

並列分散コンピューティング (11)合意問題

大瀧保広

今日の内容

- 合意問題
 - プロセスが故障しない場合の合意問題
 - 故障の種類
 - 故障がある時の合意問題
 - 同期システムの場合
 - 非同期システムの場合
- ビザンチン合意問題
 - ビザンチン故障とは

合意問題

- 分散システムの各プロセスで同じ結論を得る問題。
- 何が難しいのだろうか？

合意問題 待ち合わせ（設定）

- A,B,Cの3人は離れて住んでおり、手紙で連絡しあっている。
久しぶりに会うために待ち合わせ場所を決めたい。

前提

- 郵便事情はとても悪く、手紙はいつ届くかわからない。
- 3人とも持病持ちで、入院して手紙が出せなくなる可能性がある。
- 待ち合わせ場所の候補は、**山上駅** と **川下駅** の2択とした。
- 誰かが入院した場合、残りの2人だけで合意できれば良い。

目標

- 待ち合わせ場所について、（入院していない）**全員が同じ結論**を得る。

合意問題 待ち合わせ（アルゴリズム構築）

多数決で決定する方法を考えてみよう

■ルール1（多数決）

それぞれが山上駅と川下駅のどちらを希望するかを、他の二人に送る。

3人分の情報が揃ったら多数決を求める。

- 欠点：例えばCさんが入院していた場合、AとBは、Cからの手紙を永遠に待ち続けてしまうことになる。そこで...

■ルール2（タイムアウト）

一定時間待っても手紙が来なかったら、入院しているとみなすことにする。

- Cからの連絡がない場合、AとBの意見しか集まらない。2票が一致すればいいが、AとBの意見が合わなかったら多数決で決まらない。そこで...

■ルール3（決裂時のデフォルト値の準備）

多数決で決まらない場合には、山上駅とすることに決めておく。

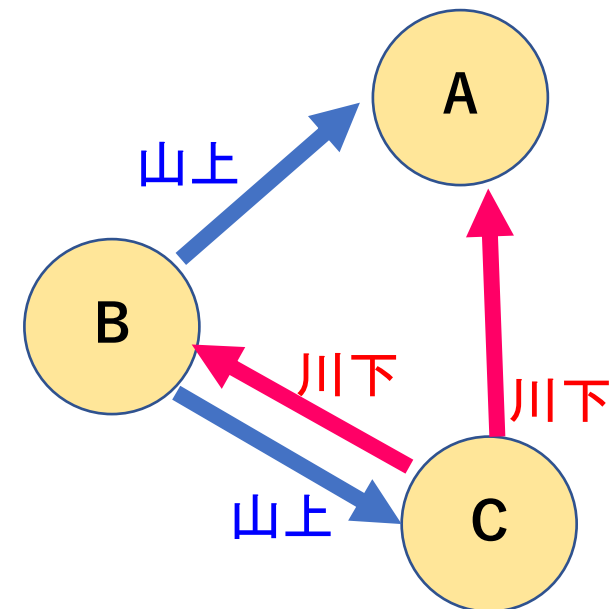
合意問題 待ち合わせ（どうなった？）

ルールに従って待ち合わせ場所を決めて、久しぶりに会おう！

- BにはCから「川下駅が希望」という手紙が届いた。
- Bには 一定時間たってもAからの手紙が届かなかった。
- Cには Bから「山上駅 希望」という手紙が届いた。
- Cには 一定時間たってもAからの手紙が届かなかった。

- Bは、ルールに従い、
「Aは入院している に違いない。
BとCの意見は一致していない。
多数決では決まらない。
ルール3により待ち合わせ場所は山上駅」
と決定する。Cも同じ。

- BとCは山上駅に向かった ...



合意問題 待ち合わせ（どうなった？）

■ BとCは**山上駅**に向かった ...

一方Aは実は入院しておらず、手紙の配達が遅れたただけだった。

■ Aは**川下駅を希望**していた。

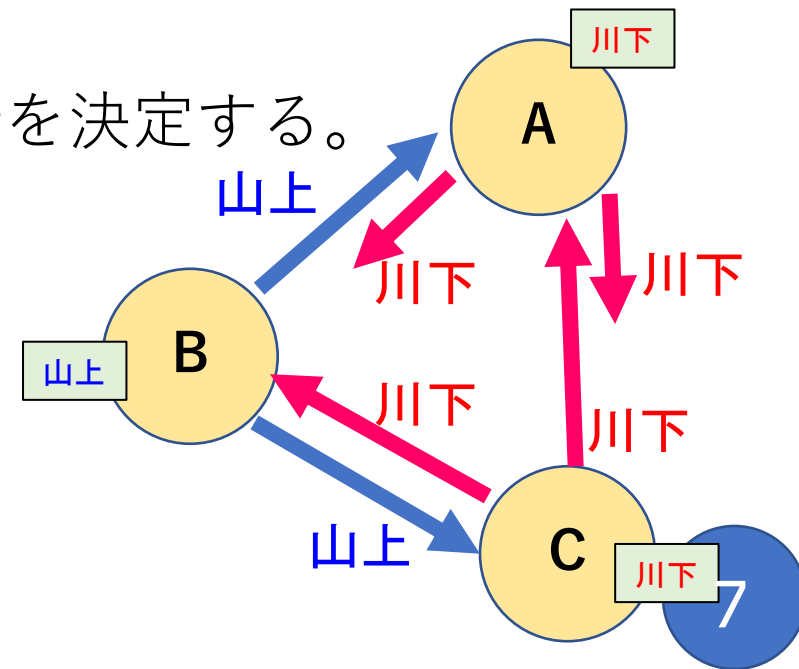
■ Aには、Bの**山上駅希望**、Cの**川下駅希望**の手紙が届いている。

■ ルールに従い、Aも待ち合わせ場所を決定する。

■ 山上駅1票、川下駅2票である。

■ 多数決により、
待ち合わせ場所は**川下駅に決定**する。

Aは**川下駅**に向かった....



合意問題 何が難しいのか

- 分散システムを構成するプロセスの故障 や通信障害（遅延、不達）などを想定することにより、合意問題は突如として難しい問題となる。
- 待ち合わせの例は極端な遅延（ほぼ不達）によって、プロセスの死活判定が失敗した例である。

このように、故障と遅延が判別できない場合には、問題を解くことができなくなる。

（各プロセスではそれぞれ合意のための計算ができるのに、実際には合意できていない！）

合意問題の定式化

- 各プロセス P_i はそれぞれ初期値 x_i をもち、アルゴリズム実行後、合意値 y_i を出力する。
各プロセスにおいて合意値 y_i は一度だけ出力する。
(後から訂正して揃える などということはしない)
- 前提
プロセスは故障する可能性がある。
- ゴール
 - 故障していないプロセスは、すべて**同一の合意値**を得る。
 - 故障していないプロセスは、**有限ステップ**で合意値を得る。
 - 合意値はプロセスの**初期値 x_i に依存**する。
例えば、「最初からいつも $y_i=0$ にすることに決めておく」はナシ。
→問題として面白くない というか 問題にならない。

プロセスの「故障」のモデル

故障プロセスの振る舞いには、いくつかのモデルがある。

■停止故障

- 最も簡単な故障のモデル

- 停止故障したプロセスは、ある時点から先 全く動作しない

■ビザンチン故障 (Byzantine failure)

- 不作為障害 (omission failures)

- 作為障害 (commission failures)

あとで少し
詳しくやります

同期システムにおける 合意問題

同期システムにおける「故障の判断」

■分散システムが同期システムであるならば、
メッセージの送信タイミングがわかっているので、
これを利用して故障しているか否かを判断できる。

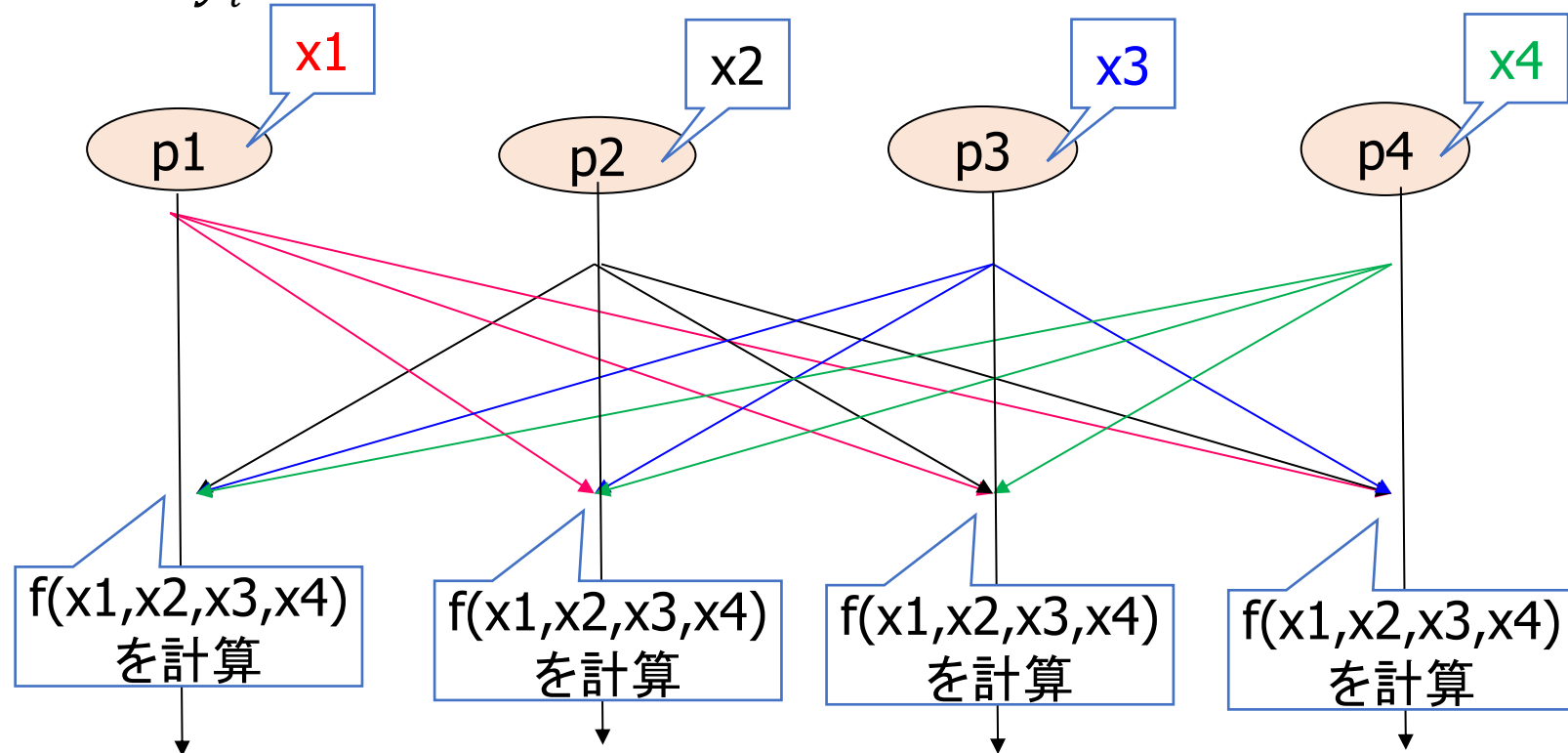
■故障の判断

プロセス P_i がプロセス P_j から

（来るはずの）値を一定時間内に受信しない場合、
 P_i は「 P_j が故障している」と判断する。

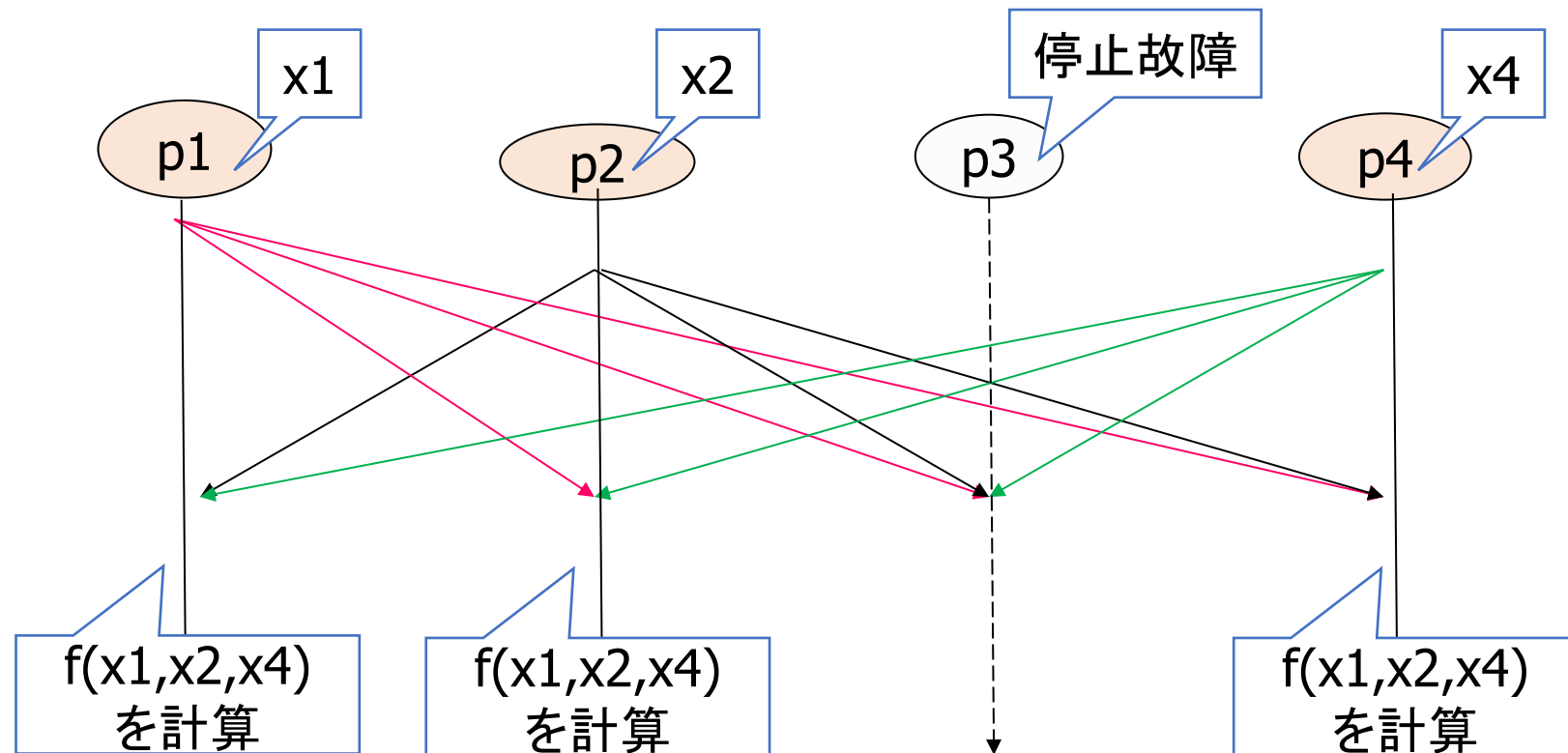
プロセスが故障しない場合（基本形）

- 各プロセス P_i は、自分の初期値 x_i を他の全プロセスに送信する。
- どのプロセスも、 x_1, x_2, \dots, x_n を得る。
- 各プロセスではある関数 $f(x_1, x_2, \dots, x_n)$ を計算し、その結果を y_i とすることで合意値を得ることができる。



プロセスが故障する場合を考える

■ **基本形**と同様に考えてみる。[アルゴリズム1]



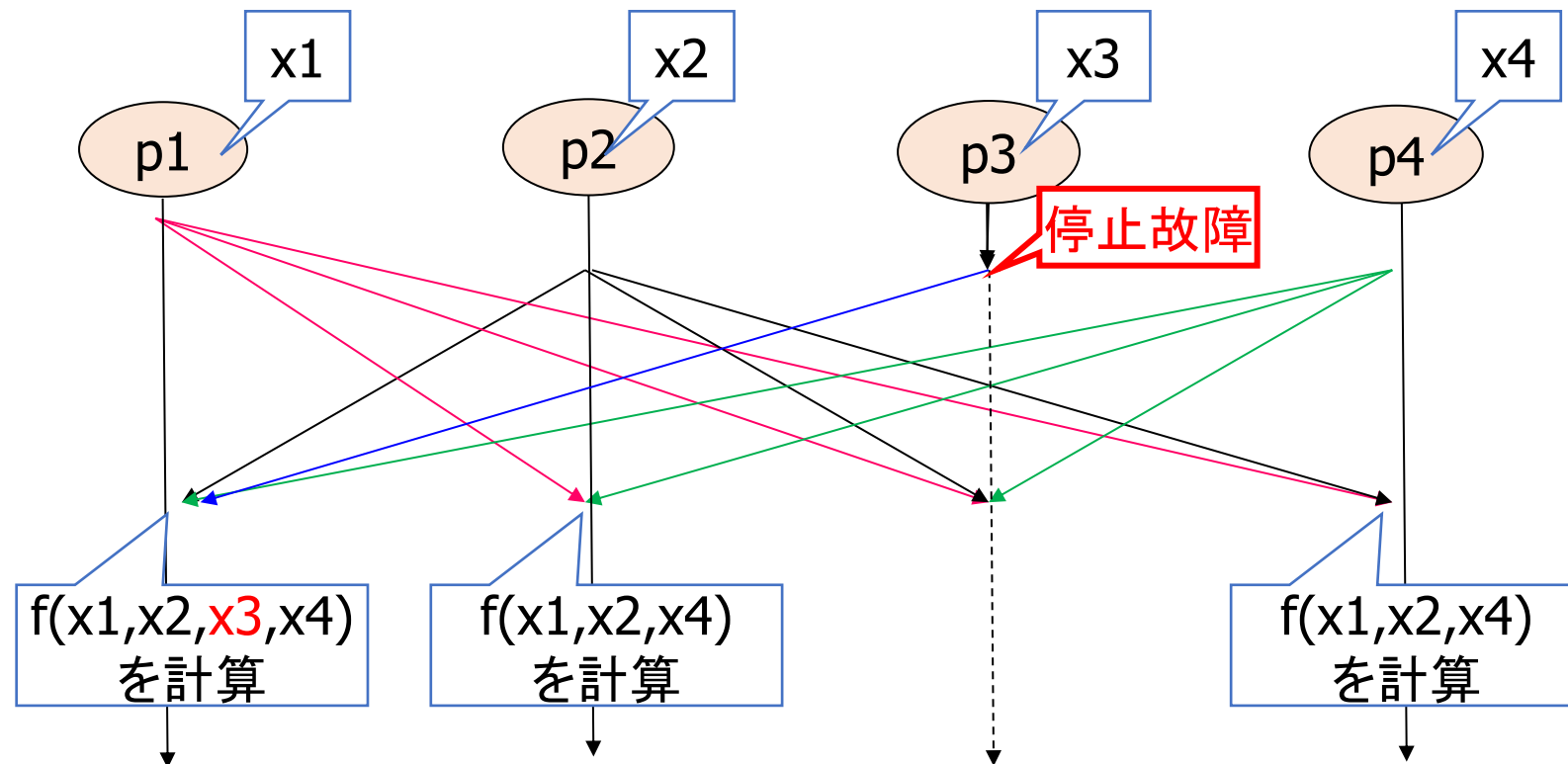
うまくいくように見える。

プロセスが故障する場合を考える

- [アルゴリズム1]は一見うまくいくように見える。
- しかし、故障プロセスが最初から停止しているときしか正しく動かない。
- プロセスは**処理の途中で故障する可能性**があり、その場合は合意が得られない。

プロセスが途中で故障する場合

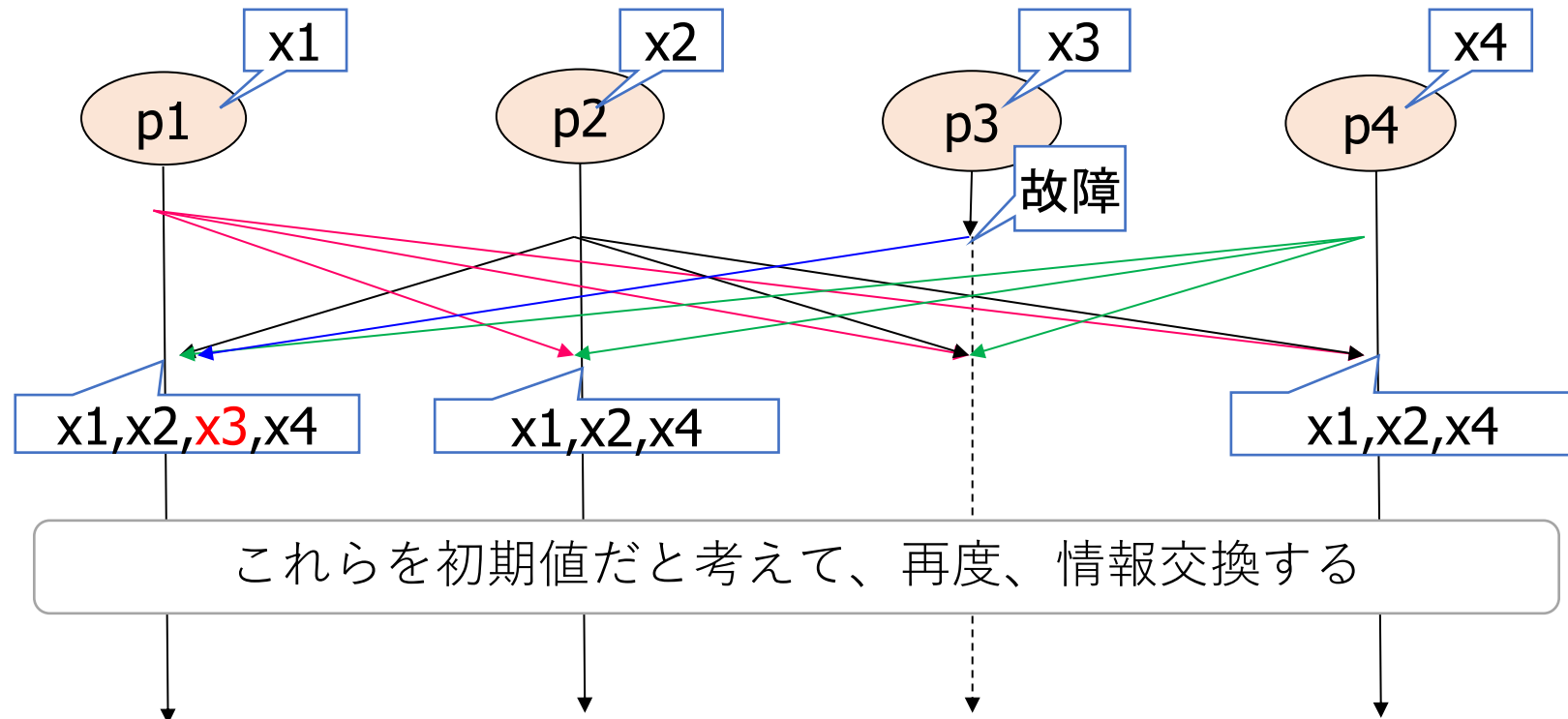
- 「全プロセスへの送信」 は実際には真に同時ではない。
- 例えば、 P_3 が「 P_1 に x_3 を送信した直後に停止故障」したとする。この場合、 P_1 は異なる合意値を計算してしまう。



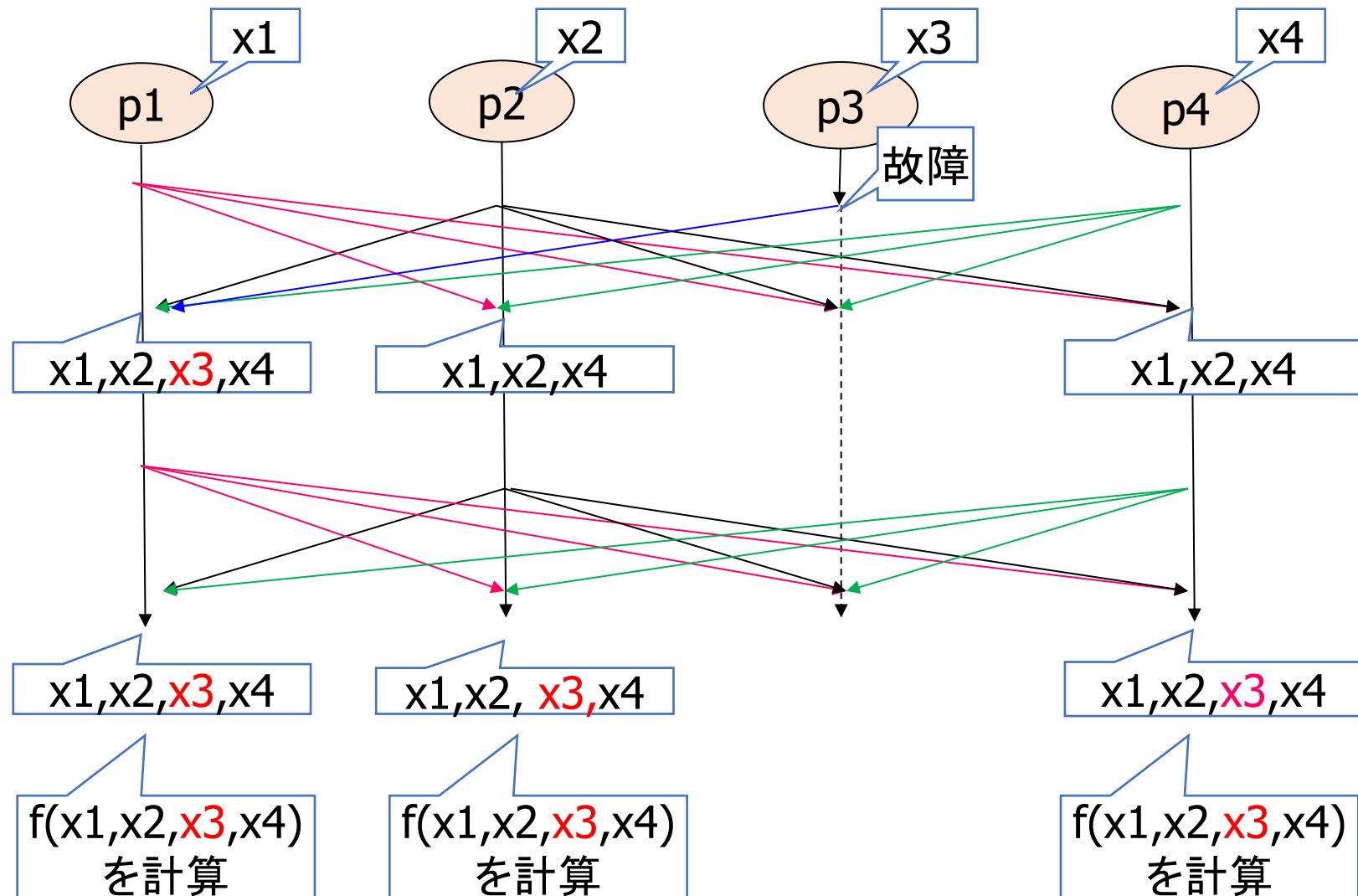
プロセスが途中で故障する場合

対応策の考え方

- 各プロセスが保持している情報の差をなくす必要がある。
- そのためには、保持している情報をプロセス間で再度交換すればよい。この情報交換を必要な回数繰り返す。

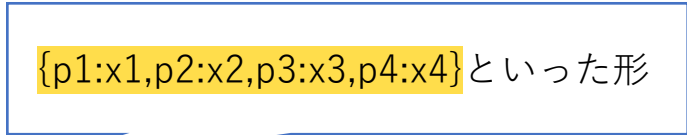


プロセスが途中で故障する場合



同期システムにおける合意アルゴリズム

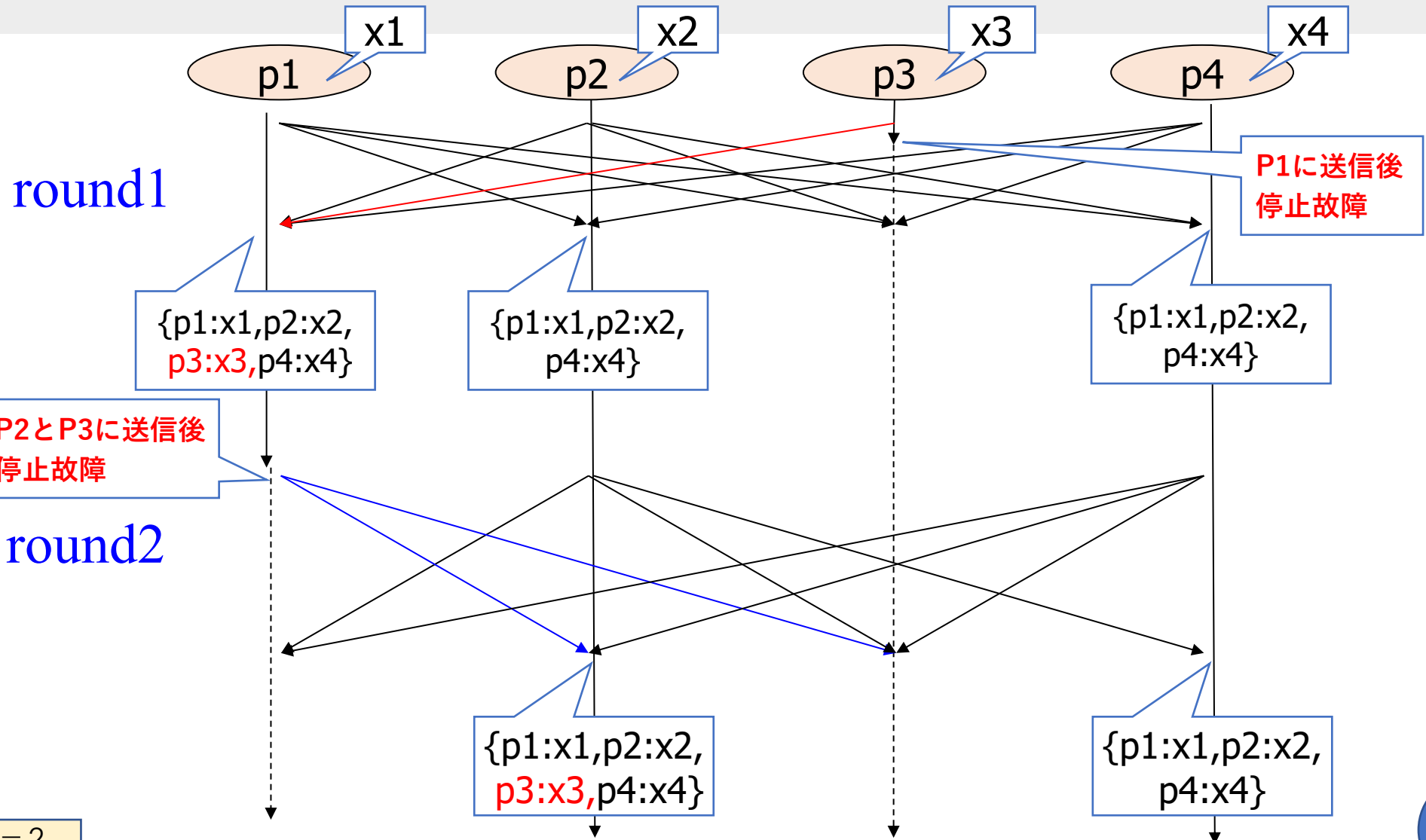
[アルゴリズム2]

- 停止故障の数の上限値を t とする。
- 以下のラウンドを **$t+1$ 回** 繰り返す。
 - 各プロセスは、各ラウンドごとに「自分が知っている各プロセスの値のリスト」を他の全プロセスに送る。

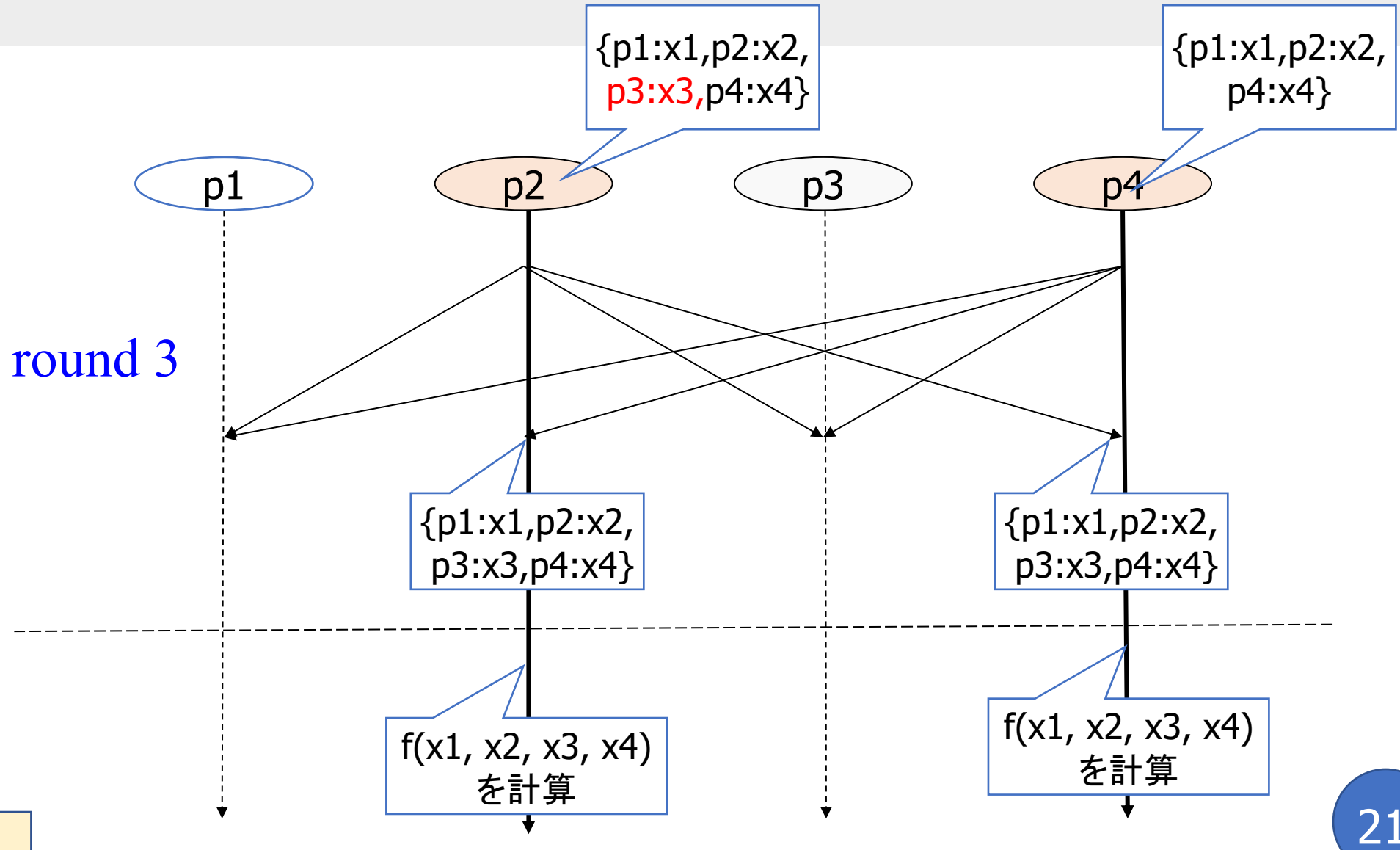
```
{p1:x1,p2:x2,p3:x3,p4:x4}
```

といった形
 - 各プロセスは、受信したリストをもとに、自分がもつリストを更新する。
- $t+1$ 回実行した後、故障していないプロセスは、全員同じリストを保持することができる。
- 各プロセスは、このリストに基づいて 合意値を計算する。

同期システムにおける合意アルゴリズム



同期システムにおける合意アルゴリズム



演習：状況設定

4プロセス P_i ($i=1, \dots, 4$) における合意問題。

- 各プロセス P_i は、それぞれ初期値 1, 2, 3, 4 をもつ。

- 合意値を求める計算式 $f()$ は、総和とする。

- 各プロセス P_i は、 P_j ($j=1, \dots, 4$ の順 ただし $i \neq j$) にデータを送信する。

- P_4 は、round1において、 P_1, P_2 に送信した直後に停止故障

- P_3 は、round2において、 P_1 に送信した直後に停止故障

- 最大故障プロセス数 $t=2$ とする合意アルゴリズムを実行したときの様子を見せ。

データがないところは - で書く。例：(1, -, 3, 4)

故障が起きなかったら 繰り返し回数を減らしていいのか

停止故障の数の上限値を t とする。
以下のラウンドを **$t+1$ 回** 繰り返す。

これは $t+1$ ラウンド実行すれば、少なくとも 1 回は
「途中で故障が起きないラウンド」が存在するということ。

では、途中で「途中で故障が起きないラウンド」が発生したら、
そこでくり返しをやめて合意値の計算をして良いか？

非同期システムの場合

- **分散システムが非同期システムの場合、**
故障プロセスの数を1以下に限る
故障は**停止故障**に限る
という最も簡単な設定としても、
合意問題を解くアルゴリズムは存在しない。

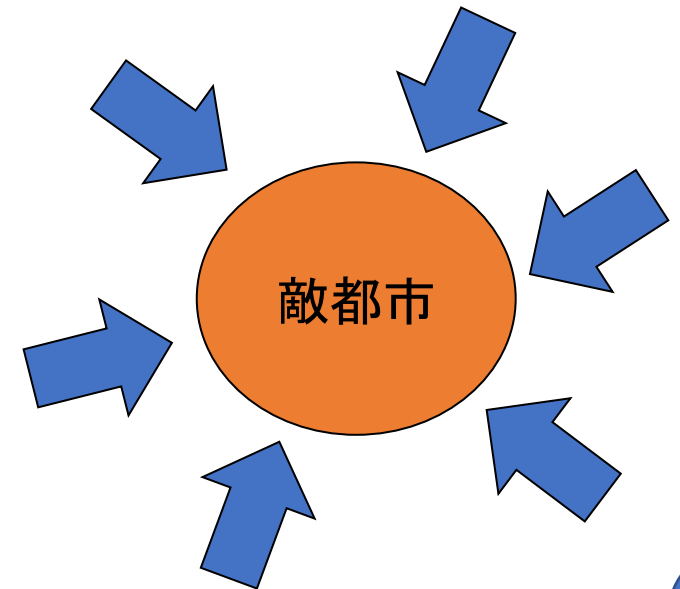
■直感的な説明

非同期システムでは、メッセージがいつ送信されるのかわからない。プロセス P_i からのメッセージが届かない時、 P_i が停止故障しているのか、それとも単にメッセージが遅れているだけなのかを、他のプロセス P_j で判別することができない。

ビザンチン合意問題

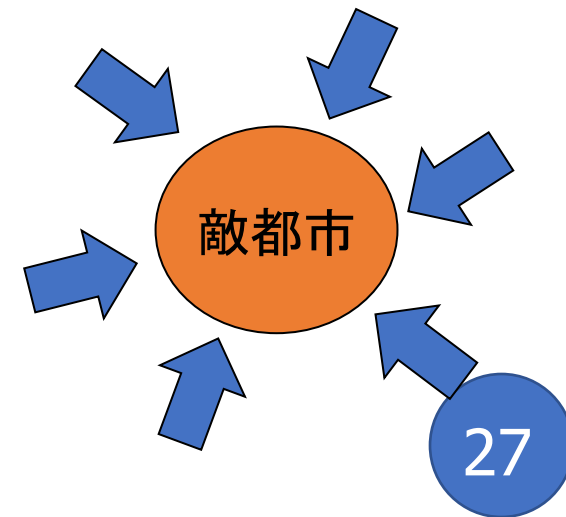
「ビザンチン将軍」問題

- ビザンチン帝国（＝東ローマ帝国）の複数の将軍たちが、それぞれ軍団を率いてひとつの都市を包囲している。将軍たちは、今後の都市攻撃計画について合意したいと考えている。
- 最も単純な形では、将軍たちは、攻撃するか撤退するかだけを合意したい。



「ビザンチン将軍」問題（つづき）

- 一部の将軍たちは攻撃したいと言うだろうし、他は撤退を望むかもしれないが、将軍たちはひとつの結論で合意しなければならない。
 - つまり合意内容に基づいて、**全員で攻撃**か**全員で撤退**をする。
 - 一部の将軍だけで攻撃を仕掛けると都市攻撃は失敗し、甚大な被害が出る。
- 将軍たちは、それぞれ離れた場所に各軍団を配置しているので、使者を相互に送ることで自分の希望を伝え合い、合意を目指す。
- 基本的には多数決**で決めるイメージ。



「ビザンチン将軍」問題（つづき）

今日 最初にやった合意問題と同じ方法じゃダメなの？

ビザンチン将軍問題を複雑にさせる要因：

- 一部の将軍たちは**反逆者である可能性があり**、
最適でない戦略に票を投じたりして混乱させるかもしれない。
- 投票内容は使者によって運ばれるのだが、
票を届けるのに失敗する場合もあるし、
途中で偽の票（異なる内容）に改ざんされる可能性もある。

「反逆」とは

9人の将軍が投票する場合を考える。

- 4人が**攻撃**に投票し、別の4人は**撤退**に投票する。
- 9人目の（反逆者である）将軍は、他の将軍からの票をみて、将軍ごとに異なる内容を送る可能性がある。例えば：
 - 撤退**に投票した将軍たちには**撤退票**を送り、
 - 攻撃**に投票した将軍たちには**攻撃票**を送る。
- この9人目の将軍から**撤退票**を受けとった将軍たちは（**撤退**が過半数であると判断して）**撤退**する。
- 9人目の将軍から**攻撃票**を受けとった将軍たちは（**攻撃**が過半数であると判断して）**攻撃**を開始する。
- その結果、**都市攻撃は失敗**することになる。

プロセスの故障の種類 再び

- **停止故障**：プロセスが一定時間以上 停止する。
- **ビザンチン故障** (Byzantine failure)
 - **不作為障害** (omission failures)
クラッシュ、要求を受信しそこなう、応答を返しそこなう など
 - **作為障害** (commission failures)
要求を不正に処理する、局所状態が壊れる、
要求に対して不正または一貫しない応答を返す など。
- ビザンチン故障が発生すると、(対策が取られていないシステムは) 予期しない動作をする恐れがある。

ビザンチン合意問題

ビザンチン故障の可能性があるプロセス間での合意問題。

ゴールの定式化

- プロセス p_1 が他の $(n-1)$ 個のプロセス p_2, \dots, p_n に値 m をもつメッセージを送ったとき、 p_1, \dots, p_n の中のすべての正しいプロセスは、**同一の値** に合意する。
 - 送信プロセス p_1 が正しければ、正しいプロセスは **値 m に基づく値に合意** する。
 - 送信プロセス p_1 が正しくなければ、 p_1 は値 m とは異なる値 m' を送ることになる。したがって、すべての 正しいプロセスは **m' に基づく値に合意** する。

ビザンチン合意問題（続き）

ビザンチン合意問題には、さまざまな条件設定のモデルがある。

ここでは最も簡単な条件設定を考える。

通信路について以下のように前提をおく。

- 通信ネットワークは信頼性がある。
⇒メッセージは紛失せず、途中で改変されず、宛先に届く。
- メッセージの遅延時間には上限がある。
- 各プロセスからすべてのプロセスに通信路がある。
(完全グラフ)

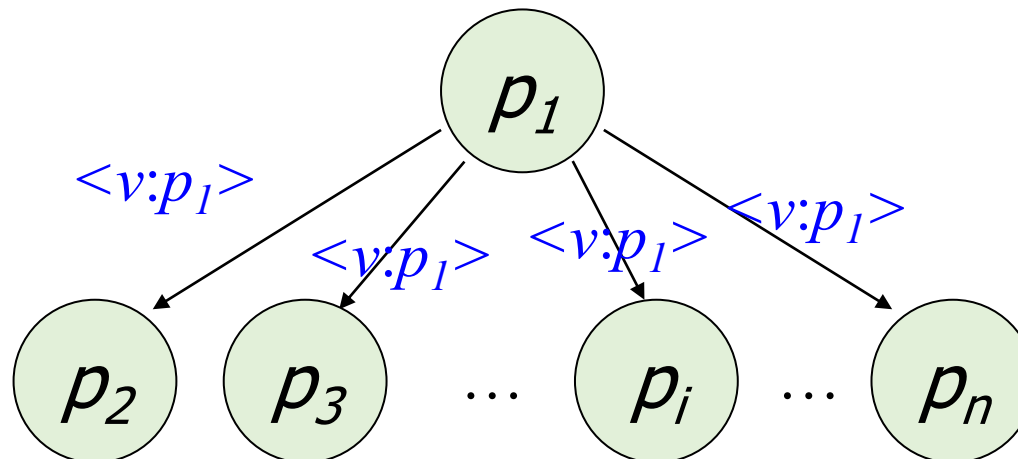
ビザンチン合意アルゴリズム

Lamport, Shostak, Peaseのアルゴリズム(1982)
(LSPアルゴリズム)

■ n プロセス p_1, \dots, p_n のうち、開始プロセスを p_1 とする。

■ (第1フェーズ)

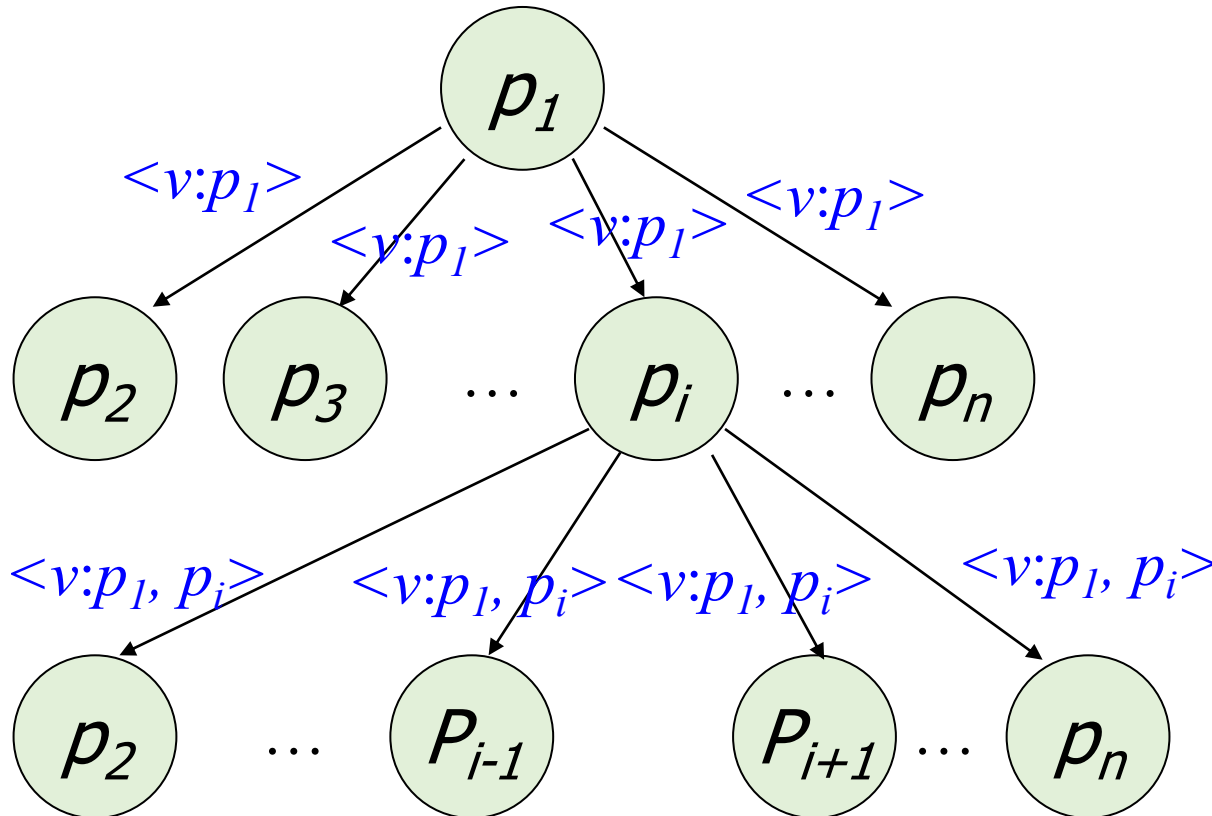
プロセス p_1 は、値 v に自分の識別子をつけたメッセージ $\langle v:p_1 \rangle$ を他のプロセス p_2, \dots, p_n に送る。



ビザンチン合意アルゴリズム

■(第2フェーズ)

$i \neq 1$ のすべてのプロセス p_i は、メッセージ $\langle v:p_1, p_i \rangle$ を p_1 と p_i 以外の $(n-2)$ 個のプロセス $p_2, \dots, p_{i-1}, p_{i+1}, \dots, p_n$ に送る。



ビザンチン合意アルゴリズム

■(第3フェーズ)

$j \neq 1, i$ のすべてのプロセス p_j は、メッセージ $\langle v: p_1, p_i, p_j \rangle$ を p_1, p_i, p_j 以外の $(n-3)$ 個のプロセスに送る。

以下同様に

■(第 k フェーズ)

メッセージを受信したプロセスは、
 k 個の識別子を含むメッセージを、メッセージに
識別子が含まれていない $(n-k)$ 個のプロセスに送る。

ビザンチン合意アルゴリズム

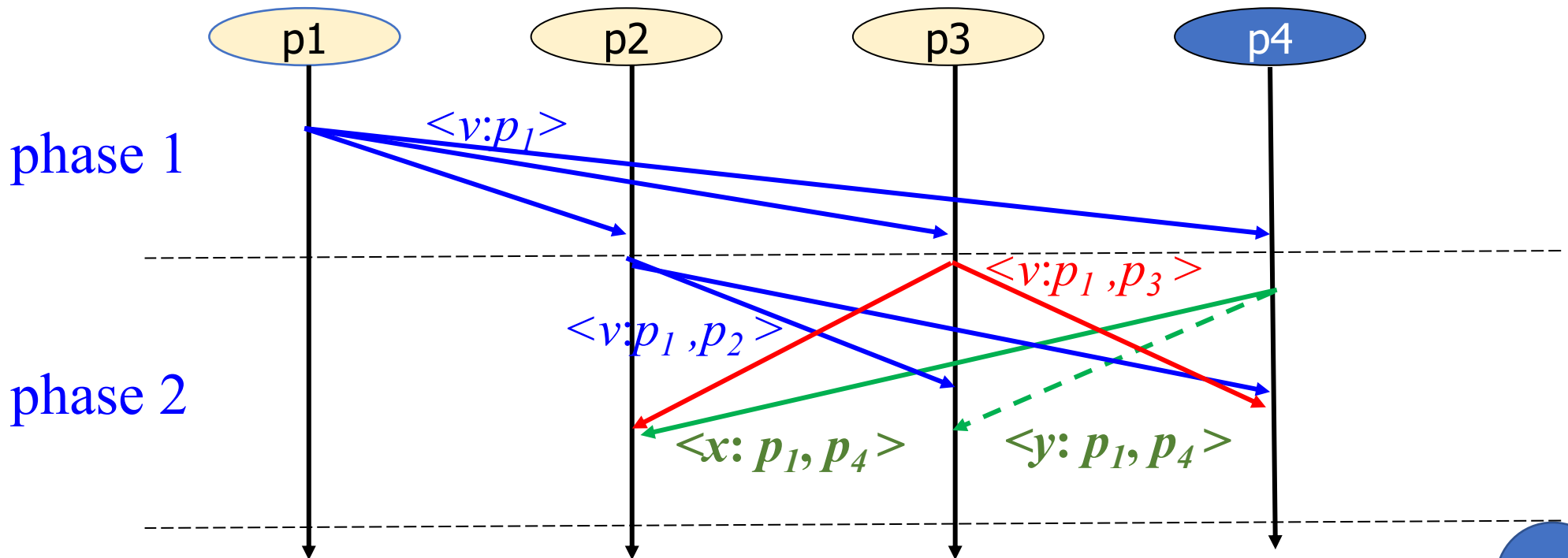
合意値の決定方法

- 最大故障数を t とする。
- 第 $(t+1)$ フェーズのメッセージ送信後、各プロセスは得られたメッセージの過半数の値 v を求め、これを合意値とする。
- 過半数の値がない場合は、メッセージの中の適当な値、例えば v_1 を選ぶ。
(どうやって選ぶかは事前に決めておく。
たとえば一番大きい数字)

4プロセスの場合（シナリオ1）

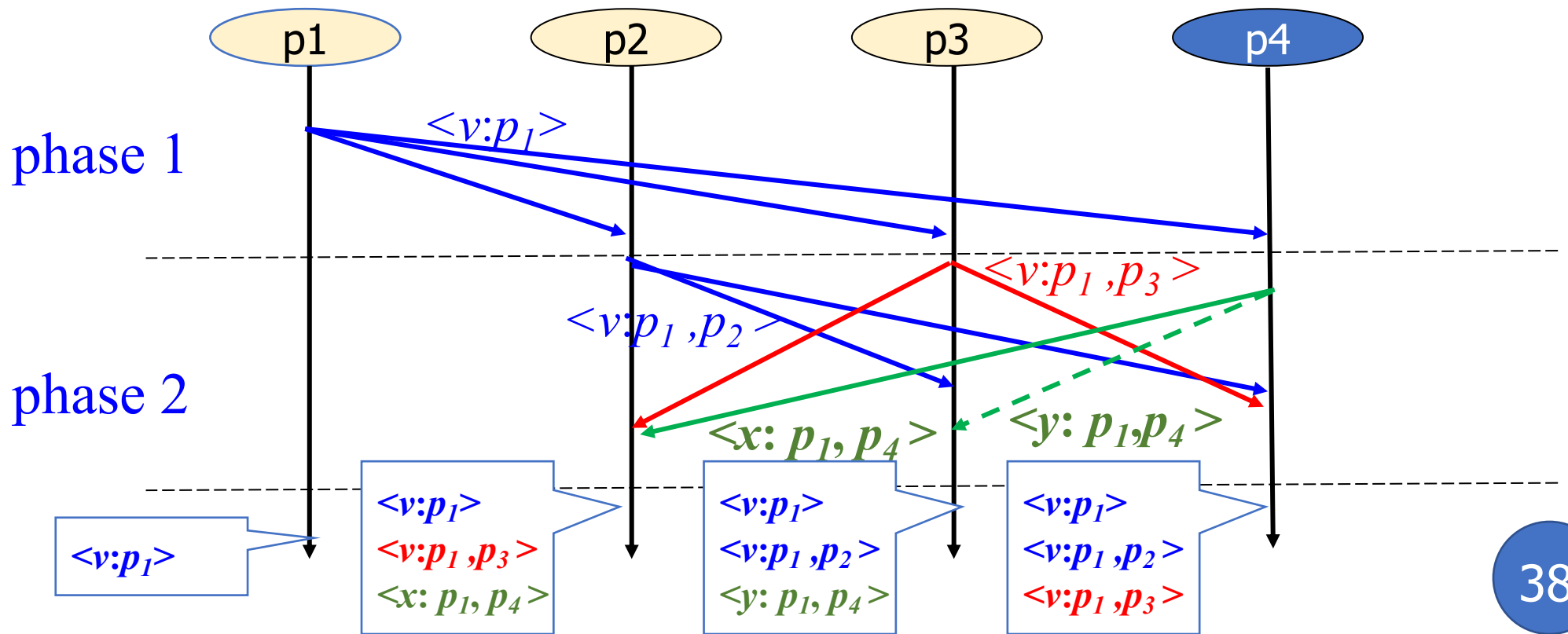
- 開始プロセスP1とする。
- 最大故障数1、プロセスP4がビザンチン故障。

プロセスP4は、
プロセスp2とp3
に対して、
値vとは異なる値
x, y をそれぞれ送る。



4プロセスの場合（つづき）

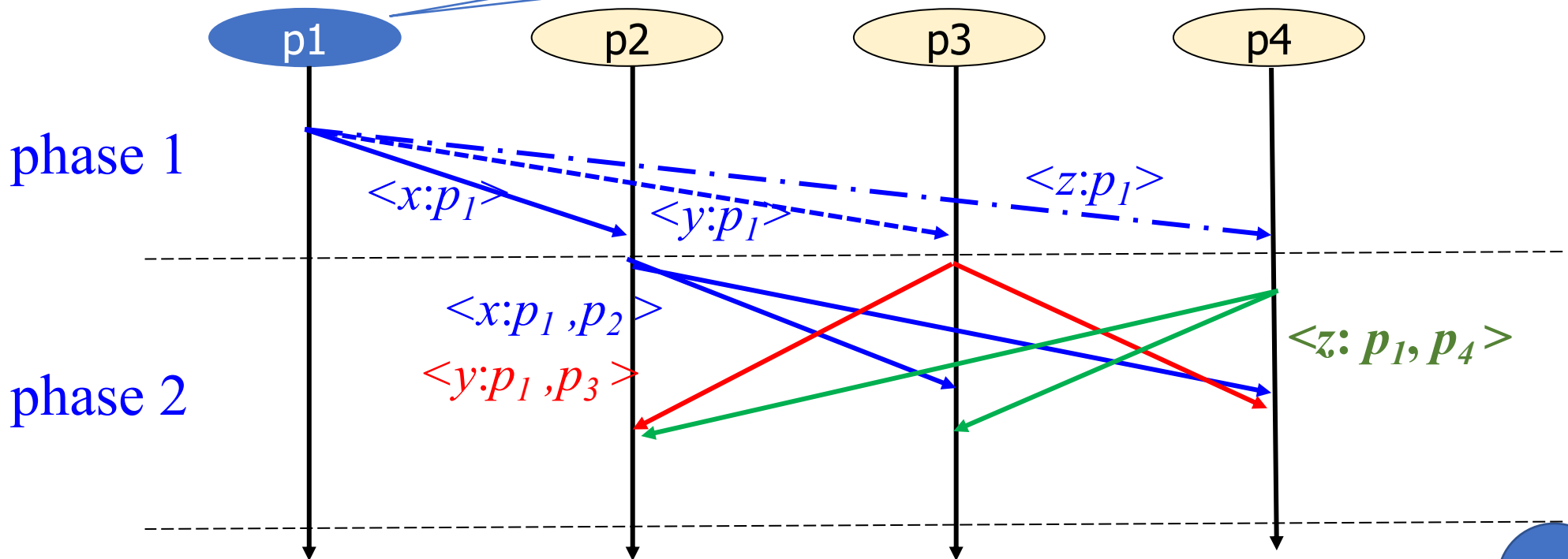
- プロセスp2が(p1, p3, p4)から受信した値は{v, v, x}、プロセスp3が(p1, p2, p4)から受信した値は{v, v, y}となる。
- 過半数を占める値に合意する。→ vで合意。



4プロセスの場合（シナリオ2）

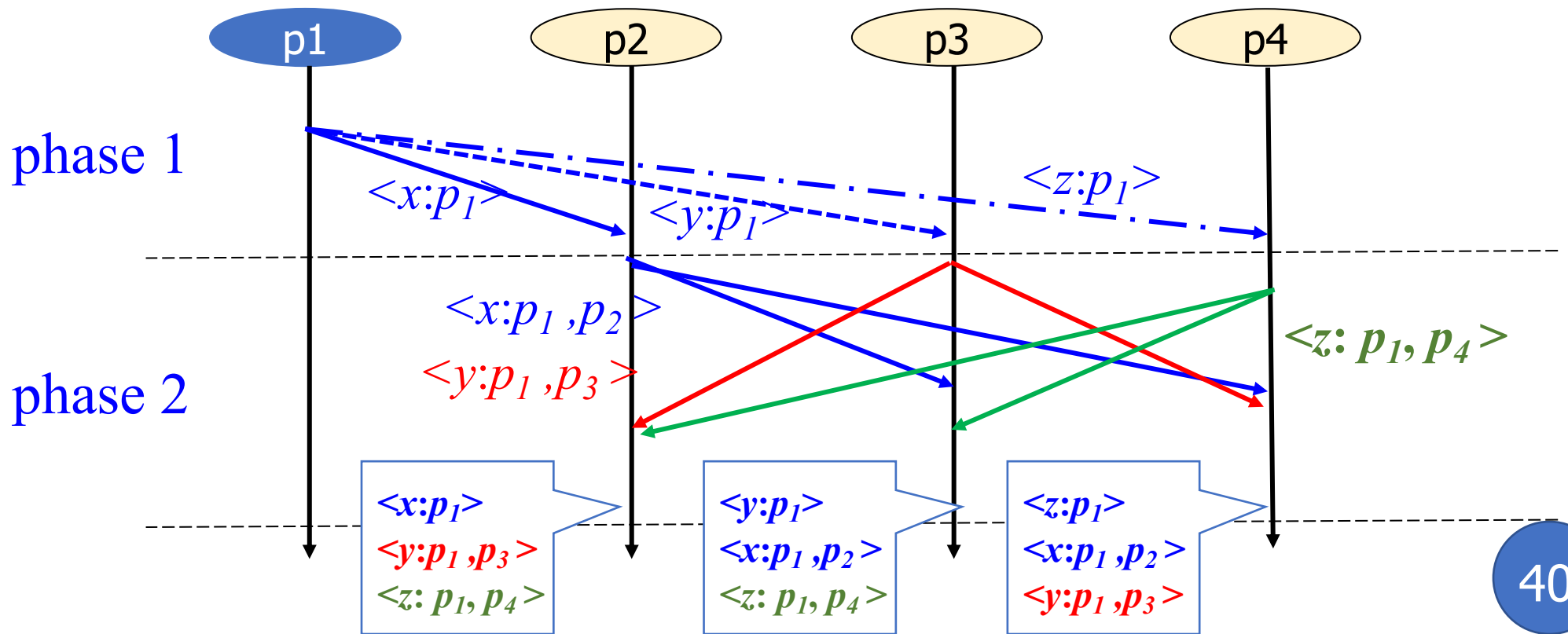
- 開始プロセスP1とする。
- 最大故障数1、プロセスP1がビザンチン故障。

プロセスP1は、
プロセスp2,p2,p3
に対して、値
x,y,zをそれぞれ送る



4プロセスの場合（続き）

- 第2フェーズまでで、プロセスp2, p3, p4が受信した値はともに{x, y, z}で、過半数を占める値が存在しない。
- 前もって決めた手順によって、何らかの値を選ぶ。



LSPアルゴリズムの解析

- 故障プロセス数を最大 t とするとき、 $(t+1)$ フェーズの処理が必要。

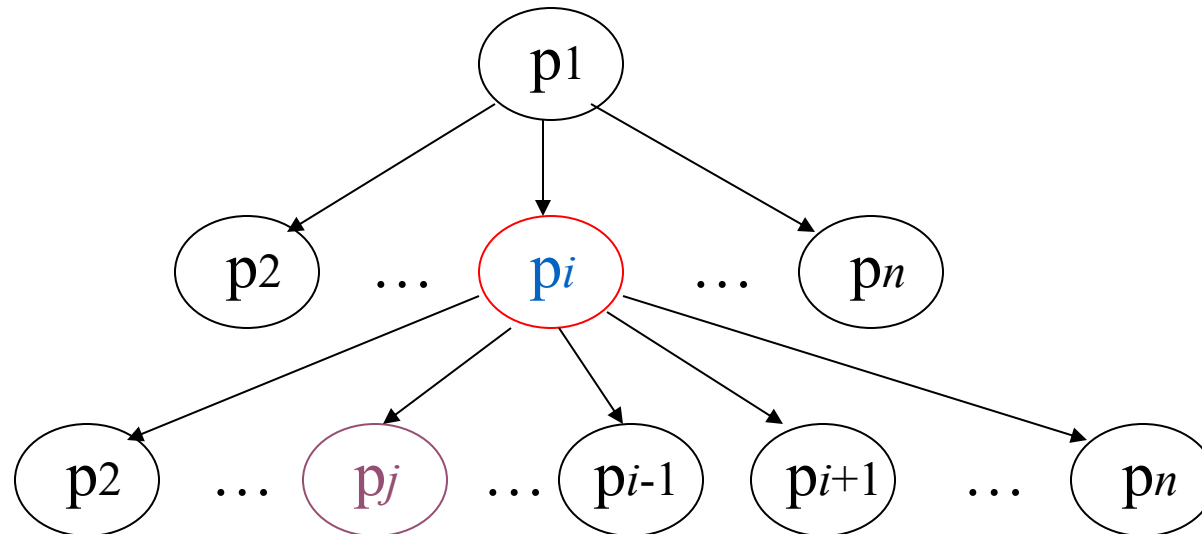
- メッセージ数

第1フェーズでのメッセージ数： $(n-1)$

第2フェーズでのメッセージ数： $(n-1)(n-2)$

⋮

第 k フェーズでのメッセージ数： $(n-1)(n-2) \cdots (n-k)$



LSPアルゴリズムの解析 (つづき)

第(t+1)フェーズで合意を作る。

■第(t+1)フェーズのメッセージ数： $(n-1)(n-2) \cdots (n-t-1)$

■第(t+1)フェーズまでのメッセージの総数：

$$\begin{aligned} & (n-1) + (n-1)(n-2) + \cdots + (n-1)(n-2) \cdots (n-t-1) \\ = & O(n) + O(n^2) + \cdots + O(n^{t+1}) \\ = & O(n^{t+1}) \end{aligned}$$

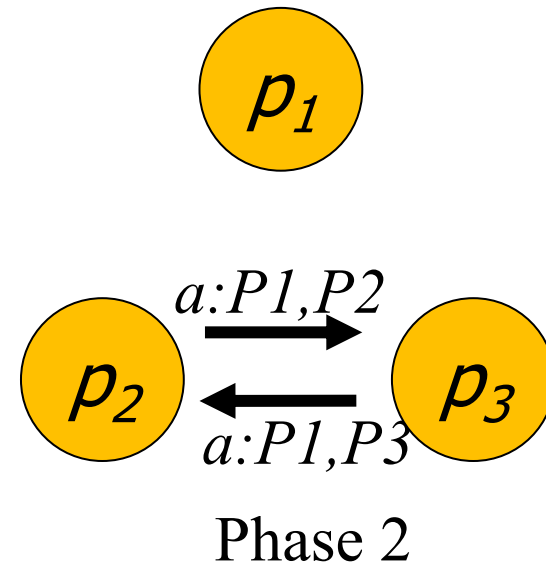
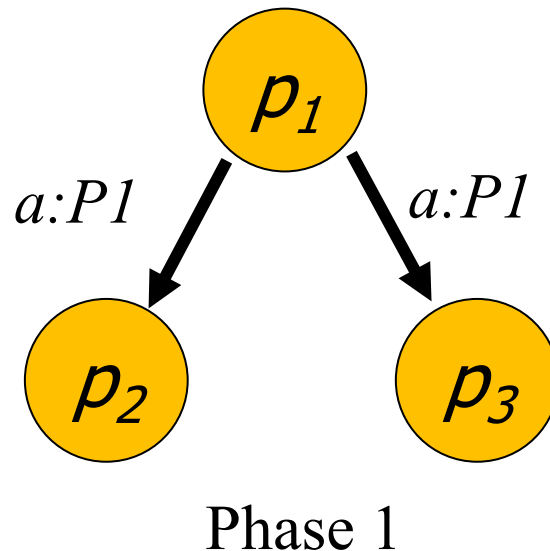


3プロセスの
ビザンチン合意問題を
考えてみよう

3プロセスのビザンチン合意問題

Case0: 故障がない場合

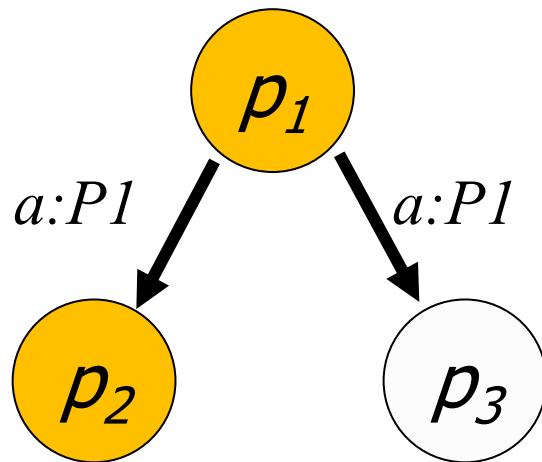
- 開始プロセスを p_1 とする。
- 開始プロセス p_1 が値 a のメッセージを送り、プロセス p_2 と p_3 が受け取る。
- プロセス p_2 と p_3 は、受信したメッセージ a を互いに交換する。



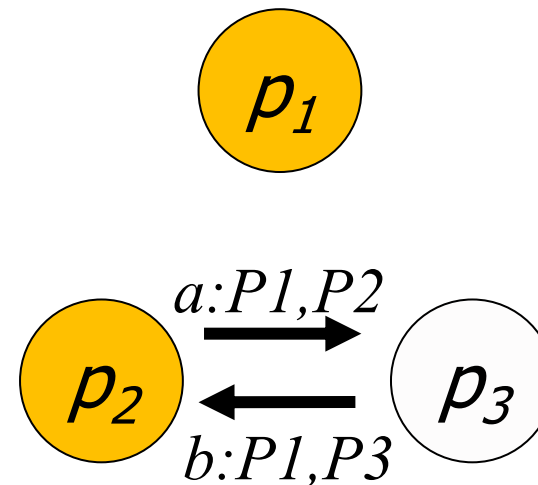
3プロセスのビザンチン合意問題

Case1: プロセス p_3 がビザンチン故障している場合

- 開始プロセス p_1 が値 a をプロセス p_2 と p_3 に送る。
- プロセス p_3 は、受信した値 a とは異なる値 b を p_2 に送る。
- プロセス p_2 は、 p_1 から値 a を受け取り、 p_3 からは値 b を受け取る。



Phase 1

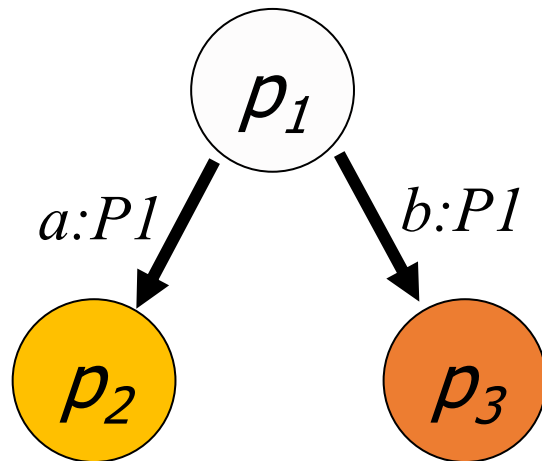


Phase 2

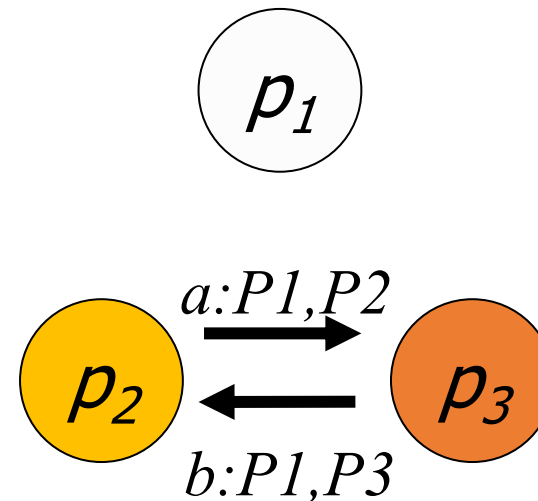
3プロセスのビザンチン合意問題

Case2: プロセス p_1 がビザンチン故障している場合

- 開始プロセス p_1 は、 p_1 と p_3 に それぞれ異なる値 a, b を送る。
- プロセス p_2 は、 p_1 から値 a を受け取り、 p_3 から値 b を受け取る。



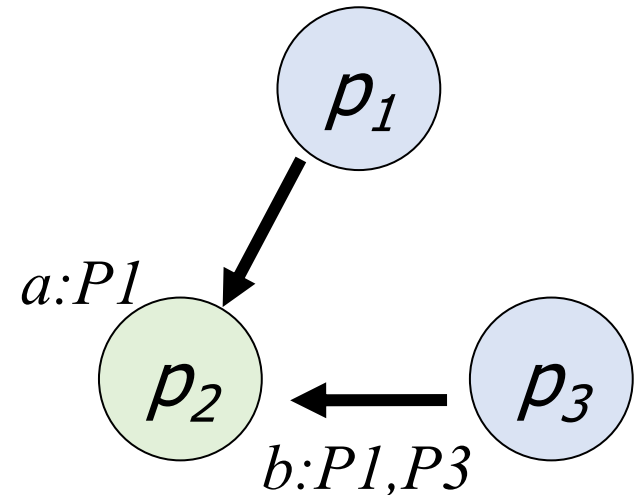
Phase 1



Phase 2

3プロセスのビザンチン合意問題

- プロセスP2の立場としては、case1, case2のいずれのケースでもp1から値aを受け取り、p3から値bを受け取っている。
 - P2は、p1とp3のどちらが故障しているのか判別できない。
 - 過半数が計算できない
- プロセスp3も同様。
- 適当な値を選んでも合意できていない。



3プロセスの場合、ビザンチン合意問題を解くアルゴリズムは存在しない。

- 正確に言えば、 $n > 3t$ を満たさない場合に解くことができない。
- 全プロセスの**3分の1未満**までのビザンチン故障には対応できる。
(9人の将軍のビザンチン将軍問題では、反逆者は二人まで)

今回のまとめ

■合意問題

- システムに含まれるプロセスが共通の値を得る。
- プロセスが故障しない場合の合意問題は簡単
- 故障がある時の合意問題
 - 故障の種類
 - 停止故障とビザンチン故障（嘘をつくプロセス）
 - 同期型のシステム＋停止故障
 - 非同期のシステム：合意アルゴリズムは存在しない
- ビザンチン合意問題
 - LSPアルゴリズム
 - 3プロセスの場合にはビザンチン合意問題を解くことはできない。