並列分散コンピューティング
(9)論理時計

大瀧保広

今日の内容

- 物理時計と論理時計
  - 事象の順序のみに着目した「時計」
- 事象と順序関係
- 論理時計
  - Lamportのタイムスタンプ
  - 論理ベクトル時間

物理時計と論理時計

- 通常「時計」というと、なんらかの物理量に基づく時計をイメージする。これを物理時計と呼ぶ。
  (Clock-on-the-Wall などと表現されることもある)
- 論理時計とは、システム内でだけ通用する、物理量に基づかない時計。
  （すなわち物理世界の時間の流れと独立でかまわない）
- 分散システムの動作を考える際には、必要な順序関係が正しく判断できれば十分であることが多い。
- 順序関係を判断するだけであれば、単調増加するだけのカウンタで十分といえる。

$$t' > t \Rightarrow C(t') > C(t)$$

事象(イベント)と順序

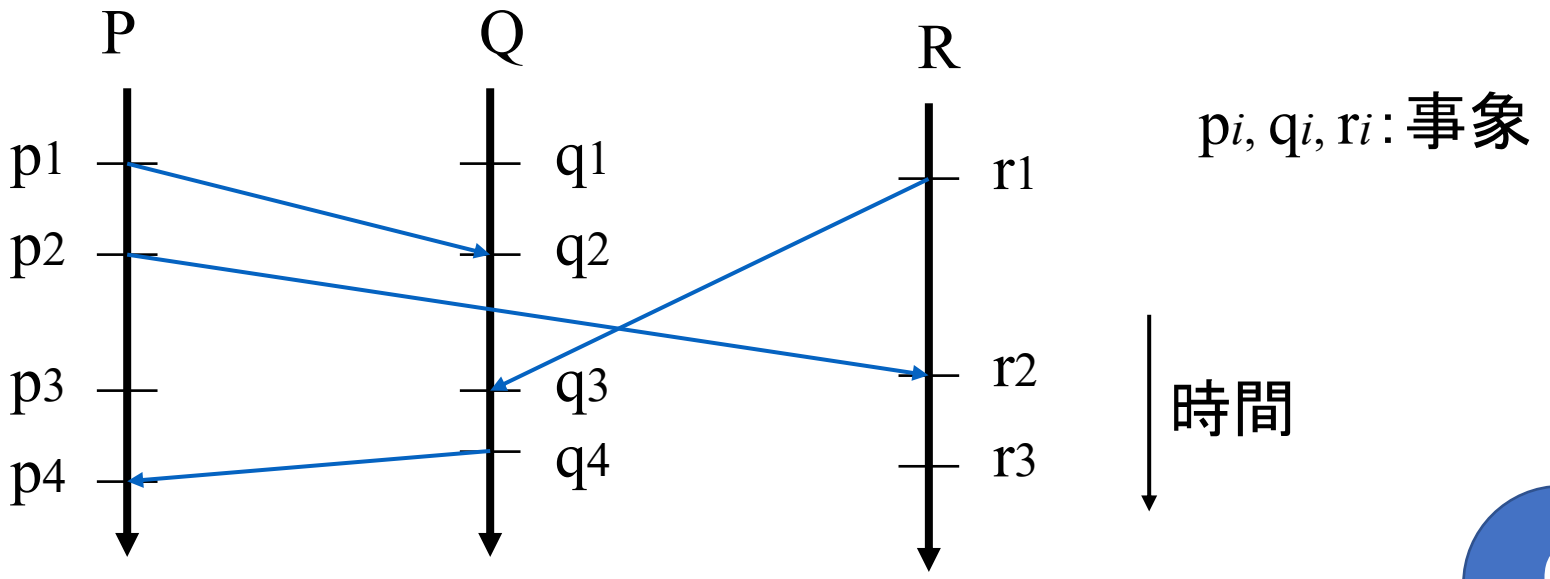
事象の順序

- あるプロセス内で生じる様々な出来事を、事象（event）と呼ぶ。
- 事象の性質
  あるプロセスの事象が原因で、他のプロセスの事象が影響を受ける場合がある。
- 事象についての仮定
  - 同一のプロセス内のすべての事象は、発生した順に順序付けることができる。（発生順を認識できる）
  - 事象は瞬時に終了する。（事象は時間軸上の点となる）
  - 同一のプロセス内では同時に2つの事象は起きない。

- 分散システムにおいて重要なことは、複数プロセスにおける事象の順序について正しく合意できることである。

時空ダイアグラム(Time-Space-Diagram)

- プロセス間でのメッセージのやり取りを表現する方法
  - 水平方向にプロセスを配置し、垂直方向に時間を表す。
- 事象(event)の起きた時点に横棒（とラベル）をつける。
- プロセス間の矢印は、メッセージの送受信を表す。



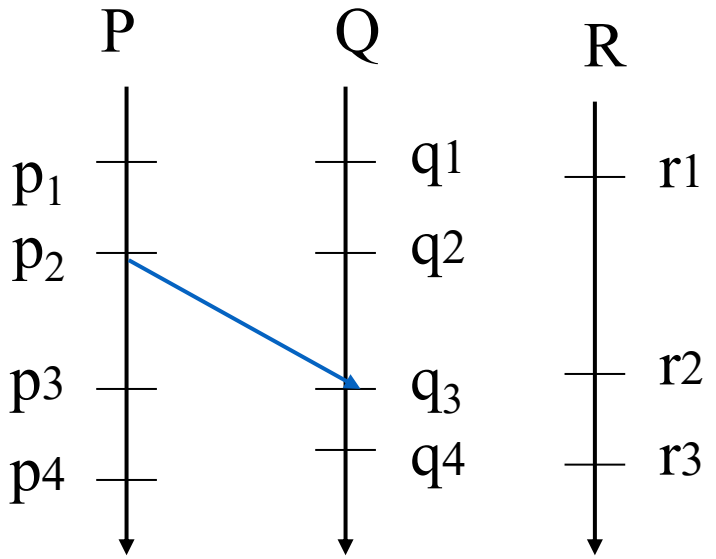
事象の順序関係

事象の順序関係を記号→で表す。

■事象aが事象bより先に起こるとき、
a→b と書く。

■同一のプロセス内では
事象の順序関係は明らか。
例：p1→p2

■あるメッセージについて、
aが送信したという事象で
bがそれを受信したという事象ならば、
a→b である。
例：p2→q3

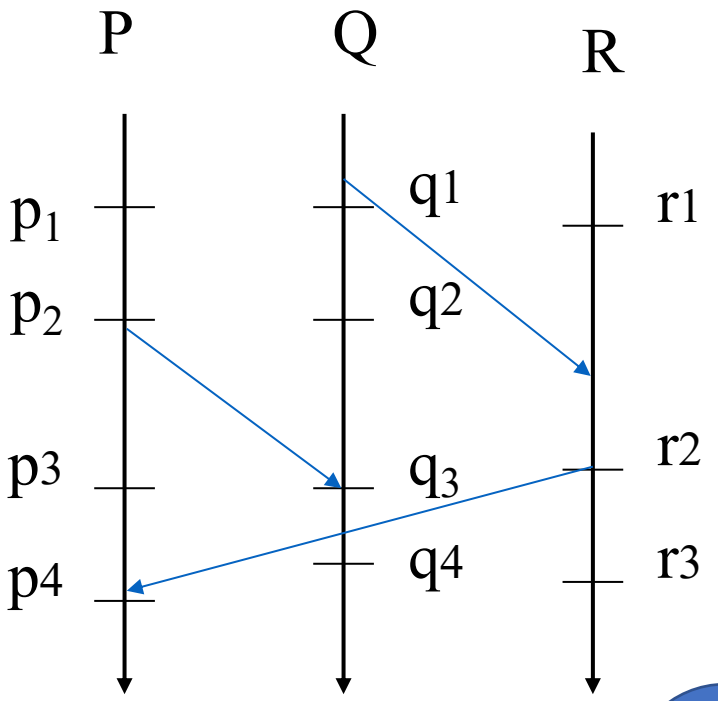


■順序関係には推移律が成り立つ。すなわち、
事象a, b, cについて a→b かつ b→c ならば a→c である。

事象の順序関係（つづき）

- 2つの事象 a, bの間に、a→b と b→a のいずれも成り立たないとき、事象aとbは並行(concurrent)であるといい、a || b と書く。

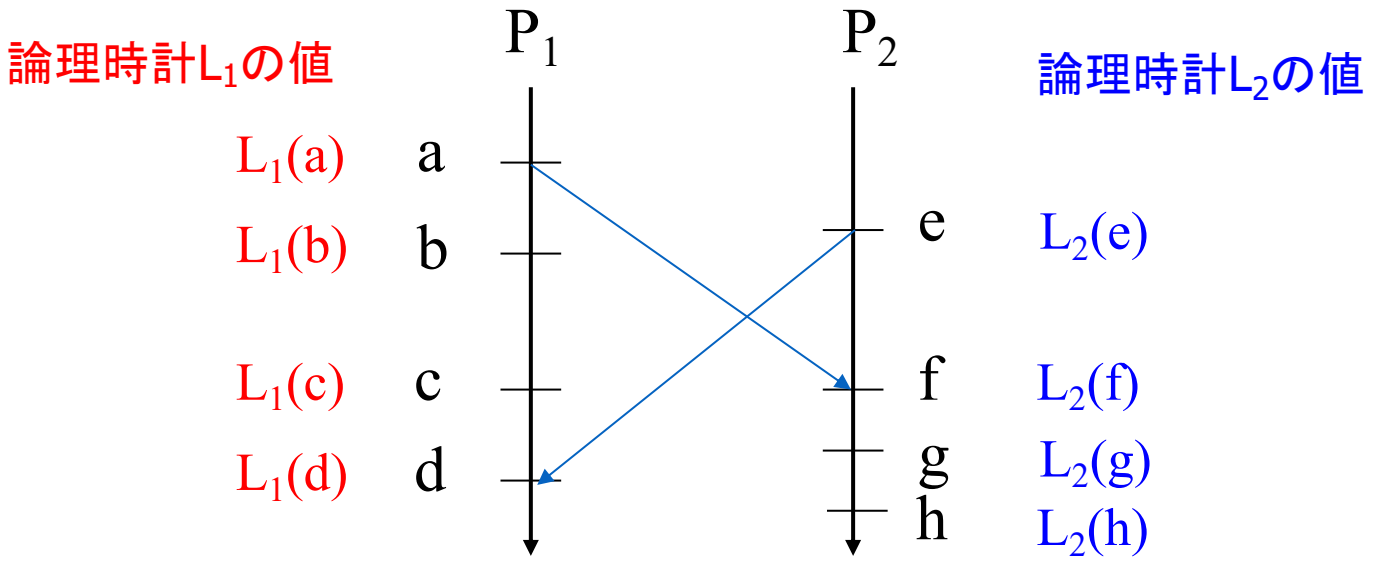
- 同一プロセス内の先行関係：
  p1→p2, q2→q3, r1→r2　など
- メッセージの送受信による関係：
  p2→q3, q1→r2, r3→p4　など
- 推移律によって導かれる関係：
  p1→q3, q1→p4　など
- 並行
  q2 || r1, p3 || q4 など



事象と論理時計

論理時計

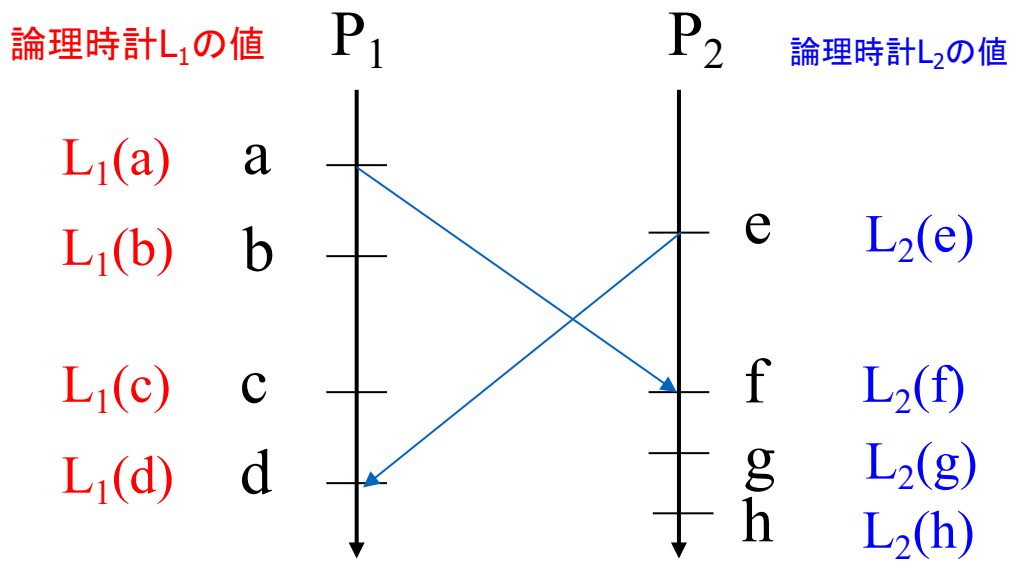
- プロセスP_i における論理時計を論理時計L_iとする。
  （論理時計はプロセスごとにある）
- プロセスP_i における事象aの論理時計L_iの値を
  $L_i(a)$と表す。



論理時計の要件ver1

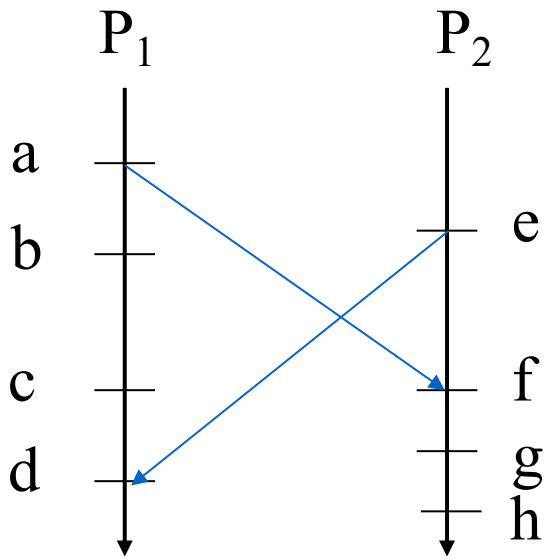
事象の順序関係に基づき、
後に発生したイベントほど大きい値であること。

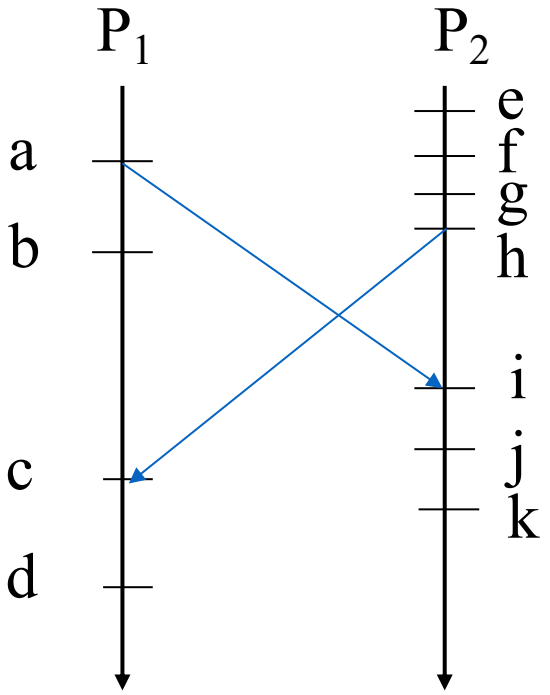
- あるプロセスP_i において、
  $a \rightarrow b$ ならば$L_i(a) < L_i(b)$
- aがプロセスP_i における
  メッセージ送信事象で、
  bがそのメッセージの
  プロセスP_j における
  受信事象ならば、
  $L_i(a) < L_j(b)$



要件を満たすような値 L(・)は
どうやったら実現できるだろうか？

単純に番号を振ってみる？





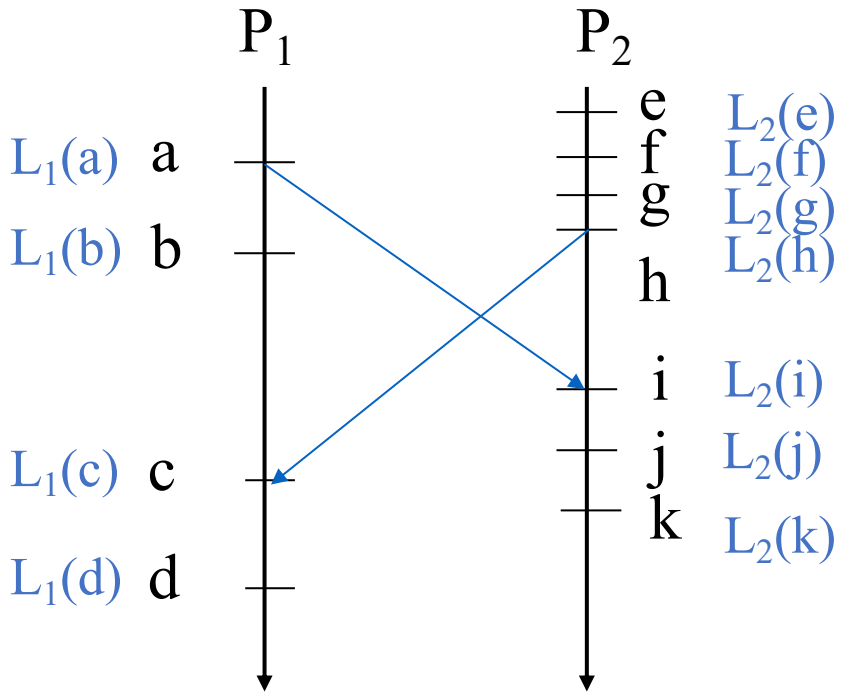
Lamportのタイムスタンプ

基本的な考え方

■L_i ：プロセスP_iの論理時計

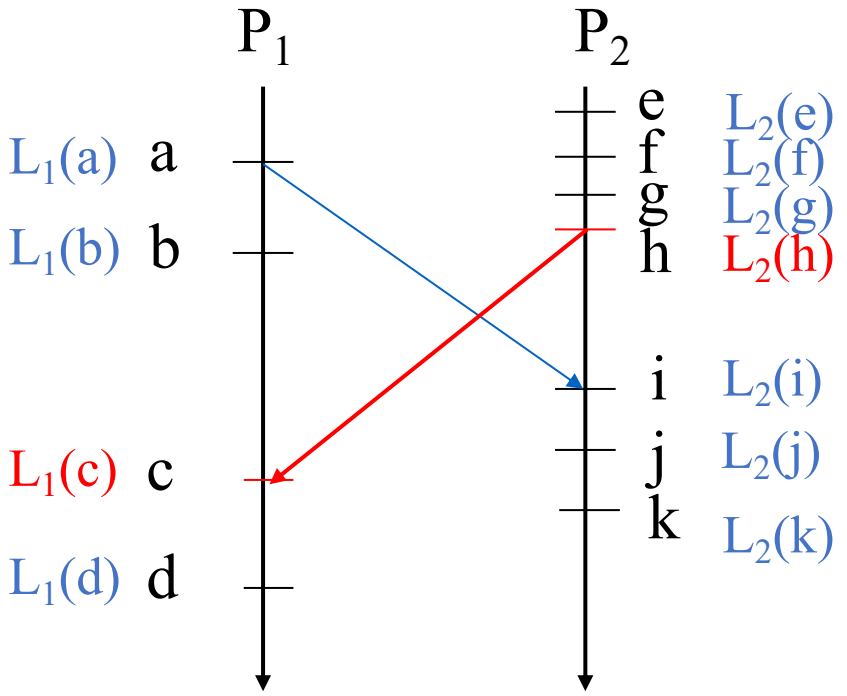
■$L_i(e)$ ：事象eのタイムスタンプ
（＝時刻）

■**$L_1(a) < L_1(b)$** を実現するには？
論理時計L_1の値を増やし、
その値をタイムスタンプとして
付与すればよい。



基本的な考え方（つづき）

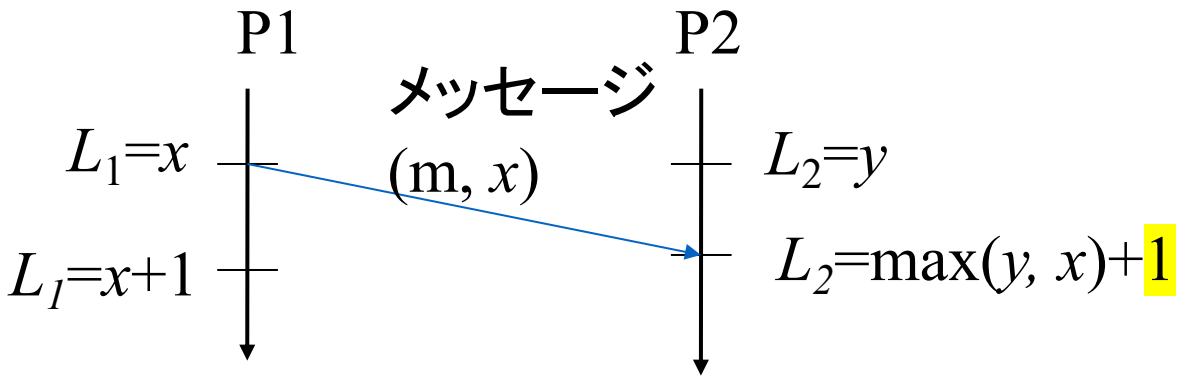
- $L_2(h) < \mathbf{L_1(c)}$ を実現するには？
  - 正確には
    $L_2(h) < \mathbf{L_1(c)}$ と $L_1(b) < \mathbf{L_1(c)}$
    を実現するには？
- 論理時計L_1の値を
  $L_1(b)$ と $L_2(h)$ のどちらよりも
  大きい値まで増やし、
  その値をタイムスタンプとして
  付与すればよい。
- しかし、P_1では$L_1(b)$はわかるが、
  $L_2(h)$の値が不明である。
  →メッセージにタイムスタンプを
  つけて教えてもらう。



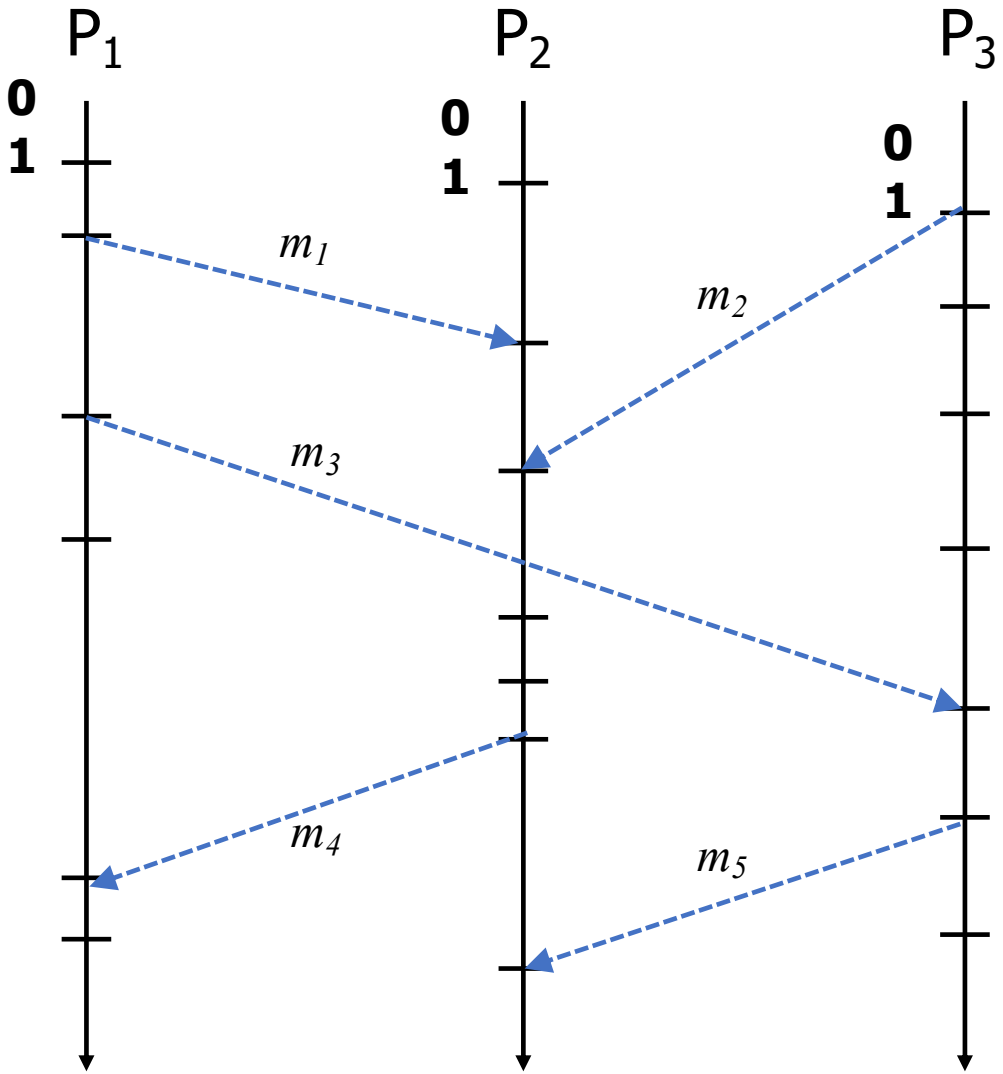
Lamportのタイムスタンプ

- 各プロセスP_iで各自の論理時計L_iを0に初期化
- 各プロセス P_i では:
  事象が発生するたびに $L_i = L_i + 1$; $L_i(event)=L_i$
- プロセス P_i がプロセス P_j にメッセージ m を送信するとき
  - 送信側 P_i:
    $L_i = L_i + 1$; $t=L_i$, message = (m, t) ; L_i(send event) = L_i
  - 受信側 P_j:
    $L_j = \max(L_j, t)$; $L_j = L_j + 1$; L_j(receive event) = L_j ;

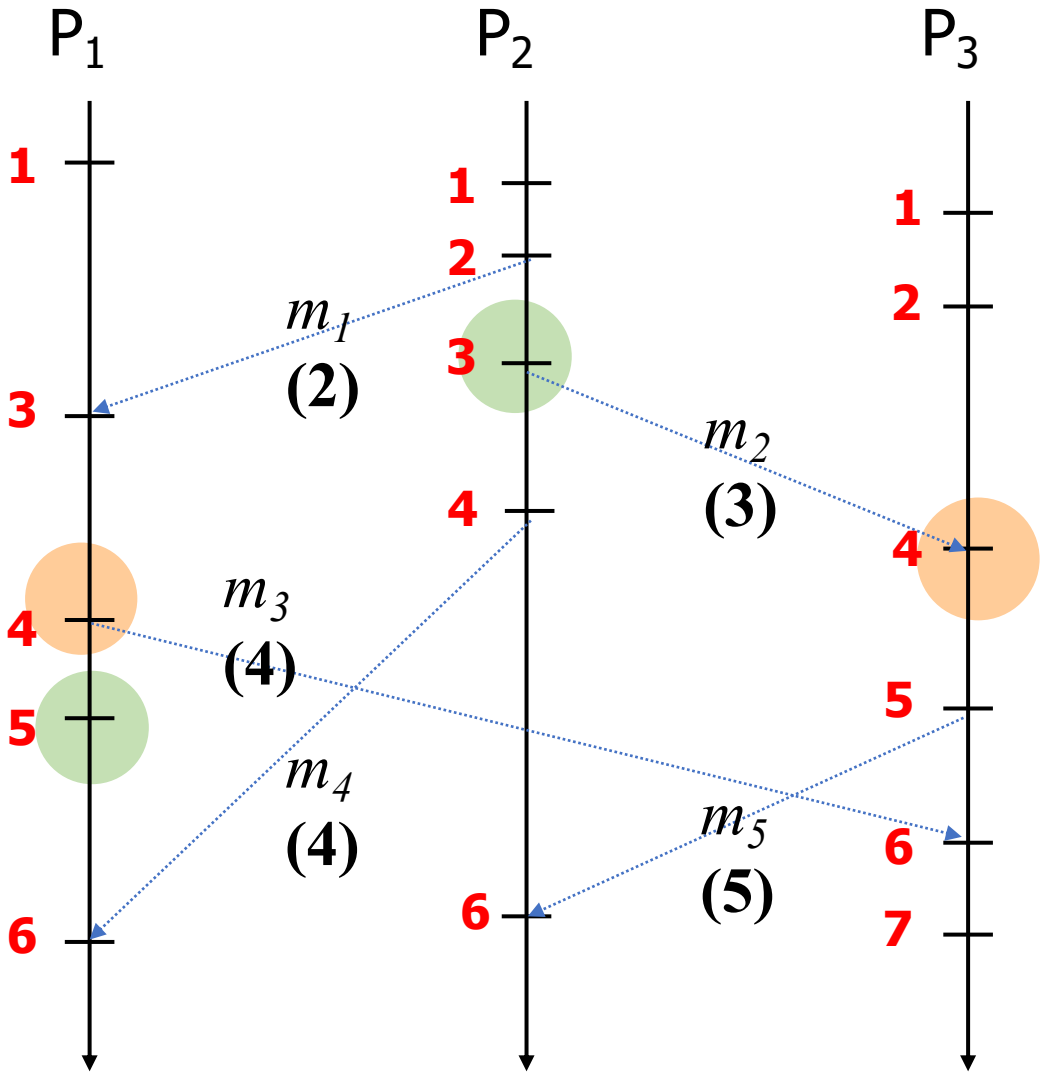
正の数であれば
何でもよい



演習：Lamportのタイムスタンプ



Lamportのタイムスタンプの問題点



Lamportのタイムスタンプの問題点

「a→b ならば L(a) < L(b) である」ようには構成することができた。しかし...

タイムスタンプの値の大小関係から
事象の順序関係が判断できない。

■問題点1：

L(a) < L(b) だからといって、 a→b とは限らない。

■問題点2：

$L_i(a) = L_j(b)$ のとき、事象a,bの順序はどう考えればいい？
（「同時に発生した」という意味？）

Lamportのタイムスタンプの問題点

■タイムスタンプの値が事象の順序を意味しない。

$L_1(a) < L_2(b)$ だからといって、 $a \rightarrow b$ とは限らない。

■これは、実際には順序関係のないところ、
すなわち $a \,||\, b$ であるところ　に
$L_1(a)<L_2(b)$といった順序関係をつけてしまったということ。
→「並行に処理可能だった」ことが わからなくなっている。

■分散システムでは、論理時計の値に基づいて
順序関係を判断したいのに、これでは困る。

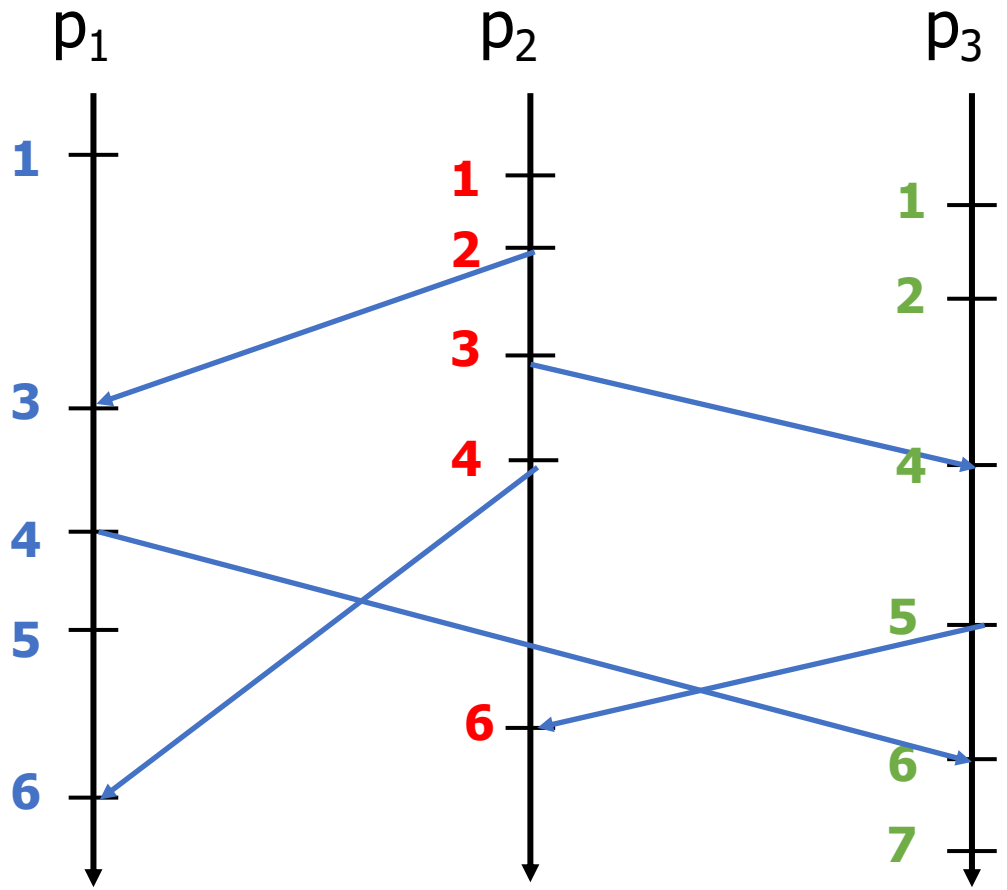
論理時計の要件ver2

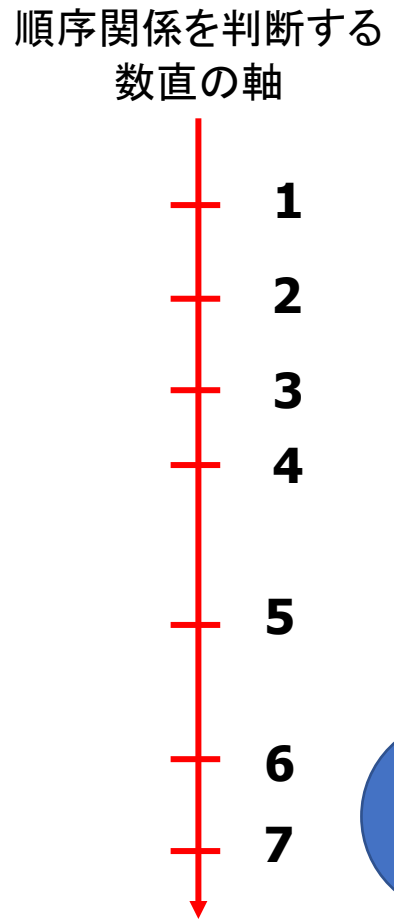
- 次のような要件を満たせる論理時計が欲しい
  1. a→b ならばL(a)＜L(b) であること。
  2. 「L(a)<L(b)ならば a→b 」といえること。
  3. L(a)とL(b)をみて、a||b であるかどうか 判断できること。

- どうやったら構成できるのだろう？
  - Larmportのタイムスタンプで要件2 と 要件3が実現できなかった真の原因はどこなのか？

Lamportのタイムスタンプの問題の原因

- 各プロセスではそれぞれ固有の論理時計でカウントしているのに、区別せずに扱っていること。





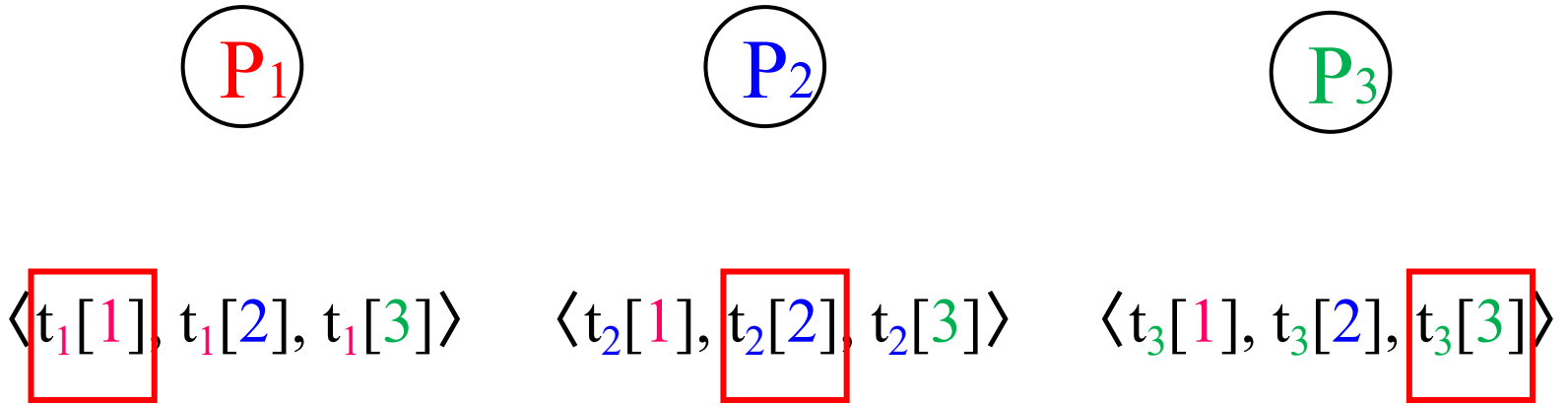
論理ベクトル時間

1つの数値（スカラー）で表すのではなく、
数値の組み（ベクトル）で表す

論理ベクトル時間

■各プロセスの論理時計の値を一緒にせず、常に区別する。
■各プロセスの値を組にして、一つのベクトルとして扱う。

■システムとしては、このベクトルをタイムスタンプとして使用する。（＝論理ベクトル時間）



$t_i[j]$：プロセスP_i が保持するプロセスP_j の論理時計の値

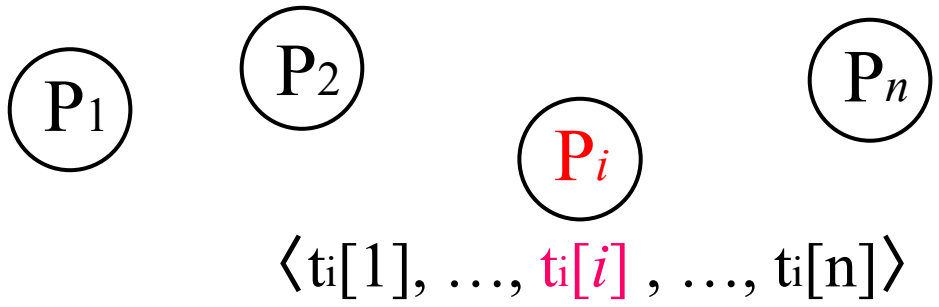
論理ベクトル時間

nプロセスからなるシステムの場合、
論理ベクトル時間は n個の数値の組みのベクトル で表す。

プロセスP_iがもつ論理ベクトル時間
$t_i = \langle t_i[1], t_i[2], \ldots, t_i[n] \rangle$

$t_i[j]$：プロセスP_i が保持するプロセスP_j の論理時計の値

$t_i[i]$：P_i 自身の論理時計の値



論理ベクトル時間の処理(1)

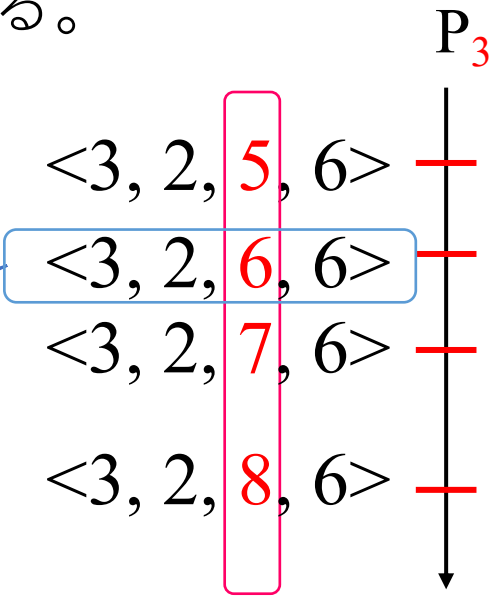
[メッセージの送受信]に関係しない事象の場合

■プロセスP_iで事象が起きたら：

1. P_i は自分自身の論理時計を進める。

   $t_i[i] \leftarrow t_i[i]+1$

2. 事象に論理ベクトル時間を付与する。



イベントに付与される
論理ベクトル時間
（タイムスタンプ）

P3の論理時計

重要：P3が保持する他のプロセスの論理時計の値は変化しない。

論理ベクトル時間の処理(2)

[メッセージの送受信]に関わるイベントの場合

- 送信イベント：通常どおり論理ベクトル時間を付与し、メッセージに「送信イベントの論理ベクトル時間」をつけて送信。
- 受信イベント：受信プロセスでは論理ベクトル時間を次のように更新し、受信イベントに付与する。

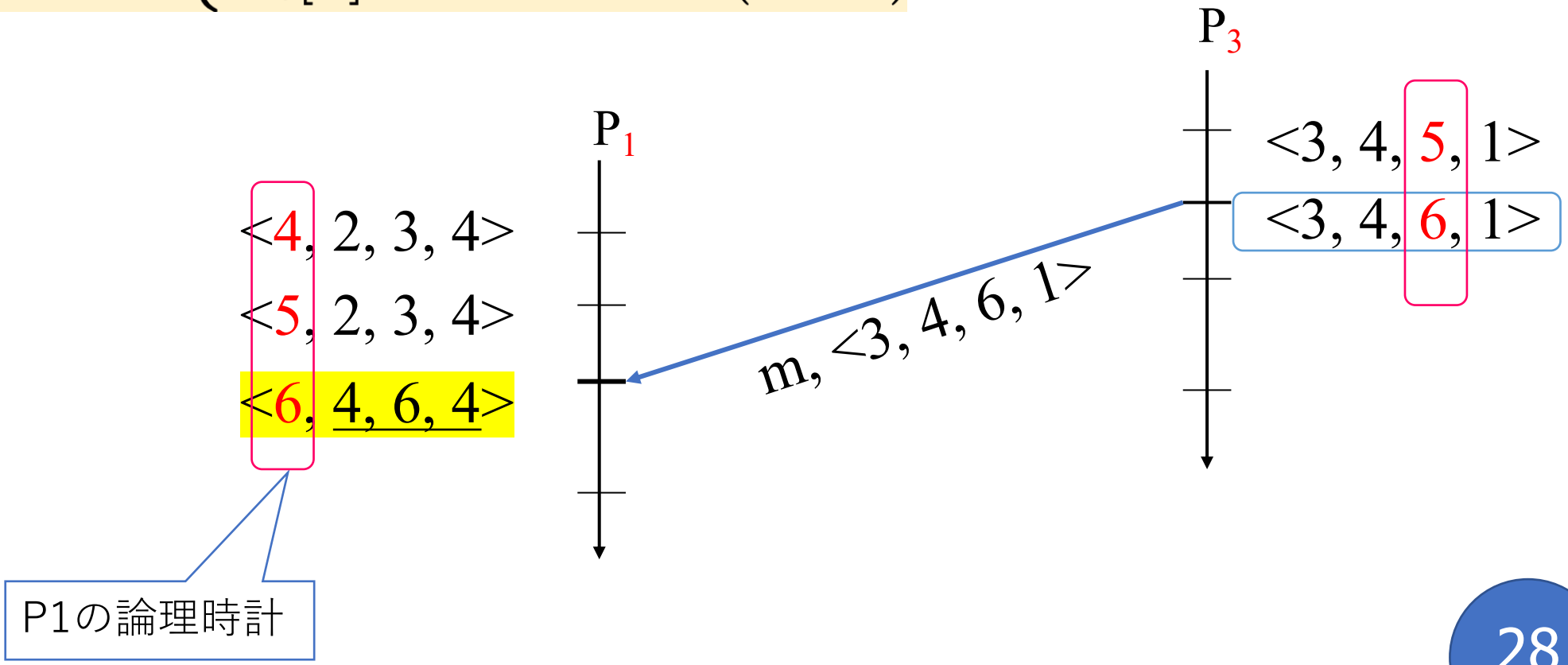
受信プロセスPiの論理ベクトル時間の各要素 $t_i[h]$ について

$$t_i[h] = \begin{cases} \max(t_i[h], t_j[h]) & (h \neq i) \\ t_i[h] + 1 & (h = i) \end{cases}$$

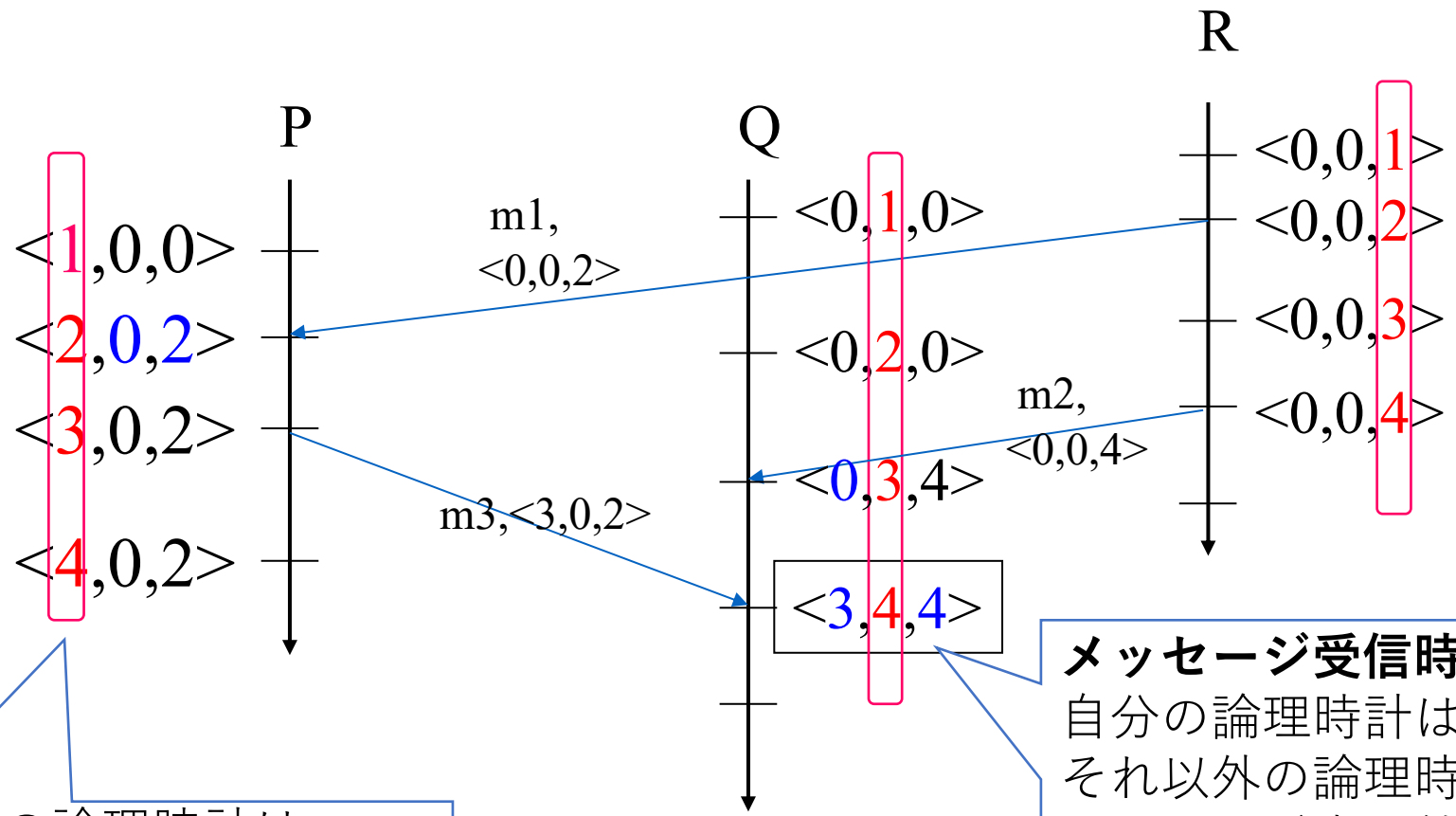
ここで $t_j[h]$ は受信した「送信イベントの論理ベクトル時間」の要素。

論理ベクトル時間の処理(2)

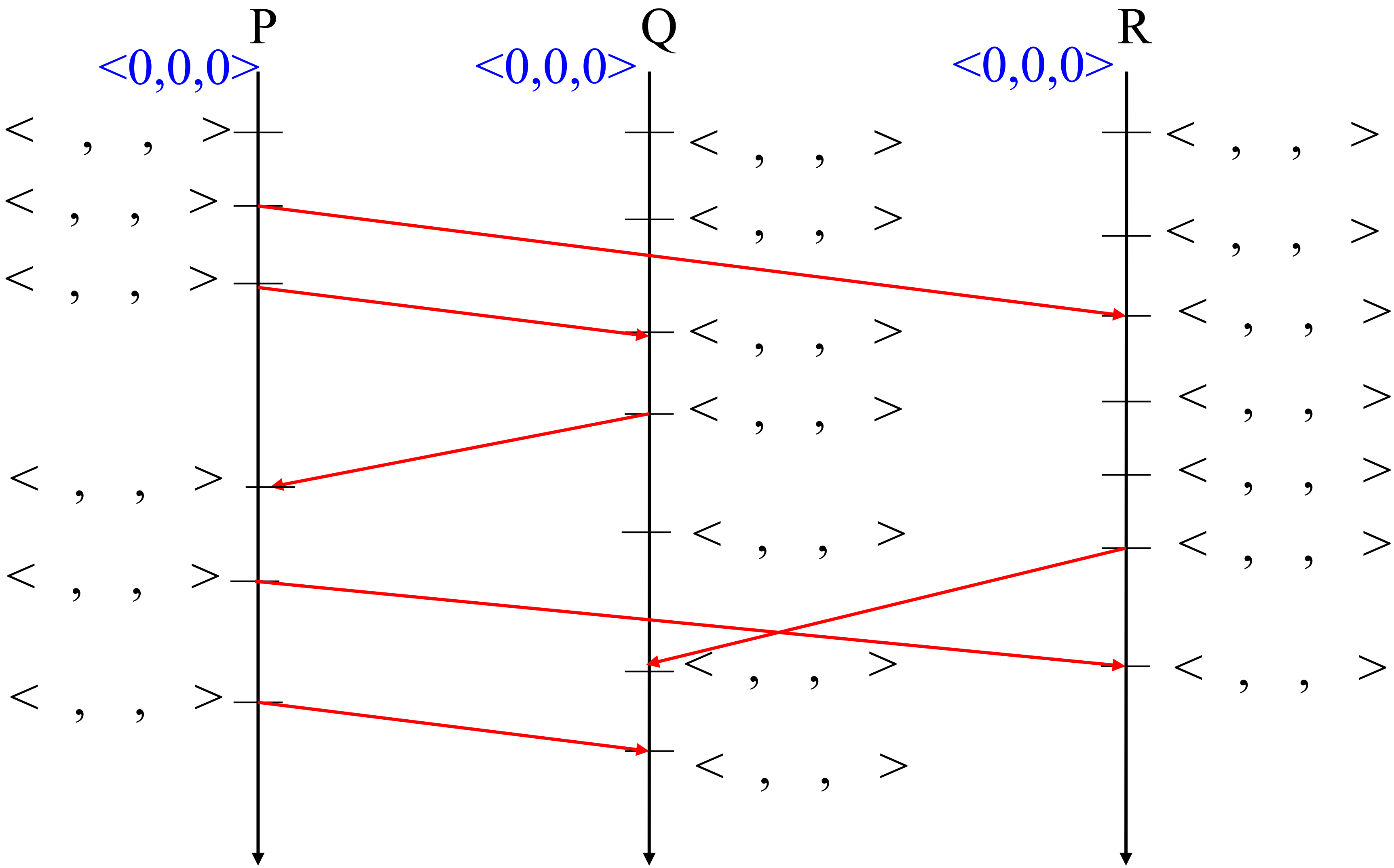
$$t_i[h] = \begin{cases} \max(t_i[h], t_j[h]) & (h \neq i) \\ t_i[h] + 1 & (h = i) \end{cases}$$



論理ベクトル時間の例


P
Q
R
<1,0,0>
<2,0,2>
<3,0,2>
<4,0,2>
m1,
<0,0,2>
<0,1,0>
<0,2,0>
m2,
<0,0,4>
<0,3,4>
m3,<3,0,2>
<3,4,4>
<0,0,1>
<0,0,2>
<0,0,3>
<0,0,4>
自分の論理時計は、
プロセス内のイベントの
たびに +1 する
メッセージ受信時
自分の論理時計は+1
それ以外の論理時計は、
メッセージ内の値と
自分が保持する値の
大きい方

演習：論理ベクトル時間



論理ベクトル時間の大小関係

■2つの論理ベクトル時間
$V=(v_1, v_2, v_3, \ldots, v_n)$ と $V'=(v'_1, v'_2, v'_3, \ldots, v'_n)$ があるとき、

■「$V<V'$ である」とは、
全ての $i \in \{1,2,3,\ldots,n\}$ に対して $v_i \leqq v'_i$ であり、かつ、
ある $i \in \{1,2,3,\ldots,n\}$ に対して $v_i < v'_i$ が成り立つことである。

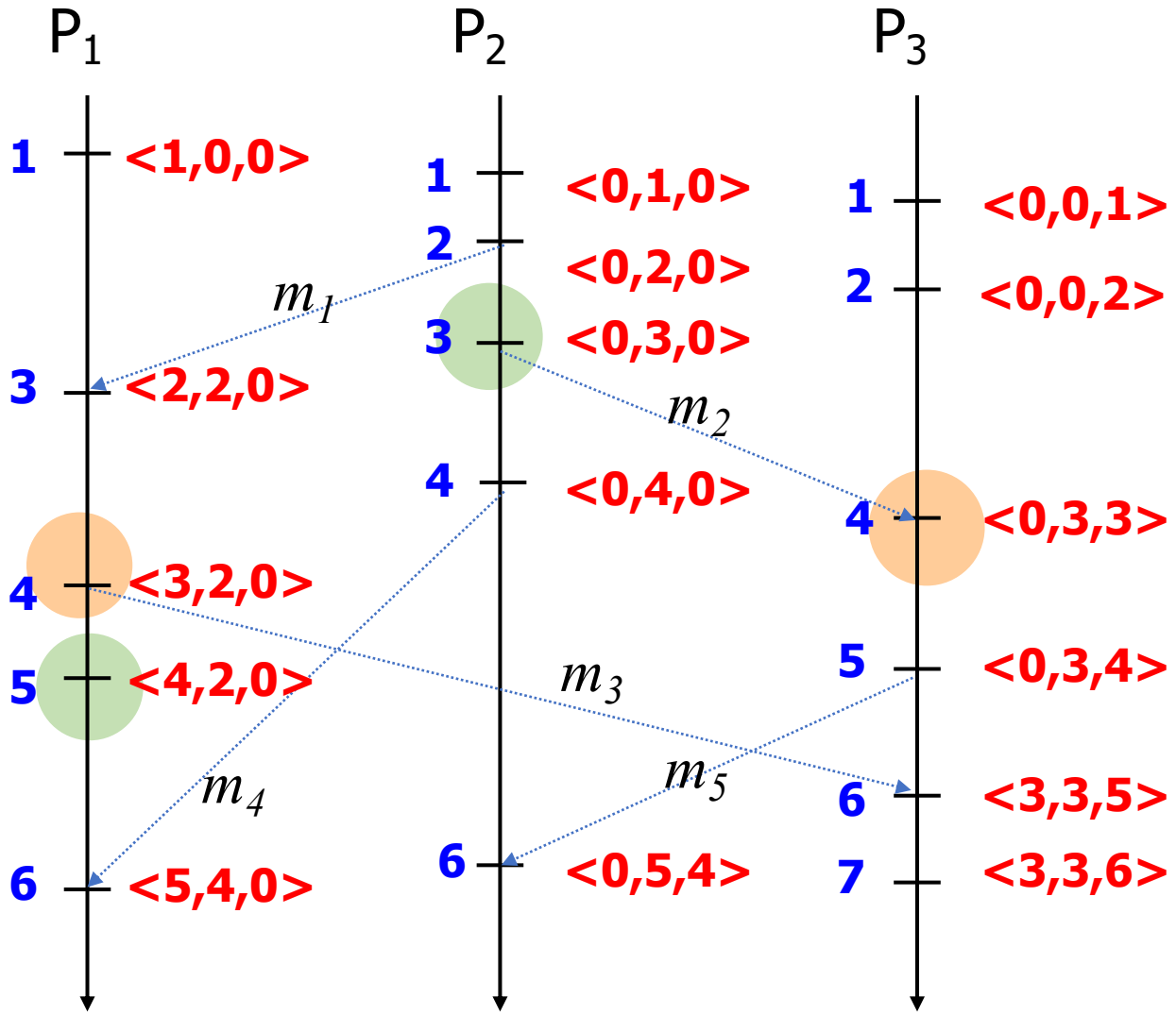
■成り立たない時には、 VとV'の間には大小関係は定義されない。

「<3,4,5,3,2>」 < 「<3,5,5,3,2>」

「<3,5,5,3,2>」 > 「<3,4,5,3,1>」

「<3,4,5,3,2>」 || 「<4,4,4,3,2>」

LamportのTSで問題があった さっきの事例



<0,3,0>と<4,2,0>
大小関係がない。
→並行

<0,3,3>と<3,2,0>
大小関係がない。
→並行

過去イベント、未来イベント、並行イベント

あるイベント e に対して、それ以外の全てのイベントは、
そのイベントeに対する
　過去イベント、未来イベント、並行イベント
のいずれかに分類できる。

■過去イベントの集合

past(e) = { e' | V(e')<V(e) }

■未来イベントの集合

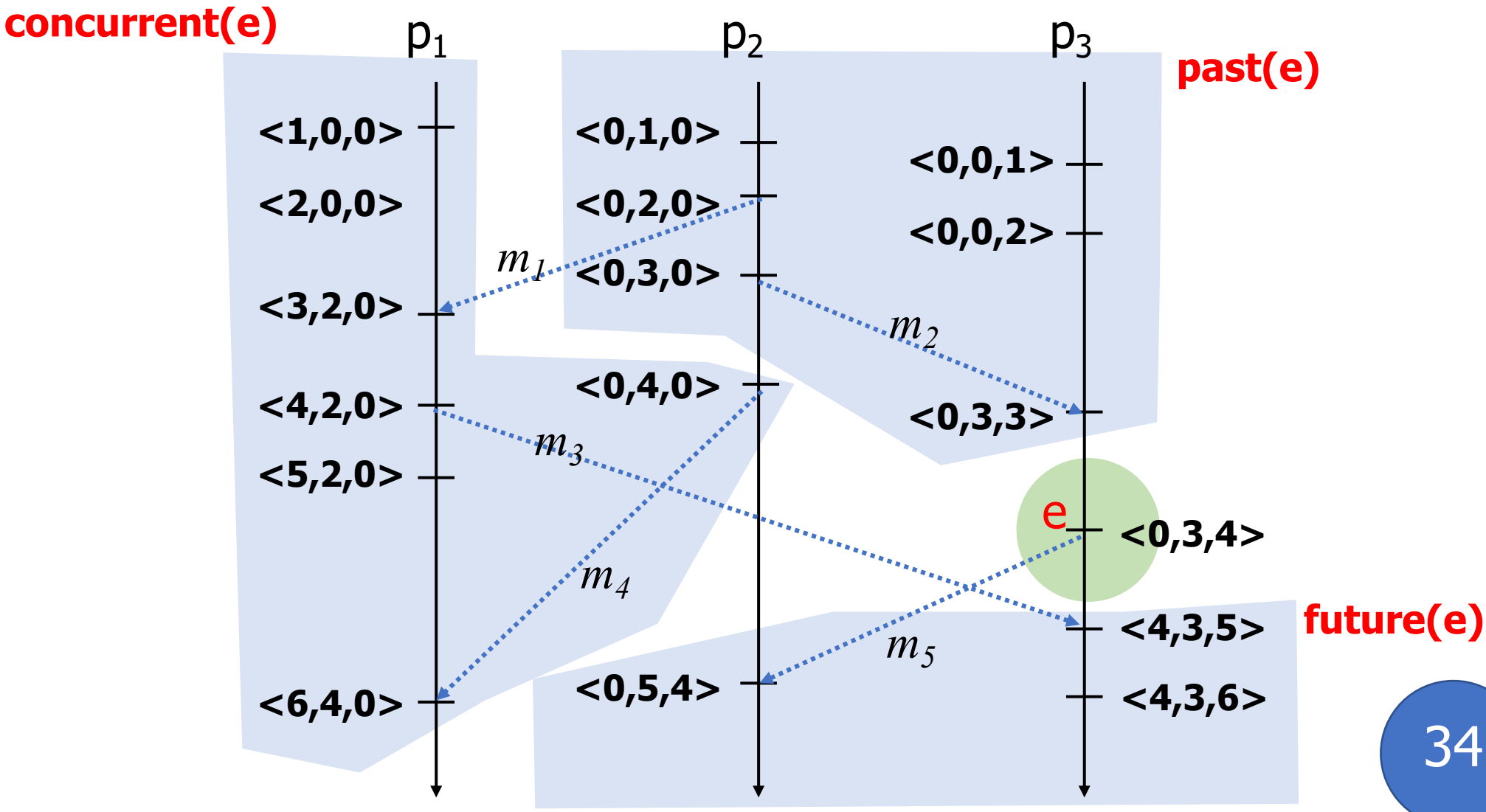
future(e) = { e' | V(e') > V(e) }

■並行イベントの集合

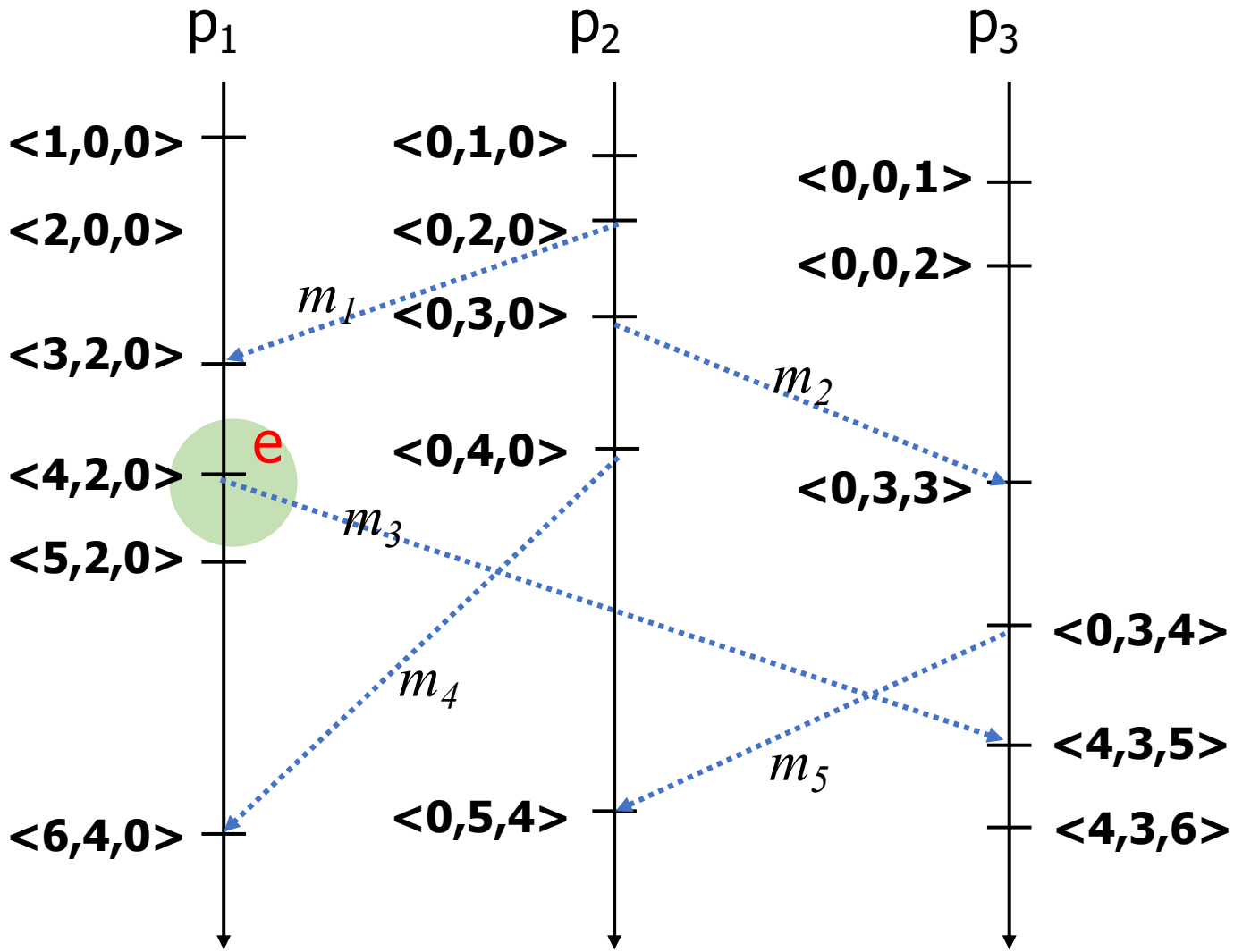
concurrent(e) ={ e' | (V(e') $\not>$ V(e)) ^ (V(e') $\not<$ V(e)) }

ここでV(x)は事象xの論理ベクトル時間。

過去イベント、未来イベント、並行イベント



演習：past(e),future(e),concurrent(e) の範囲を図示せよ



{過去｜未来｜並行}イベントは どう使う？

■**原因の絞り込み**
イベント e を実行中にエラーが見つかった場合、
その原因は past(e)の中にある。

■**汚染範囲の特定**
イベントeの時点でウィルスに感染したことが判明した場合、
そのウィルスの影響を受けた可能性がある処理は、
future(e)の範囲に限られる。
汚染範囲の特定

■**スケジューリングの改善**
concurrent(e)のイベントは、
eと並行して処理することが可能なイベントである。

今日のまとめ

- 事象（イベント）の順序のみに着目
  - 前後と並行

- 論理時計
  - Lamportのタイムスタンプ
    - 問題点：タイムスタンプの値→順序関係　に難がある。
  - 論理ベクトル時間

- 論理ベクトル時間に基づき、あるイベントに対する過去イベント、未来イベント、並行イベントが決まる。
  →分散システム運用上の判断基準