LẬP TRÌNH ĐỒNG THỜI & PHÂN TÁN

BÀI 5: MÔ HÌNH VÀ ĐỒNG HỒ TRONG TÍNH TOÁN PHÂN TÁN

Giảng viên: Lê Nguyễn Tuấn Thành
Email: thanhlnt@tlu.edu.vn

Giới thiệu

- Khi một chương trình phân tán thực thi, một tập các **sự kiện** được tạo ra
- **Tập sự kiện** này và **Mối quan hệ thứ tự**, mối quan hệ trước sau, trên tập sự kiện đó sẽ quy định cách hành xử của một hệ thống phân tán
- Mỗi máy tính trong hệ thống phân tán có đồng hồ riêng

Source: https://cloud.addictivetips.com/wp-content/uploads/2012/07/Clock-grid-Advanced-World-Clock.png

Trong hệ thống phân tán, các sự kiện xảy ra khi nào và thứ tự thực hiện của chúng là gì?

NỘI DUNG

- Mô hình *đã-xảy-ra-trước*
- Cơ chế đồng hồ để lưu vết thứ tự trên tập các sự kiện đã xảy ra
  - *Đồng hồ logic*
  - *Đồng hồ vector*
  - *Đồng hồ phụ-thuộc-trực tiếp*
  - *Đồng hồ ma trận*

Đặc điểm của Hệ thống phân tán (1)

1. **Thường thiếu một đồng hồ chia sẻ**
   - Không thể đồng bộ đồng hồ của các BXL khác nhau do độ trễ của việc truyền thông điệp
   - Hiếm khi sử dụng đồng hồ vật lý để đồng bộ
   - Sử dụng *khái niệm nhân quả* thay cho thời gian vật lý để đồng bộ các sự kiện

Đặc điểm của Hệ thống phân tán (2)

2. **Thiếu bộ nhớ chia sẻ**
   - Không có một BXL nào biết được trạng thái toàn cục của hệ thống phân tán
   - Khó khăn trong việc quan sát một thuộc tính bất kỳ của hệ thống

Đặc điểm của Hệ thống phân tán (3)

3. **Khó phát hiện các nguyên nhân sai lệch**
   - Trong một hệ thống phân tán bất đồng bộ, không thể phân biệt giữa một BXL chậm và một BXL bị lỗi
   - Khó khăn trong việc phát triển các thuật toán cho các *bài toán đồng thuận*, *bài toán bầu cử*,... trong hệ thống phân tán

Hệ thống phân tán: đồng bộ và bất đồng bộ

HT phân tán đồng bộ

- Tốc độ và thời gian thực thi bị giới hạn
- Quá trình truyền thông điệp có độ trễ bị giới hạn
- Thứ tự phân phối thông điệp được đảm bảo (e.g. FIFO)

HT phân tán bất đồng bộ

- Tốc độ và thời gian thực thi không bị giới hạn
- Quá trình truyền thông điệp có độ trễ không bị giới hạn
- Thông điệp truyền đi theo thứ tự ngẫu nhiên

Giả định cho hệ thống phân tán được nghiên cứu

Hệ thống phân tán được nghiên cứu (1)

- Hệ thống phân tán bất đồng bộ
- Một chương trình phân tán sẽ bao gồm:
  - Tập N tiến trình được biểu thị bằng $\{P_1, P_2, \ldots, P_N\}$
  - Tập các *kênh đơn hướng*, mỗi kênh kết nối hai tiến trình
- Topology có thể được xem như là một *đồ thị có hướng*

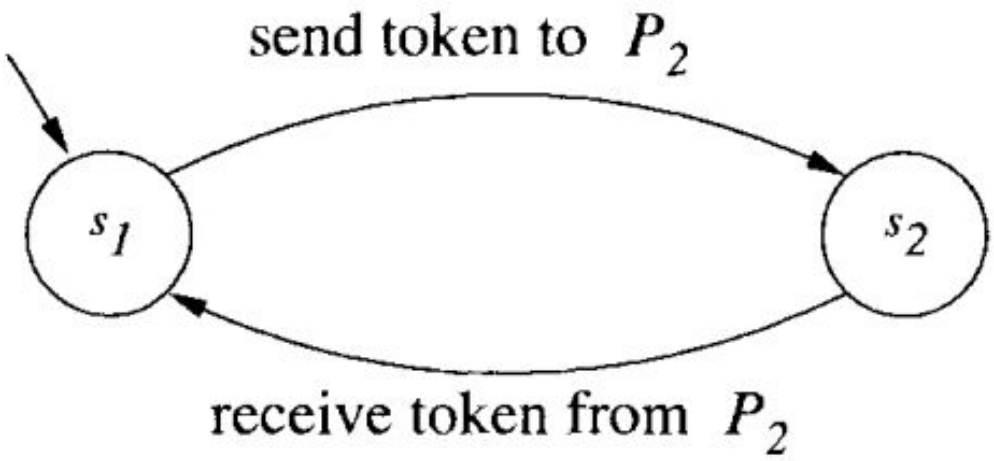
Hệ thống phân tán được nghiên cứu (2)

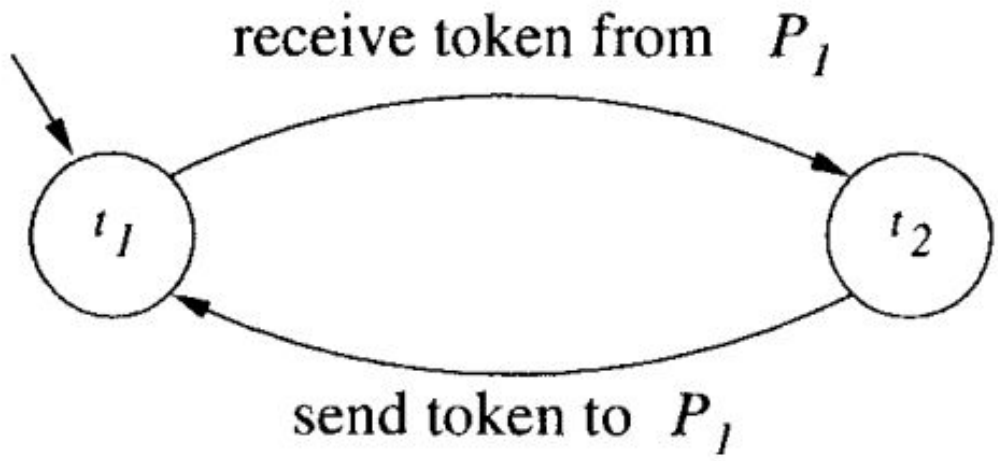
- Một kênh truyền được giả định có bộ đệm vô hạn và không có lỗi trong quá trình truyền thông điệp trên kênh đó
  - Không yêu cầu về thứ tự của các thông điệp
  - Thông điệp gửi trên kênh có thể có độ trễ tùy ý nhưng không thể vô hạn
- **Trạng thái của kênh** tại một điểm được định nghĩa là ***chuỗi các thông điệp được gửi đi trên theo kênh đó***

Hệ thống phân tán được nghiên cứu (3)

- Một tiến trình trong hệ thống phân tán được định nghĩa gồm:
  - *Tập các trạng thái* (e.g. chuỗi các thông điệp gửi)
  - *Tập các sự kiện* (e.g. sự kiện nhận, gửi thông điệp, …)
  - *Điều kiện ban đầu* (e.g. tập con của tập trạng thái)
- Khi một sự kiện xảy ra có thể thay đổi trạng thái của tiến trình và trạng thái của tối đa một kênh trên tiến trình đó

Sơ đồ chuyển trạng thái của hai tiến trình





Mô hình trong tính toán phân tán

Happened-before Model

Mô hình đã-xảy-ra-trước (1)

- Trên từng bộ xử lý, có thể quan sát được thứ tự toàn bộ của các sự kiện xảy ra trên bộ xử lý đó
- Nhưng một bộ xử lý chỉ quan sát được một **thứ tự bộ phận**, hay từng phần, của các sự kiện xảy ra trên các bộ xử lý khác

Mô hình đã-xảy-ra-trước (2)

- Lamport lập luận rằng trong một hệ thống phân tán thực sự thì chỉ có ***một trật tự từng phần***, được gọi là **mối quan hệ *đã-xảy-ra-trước***, có thể được xác định giữa các sự kiện
- Làm sao để xác định thứ tự toàn cục của tập các sự kiện của các tiến trình khác nhau trong hệ thống phân tán?

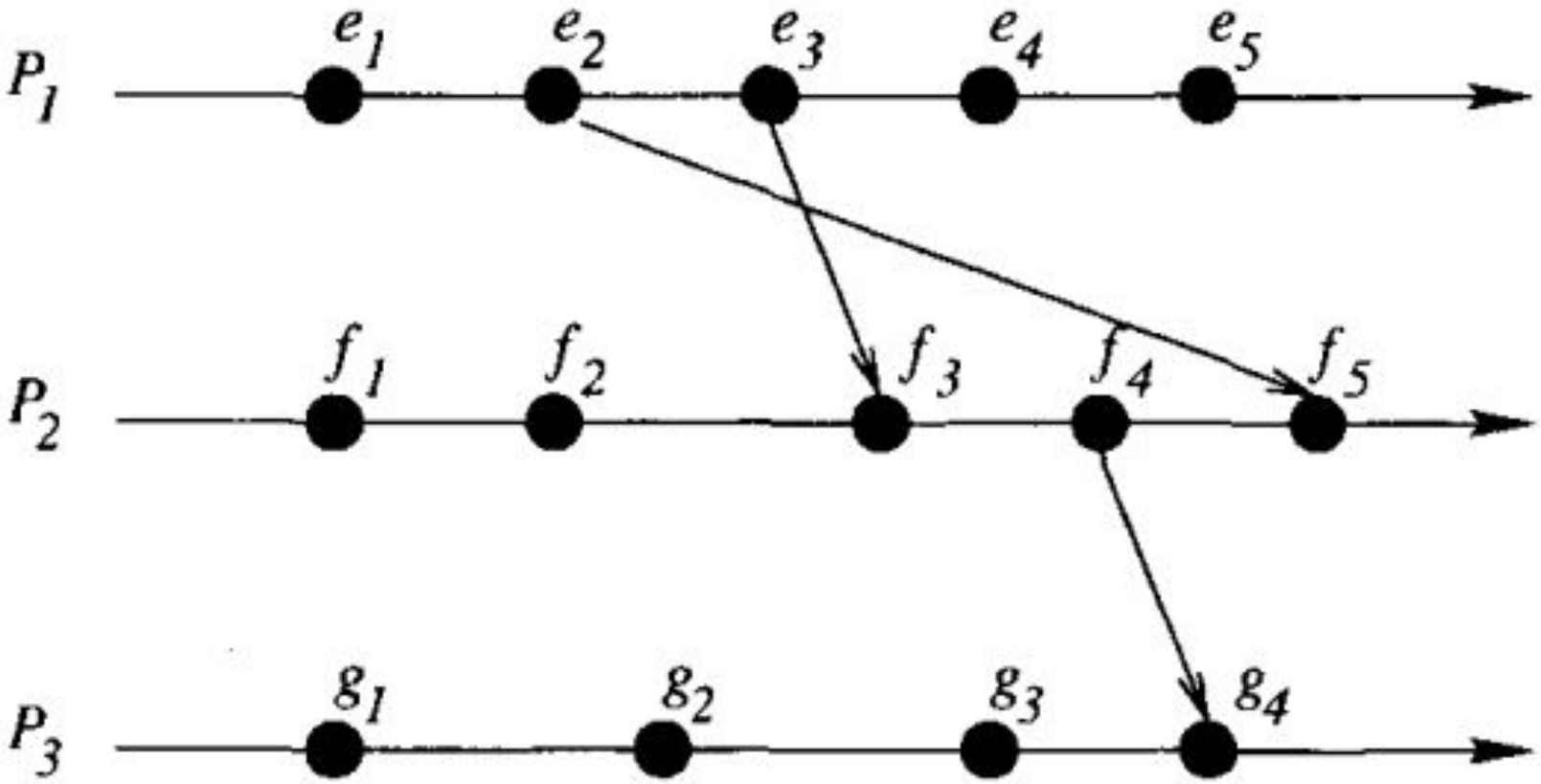
Mô hình đã-xảy-ra-trước (3)

Định nghĩa: Quan hệ *đã-xảy-ra-trước* (→) giữa 2 sự kiện *là mối quan hệ thứ tự nhỏ nhất thỏa mãn các điều kiện sau:*

- *Nếu **e** xảy ra trước **f** trong cùng một tiến trình và thời gian của e nhỏ hơn của f thì **e** → **f***
- *Nếu **e** là sự kiện gửi của một thông điệp và **f** là sự kiện nhận của cùng thông điệp đó (ở tiến trình khác), thì **e** → **f***
- *Nếu tồn tại một sự kiện **g** sao cho **(e** → **g)** và **(g** → **f)**, thì **(e** → **f)***

Mô hình đã-xảy-ra-trước (4)

- Một *tính toán* (*run*) trong mô hình *đã-xảy-ra-trước* được định nghĩa là một cặp ***(E , →)***
  - ***E*** là tập tất cả các sự kiện
  - → là thứ tự từng phần các sự kiện trên *E*



Sơ đồ tiến trình - thời gian hoặc *Sơ đồ đã-xảy-ra-trước*

$e_2 \rightarrow e_4$, $e_3 \rightarrow f_3$, và $e_1 \rightarrow g_4$

Mô hình đã-xảy-ra-trước (5)

- Trong sơ đồ tiến trình-thời gian, $e \rightarrow f$ khi và chỉ khi có một đường dẫn trực tiếp từ sự kiện ***e*** đến sự kiện ***f***.
- Ngoài ra, hai sự kiện ***e*** và ***f*** có thể không liên quan với nhau bởi mối quan hệ *đã-xảy-ra-trước*
- Chúng ta nói rằng ***e*** và ***f*** là *đồng thời* (biểu diễn bằng $e \mid\mid f$) nếu $\neg(e \rightarrow f) \wedge \neg(f \rightarrow e)$
  - Trong ví dụ trước: $e_2 \mid\mid f_2$, và $e_1 \mid\mid g_3$

22

Những cơ chế đồng hồ

Lưu vết mối quan hệ thứ tự thực hiện trên tập các sự kiện

(Lưu dấu thời gian thứ tự thực hiện của các sự kiện)

Đồng hồ logic

Logical Clocks

Đồng hồ logic (1)

- Cơ chế cho phép chúng ta biết được thứ tự toàn cục của các sự kiện ***có thể đã xảy ra*** thay vì thứ tự toàn cục đã thực sự xảy ra
- Đồng hồ logic chỉ đưa ra thứ tự thực hiện giữa các sự kiện
  - Không sử dụng bất kỳ thuộc tính nào khác liên quan tới thời gian vật lý
- Mỗi sự kiện sẽ được gắn với **một số nguyên dương**
  - Số này không liên quan đến thời gian vật lý thực sự của sự kiện đó

Định nghĩa Đồng hồ logic

- Một đồng hồ logic C là một ánh xạ từ tập các sự kiện **E** đến $\mathcal{N}$ (tập các số tự nhiên) với rằng buộc:

$$\forall e, f \in E : e \rightarrow f \Rightarrow C(e) < C(f)$$

Lưu ý: Chúng ta cũng có thể sử dụng trạng thái của tiến trình thay cho sự kiện trong định nghĩa trên, khi đó đồng hồ logic C phải thỏa mãn rằng buộc:

$$\forall s, t \in S : s \rightarrow t \Rightarrow C(s) < C(t)$$

Thuật toán cho Đồng hồ logic (1)

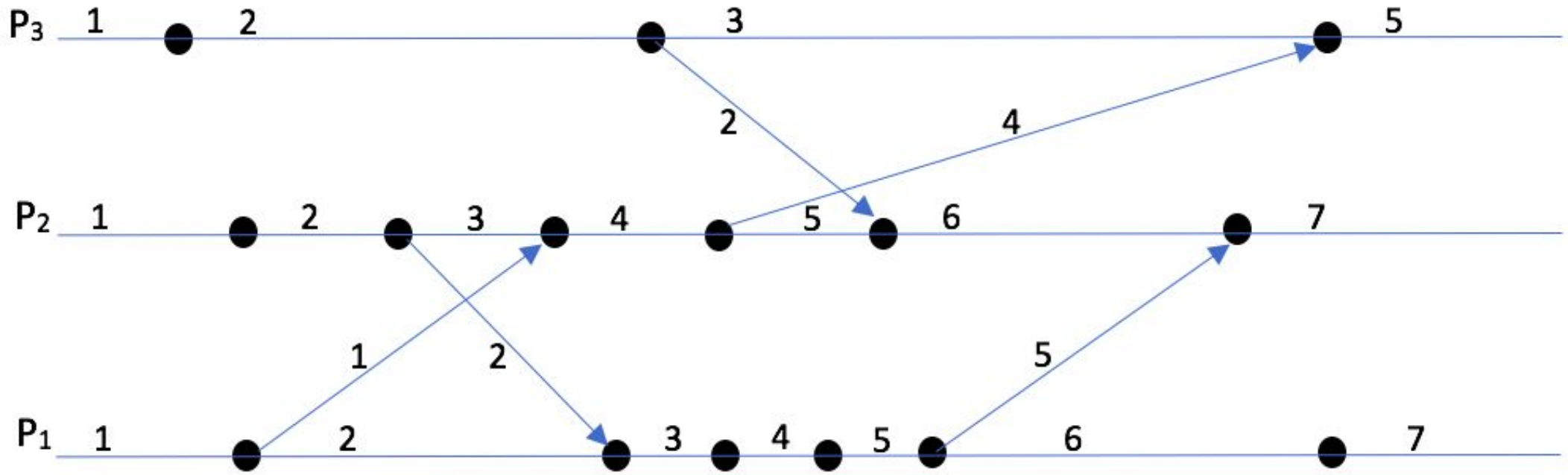
- Thuật toán được miêu tả bằng những điều kiện ban đầu và ba phương thức cho ba kiểu sự kiện:
  1. Sự kiện *gửi thông điệp*
  2. Sự kiện *nhận thông điệp*
  3. Sự kiện *nội bộ*
- Thuật toán sử dụng biến kiểu số nguyên **c** để gán giá trị đồng hồ logic cho sự kiện hoặc trạng thái

```
1  public class LamportClock {
2      int c;
3      public LamportClock() {
4          c = 1;
5      }
6      public int getValue() {
7          return c;
8      }
9      public void tick() { // on internal events
10         c = c + 1;
11     }
12     public void sendAction() {
13         // include c in message
14         c = c + 1;
15     }
16     public void receiveAction(int src, int sentValue) {
17         c = Util.max(c, sentValue) + 1;
18     }
19 }
```

Thuật toán cho Đồng hồ logic (2)

- Khi sự kiện gửi xảy ra, giá trị hiện tại của đồng hồ được gửi cùng với thông điệp và sau đó giá trị này tăng lên 1 đơn vị (dòng 14)
- Khi nhận được một thông điệp trả về (e.g. sự kiện nhận), tiến trình nhận sẽ so sánh để lấy giá trị lớn nhất trong 2 giá trị:
  1. Giá trị đồng hồ hiện tại của nó và
  2. Giá trị đồng hồ mà nó nhận được trong thông điệp (dòng 17)
     - Sau đó, tiến trình nhận tăng giá trị này lên 1 và gán cho giá trị đồng hồ của nó
- Với một sự kiện nội bộ, tiến trình chỉ đơn giản tăng giá trị đồng hồ của nó lên 1 (dòng 10)

Một tính toán với thuật toán đồng hồ Logic



Đồng hồ Vector

Vector Clocks

Đồng hồ Vector

- Đồng hồ logic thỏa mã thuộc tính :

$$\forall s,t \in S : s \rightarrow t \Rightarrow s.c < t.c$$

$$\forall e,f \in E : e \rightarrow f \Rightarrow C(e) < C(f)$$

- Tuy nhiên, điều ngược lại là không đúng !!!
  - **C(s) < C(t)** không ám chỉ rằng $s \rightarrow t$
- Do đó, đồng hồ logic không cung cấp thông tin hoàn chỉnh về mối quan hệ *đã-xảy-ra-trước*
- Cơ chế *đồng hồ vector* cho phép chúng ta nội suy hoàn toàn mối quan hệ *đã-xảy-ra-trước*

Định nghĩa Đồng hồ Vector (1)

Một đồng hồ vector ***v*** *là một ánh xạ từ tập trạng thái* ***S*** *đến* $\mathbf{N}^k$ *(vectơ của các số tự nhiên) với rằng buộc:*

$$\forall s,t : s \rightarrow t \Leftrightarrow s.v < t.v.$$

- *trong đó* ***s.v*** *là vector được gán với trạng thái* ***s***
- *s.v[i] biểu thị giá trị/độ hiểu biết mà tiến trình s.p biết về tiến trình i khi s.p đang ở trong trạng thái s*

Định nghĩa Đồng hồ Vector (2)

- Cho hai vector ***x*** và ***y*** có N chiều, chúng ta định nghĩa phép so sánh sau:

$$x < y = (\forall k : 1 \leq k \leq N : x[k] \leq y[k]) \wedge (\exists j : 1 \leq j \leq N : x[j] < y[j])$$

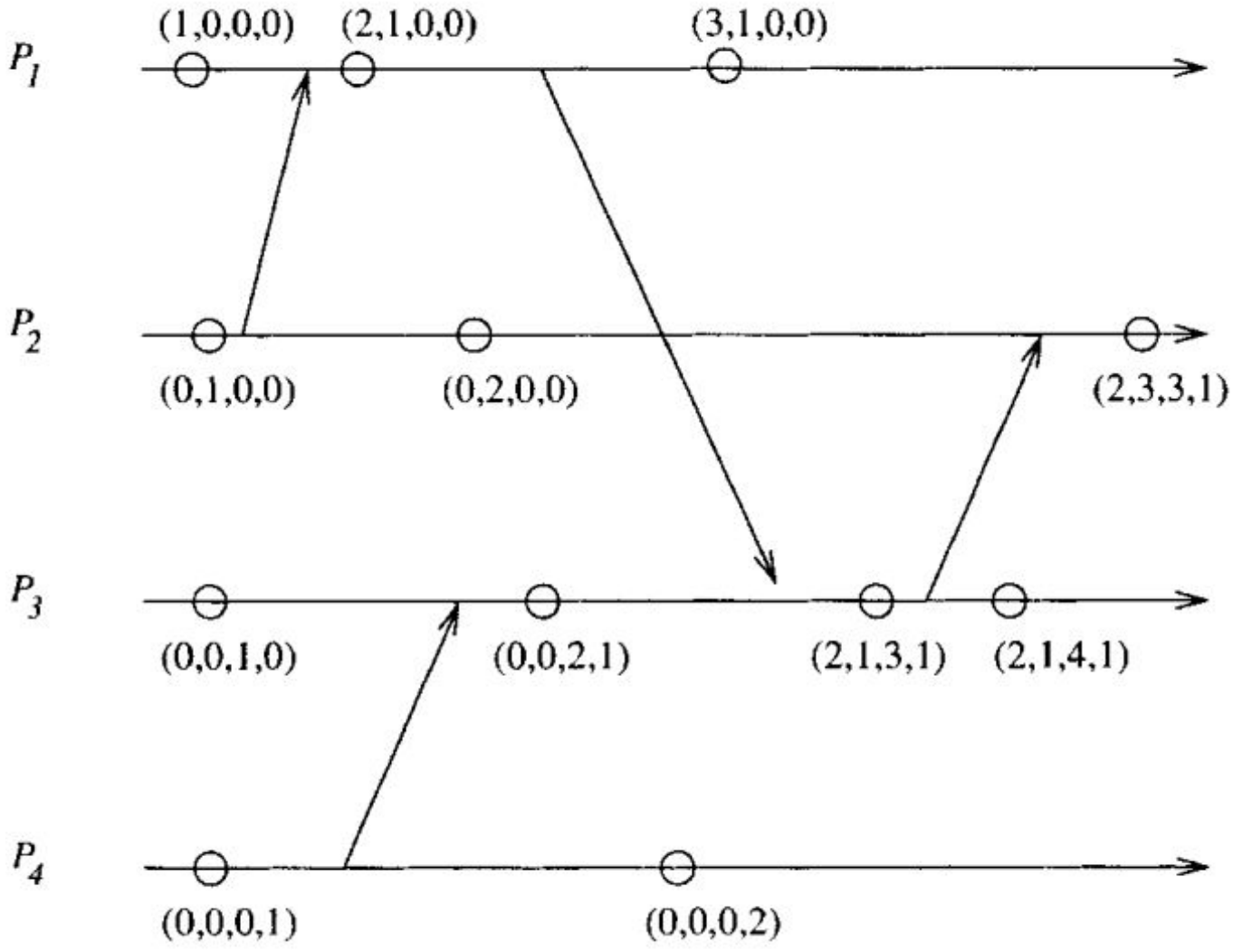
$$x \leq y = (x < y) \vee (x = y)$$

```
1  public class VectorClock {
2      public int[] v;
3      int myId;
4      int N;
5      public VectorClock(int numProc, int id) {
6          myId = id;
7          N = numProc;
8          v = new int[numProc];
9          for (int i = 0; i < N; i++) v[i] = 0;
10         v[myId] = 1;
11     }
12     public void tick(){
13         v[myId]++;
14     }
15     public void sendAction() {
16         //include the vector in the message
17         v[myId]++;
18     }
19     public void receiveAction(int[] sentValue) {
20         for (int i = 0; i < N; i++)
21             v[i] = Util.max(v[i], sentValue[i]);
22         v[myId]++;
23     }
24     public int getValue(int i) {
25         return v[i];
26     }
27     public String toString(){
28         return Util.writeArray(v);
29     }
30 }
```

Thuật toán cho Đồng hồ vector

- Tiến trình tăng giá trị đồng hồ vector của nó sau mỗi sự kiện nội bộ
- Khi gửi thông điệp:
  - Tiến trình gửi sẽ gửi kèm một bản sao đồng hồ vector của nó trong thông điệp
  - Sau đó, tiến trình gửi tăng giá trị thành phần của nó trong vector lên 1 đơn vị
- Khi nhận thông điệp:
  - Tiến trình nhận cập nhật đồng hồ vector bằng cách so sánh và lấy giá trị lớn nhất giữa các thành phần trong đồng hồ vector của nó và thành phần tương ứng trong đồng hồ vector nhận được
  - Sau đó, tiến trình nhận tăng giá trị thành phần của nó trong vector lên 1 đơn vị

Một tính toán với Thuật toán đồng hồ vector



Đồng hồ phụ thuộc trực tiếp

Direct-Dependency Clocks

Đồng hồ phụ thuộc trực tiếp (1)

- Một hạn chế trong thuật toán đồng hồ vector là yêu cầu O(N) số nguyên được gửi đi trong mỗi thông điệp
- Đối với nhiều ứng dụng, có thể sử dụng một phiên bản yếu hơn của đồng hồ vector gọi là đồng hồ *phụ-thuộc-trực-tiếp*
  - Chỉ yêu cầu một số nguyên được gắn thêm vào mỗi thông điệp khi gửi đi

Đồng hồ phụ thuộc trực tiếp (2)

Đồng hồ *phụ-thuộc-trực-tiếp* thỏa mãn rằng buộc sau:

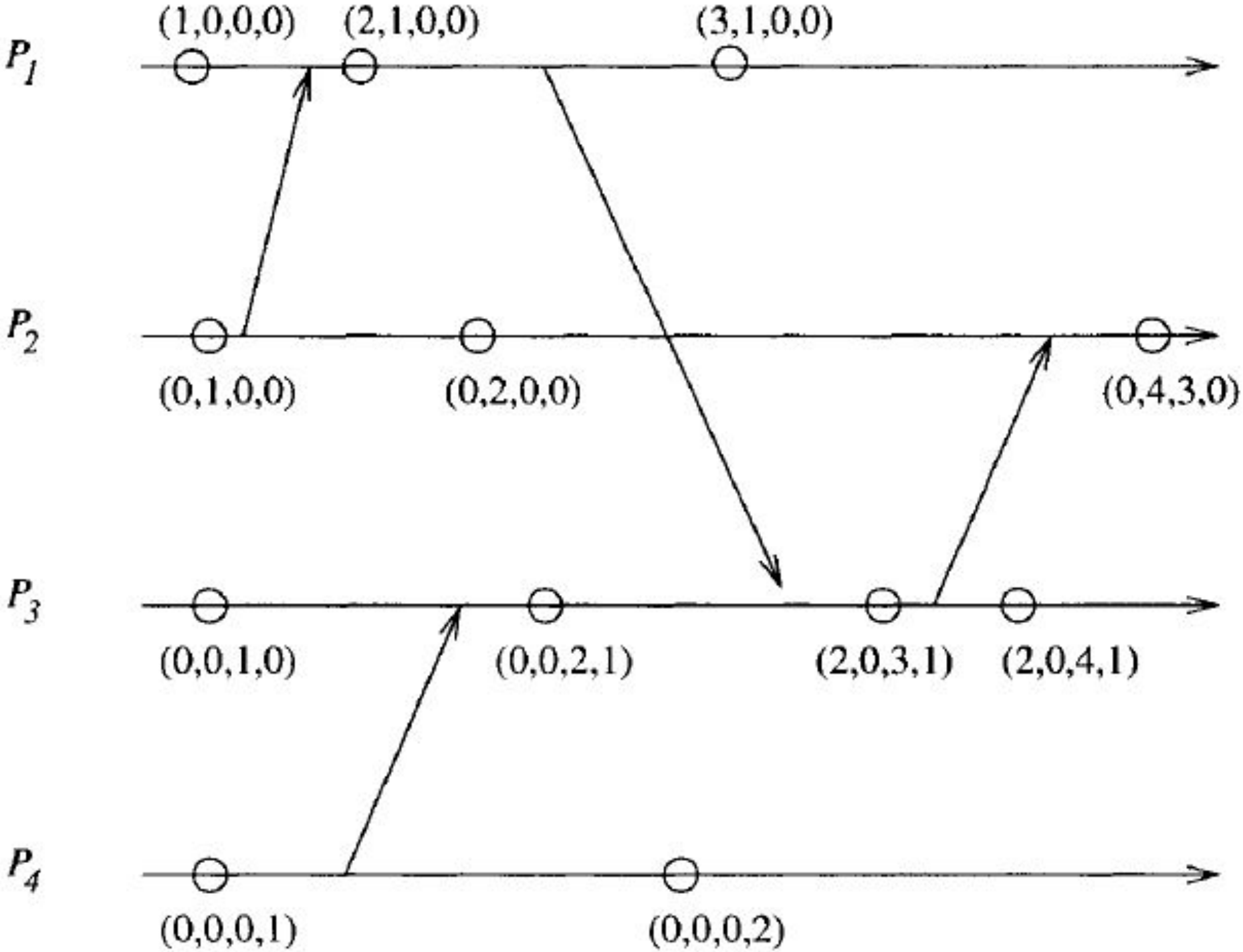
$$\forall s,t : s.p \neq t.p : (s \rightarrow_d t) \Leftrightarrow (s.v[s.p] \leq t.v[s.p])$$

- *Vị trí tương ứng với tiến trình gửi trong vector của tiến trình gửi ở trạng thái s nhỏ hơn hoặc bằng vị trí tương ứng với tiến trình gửi trong vector của tiến trình nhận ở trạng thái t*

Thuật toán cho Đồng hồ phụ thuộc trực tiếp

- Khi gửi thông điệp:
  - Tiến trình gửi **chỉ gửi kèm giá trị của thành phần đồng hồ của nó**, mà không phải là toàn bộ đồng hồ vector
  - Sau khi gửi, tiến trình gửi sẽ tăng giá trị của thành phần đồng hồ đó lên 1 đơn vị
- Khi nhận thông điệp, tiến trình nhận sẽ cập nhật 2 thành phần:
  1. Thành phần đồng hồ của nó
  2. Thành phần đồng hồ của tiến trình gửi
- Với các sự kiện nội bộ, thuật toán làm giống với thuật toán đồng hồ vector

```
public class DirectClock {
    public int[] clock;
    int myId;
    public DirectClock(int numProc, int id) {
        myId = id;
        clock = new int[numProc];
        for (int i = 0; i < numProc; i++) clock[i] = 0;
        clock[myId] = 1;
    }
    public int getValue(int i) {
        return clock[i];
    }
    public void tick() {
        clock[myId]++;
    }
    public void sendAction() {
        // sentValue = clock[myId];
        tick();
    }
    public void receiveAction(int sender, int sentValue) {
        clock[sender] = Util.max(clock[sender], sentValue);
        clock[myId] = Util.max(clock[myId], sentValue) + 1;
    }
}
```


(1,0,0,0)
(2,1,0,0)
(3,1,0,0)
P_1
P_2
(0,1,0,0)
(0,2,0,0)
(0,4,3,0)
P_3
(0,0,1,0)
(0,0,2,1)
(2,0,3,1)
(2,0,4,1)
P_4
(0,0,0,1)
(0,0,0,2)

Đồng hồ ma trận

Matrix Clocks

Đồng hồ ma trận (1)

- Sử dụng một ma trận N x N trong một hệ thống phân tán với N tiến trình.
- Đồng hồ ma trận giúp biểu thị một mức độ biết cao hơn so với đồng hồ vector
  - Giá trị **M[i,j]** của tiến trình k biểu diễn điều mà tiến trình **k** biết về *"độ hiểu biết của tiến trình **i** với tiến trình **j**"*
  - Ví dụ: nếu **s.v[j, s.p] > k** với mọi **i**, thì tiến trình **s.p** có thể kết luận rằng mọi tiến trình khác đã biết trạng thái hiện tại của nó lớn hơn **k**

```
public class MatrixClock {
    int[][] M;
    int myId;
    int N;
    public MatrixClock(int numProc, int id) {
        myId = id;
        N = numProc;
        M = new int[N][N];
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                M[i][j] = 0;
        M[myId][myId] = 1;
    }
    public void tick() {
        M[myId][myId]++;
    }
    public void sendAction() {
        //include the matrix in the message
        M[myId][myId]++;
    }
    public void receiveAction(int[][] W, int srcId) {
        // component-wise maximum of matrices
        for (int i = 0; i < N; i++)
            if (i != myId) {
                for (int j = 0; j < N; j++)
                    M[i][j] = Util.max(M[i][j], W[i][j]);
            }

        // update the vector for this process
        for (int j = 0; j < N; j++)
            M[myId][j] = Util.max(M[myId][j], W[srcId][j]);

        M[myId][myId]++;
    }
    public int getValue(int i, int j) {
        return M[i][j];
    }
}
```

Thuật toán cho Đồng hồ ma trận

- Nếu ta chỉ tập trung vào hàng ***myId*** cho tiến trình P_{myId} thuật toán này sẽ trở thành thuật toán đồng hồ vector
- Khi gửi thông điệp, tiến trình gửi sẽ gửi kèm toàn bộ ma trận
- Khi nhận thông điệp, tiến trình nhận sẽ cập nhật ma trận:
  - Bước 1: cập nhật giá trị tại các hàng khác *myId*
  - Bước 2: cập nhật giá trị tại hàng *myId* với ma trận *W* nhận được từ tiến trình *srcId*, chúng ta chỉ sử dụng hàng *srcId* của ma trận *W* để cập nhật hàng *myId*

Tài liệu tham khảo

- *Concurrent and Distributed Computing in Java*, Vijay K. Garg, University of Texas, John Wiley & Sons, 2005
- Tham khảo:
  - *Principles of Concurrent and Distributed Programming*, M. Ben-Ari, Second edition, 2006
  - *Foundations of Multithreaded, Parallel, and Distributed Programming*, Gregory R. Andrews, University of Arizona, Addison-Wesley, 2000
  - *The SR Programming Language: Concurrency in Practice*, Benjamin/Cummings, 1993
  - *Xử lý song song và phân tán*, Đoàn văn Ban, Nguyễn Mậu Hân, Nhà xuất bản Khoa học và Kỹ thuật, 2009