Distributed System

Genap 2011/2012

10

Koordinasi Antar Proses

Dahlia Widhyaestoeti, S.Kom
dahlia.widhyaestoeti@gmail.com
dahlia74march.wordpress.com

Jika ada lebih dari satu proses yang siap running, maka System Operasi akan menentukan salah satu proses untuk running lebih dulu.

Untuk menjadwalan proses, perlu kreteria :

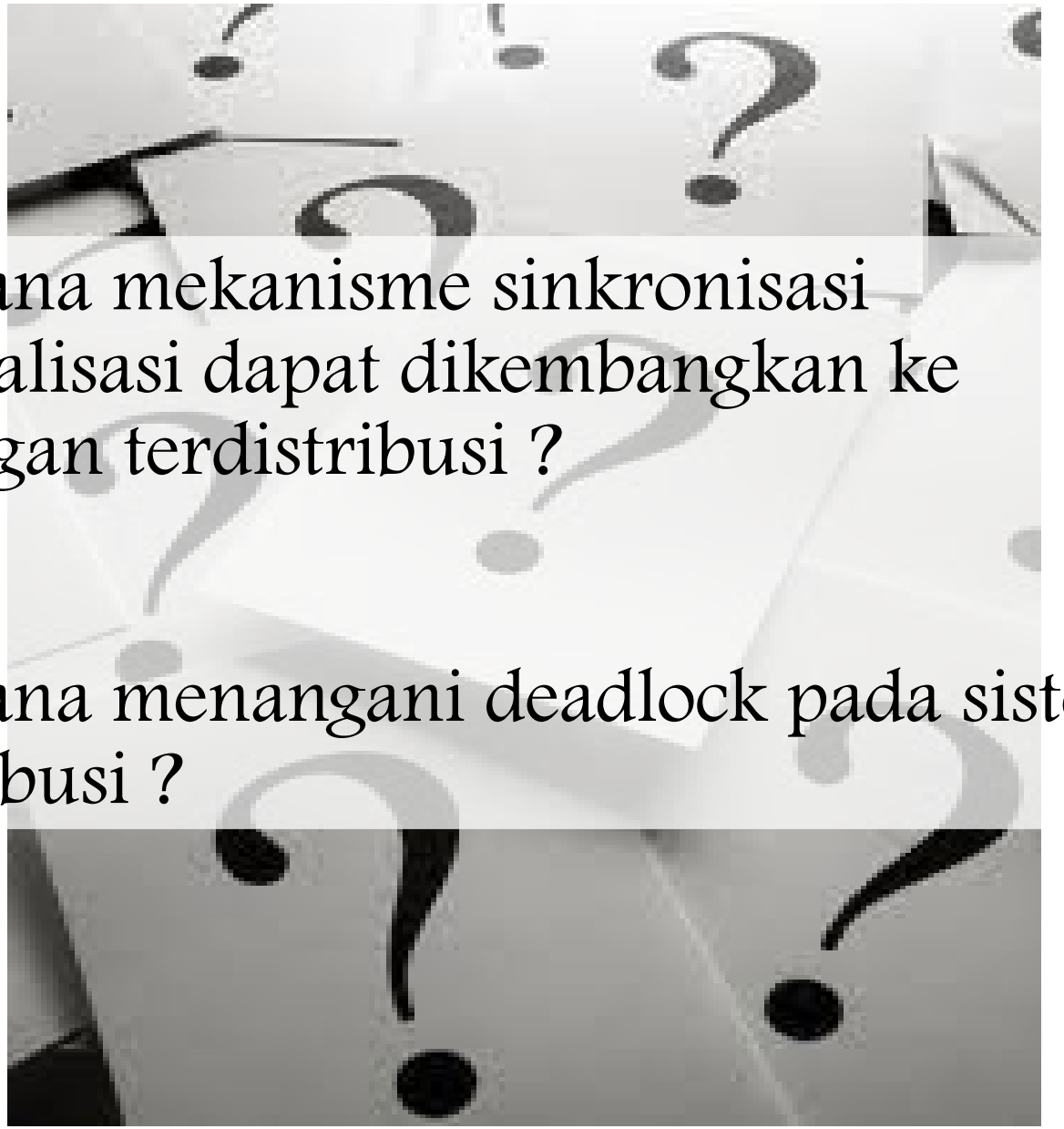
- Fairness
- Efisiensi
- Respontime
- Turn arround time
- Through put

Dead Lock

Suatu kondisi dimana 2 proses/lebih tidak dapat meneruskan eksekusinya.

Penyebab :

- Mutual Exclusion
- Hold & wait
- No Preemption
- Circular wait



Bagaimana mekanisme sinkronisasi tersentralisasi dapat dikembangkan ke lingkungan terdistribusi ?

Bagaimana menangani deadlock pada sistem terdistribusi ?

1. EVENT ORDERING

Sistem tersentralisasi :

- ➔ Selalu mungkin menentukan urutan kejadian, sebab hanya terdapat satu memory dan clock
- ➔ Sejumlah aplikasi sangat menekankan urutan, misal : alokasi resource resource dapat dipakai setelah resource tsb dipesan dan dijamin bebas.

Sistem terdistribusi :

- Memori & clock tdk tunggal
- Tdk mungkin menyatakan urutan dua kejadian
- Hanya dpt ditentukan partial ordering (urutan bagian) relasi Happened-Before

1.1. Relasi Happened - Before

Proses-proses sekuensial → semua kejadian dijalankan oleh satu pemroses → total ordering

Hukum sebab-akibat : suatu pesan dapat diterima setelah pesan tersebut dikirim

Simbol relasi happened-before : →

Sejumlah kejadian sebagai berikut :

- Jika kejadian A & B berada dlm satu proses dan A dieksekusi sebelum B, maka ditulis A→B
- Jika A adalah kejadian pengiriman pesan (sending) oleh sebuah proses dan B adalah sebuah kejadian penerima pesan (receiving) oleh proses lain, maka ditulis A → B
  (sending dilakukan sebelum receiving)
- Jika A → B dan B→ C, maka A → C

Karena suatu event tidak dapat terjadi sebelum dirinya sendiri terjadi, maka relasi happened-before tidak bersifat refleksi (dipasangkan ke dirinya sendiri)

Jika 2 event A & B tidak memenuhi Relasi R
(A tidak terjadi sebelum B dan B tidak terjadi sebelum A), maka kedua event dieksekusi bersamaan (concurent) kedua event tidak dapat saling mempengaruhi.

(A B : *ada kemungkinan A mempengaruhi B*)

Konkurensi (concurency) & happened-before dapat dijelaskan dengan baik melalui diagram *space –time* sebagai berikut:

- Arah horisontal menyatakan space (proses berbeda)
- Arah vertikal menyatakan time
- Garis vertikal menunjukkan processor (Huruf besar) dan lingkaran kecil menunjukkan event (huruf kecil)
- Garis gelombang () menunjukan pengiriman pesan / lintasan (path)
- 2 event konkuren jika tidak ada path antara keduanya.

Contoh :
sejumlah event direlasikan dgn happened-before.

p1→q2

r0→q4

q3→r1

p1→q4 (tentu : p1 → q2 dan q2 → q4)

dan sejumlah event konkuren dari sistem adalah :

q0 dan p2

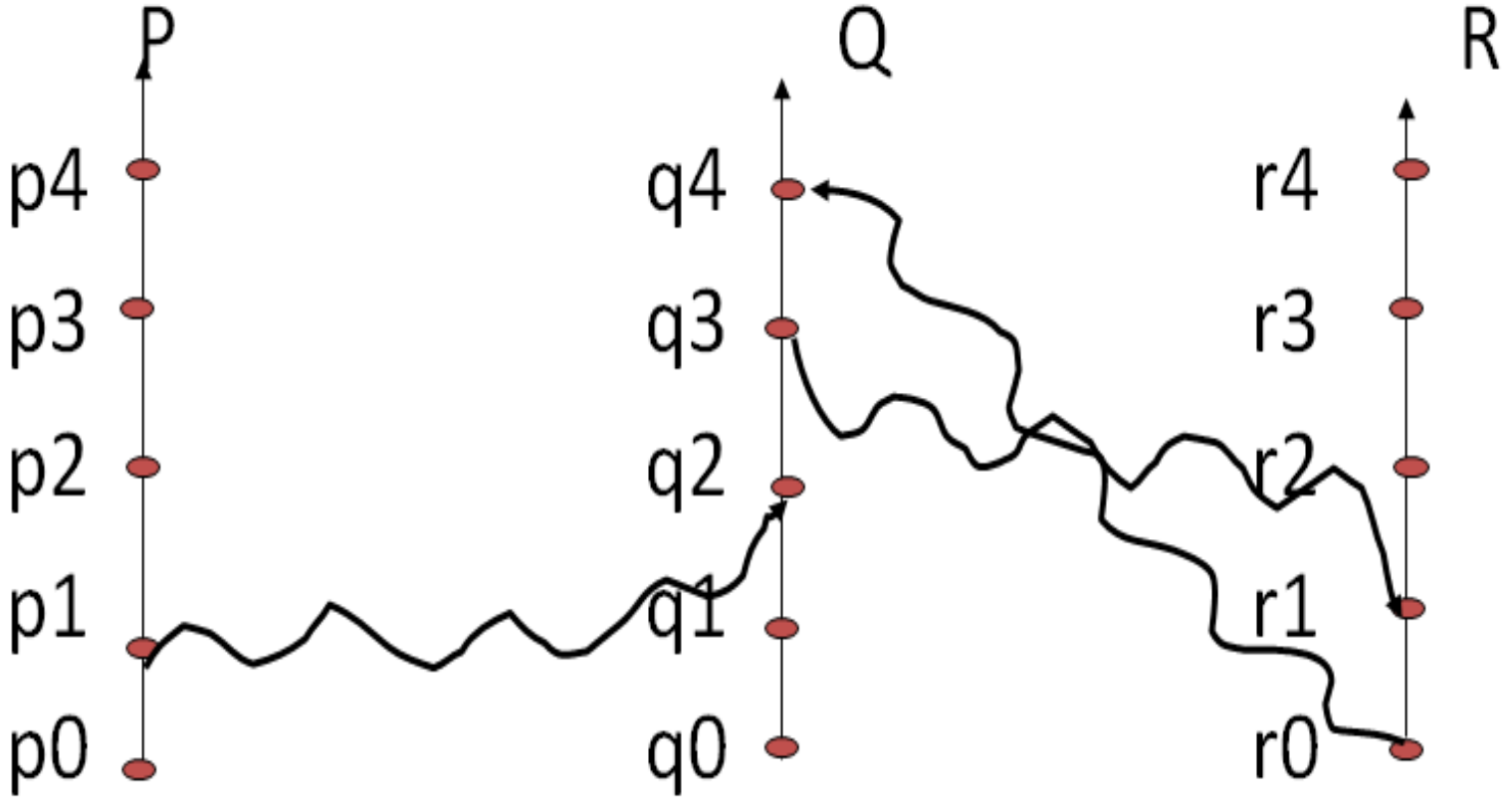
r0 dan q3

r0 dan p3

q3 dan p3

(event p pada processor P, q pada Q dan r pada R)

Diagram space-time dapat digambarkan sebagai berikut:



Tidak dapat ditentukan dari 2 event yang konkuren (q0 dan p2) mana yang terjadi dahulu. Namun, karena satu sama lain tidak saling mempengaruhi, maka tidaklah penting untuk mengetahui mana yang dahulu terjadi.

1.2. Implementasi

Untuk menentukan event A terjadi sebelum event B diperlukan sebuah clock umum atau suatu set pensinkron clock yang perfect.

Pada sentralisasi ini bisa dilakukan, namun pada terdistribusi kedua hal tersebut tidak mungkin dilakukan.

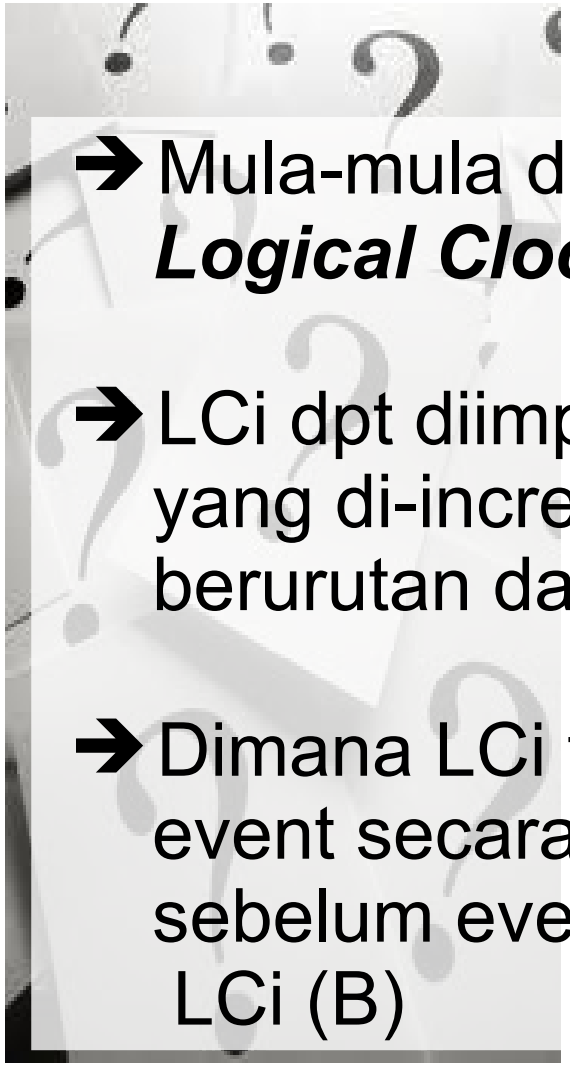
Sehingga diperlukan definisi khusus dari relasi Happened-before tanpa menggunakan clock physics.

Dikenalkan konsep Timestamp;

Setiap event sistem diasosiasikan dengan Timestamp, kemudian didefinisikan ***Global Ordering Requirement (GOR)***

Untuk setiap pasangan event A & B,
jika A → B, maka Timestamp A < Timestamp B

Bagaimana menerapkan GOR dlm lingkungan terdistribusi?



- ➔ Mula-mula didefinisikan dlm tiap proses Pi suatu ***Logical Clock (LCi)***.
- ➔ LCi dpt diimplementasikan sbg counter sederhana yang di-increment-kan antara 2 event tereksekusi berurutan dalam suatu proses.
- ➔ Dimana LCi tersebut dapat menunjukan nomer event secara unik, sehingga jika event terjadi sebelum event B dalam proses Pi, maka LCi (A) < LCi (B)

2. MUTUAL EXCLUSION

Ada n proses yang masing-masing terletak pada prosesor yang berbeda. Pemetaan one to one terjadi antar proses & processor.

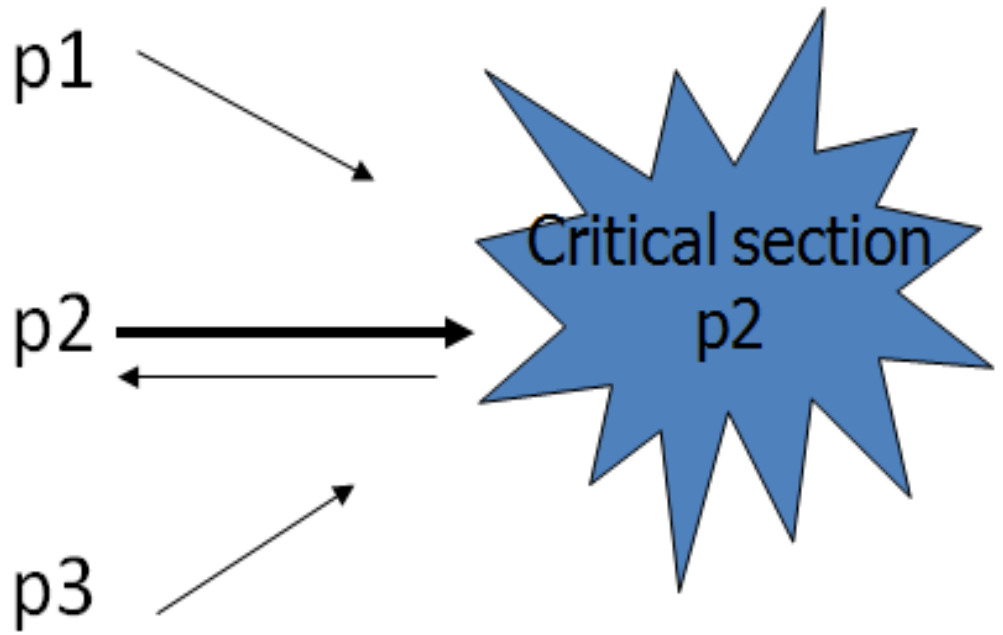
Bagaimana implementasi mutual exclusion pada sistem terdistribusi ?

Ada algoritma :

- *Centralized Approach (CA)*
- *Fully distributed Approach (FDA)*
- *Token Parsing Approach (TPA)*

2.1. CENTRALIZED APPROACH

Dalam CENTRALIZED APPROACH untuk menentukan mutual exclusion, salah satu proses akan dipilih untuk mengkoordinasikan masuk ke critical section.



- tiap proses yg menginginkan mutual exclusion mengirim pesan **request** ke koordinator, dan baru dpt masuk ke c.s setelah menerima jawaban dari koordinator
- Setelah keluar dari c.s, proses tsb akan mengirim pesan **release** ke kordinator dan meneruskan eksekusinya.

Dalam pendekatan ini suatu Graf global dibentuk dengan menggabungkan (union) semua Graf Lokal. Graf ini dimaintenance oleh sebuah proses single sebagai koordinator.

Karena ada delay komunikasi dlm sistem, maka dibedakan dua tipe graf wait-for :

- ***Real Graf*** :
  Keadaan real, tetapi tidak dapat mengetahui status sistem pada suatu saat
- ***Contructed Graf*** :
  Suatu pendekatan yang dibangkitkan oleh koordinator pada saat algoritma dieksekusi

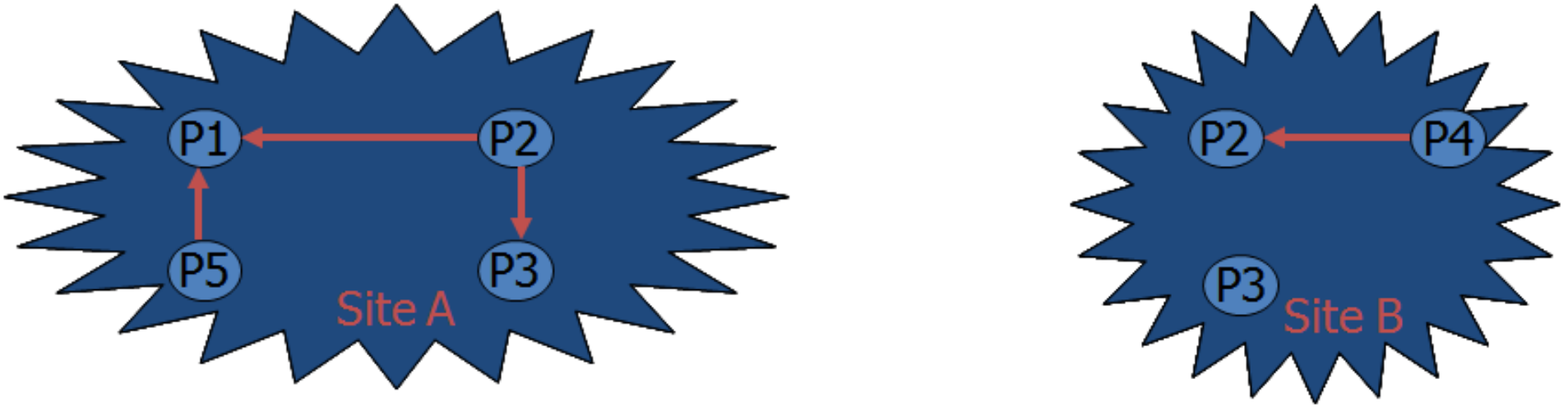
Ada 3 option berbeda ketika graf dikontruksikan:

a) Bila ada edge baru disisipkan/dibuang pada salah satu graf lokal
b) Bila terjadi sejumlah perubahan pada graf
c) Bila koordinator ingin mendptkan algoritma deteksi siklus (penanganan deadlock)

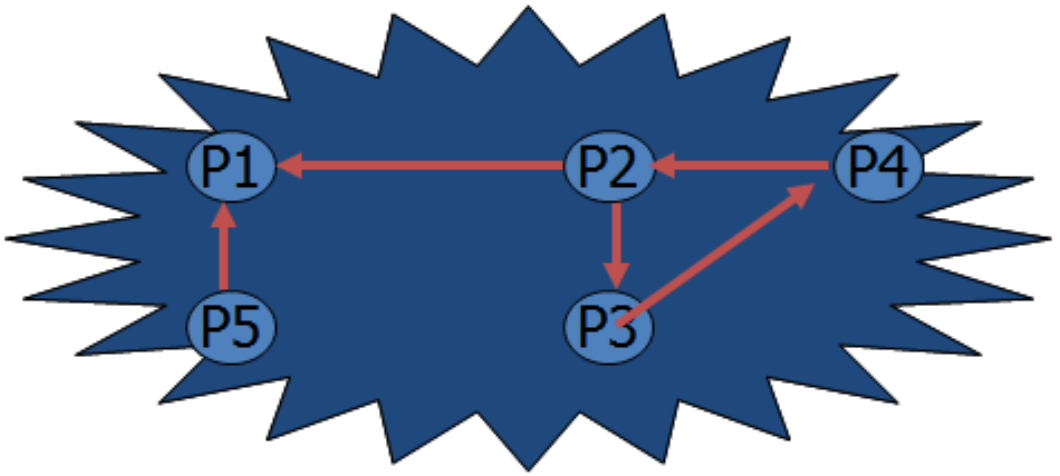
OPTION (a)

Bila sebuah edge disisipkan/dibuang pada graf lokal, maka site lokal tersebut juga harus mengirim pesan ke koordinator untu memodifikasi. Dan ketika koordinator menerima pesan tersebut, koordinator mengupdate graf lokal.

Misal : ada 2 site lokal



Graf Global

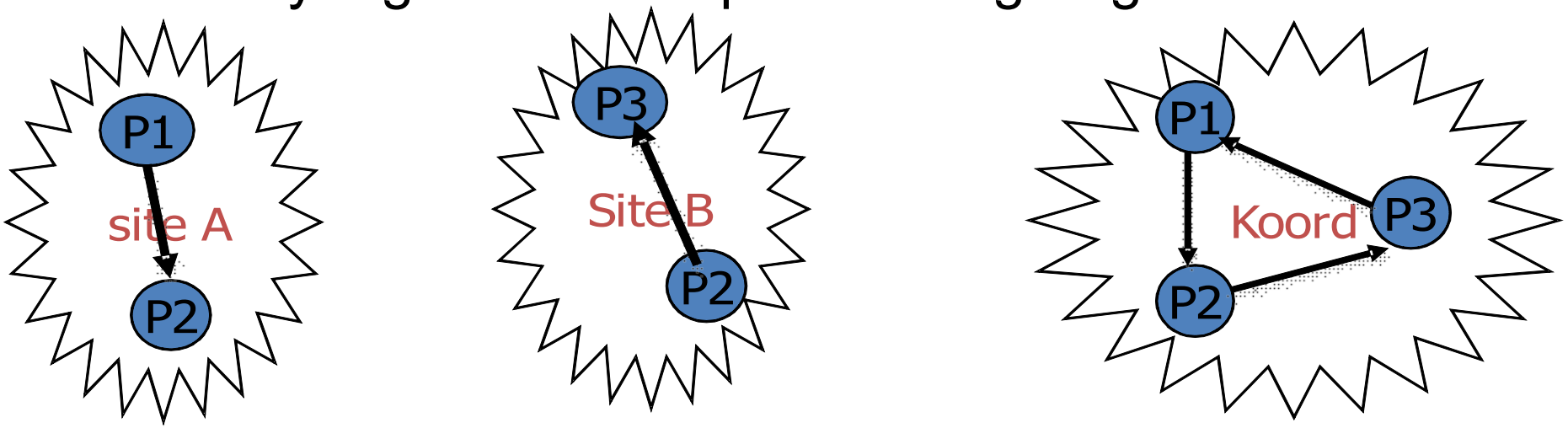


Bila algoritma *DEADLOCK DETECTION (D-D)* dijalankan, maka koordinator akan melakukan searching pada graf global.
Dan bila ditemukan siklus , maka :

- Dipilih victim untuk ***rolled back***
- Koordinator mencatat semua site yang sebagian proses nya di-victim

Rolled back tidak perlu terjadi bila:

1. Siklus yang salah terdapat dalam graf global



- Jika di site A, P2 melepaskan resource yang dipegang, maka edge P1 → P2 akan dihapus
- Jika di site B, P2 request resiurce yang dipegang P3, maka edge P2 →P3 akan dibuat
- Jika pesan insert P2 → P3 dari site B tiba sebelum pesan delete P1 → P2 dari site A, maka koordinator akan menemukan siklus yang salah:

  P1 → P2 → P3 → P1.

  Recovery deadlock dimulai, padahal tidak terjadi Deadlock

2. Suatu Deadlock benar-benar terjadi dan victim telah ditetapkan, tetapi pada waktu yang sama salah satu proses di-abort

Misal :
Site A akan meng-abort P2. Pada waktu yang sama koordinator menemukan siklus pada graf global dan memutuskan P3 akan divictim.

Maka P2 & P3 sekarang di *rolled back*, tetapi sesungguhnya hanya P2 yang perlu di *rolled back*

2.2. FULLY DISTRIBUTED APPROACH

Pendistribusian Decision – Making ke seluruhan sistem sukar & kompleks.

Dikenalkan algoritma berbasis “Event Ordering “, sebagai:

- ketika suatu proses Pi ingin masuk ke critical section,
- maka ia akan meng-create timestamp baru (Ts) dan mengirim pesan request (Pi,Ts) ke semua proses lain dalam sistem (termasuk dirinya sendiri)

- Pada penerimaan pesan request, sebuah proses dapat :
  - ➔ segera menjawab kembali ke Pi
  - ➔ menunda jawaban (karena masih berada di dalam critical section)

- Proses yg telah menerima pesan jawaban dari semua proses lain dlm sistem dpt memasuki critical section mengantrikan request yang datang dan menahan mereka.

- Setelah keluar dari critical section, proses mengirim pesan jawaban kepada semua proses yang requestnya ditahan

Keputusan suatu proses Pi menjawab langsung atau menahan dahulu request (Pj,Ts) didasarkan pada 3 faktor :

1. Jika proses Pi dalam critical section , maka ia akan menahan jawaban ke Pj
2. Jika proses Pi tidak dalam critical section, maka ia akan segera menjawab ke Pj
3. Jika proses Pi ingin masuk ke critical section, tetapi belum dapat masuk, maka ia akan membandingkan timestamp request miliknya dengan timestamp request yang datang dari Pj.
4. jika Ts miliknya > Ts (Pj) , maka ia segera memberi jawaban ke Pj.

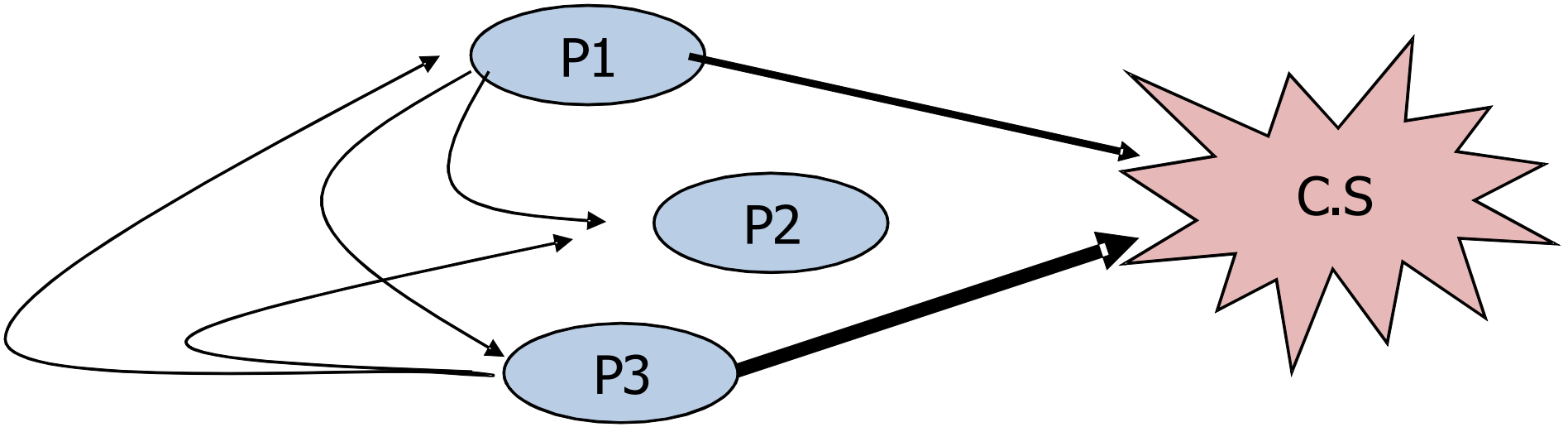
Algoritma ini dapat berhasil dengan syarat :

- Mutual exclusion dapat ditentukan
- Dijamin bebas dead-lock
- Dijamin bebas starvation (masuk critical section dijadwalkan berdasarkan urutan timestamp, dimana urutan timestamp menjamin FIFO)
- Jumlah pesan per critical section entry = 2x (n-1). Suatu jumlah minimum yang diperoleh per critical section entry, bila proses-proses bekerja independen dan konkuren

Cara kerja algoritma
FULLY DISTRIBUTED APPROACH :

Sistem terdiri dari 3 proses P1, P2, P3.

- P1 dan P3 ingin masuk ke critical section mereka.
- P1 kemudian mengirim pesan request (P1,Ts=10) ke proses P2 dan P3.
- Sementara itu P3 mengirim pesan request (P3,Ts=4) ke P1 dan P2



Skenario ini memerlukan partisipasi dari semua proses dalam sistem. Pendekatan ini mempunyai 2 konsekuensi :

- Proses perlu tahu identitas semua proses lain dalam sistem. Jika ada proses baru bergabung dengan group proses yang berpartisipasi dalam algoritma mutual exclusion, maka perlu langkah :
  - ➔ Proses harus menerima semua nama proses lain dalam group
  - ➔ Nama proses baru harus di distribusikan ke semua proses lain dalam group

- Jika salah satu proses fail, skema keseluruhan collaps
  Dapat diatasi dengan memonitor secara kontinyu status semua proses dalam sistem
  - ➔ Jika sebuah proses fail, maka semua proses yang lain diberitahu agar tidak mengirim pesan ke proses yang fail tersebut
  - ➔ Jika proses direcover, ia hrs menginisialisasikan procedure yang mengizinkan untuk bergabung dengan group tersebut.

2.3. TOKEN PARSING APPROACH

TOKEN :

Pesan bertipe khusus yang beredar pada seluruh sistem

- Dengan menggerakkan token diantara proses-proses dalam suatu sistem dapat menentukan mutual exclusion
- Hanya ada satu token dalam sistem hanya ada satu proses yang dapat masuk ke critical section pada suatu saat
- Proses-proses diorganisasikan secara logika dalam bentuk Ring (fisiknya jaringan tidak perlu Ring)
- Selama proses-proses saling berhubungan,sangat mungkin di implementasikan suatu Logical-Ring

Cara Implementasi :

➔Kirim token ke seluruh Ring

➔Ketika sebuah proses menerima token, ia dapat masuk ke critical section dan mempertahankan token tersebut

➔Setelah keluar dari critical section, token dilepas dan digerakkan ke seluruh Ring

➔Jika sebuah proses menerima token, tetapi tidak ingin masuk ke critical section, maka ia akan memberikan token ke tetangganya

Dua jenis kesalahan yang mungkin terjadi :

- Token hilang : memanggil “ election” untuk membangkitkan Token baru.
- Proses fail : struktur Ring baru harus dibentuk.