6. Coordonarea proceselor distribuite

6.1. Resurse

6.2. Starea globală și consistența calculelor distribuite

6.3 Sistem de monitorizare a căderilor de tensiune (voltage sag)

6.4. Sincronizarea proceselor distribuite

6.5. Alocarea resurselor în sisteme distribuite

6.6. Algoritmi de alegere (election algorithms)

6.7. Alocarea resurselor în managementul traficului

6.1. Resurse

Resurse:

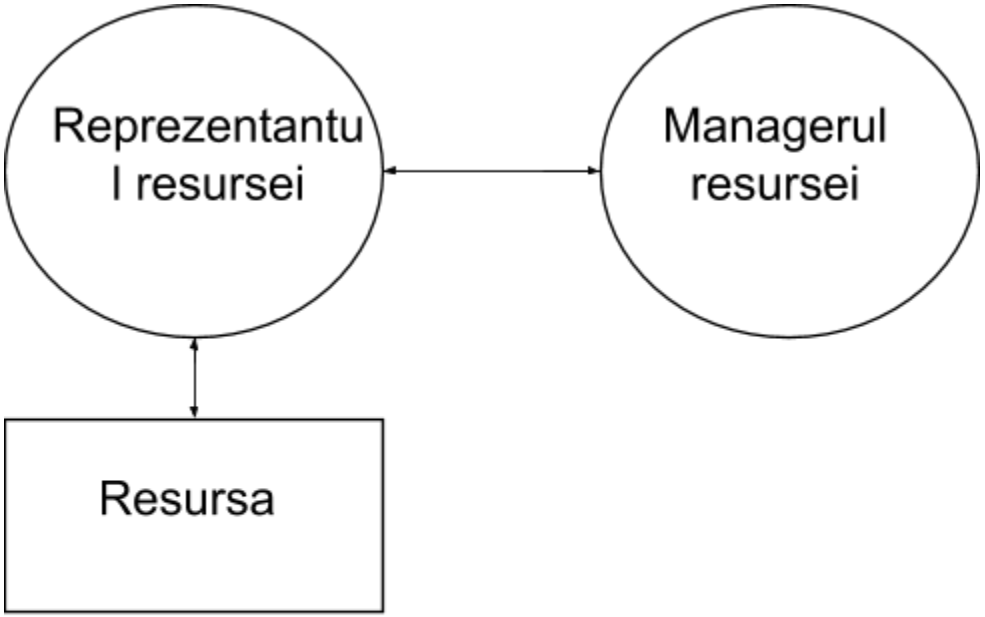
- interne:
  - hardware
  - software:
    - zone de memorie
    - secvențe de instrucțiuni (proceduri/funcții reentrante sau non-reentrante)
    - objects
- externe (device-uri periferice, actuatori, etc) – cu comportamente:
  - static
  - dinamic
  - pasiv
  - activ

Resursele pot fi:

- concentrate într-un singur nod
- distribuite în mai multe noduri

Resoursele pot fi (din punctul de vedere al utilizării):

- *dedicate*
- *comune (shared)*
- *cu acces multiplu*



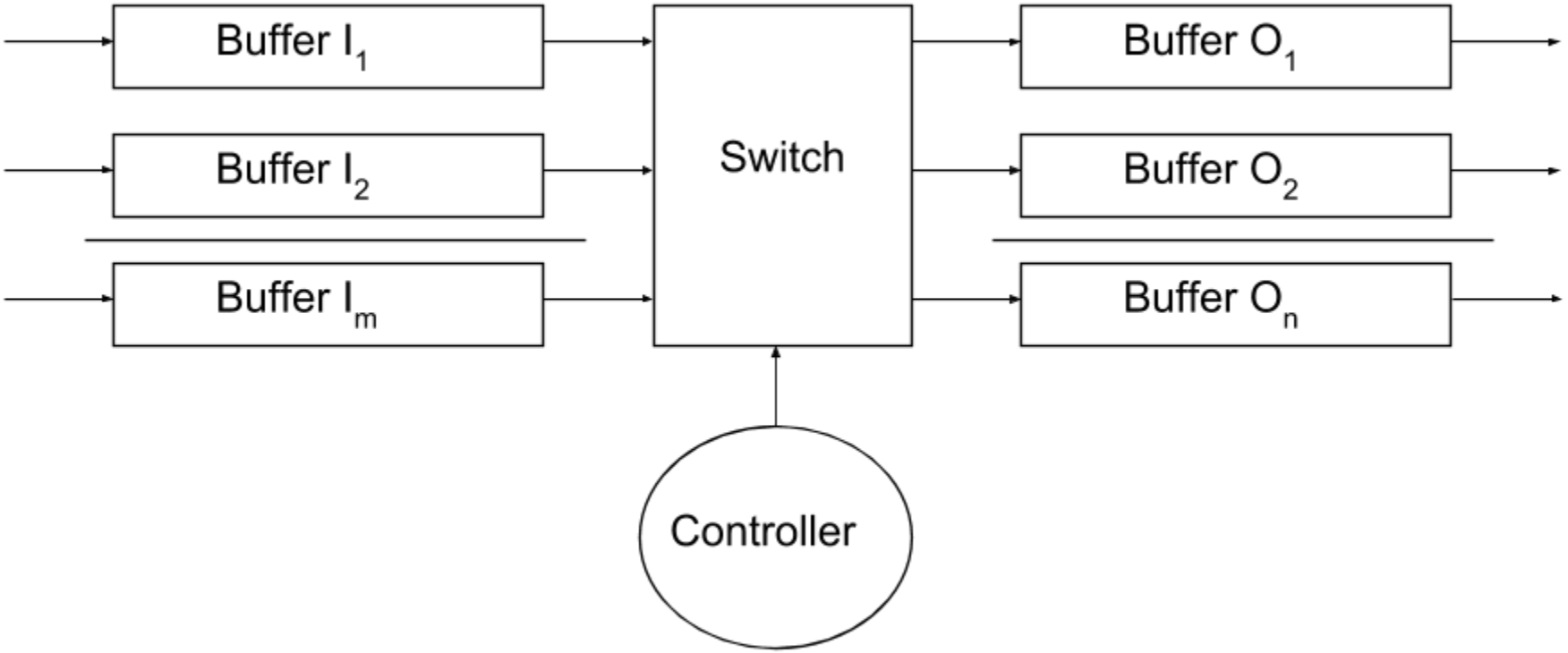
Reprezentarea resurselor logice

Alocarea resurselor: statică sau dinamică

Resursele aparțin::

- sistemului controlat ➔ resursele sistemului

- sistemului de control ➔ ex.: resurse software, hardware etc.



Switch resource - reprezentare logică

Alocarea greșită a resurselor poate duce la:

- interblocaj (deadlock)
- înfometare (starving)

Alocarea statică a resurselor ➔ evitarea deadlock-ului off-line
Alocarea dinamică a resurselor ➔ verificare on-line

Manageri de alocare a resurselor pentru evitarea deadlock-ului:

- prevenție
- evitare
- detecție
- recuperare

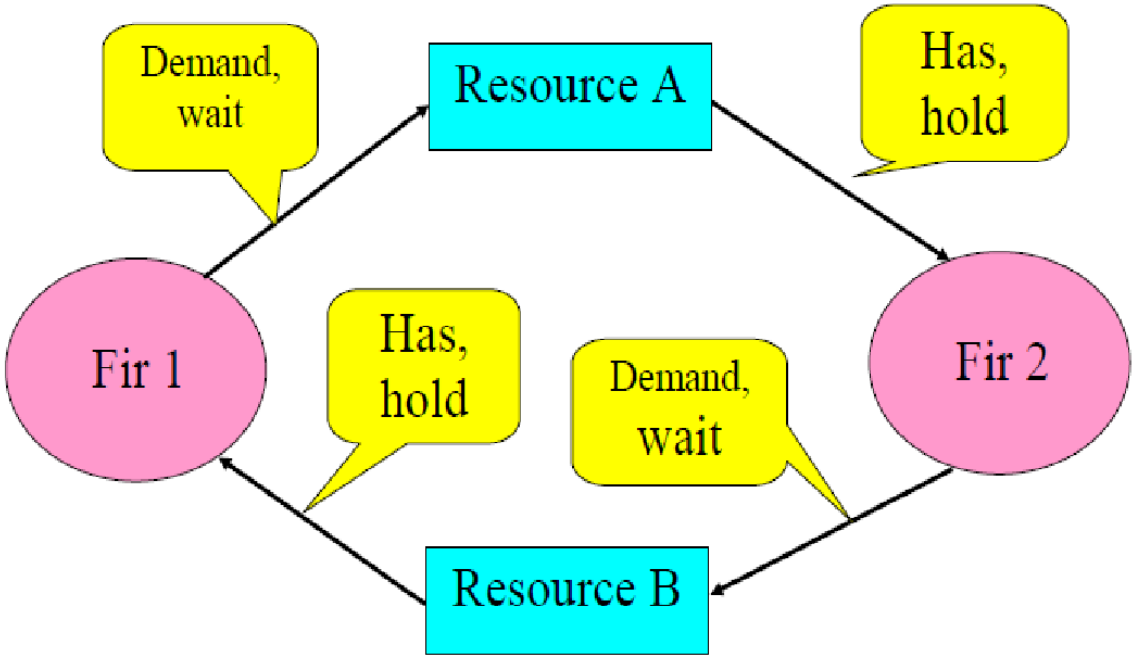


Fig. x. Ciclic waiting (deadlock).

6.2. Consistența stării globale și consistența calculelor distribuite

Starea unui sistem conține:
Toate informațiile necesare pentru a obține evoluția sistemului dacă mărimile de intrare sunt cunoscute. Starea determină evoluția viitoare a sistemului.
Variabile de stare:
- abstracte
- interpretare fizică

Elemente de stare:
- valorile variabilelor (s-a studiat până acum)
- secvența activităților îndeplinite
- starea resurselor (eliberate, rezervate sau ținute) etc.

Resursele la sistemul traficului feroviar:
Resursele sistemului controlat (ale căilor ferate)
Resursele sistemului de control

Trenul corespunde la un task .

Starea unui sistem de control distribuit:

- *Stările programului (activități executate, rezultate, variabile, zone de memorie etc.)*
- starea sistemului
- informații despre starea sistemului (plant)

Decizii de control ← unele (de obicei nu toate) variabile de stare sunt necesare

Multe sisteme de control implică semnalizarea sau reacția la evenimente sau atunci când sistemul îndeplinește o anumită condiție:

- monitorizare,
- depanare,
- detecția unei anumite stări (deadlock, terminare etc.),
- configurarea dinamică - adaptarea programului (de ex: echilibrul sistemului)

Starea globală a unui sistem este reuniunea stărilor proceselor sale.

Procesele unui sistem distribuit nu au ***memorie comună*** și pot comunica numai prin mesaje.

Comunicarea între procese implică schimbarea stării proceselor atunci când ele primesc/trimit mesaje.

Sisteme distribuite sincrone și asincrone - (noțiuni noi)

Sisteme distribuite sincrone

➔ Viteza de transmitere a mesajului este cunoscută

➔ Timpii de distribuire la destinație sunt cunoscuți

GREU DE FOLOSIT:

Sisteme distribuite asincrone

➔ Viteza de transmitere a mesajului este necunoscută

➔ Timpii de distribuire la destinație sunt necunoscuți

Aplicații practice: aproape toate sistemele distribuite sunt asincrone

Transmisia mesajelor este variabilă datorită:

- sistemul de operare (computer) al expeditorului
- Componente de transmisie ➔ buffere ➔ cozi de așteptare
- Transmisia mesajelor prin rețeaua de comunicație
- sistemul de operare (computer) al destinatarului
- procesul destinatar (primitorul de mesaje) etc.

Dacă mesajele într-un sistem de control întârzie nedefinit, unele informații necesare pentru luarea deciziilor nu sunt disponibile la timp

➔Unele decizii nu pot fi luate niciodată

E.g.: IF (x_1 is X_1) ^ (x_2 is X_2) ^ ··· THEN (x_k is X_k) ^ (x_l is X_l) ^ ···

Evitare ➔ presupunerea întârzierii == întârzierea informațiilor pentru a lua decizii nu trebuie să fie mai mare decât o durată specificată

➔ ***sisteme pseudo-sincrone***

Transmisia mesajelor

$$\mathbf{T_{min} + T_x}$$

T_{min} – durata minimă a transmisiei
T_x – întârzierea transmisiei (necunsocută și variabilă)

Concluzie:

- Sisteme sincrone: T_x = *constant (cunoscut)*
- Sisteme asincrone: $T_x < \infty$
- **Sisteme pseudo-sincrone**: $T_x < B$, unde B este durata maximă (acceptată) a întârzierii.

Sistem pseudo-sincron **➔ sistem de comunicație de timp real**

➔ cerințele temporale ale comunicației

Consistență: exemplu

Ex.: legea lui Ohm $U=R\cdot I$

Rezistență: R (constantă)
Tensiune variabilă: u(t).
Curent: i(t)

legea lui Ohm : $u(t)=R\cdot i(t)$.

La t_1 se citește tensiunea U_1.
La t_2 se citește curentul I_2.

➔ $U_1 \neq R\cdot I_2$.

➔ Mulțimea (U_1, I_2) este *inconsistentă*

Consistența necesită menținerea proprietăților necesare pentru un comportament corect al sistemului.

Consistență distribuită: se referă la menținerea proprietăților în noduri (calculatoare).

Asigurarea consistenței: sistemul trebuie să mențină rețeaua de comunicație funcțională și cooperarea dintre noduri în ceea ce privește execuția.

Consistență:

- Consistența execuției (a rulării) – ordinea evenimentelor și
- Consistența datelor – starea sistemului

Consistența la replicare

Motive pentru replicare:
- eficiență
- siguranță

Toate procesele trebuie să citească aceleași date ➔ întârziere la actualizarea datelor

Sistemele distribuite au:
- fișiere
- zone de memorie
- baze de date
- obiecte

Consistența datelor este dată de ordinea în care se execută anumite operații

Lipsa ceasului global ➔ dificil de obținut timpul operațiunilor de scriere.
Ceasurile logice pot fi folosite în locul ceasului global.

Ex.: Consistența la traficul feroviar

- stările segmentelor
- stările macazelor
- stările semafoarelor

Cerințe: operații de scriere/citire: atomice !
Cum poate fi atomicitatea implementată ?

Consistența stării globale

Cauzele inconsistenței:

- operațiile de scriere nu pot fi făcute în același moment în toate replicile (copiile)
- actualizarea informației nu este făcută în așa fel încât ordinea temporală să fie asigurată
- unele informații sunt prea vechi (*deprecated*)

Creșterea vitezei de comunicație **nu** garantează consistența stării globale. Aceasta poate da o informație închisă despre starea globală.

Cauza principală: întârzierile variabile !
Sisteme distribuite:

- Consistența proceselor software
- Consistența informației achiziționate din sistem

informații inconsistente ➔ decizii greșite

E.g.: IF (x_1 is X_1) ^ (x_2 is X_2) ^ ··· THEN (x_k is X_k) ^ (x_l is X_l) ^ ···

Abstractizări:

- evenimentele sistemului sunt considerate similare cu evenimentele unui proces software
- Sistemul este modelat de un set de mașini de stare care comunică (și care sunt similare proceselor software)
- Stările sistemului sunt similare cu stările proceselor software.

Decizii de control

Pentru luarea deciziilor de control este necesară:

- Evaluarea stării curente a sistemului
- Evaluarea stării trecute (stocată în memoria unui calculator sau alte periferice)

Cerințe:

- Evaluarea predicatelor logice
- Consistența variabilelor de stare

Predicatul global

Predicatele globale sunt construite prin codificarea proprietăților sistemului
Sunt folosite variabilele de stare necesare

Predicatul global folosește starea globală. *Zone de consistență ?*

E.g.: IF $(x_1$ ***is*** $X_1)$ ^ $(x_2$ ***is*** $X_2)$ ^ ··· ***THEN*** $(x_k$ ***is*** $X_k)$ ^ $(x_l$ ***is*** $X_l)$ ^ ···

Folosirea predicatelor globale:
- detecția deadlock-urilor (hardware sau software)
- detecția terminării unui proces
- decizii de control
- detecția funcționării defectuoase
- puncte de control intermediare și restartare
- monitorizare
- reconfigurări, etc.

Întrebări

- Cum poate fi detectat deadklock-ul la traficul feroviar ?
- Cum poate fi detectată înfometarea trenurilor ?
- Ce decizii de control sunt perturbate de inconsistența stării?
- Poate fi implementat rollback-ul la traficul trenurilor ?
- Care sunt elementele de stare la sistemul traficului feroviar ? Cum este influențată monitorizarea de inconsistența stării ?
- Cum poate fi implementată schimbarea de la control manual la automat pentru traficul feroviar ?

Calcule distribuite (executări distribuite)

Proces software sau tehnic

➔ secvență de evenimente (interne, externe)

Orice eveniment intern schimbă doar starea locală.
Evenimentele externe sunt semnalizate celorlalte procese:

- *send(m)* și *signal(e)*
- *receive(m)* și *handle(e)*

  Același canal este folosit pentru:
  - send - receive
  - signal - handle

Un proces primește un mesaj sau un eveniment numai dacă este pregătit (ready) pentru a face această activitate. Activitatea este întârziată dacă procesul nu este pregătit pentru aceasta.

Istoria locală

Istoria locală h_i a procesului (software sau tehnic) p_i constă într-o secvență de evenimente (posibil infinită) notată cu e_i^j.

$$h_i = e_i^1 \, e_i^2 \cdots e_i^j \cdots$$

Enumerare canonică:

$$h_i = e_i^1 \, e_i^2 \cdots e_i^k$$

***Prefix inițial**:* primele k evenimente din istoria locală

Istoria globală

Istoria globală a execuției unui ***set compus din n procese*** (software sau technice)

$$H = h_1 \square h_2 \square \cdots \square h_n$$

Evenimentele nu sunt ordonate. Nu există temporizare relativă a evenimentelor
Nu există ceas global.
Evenimentele pot fi ordonate prin relația cauză - efect (precedență cauzală).

Starea locală și globală

Starea locală (notată σ_i) a unui proces (software sau tehnic) conține:

- variabile locale
- starea senzorilor
- secvențe *send()* și *receive()*, sau *signal()* și *handle()*

Starea globală a unui sistem format din n procese este un n-tuplu de stări locale:

$$\Sigma = (\sigma_1, \sigma_2 \cdots \sigma_n)$$

Precedența cauzală

send(m) $\rightarrow$ *receive(m)*.

signal(e) $\rightarrow$ *handle(e)*.

Tranzitivitate:

$(e_1 \rightarrow e_2)$ și $(e_2 \rightarrow e_3)$ implică $(e_1 \rightarrow e_3)$

Evenimentele care nu sunt **precedente cauzal** sunt **concurente**.

$(e_1^1 \rightarrow e_3^1)$, $(e_1^1 \rightarrow e_1^2)$, $(e_1^1 \rightarrow e_1^3)$, $(e_3^2 \rightarrow e_1^4)$

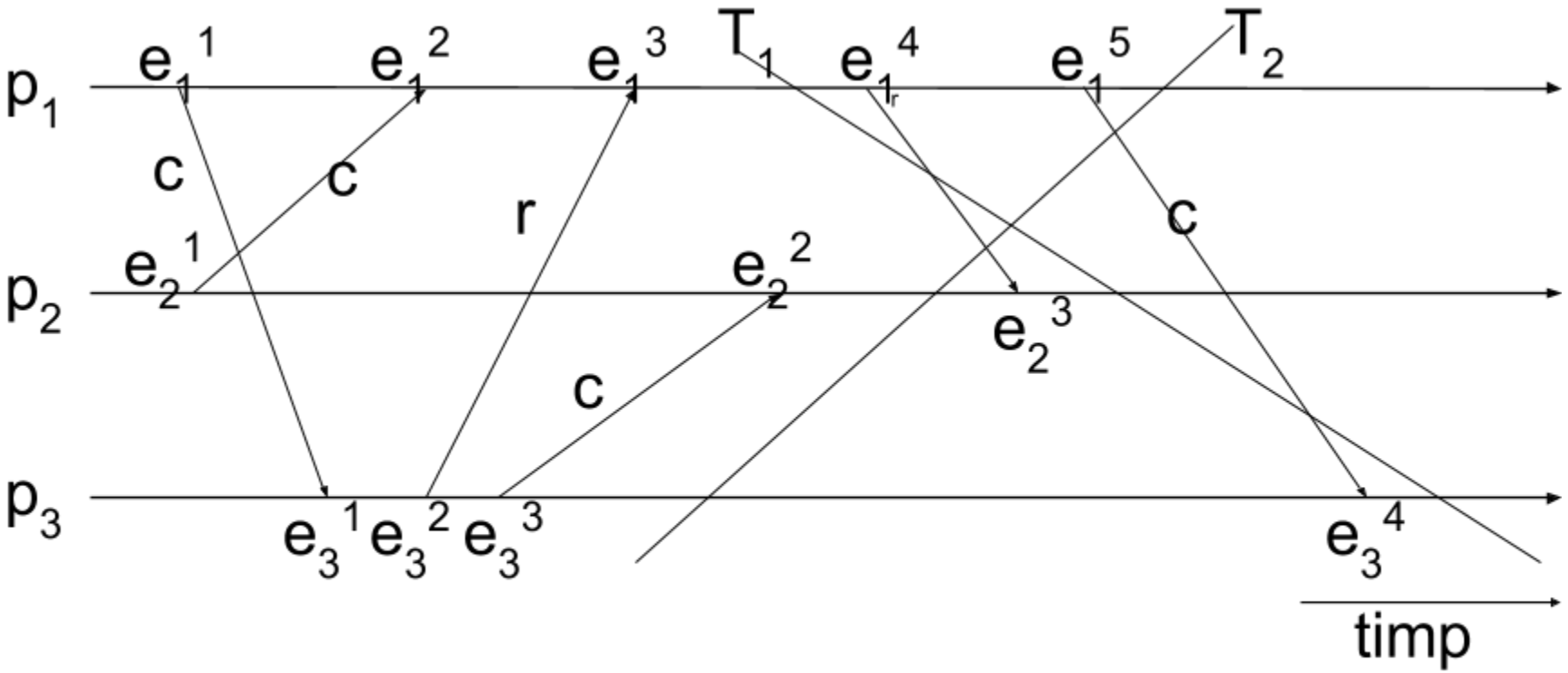


Diagrama spațiu-timp (tăieturi)

e_2^2 şi e_1^4 sunt concurente.

Tărietura unei execuții

Tăietura unei execuții distribuite a unui subset T a istoriei globale conține un prefix inițial a fiecărei stări locale.

Tăietura poate fi specificată folosind un tuplu de numere naturale (indexul ultimului eveniment din fiecare proces)

$T = (t_1, t_2 , \cdots, t_n)$

$T_1 = (3,3,4)$

$T_2 = (5,2,3)$

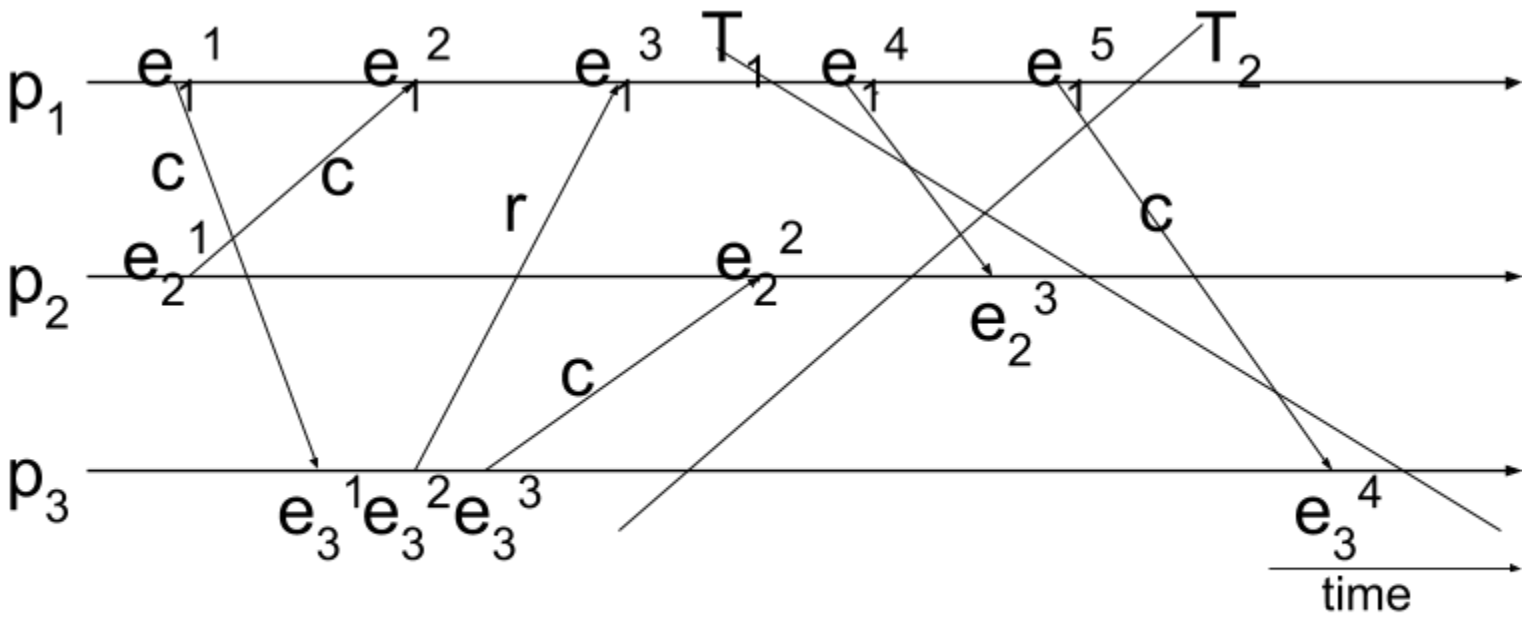
Mulțimea este parțial ordonată.

O tăietură T este ***consistentă*** dacă pentru toate evenimentele e, e' incluse în tăietură se satisface regula:

$$if ((e \rightarrow e') \text{ and } (e' \in T)) \text{ then } (e \in T)$$

Concluzie: Într-o tăietură nu poate fi conținut efectul fără a fi inclusă și cauza!

Tăietura este consistentă dacă este închisă relativ la relația de precedență cauzală.

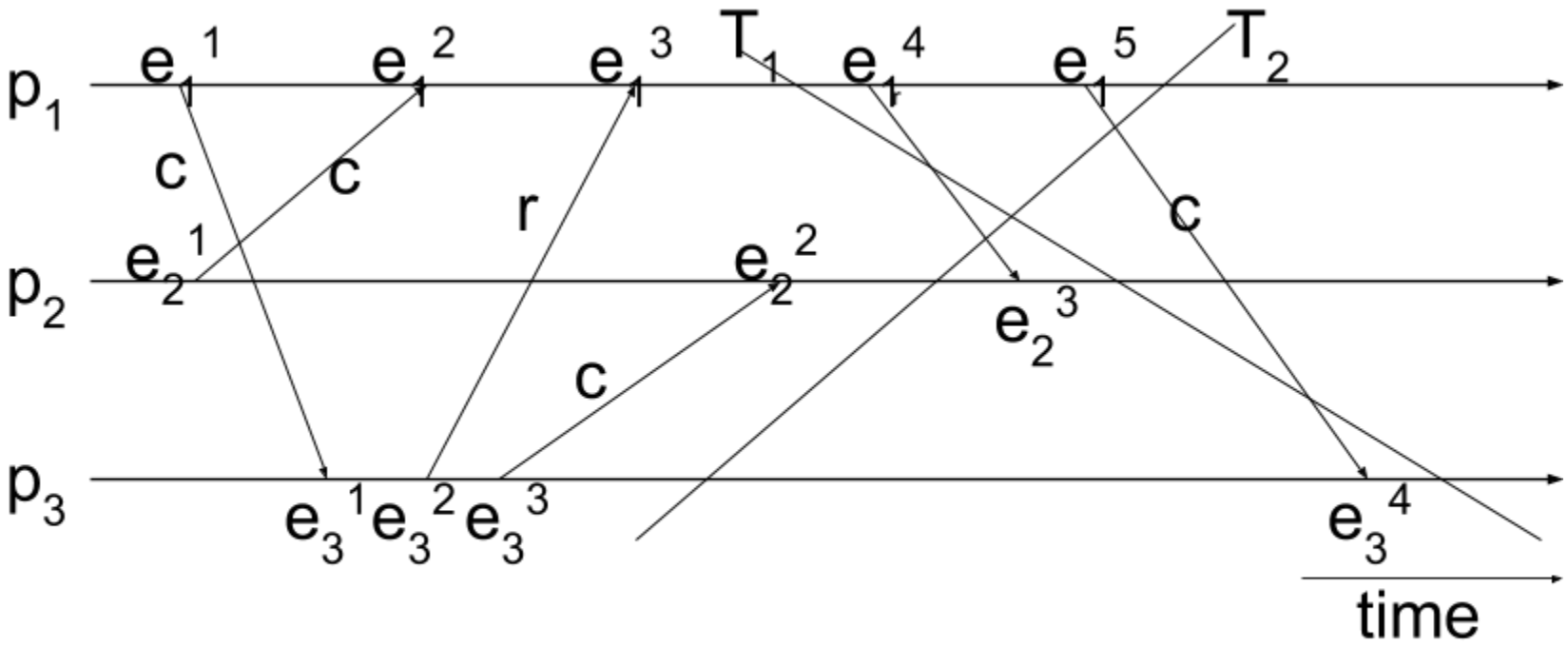


Space-time diagram

Starea globală este consistentă dacă corespunde unei tăieturi consistente.

Execuția unor calcule distribuite produce o ordine totală E a tuturor evenimentelor sistemului care s-au întâmplat până la acel moment de timp. Include toate evenimentele istoriei globale și este consistentă cu fiecare istorie locală.

O execuție este consistentă dacă ordinea evenimentelor sale respectă relația de precedență cauzală.



Space-time diagram

Monitorizarea proceselor distribuite

Monitor:
- pasiv sau
- activ

Rolul monitorului ➔ să ordoneze evenimentele ținând cont de relația cauză-efect (precedență cauzală)

Există întârzieri de comunicație.

Ordonarea corectă a mesajelor este posibilă dacă:

- Întârzierea maximă și drift-ul maxim al ceasului sunt cunoscute. Valoarea timpului local este inclusă în fiecare mesaj. Fiecare mesaj este întârziat cu suma întârzierilor menționate. Mesajele trimise în timpul intervalului de timp - toleranță sunt considerate concurente. Astfel de evenimente pot fi ordonate folosind ceasuri logice.
- Se folosesc ceasuri logice pentru procese. Monitorizarea mesajelor se face considerând ordinea dată de instanța temporală a mesajului. Un mesaj este trimis numai dacă este garantat că nici un alt mesaj cu un timp logic mai mic (înainte) nu poate ajunge mai târziu.

Detecția omisiunii

Ceasurile logice pot fi folosite pentru acest scop. Altă variantă: ceasurile logice vectoriale.

Consistența trebuie garantată. Poate fi obținută prin întârzierea transmiterii informației. Concluzie: deciziile de control sunt întârziate.

Analiza cazului de control pentru traficul feroviar.

Soluția ➔ partiționarea sistemului din punct de vedere al deciziilor care trebuie luate.

ex: gara Cluj - 0.1sec

zona Transilvania: 10sec

6.3 Sisteme distribuite de monitorizare a căderilor de tensiune

6.3.1 Introducere
6.3.2 Aspecte legate de calitatea puterii
6.3.3 Structura sistemului de monitorizare
- **arhitectură Hardware**
- **arhitectură Software**

6.3.4 Testare și validare
6.3.5 Concluzii

6.3.1 Introducere

Există mai multe tipuri de probleme legate de calitate ce pot apărea într-o rețea publică de energie electrică.

Un număr considerabil de evenimente care apar în rețea sunt ***căderile de tensiune (voltage sags)***.

Căderea de tensiune, conform EPRI (Electric Power Research Institute), este cea mai importantă problemă legată de puterea electrică.
Afectează sectoarele serviciilor și industriei, clienți majori care folosesc echipamente de control, electronică de putere sau calculatoare sensibile.

Căderea de tensiune poate cauza sau nu probleme în funcție de:

- ***mărimea și durata căderii***,
- sensibilitatea echipamentelor
- tipul de aplicație

Posibilitatea de a identifica **locul (sursa) căderii de tensiune** devine importantă.

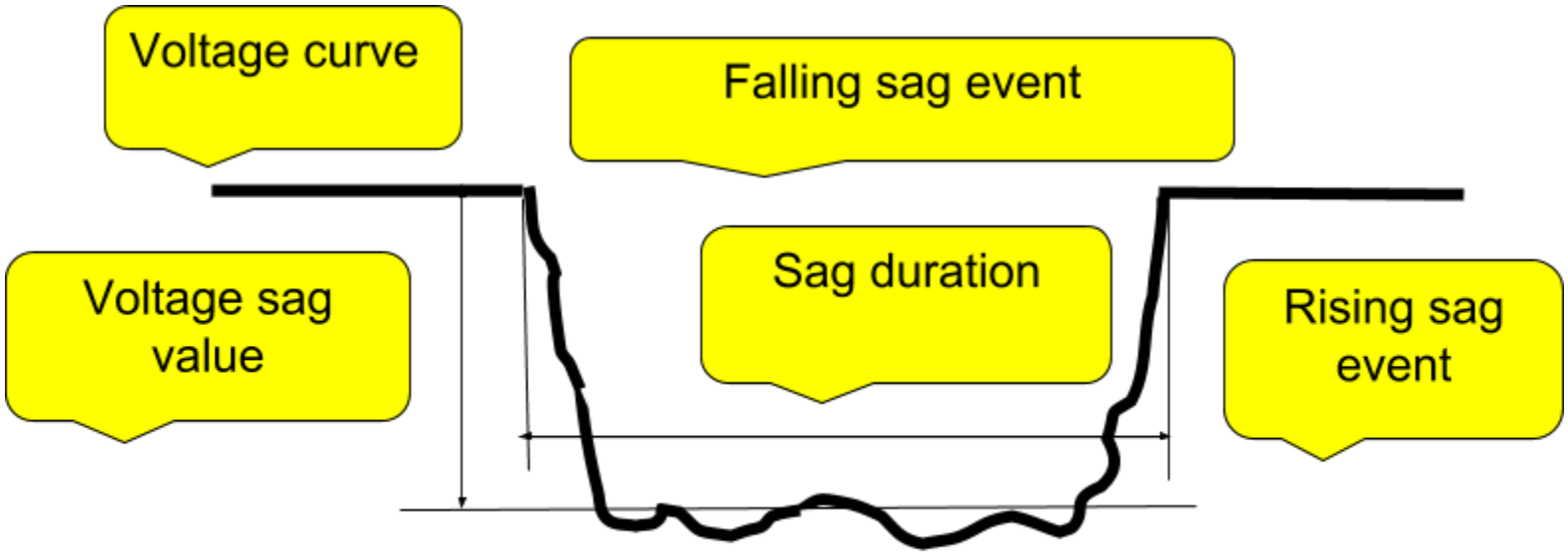
Trebuie avută mare grijă la căderile de tensiune care se întâmplă simultan pe diferite faze sau în succesiune rapidă.

S-a făcut multă cercetare pe acest subiect.

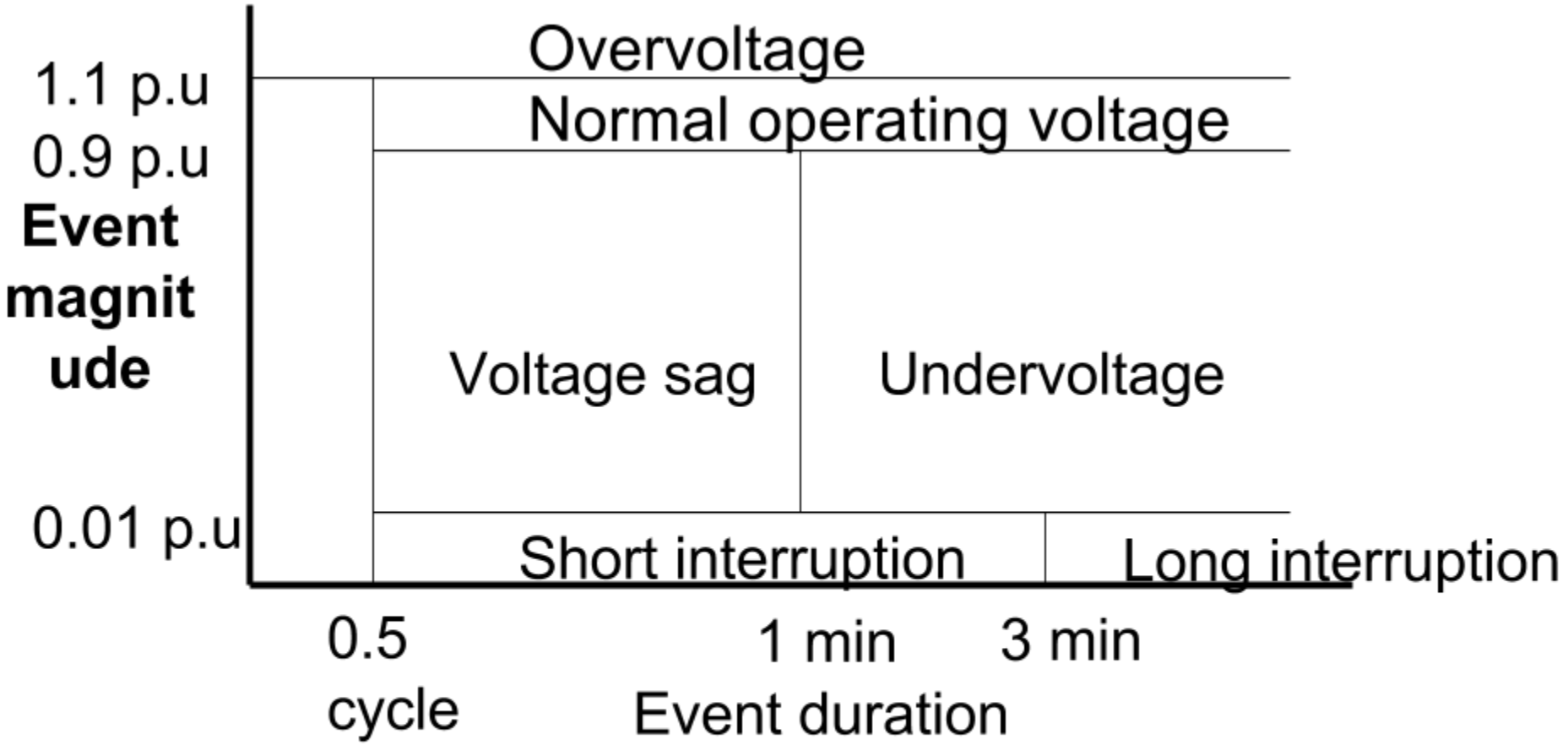
Aplicații practice, dificil de rezolvat:
1) Sistem de monitorizare distribuit, care ”vânează” evenimentele relevante în rețeaua electrică - constă dintr-un set de elemente împrăștiate pe o arie mare. Distanțele lungi dintre elementele sistemului de monitorizare implică folosirea unor noduri de comunicație care introduc ***întârzieri variabile*** la schimbul de mesaje. Aceste întârzieri fac dificilă stabilirea concluziei dacă toate evenimentele semnalate într-o perioadă foarte scurtă de timp corespund la aceeași cauză sau nu.
2) O altă problemă dificilă este aceea de a sincroniza ceasurile diferitelor componente implicate în monitorizare.

Scopul principal nu este numai de a determina apariția căderilor, ci și adâncimea, durata și frecvența, dar și zona geografică afectată.

6.3.2 Aspecte despre calitatea puterii electrice



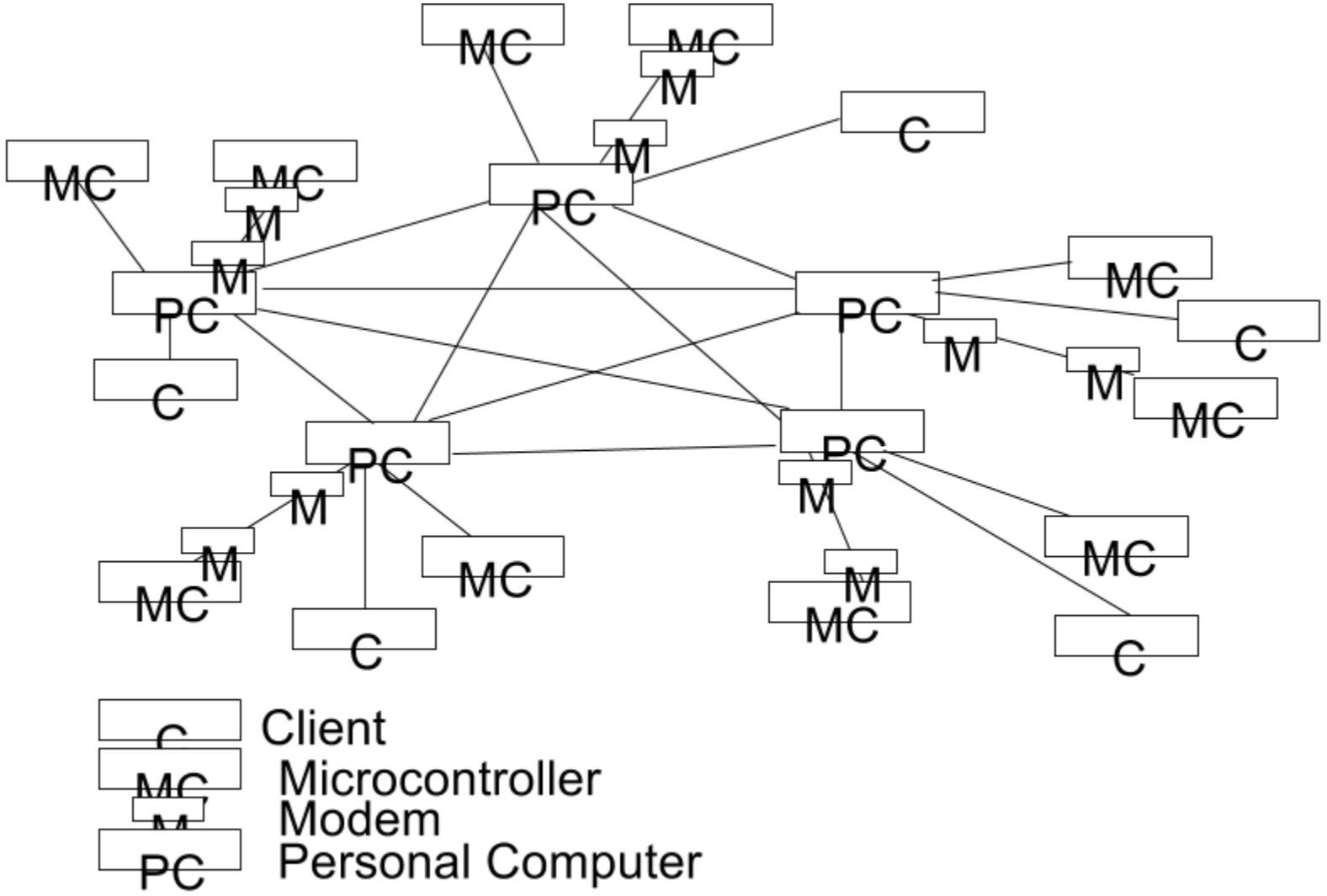
Evenimentele și parametrii unei căderi de tensiune.



Clasificarea magnitudinii evenimetelor la o cădere de tensiune

6.3.3 Structura sistemului de monitorizare

arhitectură hardware



Arhitectură software

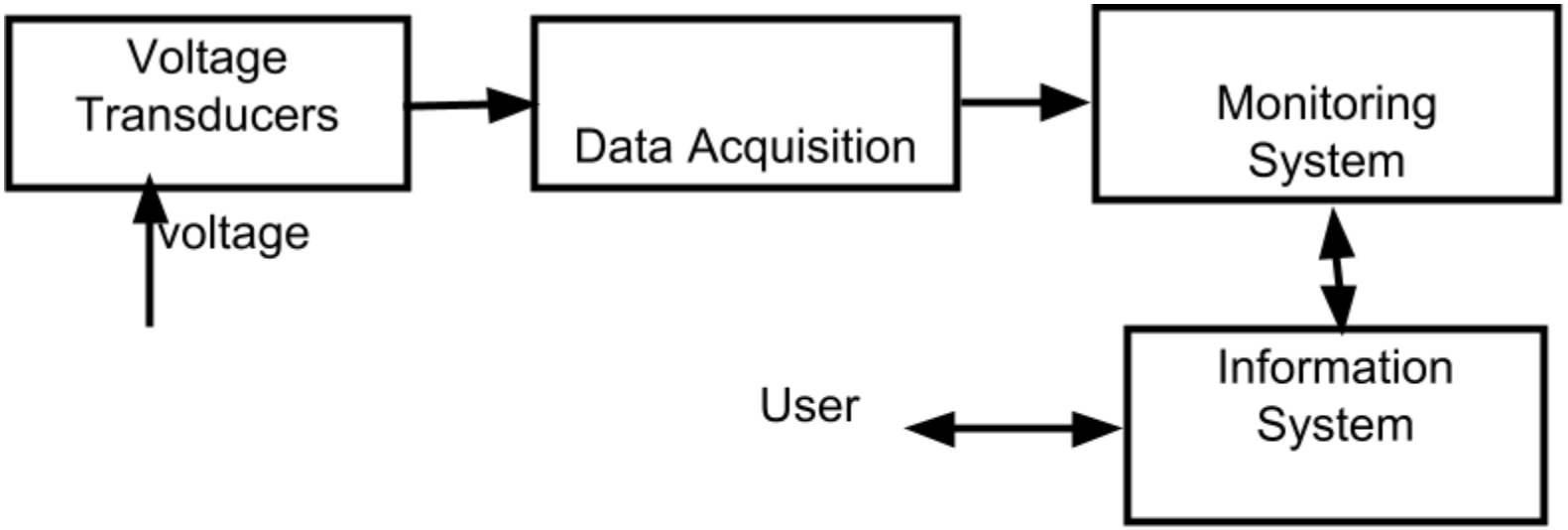
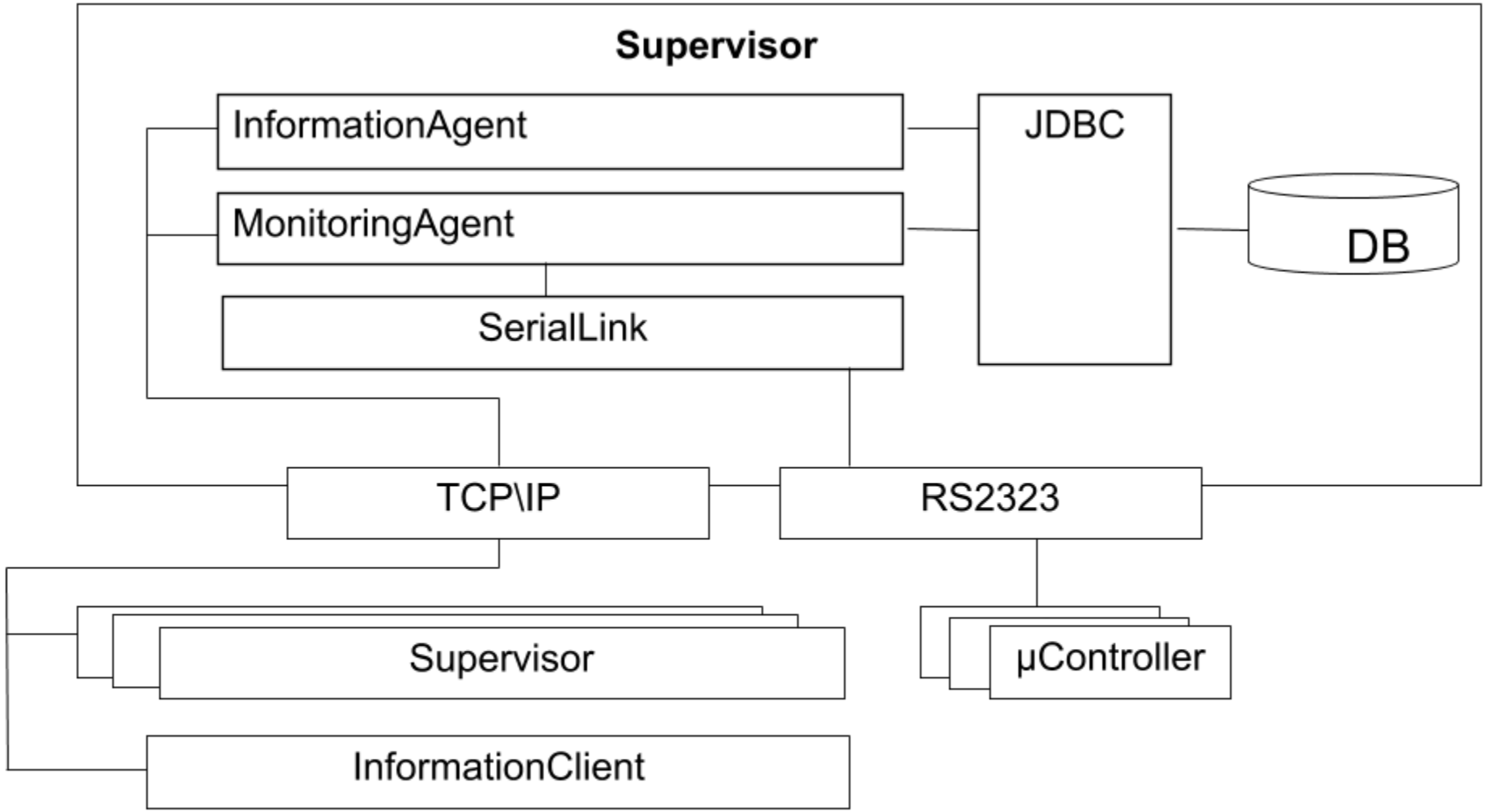
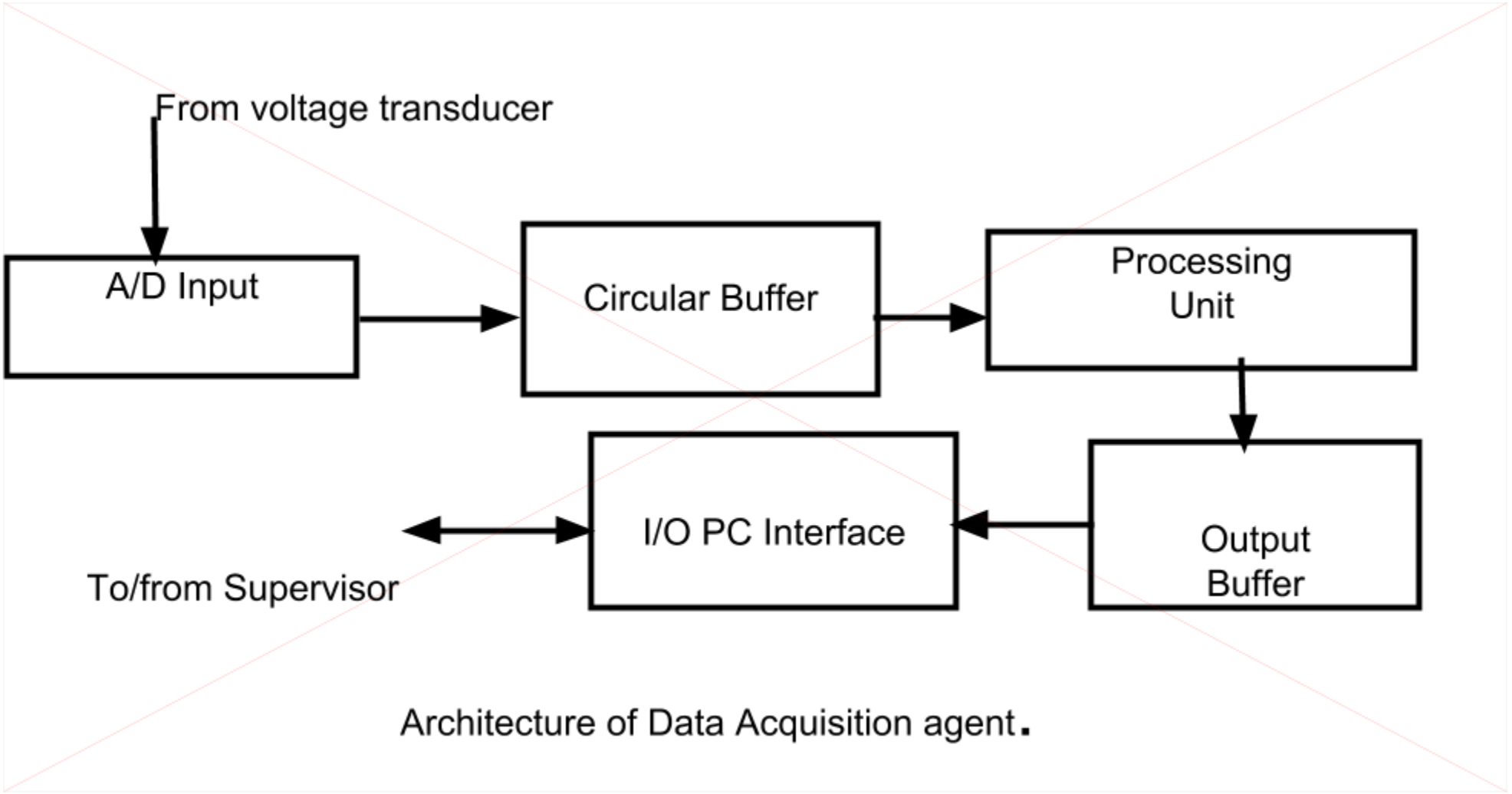


Diagrama bloc a sistemului de monitorizare

Arhitectură software



Supervisor structure and connections.



Architecture of Data Acquisition agent.

Area of vulnerability of a short-circuit at bus 16.

6.3.4. Testare și validare

EEE 39-Bus Test System

Table 1 Tensiunea pe linia i pentru un scurtcircuit pe linia j.

n	1	2	3	4	5	…	37	38	39
1	0	0.5	0.7	0.8	0.8	…	0.8	0.9	0.3
2	0.7	0	0.3	0.6	0.7	…	0.6	0.8	0.7
3	0.8	0.4	0	0.4	0.6	…	0.8	0.9	0.8
4	0.9	0.7	0.5	0	0.3	…	0.9	0.9	0.9
5	0.9	0.8	0.8	0.5	0	…	0.9	1	0.9
6	0.9	0.9	0.8	0.6	0.2	…	0.9	1	0.9
7	0.9	0.9	0.8	0.6	0.2	…	0.9	1	0.8
…	…	…	…	…	…	…	…	…	…
37	0.9	0.6	0.7	0.8	0.9	…	0	0.9	0.9
38	1	0.9	0.9	0.9	0.9	…	0.9	0	1
39	0.7	0.9	0.9	0.9	0.8	…	0.9	1	0

”Vânarea” evenimentului a fost implementată cu un task Signaller care execută următorul algoritm:

Initial: *set* SystemSate=normal
REPEAT (at each 0.001 second)
{
IF (SystemState=normal AND (NominalValue + 10%) < CurrentValue)))
THEN (*signal* RisingOvervoltageEvent AND
set SystemState=overflow)
IF (SystemSate=overflow AND (NominalValue + 10%) > CurrentValue)))
THEN (*signal* FallingOvervoltageEvent AND *set* SystemState=normal)
IF (SystemState=normal AND (NominalValue - 10%) > CurrentValue)))
THEN (*signal* FallingSagEvent AND *set* SystemState=lack)
IF (SystemSate=lack AND (NominalValue - 10%) > CurrentValue)))
THEN signal RisingSagEvent AND *set* SystemState=normal)
}

Analiza temporală

Sistemul de vânare (hunting) se confruntă (în cel mai rău caz) cu un set de **semnale care corespund la același eveniment**, care are efecte în mai multe locuri.

Sistemul trebuie să diferențieze între semnale ale aceluiași eveniment și semnale care au surse diferite.

Decizia este luată luând în considerare timpul apariției și locul unde evenimentele au fost detectate.

Microcontroler

Timpul când apare un eveniment este dat de ceasul de timp real al microcontrolerului.
Ca să se compenseze drift-ul, s-a implementat un task Sinchronizer. Are rolul de a actualiza ceasul de timp real cu ceasului PC-ului.
Datorită duratelor schimbului de mesaje, precizia sincronizării ceasurilor este de 1 millisecundă.
Pentru a menține precizia, sincronizarea trebuie făcută cu o perioadă de o ora.
Ceasurile calculatoarelor funcționează cu precizie de 55 milisecunde.
Sincronizarea cu ceasul microcontrolerului are precizie de 10 milisecunde.

Personal Computer (PC)

Sincronizarea ceasurilor calculatoarelor este o alta problemă. Ele pot fi sincronizate unul cu altul sau cu un ceas global (ales !). Aceasta determină precizia sistemului de monitorizare(pentru a discerne evenimentele diferite dintr-un set de semnale.

Achiziția datelor

Agenții responsabili de achiziția datelor trebuie să-și îndeplinească activitățile sub constrângeri temporale.. Fiecare agent este compus din 5 task-uri, parametrii: perioada (T), timpul efectiv de calcul (C), deadline (D) și prioritate (P). Task-ul de A/D Input măsoară, periodic, tensiunea curentă și o stochează în Circular Buffer.

Table 2. Parametrii taks-ului de achiziție a datelor

Task	T (sec.)	C (sec.)	D (sec.)	P
A/D Input	0.0001	0.00002	0.0001	5
Processing	0.001	0.00067	0.001	4
Signaller	0.1	0.00003	0.1	3
Sender	0.1	0.00005	0.1	2

Synchronizer	36000	0.00002	0.3	1

Task-urile sunt independente și sunt executate folosind un algoritm de planificare static. Ele respectă constrângerile de timp real dacă testul *Joseph and Pandya* este îndeplinit:

For all tasks i, $R_i \leq D_i$

where $R_i = C_i + I_i$ *is the task response time,*
C_i *the worst case computation time, and the tasks interference* I_i *is given by*

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$\lceil x \rceil$ este partea întreagă a lui x.

6.3. 5. Concluzii

Sistemul de monitorizare este folosit pentru a vâna evenimente pe o arie extinsă.
Este un sistem deschis, care acceptă intrarea și ieșirea supervizorilor. Din alt punct de vedere, datorită implementării în Java, nu există restricții legate de tipul calculatorului sau sistemul de operare.

Sistemul poate fi scalat fără să fie necesare modificări.
Utilizatorul poate accesa online lista de evenimente legate de poziția sa.
Dacă dotăm sistemul de monitorizare cu traductori de tensiune în punctele relevante, el poate da informații despre evenimentele care apar în punctele unde traductoarele lipsesc.
Folosind sincronizarea cu Global Positioning System ar putea crește precizia detecției până la 0.12 secunde.

6.4. Sincronizarea proceselor distribuite

Justificarea sincronizării

De multe ori unele activități (aplicații) trebuie:

- să fie startate în același timp,
- să fie/să nu fie executate concurent,
- să se termine în același moment de timp sau
- să fie executate secvențial

Problema legată de sincronizare:

- lipsa unui ceas fizic global
- ***lipsa unei memorii comune (globale)***
- apariția căderilor parțiale și temporare ale sistemului de comunicație

Sistemele de control distribuit:

➔ problema sincronizării activităților executate de sistemul controlat (plant)

➔ sincronizarea proceselor hardware și software

Toleranța temporală

Metode de sincronizare

În același calculator pot fi folosite:

- semafoare
- semnalizarea evenimentelor (???)
- monitoare
- regiuni critice
- lock-uri
- rendezvous etc.

De obicei nu sunt disponibile (dar unele pot fi implementate) în sisteme distribuite.

Excluderea mutuală

Motivele excluderii mutuale:

- pentru a preveni interferența task-urilor la execuția secțiunilor critice
- pentru asigurarea consistenței informației (stocată în resurse).

Secțiuni (regiuni) critice
Set de secțiuni critice

proprietatea de Siguranță ➔ doar un singur proces poate executa secțiunea critică la un moment dat

exemple: sinu, tranzacții bancare, etc

Implementarea sincronizării în sisteme distribuite
prin:

- semafoare distribuite (NU în aplicații reale)
- server de sincronizare
- ceasuri logice
- jeton într-un inel logic

Semafoare distribuite

Semafor ➔ o variabilă întreagă care poate fi citită sau scrisă (accesată) prin operații atomiceaccessed) by atomic operations

Syscalls = primitives:

- *P()* și *V()*,
- *set()* și *reset()*,
- *wait()* și *signal()*,
- *acquire() și release()*

Probleme:
- pierderea mesajelor
- Inactivitatea serverului

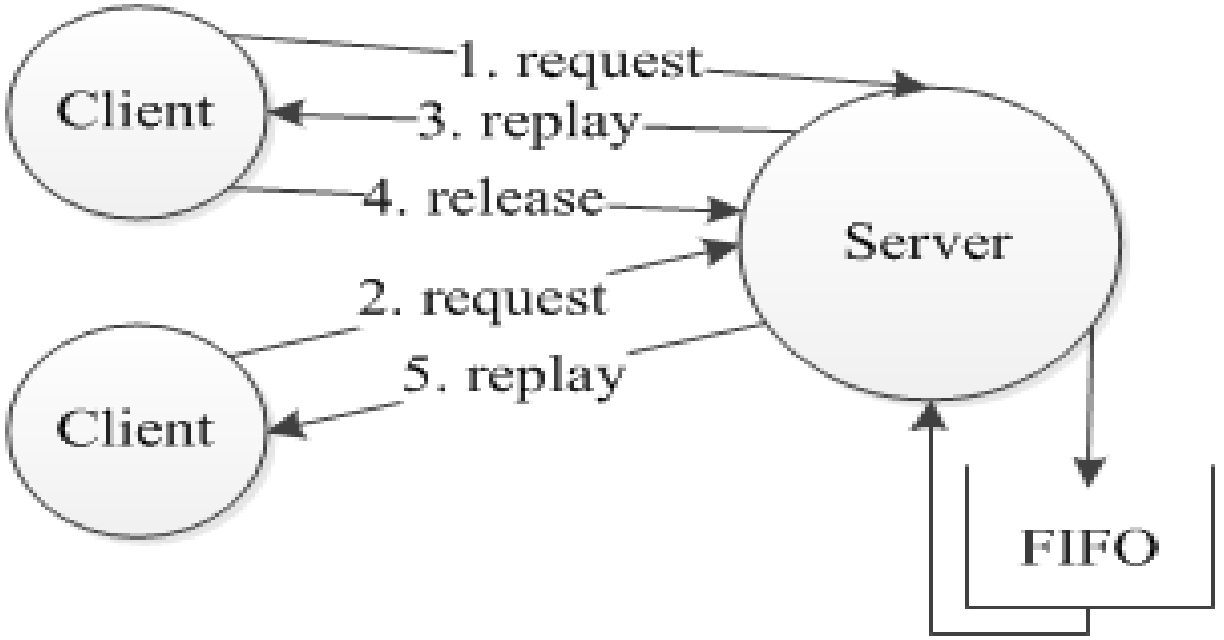
Structura: valoare + waitingQueue (coada de așteptare)

implică folosirea unei memorii virtuale comune cu operații atomice

← comunicarea prin mesaje

Server de sincronizare (centru de coordonare)

Serverul gestionează procesul de intrare în regiunile critice.



Procedura:

- Procesele solicitatoare (clienți) trimit mesaje *request* la server
- Serverul central de sincronizare adaugă solicitatorul într-o coadă de așteptare (FIFO). Aceasta menține cererile de sincronizare globale.
- Serverul trimite un mesaj *replay* (permite intrarea în regiunea critică) primului client (solicitatorul din capul coadei de așteptare) și îl șterge din listă.
- Clientul care execută regiunea critică trimite un mesaj *release* când nu mai are nevoie de ea.
- Serverul primește mesajul *release* și trimite un mesaj *replay* la următorul client care este în capul listei de așteptare.

Dezavantaje/probleme:

- Căderea serverului (restart → starea ?)
- Căderea clientului care este în regiunea critică
- Pierderea unui mesaj de sincronizare.

Poate fi implementată sincronizarea astfel pentru trenuri ?

Sincronizarea cu ceasuri logice

Decizia de intrare în regiunea critică este luată distribuit.

Procedura:

- Fiecare proces implicat în sincronizare menține un ceas logic (actualizat la fiecare eveniment legat de sincronizare)
- Procesele implicate în sincronizare cunosc identitatea și adresele celorlalte procese.

Un proces poate fi în una din următoarele stări:

- *RELEASED* – eliberează un jeton, nu are nevoie de jeton, nu vrea jetonul
- *WANTED* – vrea jetonul
- *HELD* – deține jetonul

Identificatorii proceselor p_1, p_2, ..., p_n.

Protocolul de sincronizare a unui proces p_j:

1) Inițializare:

state=RELEASED;

2) Preluarea jetonului:

state=WANTED;

Trimite cererea la toate procesele;

T=timpul cererii;// include (T,p) in mesaj

wait until(numărul de răspunsuri primite=(n-1));

state=HELD;

Intră în regiunea critică;

3) Când primește o cerere de la un proces p_i cu (T_i,p_i):

if(state=HELD or (state=WANTED and $((T,p_j) < (T_i,p_i))$*)*

then Adaugă cererea p_i fără a răspunde

else Răspunde imediat lui p_i;

4) Când iese din regiunea critică :

state=RELEASED;

Răspunde la toate cererile din lista de așteptare;

Observație: Pierderea mesajului ➔ ???

Sincronizare cu inel cu jeton

Inel logic

Fiecare proces trimite jetonul mai departe când nu are nevoie de el.
Fiecare proces care are jetonul poate intra în regiunea critică.

Dezavantaje:

- Se pierde jetonul dacă:
  - se pierde un mesaj
  - procesul care avea jetonul cade
- Un participant cade.

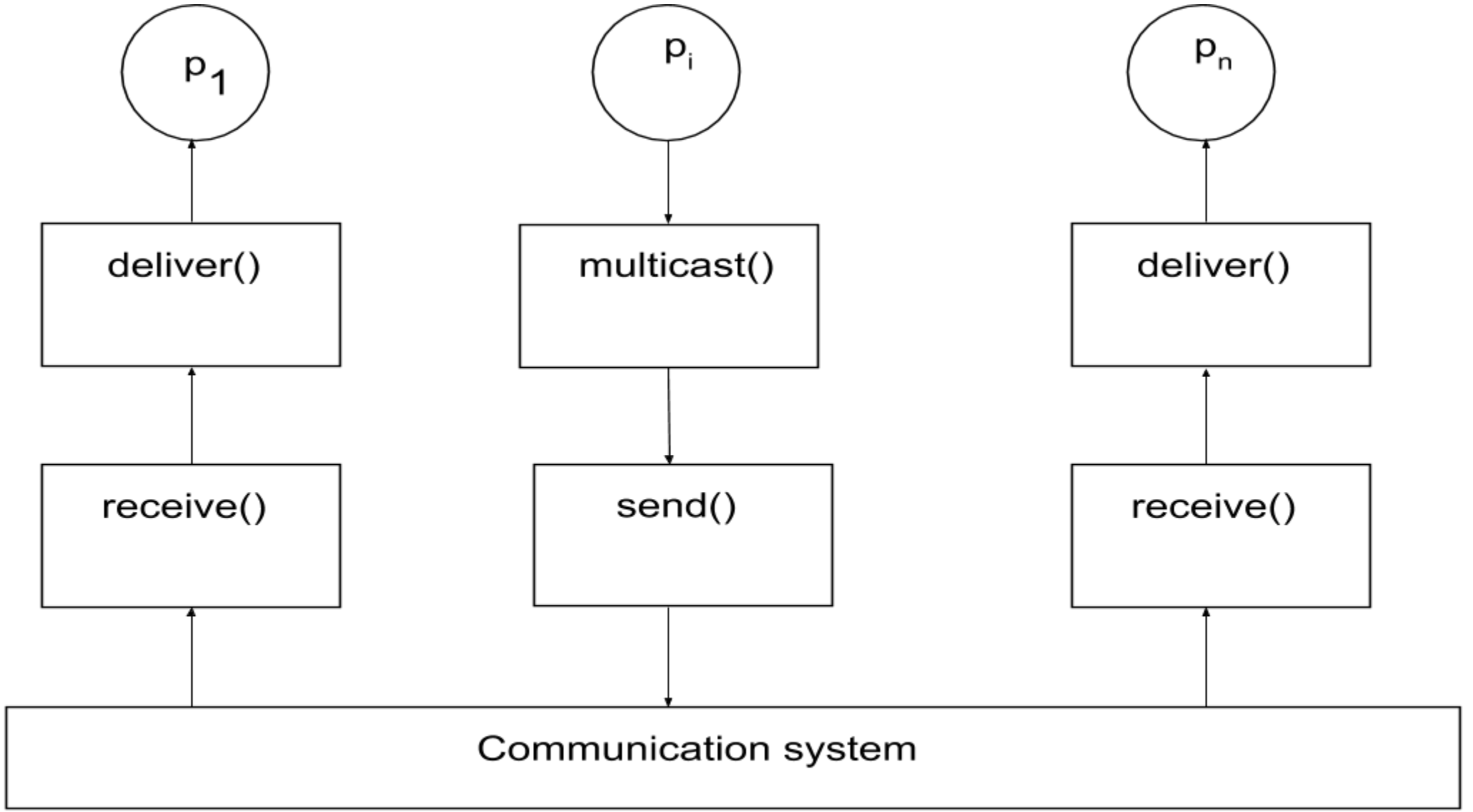
Sincronizare virtuală

Primitive:

- *send()* –
- *receive()* –
- *deliver()* –
- *multicast()* – trimite mesajul la toți participanții

Sistemul multicast îndeplinește condițiile:

- Dacă emițătorul trimite un mesaj multicast, el este distribuit la toate procesele.
- Dacă un proces de destinație trimite un mesaj, acesta este transmis la toate procesele destinație.
- Fiecare proces destinație primește un mesaj o singură dată dacă a fost trimis de multicast înainte.

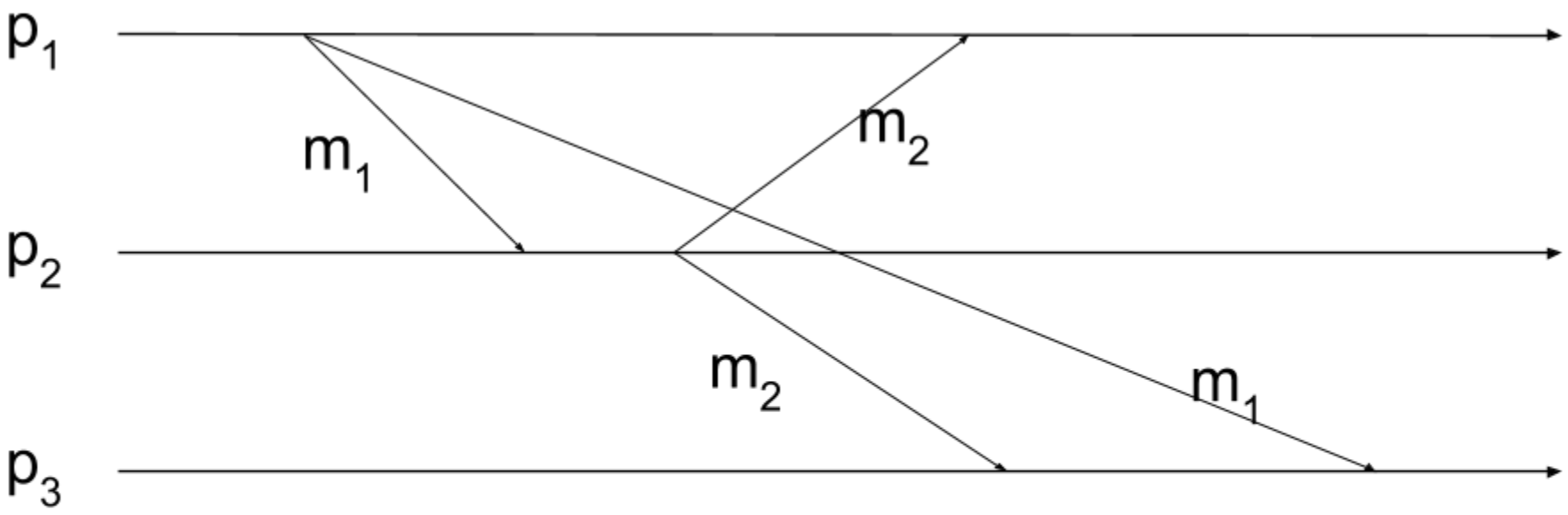


Model of a multicast system.

Probleme:

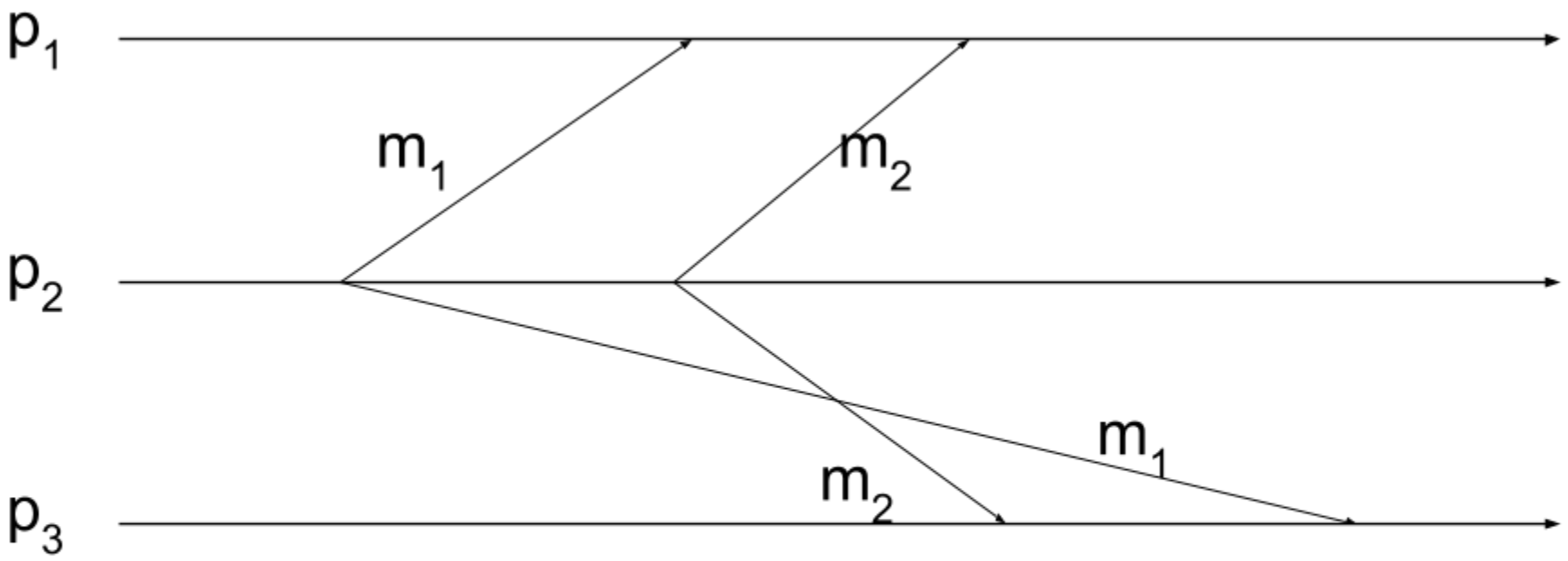
1) ***Violarea cauzalității*** transmisiei multicast

$m_1 \rightarrow m_2$ ➔ deliver(m_2) → deliver(m_1), process p_3.



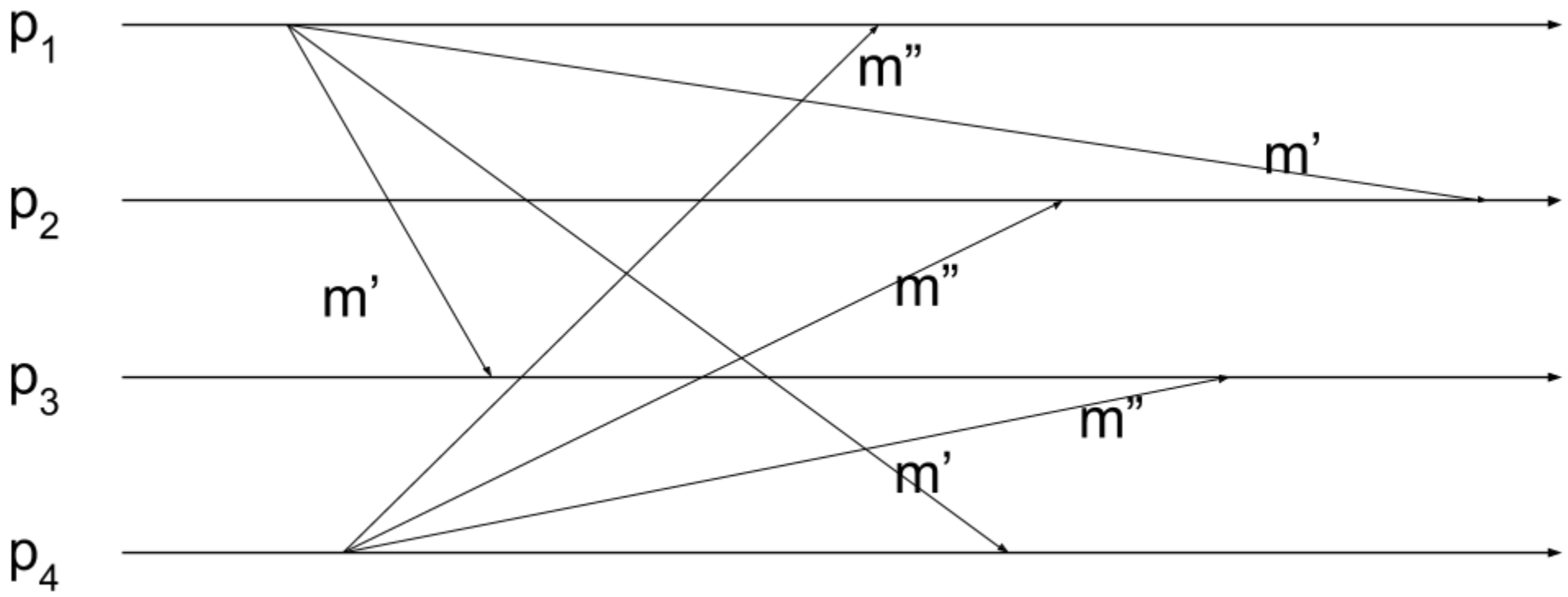
Causality violation

2) ***Inversarea ordinii de transmisie***:
$m_1 \rightarrow m_2$) ➔ deliver(m_2) → deliver(m_1))



3) ***Violarea atomicității***

Mesajele sunt distribuite în ordine diferită.



.

Tranzacții

Excludere mutuală
Operații atomice
Rollback

6.5. Alocarea resurselor în sisteme distribuite

Componente ale alocării de resurse:

- Manageri de resurse
- Agenți – cereri ale clienților
- Reprezentanți ai resurselor
- Coordonatori

Alocarea resurselor se poate face:

- *pe bază de timp* –
- *pe bază de evenimente* –

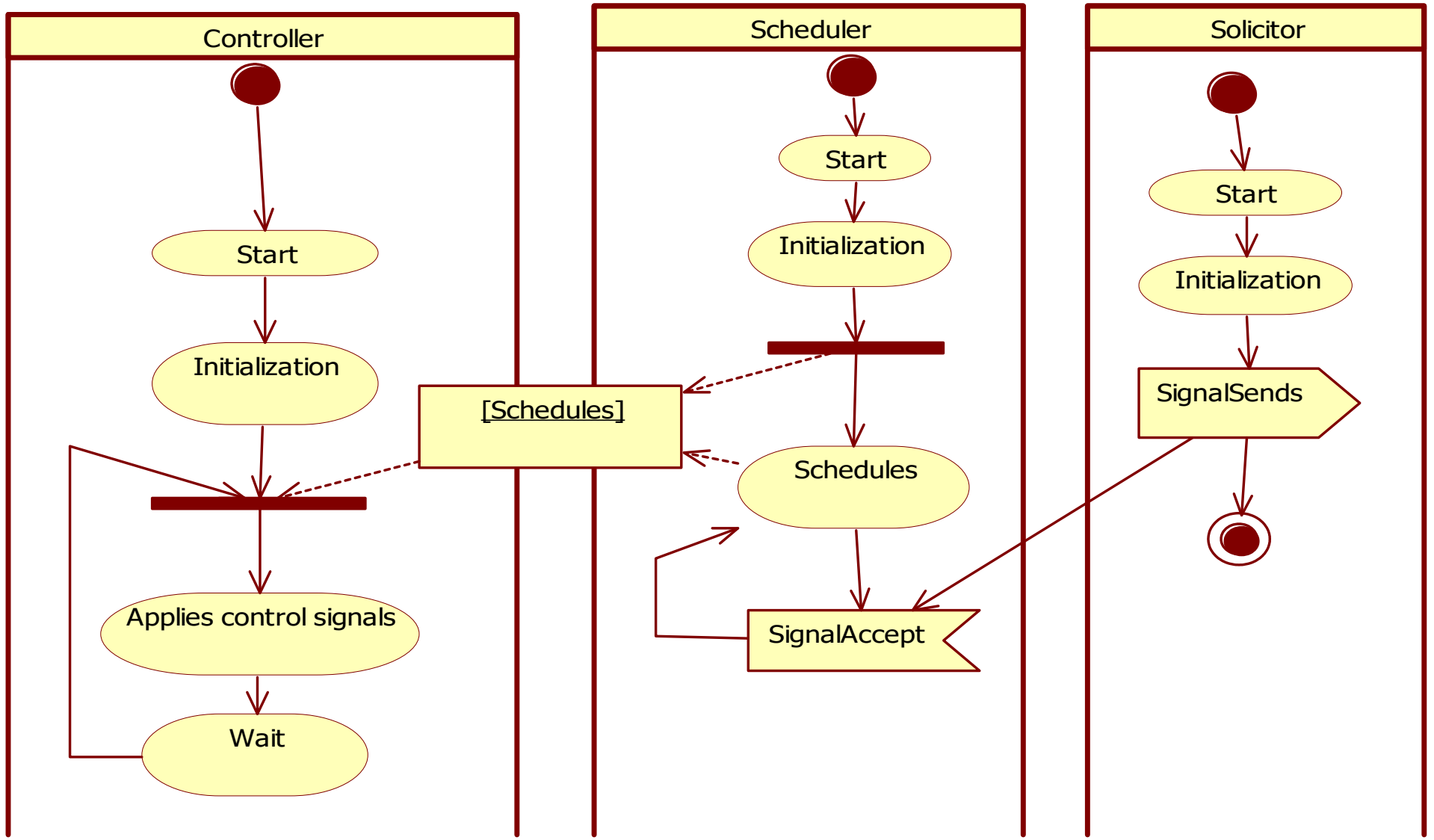
Implementarea preempțiunii?

Alocarea echitabilă (**fair allocation)**

Menține ordinea cererilor.

Alocarea prioritară (**priority allocation)**

Ia în considerare prioritatea solicitorilor.


Controller
Scheduler
Solicitor
Start
Initialization
Start
Initialization
Start
Initialization
[Schedules]
Schedules
SignalSends
Applies control signals
SignalAccept
Wait

Principii și metode pentru alocarea resurselor:

- ***Solicitorul*** (procesul) așteaptă până când deținătorul curent al resursei o eliberează.
  - Managerul resursei semnalează solicitorul.
  - Deținătorul resursei semnalează când eliberează resursa.
- ***Solicitorul*** cere resursa, dar își continuă activitatea. Verifică periodic dacă resursa fost eliberată.
  - Managerul resursei marchează că resursa este eliberată.
  - Deținătorul resursei marchează când eliberează resursa.
  - Reprezenantul resursei răspunde dacă resursa este deținută sau liberă.
- ***Solicitorul*** preemptează resursa de la deținătorul (procesul) mai puțin prioritar.

Probleme de alocare a resurselor:

- ***dining philosophers*** – set fix de solicitori & set fix de resurse cerute
- ***drinking philosophers*** – set fix de solicitor & set variabil de resurse cerute
- ***dynamic resource allocation*** – set variabil de solicitori & set variabil de resurse cerute

Timpul de răspuns

- măsoară performanța algoritmului de alocare
- depinde de:
  - numărul de procese în conflict
  - numărul de resurse cerute
  - timpul de folosire efectivă a resurselor
  - întârzieri de transmitere a mesajelor

Metode de alocare:

- semafoare
- coadă de rând sau ordine (queue of turns)
- server de alocare
- algoritmul Round-Robin
- algoritmul de alocare dinamică a resurselor multiple

Cozi de ordine

- bazate pe tichete (numere naturale).
- Dispecer de tichete.

Funcțiile reprezentantului resursei:

- Transmiterea tichetelor
- Interogarea alocării curente a resursei
- Utilizarea resursei
- Eliberarea resursei

Serverul este pasiv.

Server de alocare

- Fiecare resursă are un server.
- Serverul gestionează o coadă de așteptare (FIFO).
- Serverul primește cereri pentru resurse.
- Serverul semnalează solicitorul care are asignată prima poziție.
- Serverul este semnalizat când resursa este eliberată.

Algoritmul Round-Robin

- Există un set de procese solicitoare
- Reprezentantul resursei
  - Gestionează o coadă de așteptare și
  - Asignează un număr fiecărui solicitor
- Fiecare solicitor folosește resursa când numărul curent corespunde numărului asignat lui
- Fiecare deținător al resursei incrementează numărul current atunci când eliberează resursa

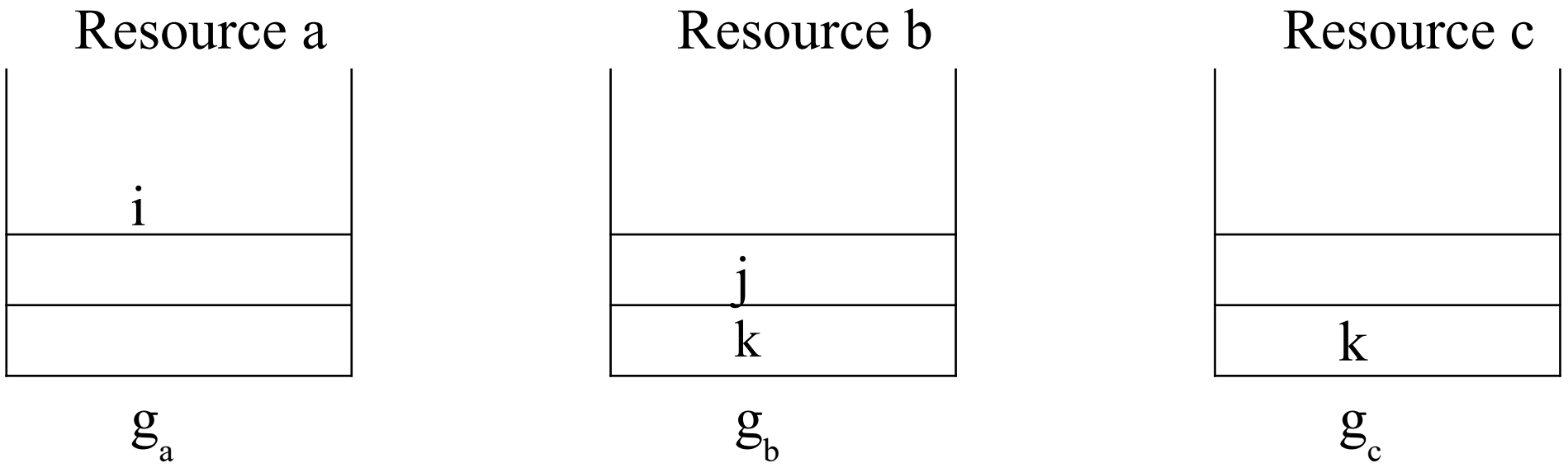
Implementare activă sau pasivă?

Algoritmul de alocare dinamică a resurselor multiple

- Mai multe resurse ➔ interblocaje (deadlocks)
- Un număr variabil de solicitori
- Un set variabil de resurse cerute

Fiecare manager de resursă gestionează o coadă de așteptare.

- n resourse
- m solicitori (clienți)



Schema de alocare.

- g_a, g_b şi g_c manageri de resurse
- a, b, c resurse
- i, j, k marcajul solicitorilor.

Protocol

- Numai solicitorul marcat în poziția curentă poate folosi resursele.
- Operația de rezervare este realizată prin excluderi mutuale.
- Un solicitor care are nevoie de un set de resurse trebuie să fie in aceeași poziție în toate cozile de așteptare corespunzătoare.
- Numai un solicitor poate fi într-o poziție într-o coadă de așteptare.
- Resursele dintr-o linie sunt allocate împreună.
- Un process care eliberează resursele notifică managerii acelor resurse.
- Cozile avanseaza simultan când toți deținătorii resurselor eliberează resursele.
- Un solicitor venit mai târziu nu poate întârzia solicitorii veniți mai devreme.

Alocarea resurselor (switch resource allocation)

- Proporțional cu volumele de intrare
- Ia în considerare prioritățile bufferelor
    - input buffer
        - până când solicitorul cu prioritate mai mare nu solicită resursa
        - preempțiune pentru prioritatea mai mare
        - durata alocată este proporțională cu prioritățile bufferului
    - output buffer – similar ca mai sus
- Se iau în considerare constrângerile impuse de bufferele de intrare și ieșire

6.6. Algorimi de alegere (election algorithms)

Implementarea fialbilității

Motivele (cauzele) alegerii:

- Jetonul este pierdut (datorită pierderii mesajului corespunzător)
- Mesajul care reprezintă eliberarea resursei este pierdut
- Procesul care avea jetonul (sau dreptul de a executa secțiunile sale critice) cade

Rolul alegerii:

- Să injecteze un jeton nou
- Să reconstruiască structura, inelul logic, rețeaua, etc.

Alegere (election) = procedura executată de un proces pentru a primi rolul de a inițializa structura de sincronizare. ➔ ***coordonator***

Alegerea este o metodă de a garanta fiabilitatea structurii de sincronizare.

Problemă hardware sau software?

Cerințele alegerii: să fie un process ales unic (coordonator, server, etc.) sau un jeton unic injectat.

Algoritmi de alegere:
- bully algorithm
- alegere în rețele unidirecționale
- alegere pe arbori
- alegere în topologii arbitrare

Inițial, procesele nu știu că coordonatorul este pierdut.
Coordonatorul informează ceilalți participanți că el are această funcție.

Bully algorithm

Se dau: n procese P_1, P_2, ..., P_n cu priorități constante
Cerințe: procesul cu cea mai mare prioritate devine coordonator
Presupuneri:

- sistemul de comunicație este fiabil
- numai un process poate să cadă

Startat de: orice proces care nu realizează un schimb de mesaje (activități) într-o perioadă de timp dată (coordonator-client)

Mesaje de alegere:

- ***election*** – anunță startarea procedurii de alegere
- ***answer*** – răspuns la un mesaj *election*
- ***coordinator*** – anunță un nou coordonator

Protocolul algorimului ”bully”

- un proces P_i trimite un mesaj ***election*** spre procesele mai prioritare
- P_i așteaptă o perioadă specificată pentru a primi mesaje ***answer***
- Dacă procesul P_i primește cel puțin un mesaj ***answer***, atunci consider că există un process mai prioritar care ar trebui să starteze procedura de alegere
  - Dacă după perioada specificată, P_i nu primește un mesaj ***coordinator***, restartează procedura de alegere.
- Dacă procesul P_i nu primește nici un mesaj ***answer***, se consider pe el însuși coordonator
  - P_i trimite un mesaj ***coordinator*** spre procesele mai puțin prioritare.
- Dacă un process P_j primește un mesaj ***election*** de la un process mai puțin prioritar, îi răspunde cu un mesaj ***answer***
  - P_j startează procedura de alegere (similarcu procesul P_i).
- Dacă un process primește un mesaj ***coordinator***, îl înregistrează ca și coordinator și urmează protocolul standard de sincronizare.

Alegere în inele unidirecționale

Structura: inel logic (funcțional)
Motivul (cauza) alegerii: jetonul este pierdut
Cerințe: să se injecteze un jeton nou și unic și să se restarteze sincronizarea
Presupunere: orice process poate executa o secțiune critică pentru un timp nu mai lung decât o durată dată
Se dă: n – numărul proceselor din inel
Asignat: fiecare proces are o anumită prioritate
Startarea alegerii: orice proces care nu primește jetonul pentru cel puțin o perioadă specificată

Protocolul de alegere:

- Un process care startează alegerea trimite mesajul ***election*** împreună cu un număr (prioritatea sa) spre următorul process
  Obs.: Mai multe procese pot starta alegerea simultan.
- Când un process care primește un mesaj ***election***, compară prioritatea primită prin mesaj cu prioritatea sa:
  - Dacă este mai mare, lasă mesajul neschimbat
  - Dacă este mai mica, schimba prioritatea inclusă în mesaj cu prioritatea sa
  - Trimite mesajul mai departe.
- Dacă un process primește un mesaj ***election*** cu o prioritate egală cu a lui proprie, se consideră îndreptățit să injecteze noul jeton.

Alegeri pe arbori

Structura: arborele este încă parțial funcțional
Presupunere: fiecare process știe adresele vecinilor săi; unele procese nu funcționează, dar procesul current poate detecta asta
Cerințe: să reconstruiască arborele curent

Fazele alegerii:
1) ***faza de explorare*** **(exploration)** – fiecare nod al arborelui este informat despre startarea alegerii
2) ***faza de asumare*** **(commitment)**– maximele locale sunt calculate și trimise spre nodul rădăcină
3) ***faza de informare*** **(information)** – nodurile sunt anunțate despre procesul de alegere

Protocolul de alegere:

- Un nod inițiator trimite mesajul ***exploration*** spre vecinii săi inferiori.
- Un nod interior care primește mesajul ***exploration*** îl trimite mai departe spre vecinii săi inferiori.
- Un nod frunză (final) care primește un mesaj ***exploration*** răspunde cu un mesaj ***commitment*** în care include identificatorul sau prioritatea sa.
- Un nod care primește un mesaj ***commitment*** actualizează informațile lui despre procesele inferioare.
- Un nod care primește un mesaj ***commitment*** de la toți vecinii săi inferiori (care sunt active) trimite un mesaj ***commitment*** (conținând identificatorul sau prioritatea sa și a vecinilor săi inferiori) spre vecinul superior.
- Un nod care primește un mesaj ***commitment*** și nu are nod superior, startează faza de ***informare*** prin trimiterea de mesaje ***information*** la toți vecinii săi.
- Un nod care primește un mesaj ***information*** îl trimite mai departe la vecinii săi.

Este posibil ca problema de control a traficului feroviar să ducă la nevoia unei algeri?

*

****

*** END ***

****

*