

Semafoare.

Probleme clasice.

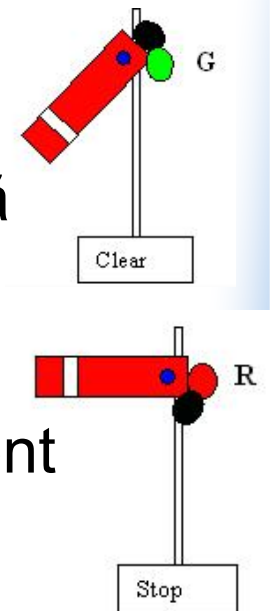
Ciprian Dobre
ciprian.dobre@cs.pub.ro

Dezvoltarea algoritmilor folosind variabile partajate (MIMD)

- Sincronizarea: **excluderea mutuală** și **sincronizarea condiționată**.
- Câteva rezultate notabile:
 - ♦ semafoare
 - ♦ regiuni critice
 - ♦ monitoare
- Câteva dintre problemele *clasice*:
 - ♦ producători-consumatori
 - ♦ problema filozofilor
 - ♦ cititori-scriitori
 - ♦ problema bărbierului

Semafoare

- Tip special de variabile partajate manipulate prin 2 operații **atomice**: P și V (*Dijkstra, 1965*)
- Valoarea semaforului: un **întreg nenegativ**
- **V** (*verhogen = to increment*)
 - ♦ semnalează apariția unui eveniment (incrementează semaforul)
- **P** (*proberen = to test*)
 - ♦ întârzie un proces până la producerea unui eveniment (decrementează semaforul)



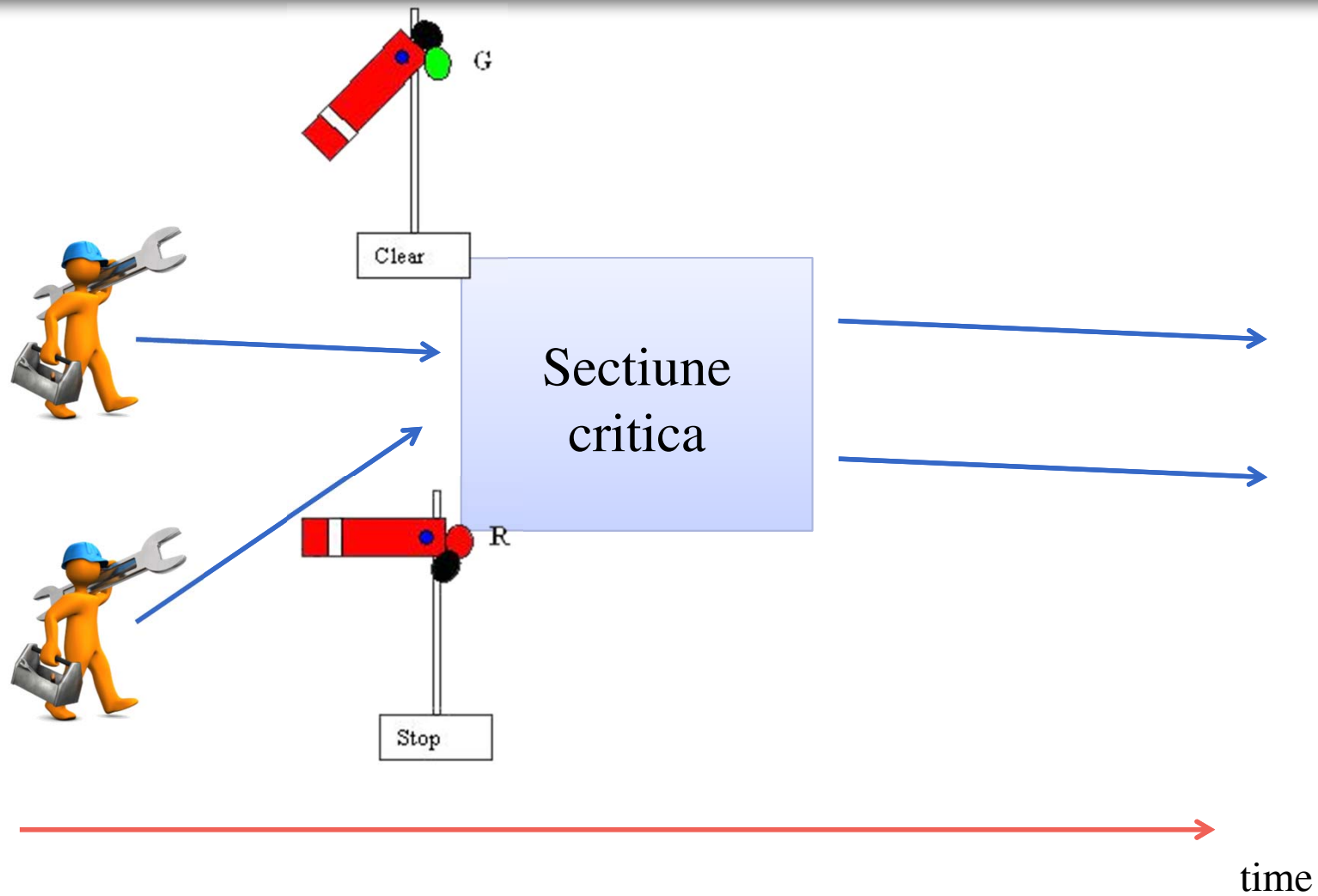
Secțiuni critice

- Problema
 - ♦ Fiecare proces $P(i)$ al unei colecții de procese $P(i:1..n)$ execută ciclic o **secțiune critică** în care are acces exclusiv la anumite resurse partajate urmată de o secțiune necritică în care folosește doar resurse locale.

- Soluție

```
var mutex: sem := 1;  
co P(i:1..n) ::  
  do true ->  
    P(mutex); /* acaparare secțiune critică */  
    Secțiune critică;  
    V(mutex); /* eliberare secțiune critică */  
    Secțiune necritică  
  od  
oc
```

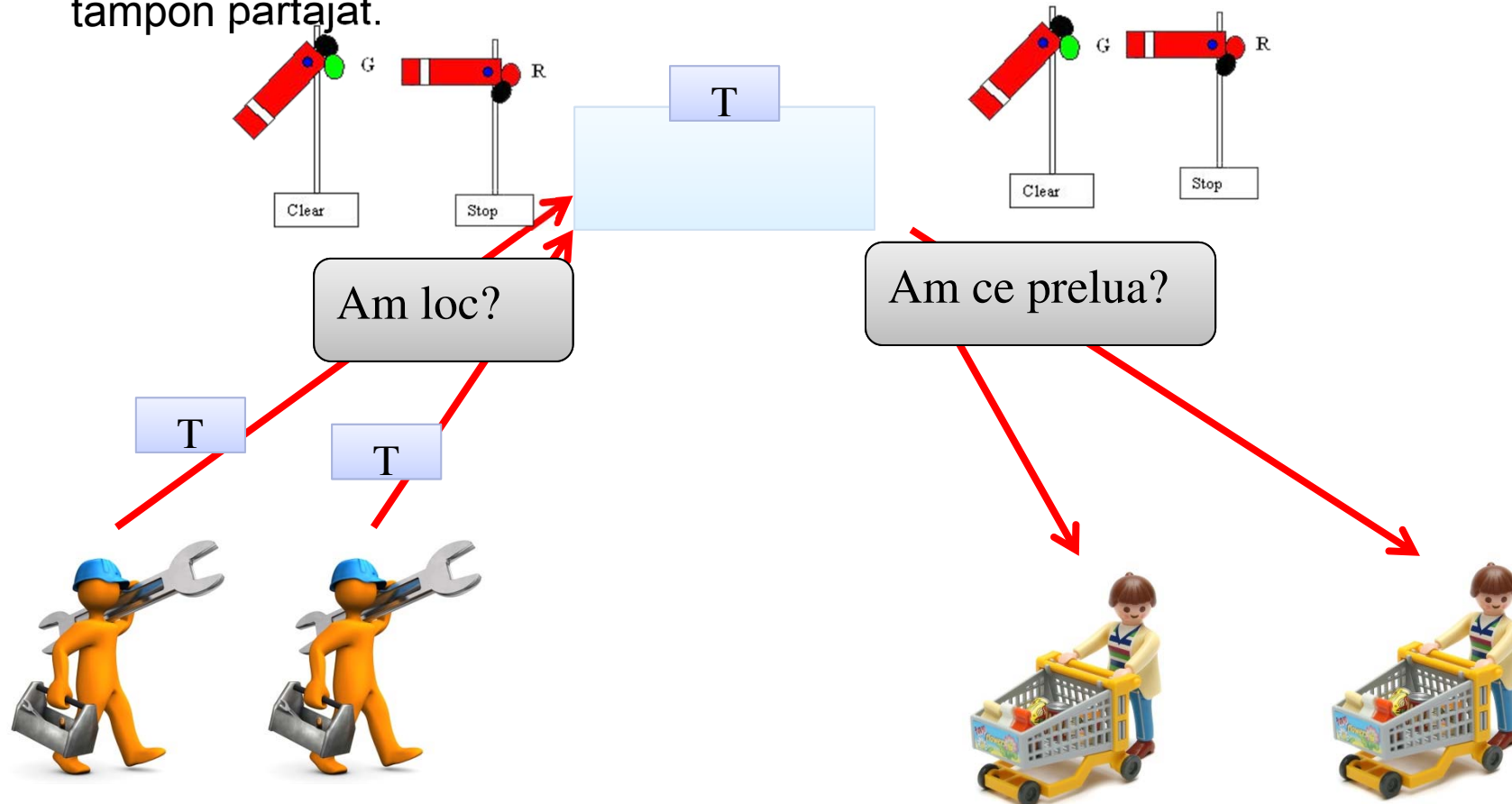
Exemplu executie...



Producători și consumatori

- Problema

- Se consideră M producători și N consumatori care comunică printr-un singur tampon partajat.



Producători și consumatori

- Soluția

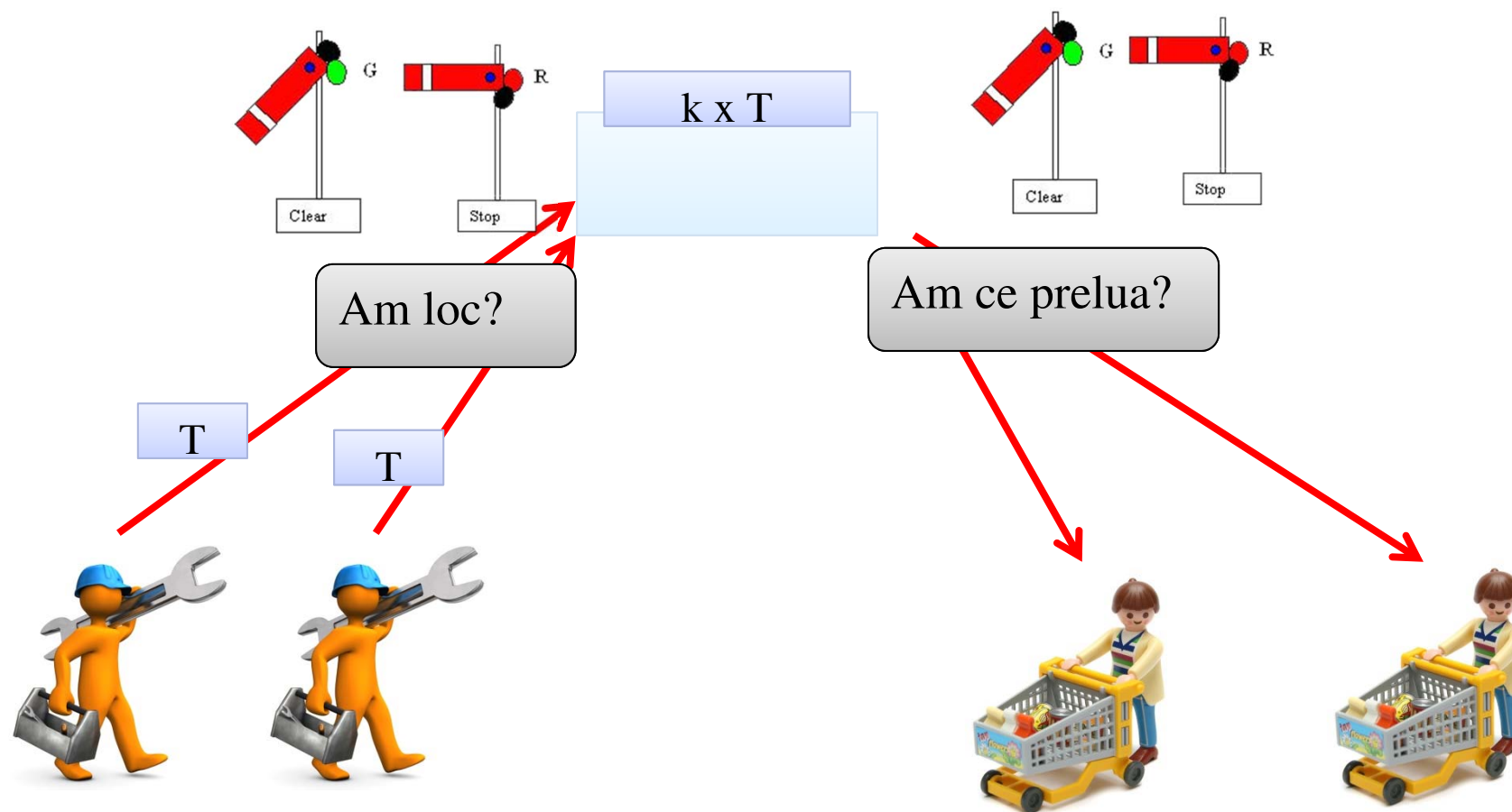
```
var buf: T; /* T - tipul datelor */
var gol: sem := 1, plin: sem := 0; /* sem binare */
co Producător (i: 1..M) ::      co Consumator (i: 1..N) ::
    var v: T;                      var w: T;
    do true ->                    do true ->
        v := produce();           P(plin);
        P(gol);                   w := buf;
        buf := v;                 V(gol);
        V(plin);                  consumă(w);
    od;                             od;
oc                                  oc
```

semnalizare

Comunicarea producător și consumator prin tampon limitat

```
var buf: array [1:k] of T;  
var gol: sem := k, plin: sem := 0; /* semafoare generale */  
co Producător::  
    var v: T;  
    var ultim: int := 1;  
    do true ->  
        v := produce();  
        P(gol); /* există locuri goale? */  
        buf[ultim] := v;  
        ultim := ultim mod k + 1; /* circular */  
        V(plin);  
    od;  
Consumator::  
    var w: T;  
    var prim: int := 1;  
    do true ->  
        P(plin); /* există valori în buffer? */  
        w := buf[prim]; prim := prim mod k + 1;  
        V(gol);  
        consumă(w);  
    od;  
oc
```


Comunicarea producător și consumator prin tampon limitat



Mai mulți producători și mai mulți consumatori

```
var buf: array [1:k] of T;  
var prim: int := 1, ultim: int := 1;  
var gol: sem := k, plin: sem := 0;  
var mutexP: sem := 1, mutexC: sem := 1;
```

```
co Producător(i:1..M) ::
```

```
  var v: T;
```

```
  do true ->
```

```
    v := produce();
```

```
    P(gol);
```

```
    P(mutexP);
```

```
    buf[ultim] := v;
```

```
    ultim := ultim mod k + 1;
```

```
    V(mutexP);
```

```
    V(plin);
```

```
  od;
```

```
oc
```

```
co Consumator (i: 1..N) ::
```

```
  var w: T;
```

```
  do true ->
```

```
    P(plin);
```

```
    P(mutexC);
```

```
    w := buf[prim];
```

```
    prim := prim mod k + 1;
```

```
    V(mutexC);
```

```
    V(gol);
```

```
    consumă(w);
```

```
  od;
```

```
oc
```

secțiune critică

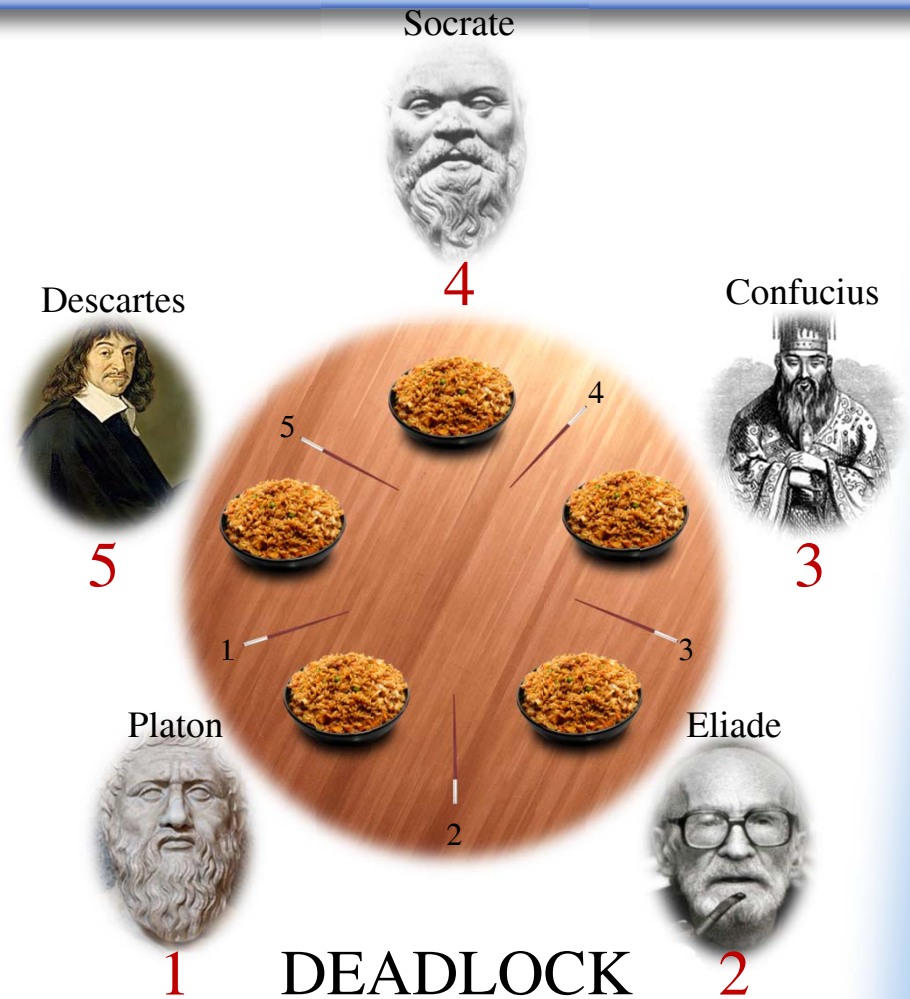
semnalizare

Problema filozofilor

```
co Filozof (i:1..5) ::  
do true ->  
    ia bețișoare;  
    mănâncă;  
    eliberează bețișoare;  
    gândește;
```

od

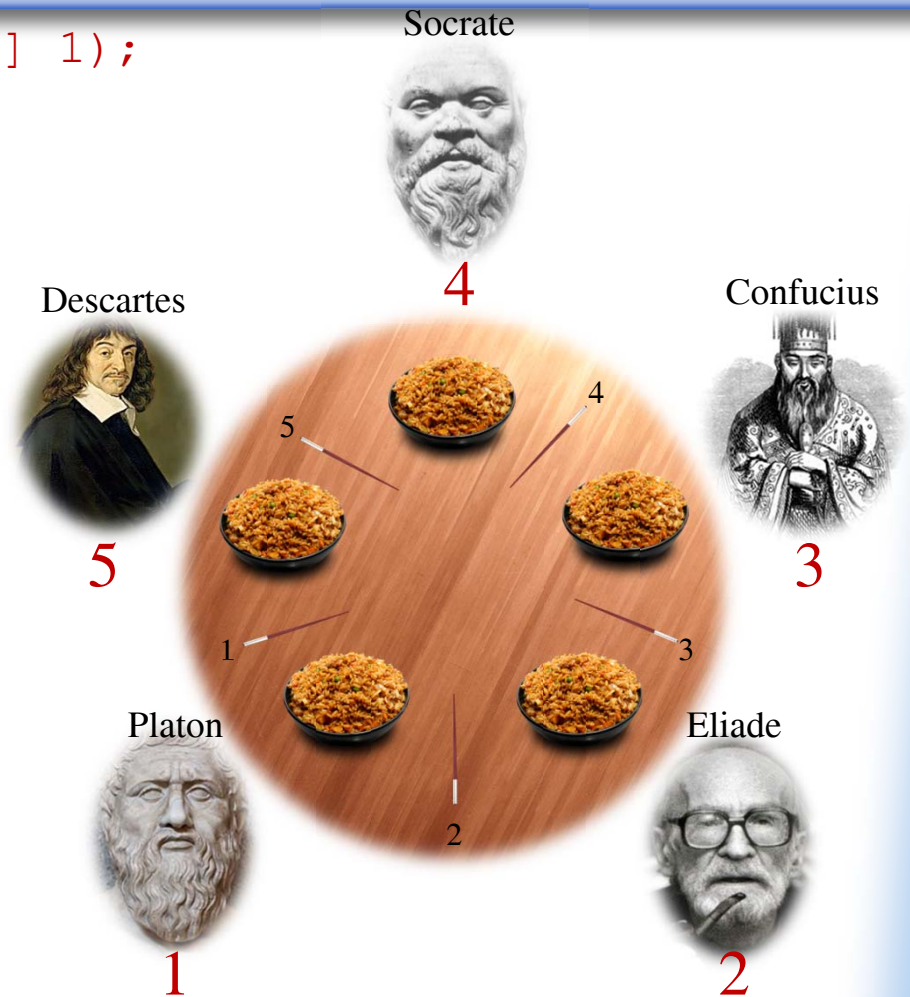
oc



Fiecare mai are nevoie de un bețișor

Problema filozofilor (2)

```
var b: array [1:5] of sem := ([5] 1);  
co Filozof (i:1..4)::  
    do true ->  
        P(b[i]); P(b[i+1]);  
        mănâncă;  
        V(b[i]); V(b[i+1]);  
        gândește;  
    od;  
Filozof (5)::  
    do true ->  
        P(b[1]); P(b[5]);  
        mănâncă;  
        V(b[1]); V(b[5]);  
        gândește;  
    od;  
co
```



Problema cititorilor și scriitorilor

Excludere mutuală

Zona de memorie
(resursa critica)



- 1) Un singur scriitor are dreptul sa scrie la un moment dat,
- 2) Doar daca nu exista un cititor curent care citeste



- 1) Un cititor poate citi din memorie, indiferent daca deja exista alti cititori ce deja citesc
- 2) Doar daca nu exista un scriitor care scrie



Problema cititorilor și scriitorilor

Excludere mutuală

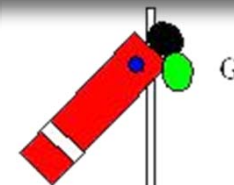
```
var rw: sem := 1;  
co Cititor (i: 1..m)::  
    do true ->  
        P(rw);  
        citește din resursa comună;  
        V(rw);  
    od;  
Scriitor (j: 1..n)::  
    do true ->  
        P(rw);  
        scrie în resursa comună;  
        V(rw);  
    od;  
oc
```

Excludere mutuala strica !
Un singur proces are voie sa
execute opeartii pe resursa
critica...

Zona de memorie
(resursa critica)



Stop



Clear



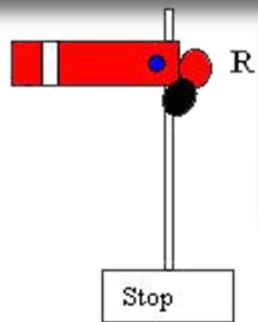
Problema cititorilor si scriitorilor

Excludere mutuală (2)

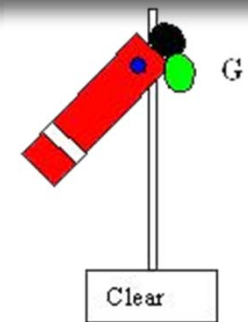
```
var nr: int := 0; mutexR: sem := 1, rw: sem := 1;
co Cititor (i: 1..m)::
  do true ->
    → P(mutexR);
    nr := nr + 1;
    if nr = 1 -> P(rw) fi; /* dacă primul cititor */
    → V(mutexR);
    citește din resursa comună;
    → P(mutexR);
    nr := nr - 1;
    if nr = 0 -> V(rw) fi; /* dacă ultimul cititor */
    → V(mutexR);
  od;
Scriitor (j: 1..n)::
  do true ->
    P(rw);
    scrie în resursa comună;
    V(rw);
  od;
oc
```

sețiune critica

instrucțiune cu garda



Zona de memorie
(resursa critica)



nr = 0
nr = 1
nr = 2



Problema cititorilor și scriitorilor

Sincronizare condiționată

- Invariant global:

RW: $(nr == 0 \mid \mid nw == 0) \ \&\& \ nw \leq 1$

- Proces Reader:

- ♦ Incrementarea **nr** e condiționată de $(nw == 0)$

- Proces Writer:

- ♦ Incrementarea **nw** e condiționată de $(nr == 0 \ \&\& \ nw == 0)$

- Decrementarea nu trebuie condiționată

Problema cititorilor și scriitorilor

Sincronizare condiționată (2)

```
var nr: int := 0, nw: int := 0;
co Cititor (i: 1..m)::
    do true ->
        <await (nw == 0) -> nr := nr + 1>
        citește din resursa comună;
        <nr := nr - 1>
    od;
Scriitor (j: 1..n)::
    do true ->
        <await (nr == 0 && nw == 0) -> nw := nw + 1>
        scrie în resursa comună;
        <nw := nw - 1>
    od;
oc
```

Problema cititorilor și scriitorilor

Sincronizare condiționată (3)

Politici:

- noile cereri de la cititori sunt întârziate dacă un scriitor așteaptă
- un cititor întârziat este trezit doar dacă nu există un scriitor în așteptare.

```
var nr: int := 0; /* nr. cititori care folosesc resursa */  
    nw: int := 0; /* nr. scriitori care folosesc resursa */  
var e: sem := 1; /* intrare secțiune atomică */  
    r: sem := 0; /* așteaptă nw == 0 */  
    w: sem := 0; /* așteaptă ca nw == 0 și nr == 0 */  
var dr: int := 0; /* nr. cititori întârziați */  
    dw: int := 0; /* nr. scriitori întârziați */
```

Problema cititorilor și scriitorilor

Sincronizare condiționată (4)

Split binary semaphore

- Folosit pentru a implementa atât **excluderea mutuală** cât și **sincronizarea condiționată**.
- Semafoarele **e**, **r** și **w** formează împreună un semafor **splitat** (*split binary semaphore*):
 - ♦ cel mult un semafor este 1 la un moment dat $0 \leq e + r + w \leq 1$
 - ♦ fiecare cale de execuție începe cu un P și se termină cu un singur V
 - ♦ instrucțiunile între P și V se execută în excludere mutuală.
- Tehnica se numește **pasarea ștafetei**:
 - ♦ Inițial un semafor este 1 și un proces poate prelua ștafeta printr-o operație P asupra semaforului
 - ♦ când un proces deține ștafeta (se execută într-o secțiune critică și toate semafoarele sunt 0), el poate pasa ștafeta altui proces printr-o operație V asupra unuia din cele trei semafoare.

Problema cititorilor și scriitorilor

Sincronizare condiționată (5)

```
co Cititor (i: 1..m) ::  
  do true ->  
    P(e);  
    if nw > 0 or dw > 0 ->  
      dr := dr + 1; V(e); P(r);  
    fi;  
    nr := nr + 1;  
    if dr > 0 -> dr := dr - 1; V(r);  
    [] dr == 0 -> V(e);  
    fi;  
    citește din resursa comună;  
    P(e);  
    nr := nr - 1;  
    if nr == 0 and dw > 0 -> dw := dw - 1; V(w);  
    [] nr > 0 or dw == 0 -> V(e);  
    fi;  
  od;  
oc
```

Problema cititorilor și scriitorilor

Sincronizare condiționată (6)

```
co Scriitor (j: 1..n)::
  do true ->
    P(e);
    if nr > 0 or nw > 0 ->
      dw := dw + 1; V(e); P(w)
    fi;
    nw := nw + 1;
    V(e);
    scrie în resursa comună;
    P(e);
    nw := nw - 1;
    if dr > 0 and dw == 0 -> dr := dr - 1; V(r);
    [] dw > 0 -> dw := dw-1; V(w);
    [] dr == 0 and dw == 0 -> V(e);
    fi;
  od;
oc
```

Problema bărbierului

- Problema:

- ◆ O frizerie cu un bărbier, un scaun de bărbier, n scaune de așteptare.
- ◆ Când nu sunt clienți, bărbierul doarme.
- ◆ Când sosește un client fie trezește bărbierul, fie așteaptă dacă acesta e ocupat.
- ◆ Dacă toate scaunele sunt ocupate, clientul pleacă.

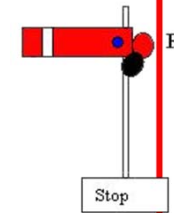
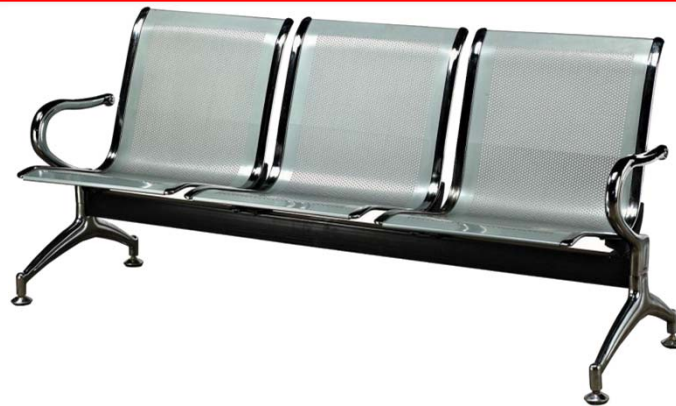


Ce se poate intampla rau?

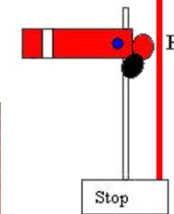


<https://www.youtube.com/watch?v=APfn72TeUkA>

Pas 1

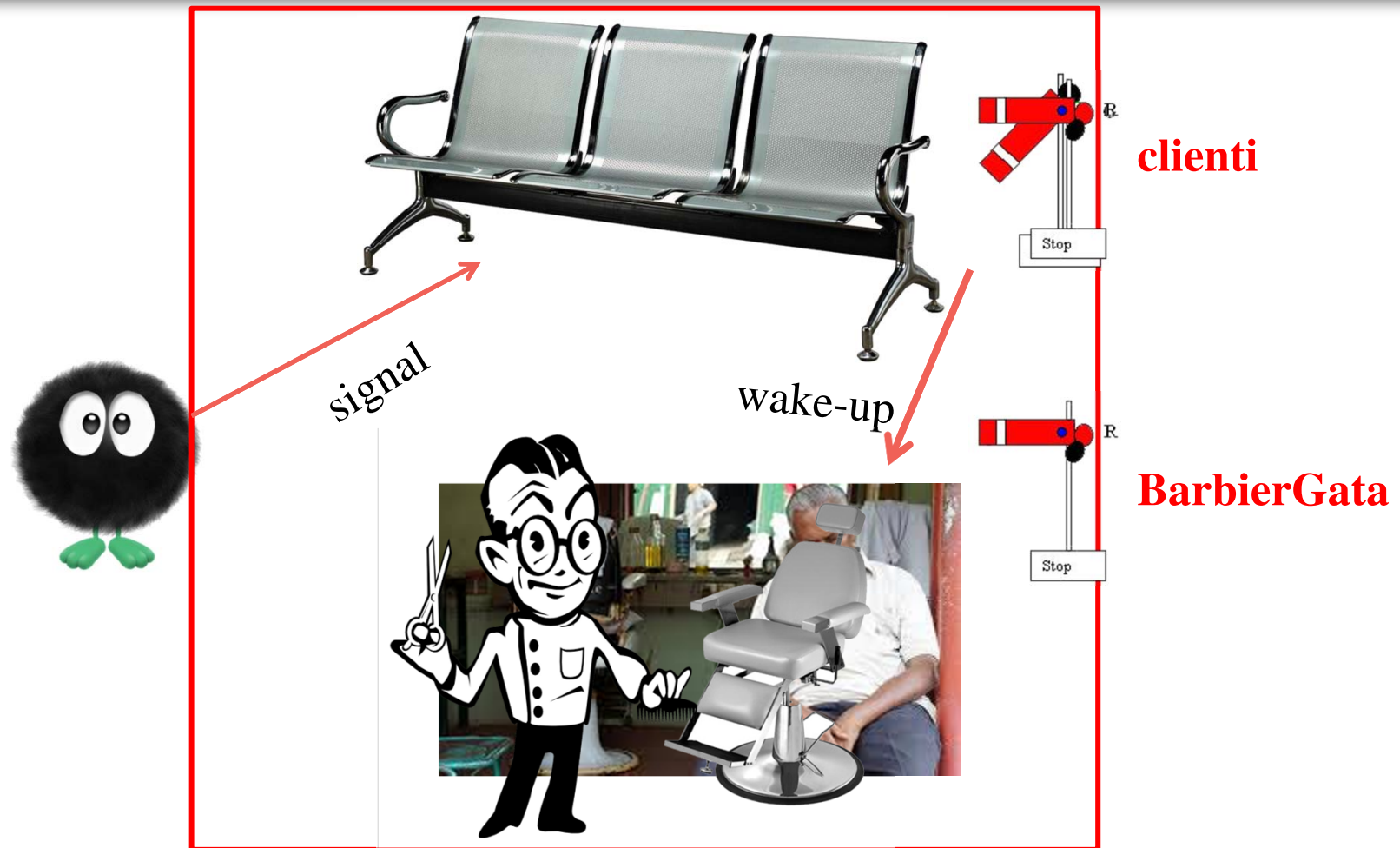


clienti

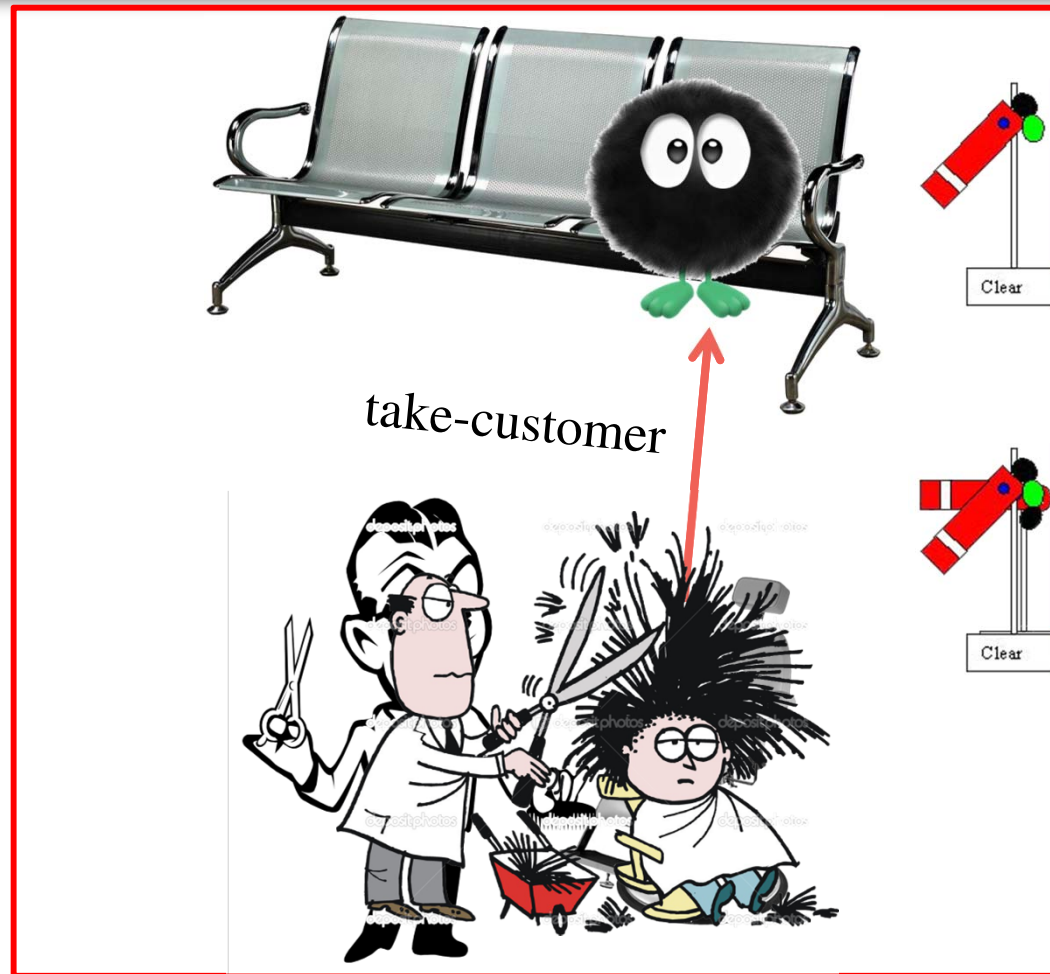


BarbierGata

Pas 2



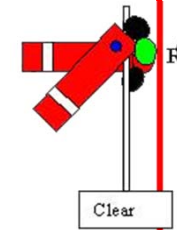
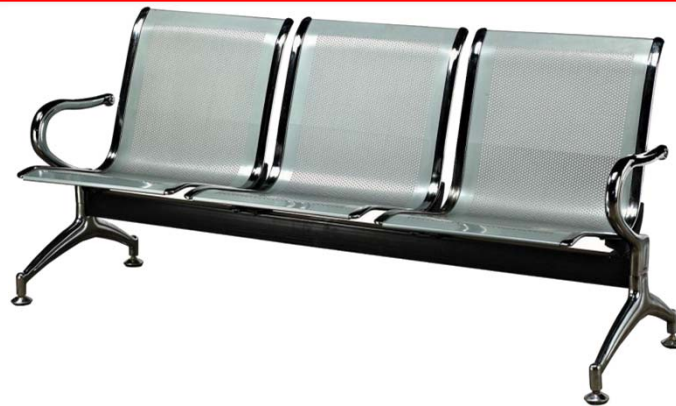
Pas 3



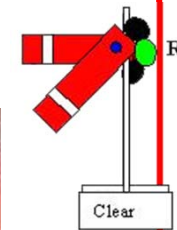
clienti

BarbierGata

Pas 4



clienti



BarbierGata

Problema bărbierului (2)

```
var NumărScauneLibere: int := n,  
    Clienți: sem := 0,  
    BărbierGata: sem := 0,  
    Scaune: sem:= 1;
```

```
co Bărbier::
```

```
  do true ->
```

```
    P(Clienți);
```

```
    P(Scaune);
```

```
    NumărScauneLibere++;
```

```
    V(BărbierGata);
```

```
    V(Scaune);
```

```
    /* Bărbierul tunde... */
```

```
  od
```

```
co
```

sincronizare cu Client

/* se caută un client;

dacă există, este chemat */

/* are client, va modifica

NumărScauneLibere */

/* se eliberează un scaun */

/* bărbierul e gata să tundă */

/* a terminat de modificat

NumărScauneLibere */

Problema bărbierului (2)

```
co Client(i:1..m)
  do true ->
    P(Scaune);
    if NumărScauneLibere > 0 ->
      NumărScauneLibere--;
      V(Clienți);
    V(Scaune);
    P(BărbierGata);

    /* Clientul e tuns... */
    [] NumărScauneLibere == 0 ->
      V(Scaune);
      /* Clientul pleacă netuns... */
  fi
od
oc
```

sincronizare cu Barbier

atomicitate

/* clientul încearcă să obțină un scaun liber */

/* există un scaun disponibil? */

/* clientul se așază */

/* se anunță bărbierul că s-a ocupat un scaun */

/* se renunță la accesul asupra NumărScauneLibere */

/* clientul așteaptă pentru a fi tuns */

/* nu sunt scaune libere */

/* eliberează mutexul */

Sumar

- Dezvoltarea algoritmilor folosind variabile partajate (MIMD)
- Semafoare
- Secțiuni critice
- Probleme:
 - ♦ Producători și consumatori
 - ♦ Problema filozofilor
 - ♦ Problema cititorilor și scriitorilor
 - ♦ Problema bărbierului

Quiz

- Cum ati implementa o *bariera* folosind semafoare?
- Ex. (suma elementelor unui vector):

```
var a: array [1:n] of int;  
co suma (k:1..n) ::  
    fa j := 1 to sup(log2 n) ->  
        if k mod 2j = 0 ->  
            a[k] := a[k-2j-1] + a[k]  
        fi  
    barrier  
    af  
oc
```

Nu dati inca lucrarea ...

Un pic de practice: Cigarette smokers problem

- Un agent și trei fumători
- Fumătorii:
 - ♦ Așteaptă ingrediente (tutun, hârtie, chibrit)
 - ♦ Confectionează țigară
 - ♦ Fumează
- Agentul deține toate 3 ingredientele
- Un fumător are tutun, un altul hârtie, al 3-lea chibrituri)
- Agentul selectează două ingrediente (random) pe care le dă fumătorilor
 - ♦ Doar fumătorul ce are nevoie de exact acele 2 ingrediente trebuie să le preia
 - ♦ Agentul nu poate semnaliza exact acelui fumător pentru că nu știe care fumător e care, respectiv ingredientele sunt random extrase

Cigarette smokers problem

```
var tobacco: sem := 0;
    paper : sem := 0;
    match  : sem := 0;
    agent  : sem := 1;
co Agent::
    do draw1 -> P(agent); V(tobacco); V(paper);
    [] draw2 -> P(agent); V(paper); V(match);
    [] draw3 -> P(agent); V(tobacco); V(match)
    od
oc
co Smoker1::
    P(tobacco); P(paper); V(agent);
oc
co Smoker2::
    P(paper); P(match); V(agent);
oc
co Smoker3::
    P(tobacco); P(match); V(agent);
oc
```

Functioneaza ???

Cigarette smokers problem - deadlock

OK!



P(match)
P(tobacco)



DEADLOCK!

P(tobacco) **P(paper)**
P(match)



Quiz

- Ganditi-va la o solutie pentru evitarea deadlock-ului...



Cigarette smokers problem

```
var tobacco: sem := 0;
    paper : sem := 0;
    match : sem := 0;
    agent : sem := 1;

    isTobacco : bool := false;
    isPaper : bool := false;
    isMatch : bool := false;

    tobaccoSem: sem := 0;
    paperSem : sem := 0;
    matchSem : sem := 0;

co Agent::
    do draw1 -> P(agent); V(tobacco); V(paper);
    [] draw2 -> P(agent); V(paper); V(match);
    [] draw3 -> P(agent); V(tobacco); V(match)
od
oc
```

Cigarette smokers problem

```
co PusherA::
  P(tobacco);
  P(e);
  if isPaper -> isPaper := false; V(matchSem);
  [] isMatch -> isMatch := false; V(paperSem);
  [] isPaper = isMatch = false -> isTobacco := true;
  fi
  V(e);
oc

co PusherB::
  P(match);
  P(e);
  if isPaper -> isPaper := false; V(tobaccoSem);
  [] isTobacco -> isTobacco := false; V(paperSem);
  [] isPaper = isTobacco = false -> isMatch := true;
  fi
  V(e);
oc

co PusherC::
  P(paper);
  P(e);
  if isTobacco -> isTobacco := false; V(matchSem);
  [] isMatch -> isMatch := false; V(tobaccoSem);
  [] isPaper = isMatch = false -> isPaper := true;
  fi
  V(e);
oc
```

Cigarette smokers problem

```
co SmokerWithTobacco::  
    P(tobaccoSem);  
    # makeCigarette  
    V(agent);  
    # smoke
```

```
oc
```

```
co SmokerWithPaper::  
    P(paperSem);  
    # makeCigarette  
    V(agent);  
    # smoke
```

```
oc
```

```
co SmokerWithMatch::  
    P(matchSem);  
    # makeCigarette  
    V(agent);  
    # smoke
```

```
oc
```



Posibila solutie... pentru bariera

```
var b: sem := 0; e : sem := 1;
    nb : int := 0;
co proc (k:1..n)::
    # enter barrier
    P(e);
    nb:=nb+1;
    if (nb = n) -> V(b);
    [] (nb > 1) -> P(b); V(b);
    [] (nb = 1) -> P(b);
fi
    nb:=nb-1;
    V(e);
    # exit barrier
```

if last
process to
enter barrier



if first
process to
enter barrier



oc

Întrebări?