



Arhitecturi Paralele

Access concurrent la memorie

Lect. Dr. Ing. Cristian Chilipirea – cristian.chilipirea@mta.ro








REMINDER

- Semafor/mutex(semafor binar)
 - La P
 - La V
- Zonă critică
 - Zonă dintre P și V pe același mutex pe un thread.
 - Garantează
- Barieră
 - Gatantează





REMINDER

- Semafor/mutex(semafor binar)
 - La P thread-ul așteaptă până ce semaforul are valori >0 .
 - La V se crește valoarea semaforului.
- Zonă critică
 - Zonă dintre P și V pe același mutex pe un thread.
 - Garantează – un singur thread poate să intre în zona critică la un moment dat.
- Barieră
 - Garantează – tot codul de dinainte de barieră, de pe orice thread, este executat înainte de orice cod de după barieră, de pe orice thread.







Producer - Consumer

EWD209 - 0

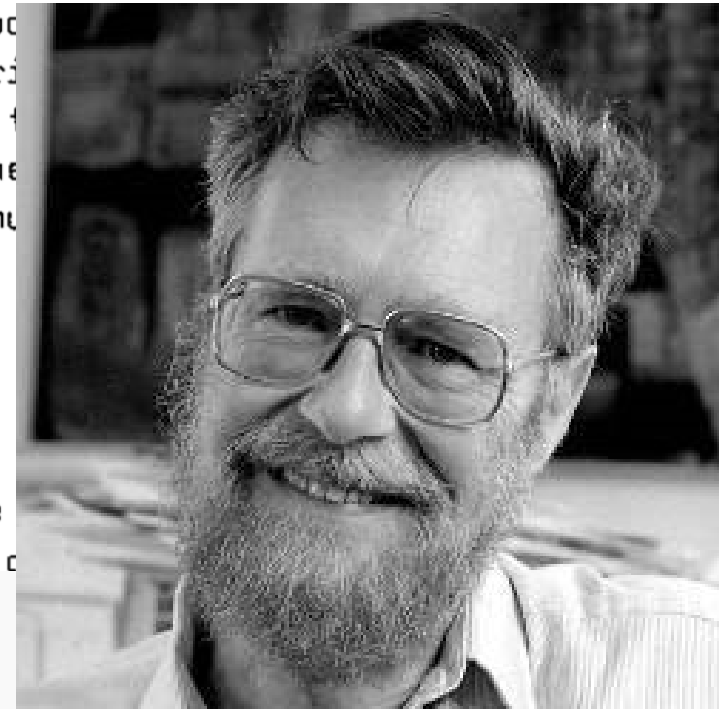
[EWD209.html](#)

A Constructive Approach to the Problem of Program Correctness.

Summary. As an alternative to methods by which the correctness of given programs can be established a posteriori, this paper proposes a method of program generation such as to produce a priori. The paper is treated to show the form that such a control structure comes from the field of parallel programming; the method is representative for the way in which a whole machine has actually been constructed.

Introduction.

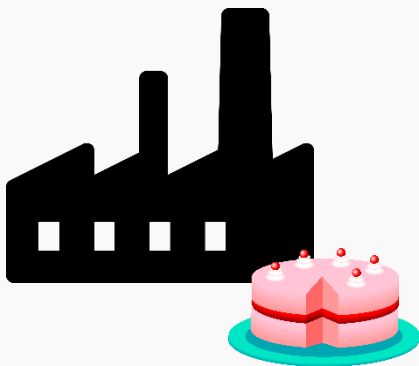
The more ambitious we become in our machine, the more becomes the problem of program correctness. The construction of a machine





Problema producător-consumator

Producător
Un Thread

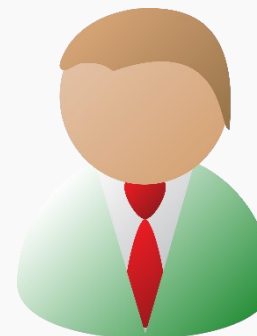


Pot să pun?
Da – bufferul este gol

Buffer



Consumator
Alt Thread

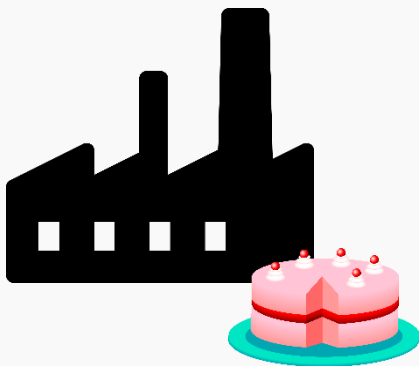


X Pot să iau?
Nu – nu este
nimic în buffer



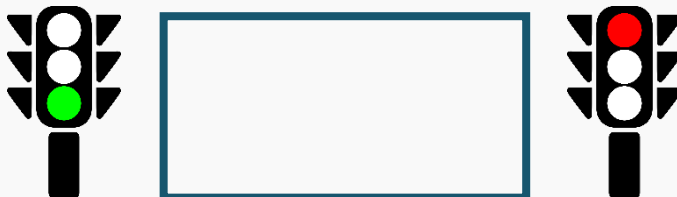
Problema producător-consumator

Producător

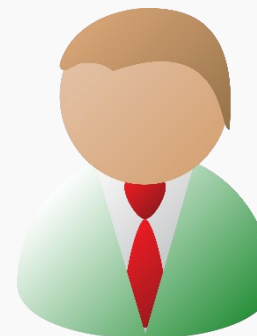


Pot să pun?

Buffer



Consumator

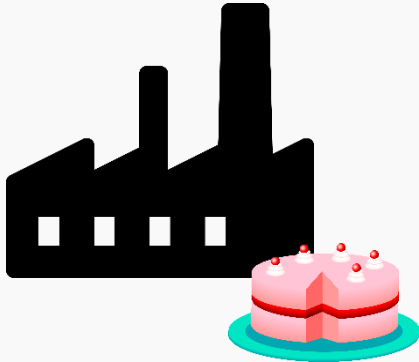


Pot să iau?



Problema producător-consumator

Producător

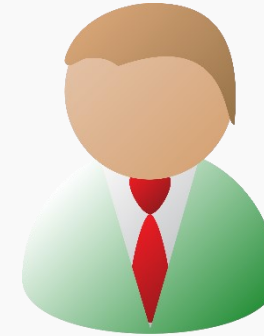


Pot să pun?

Buffer



Consumator

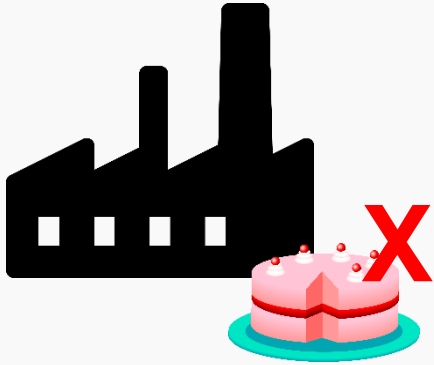


Pot să iau?



Problema producător-consumator

Producător

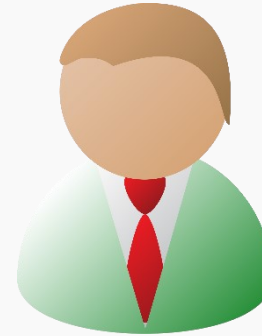


Pot să pun?

Buffer



Consumator

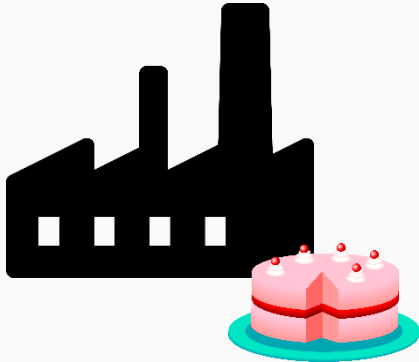


Pot să iau?



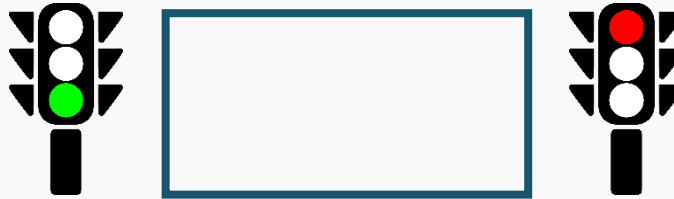
Problema producător-consumator

Producător



Pot să pun?

Buffer



Consumator

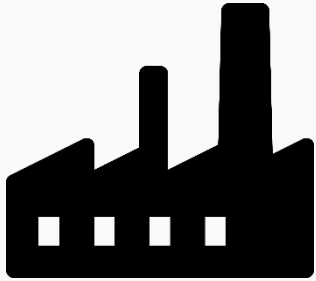


Pot să iau?



Problema producător-consumator

Producător



Pot să pun?

Buffer



Consumator



Pot să iau?

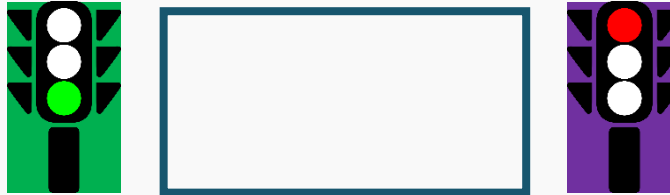


Problema producător-consumator

Producător

Buffer
B

Consumator



$B = EL;$

$EL = B;$

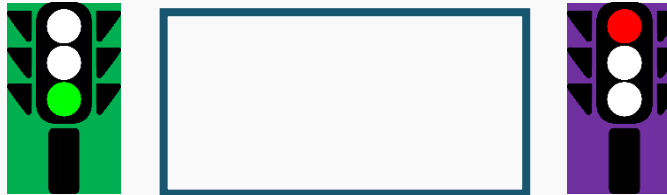


Problema producător-consumator

Producător

Buffer
B

Consumator



$B = EL;$

$EL = B;$

PROBLEME:

Read-Write conflict

Suprascrisiere element existent

Citire element inexistent

Duplicare de elemente



Problema producător-consumator

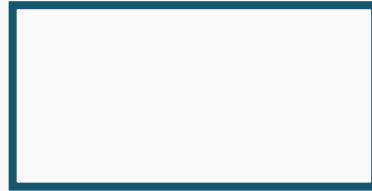
Producător

Buffer

Consumator

Full = 0
Empty = 1

P(**Empty**);



P(**Full**);

B=EL;

EL = B;

V(**Full**);

V(**Empty**);



Problema producător-consumator

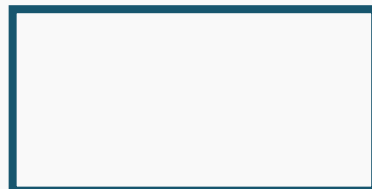
Producător

Buffer

Consumator

Full = 0
Empty = 1

Empty.lock();



Full.lock();

B=EL;

EL = B;

Full.unlock();

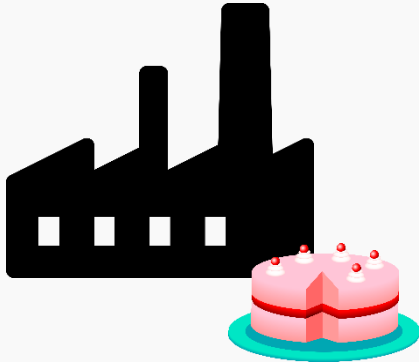
Empty.unlock();





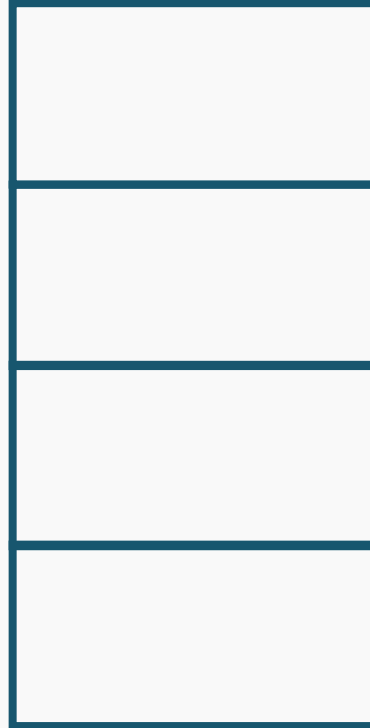
Problema producător-consumator

Producător

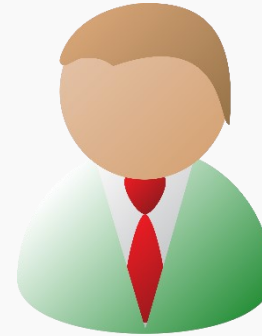


Pot să pun?

Buffer



Consumator

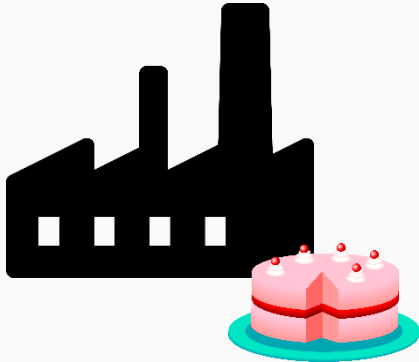


Pot să iau?



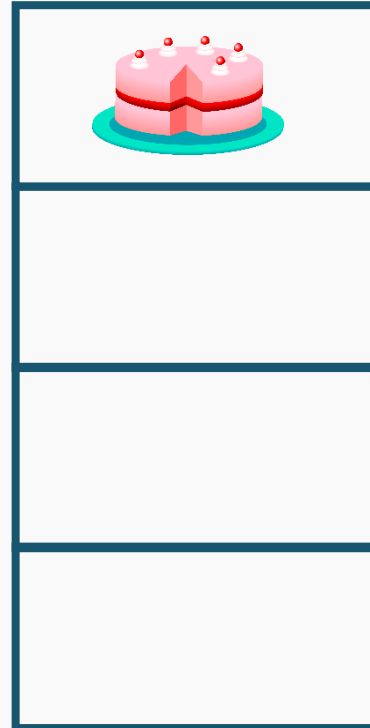
Problema producător-consumator

Producător



Pot să pun?

Buffer



Consumator

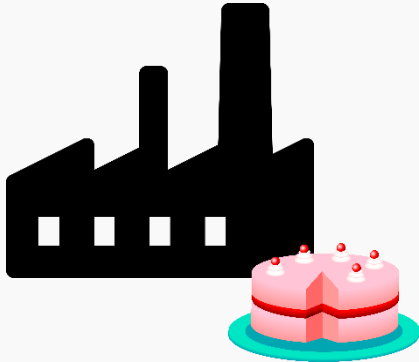


Pot să iau?



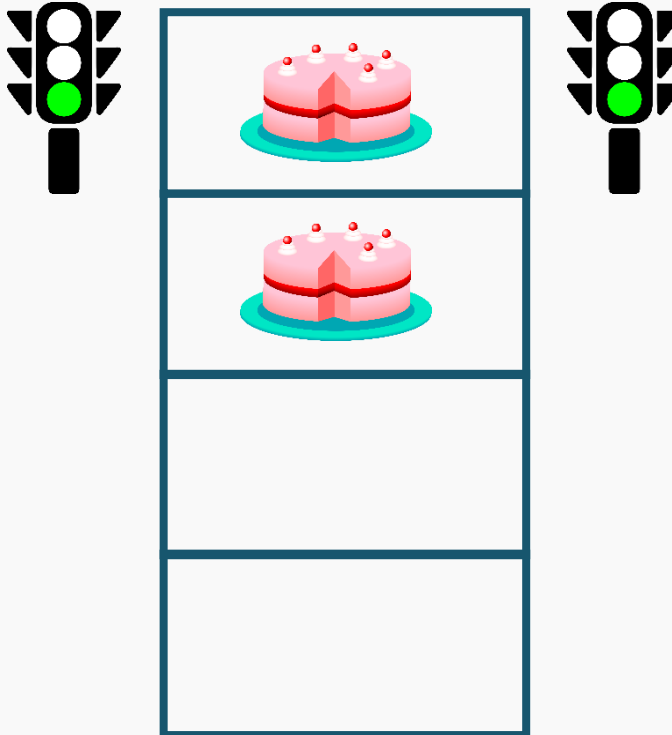
Problema producător-consumator

Producător

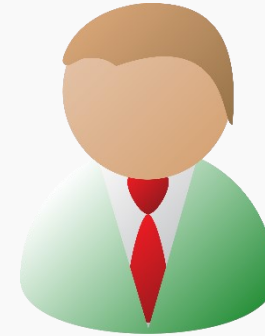


Pot să pun?

Buffer



Consumator

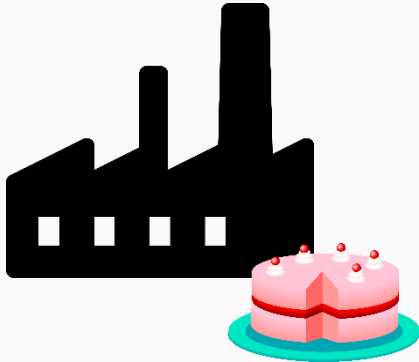


Pot să iau?



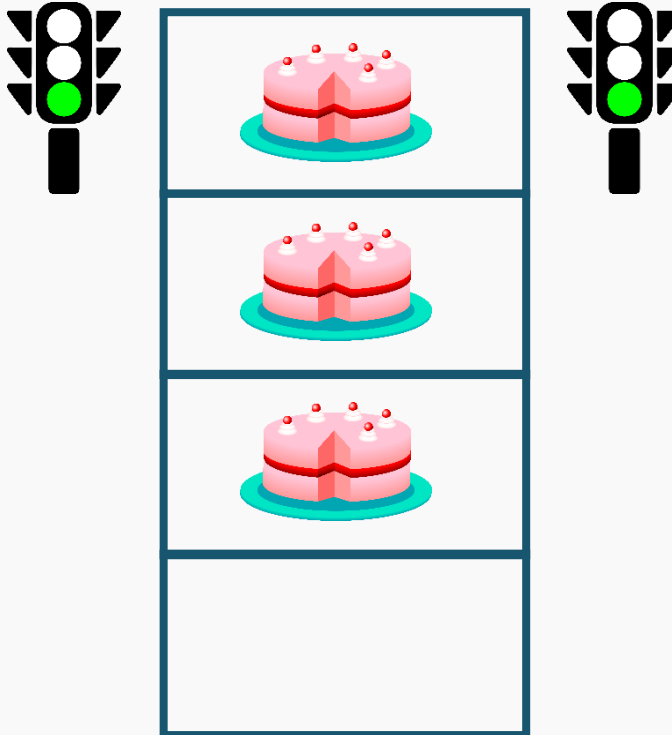
Problema producător-consumator

Producător

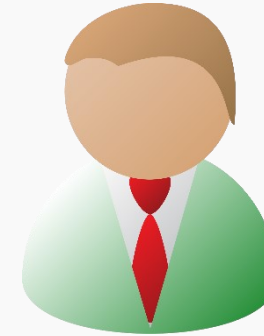


Pot să pun?

Buffer



Consumator

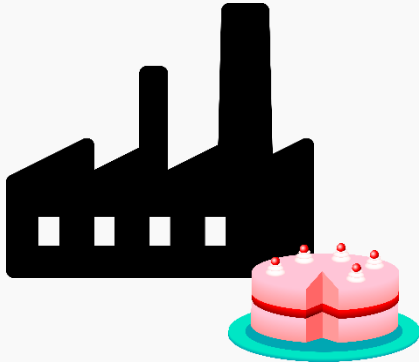


Pot să iau?



Problema producător-consumator

Producător



Pot să pun?

Buffer



Consumator

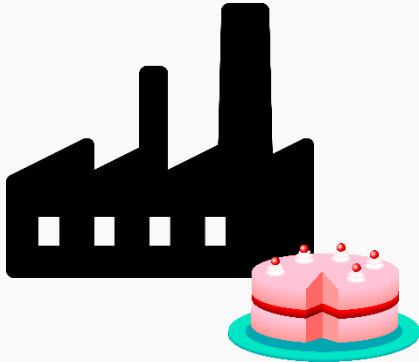


Pot să iau?



Problema producător-consumator

Producător



Pot să pun?

Buffer



Consumator

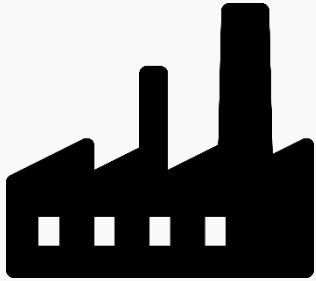


Pot să iau?



Problema producător-consumator

Producător



Pot să pun?

Buffer



Consumator

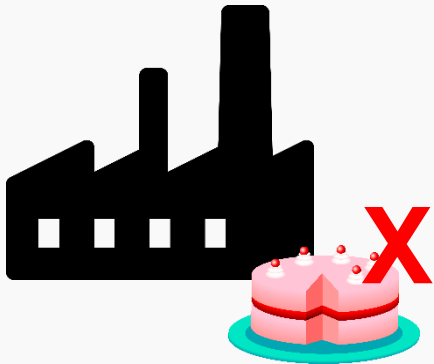


Pot să iau?



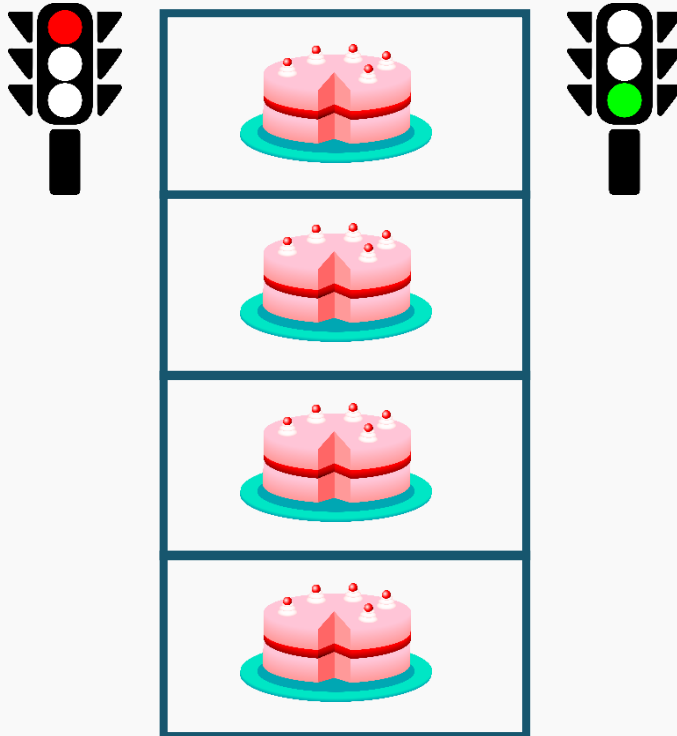
Problema producător-consumator

Producător



Pot să pun?

Buffer



Consumator

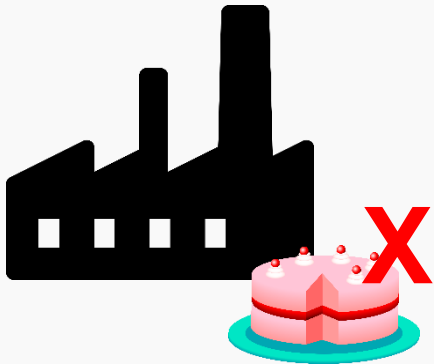


Pot să iau?



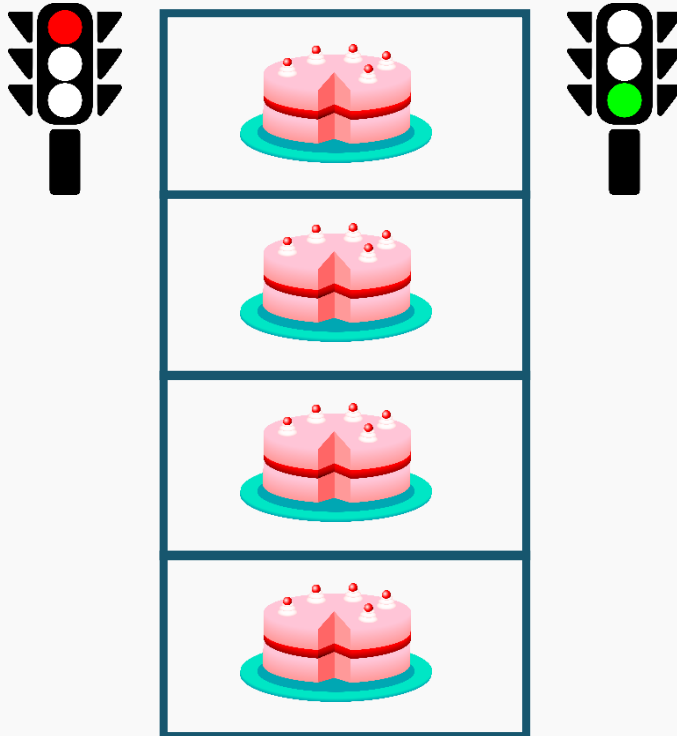
Problema producător-consumator

Producător

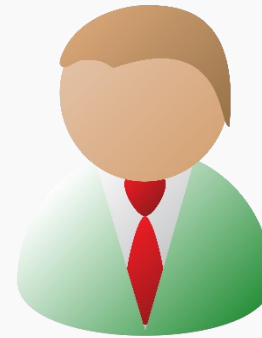


Pot să pun?

Buffer



Consumator

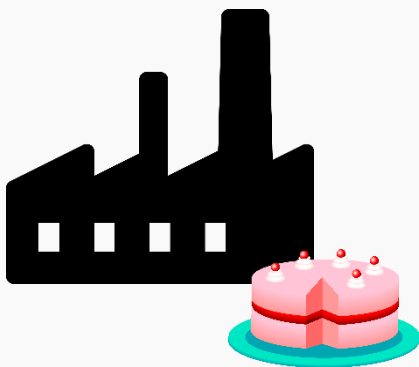


Pot să iau?



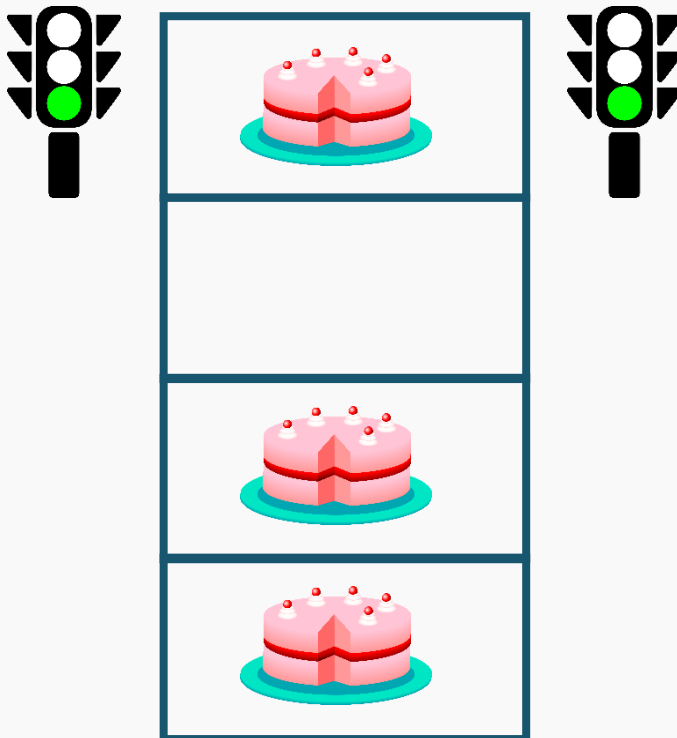
Problema producător-consumator

Producător

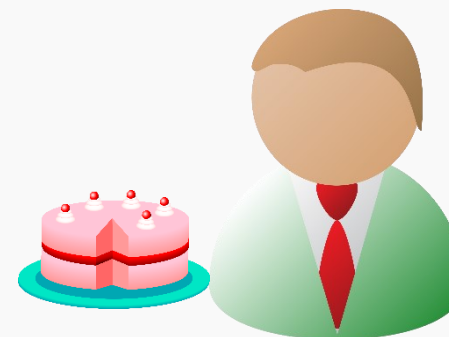


Pot să pun?

Buffer



Consumator

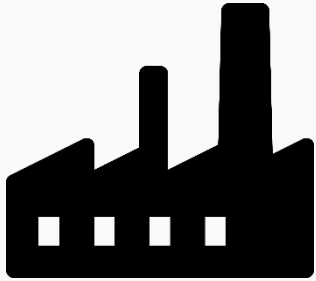


Pot să iau?



Problema producător-consumator

Producător

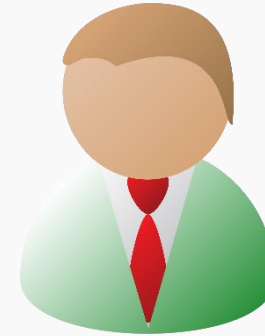


Pot să pun?

Buffer



Consumator

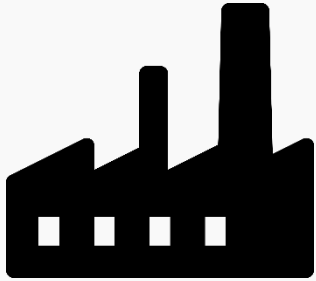


Pot să iau?



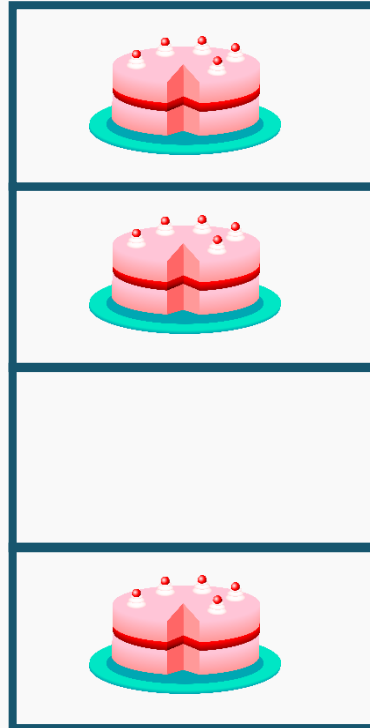
Problema producător-consumator

Producător



Pot să pun?

Buffer



Consumator

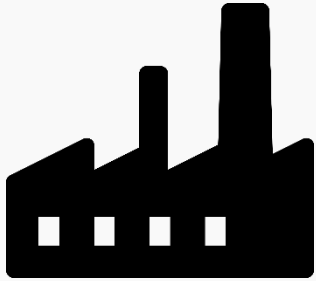


Pot să iau?



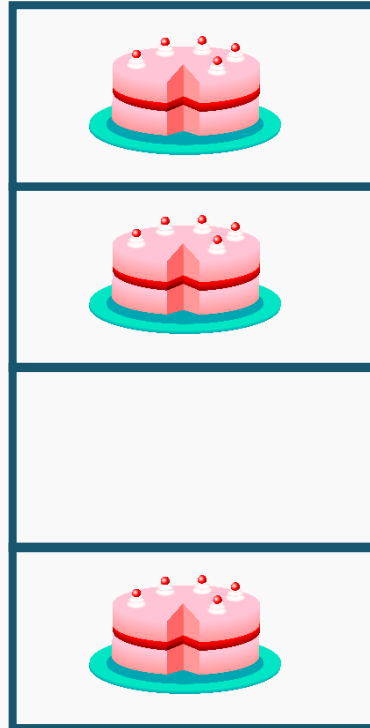
Problema producător-consumator

Producător

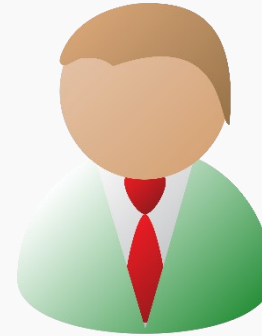


Pot să pun?

Buffer



Consumator

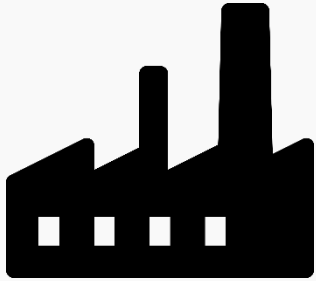


Pot să iau?



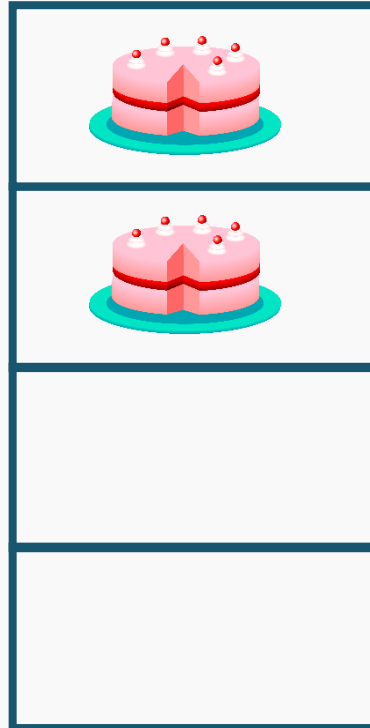
Problema producător-consumator

Producător



Pot să pun?

Buffer



Consumator

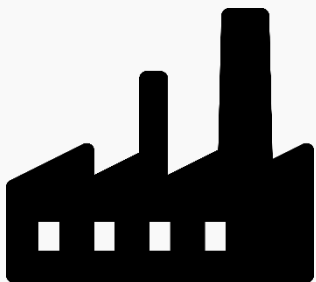


Pot să iau?



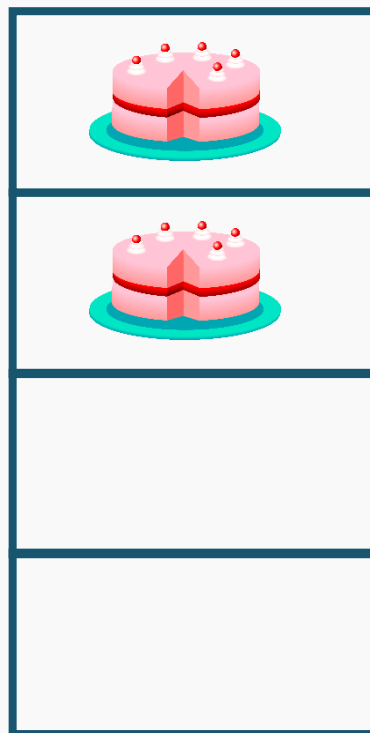
Problema producător-consumator

Producător

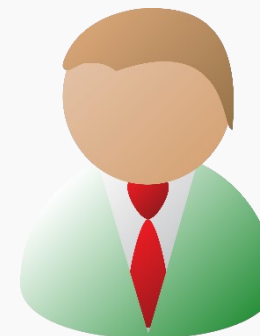


Pot să pun?

Buffer



Consumator

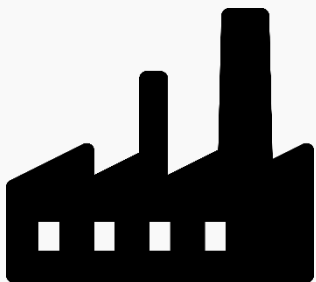


Pot să iau?



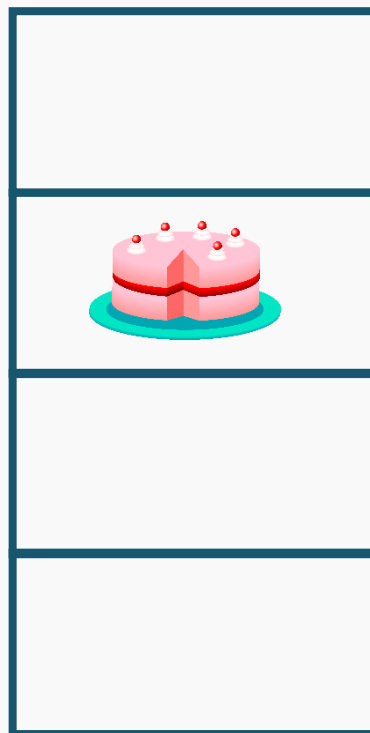
Problema producător-consumator

Producător

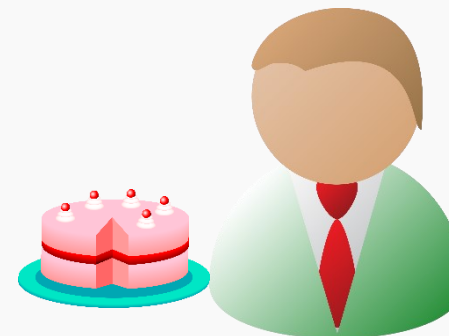


Pot să pun?

Buffer



Consumator

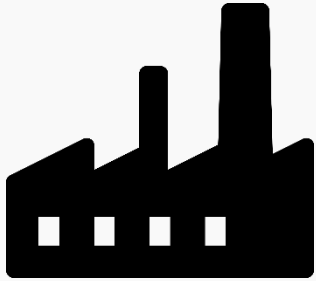


Pot să iau?



Problema producător-consumator

Producător

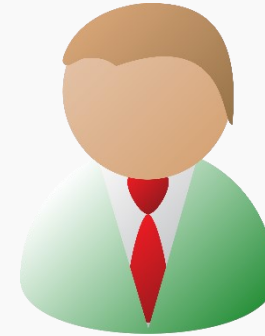


Pot să pun?

Buffer



Consumator

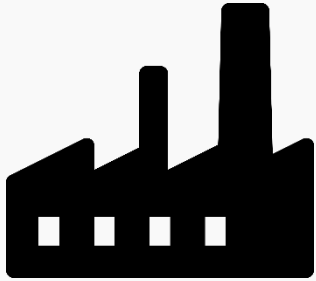


Pot să iau?



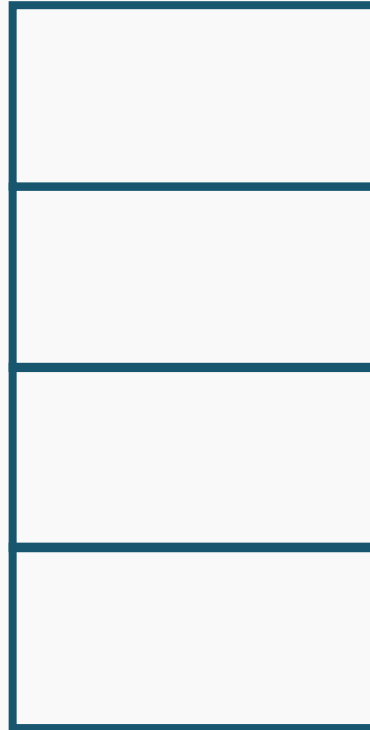
Problema producător-consumator

Producător



Pot să pun?

Buffer



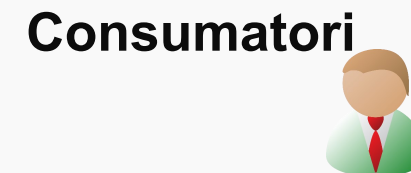
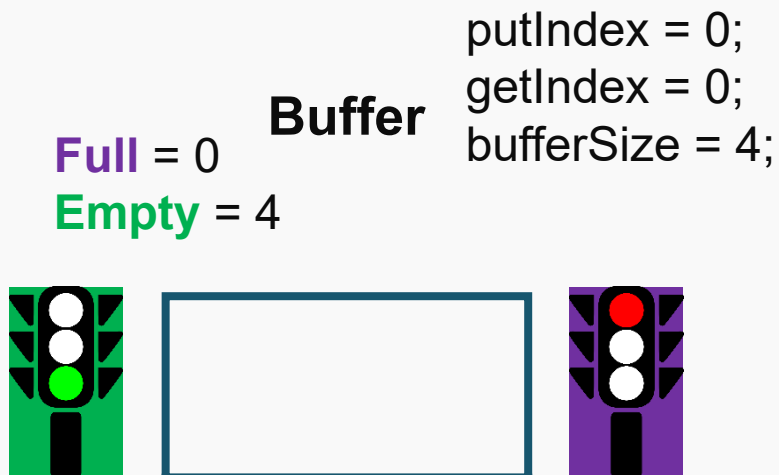
Consumator



Pot să iau?



Problema producător-consumator



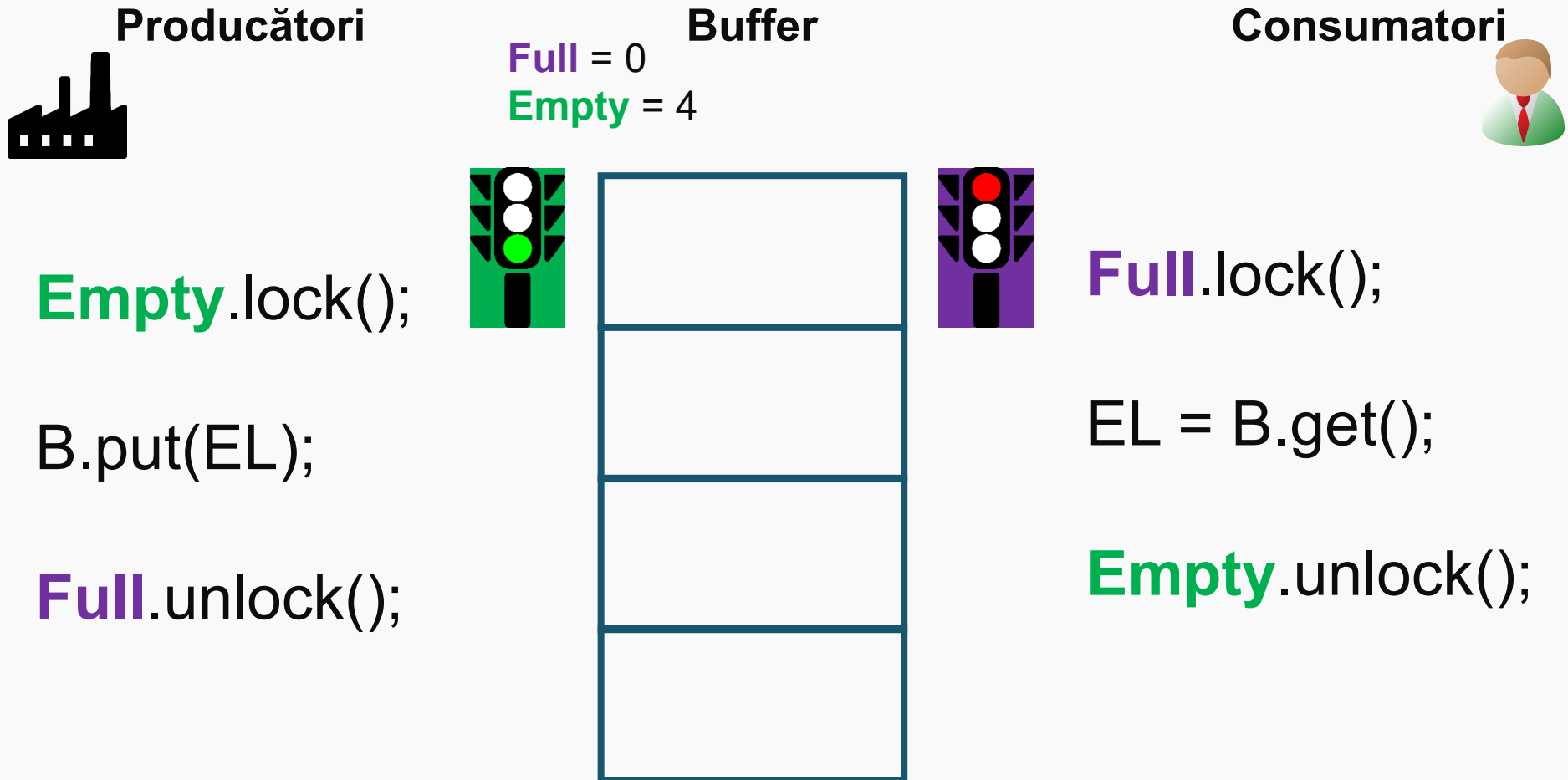
```
Empty.lock();  
B[putIndex]=EL;  
putIndex++;  
putIndex%=bufferSize;  
Full.unlock();
```

```
Full.lock();  
EL=B[getIndex];  
getIndex++;  
getIndex%=bufferSize;  
Empty.unlock();
```

Avem o problemă?



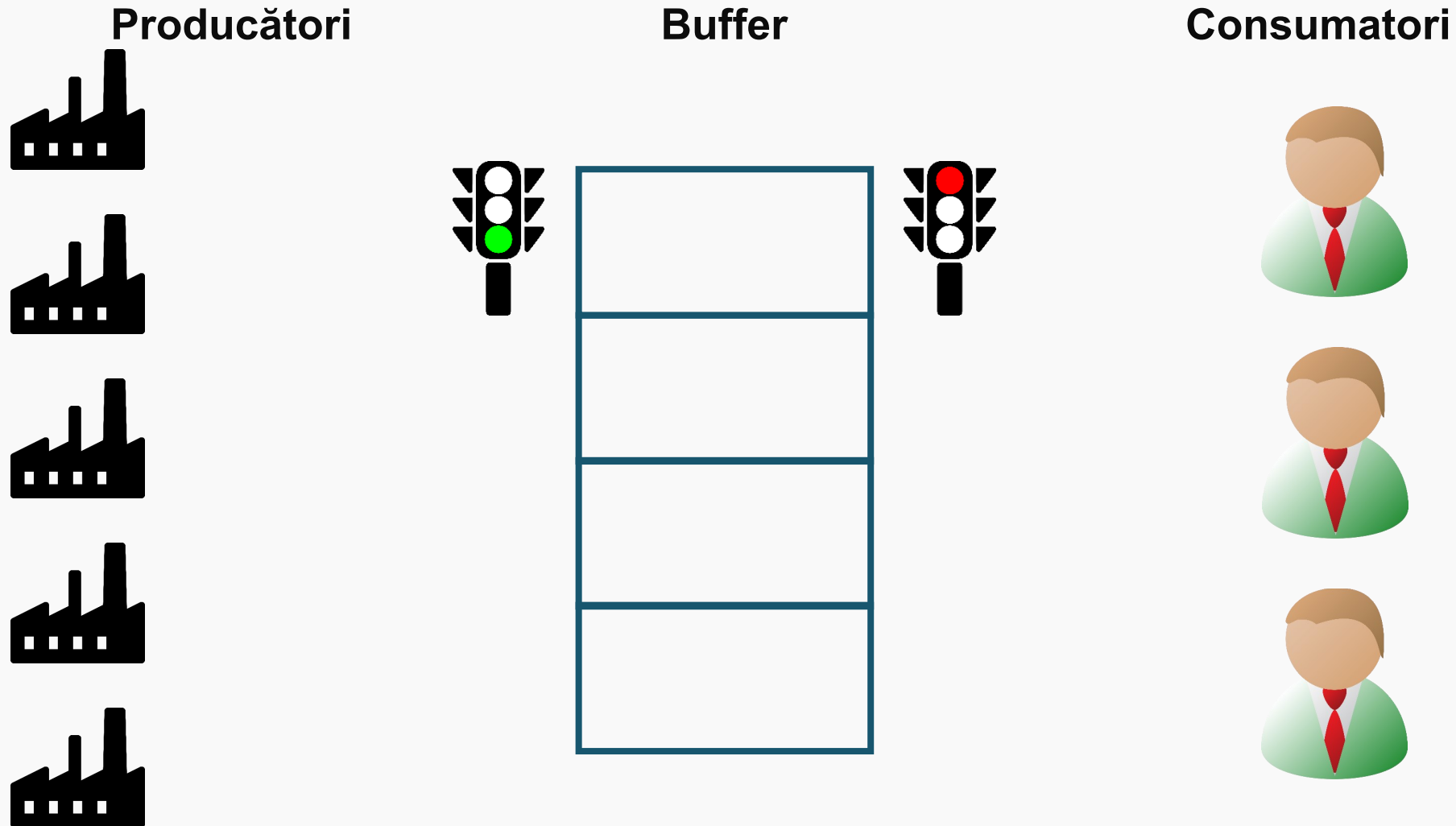
Problema producător-consumator



Avem o problemă?

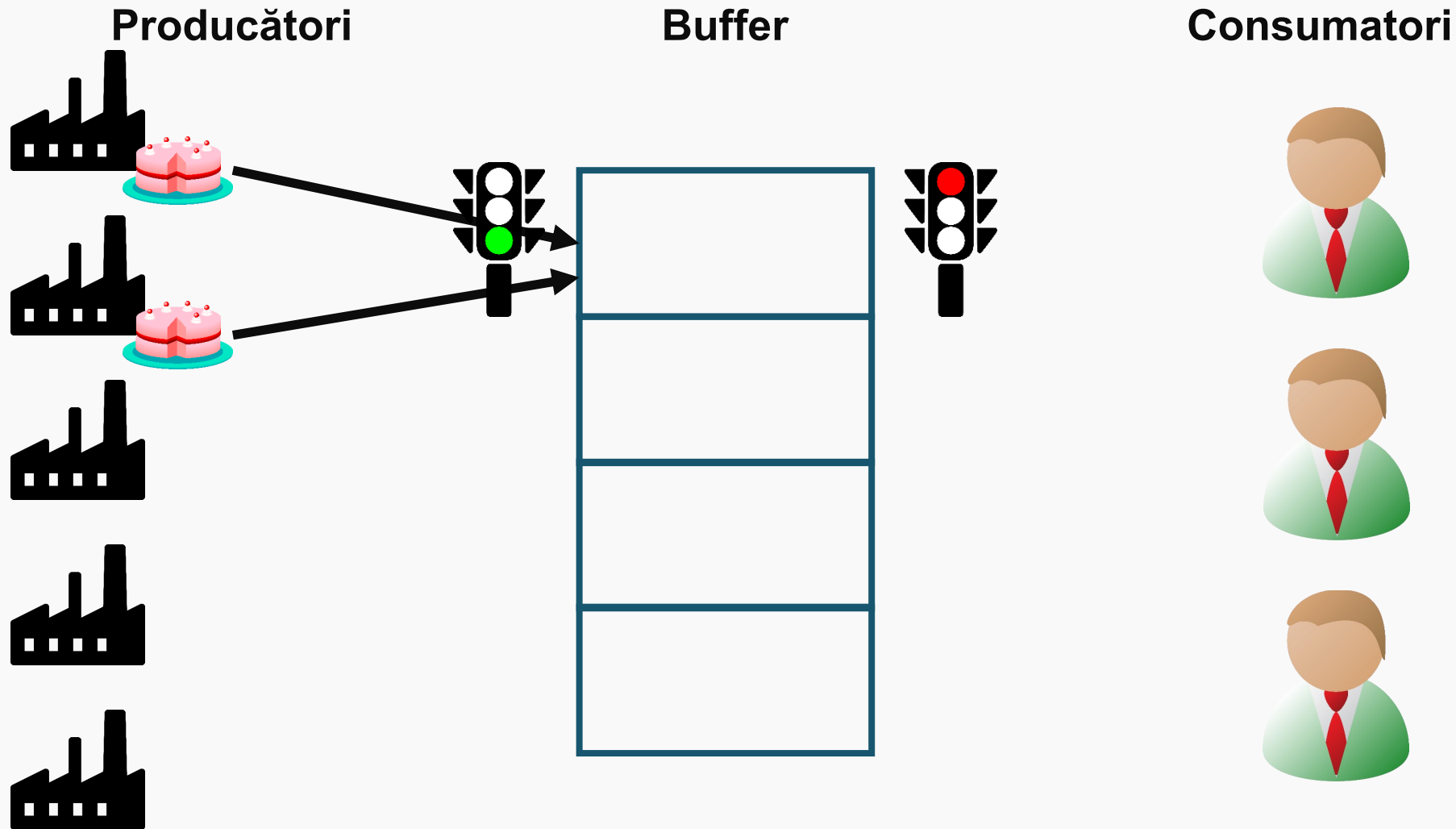


Problema producători-consumatori



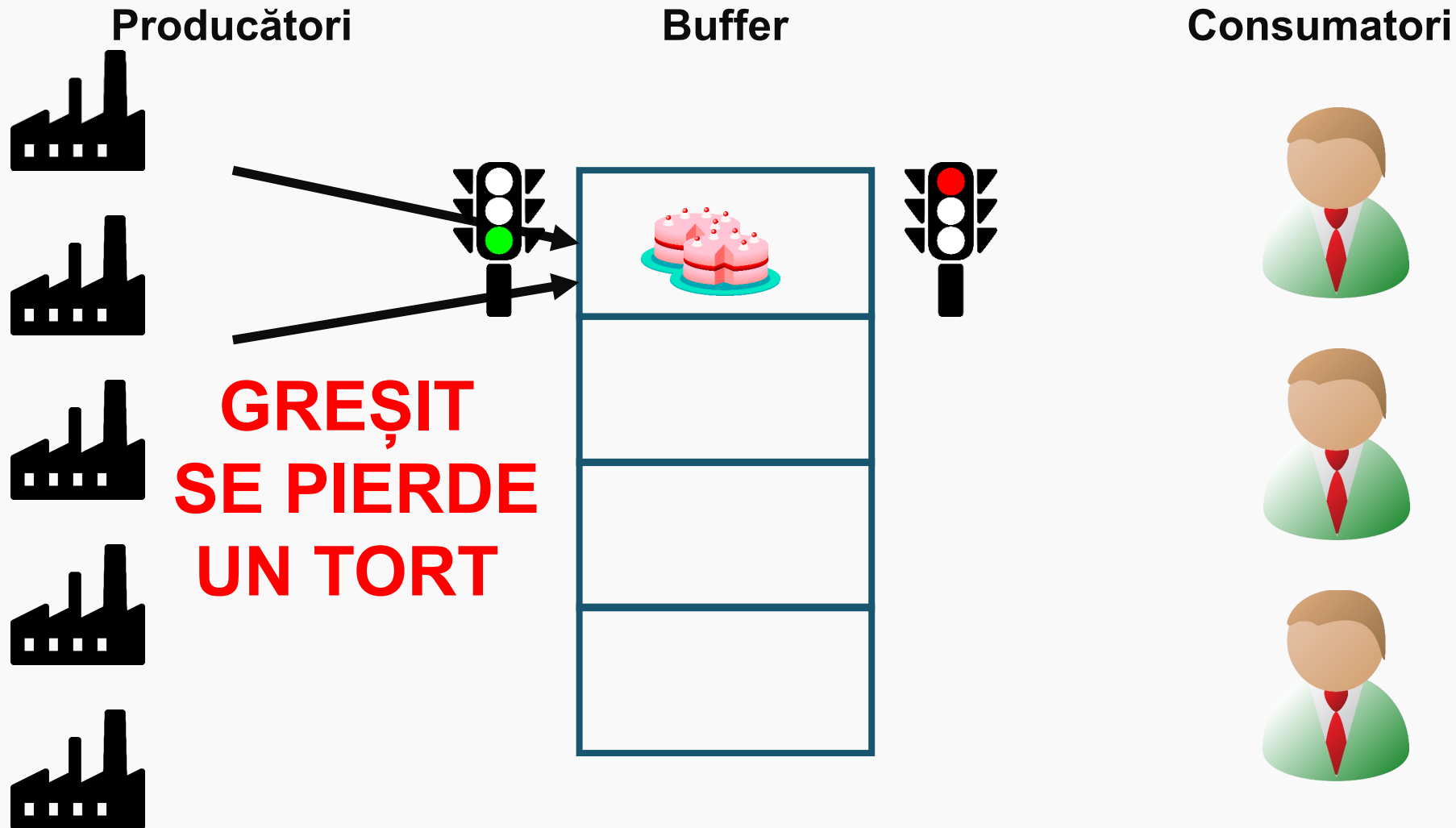


Problema producători-consumatori



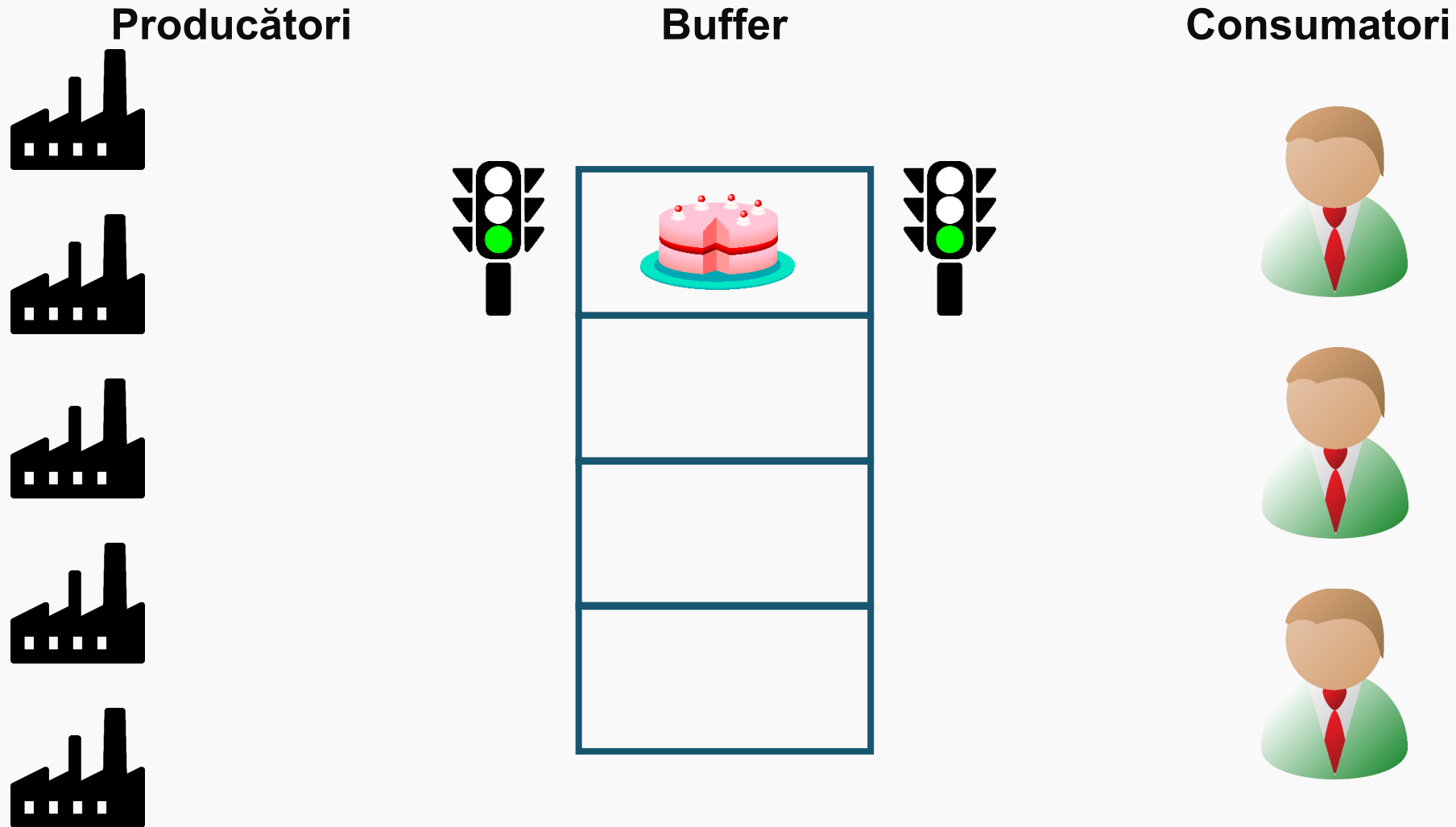


Problema producători-consumatori



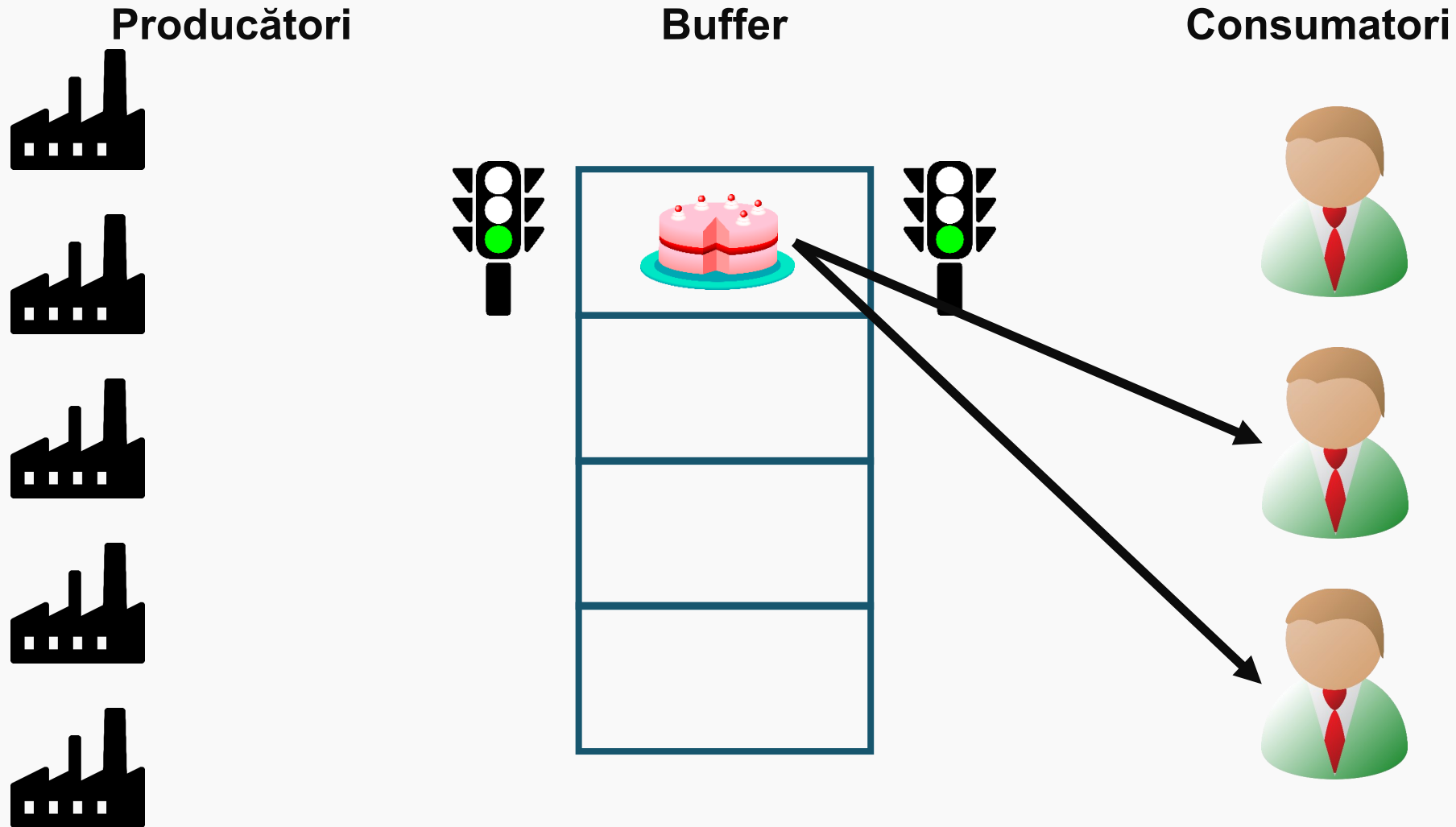


Problema producători-consumatori



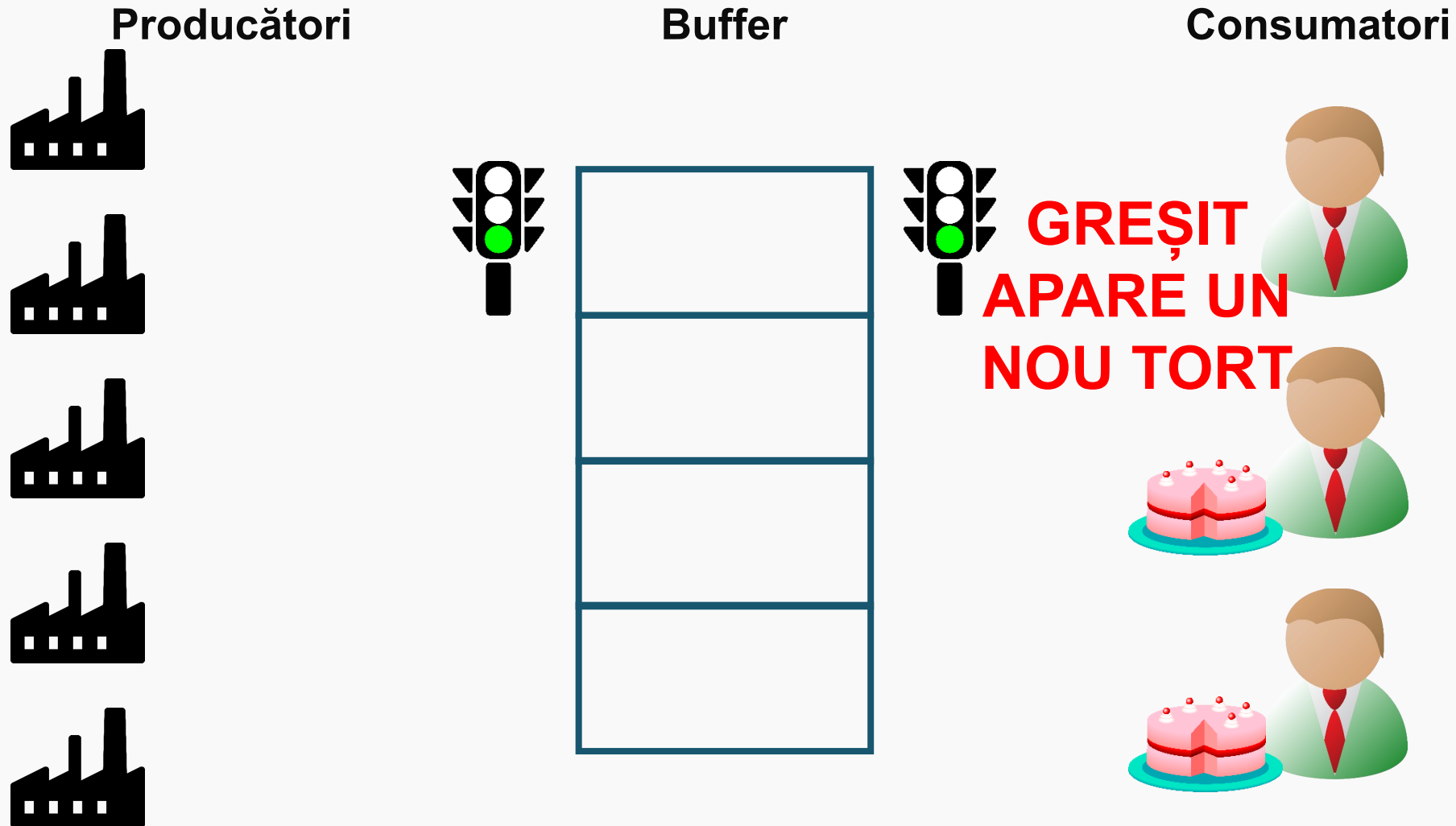


Problema producători-consumatori





Problema producători-consumatori





Problema producători-consumatori



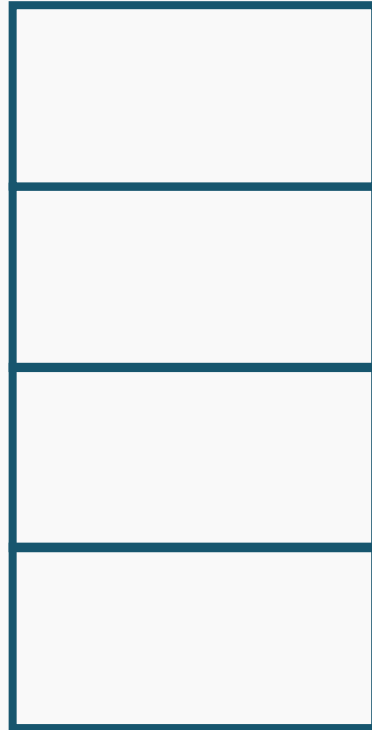
Empty.lock();

B.put(EL);

Full.unlock();



Buffer
Full = 0
Empty = 4



Consumatori



Full.lock();

EL = B.get();

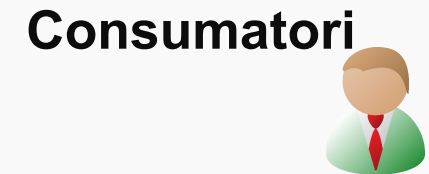
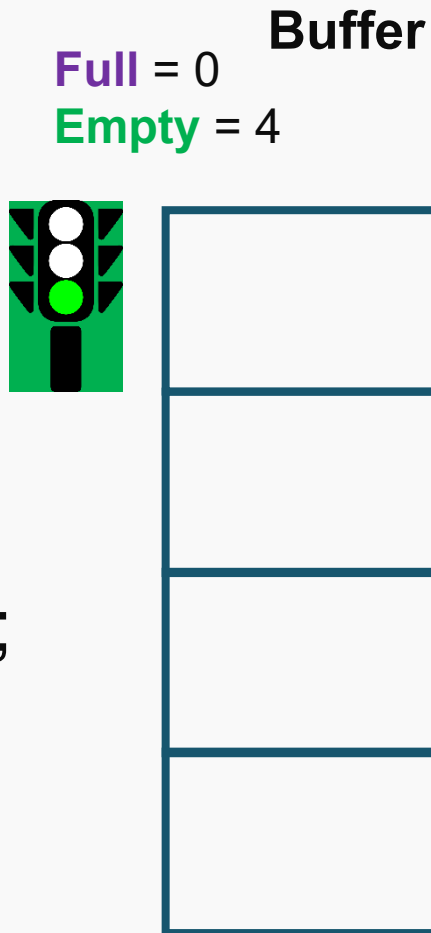
Empty.unlock();



Problema producători-consumatori



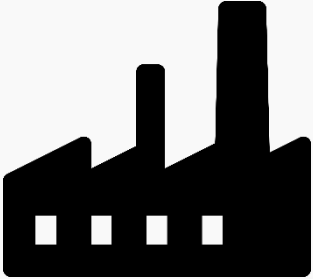
```
Empty.lock();  
Mutex.lock();  
B.put(EL);  
Mutex.unlock();  
Full.unlock();
```



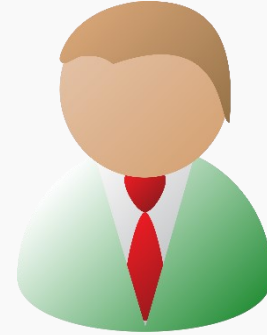
```
Full.lock();  
Mutex.lock();  
EL = B.get();  
Mutex.unlock();  
Empty.unlock();
```



Când folosim buffer?



- Un pachet se mută de la nivelul IP la TCP.
- Se strâng evenimente de la tastatură/mouse.
 - Randarea termină de construit un frame.
 - Se termină un pas i al unui pipeline.



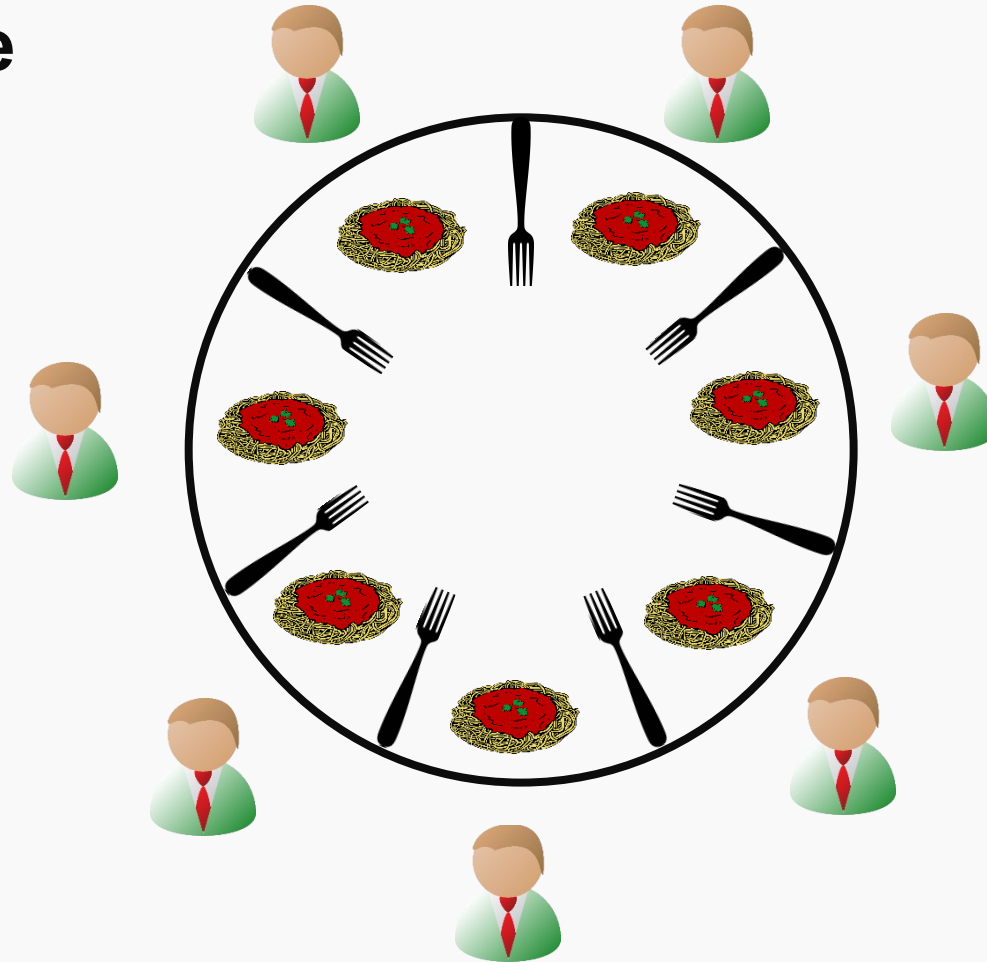
- Un pachet vine de la IP la TCP.
 - Se procesează evenimente de la tastatură/mouse.
- Se afișează un frame pe ecran.
- Se preiau datele pentru pasul $i+1$ al unui pipeline.





Problema filozofilor multi-deadlock

Furculițele
reprezintă
mutex-uri

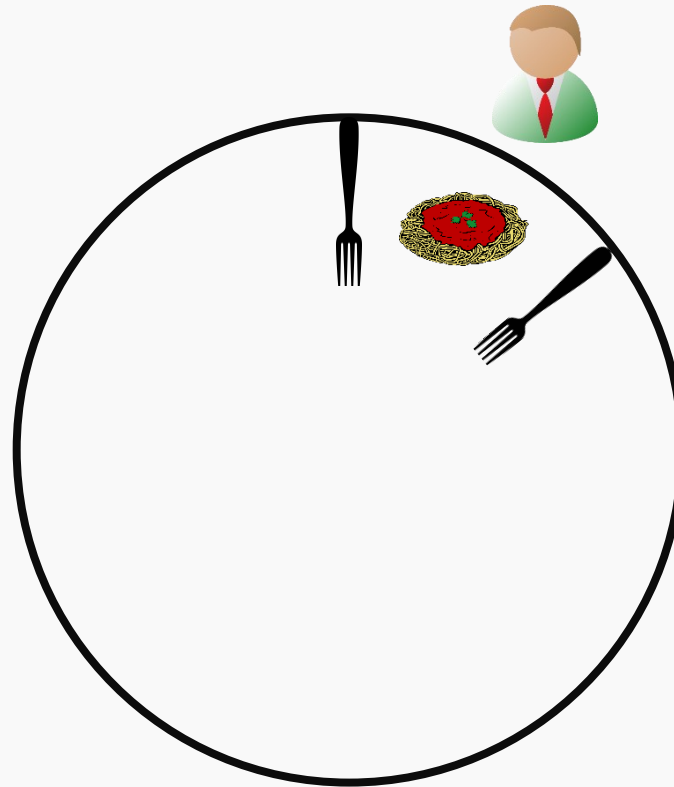


Filozofii
reprezintă
thread-uri



Un filozof

Furculițele
reprezintă
mutex-uri

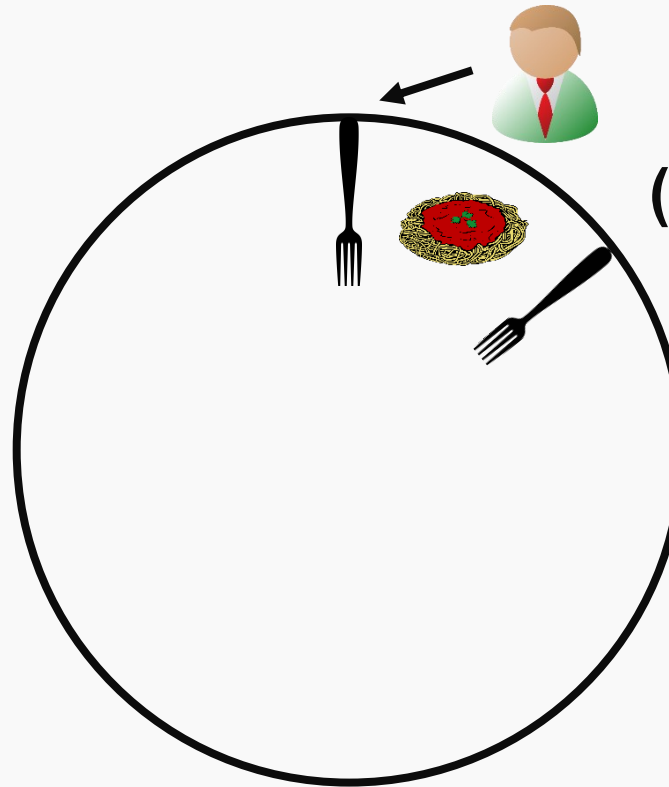


Un filozof vrea să mănânce.
Mâncatul necesită două furculițe (**mutex-uri**).
Mâncatul poate fi orice operație gen afișarea unui mesaj.



Un filozof

Furculițele
reprezintă
mutex-uri

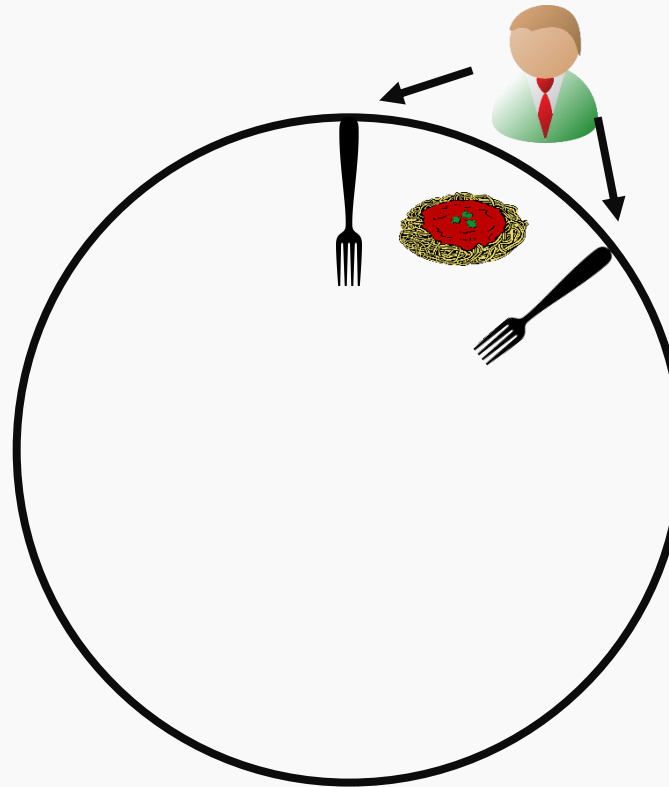


la furculița din
dreapta
(face **lock** pe **mutex**)



Un filozof

Furculițele
reprezintă
mutex-uri

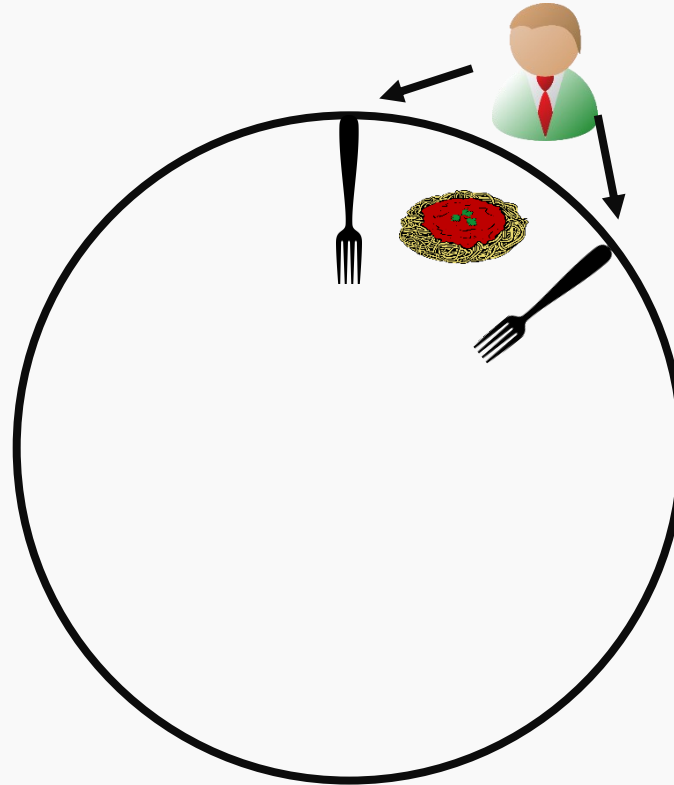


Cât timp ține
furculița din
dreapta (**mutex-ul**
este **locked**)
la furculița
stângă (face **lock** pe
mutex)



Un filozof

Furculițele
reprezintă
mutex-uri

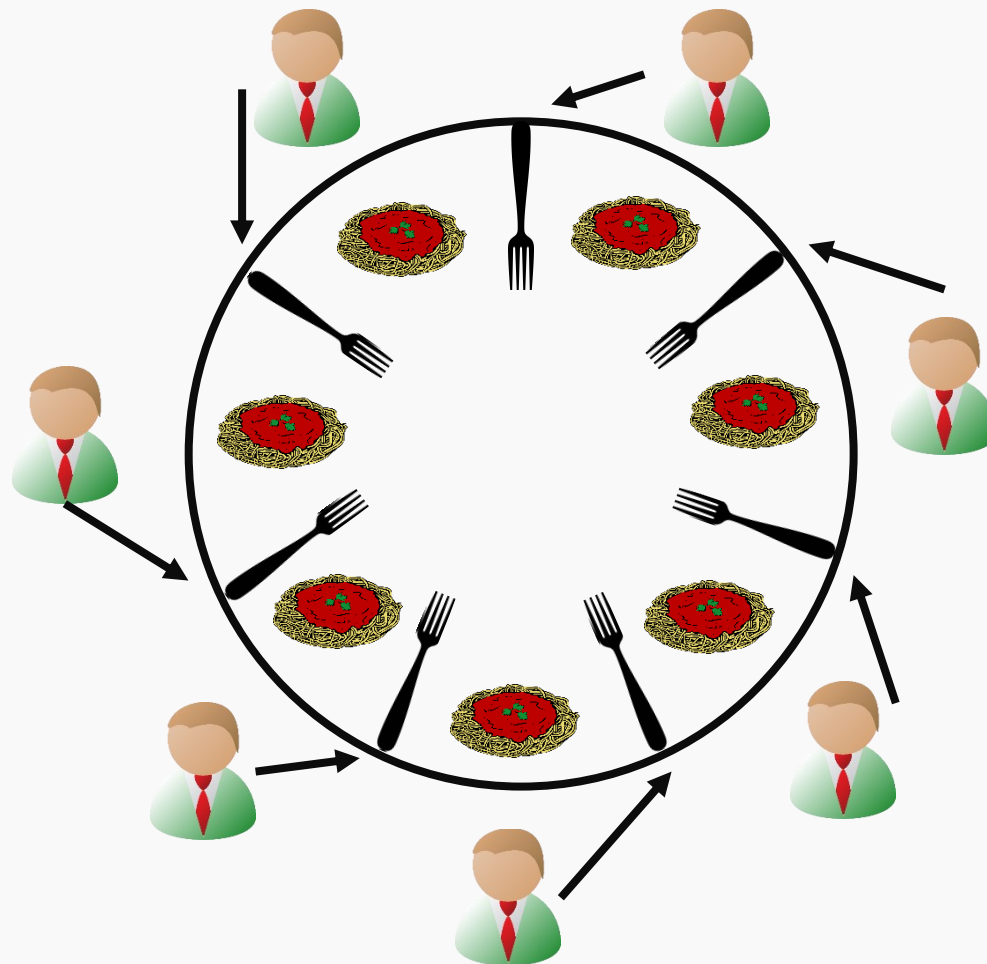


Acum poate
mânca.
Când termină va
pune joc
furculițele (face
unlock pe
mutex).
Alt filozof poate
lua acum oricare
din furculițe.



Problem filozofilor **Dead-Lock**

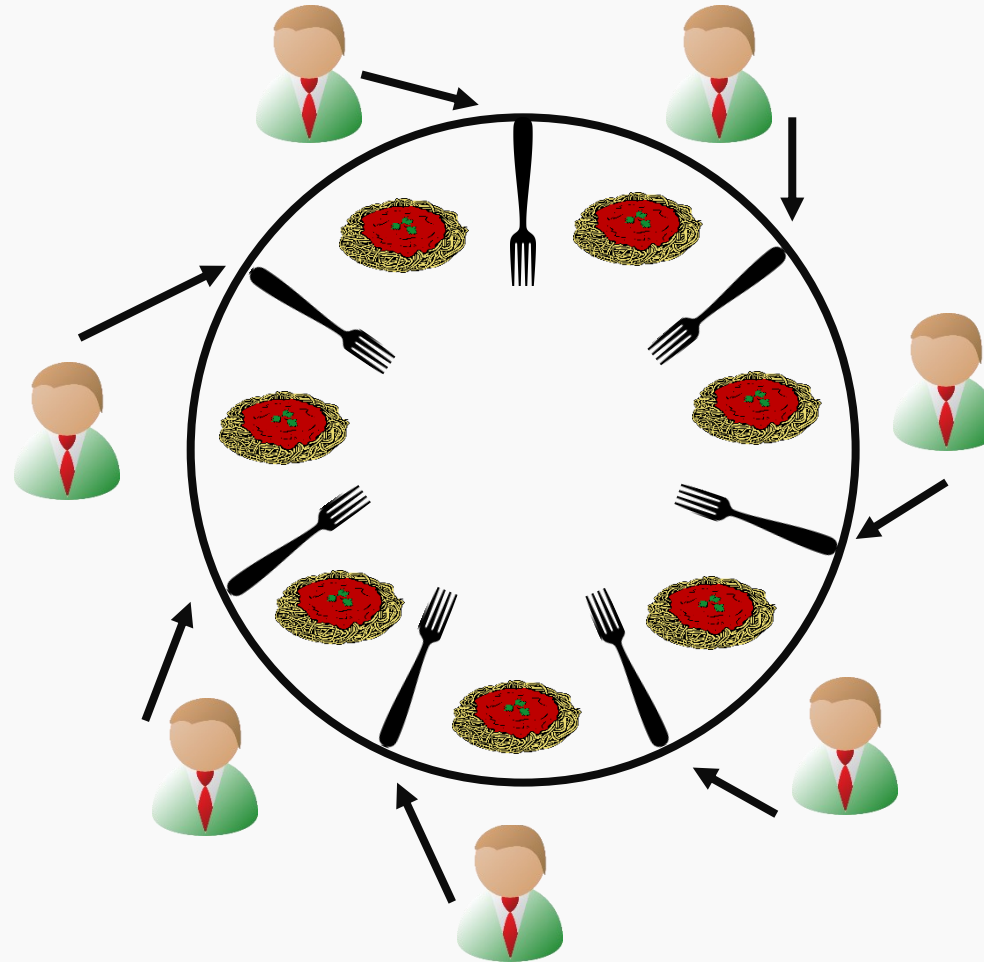
Dacă toți iau simultan furculița dreaptă (**lock** pe **mutex**) se pot bloca (**dead-lock**). Nici un filozof nu mai are furculiță în stânga care să nu fie deja luată.





Problem filozofilor **Dead-Lock**

Dacă toți iau simultan furculița stângă (**lock** pe **mutex**) se pot bloca (**dead-lock**). Nici un filozof nu mai are furculiță în dreapta care să nu fie deja luată.



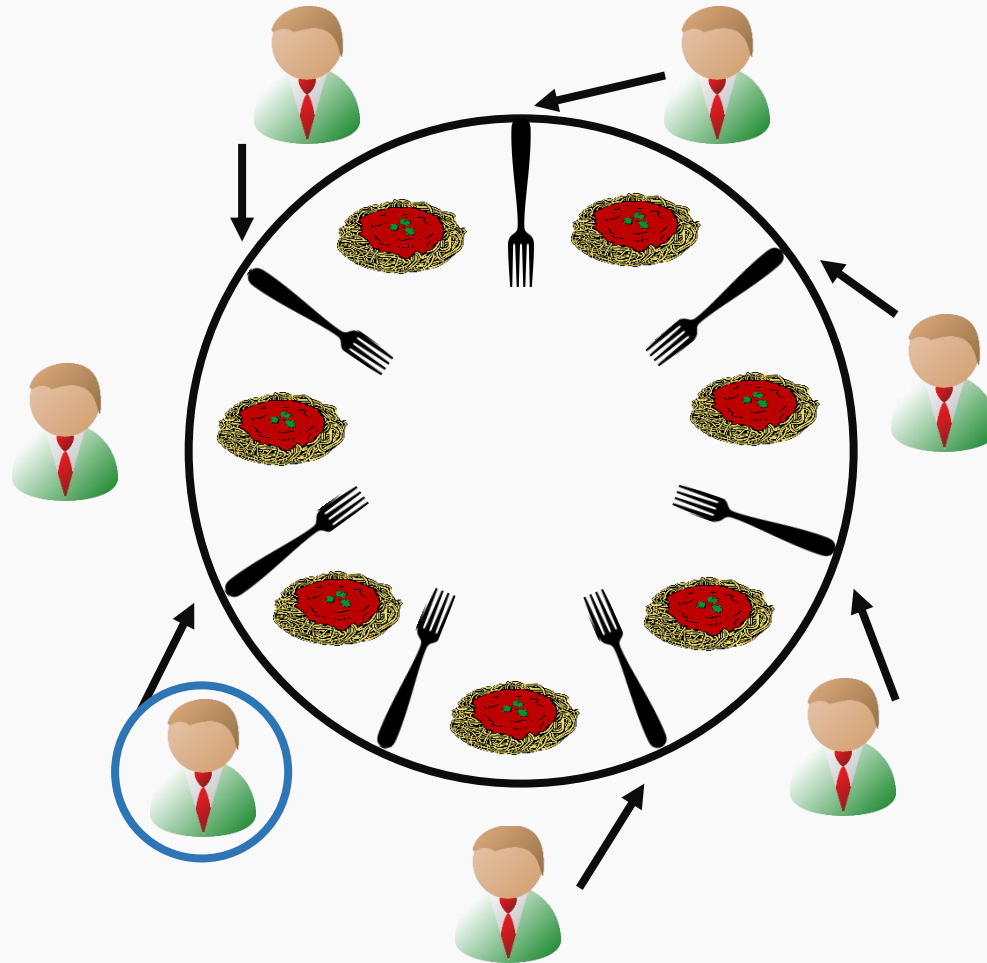


Soluție la problema filozofilor

Un filozof ridică întâi furculița stângă (**lock pe mutex**). Toți ceilalți ridică întâi furculița dreaptă (**lock pe mutex**).



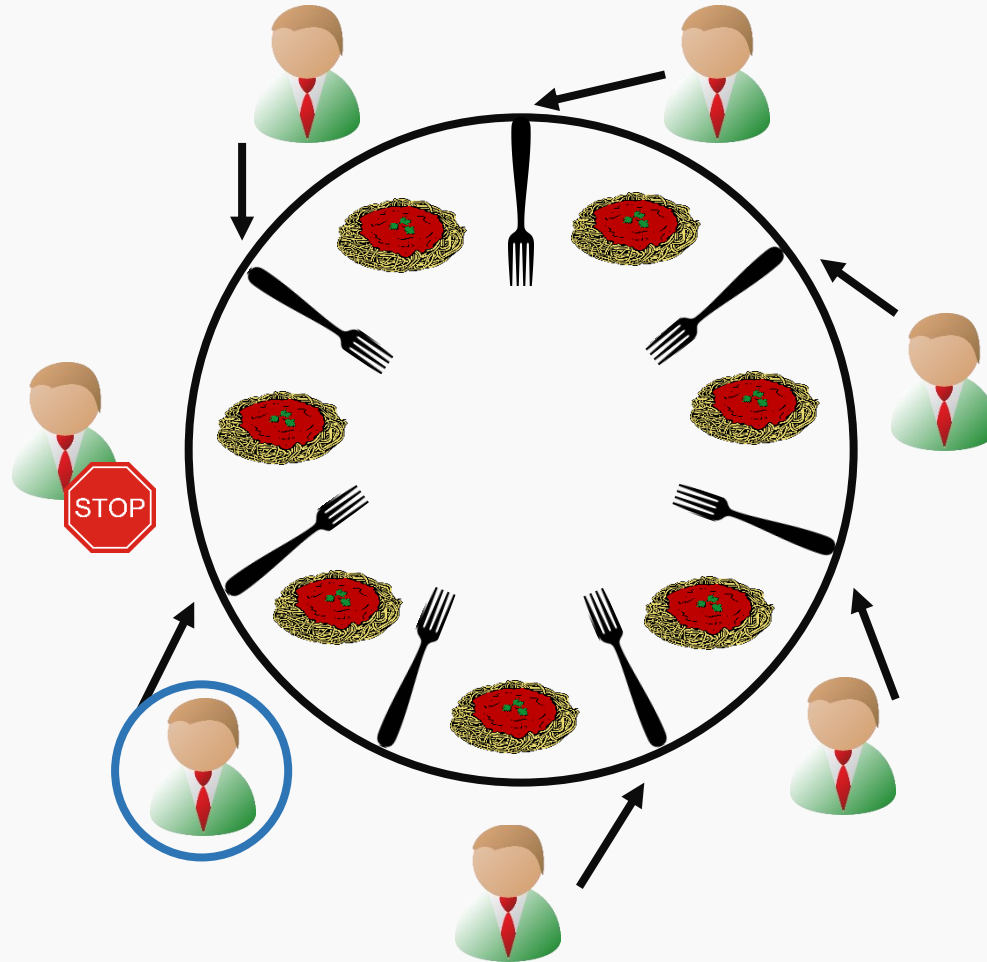
Soluție la problema filozofilor





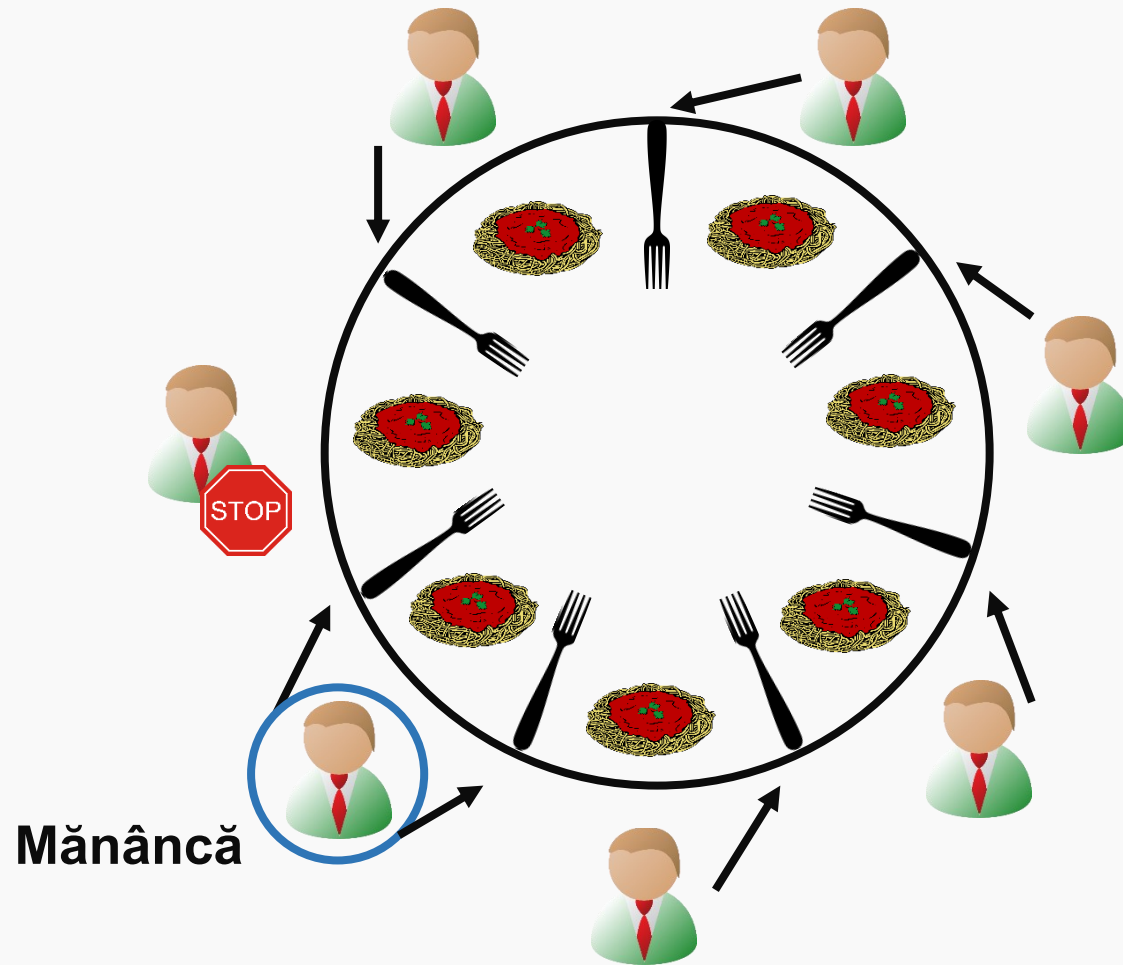
Soluție la problema filozofilor

Nu poate
ridica nici o
furculiță
(așteaptă la
mutex)



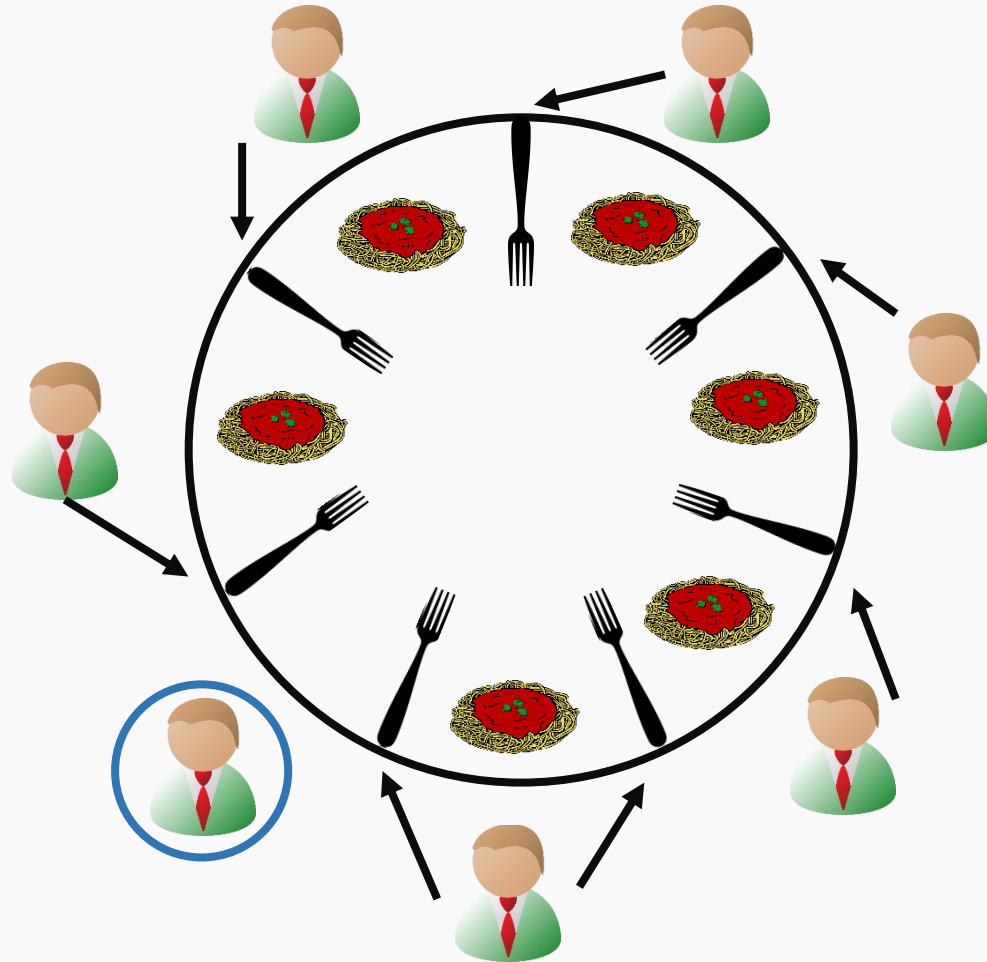


Soluție la problema filozofilor





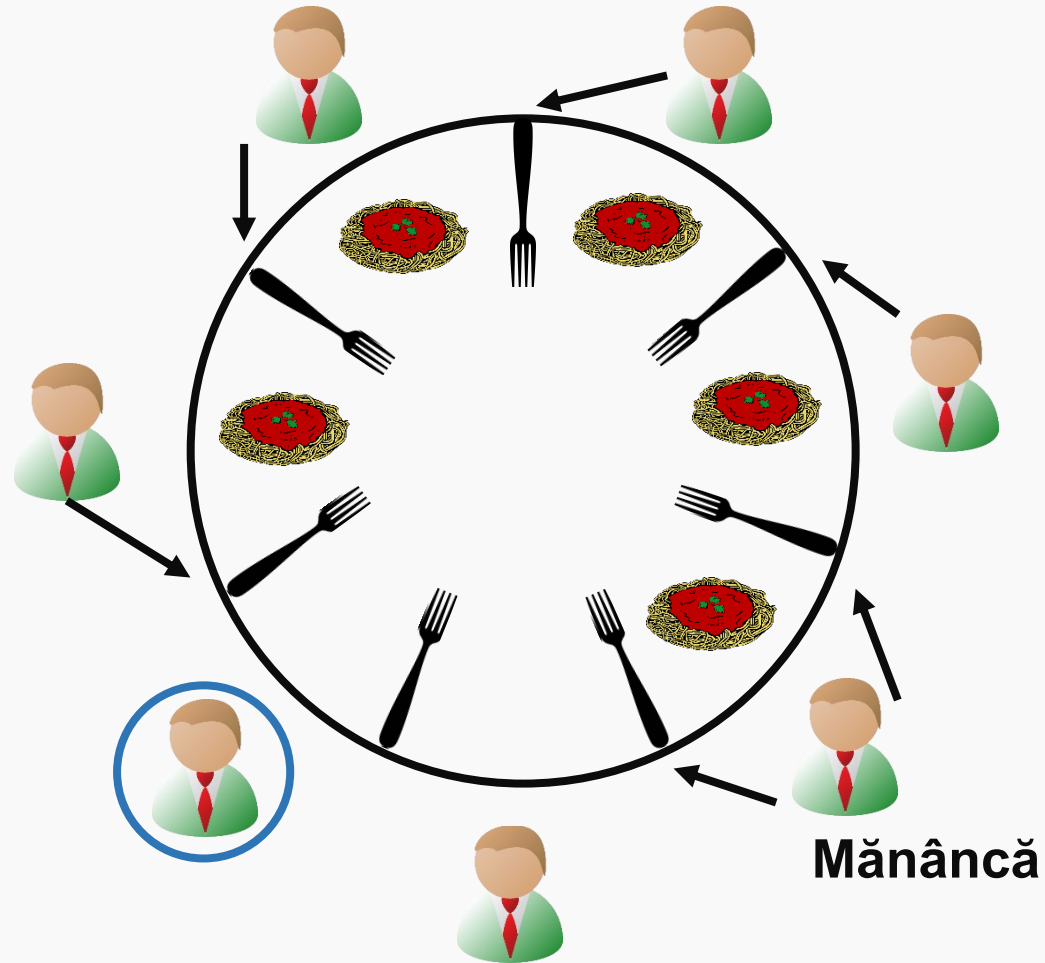
Soluție la problema filozofilor



Mănâncă

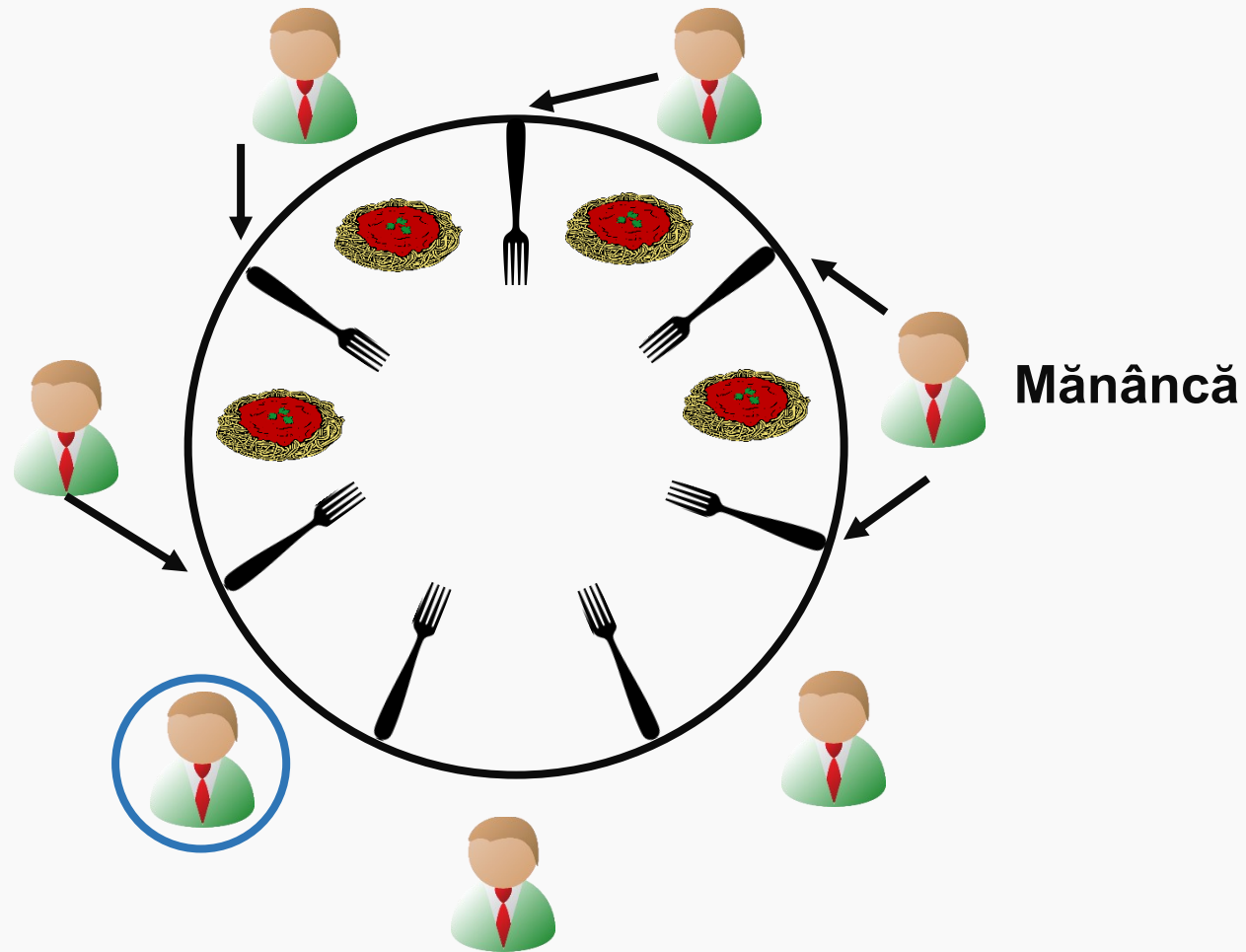


Soluție la problema filozofilor



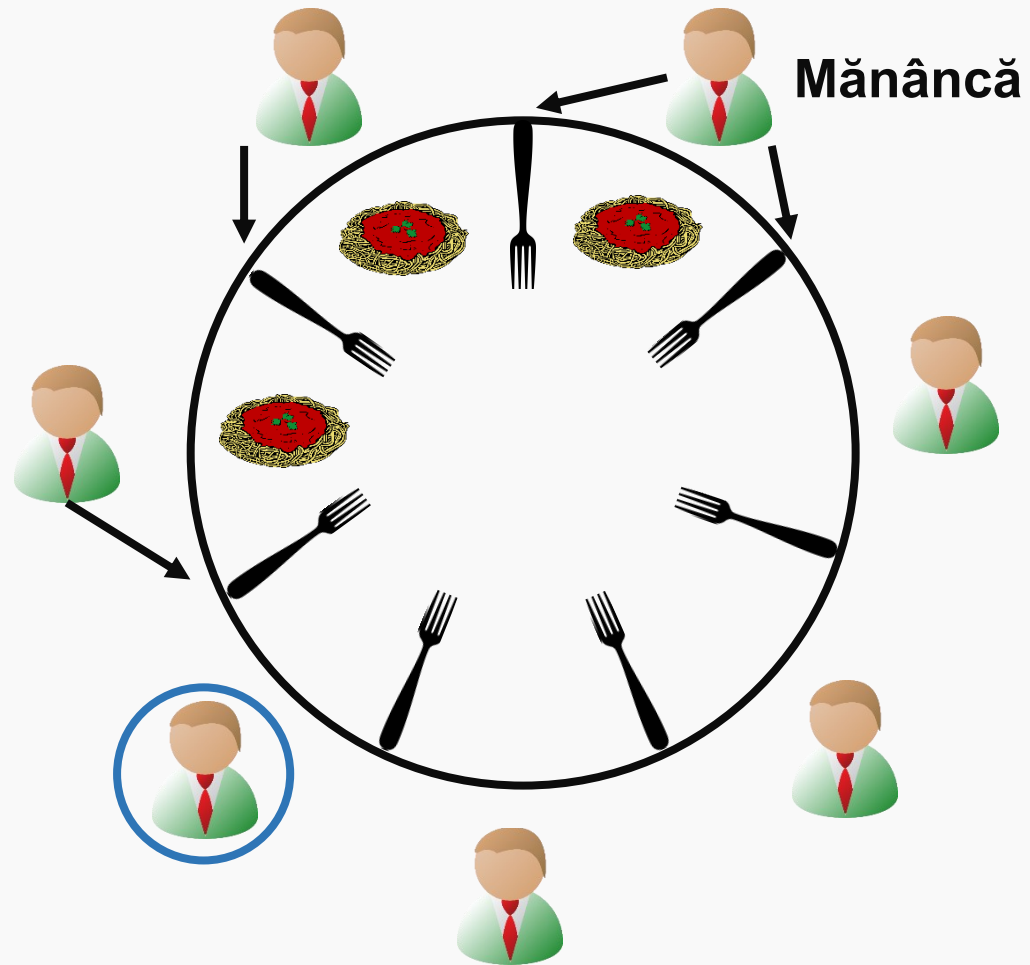


Soluție la problema filozofilor





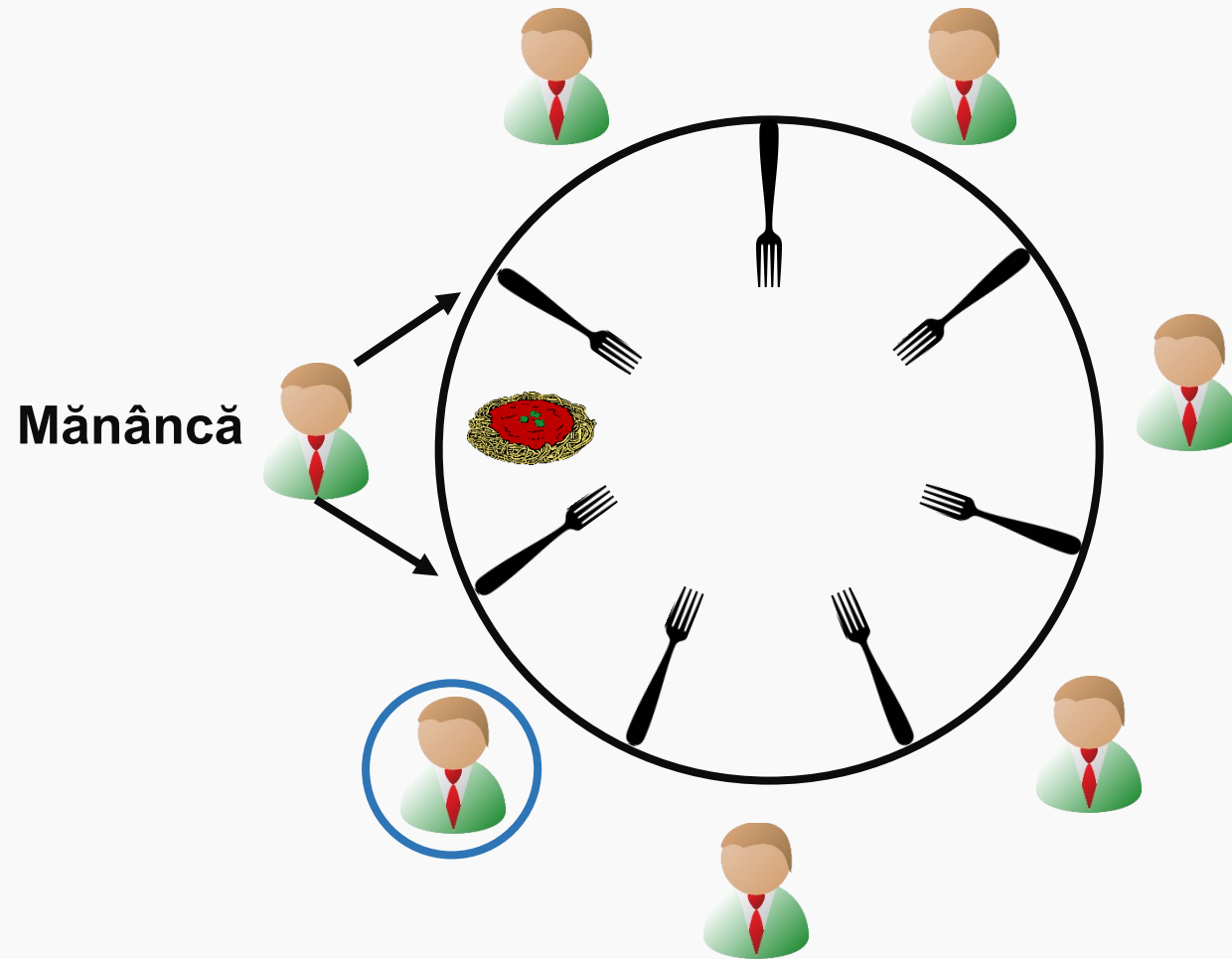
Soluție la problema filozofilor







Soluție la problema filozofilor





Problema filozofilor – exemplu **dead-lock**

**Avem un server ce are o platformă de tip rețea socială.
Fiecare din thread-urile noastre se ocupă de câte un
utilizator.**

**Este posibil ca un grup de prieteni să formeze un cerc de
prietenii (la fel ca filozofii care stau la masă).**

**Utilizatorii pot să ceară simultan ca o poză primită de la un
prieten să fie trimisă la altul.**

**Pentru a putea face această acțiune un thread trebuie să facă
lock pe mutex-urile celor două buffere.**

**Dacă se formează un cerc în care fiecare thread face lock pe
câte un buffer și așteaptă pentru un al doilea se va intra în
dead-lock.**



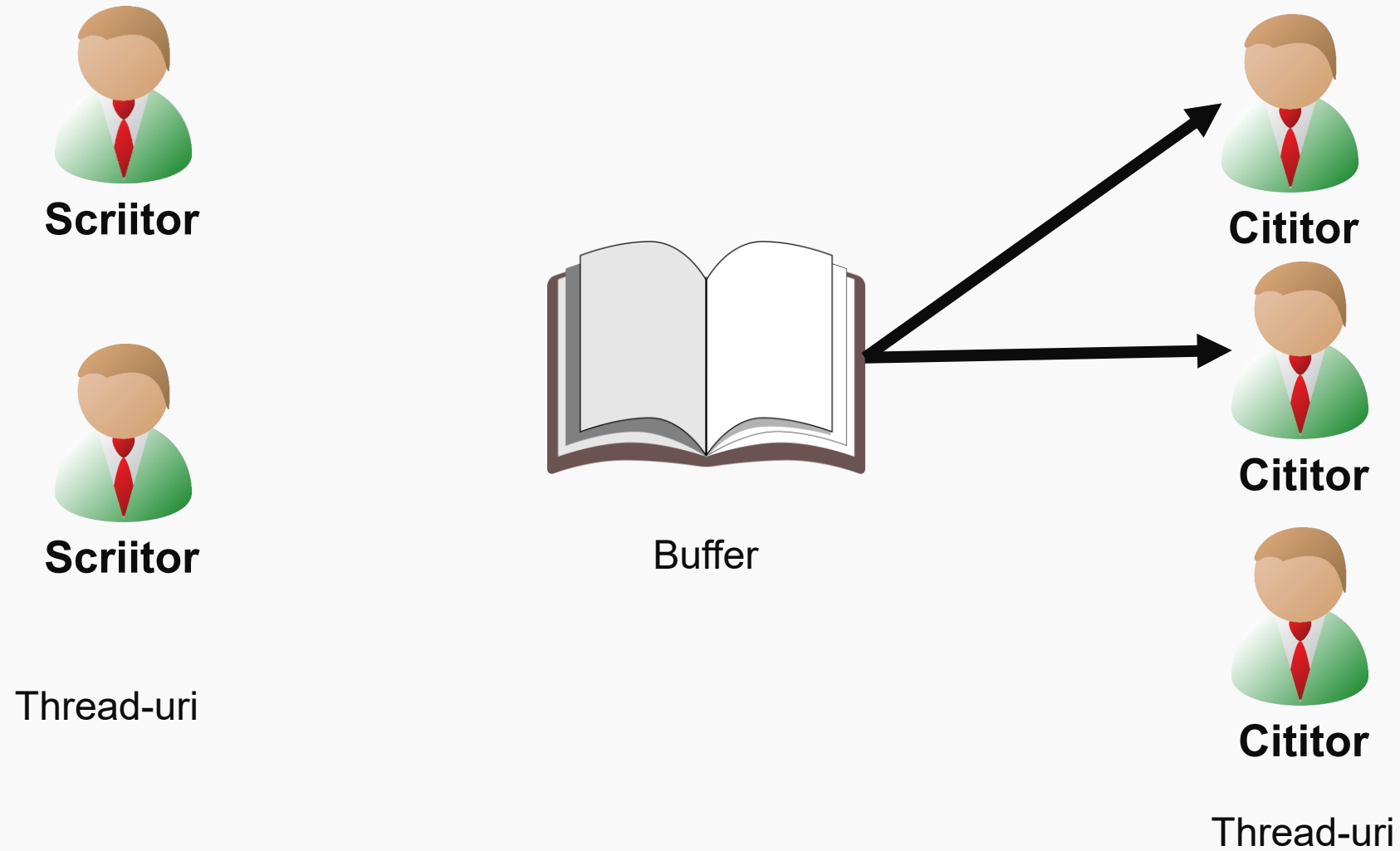


Problem Cititori - Scriitori

- Mai mulți cititori pot citi în același timp (**R-R**)
- Mai mulți scriitori **NU** pot scrie în același timp (**W-W**)
- Un cititor **NU** poate citi în timp ce se scrie (**R-W**)

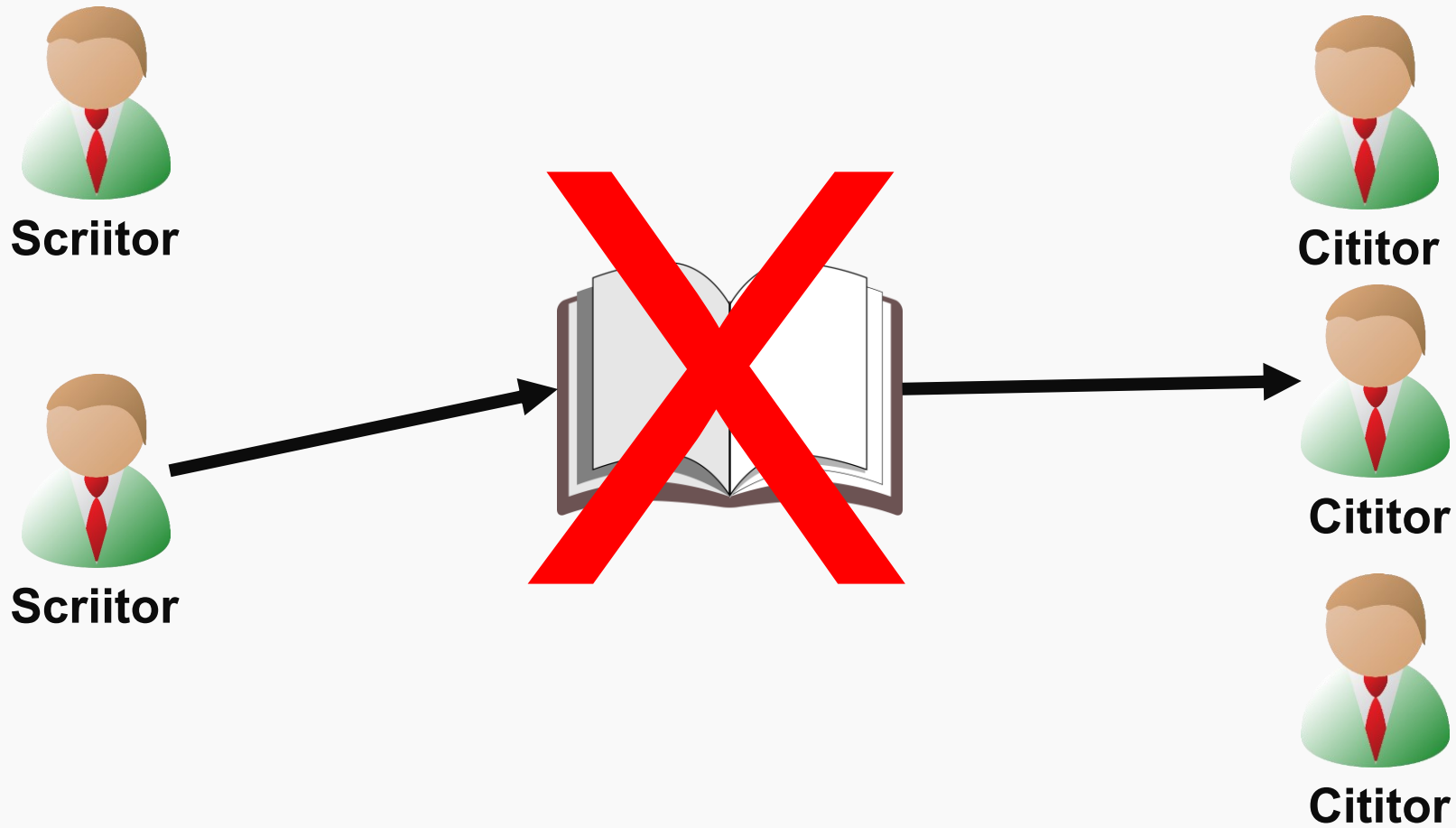


Cititori – Scriitori (R-R)



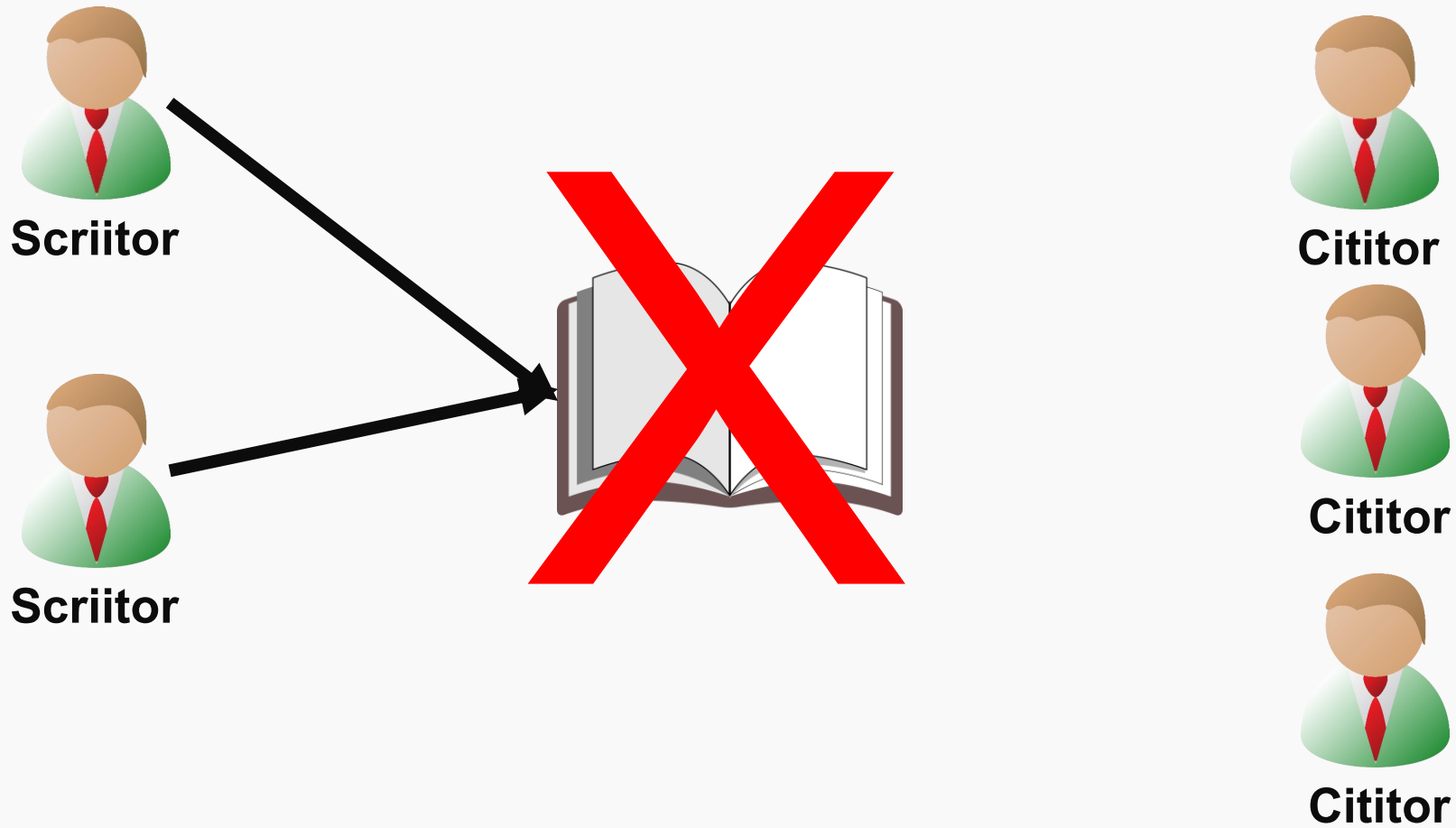


Cititori – Scriitori (R-W)





Cititori – Scriitori (W-W)





Cititori - Scriitori

Scriitor

```
B.put(EL);
```

Cititor

```
EL = B.get();
```



Cititori - Scriitori

Scriitor

```
mutex.lock();  
B.put(EL);  
mutex.unlock();
```

Cititor

```
mutex.lock();  
EL = B.get();  
mutex.unlock();
```




Cititori - Scriitori

Scriitor

```
mutex.lock();  
B.put(EL);  
mutex.unlock();
```

Cititor

```
mutex.lock();  
EL = B.get();  
mutex.unlock();
```

Greșit:

Rezolvă Write-Write

Rezolvă Read-Write

Nu permite mai multor cititori să citească simultan



Cititori - Scriitori

Scriitor

B.put(EL);

Cititor

EL = B.get();



Cititori - Scriitori

Scriitor

```
Wmutex.lock();  
B.put(EL);  
Wmutex.unlock();
```

Cititor

```
Wmutex.lock();  
  
EL = B.get();  
  
Wmutex.unlock();
```



Cititori - Scriitori

Scriitor

```
Wmutex.lock();  
B.put(EL);  
Wmutex.unlock();
```

Cititor

```
Rmutex.lock();  
countReaders++;  
if(countReaders==1)  
    Wmutex.lock();  
Rmutex.unlock();  
EL = B.get();  
Rmutex.lock();  
countReaders--;  
if(countReaders==0)  
    Wmutex.unlock();  
Rmutex.unlock();
```





Problema bărbierului

- Avem un bărbier și N scaune de așteptare
- Când nu sunt clienți, bărbierul doarme
- Un client nou venit ori trezește bărbierul și este bărbierit, ori dacă mai sunt și alți clienți, așteaptă pe unul din scaune
- Dacă toate scaunele sunt ocupate un potențial nou client pleacă



Problema bărbierului

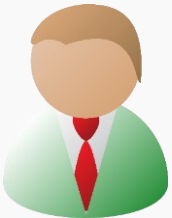
- Fiecare thread client vrea să apeleze funcția `getHairCut()`
- Dacă un client ajunge și toate scaunele sunt ocupate atunci pleacă
- Bărbierul trebuie să apeleze `cutHair()`.
- Când bărbierul apelează `cutHair()` trebuie să fie exact un client care apelează `getHairCut()`.



Problema bărbierului



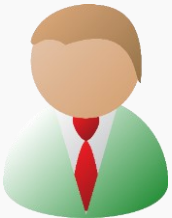
Client



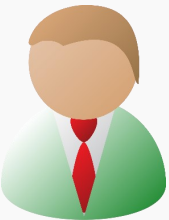
Bărbier - sleep



Problema bărbierului



Bărbier – cutHair()



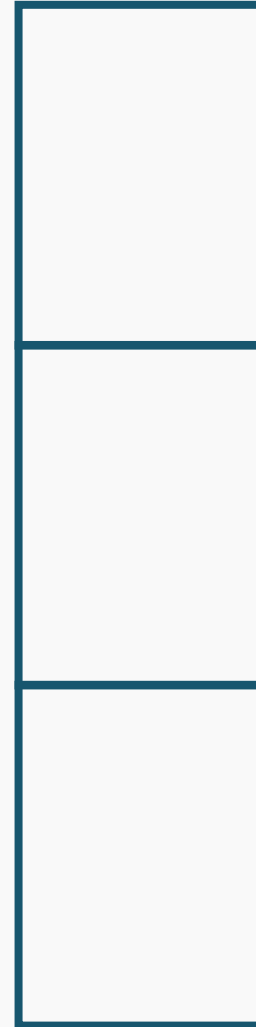
Client – getHairCut()



Problema bărbierului



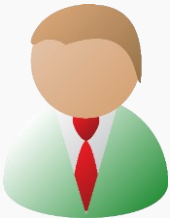
Bărbier



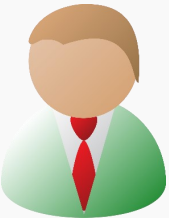
Client



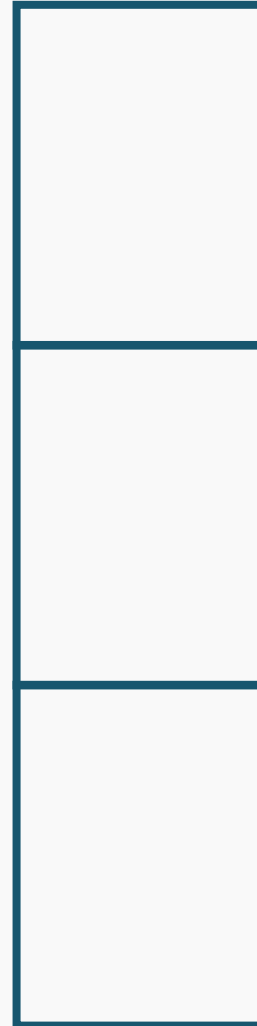
Problema bărbierului



Bărbier – cutHair()

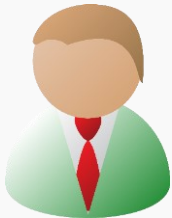


Client – getHairCut()

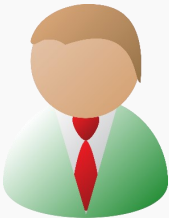




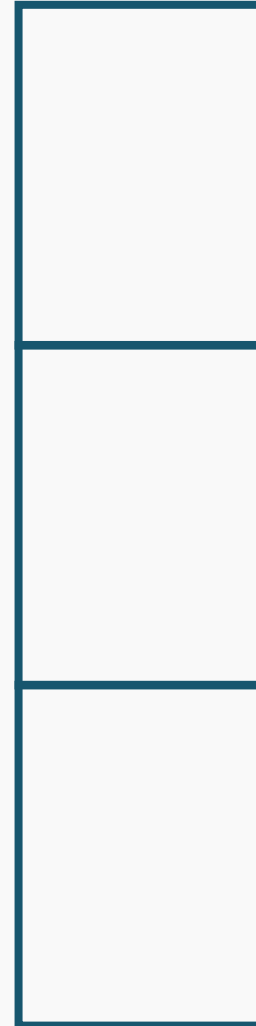
Problema bărbierului



Bărbier – cutHair()



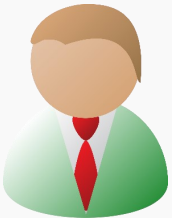
Client – getHairCut()



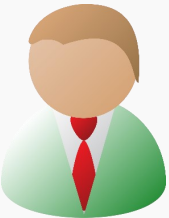
Client



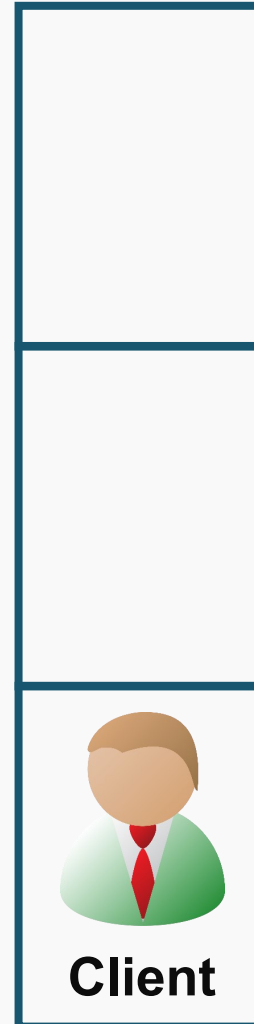
Problema bărbierului



Bărbier – cutHair()

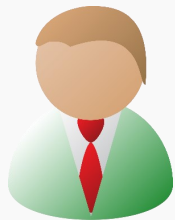


Client – getHairCut()

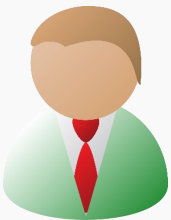




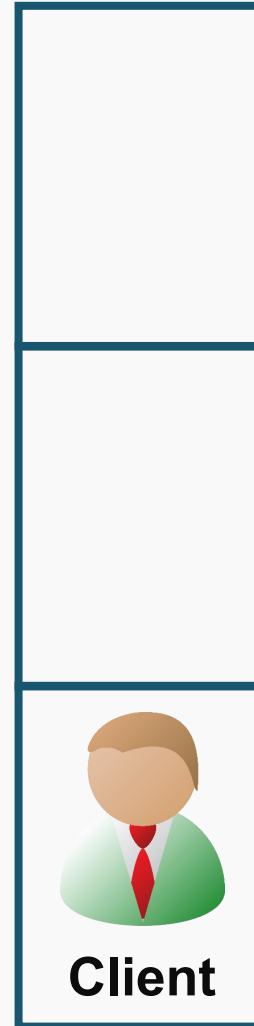
Problema bărbierului



Bărbier – cutHair()



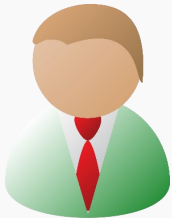
Client – getHairCut()



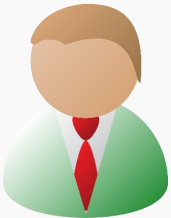
Client



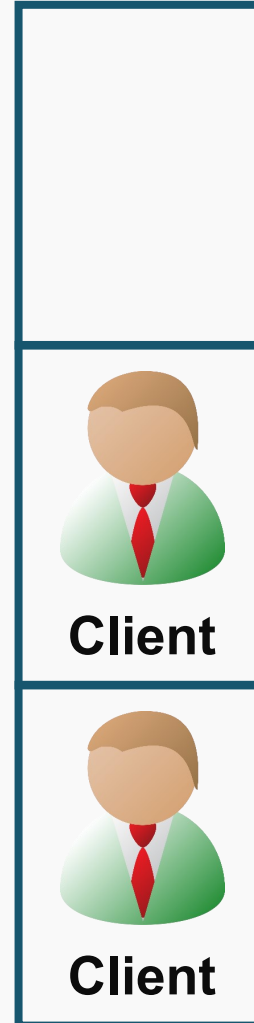
Problema bărbierului



Bărbier – cutHair()

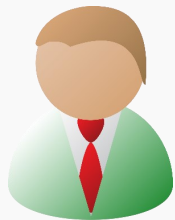


Client – getHairCut()

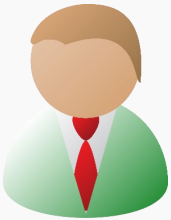




Problema bărbierului



Bărbier – cutHair()



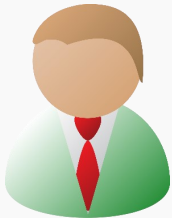
Client – getHairCut()



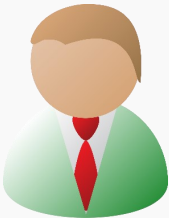
Client



Problema bărbierului



Bărbier – cutHair()



Client – getHairCut()



Client



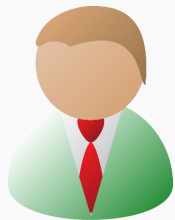
Client



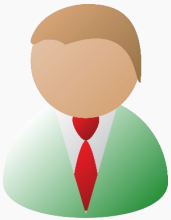
Client



Problema bărbierului



Bărbier – cutHair()



Client – getHairCut()



Client



Client



Client



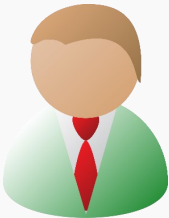
Client - leave



Problema bărbierului



Bărbier – cutHair()



Client – getHairCut()



Client



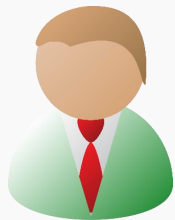
Client



Client



Problema bărbierului



Bărbier - sleep



Client



Client



Client



Problema bărbierului



Bărbier – cutHair()



Client – getHairCut()



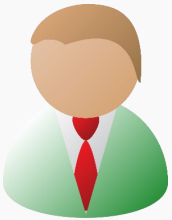
Client



Client



Problema bărbierului



Bărbier - sleep



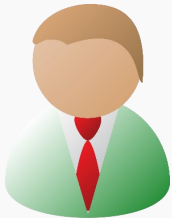
Client



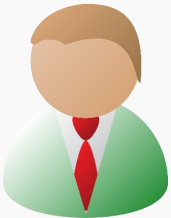
Client



Problema bărbierului



Bărbier – cutHair()



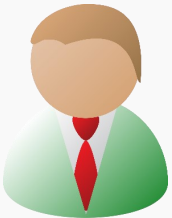
Client – getHairCut()



Client



Problema bărbierului



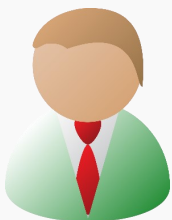
Bărbier - sleep



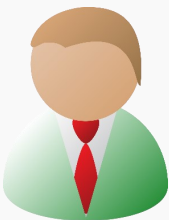
Client



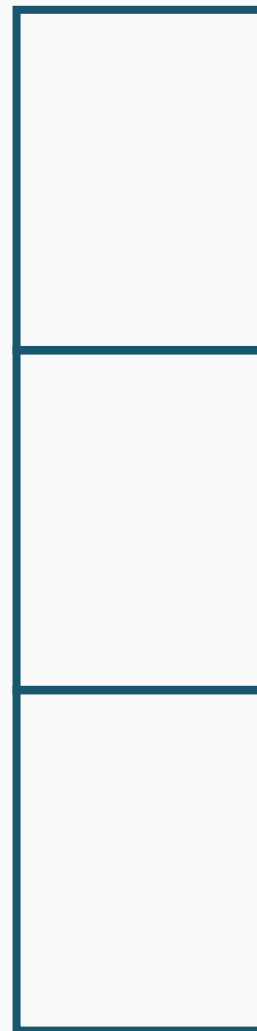
Problema bărbierului



Bărbier – cutHair()



Client – getHairCut()





Problema bărbierului

Bărbier

emptyChairs = N

Clients = 0

BarberReady = 0

Chairs = 1

Client

```
while(true) {  
    Clients.lock();  
    Chairs.lock();  
    emptyChairs++;  
    BarberReady.unlock();  
    Chairs.unlock();  
    cutHair();  
}
```

```
Chairs.lock();  
if(emptyChairs>0) {  
    emptyChairs--;  
    Clients.unlock();  
    Chairs.unlock();  
    BarberReady.lock();  
    getHairCut();  
} else {  
    Chairs.unlock();  
}
```





Problema fumătorilor

- Un agent și trei fumători
- Fumătorii:
 - **Așteaptă ingrediente (tutun, hărtie, chibrit)**
 - **Confecționează țigară**
 - **Fumează**
- Agentul deține toate 3 ingredientele
- Un fumător are tutun, un altul hărtie, 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**



Problema fumătorilor

```
sem tobacco = 0;
sem paper = 0;
sem match = 0;
Sem agent = 1;
process Agent{
    while (true){
        if (draw1){ P(agent); V(tobacco); V(paper); }
        else if (draw2){ P(agent); V(paper); V(match); }
        else if (draw3){ P(agent); V(tobacco); V(match); }
    }
}
process Smoker1{
    P(tobacco); P(paper); V(agent);
}
process Smoker2{
    P(paper); P(match); V(agent);
}
process Smoker3{
    P(tobacco); P(match); V(agent);
}
```

Funcționează ???



Problema fumătorilor - deadlock

DEADLOCK

```
sem tobacco = 0;
sem paper = 0;
sem match = 0;
Sem agent = 1;
process Agent{
    while (true) {
        if (draw1) { P(agent); V(tobacco); V(paper); }
        else if (draw2) { P(agent); V(paper); V(match); }
        else if (draw3) { P(agent); V(tobacco); V(match); }
    }
}
process Smoker1{
    P(tobacco); P(paper); V(agent);
}
process Smoker2{
    P(paper); P(match); V(agent);
}
process Smoker3{
    P(tobacco); P(match); V(agent);
}
```



Problema fumătorilor – rezolvare deadlock

**Din enunțul problemei nu avem voie să modificăm agentul.
Dar putem modifica pe cei 3 fumători.**



Problema fumătorilor – rezolvare deadlock

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

```
bool isTobacco = false;  
bool isPaper = false;  
bool isMatch = false;
```

```
sem tobaccoSem = 0;  
sem paperSem = 0;  
sem matchSem = 0;
```

```
process Agent{  
    while (true) {  
        if (draw1) { P(agent); V(tobacco); V(paper); }  
        else if (draw2) { P(agent); V(paper); V(match); }  
        else if (draw3) { P(agent); V(tobacco); V(match); }  
    }  
}
```




Problema fumătorilor – rezolvare deadlock

```
process PusherA{
    P(tobacco);
    P(e);
    if (isPaper) { isPaper = false; V(matchSem); }
    else if (isMatch) { isMatch = false; V(paperSem); }
    else if (isPaper == isMatch == false) isTobacco = true;
    V(e);
}

proces PusherB{
    P(match);
    P(e);
    if (isPaper) { isPaper = false; V(tobaccoSem); }
    else if (isTobacco) { isTobacco = false; V(paperSem); }
    else if (isPaper == isTobacco == false) isMatch = true;
    V(e);
}

process PusherC{
    P(paper);
    P(e);
    if (isTobacco) { isTobacco = false; V(matchSem); }
    else if (isMatch) { isMatch = false; V(tobaccoSem); }
    else if (isPaper == isMatch == false) isPaper = true;
    V(e);
}
```



Problema fumătorilor – rezolvare deadlock

```
process SmokerWithTobacco{
    P(tobaccoSem) ;
    # makeCigarette
    V(agent) ;
    # smoke
}
process SmokerWithPaper{
    P(paperSem) ;
    # makeCigarette
    V(agent) ;
    # smoke
}
process SmokerWithMatch{
    P(matchSem) ;
    # makeCigarette
    V(agent) ;
    # smoke
}
```