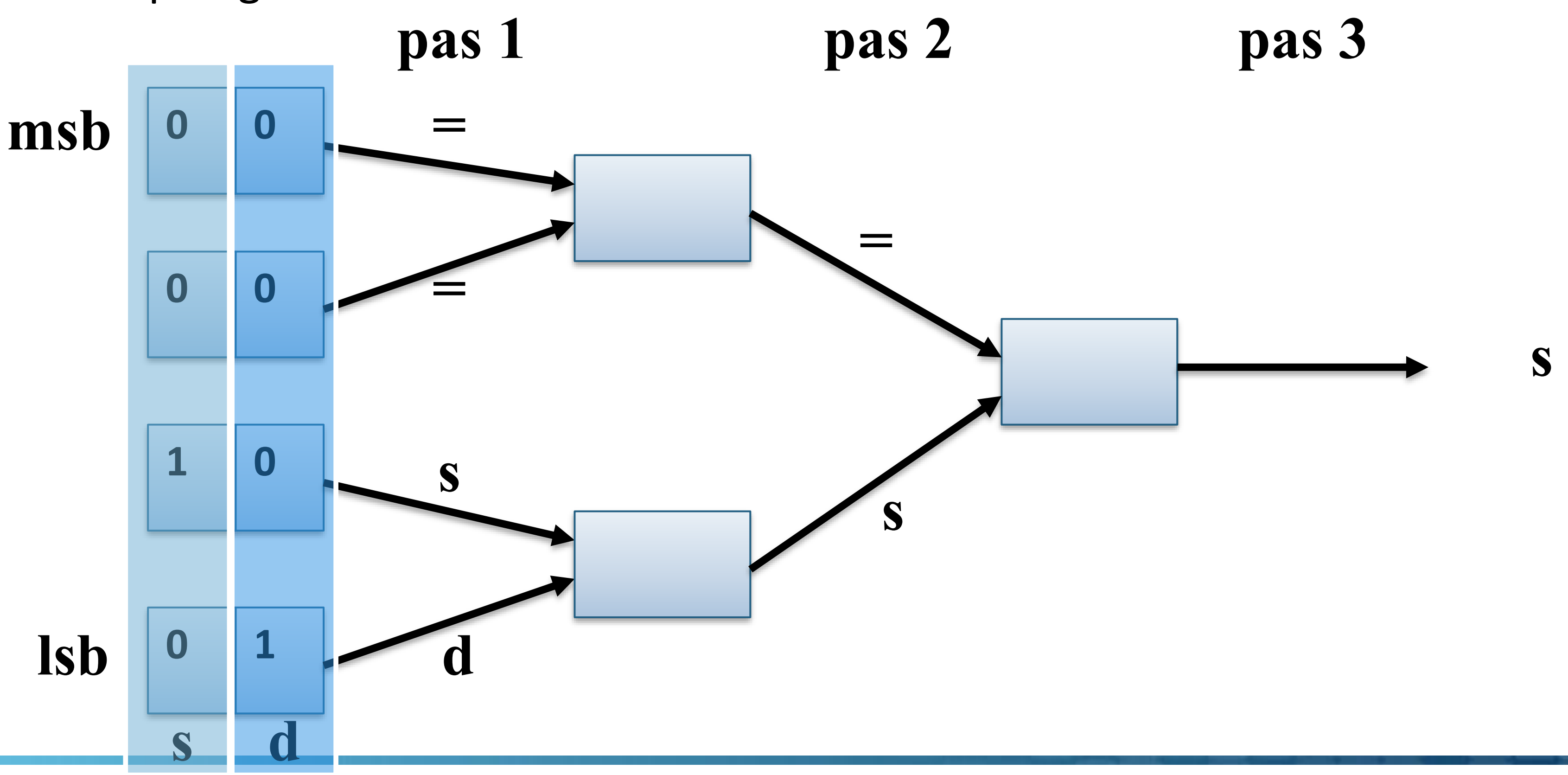
Arhitecturi Paralele
Access concurent la cozi

Prof. Florin Pop
As. Drd. Ing. Cristian Chilipirea
cristian.chilipirea@cs.pub.ro

Elemente preluate din cursul Prof. Ciprian Dobre

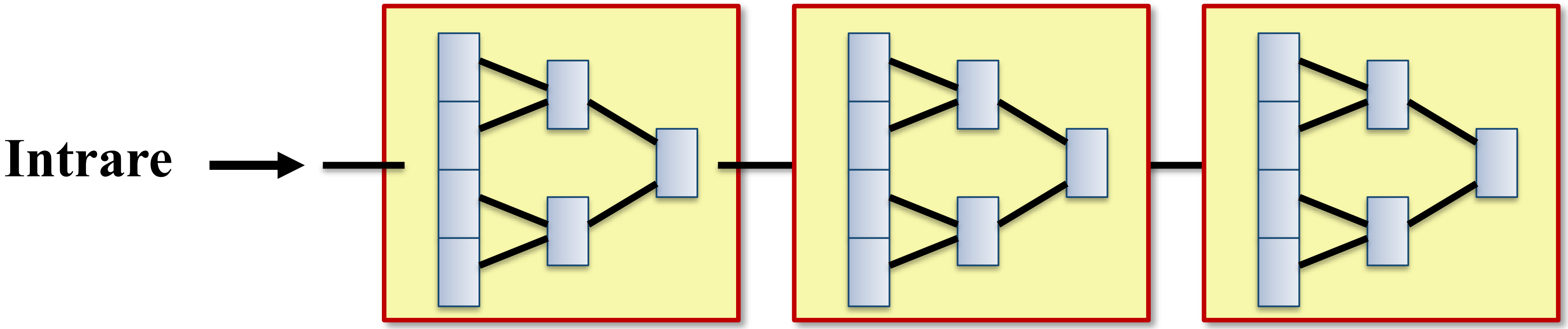
Calculul detaliat al complexităţii

- Trecere la modelul "bit"
  - Fiecare procesor operează pe 1 bit
  - Operația principală – compararea a două numere "s" și "d"
  - Topologie arborescentă



Calculul detaliat al complexităţii

- Algoritmul de sortare devine:



- Rețea de **(2k-1)*N** procesoare
- **Test**: Câți pași sunt necesari în această abordare?
- **N*logK** pași pentru faza 1

Producer - Consumer

EWD209 - 0

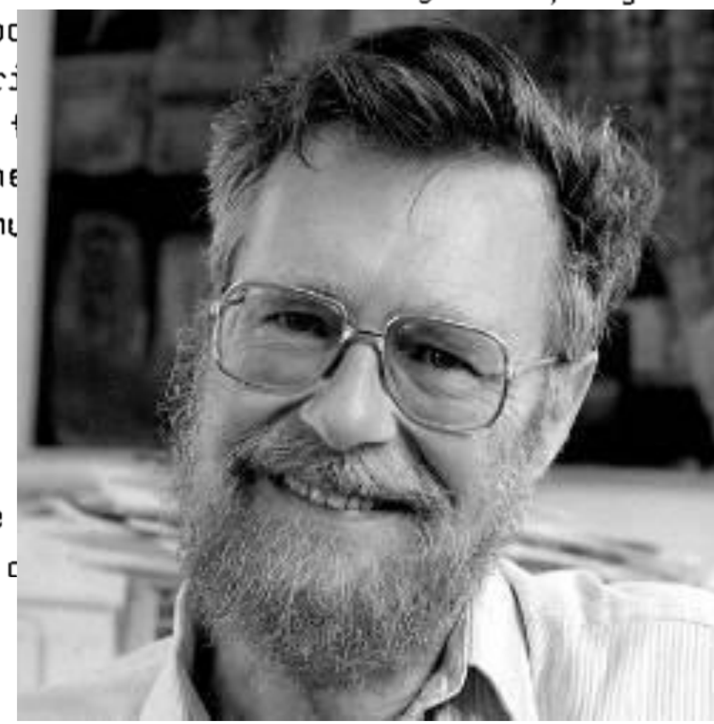
EWD209.html

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 propo
of program generation such as to produce a priori
is treated to show the form that such a control
comes from the field of parallel programming; the
is representative for the way in which a whole mu
actually been constructed.

Introduction.

The more ambitious we become in our machine
becomes the problem of program correctness. The c



Producer - Consumer

Producător **Buffer** **Consumator**

Pot să pun?

Pot să iau?

Producer - Consumer

Producător | **Buffer** | **Consumator**

Pot să pun? | Pot să iau?

Producer - Consumer

Producător | **Buffer** | **Consumator**

Pot să pun?

Pot să iau?

Producer - Consumer

Producător **Buffer** **Consumator**

Pot să pun? Pot să iau?

Producer - Consumer

Producător | **Buffer** | **Consumator**

Pot să pun?

Pot să iau?

Producer - Consumer

Producător | **Buffer** | **Consumator**

Full = 0
Empty = 1

Producător:

P(**Empty**);

B=EL;

V(**Full**);

Consumator:

P(**Full**);

EL = B;

V(**Empty**);

Producer - Consumer

Producător

Buffer

Full = 0

Empty = 1

Empty.lock();

B=EL;

Full.unlock();

Consumator

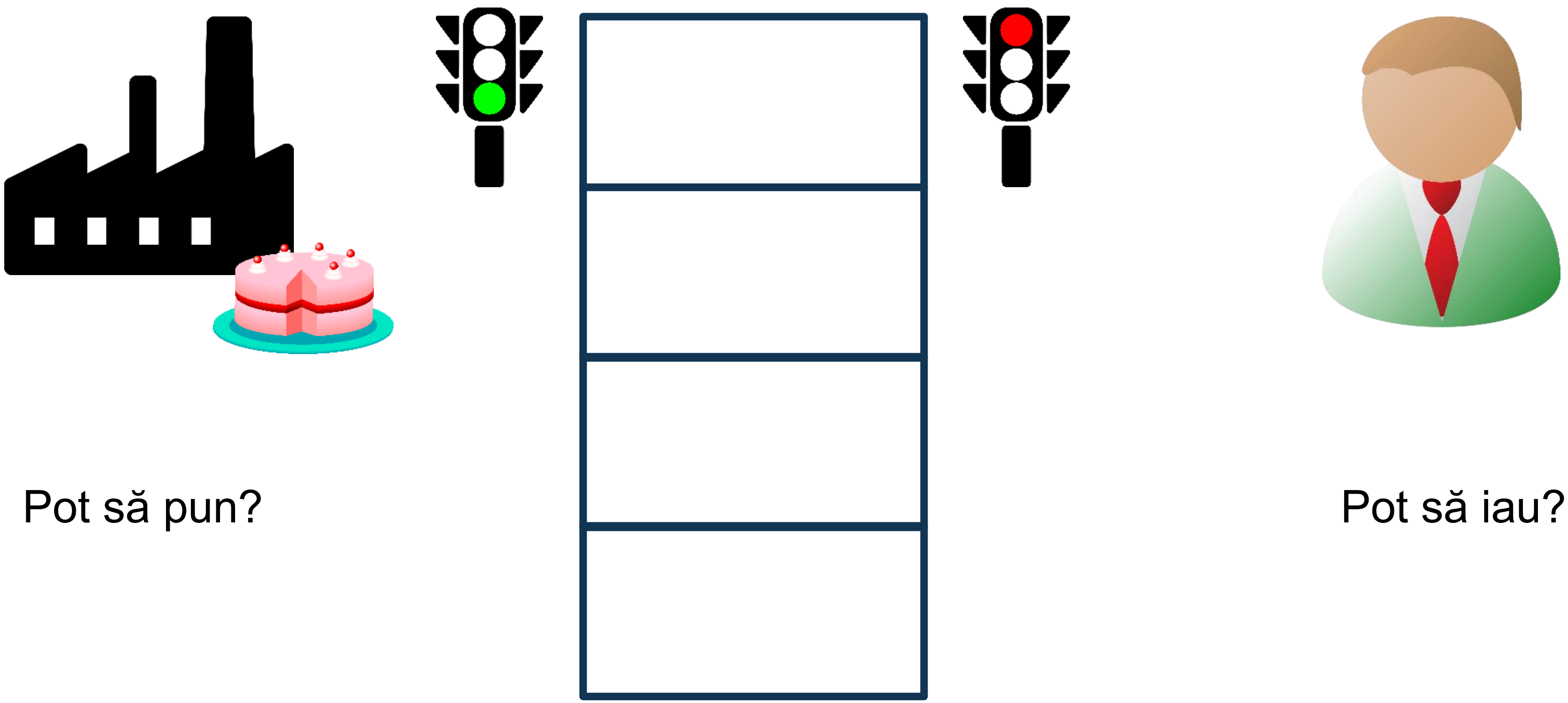
Full.lock();

EL = B;

Empty.unlock();

Producer - Consumer

Producător **Buffer** **Consumator**

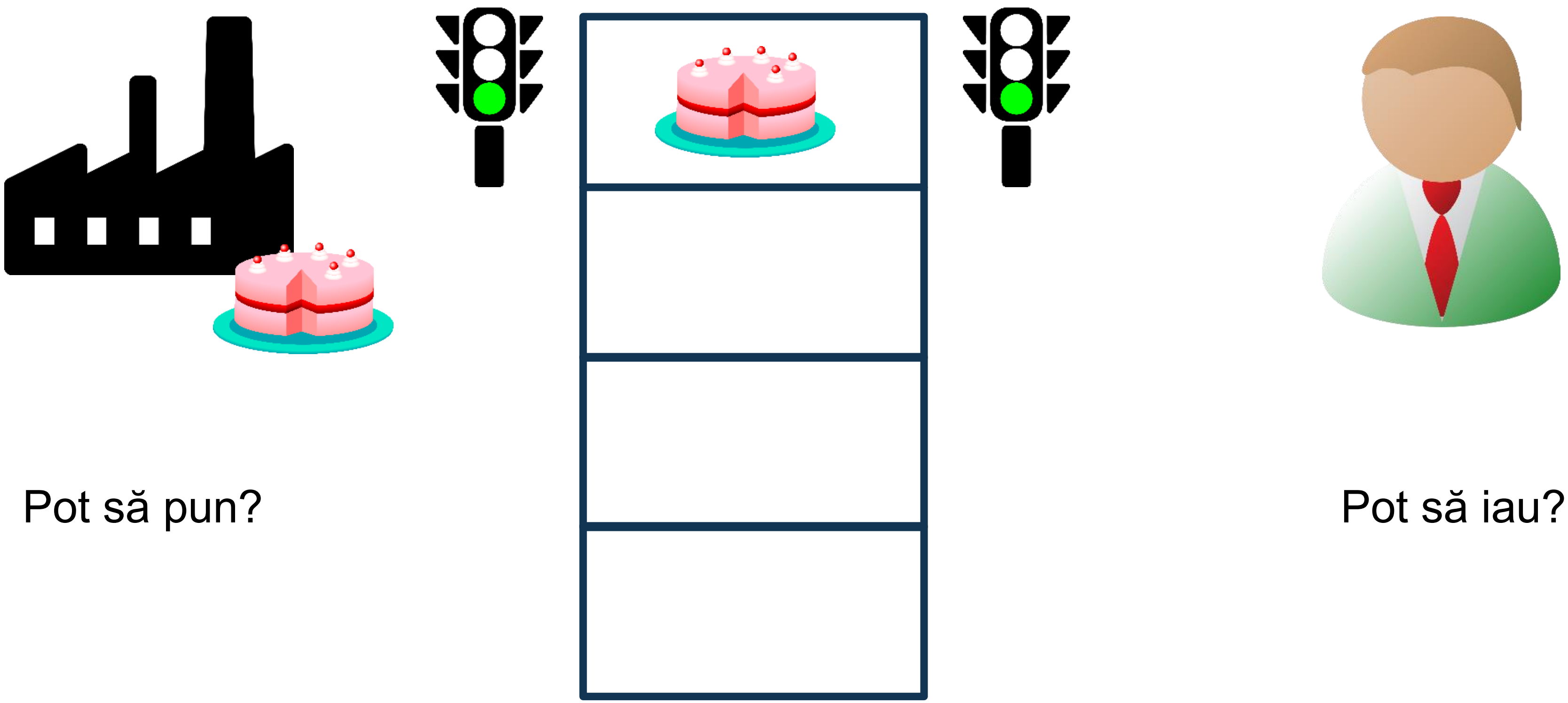


Pot să pun?

Pot să iau?

Producer - Consumer

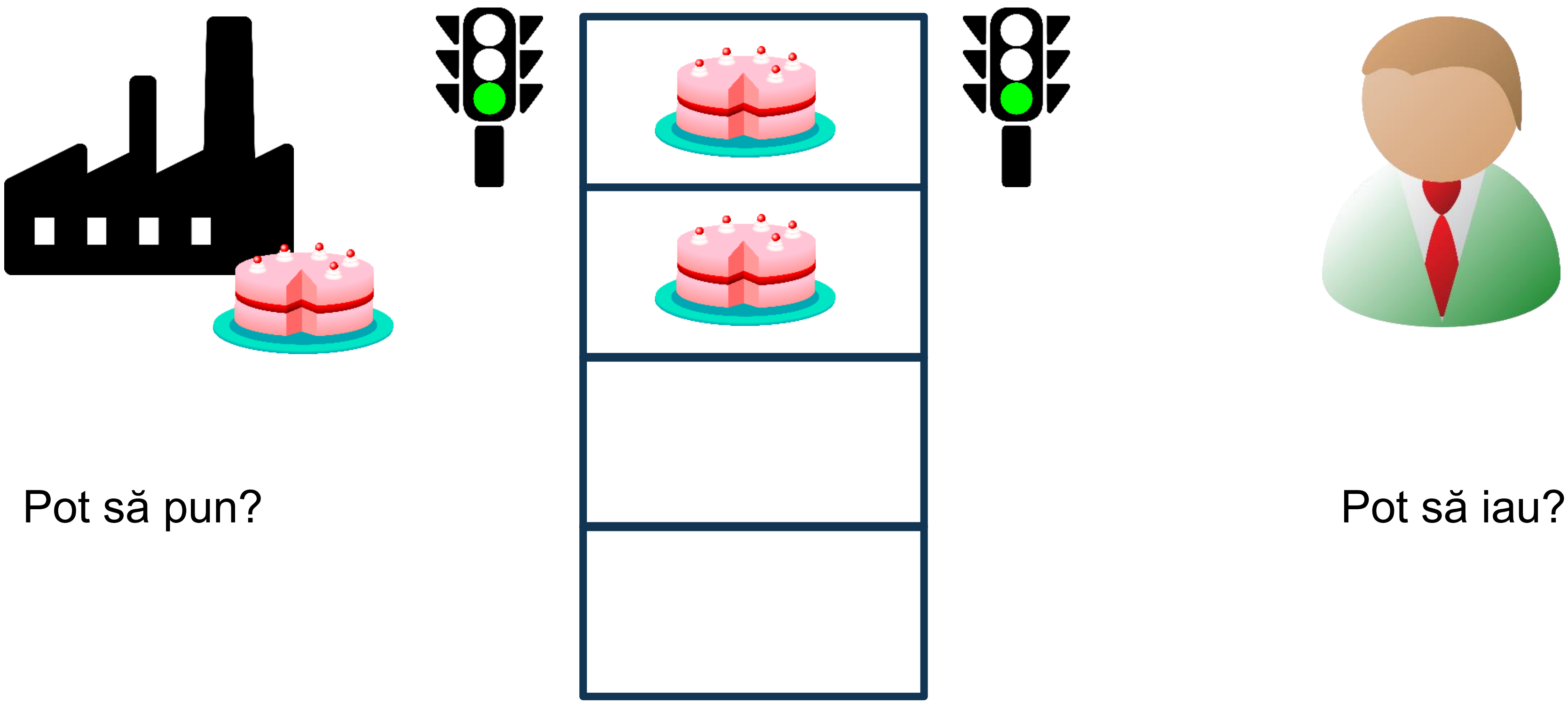
Producător **Buffer** **Consumator**



Pot să pun? Pot să iau?

Producer - Consumer

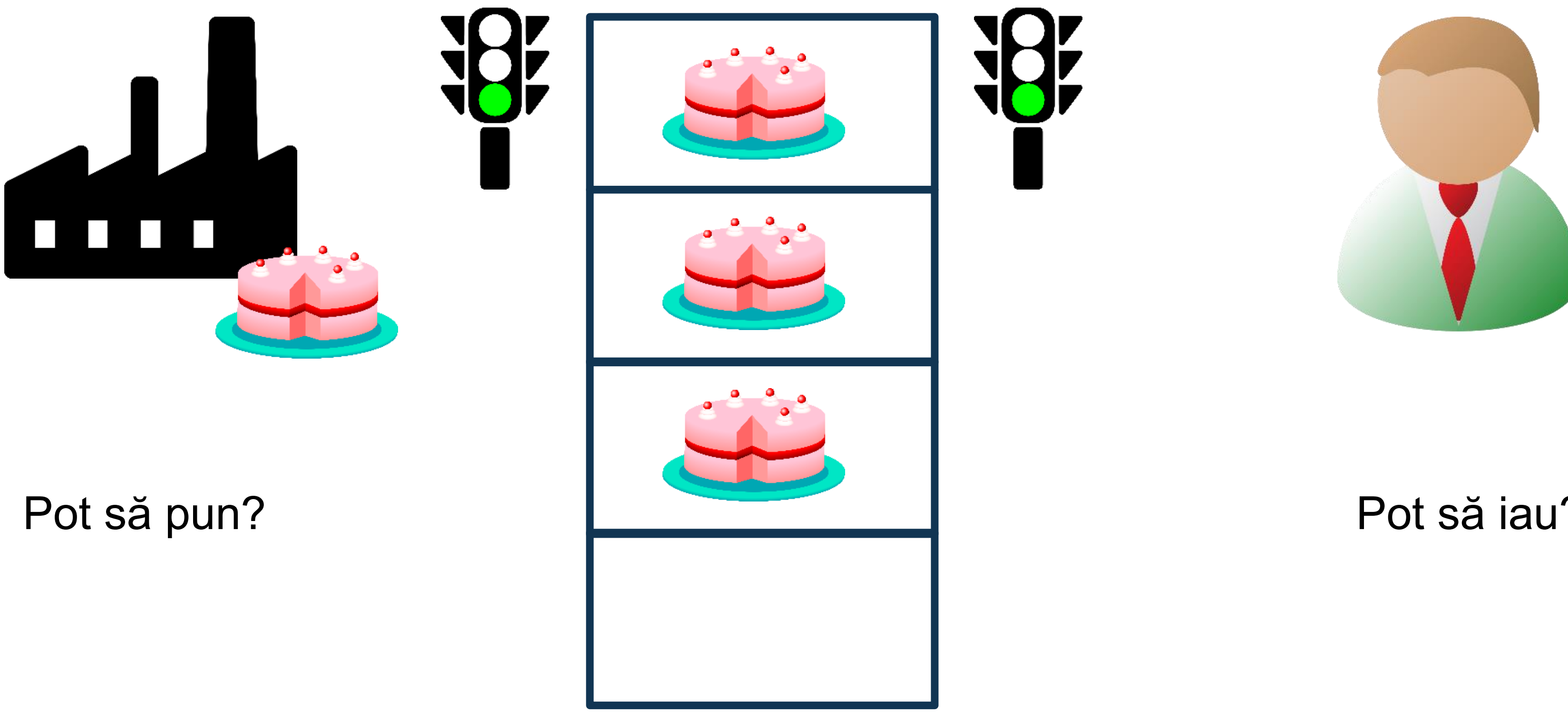
Producător **Buffer** **Consumator**



Pot să pun? Pot să iau?

Producer - Consumer

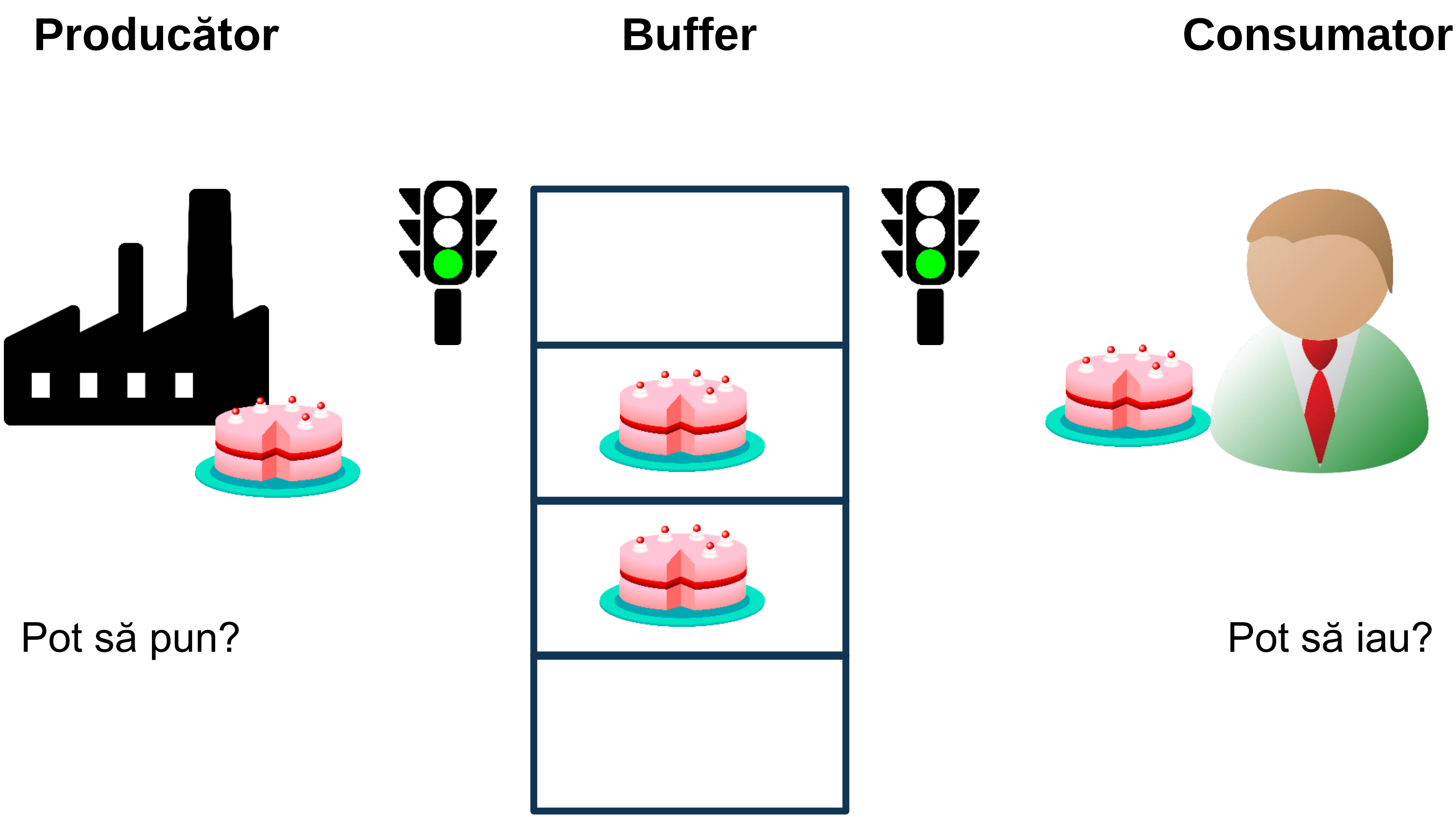
Producător | **Buffer** | **Consumator**



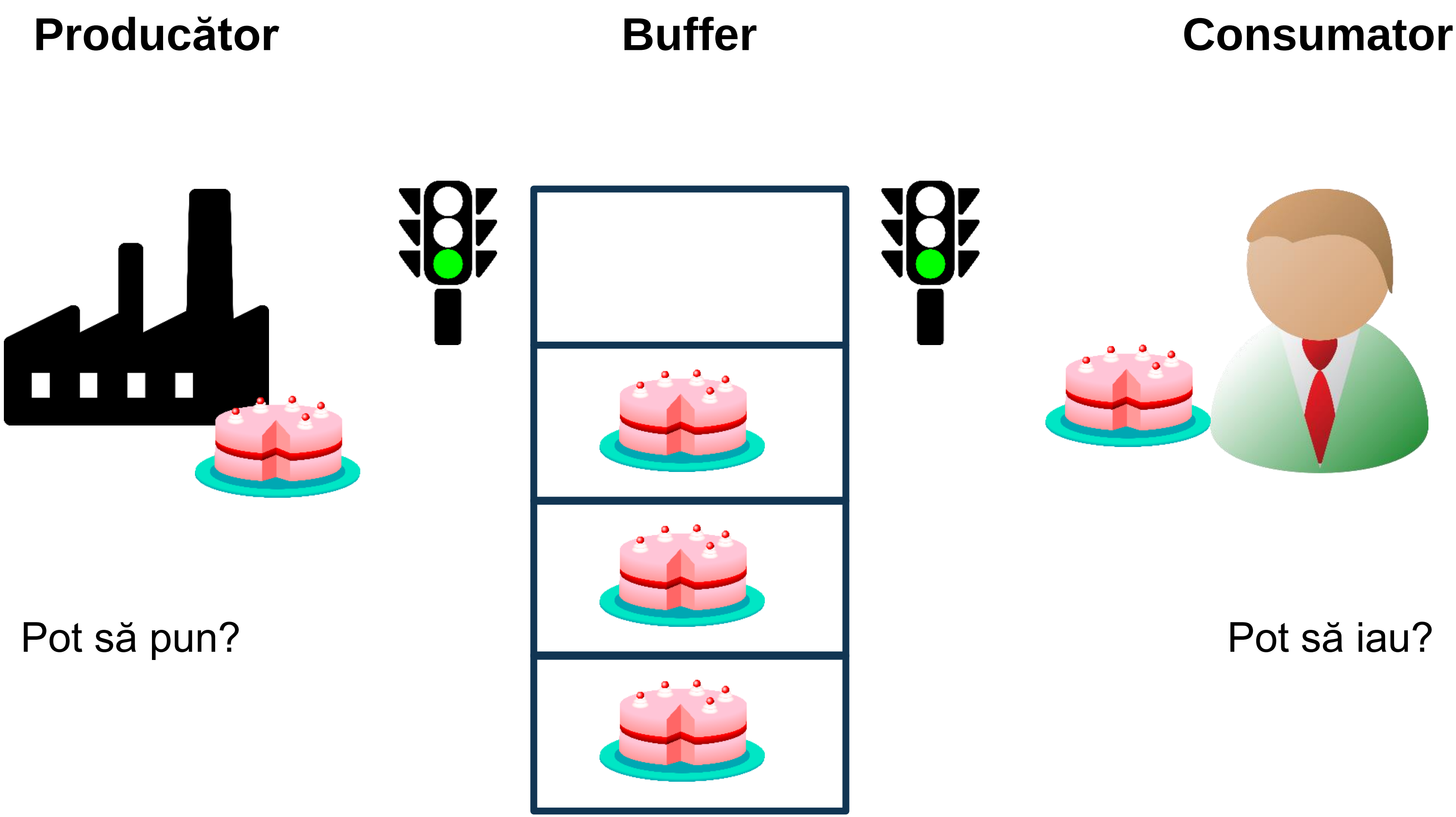
Pot să pun?

Pot să iau?

Producer - Consumer

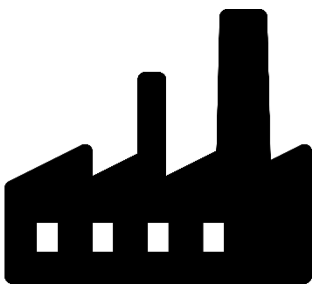


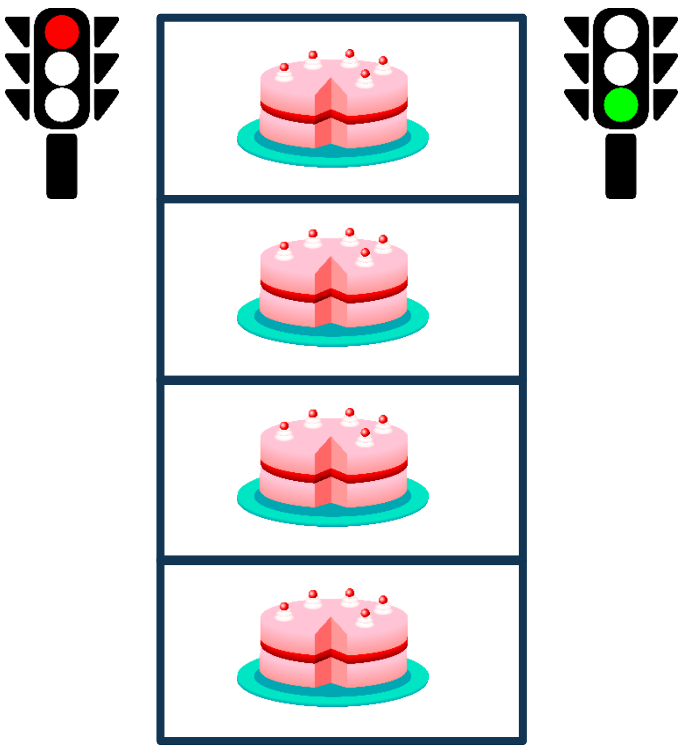
Producer - Consumer

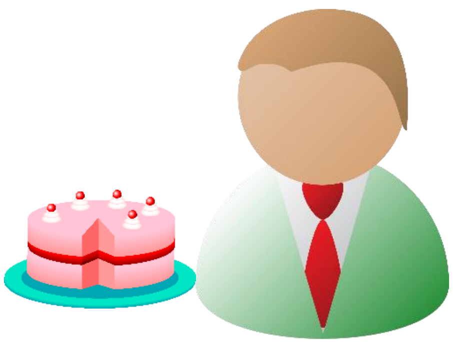


Producer - Consumer

Producător | **Buffer** | **Consumator**





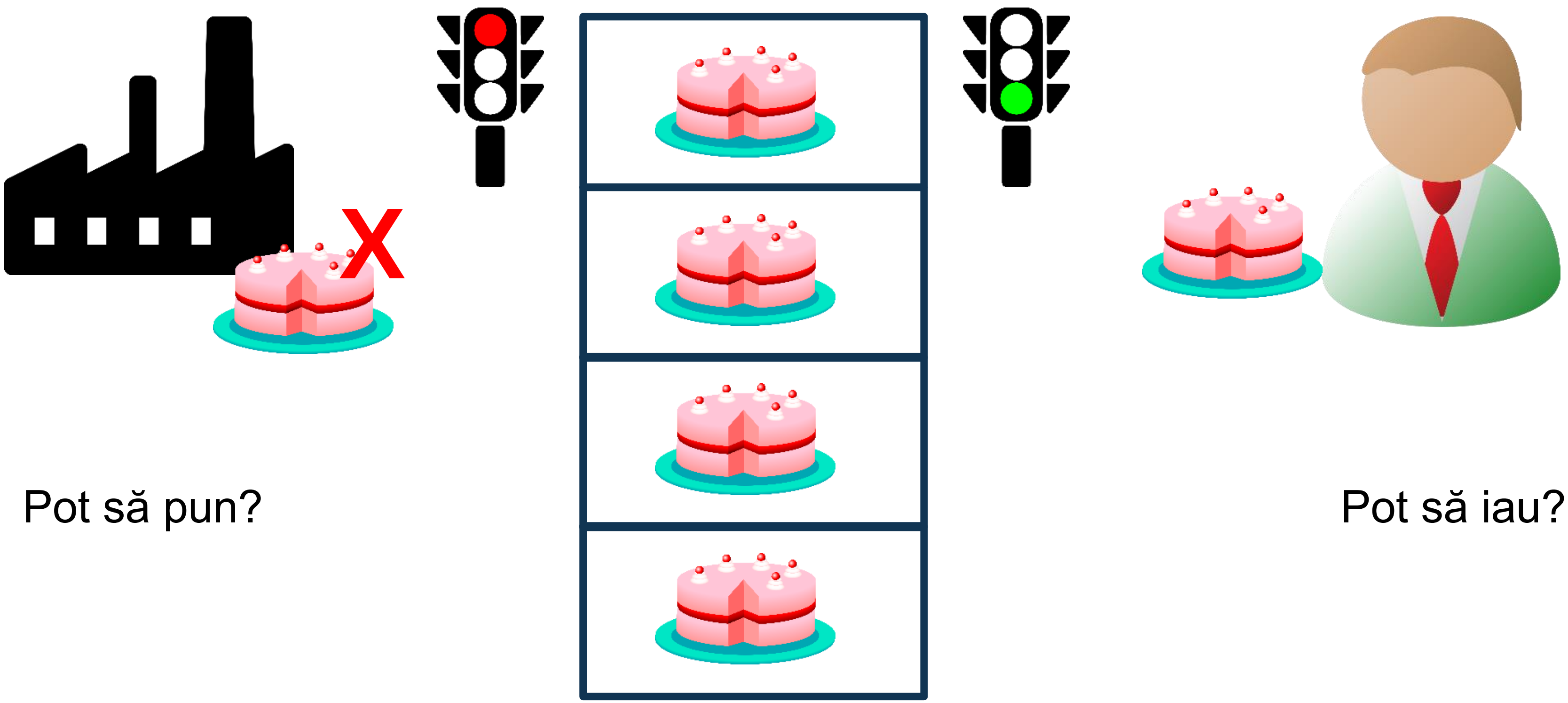


Pot să pun?

Pot să iau?

Producer - Consumer

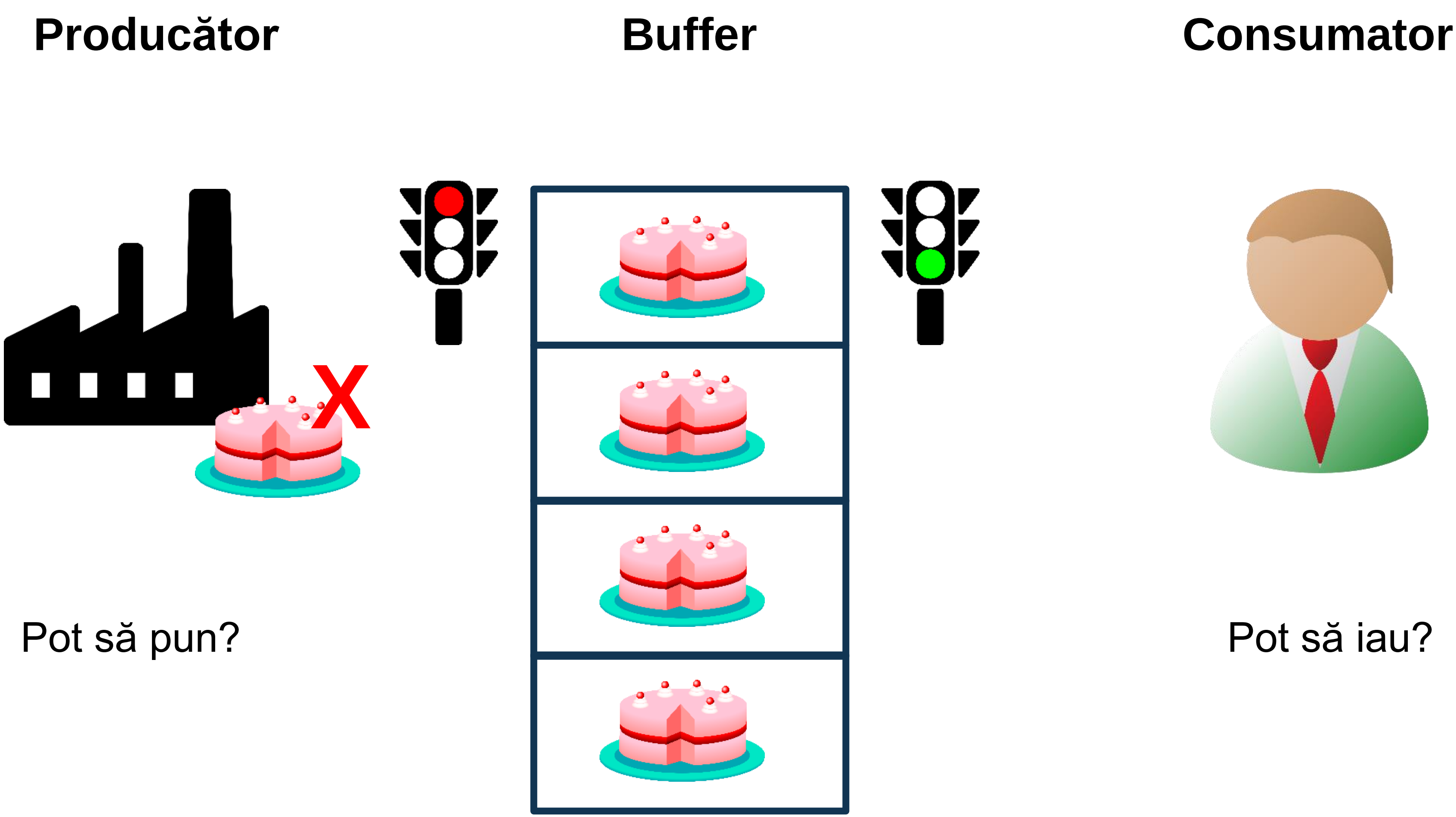
Producător | **Buffer** | **Consumator**



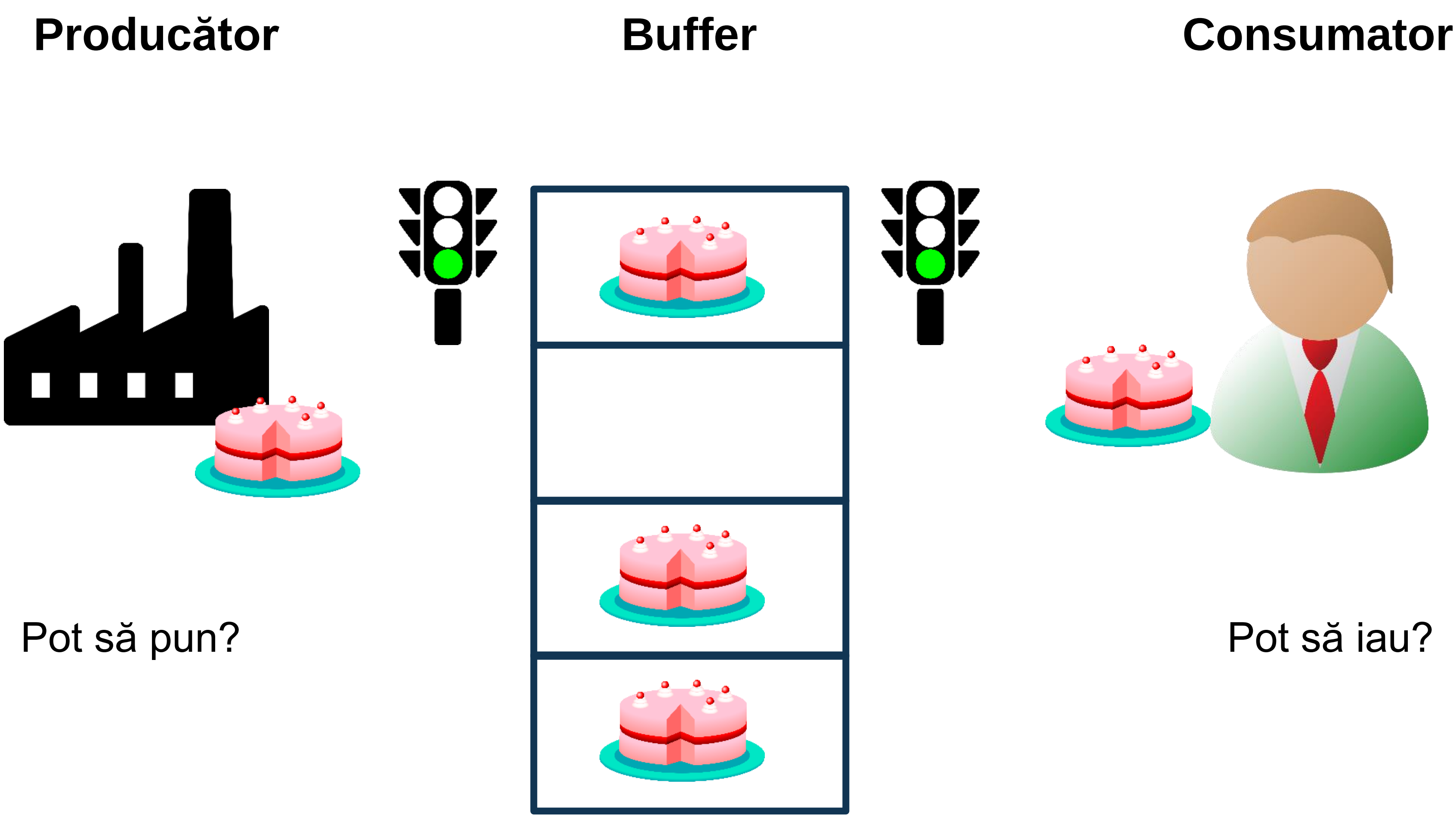
Pot să pun?

Pot să iau?

Producer - Consumer

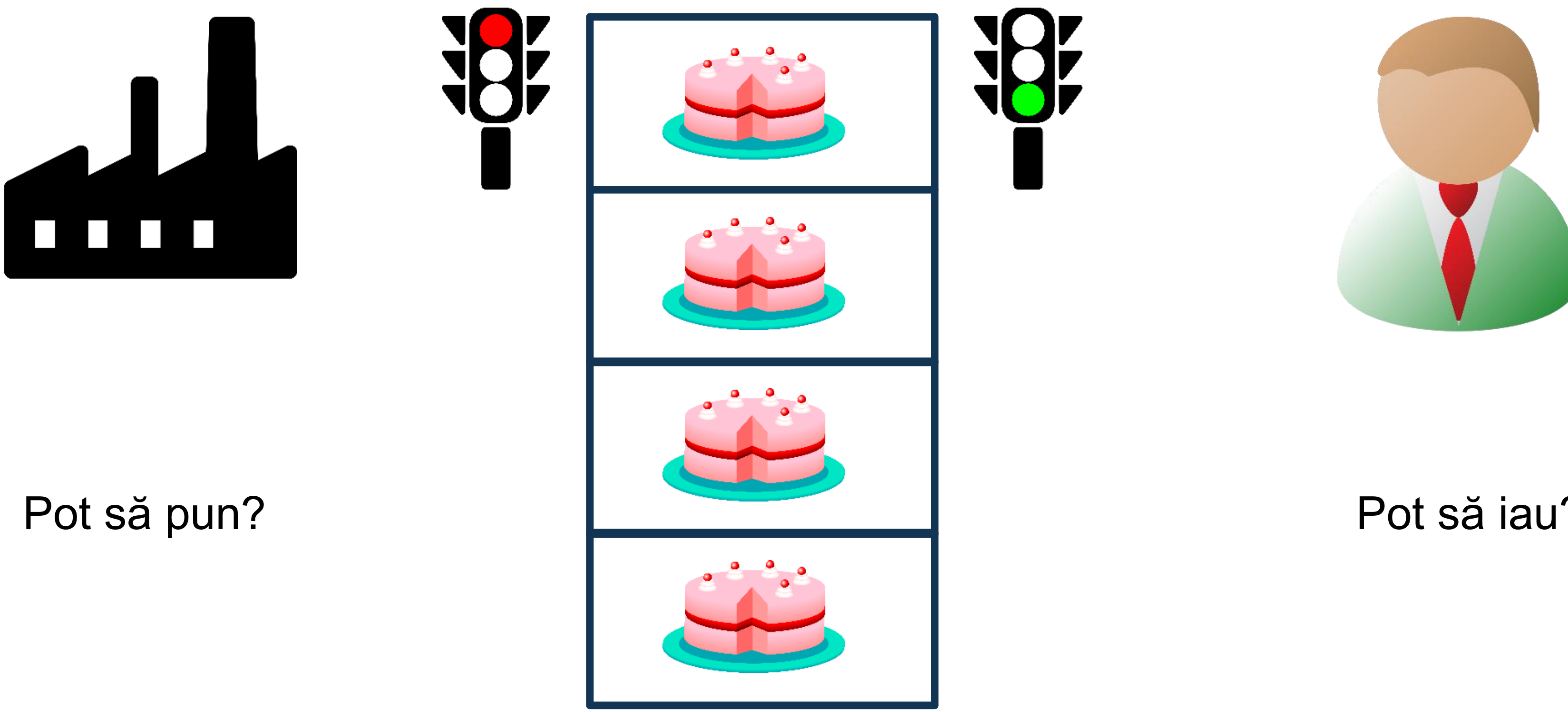


Producer - Consumer



Producer - Consumer

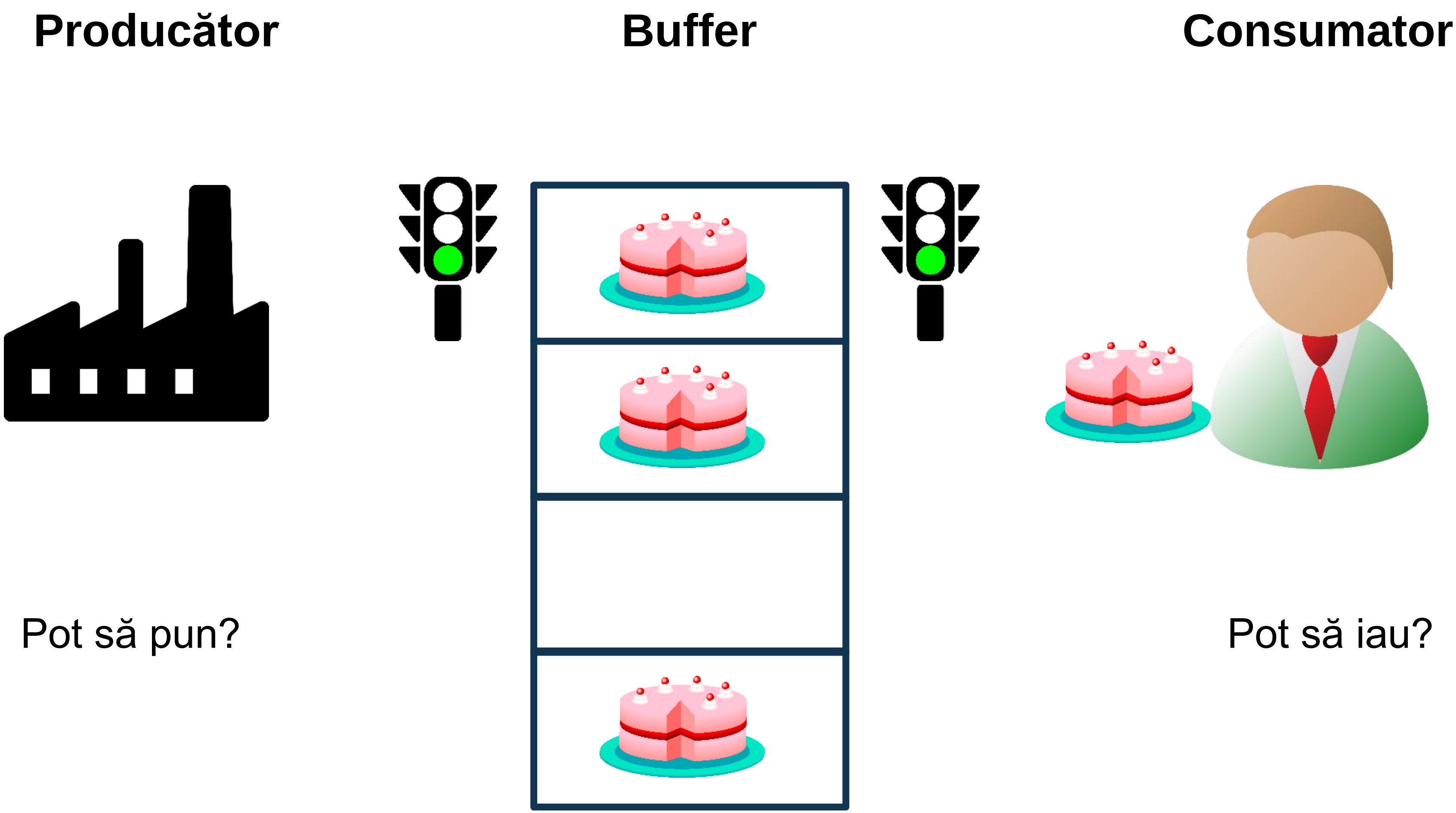
Producător | **Buffer** | **Consumator**



Pot să pun?

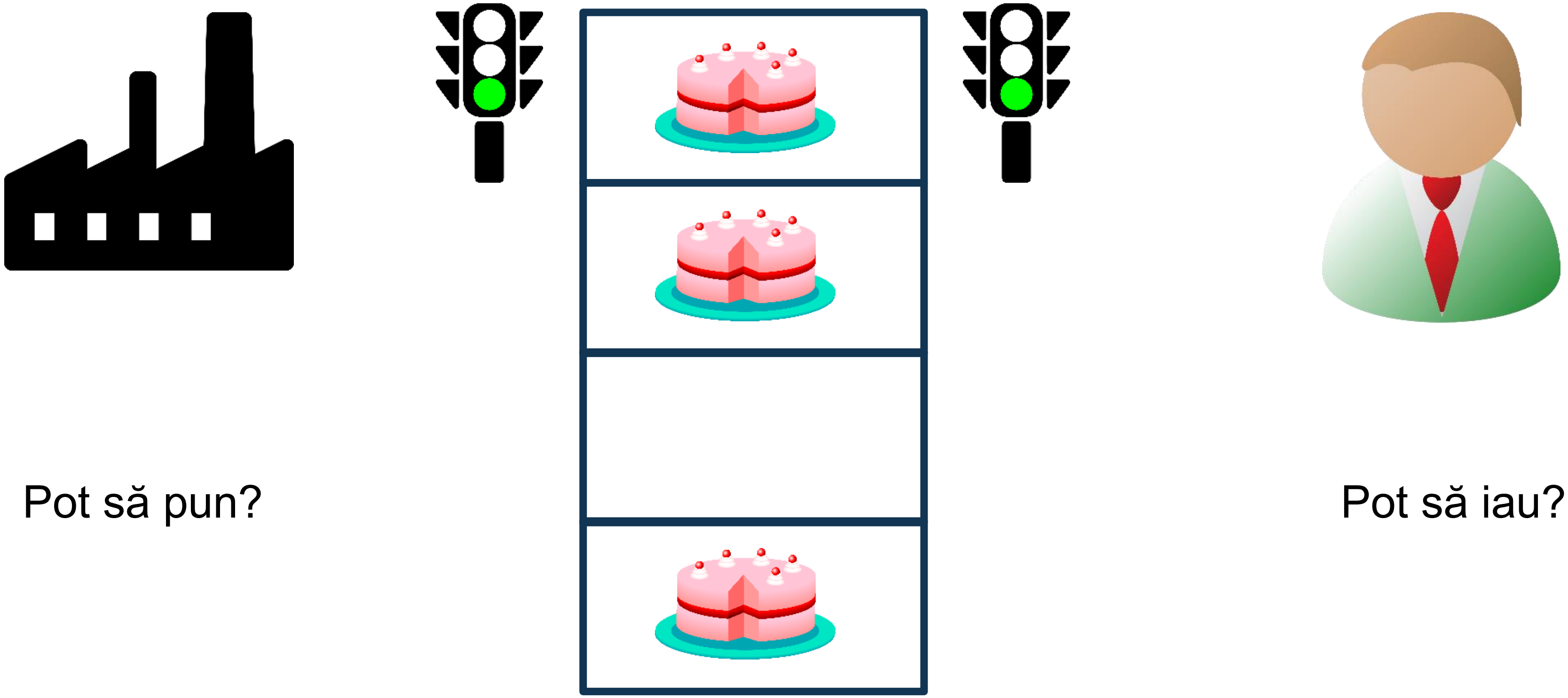
Pot să iau?

Producer - Consumer



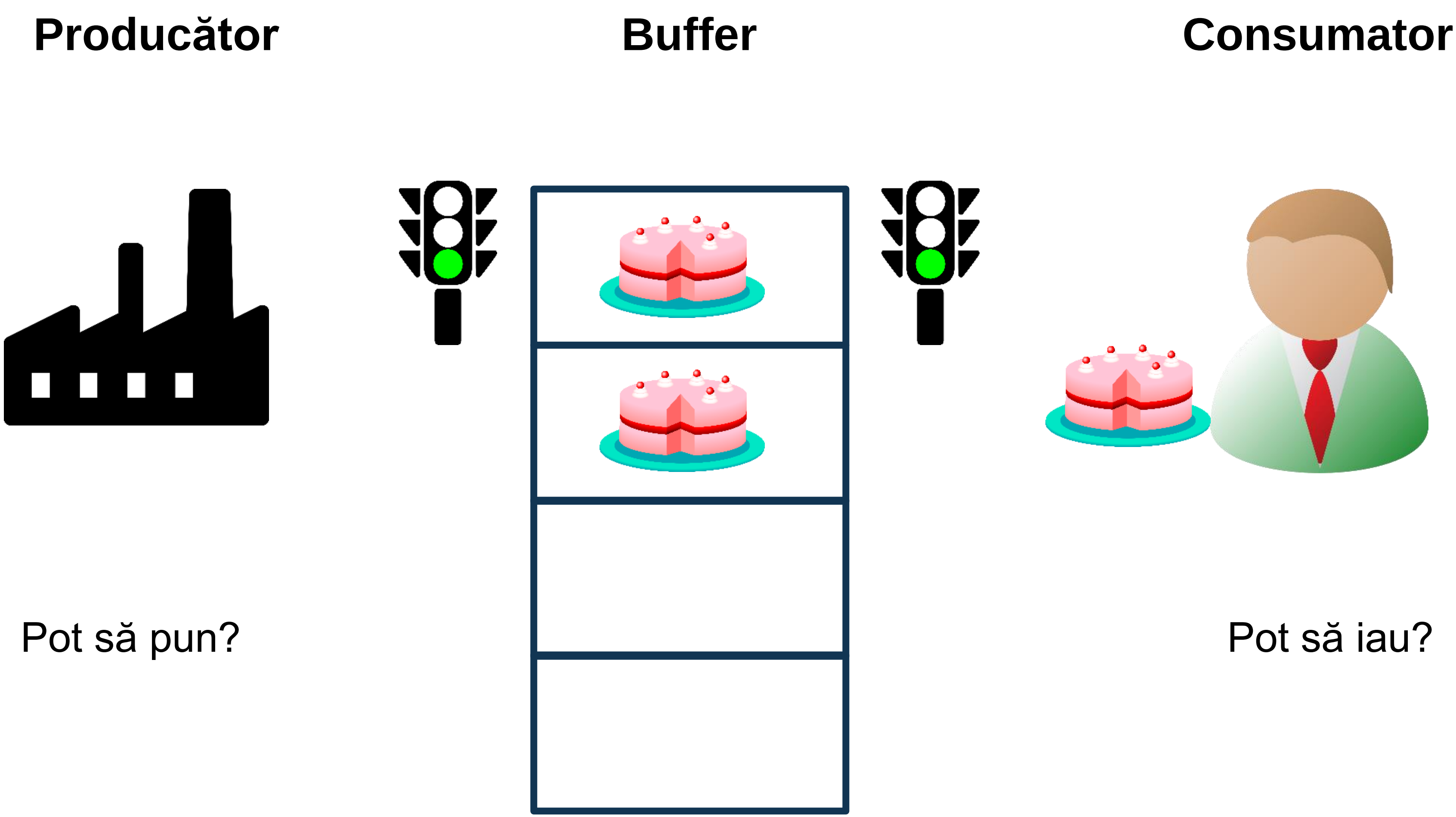
Producer - Consumer

Producător **Buffer** **Consumator**

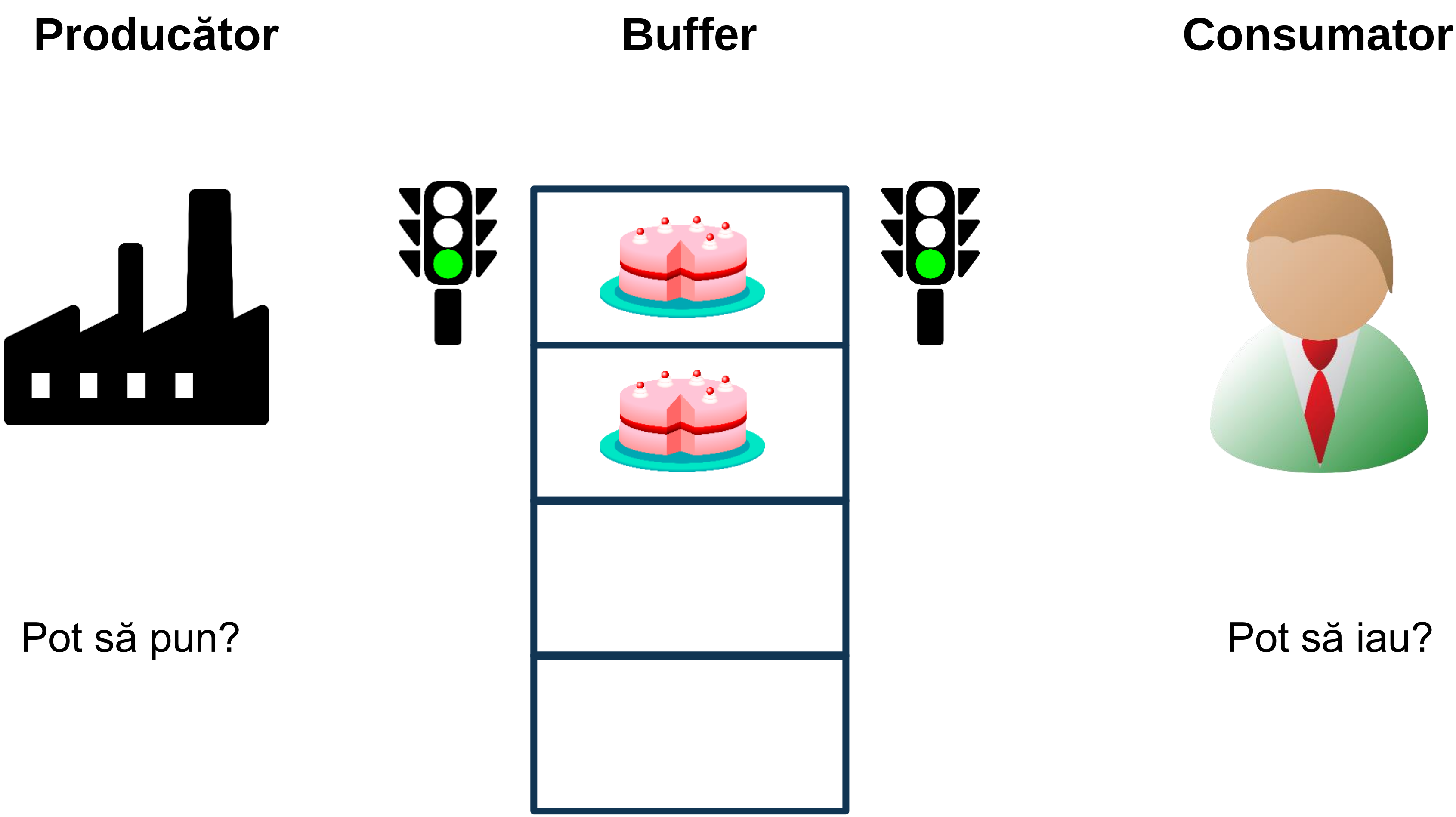


Pot să pun? Pot să iau?

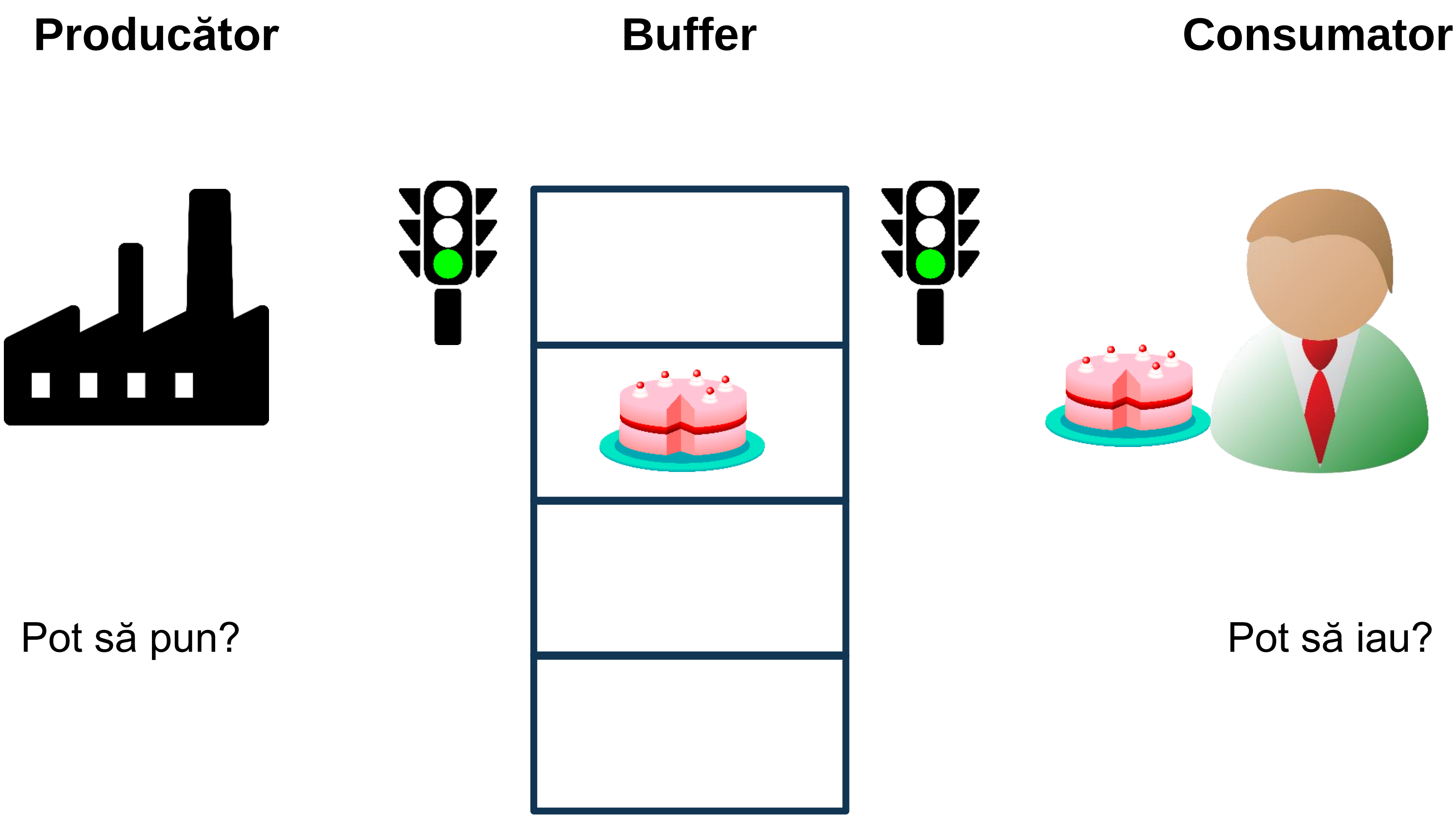
Producer - Consumer



Producer - Consumer

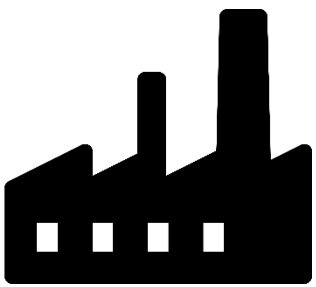


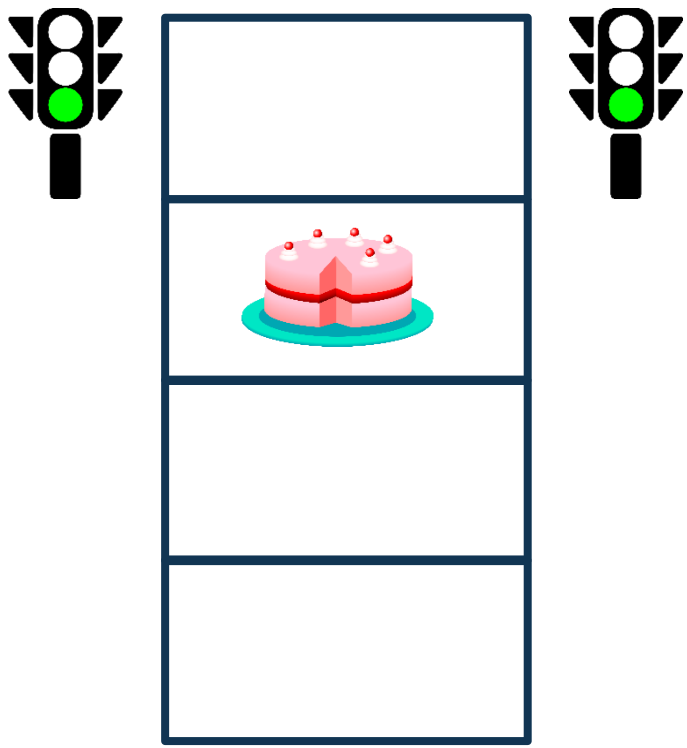
Producer - Consumer

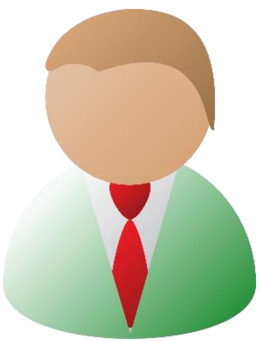


Producer - Consumer

Producător **Buffer** **Consumator**



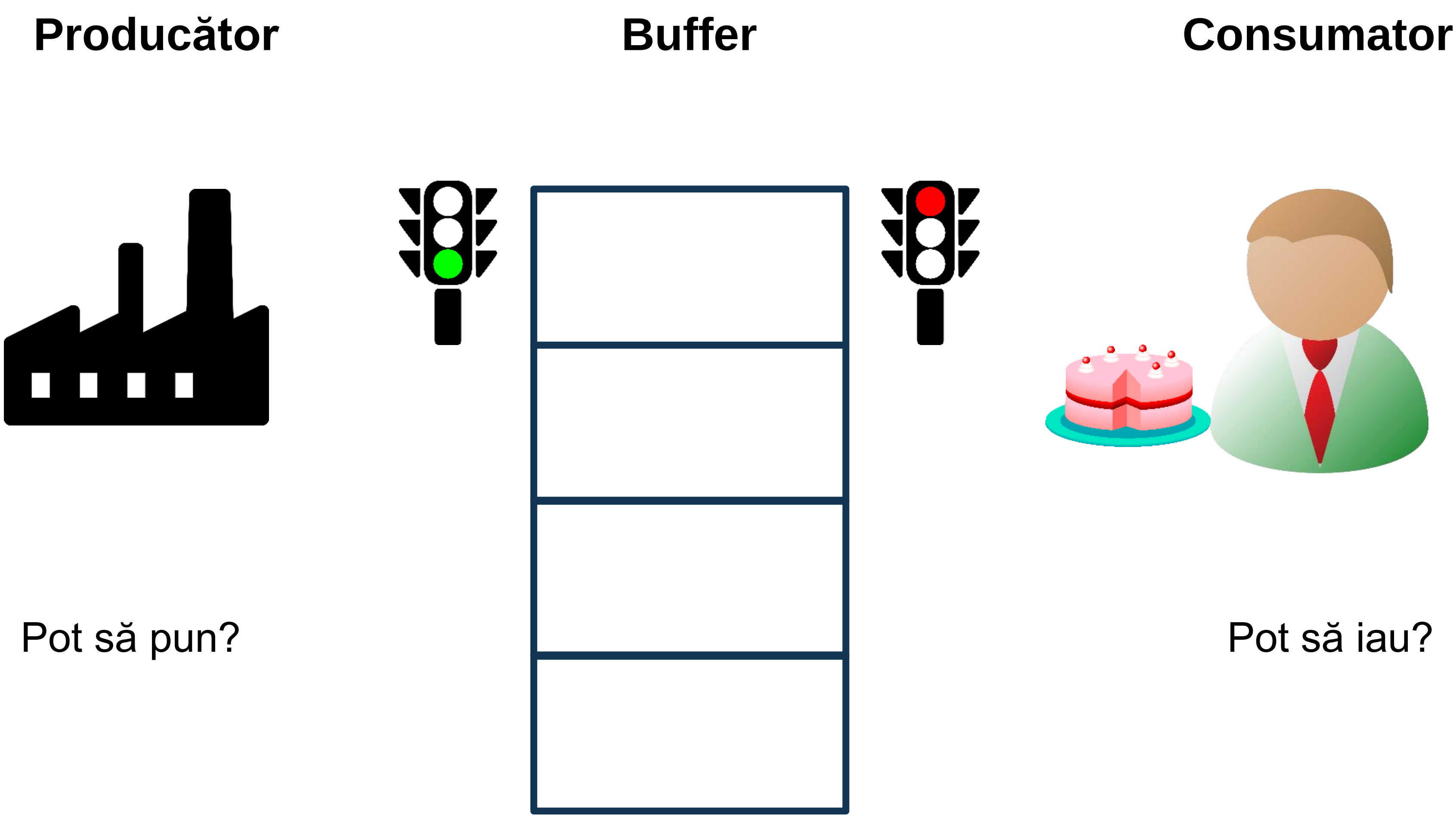




Pot să pun?

Pot să iau?

Producer - Consumer



Producer - Consumer

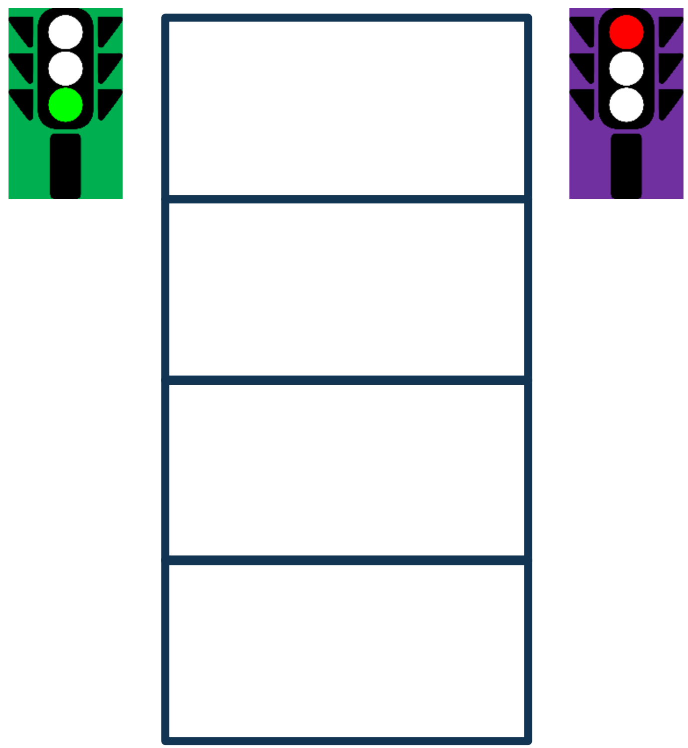
Producători

Buffer

Full = 0

Empty = 4

Consumatori



Empty.lock();

B.put(EL);

Full.unlock();

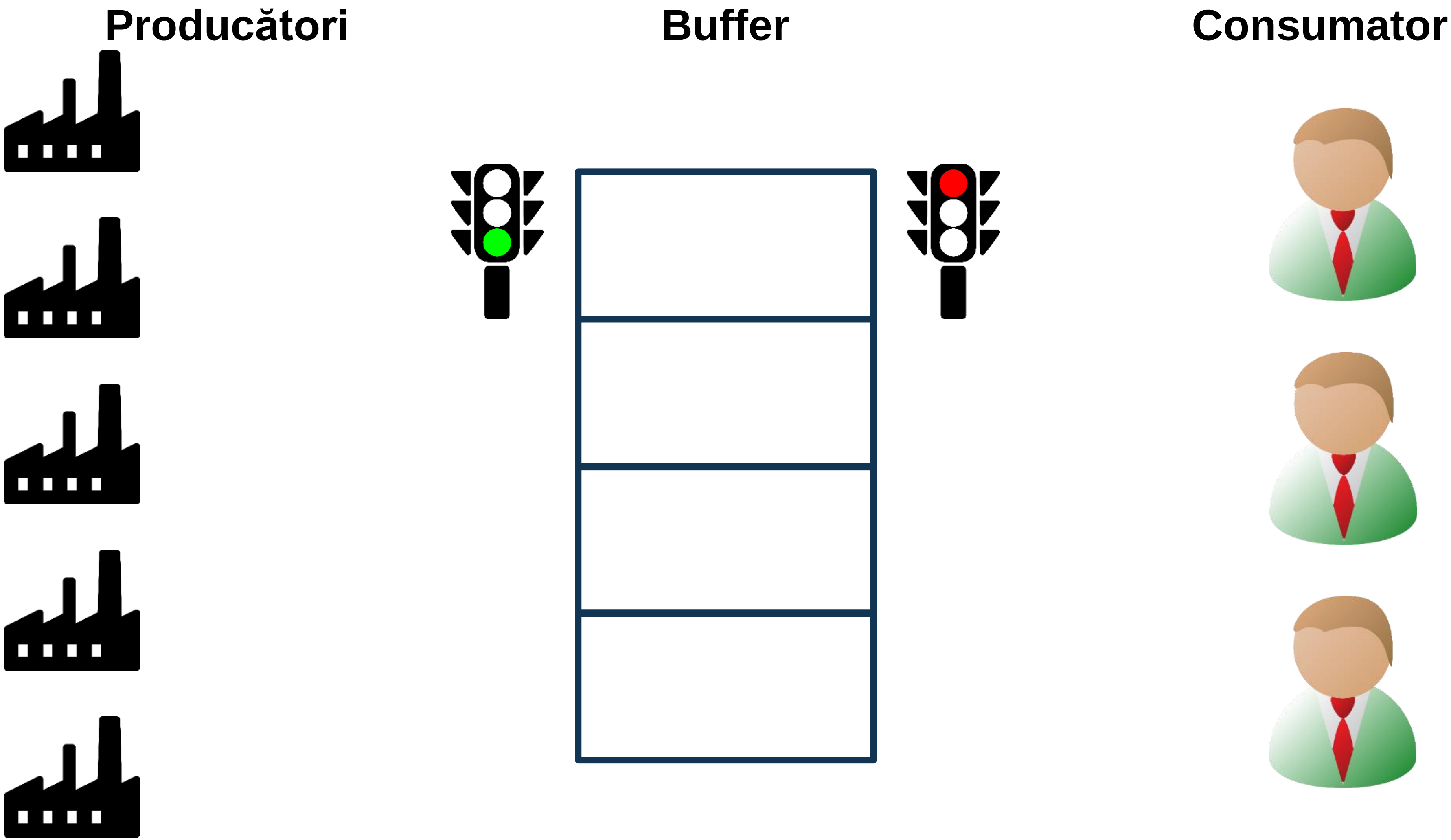
Full.lock();

EL = B.get();

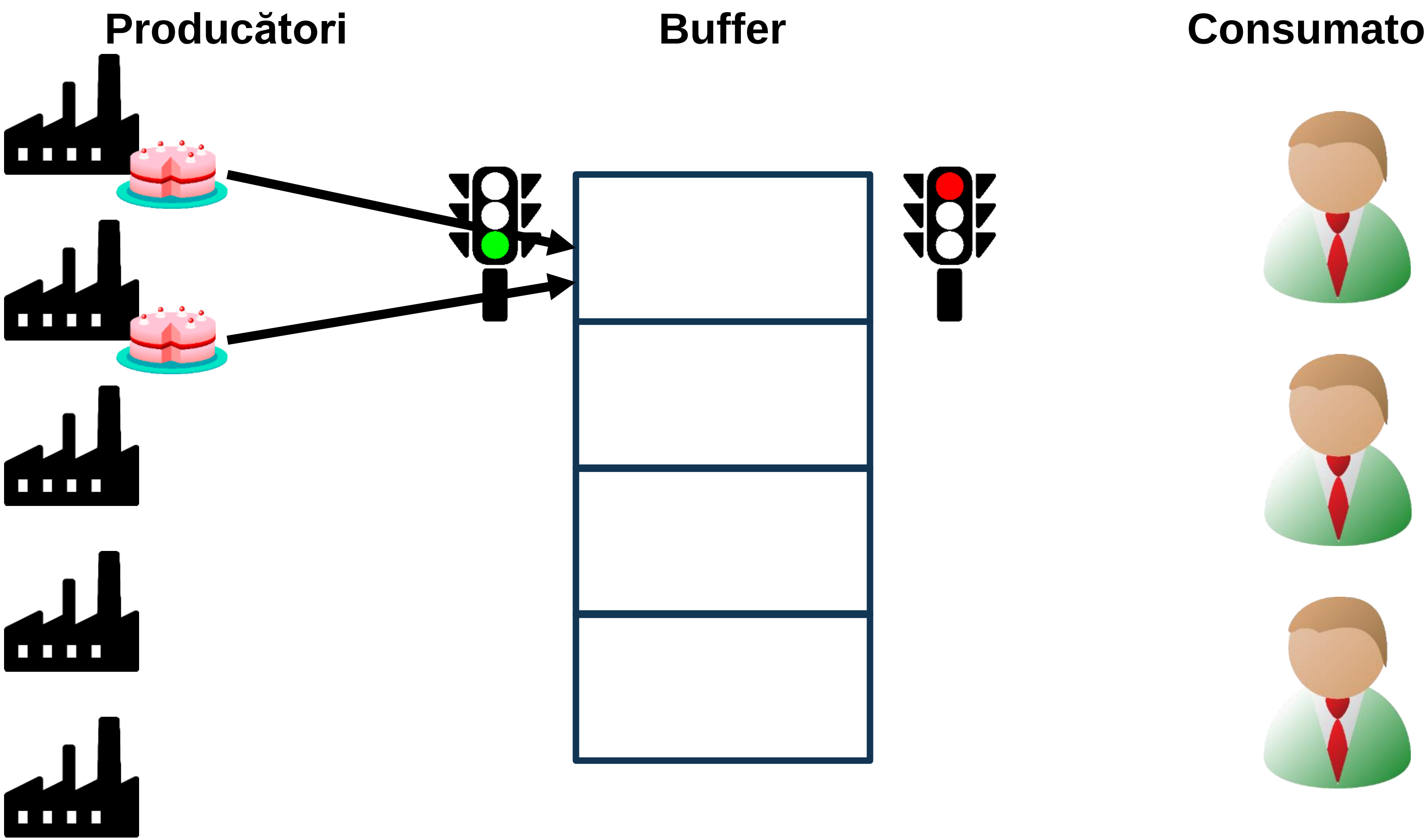
Empty.unlock();

Avem o problemă?

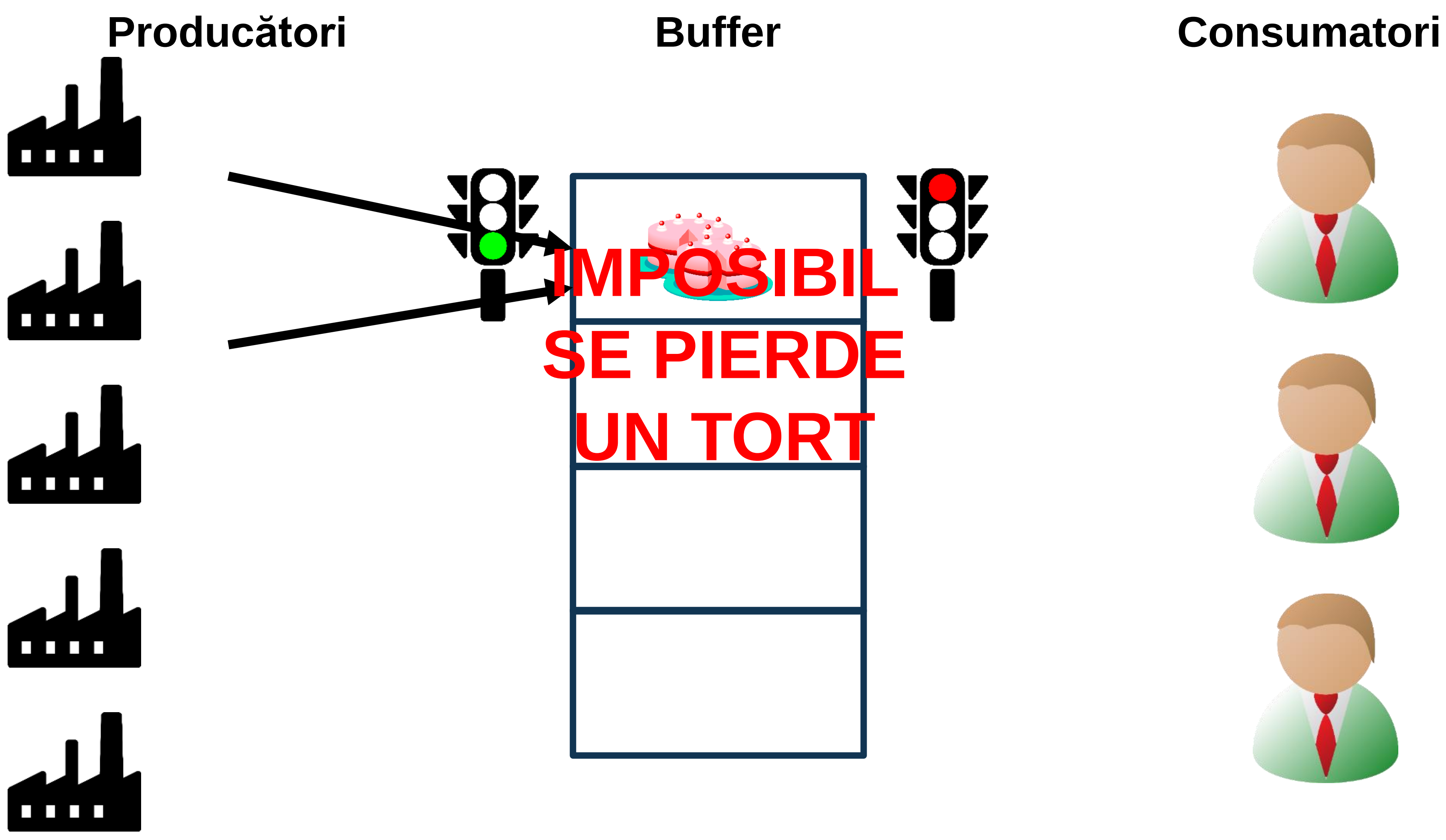
Producer - Consumer



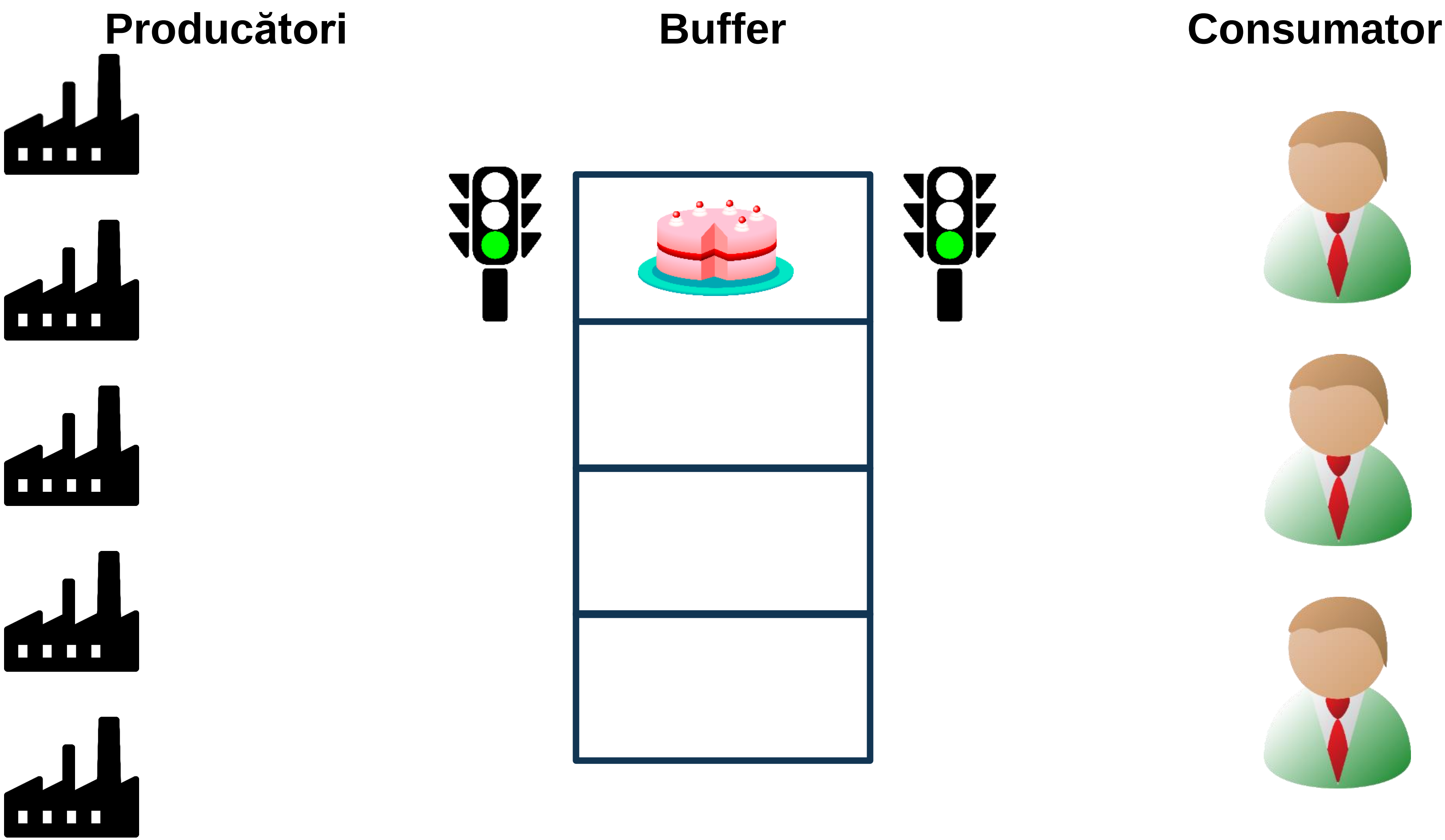
Producer - Consumer



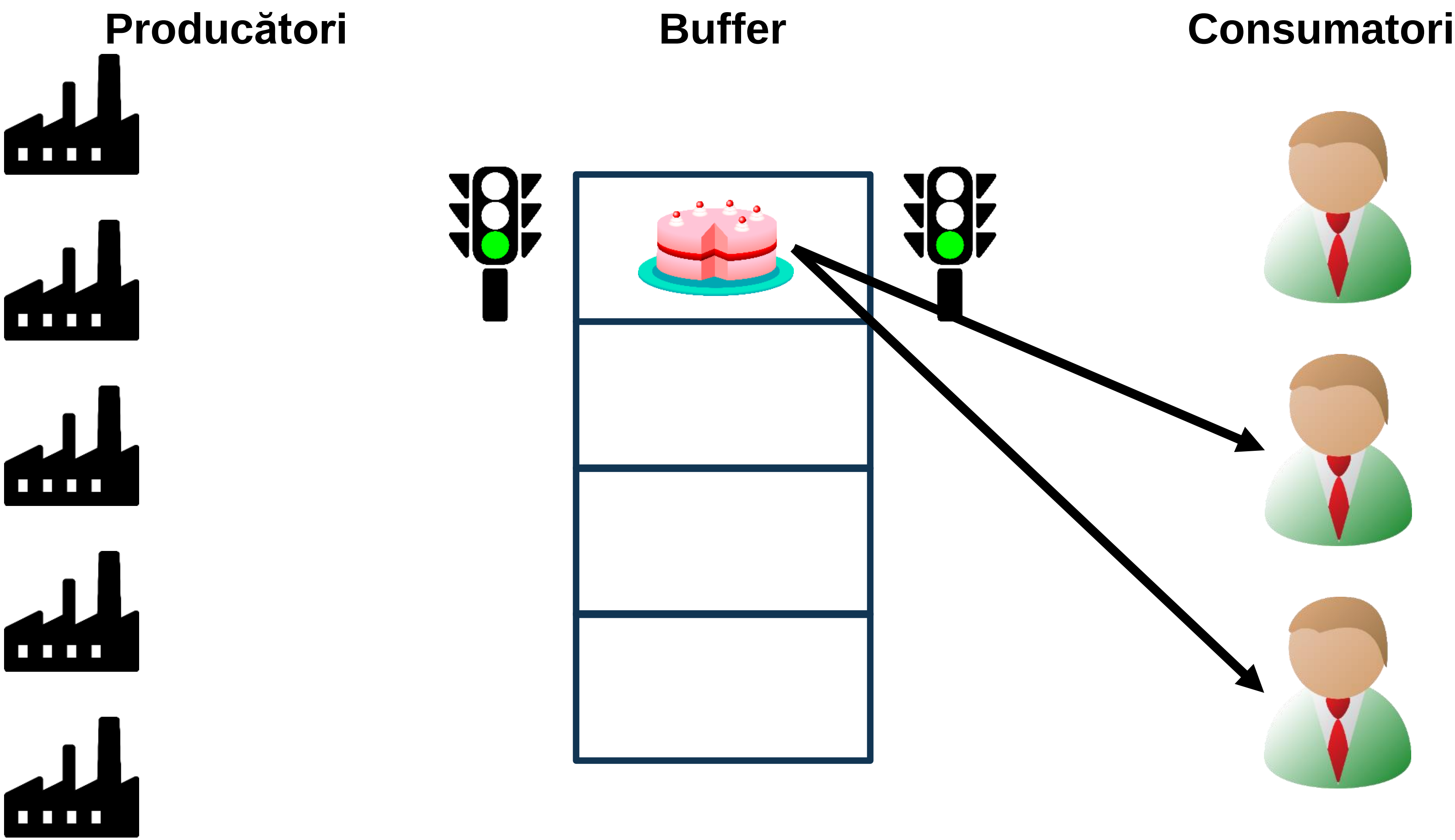
Producer - Consumer



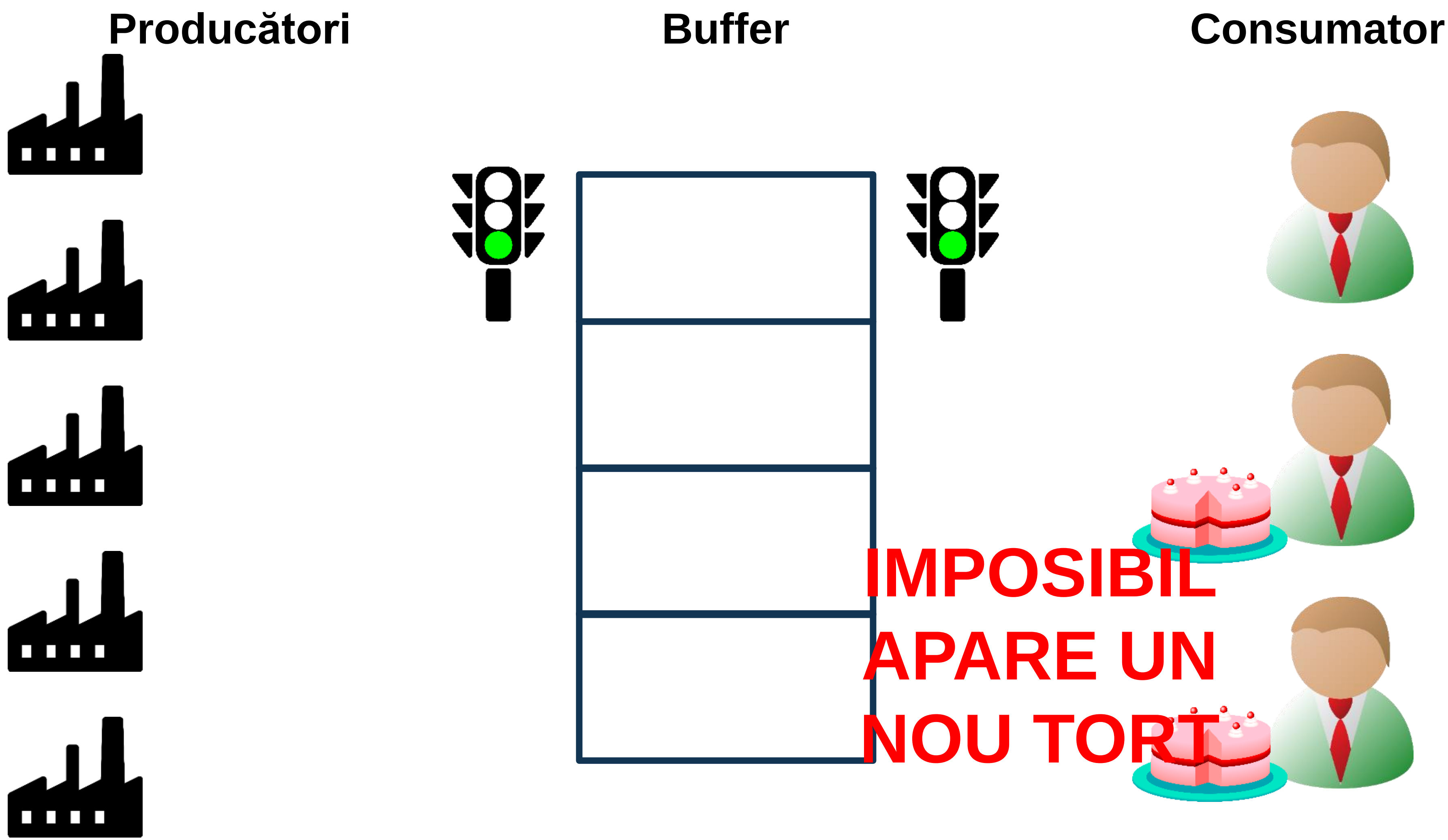
Producer - Consumer



Producer - Consumer

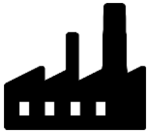


Producer - Consumer



Producer - Consumer

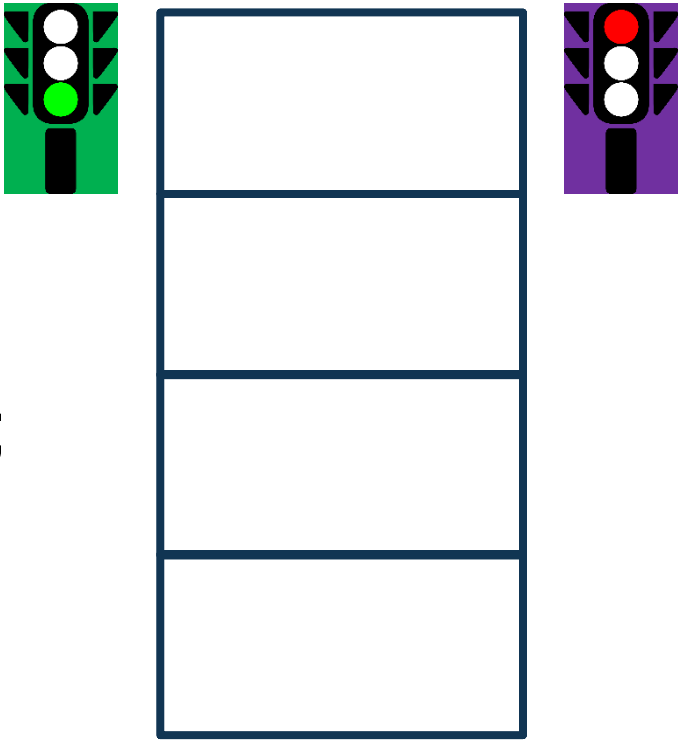
Producători



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

Buffer

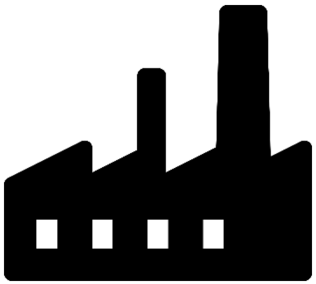
Full = 0
Empty = 4

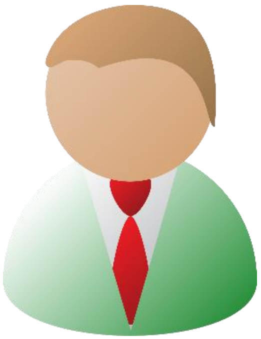


Consumatori

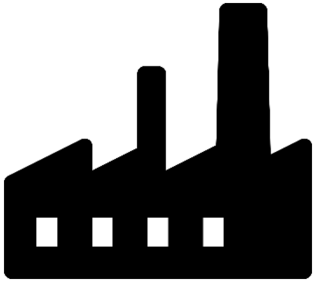
Full.lock();
Mutex.lock();
EL = B.get();
Mutex.unlock();
Empty.unlock();

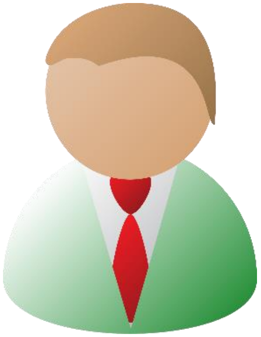
Când folosim buffer?





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.

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

Writer

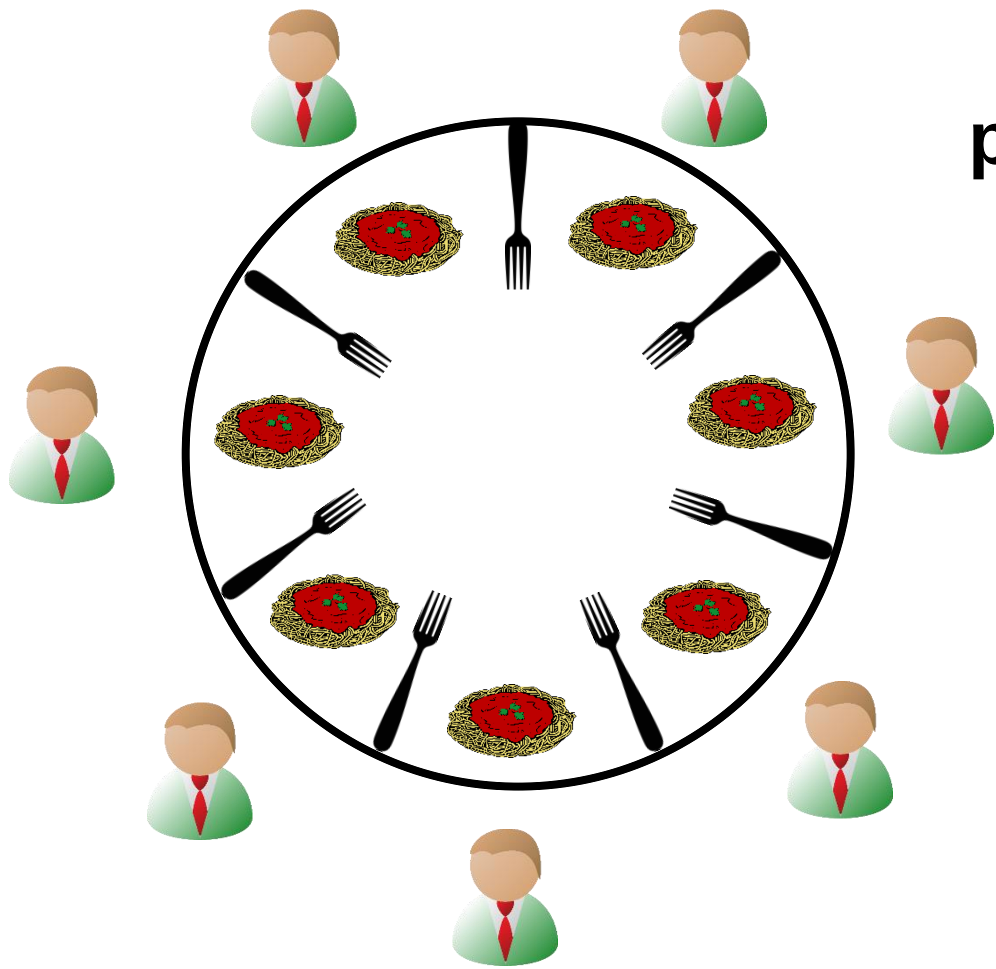
```
Wmutex.lock();
B.put(EL);
Wmutex.unlock();
```

Reader

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

Dinning Philosophers Problem

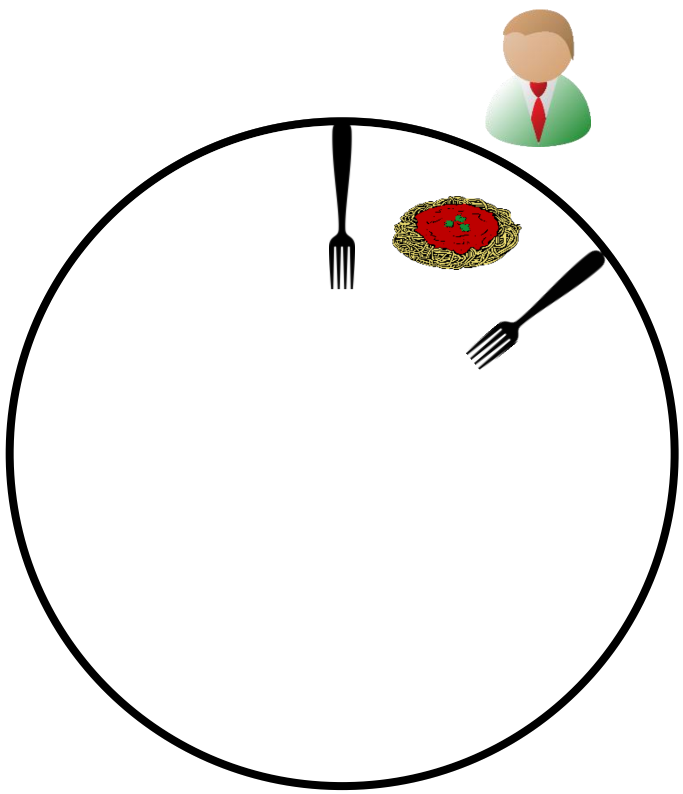
The forks are locks



The philosophers are threads

A philosopher

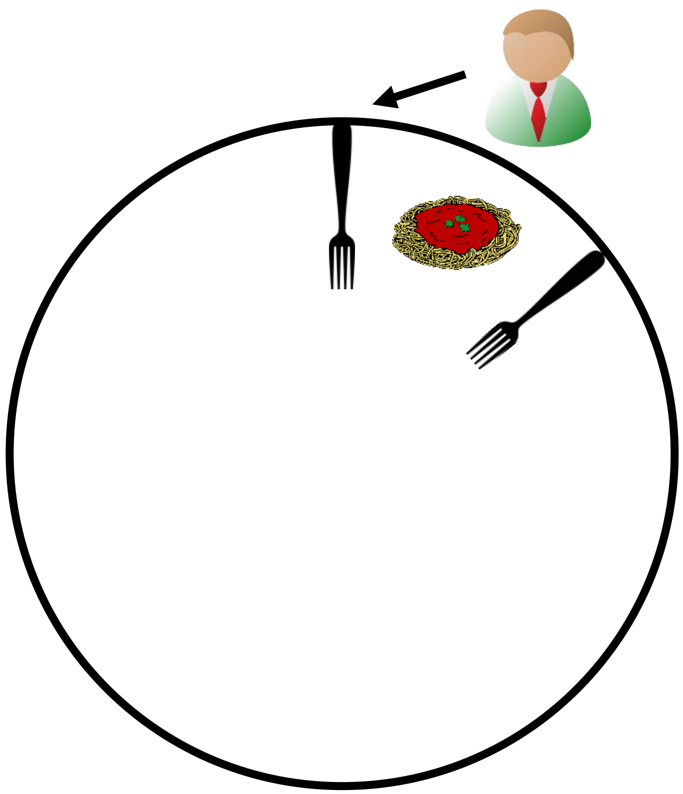
The forks are locks



A philosopher wants to eat.
Eat is a task that required both forks (**locks**).
Eat could simply mean displaying a message.

A philosopher

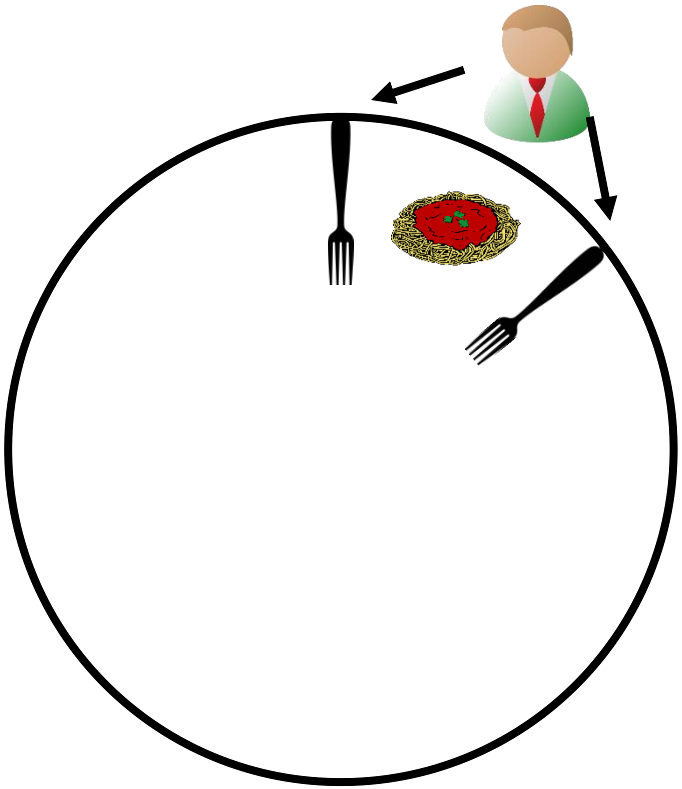
The forks are locks



Take right fork (lock)

A philosopher

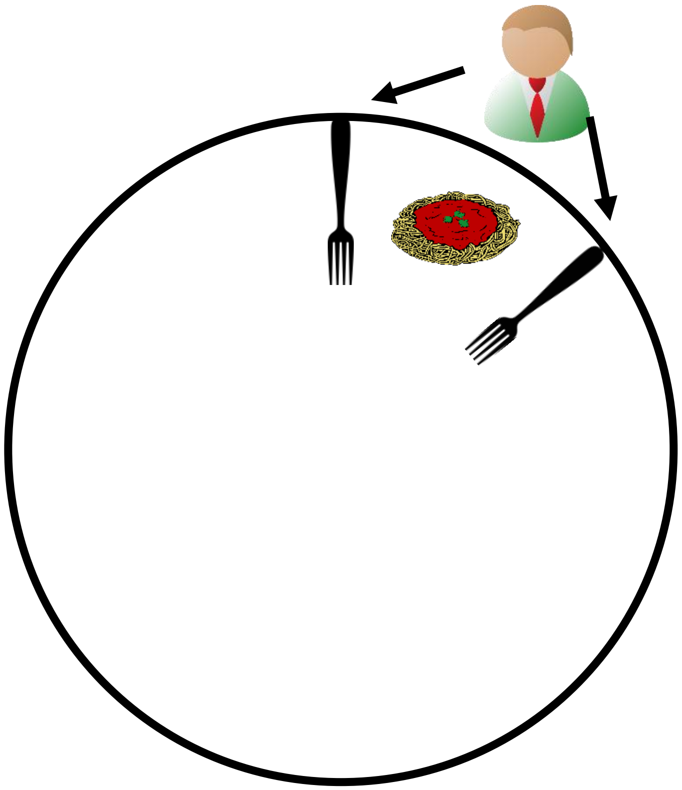
The forks
are **locks**



While holding
right fork (**lock**)
Take the left fork
(**lock**)

A philosopher

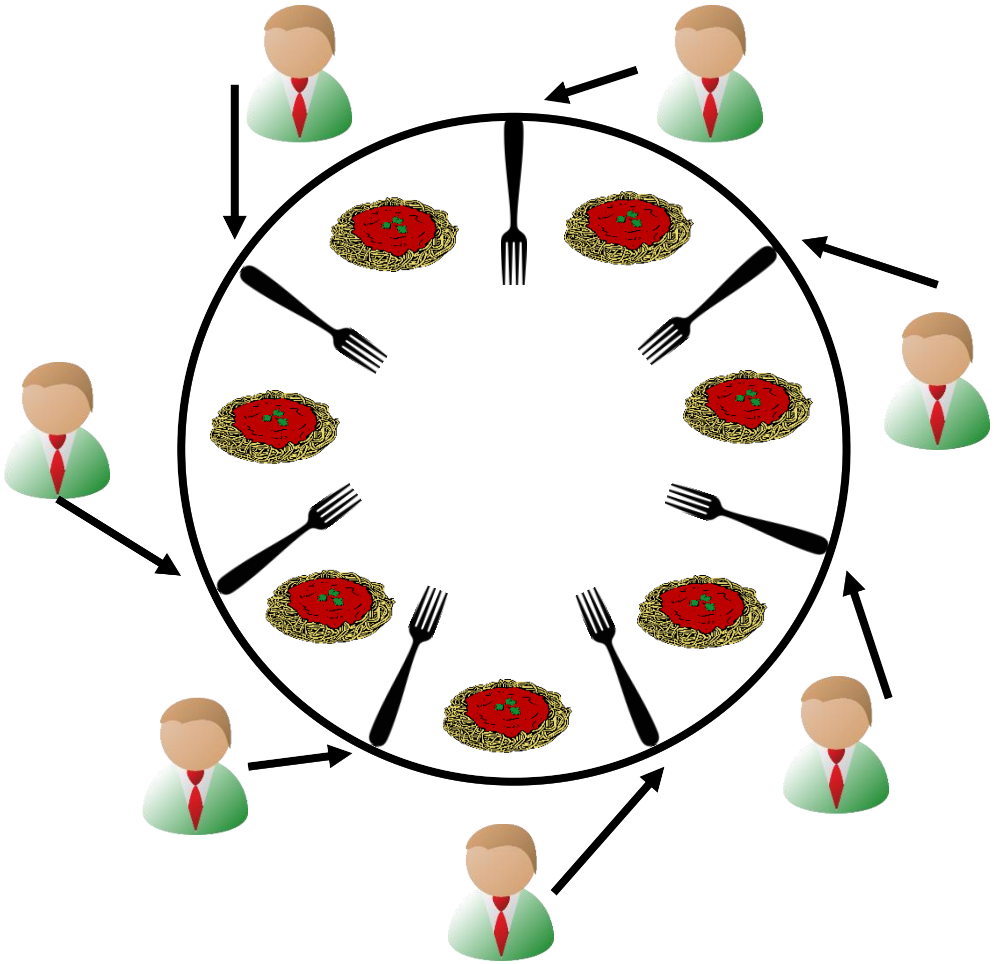
The forks are locks



Now he can eat. Then he will let go of the forks (locks). This means someone else can take them.

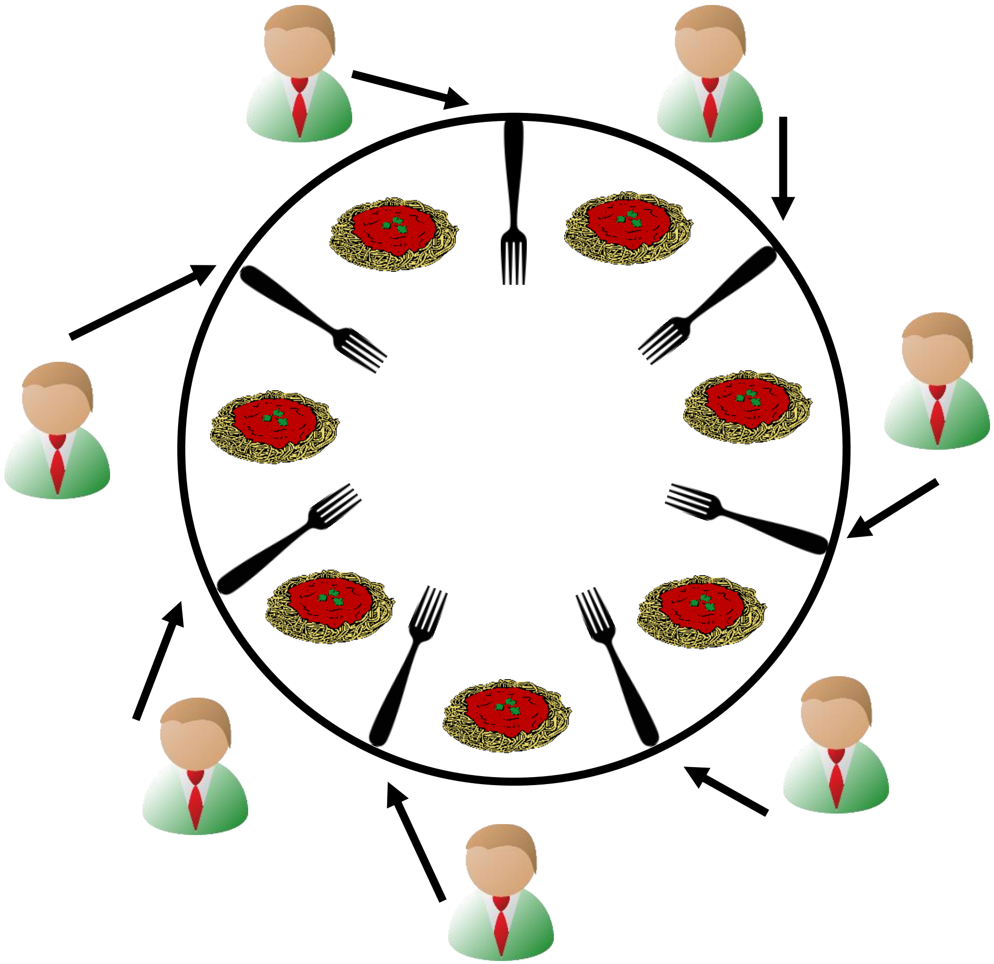
Dinning Philosophers Problem Dead-Lock

If they all get the right fork (lock) first, they can all get stuck (we have a dead-lock)



Dinning Philosophers Problem Dead-Lock

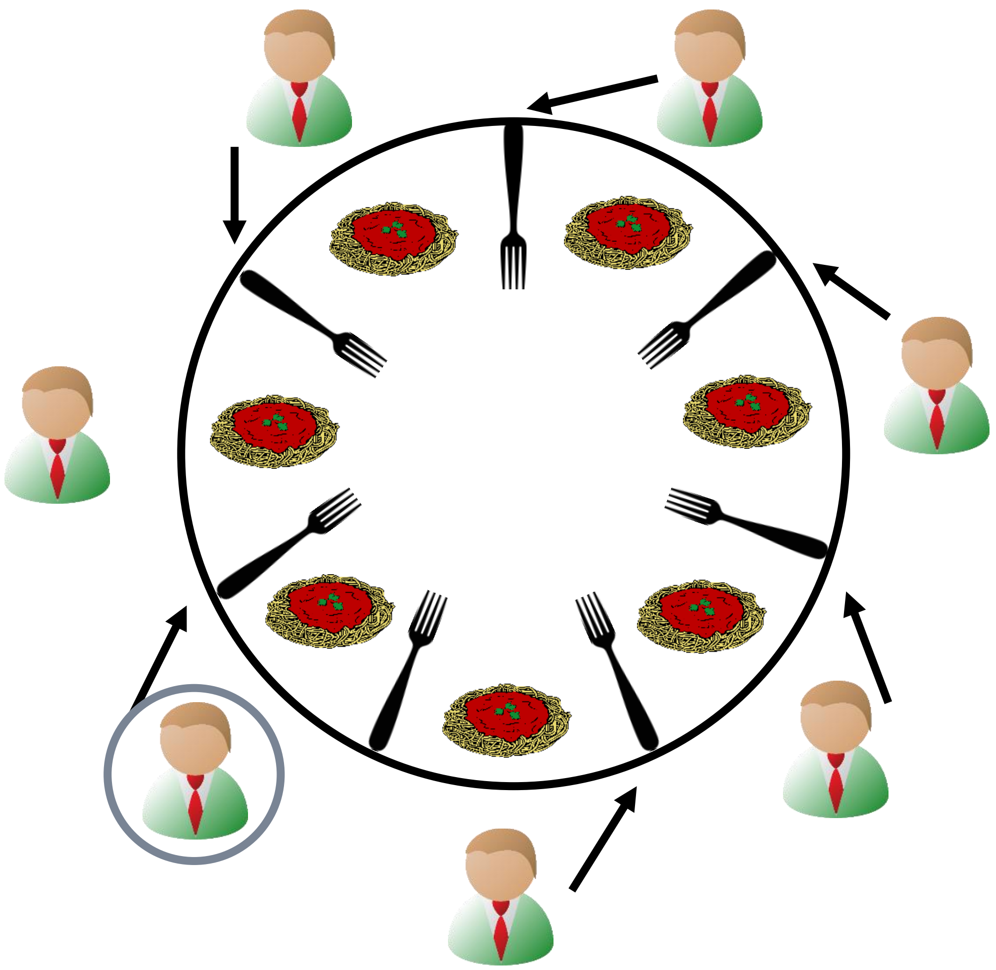
If they all get the left fork (lock) first, they can all get stuck (we have a dead-lock)



Dinning Philosophers Solution

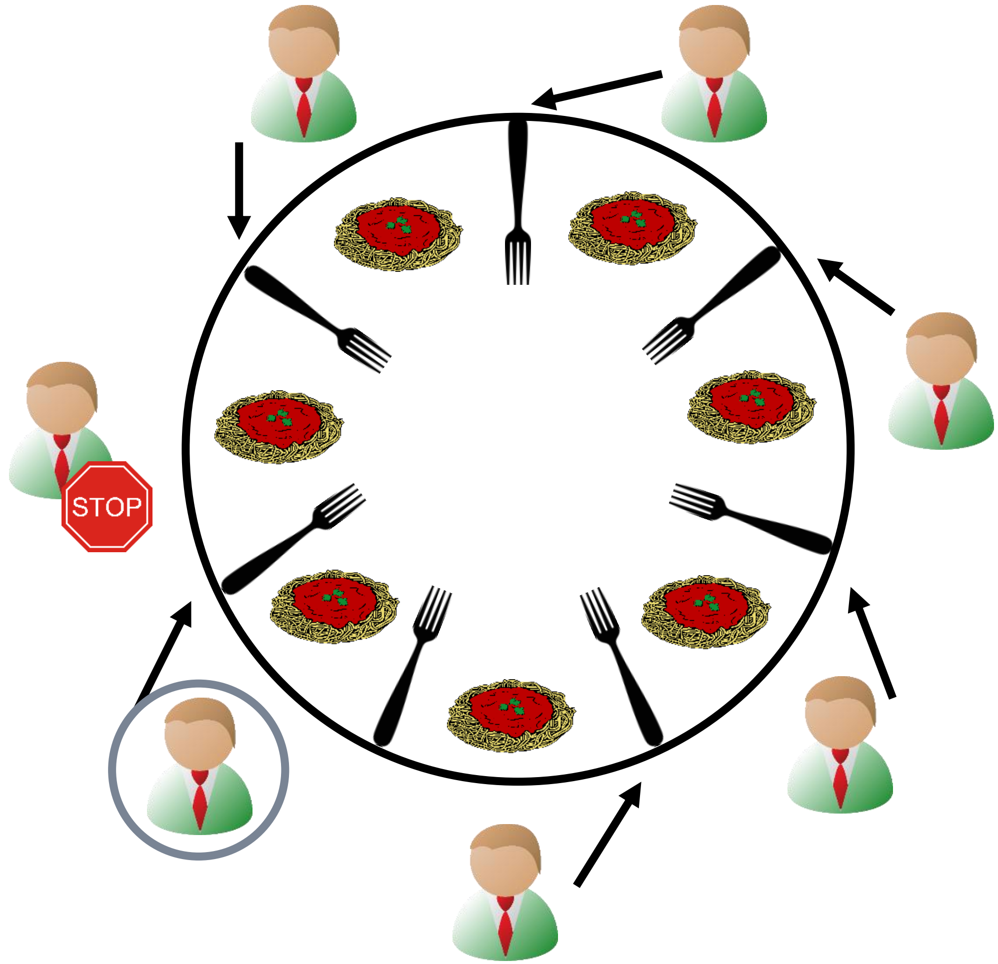
One of the philosophers takes the left fork (**lock**) **first** while the others take the right fork (**lock**) **first**.

Dinning Philosophers Solution

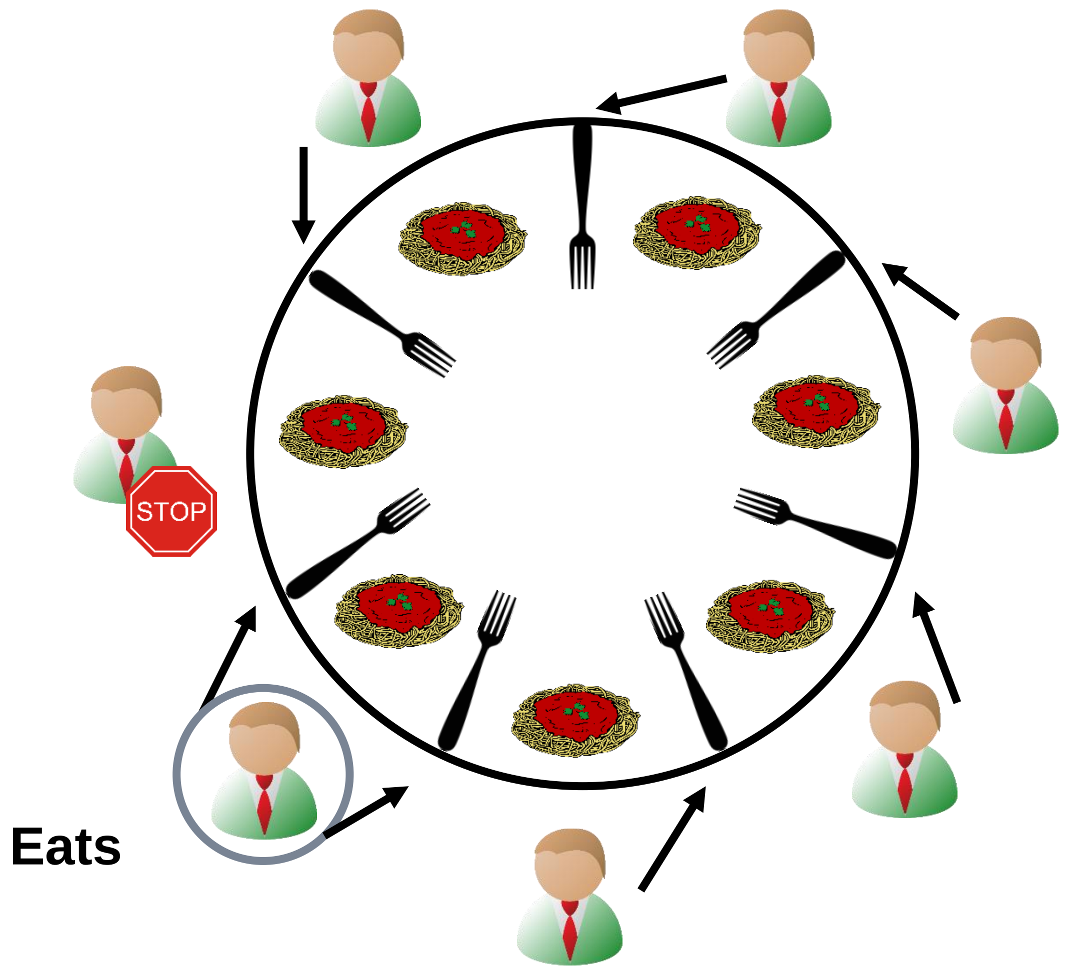


Dinning Philosophers Solution

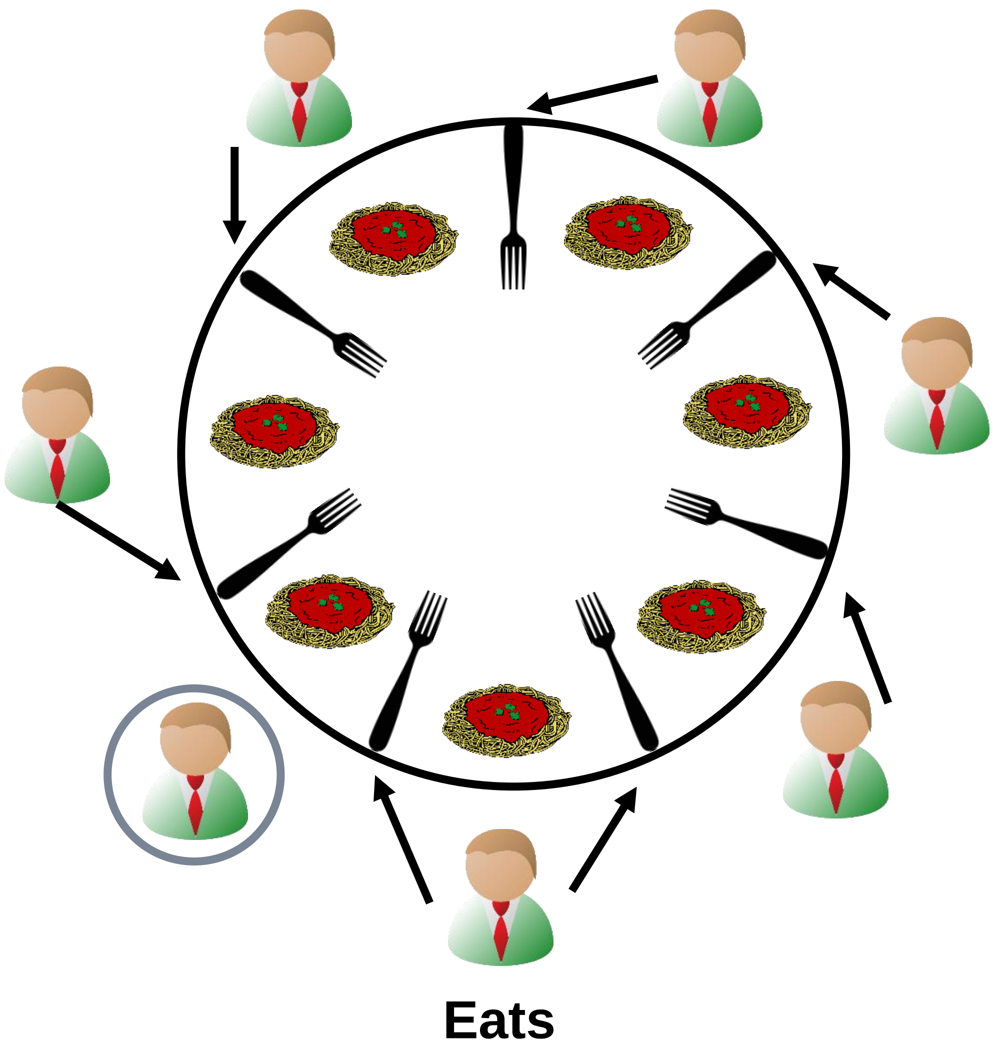
Can not take any fork (lock)



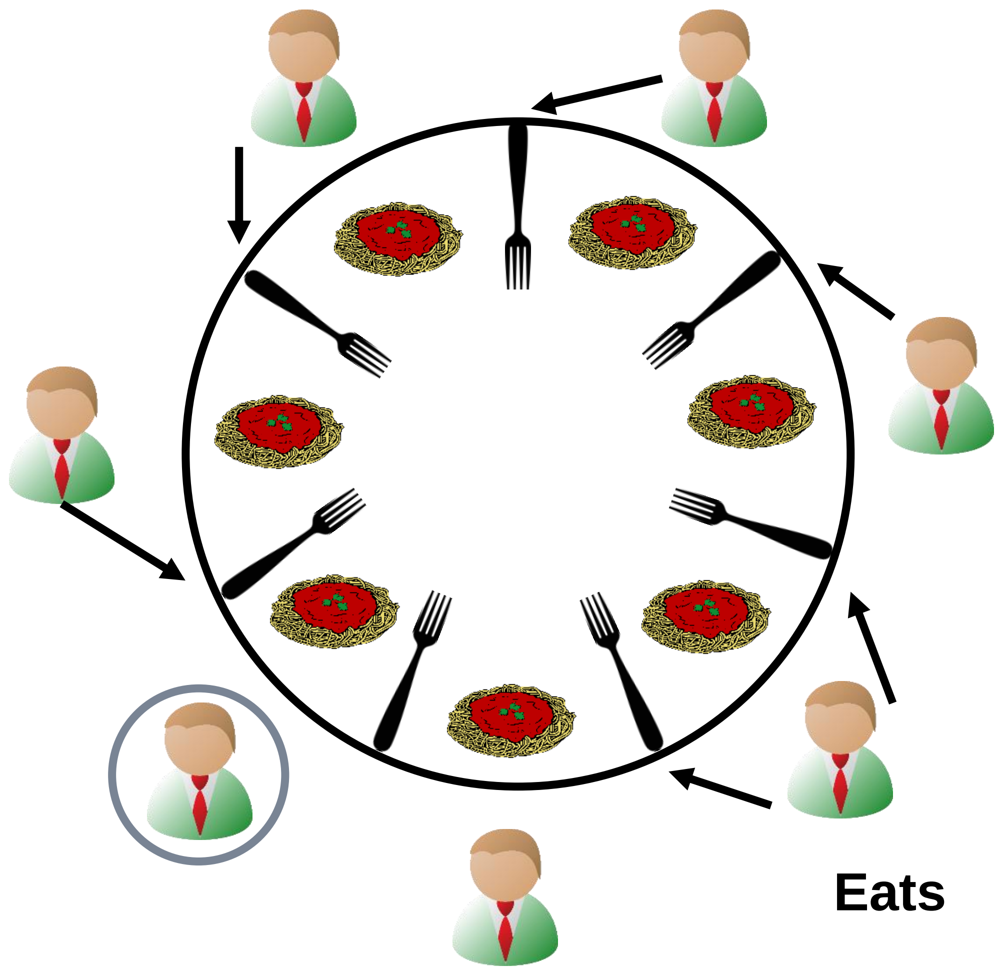
Dinning Philosophers Solution



Dinning Philosophers Solution



Dinning Philosophers Solution



Eats

Dinning Philosophers Problem in real life

**We have a server that runs a social network platform.
Each of our threads handles requests from a user.
It is possible for a group of users to be friends in a circle (like the philosophers sitting at the table).**

**The users can request sending a picture received from one of their friends to another.
In order to send the picture the thread needs to take the lock on the communication queue between A and B and between B and C.
If all people request a picture transfer at the same time the server will enter a dead-lock.**

HashMap

Keys	Values
A	11111
DD	2222
C	33333
S	4444

Hash(Keys)	Values
1	2222
2	11111
3	33333
4	
5	4444

Search **O(1)**
Insertion **O(1)**
Deletion **O(1)**

ConcurrentHashMap

- Uses locks inside the usual methods: **get**(), **put**()
- This means that when you use the HashMap in a multi-threaded program you don't need to remember to lock or what locks you used
- Has special methods such as **putIfAbsent**()

ConcurrentHashMap – under the hood

- If we use one lock for accessing the HashMap: 2 threads can not modify the hash map at the same time.
- If we use one lock for every element in the HashMap: We will need a lot of locks (difficulties with insertion/deletion)
- Store the data in groups and have a limited number of locks (parallelism level). Best results when number of locks is equal to number of threads.

Hash(Keys)	Values	
1	2222	Lock A
2	11111	
3	33333	Lock B
4		
5	4444	Lock C

Posibila solutie... pentru bariera

```
sem b = 0; e = 1;
int nb = 0;
process proc[k=1 to n]{
     # enter barrier
      P(e);
     nb=nb+1;
     if (nb = n){ V(e); V(b);}
     else { V(e); P(b); V(b);}
     P(e);
     nb=nb-1;
     V(e);
     # exit barrier
}
```

```
# if last process
to enter barrier
```

```
# if first process
to enter barrier
```

Un pic de practice: Cigarette smokers problem

- 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

Cigarette smokers problem

```
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 ???

Cigarette smokers problem - deadlock

OK!

P(match)

P(tobacco)

DEADLOCK!

P(paper)

P(match)

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

Practice

- Gandiți-vă la o soluție pentru evitarea deadlock-ului...

Cigarette smokers problem

```
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);}
    }
}
```

Cigarette smokers problem

```
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);
}
```

Cigarette smokers problem

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