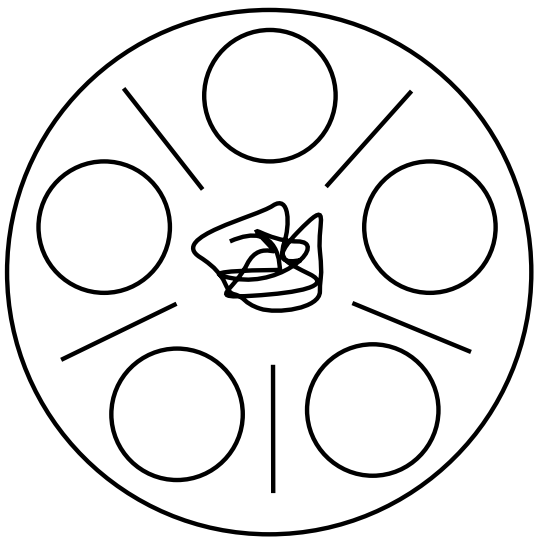
Basic Concurrency Resolution in Clock-free P Systems

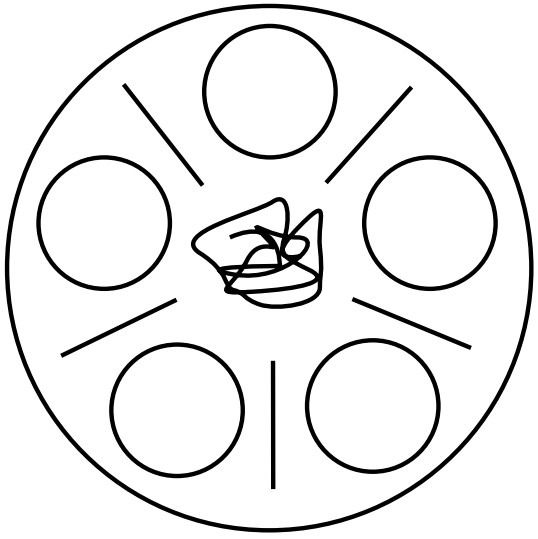


Sergiu Ivanov[1,2]

[1]Institute of Mathematics and Computer Science, Academy of Sciences of Moldova
[2]Technical University of Moldova

CMC12

Basic Concurrency Resolution in Clock-free P Systems



Sergiu Ivanov[1,2]

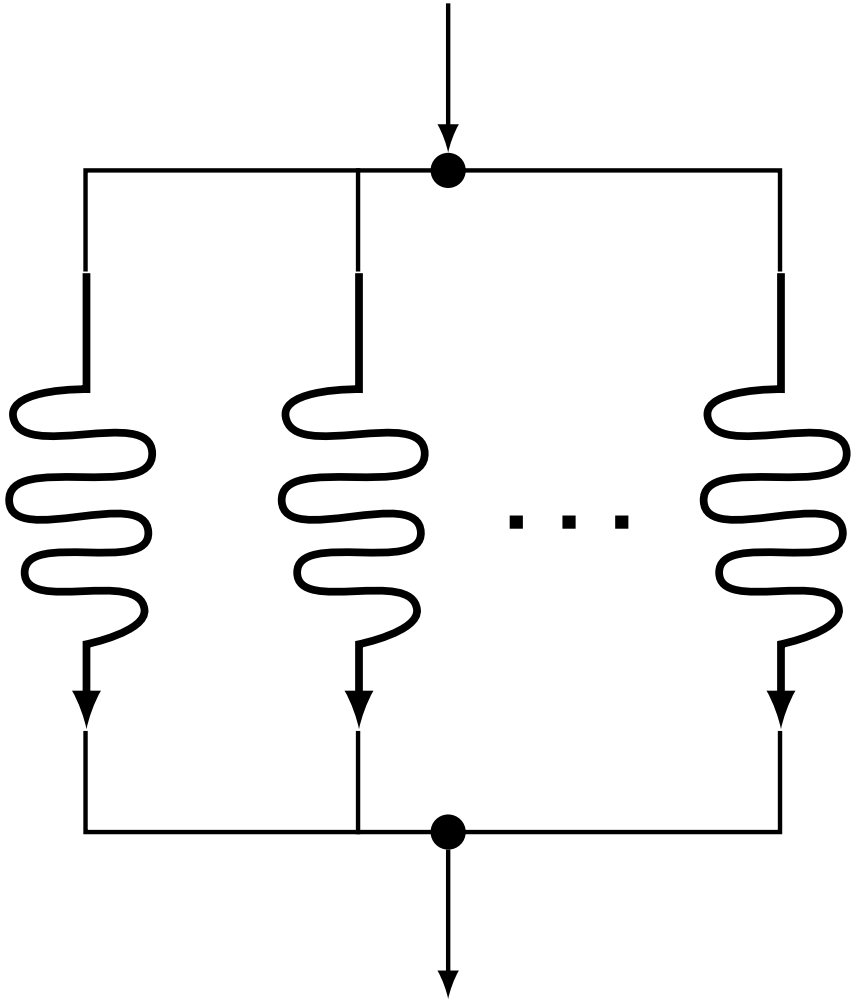
[1]Institute of Mathematics and Computer Science, Academy of Sciences of Moldova
[2]Technical University of Moldova

CMC12

Parallelism vs. Concurrency

Parallelism

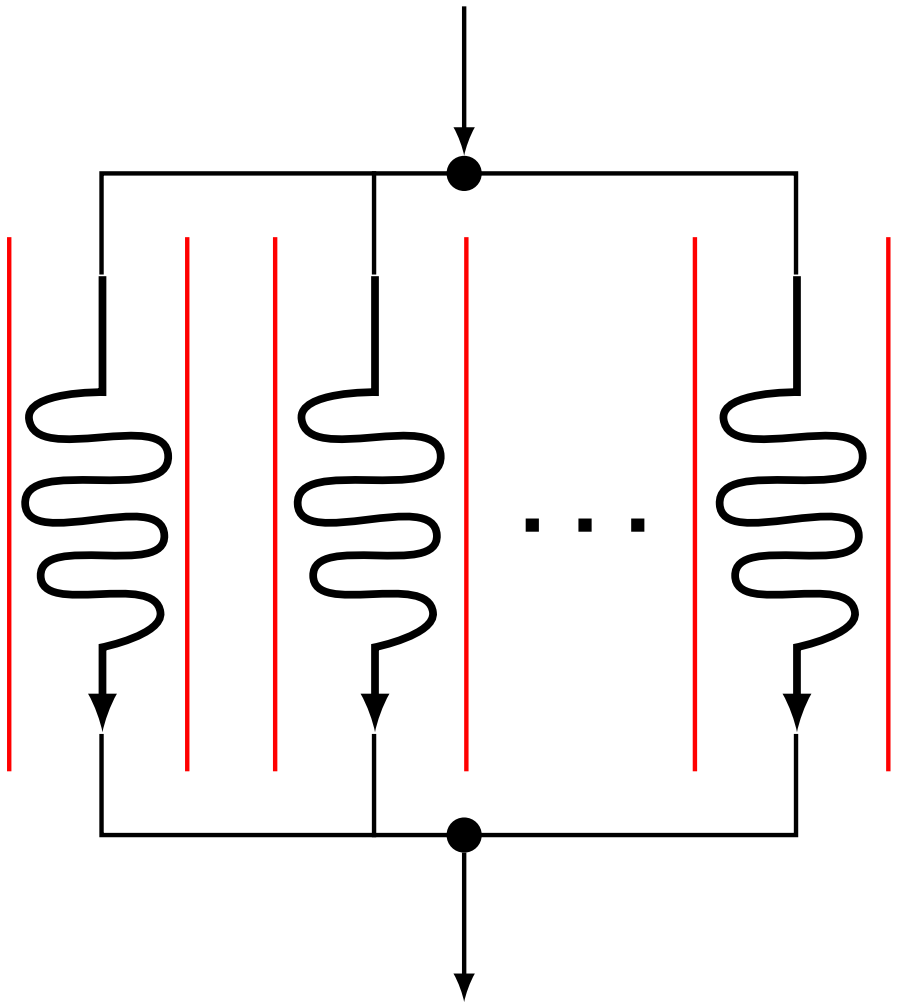
- practically valid speed-up (by $O(1)$)



Parallelism vs. Concurrency

Parallelism

- practically valid speed-up (by $O(1)$)
- far from universality



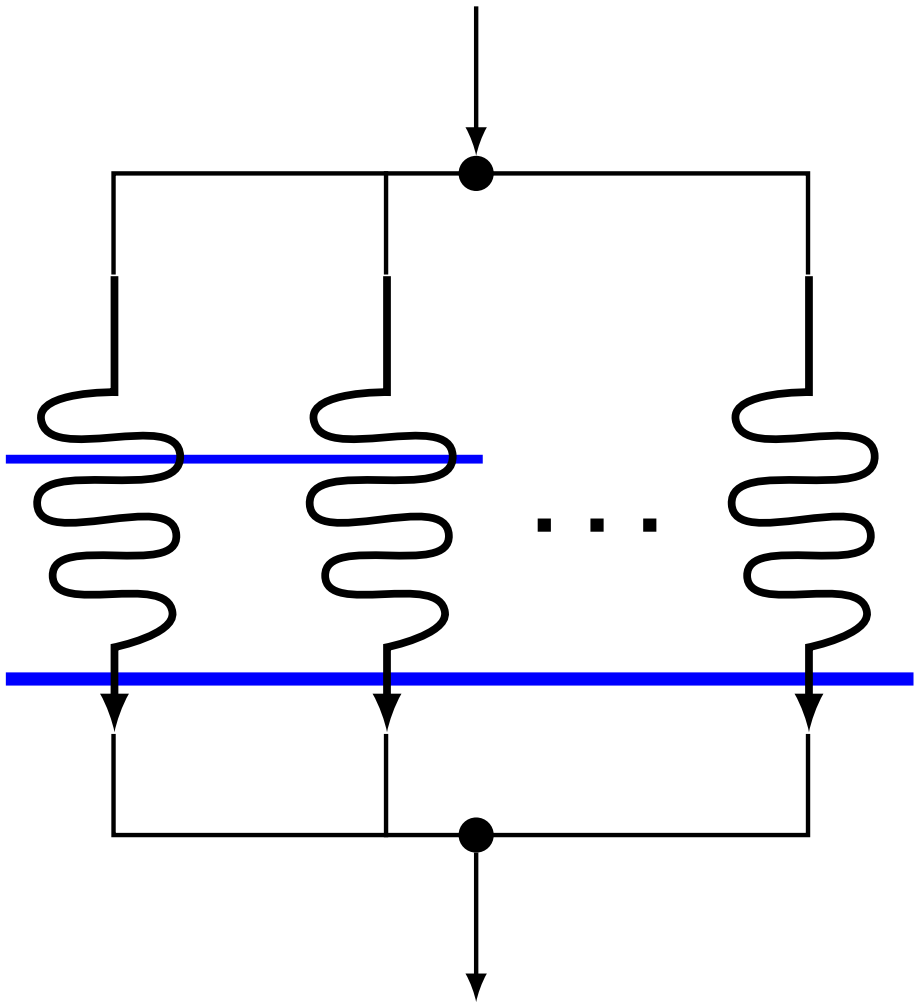
Parallelism vs. Concurrency

Parallelism

- practically valid speed-up (by $O(1)$)
- far from universality

Concurrency

- is about interactions



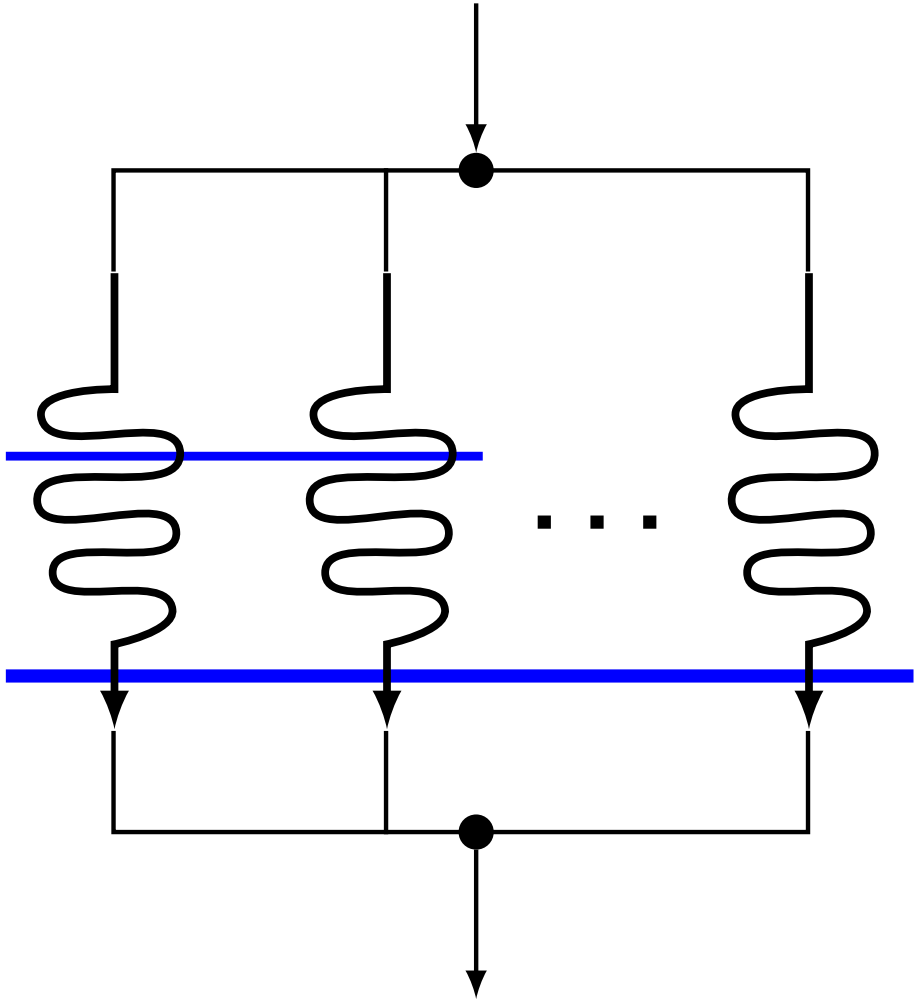
Parallelism vs. Concurrency

Parallelism

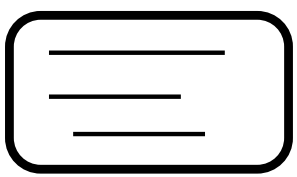
- practically valid speed-up (by $O(1)$)
- far from universality

Concurrency

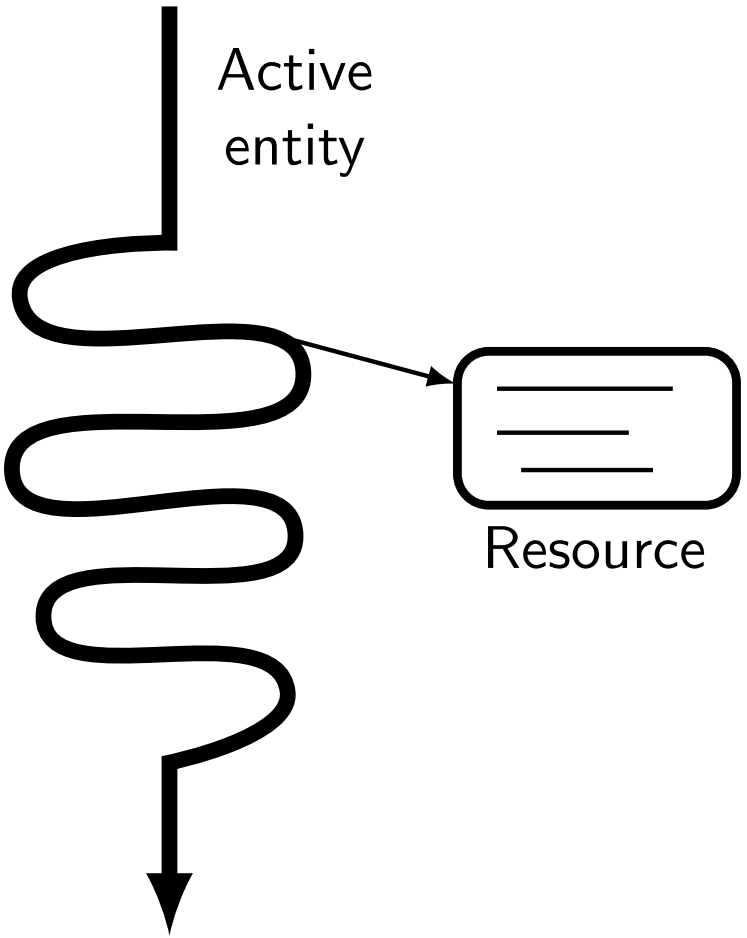
- is about interactions
- naturally arises in parallel processes



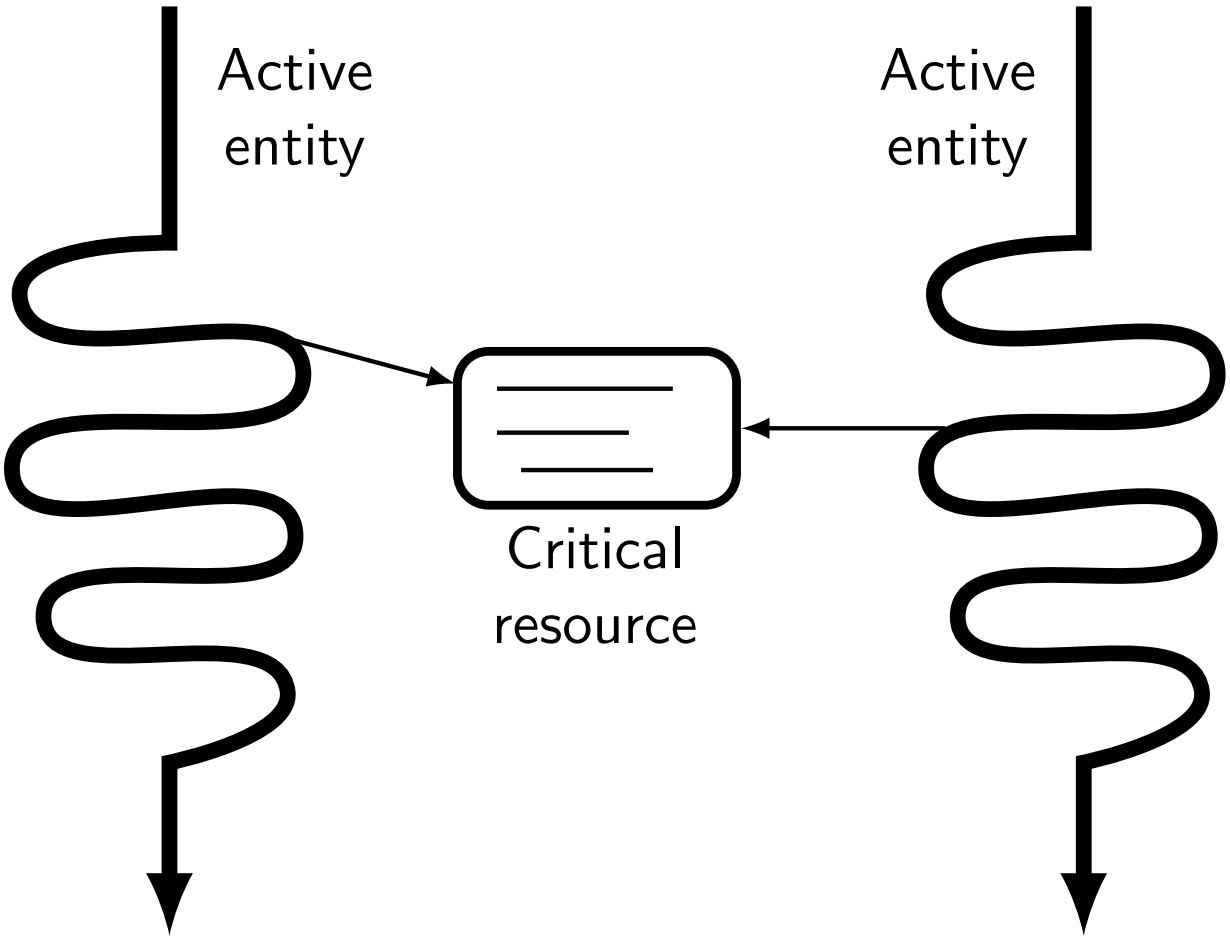
Concurrency: Basic Concepts



Concurrency: Basic Concepts

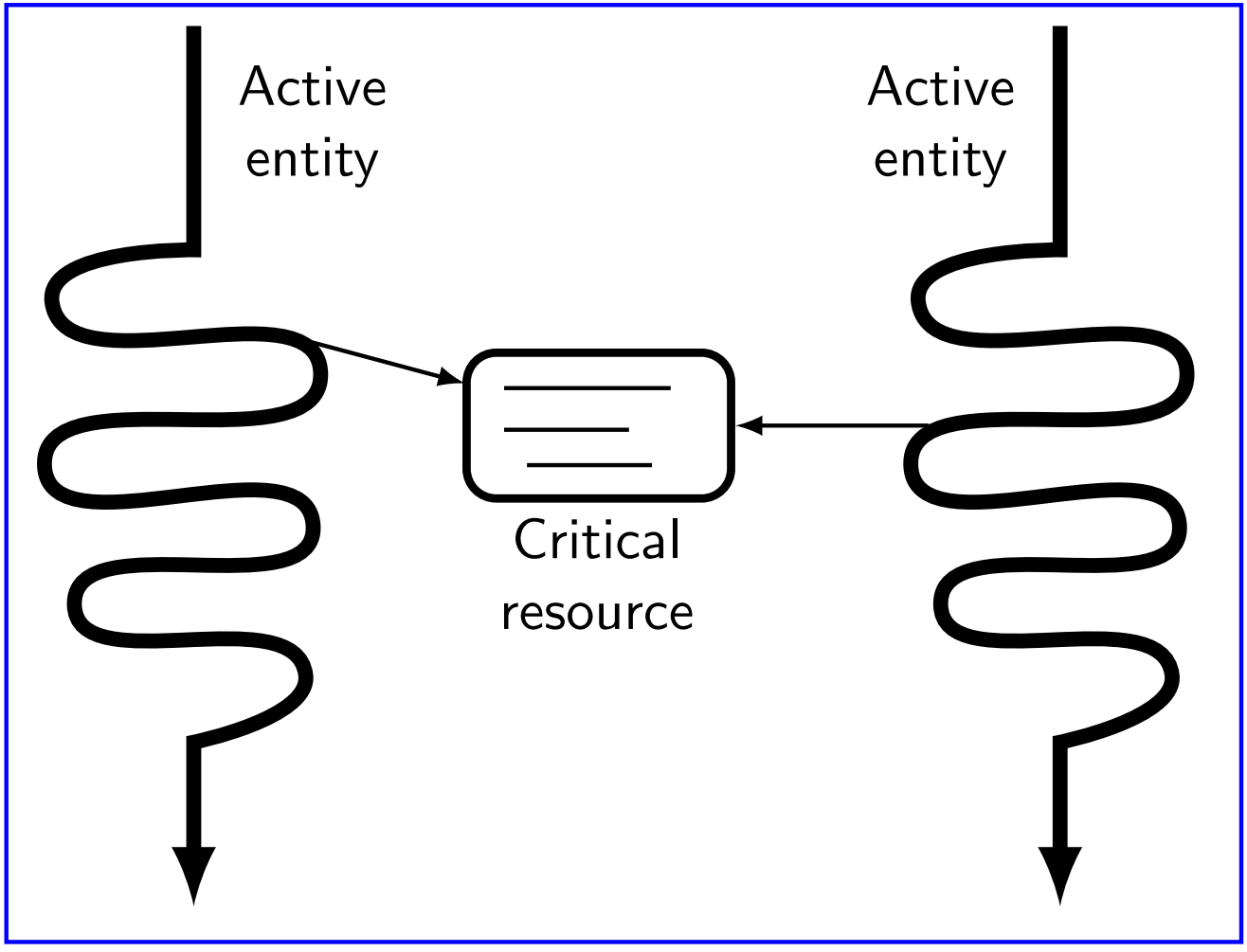


Concurrency: Basic Concepts



Concurrency: Basic Concepts

Concurrency problem



Concurrency: Semaphores and Locks

Semaphore

$$\texttt{counter} \leftarrow N$$

Concurrency: Semaphores and Locks

Semaphore

Operation P:
if $\texttt{counter} = 0$ **then**
  wait until $\texttt{counter} \neq 0$
$\texttt{counter} \leftarrow \texttt{counter} - 1$

$$\texttt{counter} \leftarrow N$$

Concurrency: Semaphores and Locks

Semaphore

Operation P:
if `counter` $= 0$ **then**
  wait until `counter` $\neq 0$
`counter` $\leftarrow$ `counter`-1

Operation V:
`counter` $\leftarrow$ `counter`$+1$

`counter` $\leftarrow N$

Concurrency: Semaphores and Locks

Semaphore

Operation P:
if `counter` $= 0$ **then**
  wait until `counter` $\neq 0$
`counter` $\leftarrow$ `counter`-1

wait

Operation V:
`counter` $\leftarrow$ `counter`$+1$

signal

`counter` $\leftarrow N$

Concurrency: Semaphores and Locks

Semaphore

Operation P:
if `counter` $= 0$ **then**
  wait until `counter` $\neq 0$
`counter` $\leftarrow$ `counter`-1

Operation V:
`counter` $\leftarrow$ `counter`$+1$

wait signal

`counter` $\leftarrow N$

Lock

- $N = 1$
  - lock $\equiv$ wait
  - unlock $\equiv$ signal

Concurrency: Semaphores and Locks

Semaphore

Operation P:
if `counter` $= 0$ **then**
  wait until `counter` $\neq 0$
`counter` $\leftarrow$ `counter`-1

wait

Operation V:
`counter` $\leftarrow$ `counter`$+1$

signal

`counter` $\leftarrow N$

Lock

- $N = 1$
  - lock $\equiv$ wait
  - unlock $\equiv$ signal

Atomic operations

Catalytic P Systems

$\Pi = (O, C, \mu, w_1, w_2, \ldots, w_m, R_1, R_2, \ldots, R_m, i_0)$, where

O is a finite set of objects,

C is a finite set of catalysts, $C \in O$,

μ is a hierarchical structure of m membranes, bijectively labeled by $1, \ldots, m$; the interior of each membrane defines a region; the environment is referred to as region 0,

w_i is the initial multiset in region $i, 1 \leq i \leq m$,

R_i is the set of rules of region $i, 1 \leq i \leq m$,

i_0 is the output region.

(the most boring slide)

Clock-free P Systems

$$
\begin{aligned}
m &= 1 \\
R_1 &= \{a \to b, b \to c, \\
&\quad a \to u, a \to x, xu \to x\}
\end{aligned}
$$

a a a

Clock-free P Systems

$$
\begin{aligned}
m &= 1 \\
R_1 &= \{a \to b, b \to c, \\
&\quad a \to u, a \to x, xu \to x\}
\end{aligned}
$$

a a a

- Each rule application lasts differently

Clock-free P Systems

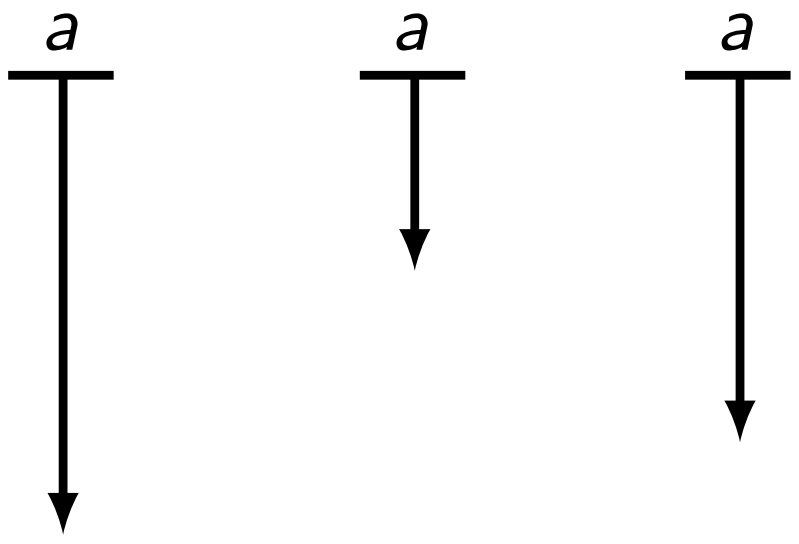
$$m = 1$$
$$R_1 = \{a \to b, b \to c, a \to u, a \to x, xu \to x\}$$

a a a

- Each rule application lasts differently
- LHS is consumed when application starts
- RHS is added when application ends

Clock-free P Systems

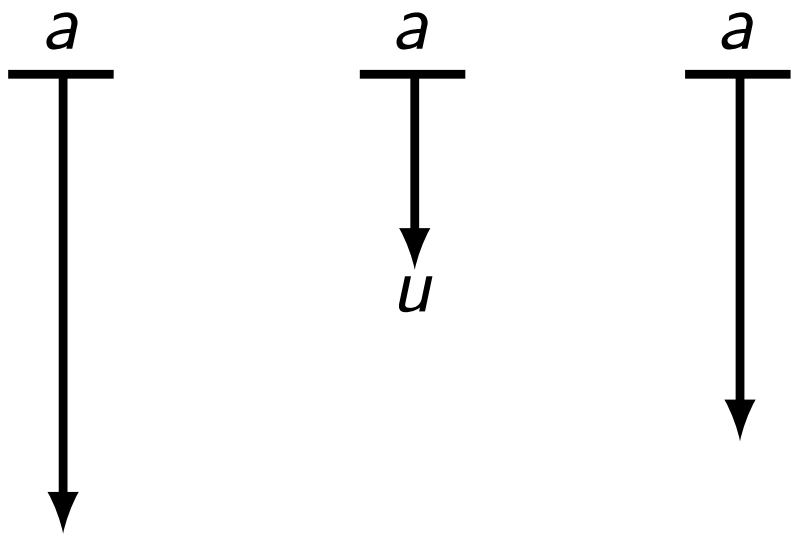
$$
\begin{aligned}
m &= 1 \\
R_1 &= \{a \to b, b \to c, \\
&\quad a \to u, a \to x, xu \to x\}
\end{aligned}
$$



- Each rule application lasts differently
- LHS is consumed when application starts
- RHS is added when application ends

Clock-free P Systems

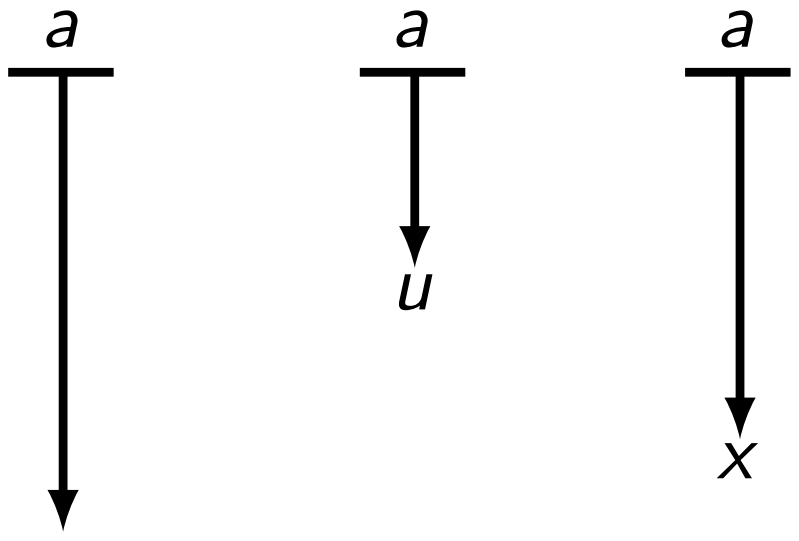
$$
\begin{aligned}
m &= 1 \\
R_1 &= \{a \to b, b \to c, \\
&\quad a \to u, a \to x, xu \to x\}
\end{aligned}
$$



- Each rule application lasts differently
- LHS is consumed when application starts
- RHS is added when application ends

Clock-free P Systems

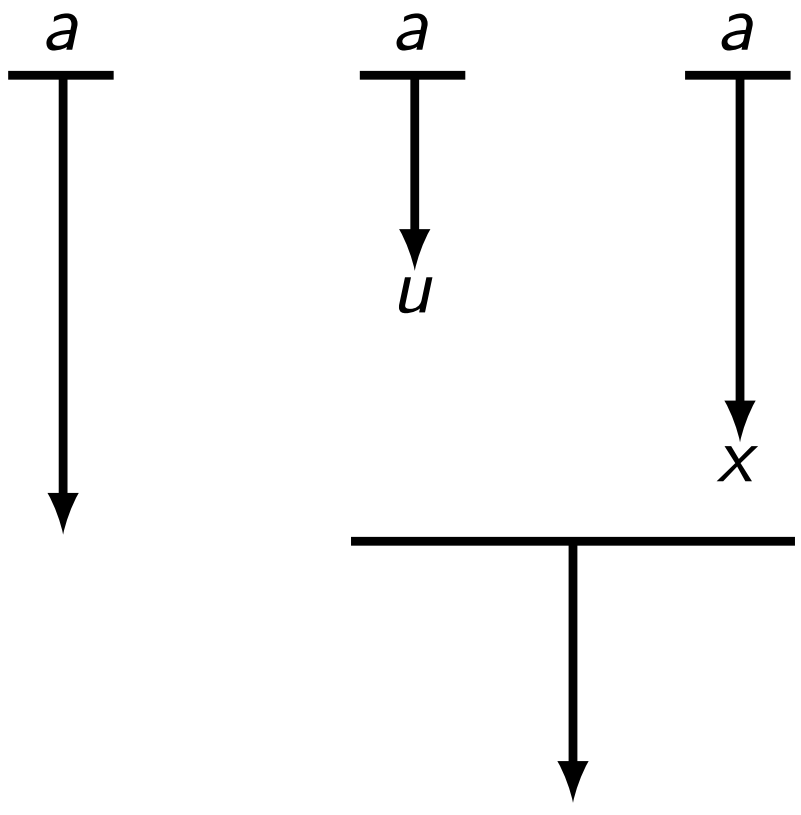
$$
\begin{aligned}
m &= 1 \\
R_1 &= \{a \to b, b \to c, \\
&\quad a \to u, a \to x, xu \to x\}
\end{aligned}
$$



- Each rule application lasts differently
- LHS is consumed when application starts
- RHS is added when application ends

Clock-free P Systems

$$
\begin{aligned}
m &= 1 \\
R_1 &= \{a \to b, b \to c, \\
&\quad a \to u, a \to x, xu \to x\}
\end{aligned}
$$

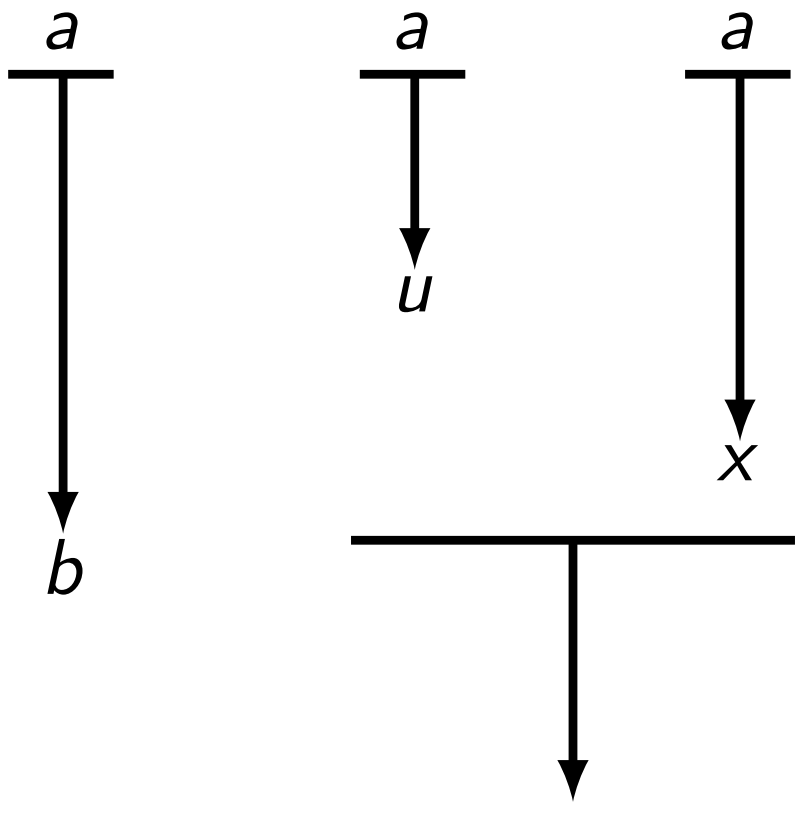


- Each rule application lasts differently
- LHS is consumed when application starts
- RHS is added when application ends

Clock-free P Systems

$$
\begin{aligned}
m &= 1 \\
R_1 &= \{a \to b, b \to c, \\
&\quad a \to u, a \to x, xu \to x\}
\end{aligned}
$$

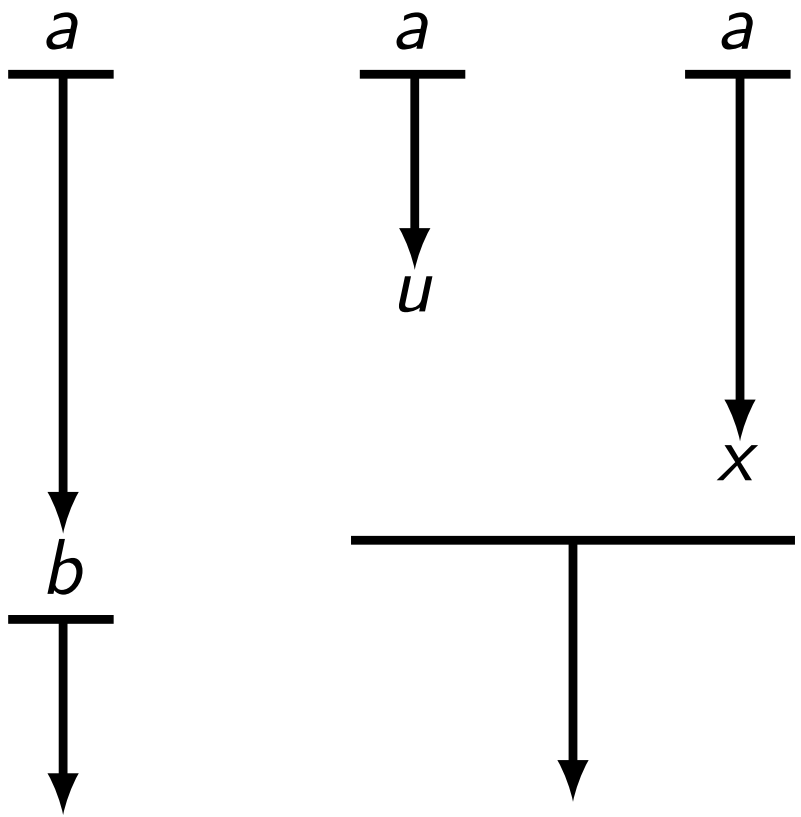
- Each rule application lasts differently
- LHS is consumed when application starts
- RHS is added when application ends



Clock-free P Systems

$$
\begin{aligned}
m &= 1 \\
R_1 &= \{a \to b, b \to c, \\
&\quad a \to u, a \to x, xu \to x\}
\end{aligned}
$$

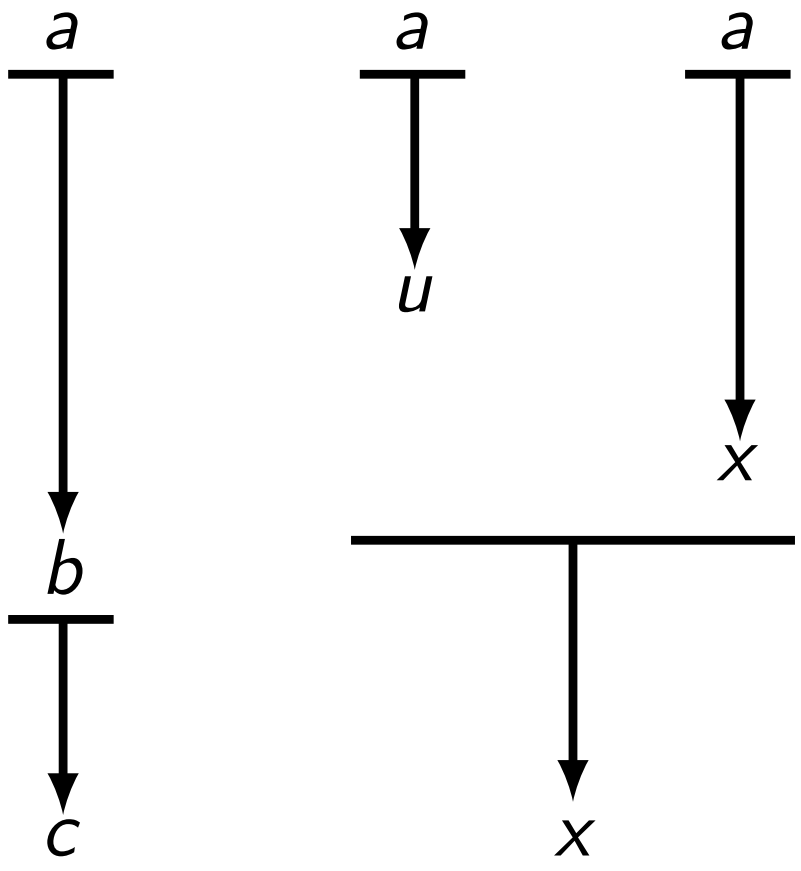
- Each rule application lasts differently
- LHS is consumed when application starts
- RHS is added when application ends



Clock-free P Systems

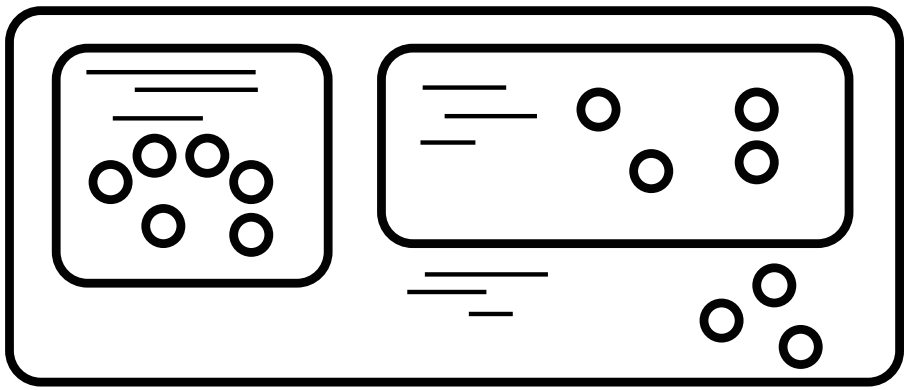
$$
\begin{aligned}
m &= 1 \\
R_1 &= \{a \to b, b \to c, \\
&\quad a \to u, a \to x, xu \to x\}
\end{aligned}
$$

- Each rule application lasts differently
- LHS is consumed when application starts
- RHS is added when application ends



Formal Strategies

Π solves L

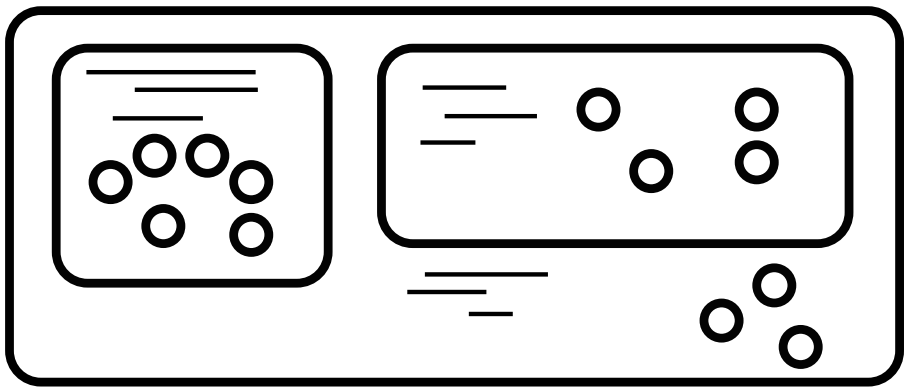
$\Pi =$ 

Formal Strategies

Consider L' similar to L

Π solves L

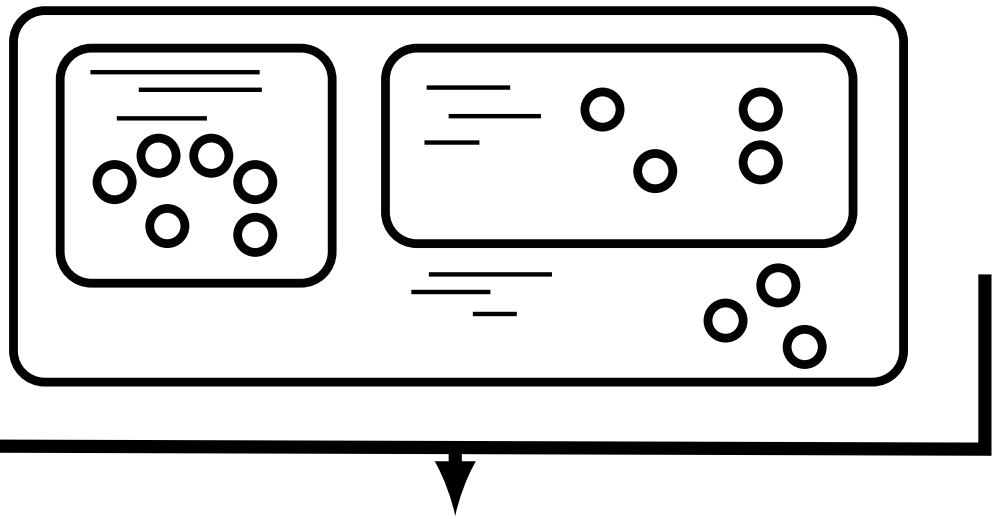
$\Pi =$



Formal Strategies

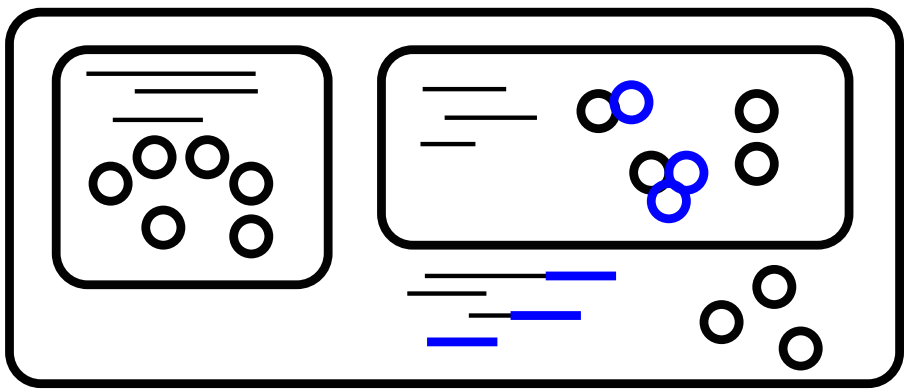
Consider L' similar to L

Π solves L $\Pi =$



Strategy

Π' solves L' $\Pi' =$

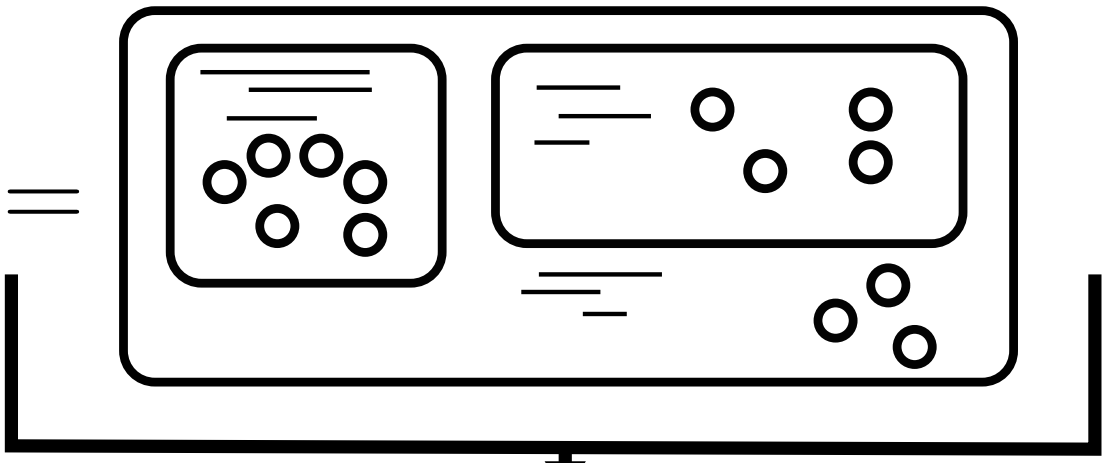


Formal Strategies

Consider L' similar to L

Π solves L

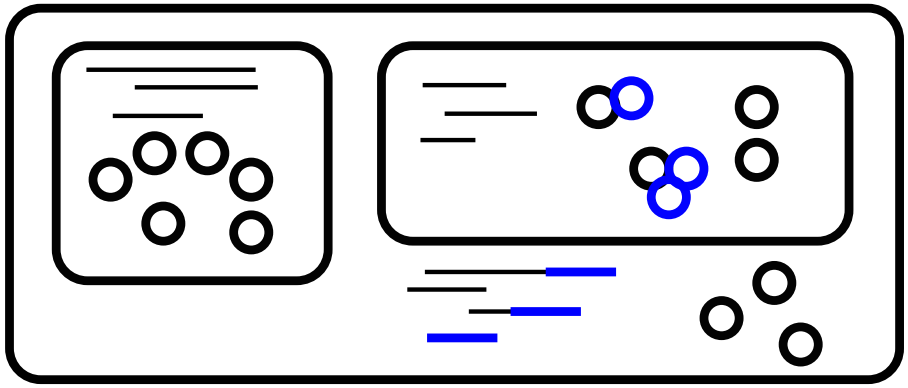
$\Pi =$



Strategy

Π' solves L'

$\Pi' =$



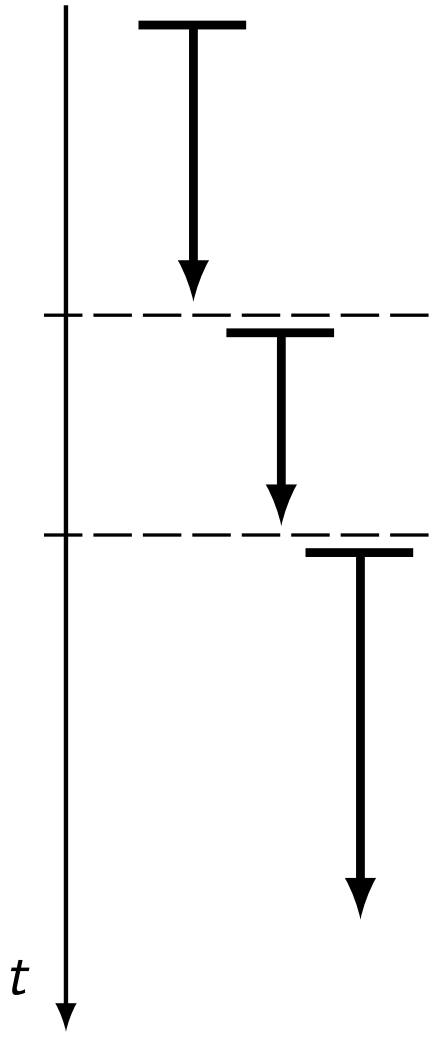
Strategy : $\mathcal{P} \rightarrow \mathcal{P}$,
$\mathcal{P}$ is the set of all P systems

Ways to Synchronise Clock-free Rules

$r : u \to \alpha$ applicable to w_0

- when all applications done?
- no other applications of r before done

Ways to Synchronise Clock-free Rules



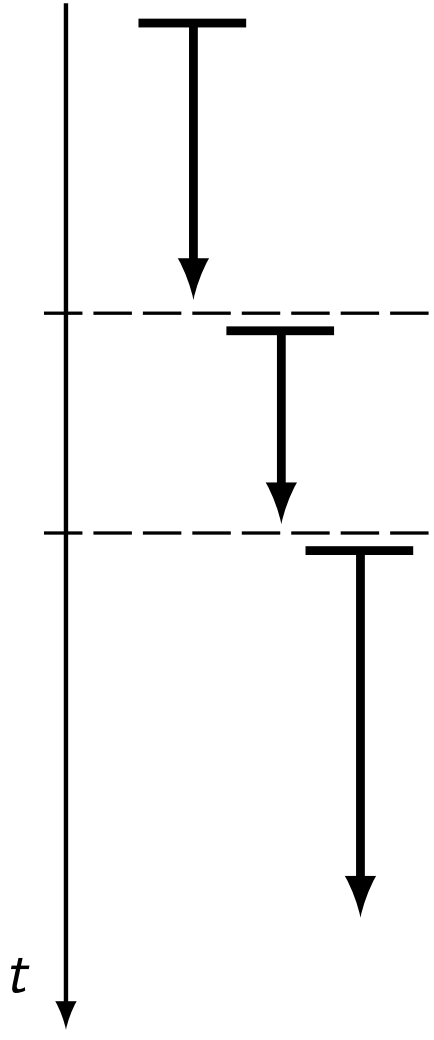
Sequential

one application at a time

$r : u \to \alpha$ applicable to w_0

- when all applications done?
- no other applications of r before done

Ways to Synchronise Clock-free Rules



Sequential

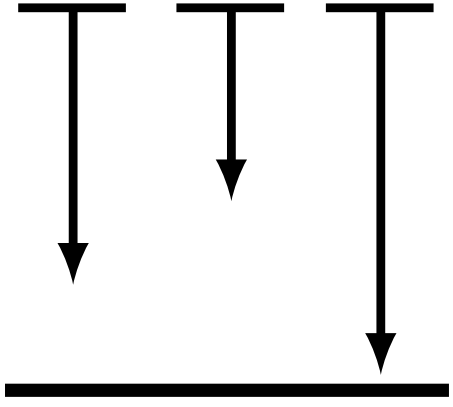
one application at a time

$r : u \to \alpha$ applicable to w_0

- when all applications done?
- no other applications of r before done

Barrier

wait for all to complete



Sequential Rule Synchronisation

Apply $a \rightarrow \alpha$ to a^n

Sequential Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{ca \to cf_1 f_2 \alpha|_p, p \to \lambda, f_1 \to \lambda|_a, f_2 \to f_3, f_3 \to p|_a, f_3 \to s|_{f_1}, f_1 \to \lambda|_{f_3}\}$

Sequential Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{ca \to cf_1 f_2 \alpha|_p, p \to \lambda, f_1 \to \lambda|_a, f_2 \to f_3, f_3 \to p|_a, f_3 \to s|_{f_1}, f_1 \to \lambda|_{f_3}\}$

1. Start with $w = a^n \cdot pc$ $a^n pc$

Sequential Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{ca \to cf_1 f_2 \alpha|_p, p \to \lambda, f_1 \to \lambda|_a, f_2 \to f_3, f_3 \to p|_a, f_3 \to s|_{f_1}, f_1 \to \lambda|_{f_3}\}$

1. Start with $w = a^n \cdot pc$ $a^n pc$
2. $ca \to cf_1 f_2 \alpha|_p$ and $p \to \lambda$; stop and check $\alpha cf_1 f_2 a^{n-1}$

Sequential Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{ca \to cf_1 f_2 \alpha|_p, p \to \lambda, f_1 \to \lambda|_a, f_2 \to f_3, f_3 \to p|_a, f_3 \to s|_{f_1}, f_1 \to \lambda|_{f_3}\}$

1. Start with $w = a^n \cdot pc$ $a^n pc$
2. $ca \to cf_1 f_2 \alpha|_p$ and $p \to \lambda$; stop and check $\alpha cf_1 f_2 a^{n-1}$
3. If $|w|_a \neq 0$, $f_1 \to \lambda|_a$, $f_2 \to f_3$ $\alpha cf_3 a^{n-1}$

Sequential Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{ca \to cf_1 f_2 \alpha|_p, p \to \lambda, f_1 \to \lambda|_a, f_2 \to f_3, f_3 \to p|_a, f_3 \to s|_{f_1}, f_1 \to \lambda|_{f_3}\}$

1. Start with $w = a^n \cdot pc$ $a^n pc$
2. $ca \to cf_1 f_2 \alpha|_p$ and $p \to \lambda$; stop and check $\alpha cf_1 f_2 a^{n-1}$
3. If $|w|_a \neq 0$, $f_1 \to \lambda|_a$, $f_2 \to f_3$ $\alpha cf_3 a^{n-1}$
4. $|w|_{f_1} = 0$ and $|w|_a \neq 0$, so $f_3 \to p|_a$; restart αcpa^{n-1}

Sequential Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{ca \to cf_1 f_2 \alpha|_p, p \to \lambda, f_1 \to \lambda|_a, f_2 \to f_3, f_3 \to p|_a, f_3 \to s|_{f_1}, f_1 \to \lambda|_{f_3}\}$

1. Start with $w = a^n \cdot pc$ $a^n pc$
2. $ca \to cf_1 f_2 \alpha|_p$ and $p \to \lambda$; stop and check $\alpha cf_1 f_2 a^{n-1}$
3. If $|w|_a \neq 0$, $f_1 \to \lambda|_a$, $f_2 \to f_3$ $\alpha cf_3 a^{n-1}$
4. $|w|_{f_1} = 0$ and $|w|_a \neq 0$, so $f_3 \to p|_a$; restart αcpa^{n-1}

$a^{n-1} pc \cdot \alpha$

Sequential Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{ca \to cf_1 f_2 \alpha|_p, p \to \lambda, f_1 \to \lambda|_a, f_2 \to f_3, f_3 \to p|_a, f_3 \to s|_{f_1}, f_1 \to \lambda|_{f_3}\}$

1. Start with $w = a^n \cdot pc$ $a^n pc$
2. $ca \to cf_1 f_2 \alpha|_p$ and $p \to \lambda$; stop and check $\alpha cf_1 f_2 a^{n-1}$
3. If $|w|_a = 0$, $f_2 \to f_3$, f_1 stays $\alpha cf_1 f_3$

Sequential Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{ca \to cf_1 f_2 \alpha|_p, p \to \lambda, f_1 \to \lambda|_a, f_2 \to f_3, f_3 \to p|_a, f_3 \to s|_{f_1}, f_1 \to \lambda|_{f_3}\}$

1. Start with $w = a^n \cdot pc$ $a^n pc$
2. $ca \to cf_1 f_2 \alpha|_p$ and $p \to \lambda$; stop and check $\alpha cf_1 f_2 a^{n-1}$
3. If $|w|_a = 0$, $f_2 \to f_3$, f_1 stays $\alpha cf_1 f_3$
4. $|w|_{f_1} \neq 0$ and $|w|_a = 0$, so $f_3 \to s|_{f_1}$ and $f_1 \to \lambda|_{f_3}$ αcs

Sequential Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{ca \to cf_1 f_2 \alpha|_p, p \to \lambda, f_1 \to \lambda|_a, f_2 \to f_3, f_3 \to p|_a, f_3 \to s|_{f_1}, f_1 \to \lambda|_{f_3}\}$

1. Start with $w = a^n \cdot pc$ $a^n pc$
2. $ca \to cf_1 f_2 \alpha|_p$ and $p \to \lambda$; stop and check $\alpha cf_1 f_2 a^{n-1}$
3. If $|w|_a = 0$, $f_2 \to f_3$, f_1 stays $\alpha cf_1 f_3$
4. $|w|_{f_1} \neq 0$ and $|w|_a = 0$, so $f_3 \to s|_{f_1}$ and $f_1 \to \lambda|_{f_3}$ αcs

- s signals completion
- $c \to \lambda|_s$, if necessary

Barrier Rule Synchronisation

Apply $a \to \alpha$ to a^n

Solution: $\{a \to c_2\alpha|_p, p \to \lambda, c_2c_1 \to c_3, c_3 \to \lambda, f_1 \to f_3, f_2 \to f_4|_{c_1}, f_3f_4 \to f_1f_2, f_2f_3 \to s\}$

Note the pairs of catalysts: $\{(c_2, c_3), (f_3, f_1), (f_2, s)\}$

Barrier Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{a \to c_2\alpha|_p, p \to \lambda, c_2c_1 \to c_3, c_3 \to \lambda, f_1 \to f_3, f_2 \to f_4|_{c_1}, f_3f_4 \to f_1f_2, f_2f_3 \to s\}$

Note the pairs of catalysts: $\{(c_2, c_3), (f_3, f_1), (f_2, s)\}$

Once the applications have started, all a are consumed

Barrier Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{a \to c_2\alpha|_p, p \to \lambda, c_2c_1 \to c_3, c_3 \to \lambda, f_1 \to f_3, f_2 \to f_4|_{c_1}, f_3f_4 \to f_1f_2, f_2f_3 \to s\}$

1. Start with $w = a^n \cdot c_1^n p f_1 f_2$ $a^n c_1^n p \cdot f_1 f_2$

Barrier Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{a \to c_2\alpha|_p, p \to \lambda, c_2c_1 \to c_3, c_3 \to \lambda, f_1 \to f_3, f_2 \to f_4|_{c_1}, f_3f_4 \to f_1f_2, f_2f_3 \to s\}$

1. Start with $w = a^n \cdot c_1^n p f_1 f_2$ $a^n c_1^n p \cdot f_1 f_2$
2. $a \to c_2\alpha|_p$ and $p \to \lambda$; all applications started $c_1^n \cdot f_1 f_2$

Barrier Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{a \to c_2\alpha|_p, p \to \lambda, c_2c_1 \to c_3, c_3 \to \lambda, f_1 \to f_3, f_2 \to f_4|_{c_1}, f_3f_4 \to f_1f_2, f_2f_3 \to s\}$

1. Start with $w = a^n \cdot c_1^n p f_1 f_2$ $a^n c_1^n p \cdot f_1 f_2$
2. $a \to c_2\alpha|_p$ and $p \to \lambda$; all applications started $c_1^n \cdot f_1 f_2$
3. Some application(s) complete(s), $c_2c_1 \to c_3$; decrement $\alpha^m c_3^m c_1^{n-m} \cdot f_1 f_2$

Barrier Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{a \to c_2\alpha|_p, p \to \lambda, c_2c_1 \to c_3, c_3 \to \lambda, f_1 \to f_3, f_2 \to f_4|_{c_1}, f_3f_4 \to f_1f_2, f_2f_3 \to s\}$

1. Start with $w = a^n \cdot c_1^n p f_1 f_2$ $a^n c_1^n p \cdot f_1 f_2$
2. $a \to c_2\alpha|_p$ and $p \to \lambda$; all applications started $c_1^n \cdot f_1 f_2$
3. Some application(s) complete(s), $c_2c_1 \to c_3$; decrement $\alpha^m c_3^m c_1^{n-m} \cdot f_1 f_2$
4. $c_3 \to \lambda$ $\alpha^m c_1^{n-m} \cdot f_1 f_2$

Barrier Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{a \to c_2\alpha|_p, p \to \lambda, c_2c_1 \to c_3, c_3 \to \lambda, f_1 \to f_3, f_2 \to f_4|_{c_1}, f_3f_4 \to f_1f_2, f_2f_3 \to s\}$

1. Start with $w = a^n \cdot c_1^n p f_1 f_2$
2. $a \to c_2\alpha|_p$ and $p \to \lambda$; all applications started
3. Some application(s) complete(s), $c_2c_1 \to c_3$; decrement
4. $c_3 \to \lambda$

f_1 and f_2 work in parallel, independently

Barrier Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{a \to c_2\alpha|_p, p \to \lambda, c_2c_1 \to c_3, c_3 \to \lambda, f_1 \to f_3, f_2 \to f_4|_{c_1}, f_3f_4 \to f_1f_2, f_2f_3 \to s\}$

1. Start with $w = a^n \cdot c_1^n p f_1 f_2$
2. $a \to c_2\alpha|_p$ and $p \to \lambda$; all applications started
3. Some application(s) complete(s), $c_2c_1 \to c_3$; decrement
4. $c_3 \to \lambda$

f_1 and f_2 work in parallel, independently

1. If $|w|_{c_1} \neq 0$, $f_1 \to f_3$ and $f_2 \to f_4|_{c_1}$

Barrier Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{a \to c_2\alpha|_p, p \to \lambda, c_2c_1 \to c_3, c_3 \to \lambda, f_1 \to f_3, f_2 \to f_4|_{c_1}, f_3f_4 \to f_1f_2, f_2f_3 \to s\}$

1. Start with $w = a^n \cdot c_1^n p f_1 f_2$
2. $a \to c_2\alpha|_p$ and $p \to \lambda$; all applications started
3. Some application(s) complete(s), $c_2c_1 \to c_3$; decrement
4. $c_3 \to \lambda$

f_1 and f_2 work in parallel, independently

1. If $|w|_{c_1} \neq 0$, $f_1 \to f_3$ and $f_2 \to f_4|_{c_1}$
2. Eventually, $f_3f_4 \to f_1f_2$; restart of the checking "thread"

Barrier Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{a \to c_2\alpha|_p, p \to \lambda, c_2c_1 \to c_3, c_3 \to \lambda, f_1 \to f_3, f_2 \to f_4|_{c_1}, f_3f_4 \to f_1f_2, f_2f_3 \to s\}$

1. Start with $w = a^n \cdot c_1^n p f_1 f_2$
2. $a \to c_2\alpha|_p$ and $p \to \lambda$; all applications started
3. Some application(s) complete(s), $c_2c_1 \to c_3$; decrement
4. $c_3 \to \lambda$

f_1 and f_2 work in parallel, independently

1. If $|w|_{c_1} = 0$, $f_1 \to f_3$ only;
2. Eventually, $f_2f_3 \to s$

Barrier Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{a \to c_2\alpha|_p, p \to \lambda, c_2c_1 \to c_3, c_3 \to \lambda, f_1 \to f_3, f_2 \to f_4|_{c_1}, f_3f_4 \to f_1f_2, f_2f_3 \to s\}$

1. Start with $w = a^n \cdot c_1^n p f_1 f_2$
2. $a \to c_2\alpha|_p$ and $p \to \lambda$; all applications started
3. Some application(s) complete(s), $c_2c_1 \to c_3$; decrement
4. $c_3 \to \lambda$

f_1 and f_2 work in parallel, independently

1. If $|w|_{c_1} = 0$, $f_1 \to f_3$ only;
2. Eventually, $f_2f_3 \to s$

The dependency is introduced by promoters $\Longrightarrow$ No mutual blocking

Barrier Rule Synchronisation

Apply $a \to \alpha$ to a^n Solution: $\{a \to c_2\alpha|_p, p \to \lambda, c_2c_1 \to c_3, c_3 \to \lambda, f_1 \to f_3, f_2 \to f_4|_{c_1}, f_3f_4 \to f_1f_2, f_2f_3 \to s\}$

1. Start with $w = a^n \cdot c_1^n p f_1 f_2$
2. $a \to c_2\alpha|_p$ and $p \to \lambda$; all applications started
3. Some application(s) complete(s), $c_2c_1 \to c_3$; decrement
4. $c_3 \to \lambda$

f_1 and f_2 work in parallel, independently

1. If $|w|_{c_1} = 0$, $f_1 \to f_3$ only;
2. Eventually, $f_2f_3 \to s$

(the most horrible slide)

The dependency is introduced by promoters $\Longrightarrow$ No mutual blocking

Rule Synchronisation vs. Process Synchronisation

Traits of rules

- atomicity
- logical independence

(even in clock-free P systems)

Rule Synchronisation vs. Process Synchronisation

Traits of rules

- atomicity
- logical independence

(even in clock-free P systems)

$\Rightarrow$

Rules are easy synchronisation targets

Rule Synchronisation vs. Process Synchronisation

Traits of rules

- atomicity
- logical independence

(even in clock-free P systems)

$\Rightarrow$

Rules are easy synchronisation targets

Process = Membrane

- skin contains m elementary membranes (processes)
- skin contains shared data
- each process has local data

Rule Synchronisation vs. Process Synchronisation

Traits of rules

- atomicity
- logical independence

(even in clock-free P systems)

$\Rightarrow$

Rules are easy synchronisation targets

Process = Membrane

- skin contains m elementary membranes (processes)
- skin contains shared data
- each process has local data

Real world modelling possible

The Lock

elementary membrane

$w_0 = \alpha \cdot \$$ for process i, $1 \leq i \leq m$

$$R_0 \leftarrow R_0 \cup \{ r_i\$ \rightarrow r_i'\$_i, r_i' \rightarrow (s_i, i), \$_i \rightarrow \lambda \}$$

symbols and rules in skin

The Lock

$$w_0 = \alpha \cdot \$ \qquad \text{for process } i,\ 1 \le i \le m$$

$$R_0 \leftarrow R_0 \cup \{ r_i\$ \rightarrow r_i'\$_i,\ r_i' \rightarrow (s_i, i),\ \$_i \rightarrow \lambda \}$$

put s_i into membrane i

The Lock

$$w_0 = \alpha \cdot \$ \qquad \text{for process } i,\ 1 \le i \le m$$

$$R_0 \leftarrow R_0 \cup \{ r_i \$ \rightarrow r_i' \$_i,\ r_i' \rightarrow (s_i, i),\ \$_i \rightarrow \lambda \}$$

the lock symbol

The Lock

$$w_0 = \alpha \cdot \$ \qquad \text{for process } i,\ 1 \leq i \leq m$$

$$R_0 \leftarrow R_0 \cup \{ r_i\$ \rightarrow r_i'\$_i,\ r_i' \rightarrow (s_i, i),\ \$_i \rightarrow \lambda \}$$

lock

1. Process i ejects r_i

The Lock

$$w_0 = \alpha \cdot \$ \qquad \text{for process } i,\ 1 \leq i \leq m$$

$$R_0 \leftarrow R_0 \cup \{ r_i\$ \rightarrow r_i'\$_i,\ r_i' \rightarrow (s_i, i),\ \$_i \rightarrow \lambda \}$$

lock

1. Process i ejects r_i
2. r_i blocks \$ by $r_i\$ \rightarrow r_i'\$_i$

The Lock

$$w_0 = \alpha \cdot \$ \qquad \text{for process } i,\ 1 \leq i \leq m$$

$$R_0 \leftarrow R_0 \cup \{ r_i\$ \rightarrow r_i'\$_i,\ r_i' \rightarrow (s_i, i),\ \$_i \rightarrow \lambda \}$$

lock

1. Process i ejects r_i
2. r_i **blocks** \$ by $r_i\$ \rightarrow r_i'\$_i$
3. r_i' **notifies** the process i by $r_i' \rightarrow (s_i, i)$

The Lock

$$w_0 = \alpha \cdot \$ \qquad \text{for process } i,\ 1 \leq i \leq m$$

$$R_0 \leftarrow R_0 \cup \{ r_i\$ \rightarrow r_i'\$_i,\ r_i' \rightarrow (s_i, i),\ \$_i \rightarrow \lambda \}$$

lock

1. Process i ejects r_i
2. r_i **blocks** \$ by $r_i\$ \rightarrow r_i'\$_i$
3. r_i' **notifies** the process i by $r_i' \rightarrow (s_i, i)$

   $\$_i$ is erased

The Lock

$$w_0 = \alpha \cdot \$ \qquad \text{for process } i,\ 1 \leq i \leq m$$

$$R_0 \leftarrow R_0 \cup \{ r_i\$ \rightarrow r_i'\$_i,\ r_i' \rightarrow (s_i, i),\ \$_i \rightarrow \lambda \}$$

lock

1. Process i ejects r_i
2. r_i blocks \$ by $r_i\$ \rightarrow r_i'\$_i$
3. r_i' notifies the process i by $r_i' \rightarrow (s_i, i)$

   $\$_i$ is erased

unlock

1. Process i ejects \$ into the skin

The Lock

$$w_0 = \alpha \cdot \$ \qquad \text{for process } i,\ 1 \leq i \leq m$$

$$R_0 \leftarrow R_0 \cup \{ r_i\$ \rightarrow r_i'\$_i,\ r_i' \rightarrow (s_i, i),\ \$_i \rightarrow \lambda \}$$

lock

1. Process i ejects r_i
2. r_i blocks \$ by $r_i\$ \rightarrow r_i'\$_i$
3. r_i' notifies the process i by $r_i' \rightarrow (s_i, i)$

   $\$_i$ is erased

unlock

1. Process i ejects \$ into the skin

Mutual exclusion is assured by mutual exclusion of rules

The Semaphore

$$\texttt{counter} \leftarrow N$$

The Semaphore

$$w_0 = \alpha \cdot \$^N \qquad \text{for process } i,\ 1 \leq i \leq m$$

$$R_0 \leftarrow R_0 \cup \{r_i\$ \rightarrow r_i'\$_i, r_i' \rightarrow (s_i, i), \$_i \rightarrow \lambda\}$$

$$\texttt{counter} \leftarrow N$$

The Semaphore

$$w_0 = \alpha \cdot \$^N \qquad \text{for process } i,\ 1 \leq i \leq m$$

$$R_0 \leftarrow R_0 \cup \{r_i\$ \rightarrow r_i'\$_i, r_i' \rightarrow (s_i, i), \$_i \rightarrow \lambda\}$$

$$\texttt{counter} \leftarrow N$$

The Semaphore

$$w_0 = \alpha \cdot \$^N \qquad \text{for process } i,\ 1 \leq i \leq m$$

$$R_0 \leftarrow R_0 \cup \{r_i\$ \rightarrow r_i'\$_i, r_i' \rightarrow (s_i, i), \$_i \rightarrow \lambda\}$$

The same rules

The Semaphore

$$w_0 = \alpha \cdot \$^N \qquad \text{for process } i,\ 1 \leq i \leq m$$

$$R_0 \leftarrow R_0 \cup \{r_i\$ \rightarrow r_i'\$_i,\ r_i' \rightarrow (s_i, i),\ \$_i \rightarrow \lambda\}$$

The same rules

lock

1. Process i ejects r_i
2. r_i blocks one \$ by $r_i\$_i \rightarrow r_i'\$_i$
3. r_i' notifies the process i by $r_i' \rightarrow (s_i, i)$

   $\$_i$ is erased

unlock

1. Process i ejects \$ into the skin

The Semaphore

$$w_0 = \alpha \cdot \$^N \qquad \text{for process } i,\ 1 \leq i \leq m$$

$$R_0 \leftarrow R_0 \cup \{r_i\$ \rightarrow r_i'\$_i,\ r_i' \rightarrow (s_i, i),\ \$_i \rightarrow \lambda\}$$

The same rules

lock

1. Process i ejects r_i
2. r_i blocks one \$ by $r_i\$_i \rightarrow r_i'\$_i$
3. r_i' notifies the process i by $r_i' \rightarrow (s_i, i)$

   $\$_i$ is erased

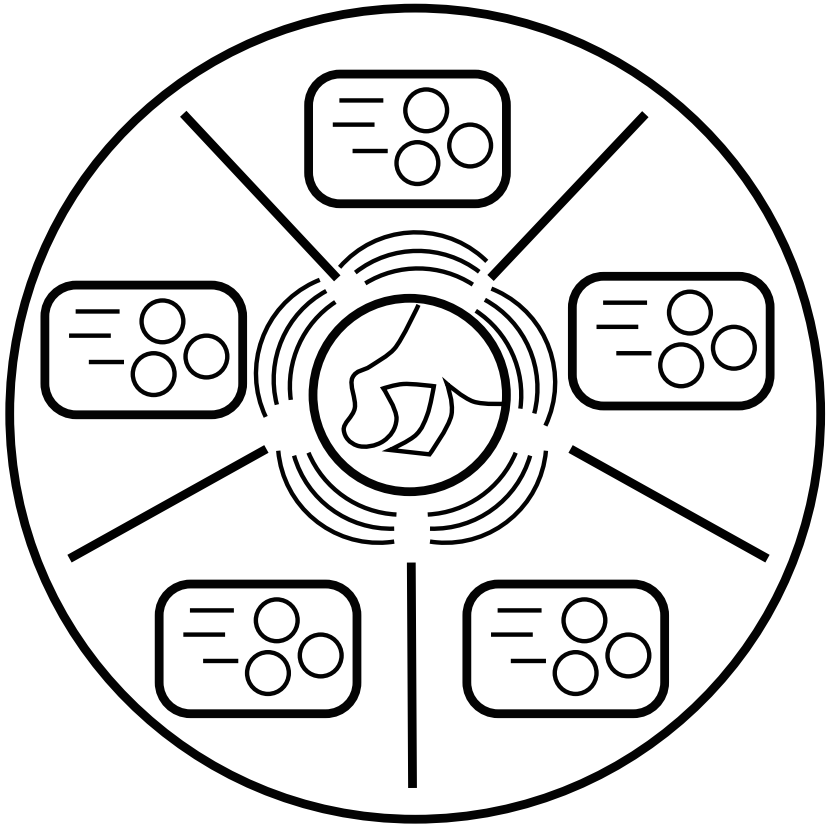
unlock

1. Process i ejects \$ into the skin

The lock is a special case

The Surfing Membranes Problem

- Five membranes; five antennae

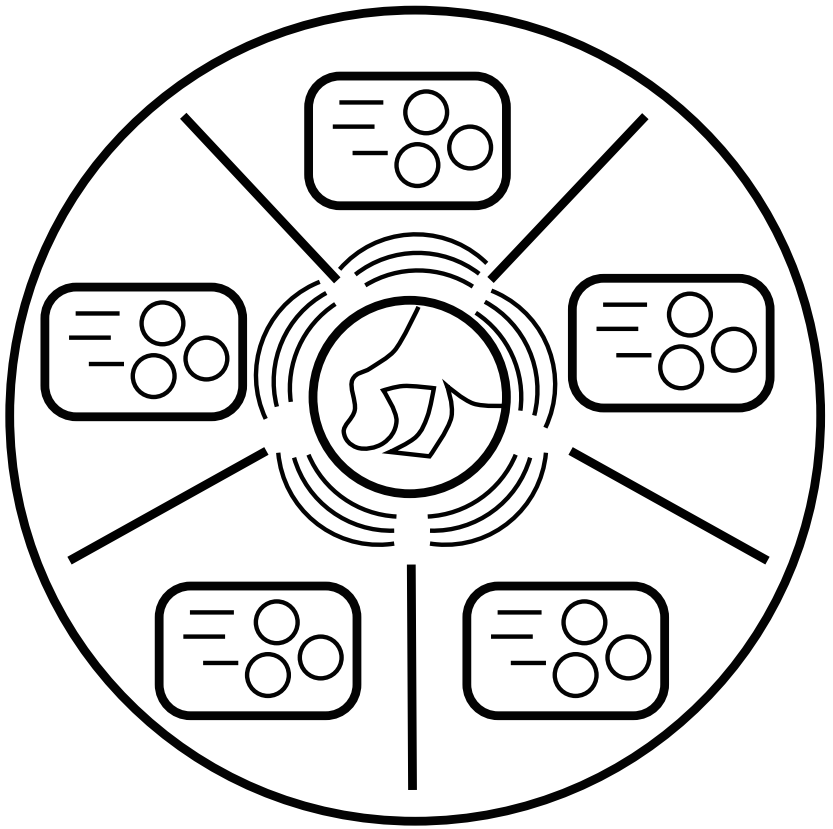


The Surfing Membranes Problem

- Five membranes; five antennae

A membrane

- is either surfing or thinking

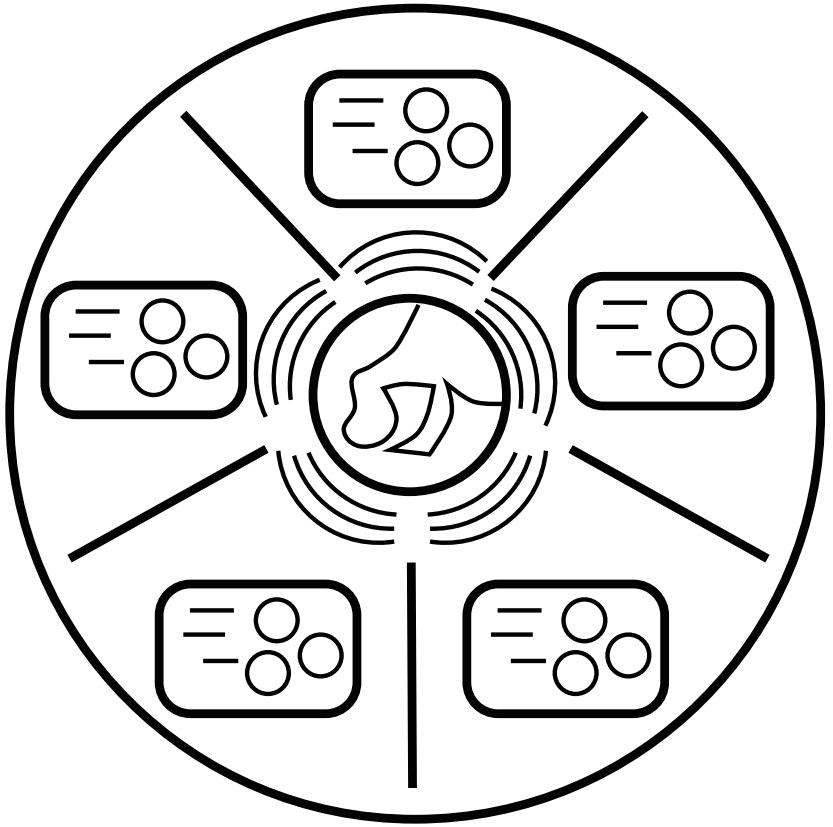


The Surfing Membranes Problem

- Five membranes; five antennae

A membrane

- is either surfing or thinking
- needs two antennae

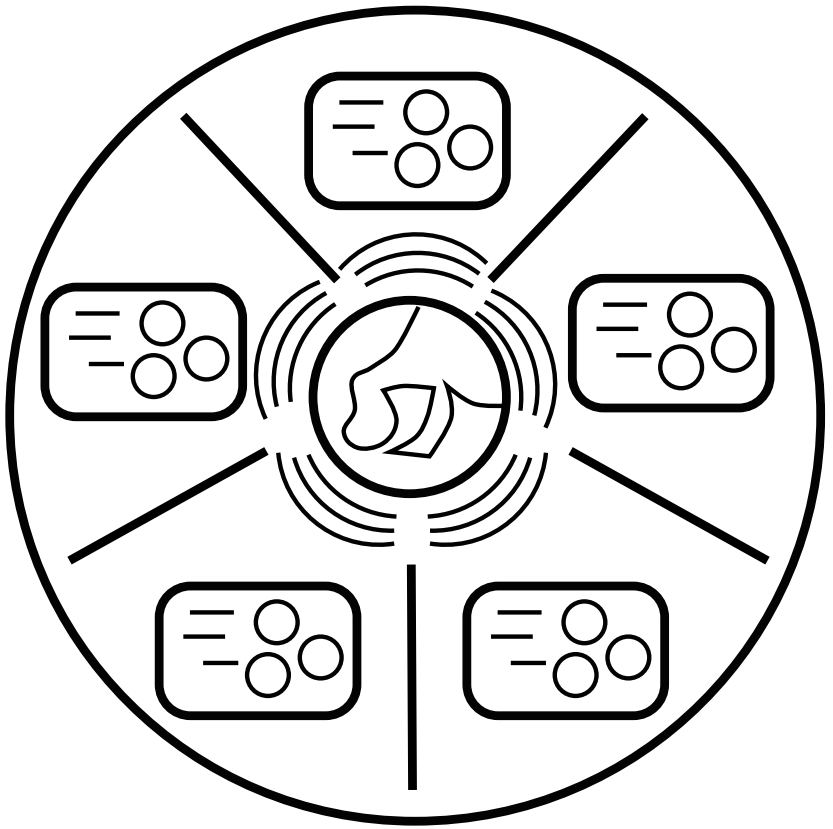


The Surfing Membranes Problem

- Five membranes; five antennae

A membrane

- is either surfing or thinking
- needs two antennae
- waits, if wants to surf, but doesn't have enough antennae



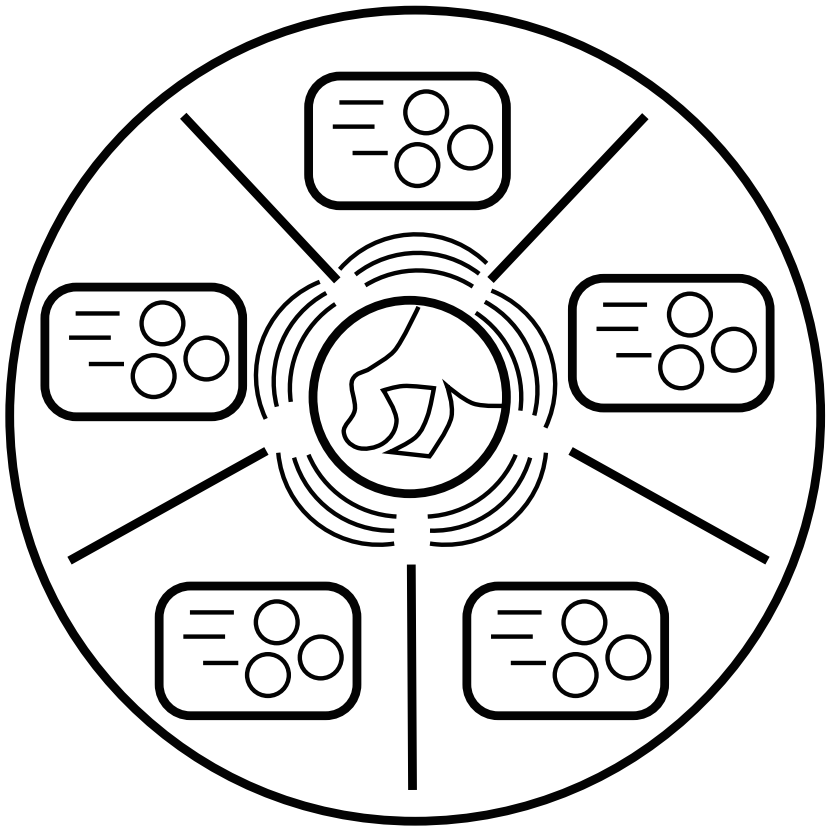
The Surfing Membranes Problem

- Five membranes; five antennae

A membrane

- is either surfing or thinking
- needs two antennae
- waits, if wants to surf, but doesn't have enough antennae

Problem

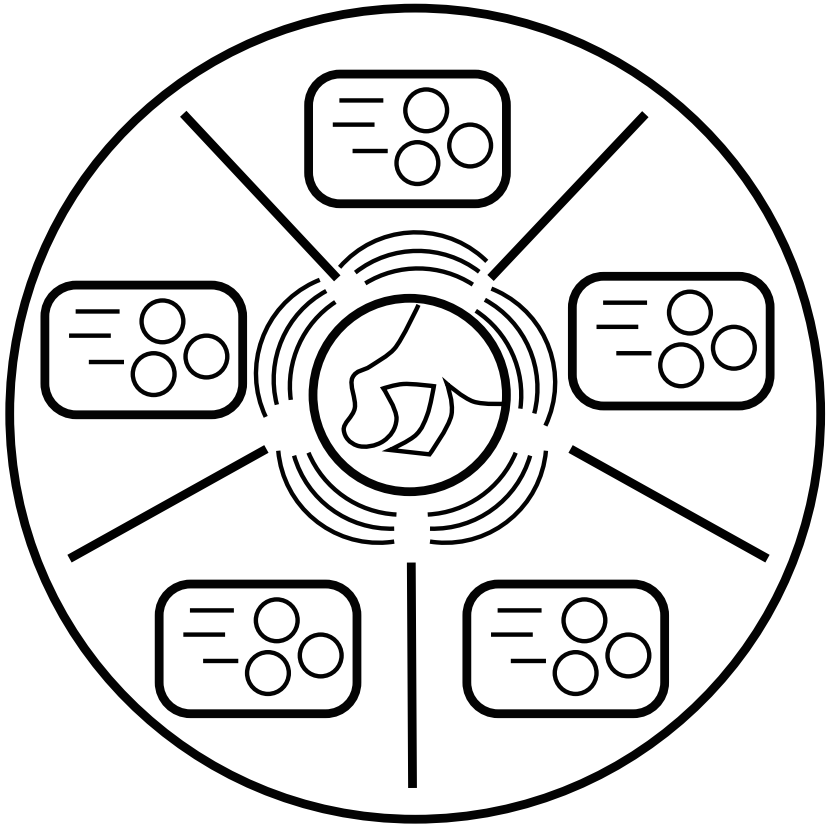


The Surfing Membranes Problem

- Five membranes; five antennae

A membrane

- is either surfing or thinking
- needs two antennae
- waits, if wants to surf, but doesn't have enough antennae



Problem

- Everyone must have a chance
- > 1 membrane can surf at a time

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\qquad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\qquad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\qquad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Regulates the number of surfing membranes

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\qquad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\qquad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Regulates taking of download antennae

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\qquad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\qquad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Regulates taking of upload antennae

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\qquad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\quad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Membrane thinks

Skin

$$R_s = \{p_i c \to p_i' c_i,\ p_i' \to (s_1, i), c_i \to \lambda,\quad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r_{id}' a_i',\ r_{id}' \to (s_2, i), a_i' \to \lambda,$$
$$r_{iu} a_{i+1} \to r_{iu}' a_{i+1}',\ r_{iu}' \to (s, i), a_{i+1}' \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\quad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Membrane tries to occupy a seat

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\quad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\qquad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Seat occupied

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\qquad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\quad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Membrane tries to take the download antenna

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\quad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\qquad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Membrane tries to take the download antenna

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\qquad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\qquad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Download antenna taken

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\qquad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\qquad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Membrane tries to take the upload antenna

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\qquad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\qquad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Membrane tries to take the upload antenna

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\qquad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\qquad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Upload antenna taken

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\qquad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\quad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Membrane surfs

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\quad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\qquad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Membrane starts thinking

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\qquad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\} \qquad w_i = t$$

Membrane stands up

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\} \qquad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$

The Surfing Membranes Problem: Solution

Membrane i

$$R_i = \{t \to (p_i, out),\ s_1 \to (r_{id}, out),\ s_2 \to (r_{iu}, out),\qquad w_i = t$$
$$s \to t(c, out)(a_i, out)(a_{i+1}, out)\}$$

Membrane frees antennae

Skin

$$R_s = \{p_i c \to p'_i c_i,\ p'_i \to (s_1, i), c_i \to \lambda,\qquad w_s = a_1 a_2 a_3 a_4 a_5 c^4$$
$$r_{id} a_i \to r'_{id} a'_i,\ r'_{id} \to (s_2, i), a'_i \to \lambda,$$
$$r_{iu} a_{i+1} \to r'_{iu} a'_{i+1},\ r'_{iu} \to (s, i), a'_{i+1} \to \lambda \mid 1 \leq i \leq m\}$$

Nasty Questions

- Random times $\Rightarrow$ barrier vs. strategy?

Nasty Questions

- Random times $\Rightarrow$ barrier vs. strategy?

  Consider less general models with less random times

Nasty Questions

- Random times $\Rightarrow$ barrier vs. strategy?

  Consider less general models with less random times

- Why parallelise at all?

Nasty Questions

- Random times $\Rightarrow$ barrier vs. strategy?

  Consider less general models with less random times

- Why parallelise at all?
  - To better understand the asynchronicity of clock-free P systems
  - To collect experience useful in real-world synchronisation

Conclusions and Open Questions

Conclusions

- Solutions to some basic, yet fundamental concurrency problems
- Link between parallel and concurrent programming and (clock-free) systems

Open Questions

- Formal definition of concurrency problem
- Applicability of results to time-free P systems
- Extension to fully asynchronous P systems
- Simple catalysts; avoidance of bistable catalysts

The Ultimately Last Slide

Clock-free

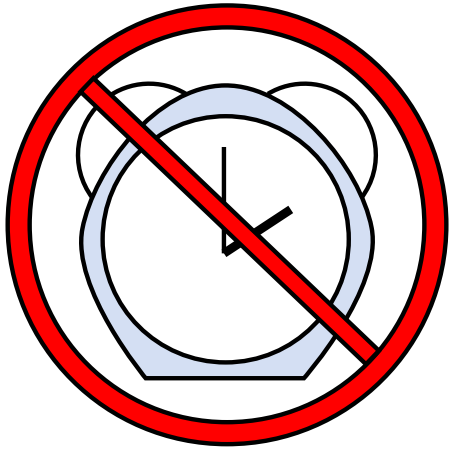
Locks

Concurrency

Catalysts

Strategies

Semaphores



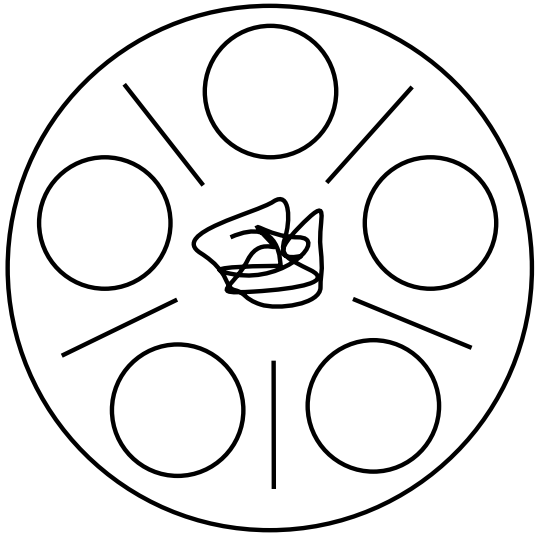
Thank you for attention and questions!

Barrier

Rule synchronisation

Sequential

Process synchronisation



Helped out the membranes!