10. Leader Election on a Ring-shaped Network

Jean-Raymond Abrial

2009

Purpose of this Lecture

- Learning more about modeling (in particular refinement)

- Learning some more modeling conventions

- Learning how to formalize an interesting structure: a ring

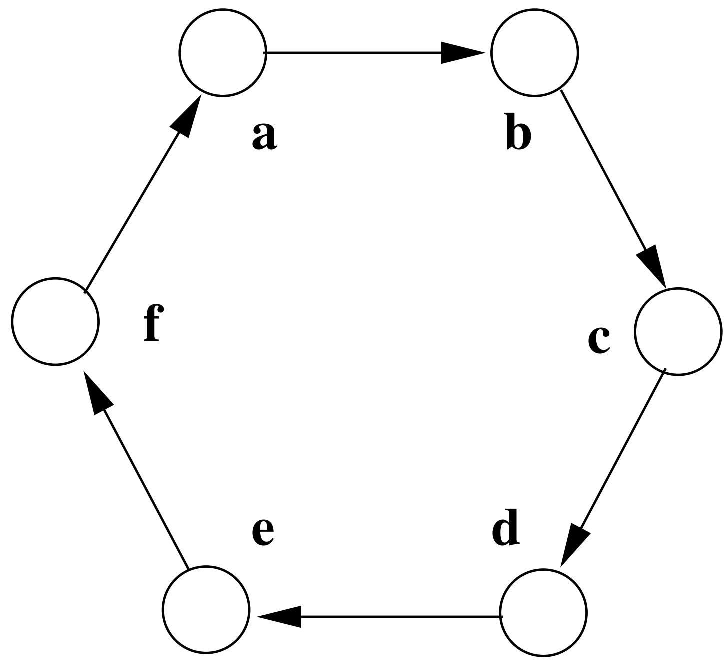
- Study a classical problem in distributed computing

- The example comes from the following paper:

G. Le Lann. *Distributed systems - towards a formal approach*.

In B Gilchrist, editor *Information Processing 77* North-Holland 1977.

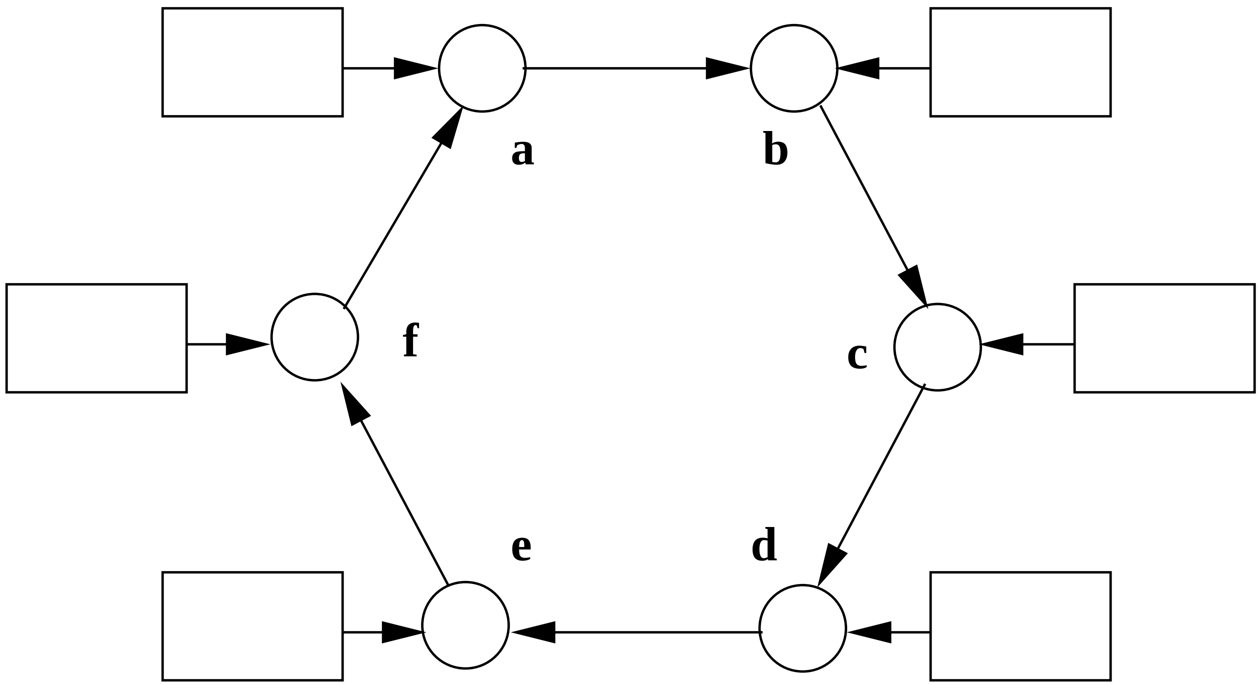
A Ring of Nodes



- Each node is able to send messages to its right neighbor

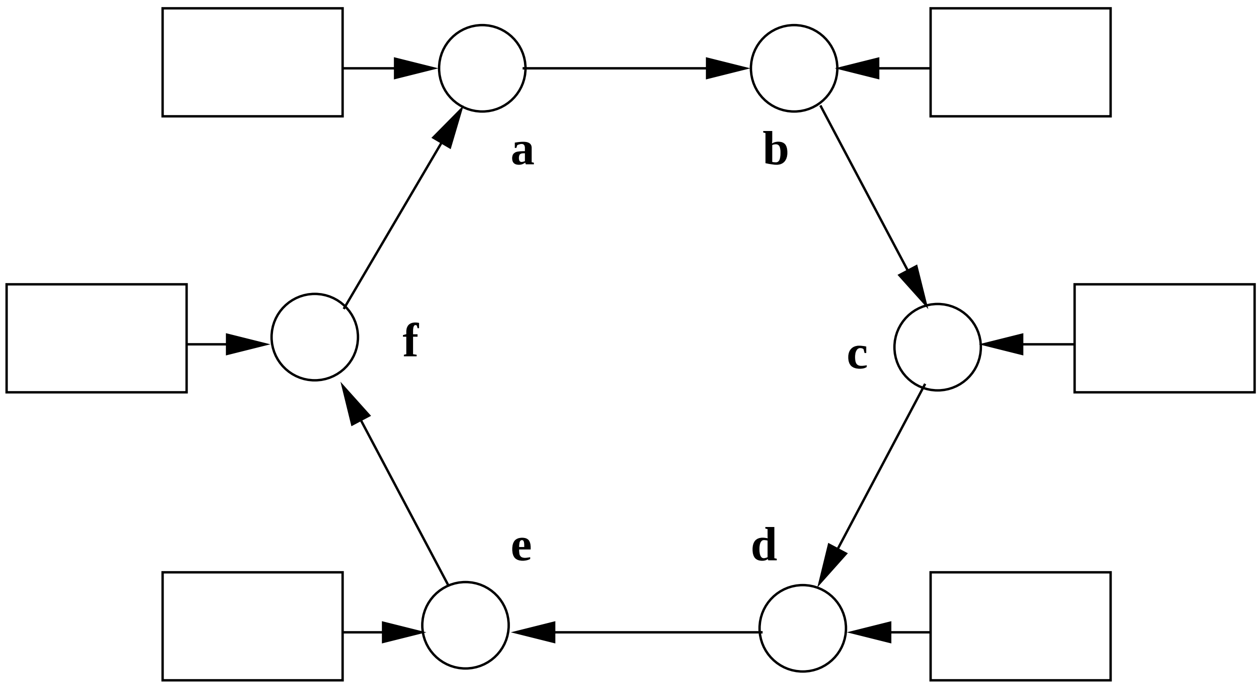
- Each node is able to receive messages from its left neighbor

A Ring of Nodes with Buffers



- Messages can be buffered in each node before being sent

- Messages can be reordered in each buffer



- After some (finite) time a unique node becomes the leader

- Constraint: each node executes the same piece of code

Reference Document (1)

We have a set of nodes forming a ring	ENV-1
Each node can send a message to the next one in the ring	ENV-2
Messages can be buffered in each node	ENV-3

Reference Document (2)

Messages can be re-ordered in their buffer	ENV-4

The distributed program is made by the same piece of code executed by each node	ENV-5

The purpose of the distributed program is to have a unique node being elected the leader	FUN-1

Discussion (1)

Discussion (1)

- Seems impossible

Discussion (1)

- Seems impossible

- Nothing makes one process different from the other

Discussion (1)

- Seems impossible

- Nothing makes one process different from the other

- The ring structure is homogeneous (no first, no last)

Discussion (1)

- Seems impossible

- Nothing makes one process different from the other

- The ring structure is homogeneous (no first, no last)

- The only difference between processes is their name

Discussion (1)

- Seems impossible

- Nothing makes one process different from the other

- The ring structure is homogeneous (no first, no last)

- The only difference between processes is their name

- But, it is not sufficient to make a difference between processes

Discussion (2)

- Giving more structure to names

Discussion (2)

- Giving more structure to names

- Names are natural numbers

Discussion (2)

- Giving more structure to names

- Names are natural numbers

- A special process is thus the one with the largest name

Discussion (2)

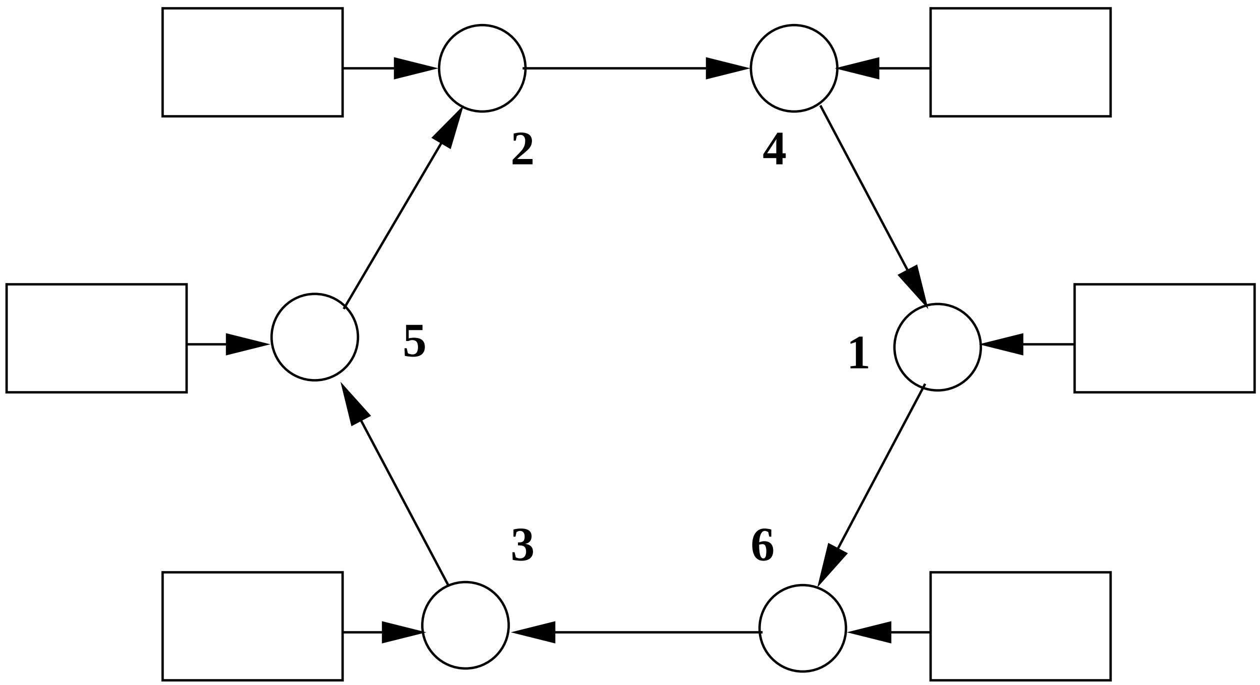
- Giving more structure to names

- Names are natural numbers

- A special process is thus the one with the largest name

- How can a process know that it bears the largest name?

Nodes with Numbered Names



Reference Document (3)

Each node has a unique name which is a Natural Number	ENV-6

The leader must be the node with the largest name	FUN-2

But remember: each node executes the same piece of code.

Initial Model: The Constants

The nodes forms a non-empty finite set of natural numbers

constants: N

axm0_1:	$N \subseteq \mathbb{N}$
axm0_2:	$\text{finite}(N)$
axm0_3:	$N \neq \varnothing$

Initial Model: The Variable and the Events

variables: w

inv0_1: $w \in N$

- A single event describes the situation at the end of the protocol

init
$w :\in N$

elect
$w := \max(N)$

- Notice that $\max(N)$ is well-defined since N is finite and non-empty.

- We do not introduce the ring yet

First Refinement: Solution 1

- Initially, each process puts its own name in its buffer

First Refinement: Solution 1

- Initially, each process puts its own name in its buffer

- Each process passes any name in its buffer to its right neighbor

First Refinement: Solution 1

- Initially, each process puts its own name in its buffer

- Each process passes any name in its buffer to its right neighbor

- Receiving a name (except its own), a process puts it in its buffer

First Refinement: Solution 1

- Initially, each process puts its own name in its buffer

- Each process passes any name in its buffer to its right neighbor

- Receiving a name (except its own), a process puts it in its buffer

- Processes also keep the names they receive

First Refinement: Solution 1

- Initially, each process puts its own name in its buffer

- Each process passes any name in its buffer to its right neighbor

- Receiving a name (except its own), a process puts it in its buffer

- Processes also keep the names they receive

- When a process receives its own name then it tests for leadership

First Refinement: Solution 1

- Initially, each process puts its own name in its buffer

- Each process passes any name in its buffer to its right neighbor

- Receiving a name (except its own), a process puts it in its buffer

- Processes also keep the names they receive

- When a process receives its own name then it tests for leadership

- Does it work?

First Refinement: Solution 1

- Initially, each process puts its own name in its buffer

- Each process passes any name in its buffer to its right neighbor

- Receiving a name (except its own), a process puts it in its buffer

- Processes also keep the names they receive

- When a process receives its own name then it tests for leadership

- Does it work? NO: because messages can be reordered in buffers

First Refinement: Solution 2

- Almost the same as solution 1

First Refinement: Solution 2

- Almost the same as solution 1

- Each process knows the number n of different processes

First Refinement: Solution 2

- Almost the same as solution 1

- Each process knows the number n of different processes

- A process starts testing after receiving n different names

First Refinement: Solution 2

- Almost the same as solution 1

- Each process knows the number n of different processes

- A process starts testing after receiving n different names

- Does it work?

First Refinement: Solution 2

- Almost the same as solution 1

- Each process knows the number n of different processes

- A process starts testing after receiving n different names

- Does it work? YES, but is rather heavy

First Refinement: Solution 2

- Almost the same as solution 1

- Each process knows the number n of different processes

- A process starts testing after receiving n different names

- Does it work? YES, but is rather heavy

- Knowing the number of different processes is not always possible

First Refinement: Solution 3

- Initially, each process puts its own name in its buffer

First Refinement: Solution 3

- Initially, each process puts its own name in its buffer

- Each process passes any name in its buffer to its right neighbor

First Refinement: Solution 3

- Initially, each process puts its own name in its buffer

- Each process passes any name in its buffer to its right neighbor

- A name is rejected if smaller than that of the receiving process

First Refinement: Solution 3

- Initially, each process puts its own name in its buffer

- Each process passes any name in its buffer to its right neighbor

- A name is rejected if smaller than that of the receiving process

- Receiving a name (except its own), a process puts it in its buffer

First Refinement: Solution 3

- Initially, each process puts its own name in its buffer

- Each process passes any name in its buffer to its right neighbor

- A name is rejected if smaller than that of the receiving process

- Receiving a name (except its own), a process puts it in its buffer

- A process receiving its own name is the leader

First Refinement: Solution 3

- Initially, each process puts its own name in its buffer

- Each process passes any name in its buffer to its right neighbor

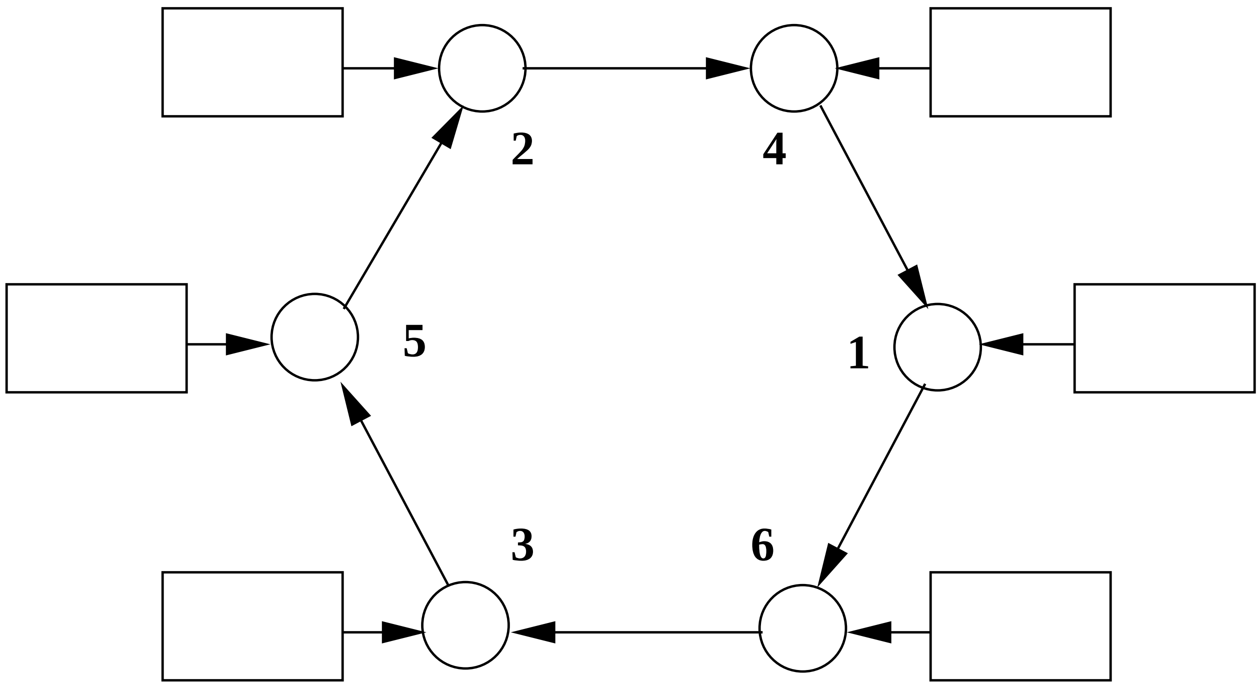
- A name is rejected if smaller than that of the receiving process

- Receiving a name (except its own), a process puts it in its buffer

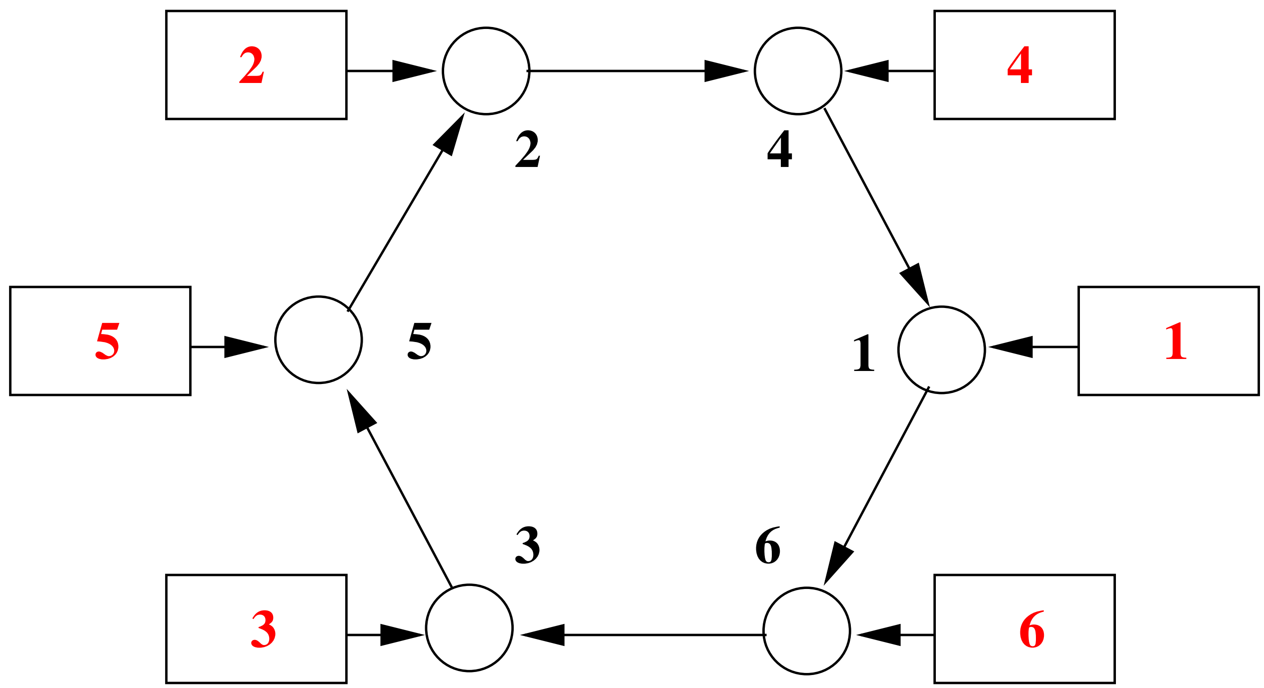
- A process receiving its own name is the leader

- Does it work???

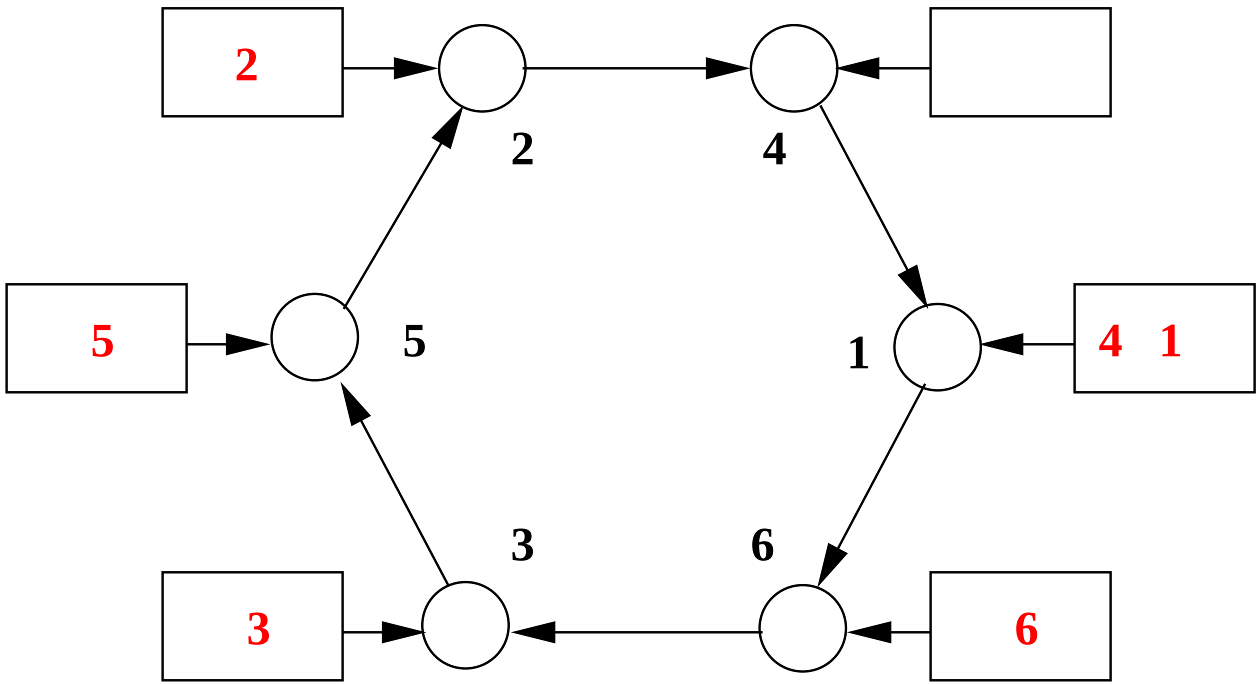
Basic Situation



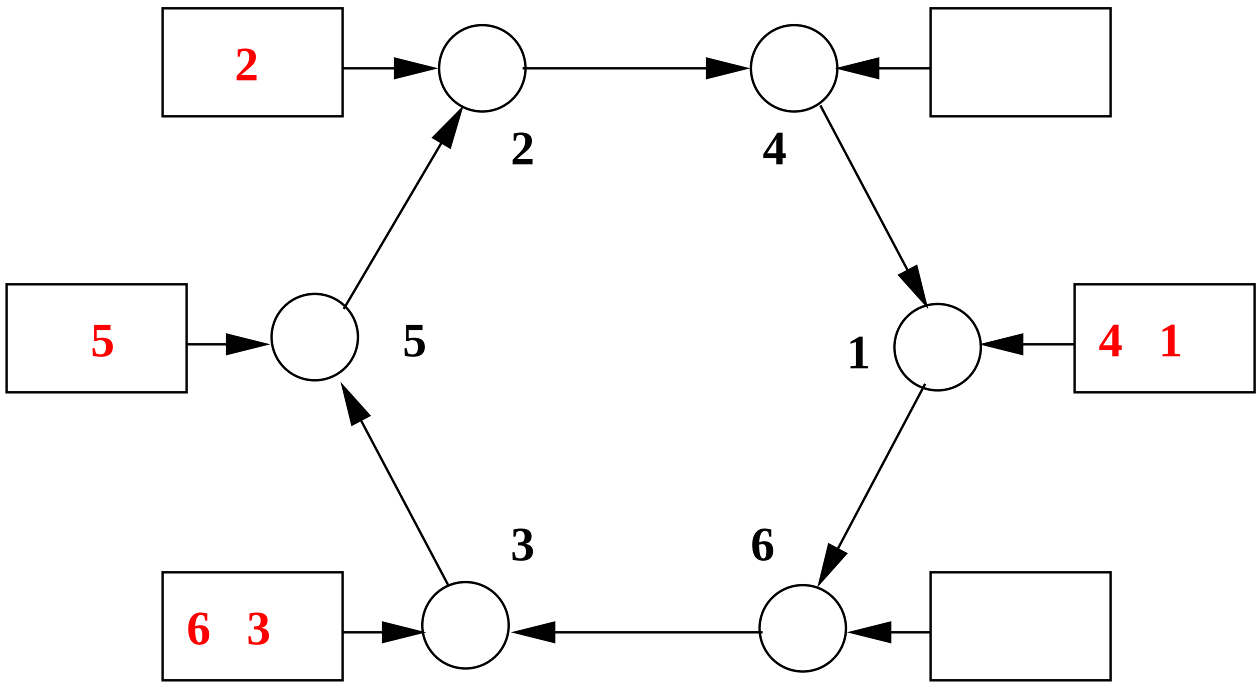
Initial Situation



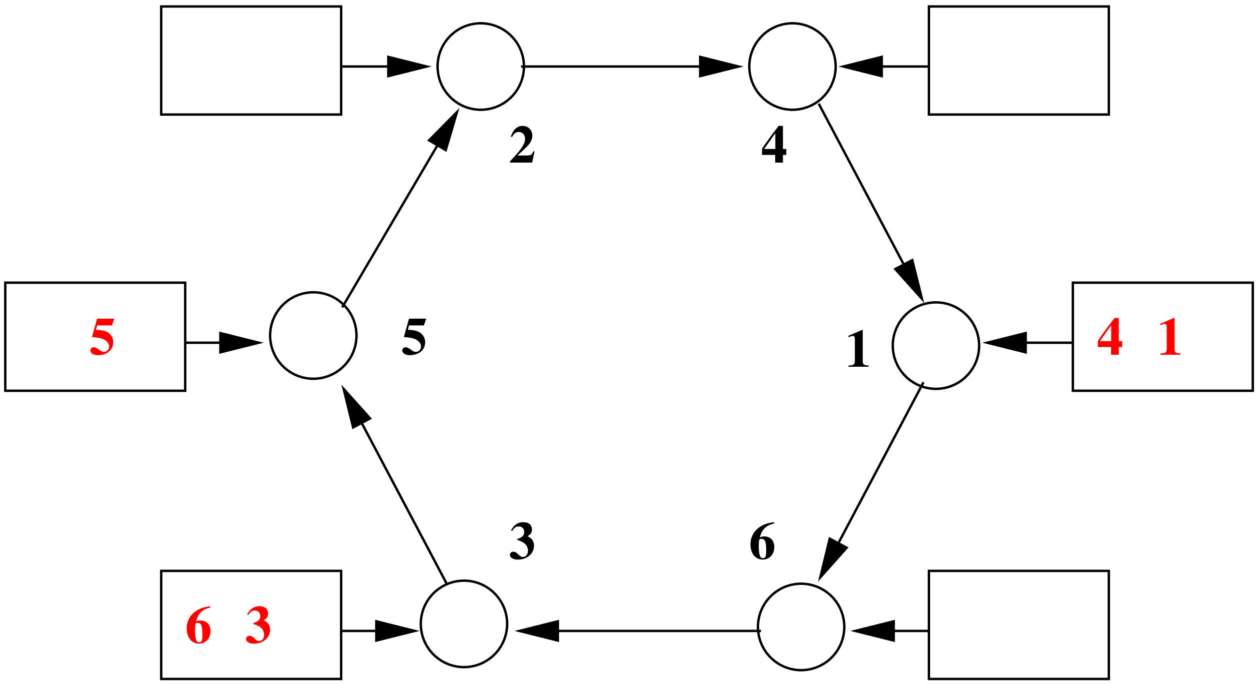
Accepting 4



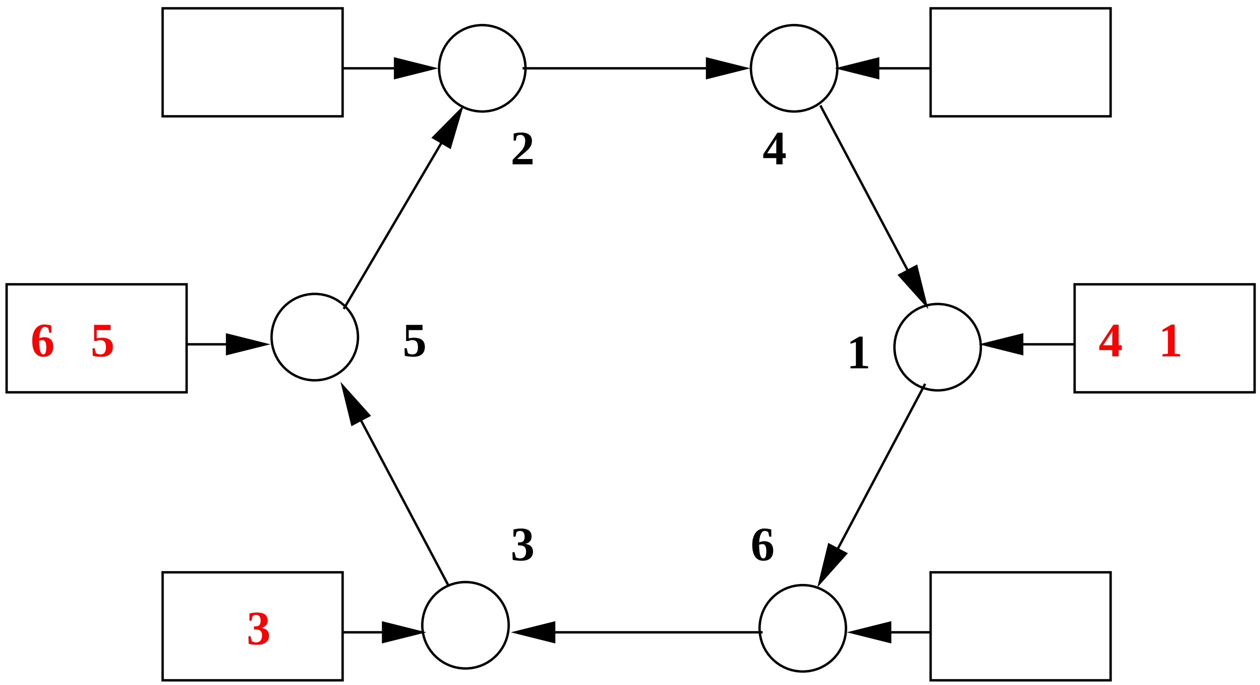
Accepting 6



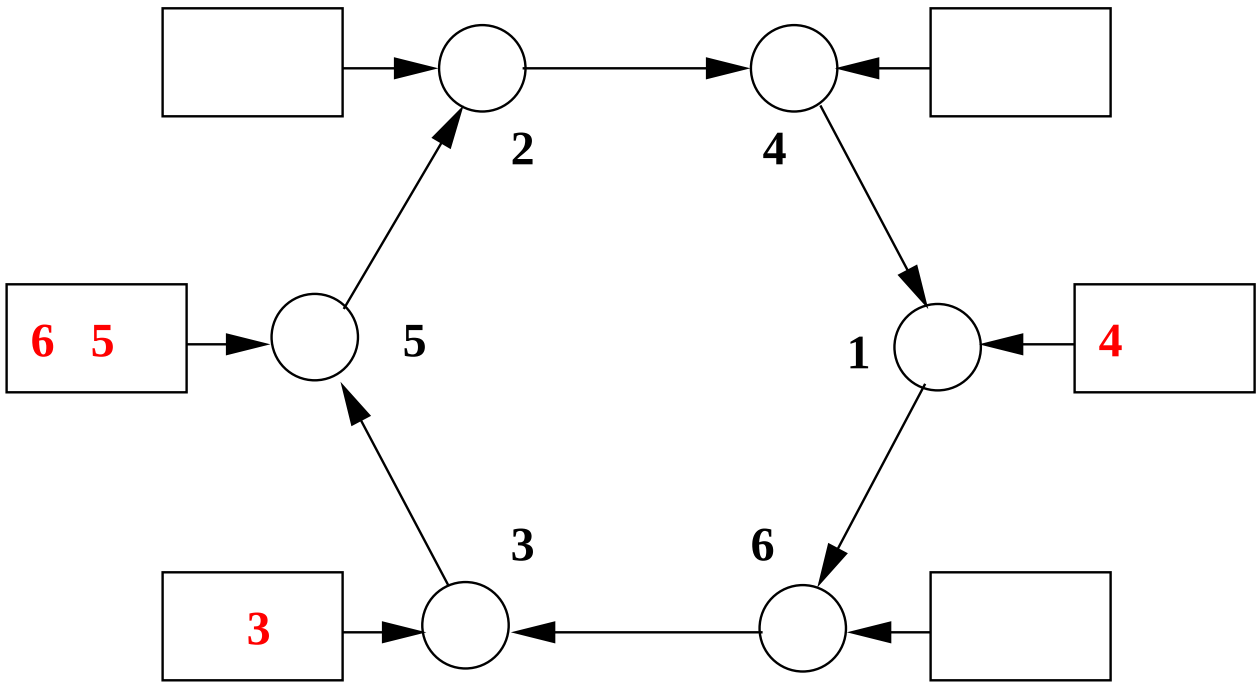
Rejecting 2



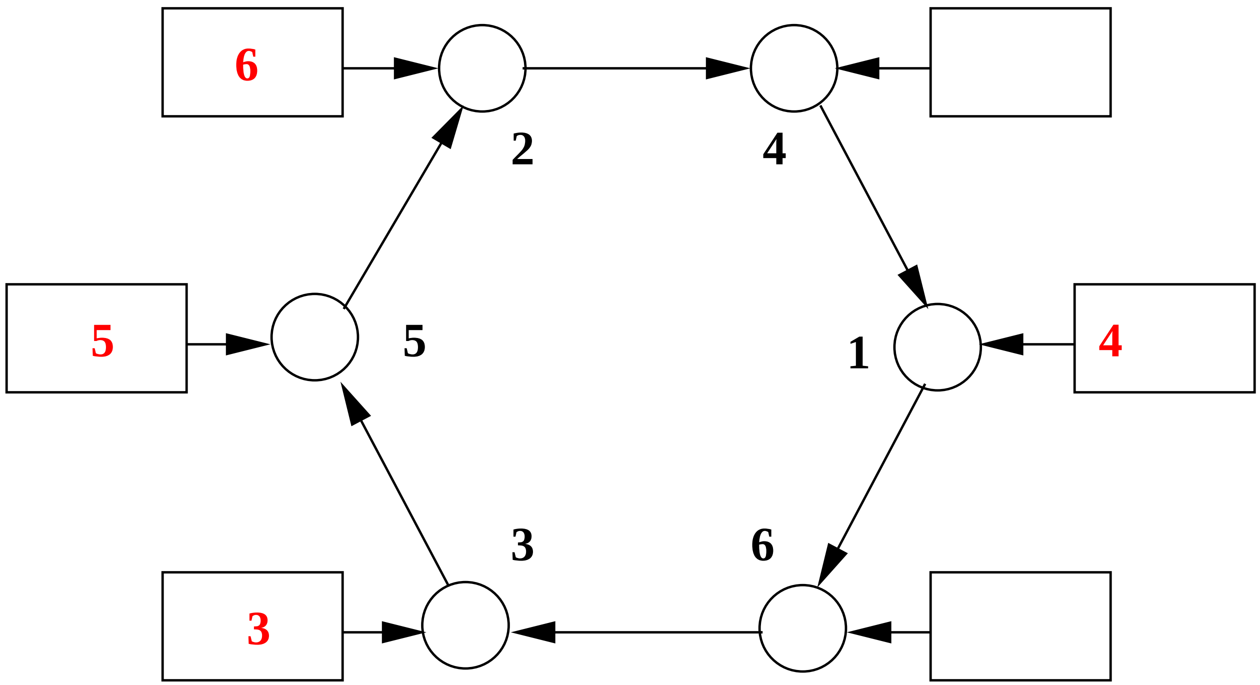
Accepting 6



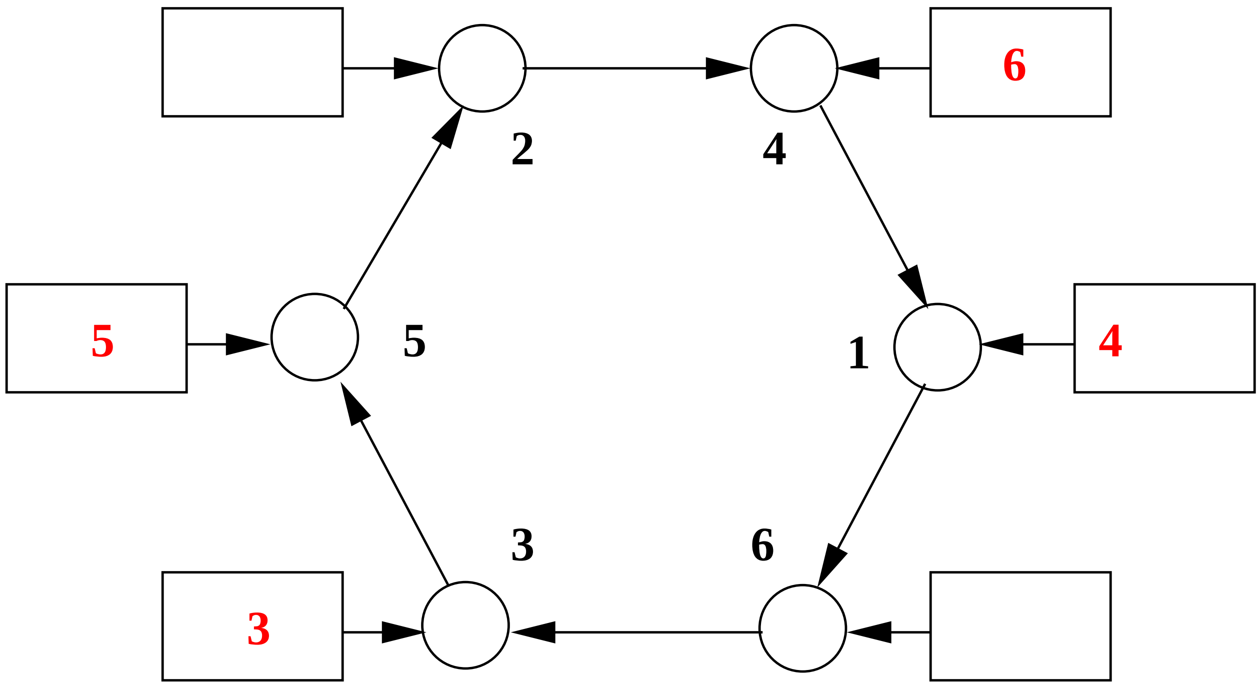
Rejecting 1



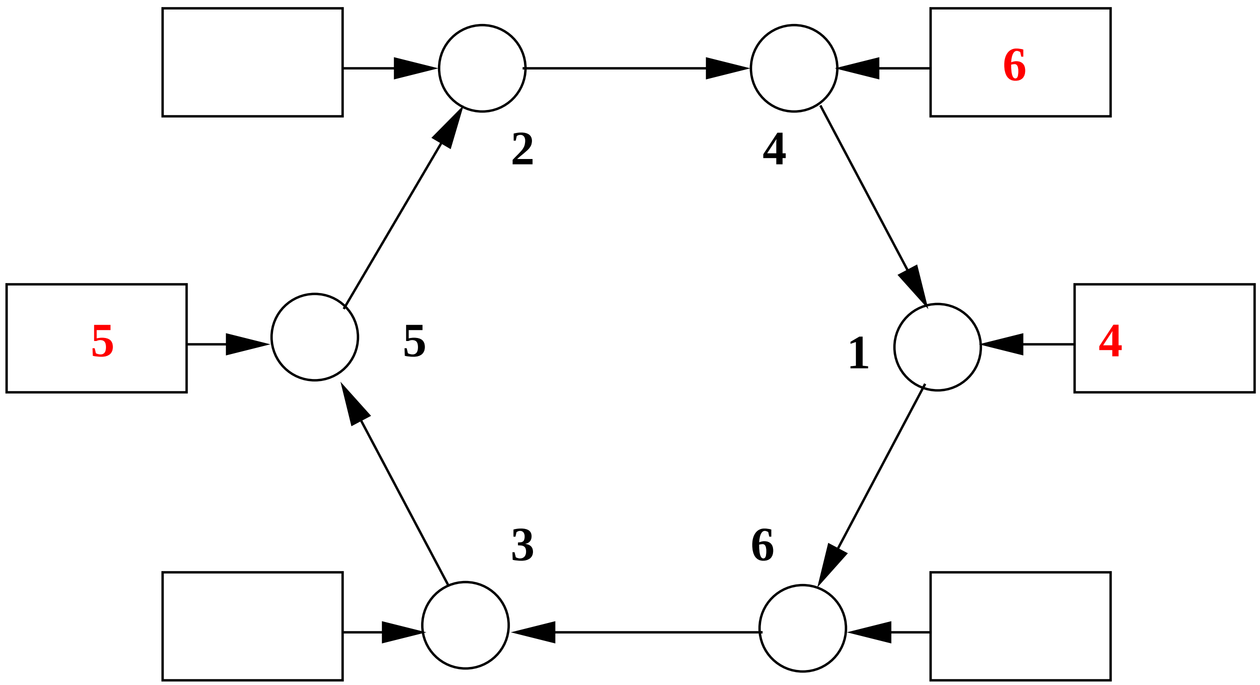
Accepting 6



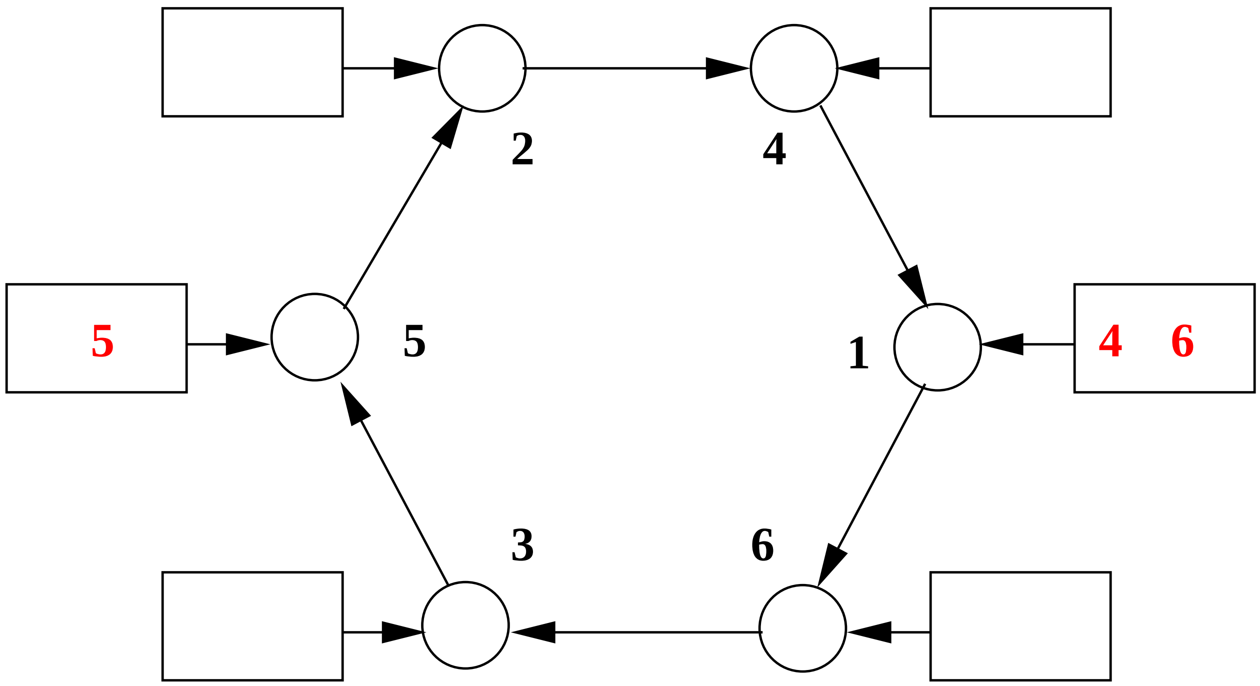
Accepting 6



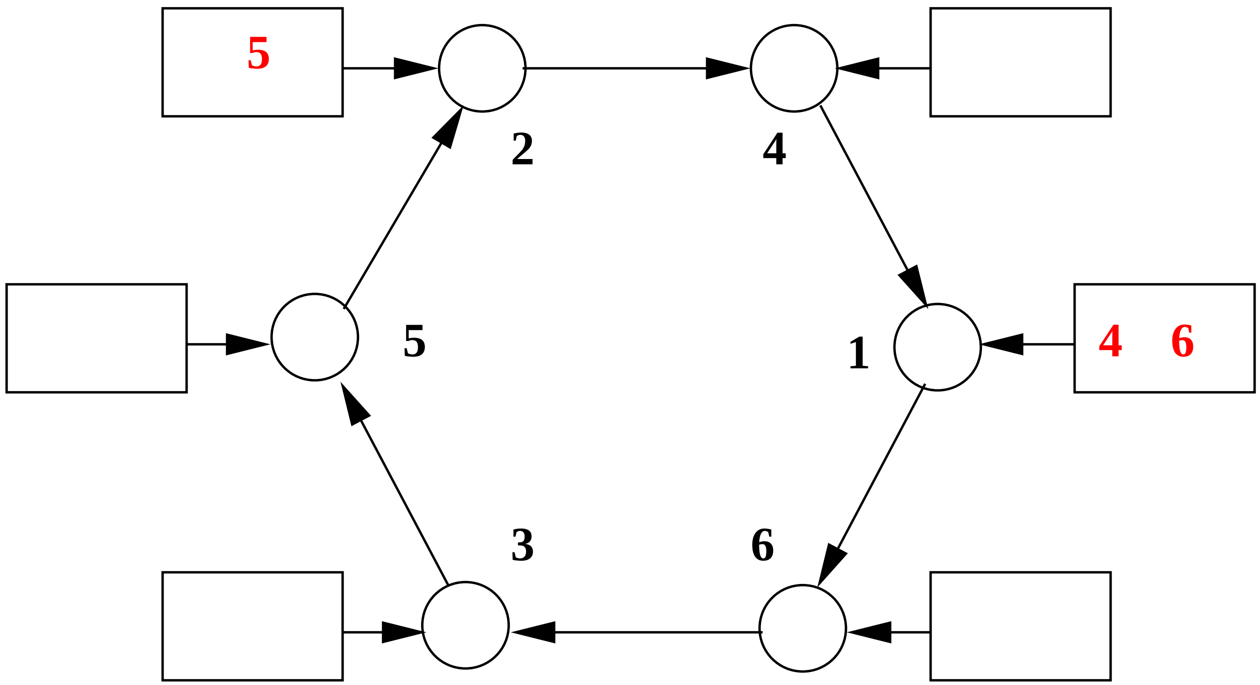
Rejecting 3



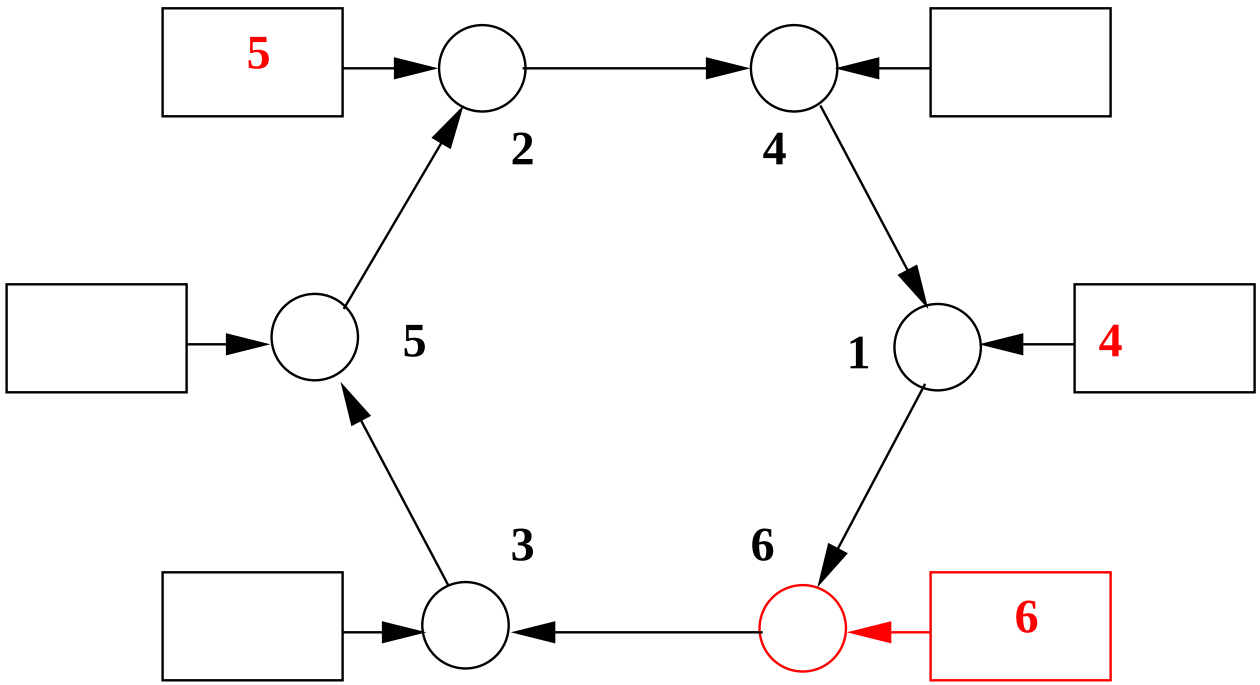
Accepting 6



Accepting 5

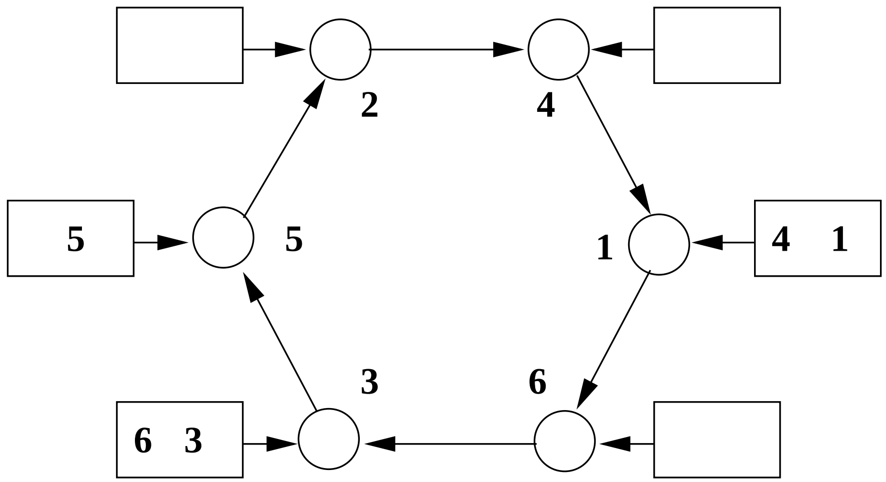


Accepting 6. It is the Winner



Being More Precise

- We define a function a linking each node to the buffer where it is



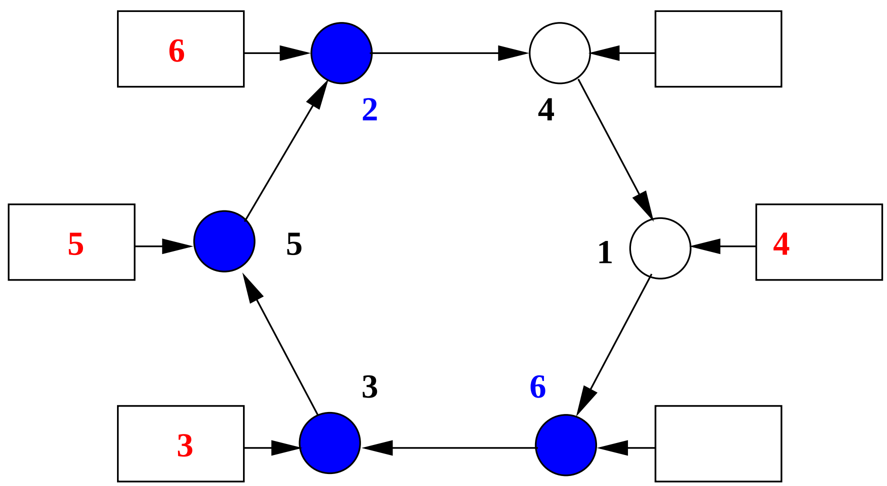
- As can be seen, the function a is the following:

$$a = \{1 \mapsto 1,\ 3 \mapsto 3,\ 4 \mapsto 1,\ 5 \mapsto 5,\ 6 \mapsto 3\}$$

- From this state, the situation can evolve in many different ways.

The Key Insight

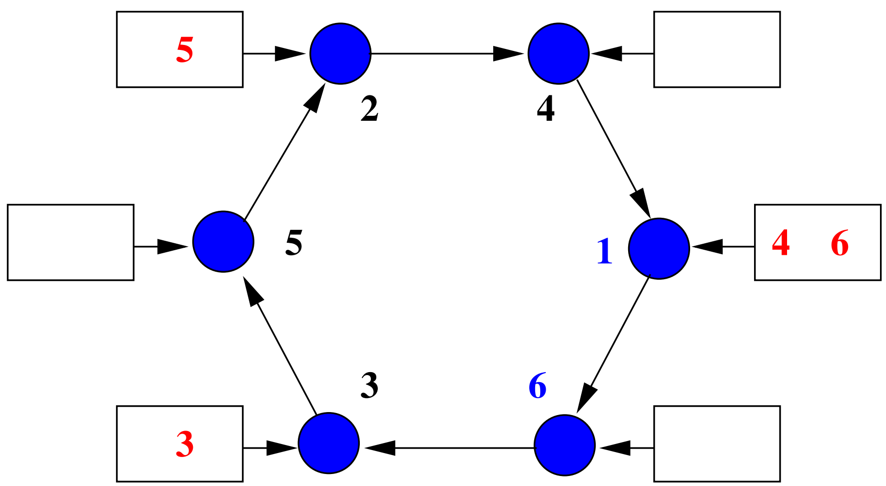
$\boldsymbol{a(6) = 2}$, then 6 is the maximum of the interval between 6 and 2



We have then:

$$6 = \max(\{6,3,5,2\})$$

- The final situation



$$6 = \max(\{6,3,5,2,4,1\}) = \max(N)$$

constants: n
i

Constant n denotes the "next" function in the ring built on N

i denotes the interval function

axm1_1: $n \in N \rightarrowtail\!\!\!\!\twoheadrightarrow N$

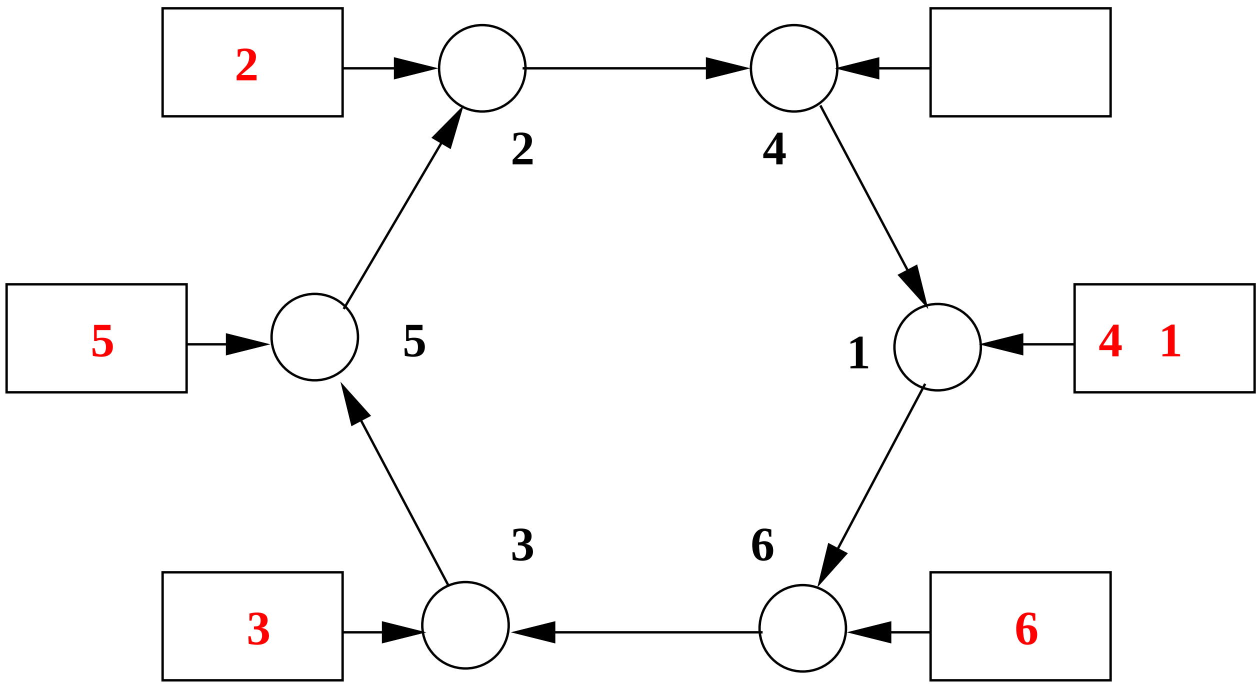
axm1_2: $\forall S \cdot n^{-1}[S] \subseteq S \;\wedge\; S \neq \varnothing \;\Rightarrow\; N \subseteq S$

axm1_3: $i \in N \times N \rightarrow \mathbb{P}(N)$

axm1_4: $\forall x \cdot x \in N \;\Rightarrow\; i(x \mapsto x) = \{x\}$

axm1_5: $\forall x \cdot x \in N \wedge y \in N \setminus \{x\} \;\Rightarrow\; i(x \mapsto y) = i(x \mapsto n^{-1}(y)) \cup \{y\}$

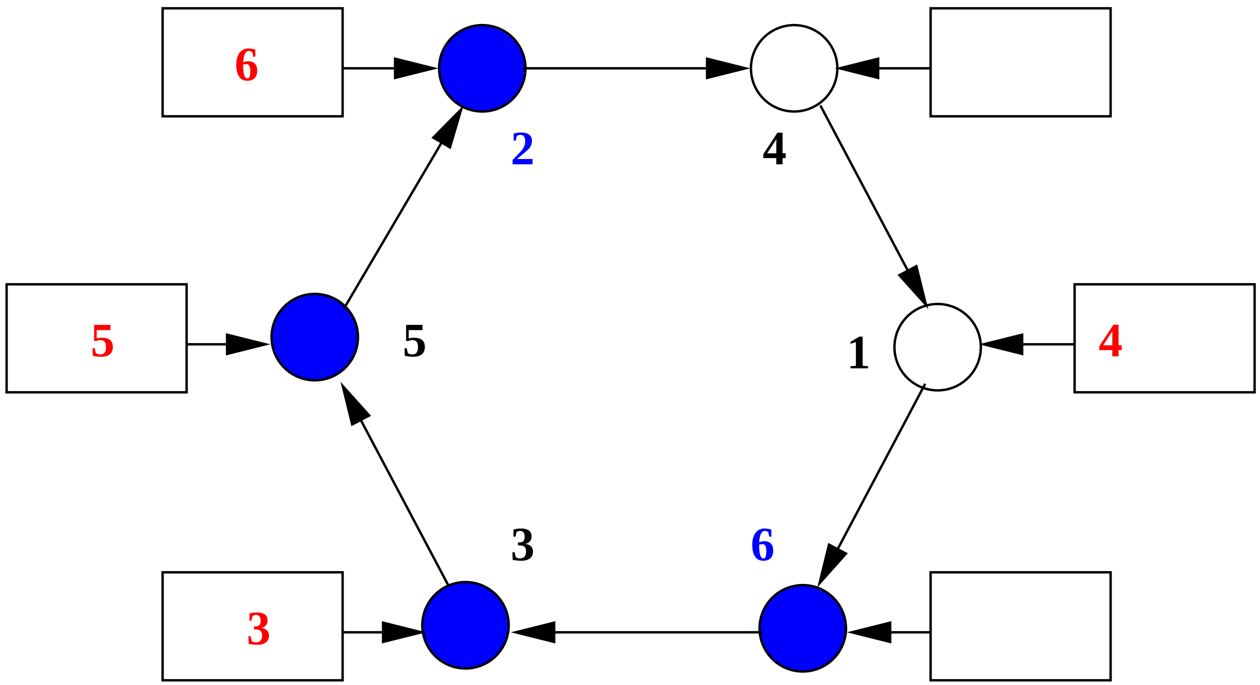
axm1_6: $\forall x \cdot x \in N \;\Rightarrow\; i(x \mapsto n^{-1}(x)) = N$



- Each name is at most in one position

- Each name is either rejected or accepted to the next position

The Key Observation



- 6 is the maximum of the blue interval $\{6, 3, 5, 2\}$

- 6 has been accepted successfully from position 6 to position 2

First Refinement: the State

variables: w, a

inv1_1: $a \in N \rightarrowtail\mkern-14mu\mapsto N$

inv1_2: $\forall x \cdot x \in \mathrm{dom}\,(a) \Rightarrow x = \mathrm{max}\,(i(x \mapsto n^{-1}(a(x))))$

- a yields the position of each node that has not yet been rejected

First Refinement: the Events

init
$\quad w :\in N$
$\quad a := n$

elect
any x **where**
$\quad x \in \operatorname{dom}(a)$
$\quad x = a(x)$
then
$\quad w := x$
end

accept
any x **where**
$\quad x \in \operatorname{dom}(a)$
$\quad a(x) < x$
then
$\quad a(x) := n(a(x))$
end

reject
any x **where**
$\quad x \in \operatorname{dom}(a)$
$\quad x < a(x)$
then
$\quad a := \{x\} \lhd a$
end