
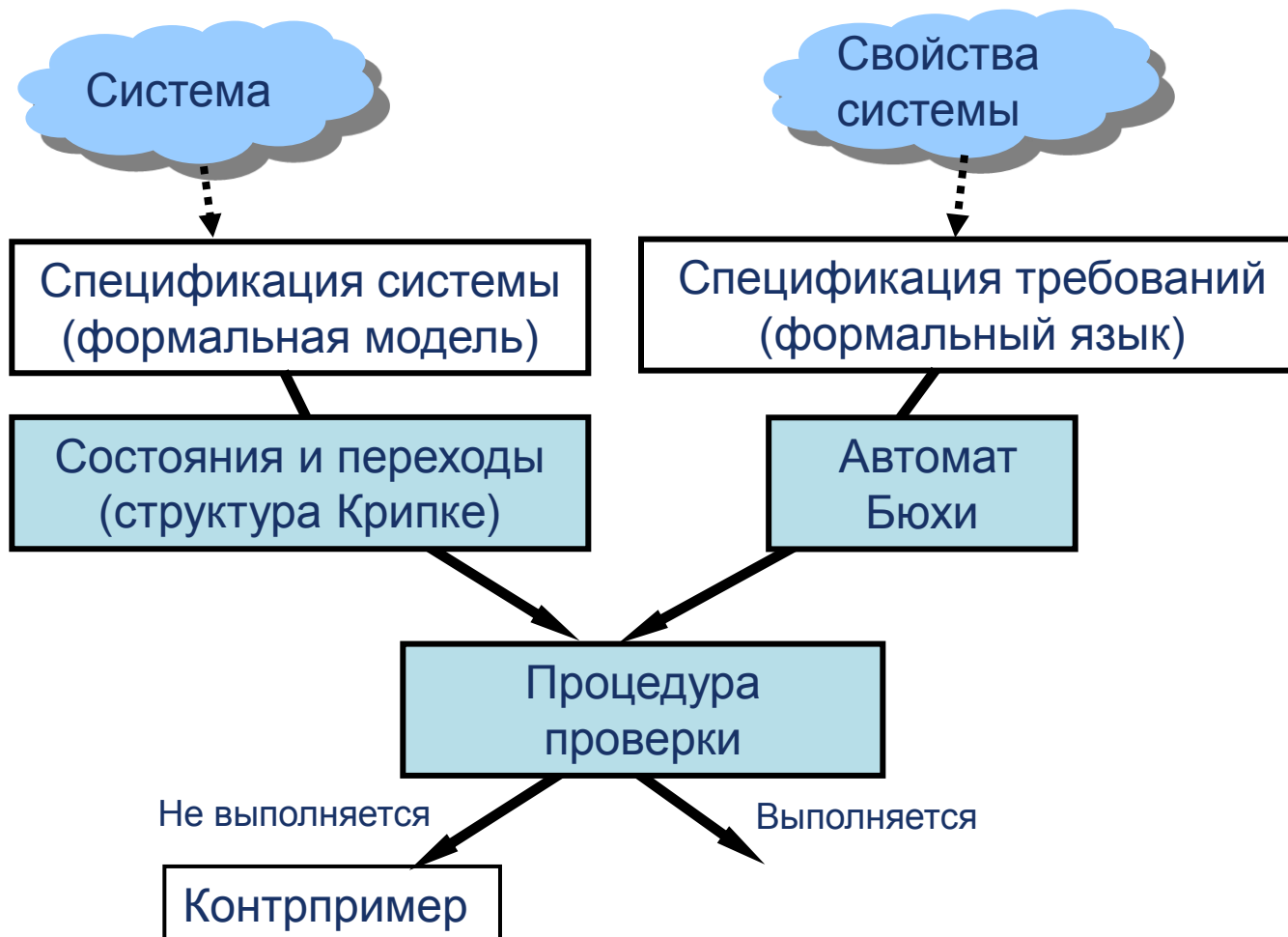


# Моделирование и верификация распределенных систем в среде SPIN

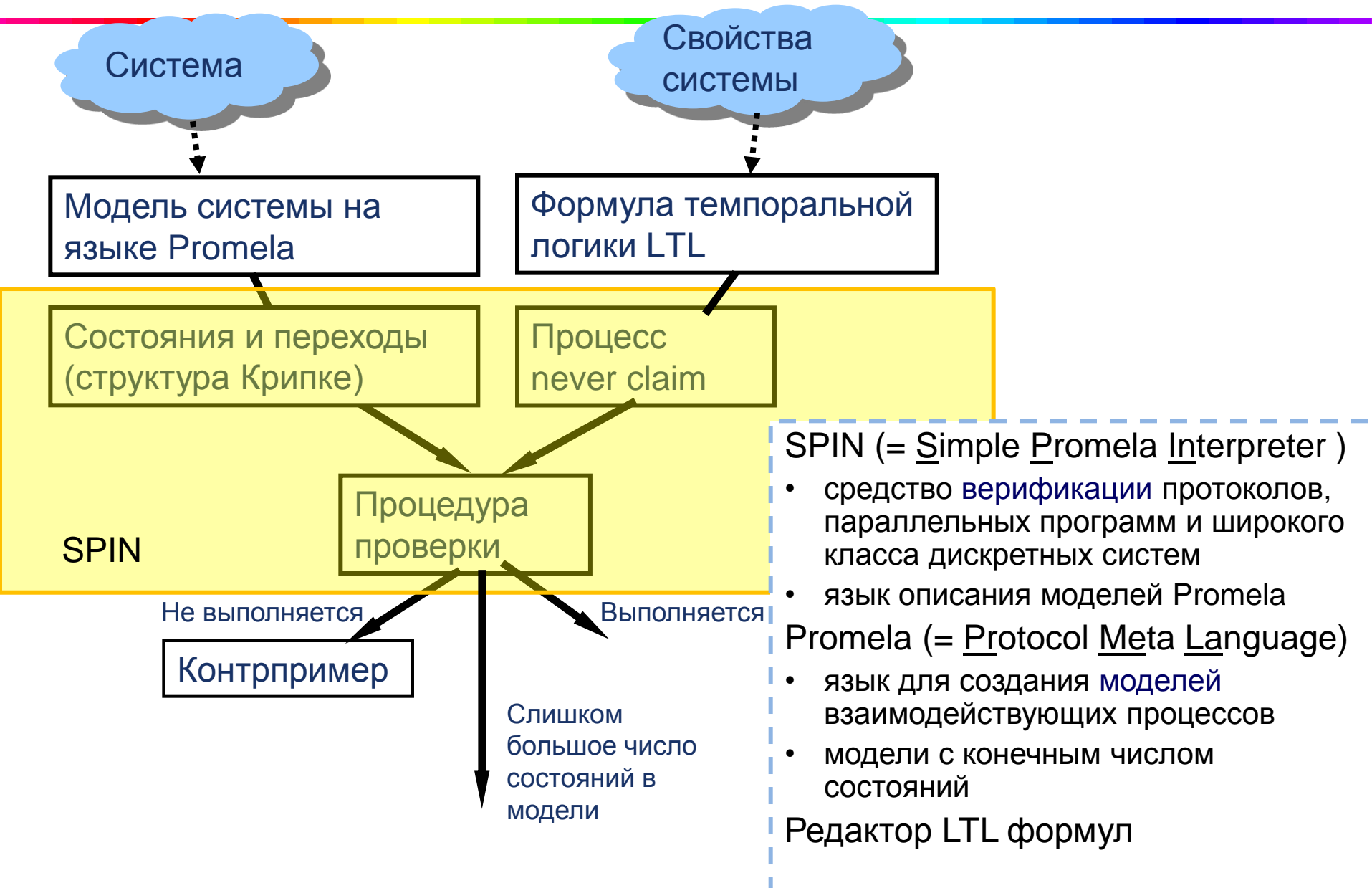


И.В.Шошмина  
ishoshmina@dcn.ftk.spbstu.ru  
РВКС, ИИТУ, СПбГПУ  
2013

# Средство верификации SPIN



# Средство верификации SPIN

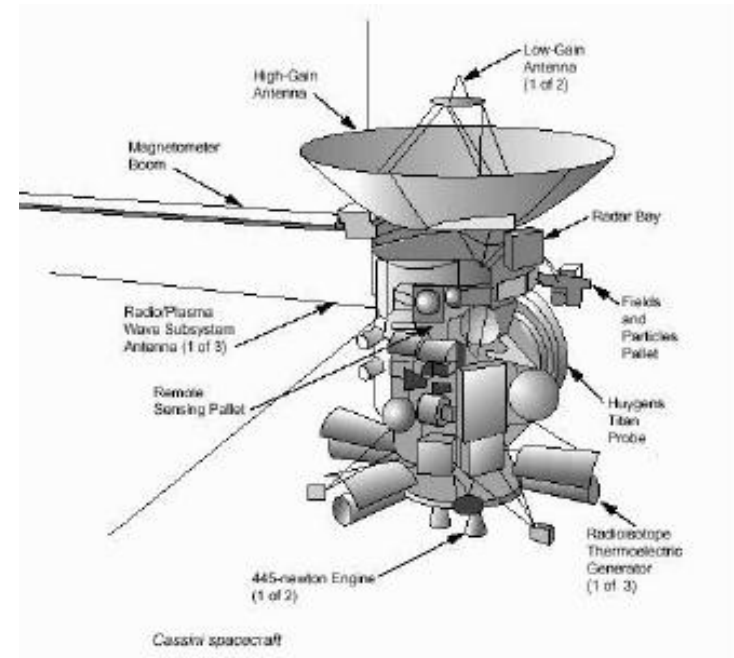


# SPIN

- разрабатывается Bell Labs с 1996
  - строить параллельные модели систем
  - выразить требования на языке LTL
  - автоматически верифицировать выполнение требования на модели
- премия ACM System Award в 2001

Использовался при верификации:

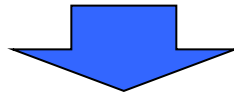
- ATC PathStar (Lucent)
- системы управления шлюзами в Роттердаме
- аэрокосмические системы Mars Exploration Rovers (NASA)
- и в других проектах



# Promela входной язык SPIN

## Модели, создаваемые на языке Promela

- абстракция реальной системы, содержащая характеристики, которые значимы для описания взаимодействия процессов
- модель не является программной реализацией системы
- модель может содержать части, которые важны только для верификации протоколов
- язык Promela имеет формальную семантику



- Promela включает примитивы для создания процессов и описания межпроцессного взаимодействия
- НО! в нем отсутствует ряд средств, которые есть в языках программирования высокого уровня
  - Например, указатели на данные и функции, не включено понятие времени или часов, отсутствуют операции с плавающей точкой и пр.

# Механизмы межпроцессного взаимодействия в Promela

---

- Разделяемые переменные
- Синхронные взаимодействия
- Асинхронные взаимодействия

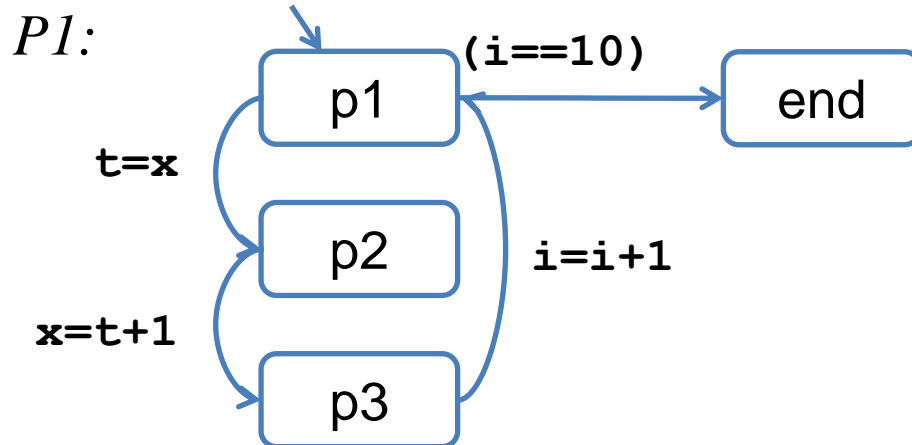
# Разделяемые переменные

Разделяемая (глобальная) переменная

`int x=0;`

```
P1: int t = 0, i = 0;  
repeat 10 times {  
  p11: t = x;  
  p12: x = t+1;  
  p13: i = i+1  
}
```

```
P2: int t = 0, i = 0;  
repeat 10 times {  
  p21: t = x;  
  p22: x = t+1;  
  p23: i = i+1  
}
```



Количество состояний общей системы переходов (без счетчиков):

$$\begin{aligned} |P_1| \times |P_2| &= \\ &= (2 \times 20)^2 \times 20 = 32 \times 10^5 \end{aligned}$$

- процессы выполняются асинхронно
- в распределенных системах отсутствуют предположения о скорости процессов

# Модель на Promela

```
#define N 10  
int x = 0;
```

объявление констант и  
глобальных переменных

```
active[2] proctype P() {
```

объявление процесса

непосредствен  
ный запуск  
процесса

тело процесса

создать 2 копии процесса

- **Процесс** – основная структурная единица языка Promela
  - в Promela нет функций
- В программе должен быть хотя бы один процесс

```
}
```



# Недетерминированный цикл

```
#define N 10  
int x = 0;
```

```
active[2] proctype P() {  
  int t = 0, i = 0;  
  do  
    :: i < N ->  
      t = x;  
      x = t + 1;  
      i ++  
    :: else -> break  
  od  
}
```

объявление локальных  
переменных

цикл

- Структура цикла

do

:: условие -> список команд

:: условие -> список команд

...

:: условие -> список команд

od

разделитель  
команд

# Вывод на экран

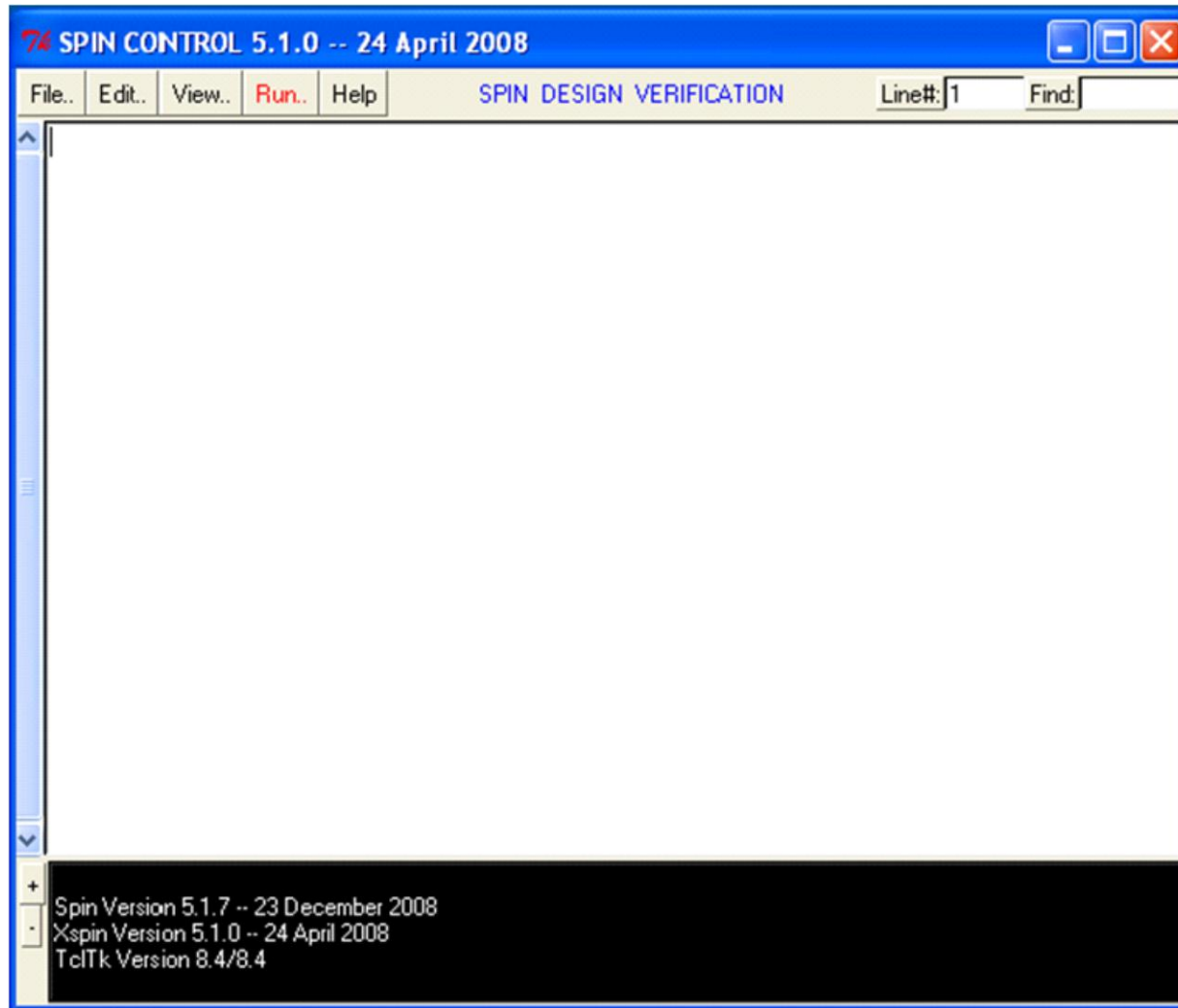
```
#define N 10
int x = 0;

active[2] proctype P() {
  int t = 0, i = 0;
  do
    :: i < N ->
      t = x; printf("MSC: t=%d", t);
      x = t + 1; printf("MSC: x=%d", x);
      i ++;
    :: else -> break
  od
}
```

# Запуск оболочки XSpin

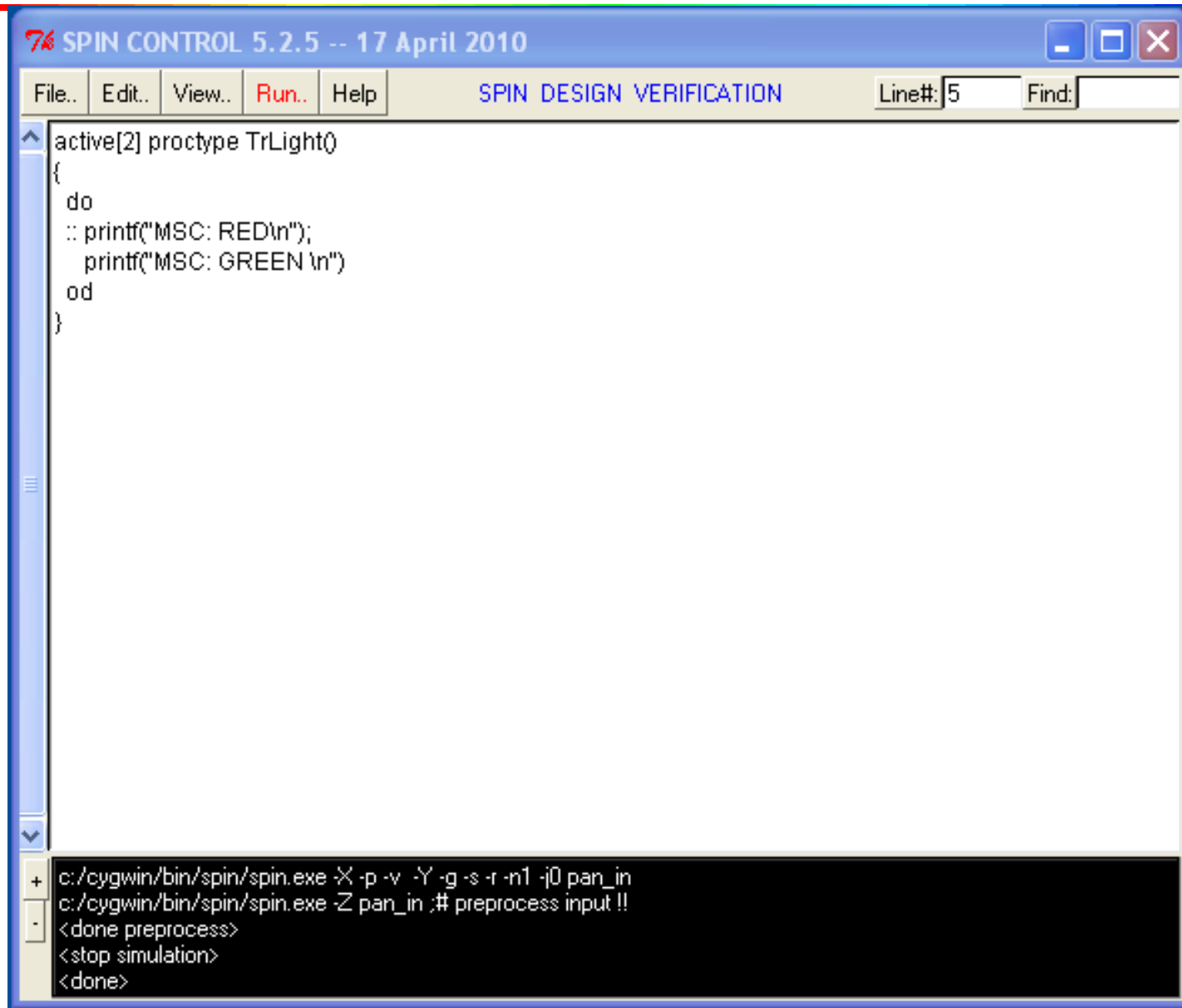
```
$ /bin/spin/xspin.tcl
```

запуск оболочки XSpin из Cygwin



Основное окно редактора XSpin

# Загрузка файла с моделью на Promela



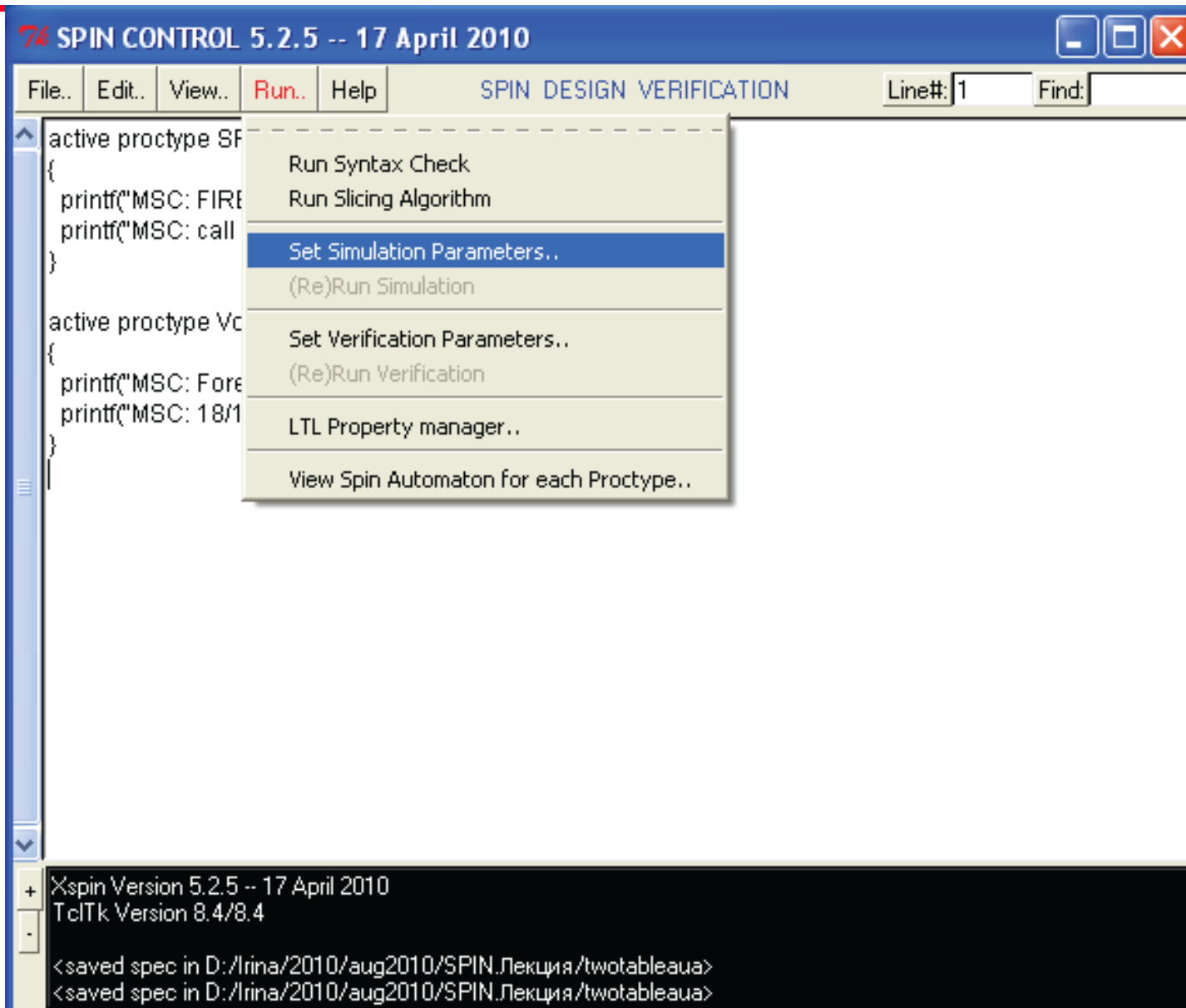
The screenshot shows the SPIN CONTROL 5.2.5 application window. The title bar reads "SPIN CONTROL 5.2.5 -- 17 April 2010". The menu bar includes "File..", "Edit..", "View..", "Run..", and "Help". The main menu is "SPIN DESIGN VERIFICATION". The "Line#" field is set to "5" and the "Find:" field is empty. The main text area contains the following Promela code:

```
active[2] proctype TrLight()
{
  do
  :: printf("MSC: RED\n");
  printf("MSC: GREEN\n")
  od
}
```

At the bottom, a command prompt window is open, showing the following commands and output:

```
c:/cygwin/bin/spin/spin.exe -X -p -v -Y -g -s -r -n1 -i0 pan_in
c:/cygwin/bin/spin/spin.exe -Z pan_in ;# preprocess input !!
<done preprocess>
<stop simulation>
<done>
```

# Запуск симуляции



# Окно параметров симуляции

**Simulation Options**

**Display Mode**

- ☒ MSC Panel - with:
  - ☒ Step Number Labels
  - ☐ Source Text Labels
  - ☒ Normal Spacing
  - ☐ Condensed Spacing
- ☐ Time Sequence Panel - with:
  - ☒ Interleaved Steps
  - ☐ One Window per Process
  - ☐ One Trace per Process
- ☒ Data Values Panel
  - ☒ Track Buffered Channels
  - ☒ Track Global Variables
  - ☐ Track Local Variables
  - ☒ Display vars marked 'show' in MSC
- ☐ Execution Bar Panel

**Simulation Style**

- ☒ Random (using seed)  
Seed Value
- ☐ Guided
  - ☒ Using pan\_in.trail
  - ☐ Use
- Steps Skipped
- ☐ Interactive

**A Full Queue**

- ☒ Blocks New Msgs
- ☐ Loses New Msgs

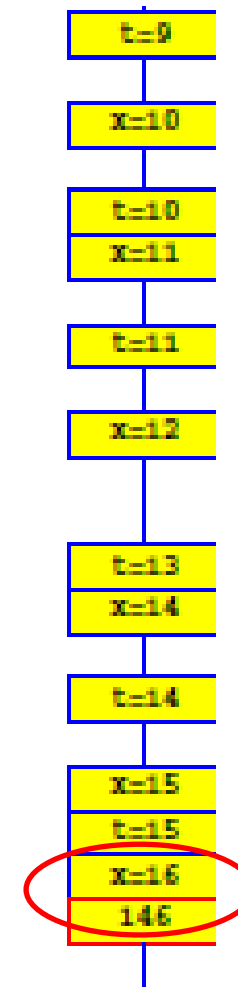
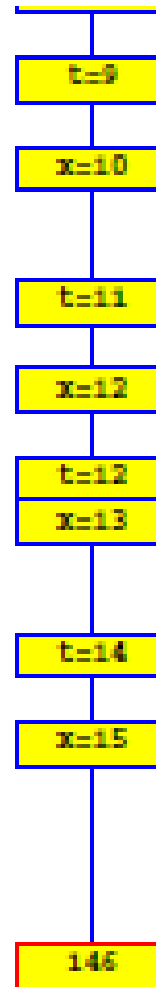
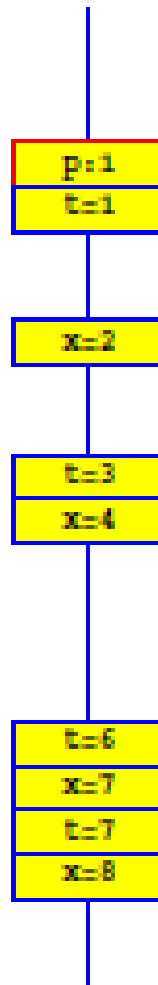
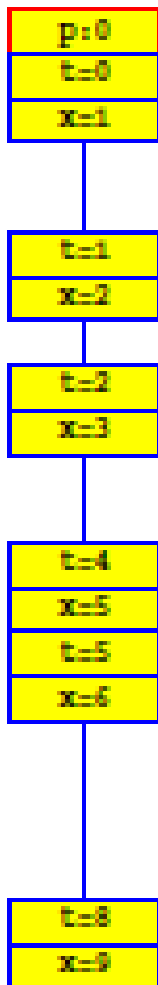
**Hide Queues in MSC**

Queue nr:

Queue nr:

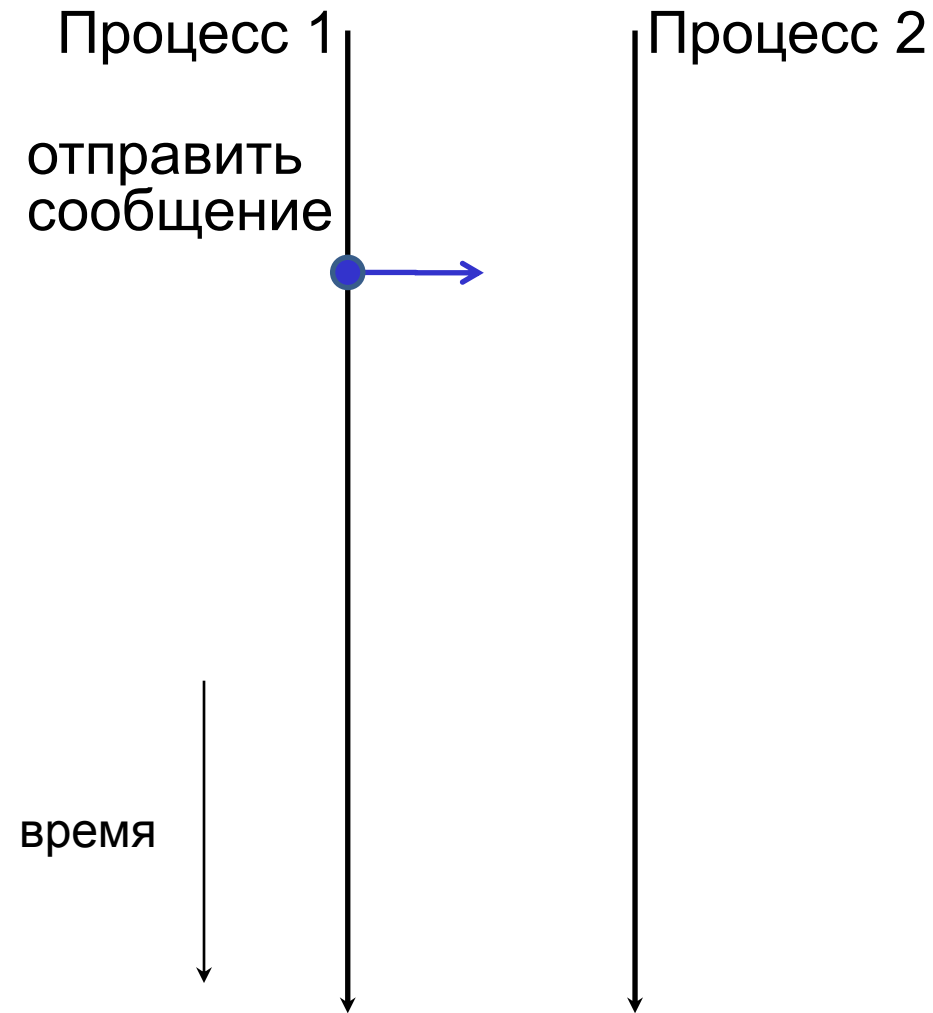
Queue nr:

# Симуляция модели



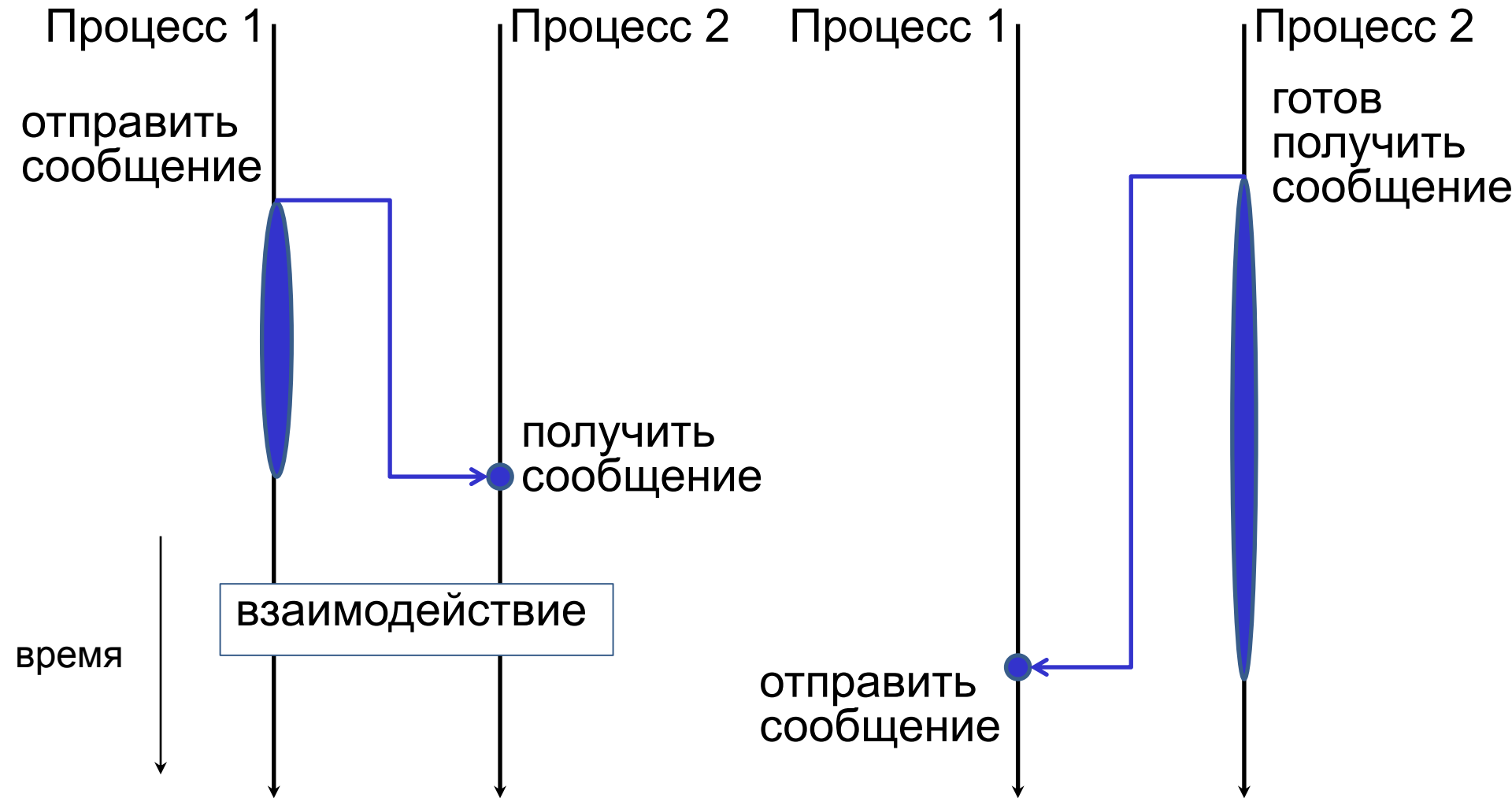
**$x=16!$**

# Синхронные взаимодействия





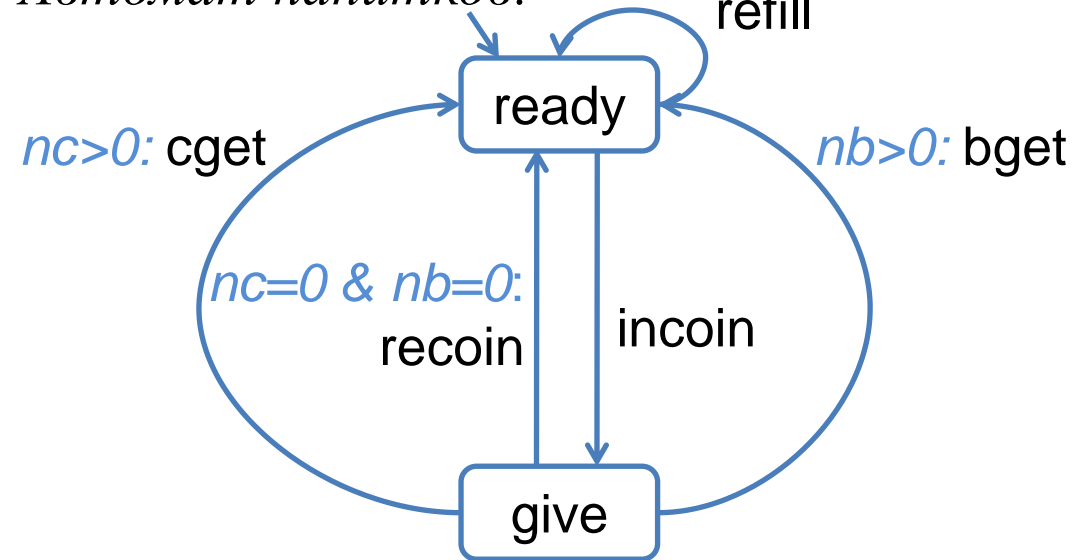
# Синхронные взаимодействия



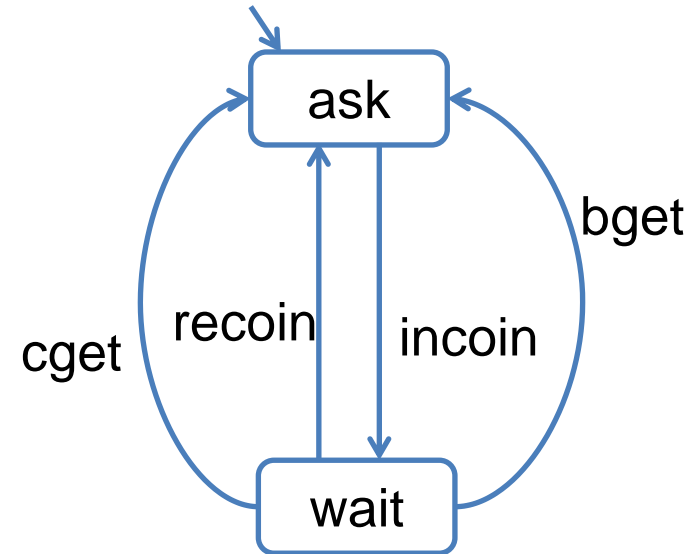
Синхронное взаимодействие называют рукопожатием (handshaking)

# Автомат напитков и студент

Автомат напитков:



Студент:



- *nc* – количество банок колы в автомате
- *nb* – количество банок пива в автомате

Сообщения в системе:

- *incoin* – в автомат бросили монету
- *recoin* – автомат вернул монету
- *cget* – автомат выдал колу
- *bget* – автомат выдал пива

Взаимодействия в системе  
синхронные

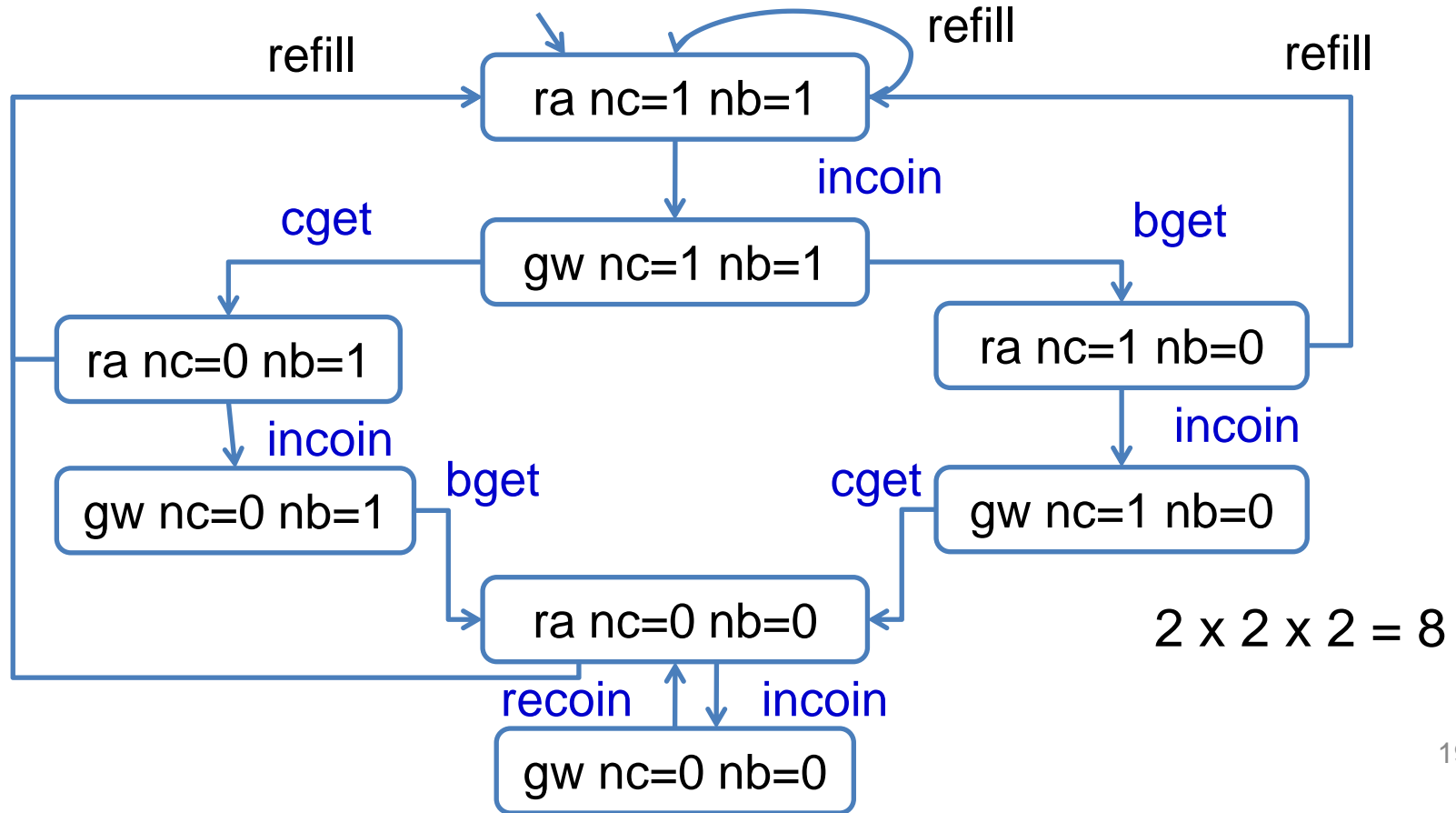
# Система переходов с синхронным взаимодействием

Сколько состояний в системе переходов «автомат напитков – студент» ?

Зависит от числа банок колы и пива!

Модели на языке Promela должны быть конечны

Ограничим:  $0 \leq nb \leq 1$      $0 \leq nc \leq 1$



# Автомат напитков и студент.

## Модель на Promela

- Поведение автомата напитков и студента моделируем процессами
- Ограничиваем количество банок напитков в автомате
- Синхронные взаимодействия – рандеву-каналы

```
#define NMAX 1
```

максимальное количество бутылок в автомате

```
mtype = {incoin, cget, bget, recoin};
```

сообщения системы

перечислимый тип, удобен для задания сообщений

```
chan mcch = [0] of {mtype};
```

объявление канала сообщений, исходящих от автомата напитков

емкость канала

формат сообщения

```
chan stch = [0] of {mtype};
```

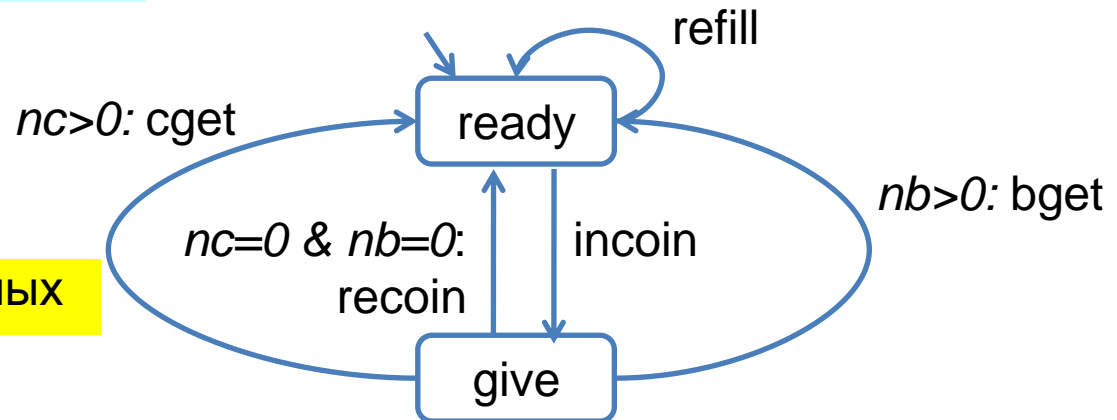
объявление канала сообщений, исходящих от студента

# Модель автомата напитков на Promela

```
active proctype Machine() {
```

```
  int nc = NMAX,  
      nb = NMAX,  
      state = 0;
```

объявление локальных переменных



guard – защита, условие

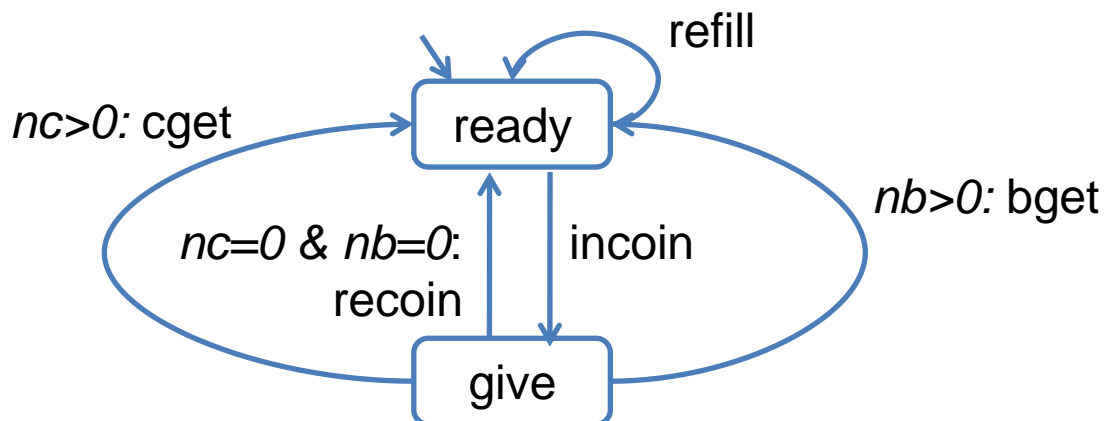
```
do  
  :: (state == 0) -> /*...*/  
  :: (state == 0) -> /*...*/  
  :: (state == 1) && (nc == 0) && (nb == 0) -> /*...*/  
  :: (state == 1) && (nc > 0) -> /*...*/  
  :: (state == 1) && (nb > 0) -> /*...*/  
od
```

ЦИКЛ

```
}
```

# Недетерминированный выбор в цикле

- Promela предоставляет простые механизмы обеспечения недетерминизма
- Если не выполняется ни одно из условий, процесс блокируется
- Операторы в Promela являются блокирующими



guard – защита, условие

```
do
  :: (state == 0) -> /*...*/
  :: (state == 0) -> /*...*/
  :: (state == 1) && (nc == 0) && (nb == 0) -> /*...*/
  :: (state == 1) && (nc > 0) -> /*...*/
  :: (state == 1) && (nb > 0) -> /*...*/
od
```

ЦИКЛ

# Операции получения и отправки сообщений в канал

```
active proctype Machine() {  
    int nc = NMAX, nb = NMAX, state = 0;  
    do  
        :: (state == 0) ->  
            nc = NMAX; nb = NMAX; printf("MSC: refilled\n")  
        :: (state == 0) ->  
            mcch ? incoin; state = 1  
        :: (state == 1) && (nc == 0) && (nb == 0) ->  
            stch ! recoin; state = 0  
        :: (state == 1) && (nc > 0) ->  
            stch ! cget; state = 0  
        :: (state == 1) && (nb > 0) ->  
            stch ! bget ; state = 0  
    od  
}
```

получить сообщение

отправить сообщение

# Структура выбора в Promela

## Модель на Promela поведения студента

```
active proctype Student() {  
  mtype msg;  
  int state = 0;  
  do  
    :: (state == 0) -> mcch ! incoin; state = 1  
    :: (state == 1) -> stch ? msg;  
    if  
      :: (msg == recoin) -> printf("MSC: try again\n")  
      :: (msg == cget) -> printf("MSC: get cola\n")  
      :: (msg == bget) -> printf("MSC: get beer\n");  
    fi;  
    state = 0  
  od  
}
```

guard – защита, условие

структура выбора



# Редактор LTL формул

При любом ли поведении системы когда-нибудь в будущем студент получит банку пива?

*getbeer* - атомарный предикат

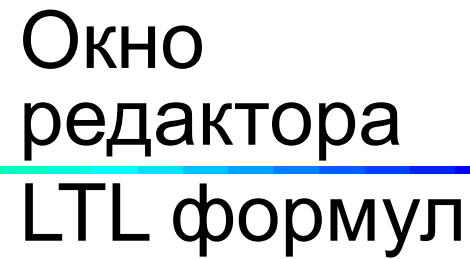
$F \text{ getbeer}$  – на всех путях когда-нибудь в будущем студент получит банку пива

Запись LTL формул в SPIN:

- <>     $F$  когда-нибудь в будущем на каком-нибудь пути будет выполняться свойство
- [ ]     $G$  всегда в будущем на всех путях будет выполняться заданное свойство
- !       $\neg$  отрицание

В строке редактора формул в SPIN запишем:

<> getbeer



## Запись формулы

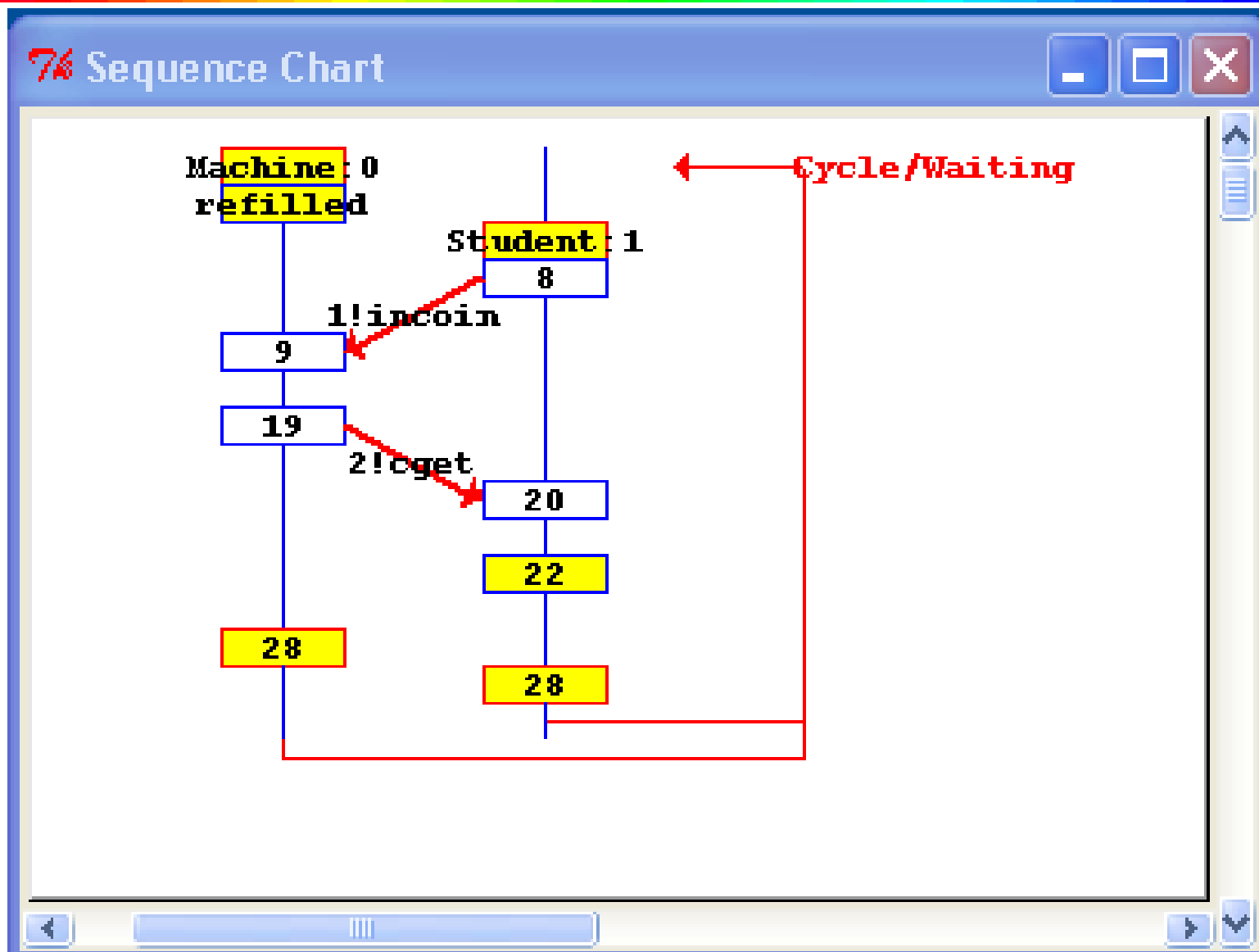
## Пропозициональная переменная

## Процесс never claim

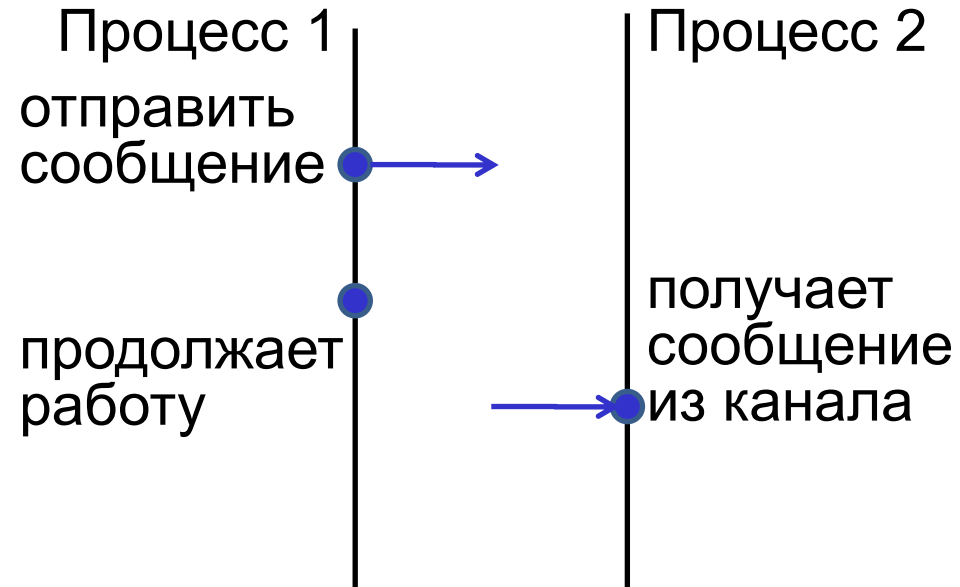
- Отрицание LTL формулы на языке Promela
- При верификации строится синхронная композиция процесса `never claim` и модели

## Результат верификации

# Окно диаграммы взаимодействия



# Асинхронные взаимодействия



- Порядок получения сообщений зависит от **дисциплины обслуживания канала**

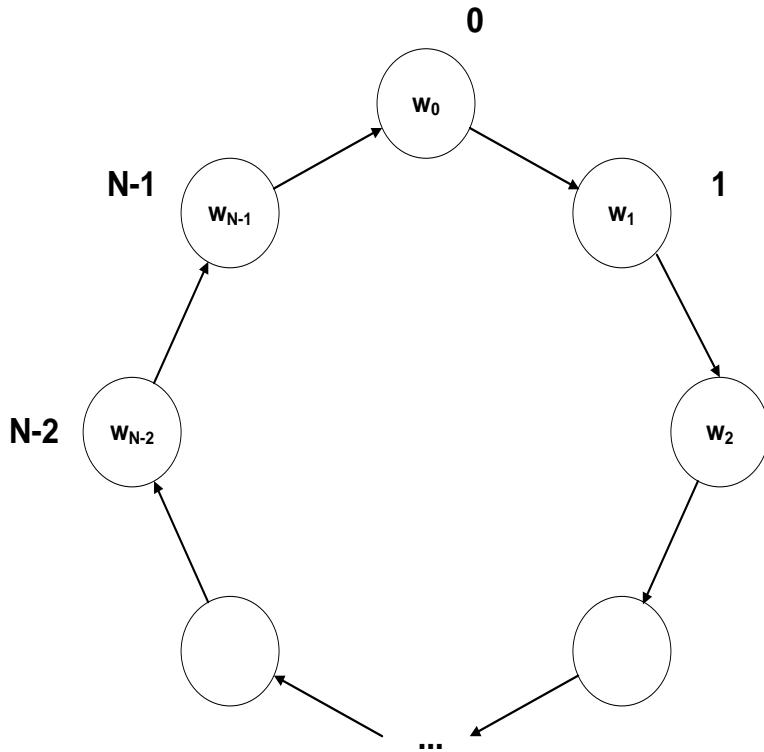
- В **Promela** дисциплина обслуживания канала FIFO
- В **Promela** Существуют механизмы сообщений в другом порядке

- Каналы имеют емкость
  - В **Promela** конечную емкость
- Сообщение может быть записано в канал, если емкость канала не исчерпана
- Процесс может прочитать сообщение из канала, если канал не пуст

# Задача выбора лидера

**Дано:** однонаправленное кольцо

- количество узлов  $N$
- веса узлов  $w_i$  ( $i=0..N-1$ ) уникальны
- узлы взаимодействуют только с соседями
- количество узлов фиксировано



**2 Требуется** построить протокол:  
набор **ЛОКАЛЬНЫХ** правил для  
каждого узла, которые позволят  
получить **ГЛОБАЛЬНЫЙ** результат -  
каждому узлу определить лидера

- например, узел с наибольшим весом

Эффективный алгоритм выбора лидера (Dolev-Klawe-Rodeh, Peterson)

количество сообщений –  $2N\log_2 N + O(N)$

# Алгоритм выбора лидера Петерсена

Цель: каждый узел должен определить максимальный вес в кольце

- В начале узел знает только свой вес

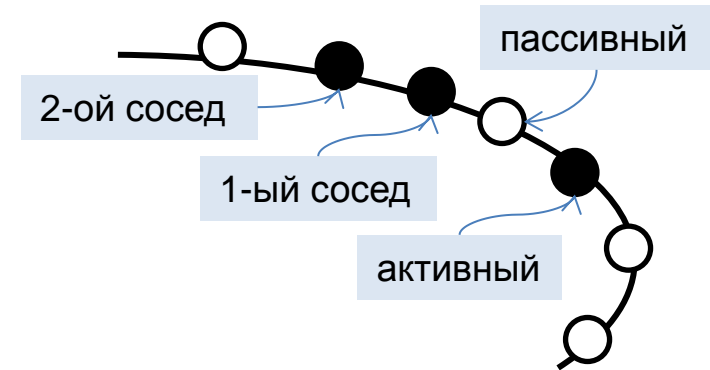
Каждый узел или *активный*,  
или *пассивный*

## Активный узел

- характеризуется **текущим весом**
- обрабатывает текущие веса **двух** ближайших активных соседей слева
- текущий вес – это **максимальный** известный узлу вес в сети
- формирует и отправляет сообщения (об известных ему текущих весах)

## Пассивный узел

- **не имеет** текущего веса
- **пропускает** через себя сообщения, не обрабатывая



# Набор локальных правил алгоритма Петерсона

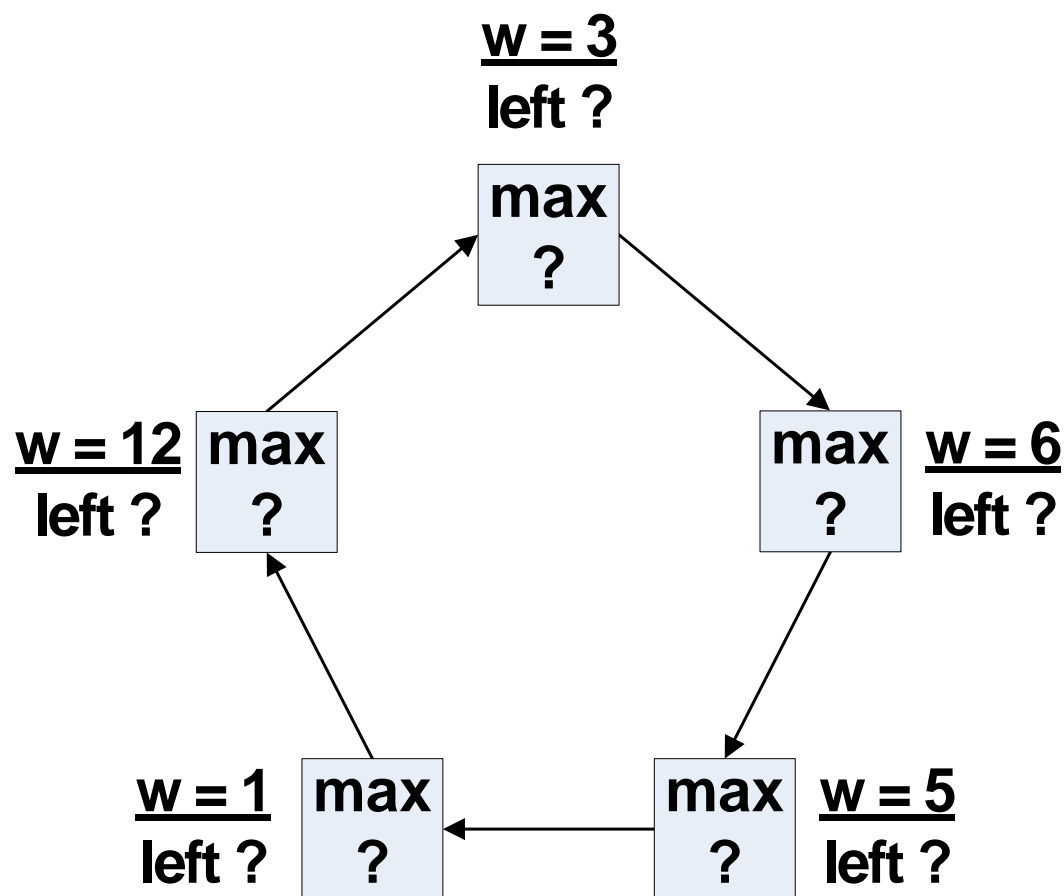
активного узла

- **max** – локальный максимум – свой текущий вес
- **left** – текущий вес активного соседа слева

```
A0. max =  $w_i$ 
    послать сообщение one(max)
A1. получить сообщение one(q)
    если (q != max) то
        left := q
        послать сообщение two(left)
    иначе послать сообщение winner(max)
    max является глобальным максимумом
A2. получить сообщение two(q)
    если (left > q) и (left > max) то
        max := left
        послать сообщение one(max)
    иначе узел становится пассивным
A3. фаза сообщения о лидере
```

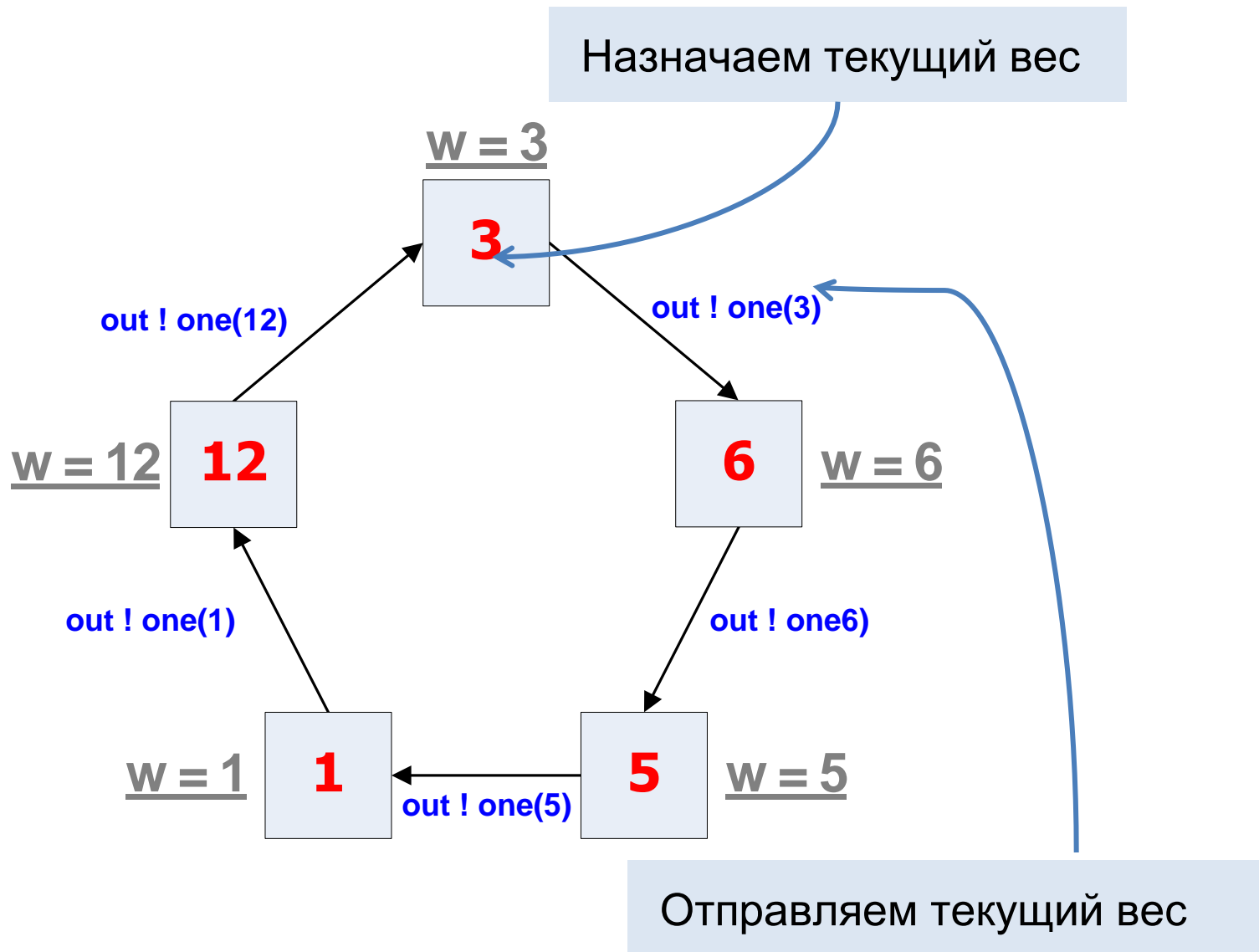
Фазы **A1**, **A2** сменяют друг друга, пока не будет найден лидер

# Пример. A0. Начальная фаза





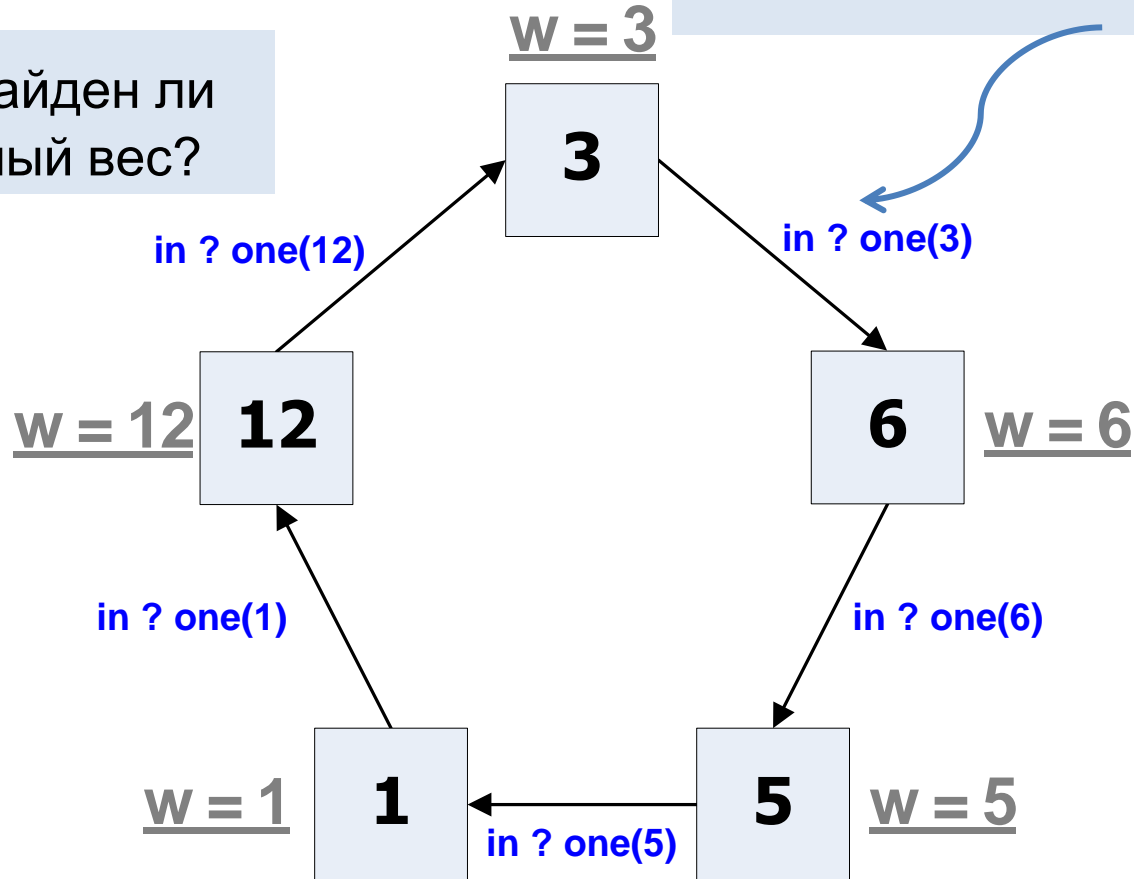
# Пример. A0. Начальная фаза



# Пример. Фаза A1

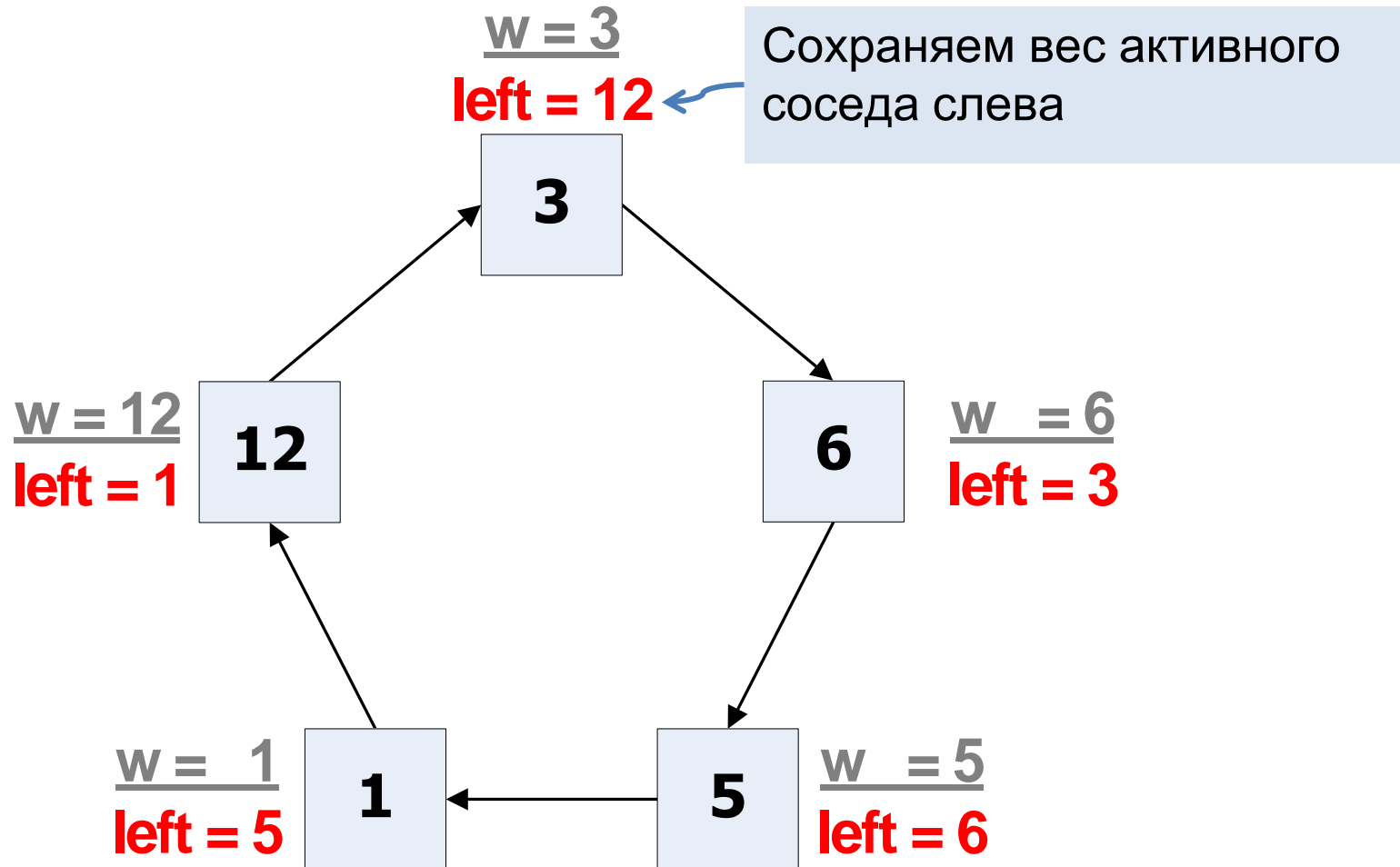
Проверяем: найден ли  
максимальный вес?

Получаем вес активного соседа

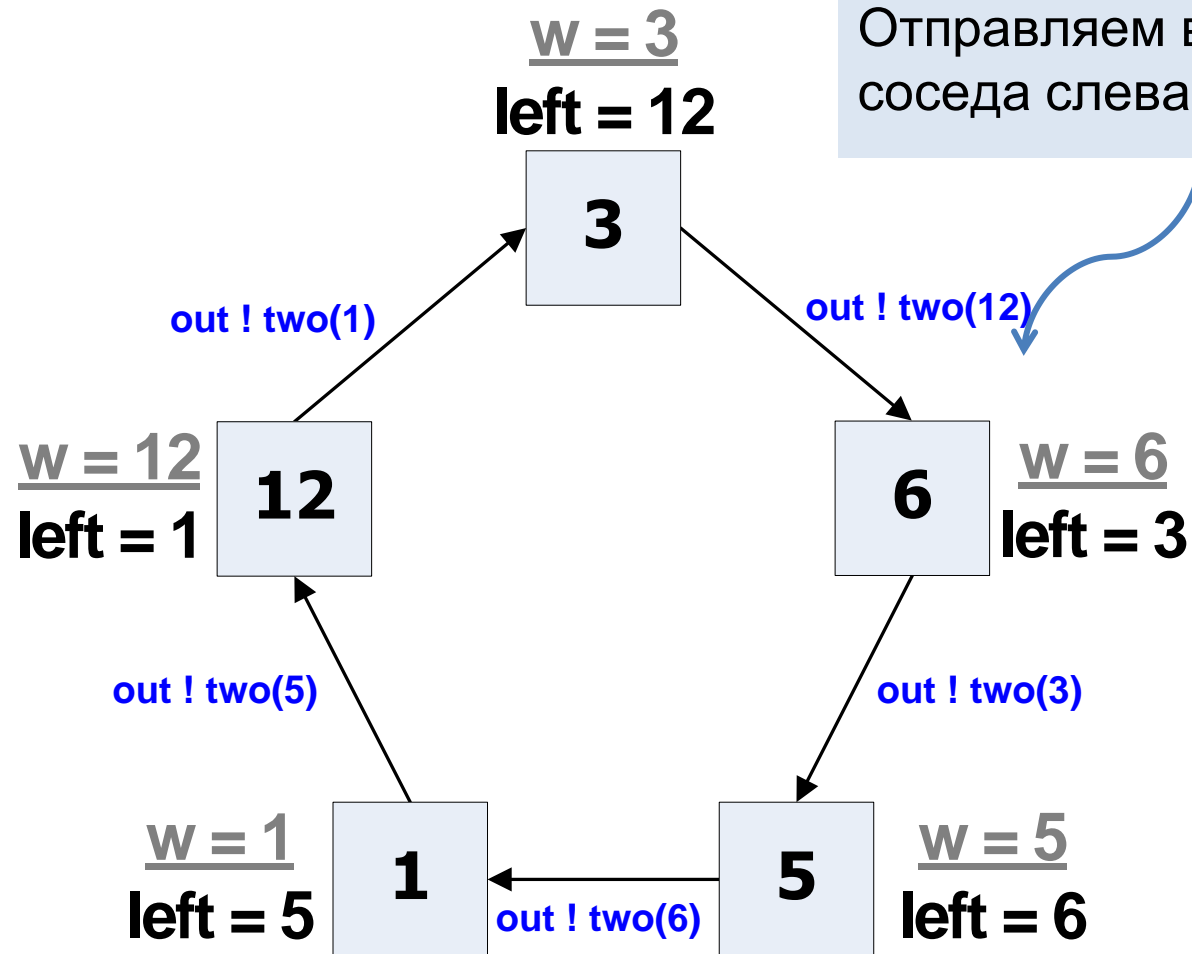


```
if (q != max) -> (left := q; send two(left))  
else победитель найден
```

# Пример. Фаза A1



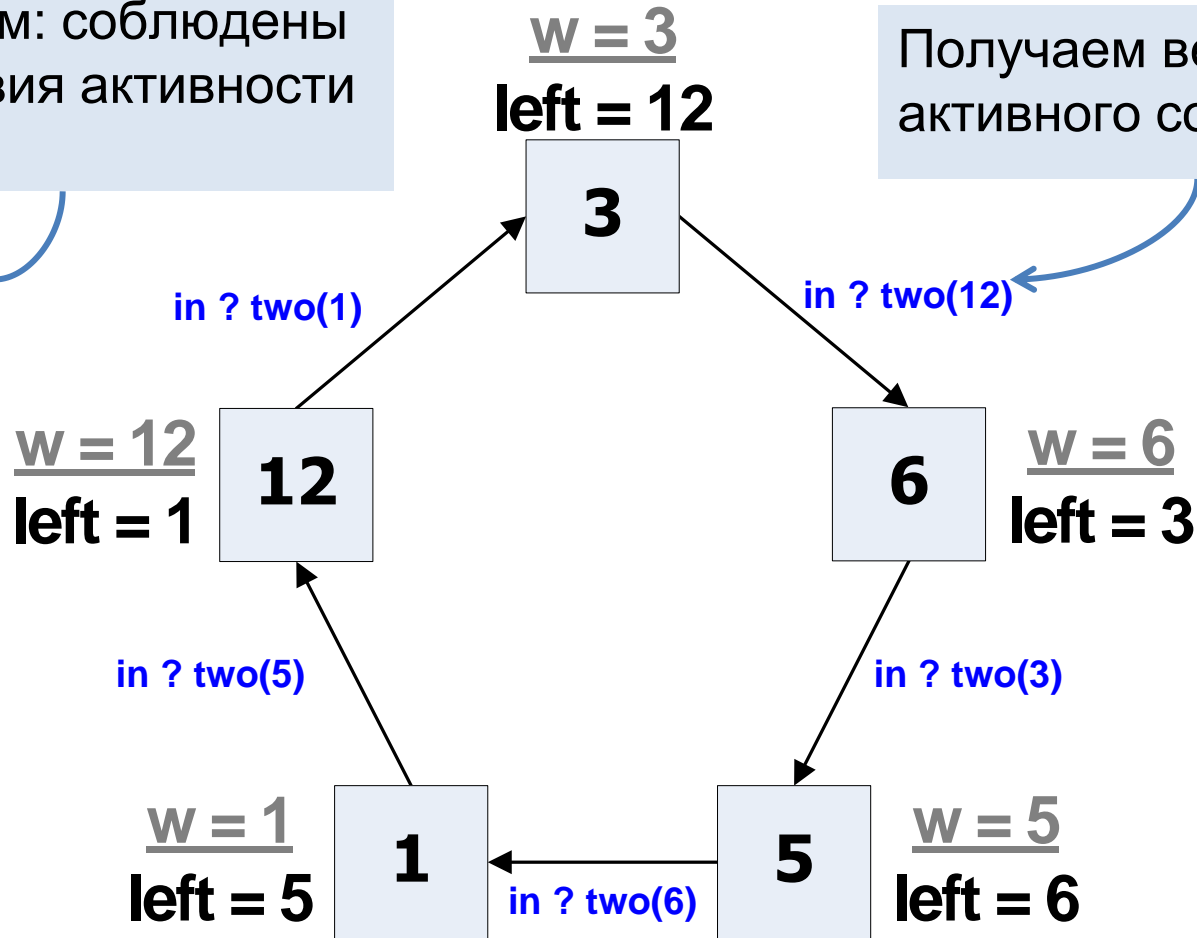
# Пример. Фаза A1



# Пример. Фаза A2

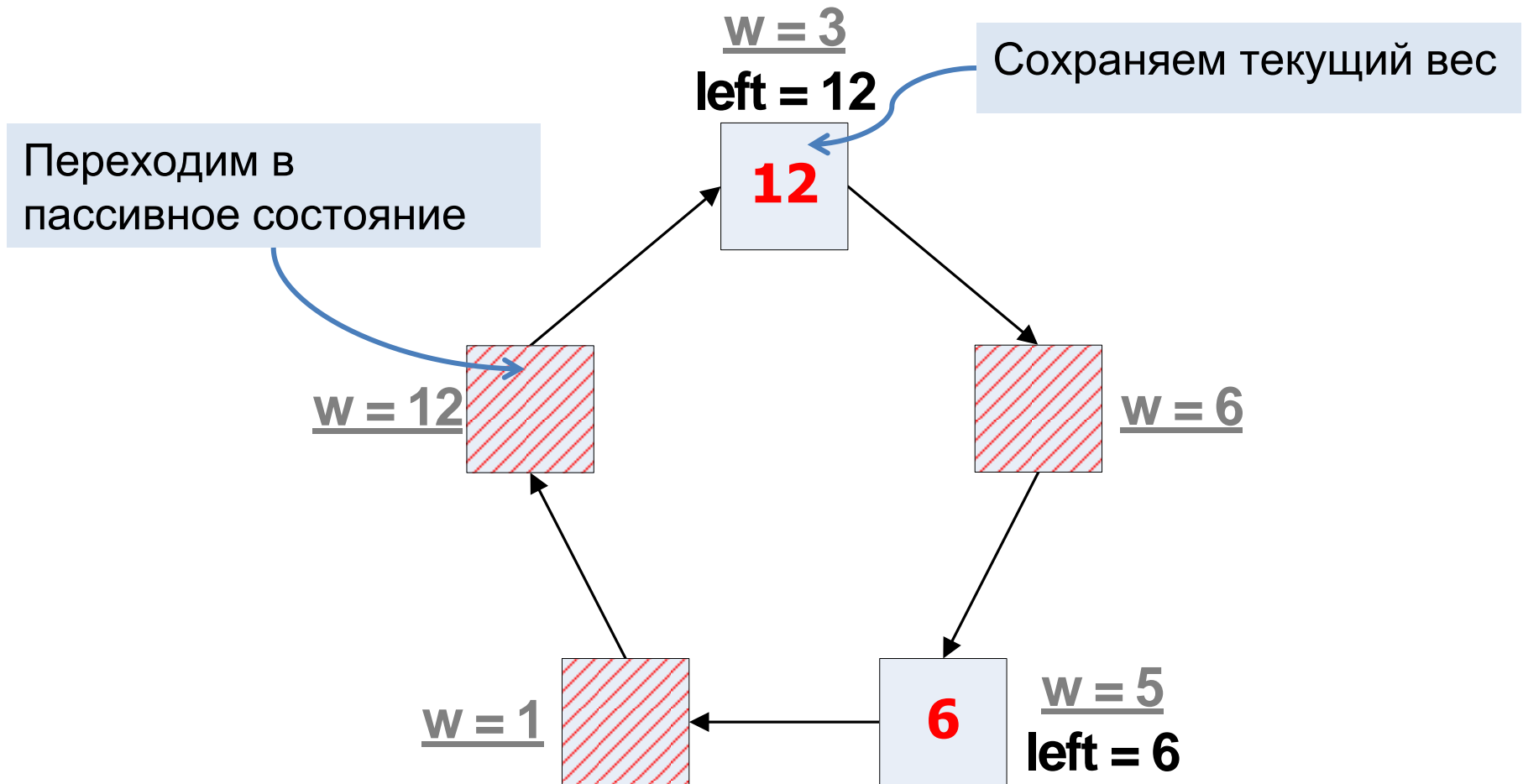
Проверяем: соблюдены ли условия активности узла?

Получаем вес активного соседа слева

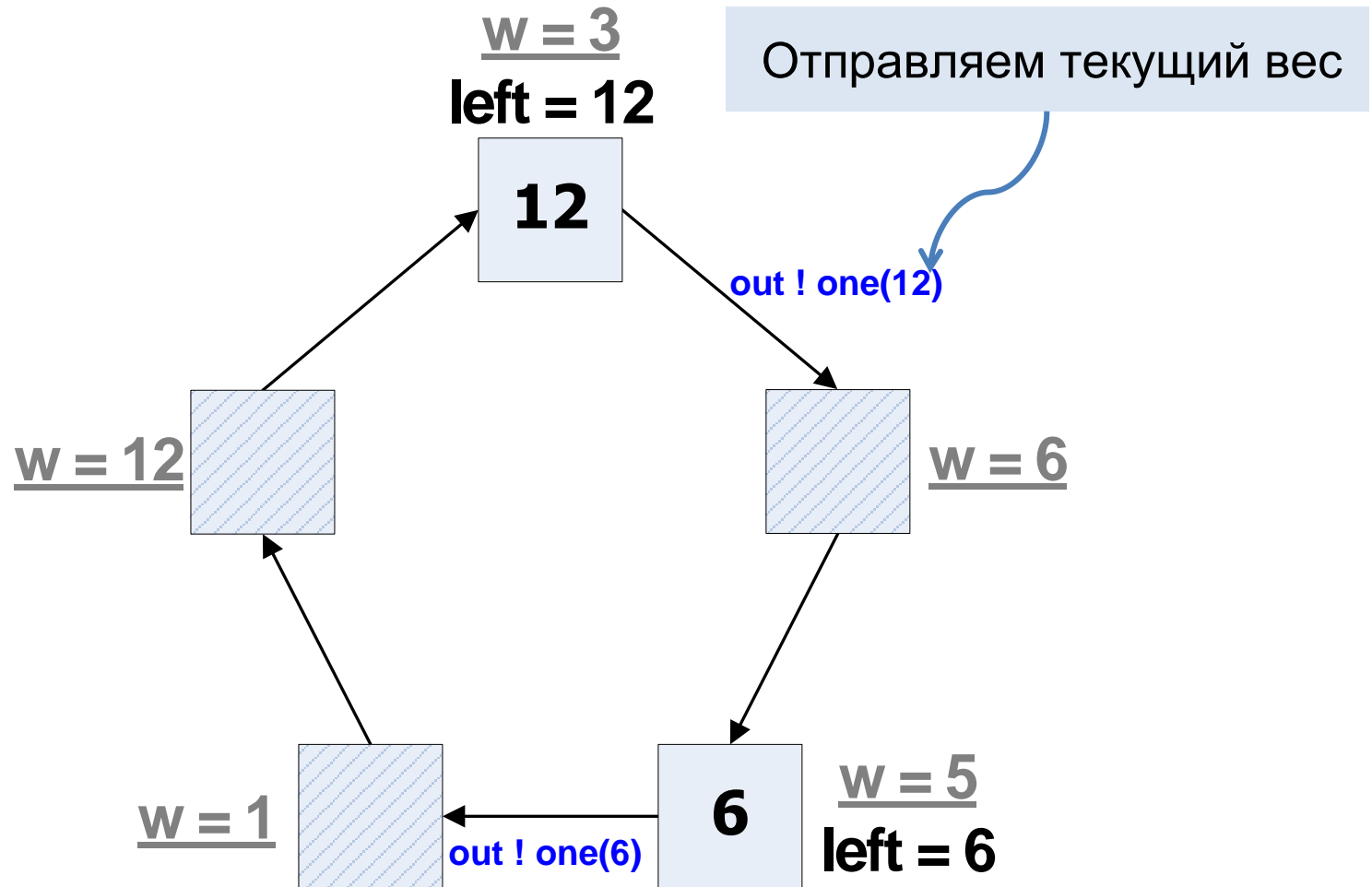


```
if((left>max) && (left>q)) -> (max:=left; send one(max))  
else узел стал пассивным
```

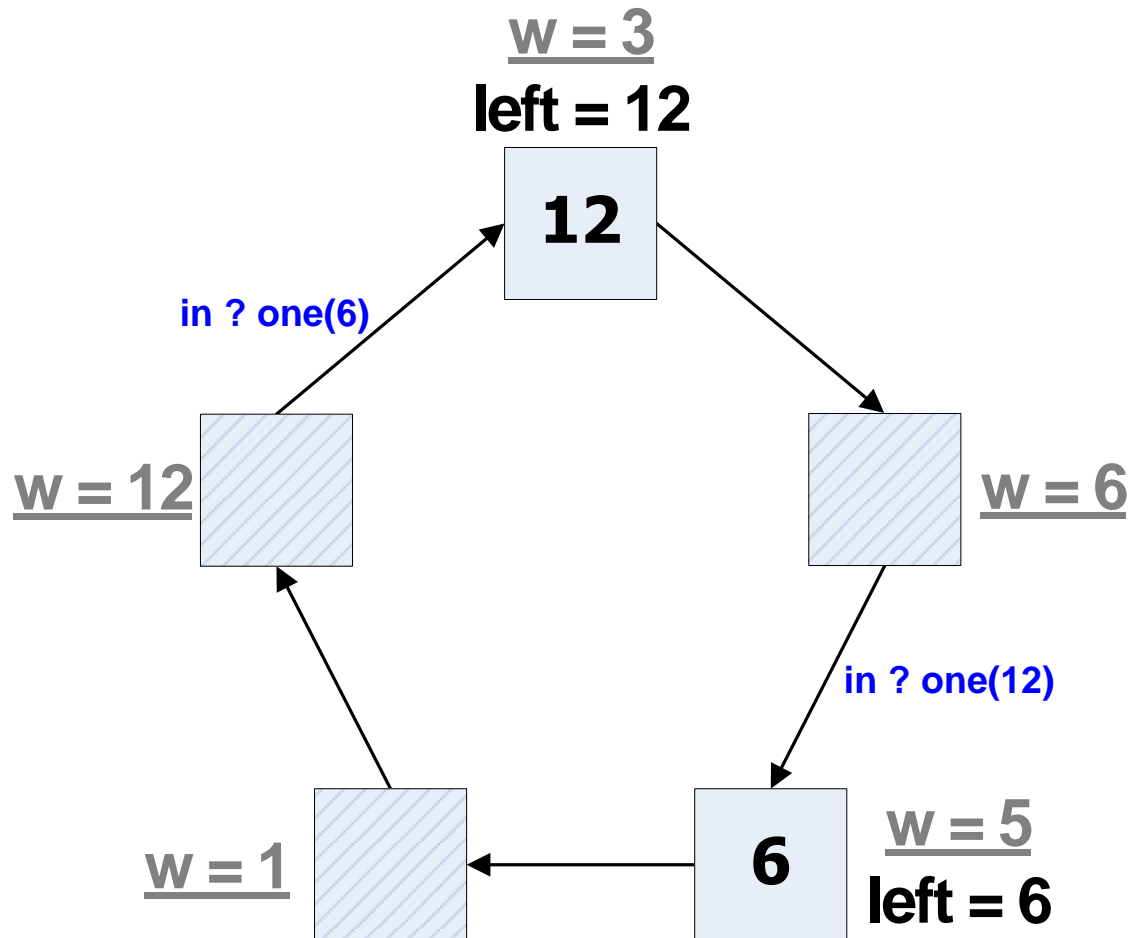
# Пример. Фаза A2



# Пример. Фаза A2

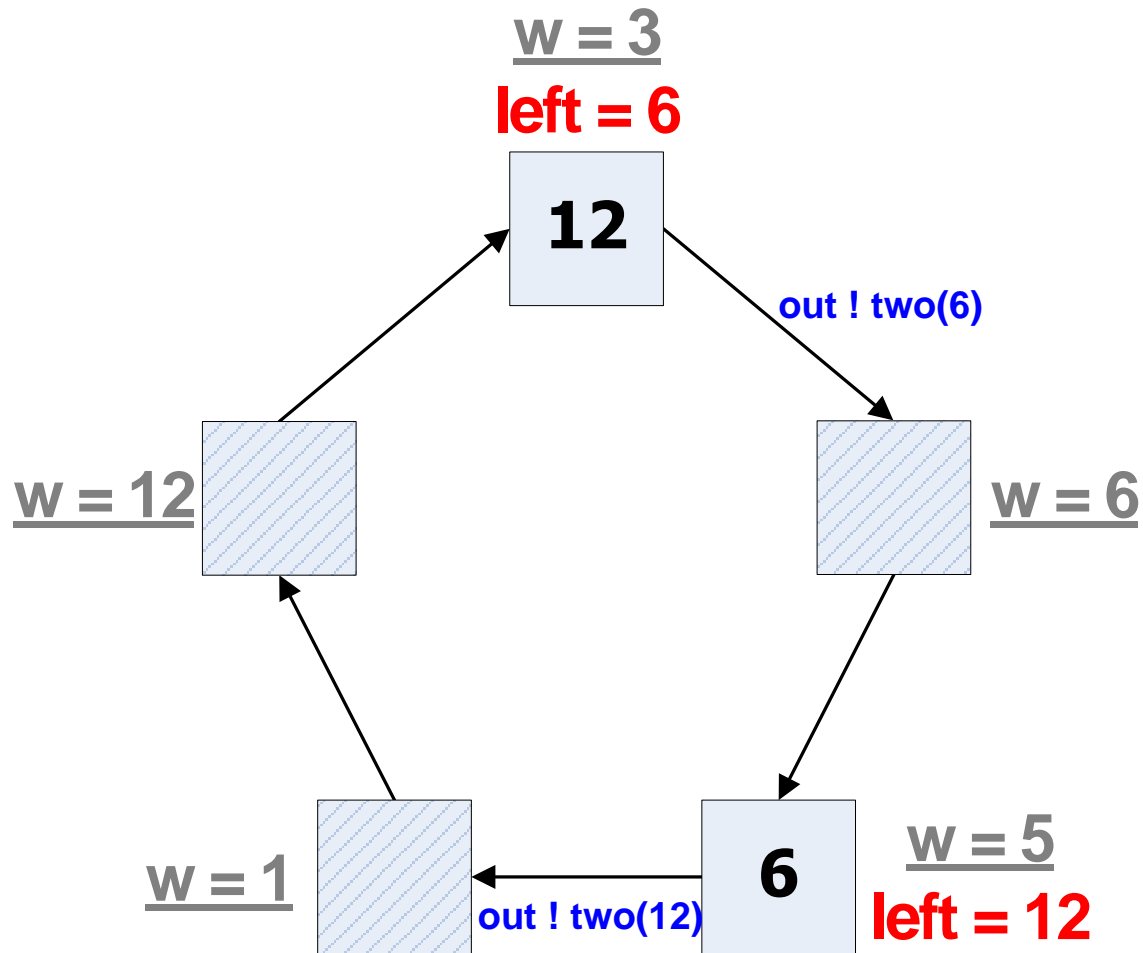


# Пример. Фаза А1

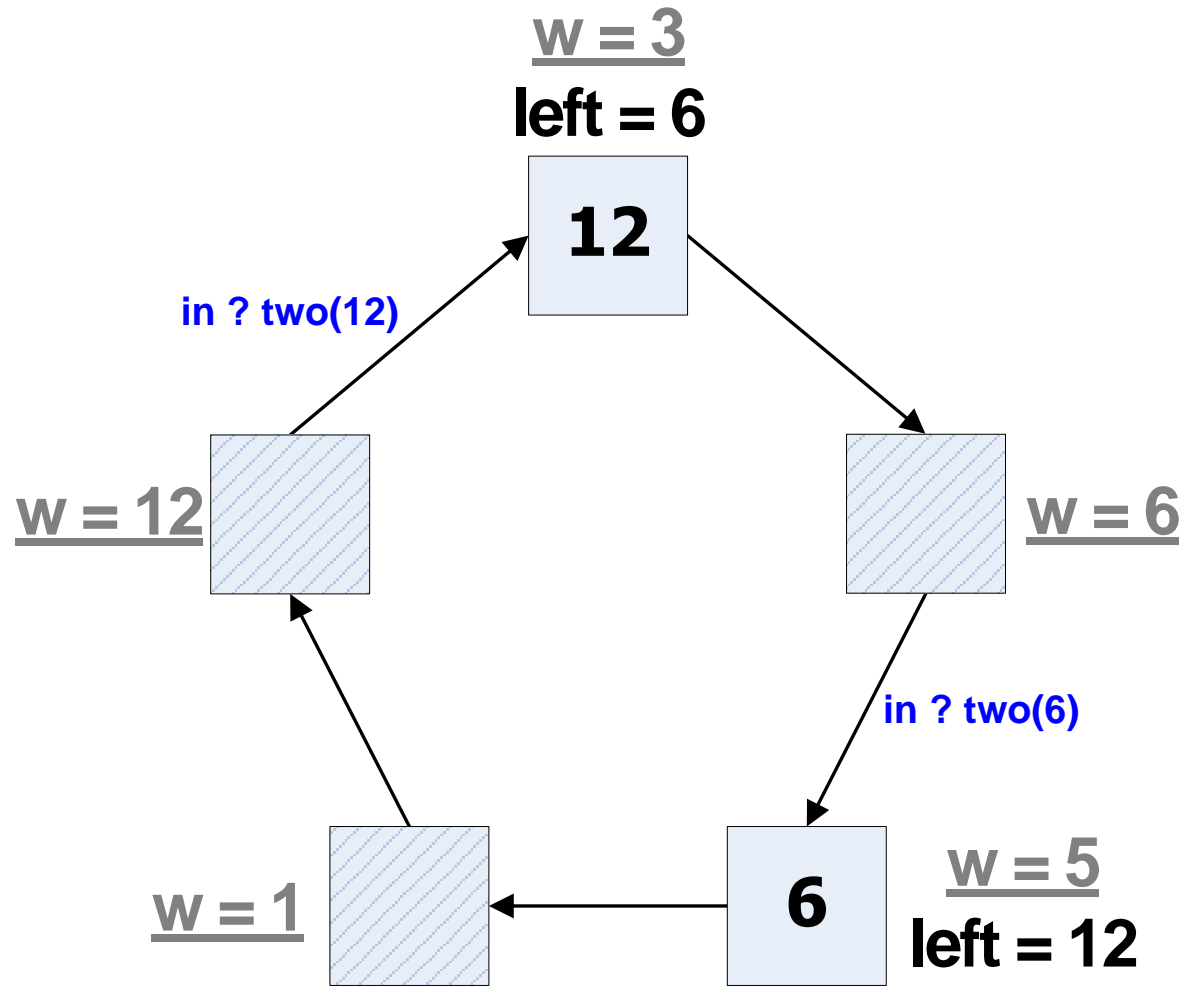




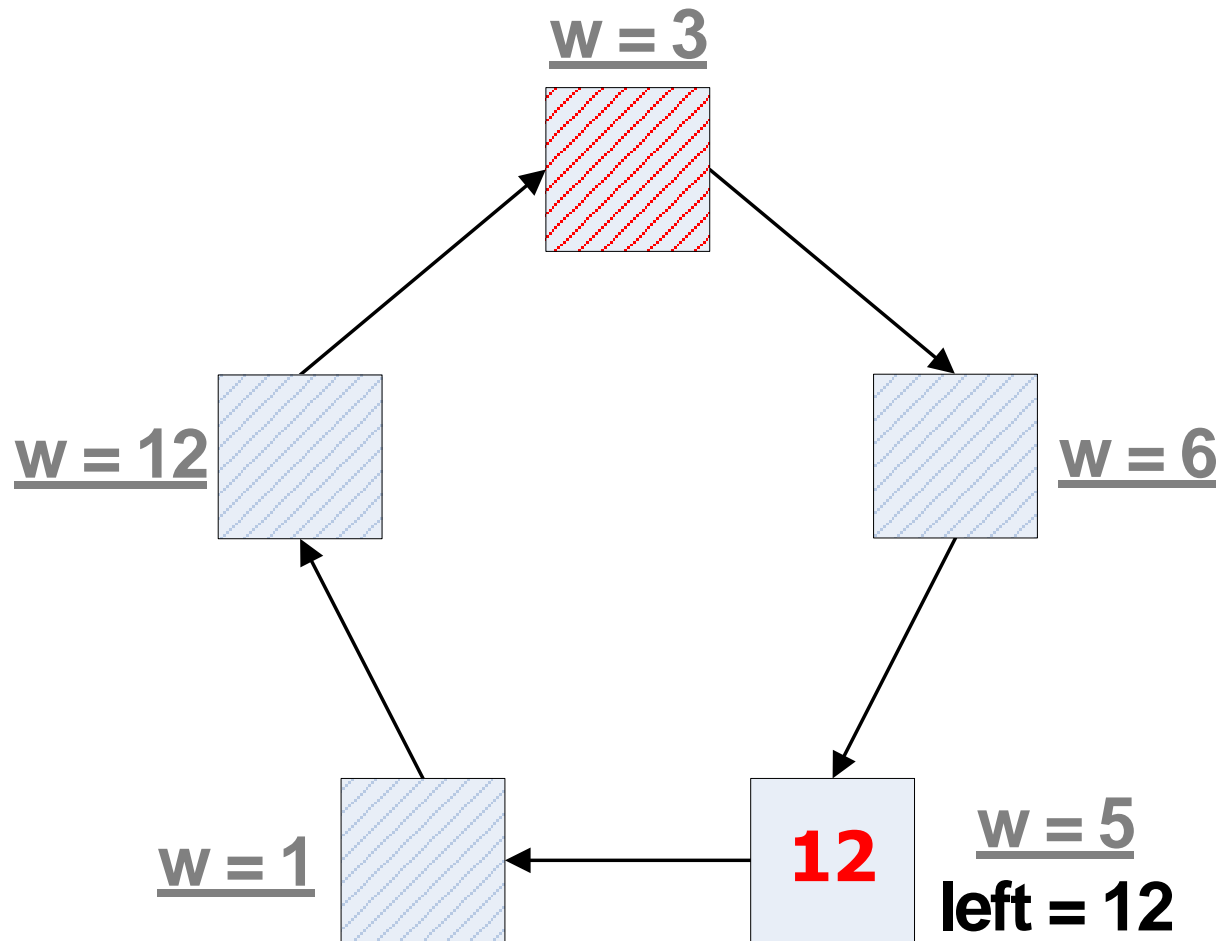
# Пример. Фаза A1



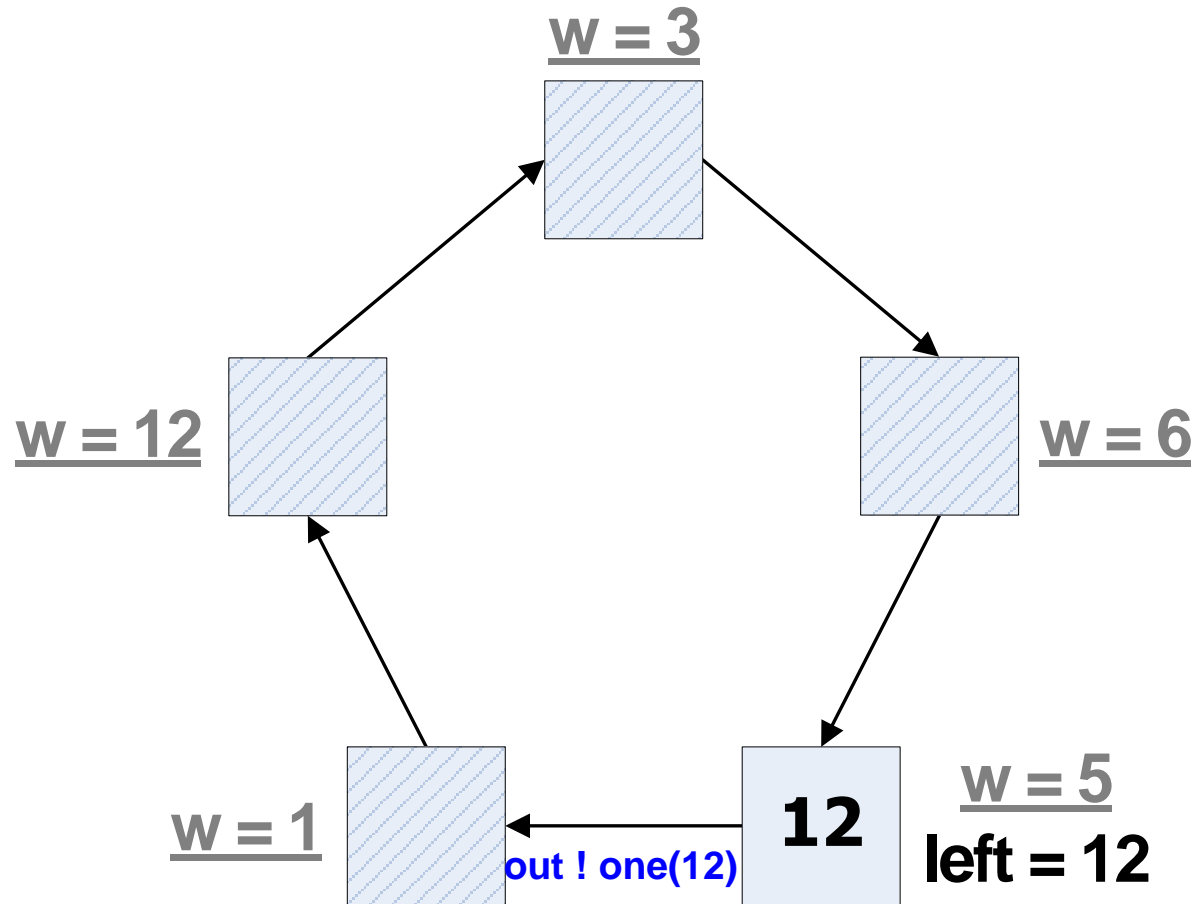
# Пример. Фаза A2



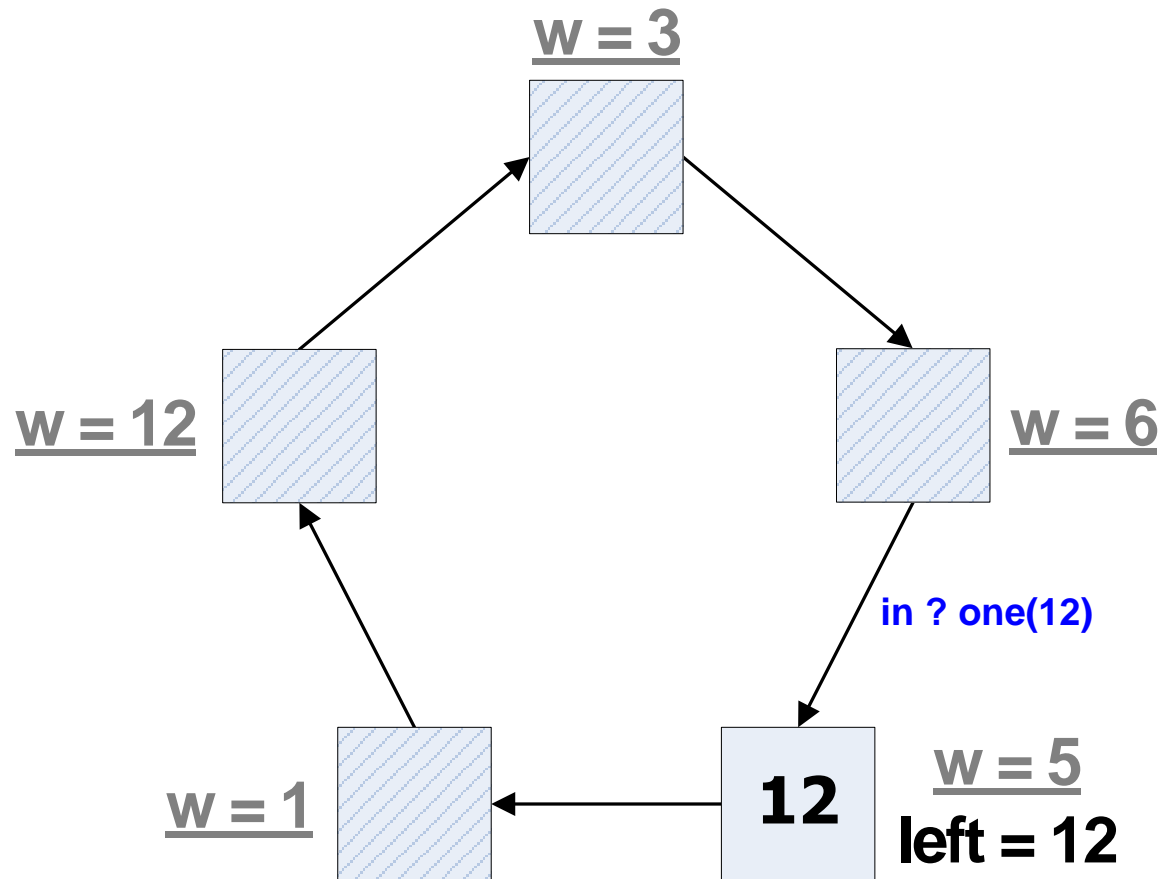
# Пример. Фаза A2



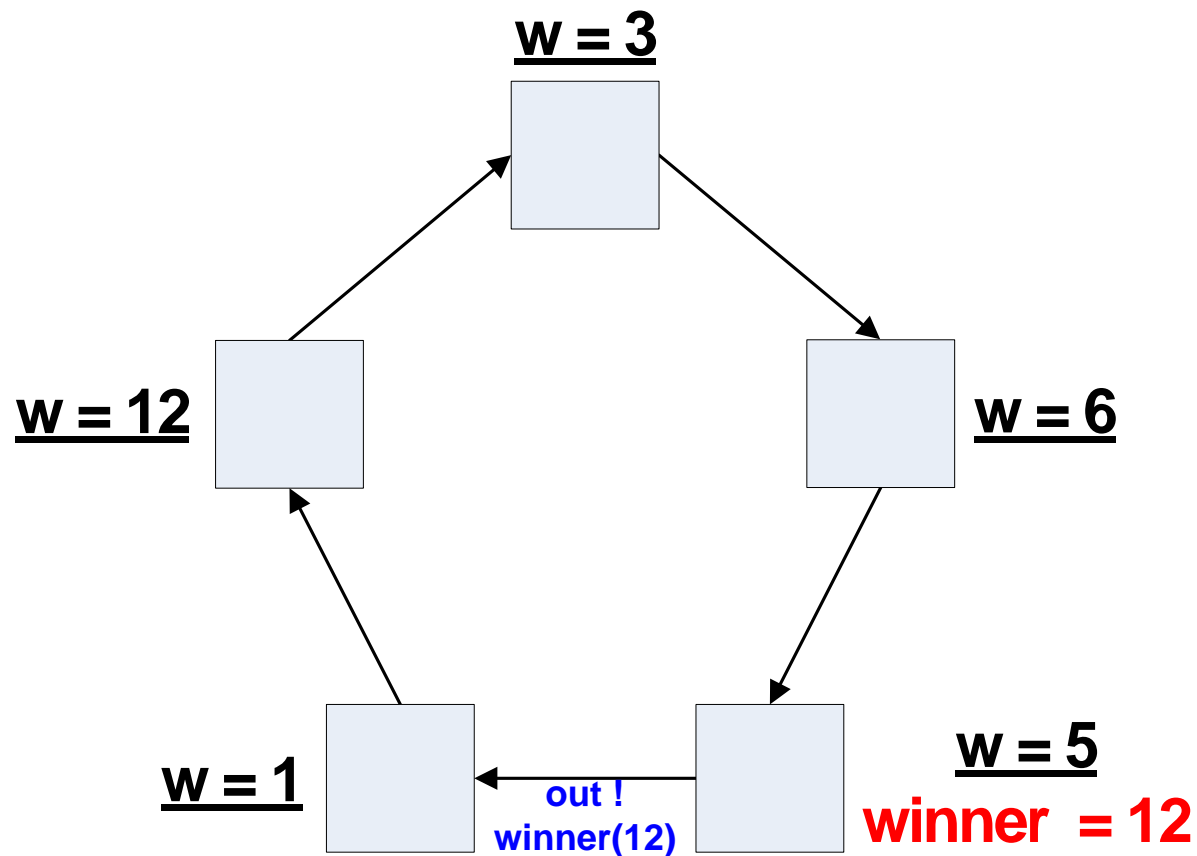
# Пример. Фаза A2



# Пример. Фаза A1



# Пример. Фаза А3. Лидер найден



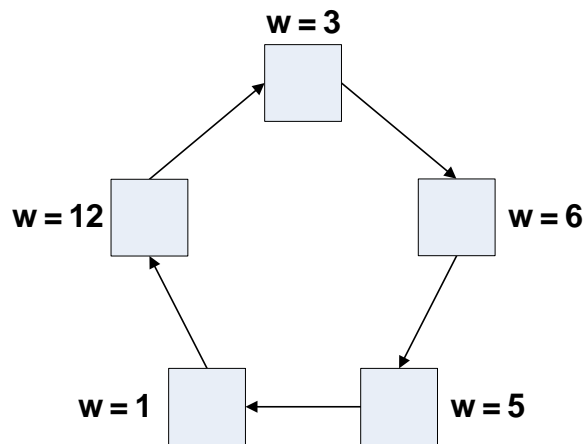
# Глобальные свойства, обеспечивающие корректность алгоритма

- На любой фазе в кольце есть только один процесс с максимальным весом
- На любой фазе активный процесс характеризуется текущим весом, который является наибольшим из двух активных соседей слева

Эти два правила являются *глобальными инвариантами*

Формулировка глобальных инвариант и формальное доказательство требуют **понимания алгоритма**

# Модель задачи о выборе лидера на языке Promela



Ограничения модели:

- Количество узлов фиксировано и ограничено
- Все узлы вступают одновременно
- Количество узлов не меняется в процессе работы

```
# define N      5
```

количество узлов

Каждый узел – независимый процесс:

```
proctype node (...)
```

объявление процесса

Узлы обмениваются сообщениями трех типов:

```
mtype = {one, two, winner};
```

Узлы обмениваются сообщениями по каналам:

```
# define L      2
```

емкость канала

```
chan p[N] = [L] of {mtype, byte};
```

формат сообщения,  
например, `one(q)`  
то же, что и `one, q`

объявление массива каналов



# Структура программы на Promela

```
// Число процессов
# define N      5

// Ограничение глубины канала
# define L      2

// Типы сообщений
mtype = {one, two, winner};

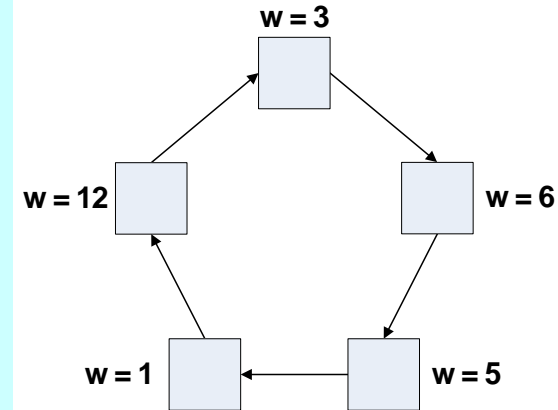
// Объявление N каналов глубиной L
chan p[N] = [L] of {mtype, byte};

// Количество лидеров
byte nr_leaders = 0;

// Объявления процессов-узлов
proctype node (chan in, out; byte my_number) {
  /* . . . */
}

// Главный процесс, запуск всех процессов узлов
init {
  /* . . . */
}
```

Глобальные переменные



# Описание алгоритма работы узла

```
proctype node (chan in, out; byte my_number)
{
    bit Active = 1,
        know_winner = 0; // флаг знаю-лидера
    byte q,
        max = my_number,
        left;
    out ! one(my_number); // A0. отправить свой параметр
end: do
    :: in ? one(q) -> /*...*/ // A1. получено сообщение one

    :: in ? two(q) -> /*...*/ // A2. получено сообщение two

    :: in ? winner(q) -> /
        break;
    od
}
```

**A0.** Начальная фаза

**A1.** Получить сообщение one(q).  
Обработать сообщение. Отправить сообщение two(q)

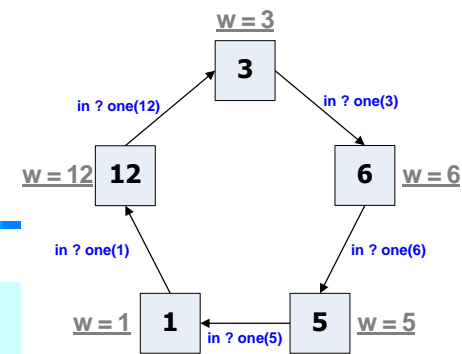
**A2.** Получить сообщение two(q).  
Обработать сообщения. Отправить сообщения one(q)

**A3.** Обработка сообщения о лидере winner(q)

# Конструкции языка Promela, связанные с верификацией

- Утверждение **assert** (любое\_булево\_условие)
  - Если условие не всегда соблюдается, то оператор вызовет сообщение об ошибке в процессе симуляции и верификации с помощью Spin
- Метка конечного состояния (**end**)
  - указывает верификатору, чтобы тот не считал определенные операторы некорректным завершением программы
- Метка активного состояния (**progress**)
  - помечает операторы, которые для корректного продолжения работы протокола должны встретиться хотя бы раз на любой бесконечной трассе

# Фаза A1



```
:: in ? one(q) ->
```

```
if
```

```
:: Active ->
```

// узел в активном состоянии

```
if
```

```
:: q != max ->
```

// проверяем полученное значение

// если не равно лок. максимуму

```
left = q;
```

// то меняем параметр соседа

```
out ! two(q)
```

// передаем параметр далее

```
:: else ->
```

// иначе - нашли глоб. максимум

```
assert(q == N);
```

```
know_winner = 1;
```

// лидер этому узлу известен

```
out ! winner(q);
```

// сообщаем о выборе лидера

```
fi
```

```
:: else ->
```

```
//
```

```
out ! one(q)
```

```
//
```

```
fi
```

A1. Получить сообщение one(q):

1. Если  $q \neq \max$ , то  $\text{left} := q$  и послать сообщение  $\text{two}(\text{left})$

2. Иначе,  $\max$  является глобальным максимумом

# Фаза A2

```
:: in ? two(q) ->

if
:: Active ->           // в активном состоянии

    if                // находимся за локальным максимумом
    :: left > q && left > max ->
        max = left;    // меняем информацию о локальном
                        // максимуме
        out ! one(max) // передаем дальше

    :: else ->         // переход в пассивное состояние
        Active = 0
fi

:: else -> // пассивны - передаем параметр без обработки
    out ! two(q)

fi
```

A2. Пришло сообщение two(q):

1. Если left больше и q, и max, то  
max:=left и  
послать сообщение one(max)
2. Иначе, узел становится пассивным.

# Фаза A3. Обработка сообщения о лидере

```
:: in ? winner(q) -> // получено сообщение «лидер выбран»

if // проверка: совпадает ли номером узла
:: q != my_number ->
    printf("MSC: LOST\n"); // узел проиграл выборы

:: else -> // узел выиграл выборы
    printf("MSC: LEADER\n");
    nr_leaders++;
    assert(nr_leaders == 1)
fi;

if // проверка: был ли найден лидер?
:: know_winner // знал и уже посылал сообщение –
                // «лидер выбран»
:: else -> out ! winner(q) // посылает сообщение
fi;

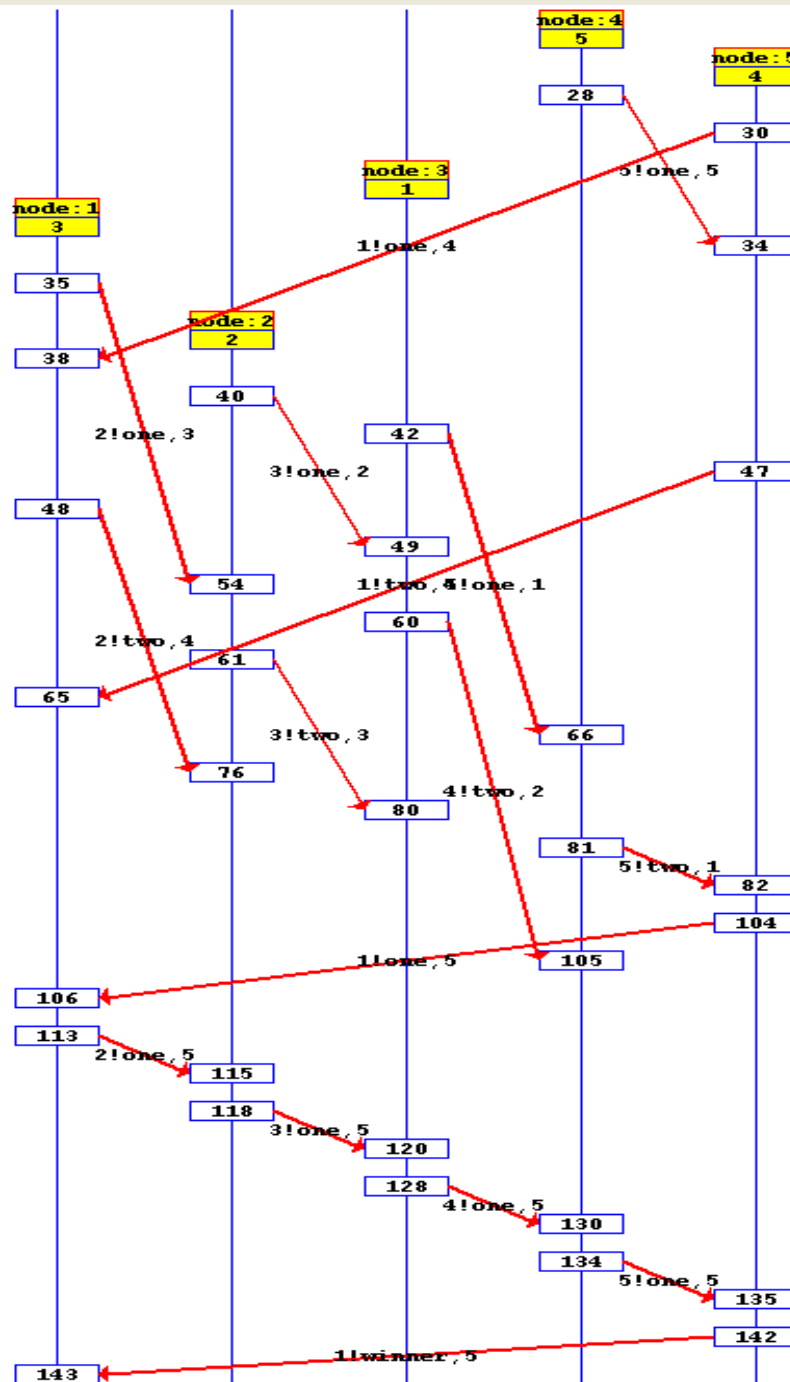
break
```

Некоторые операторы языка Promela всегда выполнимы.  
Например, `break`, `skip`, `printf`

# Основная функция, запускающая процессы

```
init {  
    byte proc;  
    atomic { // рассматривается верификатором,  
            // как одно действие  
  
        proc = 1;  
  
        do  
        :: proc <= N ->  
            run node (p[proc-1], p[proc%N], (N+1-proc)%N+1);  
            proc++  
  
        :: proc > N ->  
            break  
        od  
    }  
}
```

Условие модели – одновременный запуск процессов!



Симуляция  
программы  
выбора лидера  
в XSPIN



# Количество состояний с асинхронными взаимодействиями

- $n$  • количество процессов
- $Chan$  • множество каналов
- $Label_i$  • множество меток процесса  $i$
- $Var_i$  • множество переменных процесса  $i$
- $dom(x)$  • область определения переменной  $x$
- $dom(c)$  • область определения канала  $c$
- $cap(c)$  • емкость канала  $c$

$$\prod_{i=1}^n \left( |Label_i| \cdot \prod_{x \in Var_i} |dom(x)| \right) \cdot \prod_{c \in Chan} |dom(c)|^{cap(c)}$$

## Использование асинхронных каналов

- увеличивает размер вектора глобального состояния системы
- увеличивает число состояний системы

# Верификация алгоритма выбора лидера

Алгоритм нетривиальный. Основная цель алгоритма – эффективность на большом числе процессов.

Как можно проверить, что алгоритм корректный?

Строить LTL формулы и проверять их выполнение с помощью Spin

Проверяемые свойства:

- лидер должен быть только один

```
noMore: nr_leaders ≤ 1      // атомный предикат  
G noMore
```

- лидер в конце концов будет выбран

```
elected: nr_leaders == 1    // атомный предикат  
FG elected
```

- номер выбранного лидера всегда будет максимальным

```
nr == N      // атомный предикат  
FG nr
```

**LTL-формула выполняется для любого пути, стартовавшего в допустимом начальном состоянии**

# Linear Time Temporal Logic Formulae

Formula:  $\langle \rangle [] \text{oneLeader}$

Load...

Operators:  $[]$   $\langle \rangle$   $U$   $\rightarrow$   $\text{and}$   $\text{or}$   $\text{no}$

Property holds for: ☒ All Executions (desired behavior) ☐ No Executions (error behavior)

Notes [file C:/cygwin/bin/spin/leader.ltl]:

Some other properties:  
 $![] \text{noLeader}$   
 $\langle \rangle \text{elected}$   
 $[] (\text{noLeader} U \text{oneLeader})$

Symbol Definitions:

$\# \text{define elected} \quad (\text{nr\_leaders} > 0)$   
 $\# \text{define noLeader} \quad (\text{nr\_leaders} == 0)$   
 $\# \text{define oneLeader} \quad (\text{nr\_leaders} == 1)$

Never Claim:

Generate

```
/*
 * Formula As Typed:  $\langle \rangle [] \text{oneLeader}$ 
 * The Never Claim Below Corresponds
 * To The Negated Formula  $! (\langle \rangle [] \text{oneLeader})$ 
 * (formalizing violations of the original)
 */

never { /*  $! (\langle \rangle [] \text{oneLeader})$  */
T0_init:
    if
    ::  $! ([\text{oneLeader}]) \rightarrow \text{goto accept\_S9}$ 
    ::  $! 1 \rightarrow \text{goto T0\_init}$ 
```

Verification Result: valid

Run Verification

unreached in proctype node  
 line 53, "pan.\_\_\_\_", state 28, "out!two,nr"  
 (1 of 49 states)  
 unreached in proctype :init:  
 (0 of 11 states)  
 unreached in proctype :never:  
 line 108, "pan.\_\_\_\_", state 11, "-end-"  
 (1 of 11 states)  
 pan: elapsed time 0.006 seconds

Help Clear

Close Save As..

# Режимы верификации

- Основные режимы верификации SPIN:
  - Exhaustive – полный
  - Supertrace/Bitstate – супертрассы (с потерей состояний)
  - Hash-Compact – компактное хэширование
- В SPIN есть различные способы уменьшения размера модели при верификации
- Основные способы сокращения размеров модели:
  - Уменьшение затрат на хранение глобального состояния
  - Уменьшение числа состояний

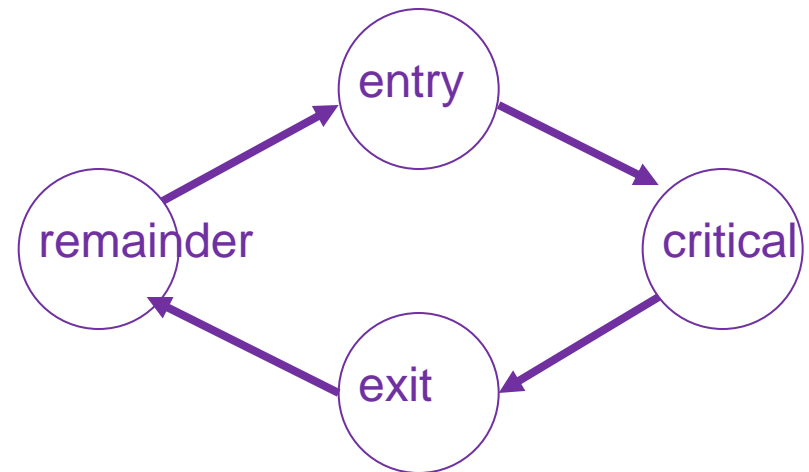
# Сложные механизмы синхронизации

---

# Проблема взаимного исключения

- Дано два последовательных процесса:  $P_1$  и  $P_2$

Критическая зона – ресурс общего пользования:  
разделяемый ресурс (shared resource)



Требуется найти алгоритм поведения каждого процесса т. ч. :

- В кр. з. в любой момент времени может быть не больше одного процесса
- Вход в кр. з. одного процесса не зависит от нахождения вне кр. з. другого процесса
- если два процесса хотят записать значение в кр.з. "одновременно", то они делают один за другим,
- если один процесс записывает значение, а второй просматривает, то второй видит либо старое, либо новое значение

# Обобщенные слабые семафоры

**Семафоры** – специальные переменные  $S = (S.C, S.L)$

- $S.C \geq 0, S.C \in \mathbb{Z}_0$
- $S.L$  – набор процессов
- две атомарные операции

Инициализация семафора  $S := (k, 0)$

**P-операция** или **wait** –  $\text{wait}(S)$

```
if S.C > 0
    S.C := S.C - 1
else
    S.L := S.L ∪ p
    p - блокируется
```

**V-операция** или **signal** –  $\text{signal}(S)$

```
if S.L == 0
    S.C := S.C + 1
else
    выбирается один из q ∈ S.L
    S.L := S.L - {q}
    q - разблокируется
```

# Еще семафоры

**Бинарные семафоры** –  $S = (S.C, S.L)$

- $S.C \in \{0,1\}$ ,  $S.C \in \mathbb{Z}_0$
- $S.L$  – набор процессов
- две атомарные операции

Инициализация семафора  $S := (k, 0)$

**P-операция** или **wait** –  $\text{wait}(S)$  – та же

**V-операция** или **signal** –  
 $\text{signal}(S)$

```
if S.L == 0
    S.C:=1
else
    выбирается один из q ∈ S.L
    S.L:=S.L-{q}
    q – разблокируется
```

**Сильные семафоры** –  $S.L$  - очередь



# Сильные семафоры на Promela

```
typedef Semaphore{  
    byte count, i, temp;  
    chan ch = [NProcs] of {pid}  
}
```

```
inline wait(S){  
    atomic{  
        if :: S.count >= 1 ->  
            S.count --;  
        :: else ->  
            S.ch ! _pid;  
            !(S.ch ?? [eval(_pid)])  
        }  
    }  
}
```

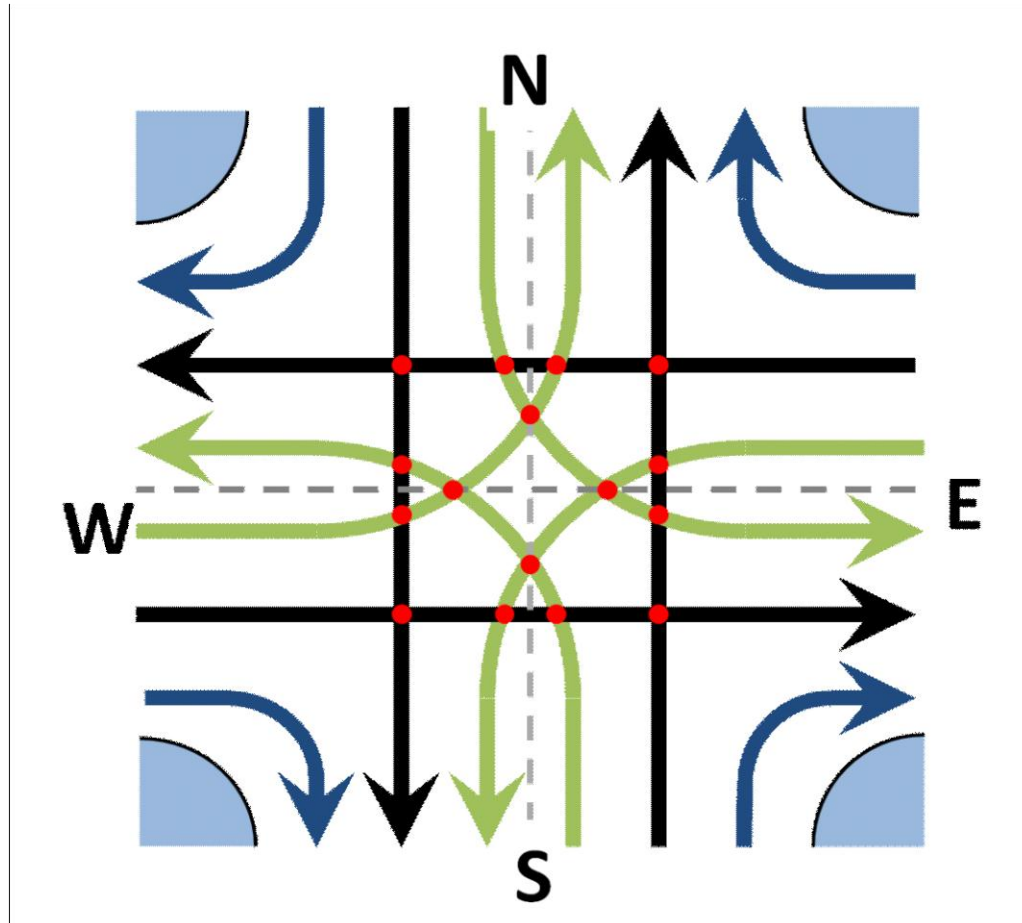
```
inline signal(S){  
    atomic{  
        S.i = len(S.ch);  
        if :: S.i == 0 ->  
            S.count ++  
        :: else -> S.ch ? _ fi  
    }  
}
```

# Курсовая «Контроллер светофоров»

---

- все группы, кроме РВКС 6 курс и примат

# Курсовой проект. Схема сложного перекрестка



# Курсовой проект (варианты)

№	Пересечение	№	Пересечение
1	WN, NS	9	SN, WE
2	WN, NE	10	SN, EW
3	WN, SW	11	SN, ES
4	WN, EW	12	NE, EW
5	NS, SW	13	NE, ES
6	NS, WE	14	SW, WE
7	NS, EW	15	SW, ES
8	SN, NE	16	WE, ES

- Вариант состоит из 3 пересечений

# Задача с лифтом (РВКС – 6 курс, примат)

---

# Задача с лифтом

- базовая задача
- 3 процесса – 1 пользователь, 1 контроллер лифта, 1 контроллер дверей лифта
- Данные
  - 5 этажей
  - на этаже есть кнопка вызова лифта, датчик движения
  - в лифте есть кнопки вызова лифта
- Пользователь:
  - случайно выбирает этаж вызова и этаж назначения
  - появляется на этаже назначения
  - если двери лифта открыты, то он входит в лифт
  - иначе, вызывает лифт и ждет
  - если лифта долго нет, то он уходит с этажа
  - иначе, входит в лифт,
  - в лифте нажимает кнопку этажа назначения
  - доезжает до своего этажа назначения и выходит

# Контроллер дверей лифта

---

1. Двери закрыты
2. Двери открываются, запускается таймер стоянки (датчик движения отключается), когда срабатывает сигнал от контроллера лифта или, когда срабатывает датчик движения
3. Если время таймера стоянки истекло, стартует закрытие дверей и активируется датчик движения
4. Если во время закрытия дверей датчик фиксирует движение, то переход на п. 1
5. Иначе, двери закрываются, датчик движения отключается

# Контроллер лифта

1. Лифт ожидает на некотором этаже
2. Если поступил вызов, лифт определяет направление движения
3. Если вызов с этажа стоянки, то он открывает двери
4. Иначе лифт начинает движение
5. При приближении к этажу контроллер проверяет, был ли вызов с площадки этого этажа или из кабины лифта
6. Если вызов с этажа был, то лифт останавливается и открывает двери
7. Если вызова с этажа не было, то лифт продолжает движение
8. Во время стоянки. Когда срабатывает таймер стоянки контроллера дверей, контроллер лифта вычисляет будущее направление движения
9. Когда двери закрылись, контроллер обновляет найденное направление движения
10. Если вызовов нет, то лифт ждет (переход на п. 1)
11. Если вызовы есть, то принимает решение о старте движения (переход на п.3)



# Цель курсовой работы

- разработать модель системы управления лифтом в соответствии с требованиями
- задачи:
  - разработать требования к системе управления на языке LTL  
– 5-10 штук
  - разработать модель системы управления лифтом в SPIN как минимум с 3 процессами (пользователь, контроллер дверей, контроллер лифта), используя базовую постановку
  - проверить требования в SPIN
  - если найдены ошибки, то исправить
  - модифицировать базовую модель и снова проверить требования

# Возможные модификации модели

- 2 кнопки вызова на этаже – ВВЕРХ, ВНИЗ
- базовый этаж – если нет вызовов, то лифт должен возвращаться на базовый этаж
- кнопка закрытия дверей в кабине лифта
- кнопка открытия дверей в кабине лифта
- кнопка экстренной остановки лифта в кабине – останавливается на ближайшем этаже по направлению движения
- 2 лифта
  - приезжает «ближайший» лифт к этажу вызова
  - приезжает первый, к которому поступил сигнал
- 2 пользователя
  - с датчиком веса в лифте (лифт может перевозить одного пользователя)
  - без датчика веса

# Заключение

---

- SPIN успешно используется для построения моделей распределенных алгоритмов и систем
- SPIN работает в режиме симуляции и верификации
- SPIN позволяет верифицировать свойства линейной темпоральной логики
- SPIN строит контрпример нарушения свойства, анализ которого позволяет выявить ошибку
- SPIN строит синхронную композицию модели и отрицания заданной формулы
  - проводит верификацию методами с сжатием без потери состояний и с потерей

# Заключение

- Основные типы объектов Promela
  - **процессы** определяют поведение
  - **каналы** используются для передачи сообщений между процессами
  - **данные** типов: `bit (boolean)`, `byte`, `short`, `int`
- Основные конструкции Promela
  - присваивание, выражение, вывод на экран и т.п.
  - отправка и получение сообщения, `skip`,
  - недетерминированный выбор по условию, оператор цикла
  - любая конструкция языка или **выполнимая**, или **блокирующая**
  - специальные конструкции для поддержки верификации
- Promela работает только с моделями **конечной** размерности

**СПАСИБО**