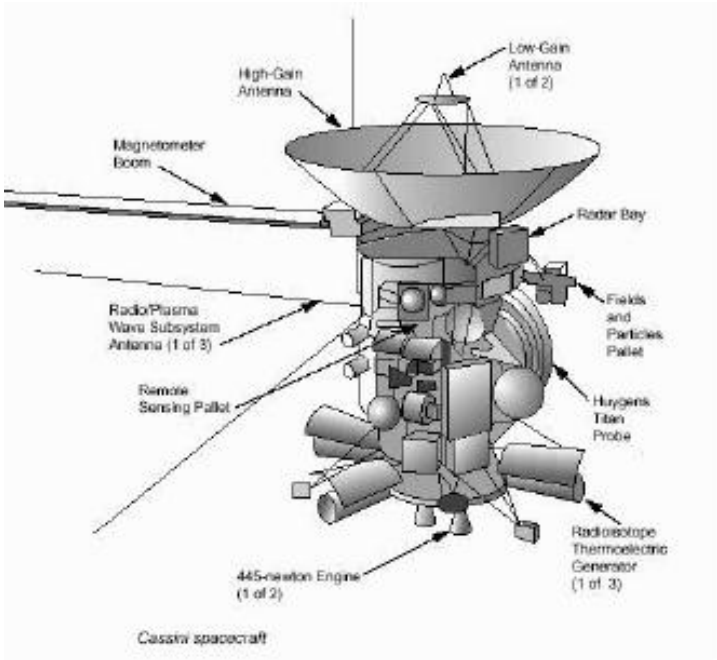
Моделирование и верификация распределенных систем в среде SPIN

И.В. Шошмина
Ю.Г. Карпов
А.Б. Беляев
ishoshmina@dcn.ftk.spbstu.ru
РВКС, ФТК, СПбГПУ
2012

SPIN

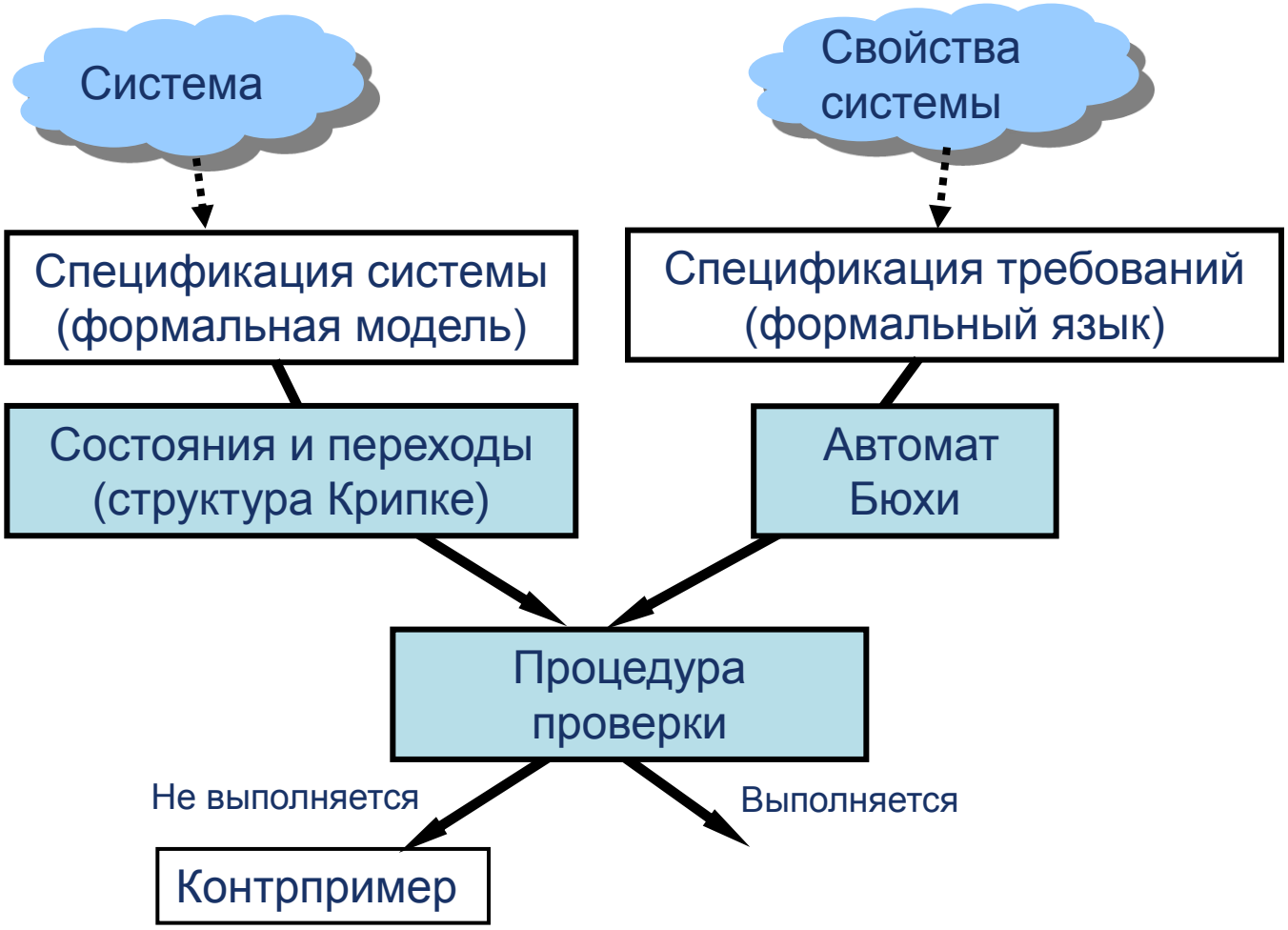
- разрабатывается Bell Labs с 1996
  - строить параллельные модели систем,
  - выразить требования на языке LTL
  - автоматически верифицировать выполнение требования на модели
- премия ACM System Award в 2001



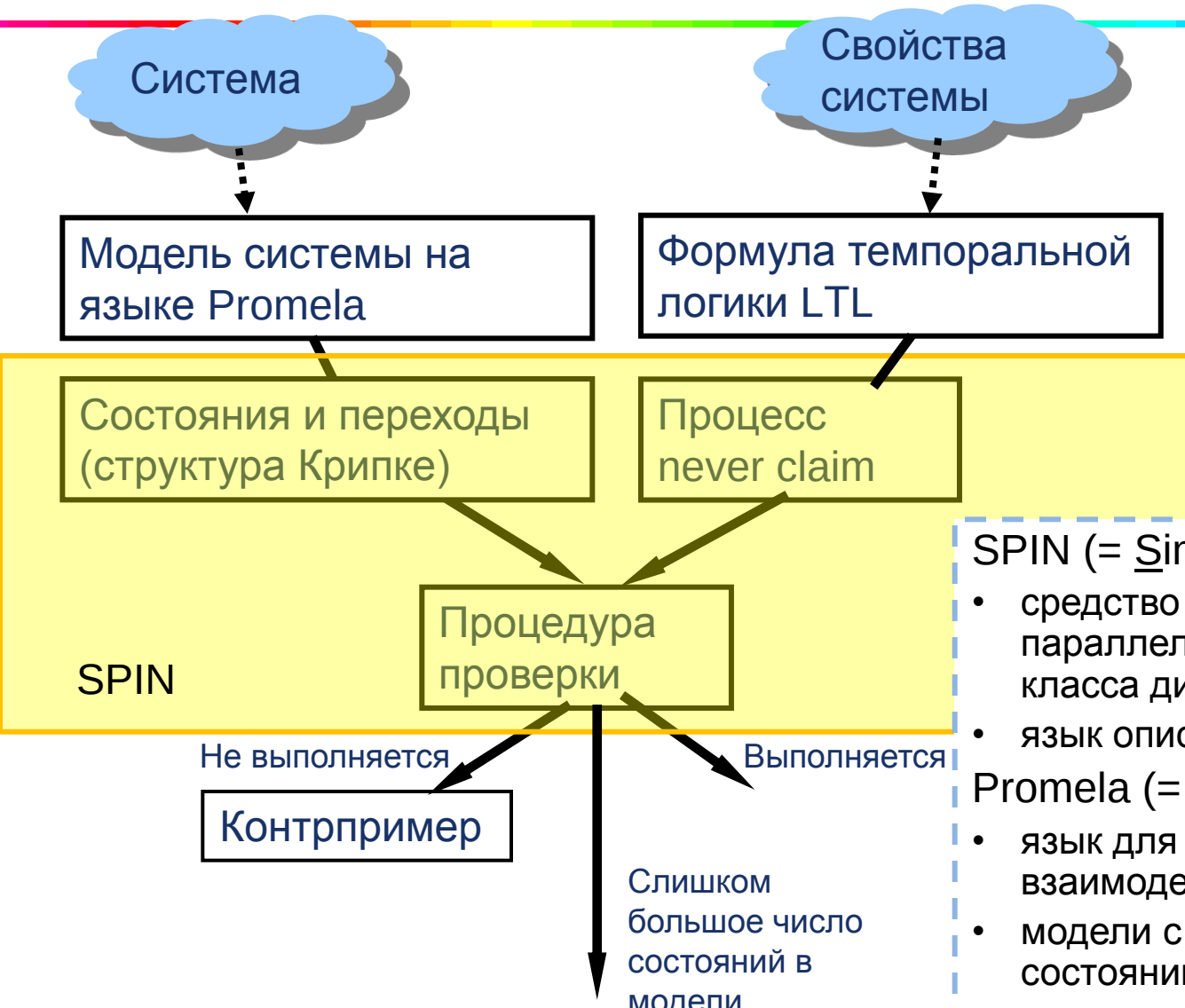
Использовался при верификации:

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

Средство верификации SPIN



Средство верификации SPIN



SPIN (= Simple Promela Interpreter)

- средство верификации протоколов, параллельных программ и широкого класса дискретных систем
- язык описания моделей Promela

Promela (= Protocol Meta Language)

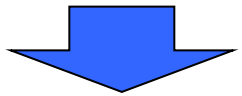
- язык для создания моделей взаимодействующих процессов
- модели с конечным числом состояний

Редактор LTL формул

Promela входной язык SPIN

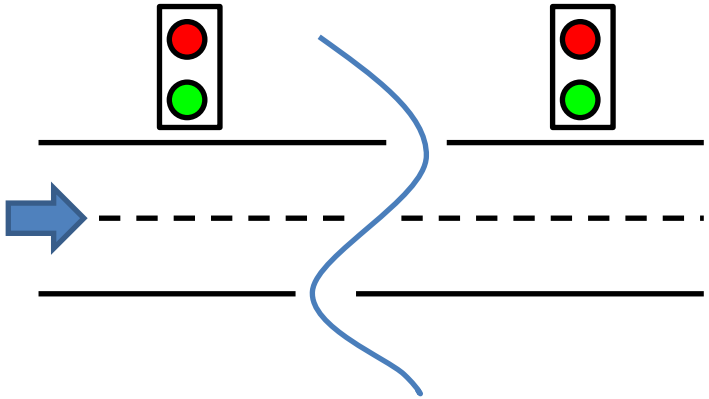
Модели, создаваемые на языке Promela

- 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

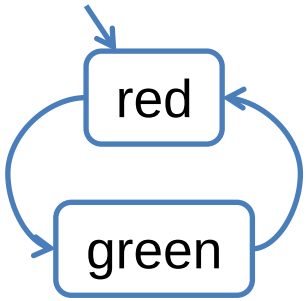


- Promela включает примитивы для создания процессов и описания межпроцессного взаимодействия
- НО! в нем отсутствует ряд средств, которые есть в языках программирования высокого уровня
  - Например, указатели на данные и функции, не включено понятие времени или часов, отсутствуют операции с плавающей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

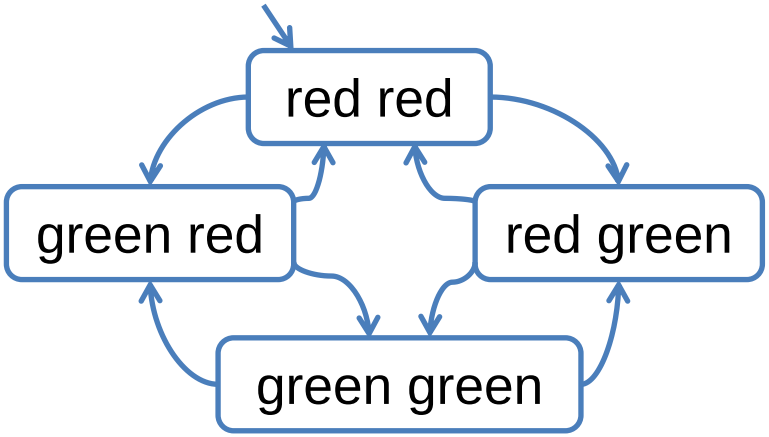
Чередование процессов. Светофоры



TrLight:



$TrLight_1 \parallel TrLight_2$:



Количество состояний общей системы переходов:

$$\left|TrLight_1\right| \times \left|TrLight_2\right| =$$

$$= 2 \times 2 = 4$$

Модель двух светофоров на языке Promela

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

тело процесса

```
proctype TrLight()
{
}
```

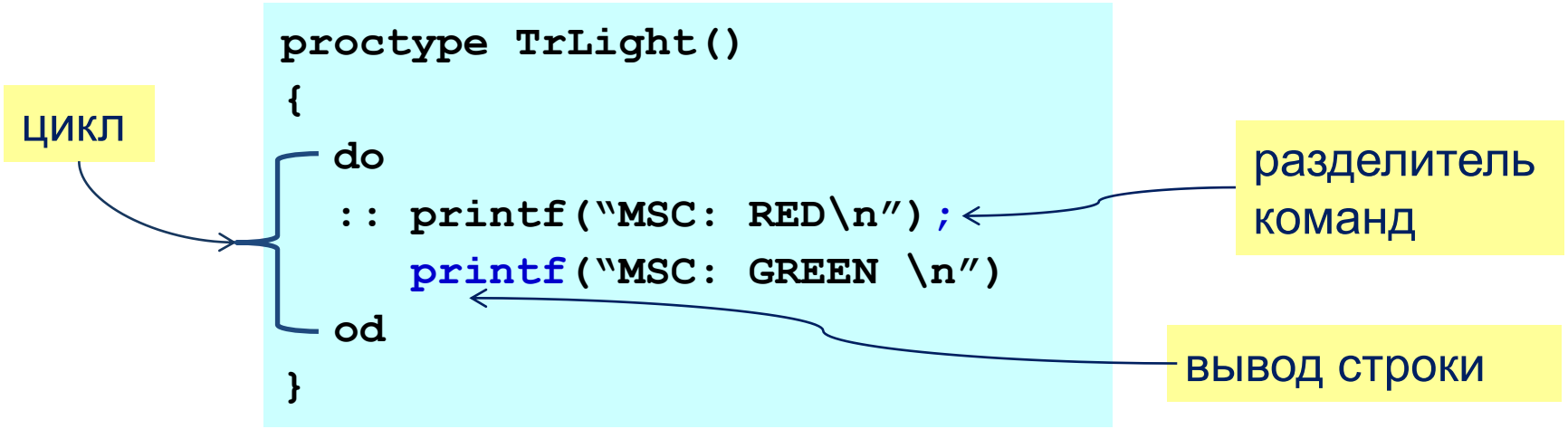
объявление процесса

```
init()
{ run TrLight();
  run TrLight()
}
```

- Основной процесс **init** (ключевое слово)
  - основной процесс может отсутствовать в программе

- Процесс может запускаться при помощи оператора **run** или непосредственно в начале работы

Модель двух светофоров на языке Promela

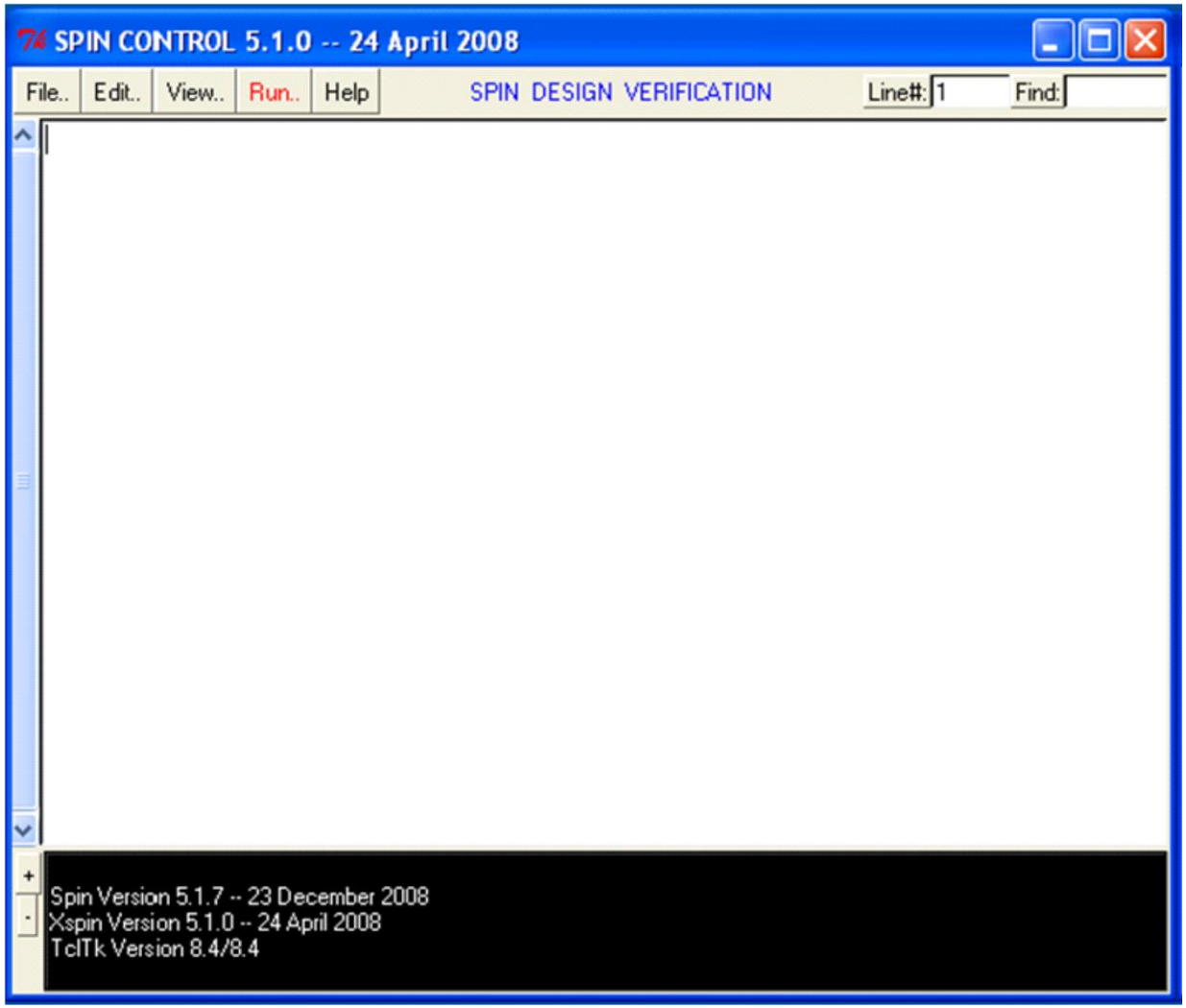


- процессы в Promela выполняются асинхронно
- в распределенных системах отсутствуют предположения о скорости процессов

Запуск оболочки XSpin

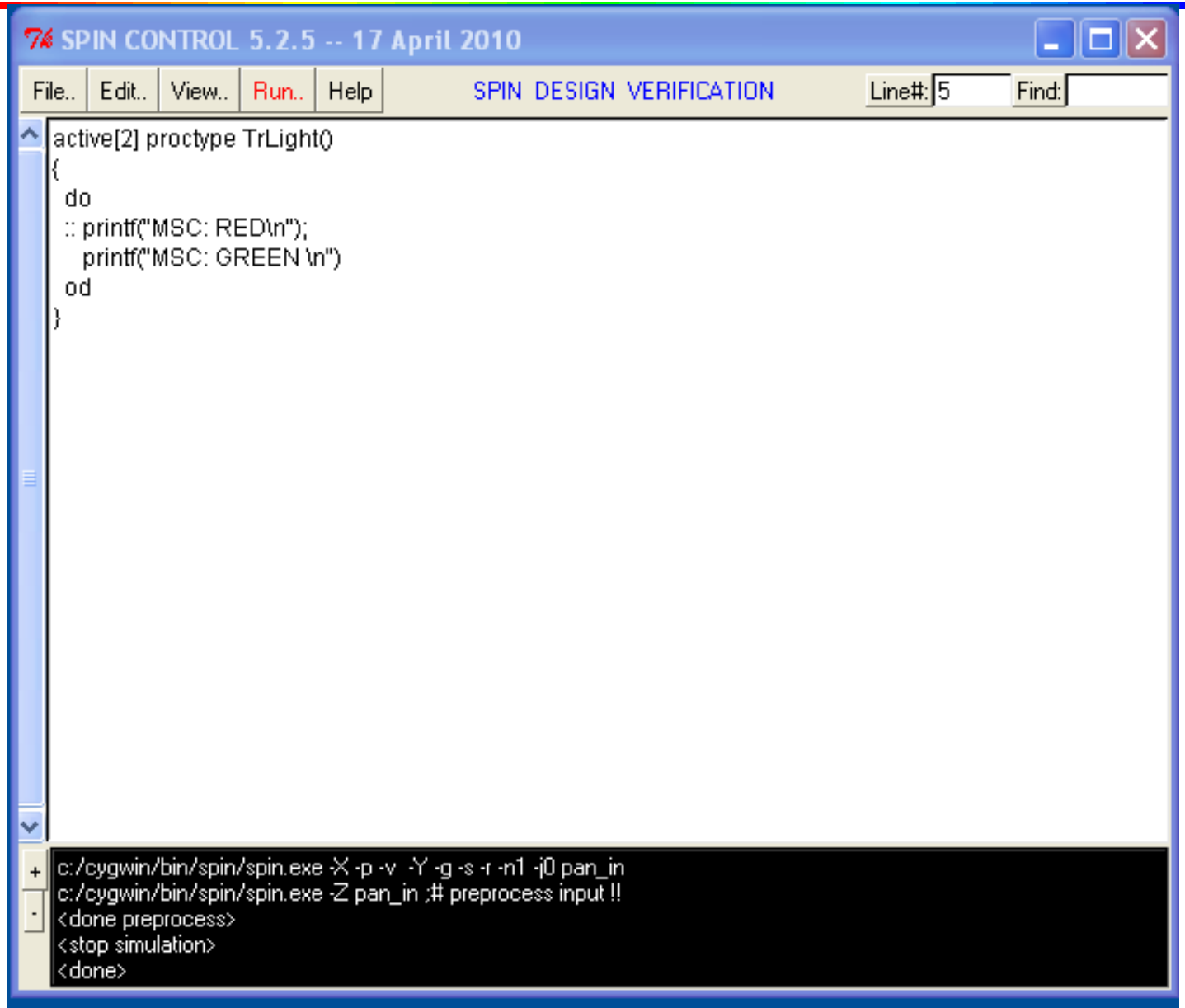
```
$ /bin/spin/xspin.tcl
```

запуск оболочки XSpin из Cygwin

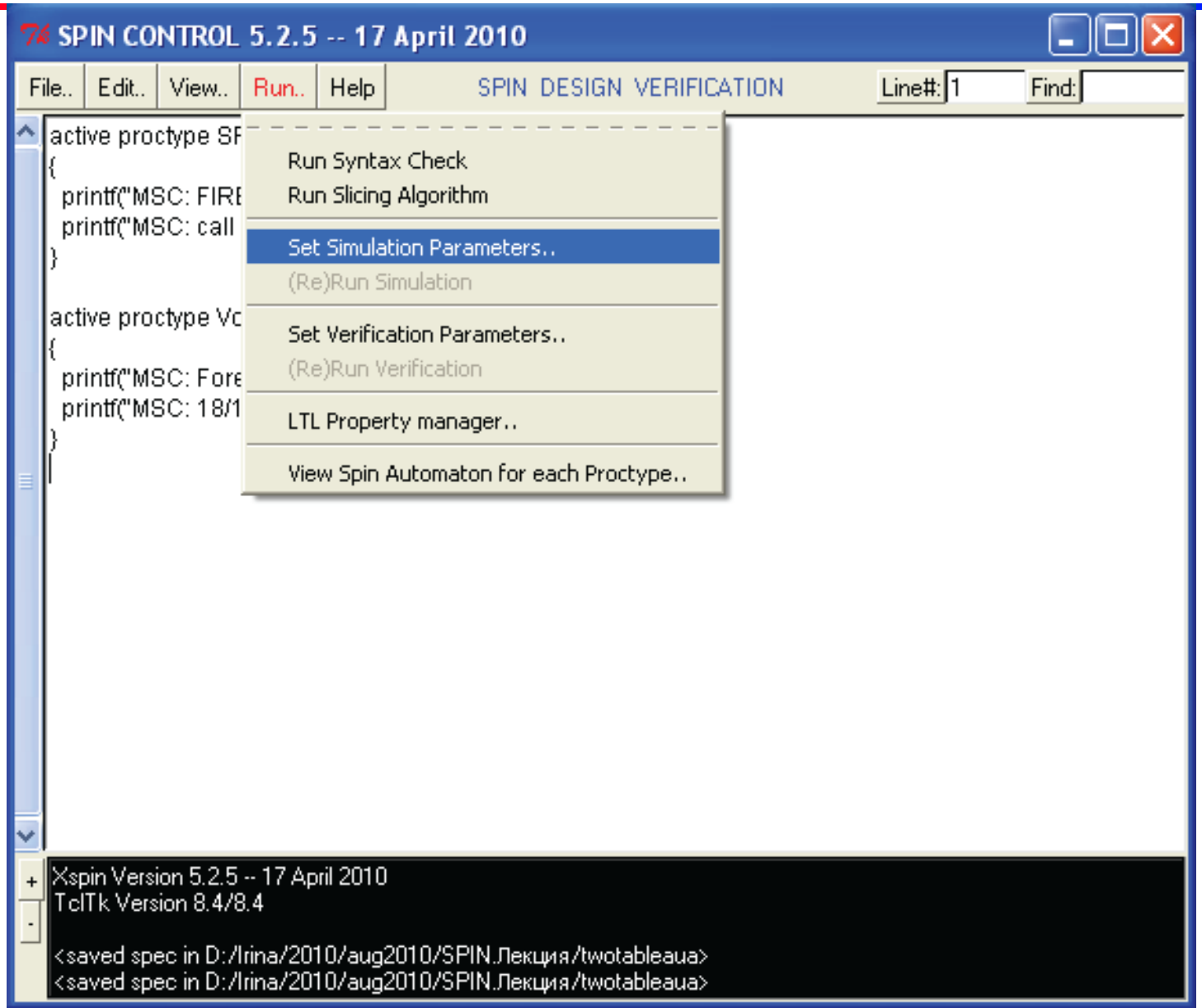


Основное окно редактора XSpin

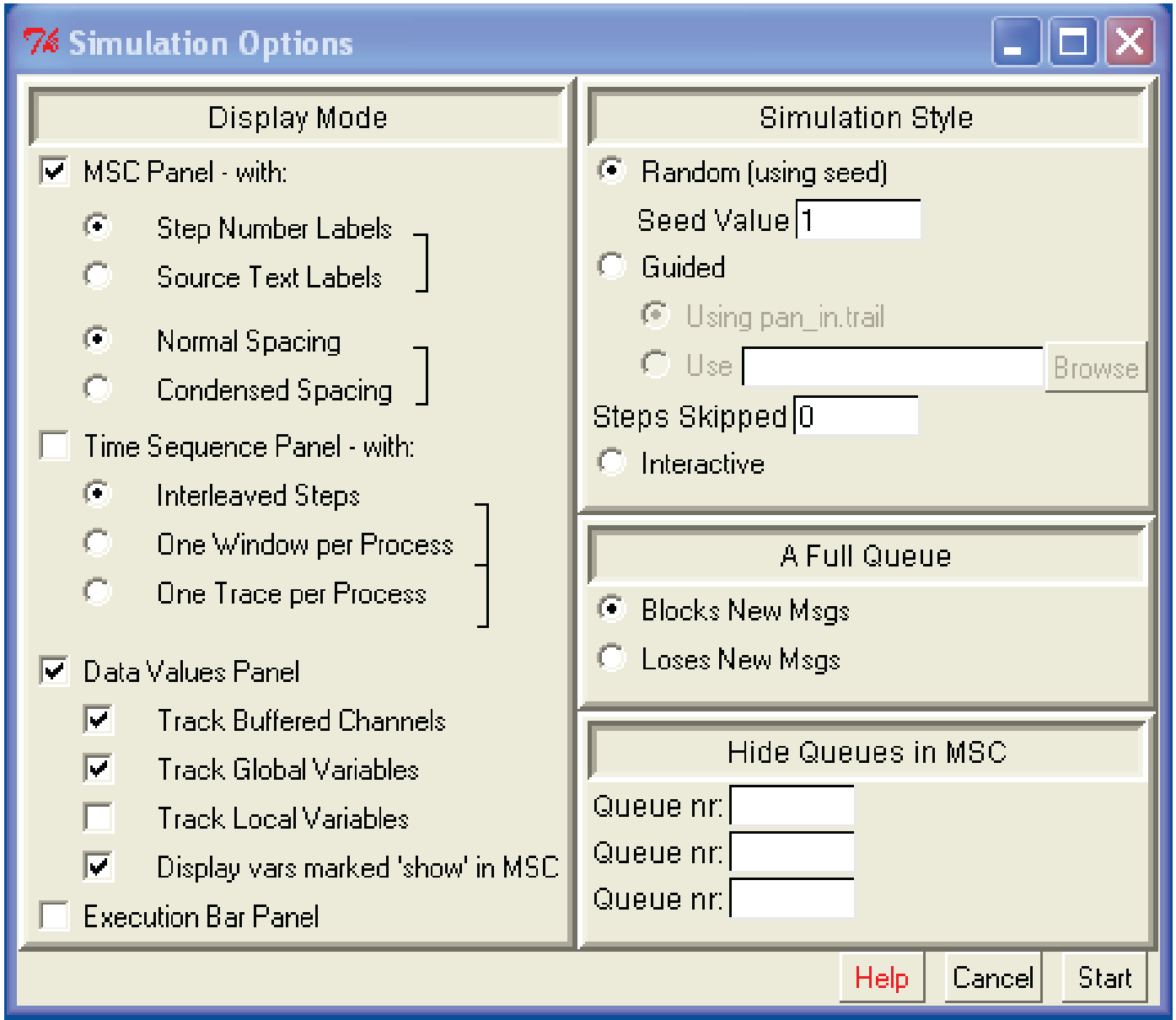
Загрузка файла с моделью на Promela



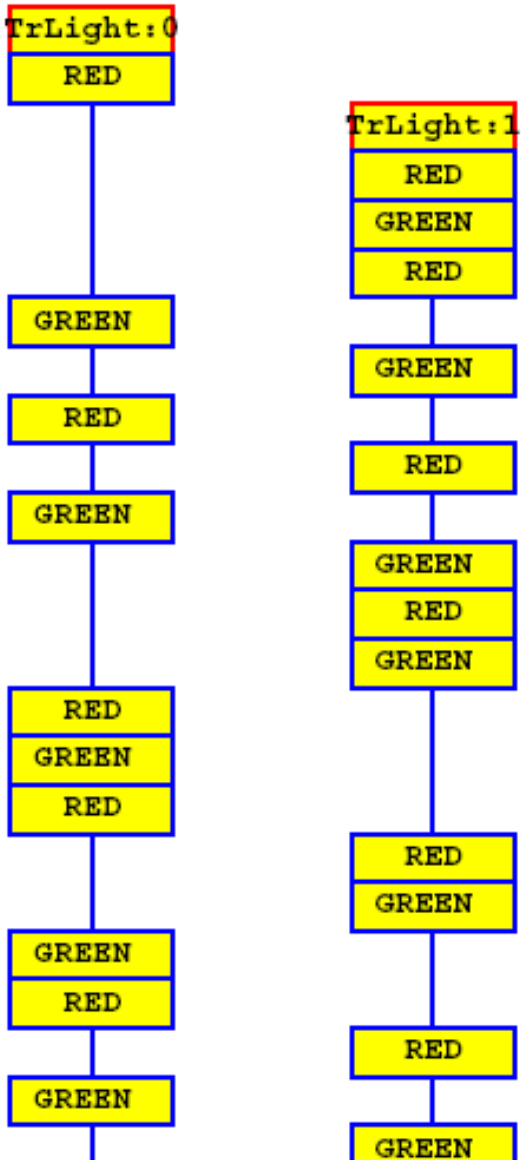
Запуск симуляции



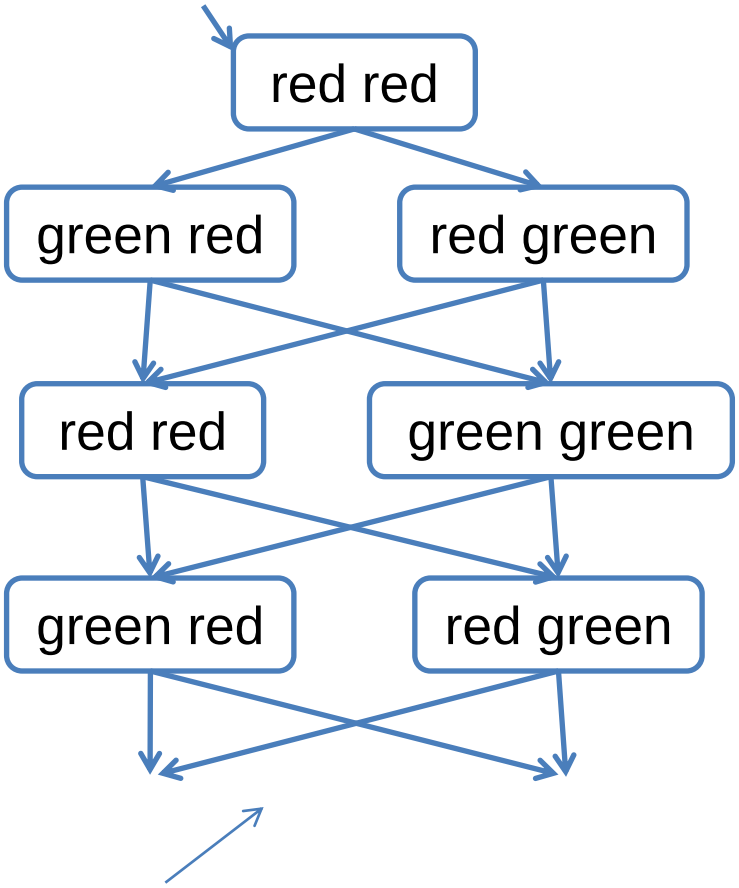
Окно параметров симуляции



Симуляция модели двух светофоров

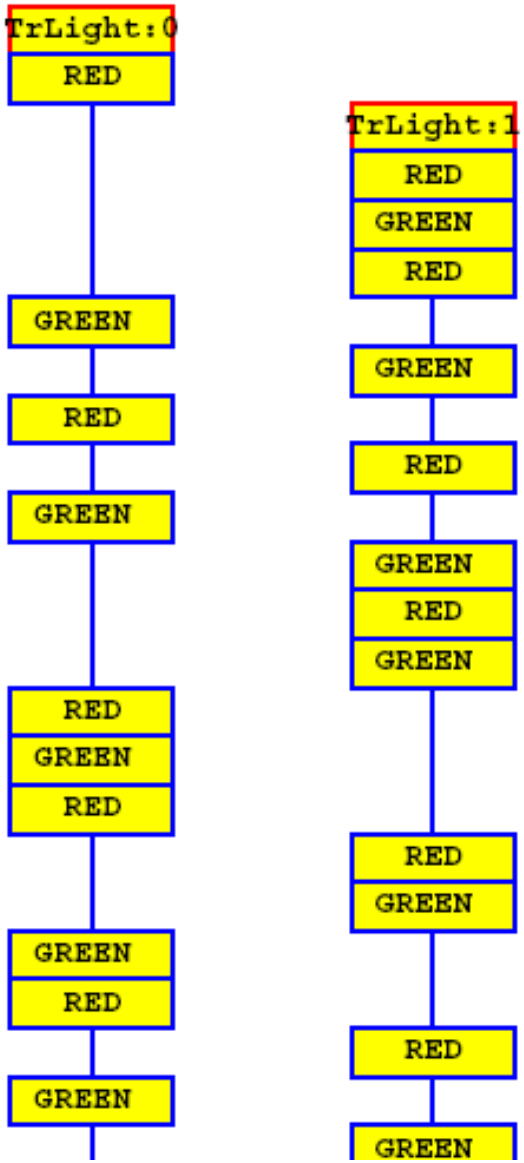


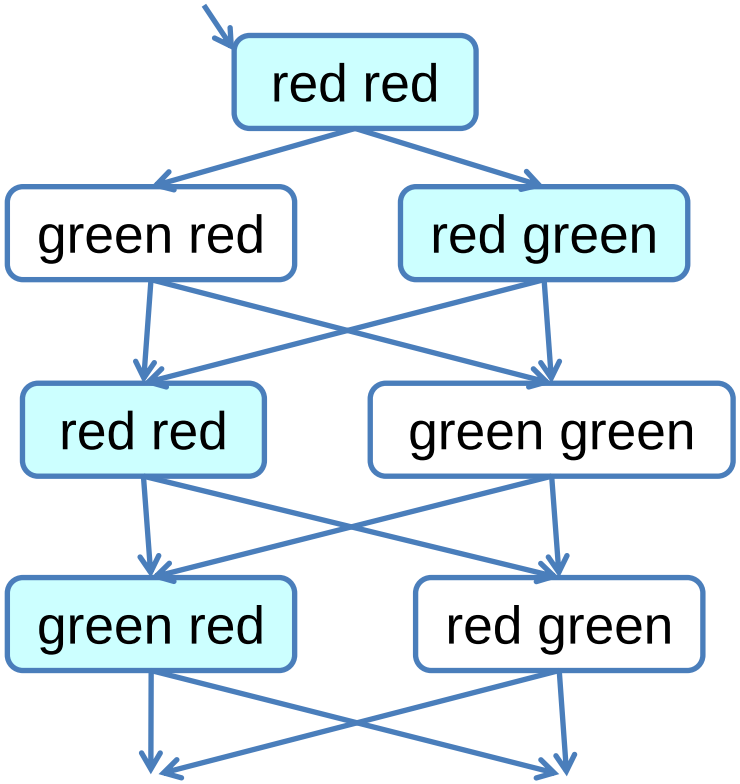
Глобальное состояние системы:
<метка состояния светофора 1, метка состояния светофора 2>



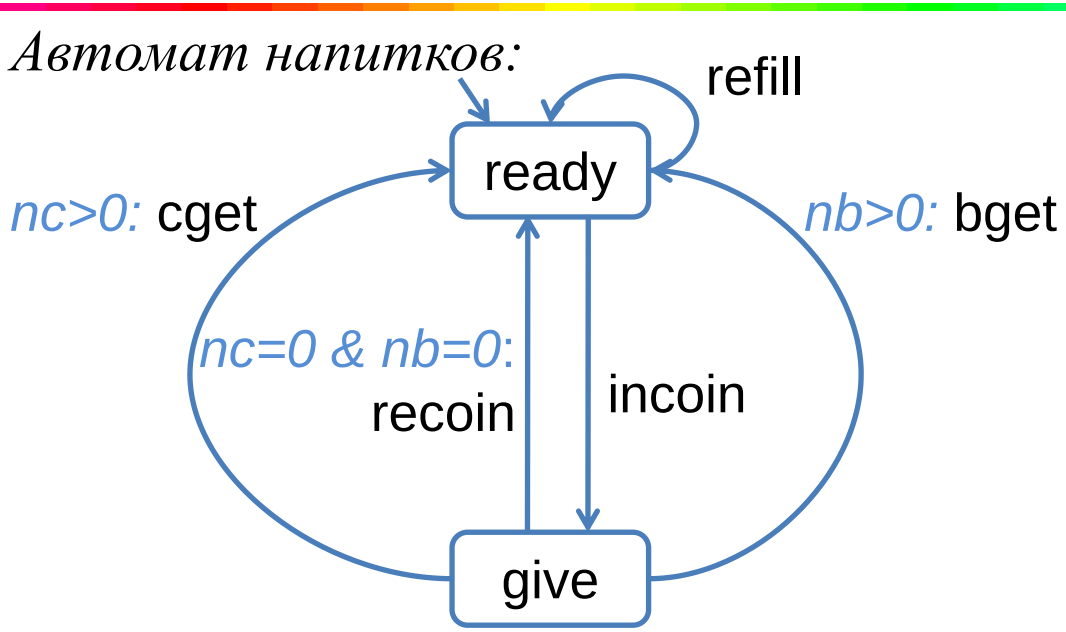
Развертка системы

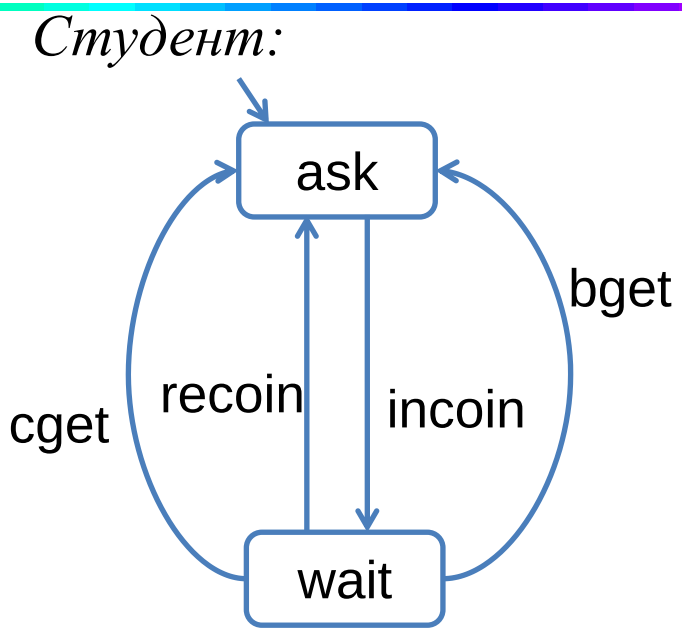
Симуляция модели двух светофоров





Автомат напитков и студент





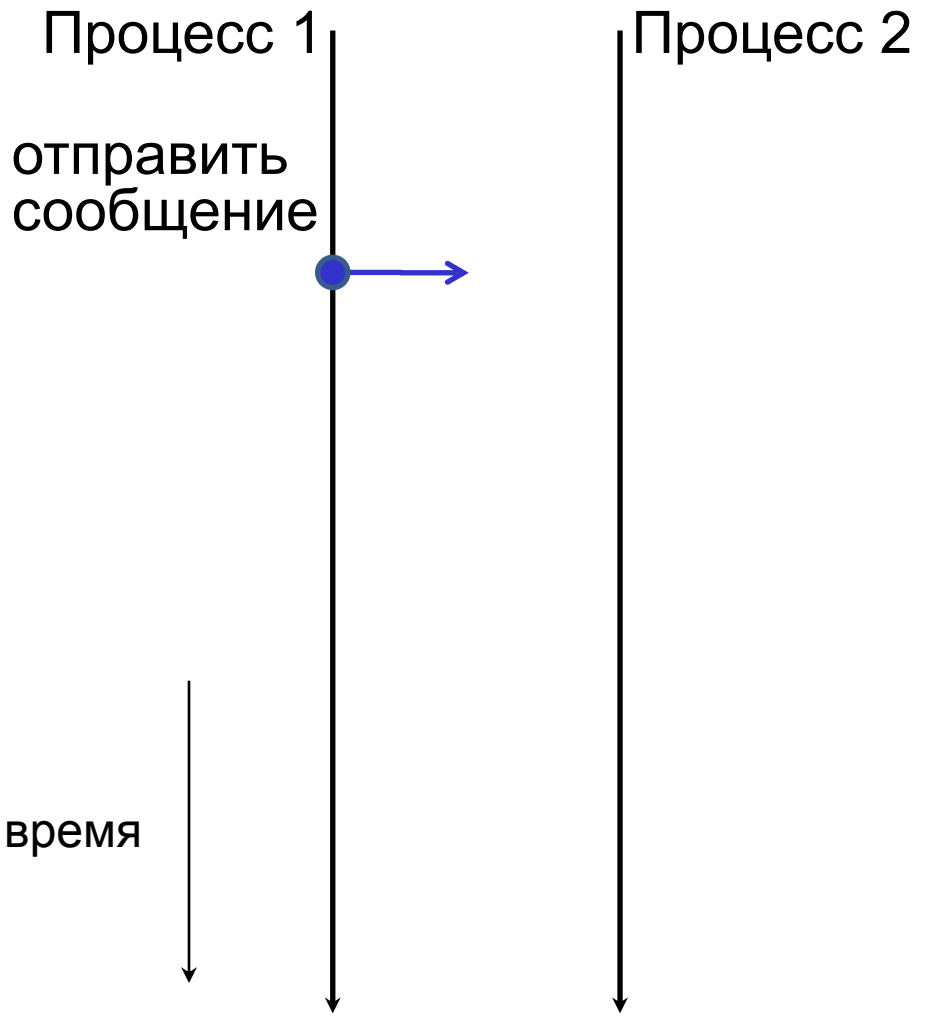
- *nc* – количество банок колы в автомате
- *nb* – количество банок пива в автомате

Сообщения в системе:

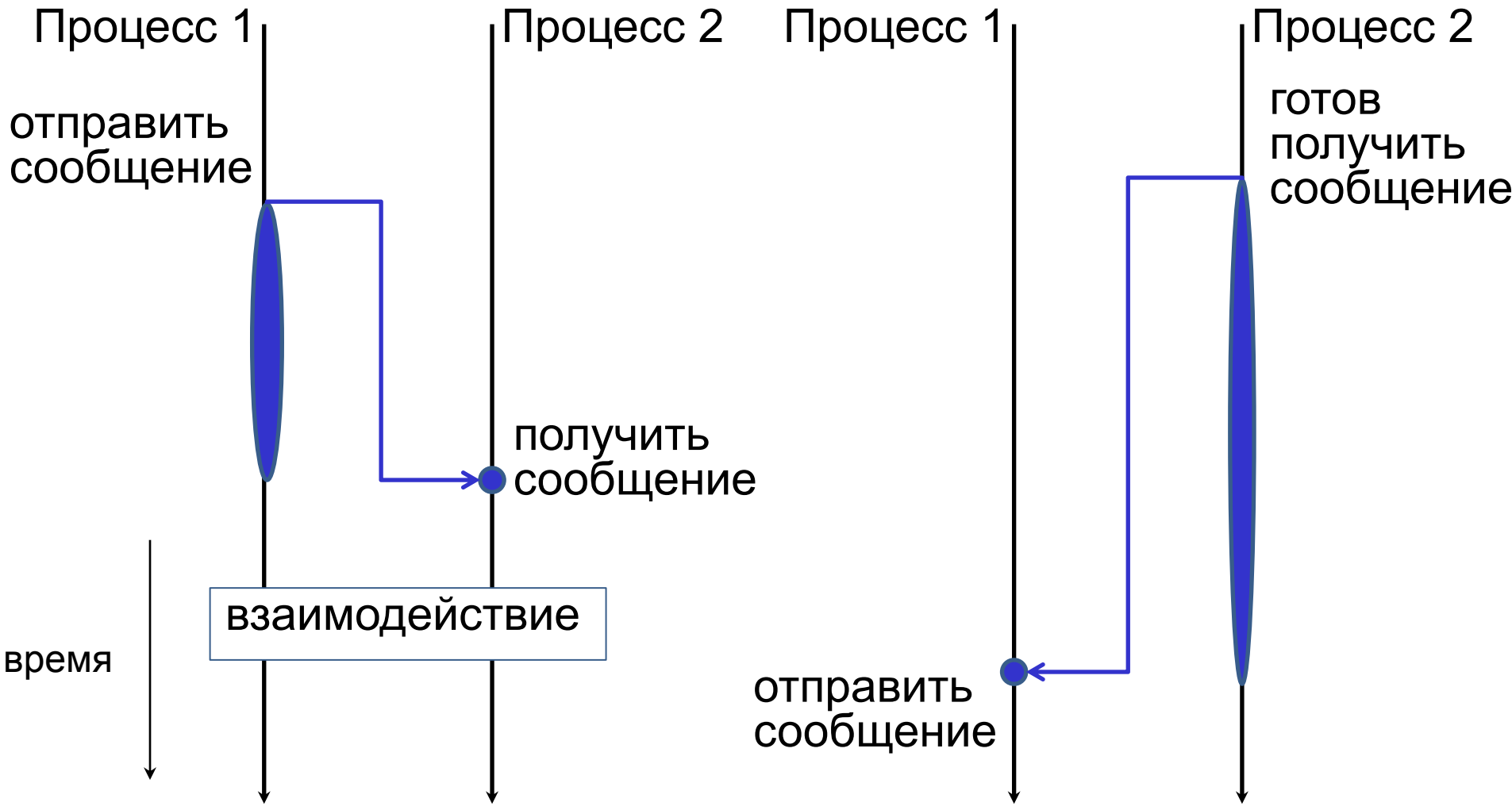
- *incoin* – в автомат бросили монету
- *recoin* – автомат вернул монету
- *cget* – автомат выдал колу
- *bget* – автомат выдал пива

Взаимодействия в системе синхронные

Синхронные взаимодействия



Синхронные взаимодействия



Синхронное взаимодействие называют рукопожатием (handshaking)

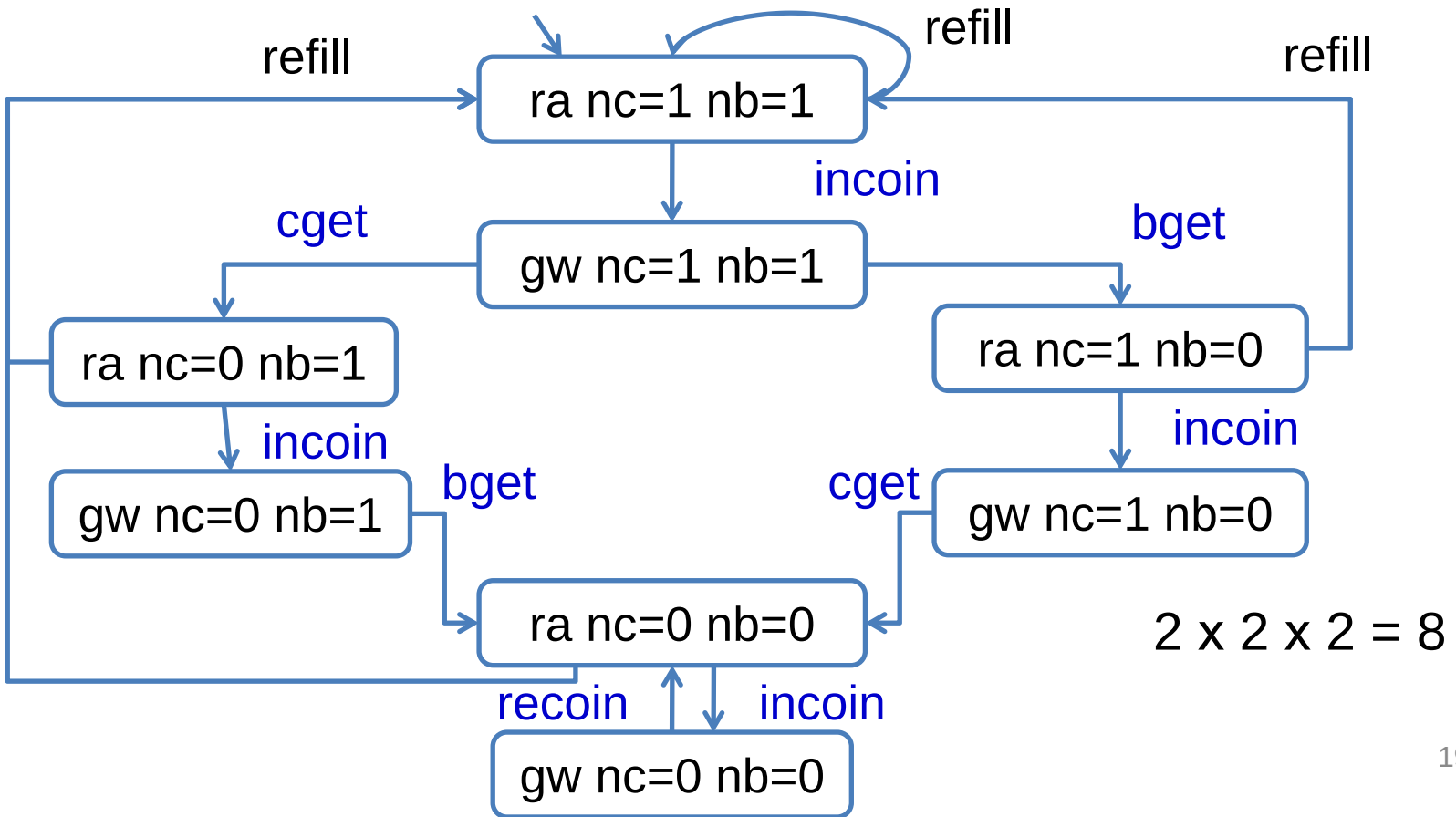
Система переходов с синхронным взаимодействием

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

Зависит от числа банок колы и пива!

Модели на языке Promela должны быть конечны

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



2 x 2 x 2 = 8

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

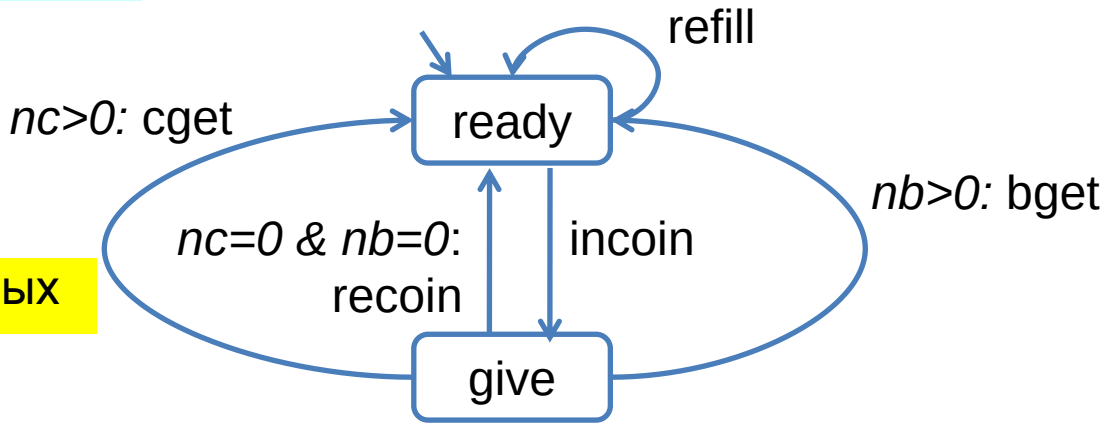
формат сообщения

```
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

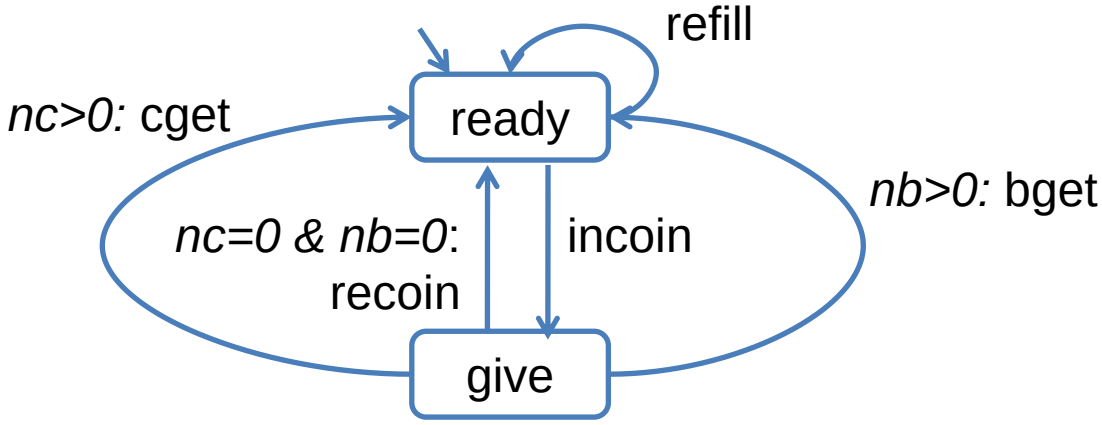
```
  do
  :: (state == 0) -> /*...*/
  :: (state == 0) -> /*...*/
  :: (state == 1) && (nc == 0) && (nb == 0) -> /*...*/
  :: (state == 1) && (nc > 0) -> /*...*/
  :: (state == 1) && (nb > 0) -> /*...*/
  od
```

цикл

```
}
```

Цикл в Promela

- 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 getbeer – на всех путях когда-нибудь в будущем студент получит банку пива

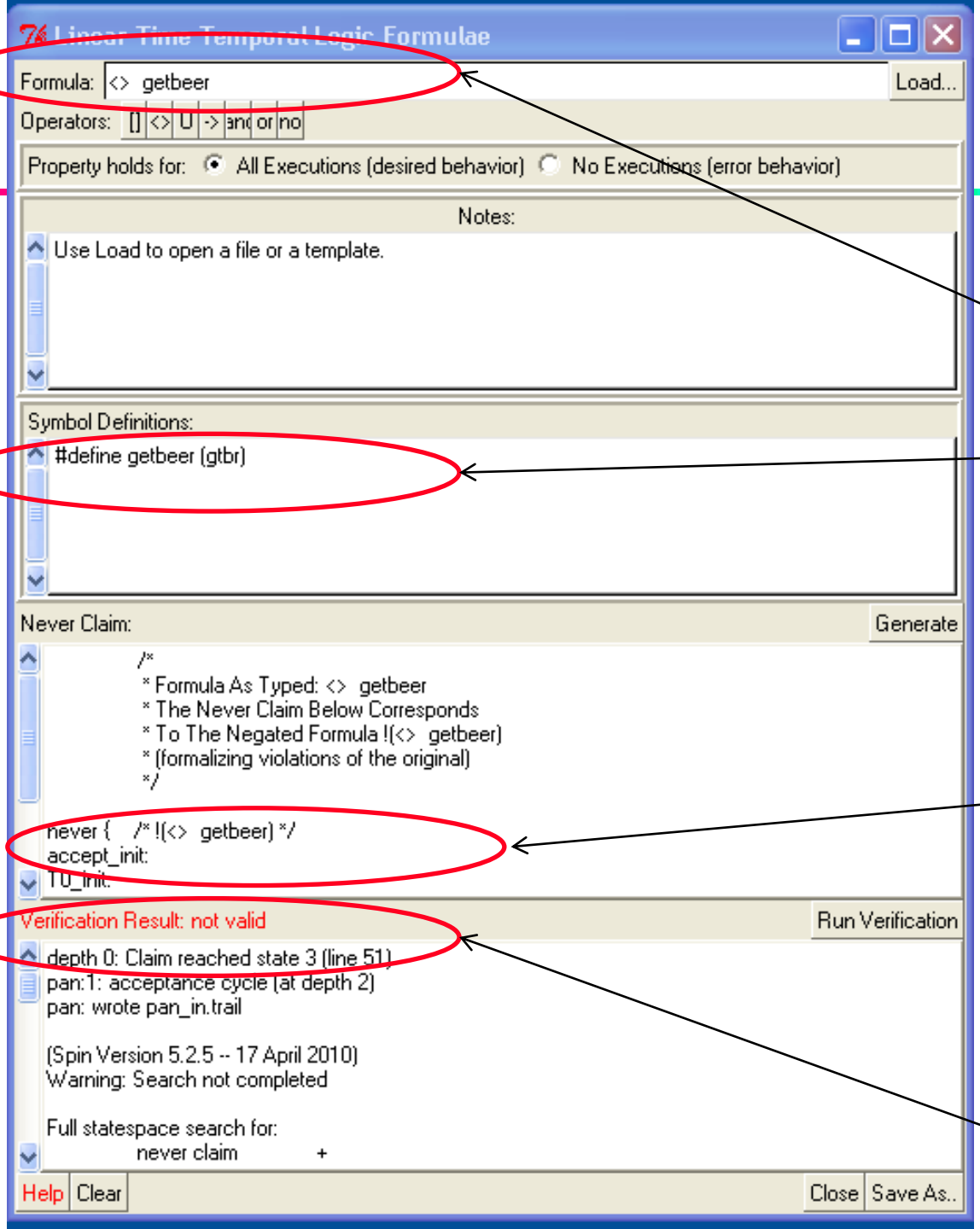
Запись LTL формул в SPIN:

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

В строке редактора формул в SPIN запишем:

<> getbeer

Окно редактора LTL формул



Запись формулы

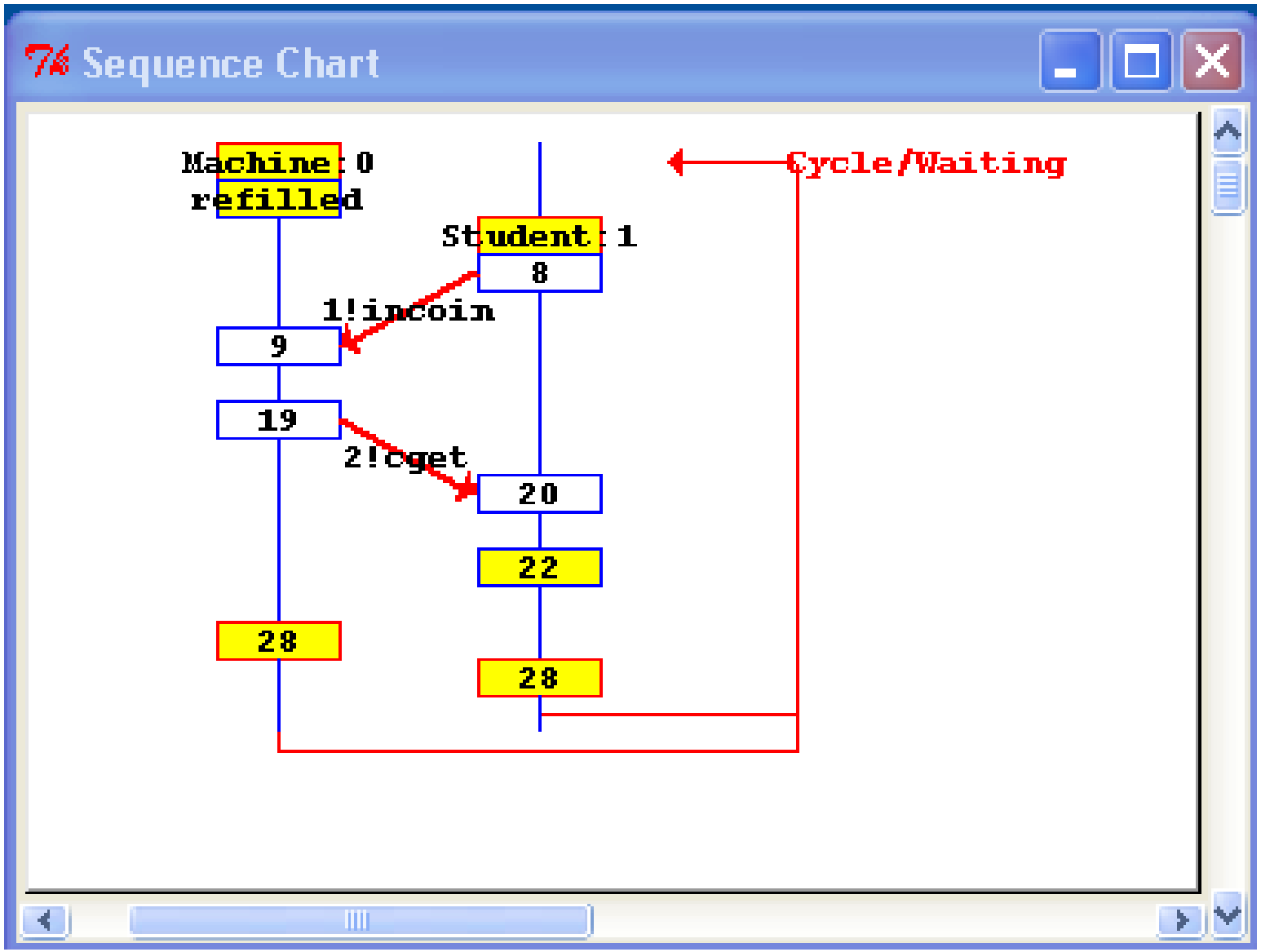
Предикат

Процесс never claim

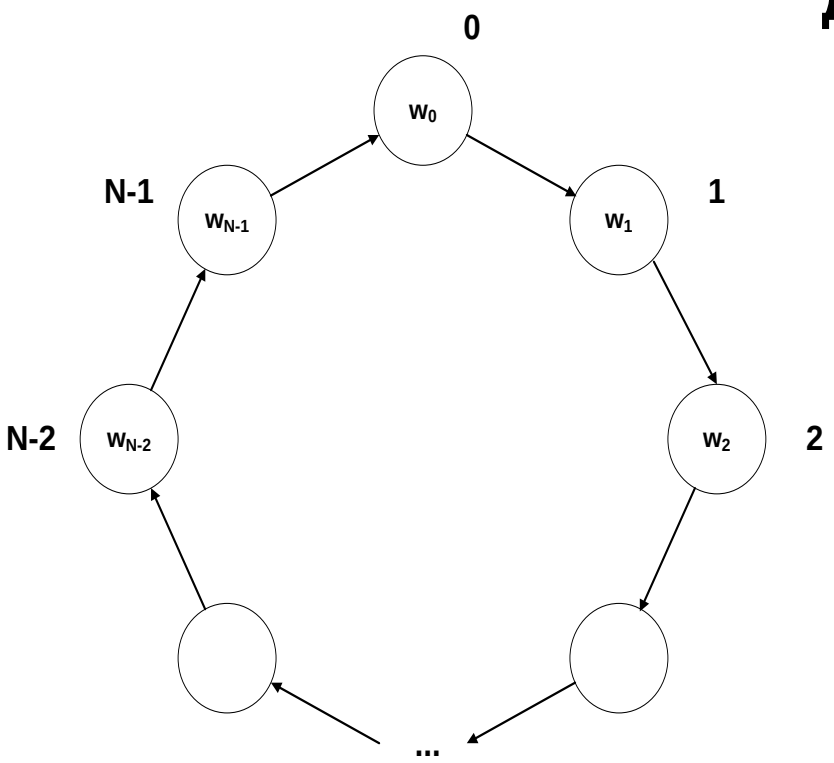
- Отрицание LTL формулы на языке Promela
- При верификации строится синхронная композиция процесса never claim и модели

Результат верификации

Окно диаграммы взаимодействия



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

Требуется построить протокол:

набор **ЛОКАЛЬНЫХ** правил для каждого узла, которые позволят получить **ГЛОБАЛЬНЫЙ** результат - каждому узлу определить лидера

- например, узел с наибольшим весом

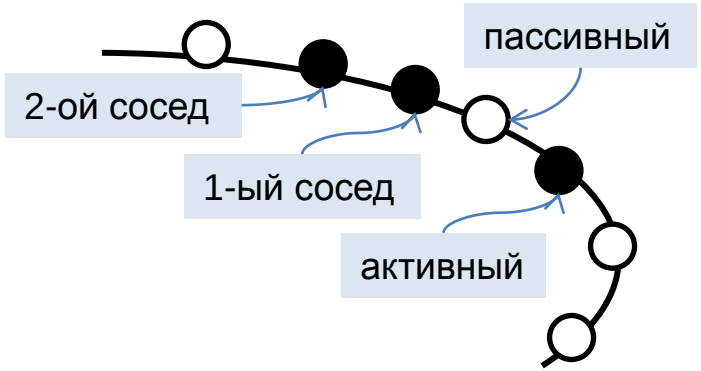
Эффективный алгоритм выбора лидера (Dolev-Klawe-Rodeh, Peterson)

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

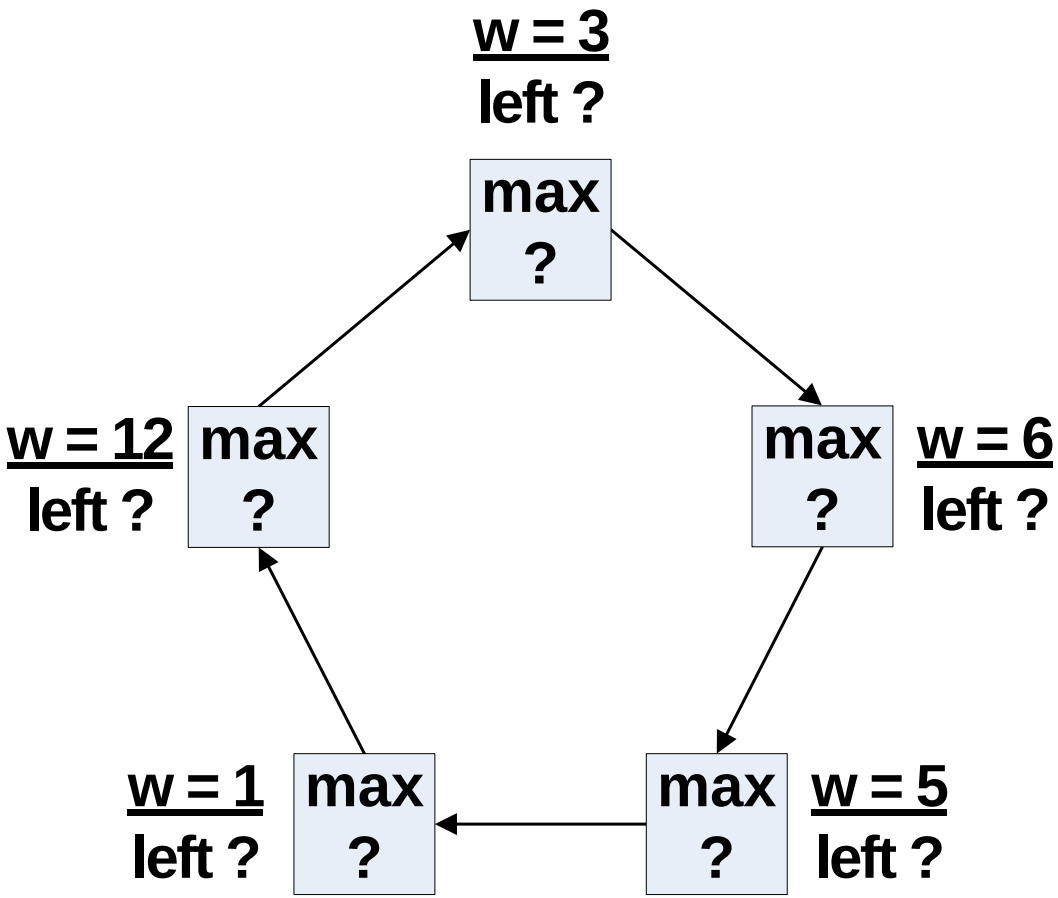
активного узла

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

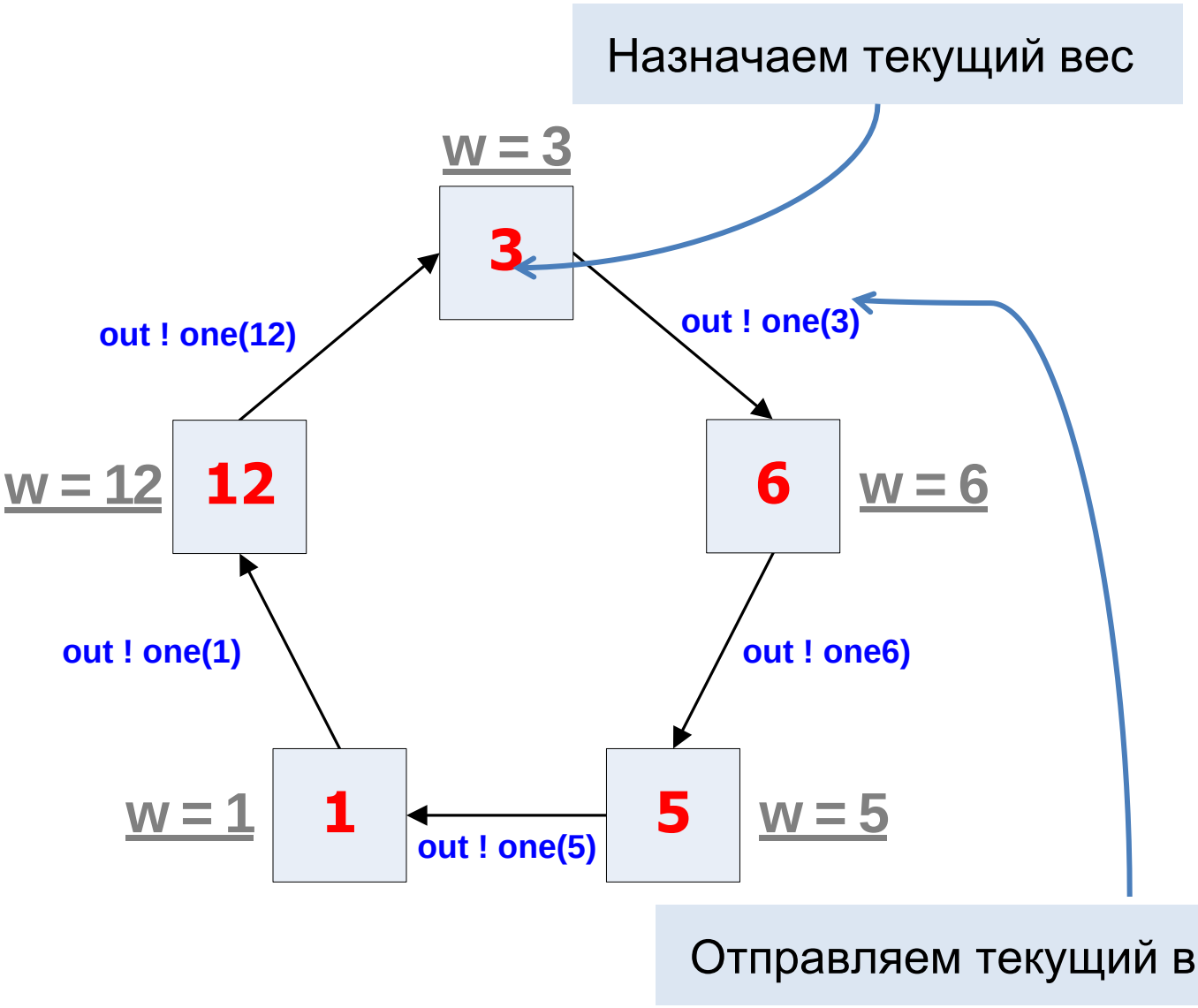
```
A0. max = w_i
    послать сообщение one(max)
A1. получить сообщение one(q)
    если (q != max) то
       left := q
       послать сообщение two(left)
    иначе послать сообщение winner(max)
      max является глобальным максимумом
A2. получить сообщение two(q)
    если (left > q) и (left > max) то
       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



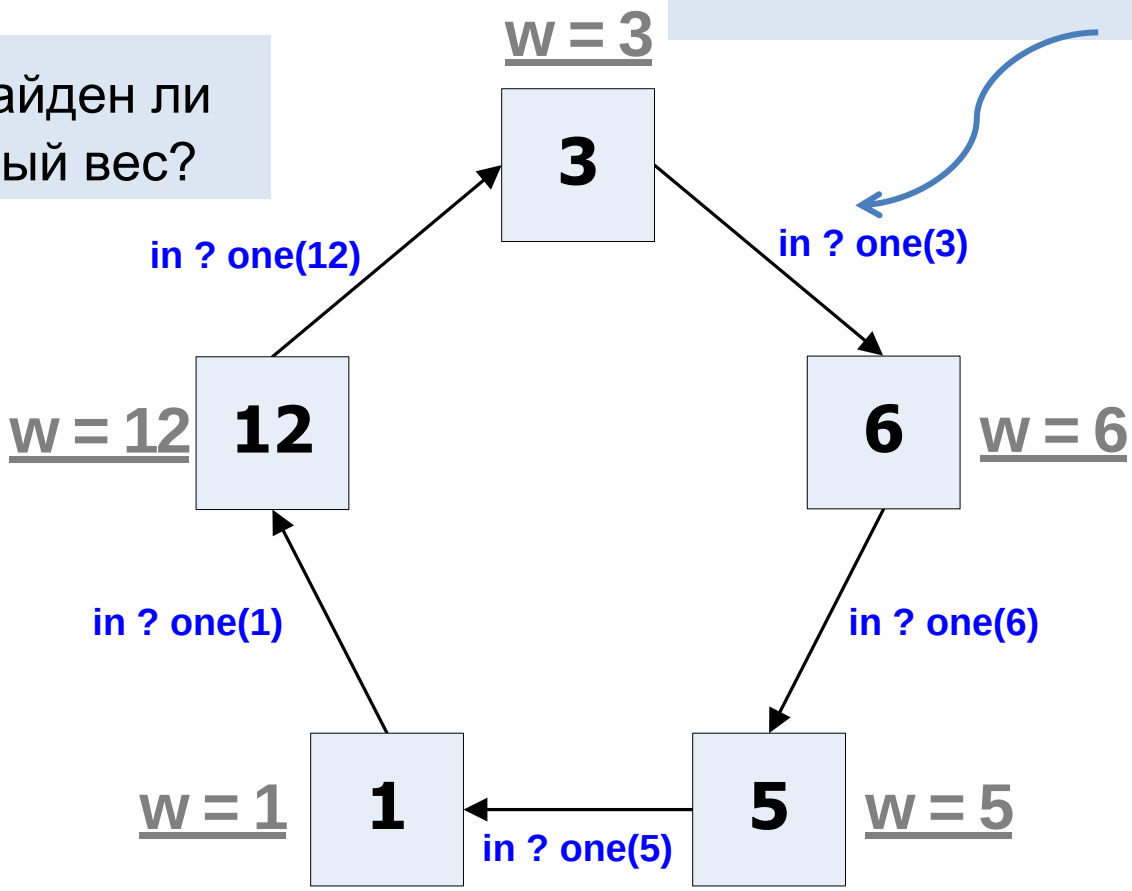
Пример. A0. Начальная фаза



Пример. Фаза A1

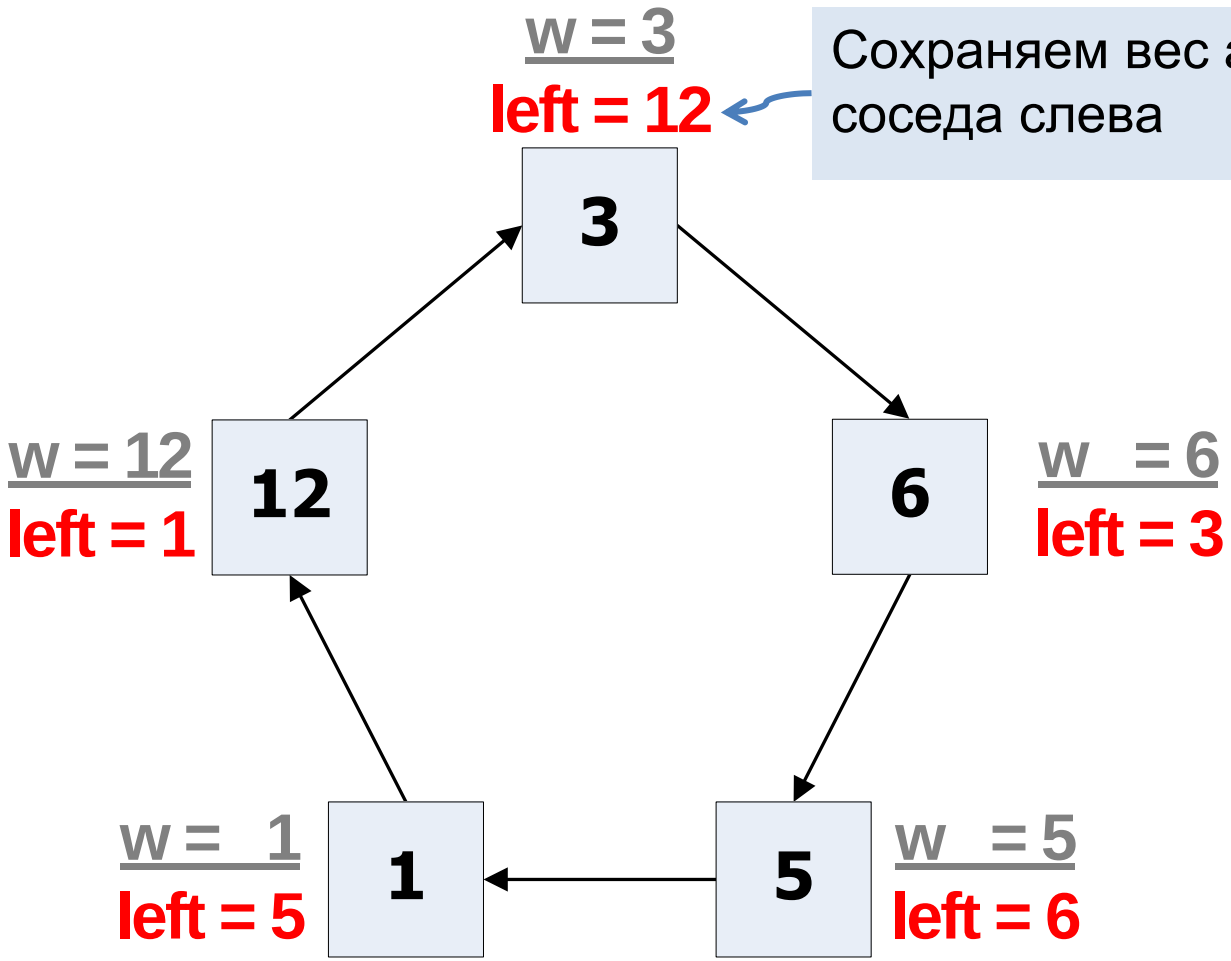
Получаем вес активного соседа

Проверяем: найден ли максимальный вес?

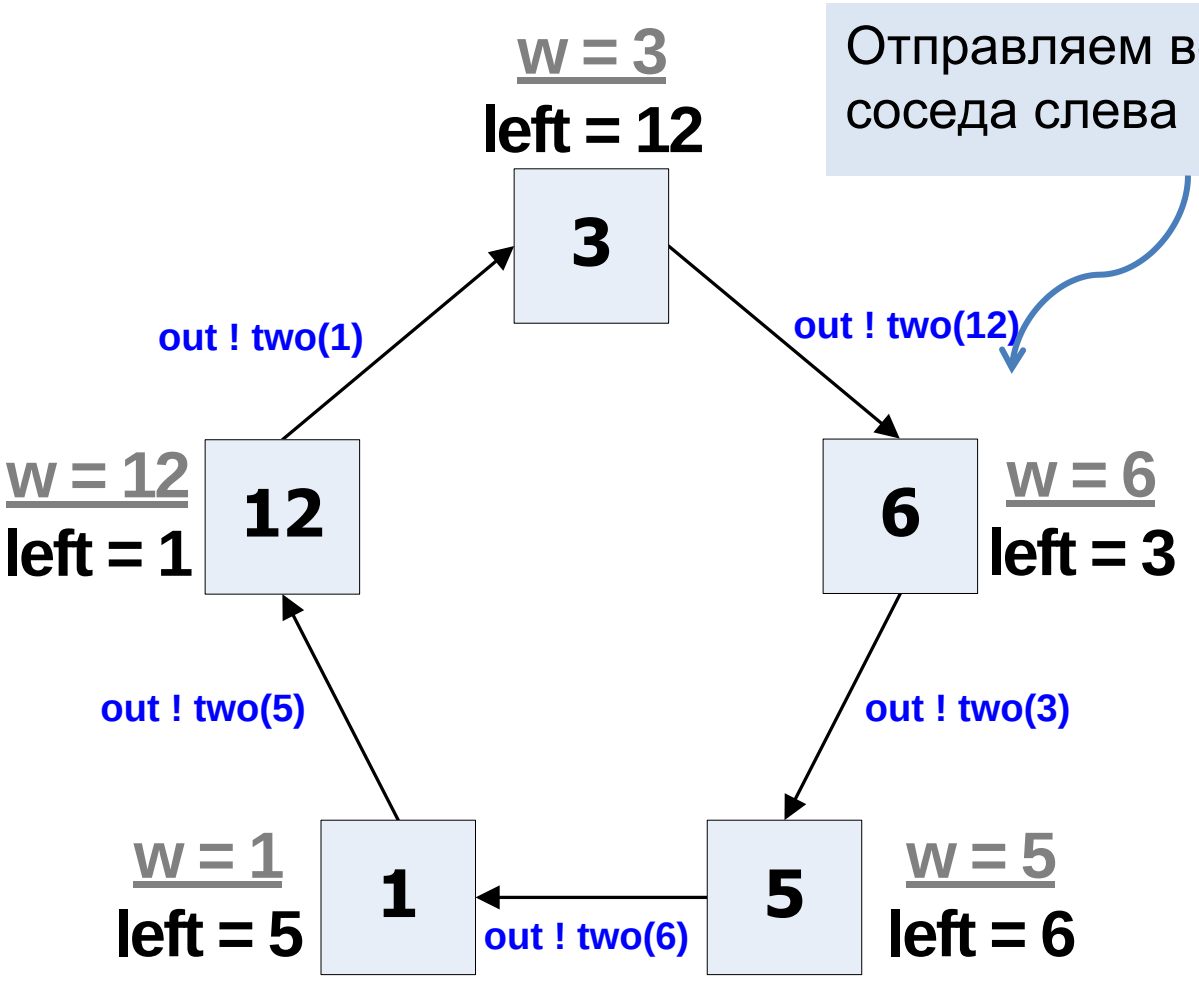



```
if (q != max) -> (left := q; send two(left))
else победитель найден
```

Пример. Фаза A1



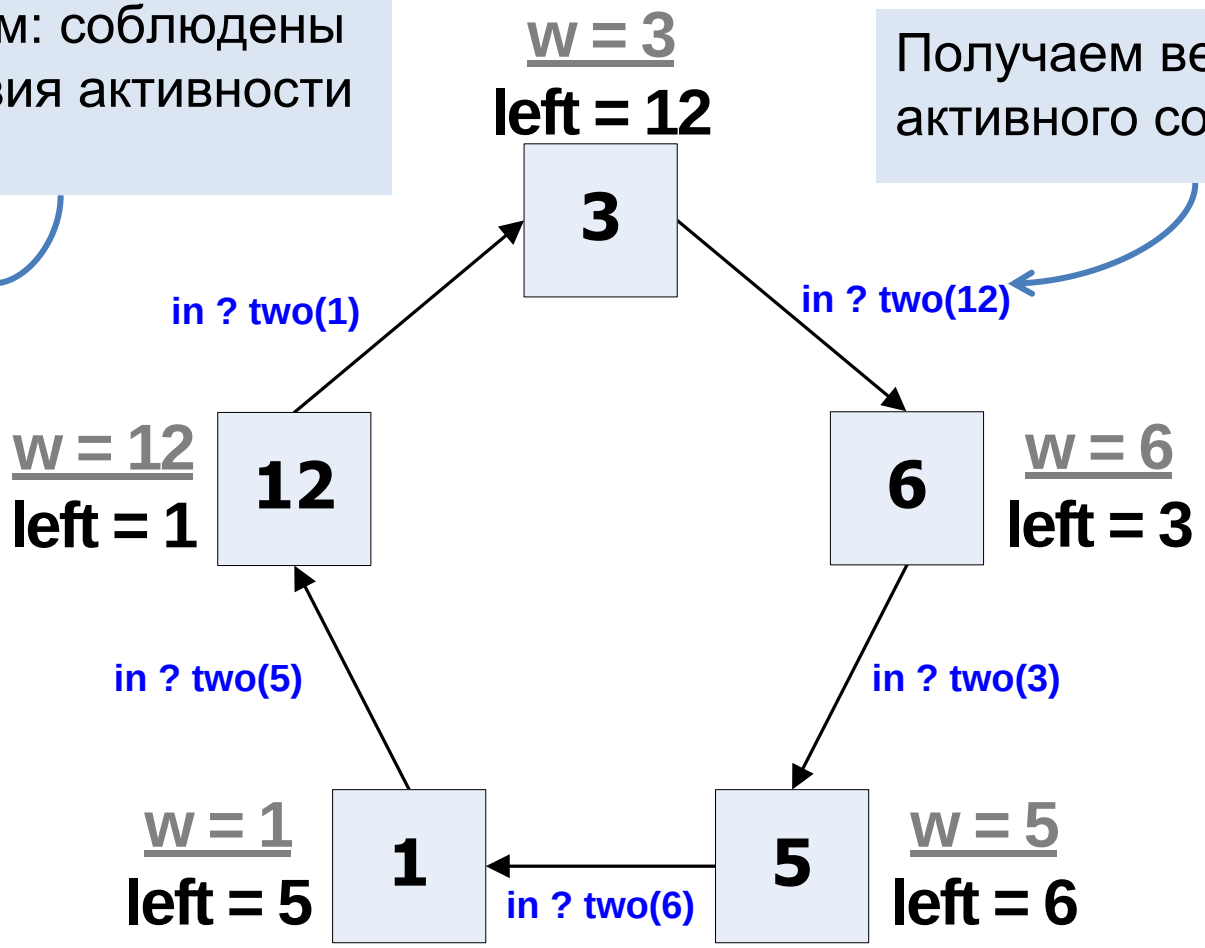
Пример. Фаза A1



Пример. Фаза A2

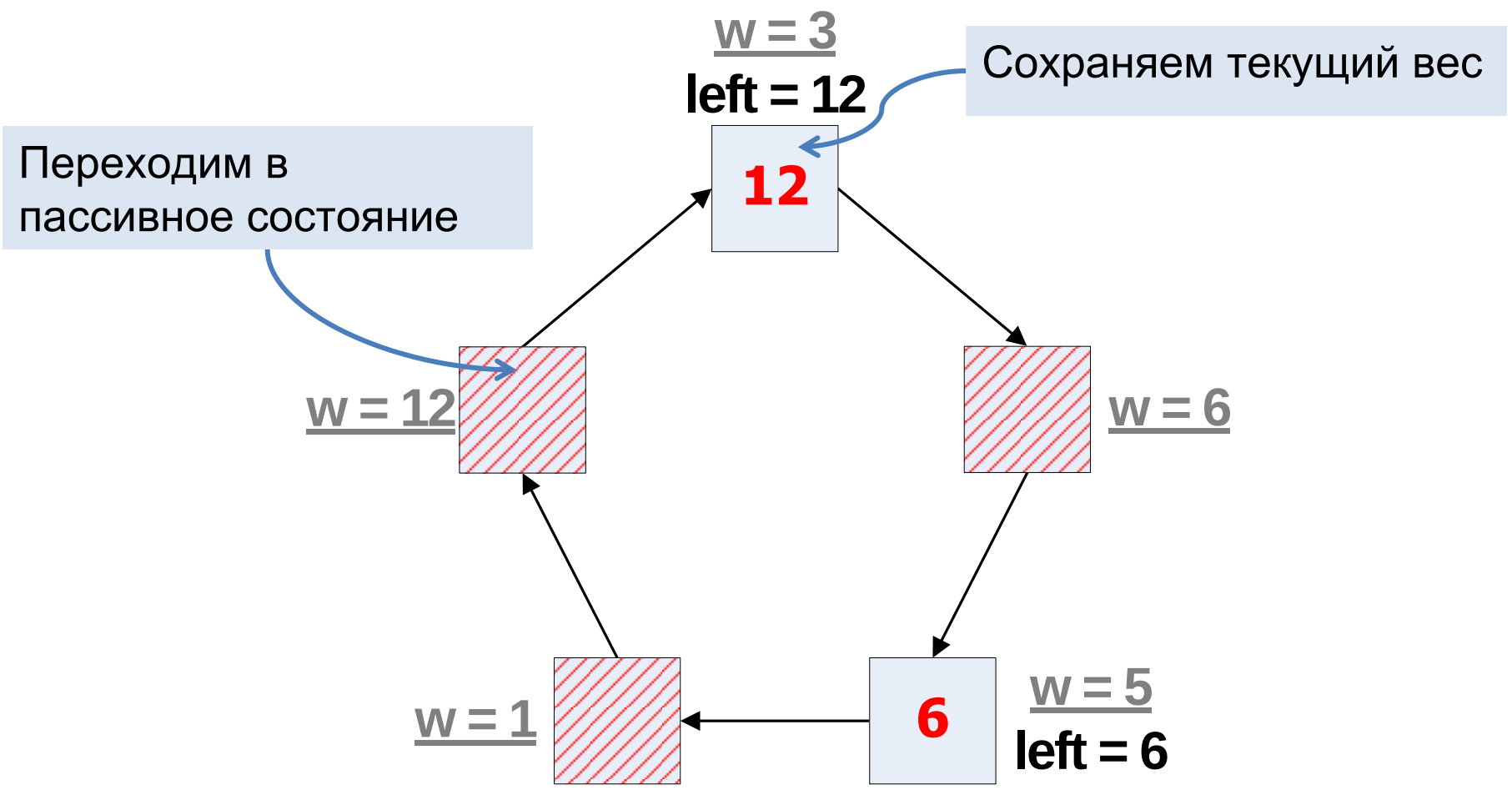
Проверяем: соблюдены ли условия активности узла?

Получаем вес активного соседа слева

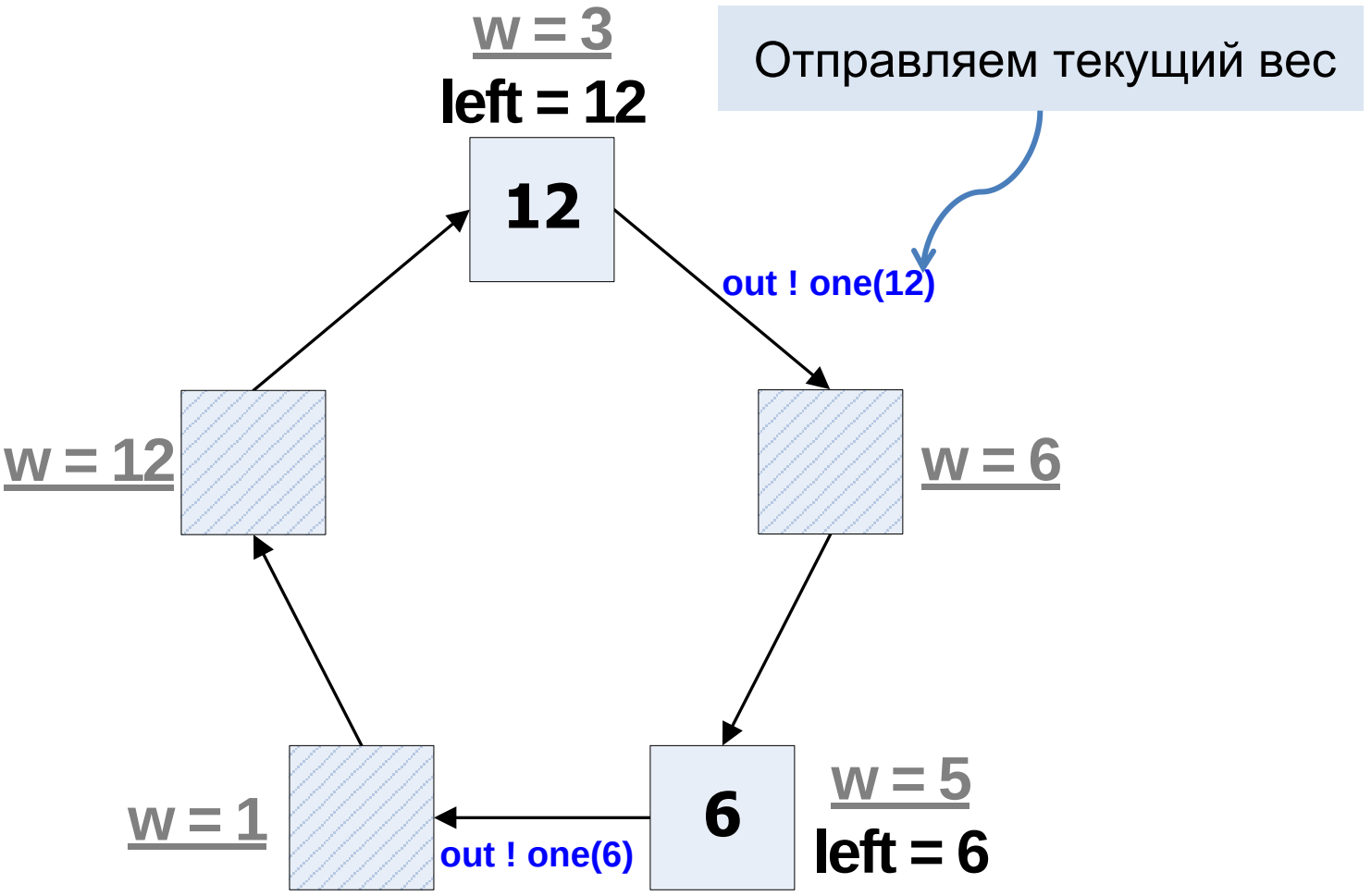



```
if((left>max) && (left>q)) -> (max:=left; send one(max))
else узел стал пассивным
```

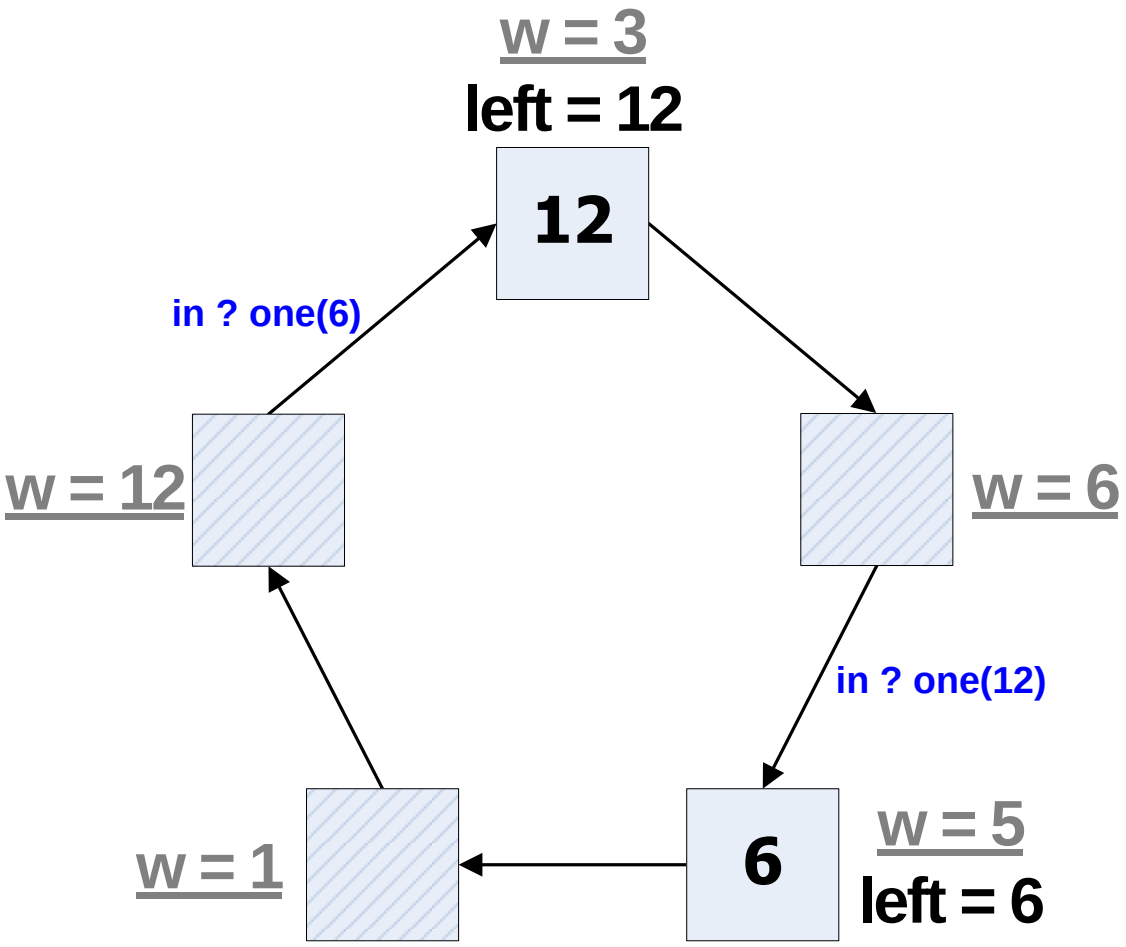
Пример. Фаза A2



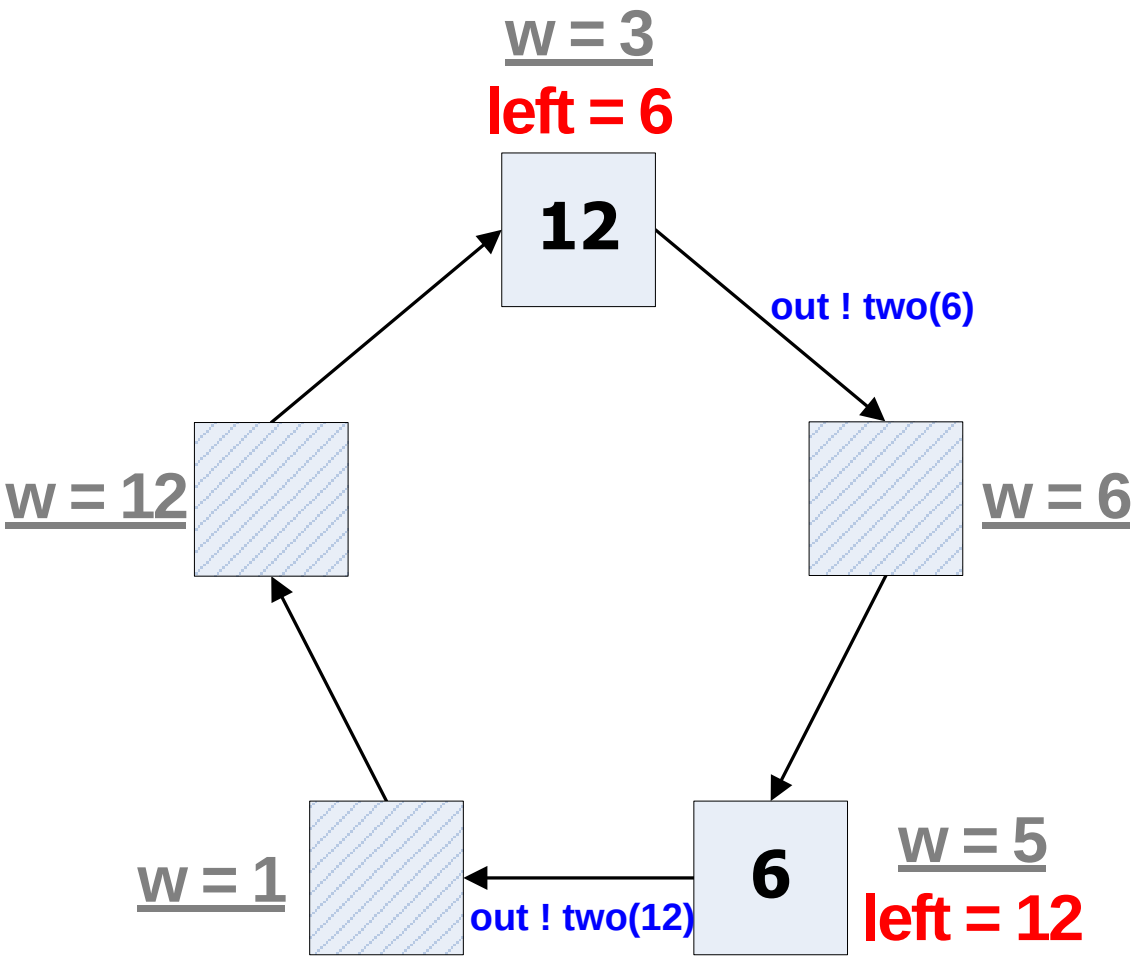
Пример. Фаза A2



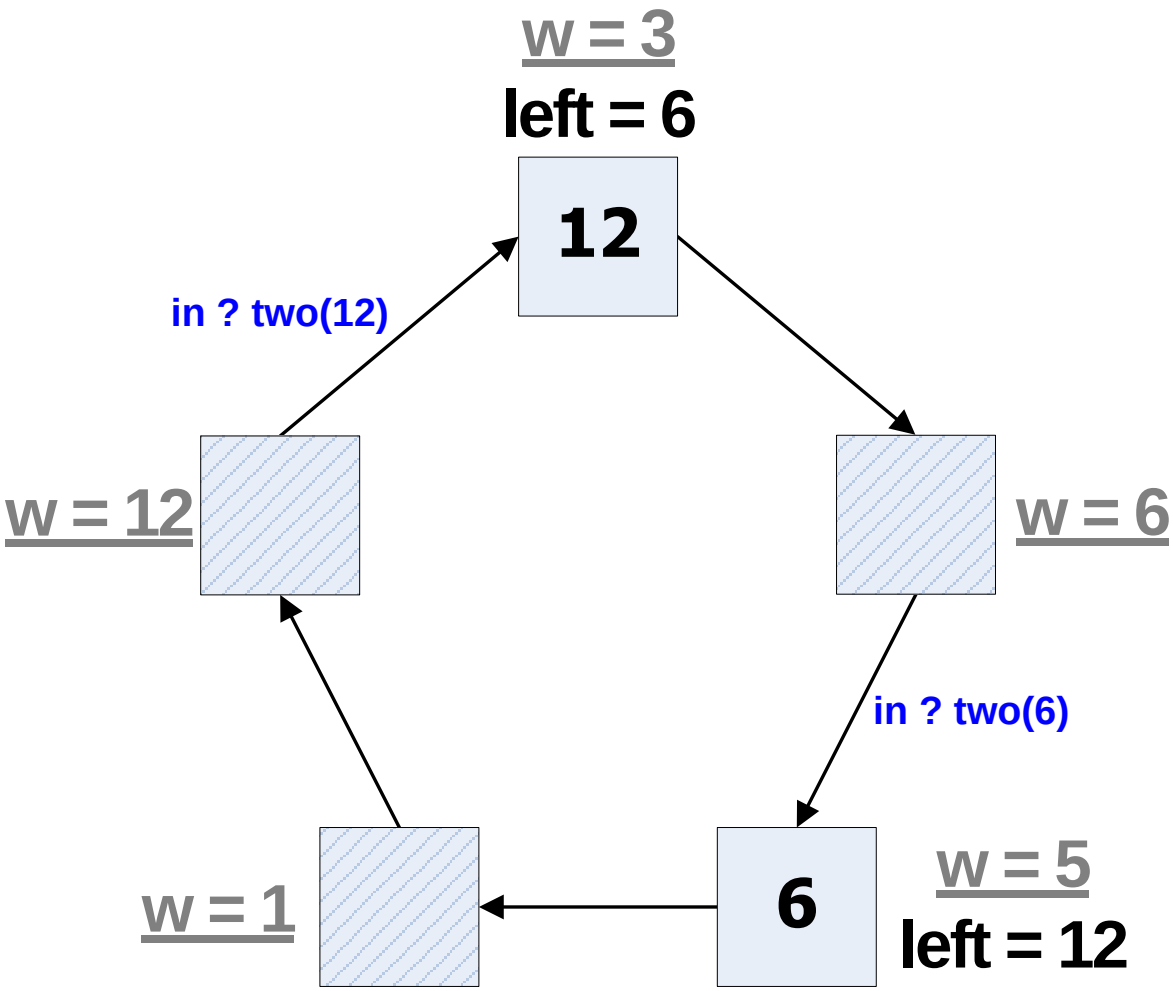
Пример. Фаза A1



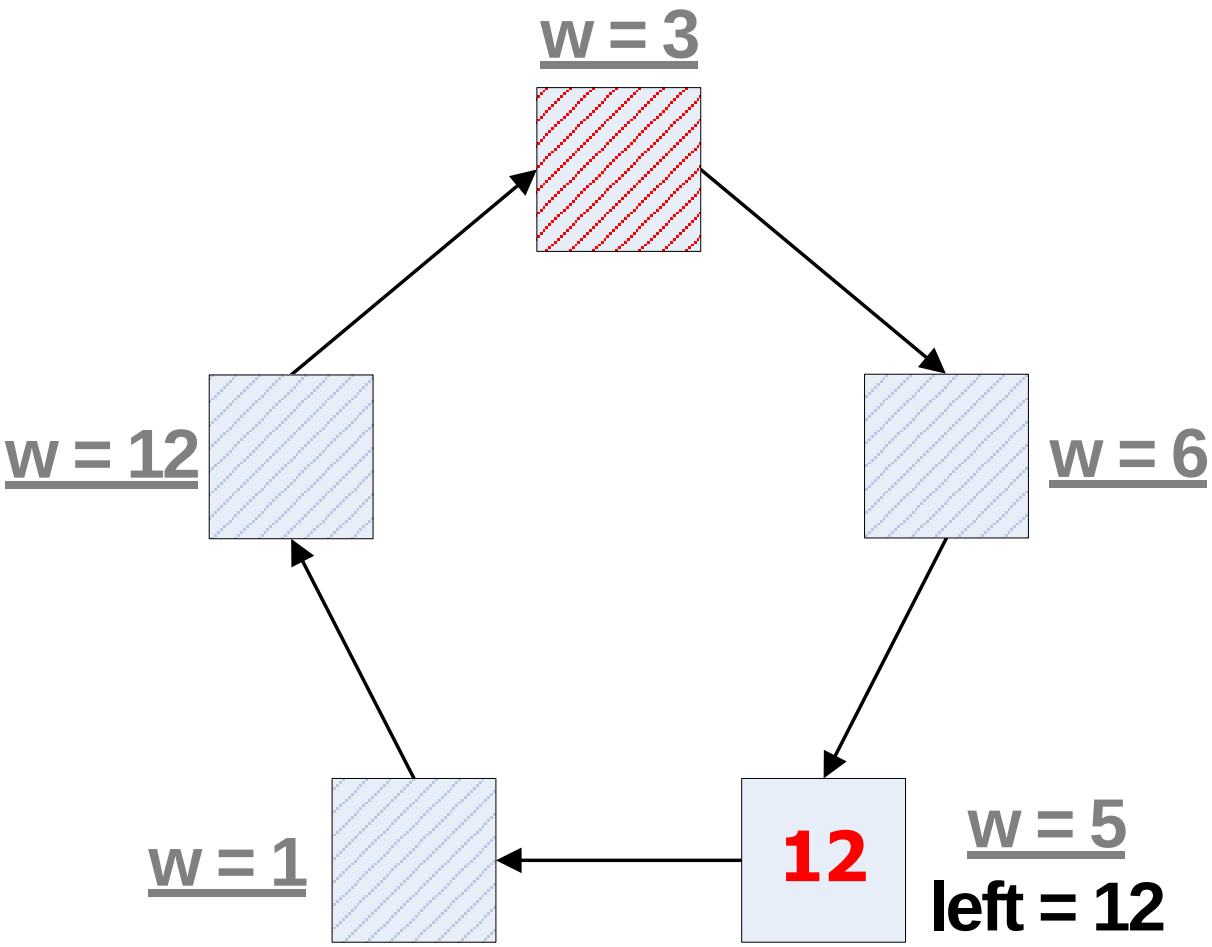
Пример. Фаза A1



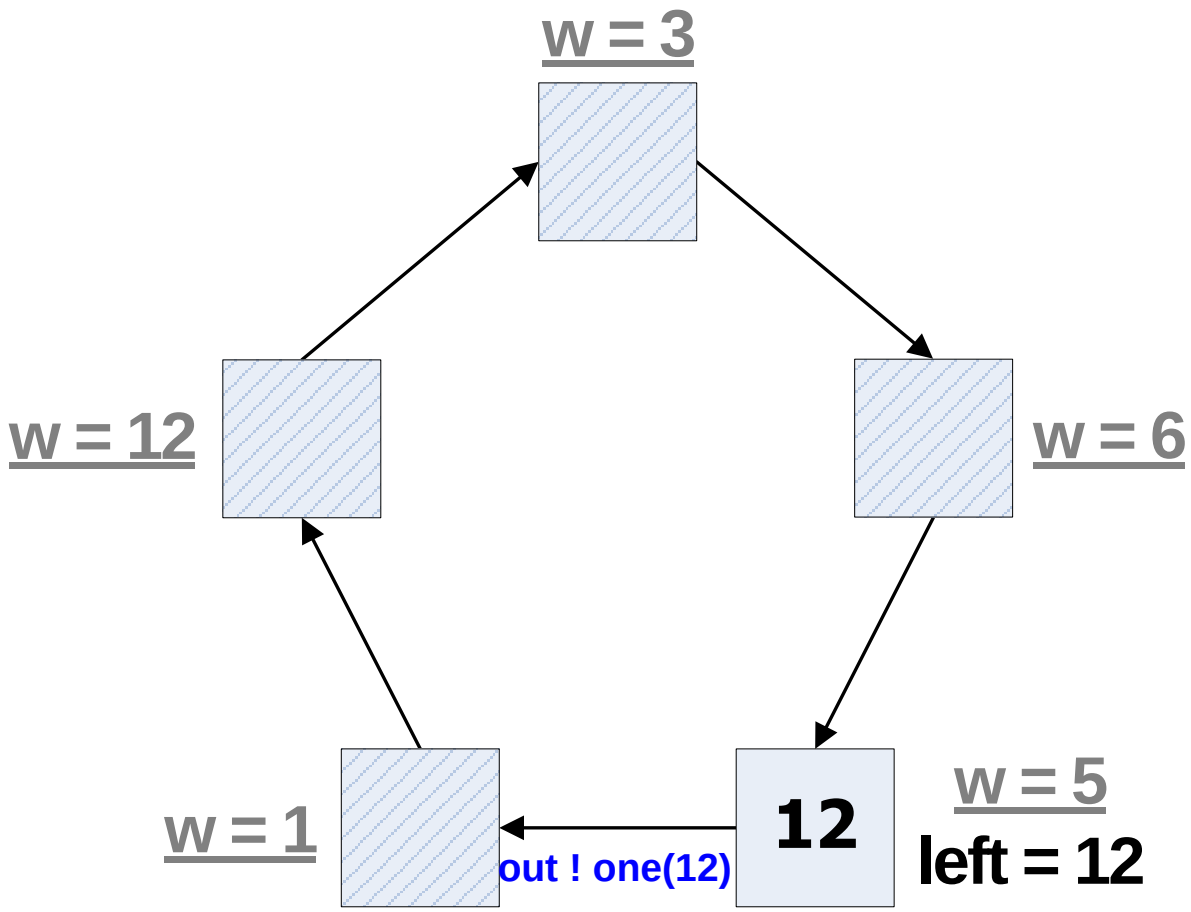
Пример. Фаза A2



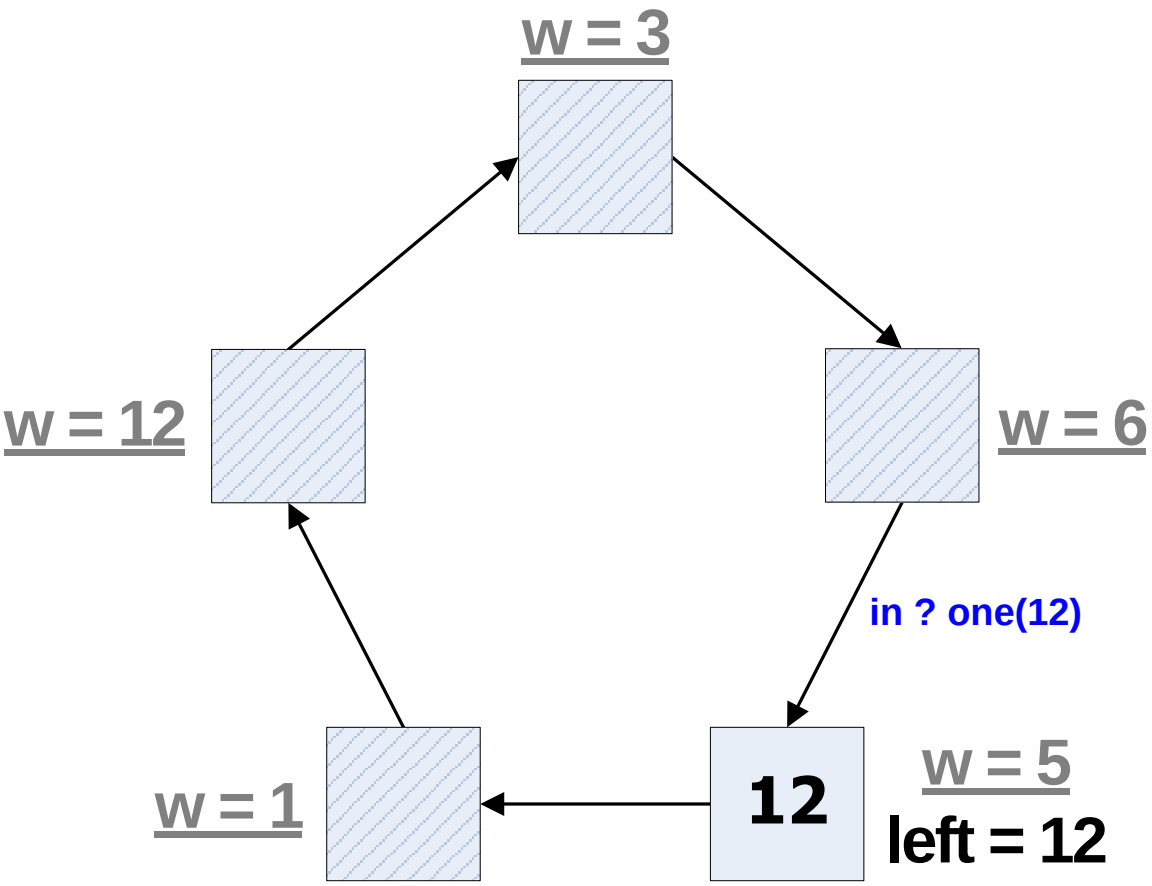
Пример. Фаза A2



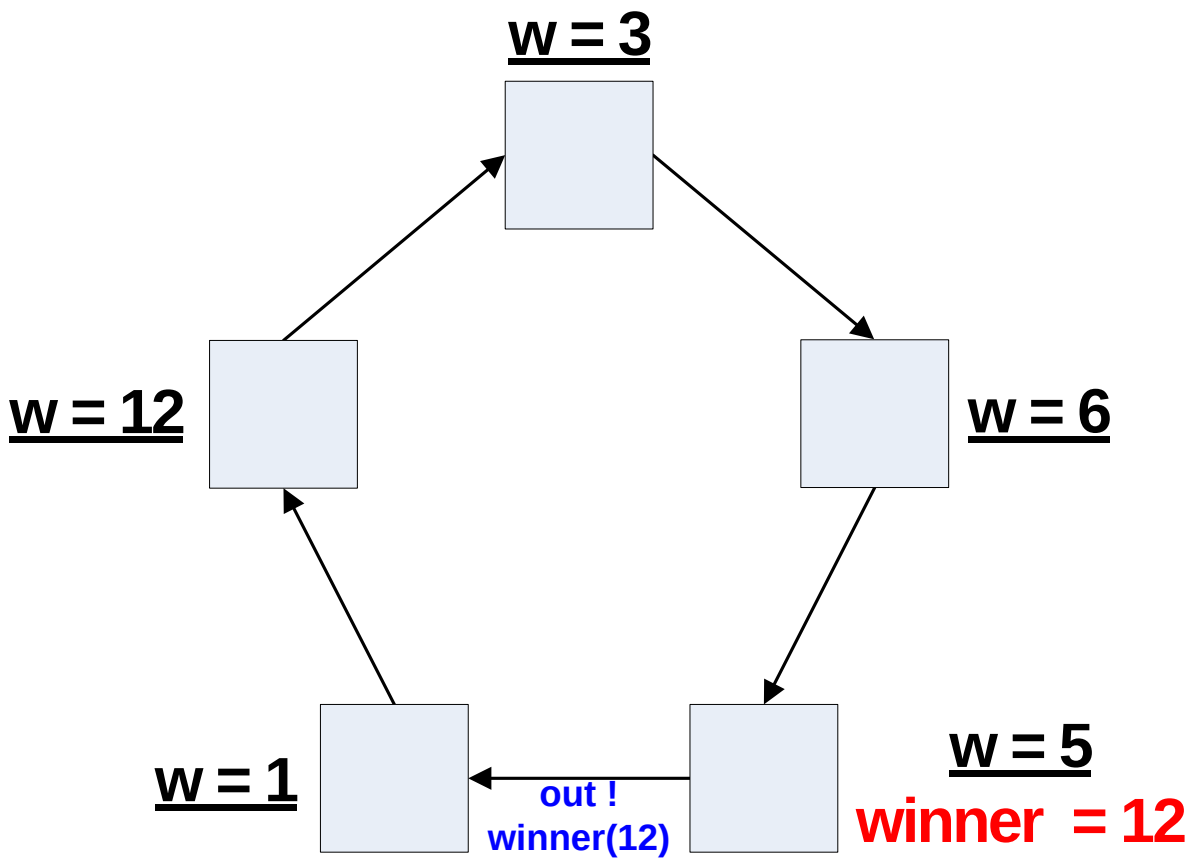
Пример. Фаза A2



Пример. Фаза A1



Пример. Фаза A3. Лидер найден



Глобальные свойства, обеспечивающие корректность алгоритма

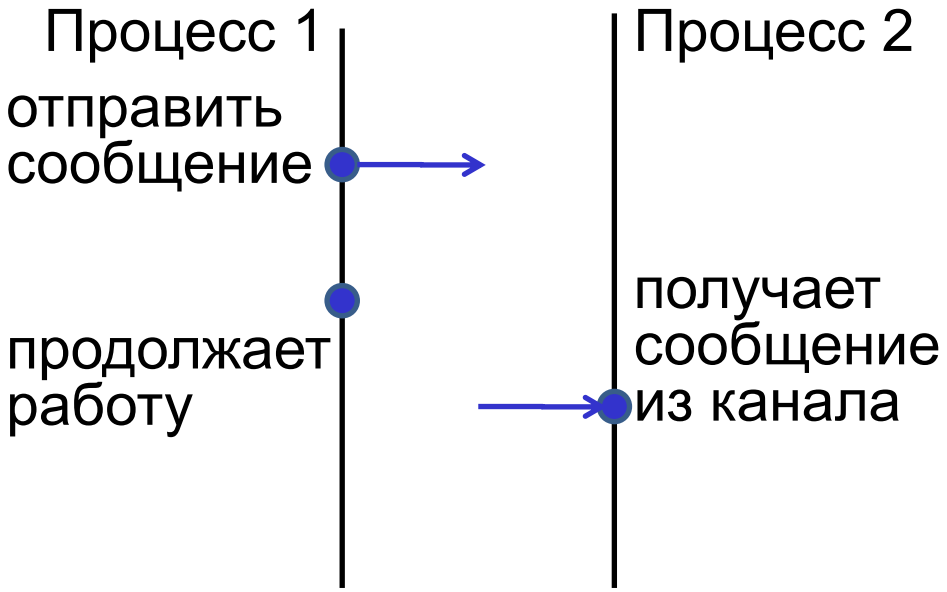
- На любой фазе в кольце есть только один процесс с максимальным весом

- На любой фазе процесс или характеризуется текущим весом, который является наибольшим из двух активных соседей слева, или не распространяет свой текущий вес

Эти два правила являются *глобальными инвариантами*

Формулировка глобальных инвариант и формальное доказательство
требуют понимания алгоритма

Асинхронные взаимодействия

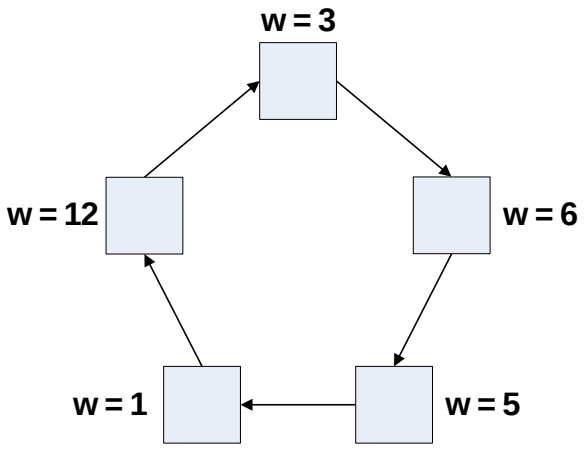


Сообщения алгоритма выбора лидера

- `one(q)` – сообщение со своим текущим весом
- `two(q)` – сообщение с текущим весом своего соседа слева
- `winner(q)` - сообщение «лидер-найден»
- `q` – один из весов, хранимых узлом

- Порядок получения сообщений зависит от дисциплины обслуживания канала
- В Promela дисциплина обслуживания FIFO
- Каналы имеют конечную емкость
- Сообщение может быть записано в канал, если емкость канала не исчерпана
- Процесс может прочитать сообщение из канала, если канал не пуст

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};
```

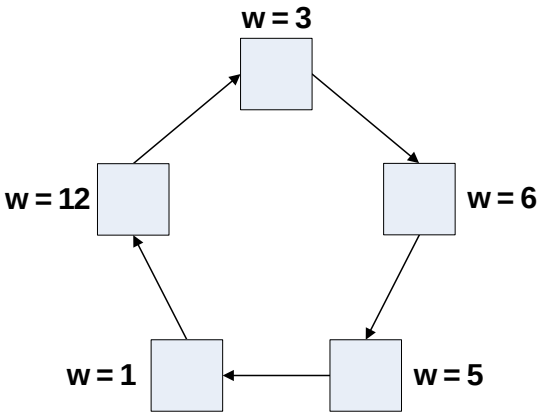
формат сообщения,
например, `one(q)`
то же, что и `one, q`

объявление массива каналов

Структура программы на Promela

Глобальные переменные

```
// 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 {
/* . . . */
}
```



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)

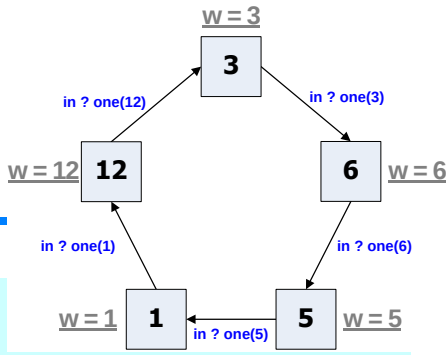
A2. Получить сообщение two(q). Обработать сообщения. Отправить сообщения one(q)

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                       // проверяем полученное значение
        :: q != max ->           // 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 != max, то left:=q и послать сообщение two(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

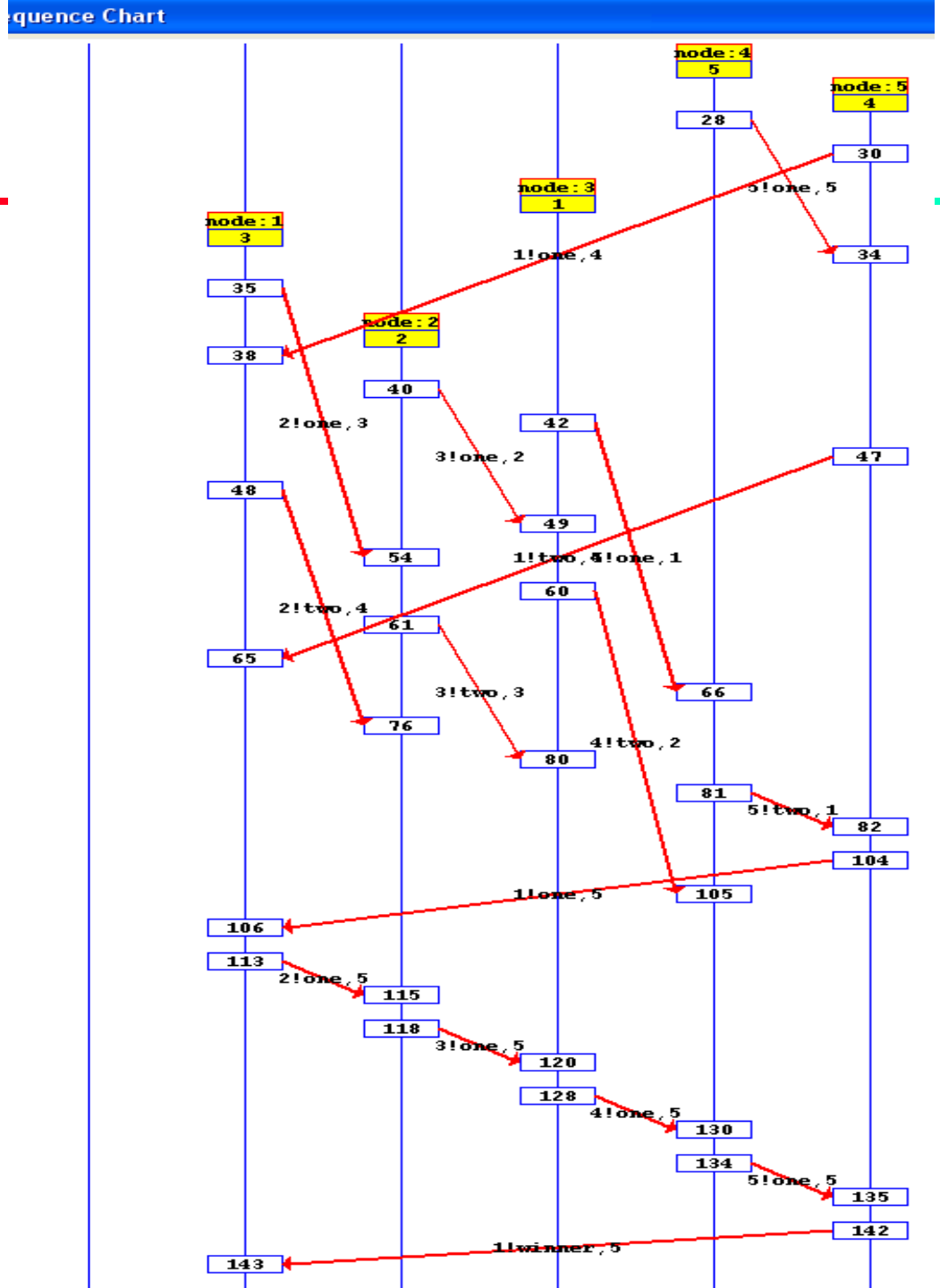
```
:: in ? winner(q) ->  // получено сообщение «лидер выбран»

   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+I-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
- $dom(\mathrm{c})$ • область определения канала c
- $cap(\mathrm{c})$ • емкость канала c

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

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
```

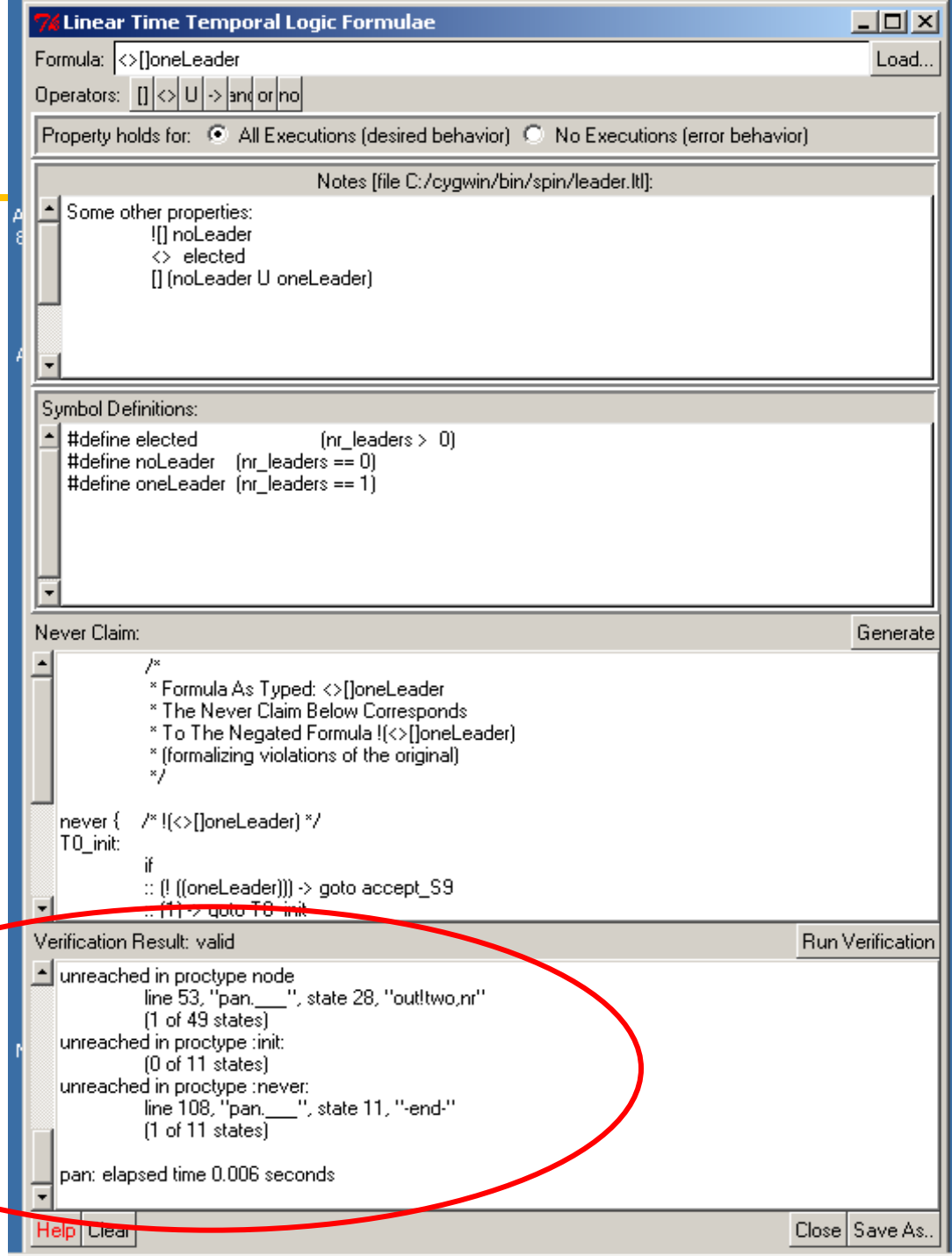
- лидер в конце концов будет выбран

```
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: <>[]oneLeader
Load...
Operators: [] <> U -> and or no
Property holds for: All Executions (desired behavior) No Executions (error behavior)
Notes [file C:/cygwin/bin/spin/leader.ltl]:
Some other properties:
![] noLeader
<> elected
[] (noLeader U oneLeader)
Symbol Definitions:
#define elected (nr_leaders > 0)
#define noLeader (nr_leaders == 0)
#define oneLeader (nr_leaders == 1)
Never Claim:
Generate
/*
* Formula As Typed: <>[]oneLeader
* The Never Claim Below Corresponds
* To The Negated Formula !(<>[]oneLeader)
* (formalizing violations of the original)
*/
never { /* !(<>[]oneLeader) */
T0_init:
if
:: (! ((oneLeader))) -> goto accept_S9
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

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

Задачи

- Найти ошибку в задаче про автомат напитков
- Изменить алгоритм выбора лидера в однонаправленном кольце так, чтобы верификация не проходила. Например, нарушим свойство «Всегда у лидера будет наибольший вес»
- Проверить свойство «На всех путях всегда общее число проголосовавших равно 100%»