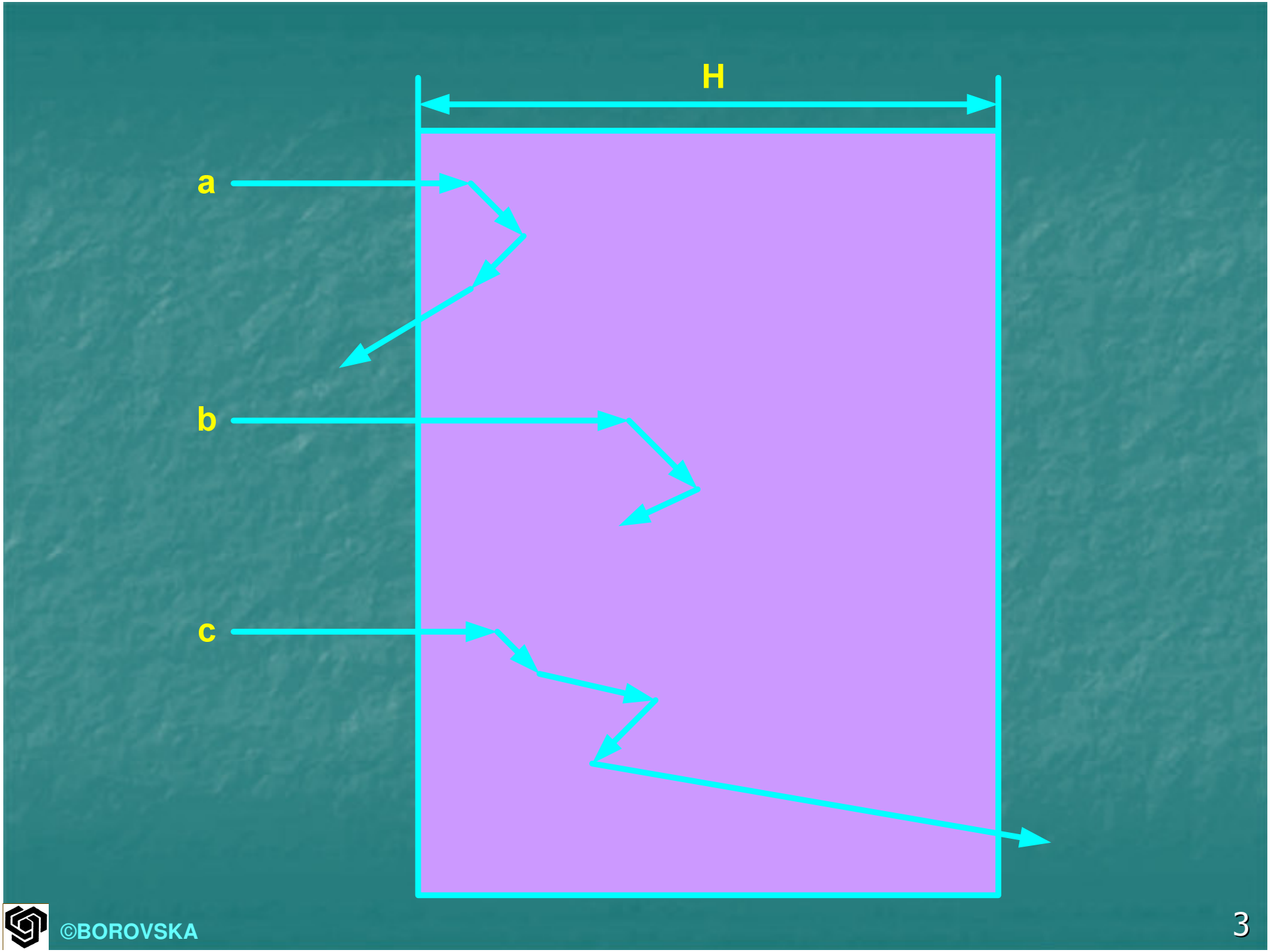
ПРИЛОЖЕНИЯ НА МЕТОДА МОНТЕ КАРЛО

ДВИЖЕНИЕ НА НЕУТРОНИ

- Разглеждаме опростен модел на движението на неутроните при 2 измерения
- Източникът излъчва неутрони срещу хомогенна повърхност с дебелина H и неограничена височина
  - *Съществуват 3 възможности:*
    1. *Отразяване на неутрона*
    2. *Абсорбиране на неутрона*
    3. *Преминаване през преградата*



Цел: Да се изчисли честотата на трите събития като функция на дебелината на преградата H.

- $C = C_c + C_s$ – константа, отразяваща взаимодействието на неутроните в преградата
  - C_c - за поглъщане
  - C_s - за разпръскване
- L – разстоянието, преди неутронът да взаимодейства с атом в преградата, се моделира с експоненциално разпределение със средна стойност $1/C$

$$L = -\frac{1}{C}\ln u$$

u – случайно число с равномерно разпределение в [0,1)

C_s/C – вероятност за удар с атом и отражение

C_d/C - вероятност за абсорбиране от атом

При преминаване през преградата – новата посока D (в rad) се моделира със случайна величина, равномерно разпределена между 0 и π – разстоянието между колизиите в преградата е ***LcosD***

Симулацията на движението на неутрона продължава докато възникне едно от следните събития:

1. Неутронът се абсорбира от атом
2. Позицията x на неутрона е отрицателна, което означава, че той е отразен от преградата
3. Позицията x на неутрона е > H, което означава, че той е преминал през преградата

Време по Монте Карло – симулацията се развива от едно събитие към друго събитие

Симулация на движението на неутроните

C – средното разстояние на взаимодействията между неутрон и атом е 1/C

C_s- компонента на C за разпръскване

C_c - компонента на C за абсорбиране

H – дебелина на преградата

L – разстояние преди колизия

d – посока на неутрона

u – случайно число с равномерно разпределение

x – координата на неутрона ($0 \leq x < H$)

a – "истина", докато неутронът се движи

r,b,t – брой на отразените, абсорбираните и преминалите през преградата неутрони

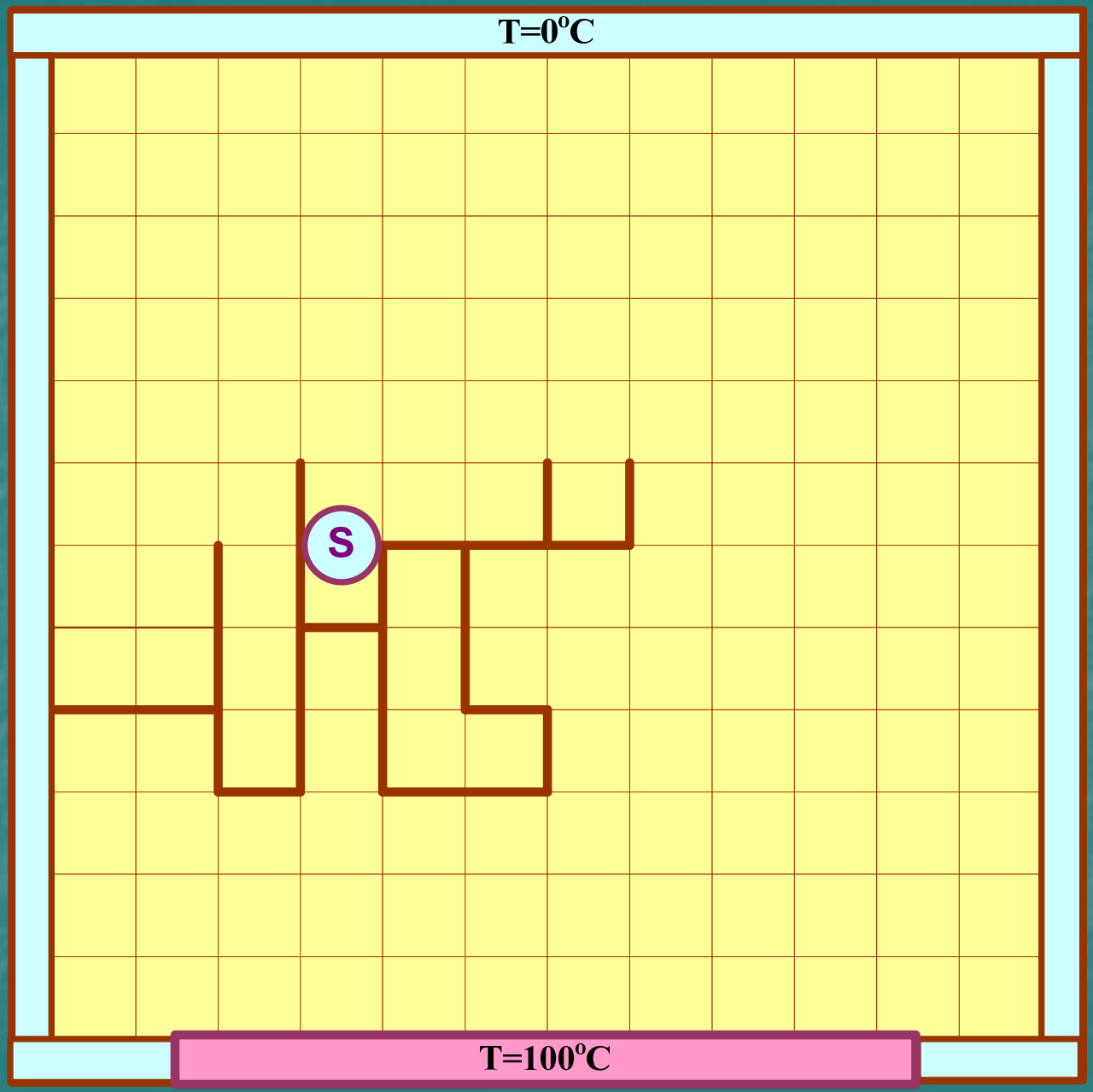
```
begin
  r,b,t ← 0
  for i ←1 to n do
     d ← 0
     x ← 0
     a ← true
     while a do
          L ← -(1/C) * ln u
          x ← x + L * cos (d)
          if x<0 then {reflected}
            r ← r + 1
            a ← false
```

```
        else if x>= H then {transmitted}
        t ← t+1
           a ← false
        else if u < Cc/C then {absorbed}
           b ← b + 1
           a ← false
        else
           d ← u * π
        endif
      endwhile
    endfor
    print r/n, a/n, t/n
end
```

Температура на точка от двумерна повърхност

- Много тънка плоча от хомогенен материал
- Да се измери t^o в дадена точка S
- В крайните точки t^o е фиксирана
- Вътрешното разпределение на t^o се описва от уравнението на Лаплас $\Delta^2 T = 0$
- В дадена точка t^o е средната стойност на t^o в съседните точки
- Произволно избираме 1 от съседите и прибавяме неговата t^o в акумулатор

- За да определим t^o в S, разделяме натрупаната сума в акумулатора на броя на пробите n, като средната очаквана стойност е $(T_n+T_s+T_e+T_w)/4$.
- Аналогично, се определя t^o на съседите (рекурсивно)
- "разходка" по случаен маршрут – условие за край - t^o в крайните точки е фиксирана
  - Алгоритъмът се терминира, при задоволителна сходимост на t^o

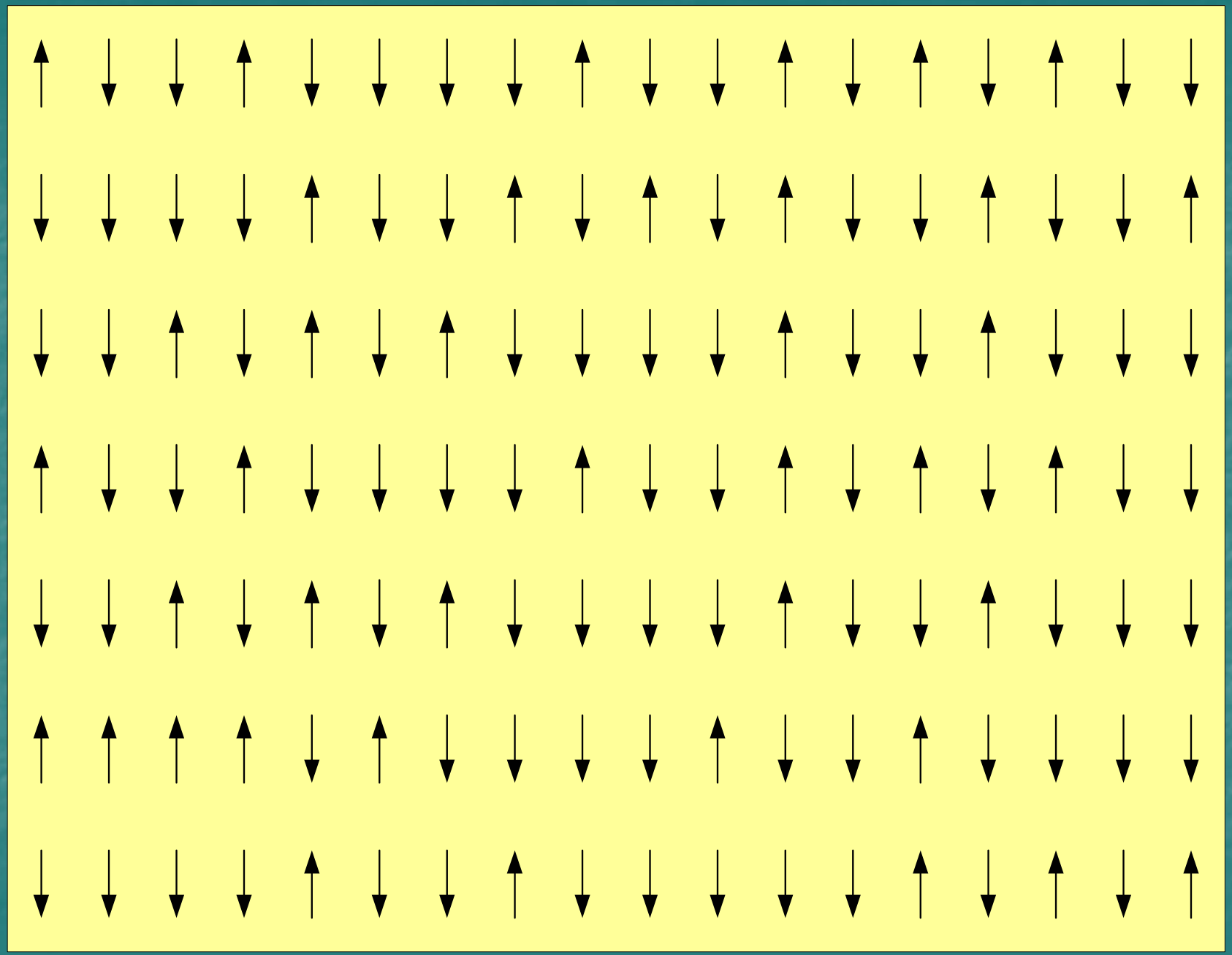


Моделиране на прост магнит

- Домейн на областта – квадратна решетка
- Всяка клетка – сайт
- Всеки сайт σ_k има спин –
  up (1/2), down (-1/2)
- Енергията на системата се определя с

$$E(\sigma) = -\sum_{i,j} J\sigma_i\sigma_j - B\sum_i \sigma_i$$

Първата сума е от най-близките съседи, J – конст. за силата на взаимодействието между спиновете, B – влияние на външното магнитно поле



- Енергията на системата е функция на спиновете
- При модел 20x20 - може да бъде в едно от 2^{400} състояния
- Вероятността да влезе в едно от тези състояния зависи от текущото състояние и температурата
- Целта е да се изчисли енергията на всяка частица
- Случайната проба x_i представя конфигурацията на спиновете
- ***Търсим конфигурации с по-висока вероятност***

Алгоритъм Metropolis

- Използва текущата конфигурация (текущата случайна проба) за да генерира следващата конфигурация (следващата случайна проба)
- При дадена проба x_i, генерира съседна конфигурация x'
  - Ако $E(x')<E(x_i)$, то $x_{i+1}=x'$
  - Ако $E(x')>E(x_i)$, то $x_{i+1}=x'$

с вероятност $e^{-[E(x')-E(x_i)]/kT}$

- ***Последователността от случайни проби се нарича верига на Марков (Markov chain) и представлява случайна разходка през възможните конфигурации***
- Късите последователности от случайни проби по алгоритъма Метрополис са силно корелирани
- При многобройни проби, алгоритъмът осигурява добро покритие на цялата функция на плътността на разпределение

Проблемът за разпределение на стаите

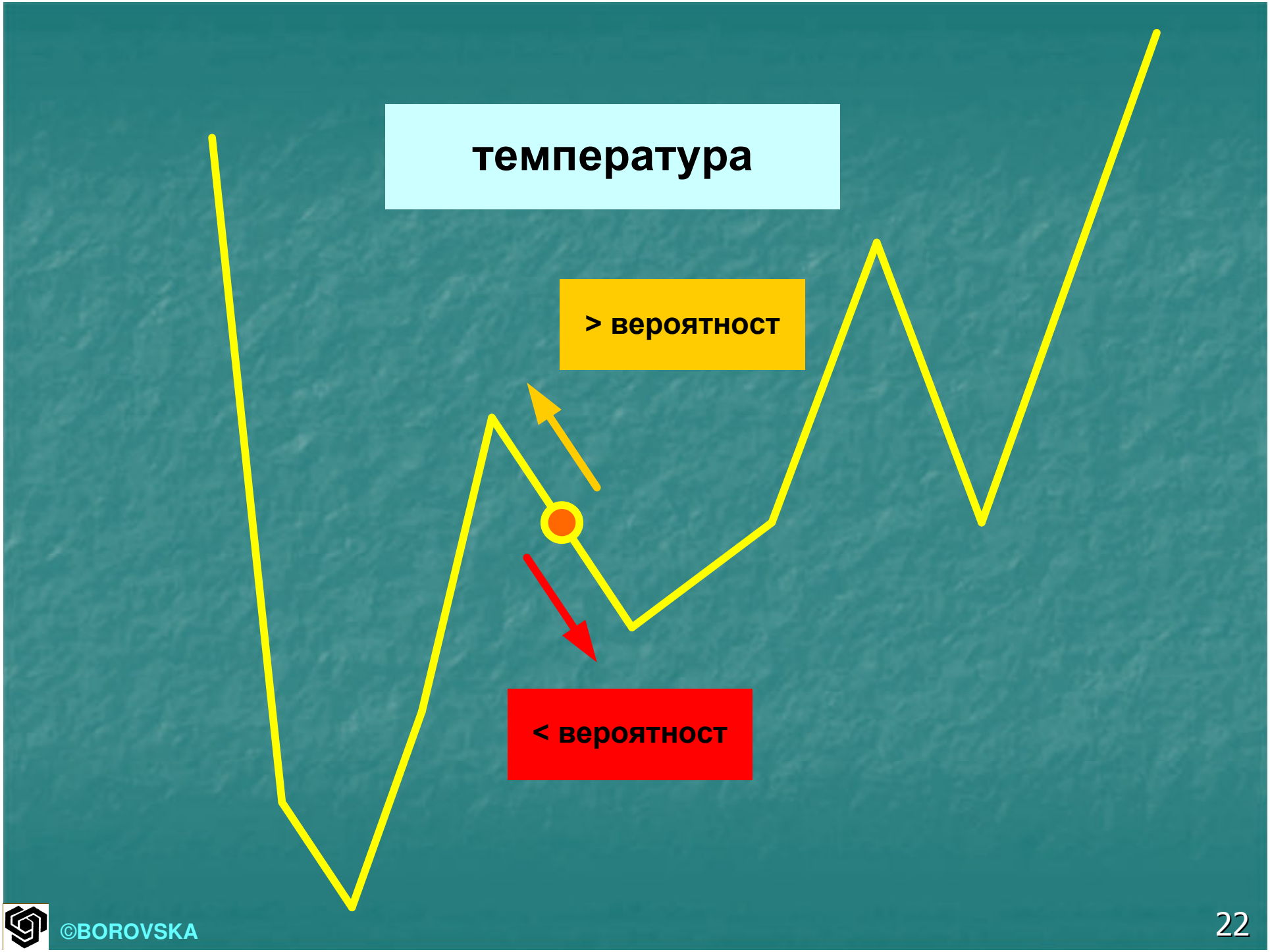
- n първокурсници трябва да бъдат разпределени в n/2 стаи като се минимизират междуличностните конфликти
- Програма е генерирала таблица – всяка позиция (i,j) показва различията между студент i и студент j
  - Таблицата е симетрична
  - Решение с техниката “закаляване”

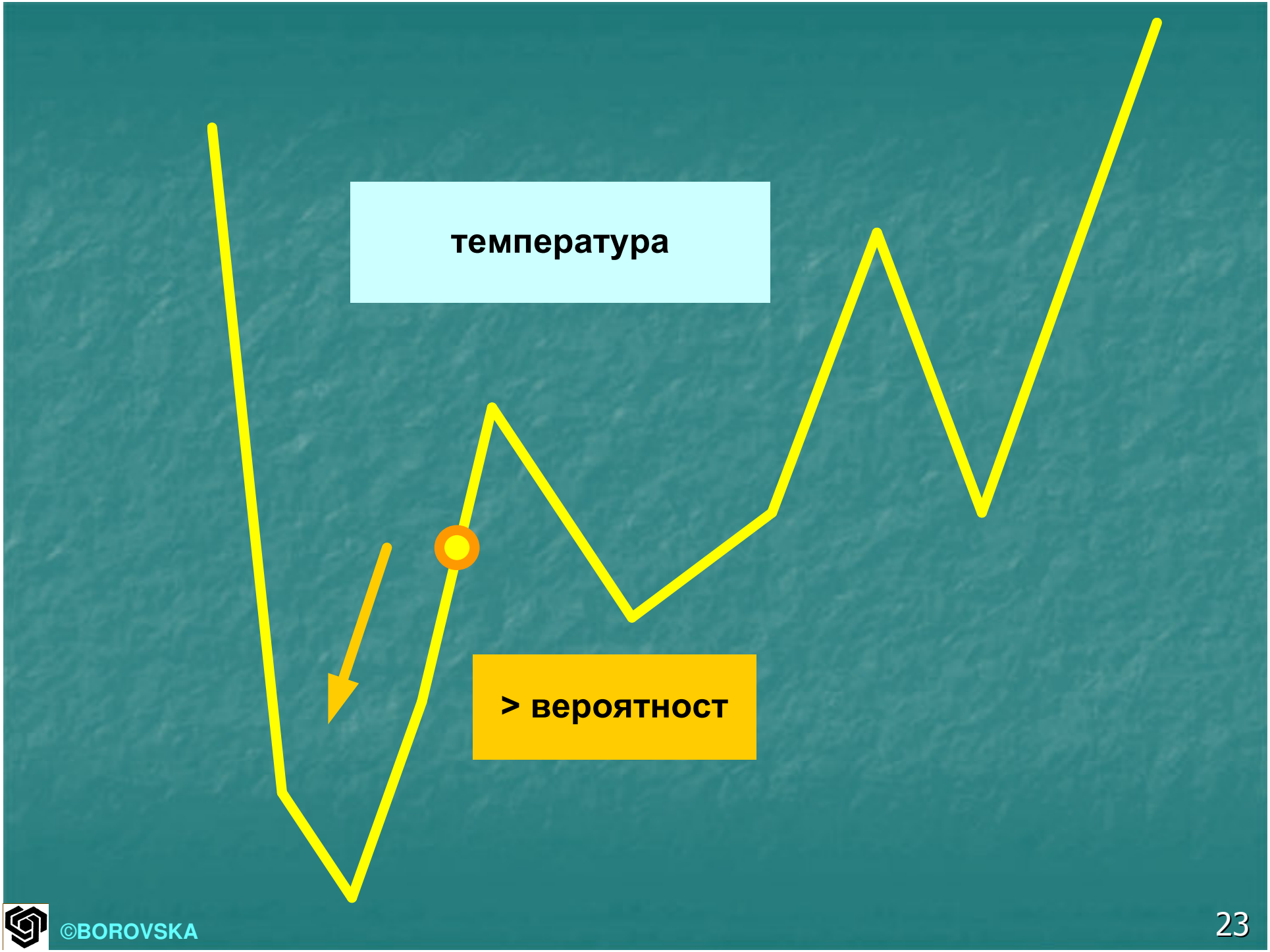
- Симулираното закаляване е итеративен алгоритъм
- При всяка итерация текущото решение се променя случайно за получаване на алтернативно решение
- Новото решение става текущо, ако стойността на целевата функция е по-малка
- В противен случай новото решение става текущо с вероятност $e^{-\Delta/T}$
- Решенията обикновено имат локални минимуми, при високи температури – лесно се излиза от локалните минимуми

- Аналогия с алгоритъма Метрополис – използват една и съща функция на плътността на разпределението за да се вземе решение за преминаване в състояние с по-висока енергия
- *Процедура за алгоритъма с каляване*

1. Как да се представят решенията?
2. Да се определи функцията на цената!
3. Как да се генерира съседно решение от съществуващо решение?
4. Да се дефинира функция на "охлаждане"!

- ***Симулираното каляване винаги осигурява възможност за търсене на ново решение с по-ниска цена***
- Вероятността за генериране на решение с по-висока цена намалява с понижаването на температурата
- Генерирането на решение с по-висока цена е с по-голяма вероятност при високи температури
- При ниски температури тази вероятност е значително по-малка





- Матрица на несъвместимостта D
- $d_{i,j}$ е число с плаваща точка между 0 и 10, показваща степента на несъвместимост
  - $d_{i,j}=d_{j,i}$
- Решението представлява разпределението на n студента в n/2 стаи
- Масив a съдържа разпределенията
- a_i е цяло число между 0 и n/2-1, представящ номера на стаята, разпределена за студента i

- Функцията на цената – сумата от несъвместимостите на студентите в стаите
- r_i – съквартирантът на студент i
- Определяме функцията на цената като

$$\sum_{i=0}^{n-1} d_{i,r_i}$$

- Генерираме ново решение като избираме произволно студенти и разменяме техните разпределения
- Накрая трябва да изберем функцията на температурата – в значителна степен определя бързодействието на алгоритъма

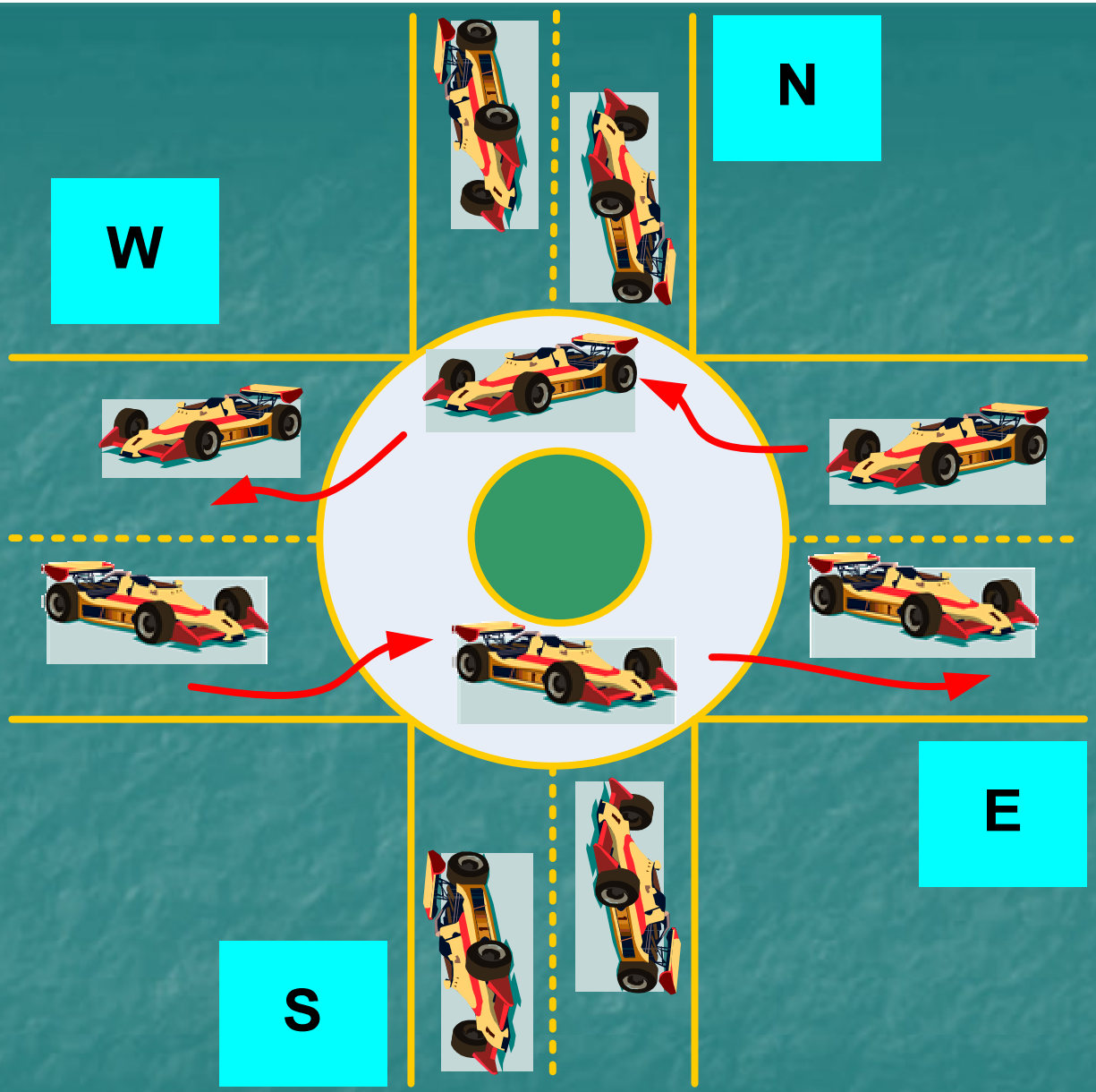
- Избираме $T_{i+1} = 0.999T_i$ при $T_0 = 1$
- Алгоритъм с $T_0=10$ е два пъти по-бавен от този с $T_0=1$.

ГАРАЖ

S клетки, колите пристигат

с Поасоново разпределение със средна стойност А минути, престой – нормално разпределение със средна стойност М минути и стандартно отклонение М/4;
да се определи средният брой заети клетки и вероятността за отказ при пълен гараж

Симулация на кръгов трафик



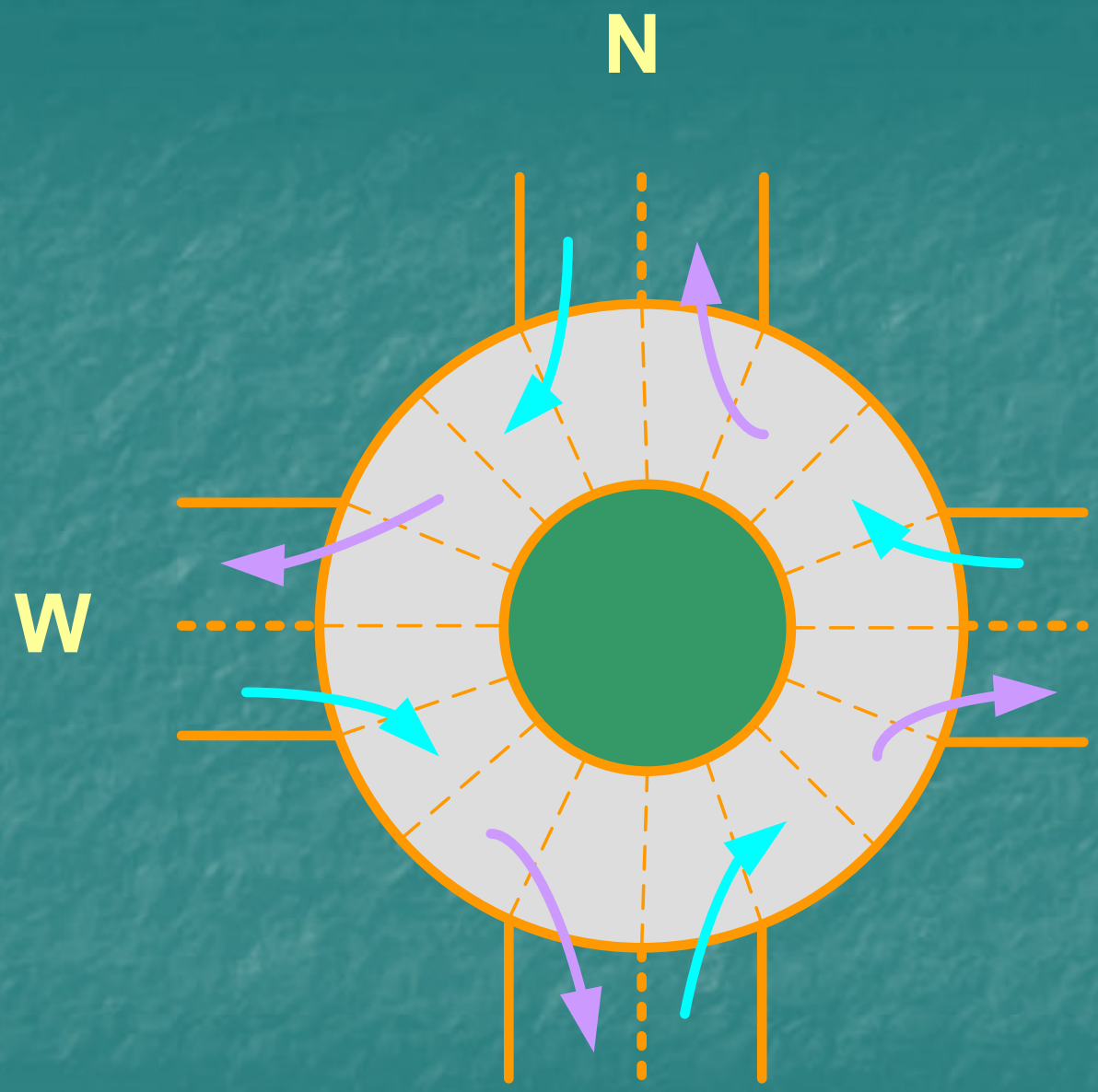
КРЪГОВ ТРАФИК
БЕЗ СВЕТОФАРИ

Цели на симулацията

1. За всяка от четирите входни точки на кръговия трафик *каква е вероятността пристигащо МПС да чака за присъединяването си към кръговия трафик*?
2. За всяка от четирите входни точки на кръговия трафик *каква е средната дължина на опашките от МПС, желаещи да се присъединят към кръговия трафик*?

Симулация на кръгов трафик

- Разделяме кръговия трафик на 16 секции
  - В рамките на 1 стъпка на симулацията МПС в кръга се придвижват към следващата секция по посока обратна на часовниковата стрелка или напускат кръга през един от четирите изхода
- МПС в кръга имат по-висок приоритет от МПС, чакащи да се включат в кръговото движение



Симулация на кръгов трафик

- МПС в кръга винаги могат да се придвижат в рамките на стъпката
- МПС, чакащи да се включат в кръговия трафик, се придвижват, ако няма МПС от кръговия трафик, което трябва да се придвижи към входната секция
- Трябва да е известна честотата на пристигане на МПС, желаещи да се присъединят към кръговия трафик

Симулационен модел

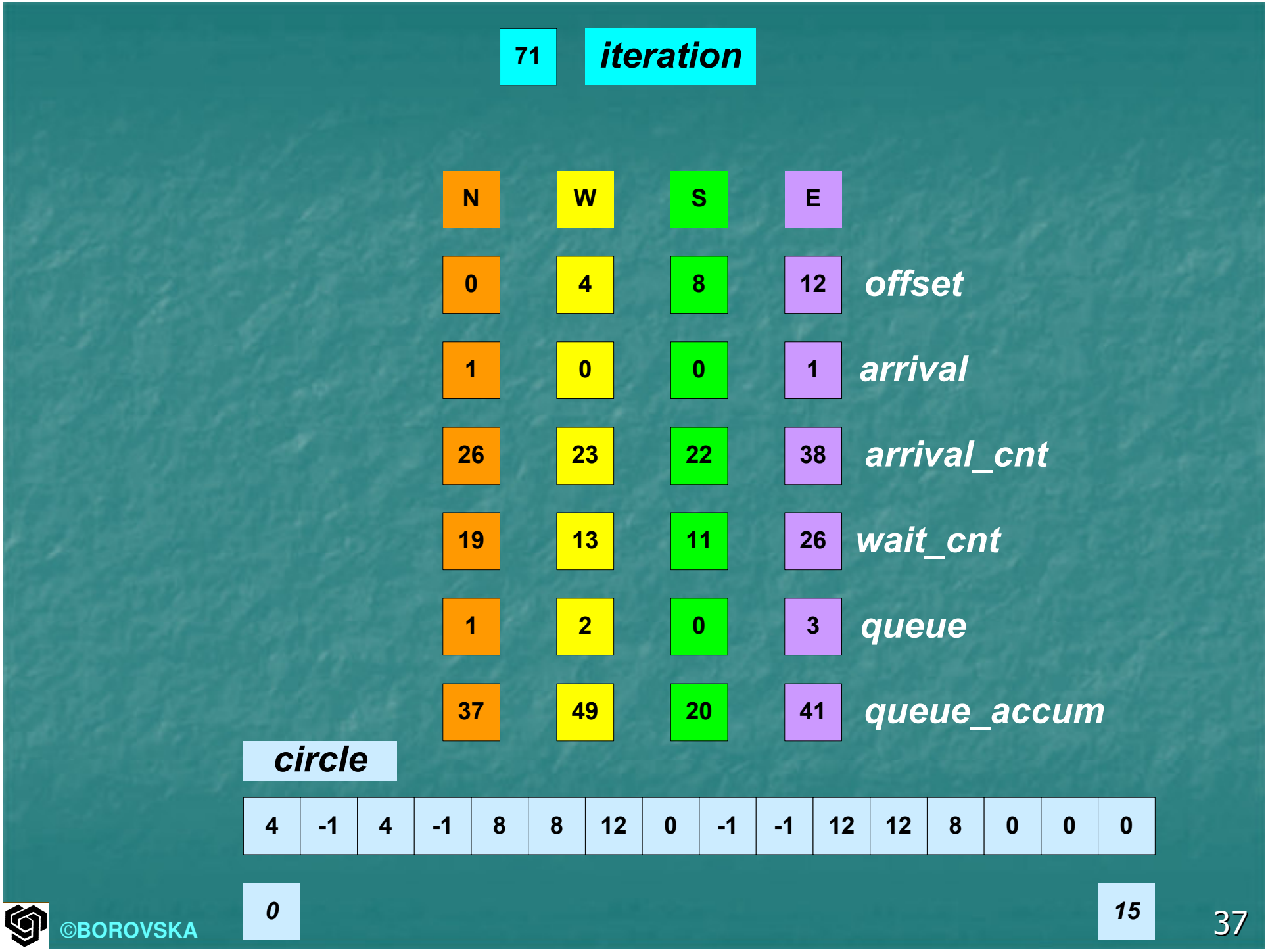
- Масив f_i съхранява средното време на пристигане на МПС във входна точка ***i***
  - Елементите на матрицата $d_{i,j}$ представляват вероятността МПС, влязло през входна точка i да напусне кръга през изходна точка j
  - Кръговият трафик се моделира посредством кръгов буфер, имплементиран като масив от 16 int

- Масив *offset* съхранява индексите в кръговия буфер, свързани с 4-те входни и 4-те изходни точки
  - Индекс 0 представя северния вход/изход, индекс 4 – западния и т.н.
  - Всеки елемент от масива *circle* представя секция, която може да бъде празна *circle[i]=-1*, или да *съдържа 1 МПС → circle[i] съдържа номера на изходната точка (0,4,8 или 12)*
- *Масив arrival – при пристигане на МПС при входна точка*

съответният елемент → 1

- Масив arrival_count – общ брой пристигнали МПС на всяка от входните точки
- Масив wait_cnt – общ брой на МПС, на които се е наложило да чакат при входните точки
- Масив queue – дължина на опашките от чакащи МПС при всяка от входните точки
- Масив queue_accum – натрупана сума от дължините на опашките queue за цялото време на симулацията


Входни точки
N
W
S
E
f
3
3
4
2
Средно време за пристигане
Изходни точки
Входни точки
D
N
W
S
E
N 0.1 0.2 0.5 0.2
W 0.2 0.1 0.3 0.4
S 0.5 0.1 0.1 0.3
E 0.3 0.4 0.2 0.1
Вероятност


71
iteration
N
W
S
E
0
4
8
12
offset
1
0
0
1
arrival
26
23
22
38
arrival_cnt
19
13
11
26
wait_cnt
1
2
0
3
queue
37
49
20
41
queue_accum
circle
4 -1 4 -1 8 8 12 0 -1 -1 12 12 8 0 0 0
0
15

ПСЕВДОКОД

Структури от данни, представящи кръговия трафик

circle[0..15] – текущо състояние на кръговия трафик

new_circle[0..15] – следващо състояние на кръговия трафик

Структури от данни, представящи 4-те входа

offset[0…3] – индекс на входа

arrival[0..3] – 1 ако е пристигнала кола

wait_count[0..3] – брой чакащи МПС

arrival_cnt[0..3] – общ брой пристигнали коли

queue[0..3] – брой чакащи коли

queue_accum[0..3] – общ брой чакащи коли

```
begin
  for i ← 0 to 15 do
    circle[i] ← -1
  endfor
  for i ← 0 to 3 do
    arrival_cnt[i], wait_cnt[i], queue[i],
   queue_accum[i] ← 0
  endfor
  for iteration ← 0 to requested iterations
    {нови МПС пристигат при входните точки}
    for i ← 0 to 3 do
      if u≤1/f[i] then {u-с равномерно разпределение}
```

```
    arrival[i] ←1
      arrival_cnt[i] ← arrival_cnt[i] +1
   else arrival[i] ← 0
   endif
endfor
{МПС в кръга се придвижват едновременно}
for i ← 0 to 15 do
   j ← (i+1)mod16
   if circle[i] =-1 or circle[i] =j then new_circle[j] ← -1
   else new_circle[j] ← circle[i]
   endif
endfor
Circle ← new_circle
```

```
{МПС се присъединяват към кръговия трафик}
for i ← 0 to 3 do
  if circle[offset[i]] = -1 then
    {има място 1 МПС да влезе}
    queue[i] ← queue[i] – 1
    circle[offset[i]] ← Choose_Exit(i)
  else if arrival[i] >0
    {новопристигнали МПС влизат в кръга}
    arrival[i] ← 0
    circle[offset[i]] ← Choose_Exit(i)
  endif
endif
```

```
      if arrival[i] > 0 then
        {новопристигналите МПС се нареждат на опашка}
        wait_cnt[i] ← wait_cnt[i] + 1
        queue[i] ← queue[i] +1
      endif
    endfor
    for i ← 0 to 15 do
      queue_accum[i] ← queue_accum[i] + queue[i]
    endfor
  endfor {iteration}
end
```

КРАЙ