РАЗВИТИЕ НА ДИНАМИЧНО ПАРАЛЕЛНИ ПРИЛОЖЕНИЯ

ЧАСТ 1

КОМБИНАТОРНО ТЪРСЕНЕ

КОМБИНАТОРНИ АЛГОРИТМИ

- Обработка на дискретни крайни математически структури
- ***Комбинаторно търсене*** – процес на намиране на едно или повече оптимални или субоптимални решения в дефинирано пространство на проблема
- Приложения: проектиране на МГИС при минимална площ, заемана от връзките, планиране на движенията на ръцете на робот при минимално изминато разстояние, доказване на теореми, игри

Проблеми при комбинаторното търсене

- Алгоритми, отговарящи на въпроса съществува ли решение на оптимизационни проблеми – “да” или “не”;
- Алгоритми, решаващи оптимизационни проблеми – намират решение, което намира минимум или максимум на стойността на функция на обекта;

Дърво на търсене

- При всички случаи коренът на дървото представя оригиналния проблем;
- Дълбочината и вида му зависят от решавания проблем;
- *Възел от тип "AND"* – проблем или подпроблем, който се решава тогава и само тогава, когато всичките му деца са решени;
- *Възел от тип "OR"* – проблем или подпроблем, който се решава когато някое от децата има решение;

Дърво на търсене

- *AND дърво* – решението на проблема се намира като се комбинират решенията на всички подпроблеми (алгоритми “разделяй-и-владей”);
- *OR дърво* – решението на проблема се намира когато поне един от проблемите е решен (търсене с обратен ход или търсене с клони и граници);
- *AND/ OR дърво* - дървета на игри.

РАЗДЕЛЯЙ-И-ВЛАДЕЙ

- Рекурсивна методология за решаване на проблеми, при която проблемът се разделя на подпроблеми, подпроблемите се решават и техните решения се комбинират за да се получи решението на изходния проблем;
- Използват AND дървета – изискват разглеждането на всеки възел;
- *По-лесно се имплементират на симетрични мултипроцесори отколкото на мултикомпютри* – централен стек като ефективен механизъм за баланс на товара, но и тясно място при голям брой на процесорите.

- *При мултикомпютри*:

1. Подпроблемите се разпределят между локалните памети на процесорите, изчислителният товар нараства динамично, след което намалява, един процесор съдържа оригиналния проблем и накрая съдържа решението;
2. Оригиналният проблем и полученото решение са разпределени между процесите;
3. Проблем – ефективен баланс на паралелния изчислителен товар.

ТЪРСЕНЕ С ОБРАТЕН ХОД

BACKTRACK SEARCH

ТЪРСЕНЕ С ОБРАТЕН ХОД

- Метод за решаване на комбинаторни оптимизационни проблеми, който се основава на обхождане по дълбочина (depth-first search) за разглеждане на алтернативите;
- Генерират се децата на основния проблем (корена) и се избира едно от тях за да продължи търсенето;
- Тази методология се повтаря рекурсивно за всеки избран възел;
- При достигане на възел, който не може да бъде разширен (dead end), или ако всички поддървета на детето са вече разгледани, управлението се връща на предходния възел (backtrack – обратен ход).

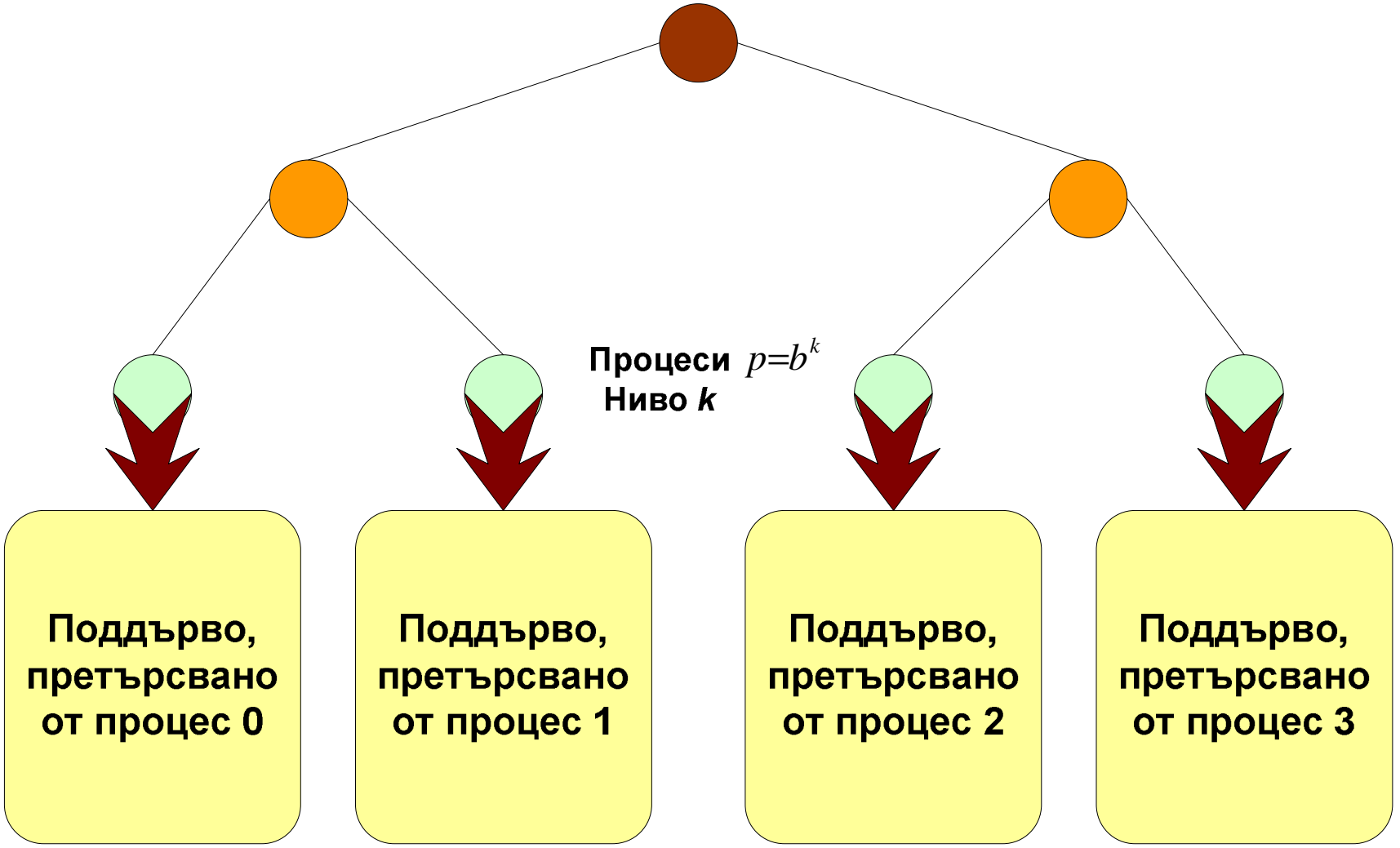
СЛОЖНОСТ

- Ако средният фактор на разклонение на дървото на пространството на търсене е *b*, търсенето в дърво с дълбочина *k* изисква разглеждането в най-лошия случай на *~Θ(b^k)* брой възли

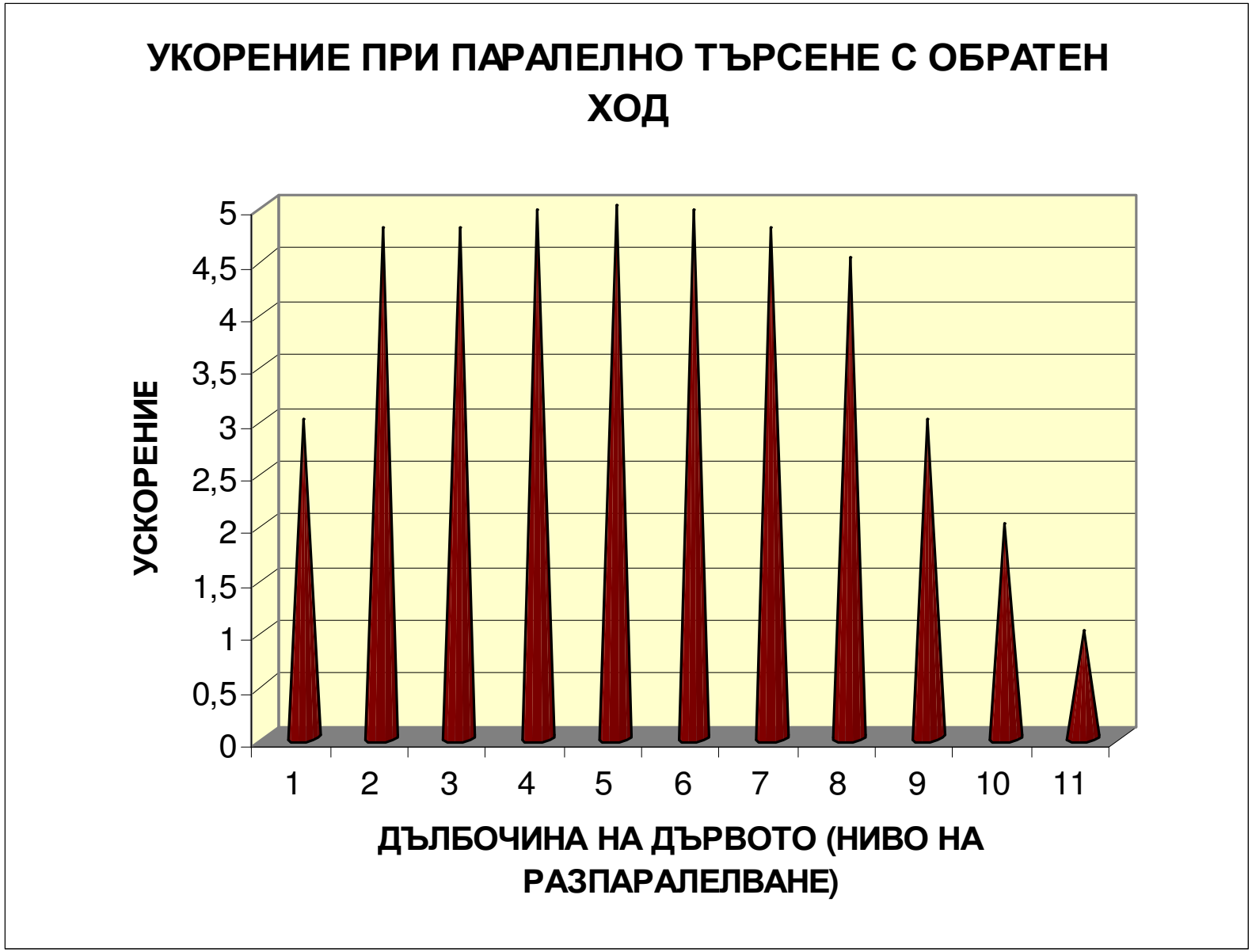
$$1+b+b^2+\ldots+b^k=\frac{b^{k+1}-b}{b-1}+1=\Theta\left(b^k\right)$$

- Търсенето с обратно обхождане в дървото на пространството на търсене изисква експоненциално време в най-лошия случай
- Необходимата памет нараства линейно с дълбочината на търсенето *~ Θ(k)*

Паралелно търсене с обратен ход



Стратегия – разделяме търсенето в поддърветата между процесите;

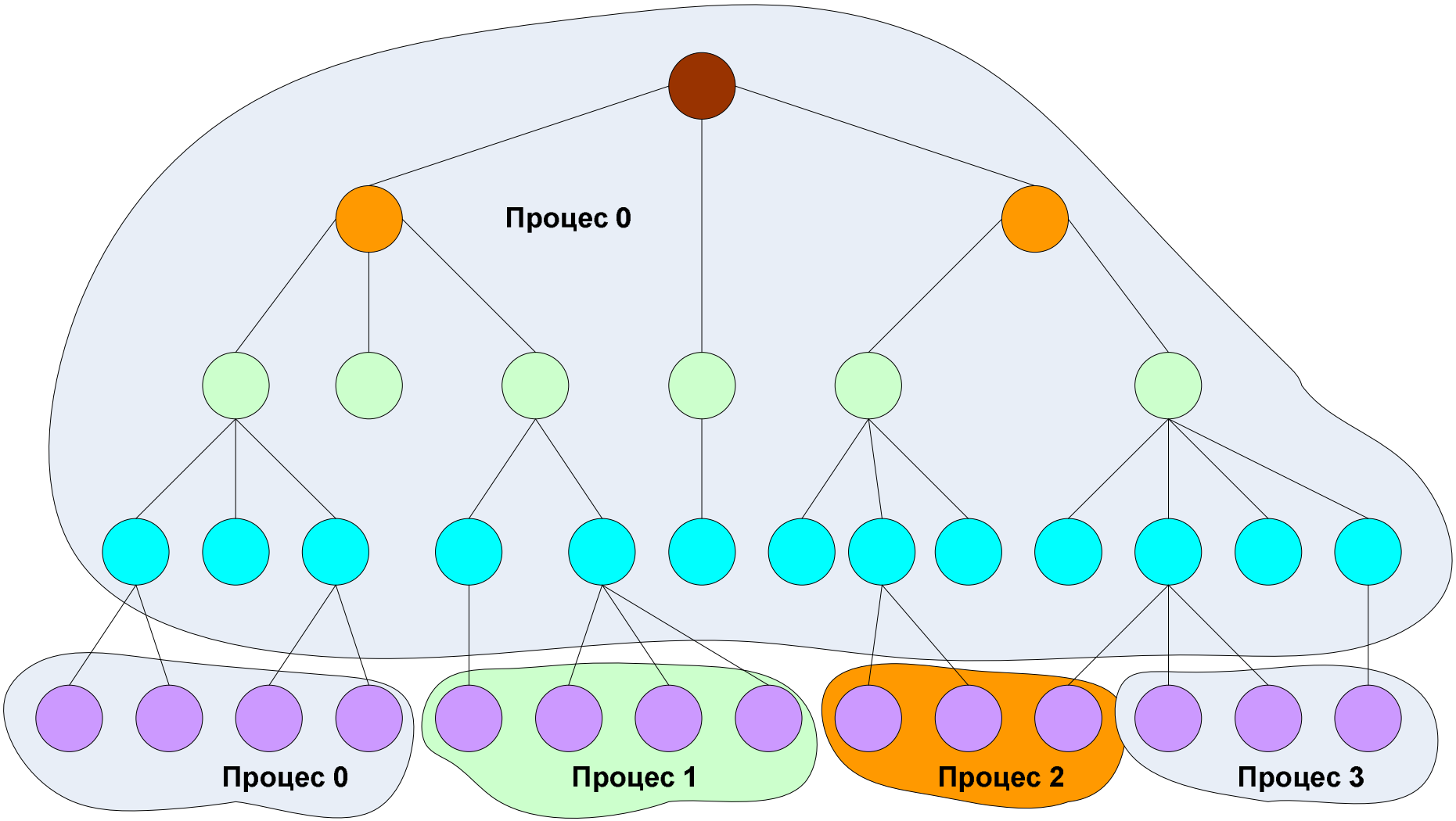


Максимално ускорение при търсене по обратна следа в дърво на търсене с фактор на разклонение 3 и дълбочина 10

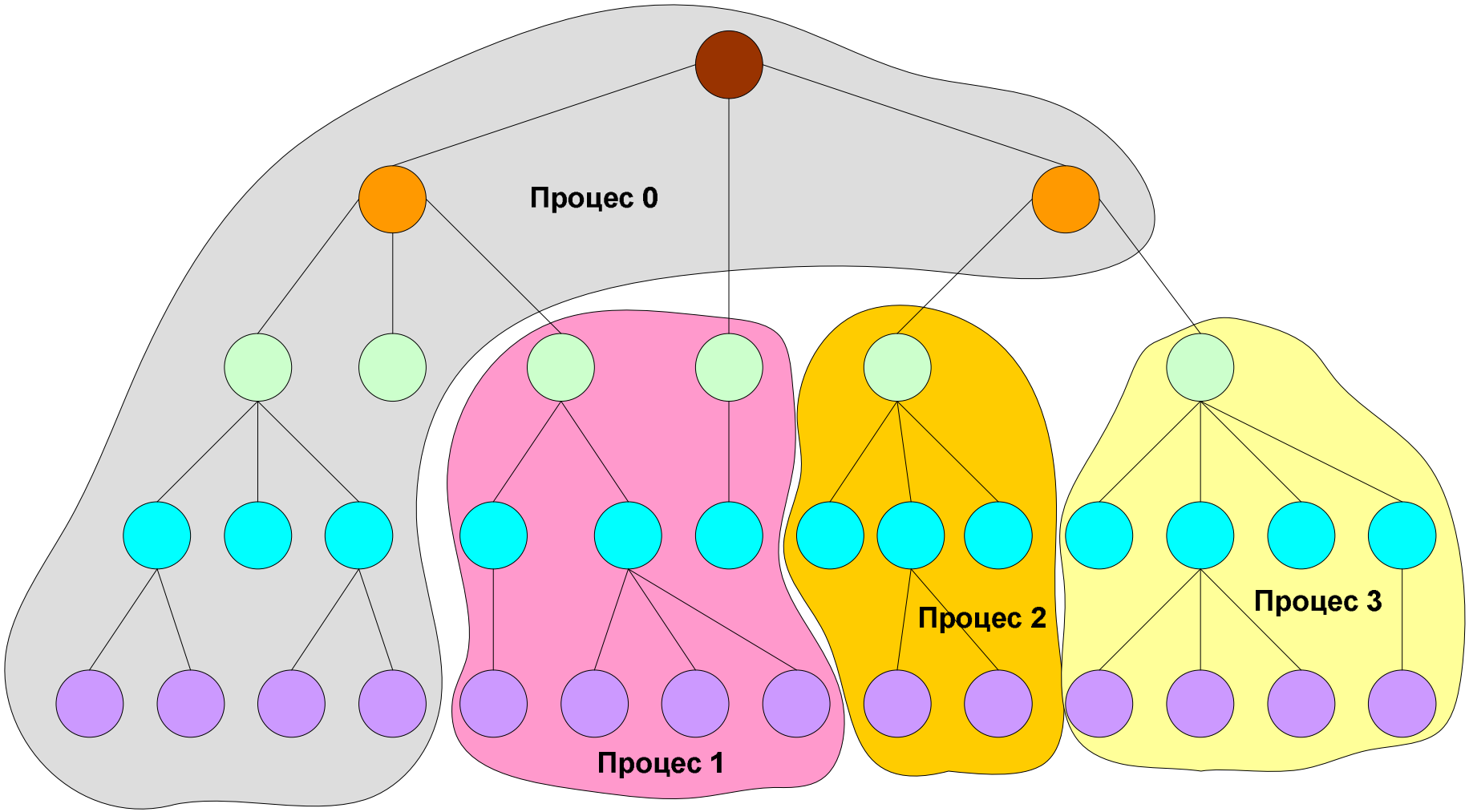
ПРОБЛЕМ

- Обикновено дърветата на пространството на търсене са силно небалансирани
- Желателно е да се обхождат повече на брой поддървета едновременно за да се балансира натоварването на процесорите

Разпределяне на товара



Разпределяне на товара



Разпределено терминиране на паралелната програма

- Всеки процес терминира след като завърши търсенето в разпределената му част от дървото до указаната му дълбочина;
- Този тип алгоритми намират всички решения и след това определят оптималното решение;
- В някои случаи е необходимо само едно решение;

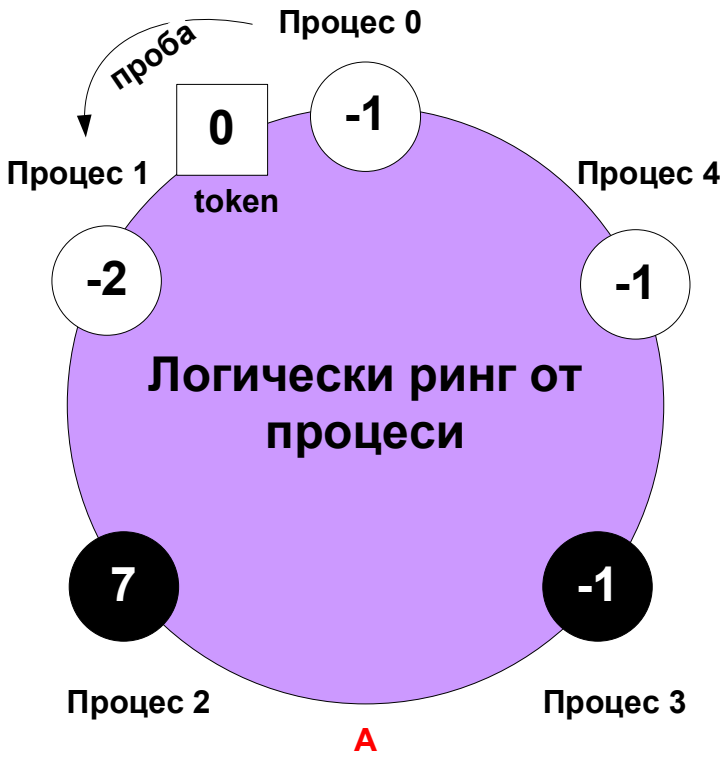
- Процес, открил решение, изпраща съобщение до всички останали процеси;
- Всички процеси периодически проверяват за получени съобщения при достигане на определено ниво *cutoff_depth* (MPI_Iprobe – без блокиране);
- Преди терминиране всички процеси трябва да са преустановили търсенето (неактивни) и да няма съобщения в процес на предаване – MPI_Finalize.

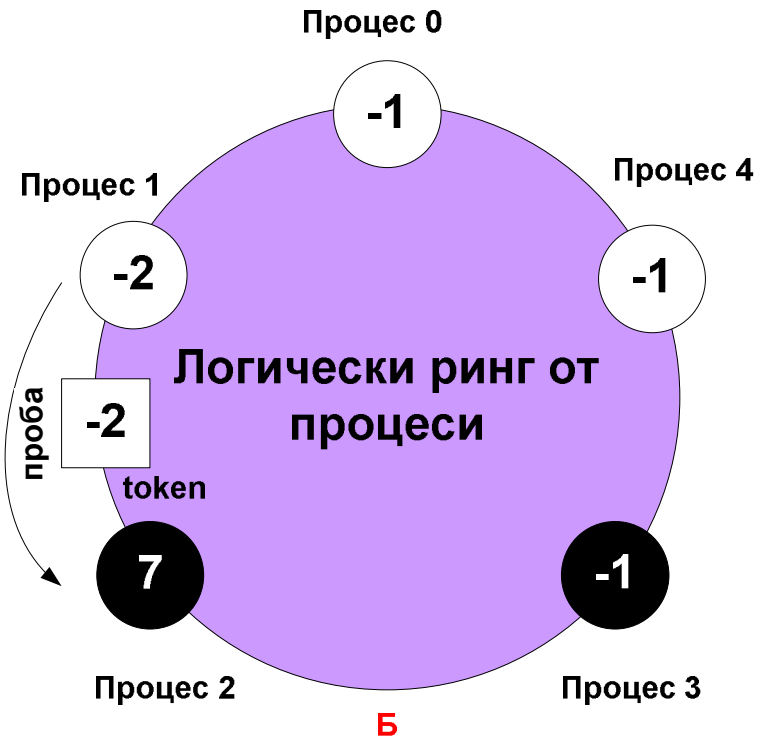
Алгоритъм на Дийкстра за разпределено терминиране

- Всеки процес има цвят и брояч на съобщенията;
- При започване на обработката всеки процес е бял и броячът на съобщенията е нулиран;
- Процесът се оцветява в черно, когато изпраща или получава съобщение;
- При изпращане на съобщение – процесът инкрементира брояча на съобщенията;
- При получаване на съобщение – процесът декрементира брояча на съобщенията;

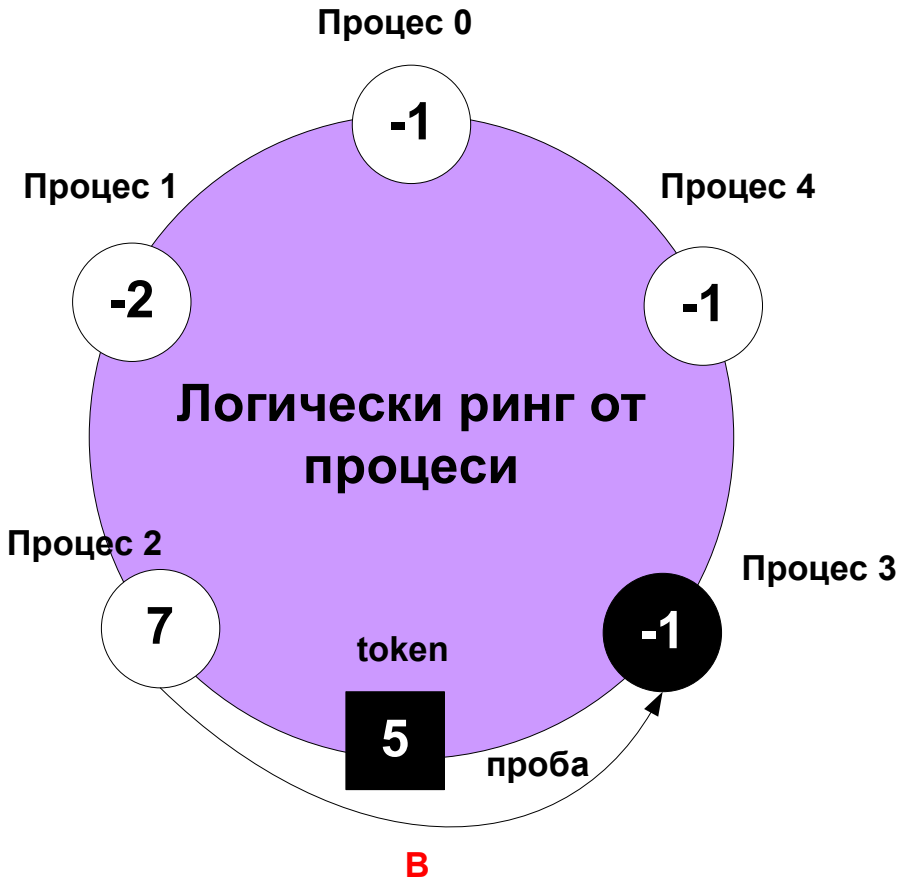
- ***Ако всички процеси са маркирани в бяло и сумата от броячите на съобщенията на всички процеси е 0 → процесите могат да бъдат терминирани;***
- ***Рамково съобщение (token)*** – има цвят и брояч;
- Когато процес 0 инициира пробата, рамковото съобщение е бяло и броячът му е нулиран;
- Когато процес получава рамково съобщение, той прибавя стойността на брояча си към тази на брояча на рамковото съобщение; ако процесът е активен, държи рамковото съобщение, докато стане неактивен;
- Процесът се оцветява в черно, ако получава съобщение или обработва подпроблем с по-малка стойност на g от тази на текущото най-добро решение;
- Ако процесът е черен, той оцветява рамковото съобщение в черно, а самият той става бял и предава рамковото съобщение към следващия процес в рамките на логическия ринг; ако процесът е бял, той не променя цвета на рамковото съобщение.

- Когато рамковото съобщение се върне към процес 0, ако рамковото съобщение е бяло и сумата от съдържанието на брояча му и брояча на съобщенията на процес 0 е 0, тогава процес 0 изпраща съобщение до всички процеси за терминиране (termination message) и извиква MPI_Finalize;
- В противен случай процес трябва да тества логическия ринг отново;
- Междинните процеси само променят цвета на рамковото съобщение, променят стойността на брояча му и го предават нататък по логическия ринг;
- Приемането или предаването на рамковото съобщение не променя съдържанието на броячите на съобщенията на процесите.
- След получаването на съобщение за терминиране, всички останали процеси извикват MPI_Finalize.

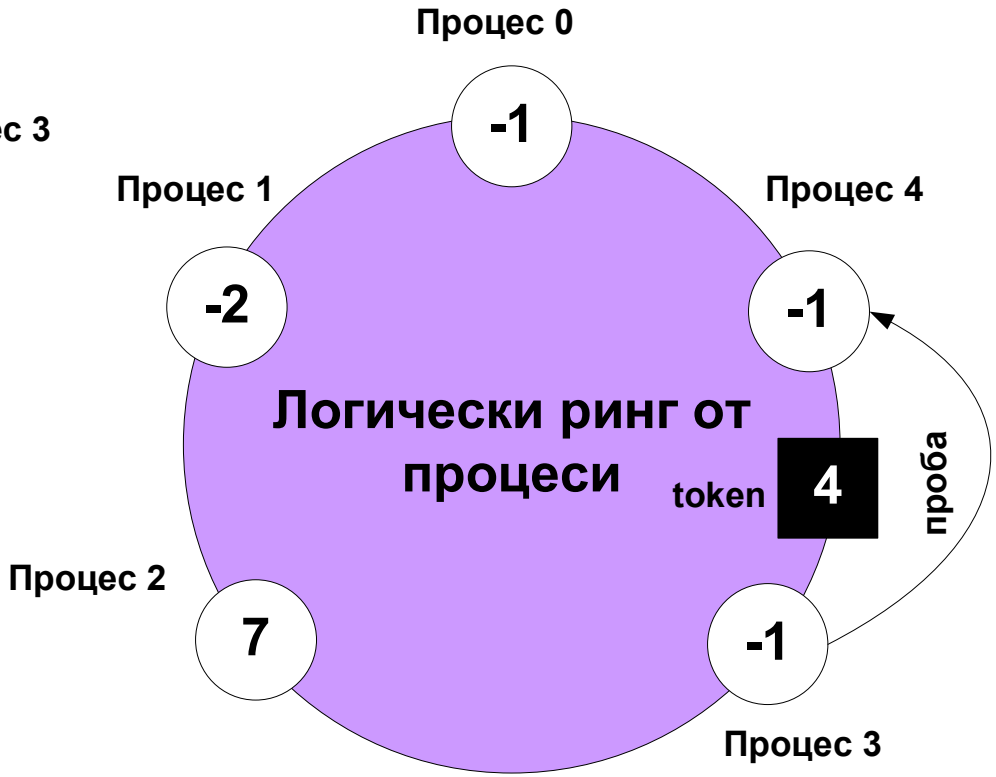




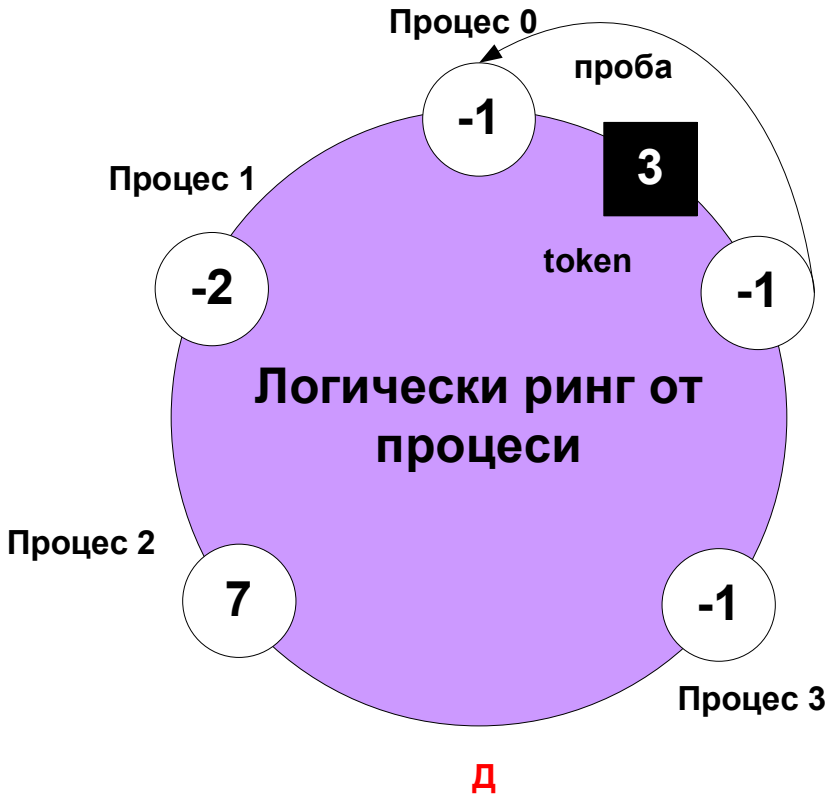
цвят	брояч	Цена на текущото най-добро решение	Ходове за решението


Процес 0
-1
Процес 1
Процес 4
-2
-1
Логически ринг от процеси
Процес 2
Процес 3
7
token
-1
5
проба

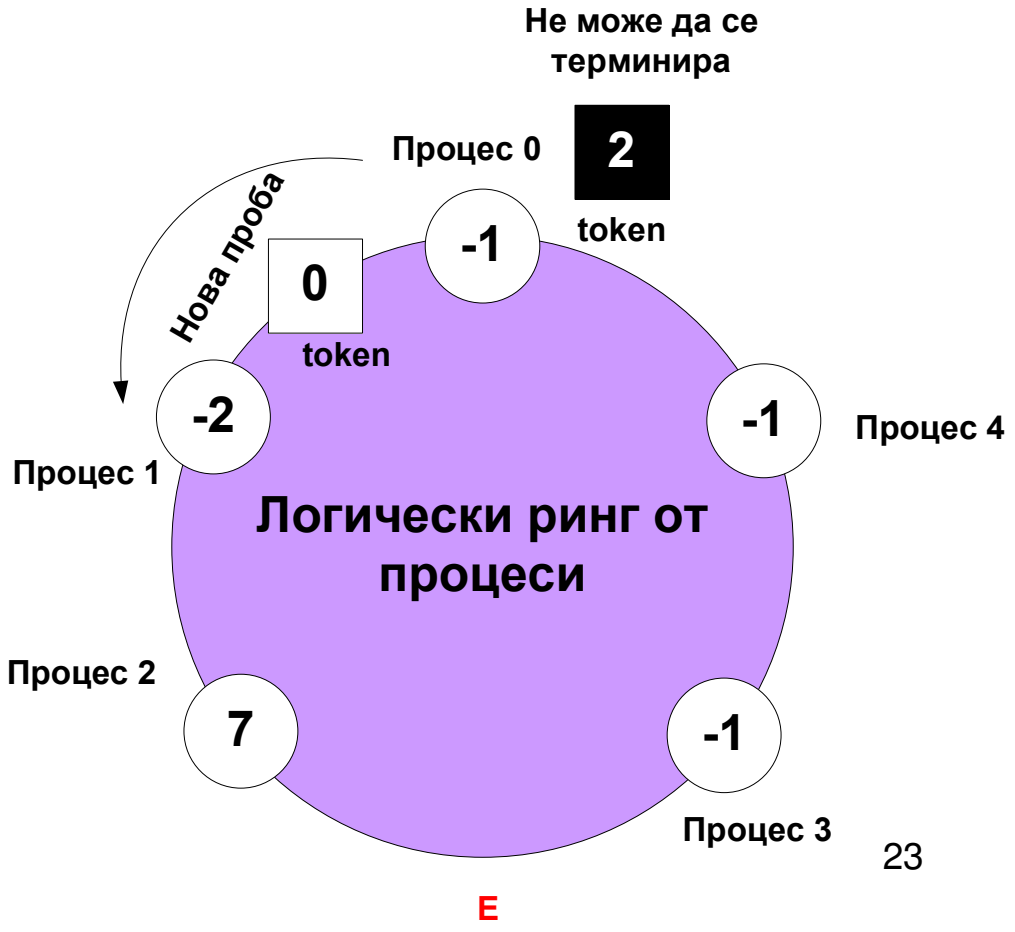
В


Процес 0
-1
Процес 1
Процес 4
-2
-1
Логически ринг от процеси
token
4
проба
Процес 2
7
-1
Процес 3

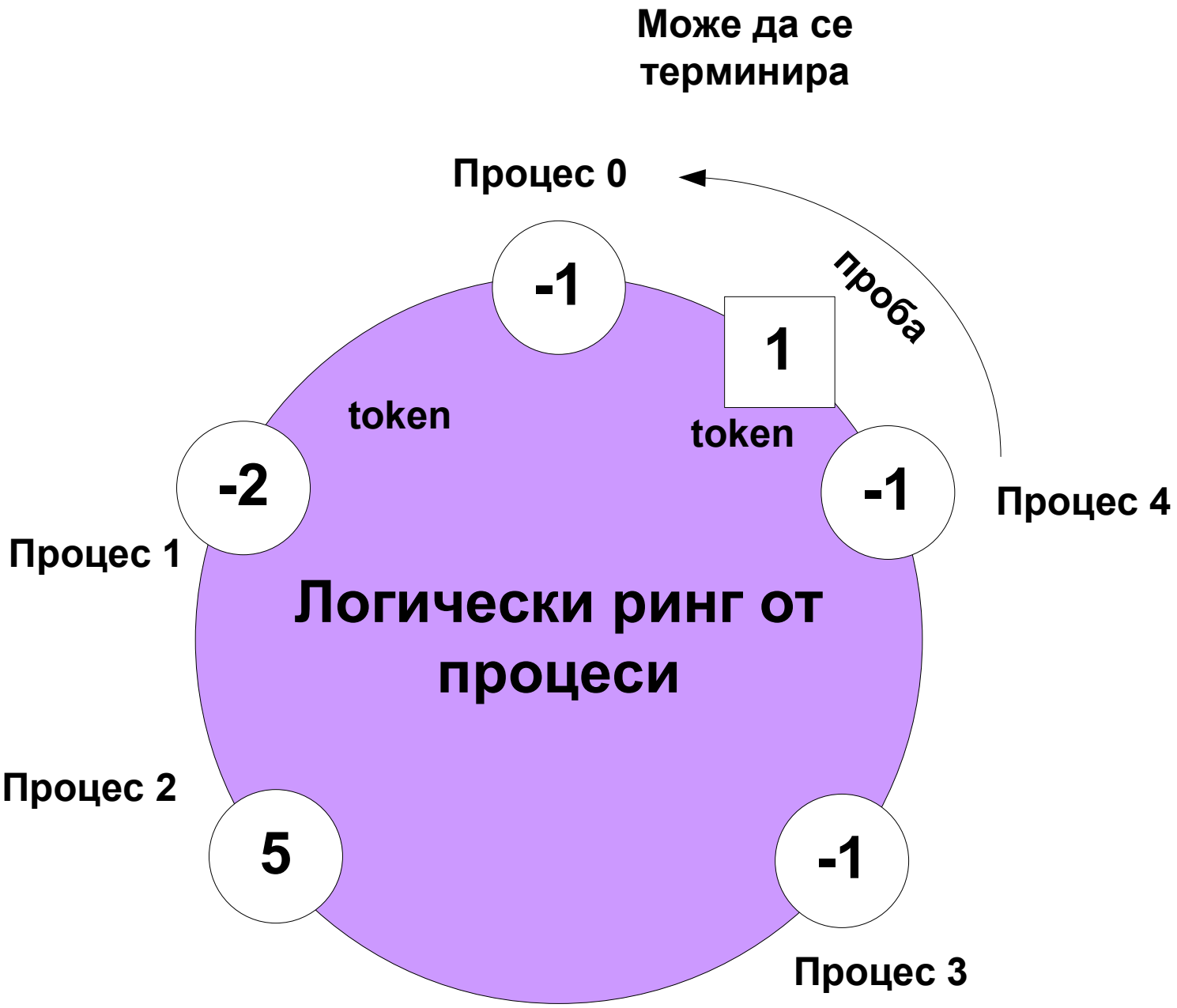
Г


Процес 0
проба
-1
3
Процес 1
token
-2
-1
Процес 4
Логически ринг от процеси
Процес 2
7
-1
Процес 3

Д


Не може да се терминира
Процес 0
2
token
Нова проба
-1
0
token
-2
-1
Процес 4
Процес 1
Логически ринг от процеси
Процес 2
7
-1
Процес 3

Е


Може да се терминира
Процес 0
проба
-1
1
token
token
-2
-1
Процес 1
Процес 4
Логически ринг от процеси
Процес 2
5
-1
Процес 3

К Р А Й