Informed Search

CS5804 Introduction to Artificial Intelligence
Virginia Tech

function TREE-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

loop do:

if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

expand the chosen node, adding the resulting nodes to frontier

function TREE-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

loop do:

if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

expand the chosen node, adding the resulting nodes to frontier

FIFO: Breadth-first
LIFO: Depth-first

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier

only if not in the frontier or explored set

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

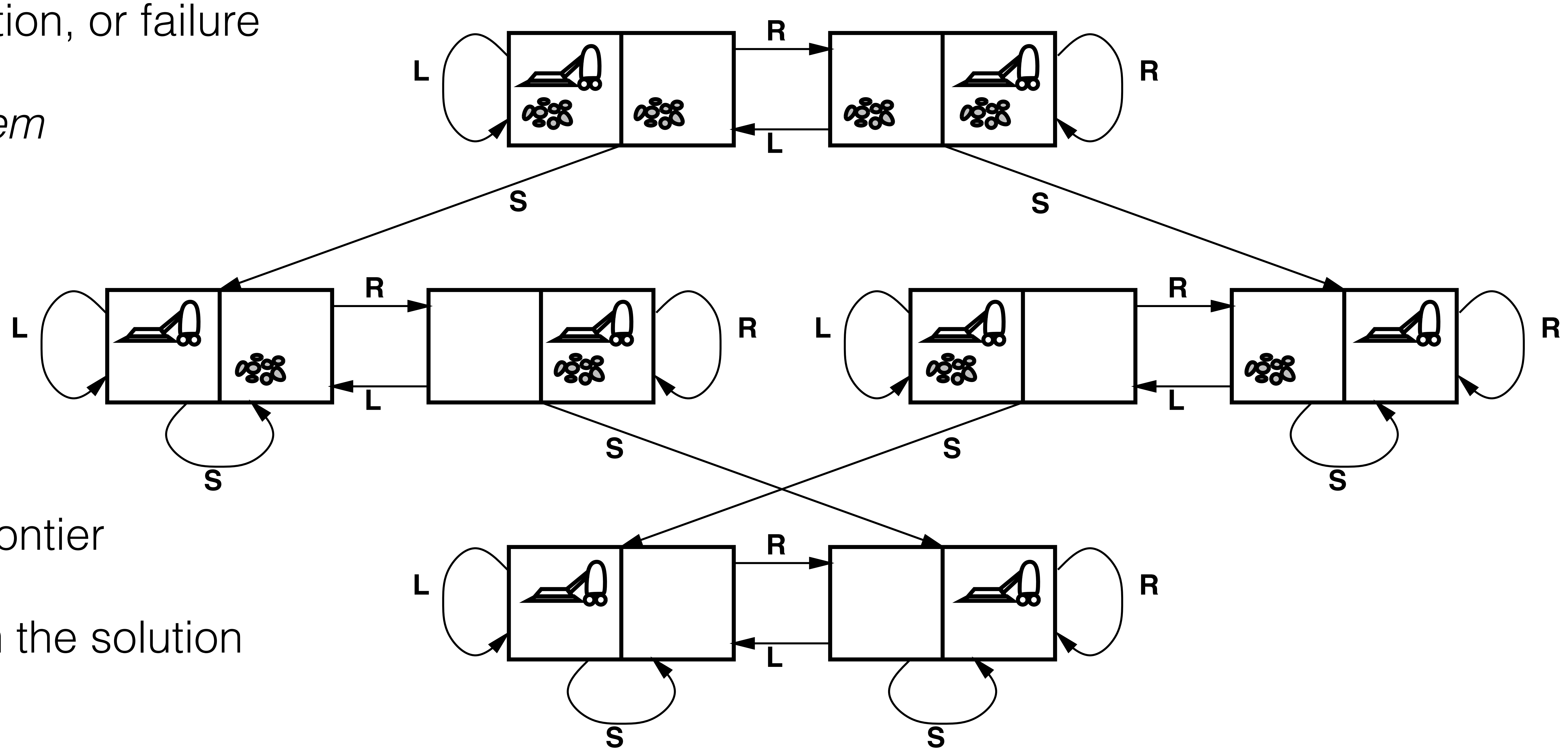
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier **only if not in the frontier or explored set**



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

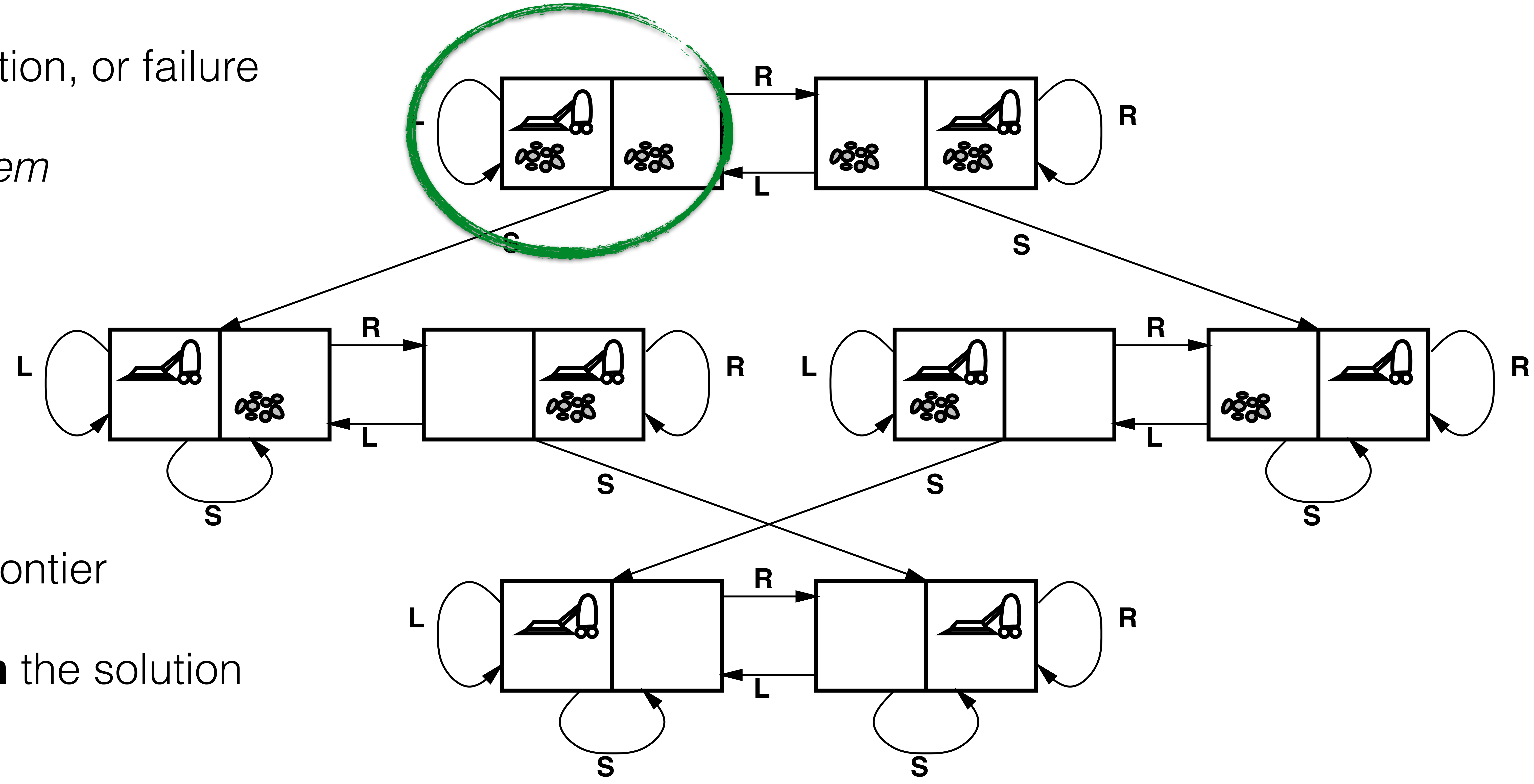
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier **only if not in the frontier or explored set**



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

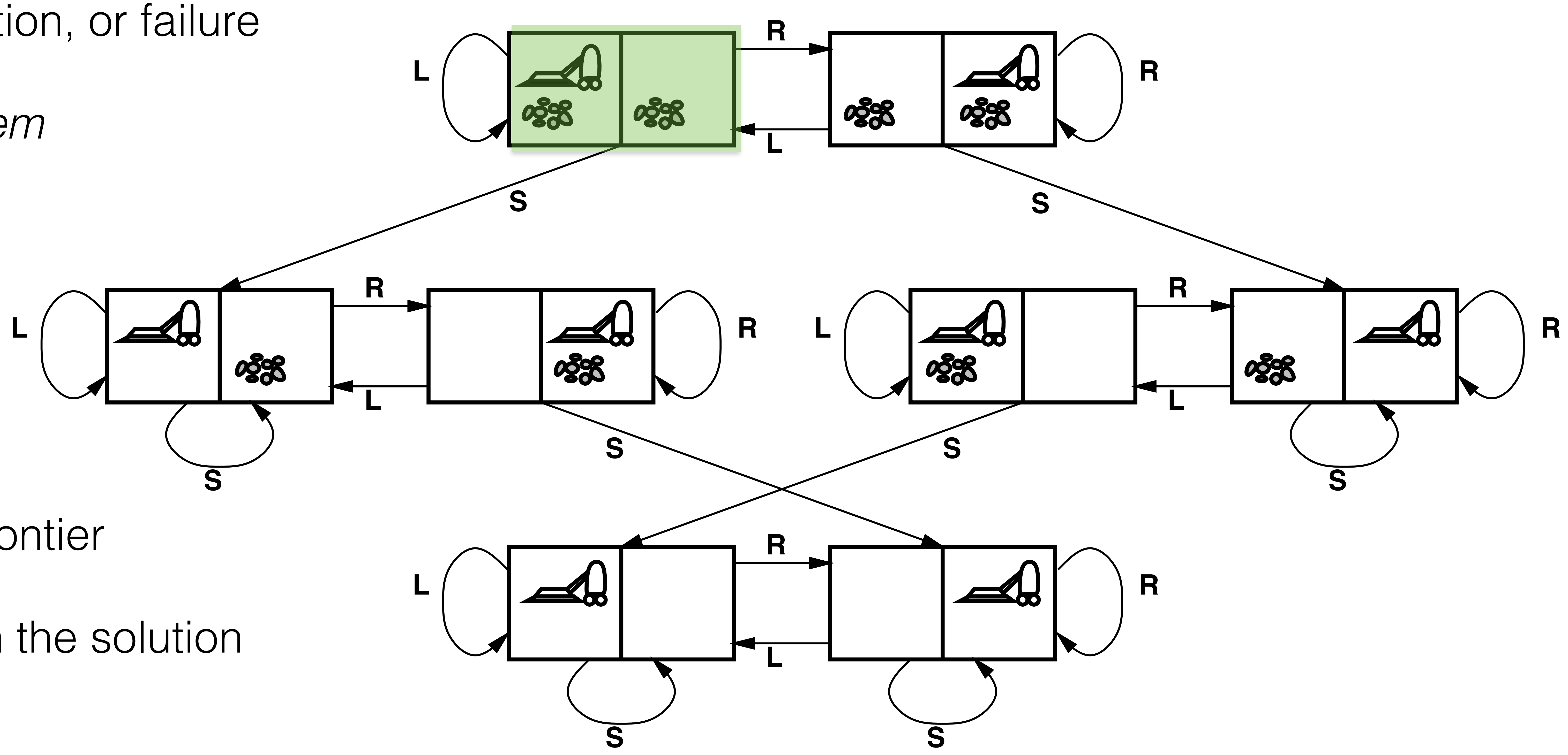
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier **only if not in the frontier or explored set**



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

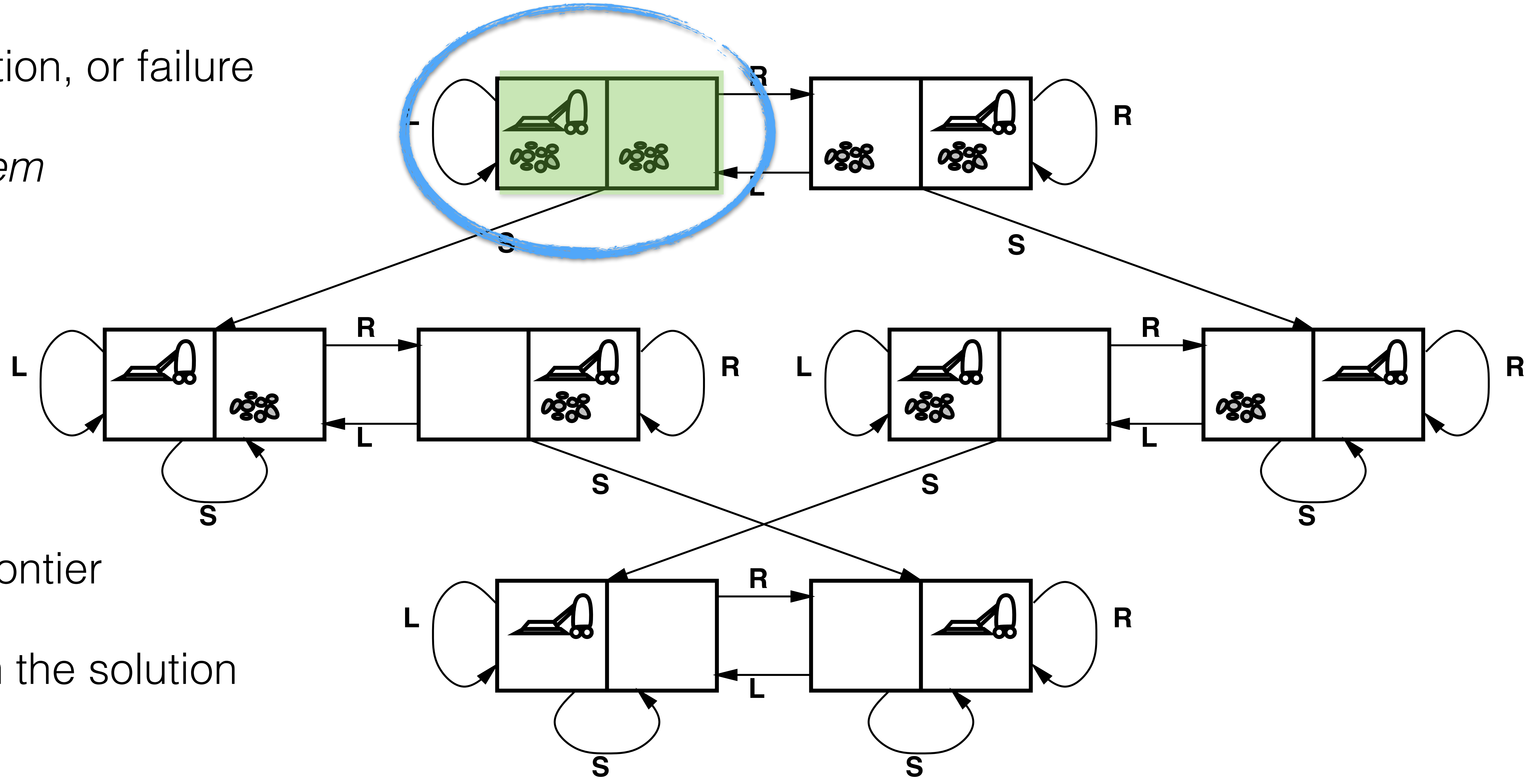
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier **only if not in the frontier or explored set**



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

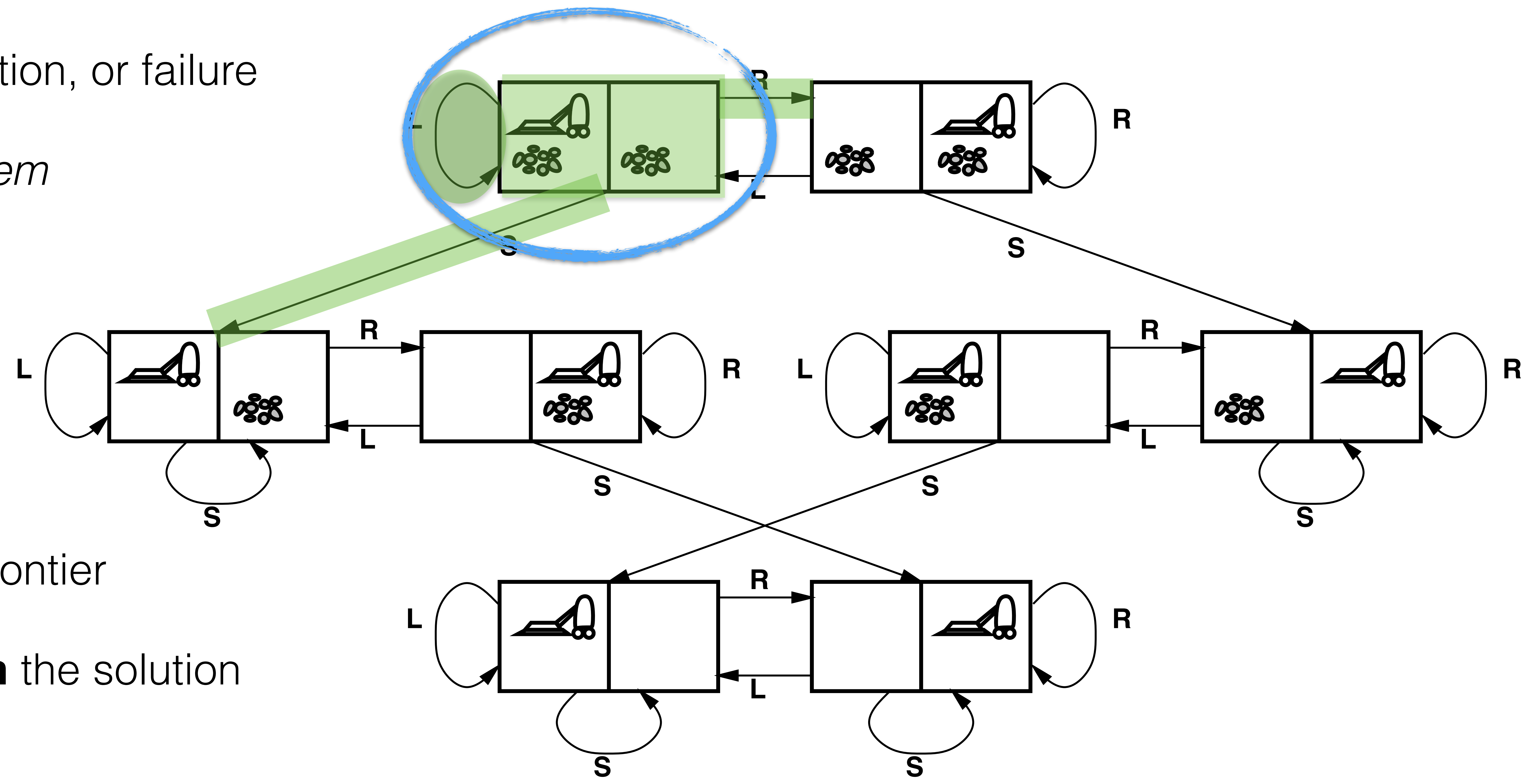
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier **only if not in the frontier or explored set**



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

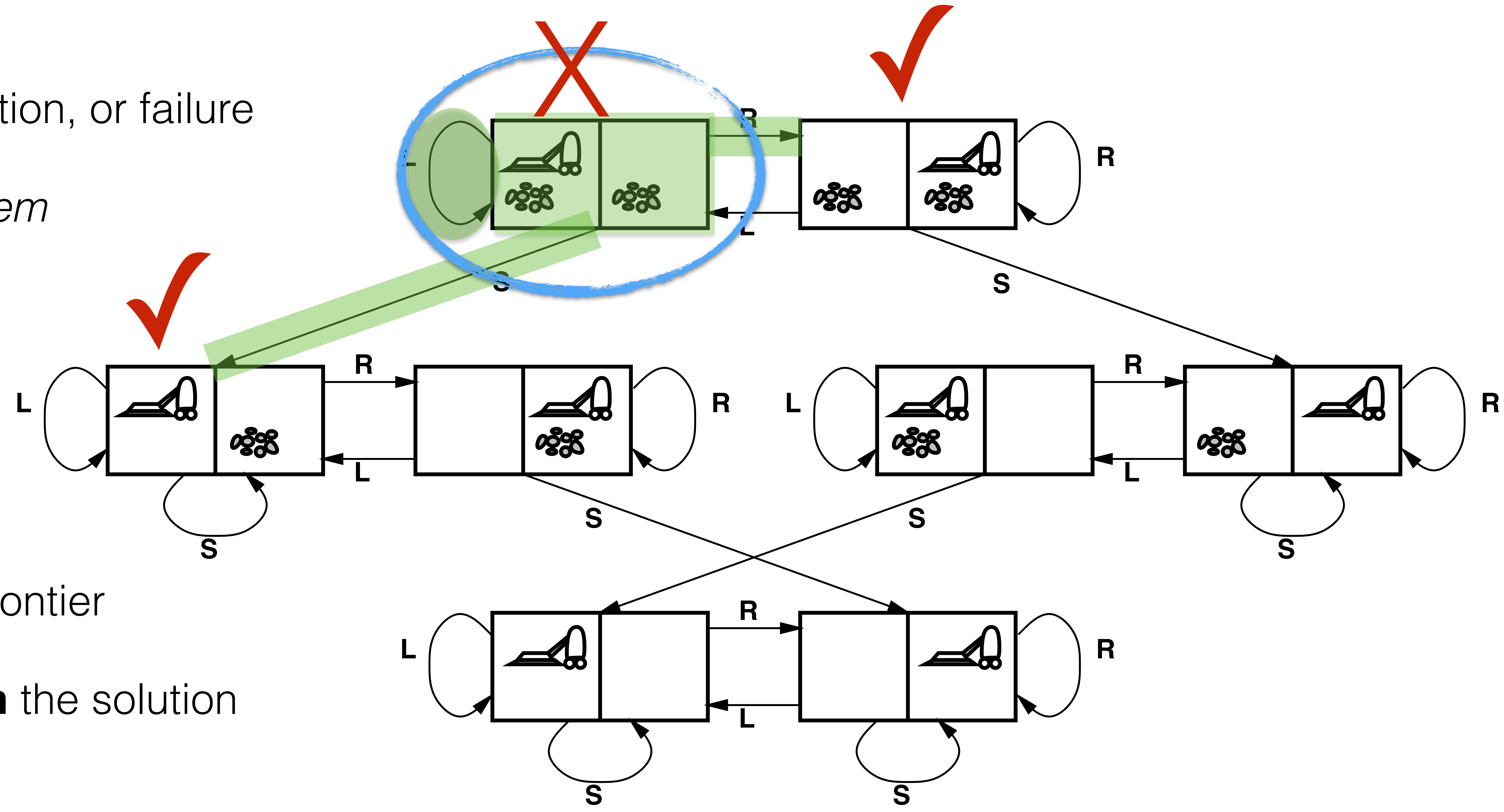
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier **only if not in the frontier or explored set**



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

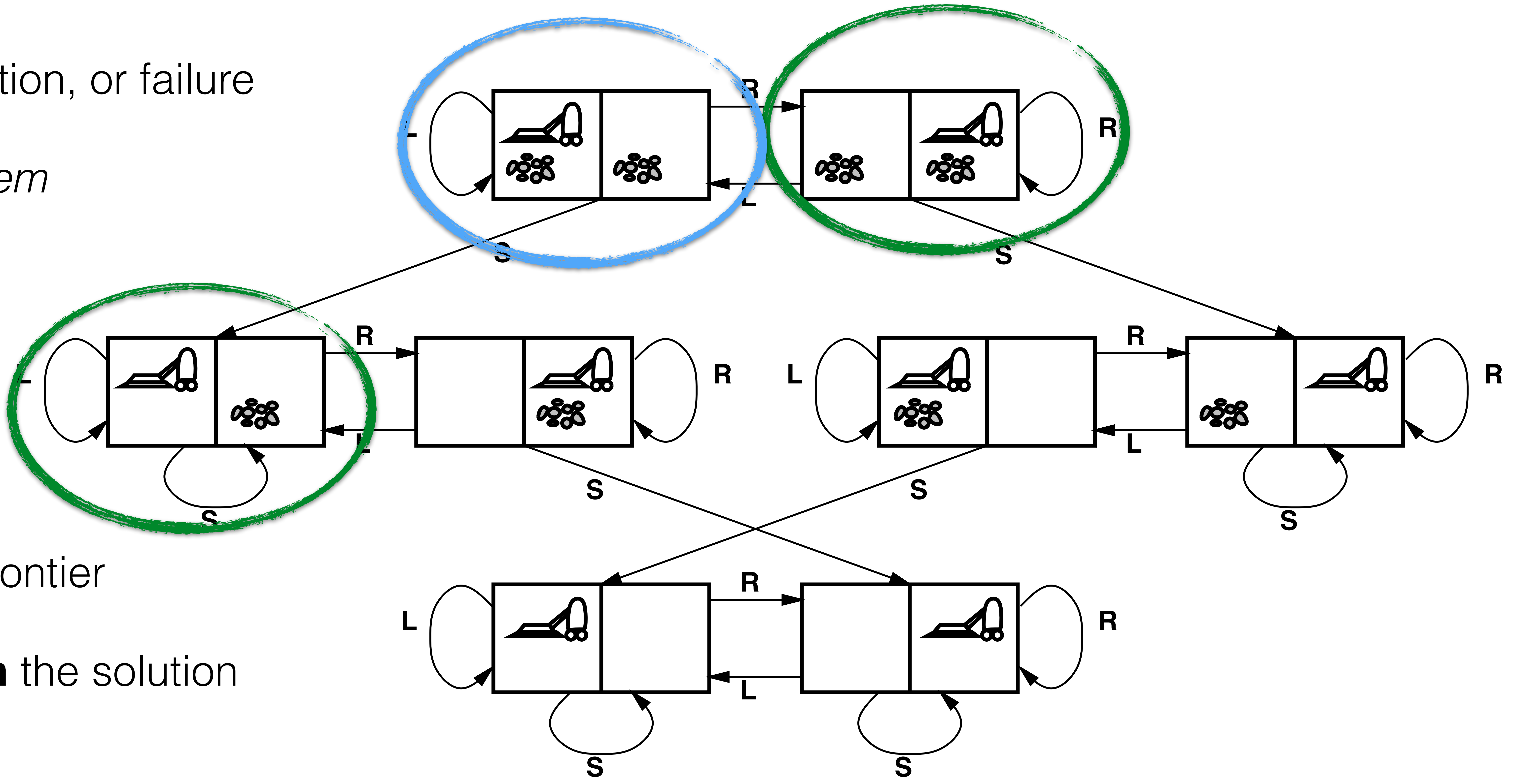
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier **only if not in the frontier or explored set**



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

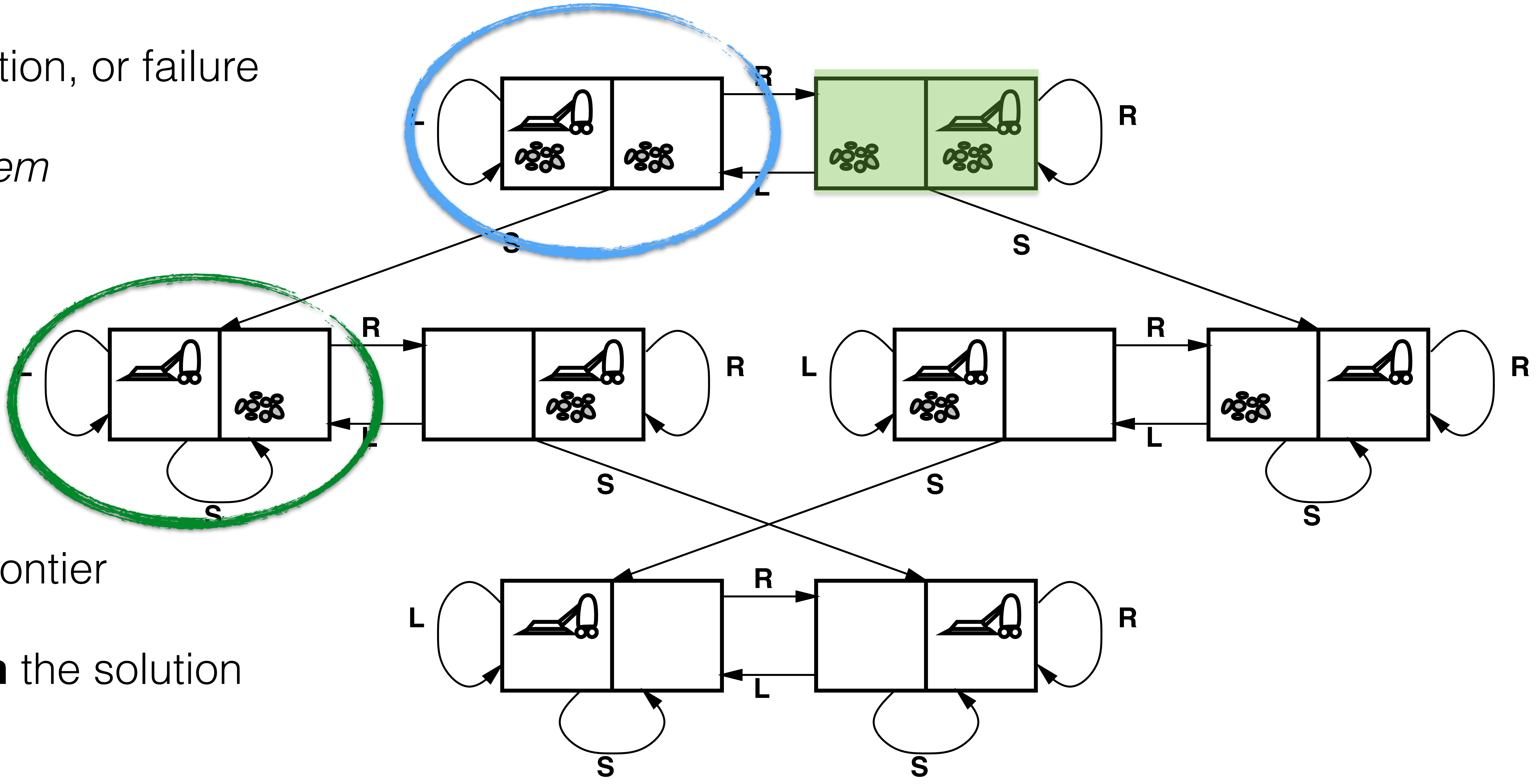
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier **only if not in the frontier or explored set**



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

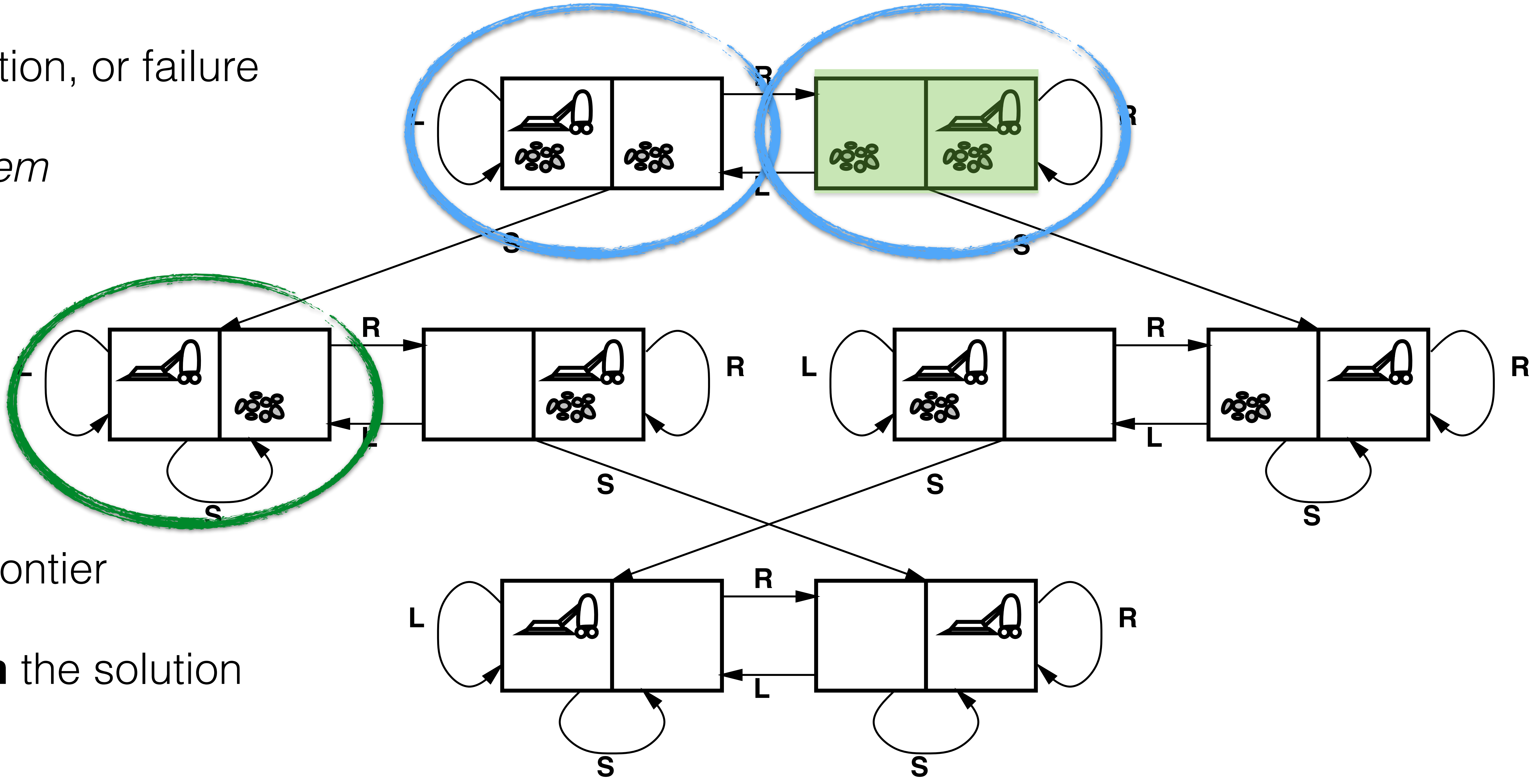
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier **only if not in the frontier or explored set**



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

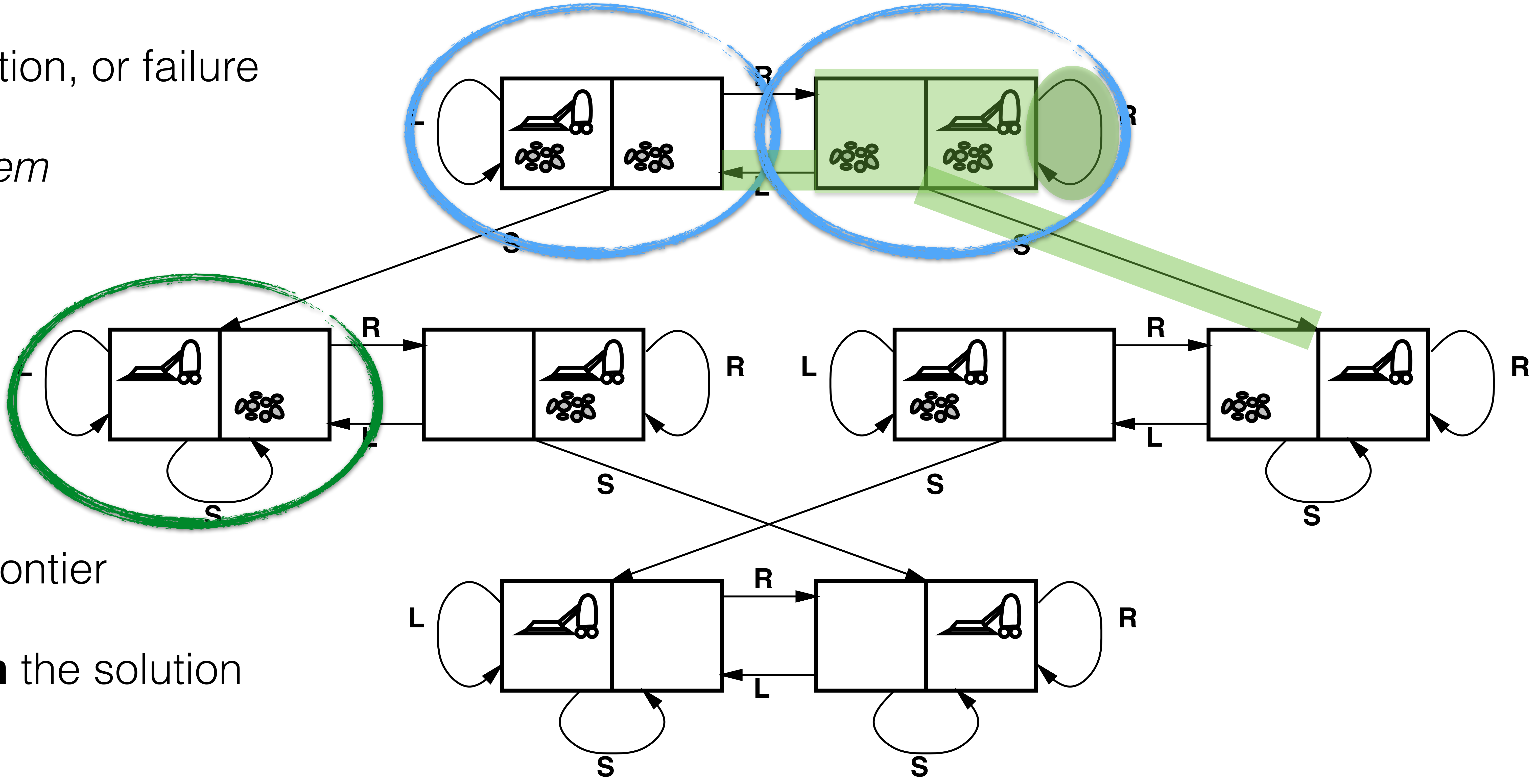
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier **only if not in the frontier or explored set**



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

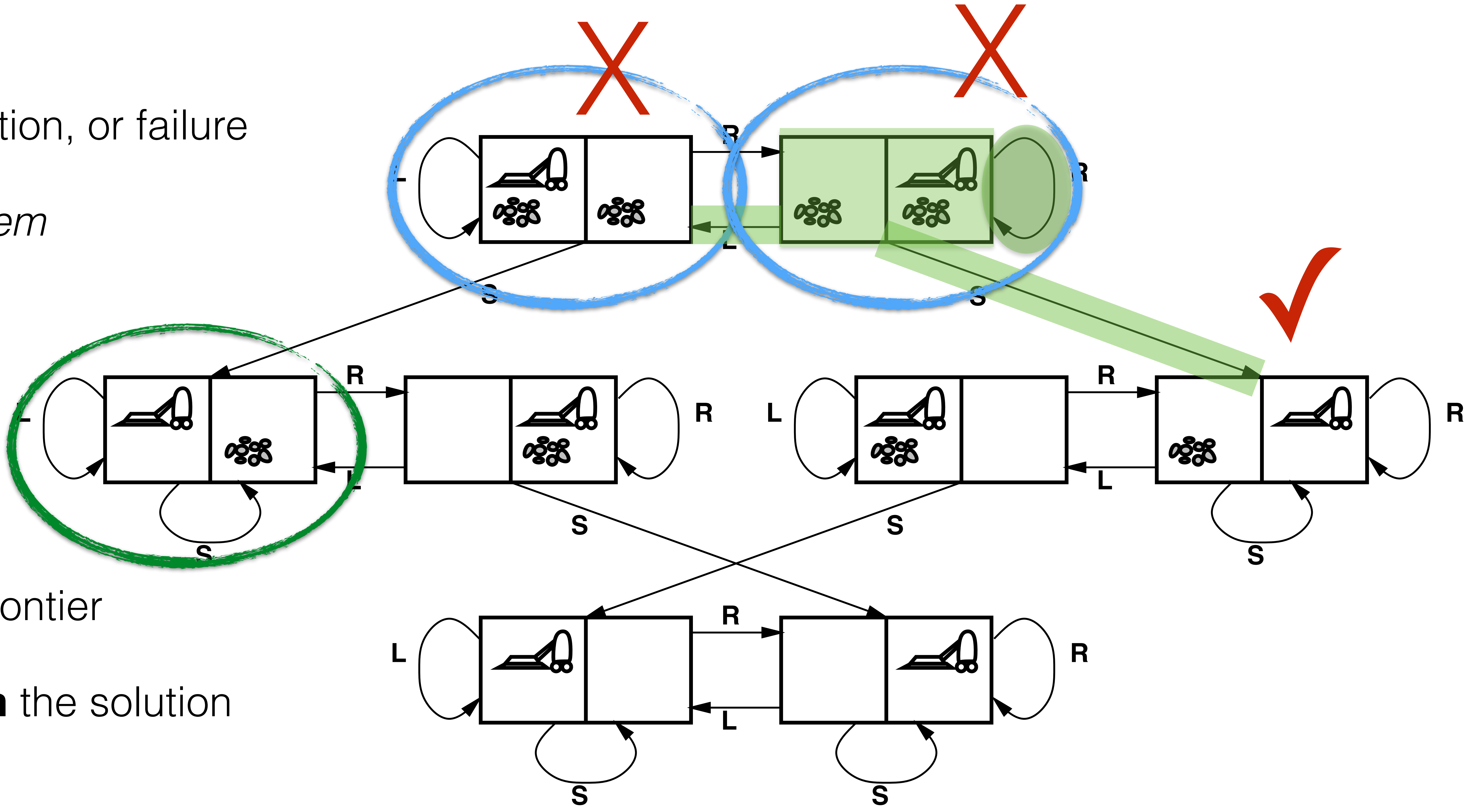
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier **only if not in the frontier or explored set**



function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do:

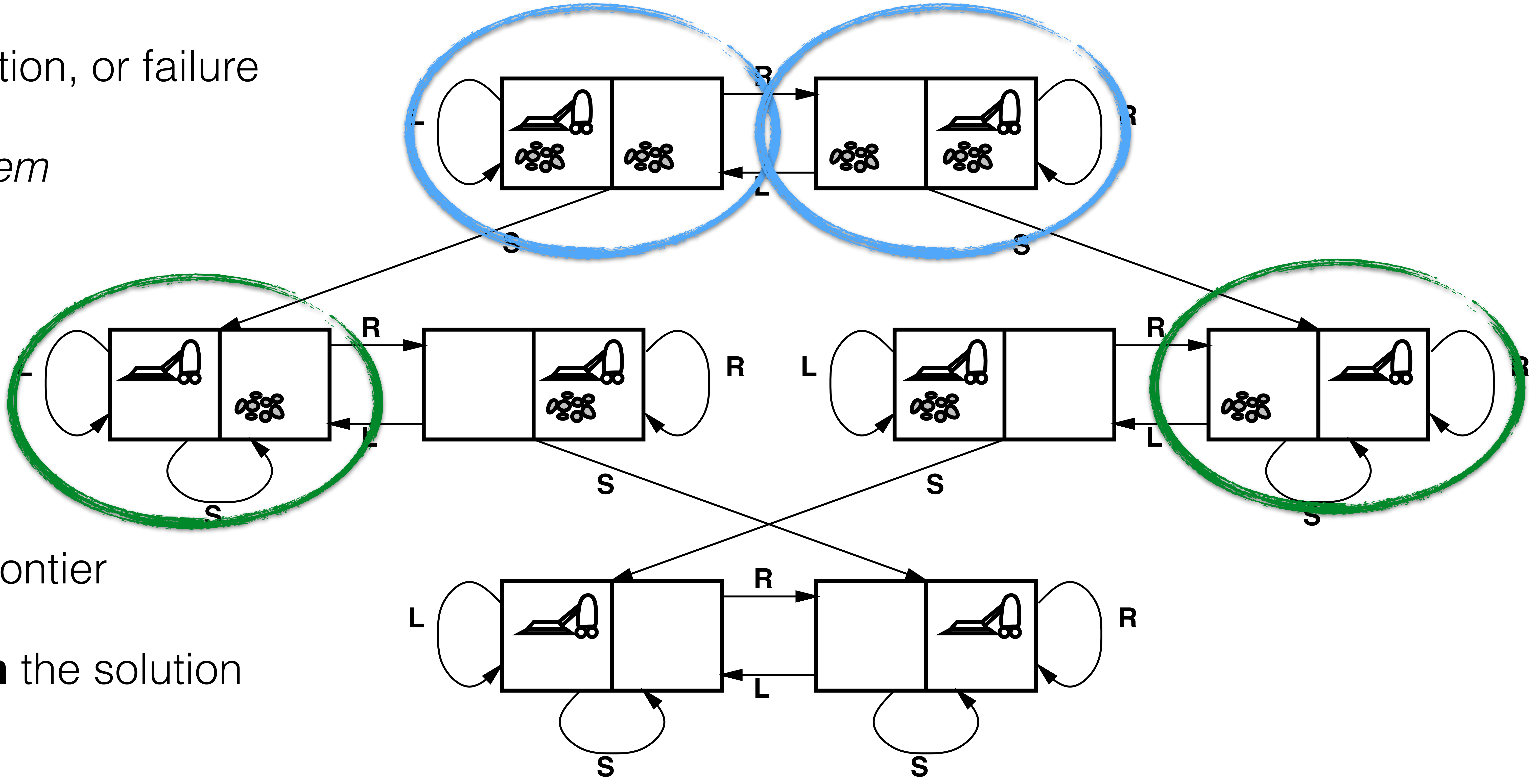
if the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to frontier **only if not in the frontier or explored set**



Search Algorithm Properties

- Completeness
- Optimality
- Time complexity
- Space complexity

Search Algorithm Properties

- Completeness: BFS, uniform cost, finite DFS, iterative deepening…
- Optimality: uniform cost
- Time complexity: BFS O(b^d), DFS terrible, iterative deep. O(b^d)
- Space complexity: BFS O(b^d), DFS O(bm), iterative deep. O(bm)

Search Algorithm Properties

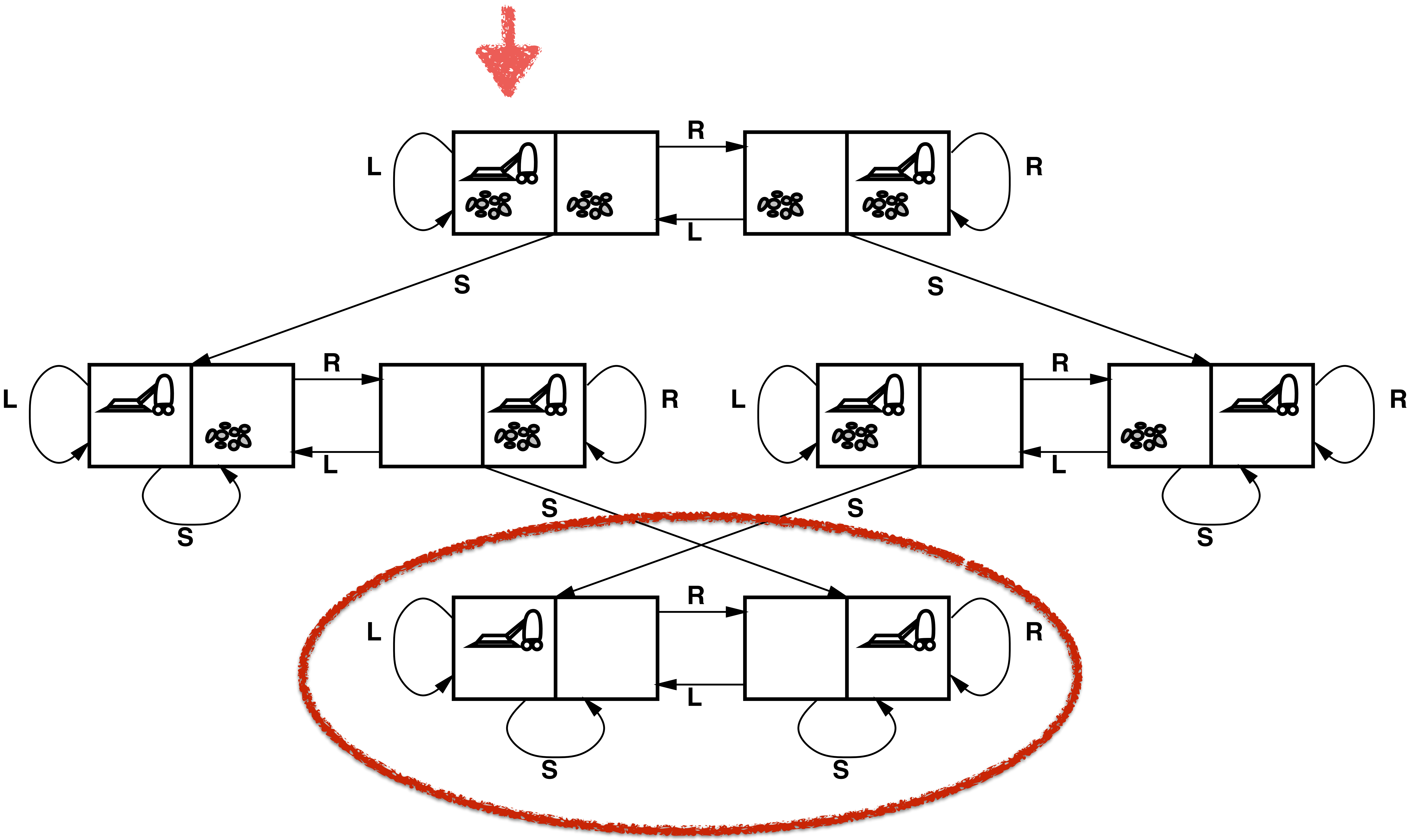
- Completeness: BFS, uniform cost, finite DFS, iterative deepening…
- Optimality: uniform cost
- Time complexity: BFS O(b^d), DFS terrible, iterative deep. O(b^d)
- Space complexity: BFS O(b^d), DFS O(bm), iterative deep. O(bm)

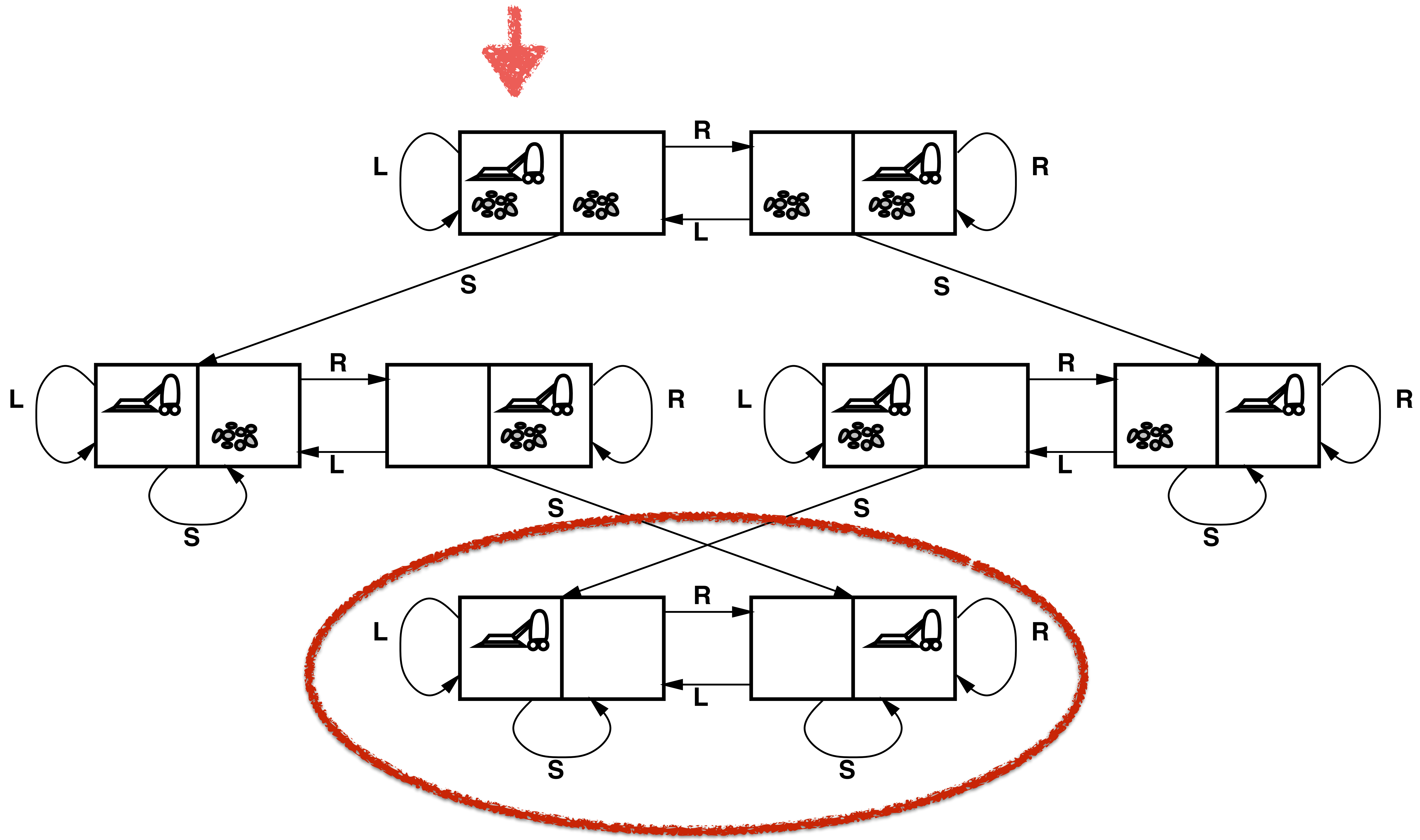
tree vs graph search?

Search Algorithms

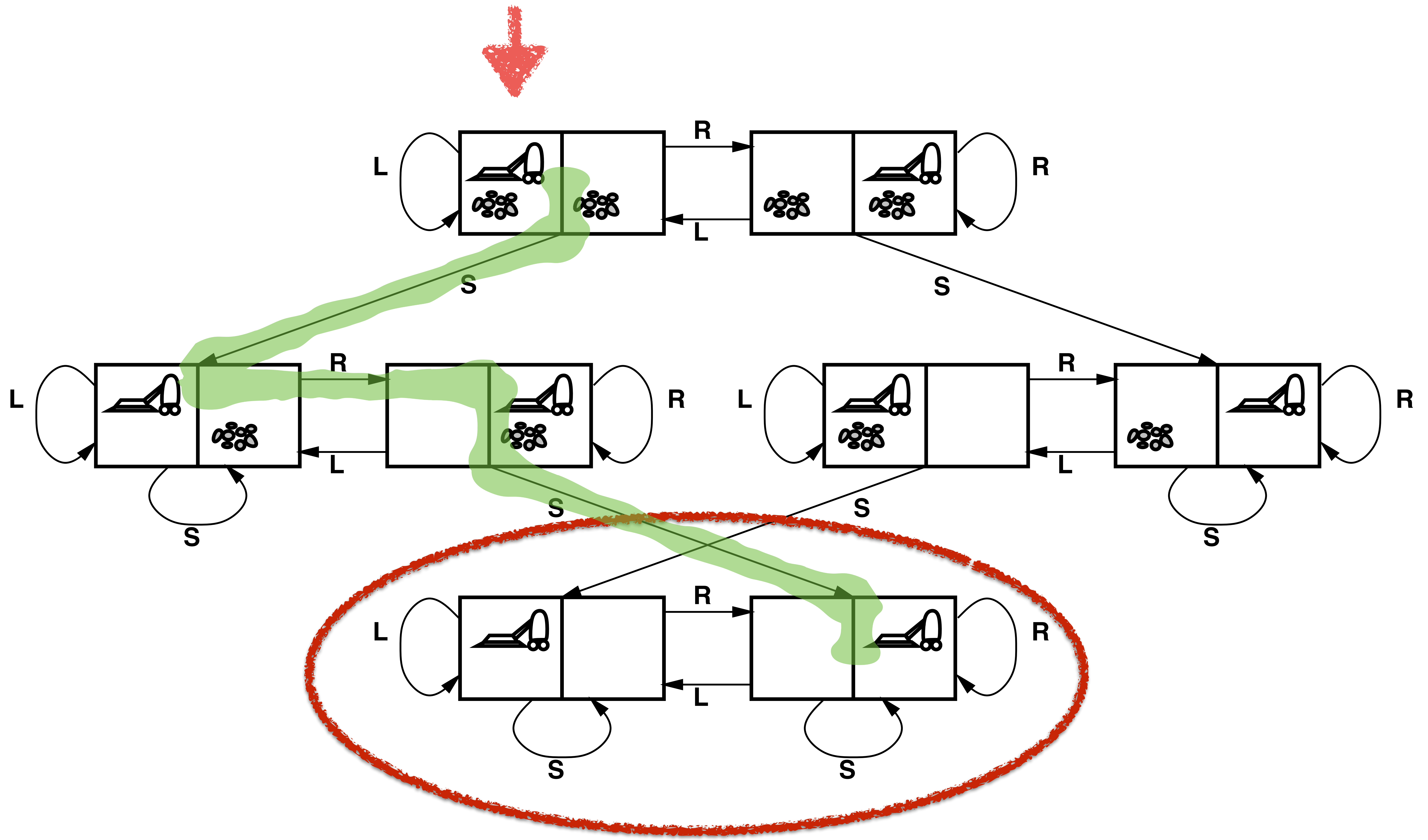
- Breadth-first family
    - Breadth-first search
    - Uniform-cost search

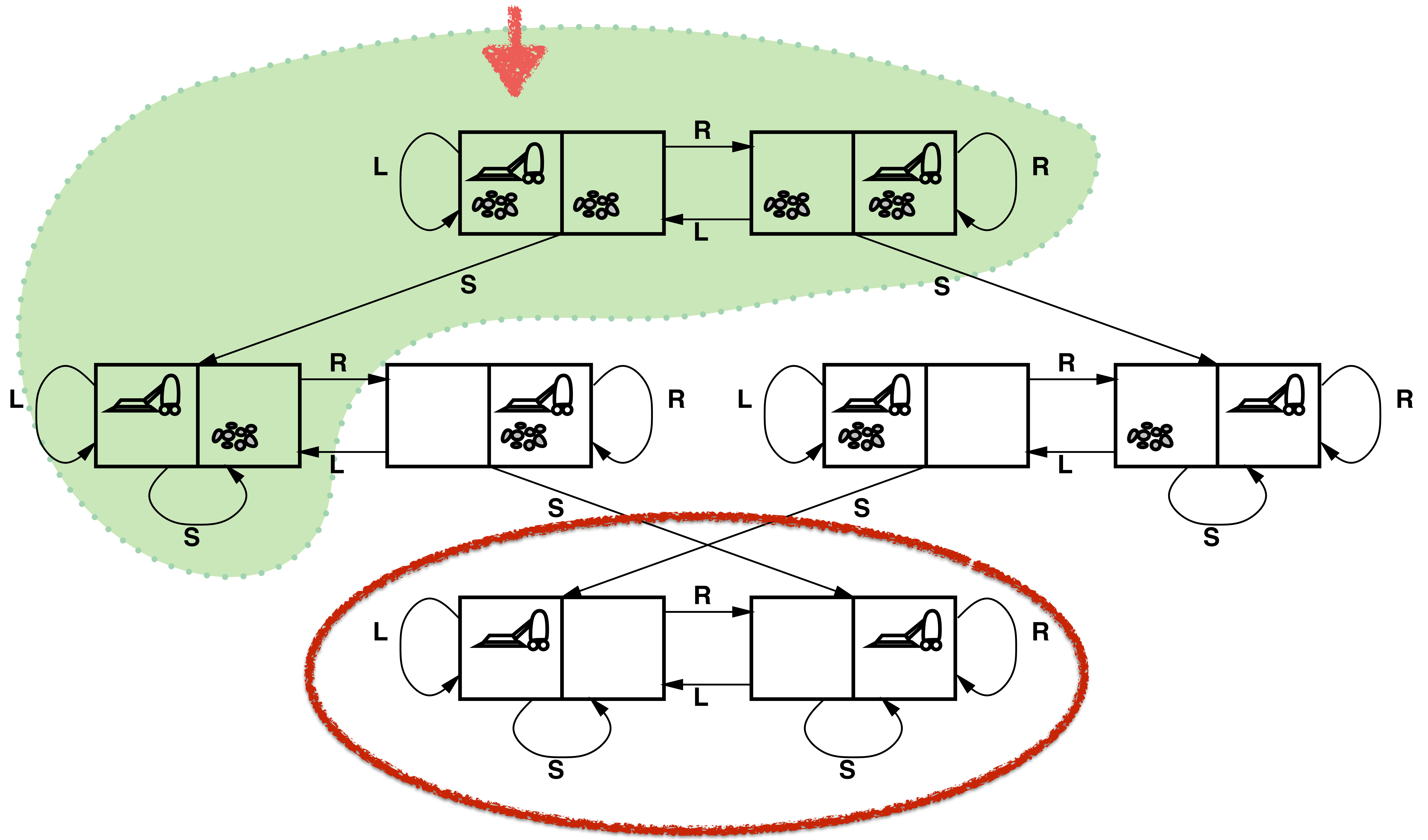
- Depth-first family
    - Depth-first search
    - Depth-limited search
    - Iterative-deepening search


R
L
L
R
S
S
R
L
L
R
L
S
R
S
L
R
L
S
S
R
L
L
R
S
S



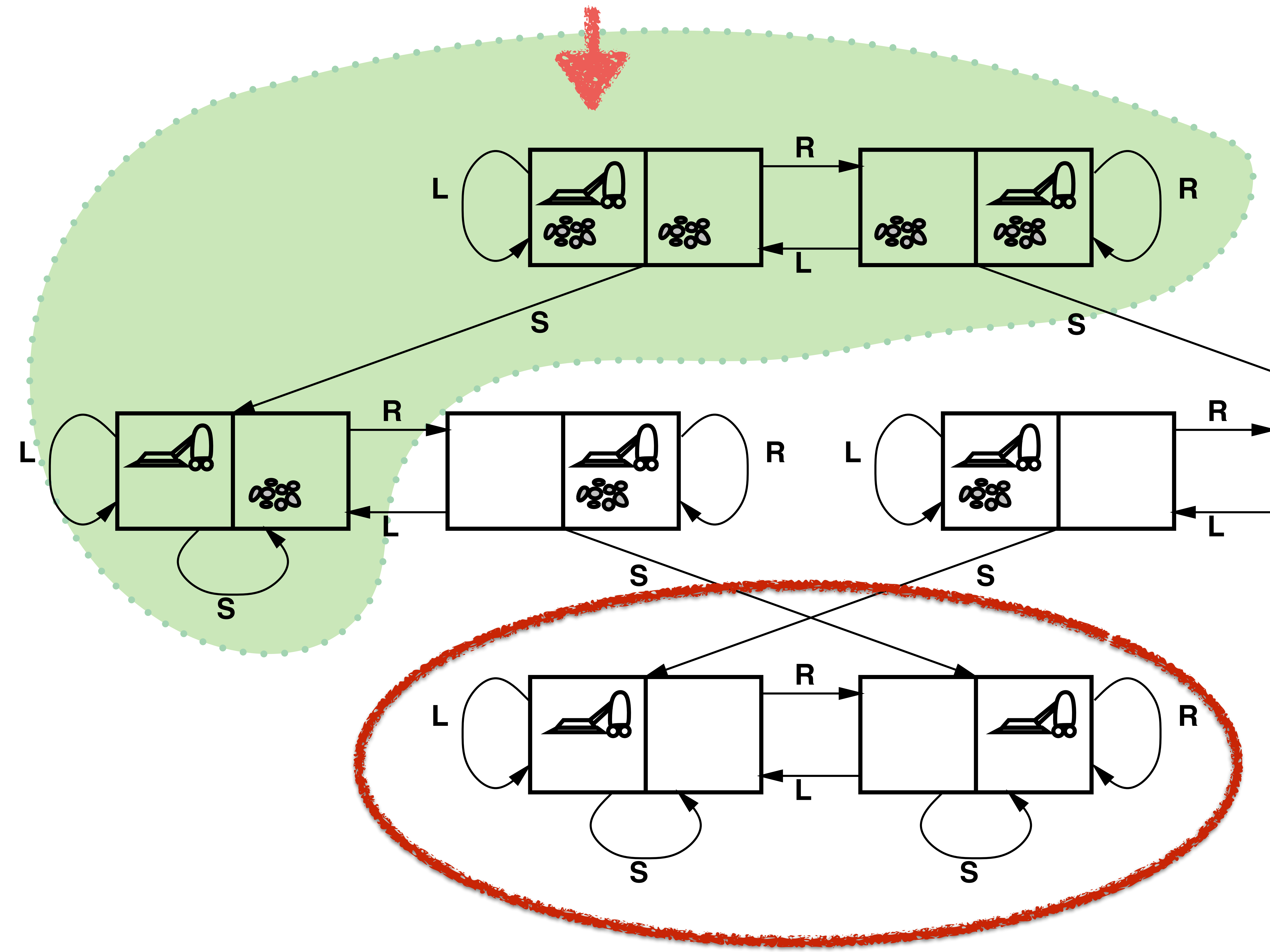
R
L
R
L
S
S
R
L
R
L
L
R
L
R
S
S
S
S
R
L
R
L
S
S

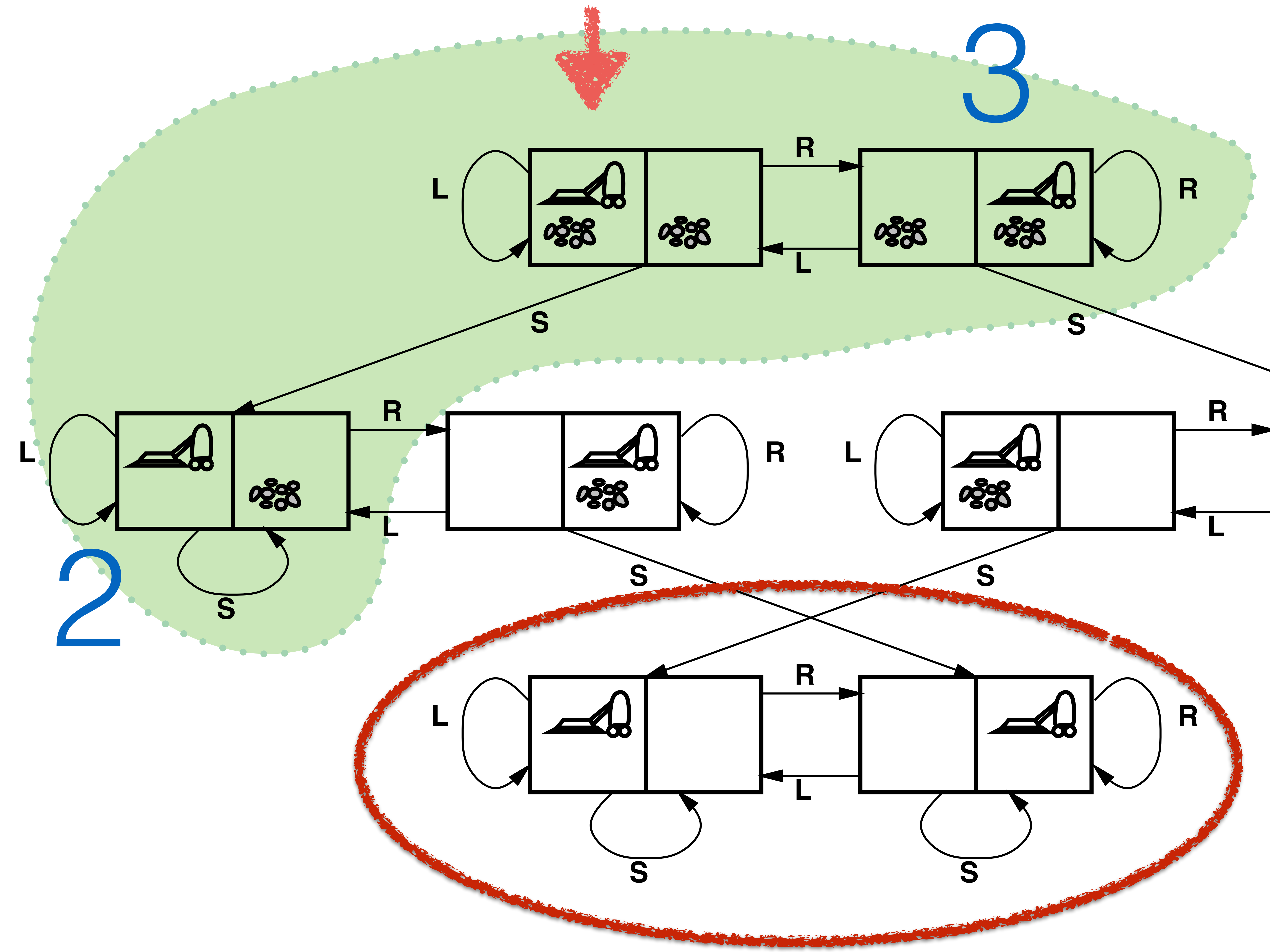

R
L
R
L
S
S
R
L
R
L
R
L
R
L
S
S
S
S
R
L
L
R
S
S



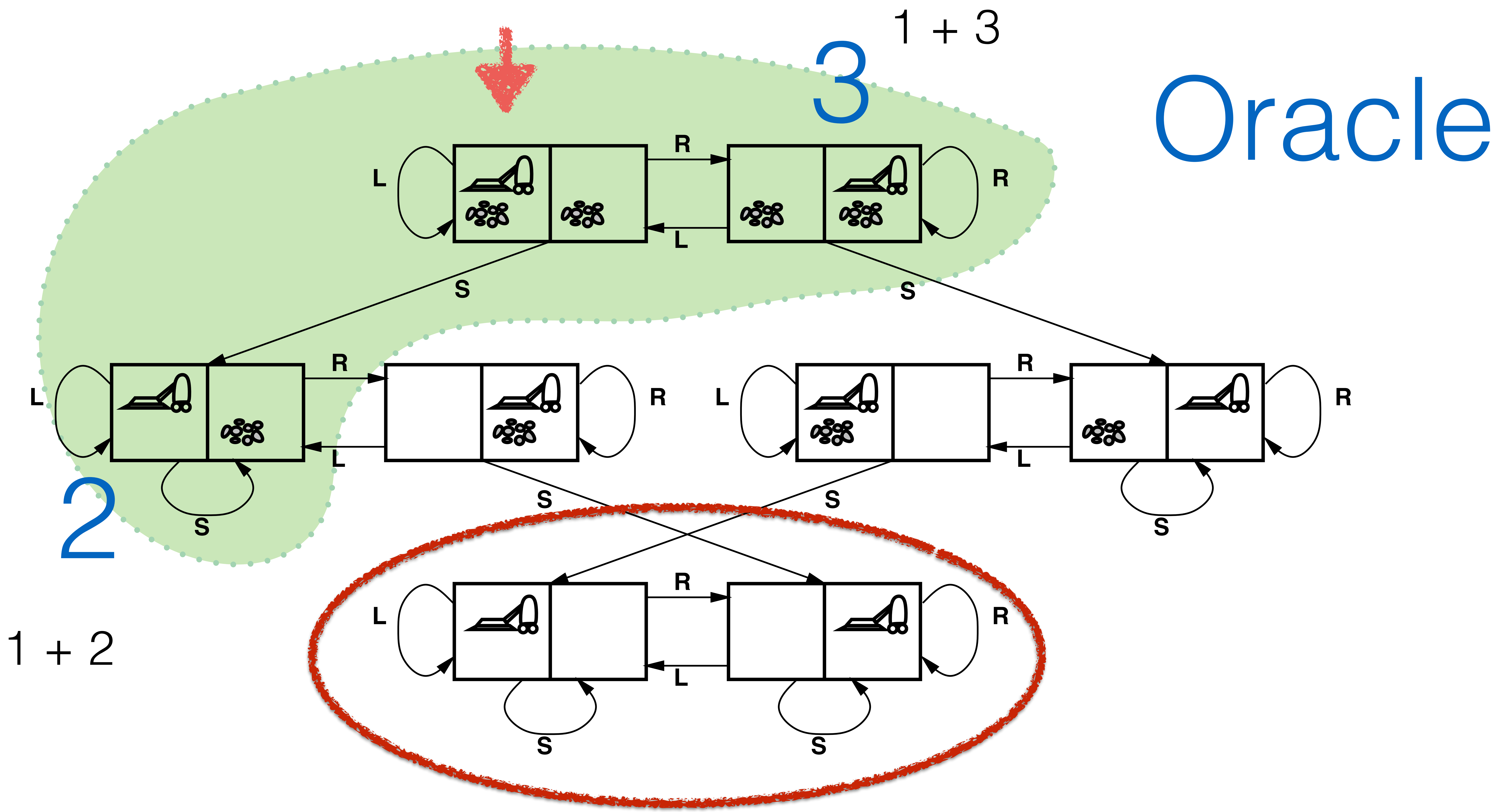
L
R
R
S
L
S
L
R
R
L
R
L
R
L
S
S
S
S
L
R
R
L
S
S

Oracle

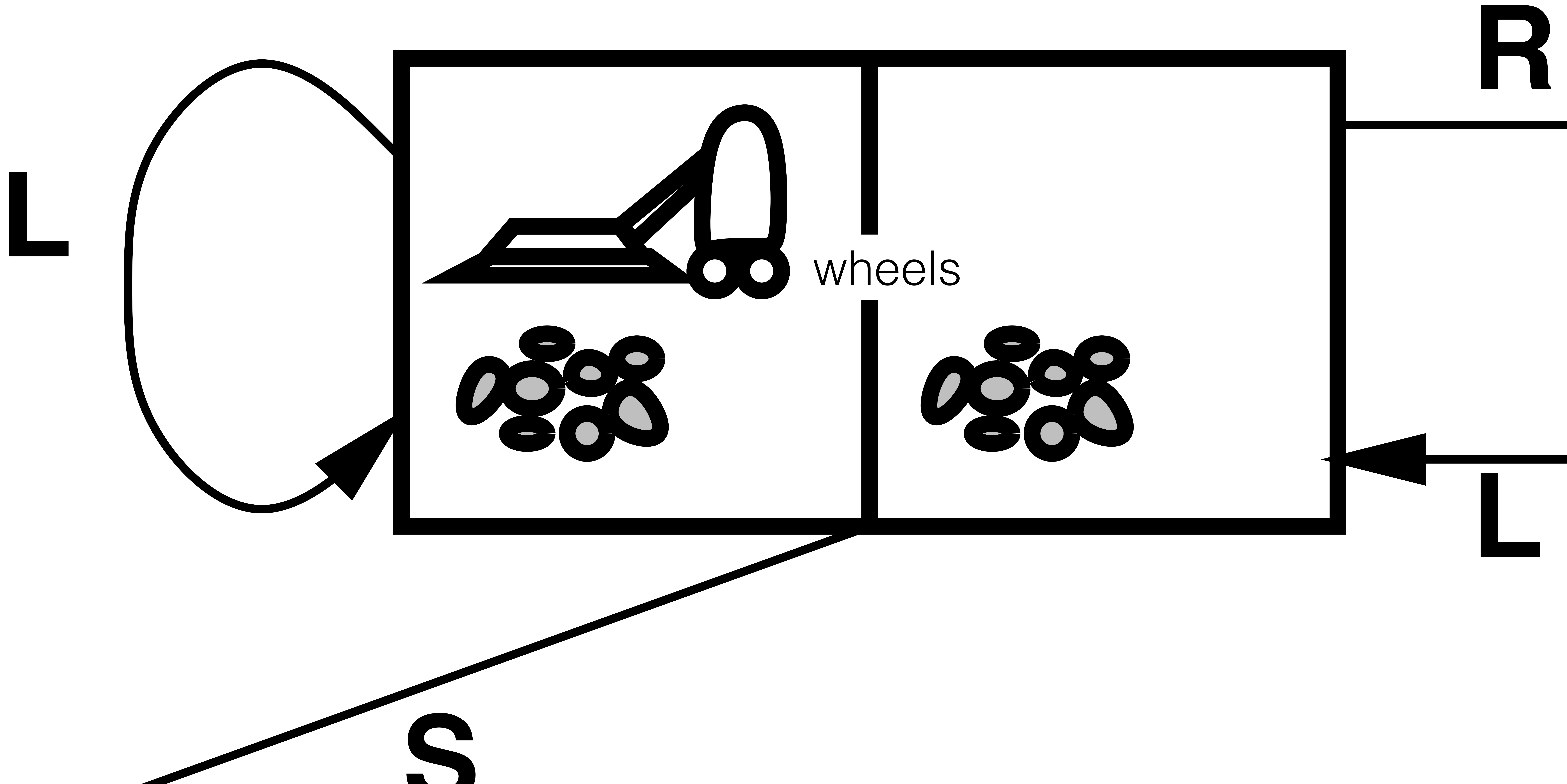




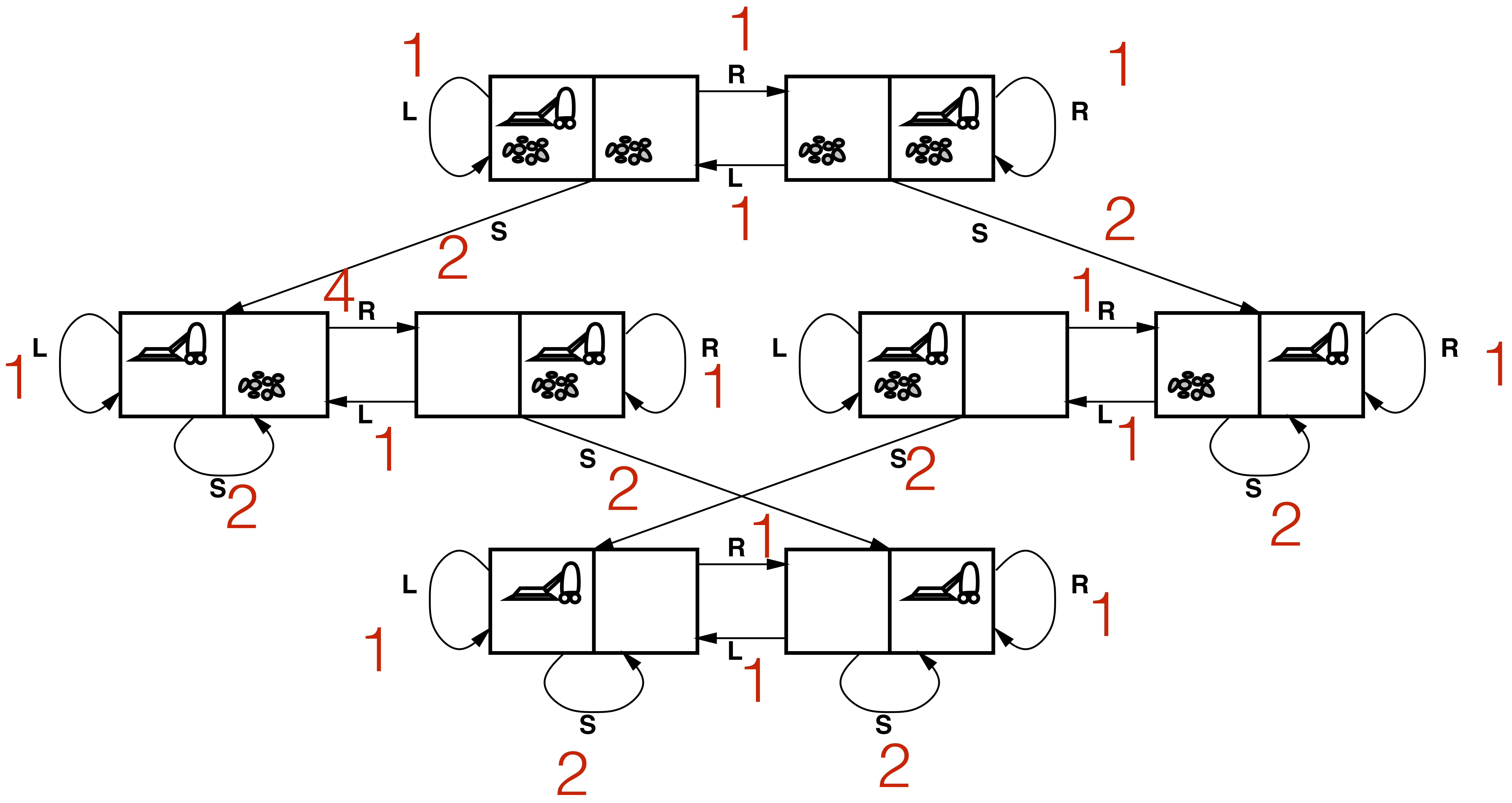
Oracle



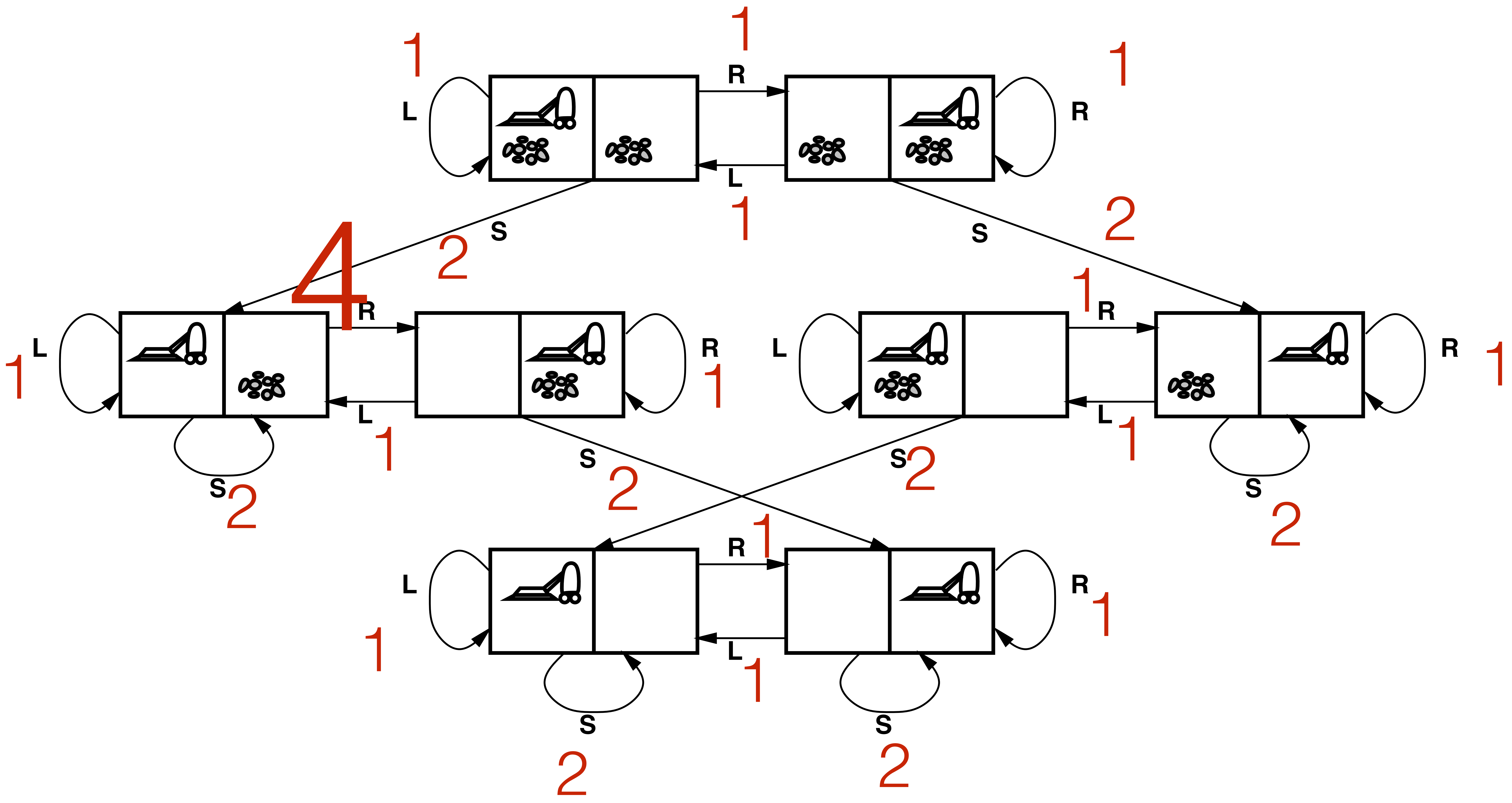
1 + 3
3
Oracle
L
R
R
L
S
S
L
R
R
L
R
L
R
L
2
S
S
S
S
1 + 2
L
R
R
L
S
S



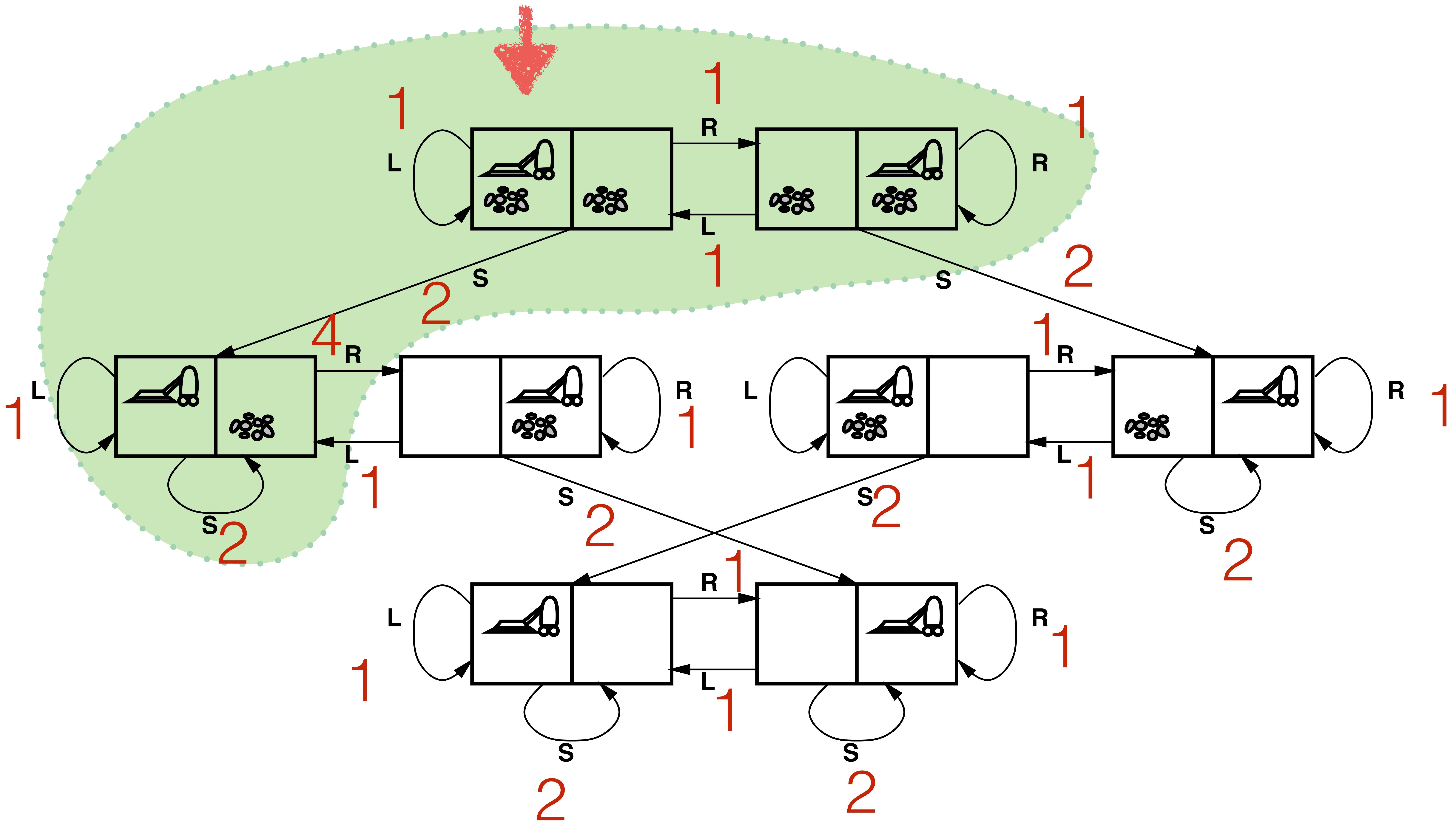
L
wheels
R
L
S



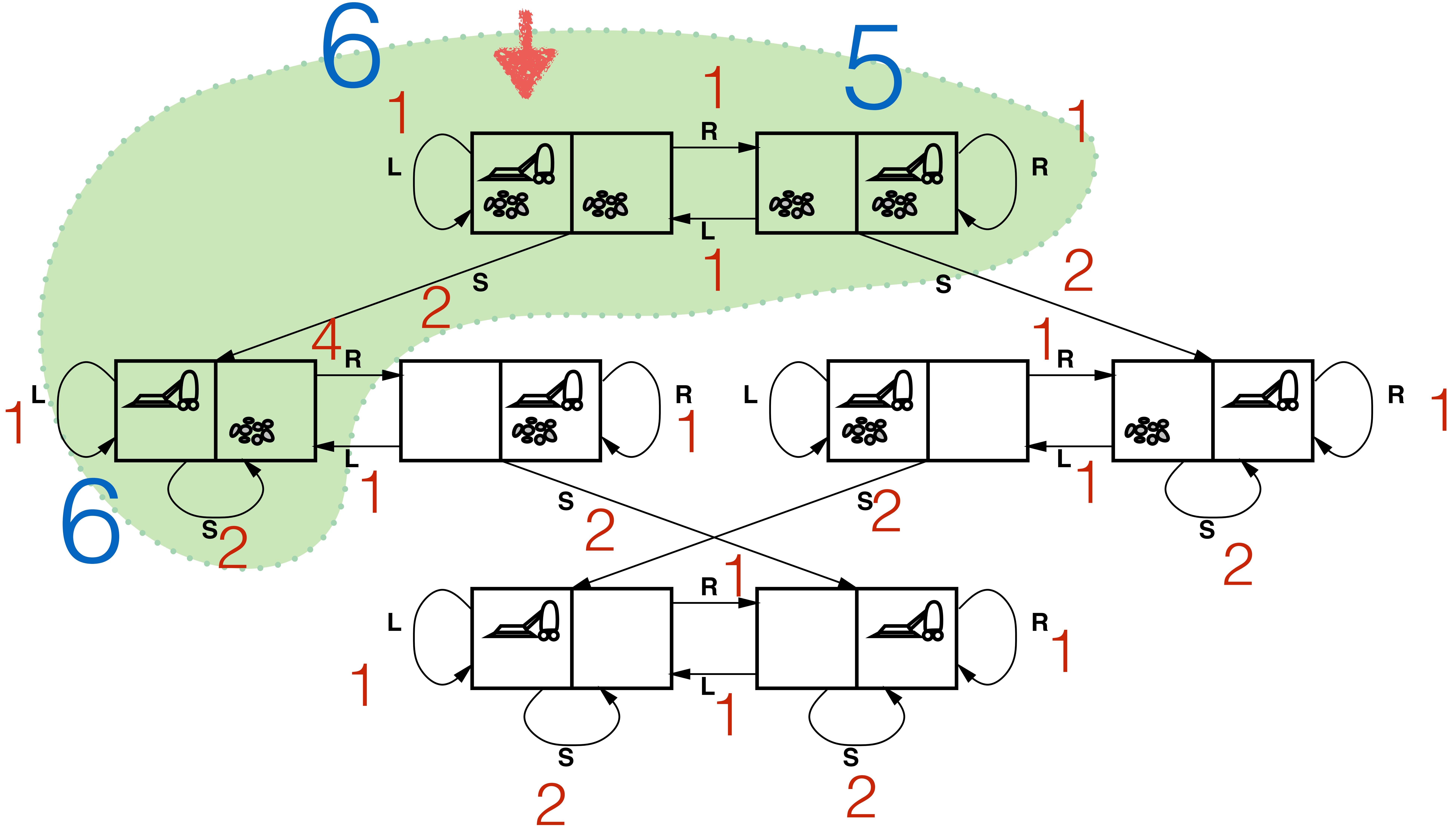
1
1
1
L
R
R
L
1
S
2
S
2
4
R
1
R
L
1
R
L
R
1
1
L
L
1
1
S
2
S2
S
2
S
2
R
1
L
R
1
1
L
1
S
S
2
2



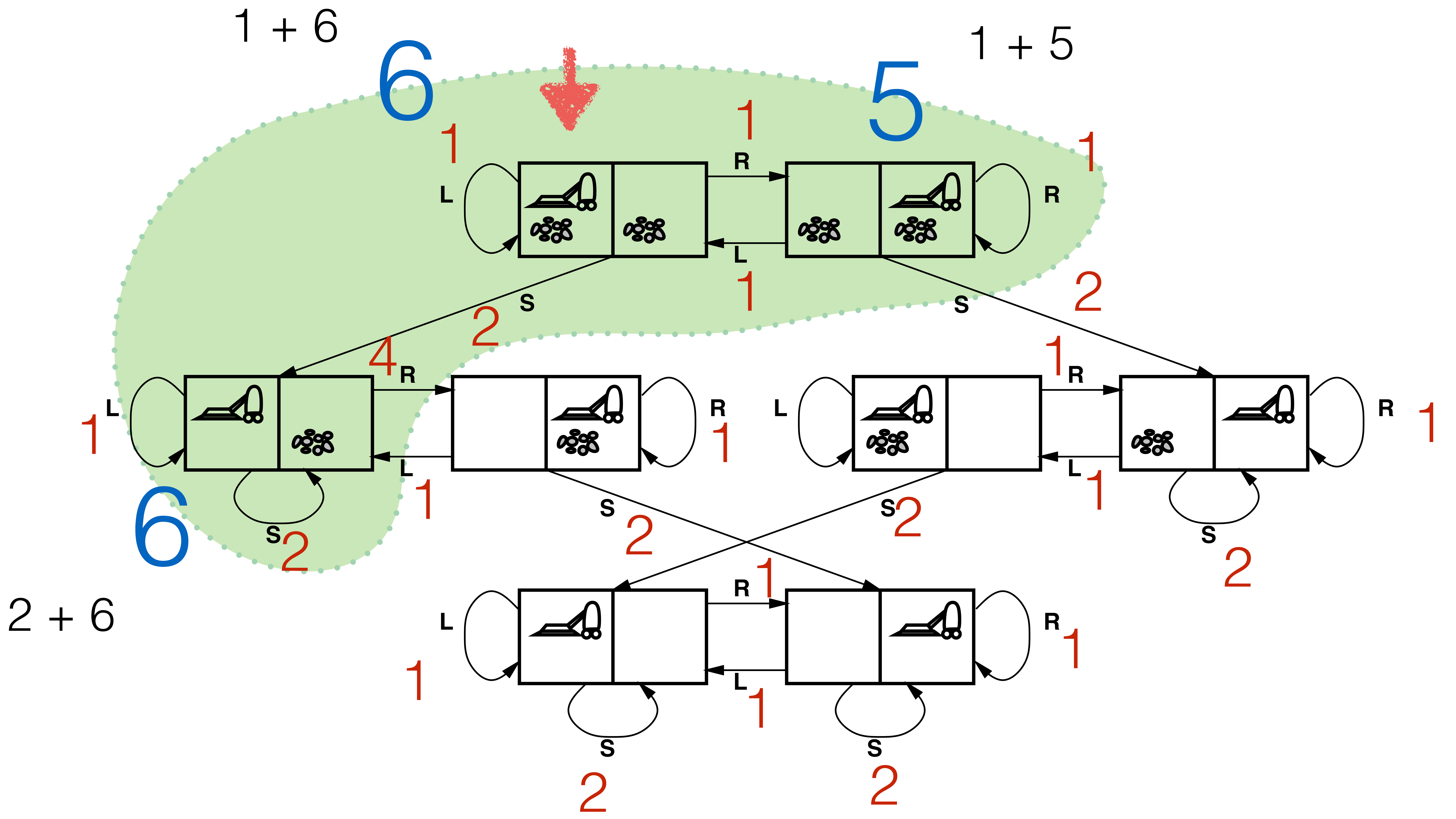
1
L
1
R
1
R
L
1
S
2
S
2
4
R
1
L
1
L
1
R
1
L
R
1
S
2
L
1
S
2
S
2
1
R
L
1
R
1
1
L
1
S
2
S
2



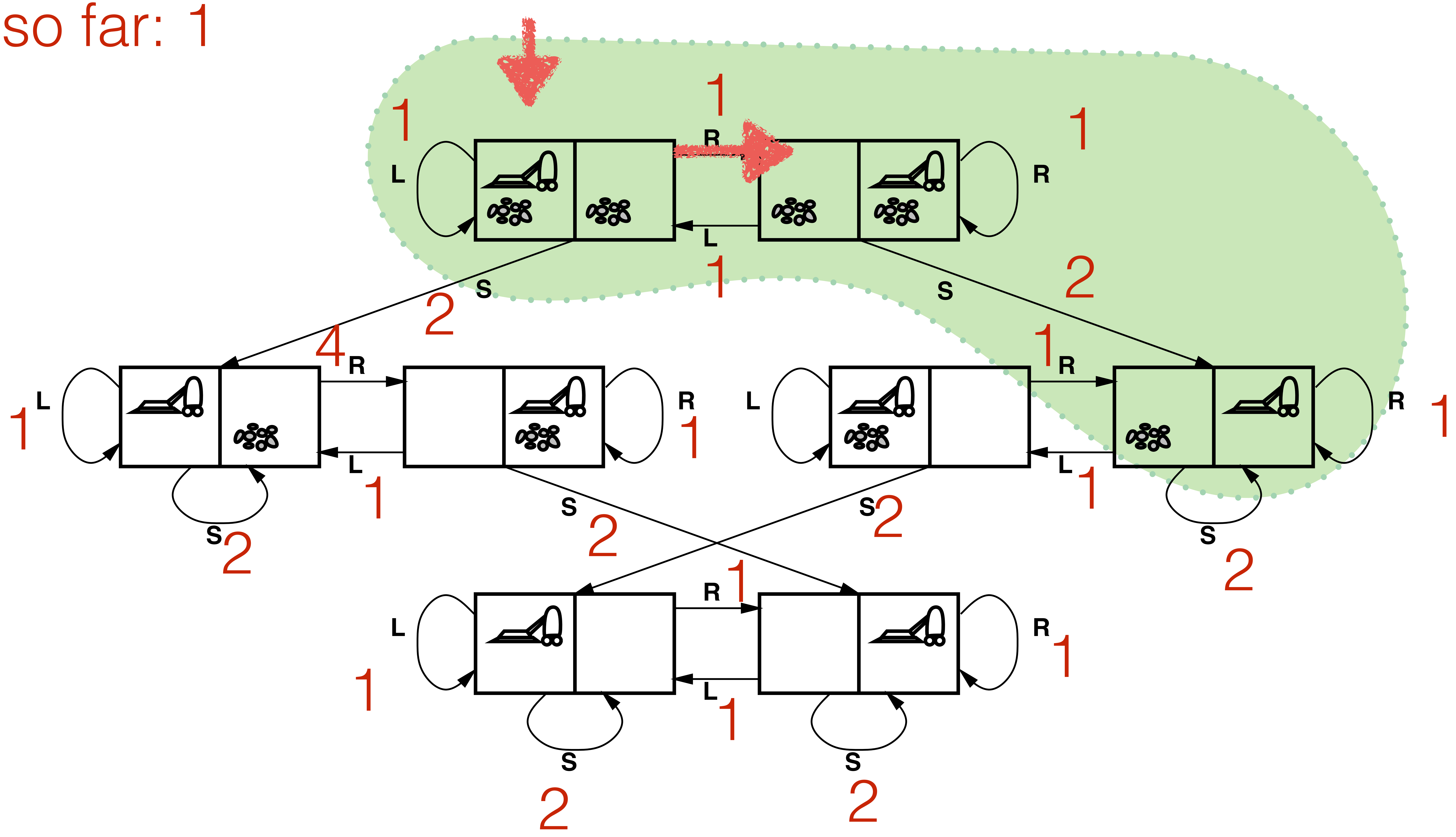
1
1
1
L
R
R
L
1
S
S
2
2
4
R
L
R
L
R
1
1
1
1
L
L
1
1
S
S
S
S
2
2
2
2
1
R
L
R
1
1
L
1
S
S
2
2



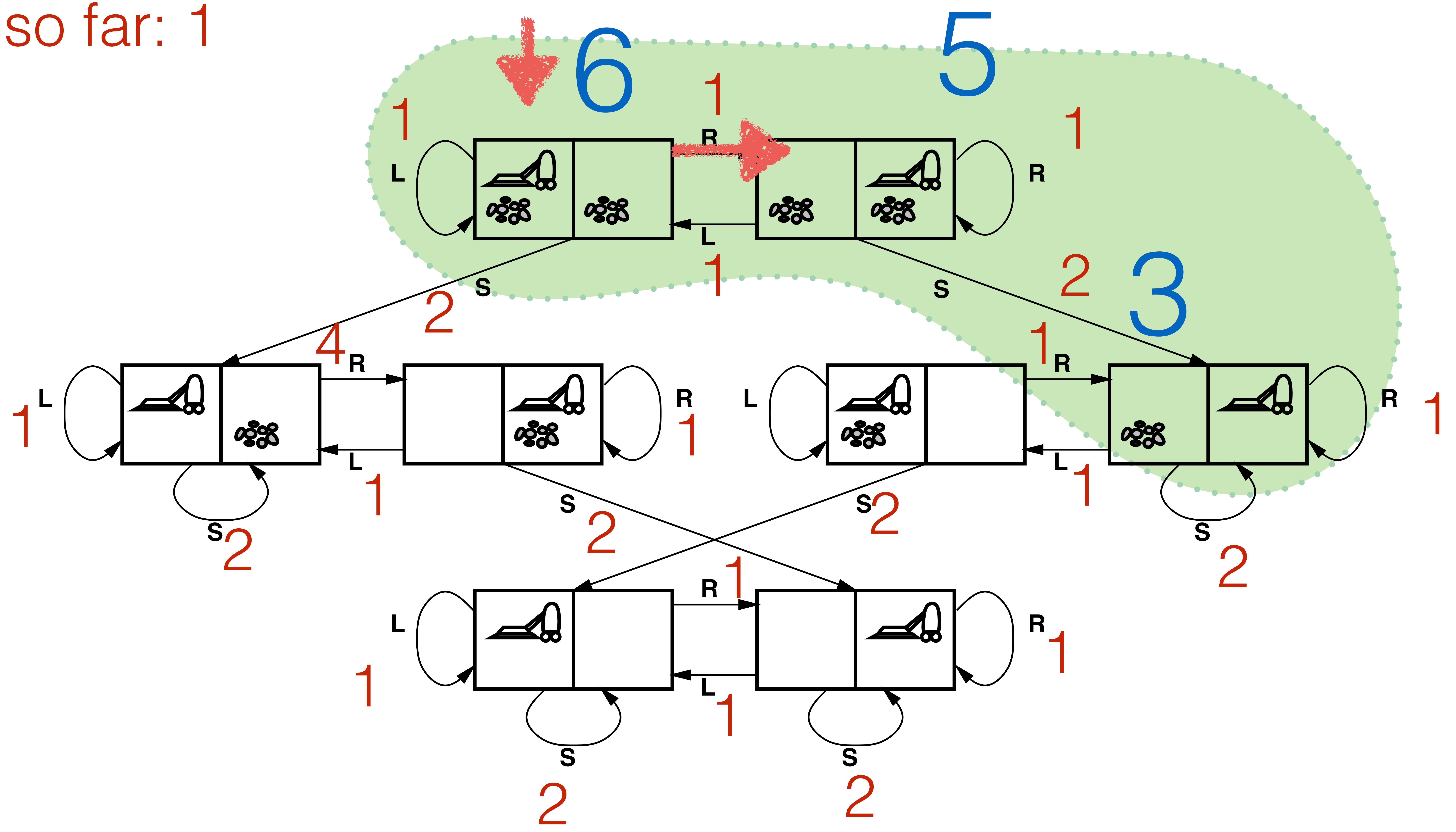
6
5
6
L
R
S


1 + 6
6
1 + 5
5
2 + 6
6
R
L
S

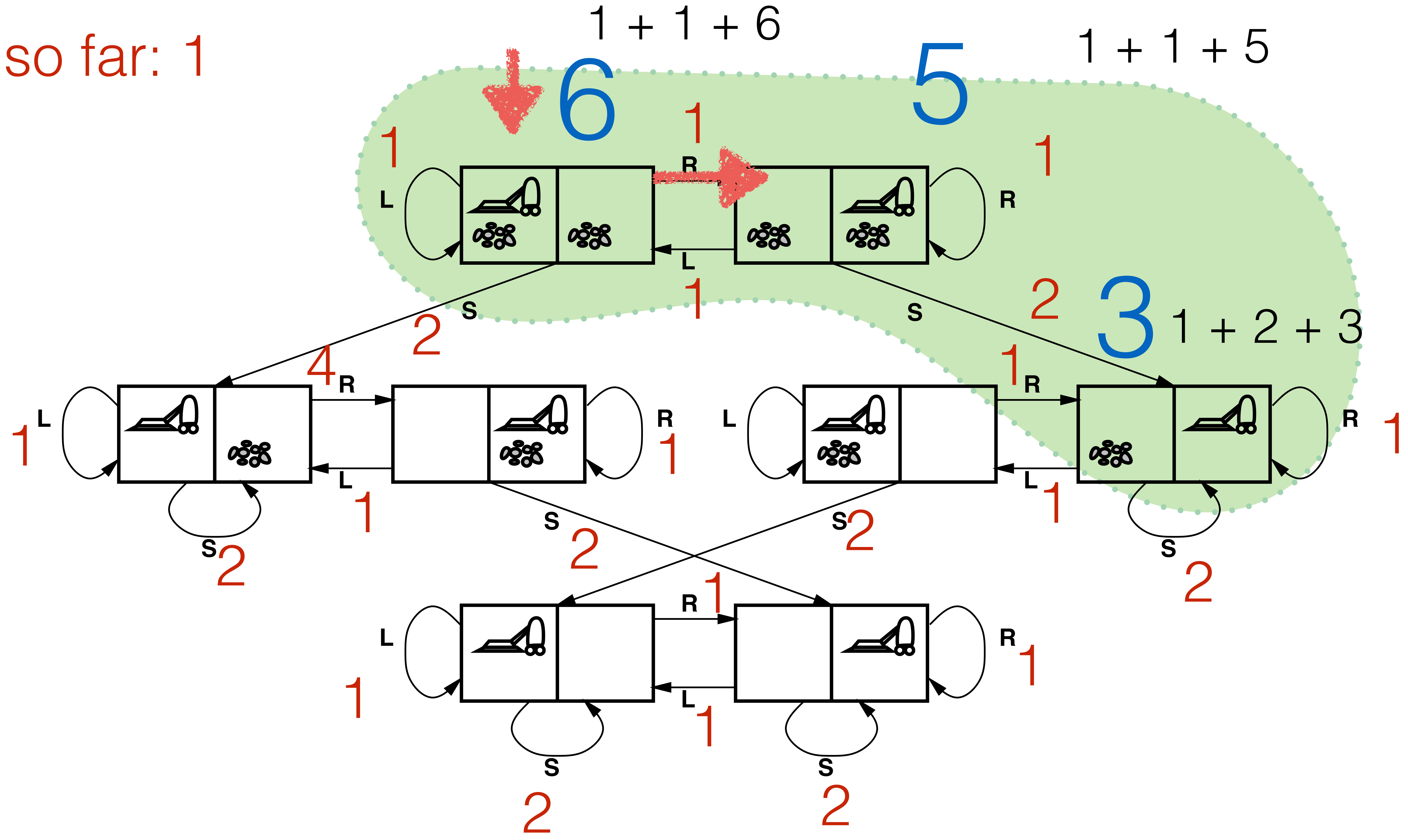
cost so far: 1



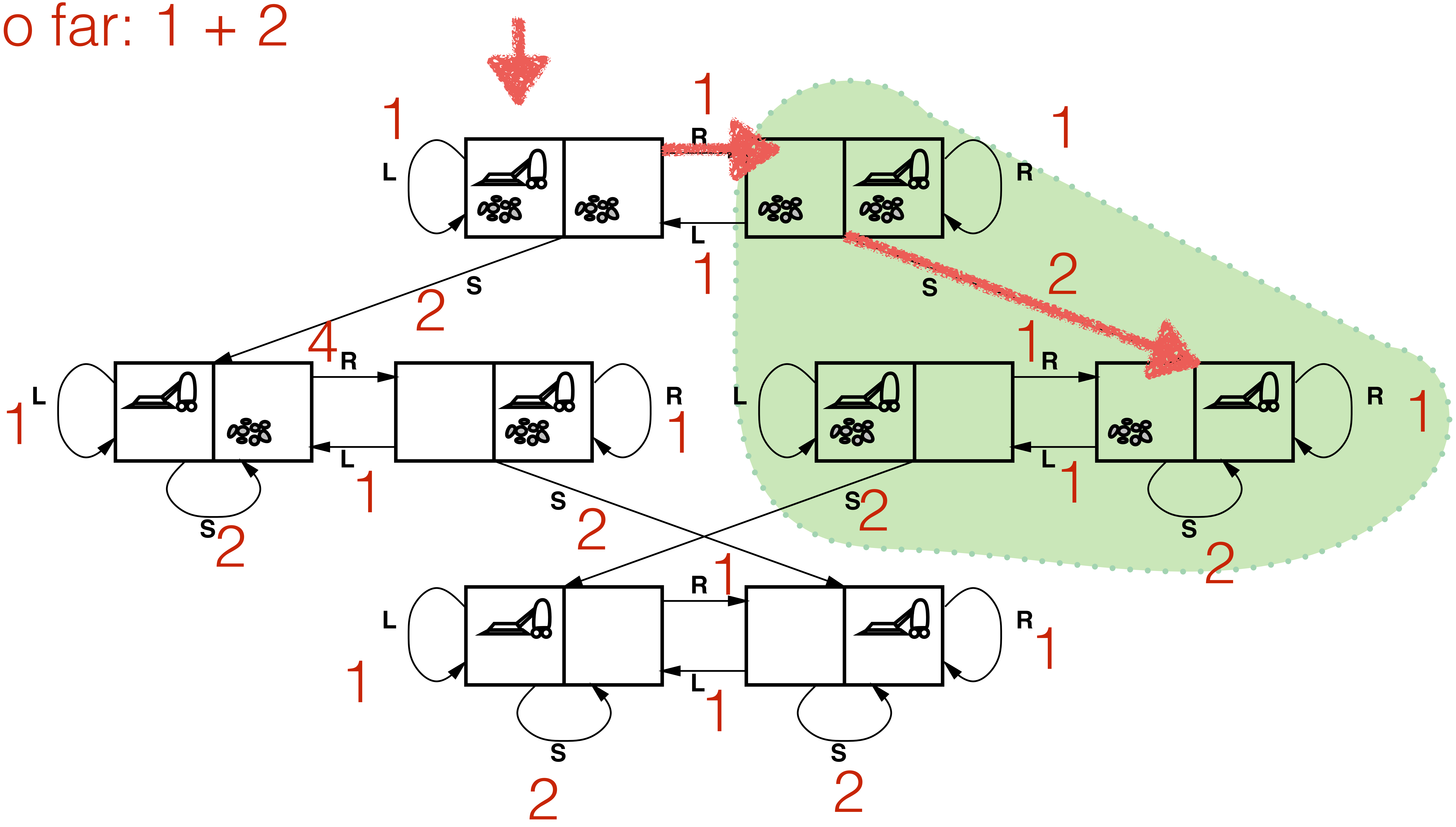
cost so far: 1



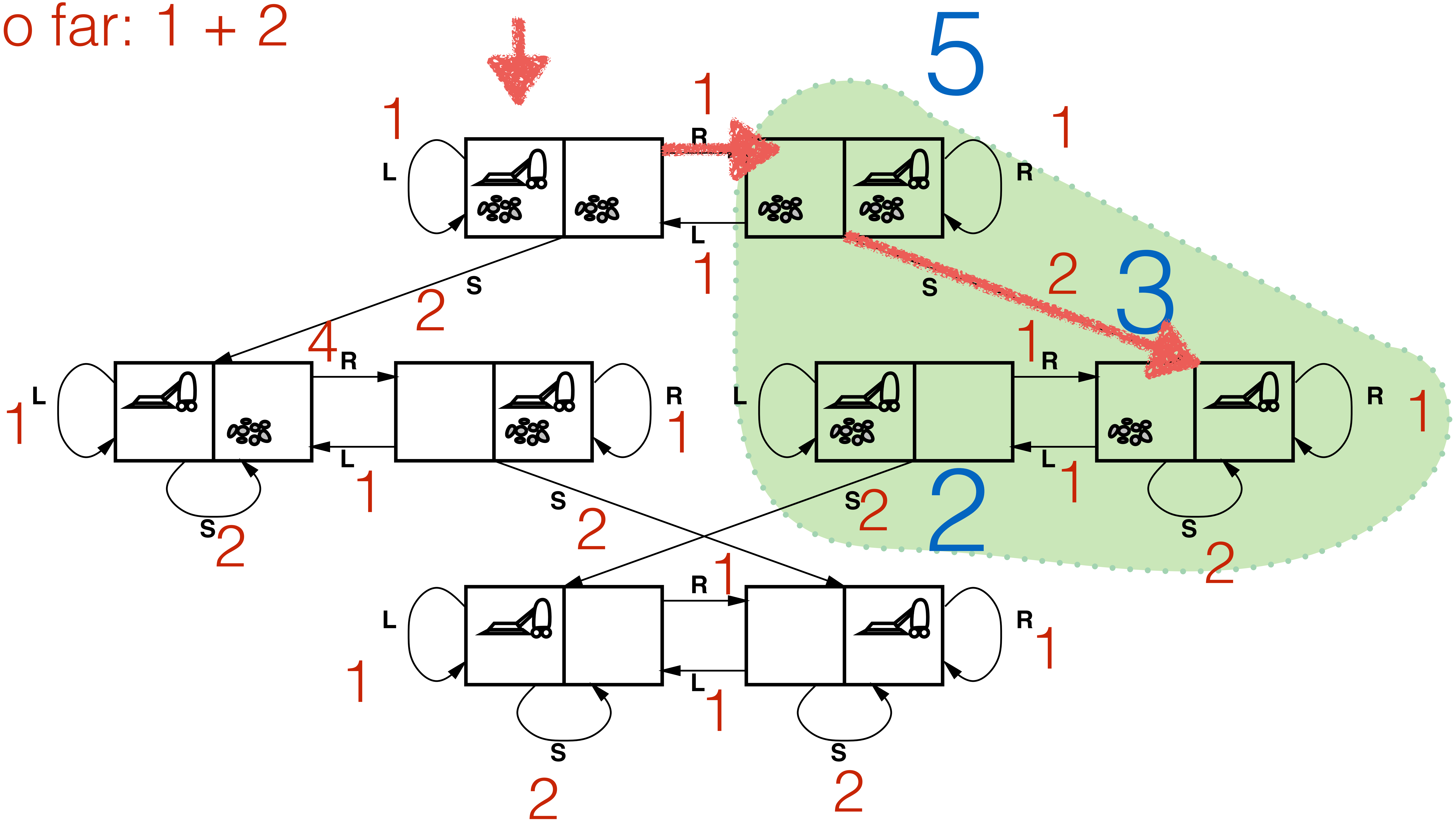
cost so far: 1



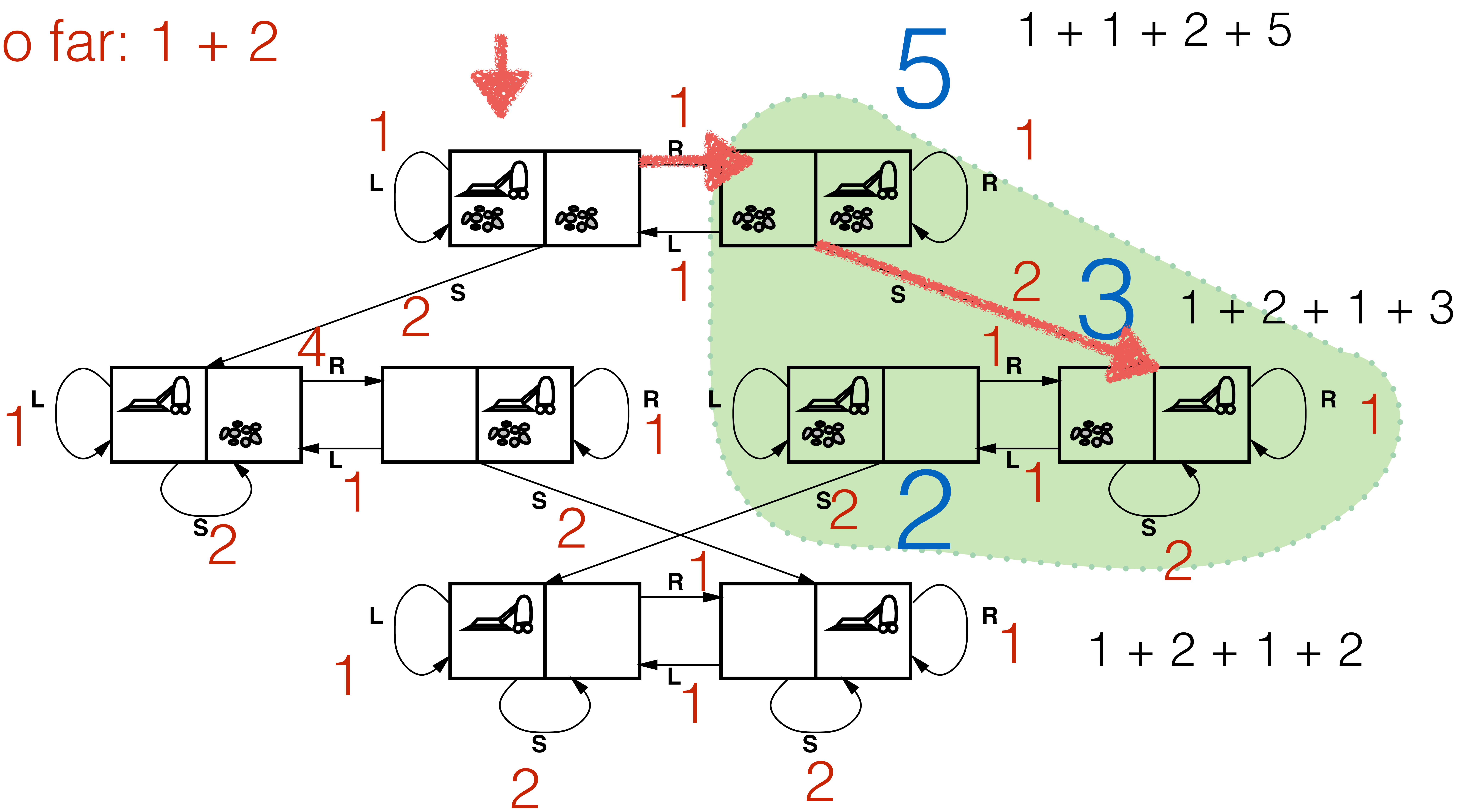
cost so far: 1 + 2



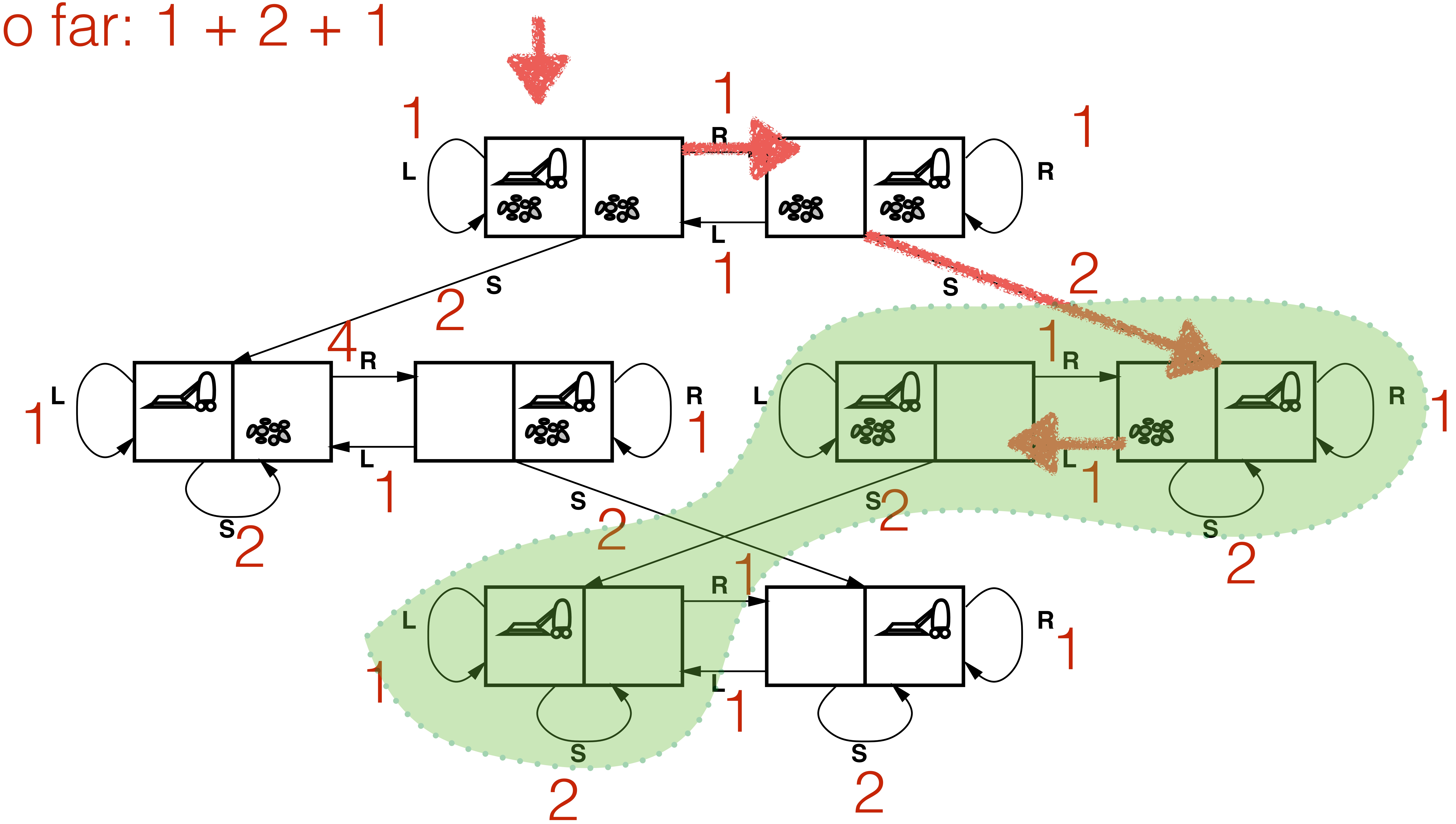
cost so far: 1 + 2



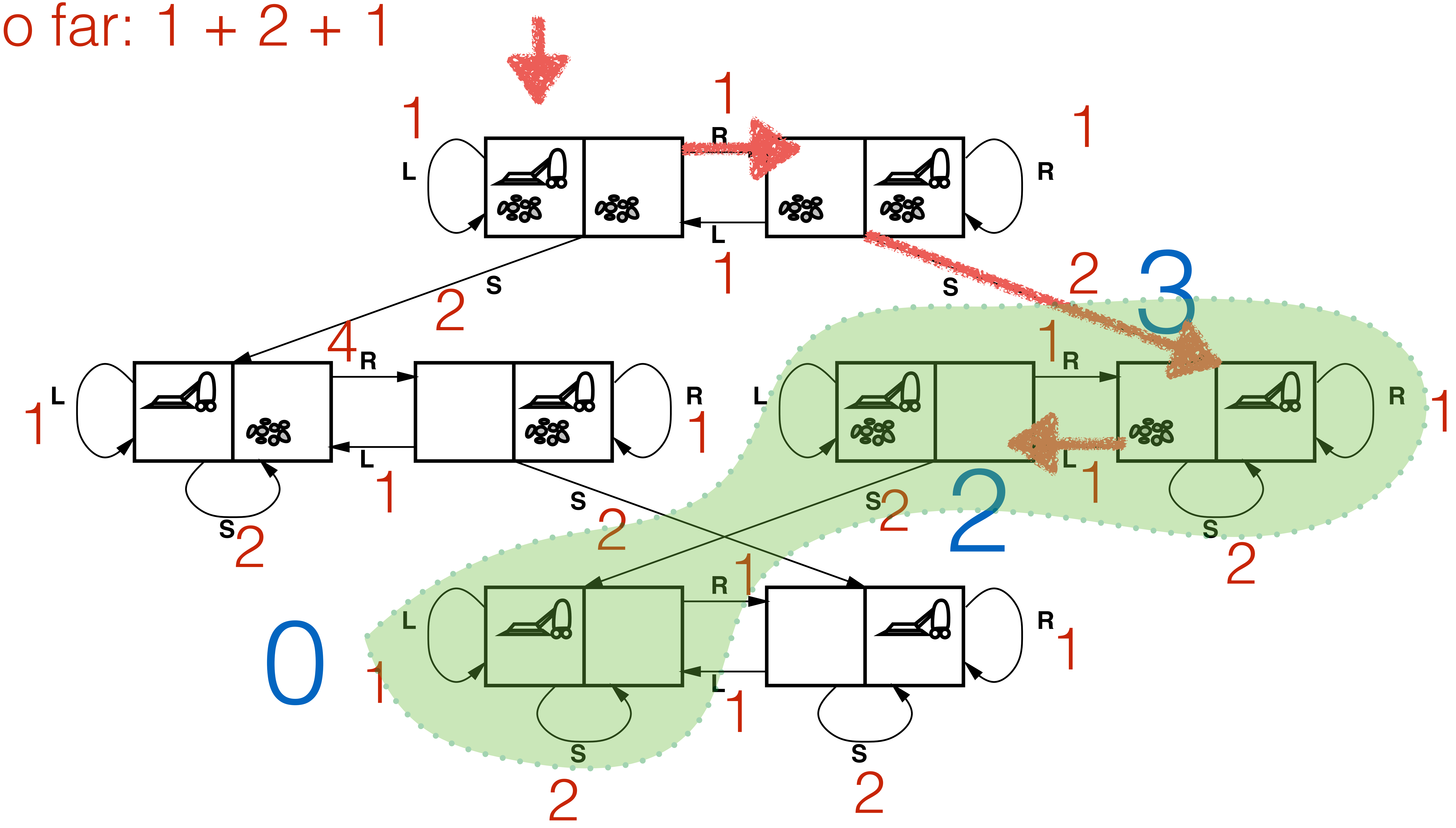
cost so far: 1 + 2



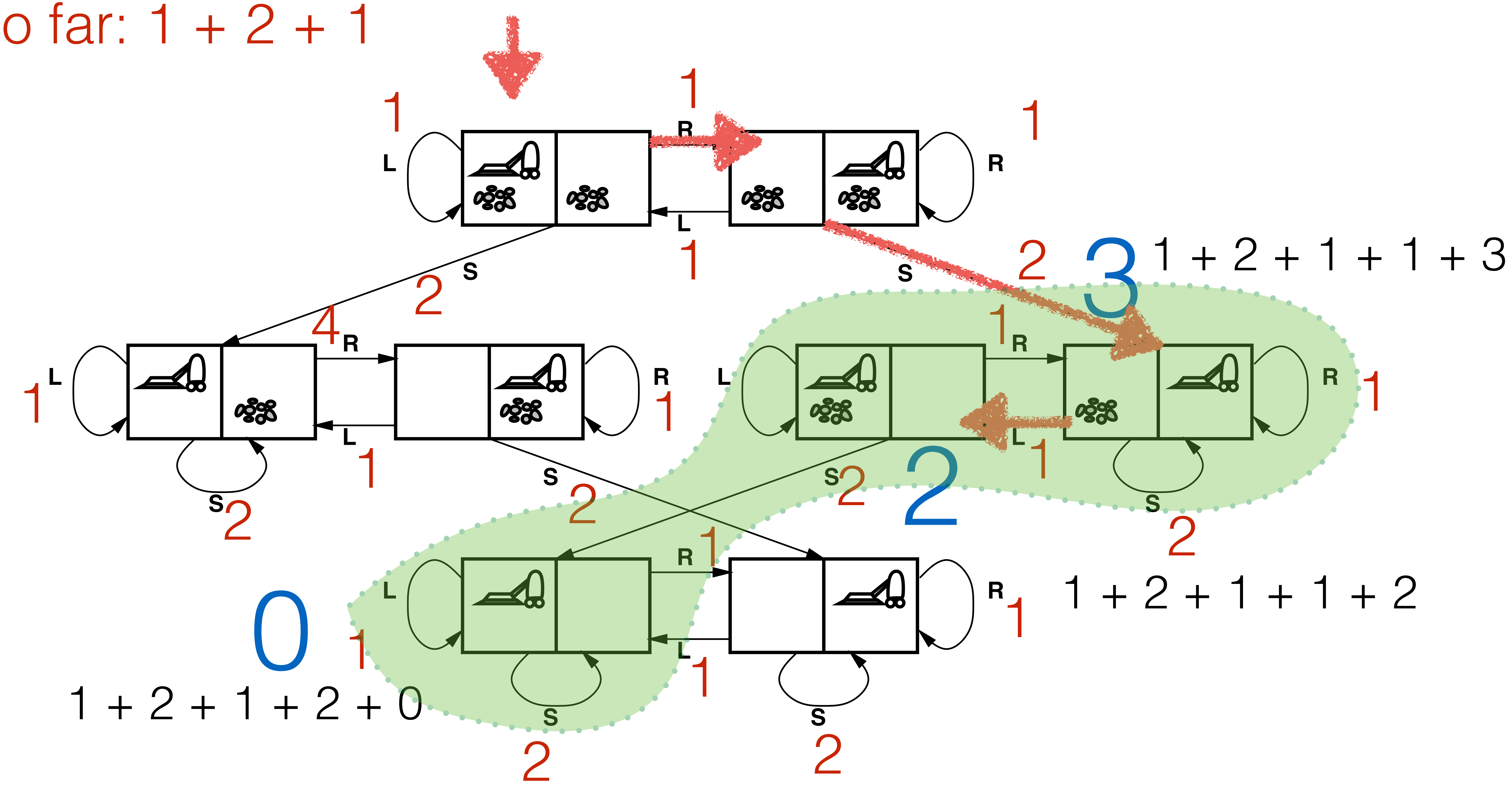
cost so far: 1 + 2 + 1



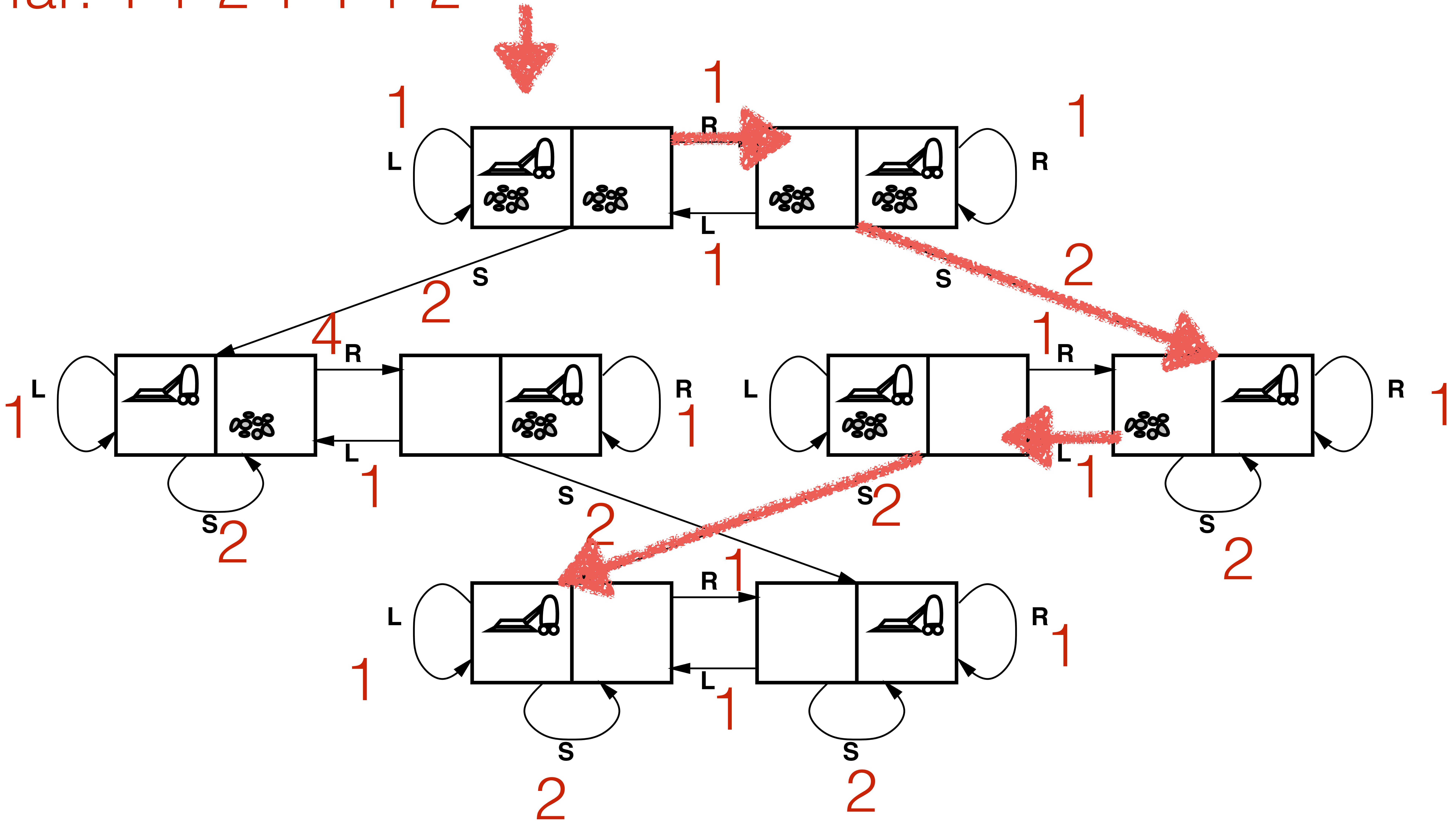
cost so far: 1 + 2 + 1



cost so far: 1 + 2 + 1



cost so far: 1 + 2 + 1 + 2



A* Search

- Expand node in frontier with best evaluation function score **f(n)**
  - **f(n) = g(n) + h(n)**
  - **g(n) :=** cost to get from initial state to **n**
  - **h(n) :=** heuristic estimate of cost to get from **n** to goal
- Completeness, optimality, and time & space depend on **h**