MODULE 1 / UNIT 3

DIVIDE & CONQUER ALGORITHMS

Today

- Insertion sort for large data
- Tower of Hanoi
- Recursion
- Divide and conquer algorithms
- Mergesort

Recap : Insertion Sort Algorithm

Data: An unsorted list $A[1 \cdots n]$
Result: The list $A[1 \cdots n]$ is sorted
for $j = 2$ **to** n **do**
    $key = A[j]$;
    $i = j - 1$;
    while $i > 0$ *and* $A[i] > key$ **do**
        $A[i+1] = A[i]$;
        $i = i - 1$;
    end
    $A[i+1] = key$;
end

Sorting many, many books

ravenoaks.net

A faster-than-insertion-sort

Suppose that insertion sort takes *T(n) = cn²*

1. **Divide the elements into half : O(1)**
2. **Sort each of the half using insertion sort**

$$c\left(\frac{n}{2}\right)^2 + c\left(\frac{n}{2}\right)^2 = \frac{c}{2}n^2$$

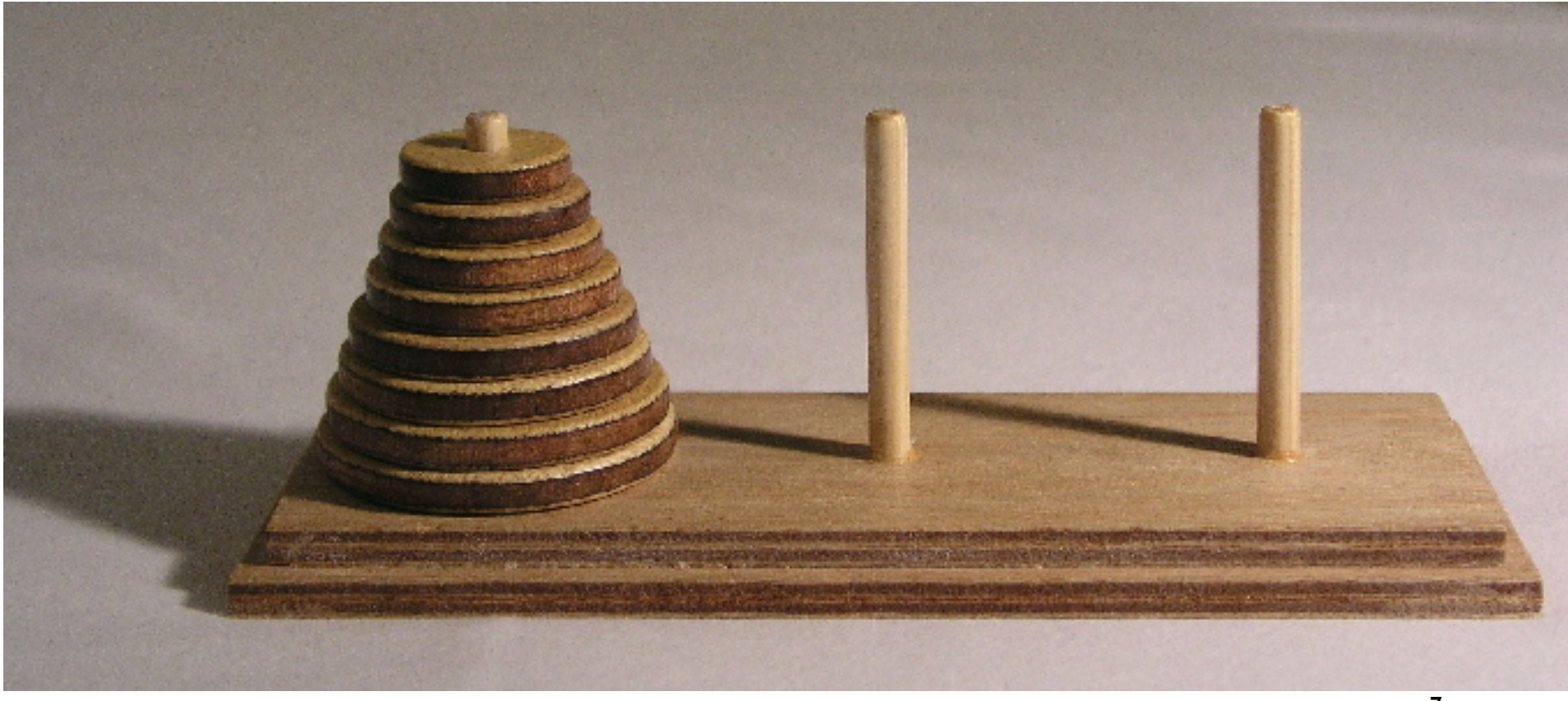
3. **Merge the two sorted halves : *O(n)***

$$T'(n) = \frac{c}{2}n^2 + O(n) + O(1) \ll T(n) = cn^2$$

A better idea that might work…

- **Divide the elements into half**
- **Sort each of the half**
  - If the half is still big, divide them into two again…
  - If small just use simple sorting
- **Merge the two sorted halves**
- **How?**

The Tower of Hanoi Problem



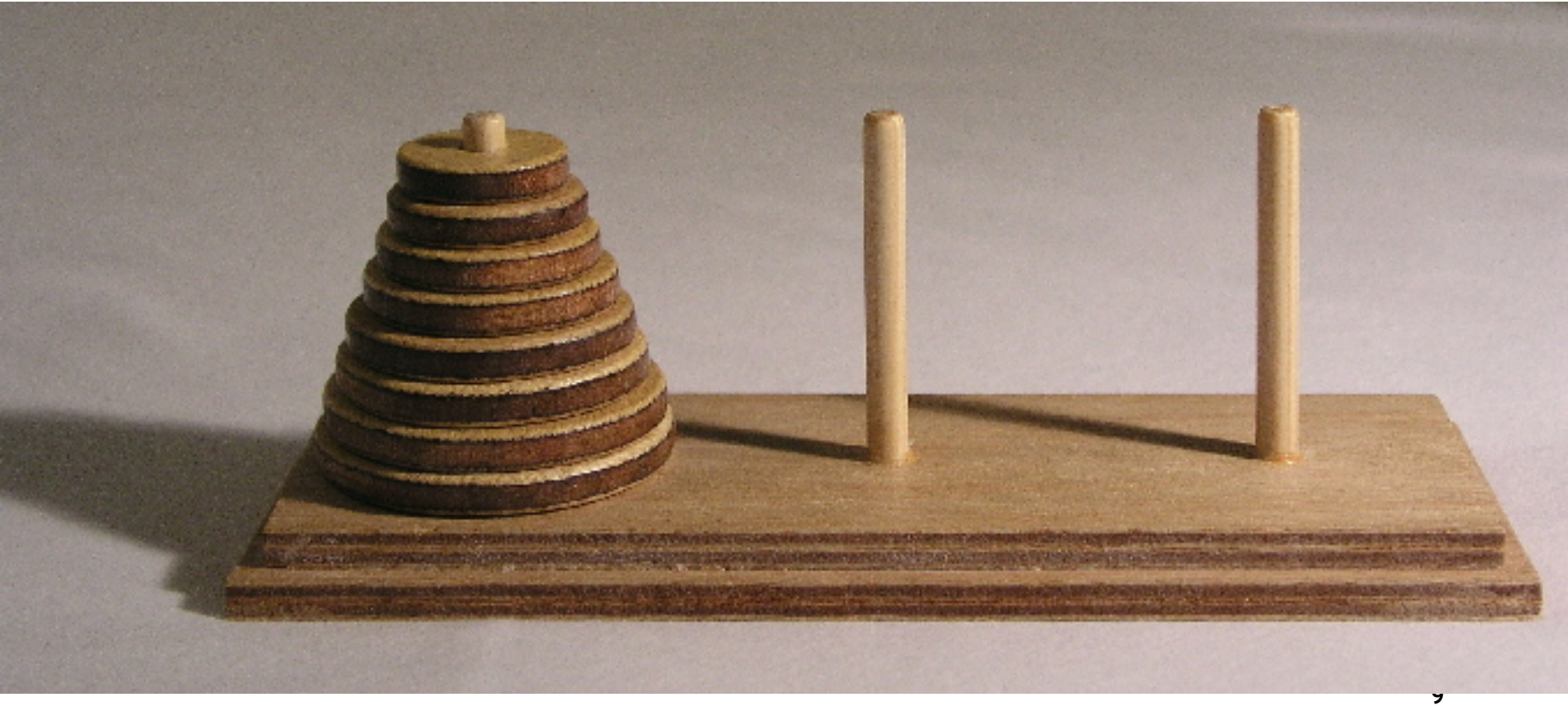
Rules

1. **The goal is to move all disks to the rightmost pole.**

2. **One disk can be moved at a time.**

3. **Larger disk cannot be placed on top of smaller disk.**

How many moves are needed for 8 disks?



An animated example solution

http://www.youtube.com/watch?v=aGlt2G-DC8c

Expected Output : A sequence of instructions

```
$ python ./solve_tower_of_hanoi.py 3 left center right
Move disk 1 from left pole to right pole
Move disk 2 from left pole to center pole
Move disk 1 from right pole to center pole
Move disk 3 from left pole to right pole
Move disk 1 from center pole to left pole
Move disk 2 from center pole to right pole
Move disk 1 from left pole to right pole
```

Divide and conquer algorithms

Solve a problem recursively,
applying three steps at each level of recursion

Divide	the problem into a number of subproblems that are smaller instances of the same problem
Conquer	the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner
Combine	the solutions to subproblems into the solution for the original problem.

Recursion: definition

Short, but incomplete definition

Recursion See "Recursion"

A complete definition

Recursion If you still don't get it, see "Recursion"

Key components of recursion

- **A function that is part of its own function**

- **Terminating condition**
  - To avoid infinite recursion

A simple recursion

```
## function to add from 1 to n
def sum(n):
    if ( n == 0 ):
        return (0)
    else:
        return (n + sum(n-1))
```

Key Idea for the Tower of Hanoi Problem

***Assume** that you already know how to move n-1 disks*

*And **divide** the problem into smaller pieces*

1. **Move the smaller n-1 disks from [src] to [med]**
2. **Move the largest disk from [src] to [dst]**
3. **Move the smaller n-1 disks from [med] to [dst]**

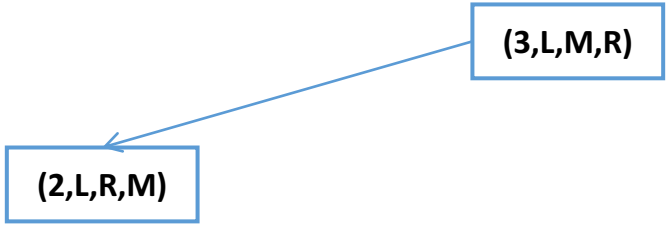
Implementing Tower of Hanoi function

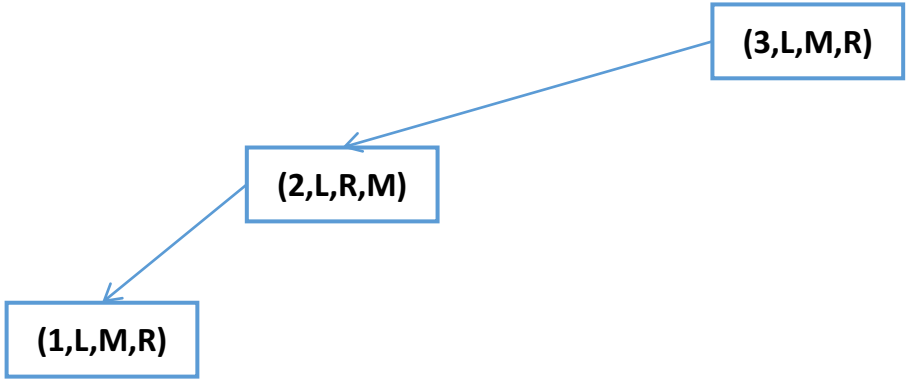
```
def tower_of_hanoi(....):
   ......?
   ......?
   ......?
```

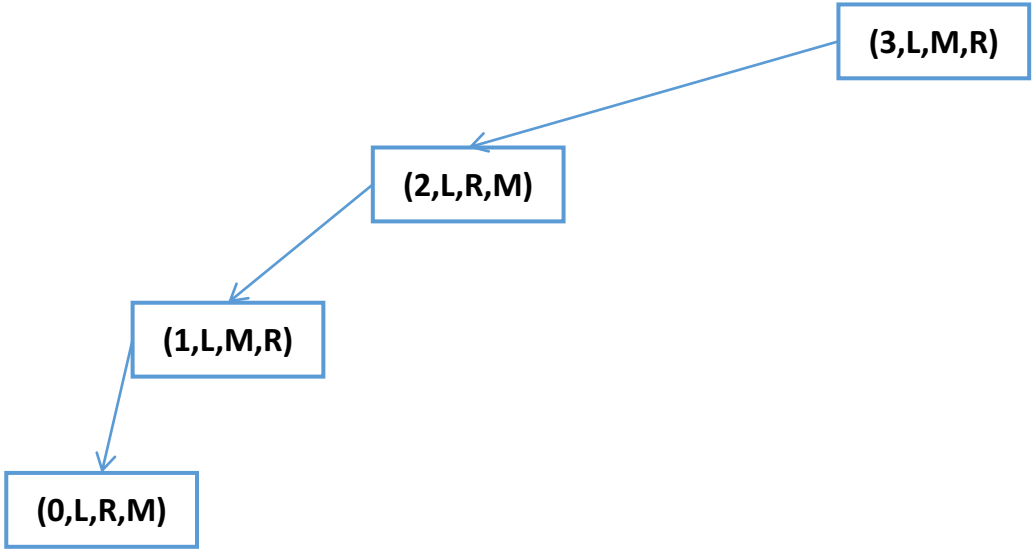
- How should the function be parameterized?
- How should recursions be called?
- When should the messages displayed?

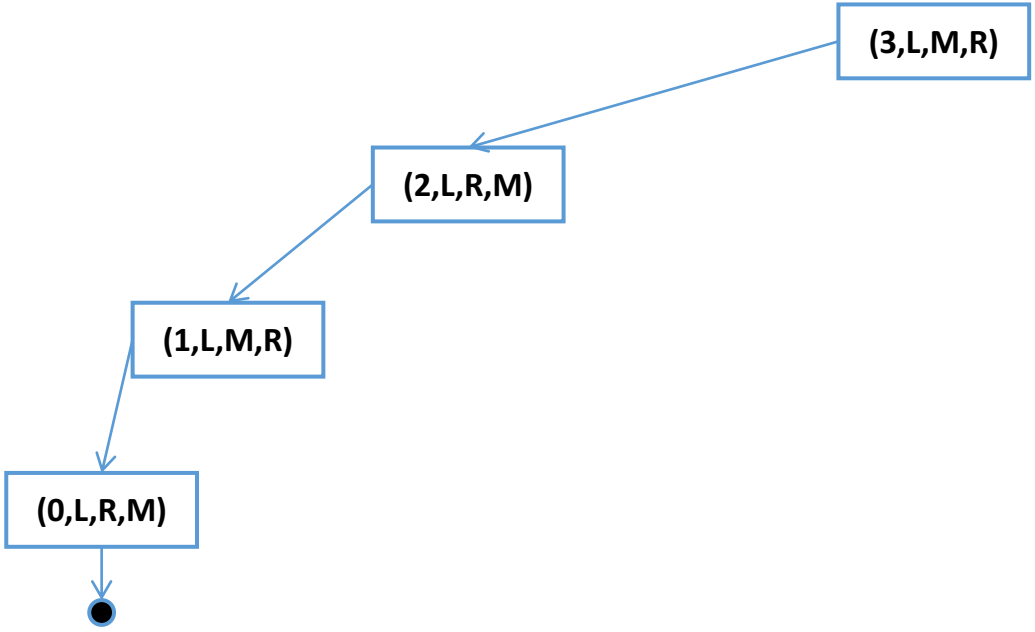
tower_of_hanoi.ipynb

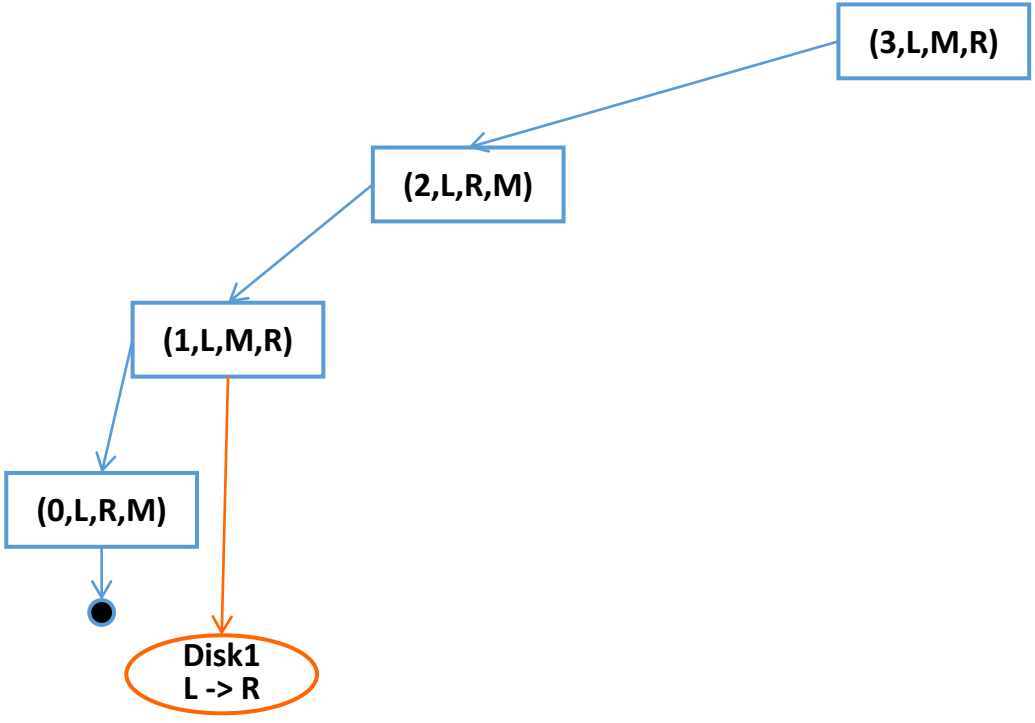
(3,L,M,R)

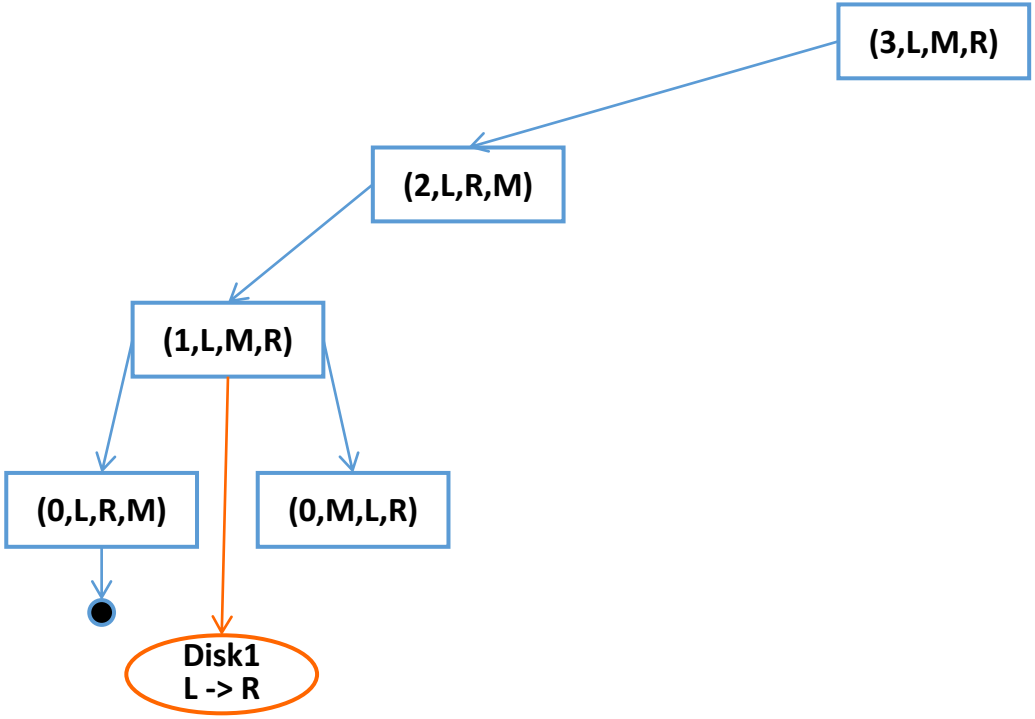

(3,L,M,R)
(2,L,R,M)

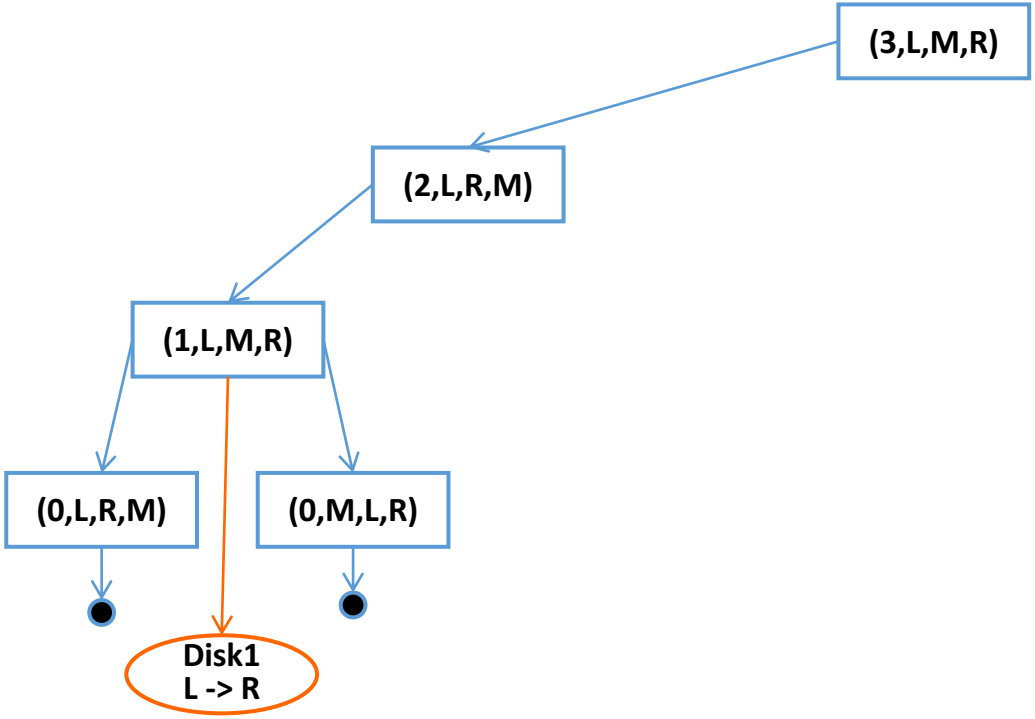

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)

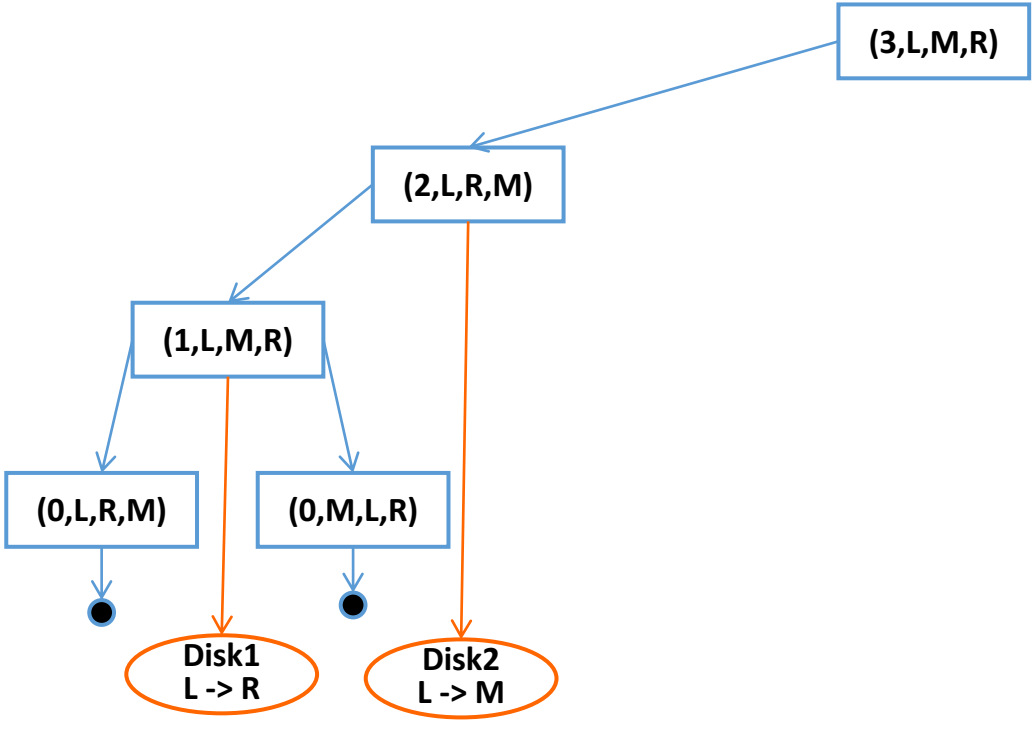

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)
(0,L,R,M)

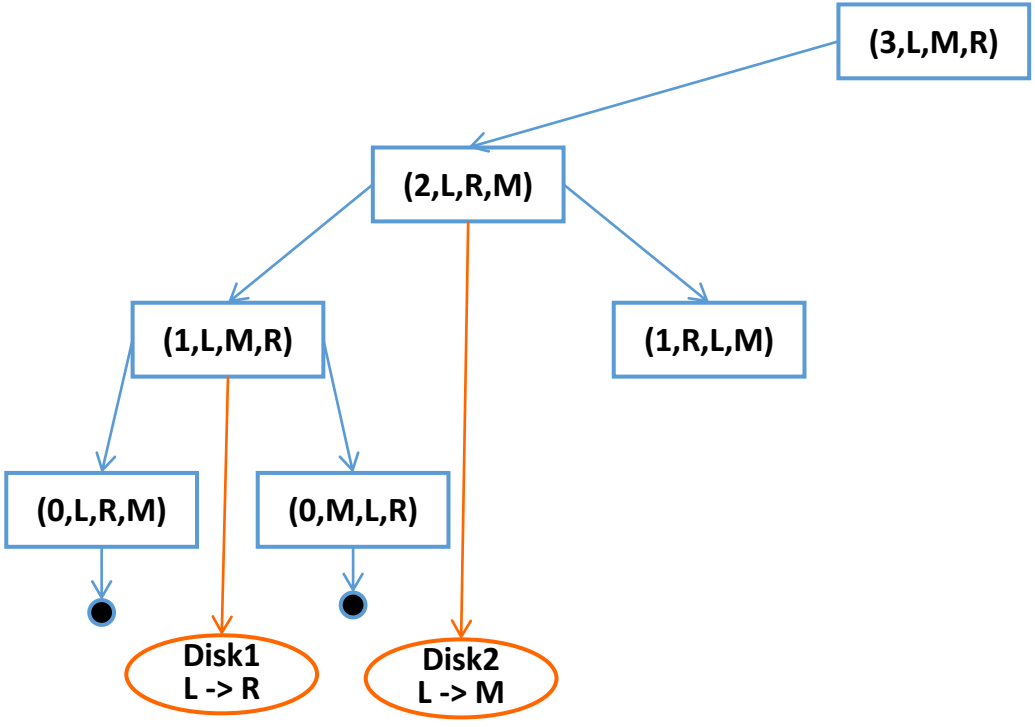

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)
(0,L,R,M)

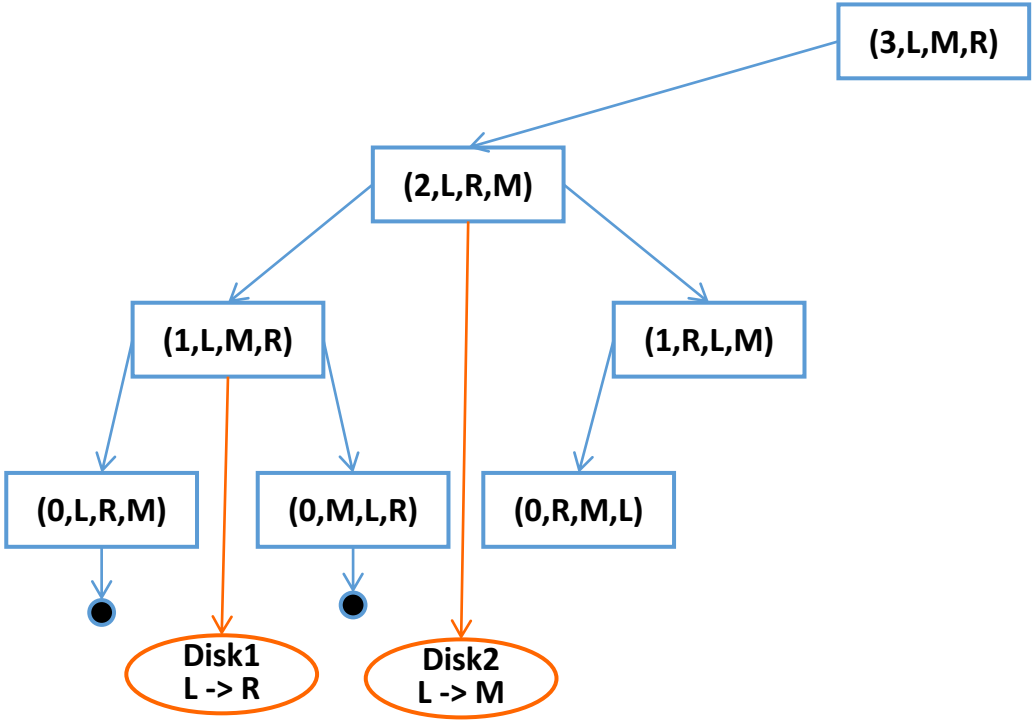

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)
(0,L,R,M)
Disk1
L -> R

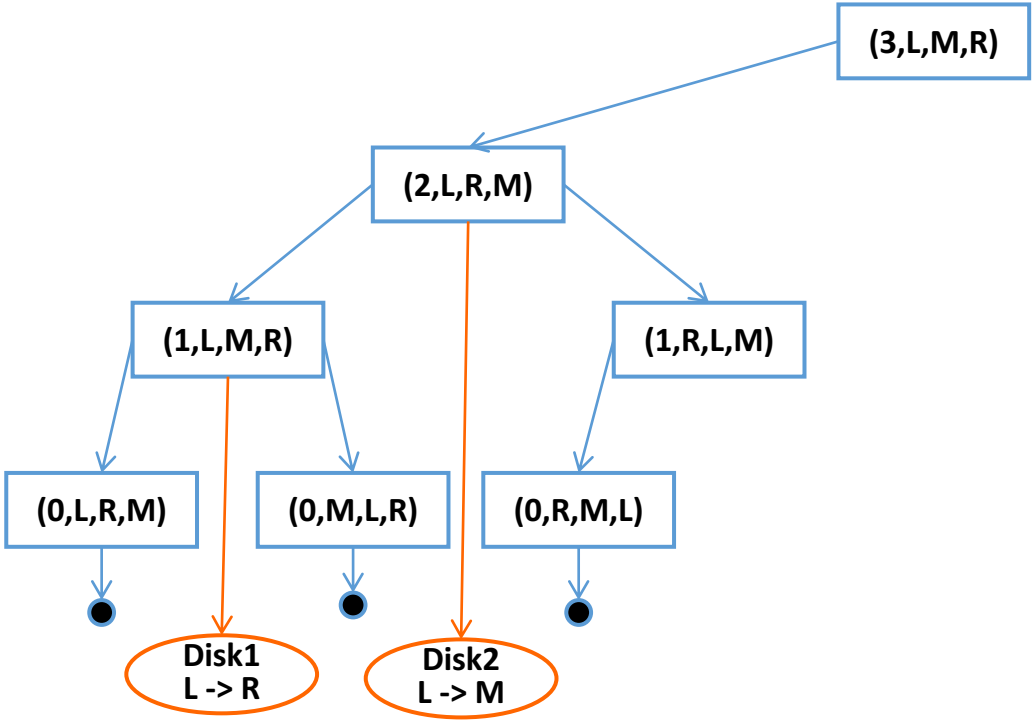

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)
(0,L,R,M)
(0,M,L,R)
Disk1
L -> R

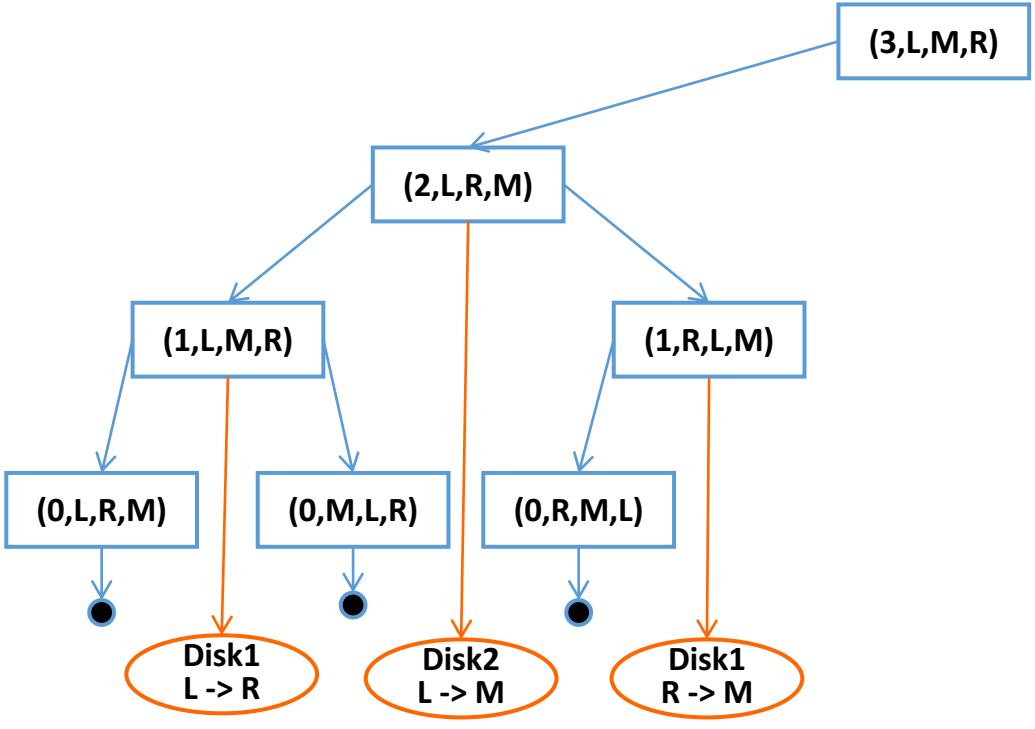

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)
(0,L,R,M)
(0,M,L,R)
Disk1
L -> R

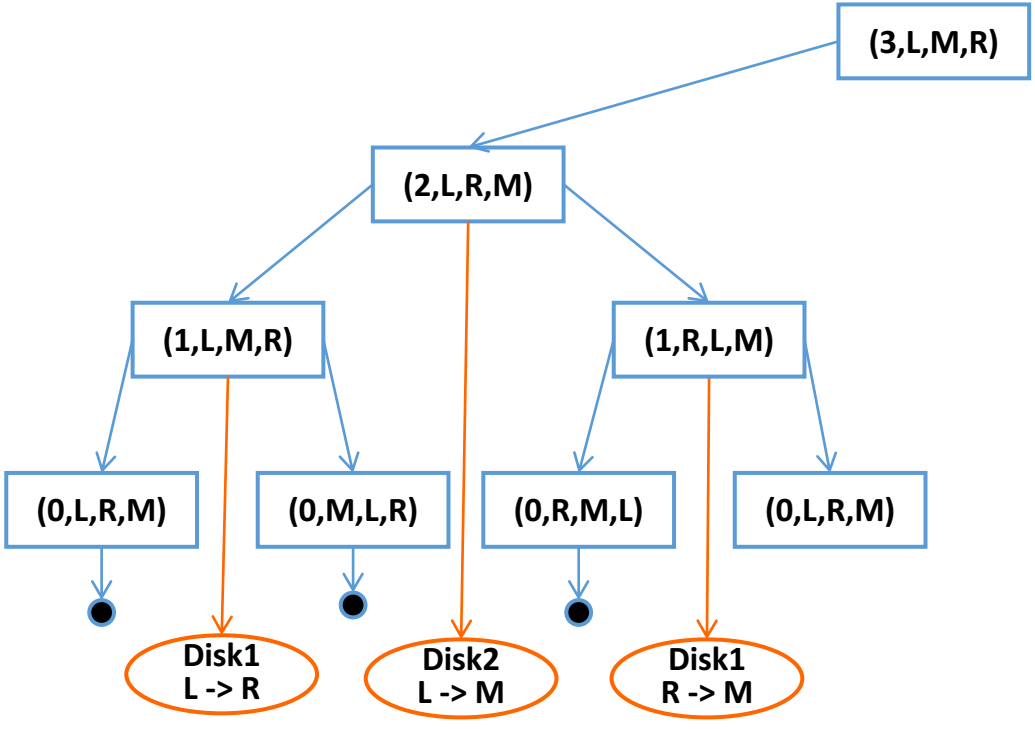

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)
(0,L,R,M)
(0,M,L,R)
Disk1
L -> R
Disk2
L -> M

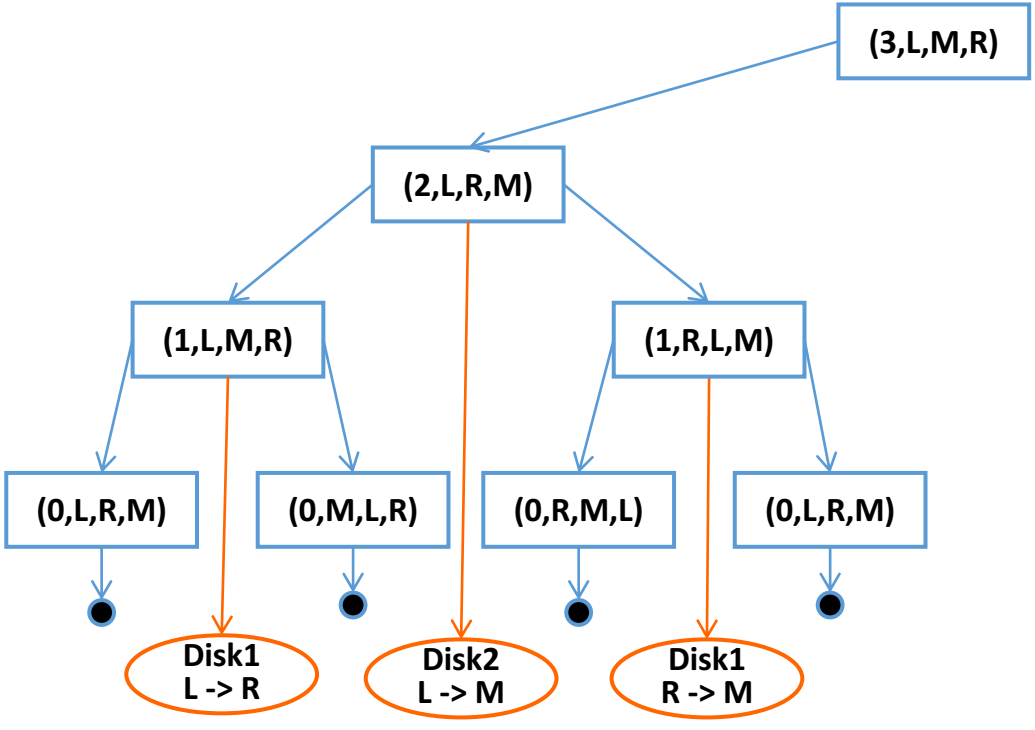

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)
(1,R,L,M)
(0,L,R,M)
(0,M,L,R)
Disk1
L -> R
Disk2
L -> M

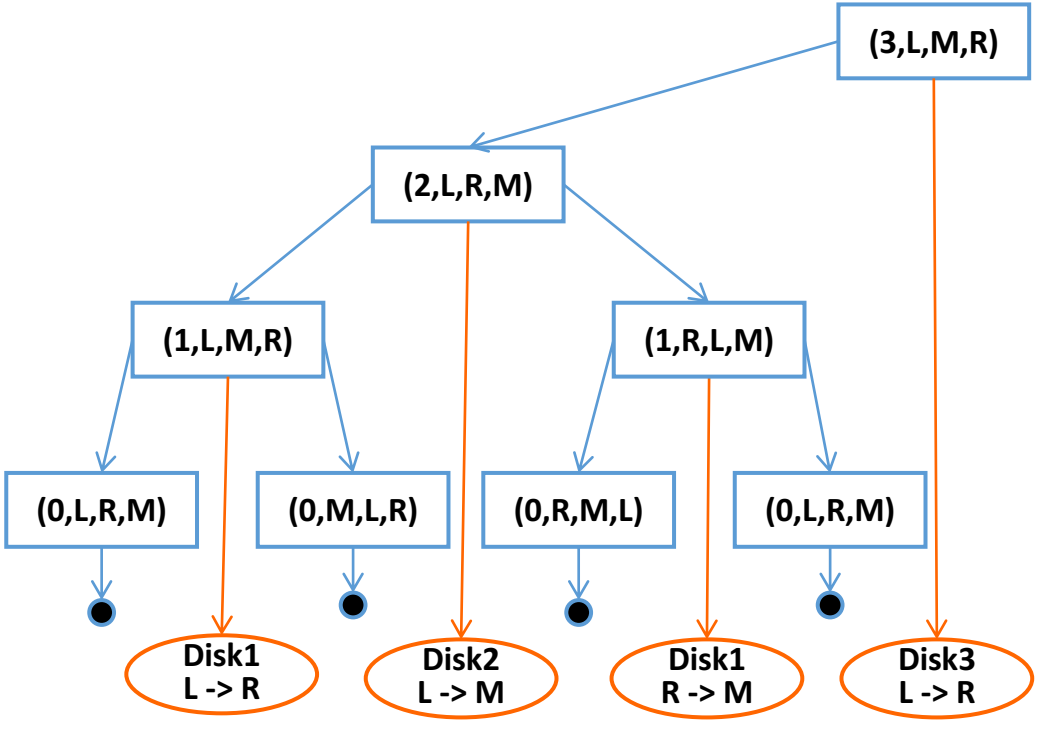

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)
(1,R,L,M)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
Disk1
L -> R
Disk2
L -> M

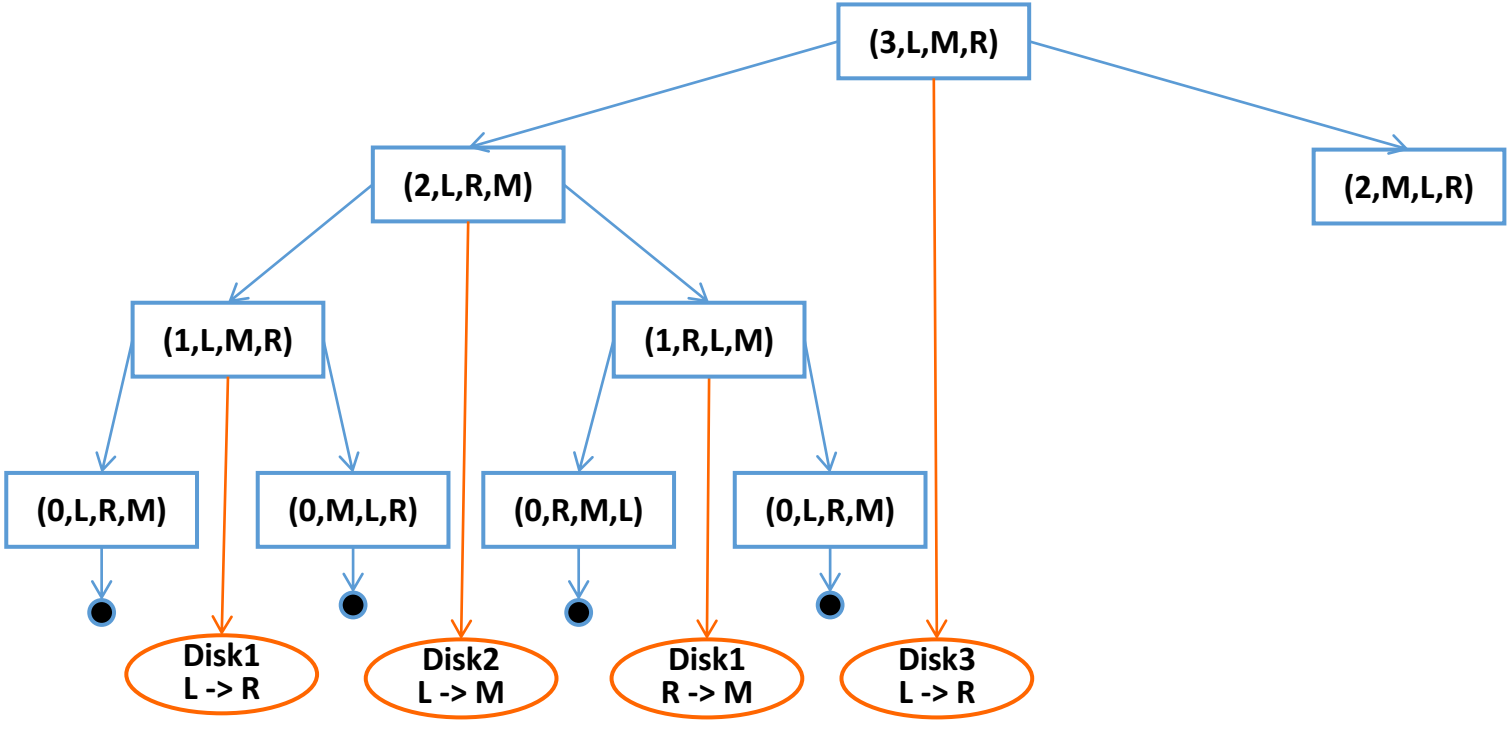

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)
(1,R,L,M)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
Disk1
L -> R
Disk2
L -> M

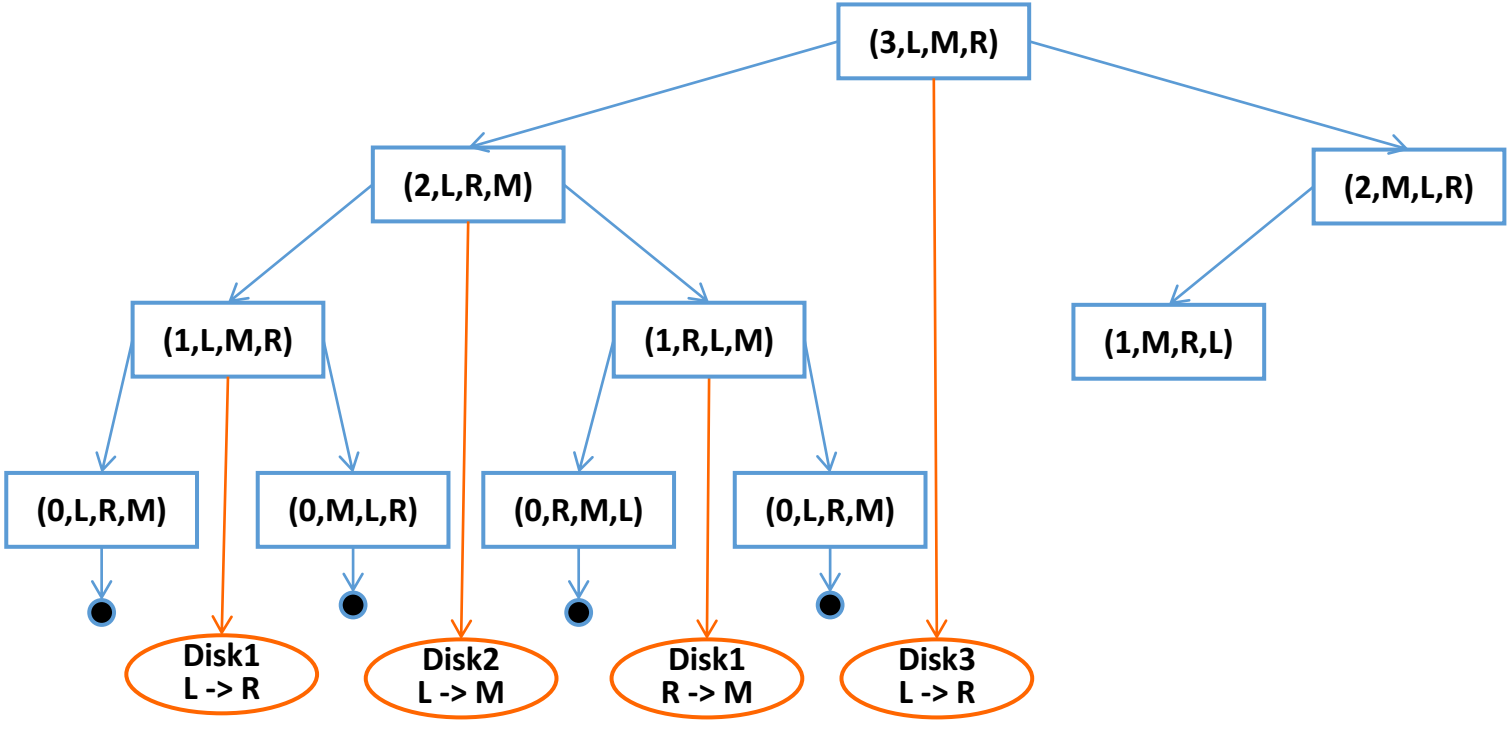

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)
(1,R,L,M)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M

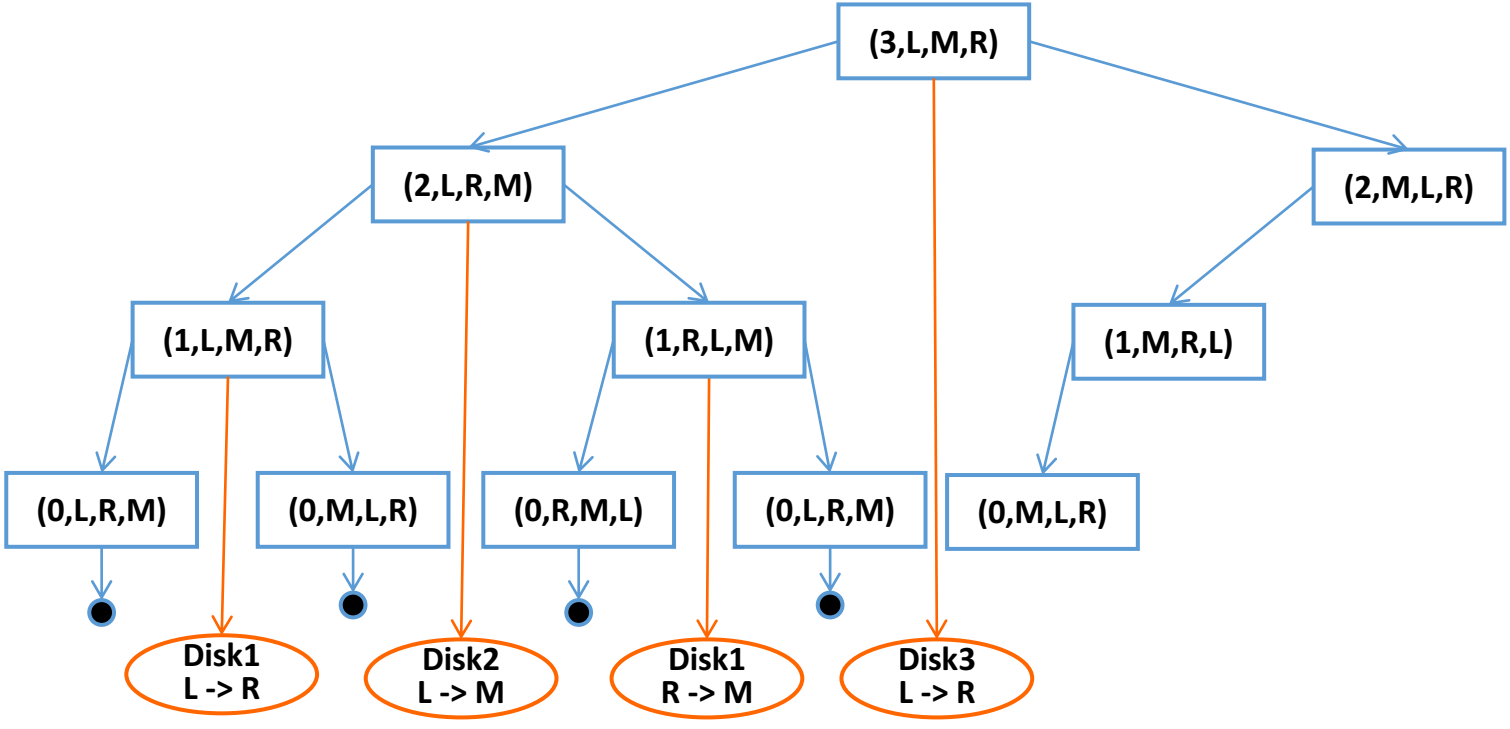

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)
(1,R,L,M)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M

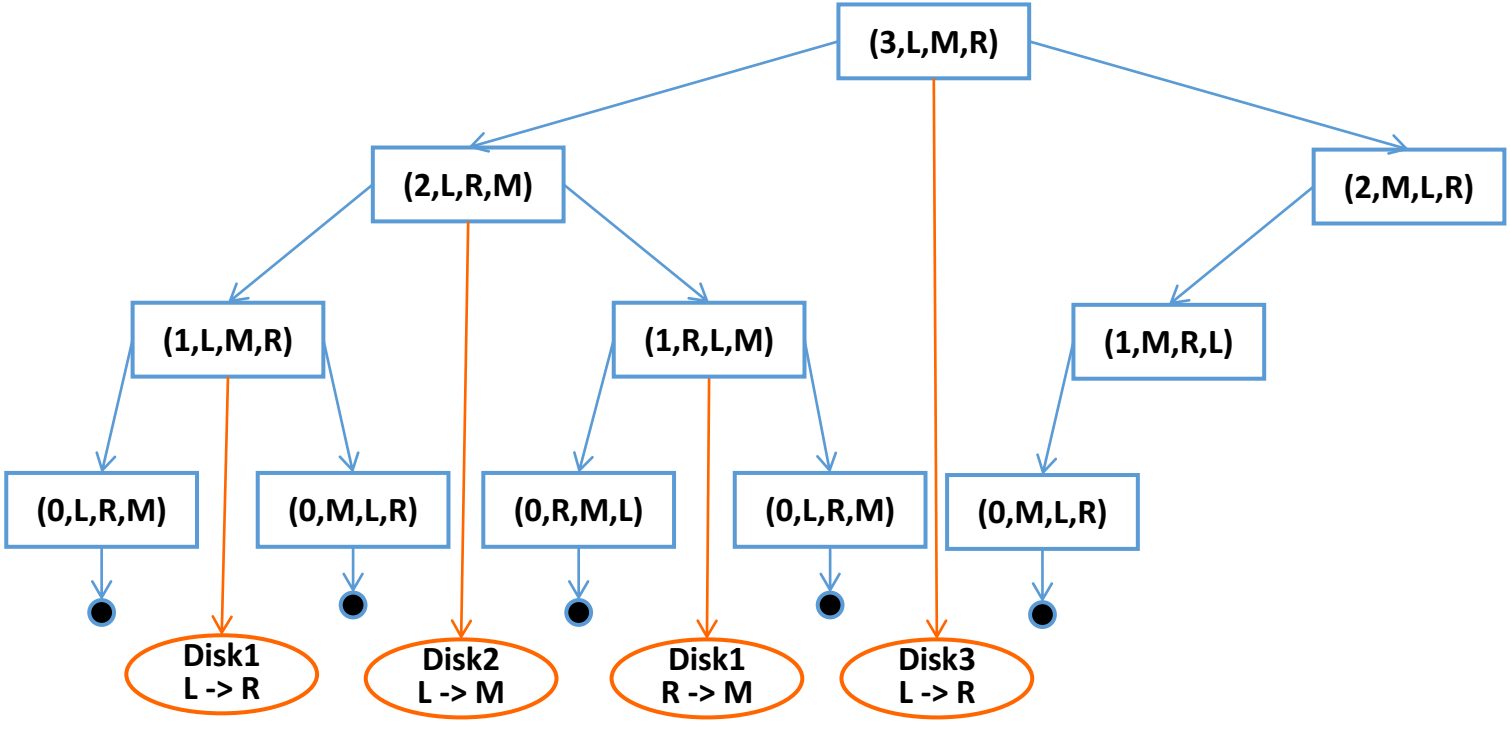

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)
(1,R,L,M)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M

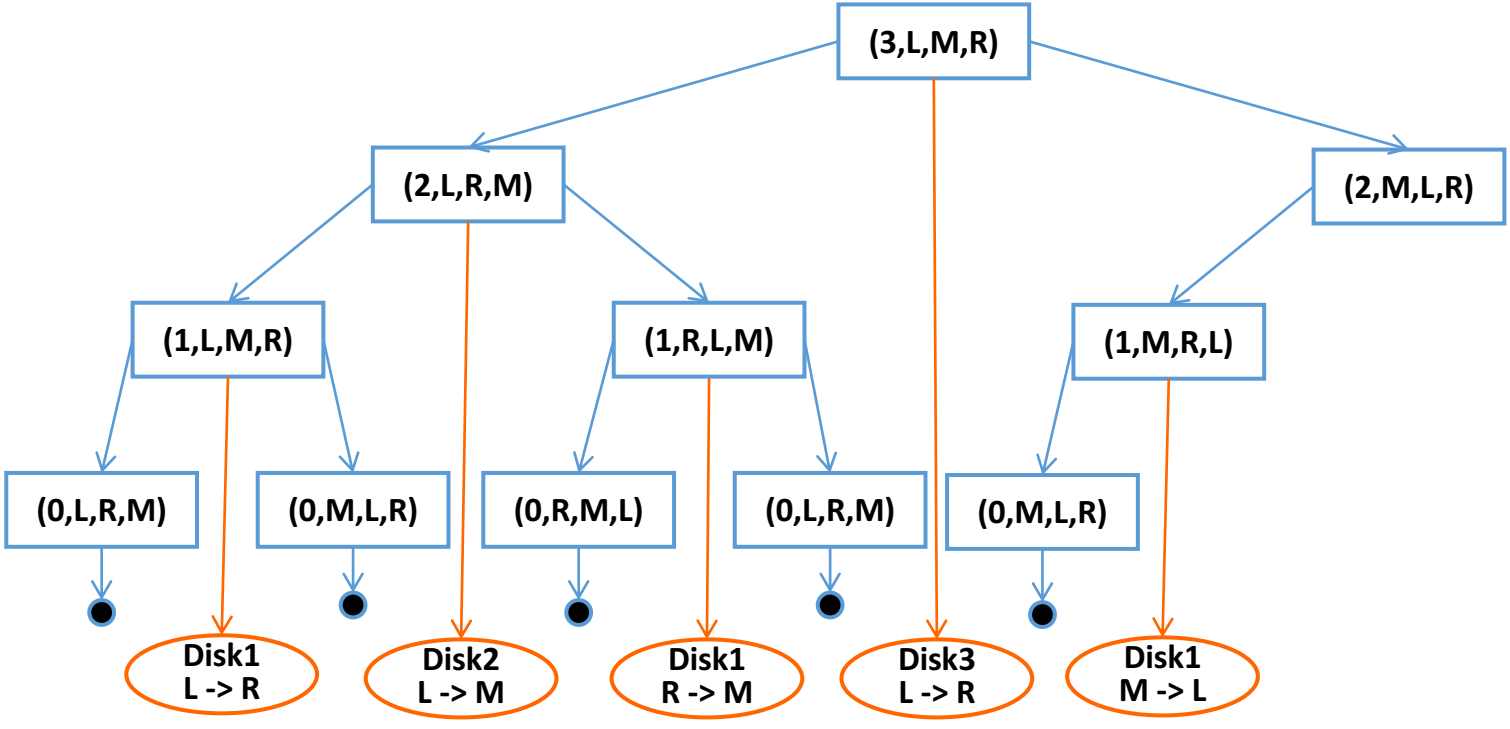

(3,L,M,R)
(2,L,R,M)
(1,L,M,R)
(1,R,L,M)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R

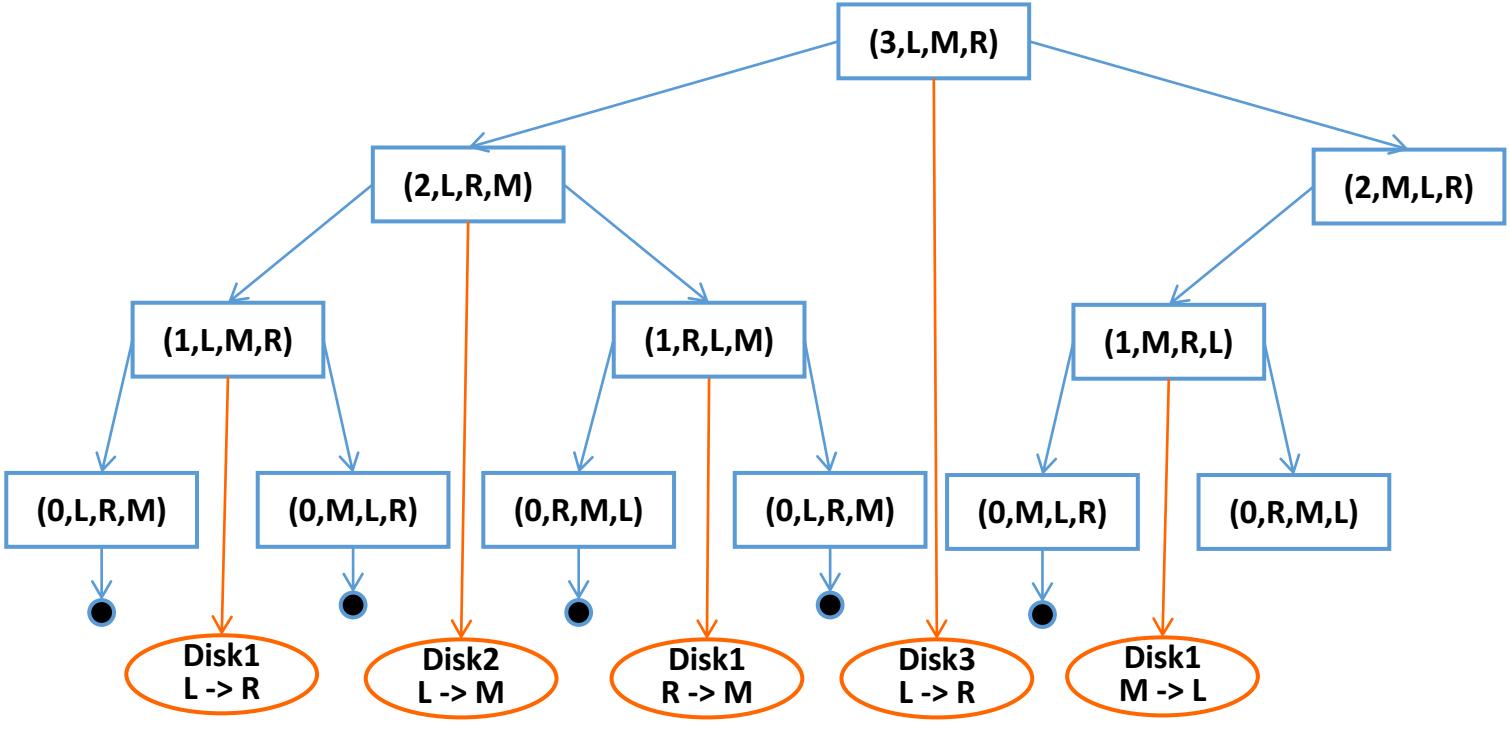

(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R

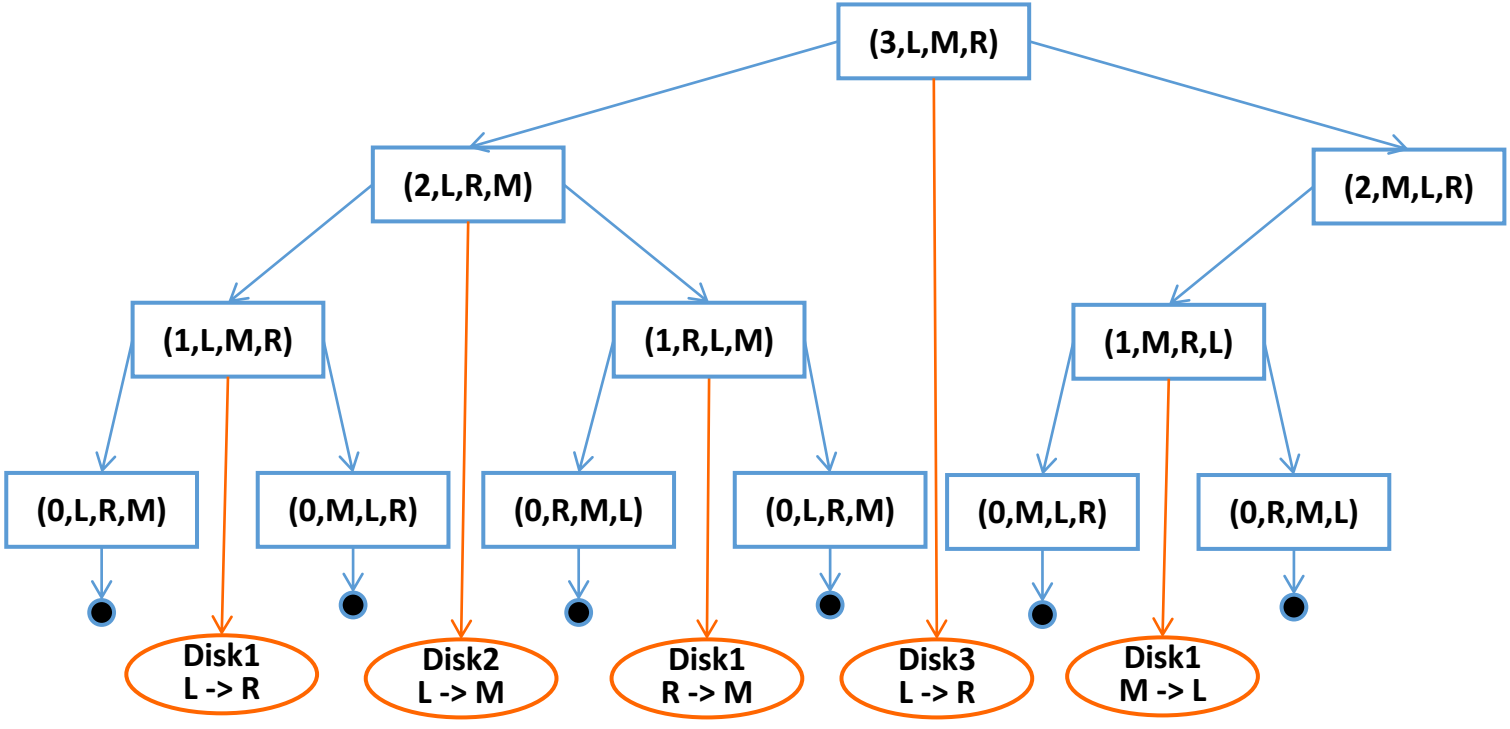

(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(1,M,R,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R

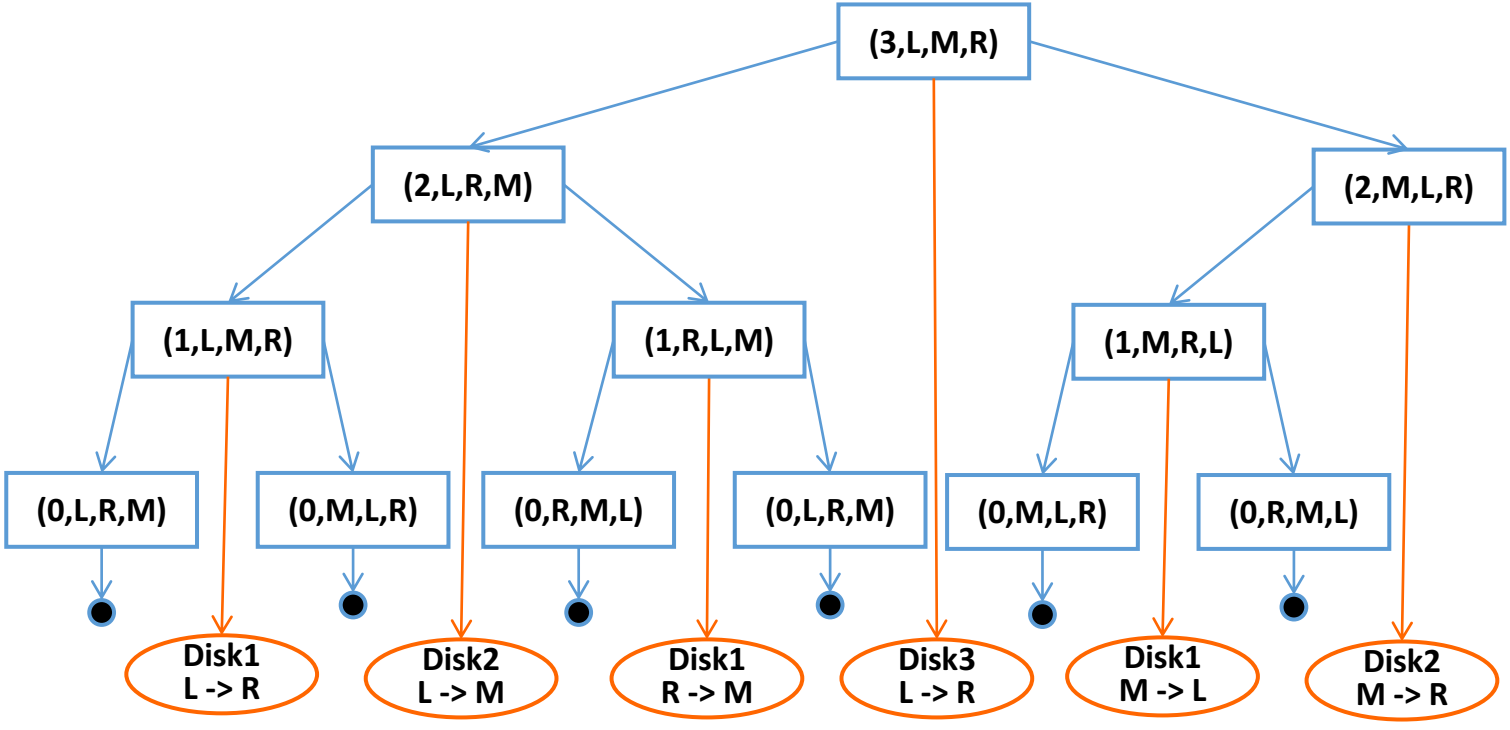

(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(1,M,R,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R

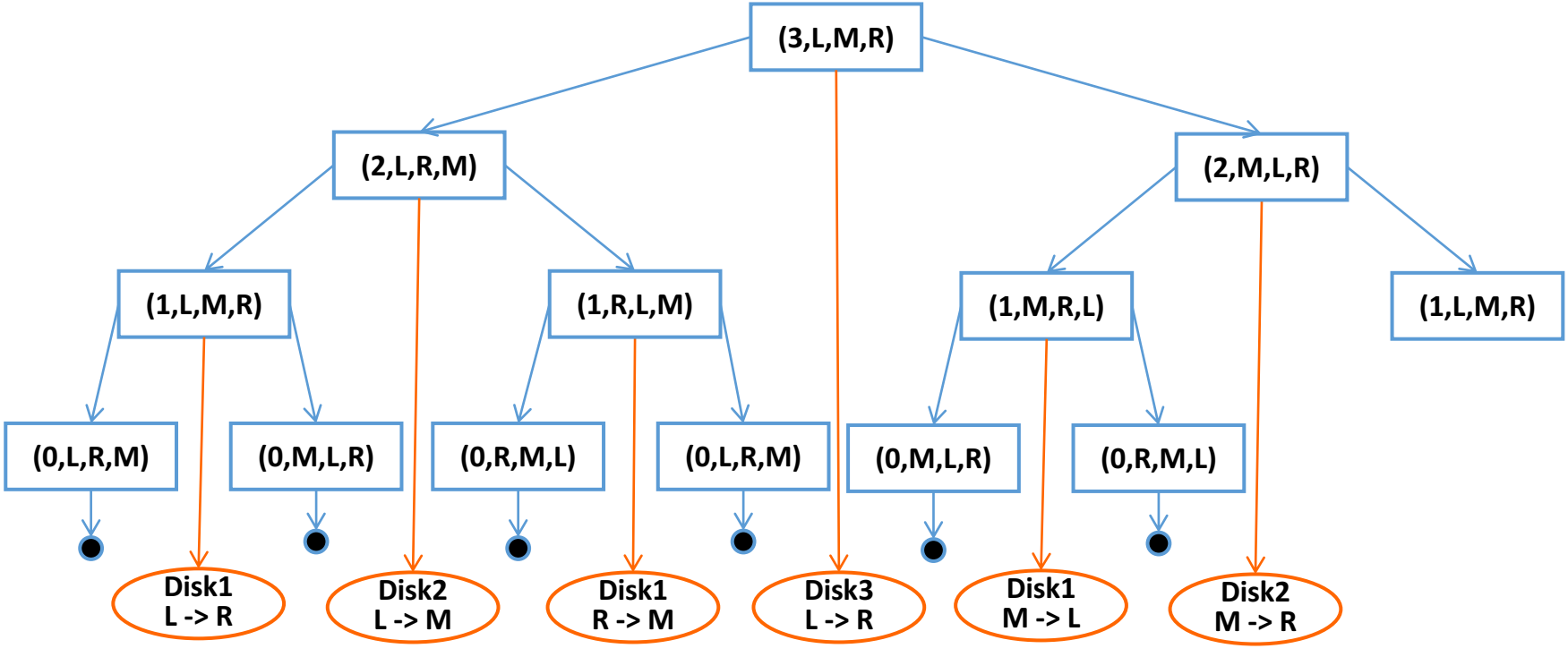

(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(1,M,R,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R

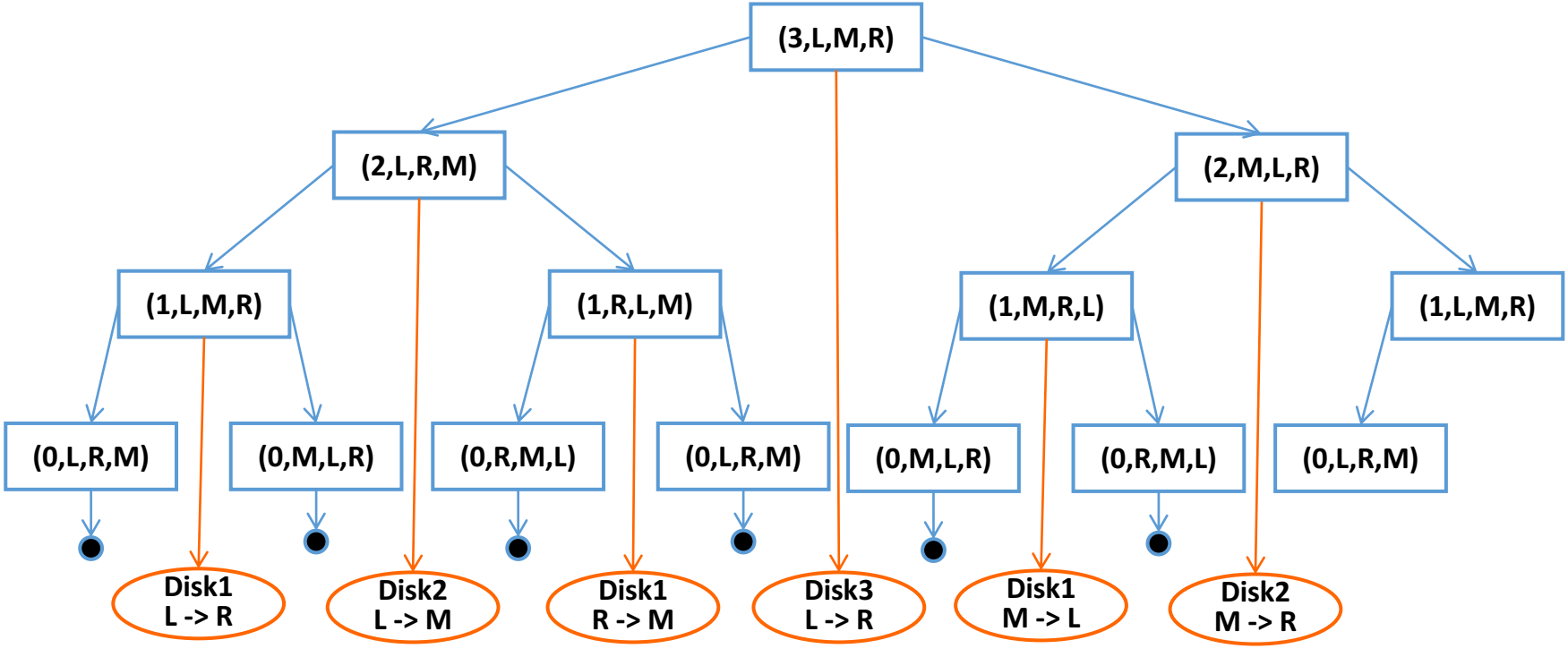

(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(1,M,R,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R
Disk1
M -> L

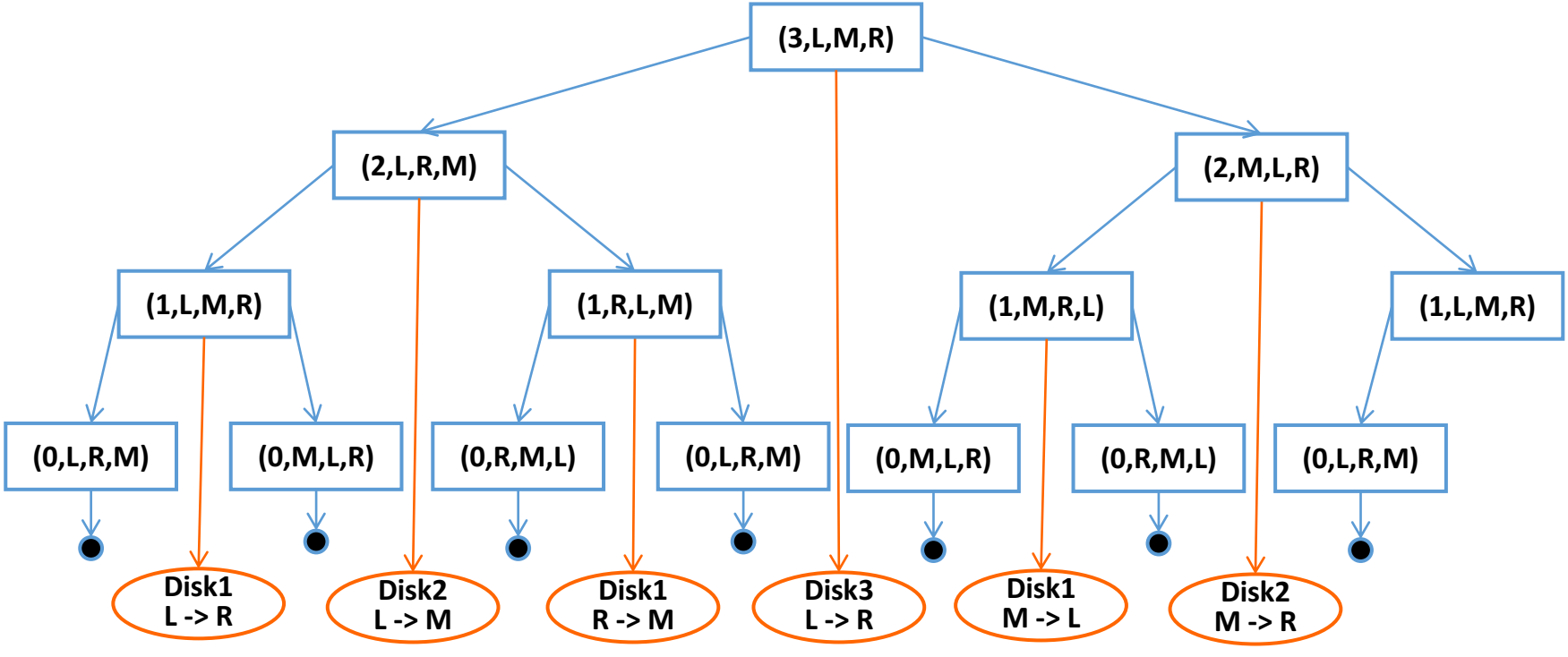

(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(1,M,R,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R
Disk1
M -> L

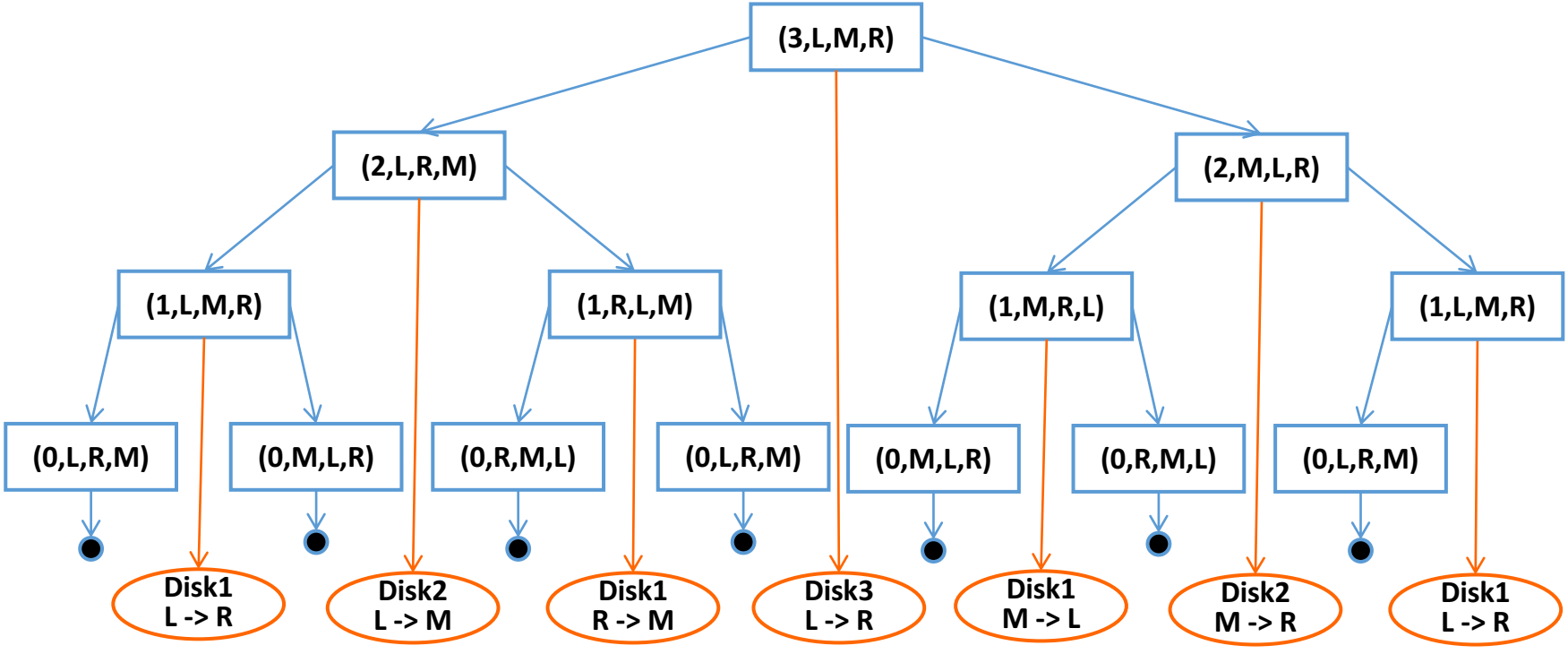

(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(1,M,R,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R
Disk1
M -> L

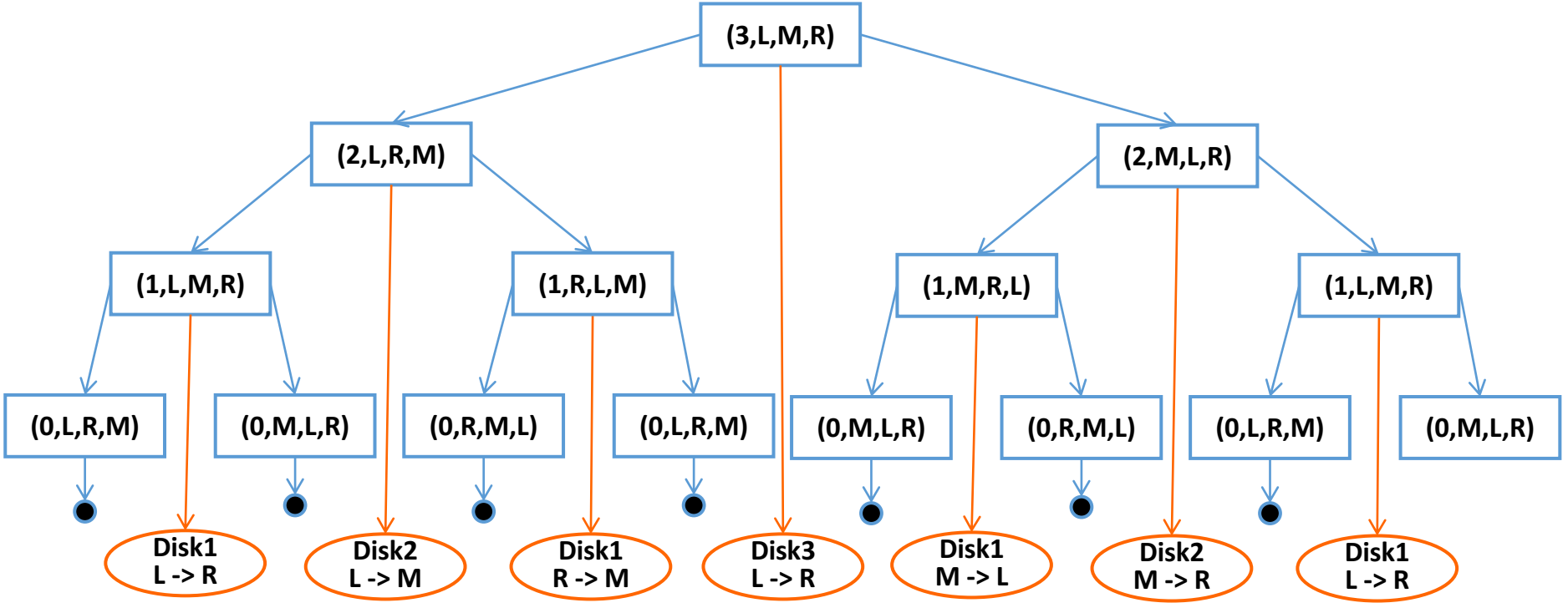

(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(1,M,R,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R
Disk1
M -> L
Disk2
M -> R

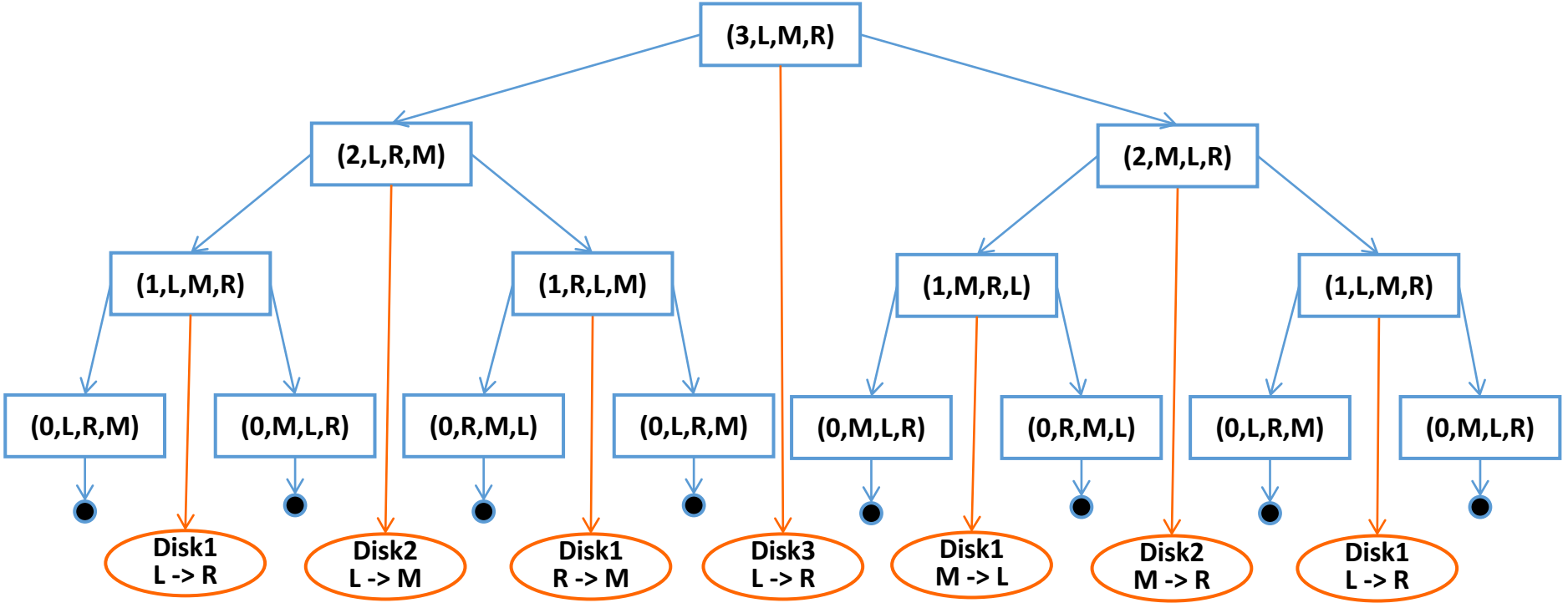

(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(1,M,R,L)
(1,L,M,R)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R
Disk1
M -> L
Disk2
M -> R


(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(1,M,R,L)
(1,L,M,R)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R
Disk1
M -> L
Disk2
M -> R


(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(1,M,R,L)
(1,L,M,R)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R
Disk1
M -> L
Disk2
M -> R


(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(1,M,R,L)
(1,L,M,R)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R
Disk1
M -> L
Disk2
M -> R
Disk1
L -> R


(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(1,M,R,L)
(1,L,M,R)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R
Disk1
M -> L
Disk2
M -> R
Disk1
L -> R


(3,L,M,R)
(2,L,R,M)
(2,M,L,R)
(1,L,M,R)
(1,R,L,M)
(1,M,R,L)
(1,L,M,R)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
(0,R,M,L)
(0,L,R,M)
(0,M,L,R)
Disk1
L -> R
Disk2
L -> M
Disk1
R -> M
Disk3
L -> R
Disk1
M -> L
Disk2
M -> R
Disk1
L -> R

Back to the sorting problem:

- **Divide the elements into half**
- **Sort each of the half**
  - If the half is still big, divide them into two again…
  - If small just use simple sorting
- **Merge the two sorted halves**
- **How should it be implemented using divide & conquer?**

Divide and conquer algorithm for sorting

- **Divide**
  - Divide A[1....n] to A_1 = A[1...m] and A_2 = A[m+1...n]
- **Conquer**
  - Recursively sort A_1 and A_2
- **Combine**
  - Combine two sorted arrays into a single sorted array

Visual illustration of Mergesort

https://www.toptal.com/developers/sorting-algorithms/merge-sort

mergesort.ipynb

Time complexity of mergesort

$$
\begin{aligned}
T(2^k) &= 2T(2^{k-1}) + c2^k \\
&= 4T(2^{k-2}) + 2c2^k \\
&= 8T(2^{k-3}) + 3c2^k \\
&= \dots \\
&= 2^k T(1) + kc2^k
\end{aligned}
$$

Time complexity of mergesort

$$2^k \leq n < 2^{k+1}$$

$$2^k T(1) + kc2^k \leq T(n) \leq 2^{k+1}T(1) + (k+1)c2^{k+1}$$

$$\frac{n}{2}T(1) + c\frac{n}{2}\log\left(\frac{n}{2}\right) \leq T(n) \leq 2nT(1) + 2cn\log(2n)$$

$$\Theta(n\log n) \leq T(n) \leq \Theta(n\log n)$$

Summary

- Recursion
- Divide and conquer algorithms
- Tower of Hanoi
- Mergesort
- Time complexity of mergesort

Reading List

- CLRS I.2.3 (pp. 29-42)