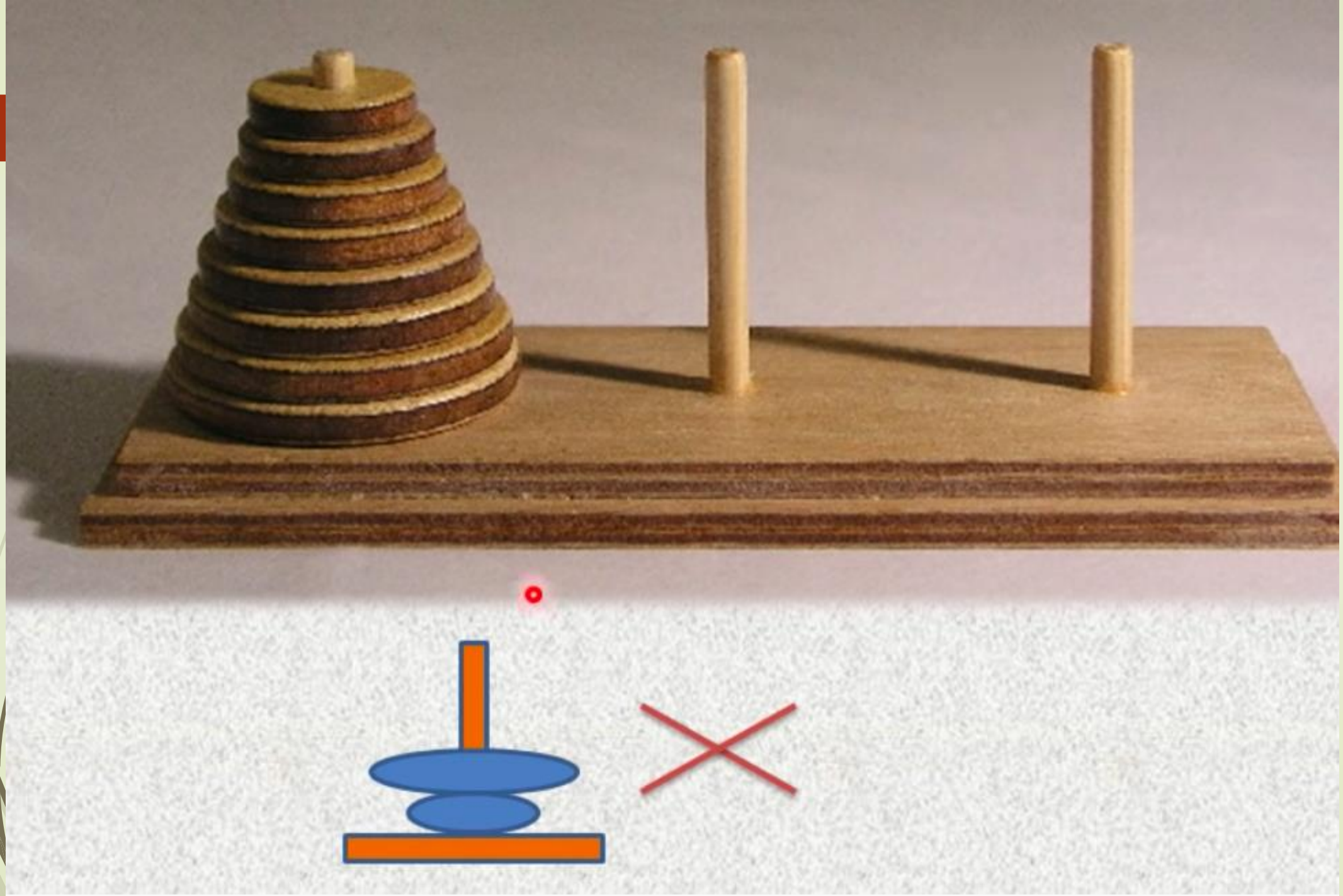
Data Structure and Algorithms

Session-10

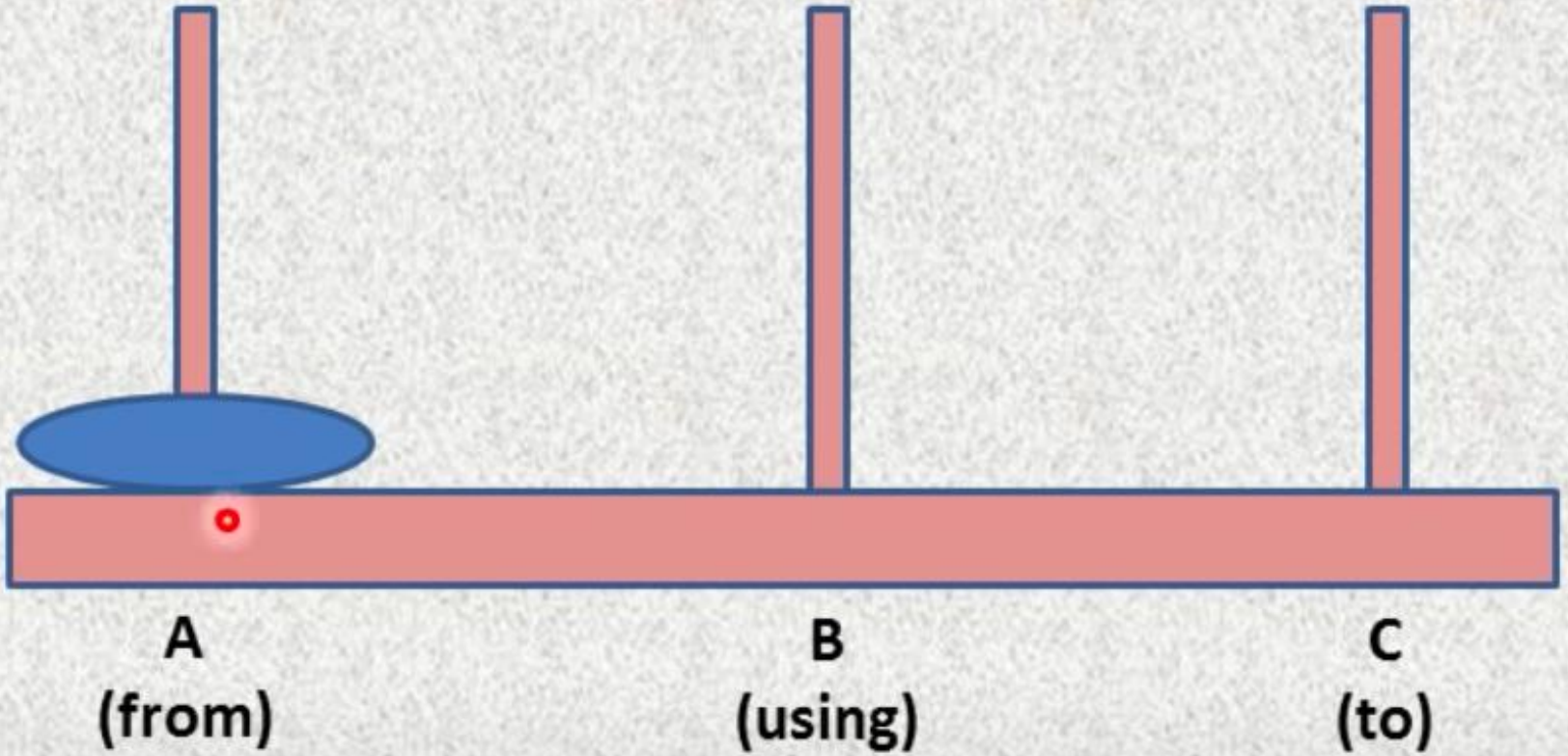
Dr. Subhra Rani Patra
SCOPE, VIT Chennai

Tower's of Hanoi

- Tower Of Hanoi is a Puzzle involving the usage of Stacks.
- Shifting of discs from source tower to target tower.
- Rules for solving puzzle:
  1. One disc can be shifted at once.
  2. Only top disc can be shifted.
  3. Large disc cannot be placed on smaller disc.
  4. Only 3 stacks can be used for solving the puzzle.

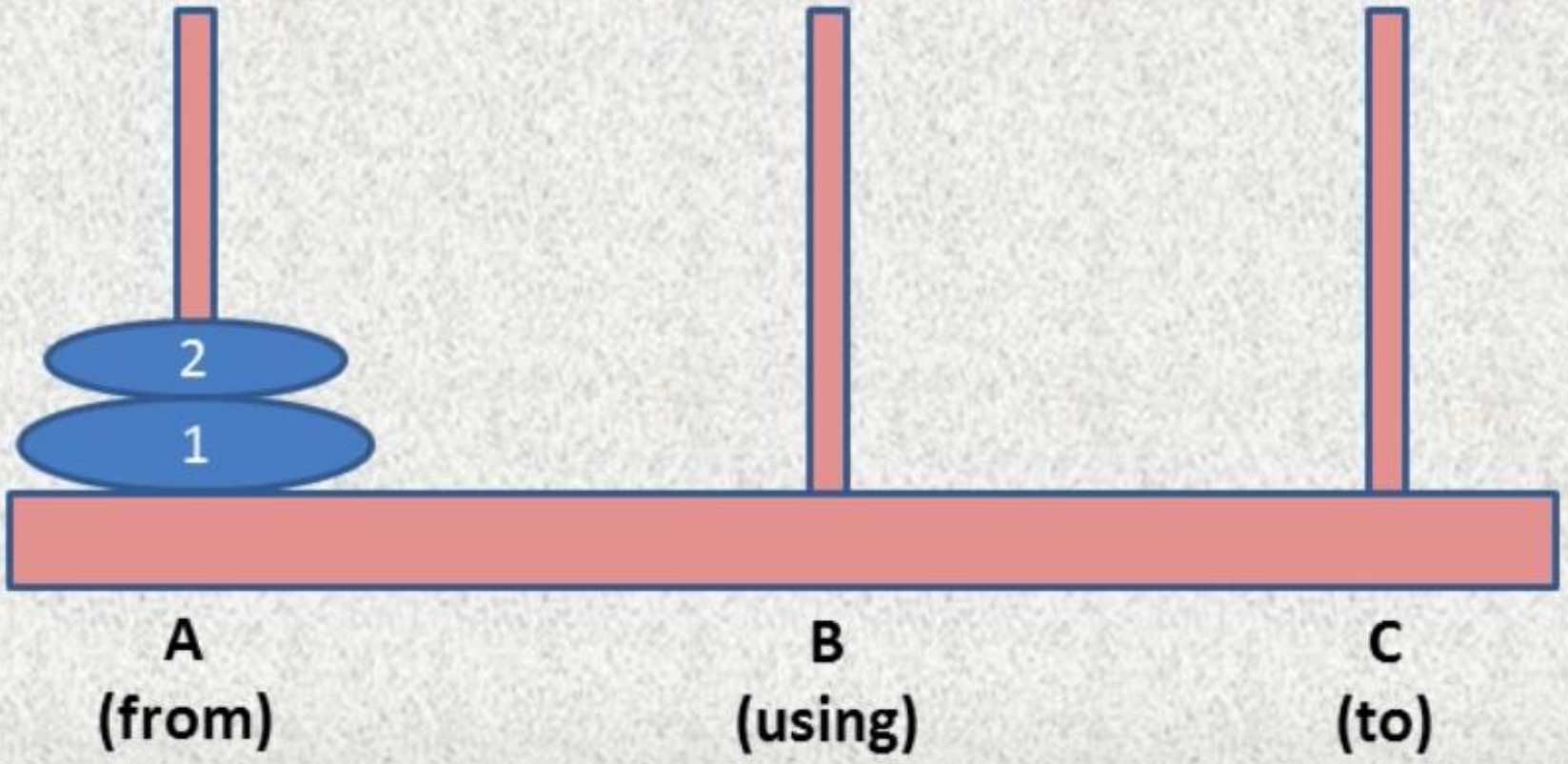


Solution for Single Disc



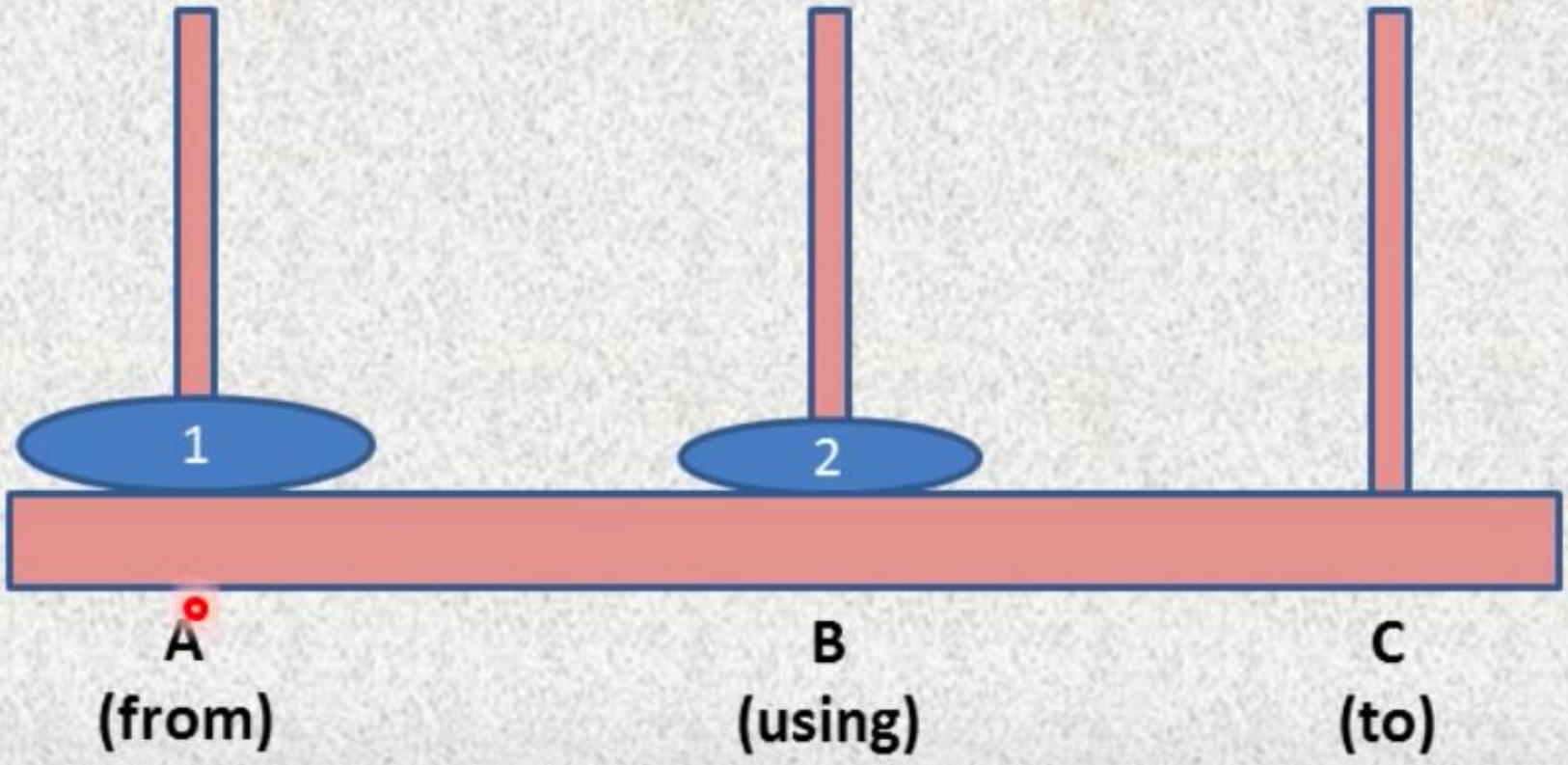
- **Move a disc from A to C**

Solution for 2 Disc



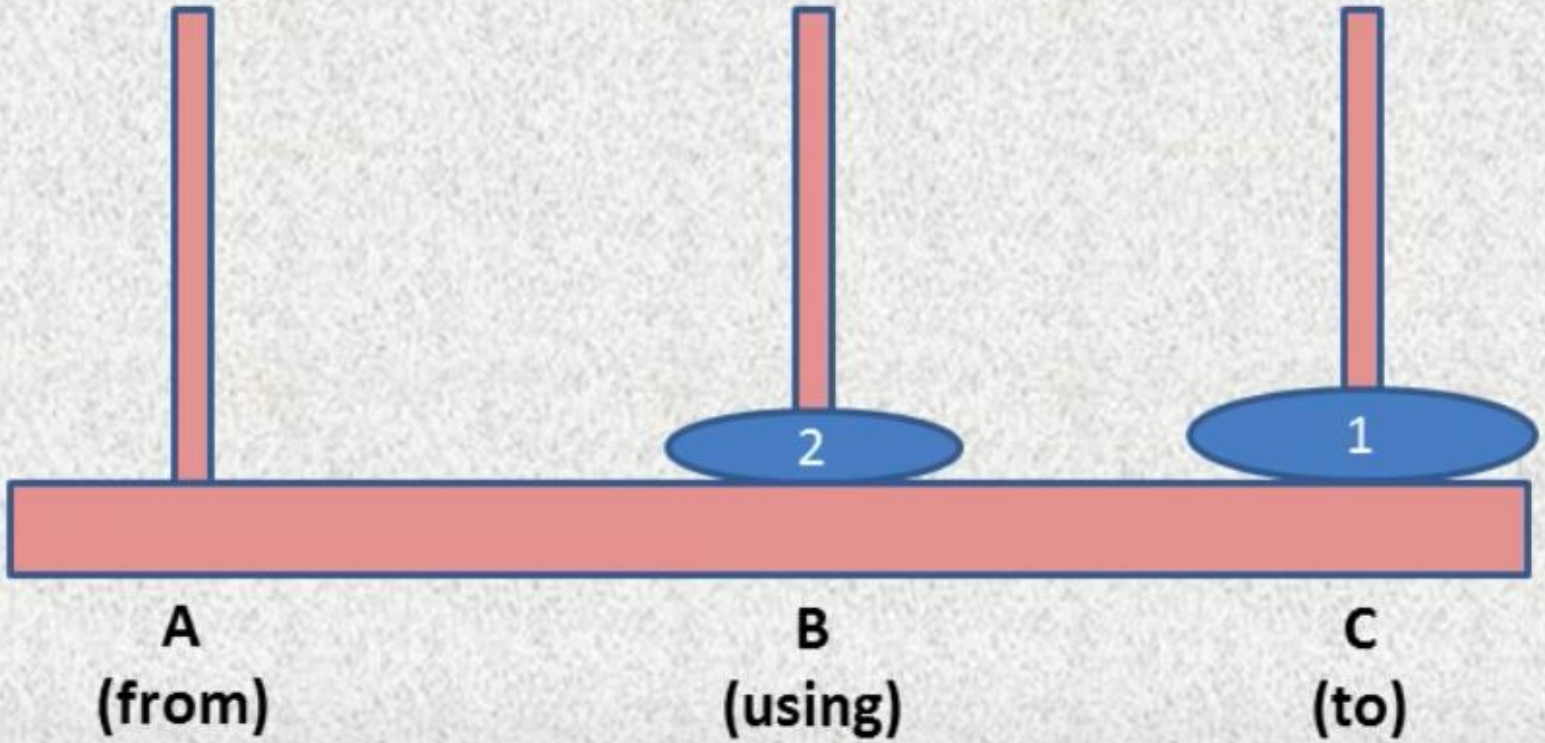
- Move a disc from A to B using C

Solution for 2 Disc



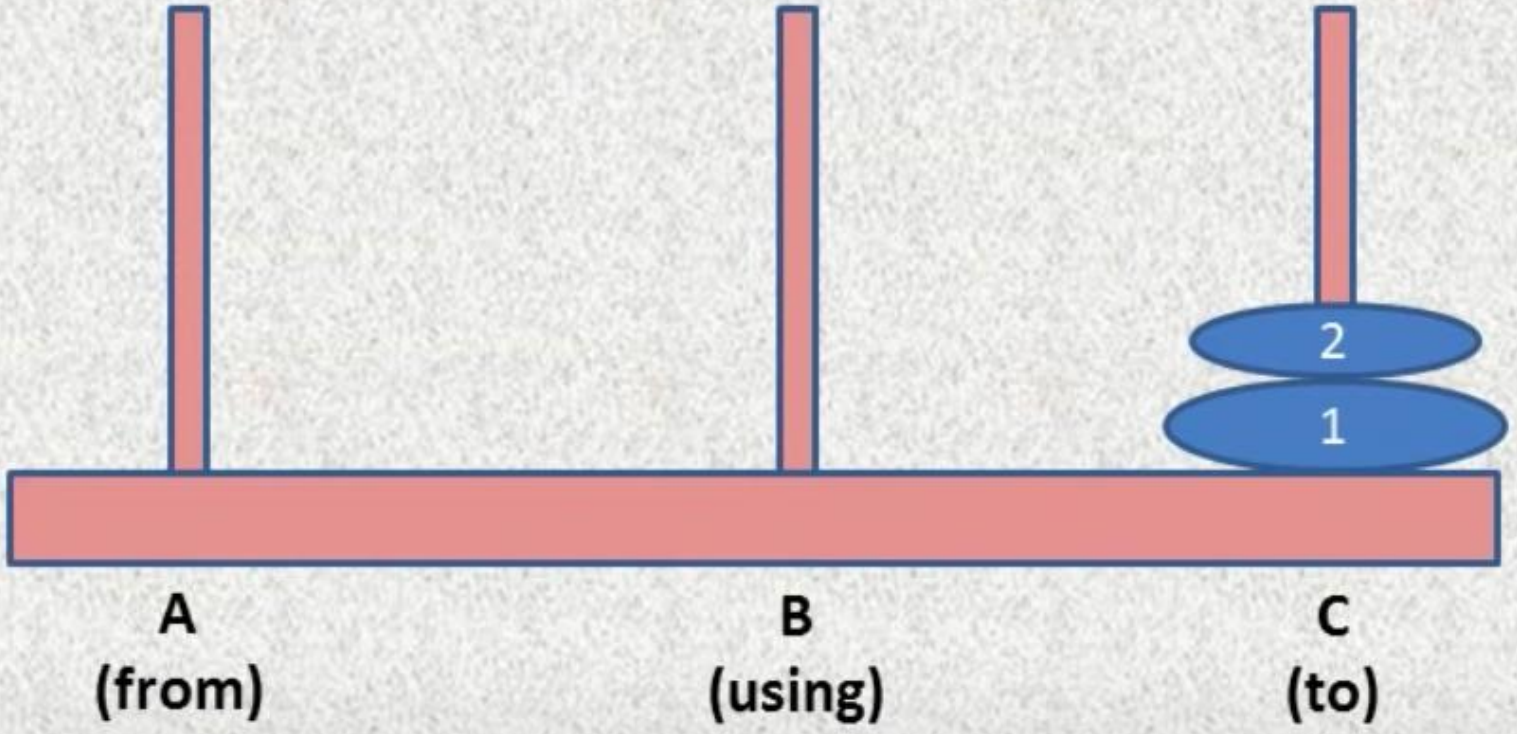
- **Move a Disc from A to B using C**
- **Move a Disc from A to C**

Solution for 2 Disc



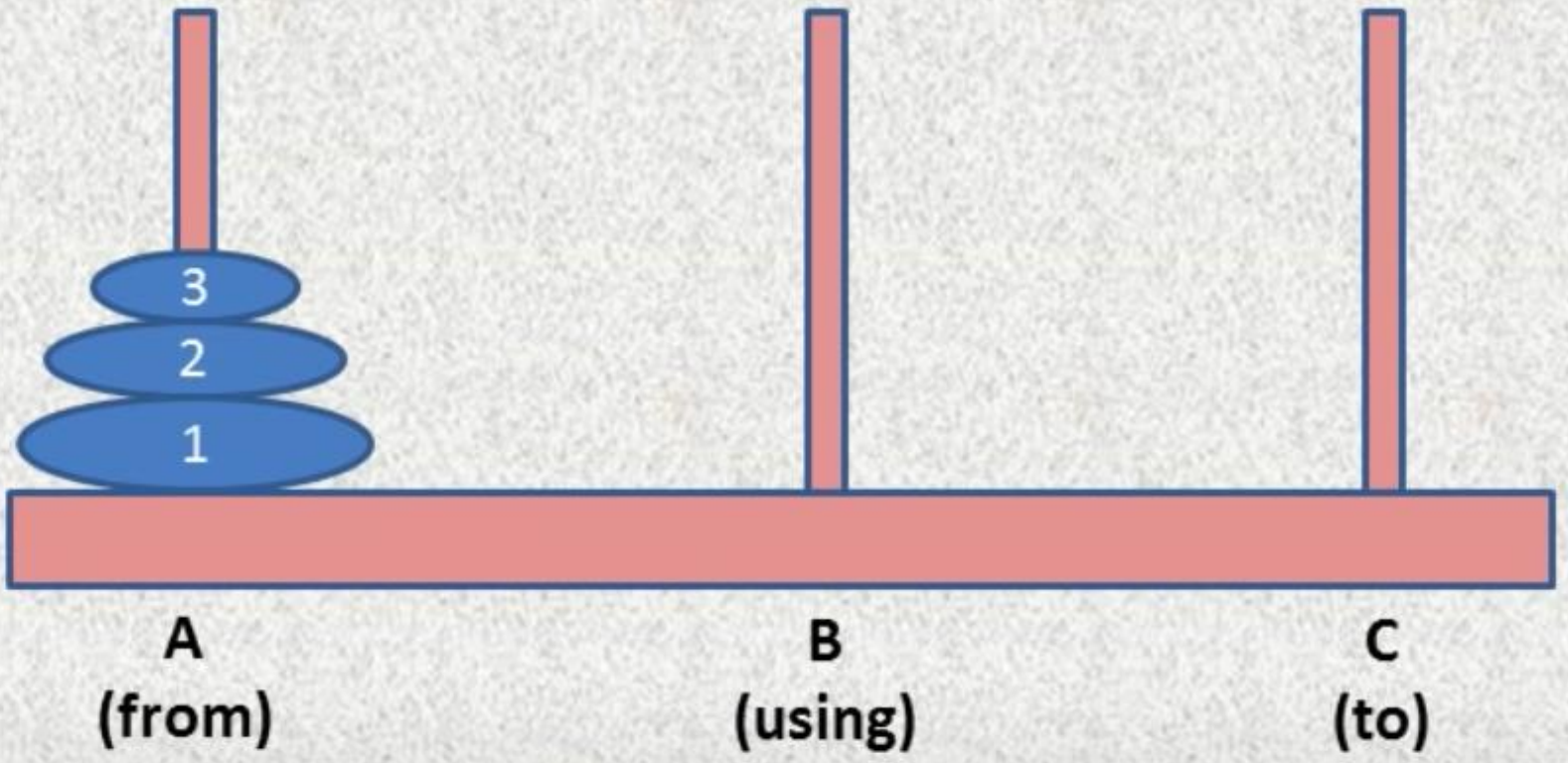
- Move a Disc from A to B using C
- Move a Disc from A to C
- Move a Disc from B to C using A

Solution for 2 Disc



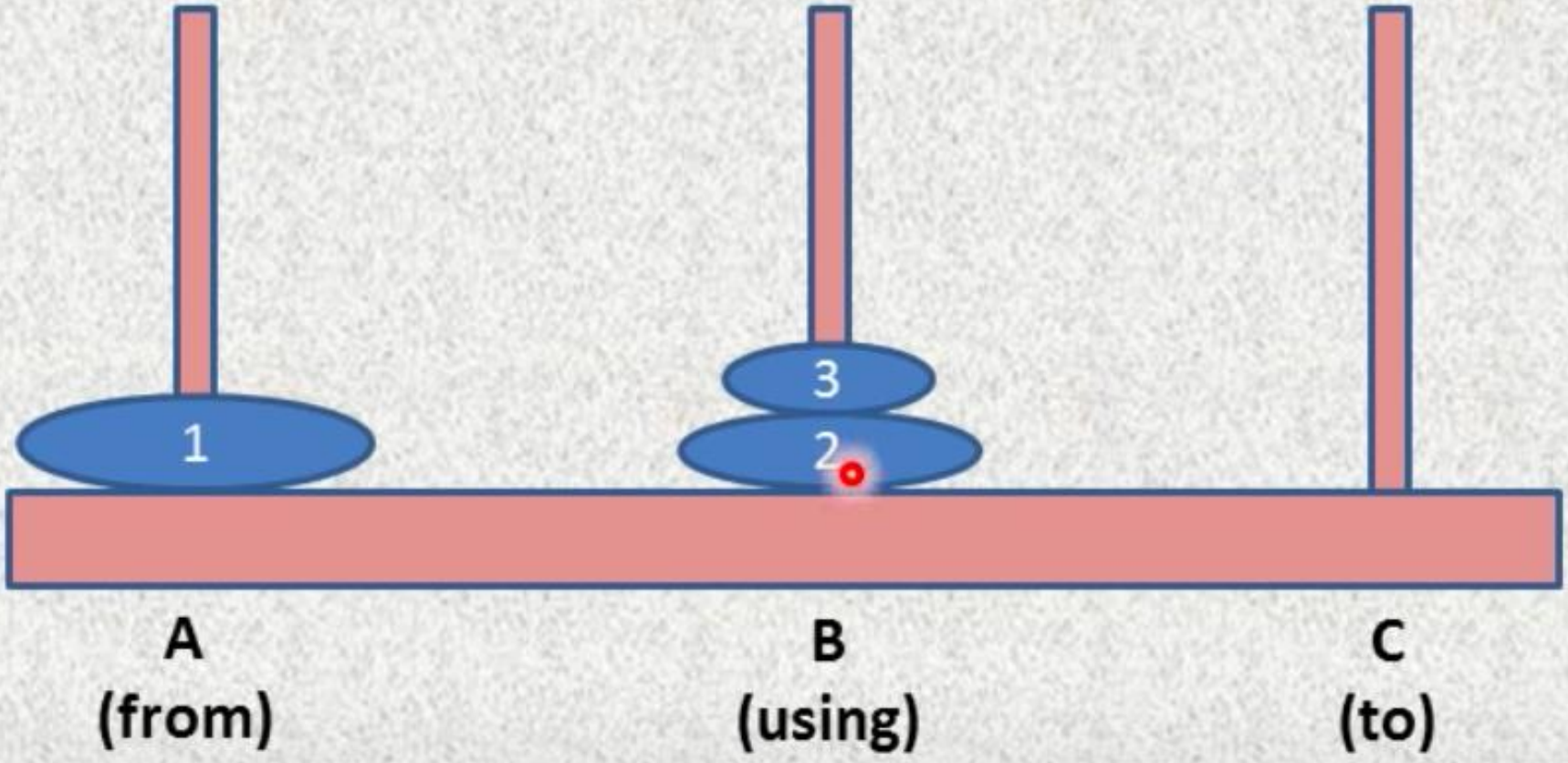
- Move a Disc from A to B using C
- Move a Disc from A to C
- Move a Disc from B to C using A

Solution for 3 Disc



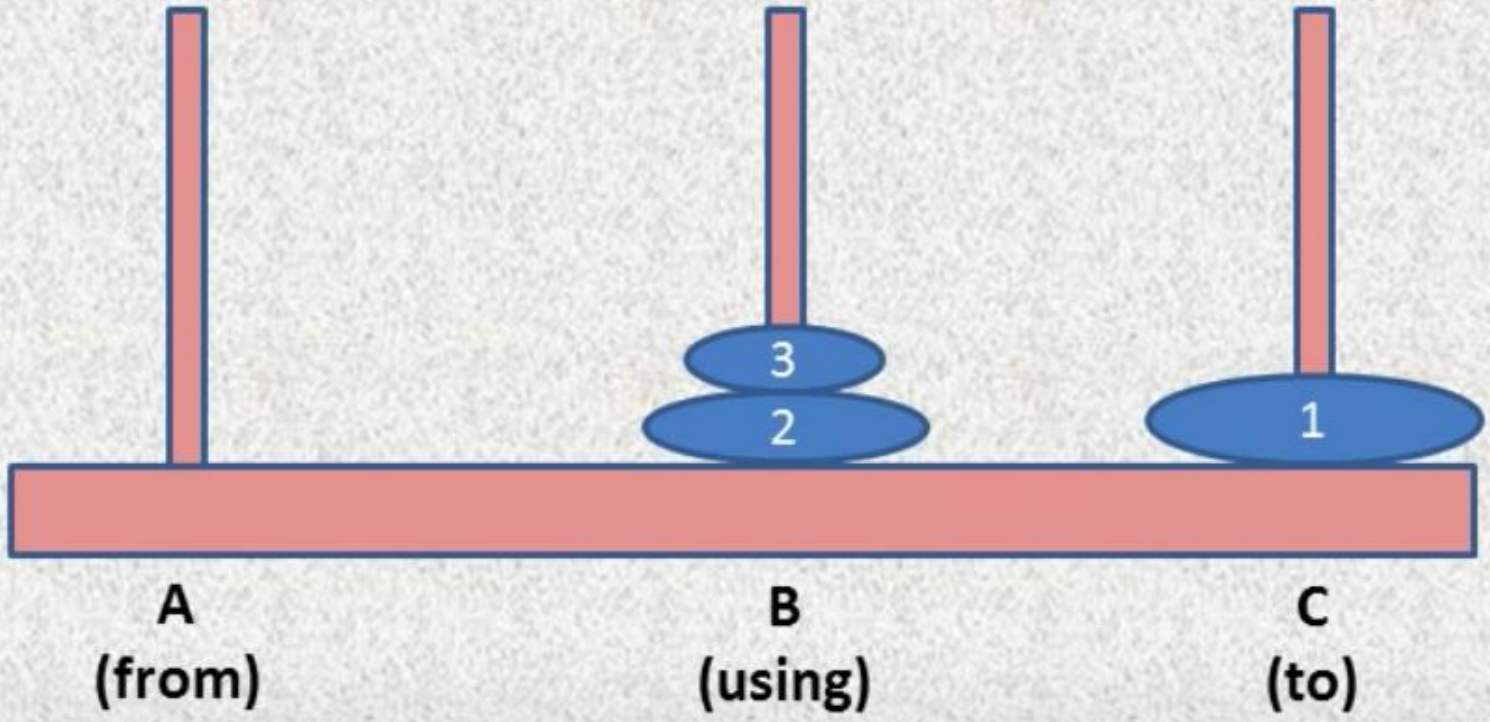
- Move 2 Discs from A to B using C

Solution for 3 Disc



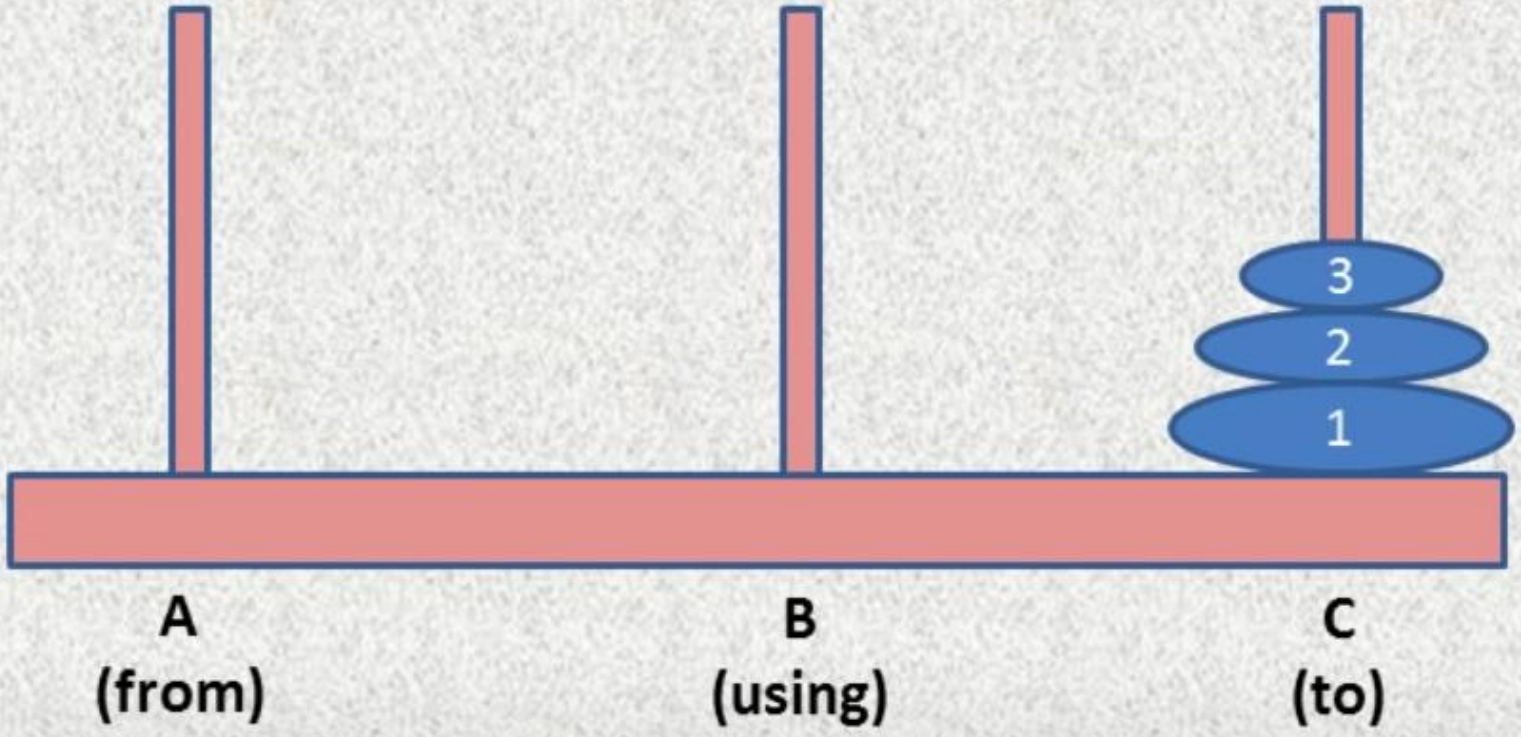
- **Move 2 Discs from A to B using C**

Solution for 3 Disc



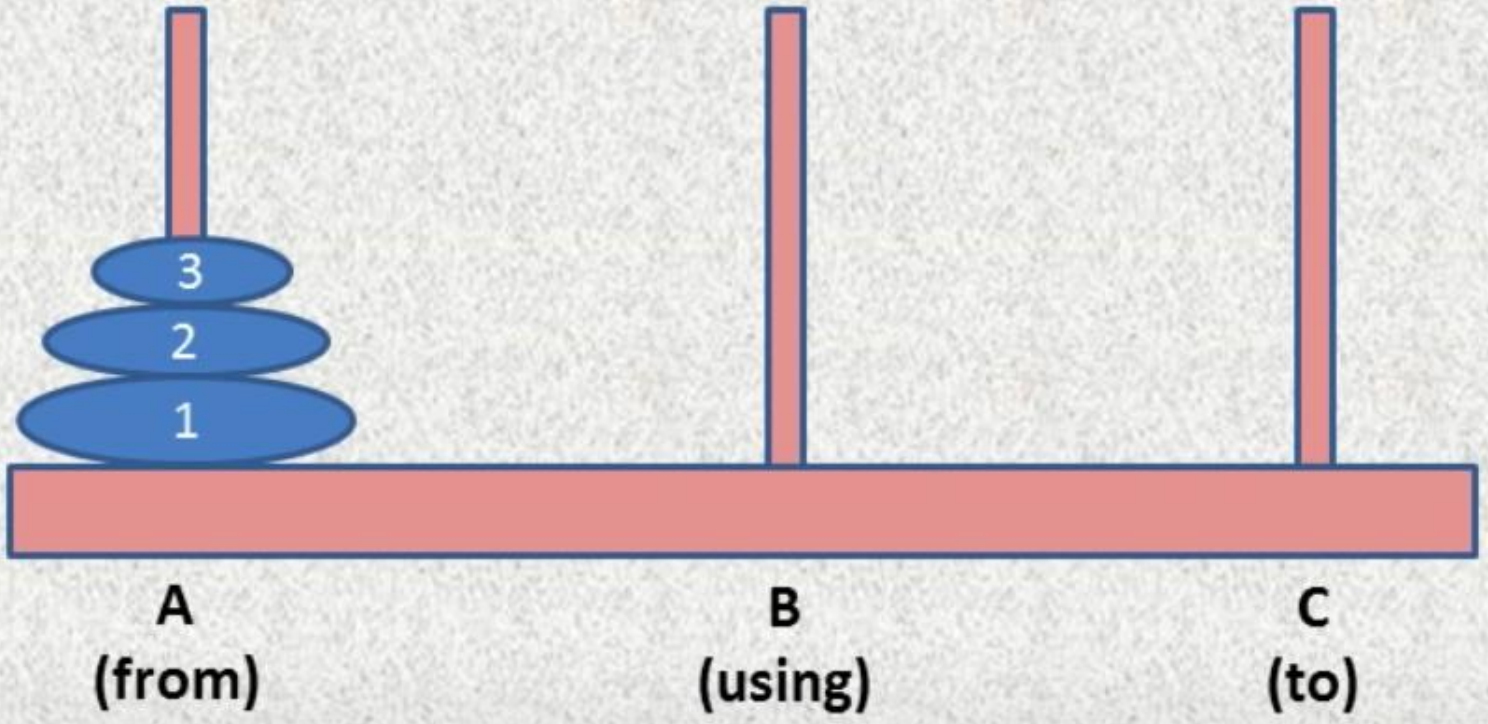
- Move 2 Discs from A to B using C
- Move a Disc from A to C
- Move 2 Discs from B to C using A

Solution for 3 Disc



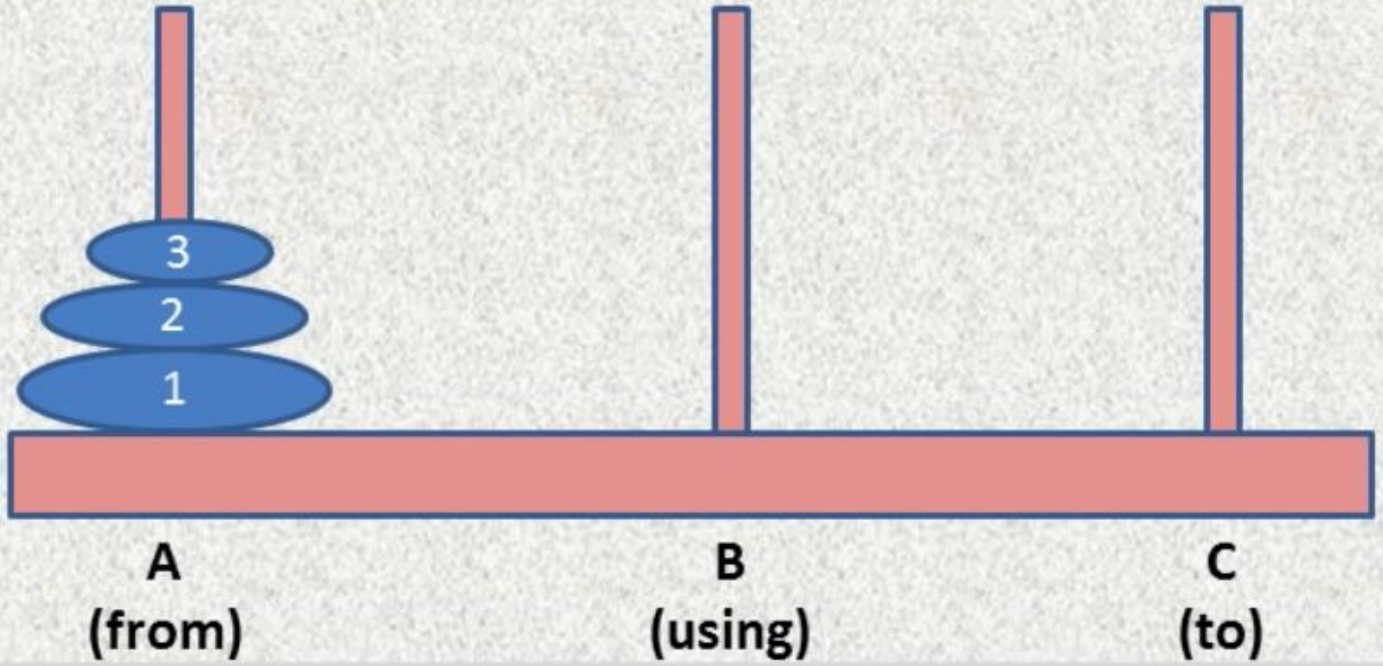
- Move 2 Discs from A to B using C
- Move a Disc from A to C
- Move 2 Discs from B to C using A

Solution for n Disc



- Move n-1 Discs from A to B using C
- Move a Disc from A to C
- Move n-1 Discs from B to C using A

Solution for n Disc




```
void TOH(int n,  int A,  int B,  int C)
{
        if(n>0)
        {
                TOH(n-1, A , C , B);
                printf( "Move a Disc from %d to %d", A , C);
                TOH(n-1, B , A , C);
        }
}
```

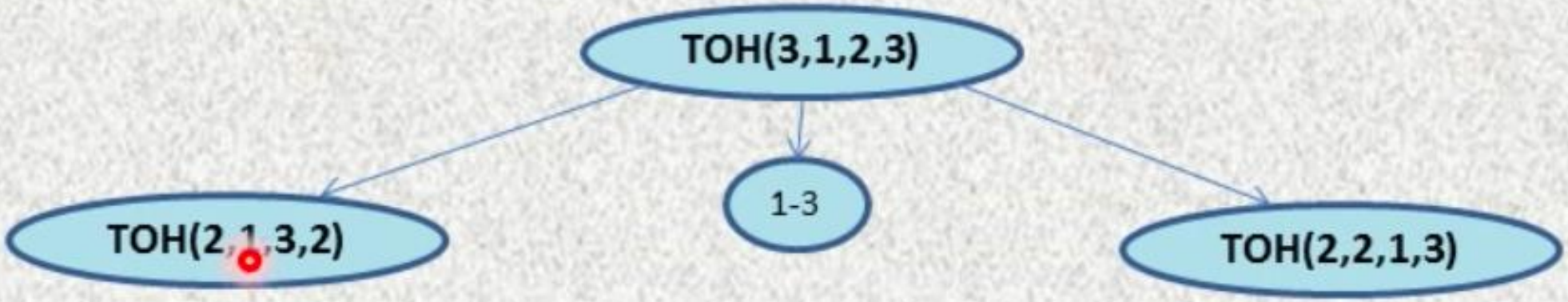
Program

- Move n-1 Discs from A to B using C
- Move a Disc from A to C
- Move n-1 Discs from B to C using A

```
void TOH(int n,  int A,  int B,  int C)
{
        if(n>0)
        {
                TOH(n-1, A , C , B);
                printf( "Move a Disc from %d to %d", A , C);
                TOH(n-1, B , A , C);
        }
}
```

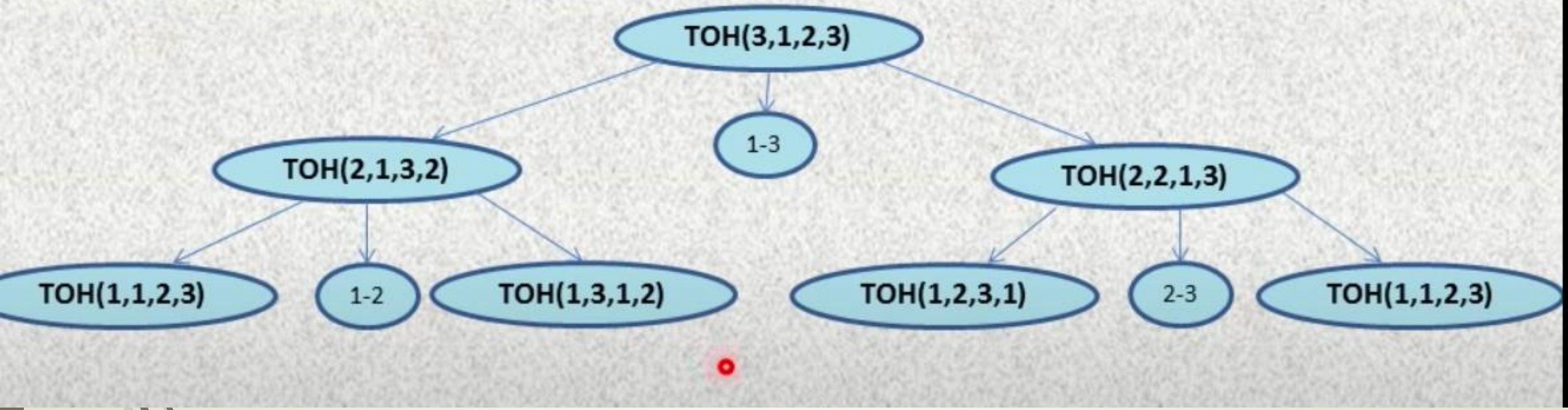
Tracing for 3 Discs

```
void TOH(int n,  int A,  int B,  int C)
{
        if(n>0)
        {
                TOH(n-1, A , C , B);
                printf( "Move a Disc from %d to %d", A , C);
                TOH(n-1, B , A , C);
        }
}
```



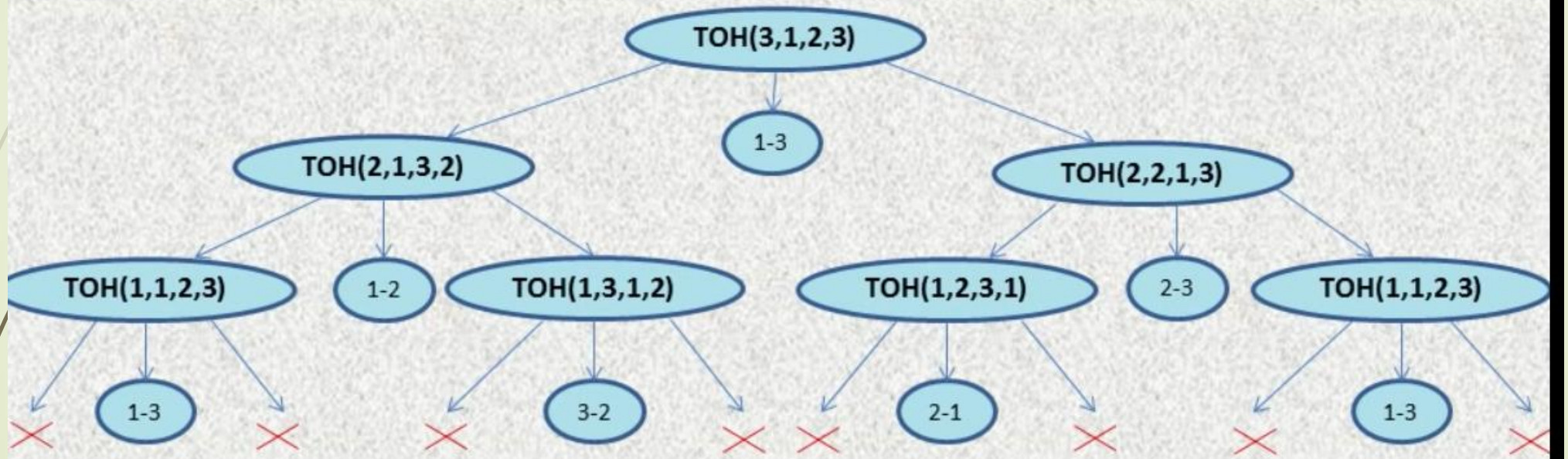
Tracing for 3 Discs

```
void TOH(int n,  int A,  int B,  int C)
{
        if(n>0)
        {
                TOH(n-1, A , C , B);
                printf( "Move a Disc from %d to %d", A , C);
                TOH(n-1, B , A , C);
        }
}
```



Tracing for 3 Discs

```
void TOH(int n,  int A,  int B,  int C)
{
        if(n>0)
        {
                TOH(n-1, A , C , B);
                printf( "Move a Disc from %d to %d", A , C);
                TOH(n-1, B , A , C);
        }
}
```



(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)

Tracing for 3 Discs

(3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)

(1-3)

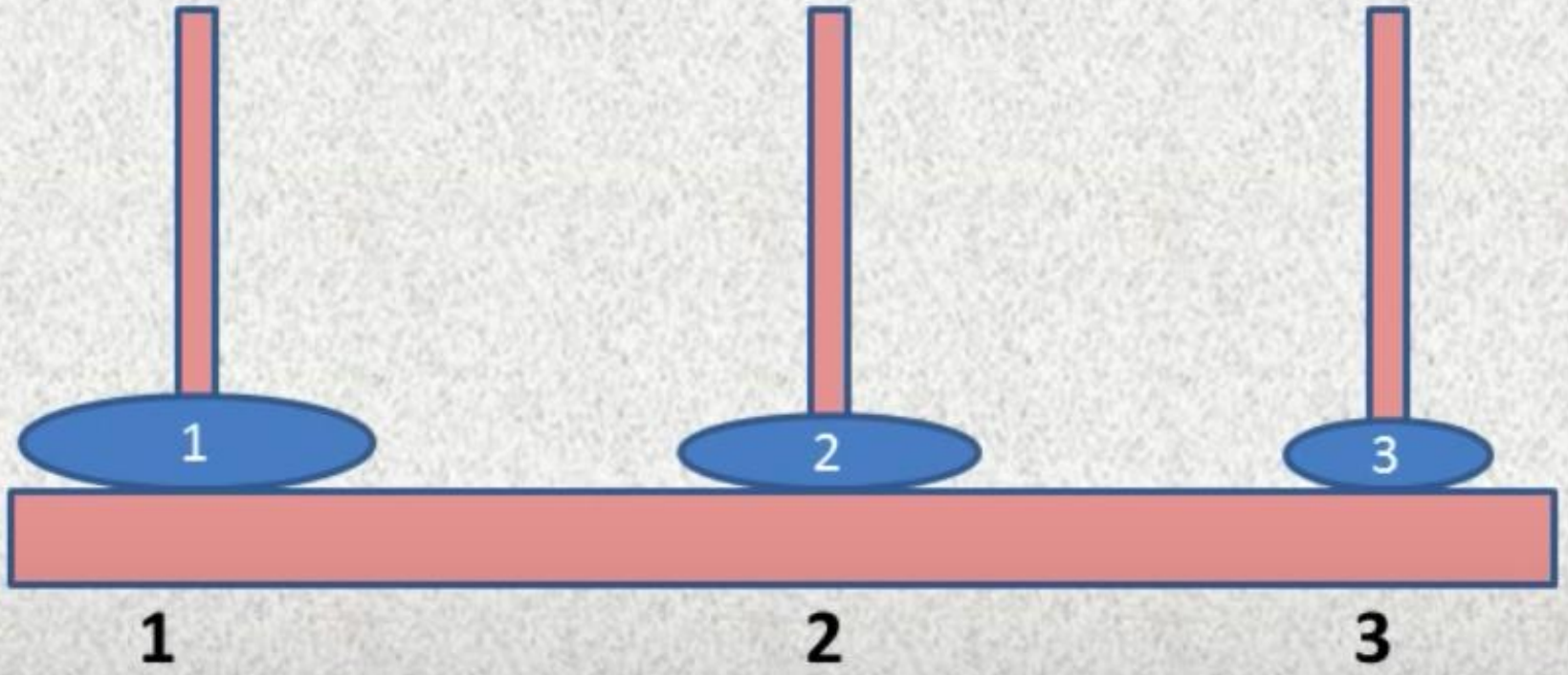
Tracing for 3 Discs

(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)

(1-2)

Tracing for 3 Discs

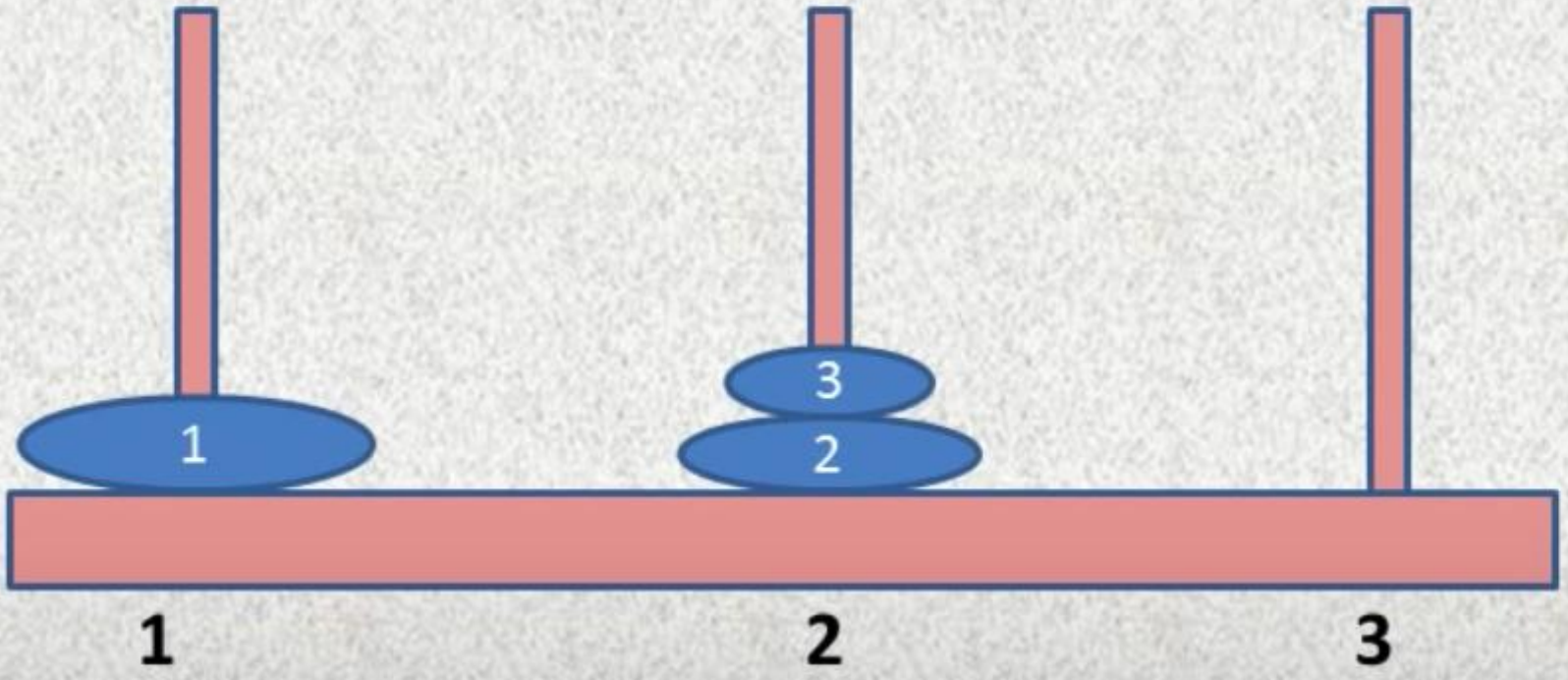
(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)



(3-2)

Tracing for 3 Discs

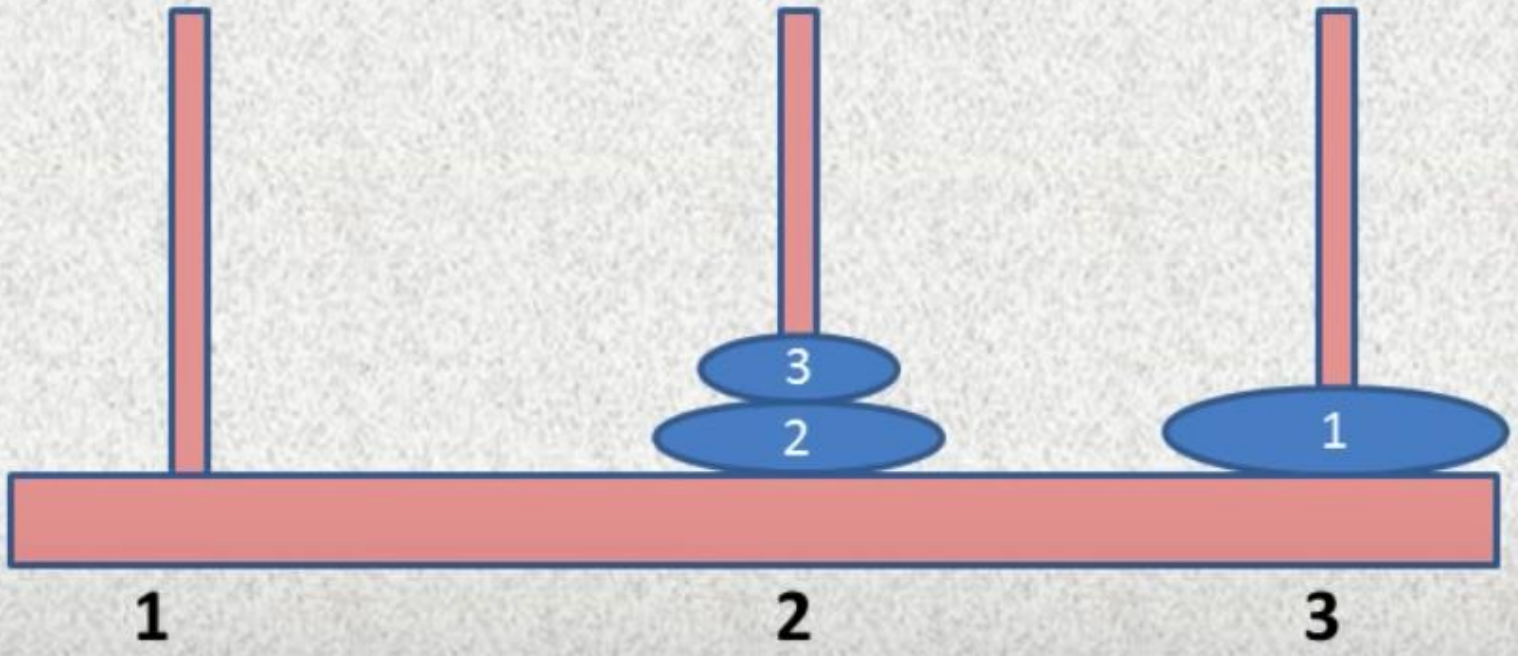
(1-3) (1-2) (3-2) (1-3) **(2-1)** **(2-3)** **(1-3)**



(1-3)

Tracing for 3 Discs

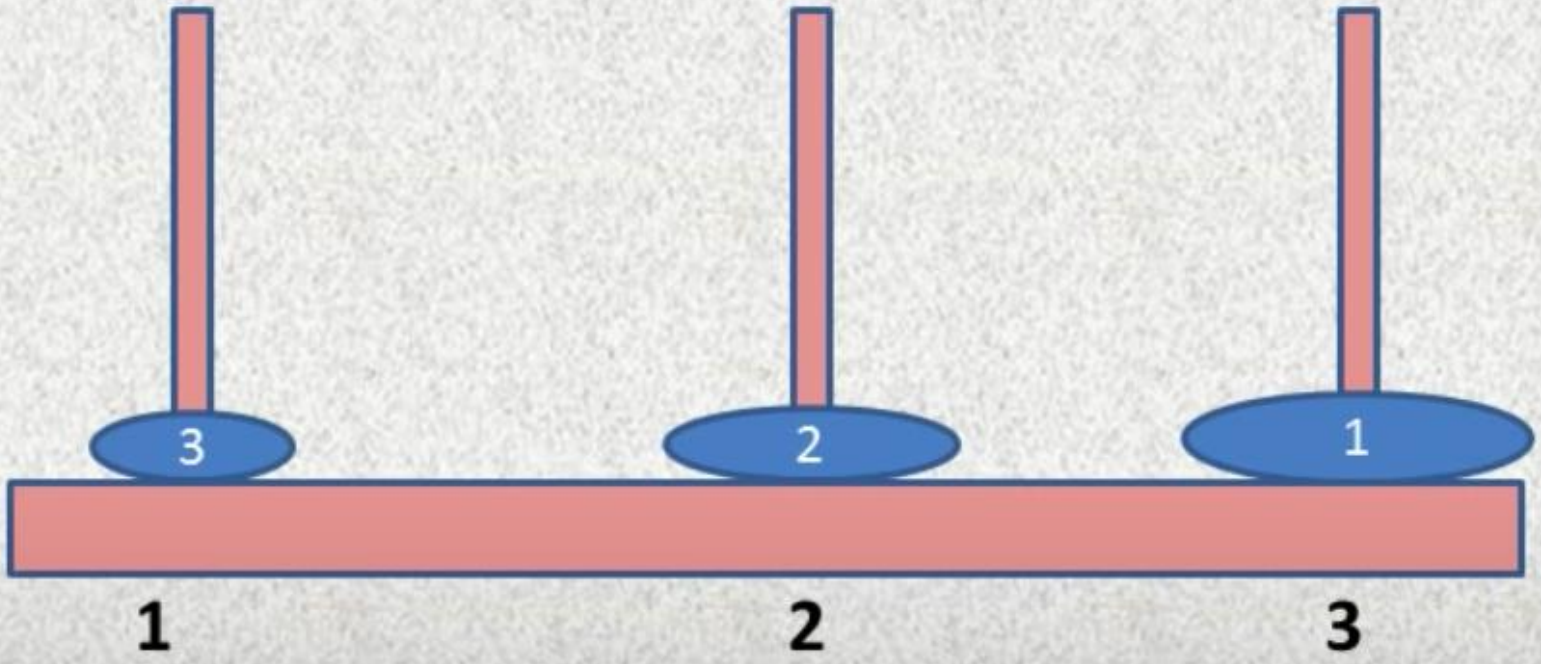
(1-3) (1-2) (3-2) (1-3) (2-1) **(2-3)** **(1-3)**



(2-1)

Tracing for 3 Discs

(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) **(1-3)**



(2-3)

Tracing for 3 Discs

(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)

(1-3)

Tracing for 3 Discs

(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)

Done

Tower of Hanoi

- Explicit Pattern
- Number of Disks Number of Moves

Number of Disks	Number of Moves
1	1
2	3
3	7
4	15
5	31

- Powers of two help reveal the pattern:
- Number of Disks (n) Number of Moves

Number of Disks (n)	Number of Moves
1	2^1 - 1 = 2 - 1 = 1
2	2^2 - 1 = 4 - 1 = 3
3	2^3 - 1 = 8 - 1 = 7
4	2^4 - 1 = 16 - 1 = 15
5	2^5 - 1 = 32 - 1 = 31

Application:

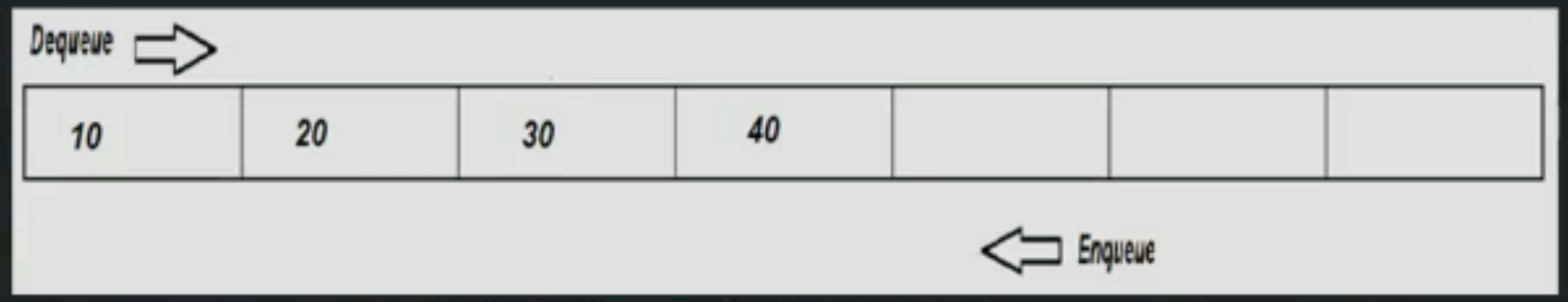
- **Used as Backup rotation Scheme (Backups of Computers having multiple tapes/media)**
  - ✓A backup rotation scheme is a system for managing your backup storage media (tapes/DVDs/HDDs).
- **Used by neuropsychologists trying to evaluate frontal lobe deficits.**
- **Used to Solve mathematical problems related to Hamilton Cycle.**

Queue Data Structure

What is a Queue:

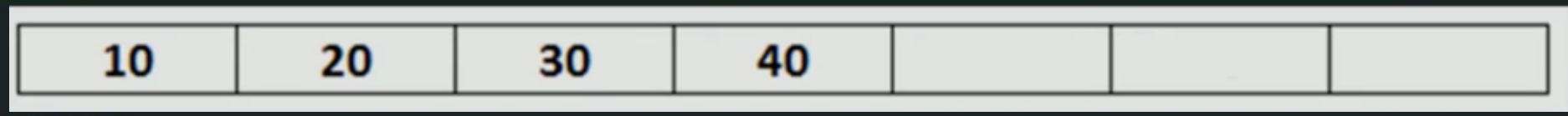
✓ Observations from above Picture:

- ✓ New addition of members happens at end of the queue.
- ✓ First person in the queue is the first to get out from queue.
- ✓ follows FIFO (First in First Out) method


Dequeue
10
20
30
40
Enqueue


COFFEE
ORDER HERE
GRAVITY

Implementation Options of Queue:



- ✓ **Array:**
  - ✓ Linear Queue
  - ✓ Circular Queue

Common operations in Queue:

- ✓ createQueue()
- ✓ enQueue()
- ✓ deQueue()
- ✓ peekInQueue()
- ✓ isEmpty()
- ✓ isFull()
- ✓ deleteQueue()

Creation of Linear Queue (Array Implementation):

```
createQueue(size)

    create a blank array of 'size'

    initialize topOfQueue, beginningOfQueue to -1
```

Enqueue operation of Linear Queue(Array Implementation):

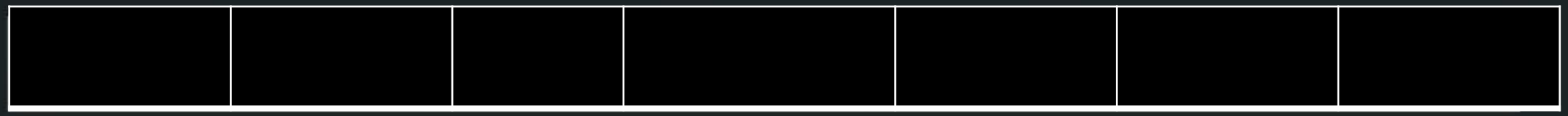
```
enQueue(Value):
    if Queue is full
        return error message
    else
        arr [topOfQueue + 1 ] = Value
        topOfQueue ++
```

Dequeue operation of Linear Queue(Array Implementation):

10	20	30	40	50		

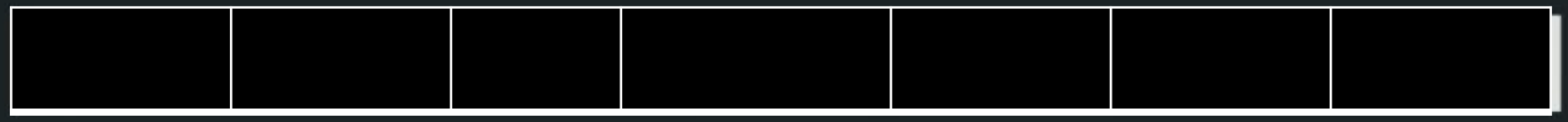
```
deQueue():
    if queue is empty
        return error message
    else
        print arr [beginningOfQueue]
        beginningOfQueue ++
        if (beginningOfQueue > topOfQueue) //If last element in the Queue is Dequeued
            beginningOfQueue = topOfQueue = -1
```

Peek operation of Linear Queue(Array Implementation):



```
peek()
    if queue is empty
        return error message
    else
        print arr [beginningOfQueue]
```

IsEmpty operation of Linear Queue(Array Implementation):



```
IsQueueEmpty()
    if ((beginningOfQueue == -1)
        return true
    else
        return false
```

IsFull operation of Linear Queue(Array Implementation):

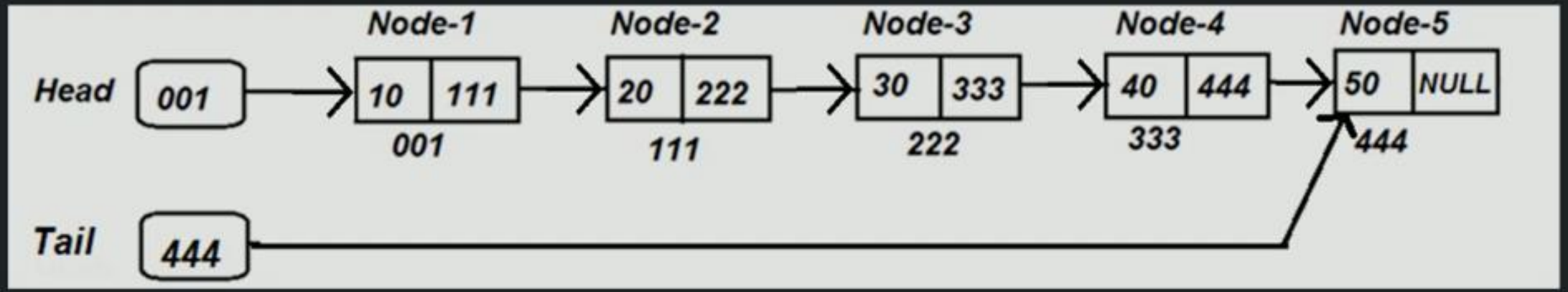
```
isQueueFull()
    if (topOfQueue == arr.length-1)
        return true
    else
        return false
```

Deleting a Linear Queue(Array Implementation):

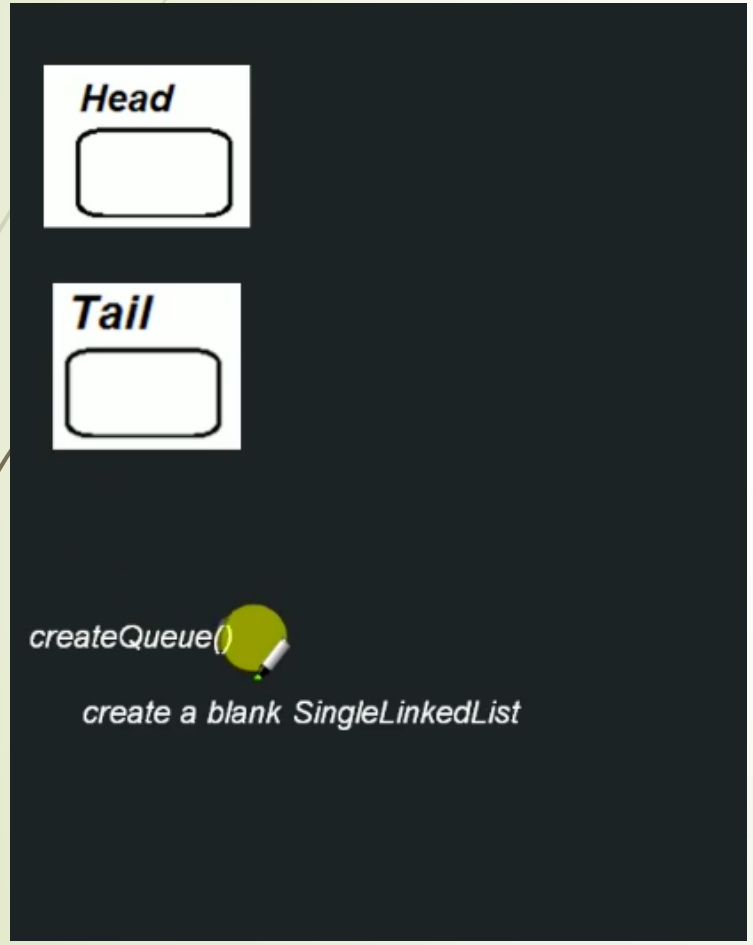
```
deleteQueue():

    array = null
```

Implementation Options of Queue:

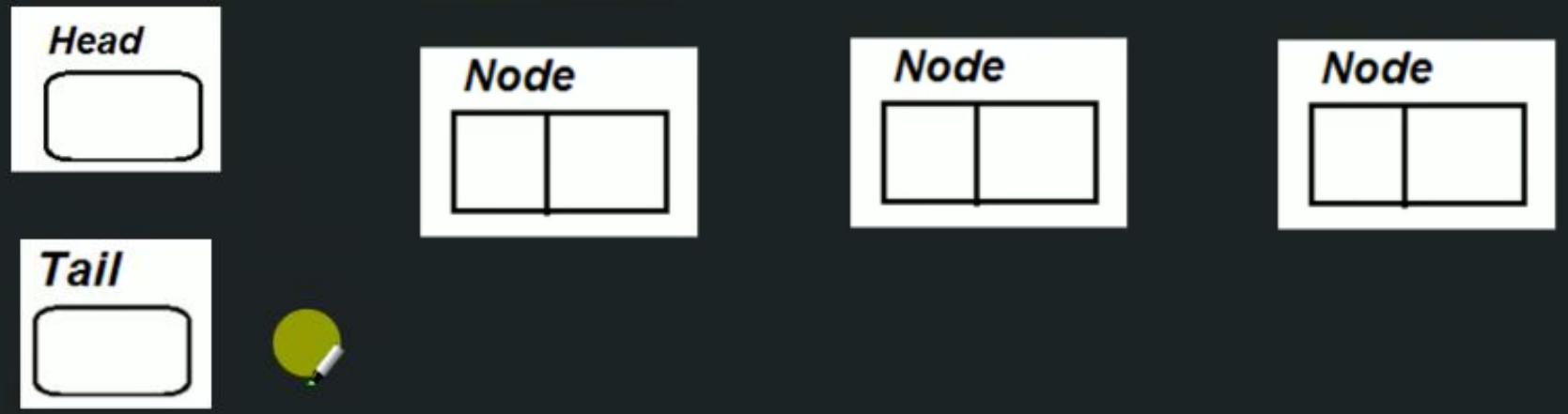
- ✓ **Linked List**
  - ✓ Linear Queue



Creation of Queue(Linked List Implementation):

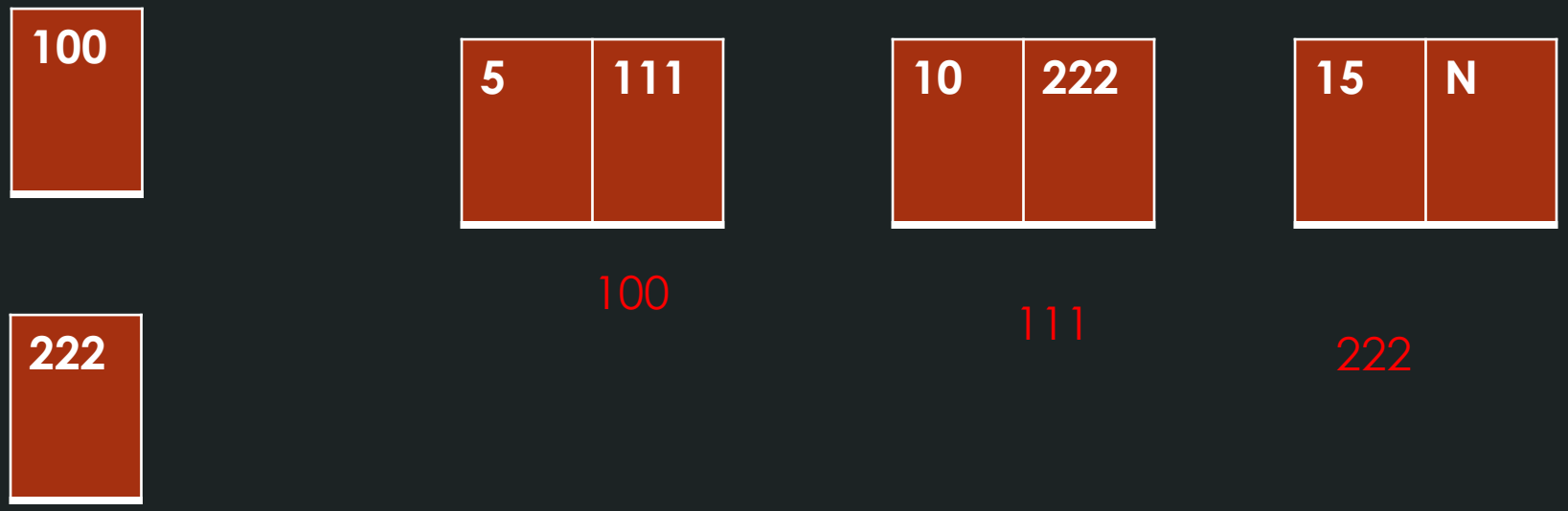


Enqueue operation of Queue(Linked List Implementation):




```
enQueue(nodeValue):
    create a node
    node.value = nodeValue
    node.next = null
    if tail equals null // if queue is empty
head =  tail = node
    else  //if queue is not empty
       tail.next = node
       tail = node
```

Dequeue operation of Queue(Linked List Implementation):



deQueue():

if header equals null

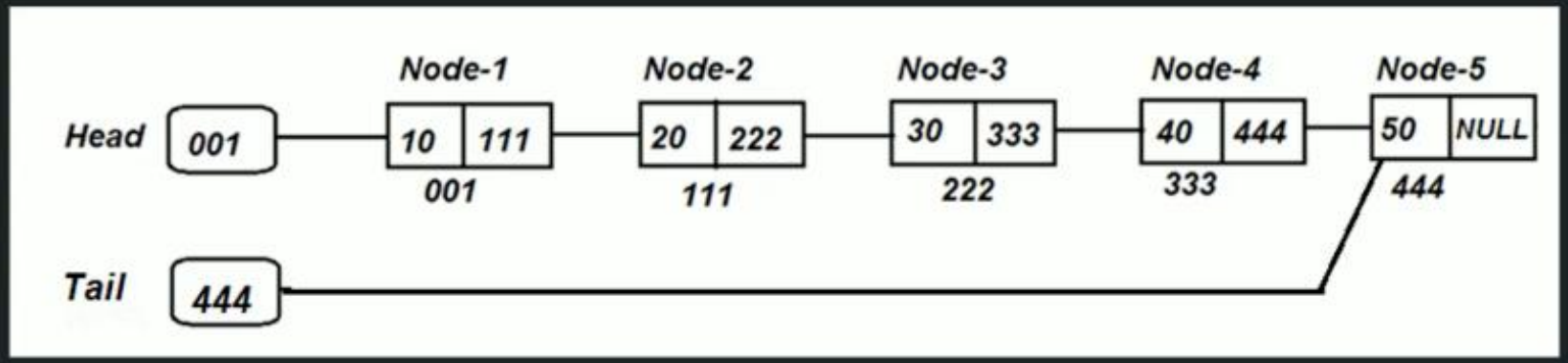
return error message

tmpNode = header

header = header.next

return tmpNode.value

Peek operation of Queue(Linked List Implementation):



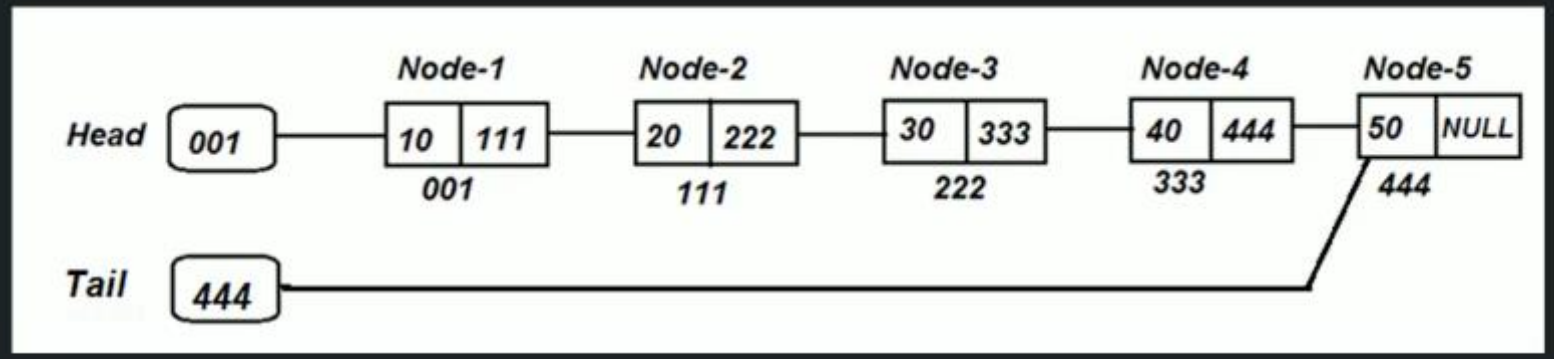
peek():

```
if header equals null
    return error
else
    return header.value
```

IsEmpty operation of Queue(Linked List Implementation):

```
isQueueEmpty():
    if (header equals null)
        return true
    else
        return false
```

Deleting a Queue(Linked List Implementation):



deleteQueue():

head = tail = null


Thank
you