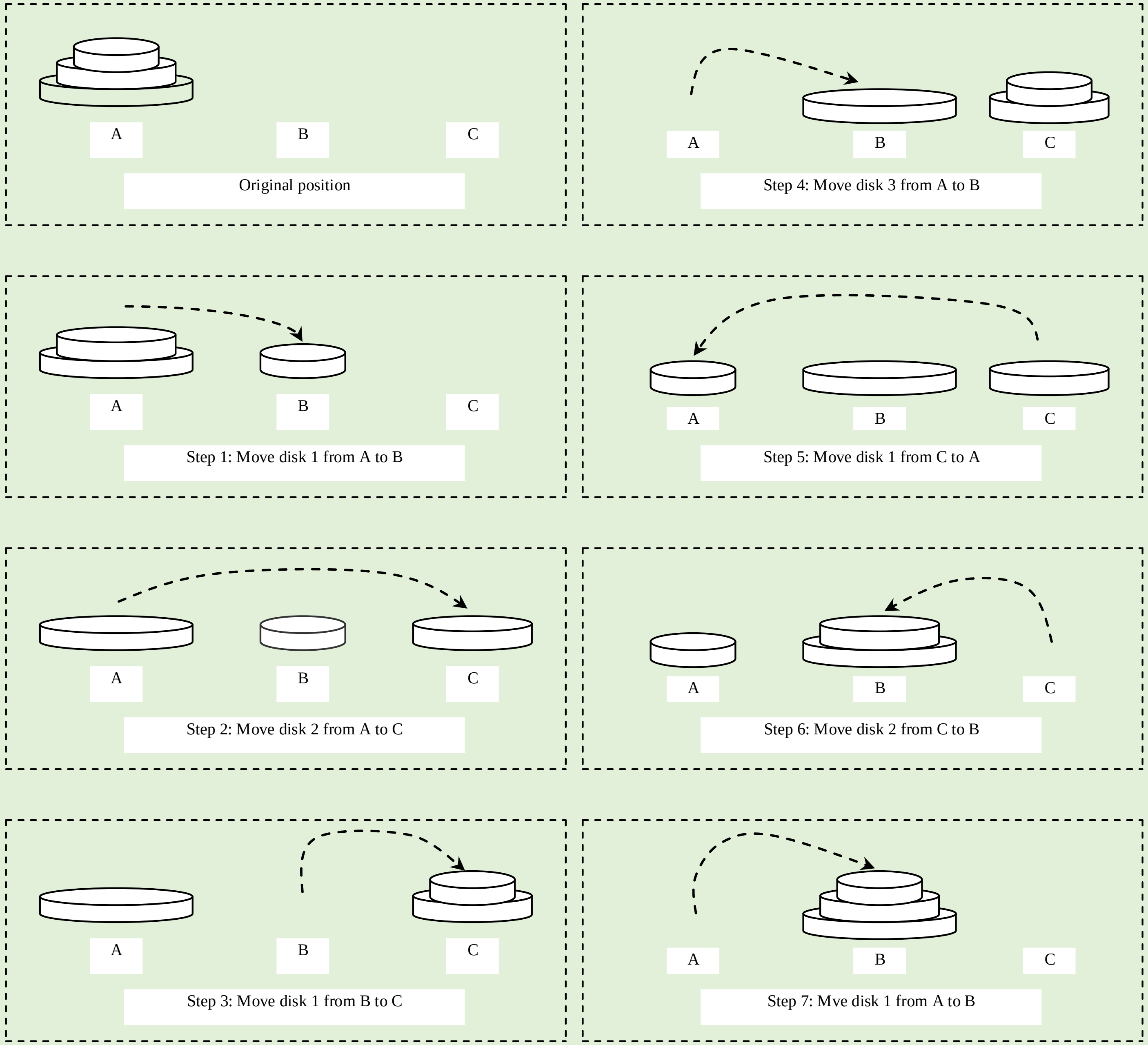
Solving the Towers of Hanoi with Recursion BackTracking Demo

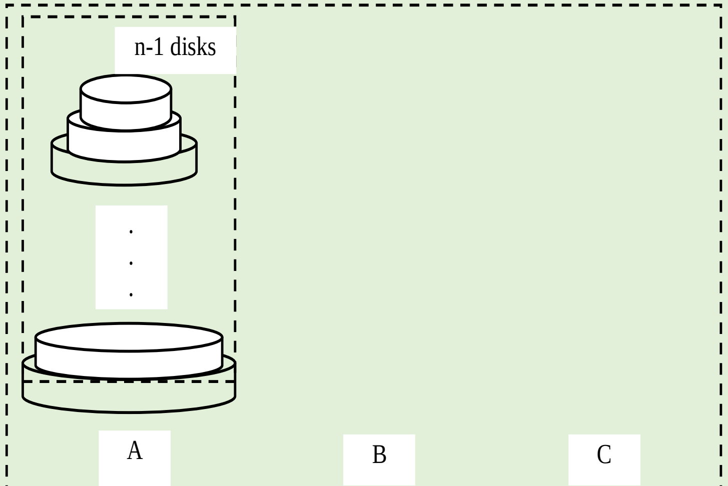
The Towers of Hanoi

- The towers of Hanoi problem has simple recursive solutions but it is otherwise very difficult to solve.
- There are *n* disks labeled 1, 2, 3, . . ., *n*, and three towers labeled A, B, and C.
- No disk can be on top of a smaller disk at any time.
- All the disks are initially placed on tower A.
- Only one disk can be moved at a time, and it must be the top disk on the tower.

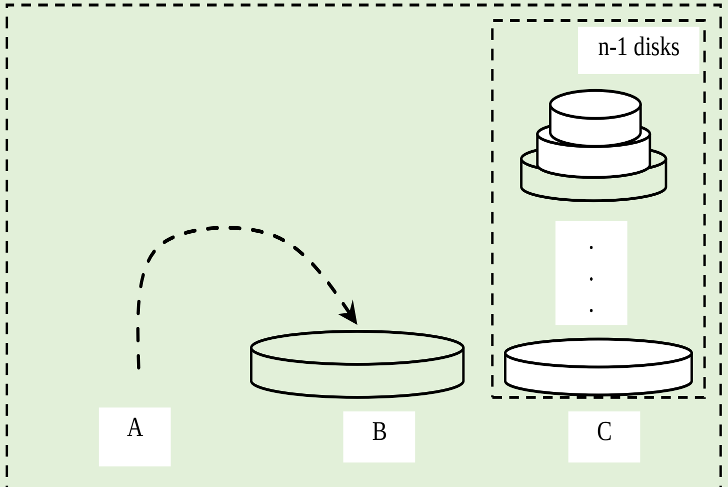
The Towers of Hanoi (cont.)


A
B
C
Original position
A
B
C
Step 1: Move disk 1 from A to B
A
B
C
Step 2: Move disk 2 from A to C
A
B
C
Step 3: Move disk 1 from B to C
A
B
C
Step 4: Move disk 3 from A to B
A
B
C
Step 5: Move disk 1 from C to A
A
B
C
Step 6: Move disk 2 from C to B
A
B
C
Step 7: Mve disk 1 from A to B

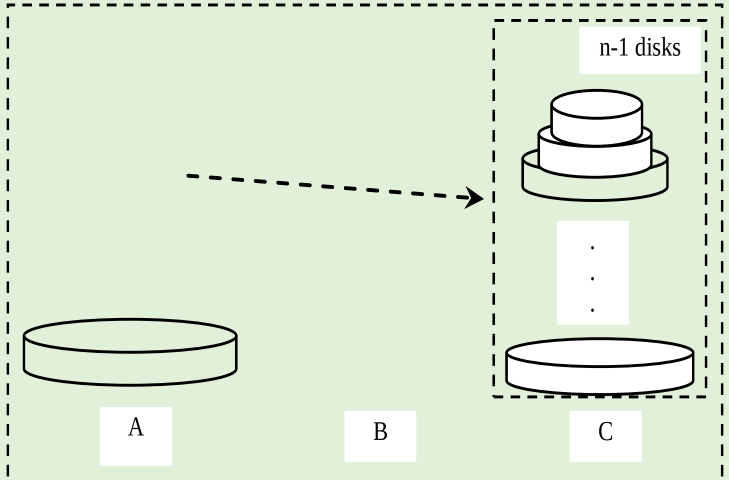
The Towers of Hanoi (cont.)



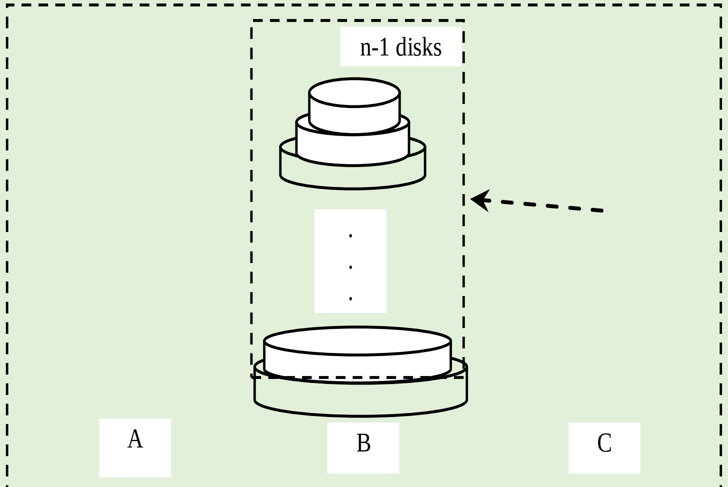
Original position



Step2: Move disk n from A to C



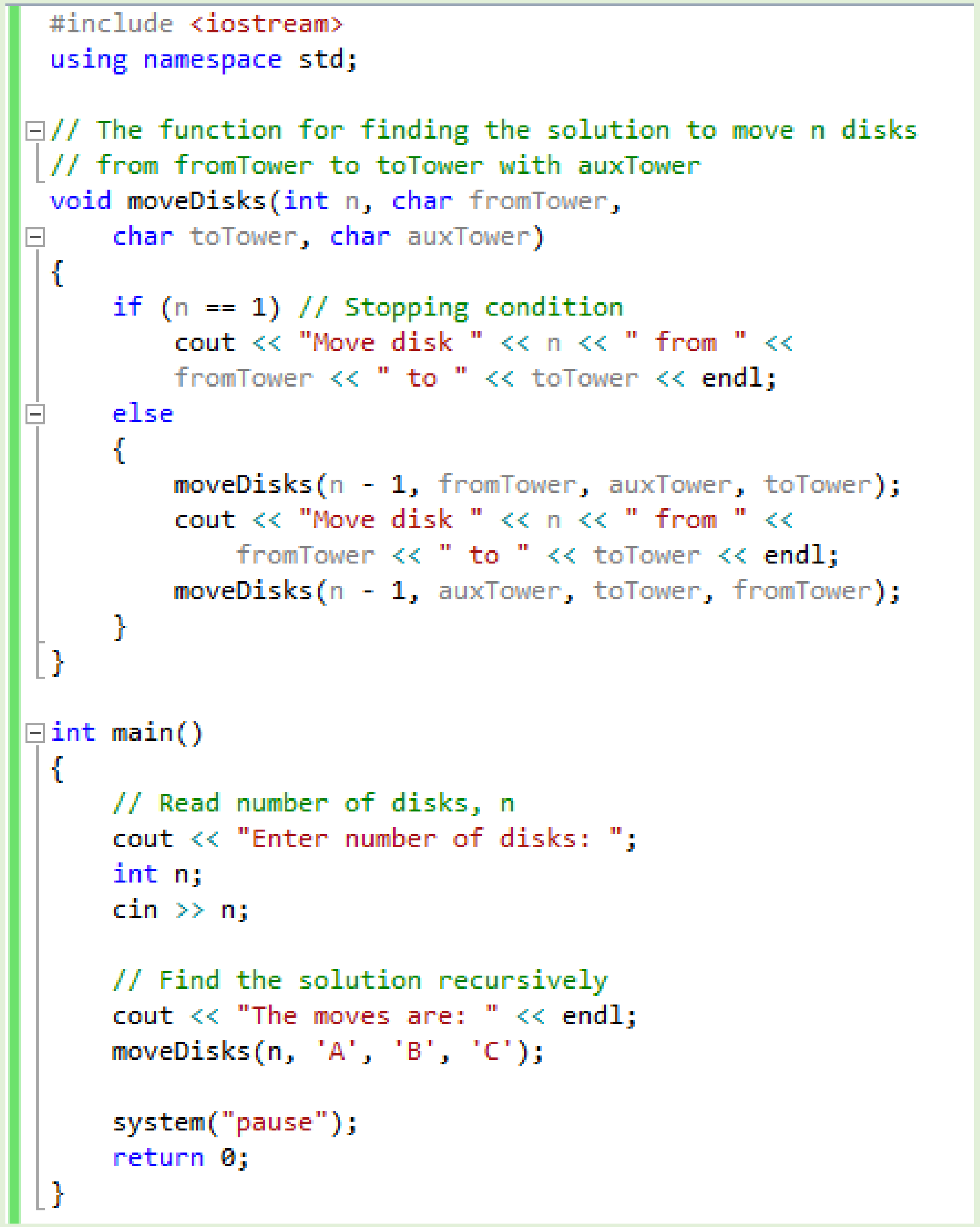
Step 1: Move the first n-1 disks from A to C recursively



Step3: Move n-1 disks from C to B recursively

- The tower of Hanoi problem can be decomposed into three sub problems.
  - Move the first <u>n - 1</u> disks from A to C with the assistance of tower B.
  - Move disk <u>n</u> from A to B.
  - Move <u>n - 1</u> disks from C to B with the assistance of tower A.

Towers of Hanoi Implementation



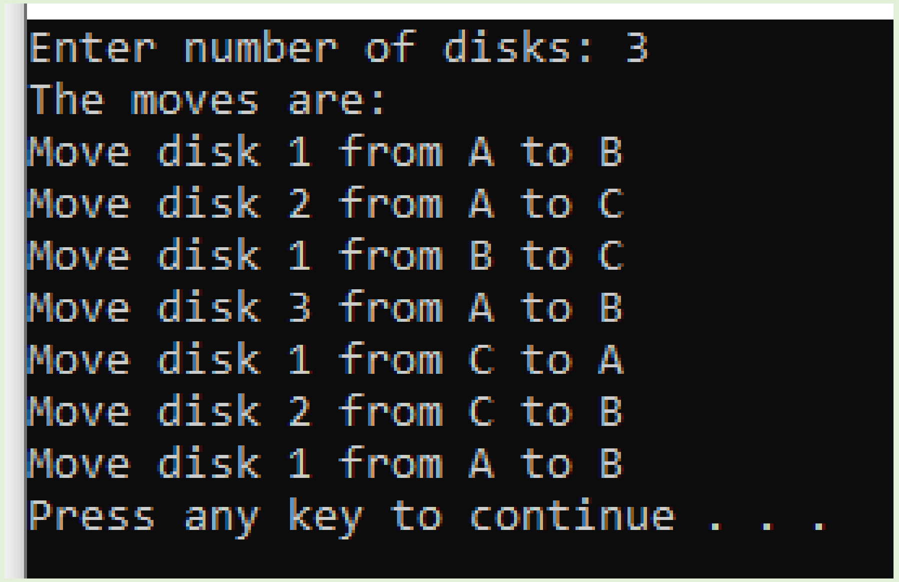
```cpp
#include <iostream>
using namespace std;

// The function for finding the solution to move n disks
// from fromTower to toTower with auxTower
void moveDisks(int n, char fromTower,
    char toTower, char auxTower)
{
    if (n == 1) // Stopping condition
        cout << "Move disk " << n << " from " <<
        fromTower << " to " << toTower << endl;
    else
    {
        moveDisks(n - 1, fromTower, auxTower, toTower);
        cout << "Move disk " << n << " from " <<
            fromTower << " to " << toTower << endl;
        moveDisks(n - 1, auxTower, toTower, fromTower);
    }
}

int main()
{
    // Read number of disks, n
    cout << "Enter number of disks: ";
    int n;
    cin >> n;

    // Find the solution recursively
    cout << "The moves are: " << endl;
    moveDisks(n, 'A', 'B', 'C');

    system("pause");
    return 0;
}
```



```
Enter number of disks: 3
The moves are:
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to B
Move disk 1 from C to A
Move disk 2 from C to B
Move disk 1 from A to B
Press any key to continue . . .
```

Backtracking

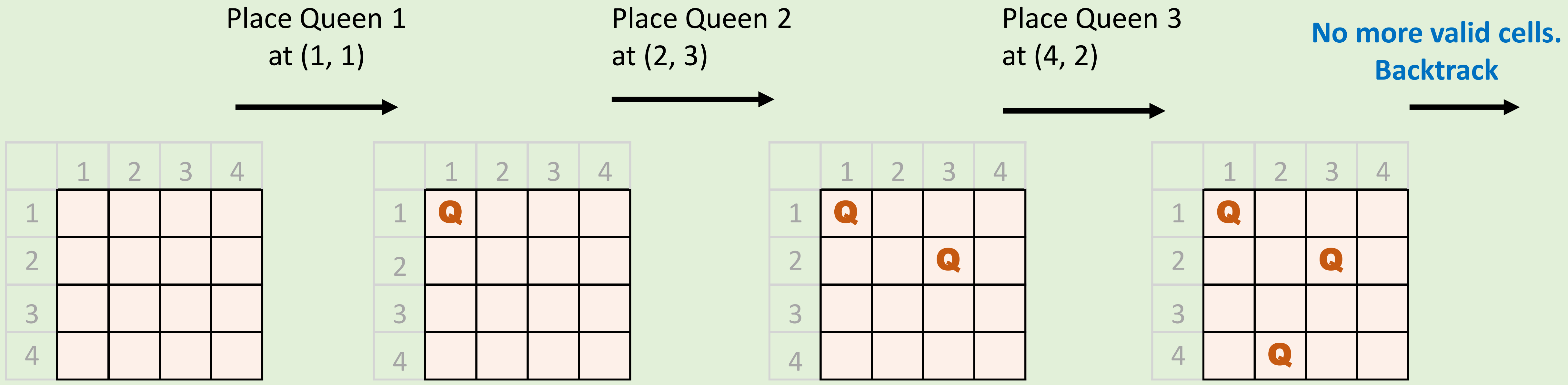
What is Backtracking?

- Suppose you are standing in front of three tunnels, one of which is having a bag of gold at the end of tunnel, but you don’t know which one. So you will go to all three:
- First go in tunnel 1, if that is not the one, then come out of it and go into tunnel 2, and again if that is not the one, come out of it and go into tunnel 3.
- So Literally! Backtracking problems are solved one step at a time.

Example of N-Queen Problem

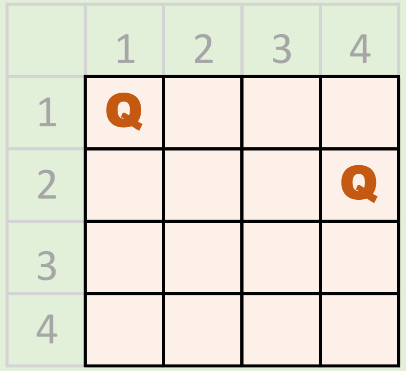
- Given a chess board having N x N cells, we need to place N queens in such a way that no queen is attached by any other queen. A queen can attack horizontally, vertically and diagonally.
- So, initially we have N x N un-attacked cells where we need place N queens. We place the first queen at a cell *(i,j),* so now the number of un-attacked cells is reduced, and number of queens to be placed is (N – 1). Place the next queen at some un-attacked cell. This again reduces the number of un-attacked cells and number of queens to be placed becomes (N – 2). Continue doing this, as long as the conditions hold.
- If the number of queens to be placed becomes 0, then it's over, we found a solution.
- If the number of un-attached cells become 0, then we need to backtrack, i.e. remove the last placed queen from its current cell, and place it at some other cell. We do this recursively.

Example N = 4

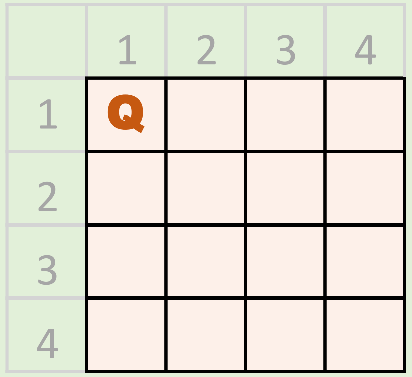

Place Queen 1
at (1, 1)
Place Queen 2
at (2, 3)
Place Queen 3
at (4, 2)
No more valid cells.
Backtrack
1
2
3
4
Q

Example N = 4 (cont.)

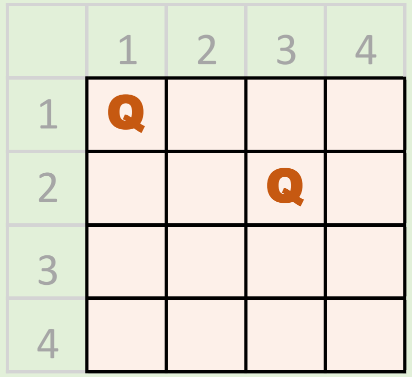
Place Queen 2
at (4, 2)

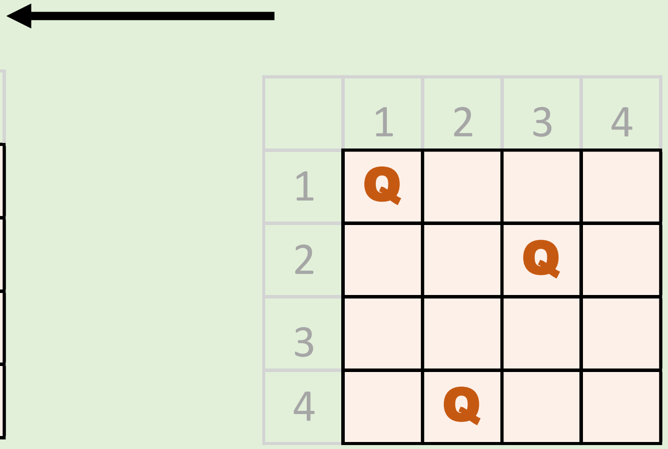


No more valid cells.
Backtrack



No more valid cells.
Backtrack





Example N = 4 (cont.)

	1	2	3	4
1	Q			
2				Q
3				
4				

Place Queen 3
at (3, 2)

$\longrightarrow$

	1	2	3	4
1	Q			
2				Q
3		Q		
4				

**No more valid cells.
Backtrack**

$\longrightarrow$

	1	2	3	4
1	Q			
2				Q
3				
4				

Place Queen 3
at (4, 3)

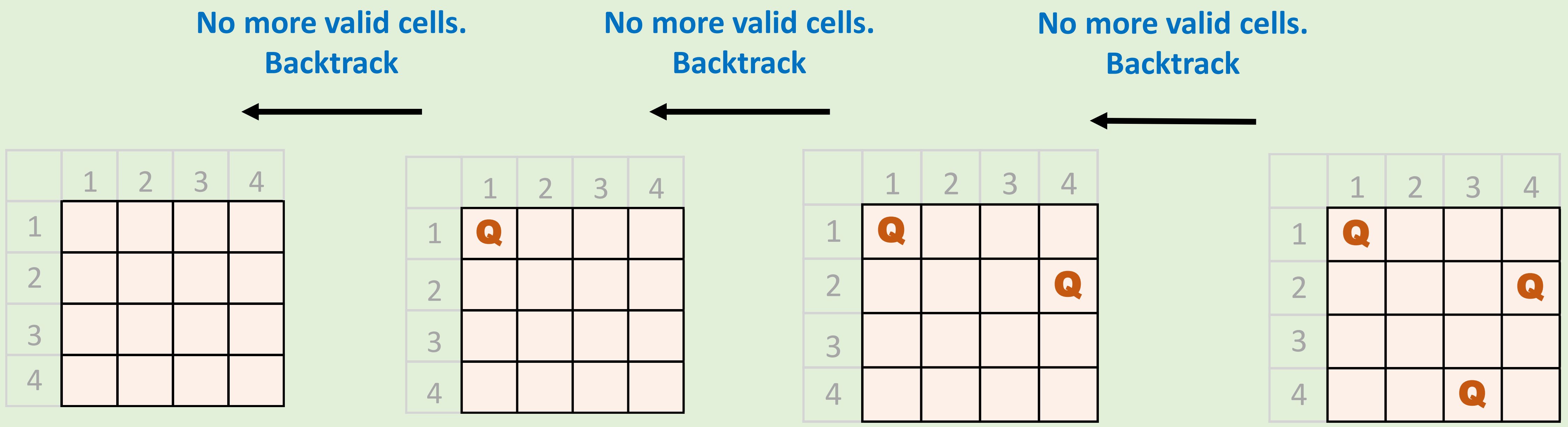
$\longrightarrow$

	1	2	3	4
1	Q			
2				Q
3				
4			Q	

**No more valid cells.
Backtrack**

$\longrightarrow$

Example N = 4;


No more valid cells.
Backtrack
No more valid cells.
Backtrack
No more valid cells.
Backtrack
1 2 3 4
1 2 3 4
Q
Q
Q
Q
Q
Q
Q

Example N = 4

And so on … until it reaches the following solution:

	1	2	3	4
1		Q		
2				Q
3	Q			
4			Q	

The End;