FIBONACCI NUMBERS

DYNAMIC PROGRAMMING

Fibonacci sequence: 0 1 1 2 3 5 8 13 21 34 ...

Fibonacci numbers are defined by the recurrence relation

$$F(N) = F(N-1) + F(N-2)$$

$$F(0) = 0 \quad F(1) = 1$$

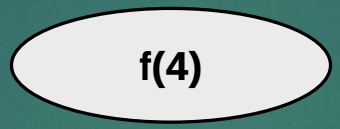
With generator functions we can get a closed form: „Binet formula”

What is the problem with the recursive formula? We calculate same problems over and over again

f(n) = f(n-1) + f(n-2)

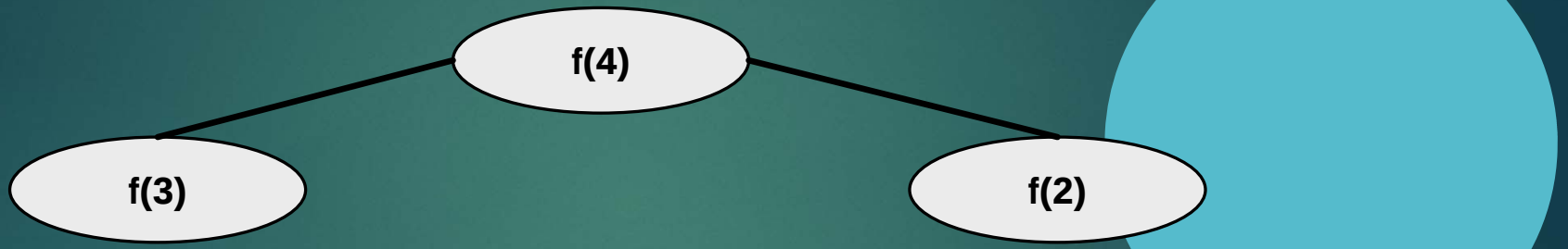
What is the problem with the recursive formula? We calculate same problems over and over again

f(n) = f(n-1) + f(n-2)



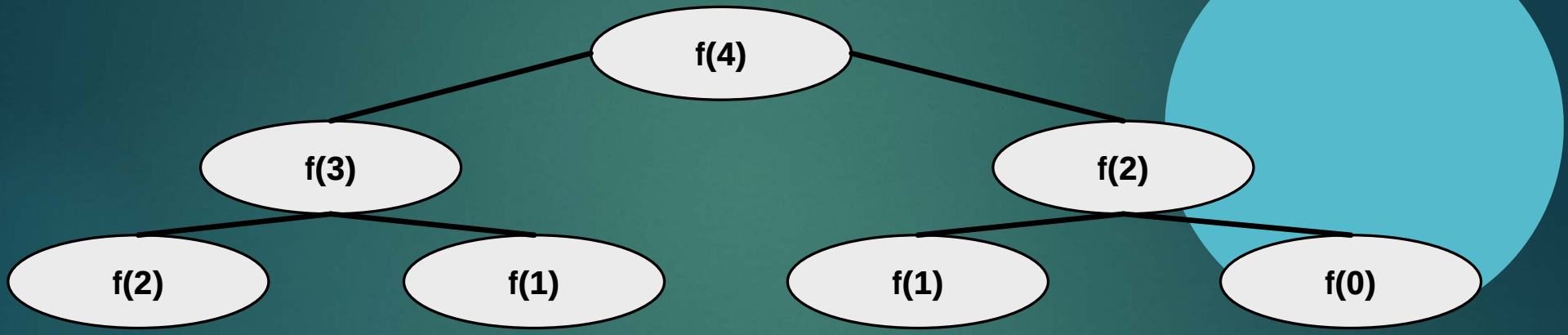
What is the problem with the recursive formula? We calculate same problems over and over again

f(n) = f(n-1) + f(n-2)



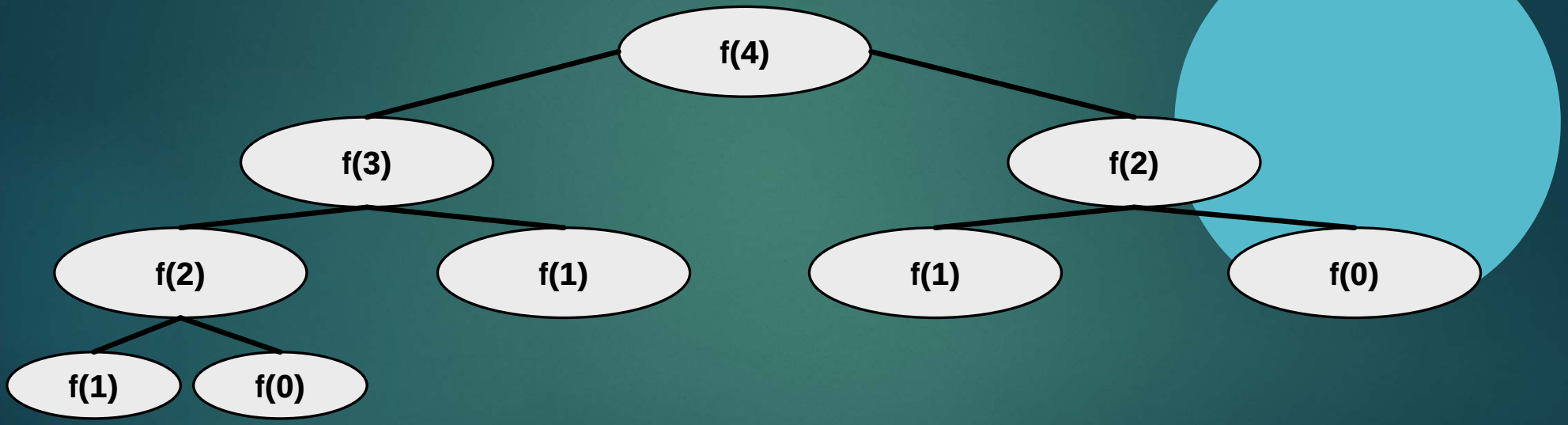
What is the problem with the recursive formula? We calculate same problems over and over again

f(n) = f(n-1) + f(n-2)



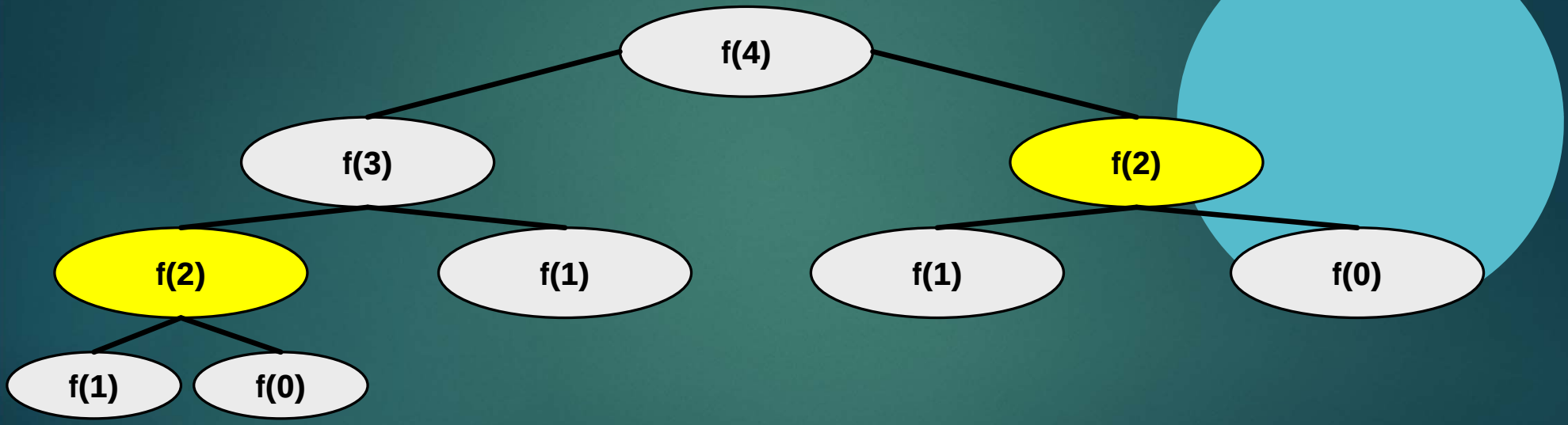
What is the problem with the recursive formula? We calculate same problems over and over again

f(n) = f(n-1) + f(n-2)



What is the problem with the recursive formula? We calculate same problems over and over again

f(n) = f(n-1) + f(n-2)



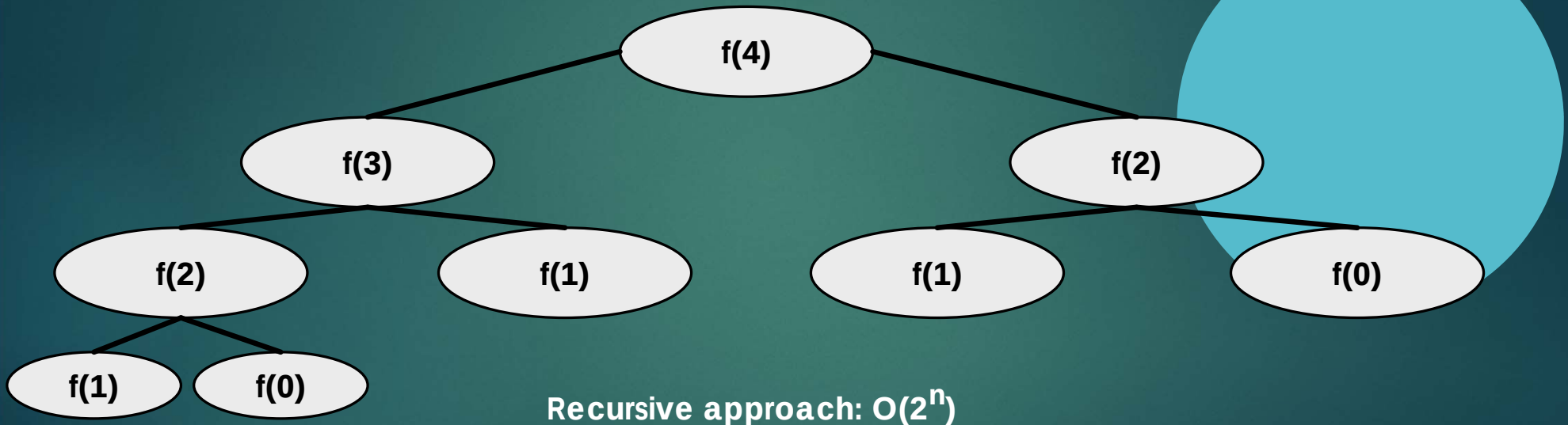
OVERLAPPING SUBPROBLEMS !!!

Fibonacci numbers

- Solution: use dynamic programming and memoization in order to avoid recalculating a subproblem over and over again
- We should use an associative array abstract data type (hashtable) to store the solution for the subproblems // **O(1)** time complexity
- On every **f()** method call → we insert the calculated value if necessary
- Why is it good? Instead of the exponential time complexity we will have **O(N)** time complexity + requires **O(N)** space

What is the problem with the recursive formula? We calculate same problems over and over again

f(n) = f(n-1) + f(n-2)



Recursive approach: $O(2^n)$

Dynamic programming: O(N)