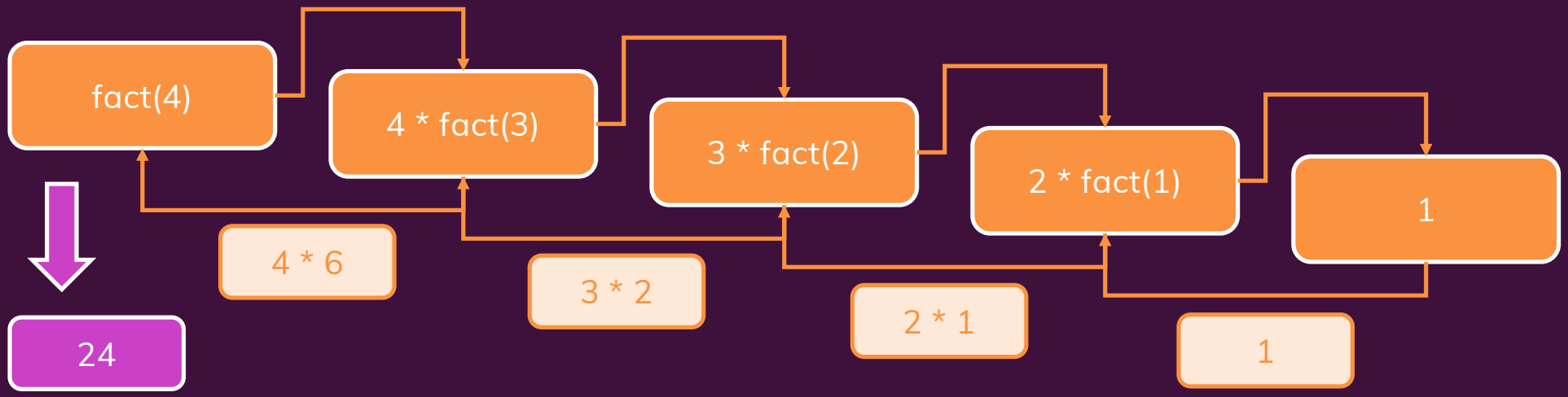
ACADEMIND

Factorial – Recursive Solution

Base Case

```
function fact(n) {
  if (n <= 1) {
    return 1;
  }
  return n * fact(n - 1);
}
```

Recursive Step

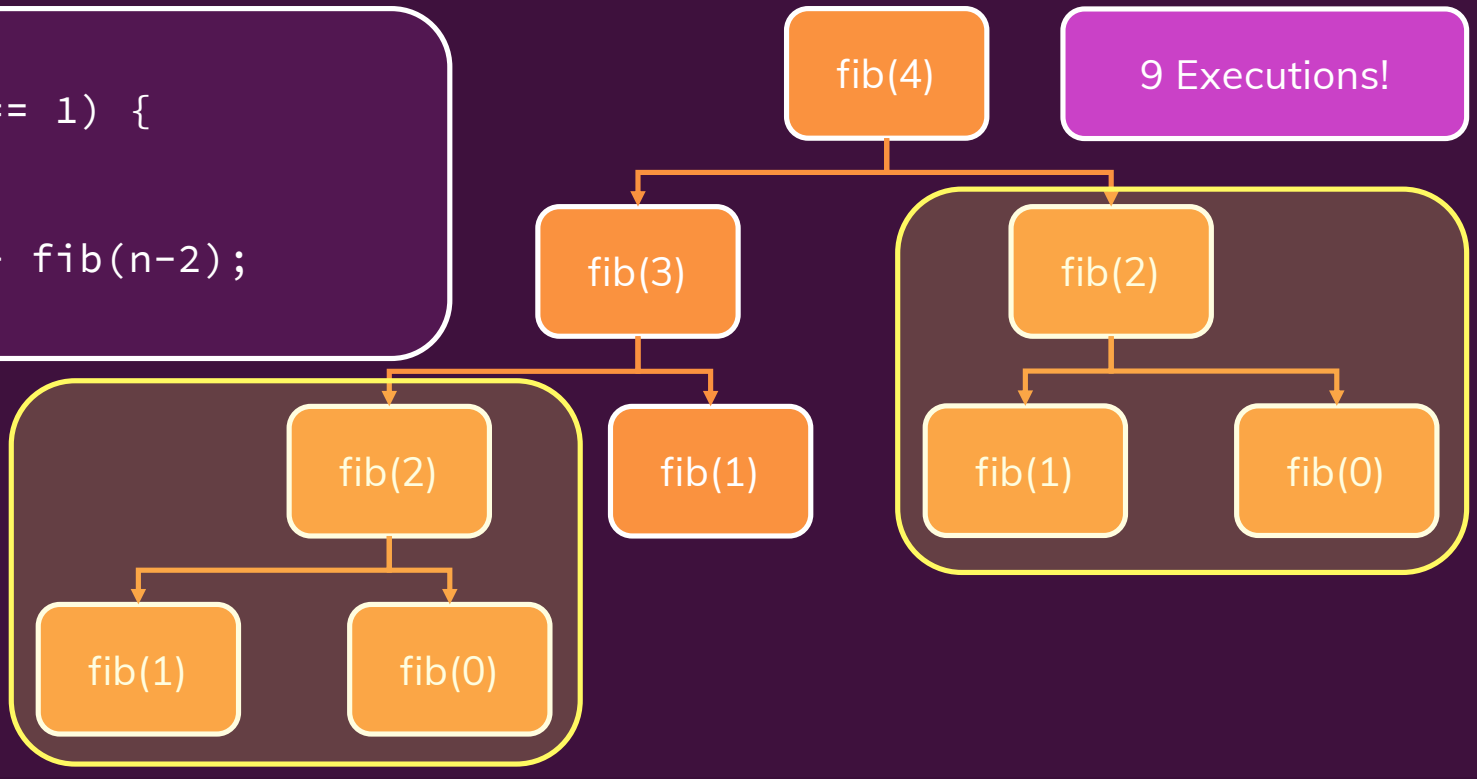


ACADEMIND

Recursion Is Not Always Best

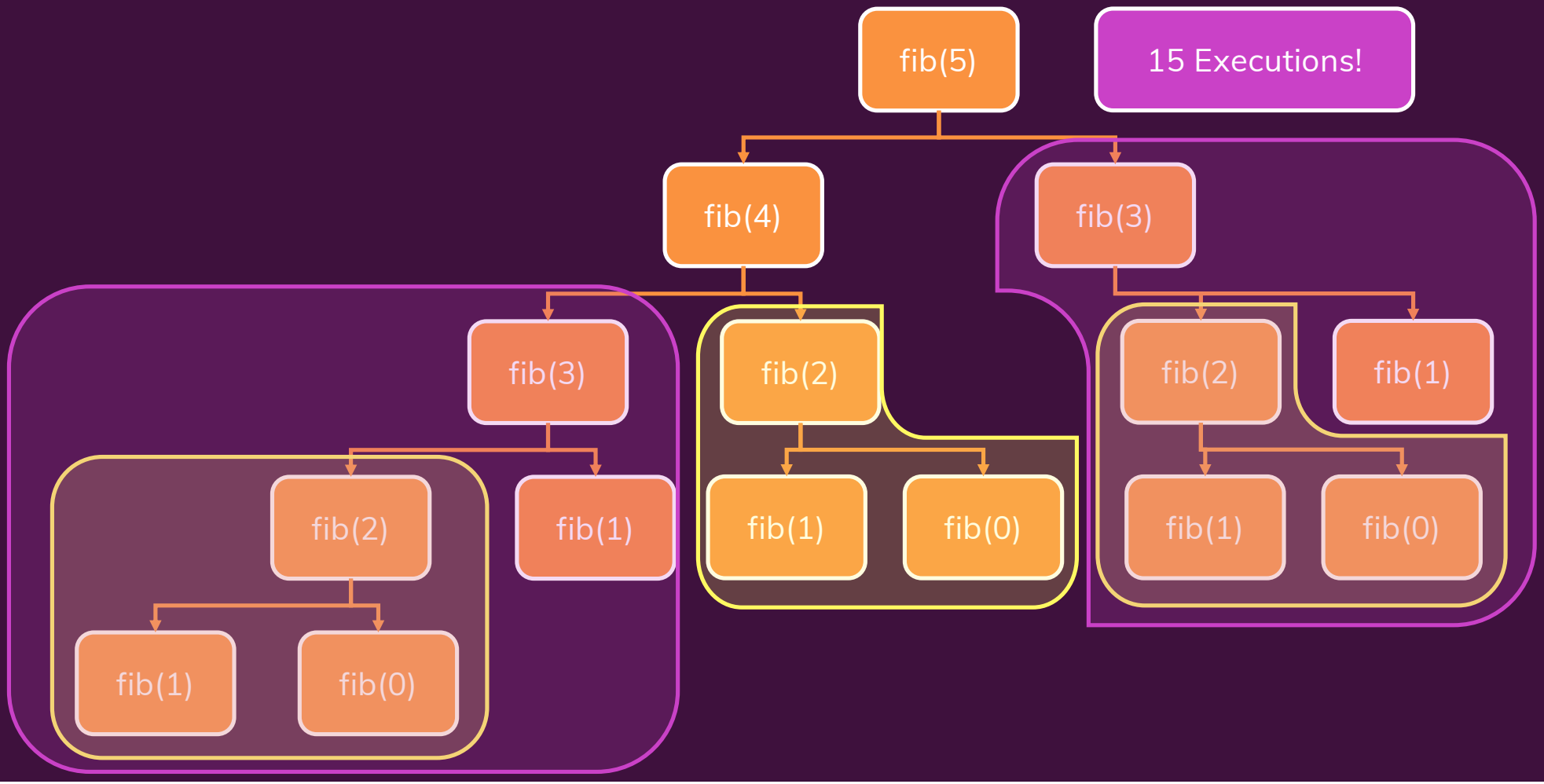
Recursive Fibonacci

```
function fib(n) {
  if (n == 0 || n == 1) {
    return 1;
  }
  return fib(n-1) + fib(n-2);
}
```



ACADEMIND

Recursion Is Not Always Best

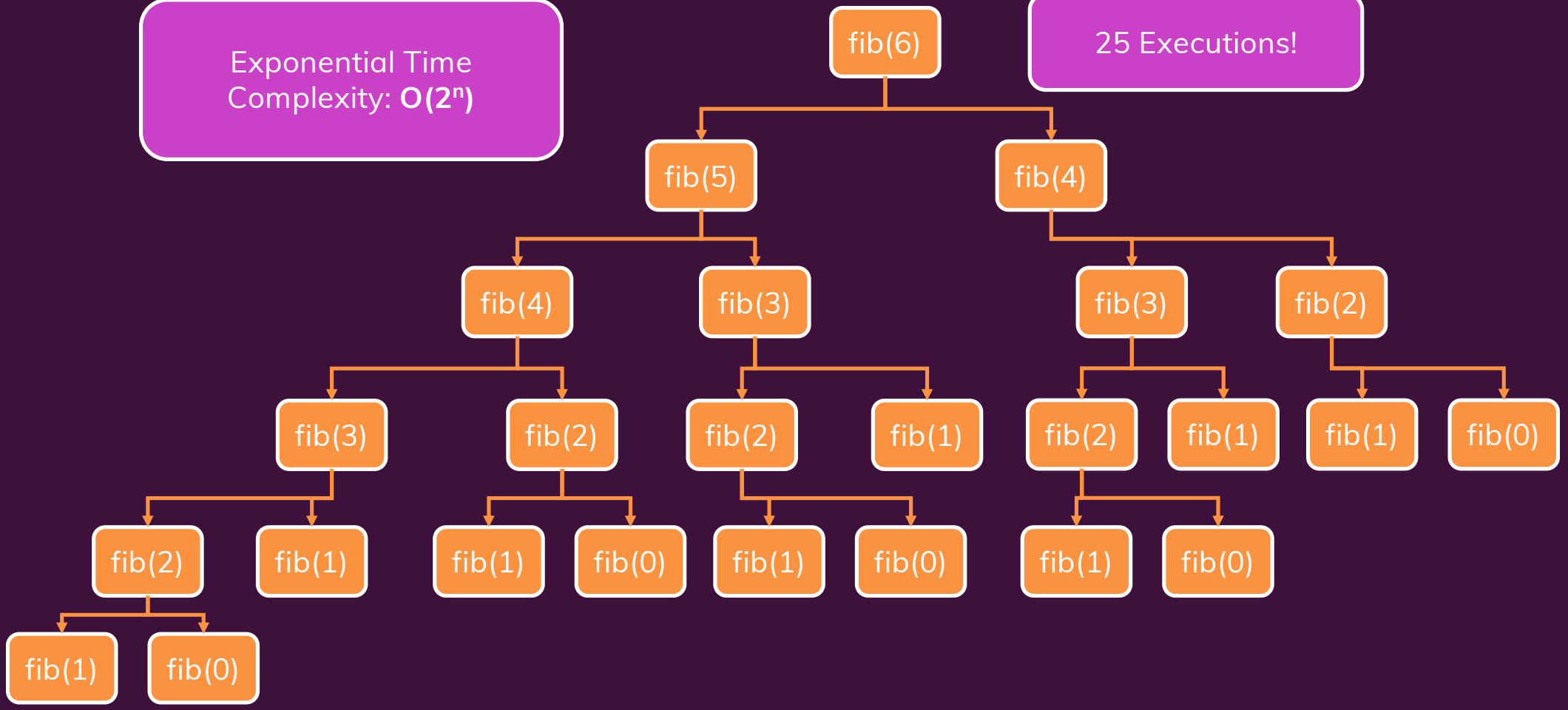


ACADEMIND

Recursion Is Not Always Best

Exponential Time Complexity: **O(2^n)**

25 Executions!



ACADEMIND

What is “Dynamic Programming”?

Recursion

+

Stored Data (“Memoization”)

Great for repeated computations

Can lead to duplicate work though

Avoid unnecessary recursive steps by storing data

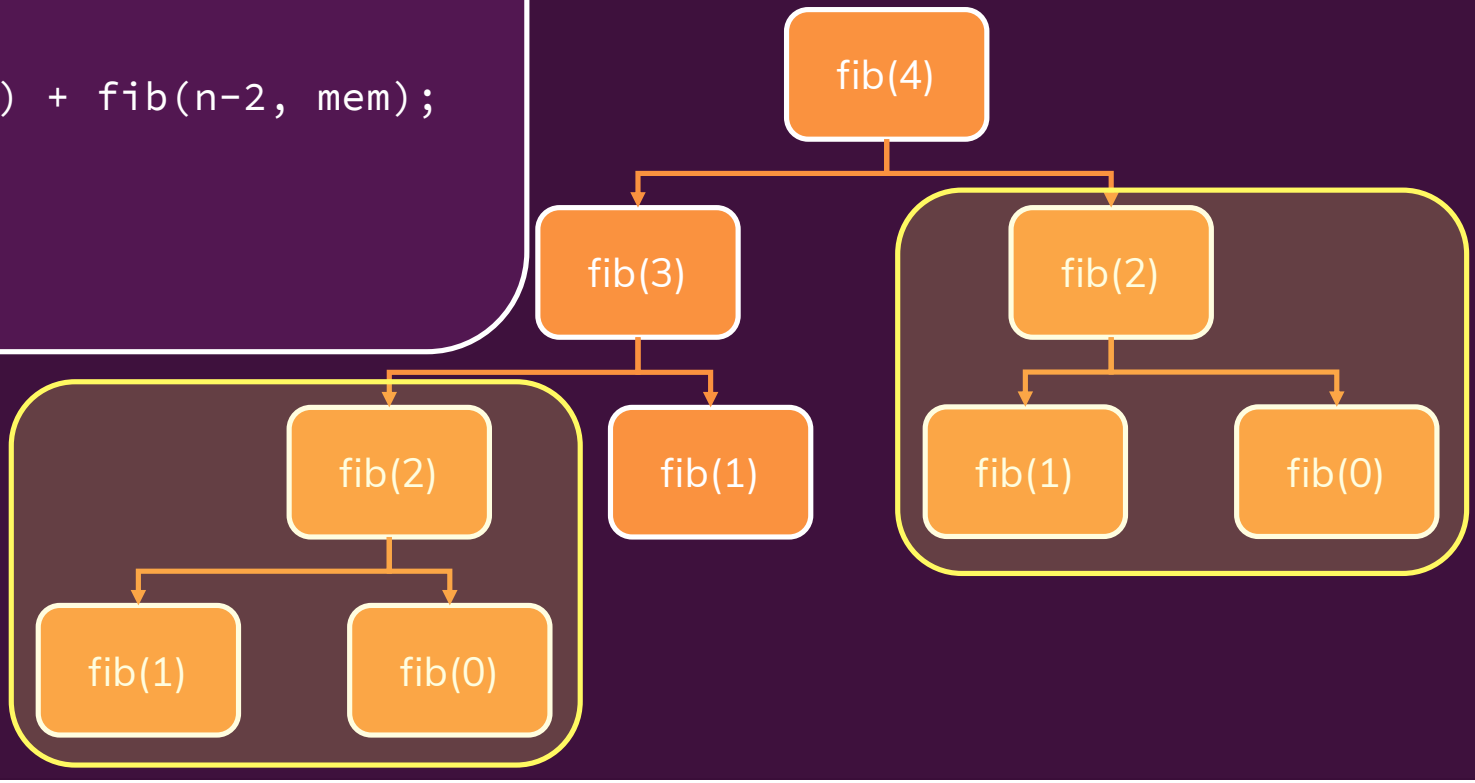
Intermediate results are stored and re-used

→ Dynamic Programming ←

ACADEMIND

Revisiting Our Solution

```
function fib(n, mem) {
  let res;
  if (mem[n]) return mem[n];
  if (n == 0 || n == 1) {
    res = 1;
  } else {
    res = fib(n-1, mem) + fib(n-2, mem);
  }
  mem[n] = res;
  return res;
}
```



ACADEMIND

Or: Use the “Bottom-Up Approach”

Memo

Build it up “from the bottom”

```
function fib(n) {
  const numbers = [1, 1];
  for (let i = 2; i < n + 1; i++) {
    numbers.push(numbers[i - 2] + numbers[i - 1]);
  }
  return numbers[n];
}
```