Recursive Function Call Tracing

Supplement to Recursion lecture

Tracing the Function Calls

Start with the original function call.

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] +  addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

Call 1

```
cards: [5, 2, 7, 3]                    addCards([5, 2, 7, 3])
```

Tracing the Function Calls

Start with the original function call.

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

Call 1

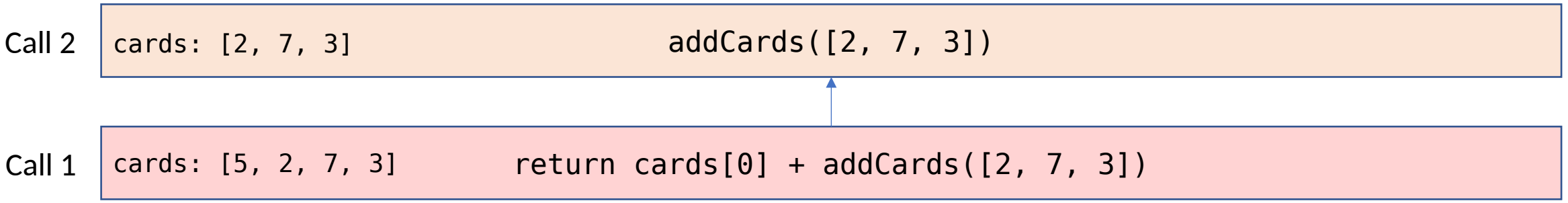
```
cards: [5, 2, 7, 3]        return cards[0] + addCards([2, 7, 3])
```

Tracing the Function Calls

We go to the recursive case and call `addCards` again on the smaller problem, making another function call.

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

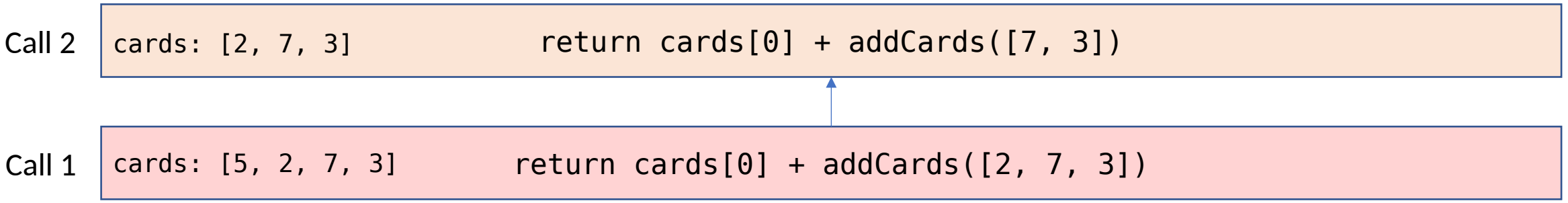


Tracing the Function Calls

When we run through `addCards` a second time, there's a **new local state**. `cards` is now `[2, 7, 3]`. The smaller problem is now `[7, 3]`.

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

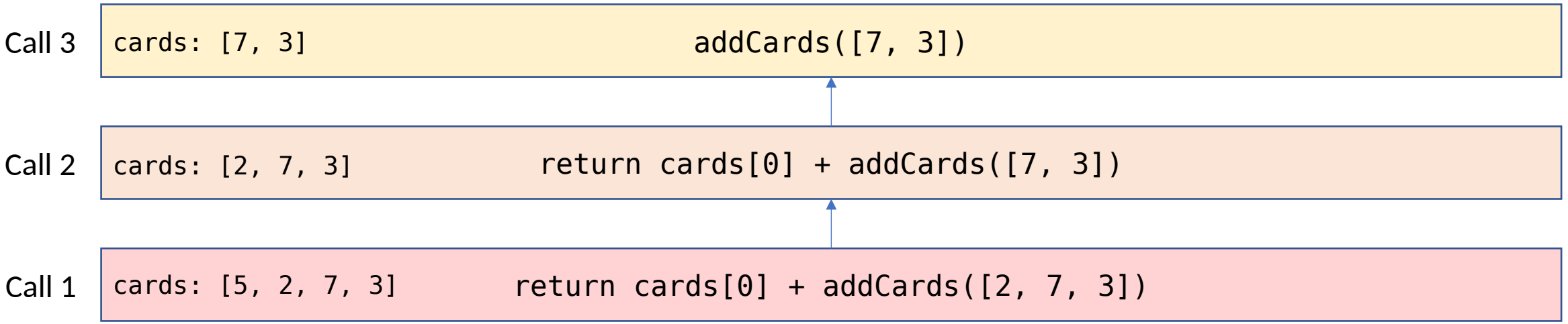


Tracing the Function Calls

Call `addCards` again, this time on `[7, 3]`. Note that the function call tracing helps us keep track of **all** previous calls.

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

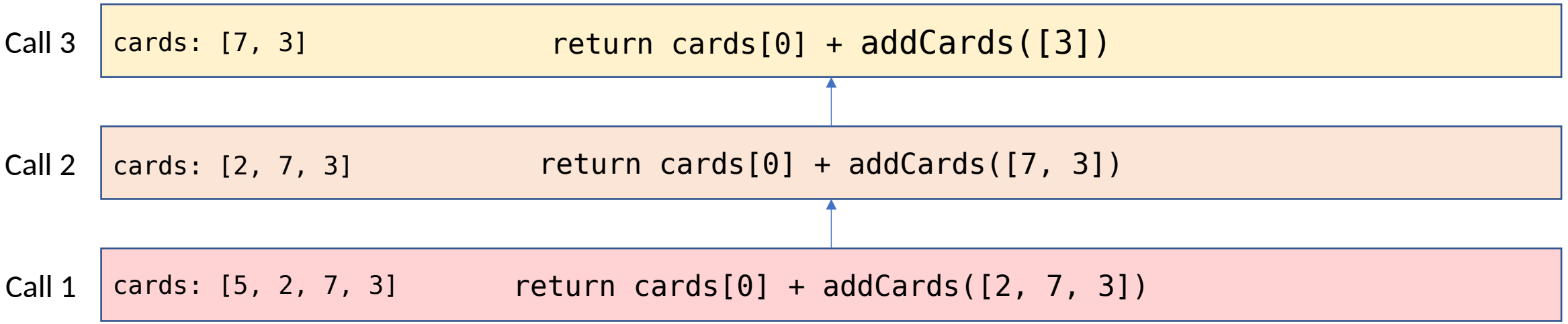


Tracing the Function Calls

Call `addCards` again, this time on `[7, 3]`. Note that the function call tracing helps us keep track of **all** previous calls.

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

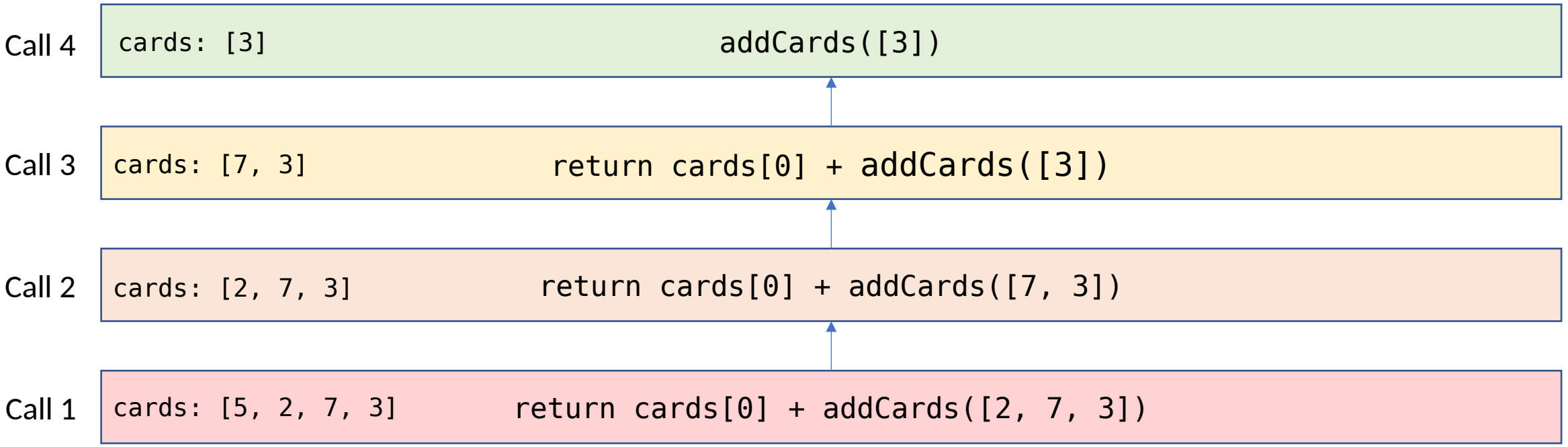


Tracing the Function Calls

Now we run the function with `cards` set to `[3]`. The smaller problem is `[]`; we call the function again.

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

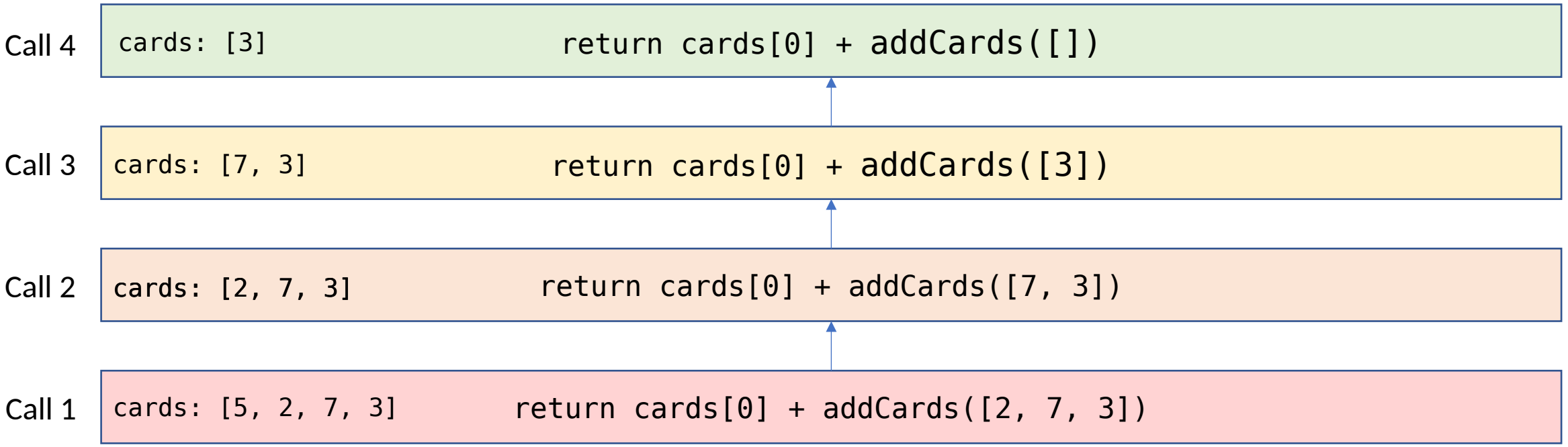


Tracing the Function Calls

Now we run the function with `cards` set to `[3]`. The smaller problem is `[]`; we call the function again.

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

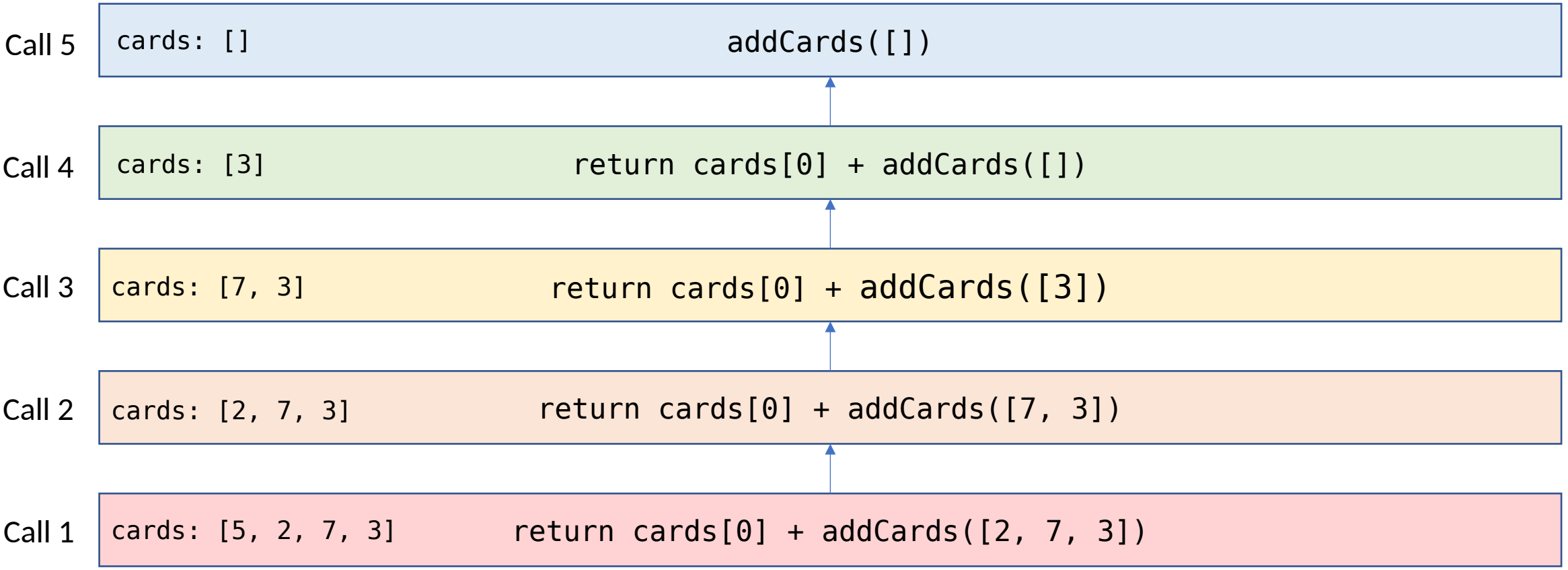


Tracing the Function Calls

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

Now we finally reach the base case. `addCards([])` returns `0` immediately, so `0` takes the place of the function call in the previous bookmark (in Call #4).

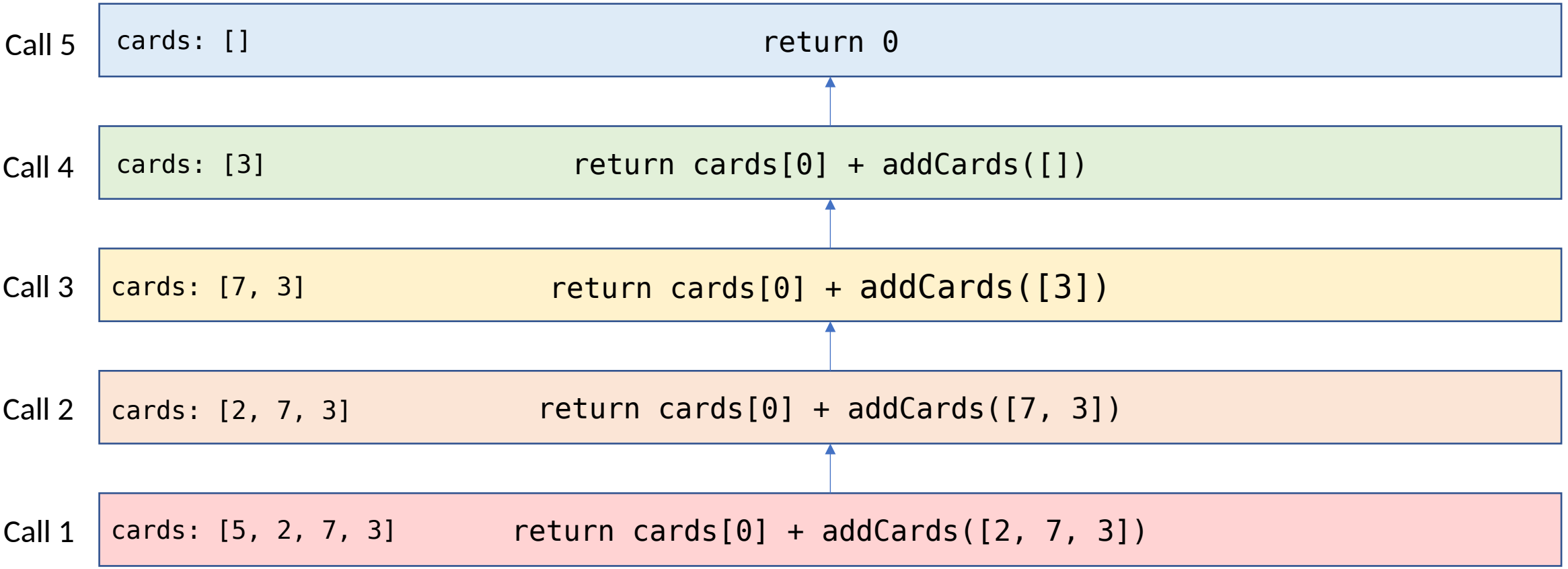


Tracing the Function Calls

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

Now we finally reach the base case. `addCards([])` returns `0` immediately, so `0` takes the place of the function call in the previous bookmark (in Call #4).

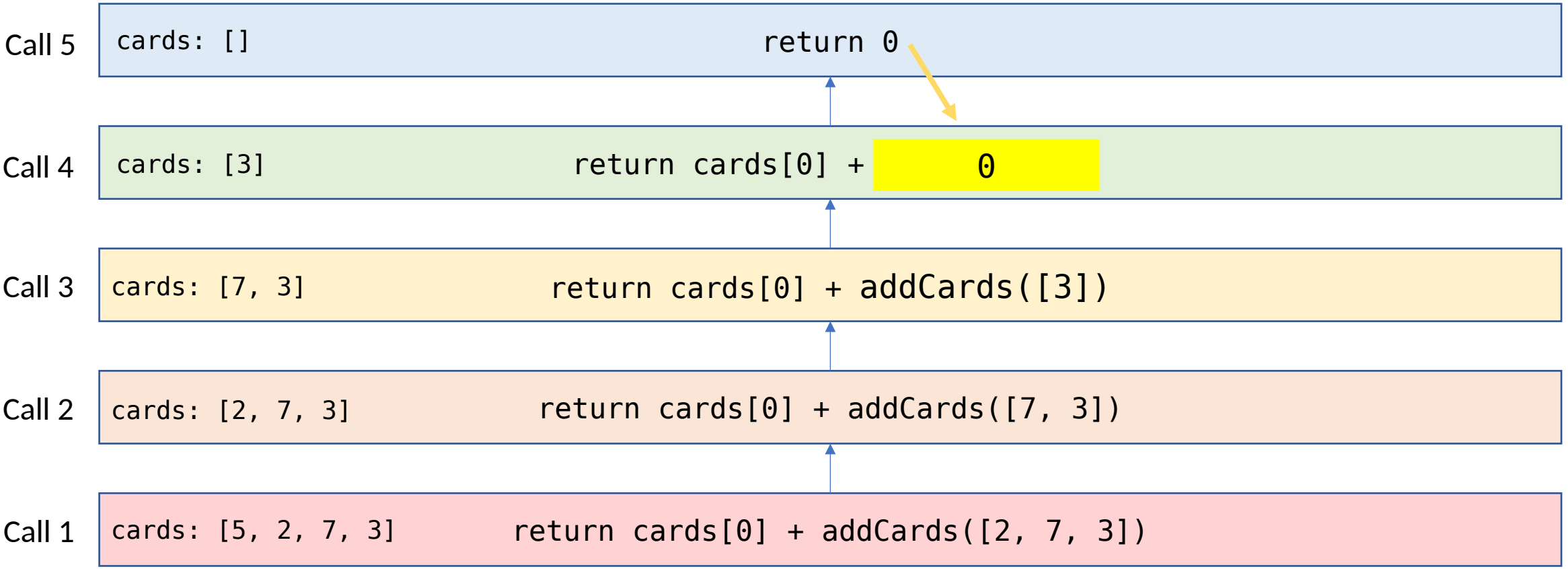


Tracing the Function Calls

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

Now we finally reach the base case. `addCards([])` returns `0` immediately, so `0` takes the place of the function call in the previous bookmark (in Call #4).

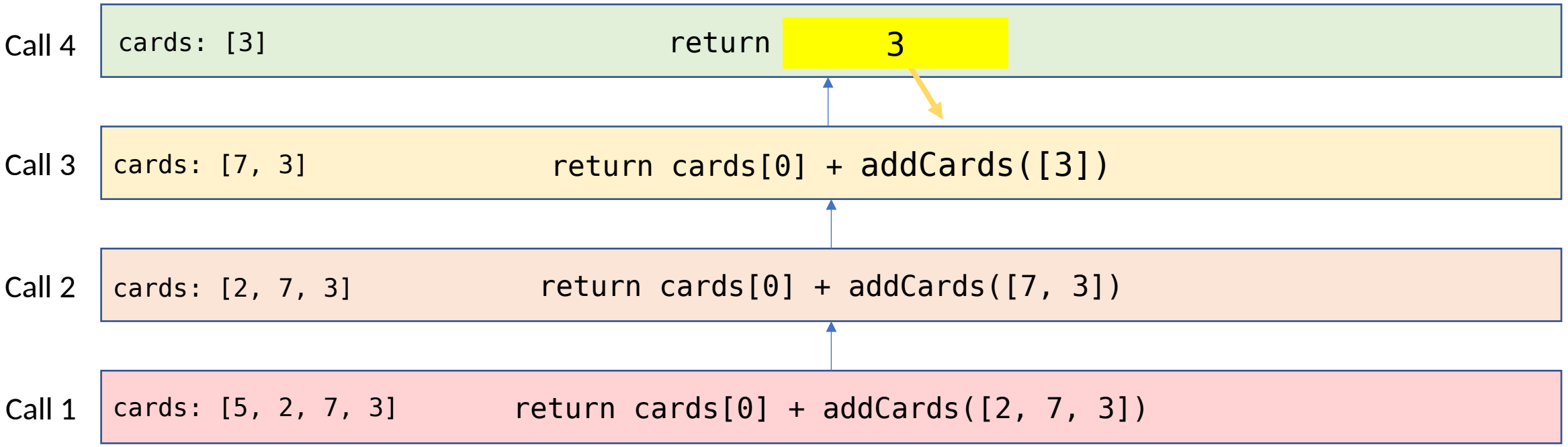


Tracing the Function Calls

Add `3 + 0` to get `3`; this can be sent back as the returned value to the previous level of the function calls (in Call #3).

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

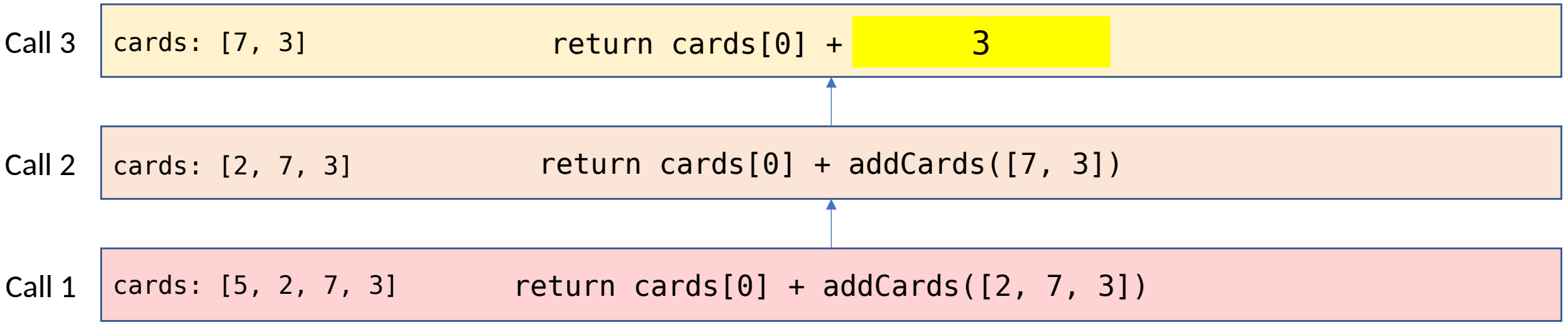


Tracing the Function Calls

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

At this level, `cards` is `[7, 3]` and the previous call returned `3`.
`7 + 3` gives us a return value of `10`.

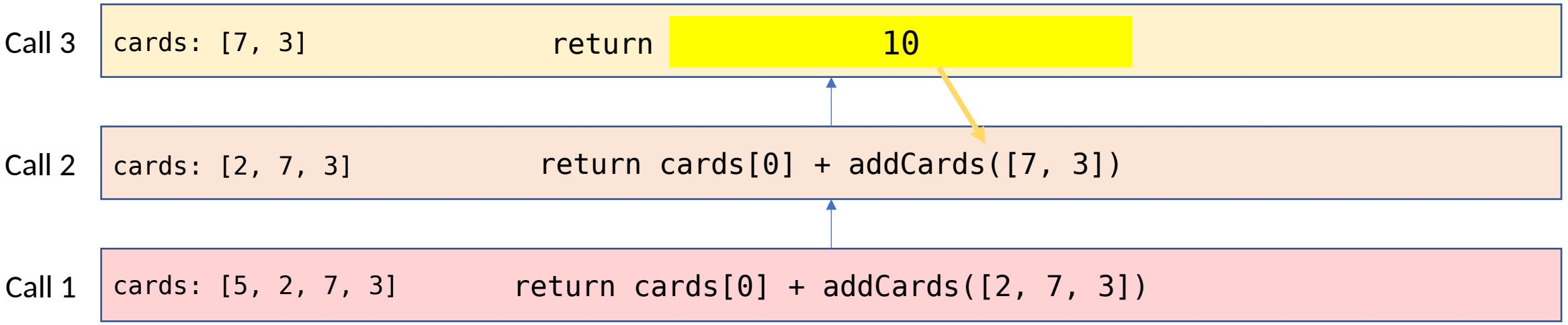


Tracing the Function Calls

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

At this level, `cards` is `[7, 3]` and the previous call returned `3`.
`7 + 3` gives us a return value of `10`.

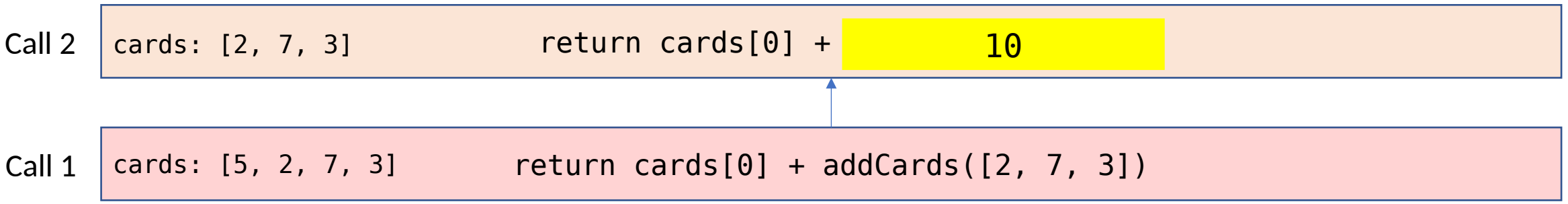


Tracing the Function Calls

At this level, `cards` is `[2, 7, 3]` and the previous call returned `10`.
`2 + 10` gives us a return value of `12`.

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

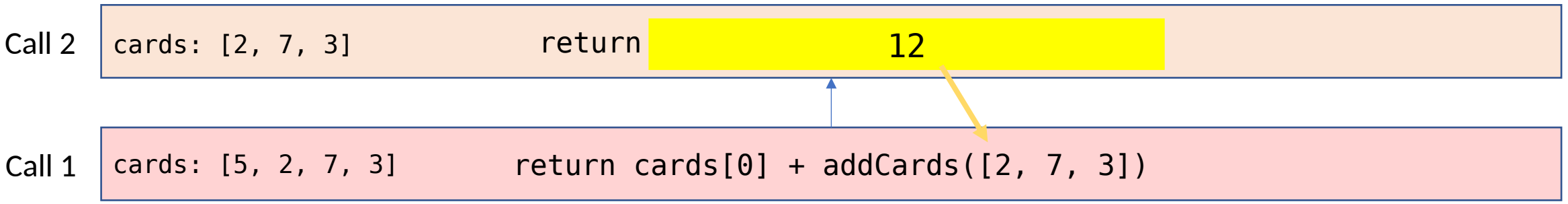


Tracing the Function Calls

At this level, `cards` is `[2, 7, 3]` and the previous call returned `10`.
`2 + 10` gives us a return value of `12`.

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```



Tracing the Function Calls

At this level, `cards` is `[5, 2, 7, 3]` and the previous call returned `12`.
`5 + 12` gives us a return value of `17`.

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        return cards[0] + addCards(smallerProblem)

addCards([5, 2, 7, 3])
```

Call 1 | `cards: [5, 2, 7, 3]` | `return cards[0] +` | `12`

Tracing the Function Calls

At this level, `cards` is `[7, 3]` and `smallerResult` is `3`. `7 + 3` gives us a returned value of `10`.

```
def addCards(cards):
    if cards == [ ]:
        return 0
    else:
        smallerProblem = cards[1:]
        smallerResult = addCards(smallerProblem)
        return cards[0] + smallerResult

addCards([5, 2, 7, 3])
```

Call 1 | `cards: [5, 2, 7, 3]` | `return` | 17