Recursive Function Call Stack

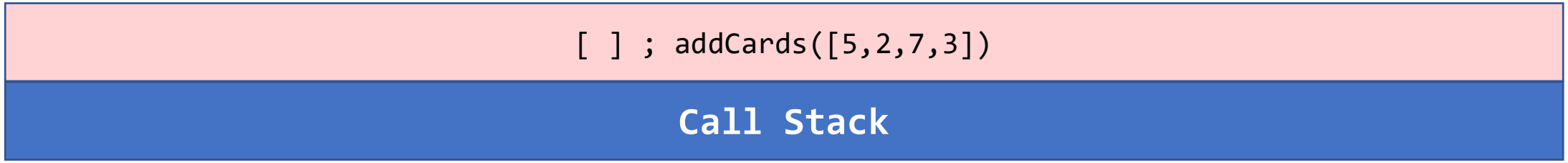
Supplement to Recursion lecture

Tracing the Call Stack

Start with the original function call at the bottom of the stack.

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

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



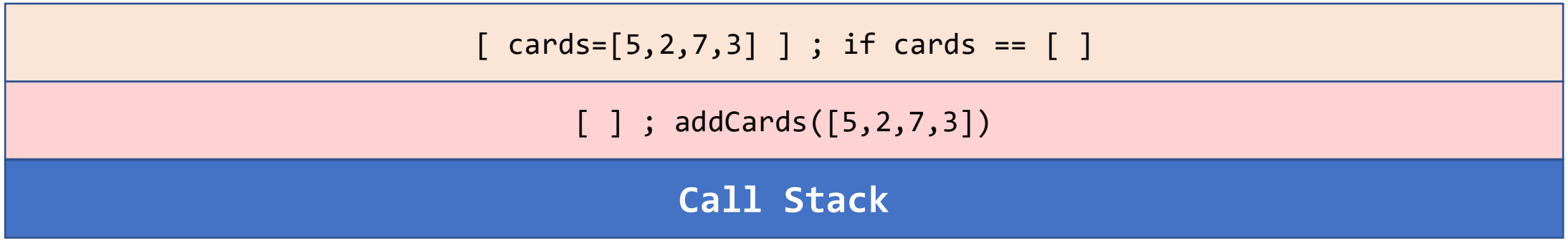
Tracing the Call Stack

The local cards variable is not an empty list, so go to the else case.

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

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

1st call



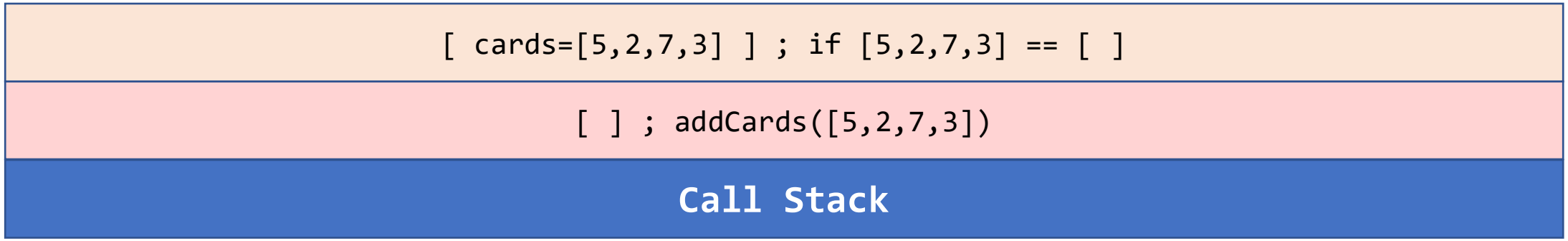
Tracing the Call Stack

The local cards variable is not an empty list, so go to the else case.

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

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

1st call



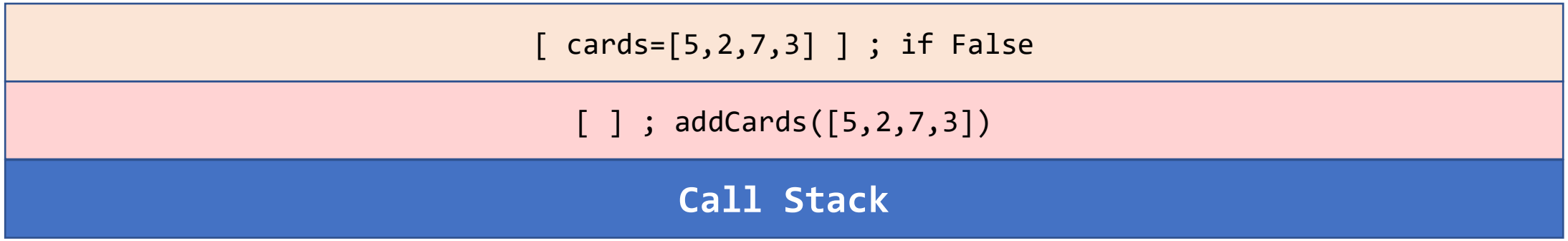
Tracing the Call Stack

The local cards variable is not an empty list, so go to the else case.

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

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

1st call



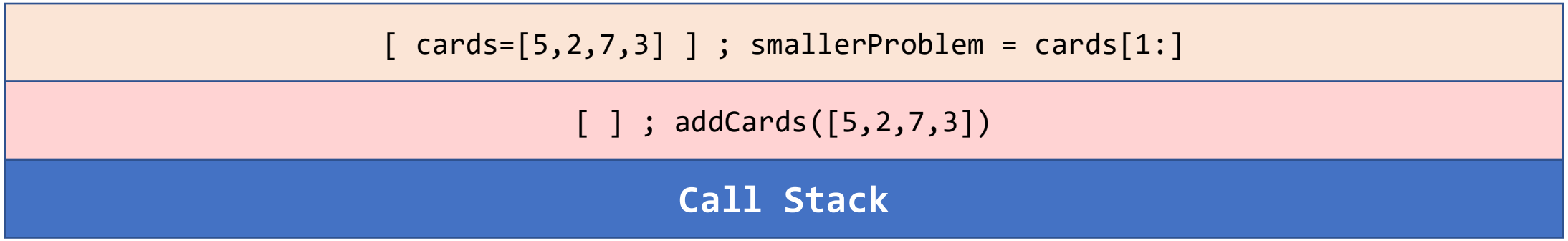
Tracing the Call Stack

Set a local var smallerProblem to all but the first card.

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

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

1st call



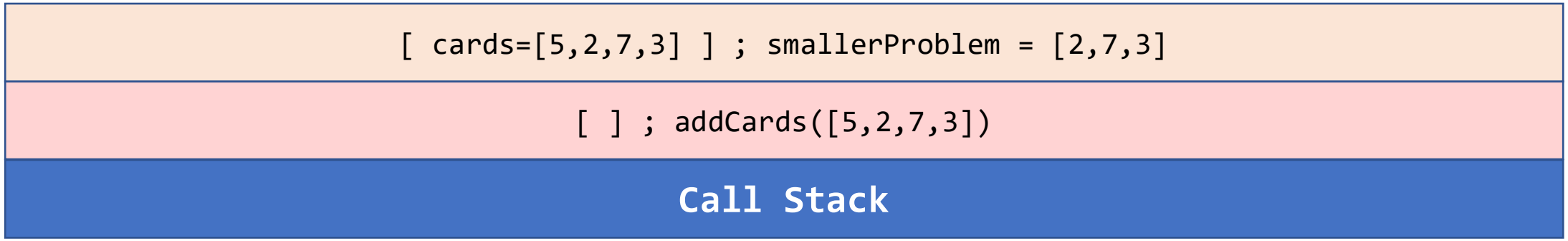
Tracing the Call Stack

Set a local var smallerProblem to all but the first card.

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

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

1st call



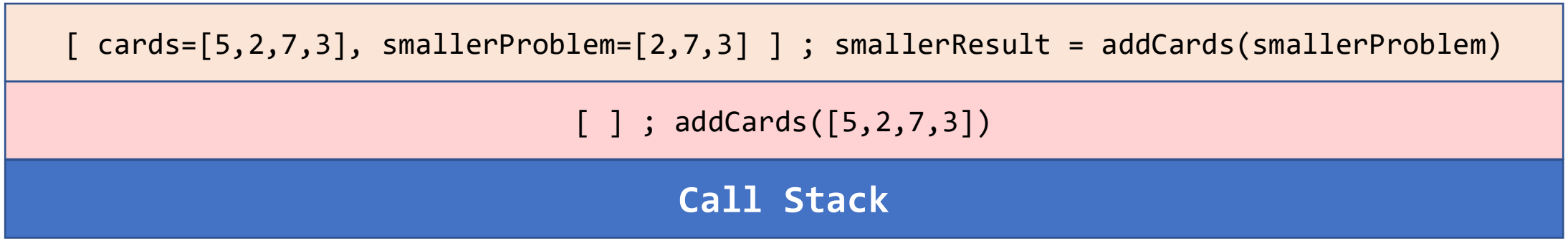
Tracing the Call Stack

Now we need to make the **recursive call**. Look up the value in local var smallerProblem, then add a new level to the stack.

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

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

1st call



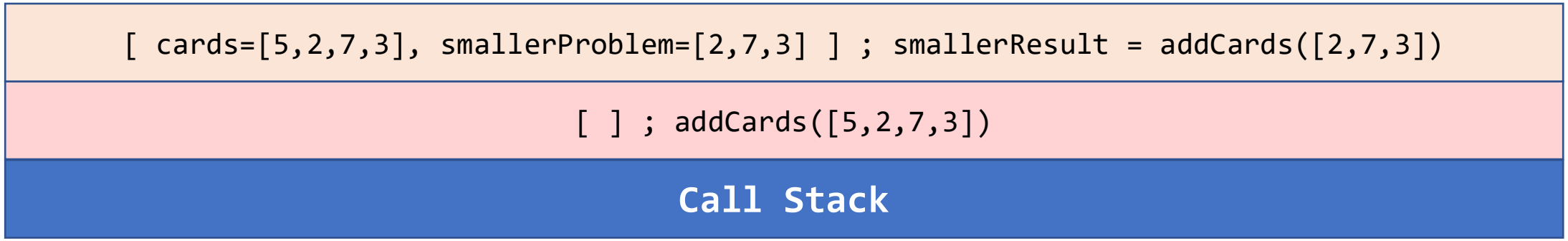
Tracing the Call Stack

Now we need to make the **recursive call**. Look up the value in local var smallerProblem, then add a new level to the stack.

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

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

1st call



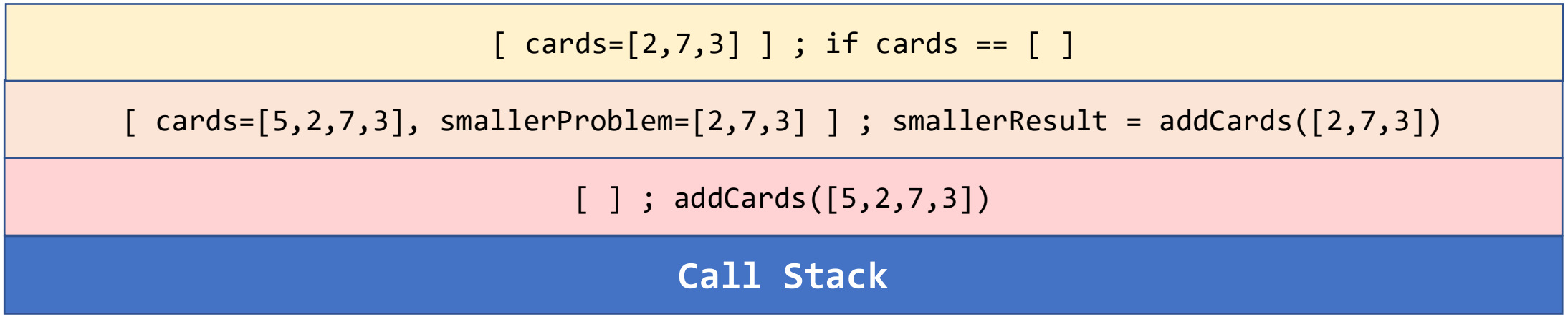
Tracing the Call Stack

The local cards variable is not an empty list, so go to the else case.

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

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

2nd call



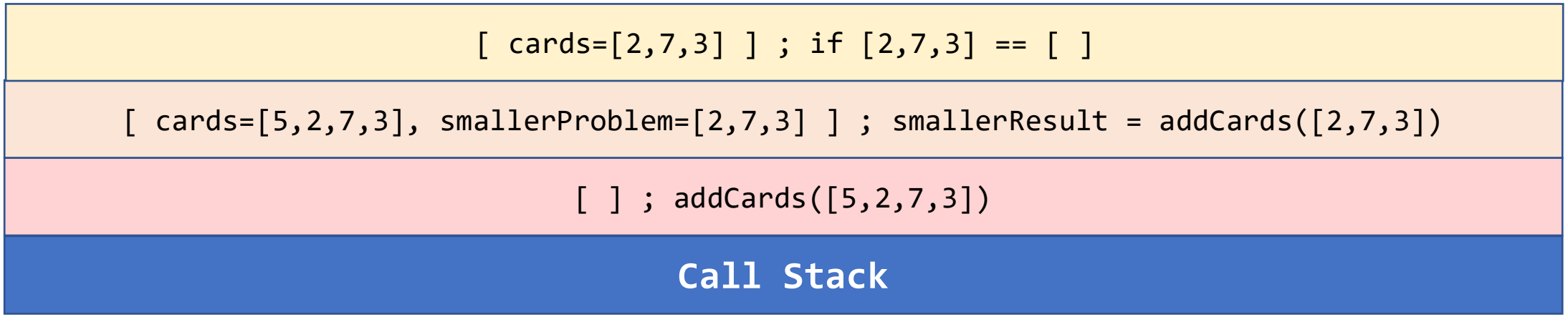
Tracing the Call Stack

The local cards variable is not an empty list, so go to the else case.

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

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

2nd call



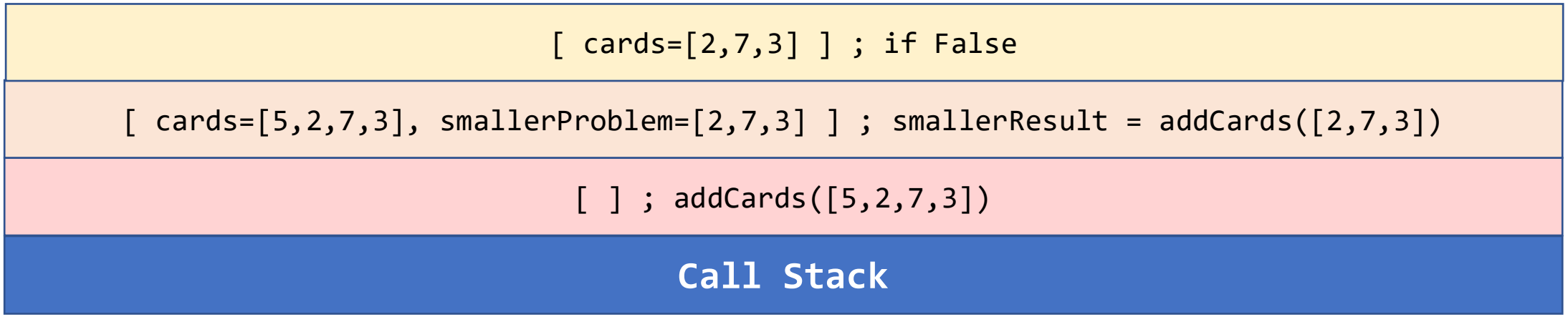
Tracing the Call Stack

The local cards variable is not an empty list, so go to the else case.

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

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

2nd call



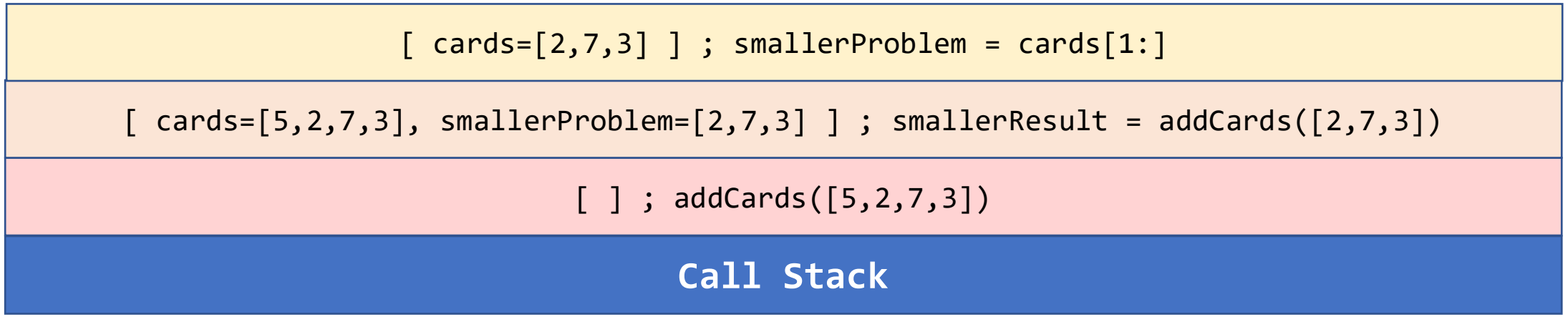
Tracing the Call Stack

Set a local var smallerProblem to all but the first card.

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

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

2nd call



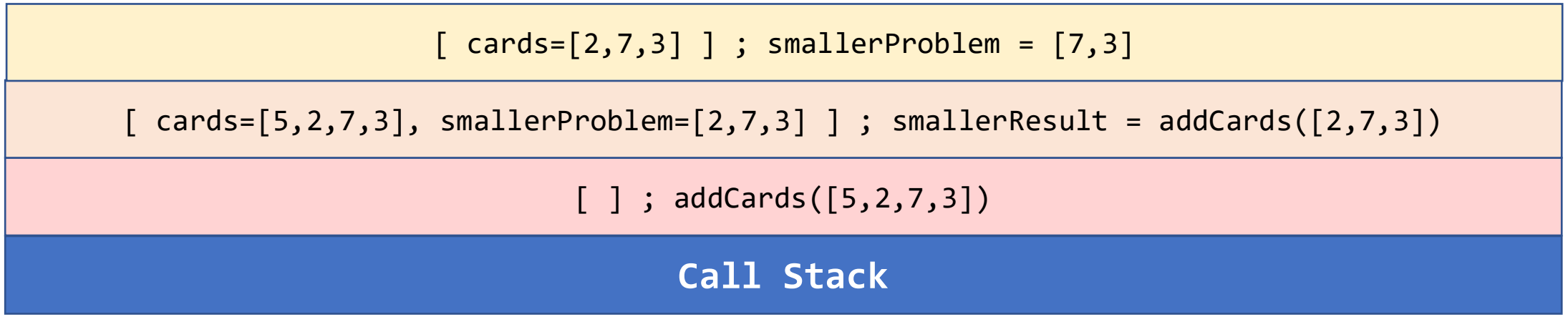
Tracing the Call Stack

Set a local var smallerProblem to all but the first card.

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

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

2nd call



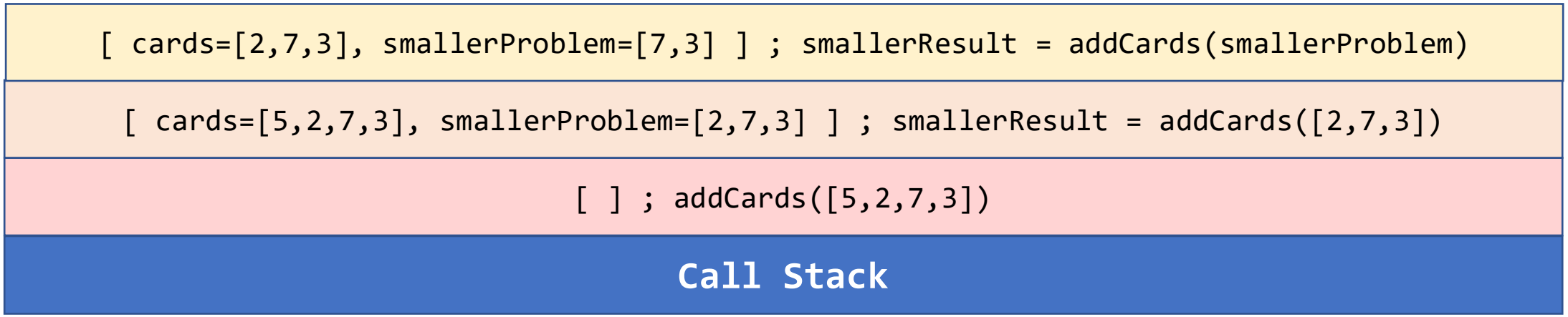
Tracing the Call Stack

Now we need to make the **recursive call**. Look up the value in local var smallerProblem, then add a new level to the stack.

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

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

2nd call



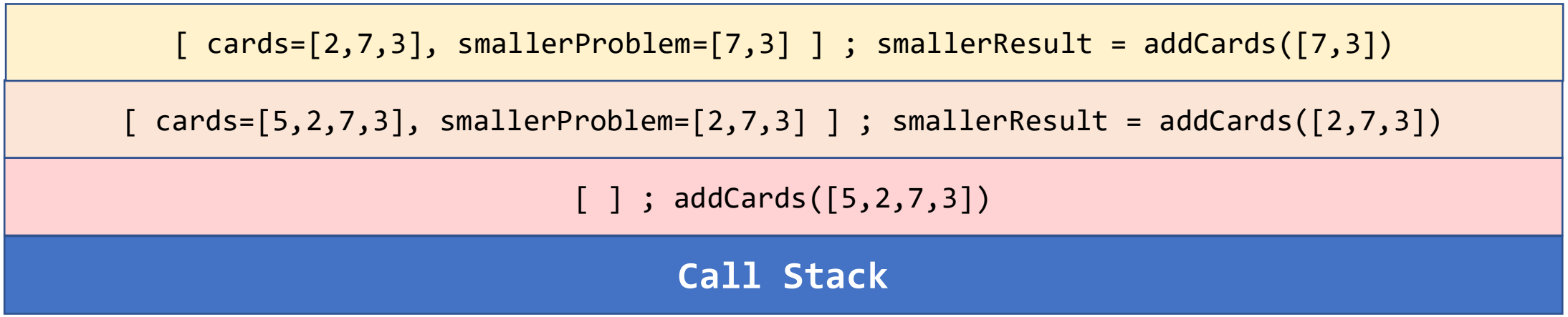
Tracing the Call Stack

Now we need to make the **recursive call**. Look up the value in local var smallerProblem, then add a new level to the stack.

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

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

2nd call



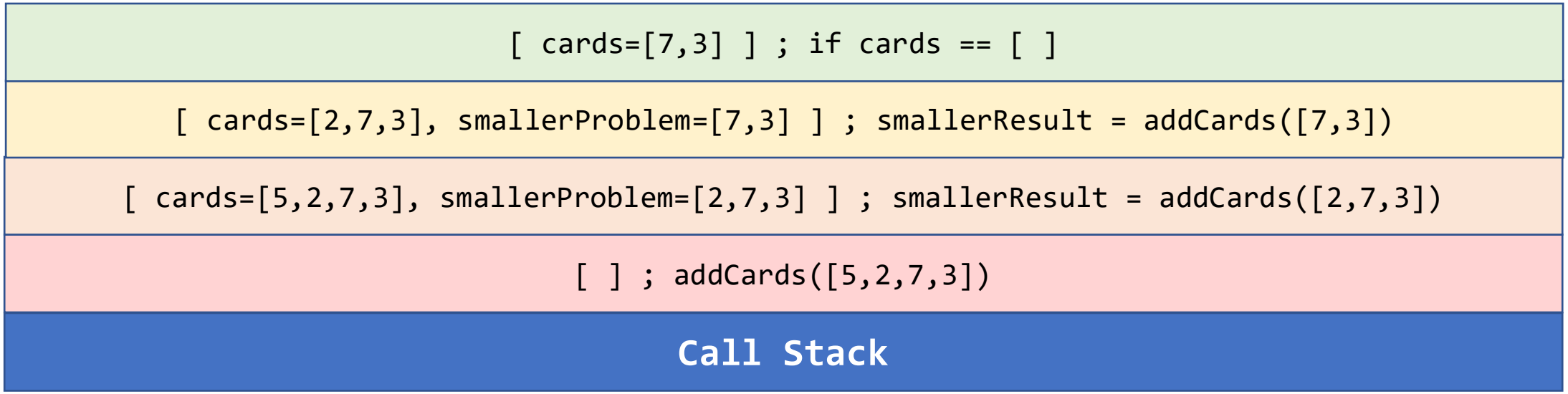
Tracing the Call Stack

The local cards variable is not an empty list, so go to the else case.

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

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

3rd call



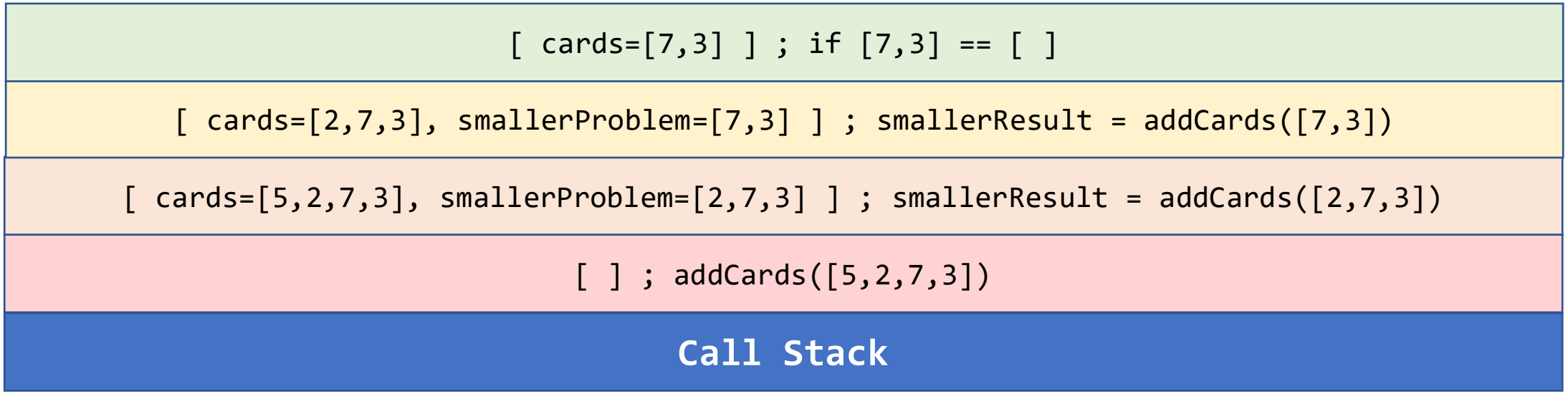
Tracing the Call Stack

The local cards variable is not an empty list, so go to the else case.

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

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

3rd call



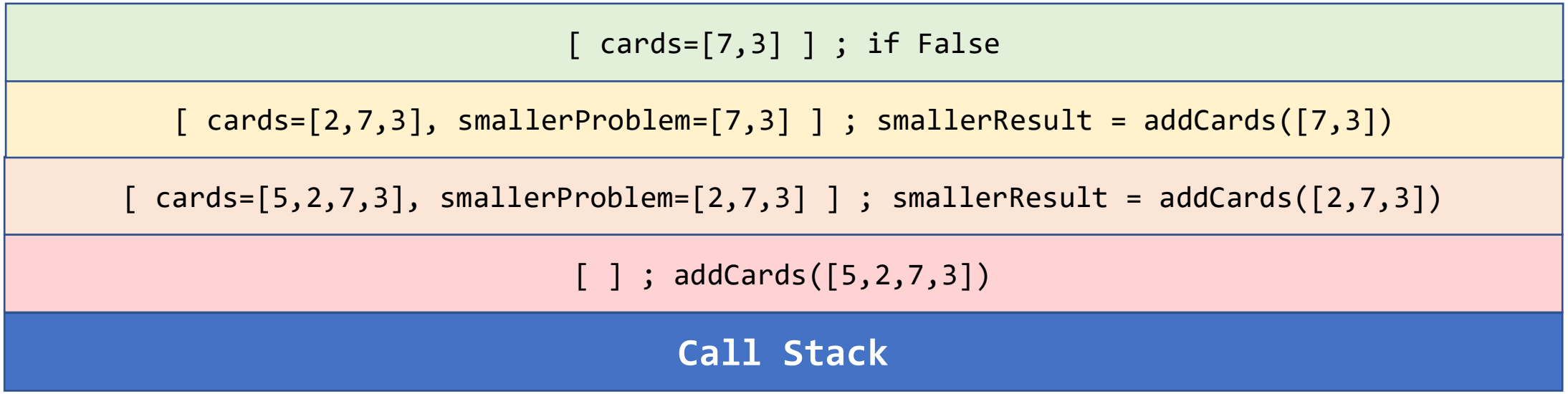
Tracing the Call Stack

The local cards variable is not an empty list, so go to the else case.

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

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

3rd call



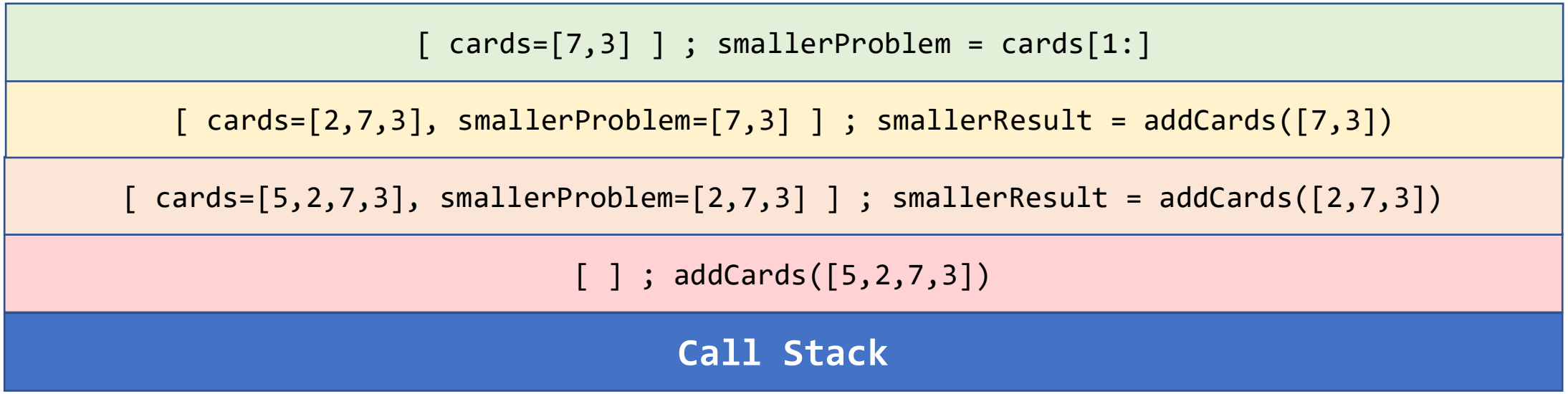
Tracing the Call Stack

Set a local var smallerProblem to all but the first card.

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

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

3rd call



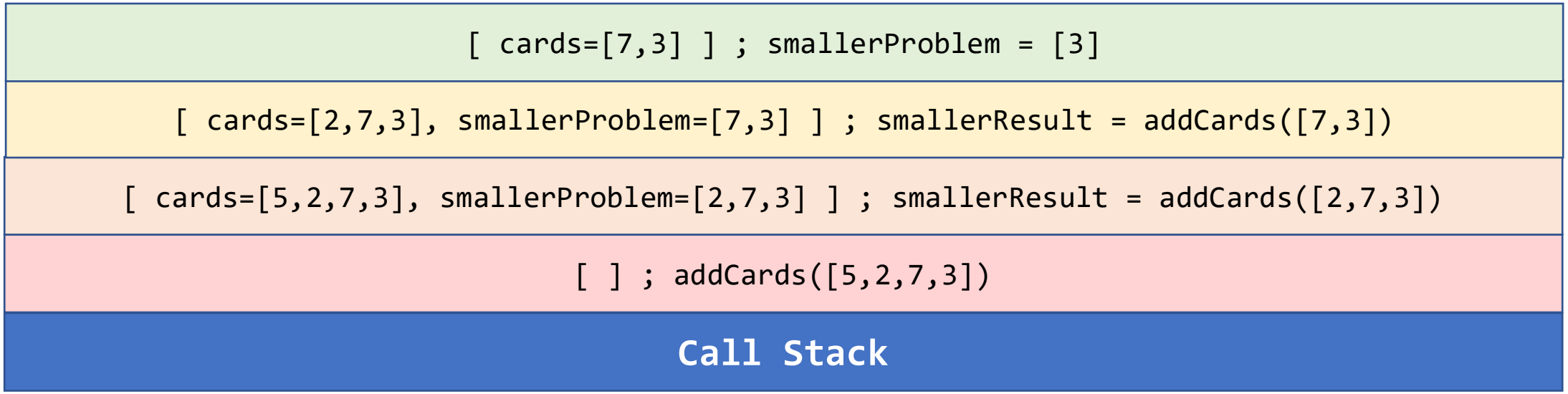
Tracing the Call Stack

Set a local var smallerProblem to all but the first card.

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

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

3rd call



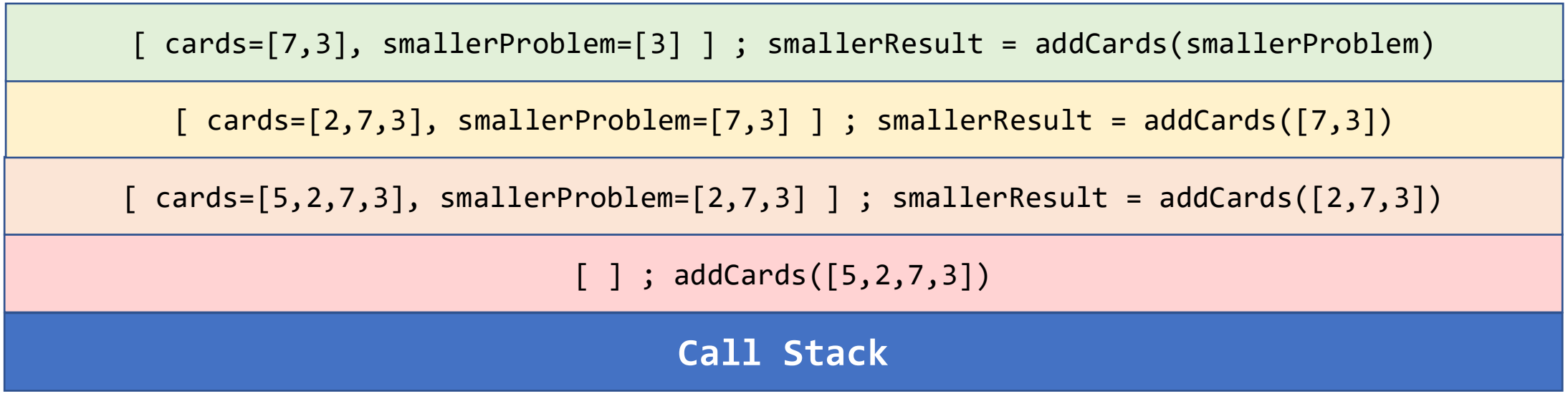
Tracing the Call Stack

Now we need to make the **recursive call**. Look up the value in local var smallerProblem, then add a new level to the stack.

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

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

3rd call



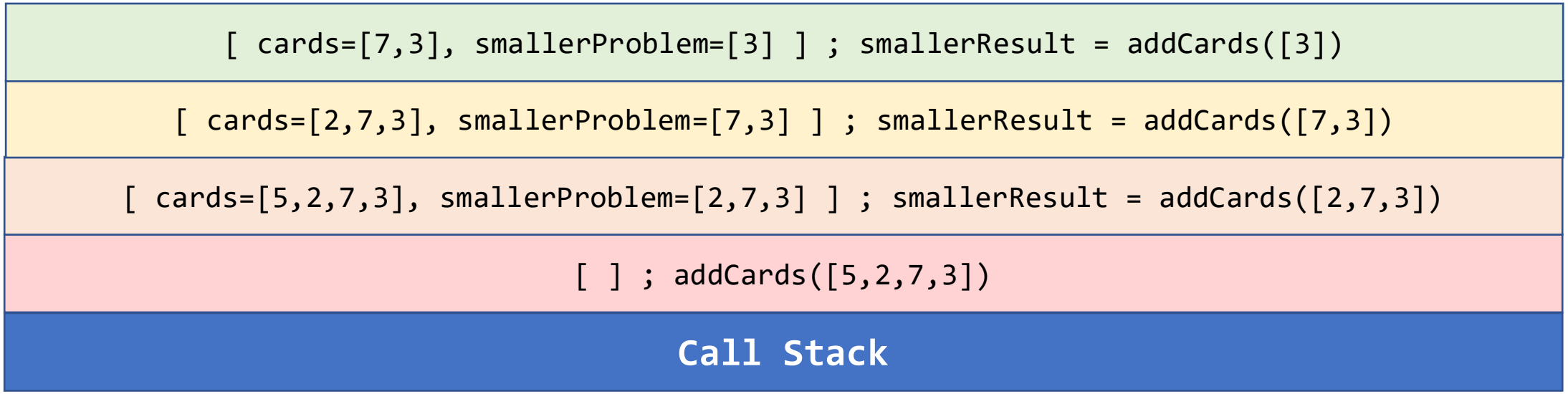
Tracing the Call Stack

Now we need to make the **recursive call**. Look up the value in local var smallerProblem, then add a new level to the stack.

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

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

3rd call



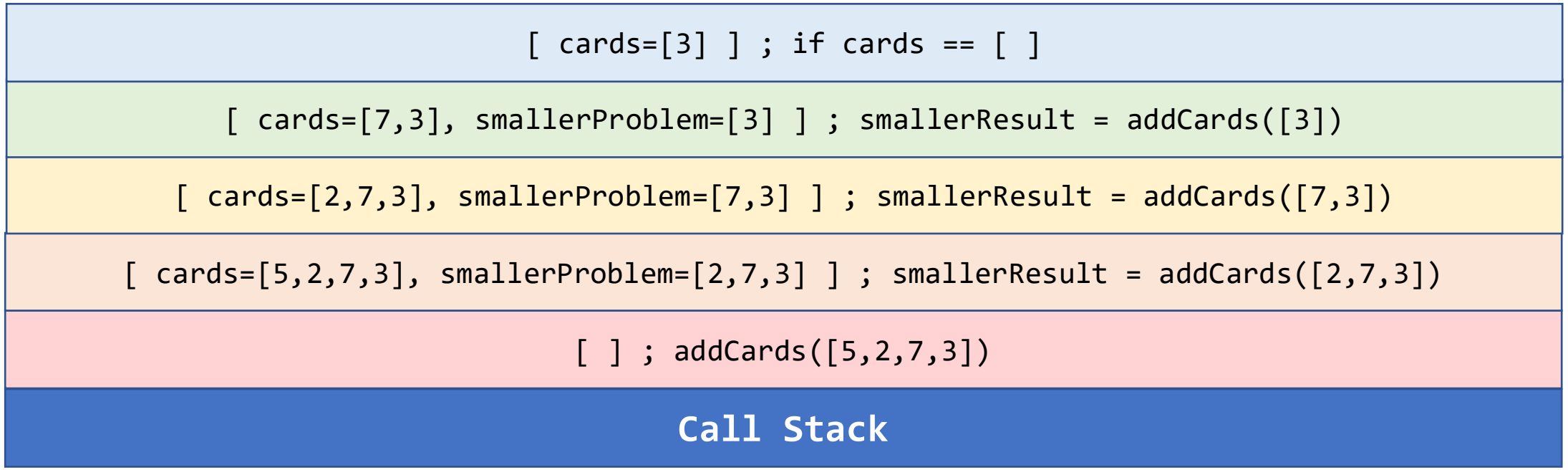
Tracing the Call Stack

The local cards variable is not an empty list, so go to the else case.

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

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

4th call



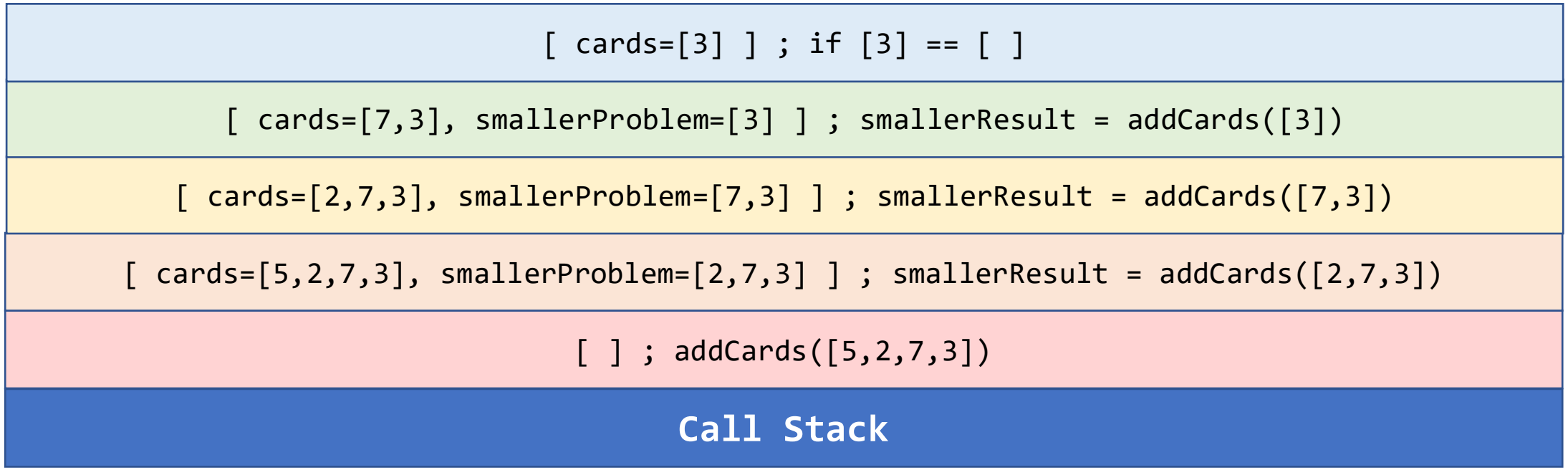
Tracing the Call Stack

The local cards variable is not an empty list, so go to the else case.

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

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

4th call



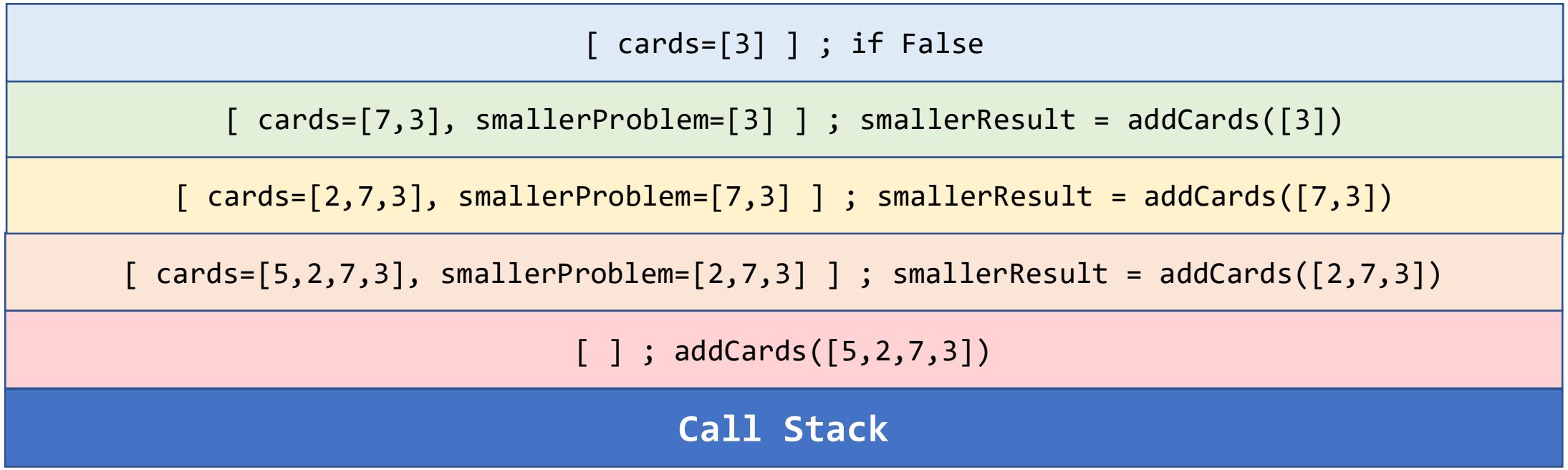
Tracing the Call Stack

The local cards variable is not an empty list, so go to the else case.

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

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

4th call



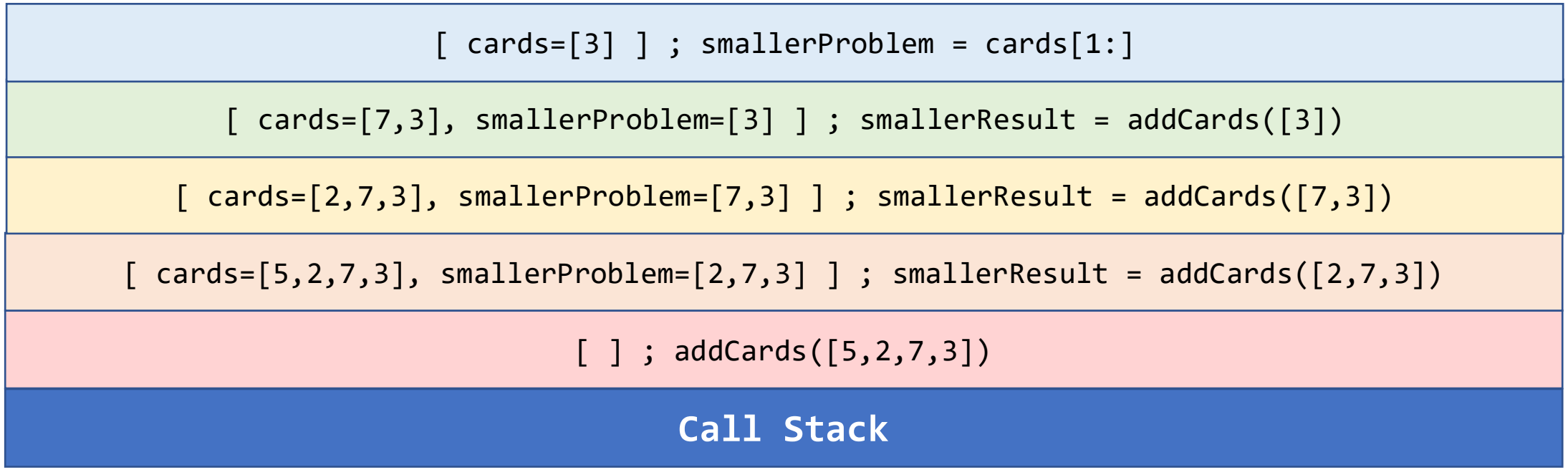
Tracing the Call Stack

Set a local var smallerProblem to all but the first card (an empty list as there are no remaining cards).

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

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

4th call



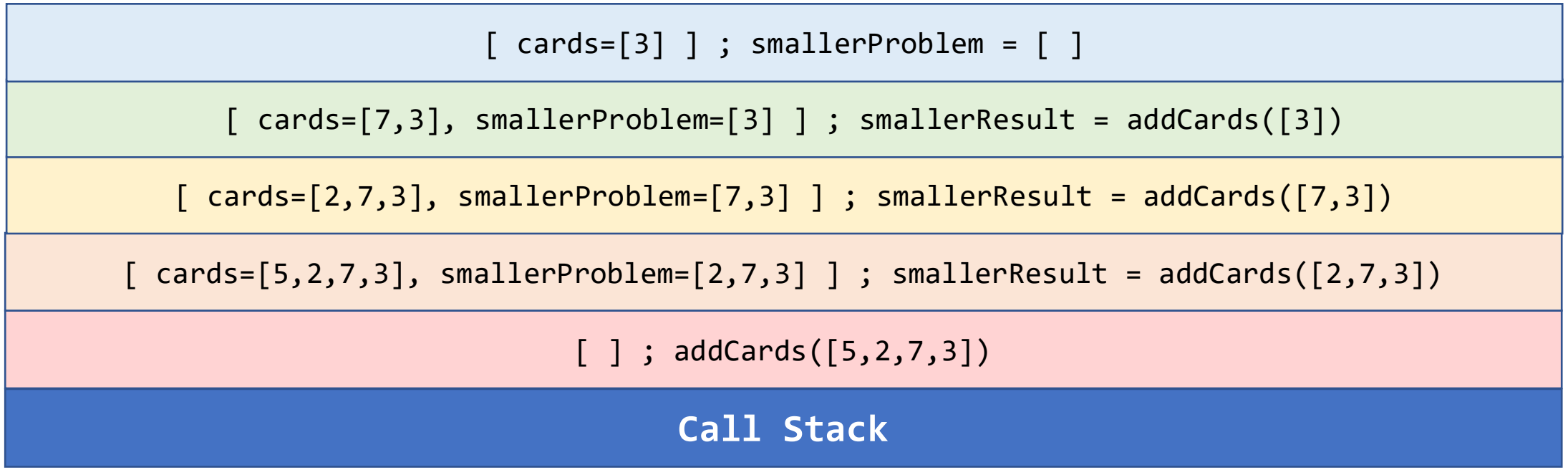
Tracing the Call Stack

Set a local var smallerProblem to all but the first card (an empty list as there are no remaining cards).

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

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

4th call



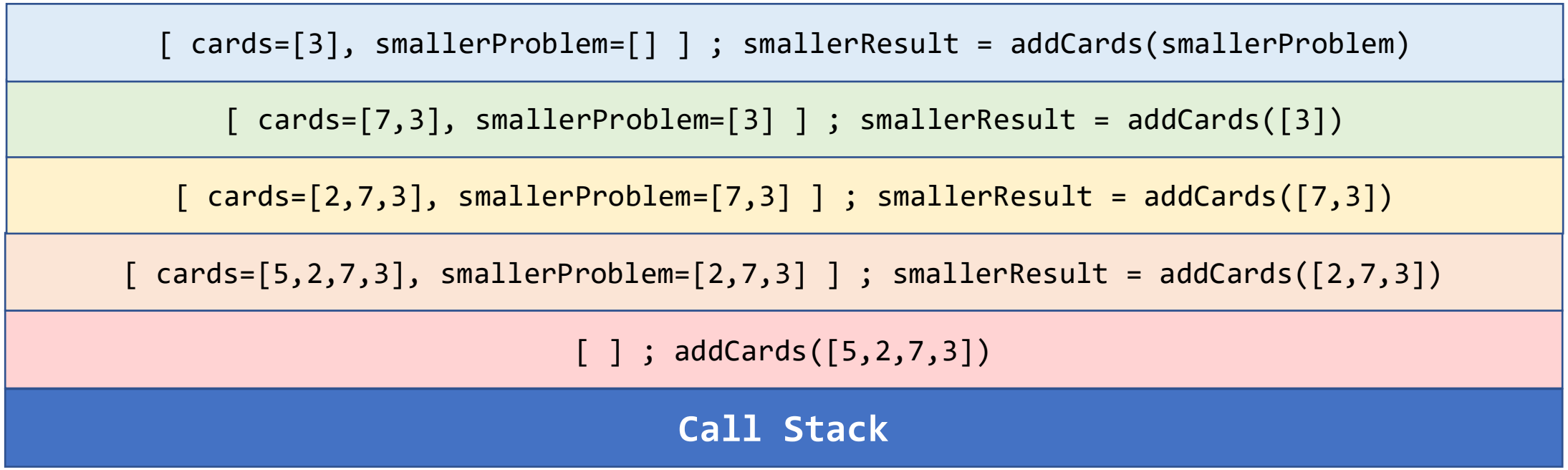
Tracing the Call Stack

Now we need to make the **recursive call**. Look up the value in local var smallerProblem, then add a new level to the stack.

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

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

4th call



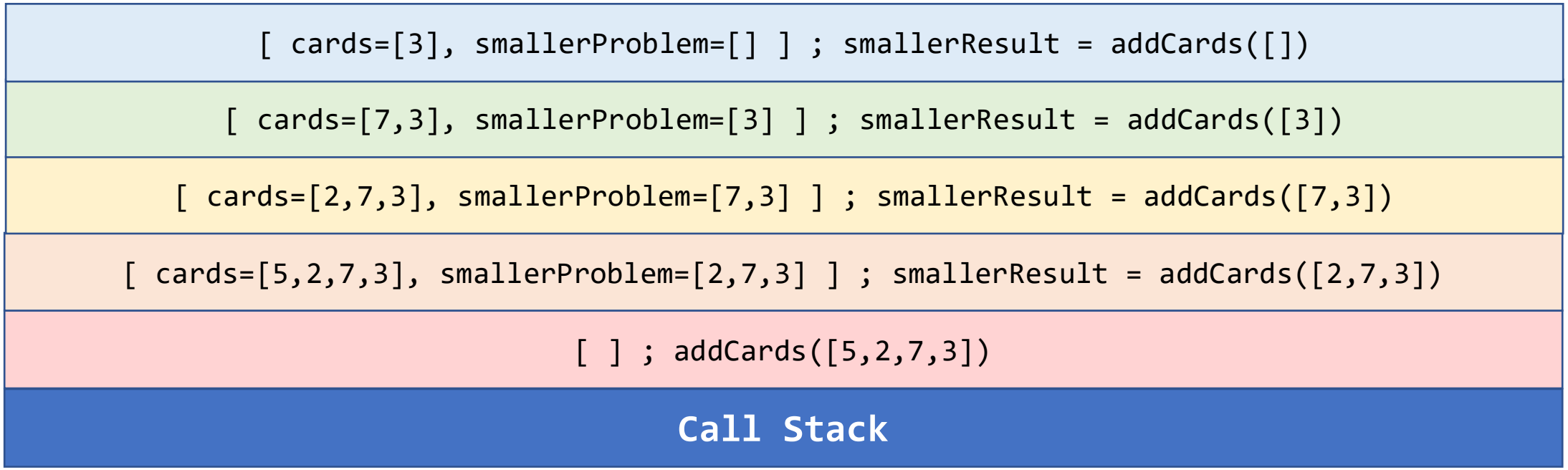
Tracing the Call Stack

Now we need to make the **recursive call**. Look up the value in local var smallerProblem, then add a new level to the stack.

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

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

4th call



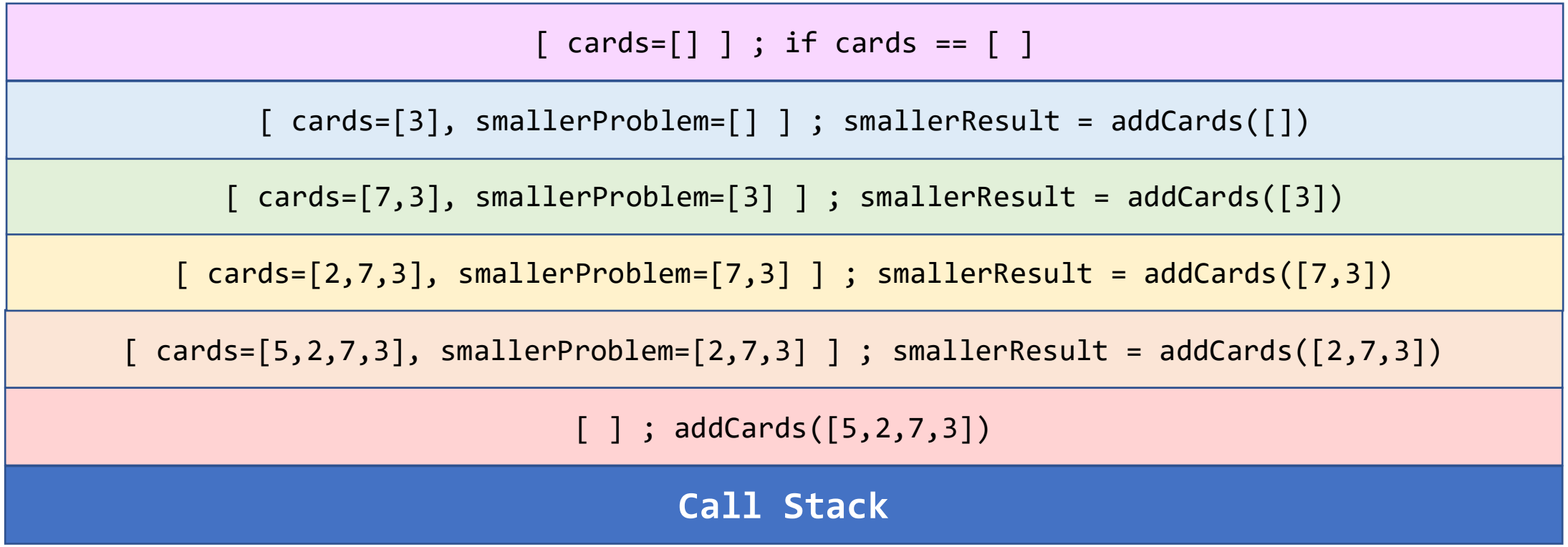
Tracing the Call Stack

Now the local cards variable is the empty list! Go to the base case.

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

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

5th call



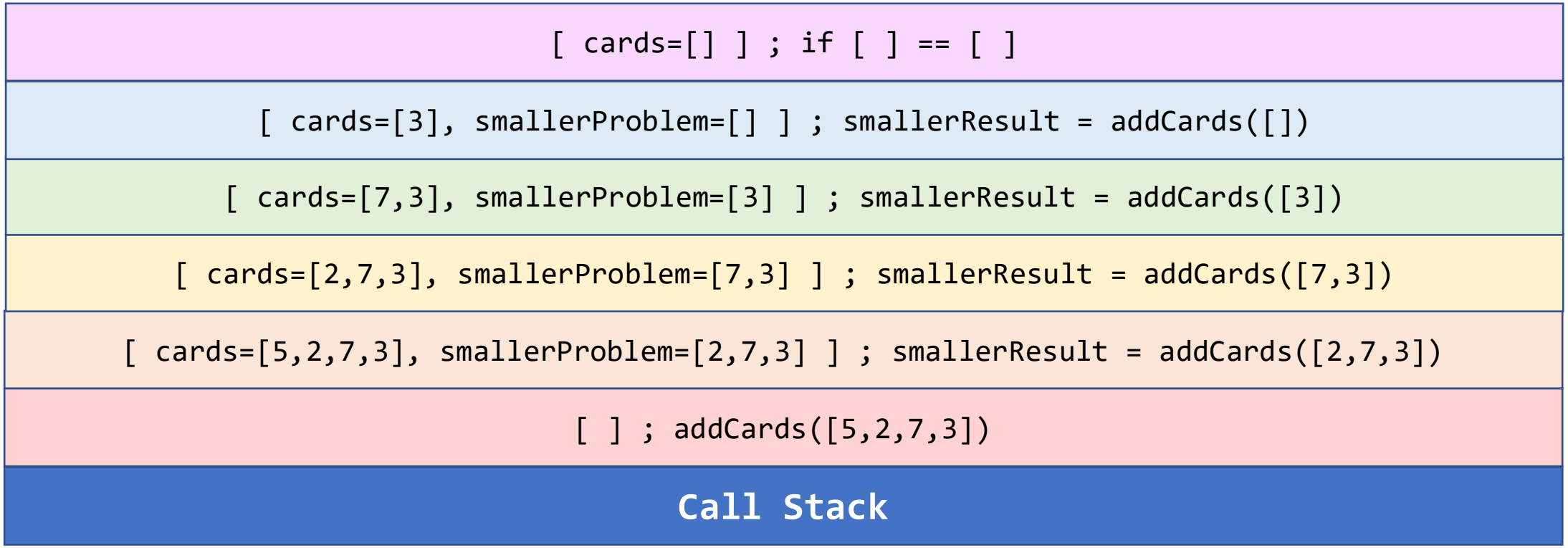
Tracing the Call Stack

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

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

5th call

Now the local cards variable is the empty list! Go to the base case.



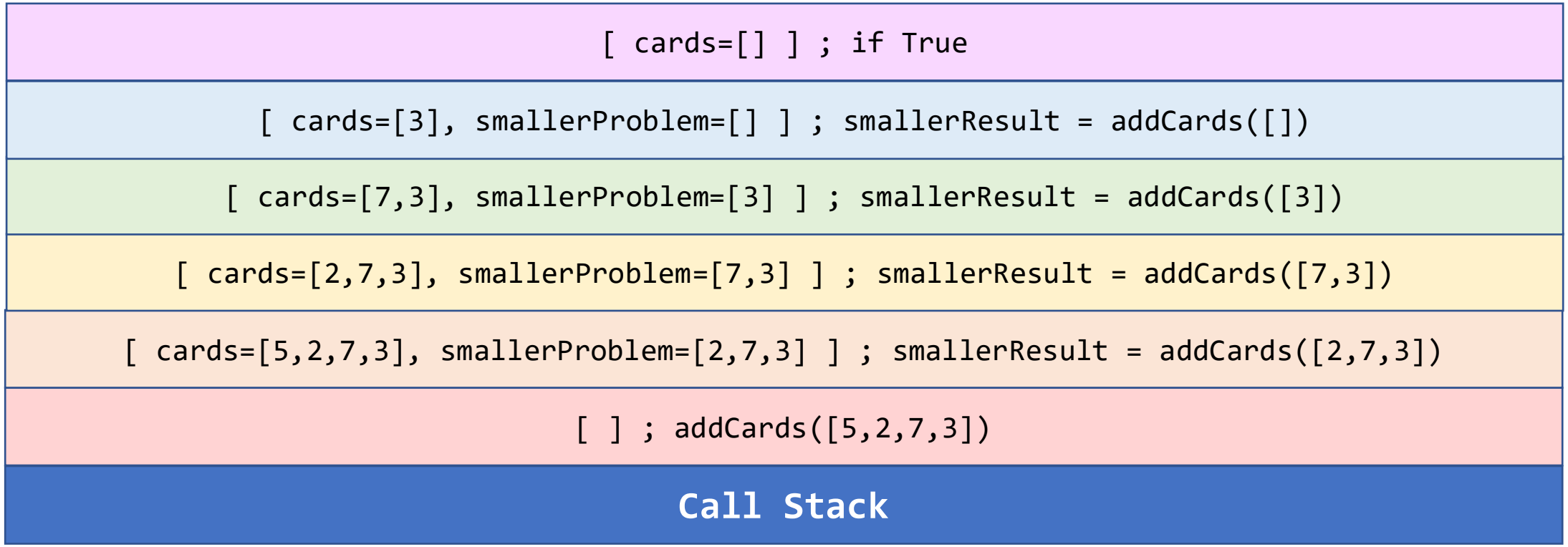
Tracing the Call Stack

Now the local cards variable is the empty list! Go to the base case.

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

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

5th call



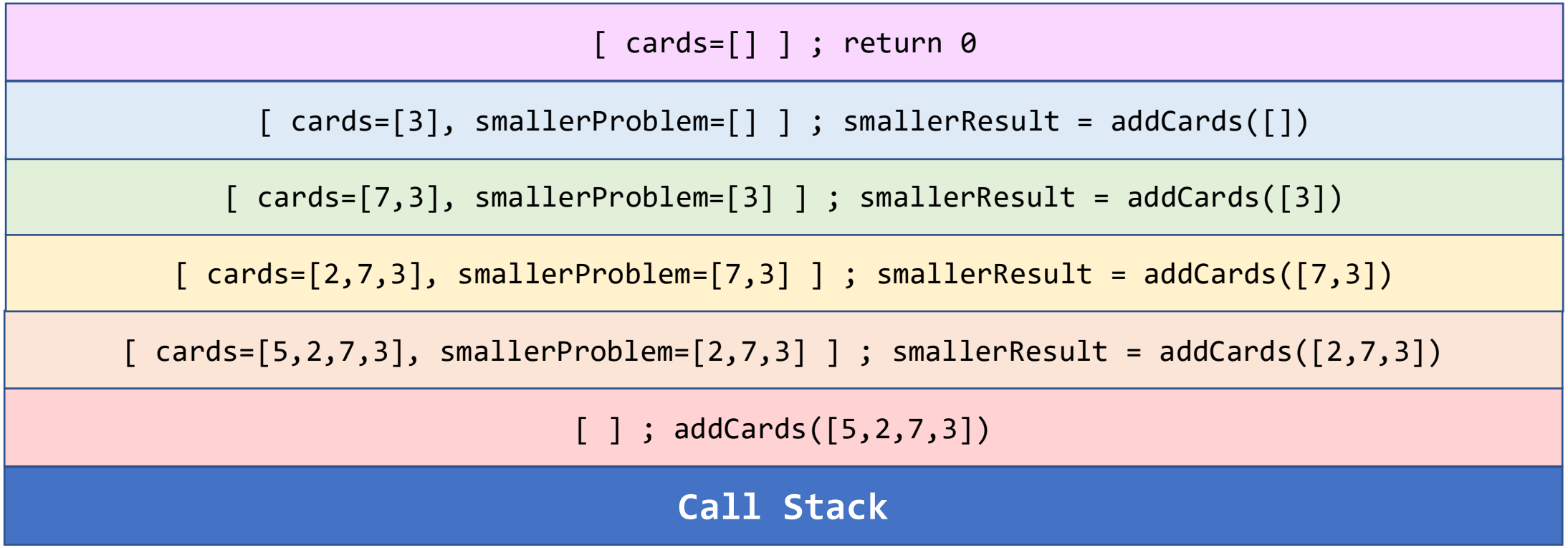
Tracing the Call Stack

We can immediately return 0 to the next layer of the stack- the **4th call**.

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

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

5th call



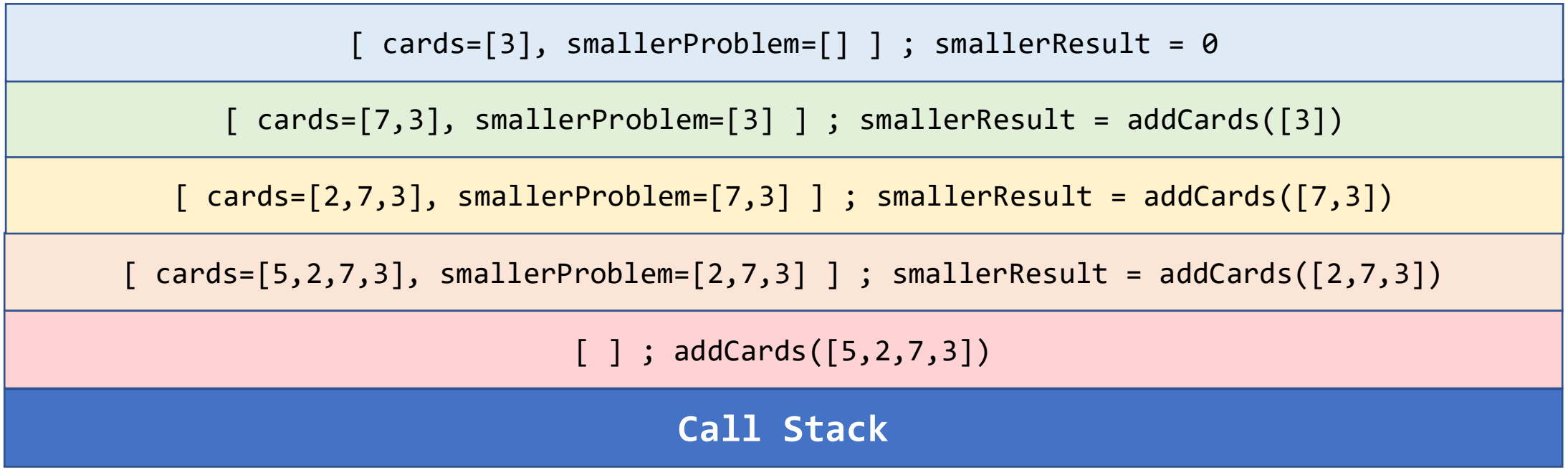
Tracing the Call Stack

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

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

4th call

Back in the 4th call, the returned value takes the place of the function call, and a new var is saved.



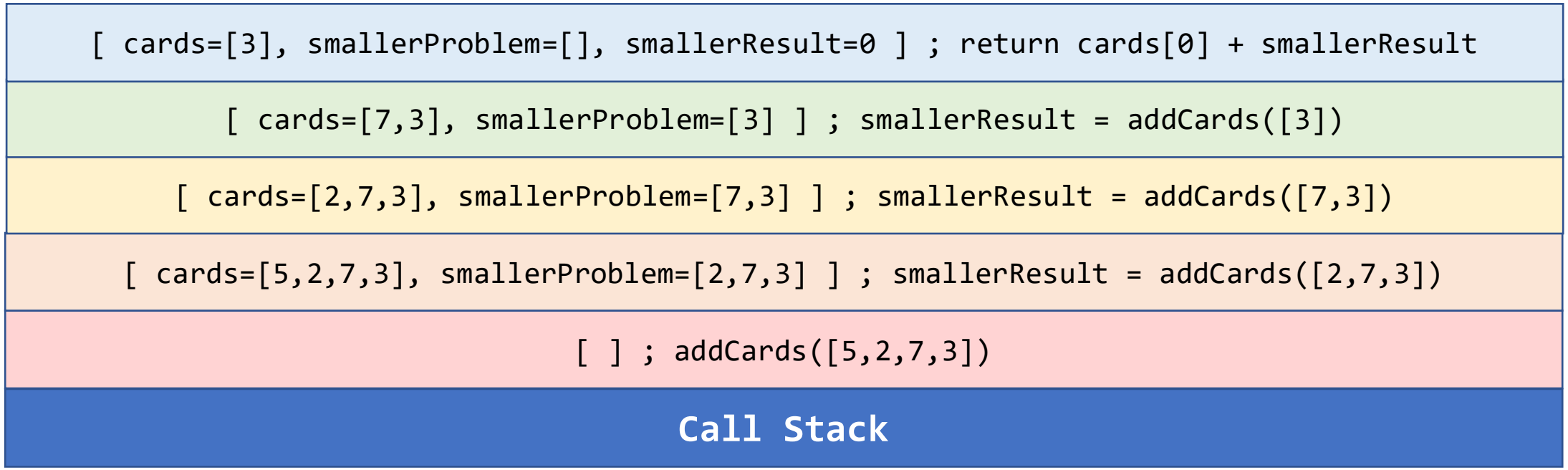
Tracing the Call Stack

Look up the values in cards[0] and smallerResult, add them, and return the result to the next layer of the stack – the **3rd call**.

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

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

4th call



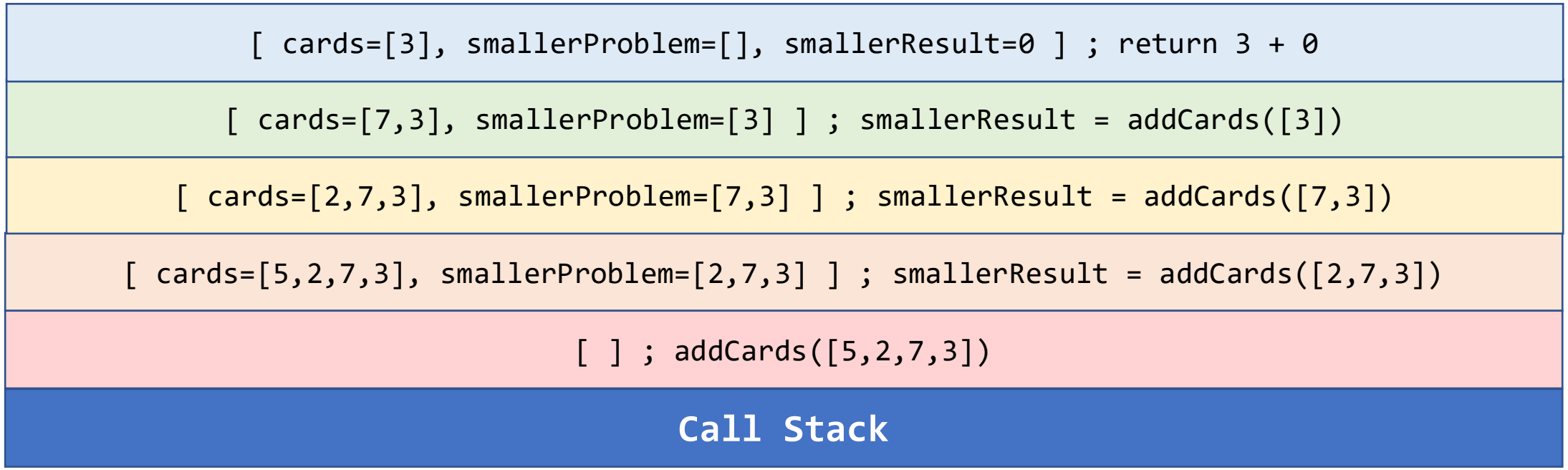
Tracing the Call Stack

Look up the values in cards[0] and smallerResult, add them, and return the result to the next layer of the stack – the **3rd call**.

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

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

4th call



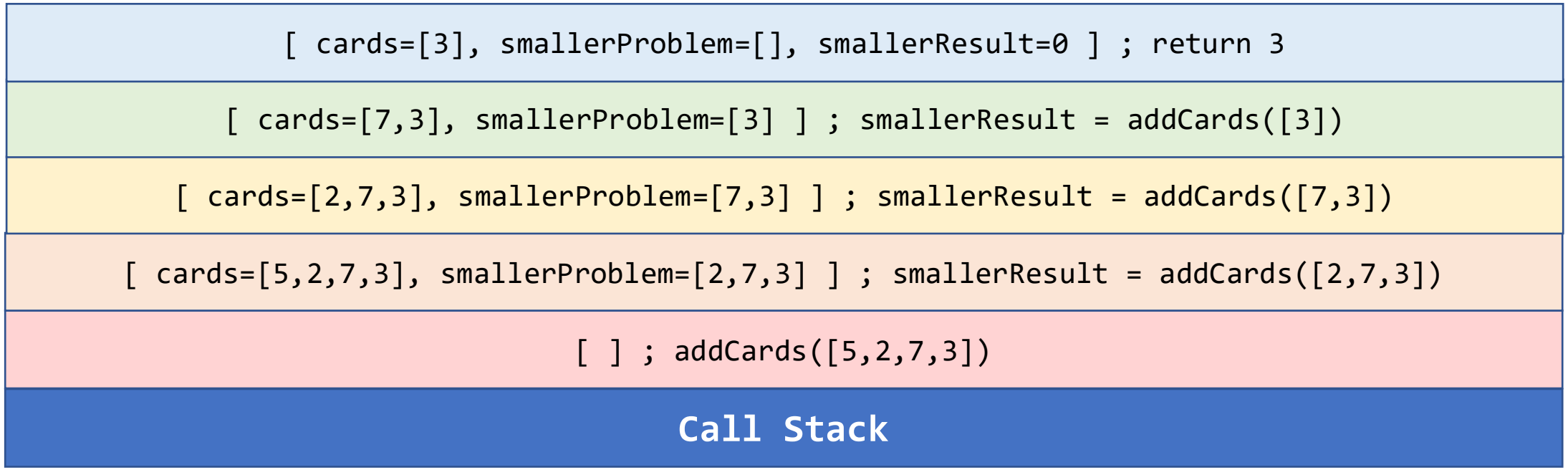
Tracing the Call Stack

Look up the values in cards[0] and smallerResult, add them, and return the result to the next layer of the stack – the **3rd call**.

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

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

4th call



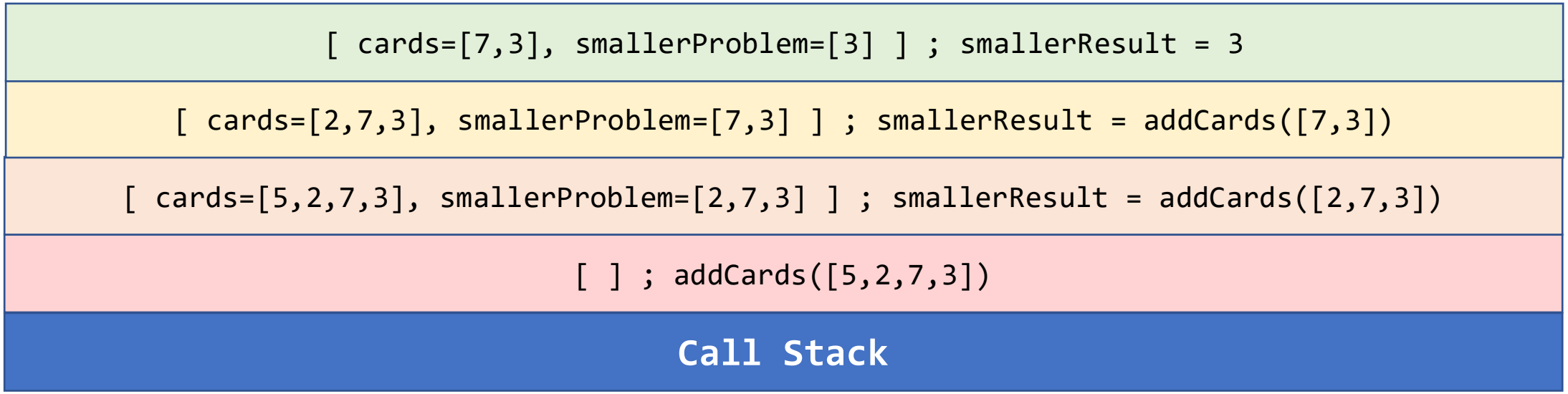
Tracing the Call Stack

Back in the 3rd call, the returned value takes the place of the function call, and a new var is saved.

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

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

3rd call



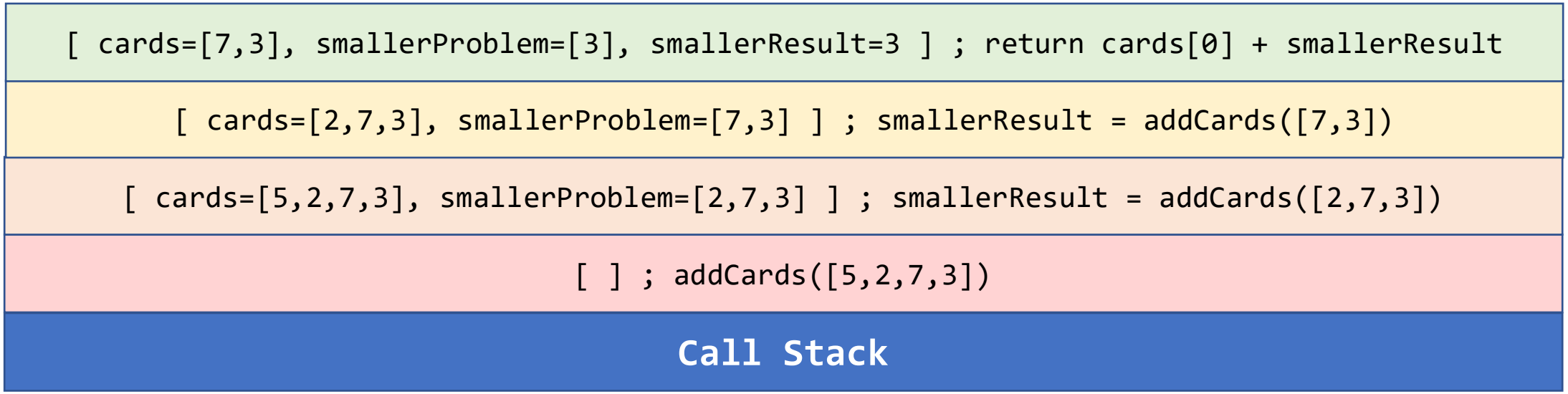
Tracing the Call Stack

Look up the values in cards[0] and smallerResult, add them, and return the result to the next layer of the stack – the **2nd call**.

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

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

3rd call



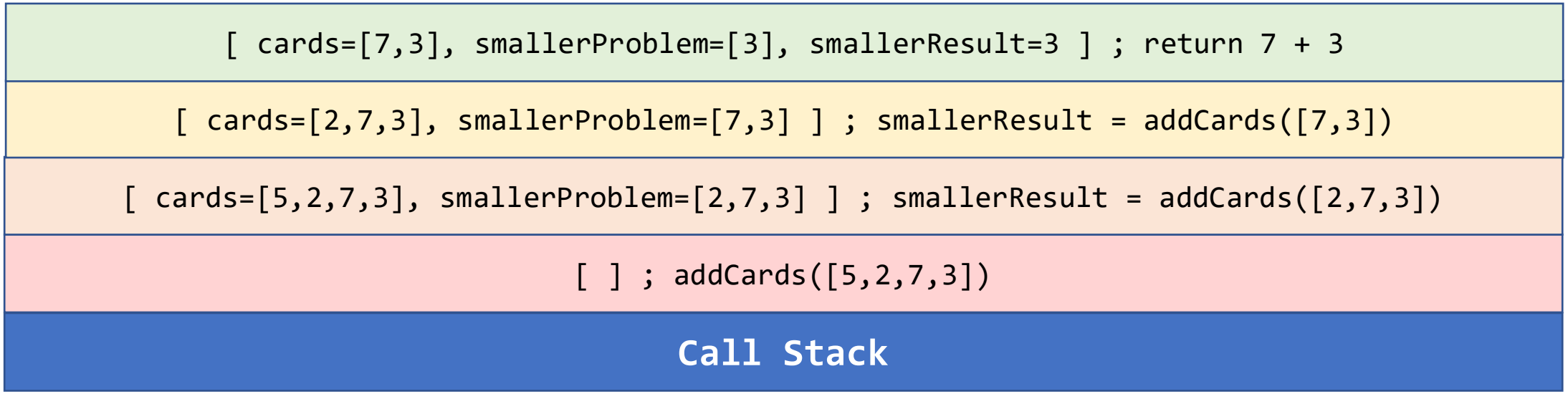
Tracing the Call Stack

Look up the values in cards[0] and smallerResult, add them, and return the result to the next layer of the stack – the **2nd call**.

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

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

3rd call



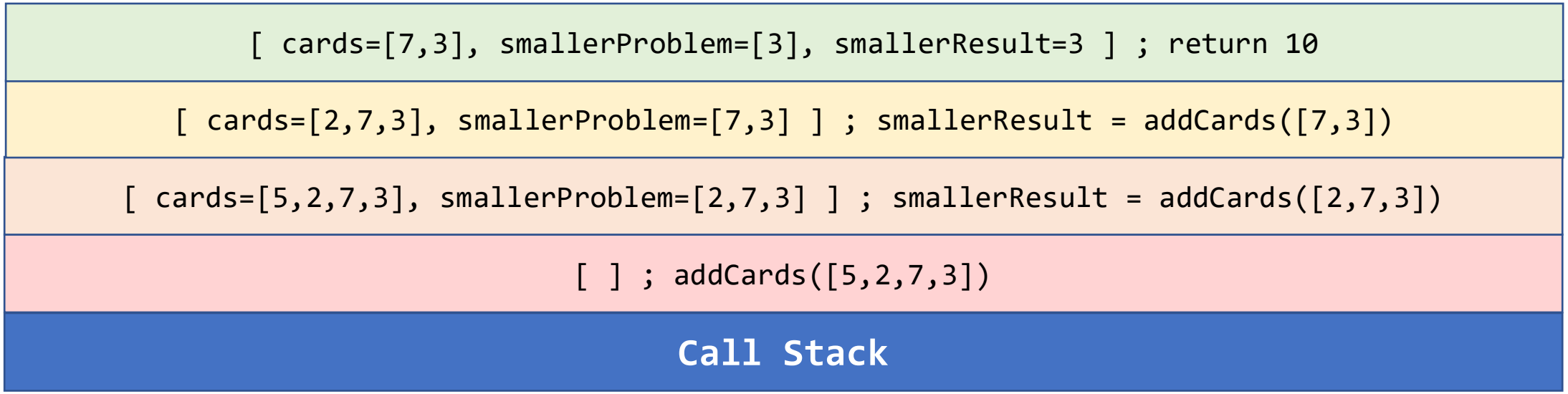
Tracing the Call Stack

Look up the values in cards[0] and smallerResult, add them, and return the result to the next layer of the stack – the **2nd call**.

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

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

3rd call



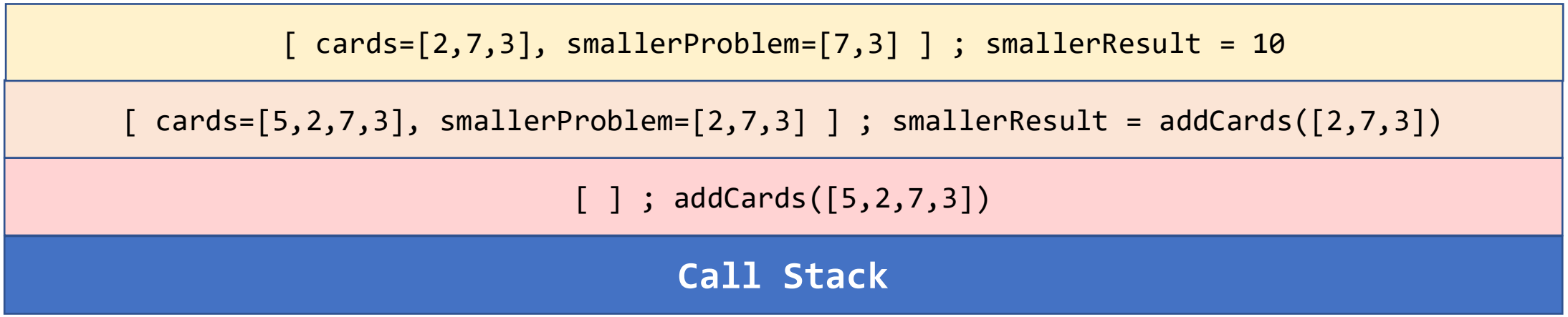
Tracing the Call Stack

Back in the 2nd call, the returned value takes the place of the function call, and a new var is saved.

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

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

2nd call



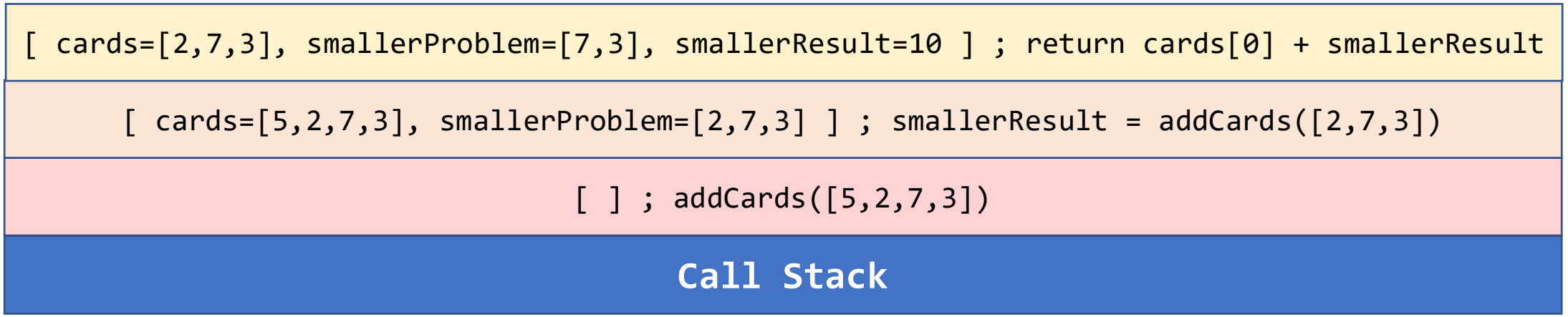
Tracing the Call Stack

Look up the values in cards[0] and smallerResult, add them, and return the result to the next layer of the stack – the **1st call**.

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

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

2nd call



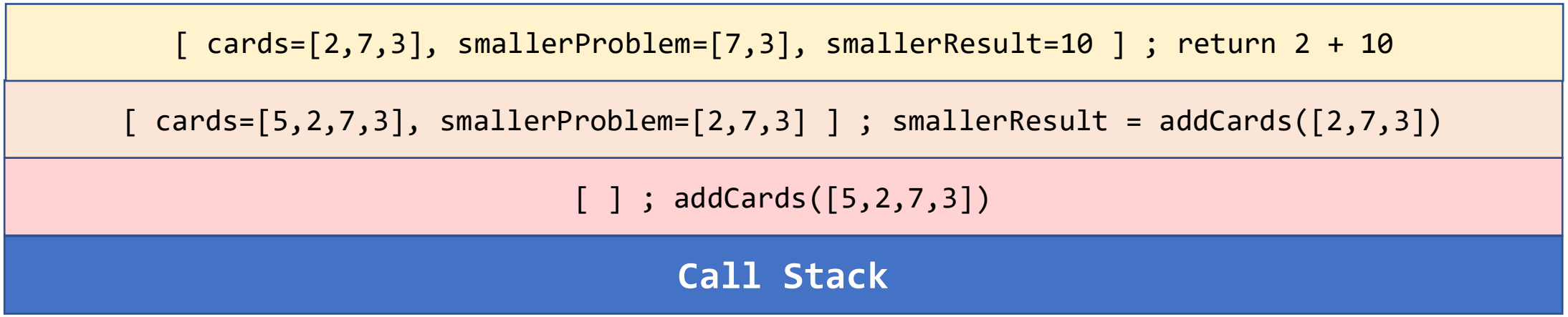
Tracing the Call Stack

Look up the values in cards[0] and smallerResult, add them, and return the result to the next layer of the stack – the **1st call**.

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

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

2nd call



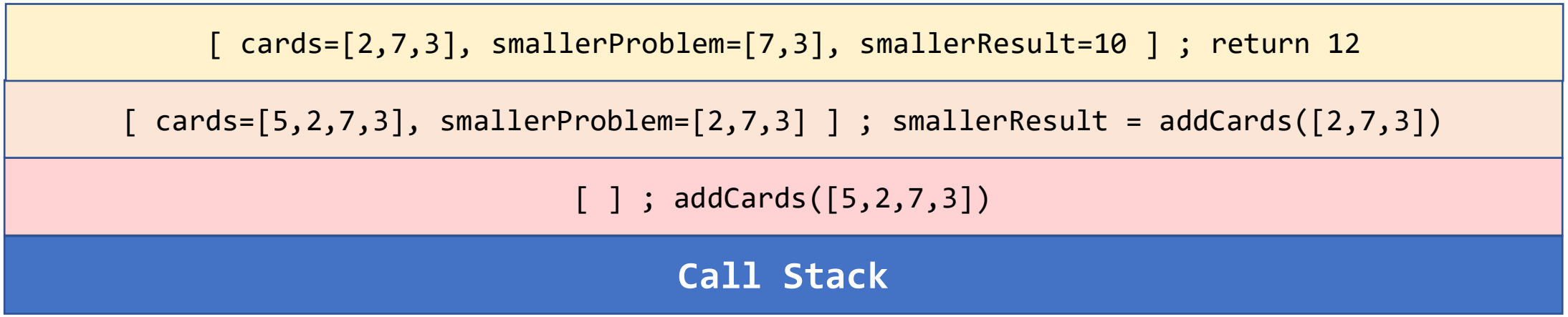
Tracing the Call Stack

Look up the values in cards[0] and smallerResult, add them, and return the result to the next layer of the stack – the **1st call**.

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

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

2nd call



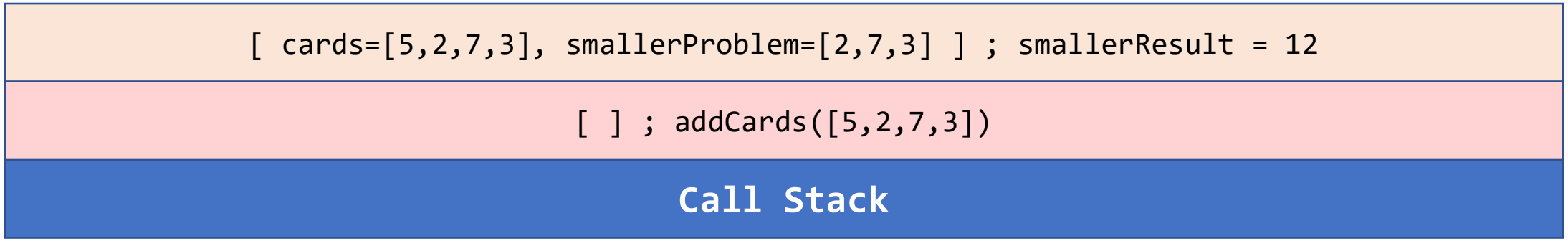
Tracing the Call Stack

Back in the 1st call, the returned value takes the place of the function call, and a new var is saved.

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

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

1st call



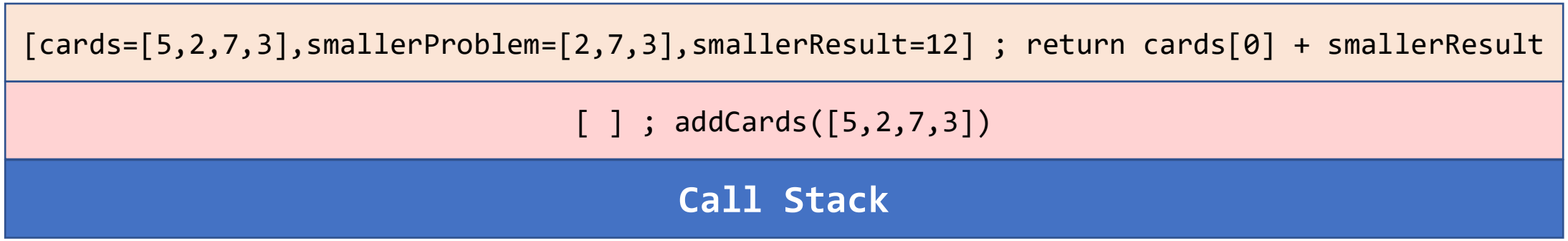
Tracing the Call Stack

Look up the values in cards[0] and smallerResult, add them, and return the result to the next layer of the stack – the **original call**.

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

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

1st call



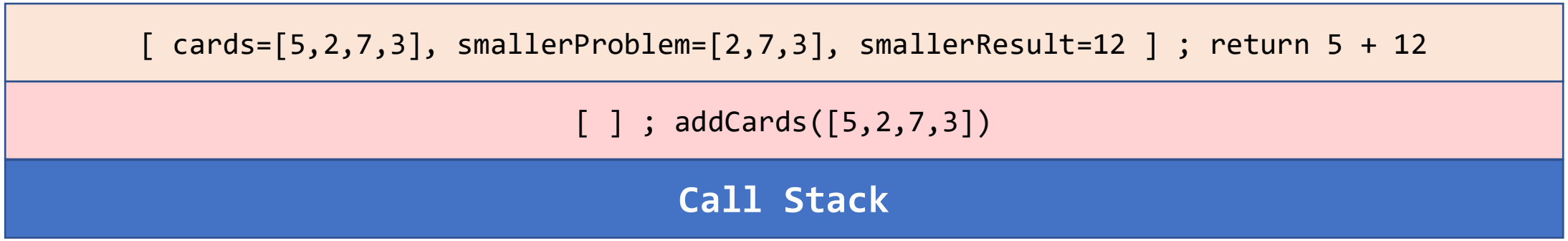
Tracing the Call Stack

Look up the values in cards[0] and smallerResult, add them, and return the result to the next layer of the stack – the **original call**.

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

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

1st call



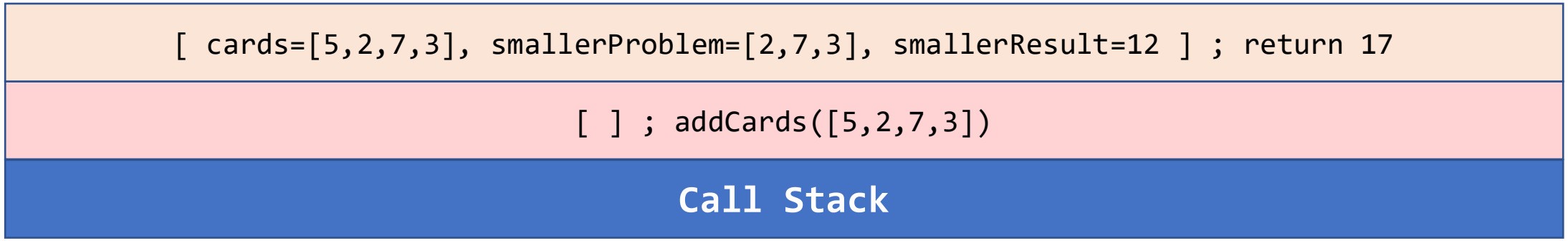
Tracing the Call Stack

Look up the values in cards[0] and smallerResult, add them, and return the result to the next layer of the stack – the **original call**.

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

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

1st call



Tracing the Call Stack

We've finally finished making recursive calls, and we're done!

addCards([5, 2, 7, 2]) evaluates to 17.

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

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

done!