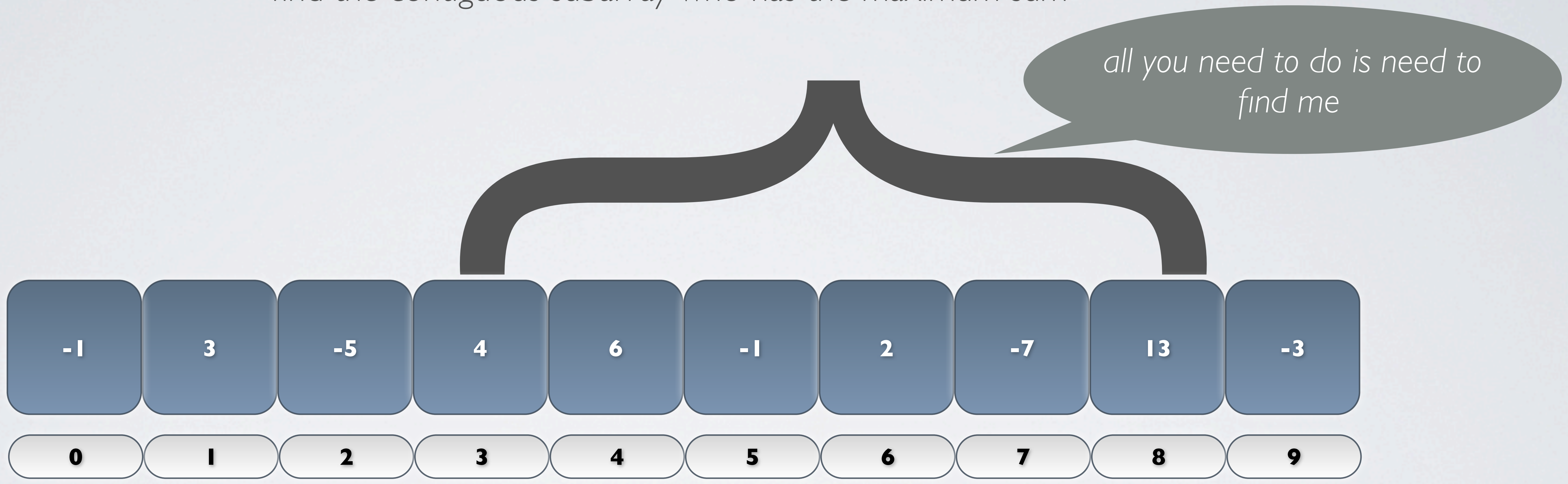
Kadane’s continuous subarray algorithm

by Arun Manivannan

Maximum sum subarray problem

- find the contiguous subarray who has the maximum sum



This program introduces few variables on top of the default implementation of Kadane's algorithm.

We'll have four major variables in all

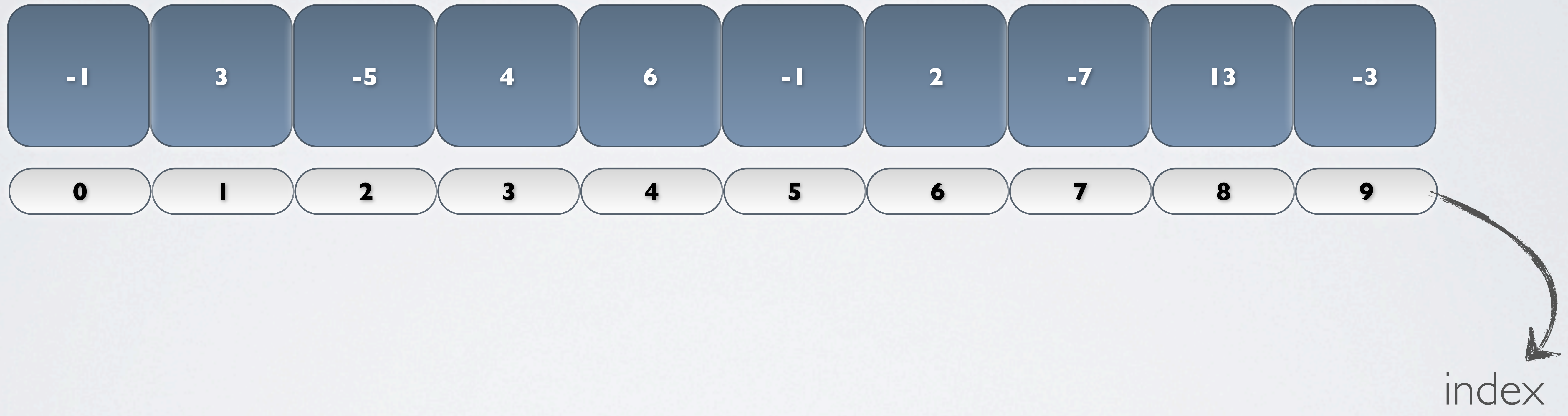
1) *cumulative sum* - holds the cumulative sum of the array

2) *maximum sum* - holds the maximum sum of continuous items in the array.

3) *maximum start index* - start index of the sub array whose total is the maximum within the array

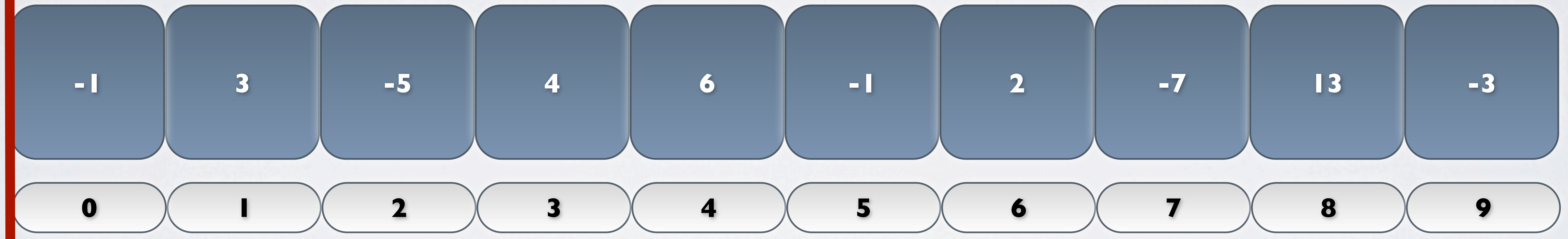
4) *maximum end index* - end index of the sub array whose total is the maximum within the array

Let's take a sample bag of numbers stored in an array



cum sum 0

max sum Integer.MIN_VALUE



-1	3	-5	4	6	-1	2	-7	13	-3
0	1	2	3	4	5	6	7	8	9

current index 0

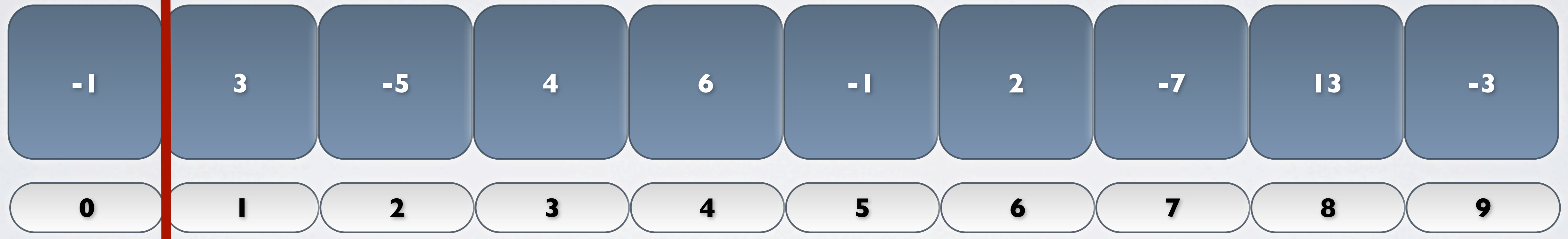
lower index 0

end index 0

cum sum -1

max sum -1

Keep incrementing our cumulative sum



current index 0

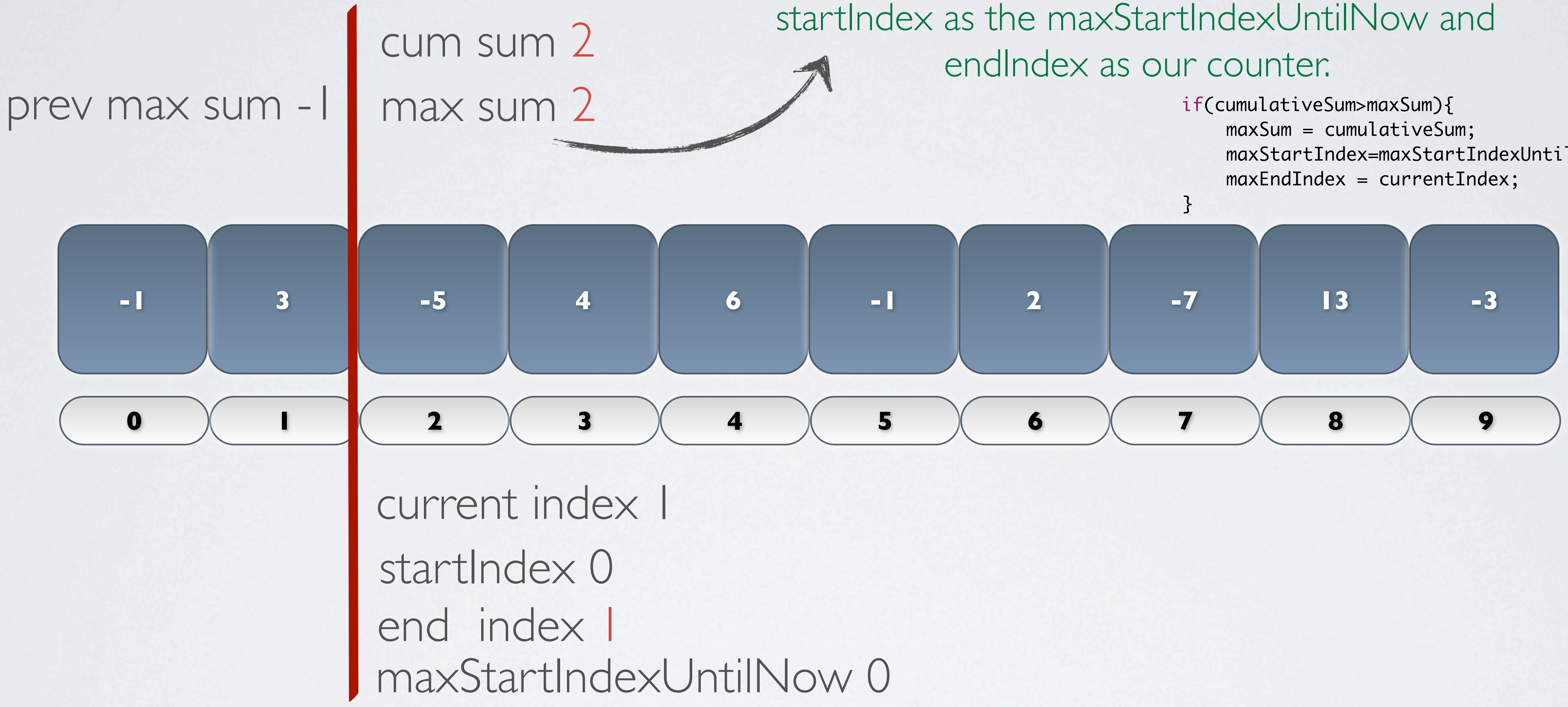
startIndex 0

end index 0

maxStartIndexUntilNow -1

When the cumulative sum is greater than max sum, it just means that the next number is an 'useful addition'.
So, lets set our
max sum as our current cumulative sum,
startIndex as the maxStartIndexUntilNow and
endIndex as our counter.

```
if(cumulativeSum>maxSum){
    maxSum = cumulativeSum;
    maxStartIndex=maxStartIndexUntilNow;
    maxEndIndex = currentIndex;
}
```



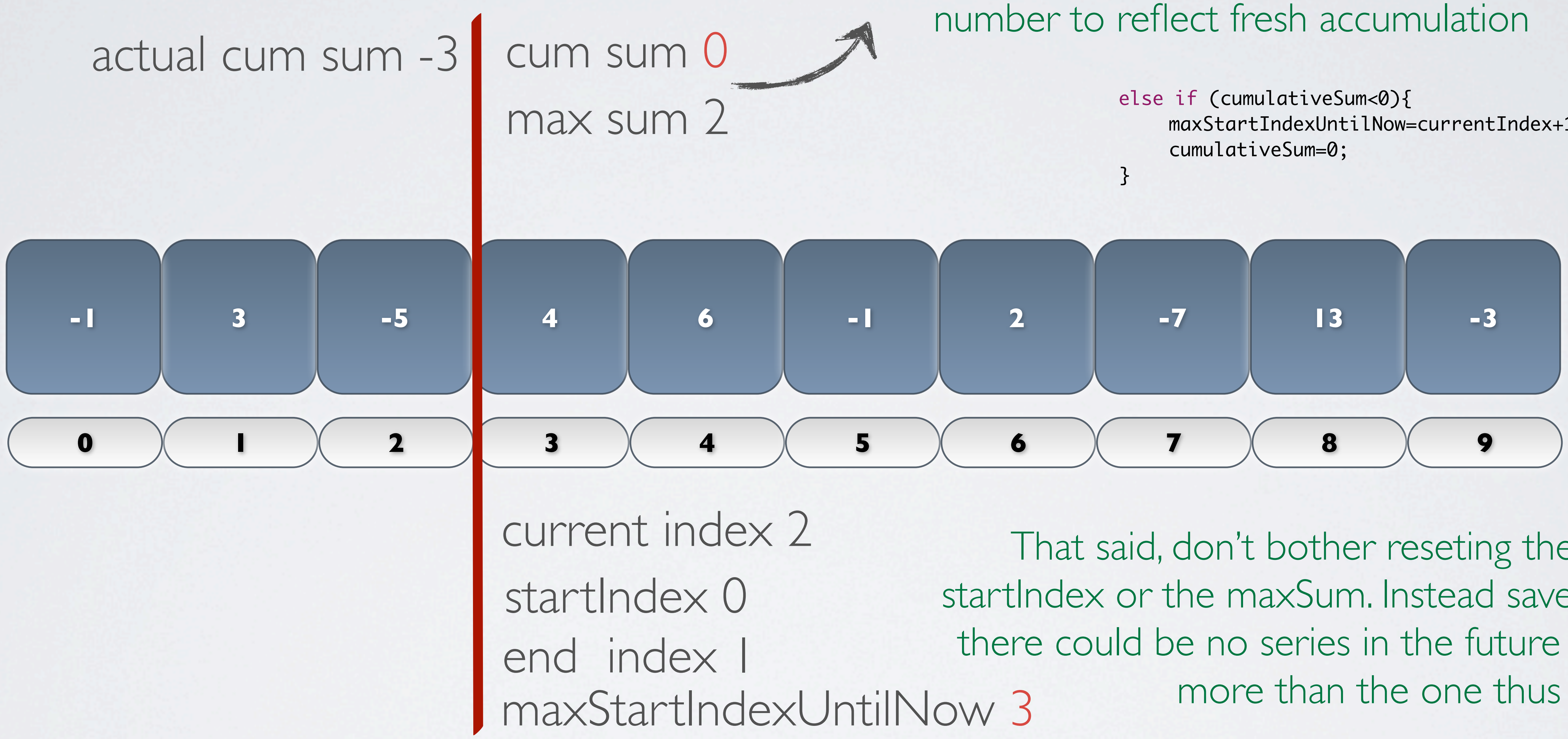
When the cumulative sum is <0, it means that the next number is negative and actually bringing down the total sum. So, reset the cumulative sum as zero to restart the accumulation process from next number. Also, let's set the fresh maxStartIndexUntilNow to the next number to reflect fresh accumulation

actual cum sum -3

cum sum 0

max sum 2

```
else if (cumulativeSum<0){
     maxStartIndexUntilNow=currentIndex+1;
     cumulativeSum=0;
}
```



current index 2

startIndex 0

end index 1

maxStartIndexUntilNow 3

That said, don't bother reseting the endIndex or startIndex or the maxSum. Instead save them aside since there could be no series in the future which has a sum more than the one thus far.

Ahaa, looks like our new cumulative sum is more than our max sum - let's set our new max sum, new startIndex and end Index.

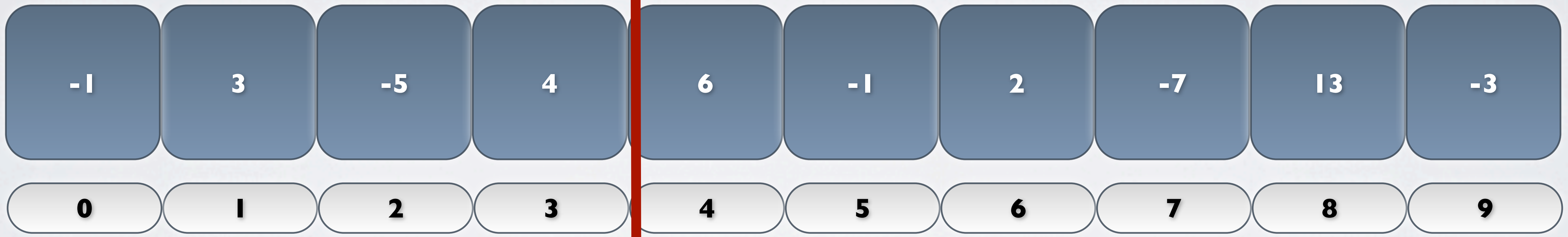
our new 'max' end index would simply be our current Index. Our startIndex is just our 'saved away' startIndex

cum sum 4

prev max sum 2

max sum 4

```
if(cumulativeSum>maxSum){
    maxSum = cumulativeSum;
    maxStartIndex=maxStartIndexUntilNow;
    maxEndIndex = currentIndex;
}
```



current index 3

prev startIndex 0

startIndex 3

prev endIndex 1

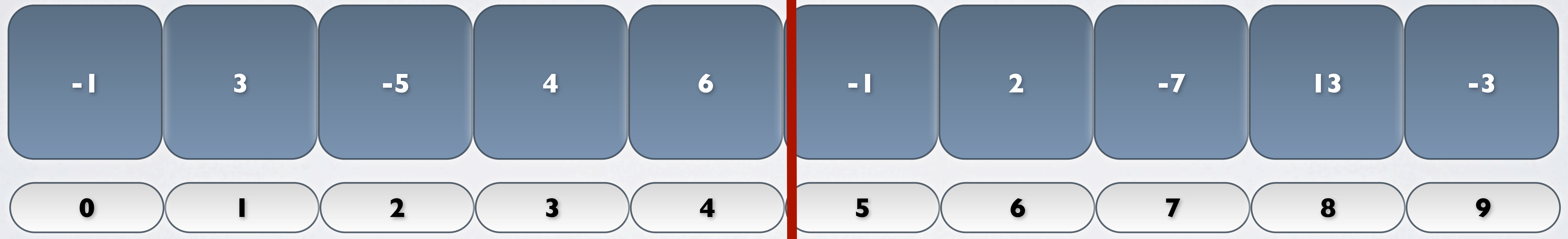
end index 3

maxStartIndexUntilNow 3

More Good news coming along. cumulative sum is increasing and is getting more than the max sum every iteration. As before, reset max sum, end Index and startIndex

cum sum 10

max sum 10



current index 4

startIndex 3

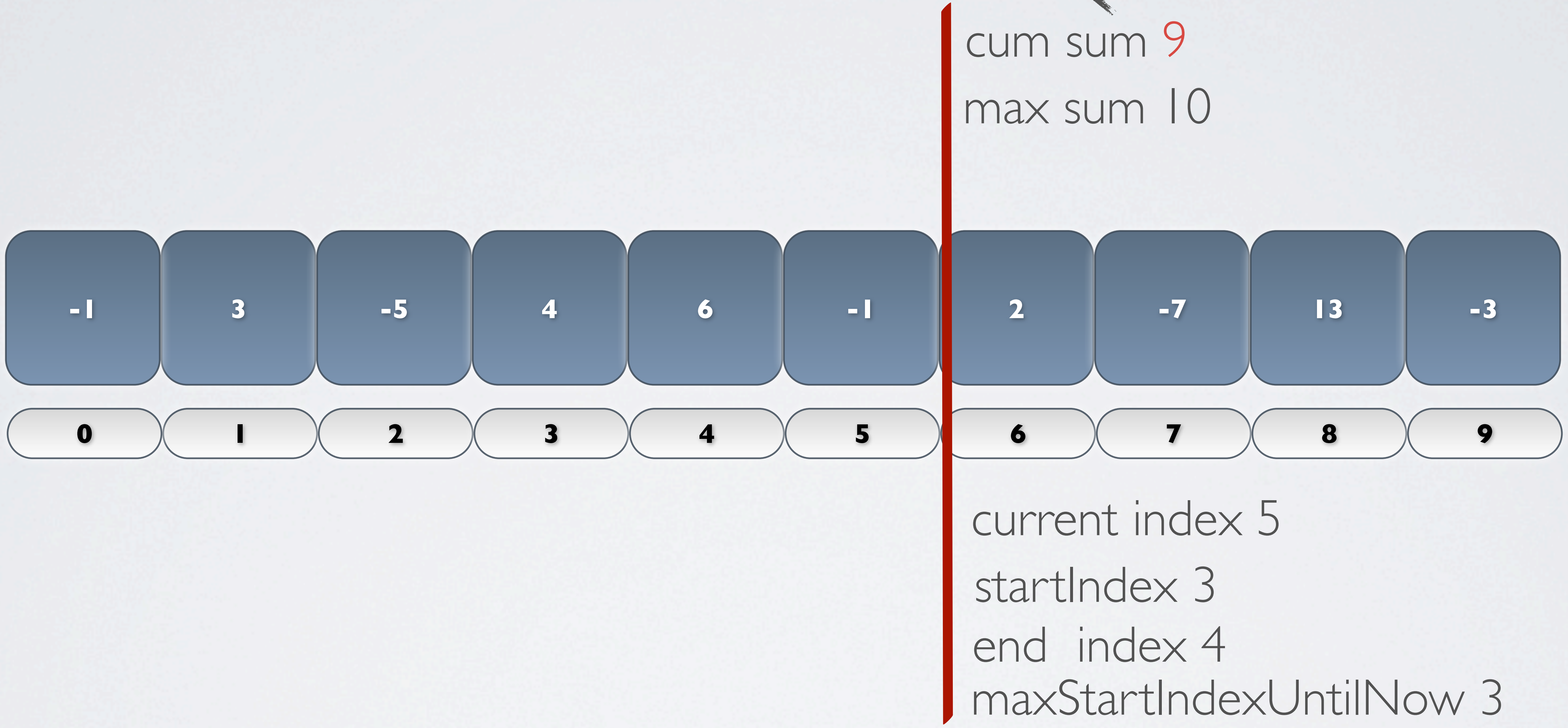
end index 4

maxStartIndexUntilNow 3

A little loss here. However, our max sum is stronger than the cumulative sum still. So, let's just ignore it and do nothing.

cum sum 9

max sum 10



current index 5

startIndex 3

end index 4

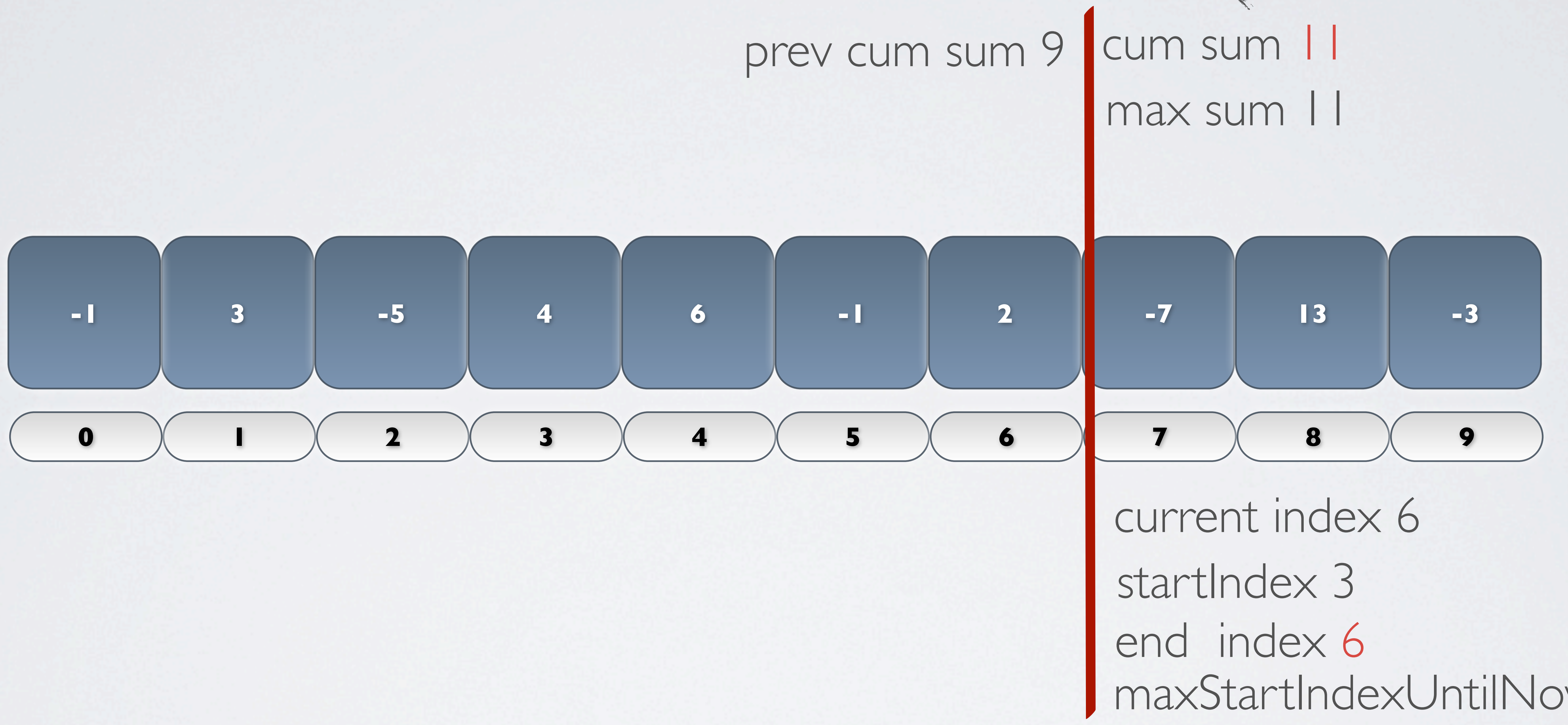
maxStartIndexUntilNow 3

Let’s gobble up the next increment to the sum. Make max sum and end index reflect it.

prev cum sum 9

cum sum 11

max sum 11



current index 6

startIndex 3

end index 6

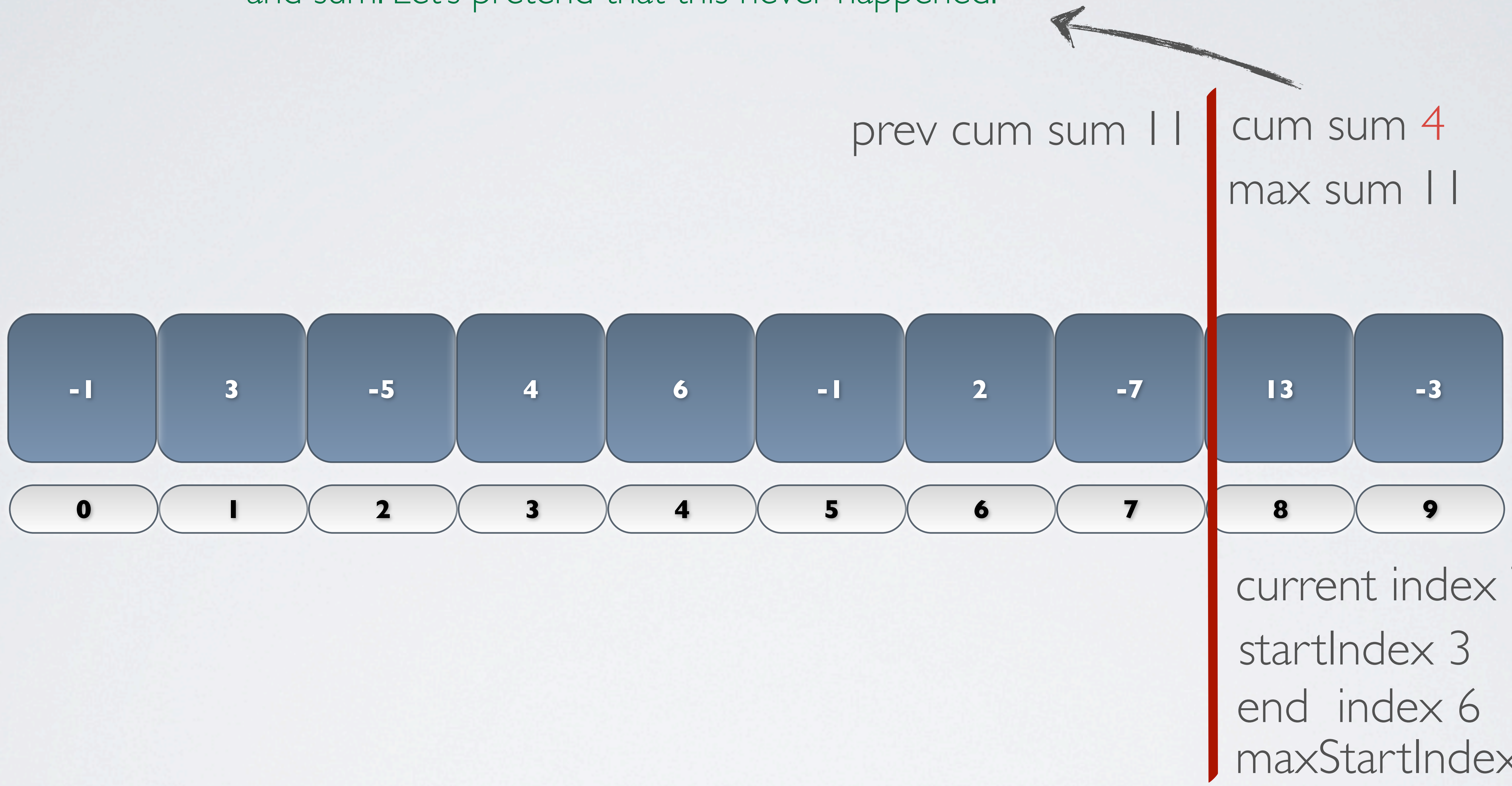
maxStartIndexUntilNow 3

A setback here but we have our savings - max indices and sum. Let's pretend that this never happened.

prev cum sum 11

cum sum 4

max sum 11



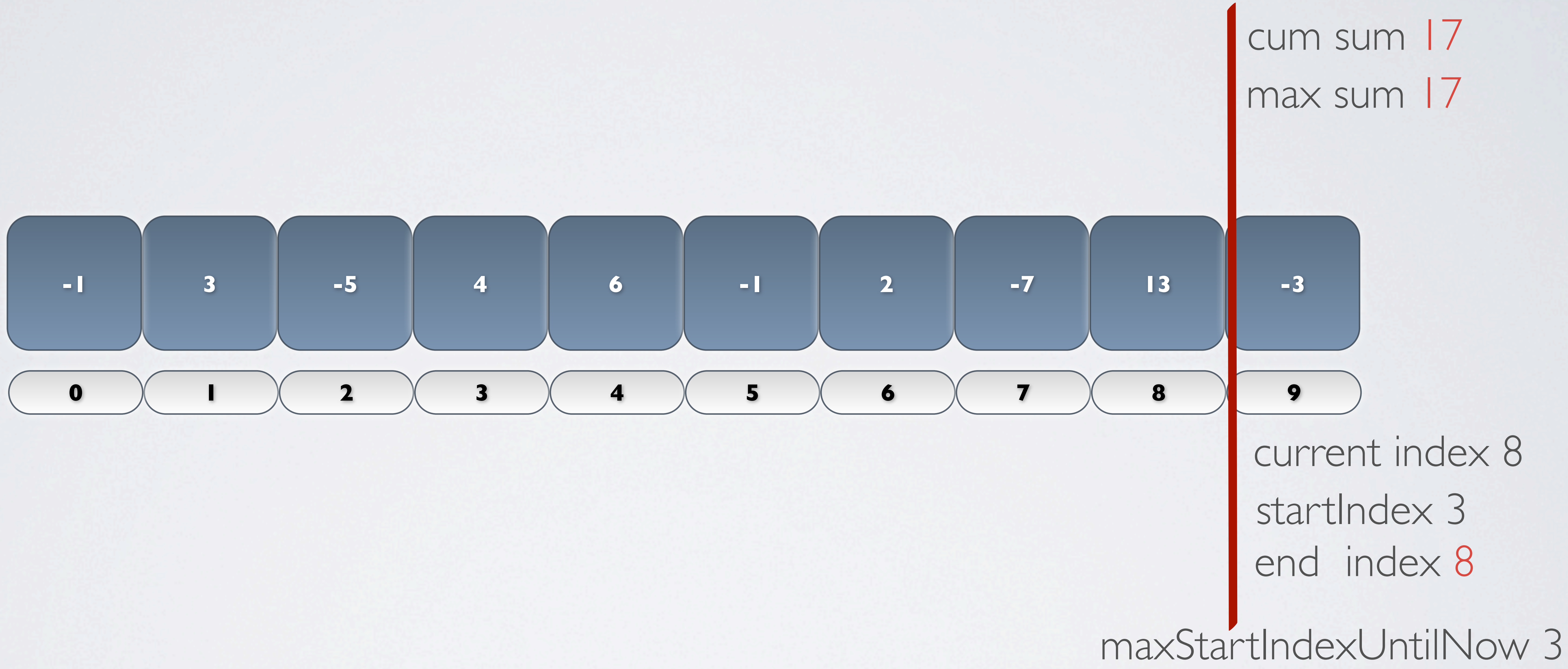
current index 7

startIndex 3

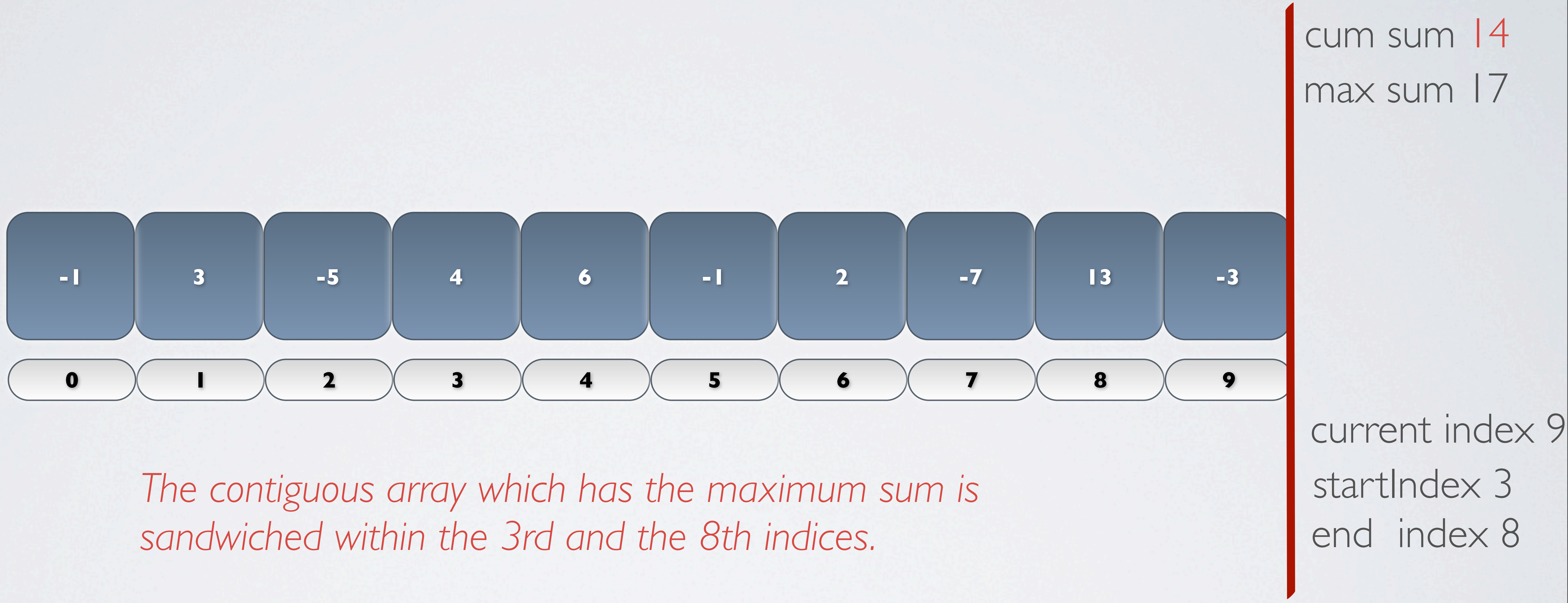
end index 6

maxStartIndexUntilNow 3

Goodness !!!


cum sum 17
max sum 17
-1
3
-5
4
6
-1
2
-7
13
-3
0
1
2
3
4
5
6
7
8
9
current index 8
startIndex 3
end index 8
maxStartIndexUntilNow 3

A final drop. But hey, we've passed the maximum in the last pass. Let's ignore this pass and return the previous results.



The contiguous array which has the maximum sum is sandwiched within the 3rd and the 8th indices.

The maximum sum of the contiguous array is 17

maxStartIndexUntilNow 3