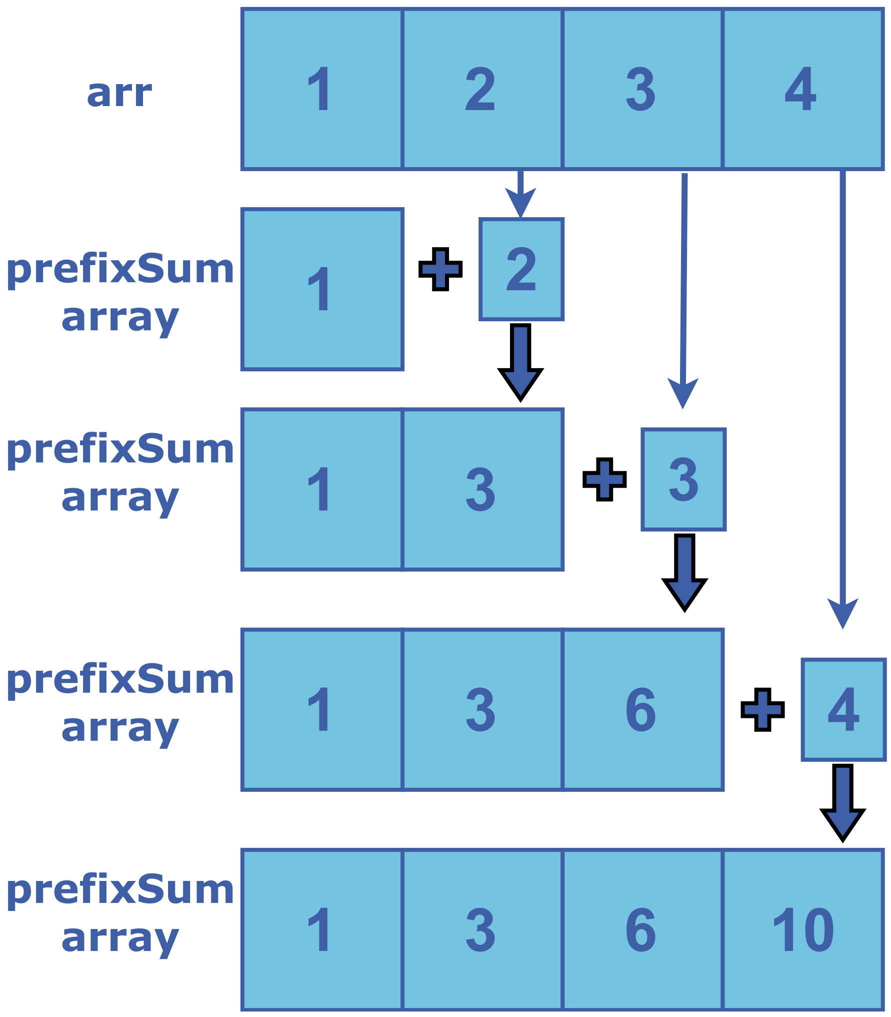
Prefix Sum Problem : A Concurrent Approach

In this lesson, you will study how to write the concurrent solution to the Prefix Sum Algorithm.

The Prefix Sum of an array `arr` of length **n** is another array `prefixSum_arr` of the same length such that the value of the `i` th index in `prefixSum_arr` is the sum of all values from `arr[0], arr[1]...arr[i]`.

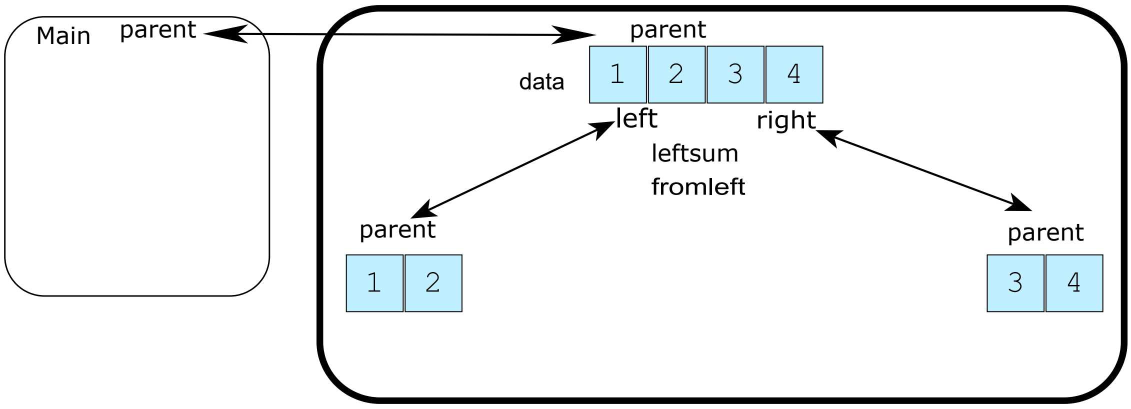


Prefix Sum

Let’s look at the code below:

```
package main
import "fmt"

func PrefixSum(my_array,my_output []int ,parent chan int) {
        if len(my_array)<2{
                parent<-my_array[0]
                my_output[0] = my_array[0] + <-parent

        }else if len(my_array)<1{
                parent<-0
                <-parent
        }else  {
                mid:=len(my_array)/2
                left:= make(chan int)
                right:=make(chan int)
                go PrefixSum(my_array[:mid],my_output[:mid],left)
                go PrefixSum(my_array[mid:],my_output[mid:],right)
                leftsum:=<-left
                parent<- leftsum +<-right
                fromleft:= <-parent
                left<-fromleft
                right<-fromleft + leftsum
                <-left
                <-right

        }
        parent<-0
}

func main () {
        data:= []int{1,2,3,4}
        output:= make([]int,len(data))
        parent :=make(chan int)
        go PrefixSum(data,output,parent)
        sum:= <-parent
        fromleft:=0
        parent<-fromleft
        donezero:=<-parent
        fmt.Println(data,output,sum,donezero)
}
```

ComputingPrefixSum

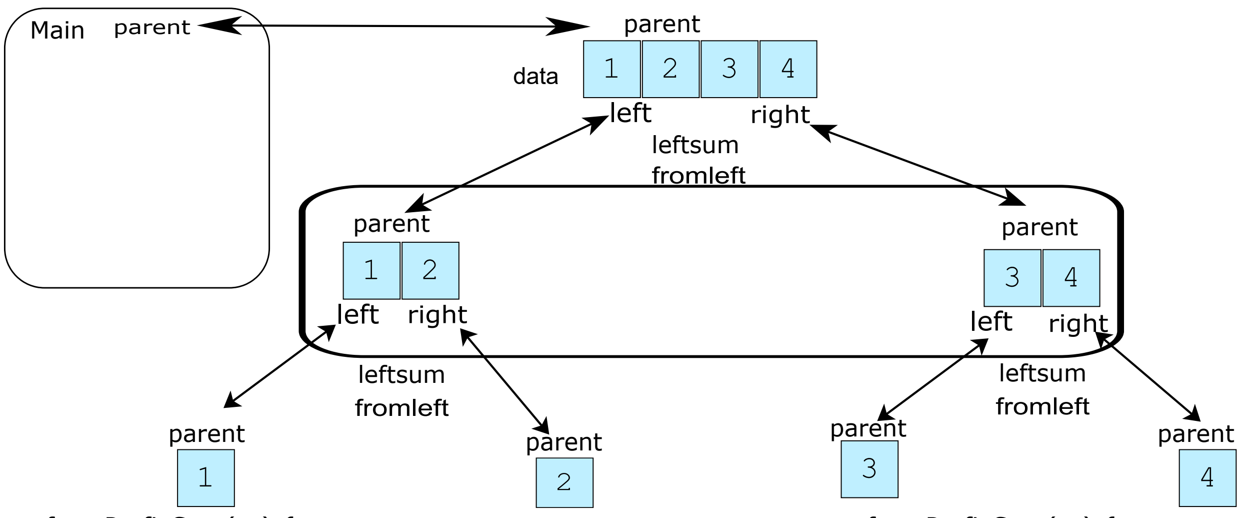


```
func main () {

data:= []int{1,2,3,4}
output:= make([]int,len(data))
parent :=make(chan int)
go PrefixSum(data,output,parent)
sum:= <-parent
fromleft:=0
parent<-fromleft
donezero:=<-parent
fmt.Println(data,output,sum,donezero)
}
```

```
func PrefixSum(my_array,my_output []int ,parent chan int) {
 if len(my_array)<2{
  parent<-my_array[0]
  my_output[0] = my_array[0] + <-parent
 }
 ...
}else {
  mid:=len(my_array)/2
  left:= make(chan int)
  right:=make(chan int)
  go PrefixSum(my_array[:mid],my_output[:mid],left)
  go PrefixSum(my_array[mid:],my_output[mid:],right)
  leftsum:=<-left
  ...
}
```

🟩	Code Executed
🟥	Code Blocked
🟦	Code Currently in Execution

We start from the main loop and call the Prefix Sum function. Both get blocked on lines highlighted in red.



```
func PrefixSum(...) {
 ...
}else {
  mid:=len(my_array)/2
  left:= make(chan int)
  right:=make(chan int)
  go PrefixSum(...)
  go PrefixSum(...)
  leftsum:=<-left
  ...
}
```

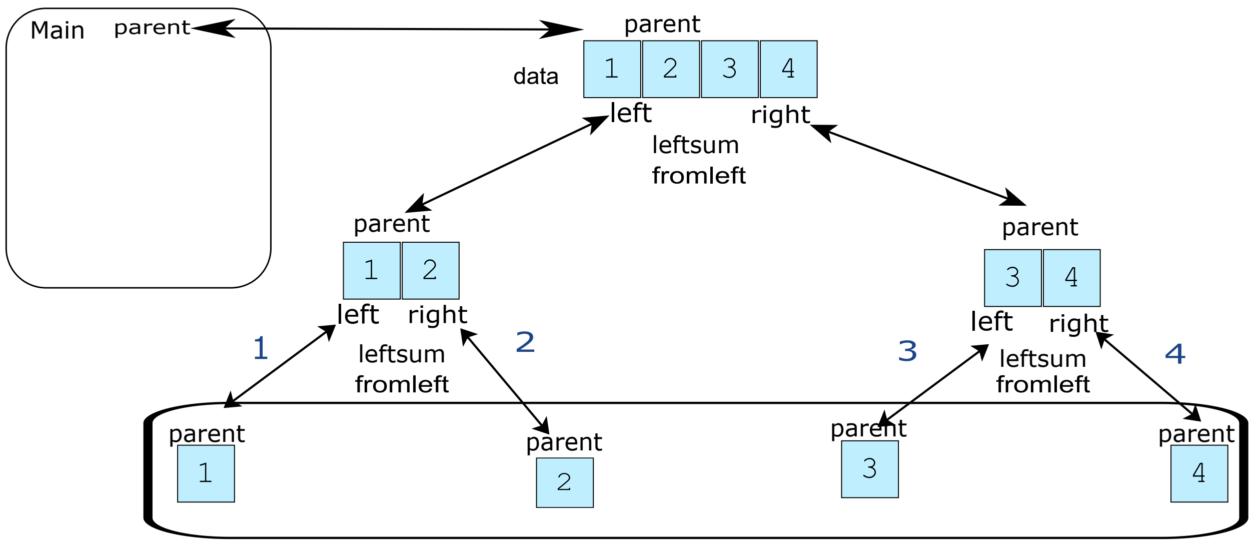
```
func PrefixSum(...) {
 ...
}else {
  mid:=len(my_array)/2
  left:= make(chan int)
  right:=make(chan int)
  go PrefixSum(...)
  go PrefixSum(...)
  leftsum:=<-left
  ...
}
```

🟩 **Code Executed**

🟥 **Code Blocked**

🟦 **Code Currently in Execution**

```
We concurrently make a second recursive call
to the Prefix Sum function.
Both get blocked on lines highlighted in red.
```



```
func PrefixSum(...) {
 if len(my_array)<2{
 parent<-my_array[0]
  my_output[0] = my_array[0] + <-parent
 }
 ...
}else {
  ...
}
```

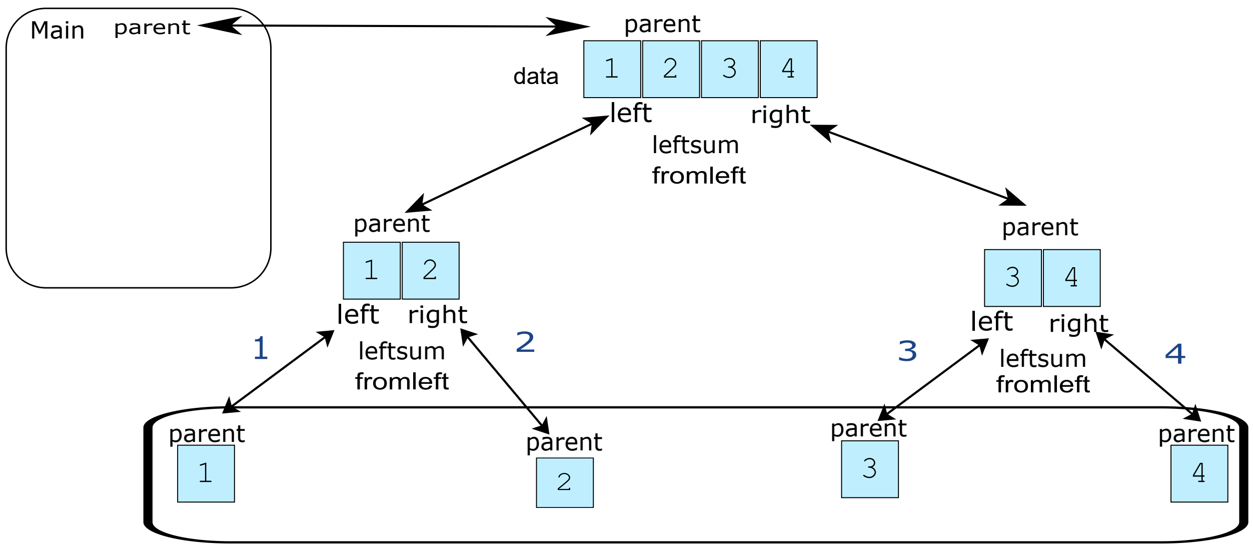
```
func PrefixSum(...) {
 if len(my_array)<2{
 parent<-my_array[0]
  my_output[0] = my_array[0] + <-parent
 }
 ...
}else {
  ...
}
```

🟩 **Code Executed**

🟥 **Code Blocked**

🟦 **Code Currently in Execution**

We send `array[0]` to the `parent` channel which is the `left` or `right` of the previous recursive calls.



```
func PrefixSum(...) {
 if len(my_array)<2{
 parent<-my_array[0]
  my_output[0] = my_array[0] + <-parent
 }
 ...
}else {
  ...
}
```

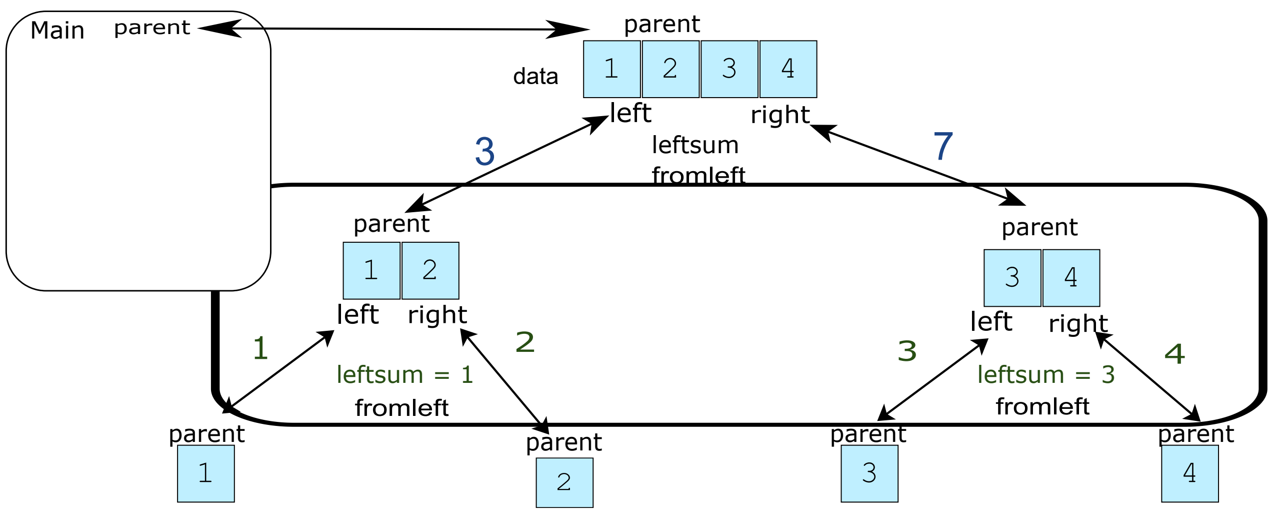
```
func PrefixSum(...) {
 if len(my_array)<2{
 parent<-my_array[0]
  my_output[0] = my_array[0] + <-parent
 }
 ...
}else {
  ...
}
```

🟩 **Code Executed**

🟥 **Code Blocked**

🟦 **Code Currently in Execution**

```
Now we are blocked on `parent` channel in
this recursive call. We'll move to the previous call which
is running concurrently.
```



```
func PrefixSum(...) {
 ...
  ...
  go PrefixSum(...)
  go PrefixSum(...)
 leftsum:=<-left
  parent<- leftsum +<-right
  fromleft:= <-parent
  ...
}
```

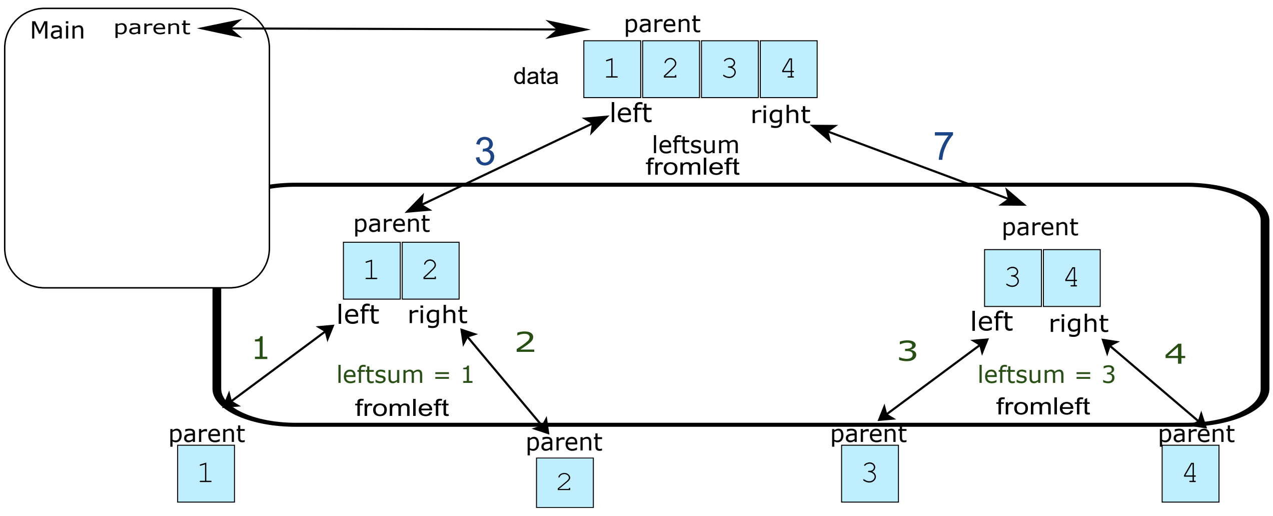
```
func PrefixSum(...) {
 ...
  ...
  go PrefixSum(...)
  go PrefixSum(...)
 leftsum:=<-left
  parent<- leftsum +<-right
  fromleft:= <-parent
  ...
}
```

🟩 **Code Executed**

🟥 **Code Blocked**

🟦 **Code Currently in Execution**

```
We receive the values from the `left` and `right`
channels and update the value to `parent` channel.
```



```
func PrefixSum(...) {
 ...
  ...
  go PrefixSum(...)
  go PrefixSum(...)
 leftsum:=<-left
  parent<- leftsum +<-right
  fromleft:= <-parent
  ...
}
```

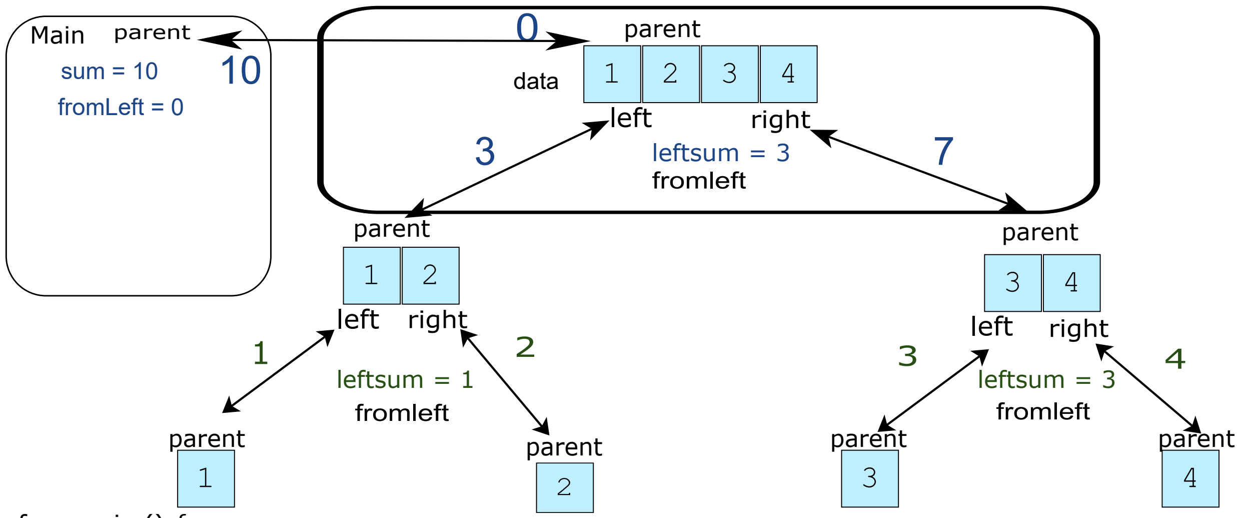
```
func PrefixSum(...) {
 ...
  ...
  go PrefixSum(...)
  go PrefixSum(...)
 leftsum:=<-left
  parent<- leftsum +<-right
  fromleft:= <-parent
  ...
}
```

🟩 **Code Executed**

🟥 **Code Blocked**

🟦 **Code Currently in Execution**

```
Now we are blocked on `parent` channel in
this recursive call. We'll move to the previous call which
is running concurrently.
```



```
func main () {

data:= []int{1,2,3,4}
output:= make([]int,len(data))
parent :=make(chan int)
go PrefixSum(data,output,parent)
sum := <-parent
fromleft := 0
parent <- fromleft
donezero:=<-parent
fmt.Println(data,output,sum,donezero)
}
```

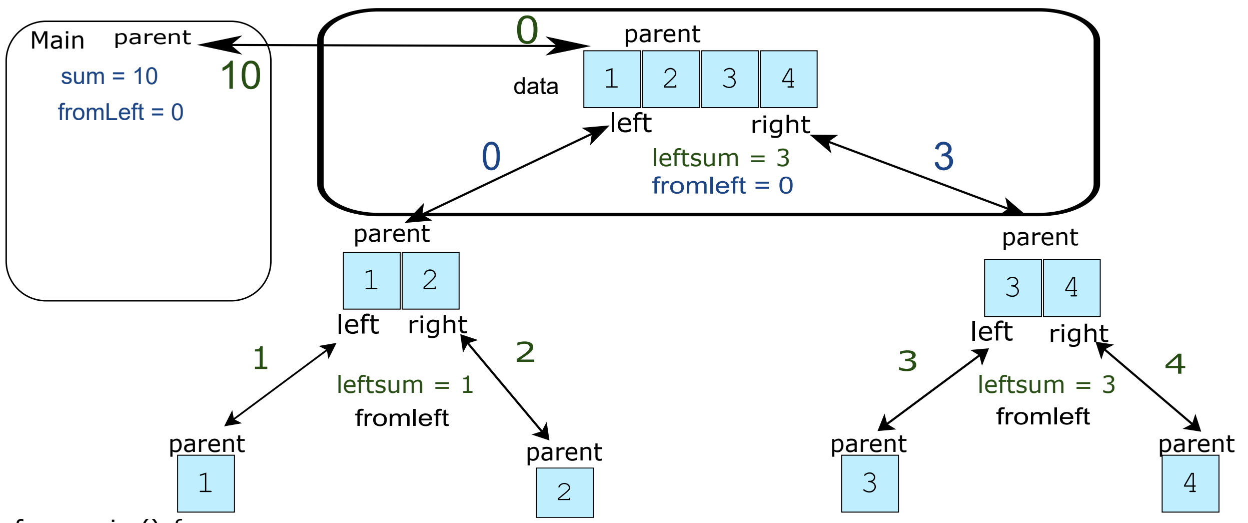
🟩 Code Executed

🟥 Code Blocked

🟦 Code Currently in Execution

```
func PrefixSum(my_array,my_output []int ,parent chan int) {
 ...
 go PrefixSum(my_array[mid:],my_output[mid:],right)
leftsum:=<-left
 parent<- leftsum +<-right
 fromleft := <-parent
 ...
}
```

Now we received `10` in main routine sent by the `parent` in the `PrefixSum` goroutine after executing the following lines: leftsum:=<-left ,parent<- leftsum +<-right We update the value of `fromLeft` in the main routine and send back `0` on the parent channel to the PrefixSum funtion.



```
func main () {

data:= []int{1,2,3,4}
output:= make([]int,len(data))
parent :=make(chan int)
go PrefixSum(data,output,parent)
sum := <-parent
fromleft := 0
parent <- fromleft
donezero:=<-parent
fmt.Println(data,output,sum,donezero)
}
```

🟩 Code Executed

🟥 Code Blocked

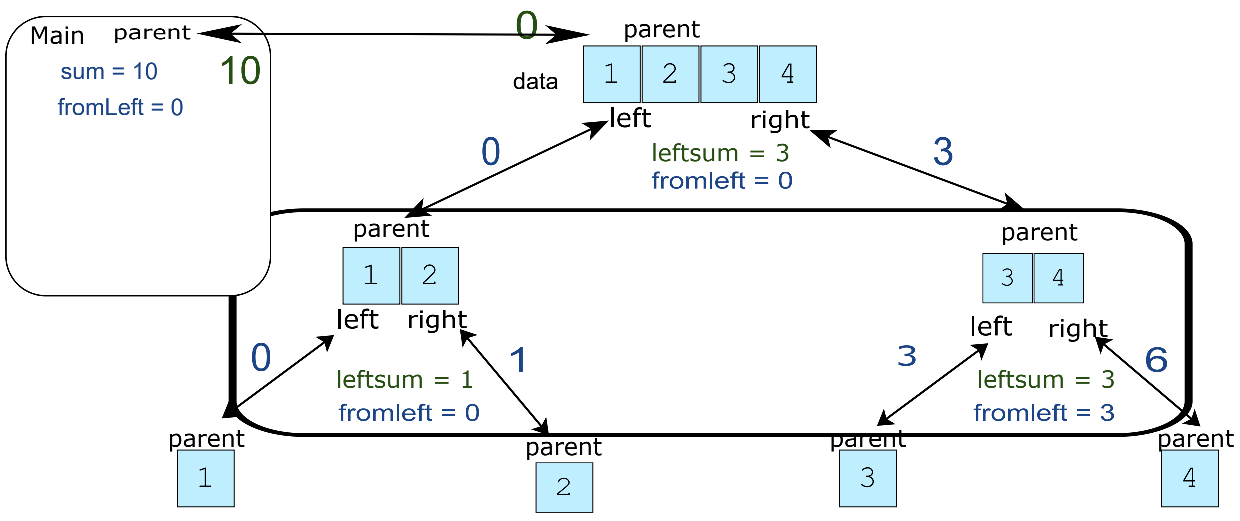
🟦 Code Currently in Execution

```
func PrefixSum(my_array,my_output []int ,parent chan int) {
 ...
  go PrefixSum(my_array[mid:],my_output[mid:],right)
 leftsum:=<-left
  parent<- leftsum +<-right
  fromleft := <-parent
 left<-fromleft
  right<-fromleft + leftsum
  <-left
  ...
}
```

We update the value of `fromLeft` in the PrefixSum goroutine and send back values to the `left` and `right` channels in the following statements:

left<-fromleft, right<-fromleft + leftsum

Both the main routine and the first ever recursive call are blocked receiving operations.



```
func PrefixSum(...)
{ ...
leftsum:=<-left
parent<- leftsum +<-right
fromleft:= <-parent
left<-fromleft
right<-fromleft + leftsum
<-left
<-right
 ...}
```

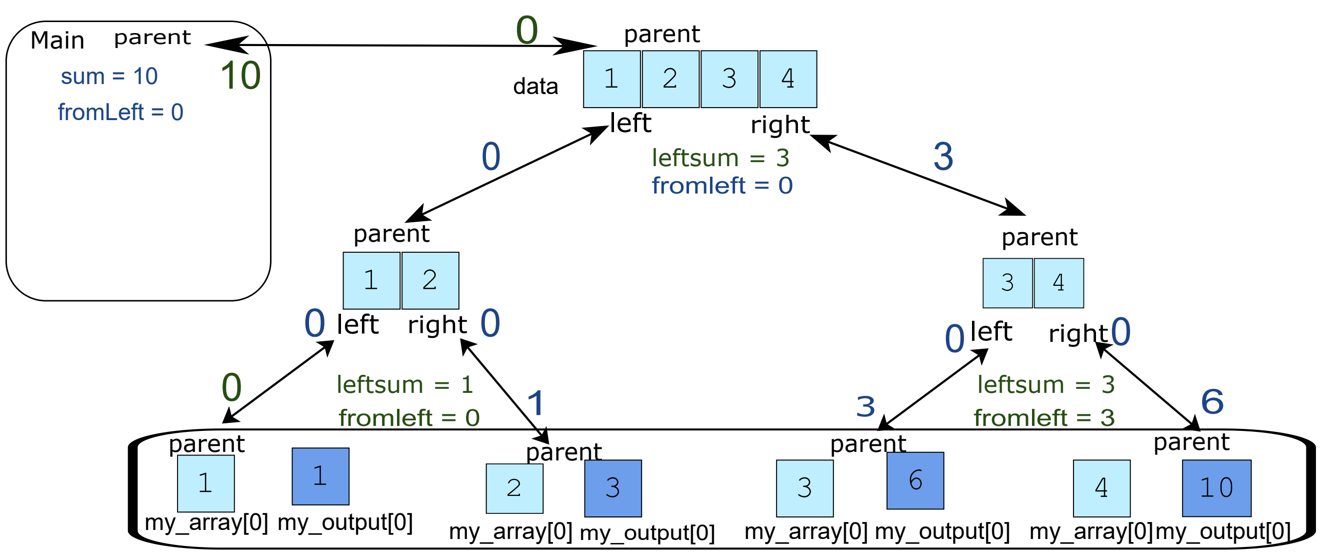
🟩 **Code Executed**

🟥 **Code Blocked**

🟦 **Code Currently in Execution**

```
func PrefixSum(...)
{ ...
leftsum:=<-left
parent<- leftsum +<-right
fromleft:= <-parent
left<-fromleft
right<-fromleft + leftsum
<-left
<-right
 ...}
```

We update the value of `fromLeft` in the PrefixSum goroutines and send values to the `left` and `right` channels in the following statements: `left<-fromleft, right<-fromleft + leftsum` Now the goroutines from this phase of recursive call are blocked on `<-left`.



```
func PrefixSum(...) {
 if len(my_array)<2{
 parent<-my_array[0]
  my_output[0] = my_array[0] + <-parent
 }
 ...
}else {
  ...
}
parent->0
```

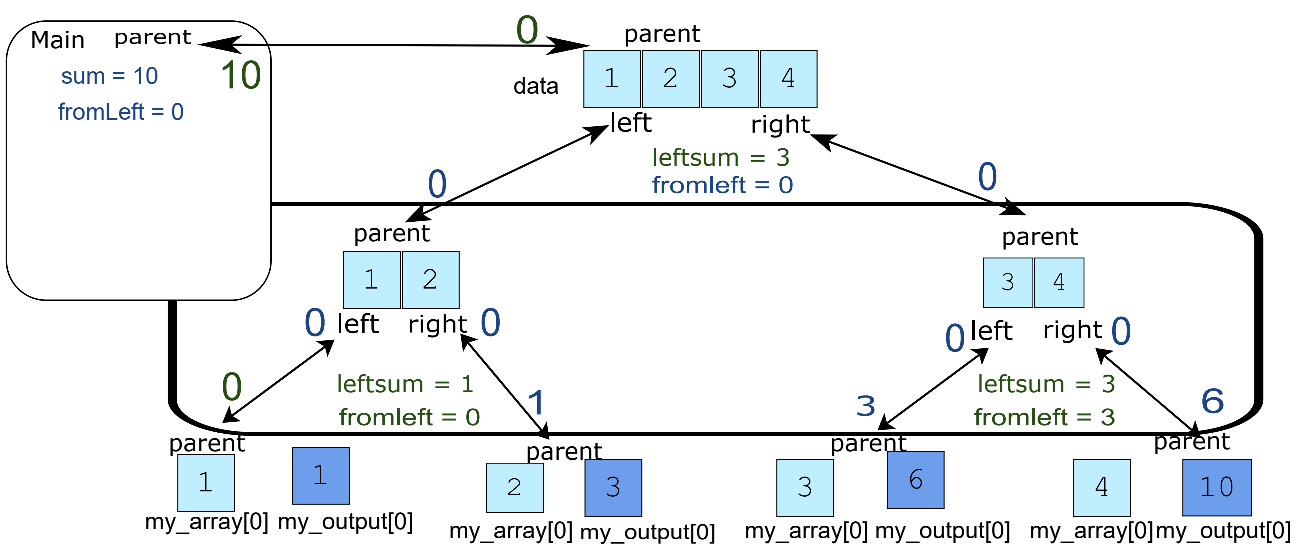
```
func PrefixSum(...) {
 if len(my_array)<2{
 parent<-my_array[0]
  my_output[0] = my_array[0] + <-parent
 }
 ...
}else {
  ...
}
parent->0
```

🟩 **Code Executed**

🟥 **Code Blocked**

🟦 **Code Currently in Execution**

`We update the ouput values in all the goroutines of the last recursive calls and send `0` to the parent channels.`



```
func PrefixSum(...) {
 ...
 leftsum:=<-left
  parent<- leftsum +<-right
 fromleft:= <-parent
  left<-fromleft
  right<-fromleft + leftsum
  <-left
  <-right
  ...
}
 parent <-0
```

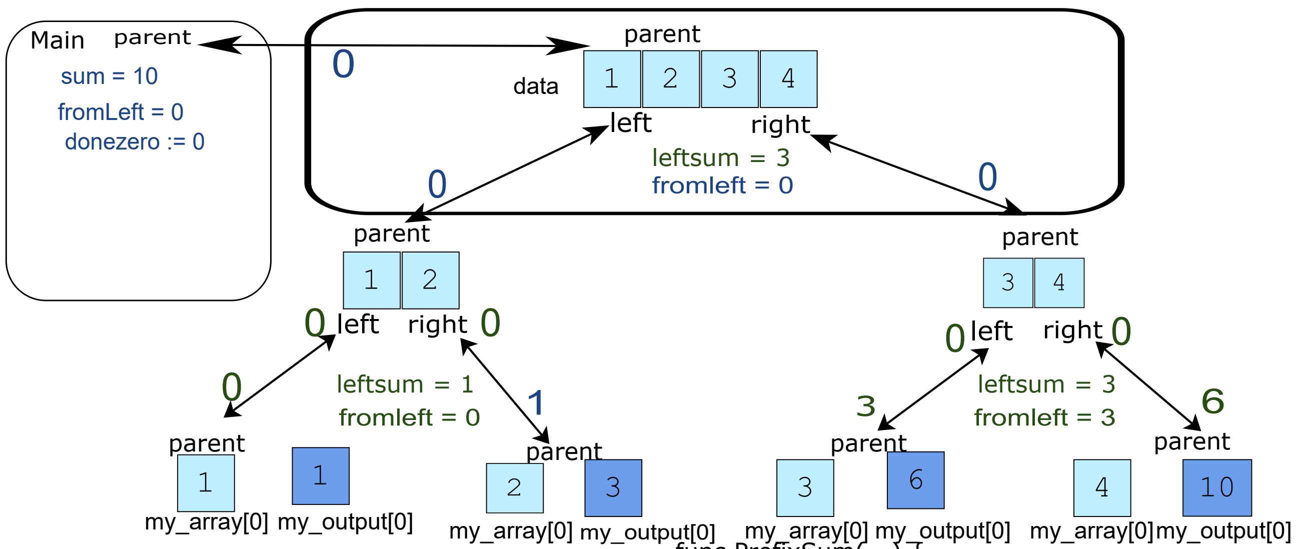
```
func PrefixSum(...) {
 ...
 leftsum:=<-left
  parent<- leftsum +<-right
 fromleft:= <-parent
  left<-fromleft
  right<-fromleft + leftsum
  <-left
  <-right
  ...
}
 parent <-0
```

🟩 Code Executed

🟥 Code Blocked

🟦 Code Currently in Execution

```
These routines that were blocked on these channels
have received their values and now send `0` to the
`parent` channel.
```



```
func main () {
data:= []int{1,2,3,4}
output:= make([]int,len(data))
parent :=make(chan int)
go PrefixSum(data,output,parent)
sum := <-parent
fromleft := 0
parent <- fromleft
donezero:=<-parent
fmt.Println(data,output,sum,donezero)
}
```

```
func PrefixSum(...) {
 ...
 leftsum:=<-left
  parent<- leftsum +<-right
 fromleft:= <-parent
  left<-fromleft
  right<-fromleft + leftsum
  <-left
  <-right
  ...
}
 parent <-0
```

🟩 **Code Executed**

🟥 **Code Blocked**

🟦 **Code Currently in Execution**

```
These routines too that were blocked on these left and right channels
have received their values and now send `0` to the
`parent` channel in the main routine which updates the `donezero` variable.
```