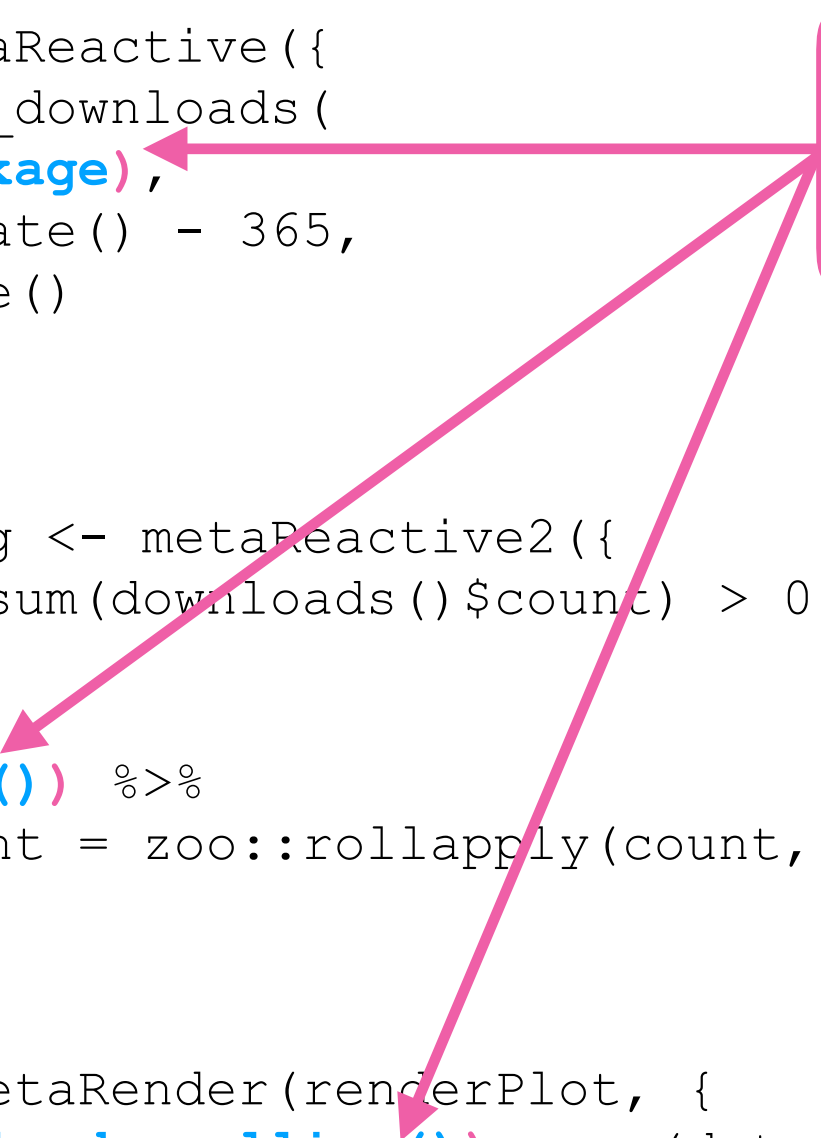


Step 2: Mark reactive reads

```
server <- function(input, output, session) {  
  
  downloads <- metaReactive({  
    cranlogs::cran_downloads(  
      ..(input$package),  
      from = Sys.Date() - 365,  
      to = Sys.Date()  
    )  
  })  
  
  downloads_rolling <- metaReactive2({  
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))  
  
    metaExpr({  
      ..(downloads()) %>%  
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))  
    })  
  })  
  
  output$plot <- metaRender(renderPlot, {  
    ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()  
  })  
}
```



Replaced by a static value or name (when code is generated)

The diagram illustrates the process of replacing reactive reads with static values or names. Three pink arrows originate from a pink box on the right and point to specific parts of the R code: the first arrow points to `..(input$package)` in the `metaReactive` call; the second arrow points to `..(downloads())` in the `metaExpr` call; and the third arrow points to `..(downloads_rolling())` in the `ggplot` call.

Step 2: Mark reactive reads

```
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      ..(input$package),
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads())$count) > 0, "Input a valid package name"))

    metaExpr({
      ..(downloads()) %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
  })
}
```