

# ➤ Shiny application development

*Migale facility - 2023/03/20*

Sandra Dérozier - Mahendra Mariadassou - Cédric Midoux

*(Original slides from Amandine Velt and Cédric Midoux)*

# Practical informations

- 9h30 - 17h00
- 2 breaks (morning and afternoon)
- Lunch break of 1 hour



# Ice breaking session

- Who are you?
  - Institution, laboratory, position...
- Why are you here?
  - What are your goals with Shiny?
  - Do you already have developments in progress?
- Have you ever used *Shiny*?



# Objectives

- After this training day, you will know:
  - General concepts of a Shiny app
  - Shiny application development
  - Shiny application deployment solutions



# Schedule

- **Morning**
  - 09h30 : Opening
  - 10h00 : Why Shiny?
  - 10h30 : Server/UI + input/output + Widgets
  - 12h00 : Layout + CSS
- **Afternoon**
  - 14h00 : Reactivity
  - 15h30 : Deployment
  - 16h30 : Conclusion + Q&A
  - 17h00 : End of day



# Why Shiny?



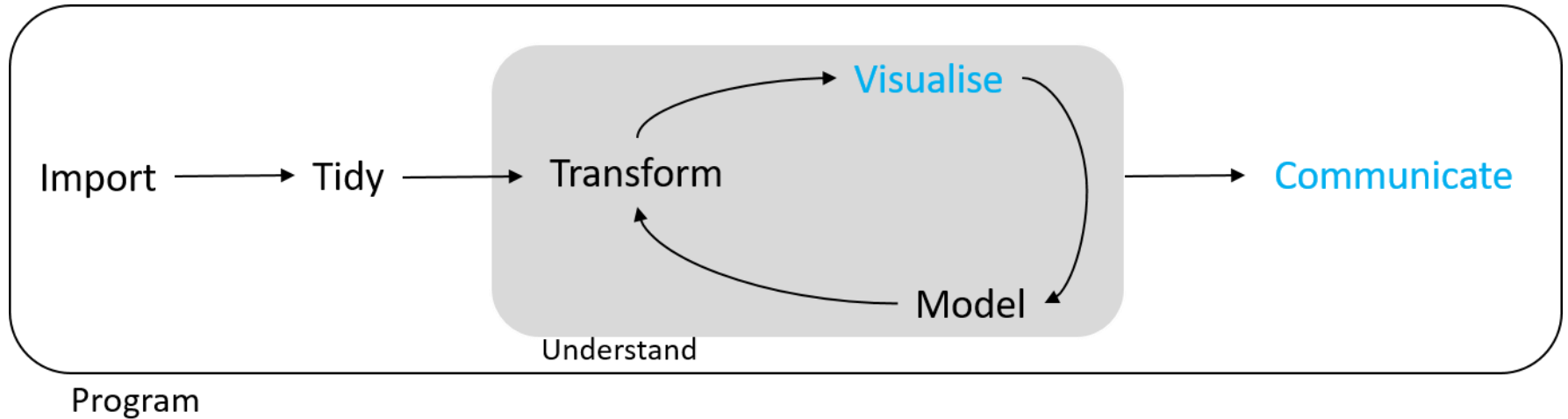
INRAE

Migale facility  
March 20, 2023 / Sandra Derozier - Mahendra Mariadassou - Cédric Midoux

migale

p. 6

# How to communicate one's results with R?



# How to communicate its results with R?

With R, to change a parameter, you have to **change the code** manually.

```
x <- faithful$waiting

# number of bins = 10
nbins <- 10
hist(x, breaks = nbins, col = "#75AADB",
     border = "white",
     xlab = "Waiting time to next eruption (in mins)",
     main = "Histogram of waiting times")

# number of bins = 50
nbins <- 50
hist(x, breaks = nbins, col = "#75AADB",
     border = "white",
     xlab = "Waiting time to next eruption (in mins)",
     main = "Histogram of waiting times")
```

We would like to have a **slider** to modify it **interactively!**

🔗 Shiny solution: <https://gallery.shinyapps.io/001-hello/>



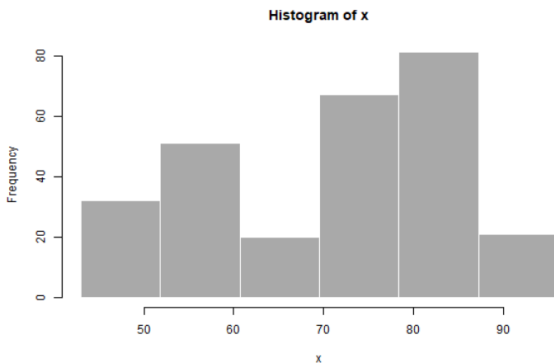


# How to communicate one's results with R?

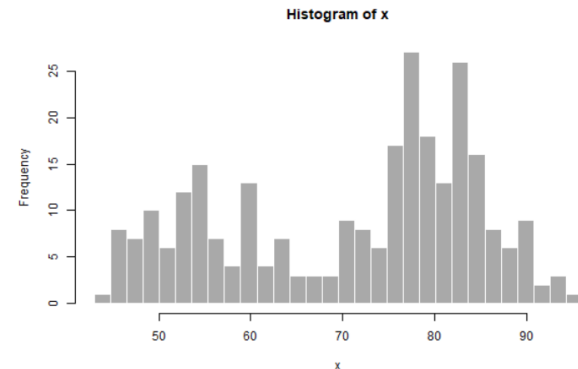
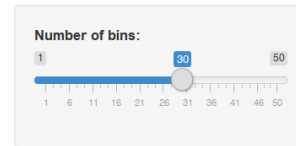
## Shiny brings reactivity

```
shiny::runExample("01_hello")
```

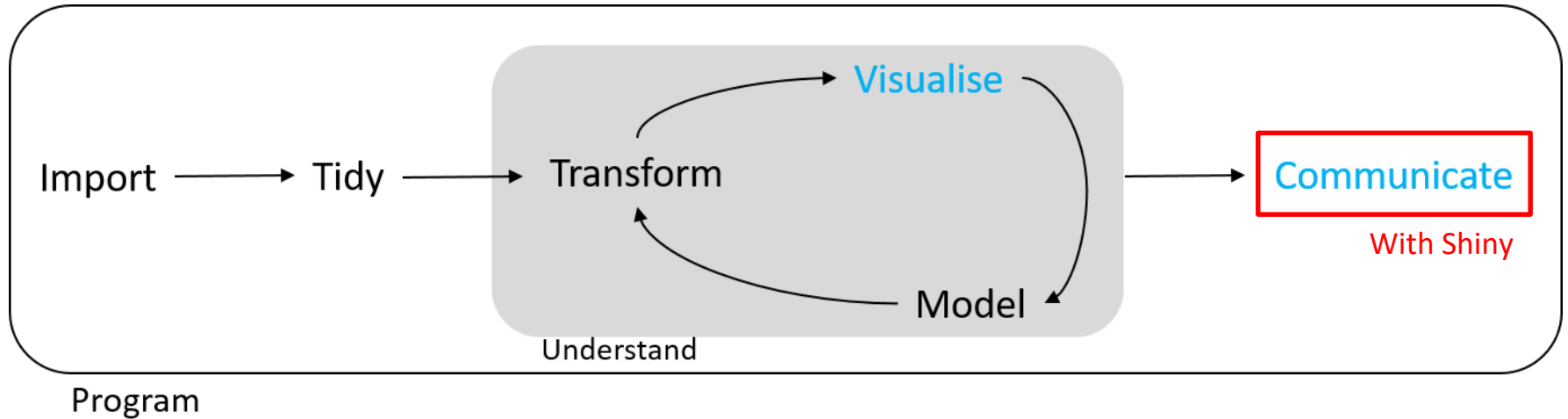
Old Faithful Geyser Data



Old Faithful Geyser Data



# How to communicate one's results with R?



# Shiny to ease access to data

- Defining features:
  - Facilitates dataviz and data mining...
  - ...through simple, readable and user-friendly interfaces
  - ...and dynamic outputs: graphs, tables, etc
  - Reduces the required amount of code to learn / write (if you already know R)

# How to communicate its results with R?

## Share your raw code

- No context or comments
- Can be difficult to interpret
- No dependency management

## Use Rmarkdown

- Static report retracing analysis
- No re-execution

## Implement a Shiny application

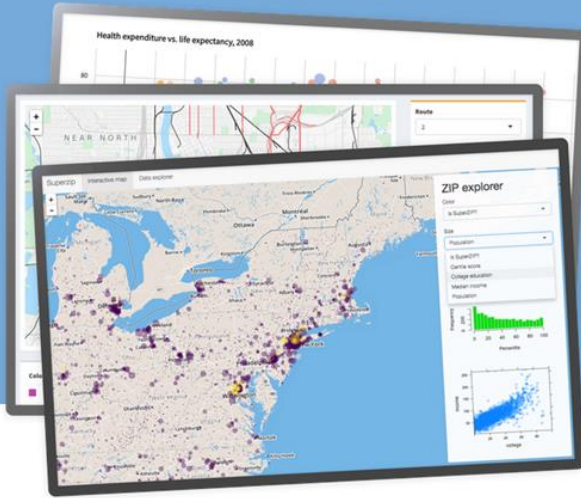
- User-friendly graphical interface
- Editable parameters and re-executable code
- Ability to export plots and reports
- User should install R and packages

## Deploy the Shiny app on a server

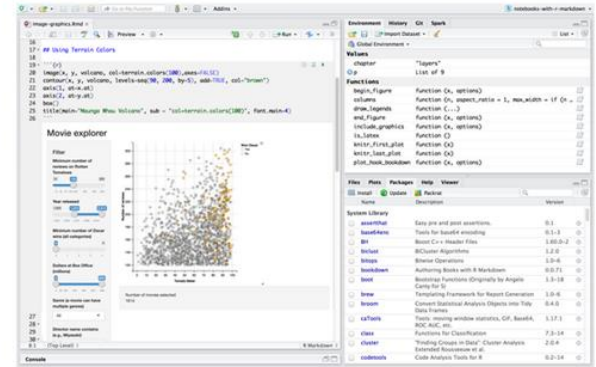
- Applications run on a dedicated server

## Interact. Analyze. Communicate.

Take a fresh, interactive approach to telling your data story with Shiny. Let users interact with your data and your analysis. And do it all with R.



Shiny is an R package that makes it easy to build interactive web apps straight from R. You can host standalone apps on a webpage or embed them in R Markdown documents or build dashboards. You can also extend your Shiny apps with CSS themes, htmlwidgets, and JavaScript actions.



## Learn Shiny

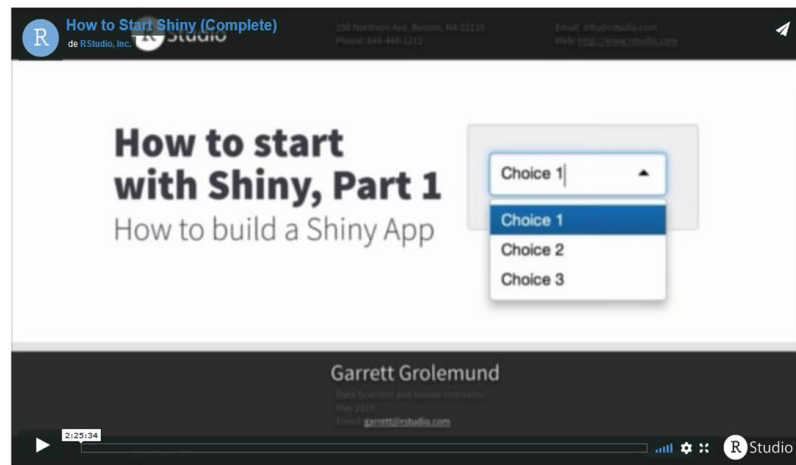
The video and written tutorials on this page are primarily designed for users who are new to Shiny and want a guided introduction.

If you use Shiny on a regular basis, you may want to skip these tutorials and visit the [articles](#) section where we cover individual Shiny topics at a more advanced level.

### Video tutorials

#### How to Start Shiny tutorial

The How to Start Shiny video series will take you from R programmer to Shiny developer. Watch the complete tutorial, or jump to a specific chapter by clicking a link below. The entire tutorial is two hours and 25 minutes long. Download the slides and exercises [here](#).



## Tutorials

<https://shiny.rstudio.com/tutorial/>

## Gallery

Welcome to the Shiny Gallery! Below you can find a myriad of Shiny apps to be inspired by and to learn from. We have organized the apps in two main categories:

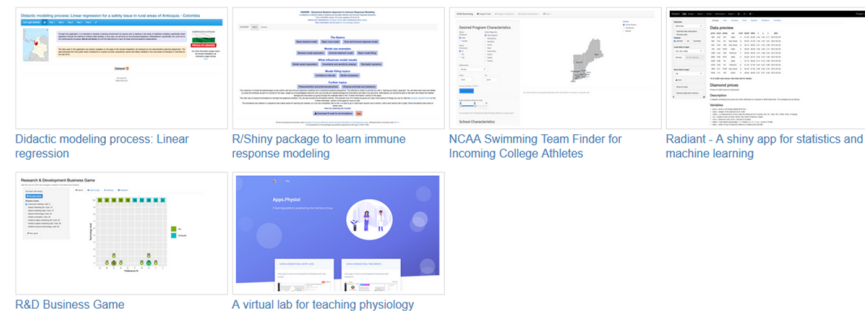
- [Shiny User Showcase](#) comprised of contributions from the Shiny app developer community.
- [Shiny Demos](#) that are designed to highlight specific features of shiny, the package.

### Shiny User Showcase

The Shiny User Showcase is comprised of contributions from the Shiny app developer community. The apps are categorized into application areas and presented with a brief description, tags, and for many, the source code. Note that many of these apps are winners and honorable mentions of our annual Shiny contest!

#### Education

Apps designed for teaching



## Gallery

<https://shiny.rstudio.com/gallery/>



# Let's play with Shiny!



- Pick one of these Shiny apps and interact with it
- Try to see how the app behaves and what can be done with it
  - [ECDC-vaccination](#) et [dataCoViz](#) de [@flodebarre](#)
  - [Covid19-tracker](#) de Edward Parker
  - [Movie explorer](#)
  - [Legislatives2017](#)
  - Any one from [the gallery](#)

# How does it work?



INRAE

Migale facility  
March 20, 2023 / Sandra Derozier - Mahendra Mariadassou - Cédric Midoux

migale

p. 16





# How does it work?

→ TP1

- Create a Shiny application with RStudio
  - File > New File > Shiny Web App
- Give a name to your first app and save it in the folder of your choice with the option "Single File"
- Browse, explore and execute the code
- Identify each element and their interactions

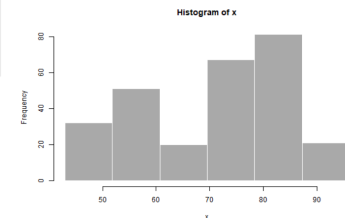
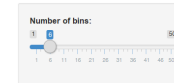
## ui.R

```
1 # This is the user-interface definition of a Shiny web application. You can
2 # run the application by clicking 'Run App' above.
3 #
4 # Find out more about building applications with Shiny here:
5 #
6 # http://shiny.rstudio.com/
7 #
8
9 library(shiny)
10 library(shinythemes)
11
12 # Define UI for application that draws a histogram
13 ui <- fluidPage(theme=shinytheme("flatly"),
14
15   # Application title
16   titlePanel("Old Faithful Geyser Data"),
17
18   # Sidebar with a slider input for number of bins
19   sidebarLayout(
20     sidebarPanel(
21       sliderInput("bins",
22                 "Number of bins:",
23                 min = 1,
24                 max = 50,
25                 value = 30)
26     ),
27
28     # Show a plot of the generated distribution
29     mainPanel(
30       plotOutput("distPlot")
31     )
32   )
33 )
34
```

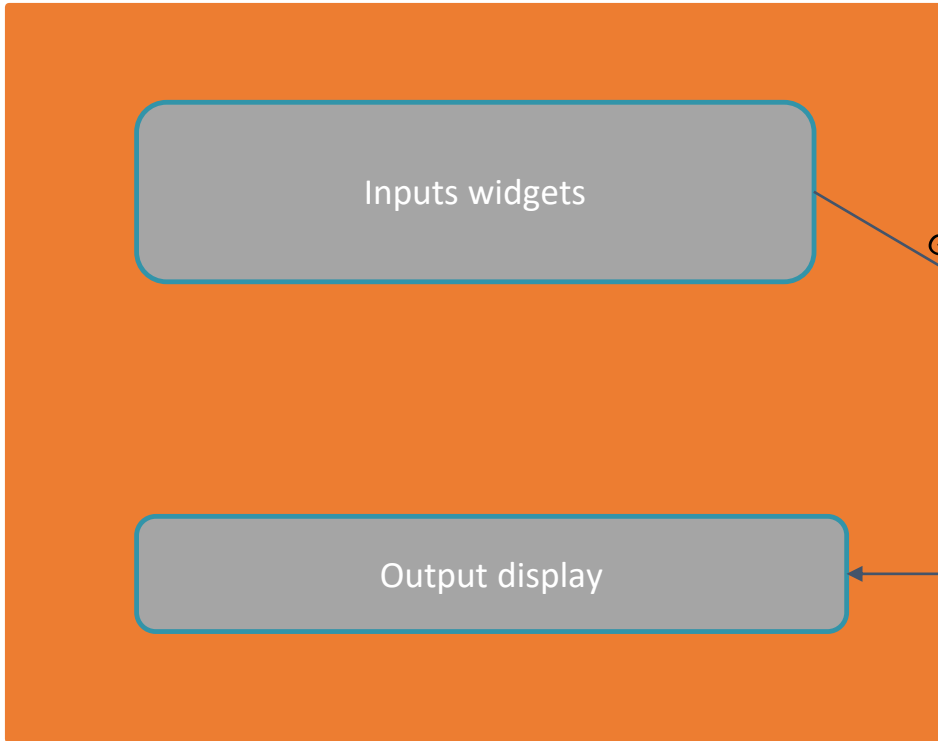
## server.R

```
1 #
2 # This is the server logic of a Shiny web application. You can run the
3 # application by clicking 'Run App' above.
4 #
5 # Find out more about building applications with Shiny here:
6 #
7 # http://shiny.rstudio.com/
8 #
9
10 library(shiny)
11
12 # Define server logic required to draw a histogram
13 server <- function(input, output, session) {
14
15   output$distPlot <- renderPlot({
16
17     # generate bins based on input$bins from ui.R
18     x <- faithful[, 2]
19     bins <- seq(min(x), max(x), length.out = input$bins + 1)
20
21     # draw the histogram with the specified number of bins
22     hist(x, breaks = bins, col = 'darkgray', border = 'white')
23
24   })
25
26 }
27
```

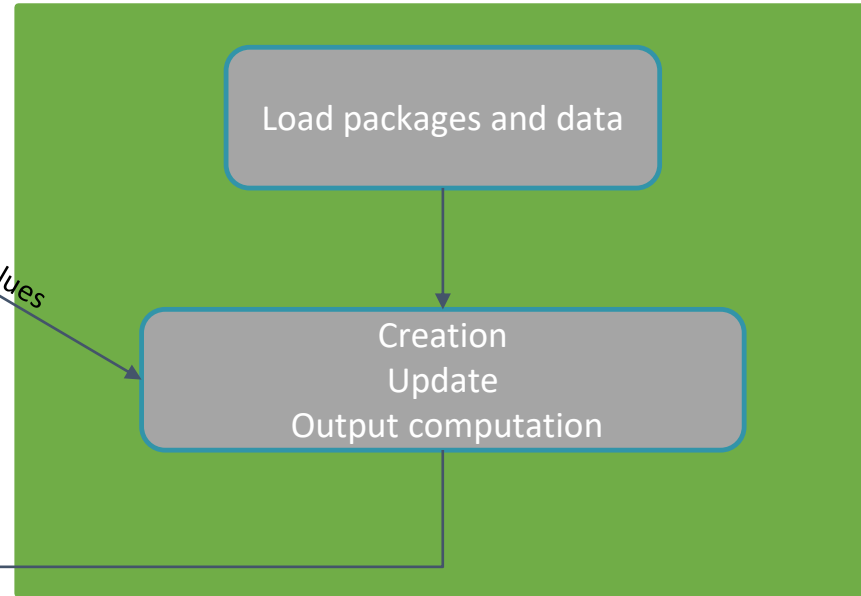
Old Faithful Geyser Data



ui.R

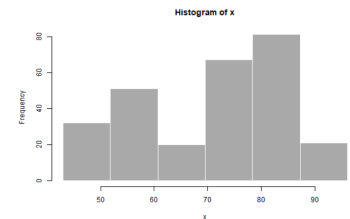
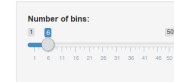


server.R



Get values

Old Faithful Geyser Data



INRAE

ui.R

```
*Input(inputId = « in1 »,  
       label = «lab»),
```

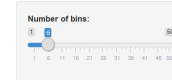
```
*Output(outputId = « out1 »)
```

server.R

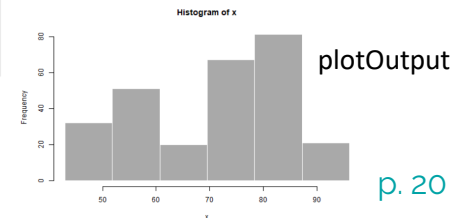
```
library(...)
```

```
output$out1 <-  
  render*(input$in1)
```

Old Faithful Geyser Data



sliderInput



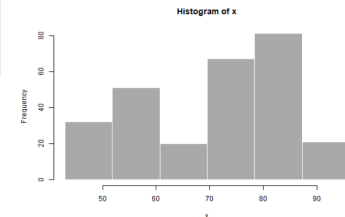
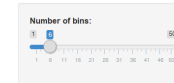
## ui.R

```
1 # This is the user-interface definition of a Shiny web application. You can
2 # run the application by clicking 'Run App' above.
3 #
4 # Find out more about building applications with Shiny here:
5 #
6 # http://shiny.rstudio.com/
7 #
8
9 library(shiny)
10 library(shinythemes)
11
12 # Define UI for application that draws a histogram
13 ui <- fluidPage(theme=shinytheme("flatly"),
14
15   # Application title
16   titlePanel("Old Faithful Geyser Data"),
17
18   # Sidebar with a slider input for number of bins
19   sidebarLayout(
20     sidebarPanel(
21       sliderInput("bins",
22                 "Number of bins:",
23                 min = 1,
24                 max = 50,
25                 value = 30)
26     ),
27
28     # Show a plot of the generated distribution
29     mainPanel(
30       plotOutput("distPlot")
31     )
32   )
33 )
34
```

## server.R

```
1 #
2 # This is the server logic of a Shiny web application. You can run the
3 # application by clicking 'Run App' above.
4 #
5 # Find out more about building applications with Shiny here:
6 #
7 # http://shiny.rstudio.com/
8 #
9
10 library(shiny)
11
12 # Define server logic required to draw a histogram
13 server <- function(input, output, session) {
14
15   output$distPlot <- renderPlot({
16
17     # generate bins based on input$bins from ui.R
18     x <- faithful[, 2]
19     bins <- seq(min(x), max(x), length.out = input$bins + 1)
20
21     # draw the histogram with the specified number of bins
22     hist(x, breaks = bins, col = 'darkgray', border = 'white')
23   })
24
25 }
26
27
```

Old Faithful Geyser Data



# Various types of inputs widgets

ui.R

```
1 # This is the user-interface definition of a Shiny web application. You can
2 # run the application by clicking 'Run App' above.
3 #
4 # Find out more about building applications with Shiny on the web page:
5 #
6 #   http://shiny.rstudio.com/
7 #
8
9 library(shiny)
10 library(shinythemes)
11
12 # Define UI for application that draws a histogram
13 ui <- fluidPage(theme=shinytheme("flatly"),
14
15   # Application title
16   titlePanel("Old Faithful Geyser Data"),
17
18   # Sidebar with a slider input for number of bins
19   sidebarLayout(
20     sidebarPanel(
21       sliderInput("bins",
22                 "Number of bins:",
23                 min = 1,
24                 max = 50,
25                 value = 30)
26     ),
27
28     # Show a plot of the generated distribution
29     mainPanel(
30       plotOutput("distPlot")
31     )
32   )
33 )
34
```

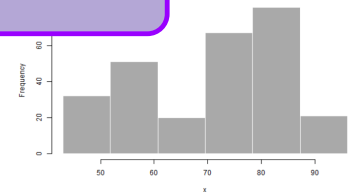
Inputs widgets

*Which ones  
would you need?*

server.R

```
1 # This is the server-side definition of a Shiny web application. You can run the
2 # application by clicking 'Run App' above.
3 #
4 # Find out more about building applications with Shiny on the web page:
5 #
6 #   http://shiny.rstudio.com/
7 #
8
9 # Define server logic to draw a histogram
10 server <- function(input, output, session) {
11
12   # Draw a histogram with the requested number of bins
13   output$distPlot <- renderPlot({
14     # Use input from ui.R
15     nbins = input$bins - 1
16     hist(x, nbins, border = 'white')
17   })
18 }
19
```

Histogram of x



INRAE

# Various types of inputs widgets

ui.R

```
1 # This is the user-interface definition of a Shiny web application. You can
2 # run the application by clicking 'Run App' above.
3 #
4 # Find out more about building applications with Shiny on the website:
5 # http://shiny.rstudio.com/
6 #
7 #
8
9 library(shiny)
10 library(shinythemes)
11
12 # Define UI for application that draws a histogram
13 ui <- fluidPage(theme=shinytheme("flatly"),
14
15 # Application title
16 titlePanel("Old Faithful Geyser Data"),
17
18 # Sidebar with a slider input for number of bins
19 sidebarLayout(
20   sidebarPanel(
21     sliderInput("bins",
22               "Number of bins:",
23               min = 1,
24               max = 50,
25               value = 30)
26   ),
27
28 # Show a plot of the generated distribution
29   mainPanel(
30     plotOutput("distPlot")
31   )
32 )
33 )
34 )
```

server.R

## Inputs widgets <https://shiny.rstudio.com/gallery/widget-gallery.html>

Access the current value of an input object with `input$<inputId>`. Input values are reactive.



`actionButton(inputId, label, icon, ...)`

Link

`actionLink(inputId, label, icon, ...)`

- Choice 1
- Choice 2
- Choice 3

`checkboxGroupInput(inputId, label, choices, selected, inline)`

- Check me

`checkboxInput(inputId, label, value)`



`dateInput(inputId, label, value, min, max, format, startview, weekstart, language)`



`dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)`

Choose File

`fileInput(inputId, label, multiple, accept)`



`numericInput(inputId, label, value, min, max, step)`



`passwordInput(inputId, label, value)`



`radioButtons(inputId, label, choices, selected, inline)`



`selectInput(inputId, label, choices, selected, multiple, selectize, width, size) (also selectizeInput())`



`sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)`



`submitButton(text, icon)`  
(Prevents reactions across entire app)



`textInput(inputId, label, value)`

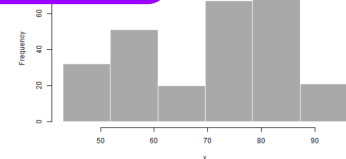
application. You can run the

with Shiny here:

histogram

```
m ui.R
= input$bins - 1)
number of bins
, border = 'white')
```

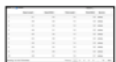
Histogram of x



# Various types of outputs

## Outputs

Outputs - render\*() and \*Output() functions work together to add R output to the UI



**DT:** `renderDataTable(expr, options, callback, escape, env, quoted)`



`dataTableOutput(outputId, icon, ...)`



`renderImage(expr, env, quoted, deleteFile)`

`imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)`



`renderPlot(expr, width, height, res, ..., env, quoted, func)`

`plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)`

`renderPrint(expr, env, quoted, func, width)`

`verbatimTextOutput(outputId)`

`renderTable(expr, ..., env, quoted, func)`

`tableOutput(outputId)`

foo

`renderText(expr, env, quoted, func)`

`textOutput(outputId, container, inline)`



`renderUI(expr, env, quoted, func)`

`uiOutput(outputId, inline, container, ...)`

`& htmlOutput(outputId, inline, container, ...)`

```

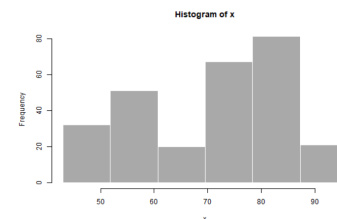
28 # Show a plot of the generated distribution
29 mainPanel(
30   plotOutput("distPlot")
31 )
32 )
33 )
34
  
```

## server.R

```

1 #
2 # This is the server logic of a Shiny web application. You can run the
3 # application by clicking 'Run App' above.
4 #
5 # Find out more about building applications with Shiny here:
6 #
7 #   http://shiny.rstudio.com/
8 #
9
10 library(shiny)
11
12 # Define server logic required to draw a histogram
13 server <- function(input, output, session) {
14
15   output$distPlot <- renderPlot({
16
17     # generate bins based on input$bins from ui.R
18     x <- faithful[, 2]
19     bins <- seq(min(x), max(x), length.out = input$bins + 1)
20
21     # draw the histogram with the specified number of bins
22     hist(x, breaks = bins, col = 'darkgray', border = 'white')
23   })
24
25
26 }
27
  
```

Old Faithful Geyser Data





# Practical work



→ **TP2:** Add text

→ **TP3:** Pick a title

→ **TP4:** Add color

# Layout



INRAE

Migale facility  
March 20, 2023 / Sandra Derozier - Mahendra Mariadassou - Cédric Midoux

migale

p. 26


# Layout

Add CSS / HTML / Javascript



- UI returns HTML document
- HTML tags can be added to the ui

```
ui <- fluidPage(  
  h1("Header 1"),  
  hr(),  
  br(),  
  p(strong("bold")),  
  p(em("italic")),  
  p(code("code")),  
  a(href="", "link"),  
  HTML("<p>Raw html</p>")  
)
```



Header 1

bold

italic

code

link

Raw html

```
fluidPage(  
  textInput("a", "")  
)  
## <div class="container-fluid">  
## <div class="form-group shiny-input-container">  
## <label for="a"></label>  
## <input id="a" type="text"  
## class="form-control" value=""/>  
## </div>  
## </div>
```

Returns HTML



- To include a CSS file, use the **includeCSS()** function and place the file in the **www** folder

```
includeCSS("styles.css"),
```



# Layout

ui.R

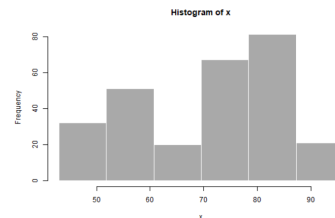
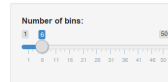
```
1 # This is the user-interface definition of a Shiny web application. You can
2 # run the application by clicking the button in the R Studio interface.
3 #
4 # Find out more about building applications with Shiny here:
5 # http://shiny.rstudio.com/
6 #
7 #
8
9 library(shiny)
10 library(shinythemes)
11
12 # Define UI for application that draws a histogram
13 ui <- fluidPage(theme=shinytheme("flatly"),
14
15   # Application title
16   titlePanel("Old Faithful Geyser Data"),
17
18   # Sidebar with a slider input for number of bins
19   sidebarLayout(
20     sidebarPanel(
21       sliderInput("bins",
22                 "Number of bins:",
23                 min = 1,
24                 max = 50,
25                 value = 30)
26     ),
27
28     # Show a plot of the generated distribution
29     mainPanel(
30       plotOutput("distPlot")
31     )
32   )
33 )
34
```

fluidPage() fits the width of the browser

server.R

```
1 #
2 # This is the server logic of a Shiny web application. You can run the
3 # application by clicking 'Run App' above.
4 #
5 # Find out more about building applications with Shiny here:
6 # http://shiny.rstudio.com/
7 #
8 #
9
10 library(shiny)
11
12 # Define server logic required to draw a histogram
13 server <- function(input, output, session) {
14
15   output$distPlot <- renderPlot({
16
17     # generate bins based on input$bins from ui.R
18     x <- faithful[, 2]
19     bins <- seq(min(x), max(x), length.out = input$bins + 1)
20
21     # draw the histogram with the specified number of bins
22     hist(x, breaks = bins, col = 'darkgray', border = 'white')
23
24   })
25
26 }
27
```

Old Faithful Geyser Data



INRAE

Migale facility

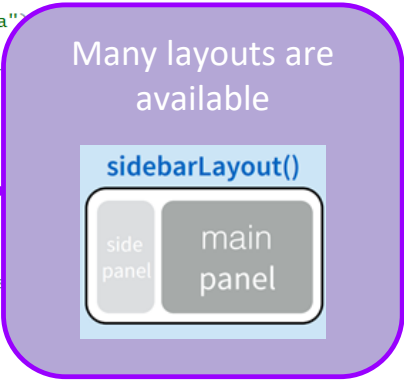
March 20, 2023 / Sandra Derozier - Mahendra Mariadassou - Cédric Midoux

# Layout

ui.R

```
1 # This is the user-interface definition of a Shiny web application. You can
2 # run the application by clicking 'Run App' above.
3 #
4 # Find out more about building applications with Shiny here:
5 #
6 #   http://shiny.rstudio.com/
7 #
8
9 library(shiny)
10 library(shinythemes)
11
12 # Define the UI for application that draws a histogram
13 ui <- fluidPage(theme=shinytheme("flatly"),
14
15   # Application title
16   titlePanel("Old Faithful Geyser Data"),
17
18   # Sidebar with a slider input for number of bins
19   sidebarLayout(
20     sidebarPanel(
21       sliderInput("bins",
22                 "Number of bins:",
23                 min = 1,
24                 max = 50,
25                 value = 30)
26     ),
27
28     # Show a plot of the generated distribution
29     mainPanel(
30       plotOutput("distPlot")
31     )
32   )
33 )
34 )
```

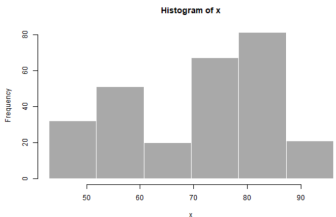
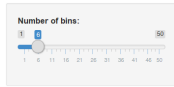
fluidPage() fits the width of the browser



server.R

```
1 #
2 # This is the server logic of a Shiny web application. You can run the
3 # application by clicking 'Run App' above.
4 #
5 # Find out more about building applications with Shiny here:
6 #
7 #   http://shiny.rstudio.com/
8 #
9
10 library(shiny)
11
12 # Define server logic required to draw a histogram
13 server <- function(input, output, session) {
14
15   output$distPlot <- renderPlot({
16
17     # generate bins based on input$bins from ui.R
18     x <- faithful[, 2]
19     bins <- seq(min(x), max(x), length.out = input$bins + 1)
20
21     # draw the histogram with the specified number of bins
22     hist(x, breaks = bins, col = 'darkgray', border = 'white')
23
24   })
25
26 }
27 )
```

Old Faithful Geyser Data



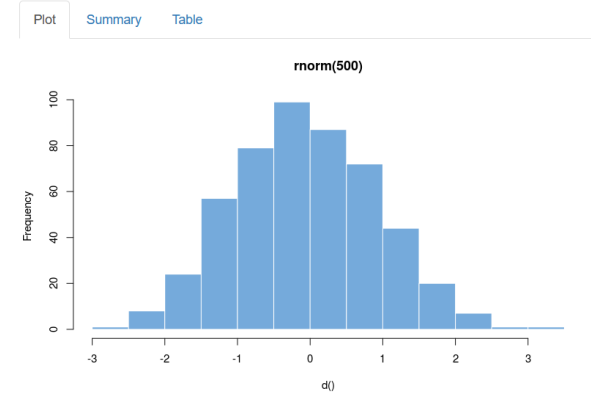
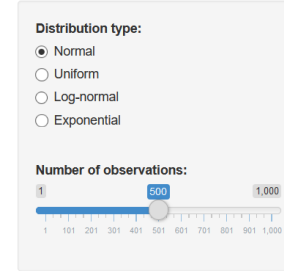
INRAE

# Layer tabPanels

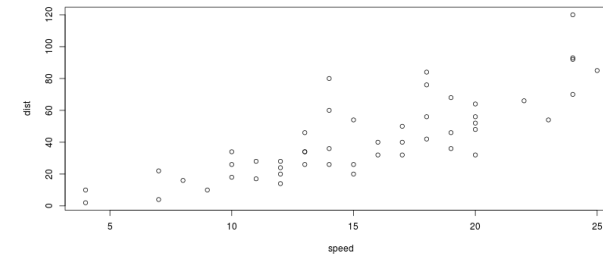
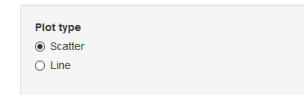
```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(...)  
    mainPanel(  
      tabsetPanel(  
        tabPanel("Plot", "content"),  
        tabPanel("Summary", "content"),  
        tabPanel("Table", "content")  
      )  
    )  
  )  
)
```

```
ui <- fluidPage(  
  navbarPage(title="Navbar!",  
    tabPanel("Plot", "content"),  
    tabPanel("Summary", "content"),  
    tabPanel("More", "content")  
  ))
```

## Tabsets



Navbar! Plot Summary More ▾



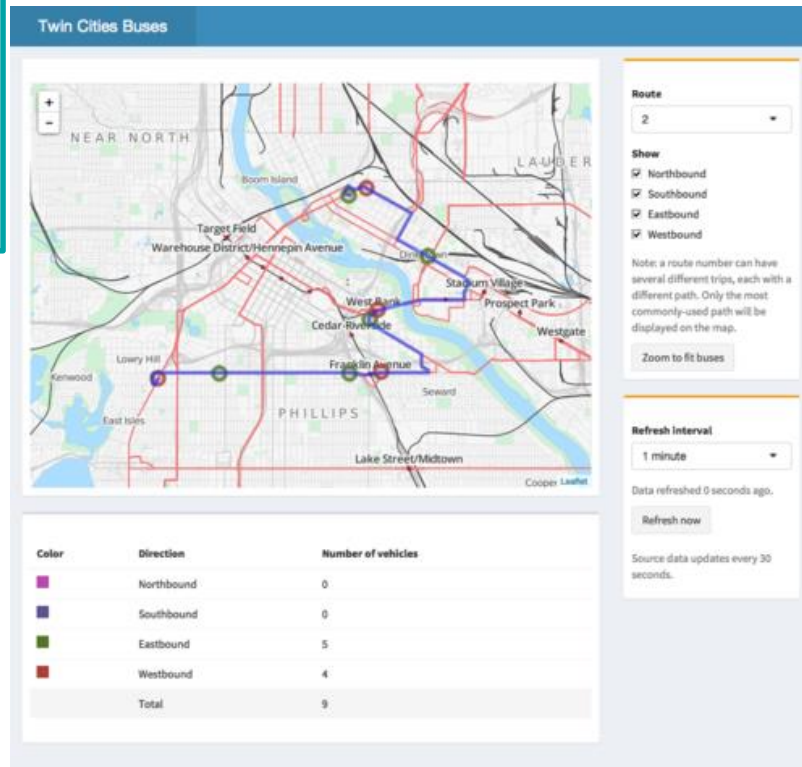
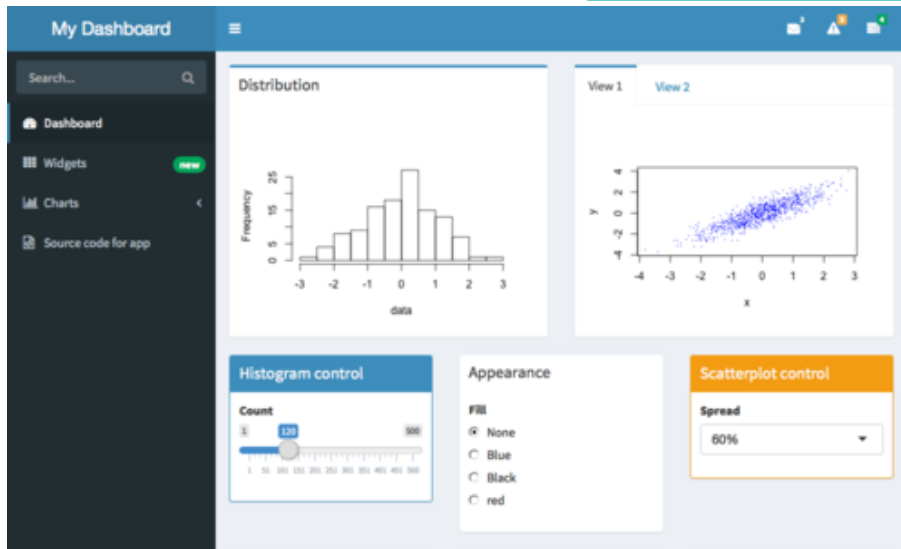
INRAE

# Shiny Dashboard

Using the shinydashboard package, you can easily create a dashboard type applications like:

```
library(shinydashboard)

dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
)
```





## 3 ways to include CSS:

- insert the style in the **head**
- insert the style at the tag level, through the **style** attribute, also known as inline CSS
- point to an external file with **includeCSS("style.css")**





## Insert the style in the head

ui.R

```
ui <- fluidPage(
  tags$head(
    tags$style(HTML("
      /* Change font of header text */
      h2 {
        font-family: 'monospace';
      }
      /* Make text visible on inputs */
      .shiny-input-container {
        color: #474747;
      }"))
  ),
  titlePanel("Old Faithful Geyser Data"),
  ...
)
```

# CSS

Insert the style at the tag level, through the **style** attribute, also known as inline CSS

ui.R

```
ui <- fluidPage(  
  
  # titlePanel("Old Faithful Geyser Data"),  
  h2("Old Faithful Geyser Data", style = "font-family: monospace;"),  
  
  ...  
)
```

## Point to an external file with `includeCSS("style.css")`

ui.R

```
ui <- fluidPage(
  includeCSS("www/style.css"),
  titlePanel("Old Faithful Geyser Data"),
  ...
)
```

style.css

```
/* Change font of header text */
h2 {
  font-family: 'monospace';
}

/* Make text visible on inputs */
.shiny-input-container {
  color: #474747;
}
```

# Practical work



→ **TP5:** Multi-tab applications and customization with CSS

→ **TP6:** Simulations

# Reactivity



INRAE

Migale facility  
March 20, 2023 / Sandra Derozier - Mahendra Mariadassou - Cédric Midoux

migale

p. 37

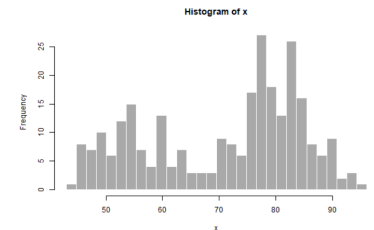
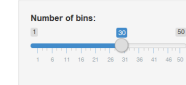
# The reactivity of Shiny app

input\$x  output\$y

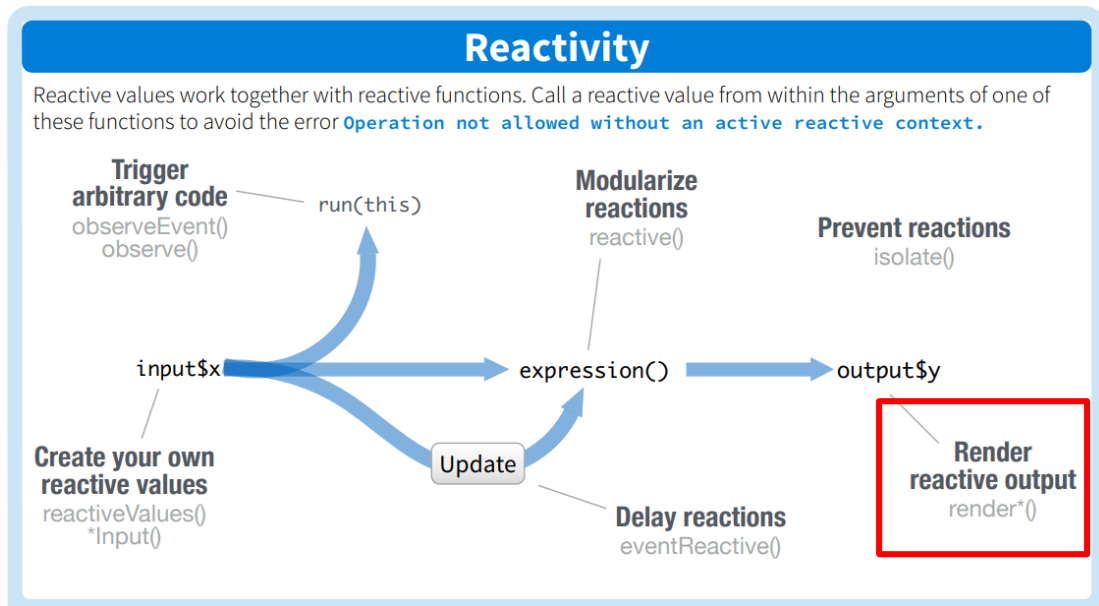
- Each change of input\$x (in the user interface) induces output\$y to be re-evaluated
  - The render\* functions family will re-execute the code as soon as one of the inputs it depends on changes
  - In the initial example, when input\$bins changes, the plot function is re-computed

```
14
15 -
16
17
18 # generate bins based on input$bins from ui.R
19 x <- faithful[, 2]
20 bins <- seq(min(x), max(x), length.out = input$bins + 1)
21
22 # draw the histogram with the specified number of bins
23 hist(x, breaks = bins, col = 'darkgray', border = 'white')
24 })
```

Old Faithful Geyser Data



# The reactivity graph of Shiny applications

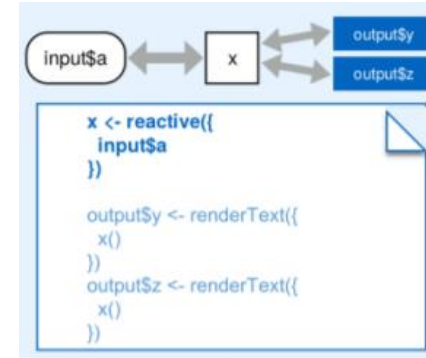
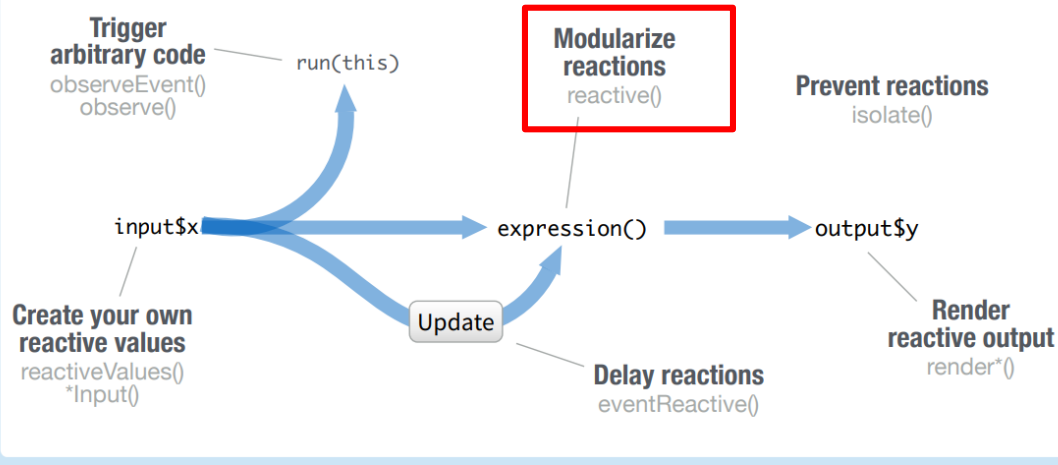


At each change of input\$x (in the user interface),  
output\$y is re-evaluated.

# Reactivity in Shiny applications

## Reactivity

Reactive values work together with reactive functions. Call a reactive value from within the arguments of one of these functions to avoid the error **Operation not allowed without an active reactive context**.



```
ui <- fluidPage(
  textInput("a", "", "A"),
  textInput("z", "", "Z"),
  textOutput("b"))

server <- function(input, output) {
  re <- reactive({
    paste(input$a, input$z)
  })
  output$b <- renderText({
    re()
  })
}
shinyApp(ui, server)
```

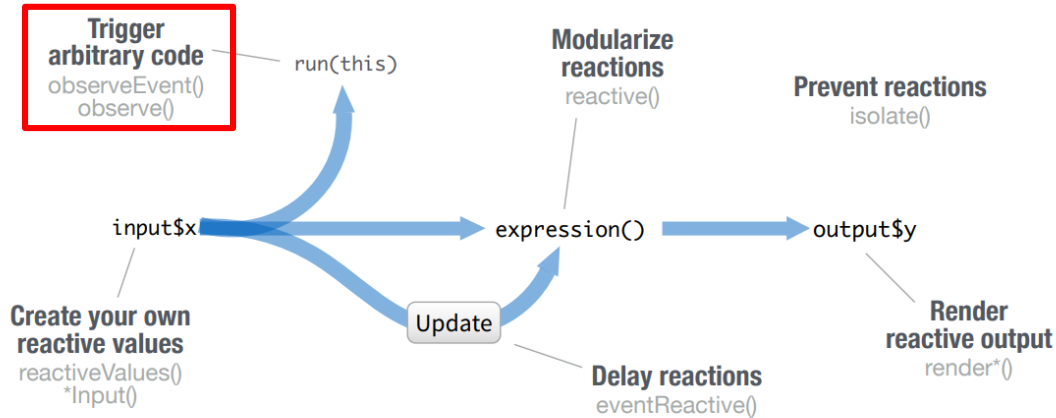
We can add *reactive* `expression()` for **caching values**.  
Call the expression with function syntax, like `re()`



# Reactivity in Shiny applications

## Reactivity

Reactive values work together with reactive functions. Call a reactive value from within the arguments of one of these functions to avoid the error **Operation not allowed without an active reactive context**.



Action

actionButton(inputId, label, icon, ...)

```
library(shiny)
ui <- fluidPage(
  textInput("a", "", "A"),
  actionButton("go", "Go")
)

server <- function(input, output) {
  observeEvent(input$go, {
    print(input$a)
  })
}

shinyApp(ui, server)
```

We run some only after an observe event like **press a button**.

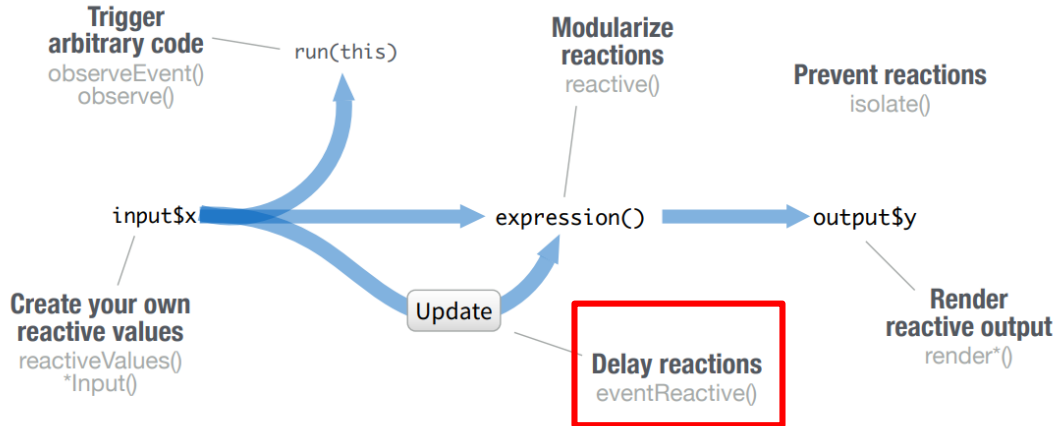


INRAE

# Reactivity in Shiny applications

## Reactivity

Reactive values work together with reactive functions. Call a reactive value from within the arguments of one of these functions to avoid the error **Operation not allowed without an active reactive context.**



In the same way, you can update an output when we observe an event.

Apply Changes

`submitButton(text, icon)`  
(Prevents reactions across entire app)

```
library(shiny)
ui <- fluidPage(
  textInput("a", "", "A"),
  actionButton("go", "Go"),
  textOutput("b")
)

server <- function(input, output) {
  re <- eventReactive(
    input$go, {input$a}
  )
  output$b <- renderText({
    re()
  })
}

shinyApp(ui, server)
```



INRAE

# Practical work



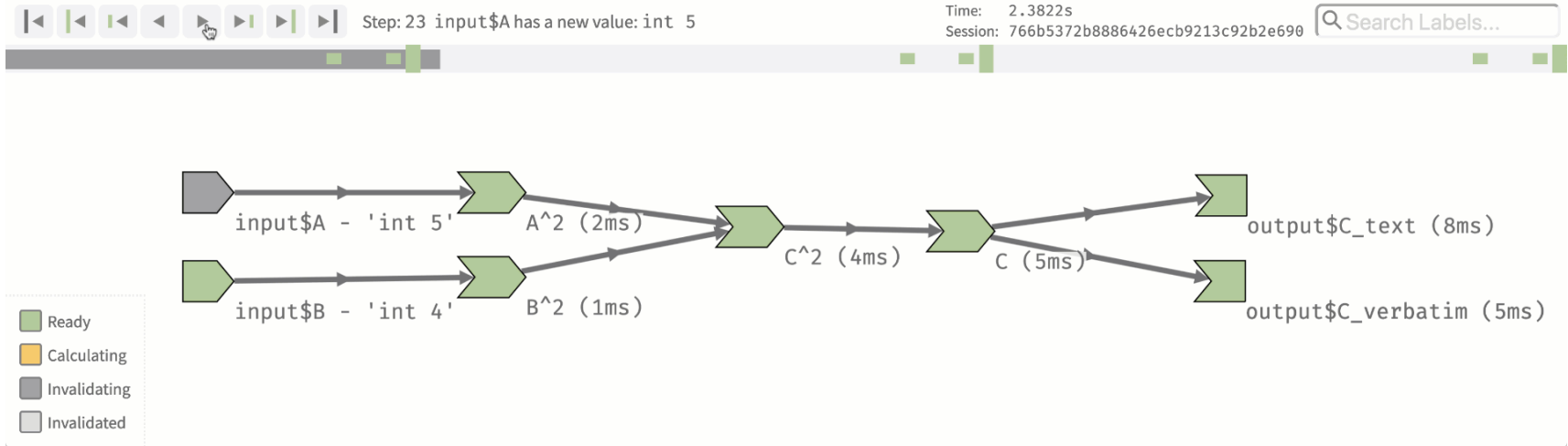
- **TP7**: Add new data
- **TP8**: Work with randomly generated data
- **TP9**: Specify the generating distribution
- **TP10**: Upload your data
- ...

# Exploring reactivity with reactlog

```
> reactlog::reactlog_enable()  
> runApp('reactlog.R')  
> shiny::reactlogShow()
```

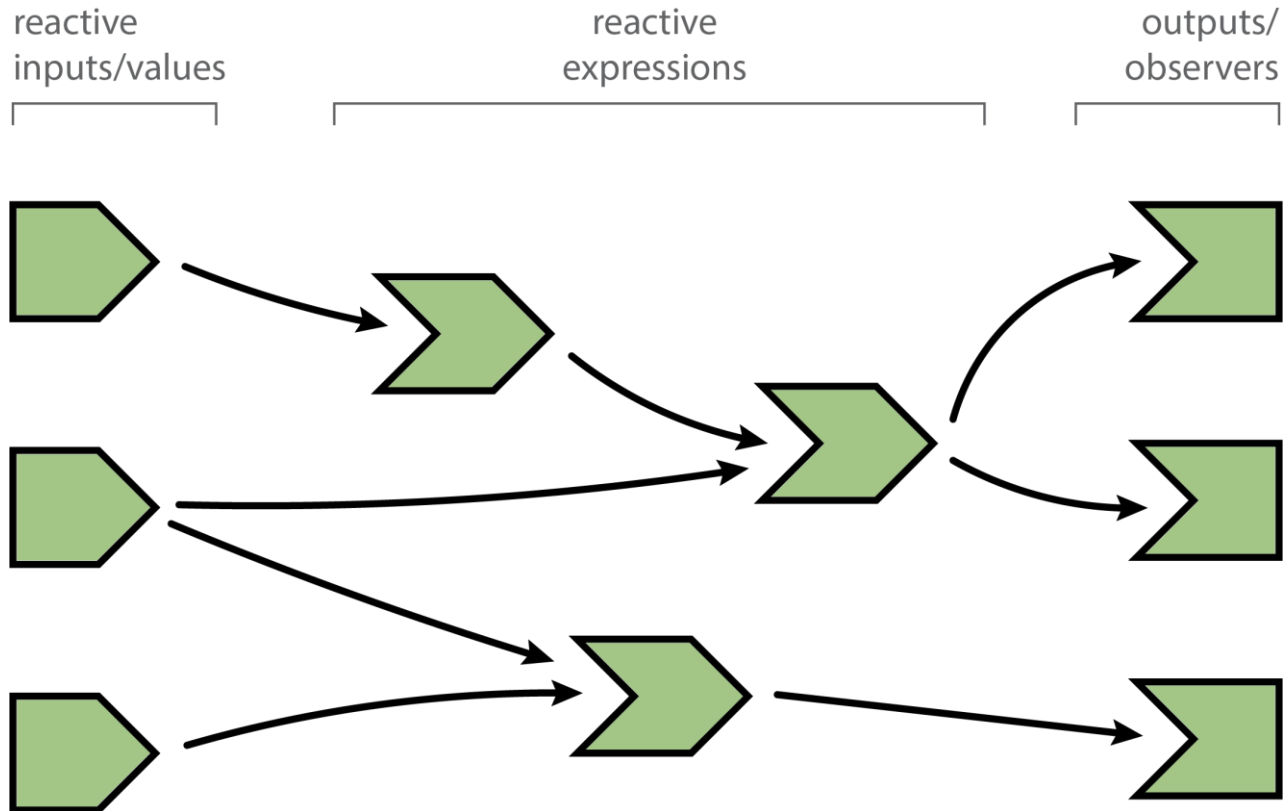
Allows you to play the app “step-by-step” and examine what’s going on under the hood.

# Exploring reactivity with reactlog

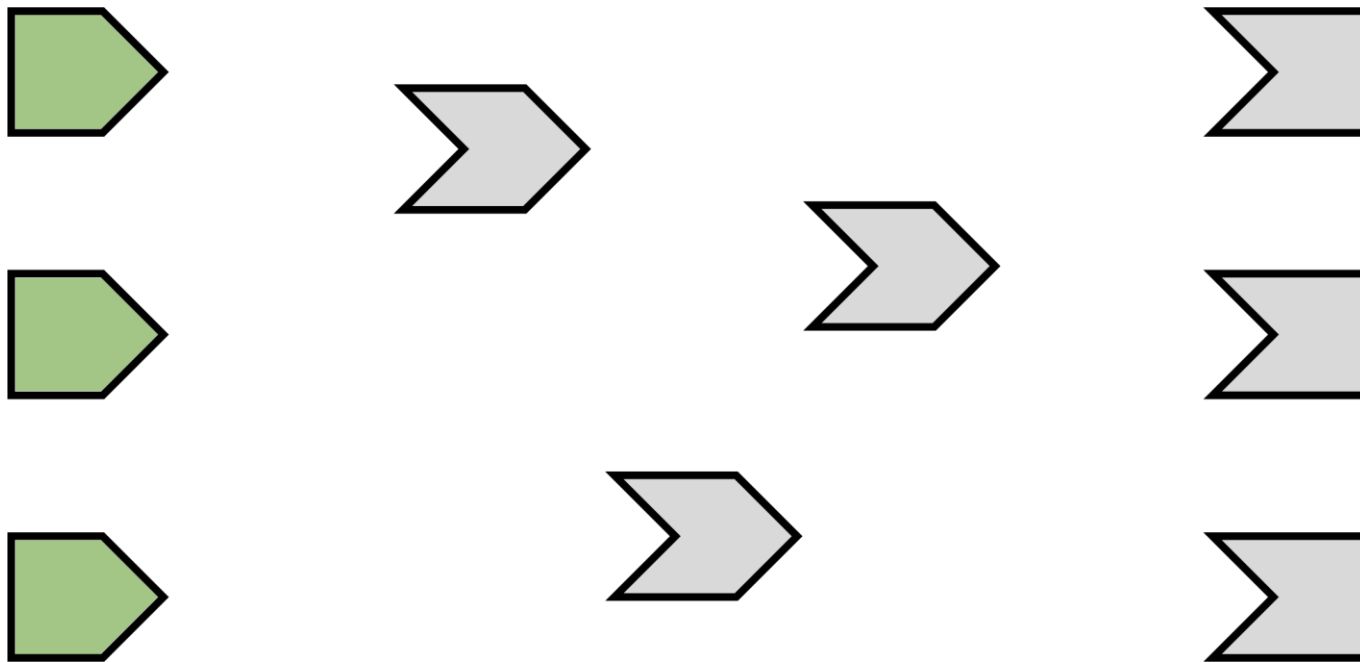


- [animated version](#)
- all images adapted from the [official documentation](#)

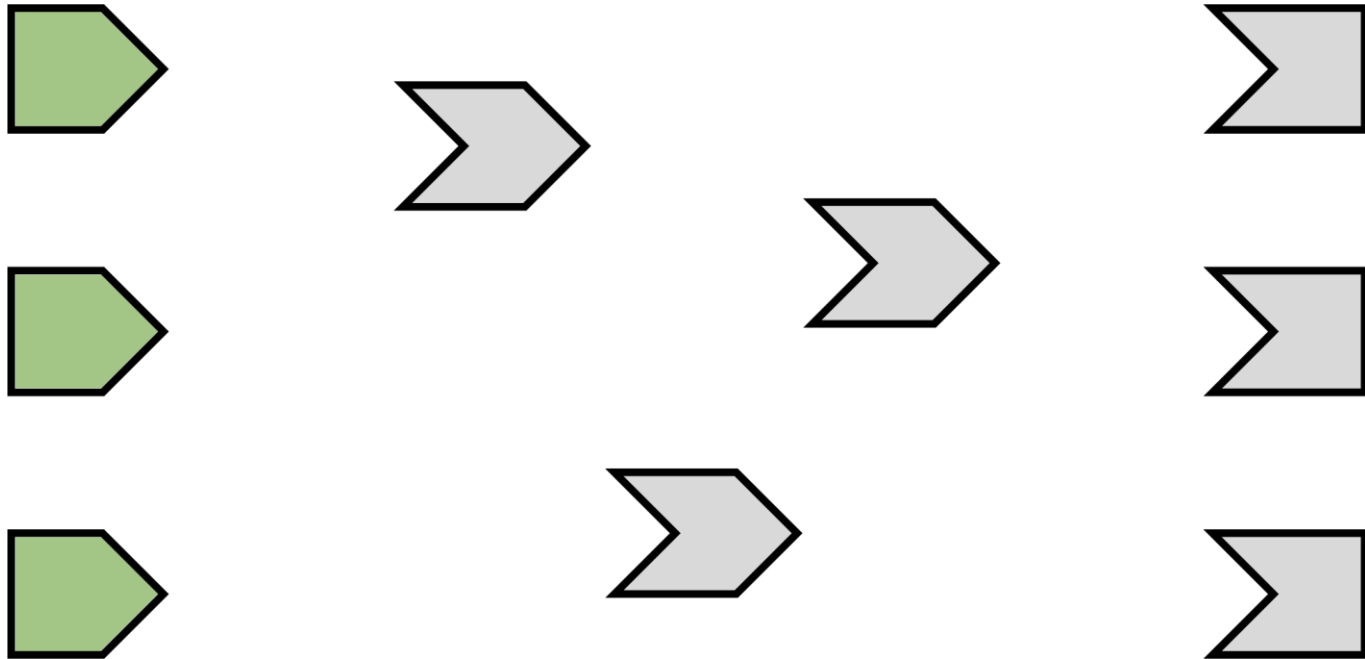
# Nodes in the graph



# A session begins

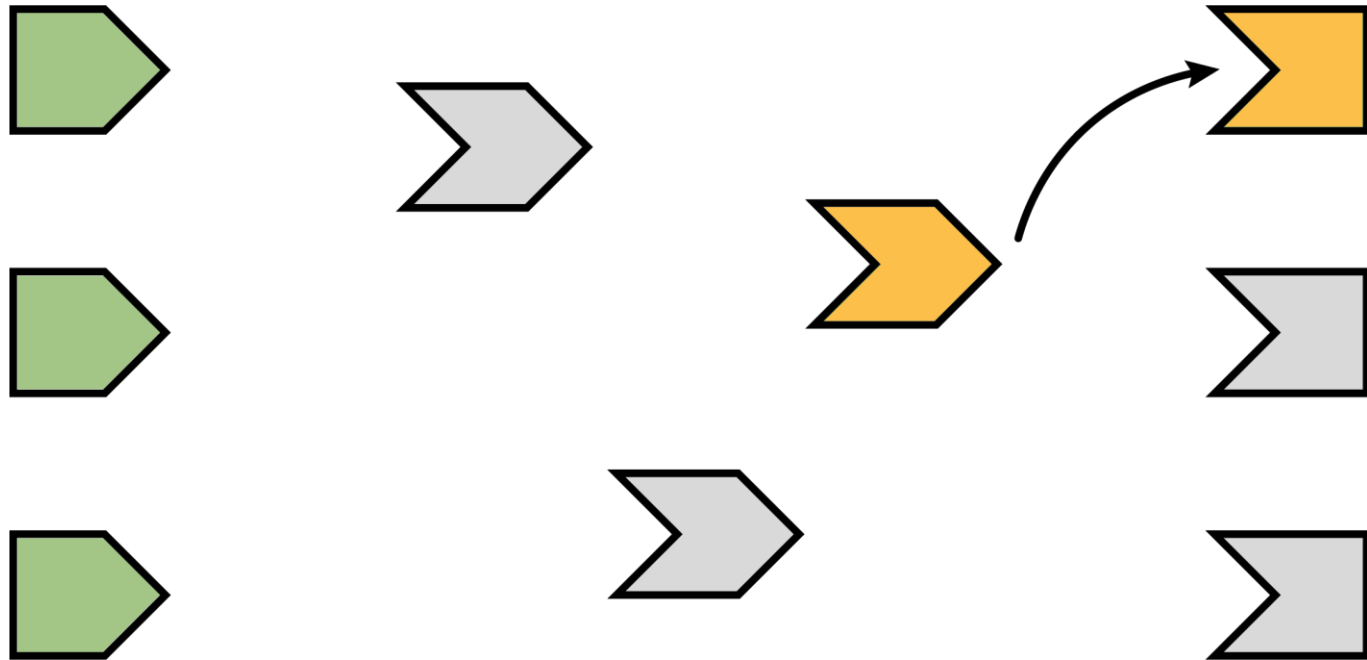


# Execution begins

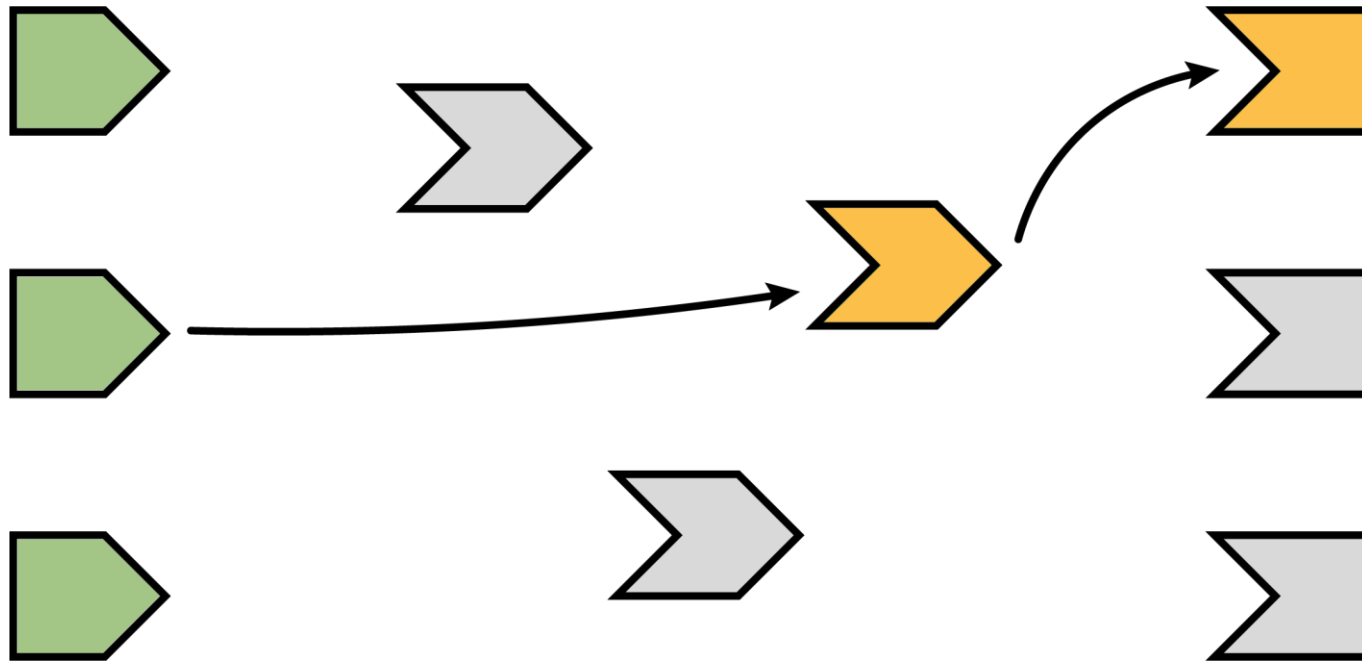




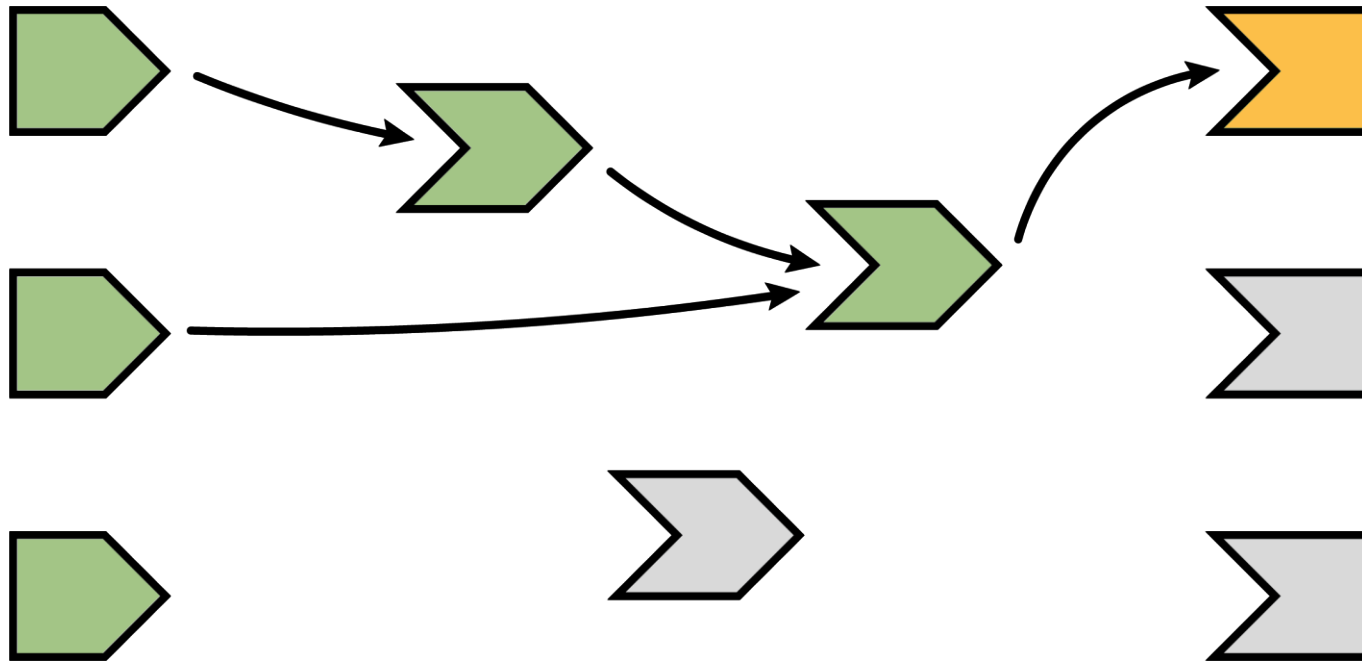
# Reading a reactive expression



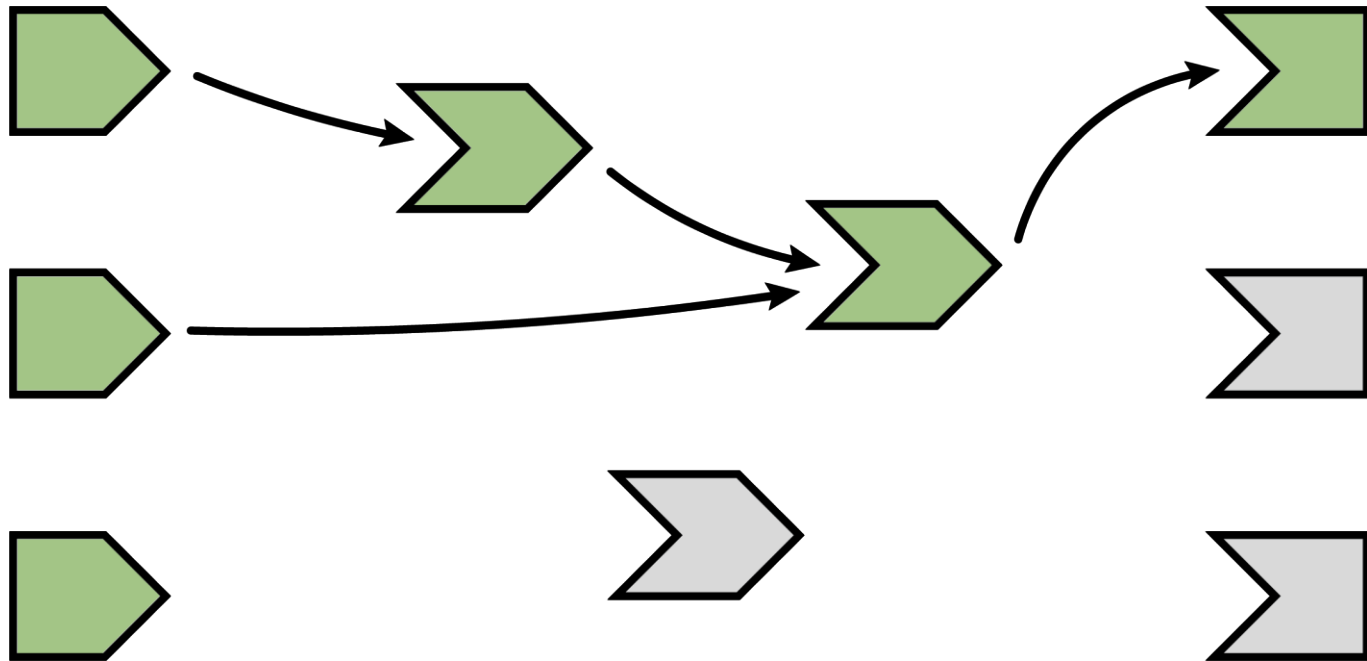
# Reading an input



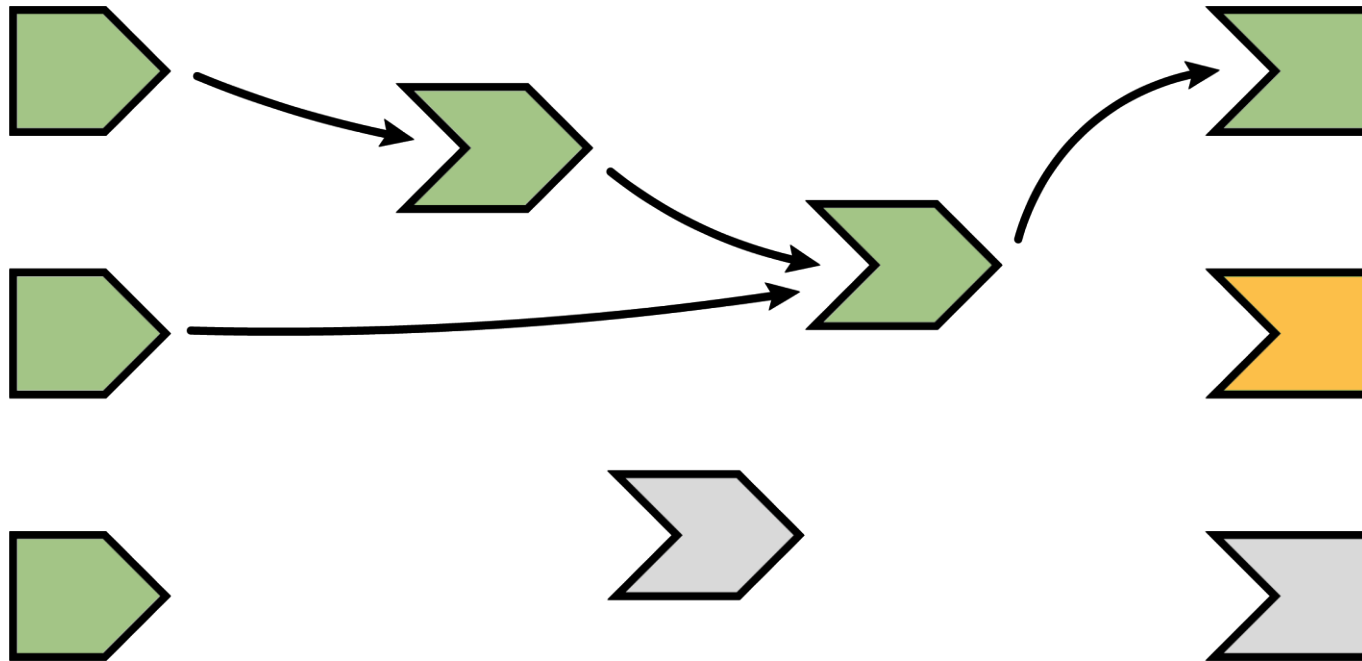
# Reactive expression completes



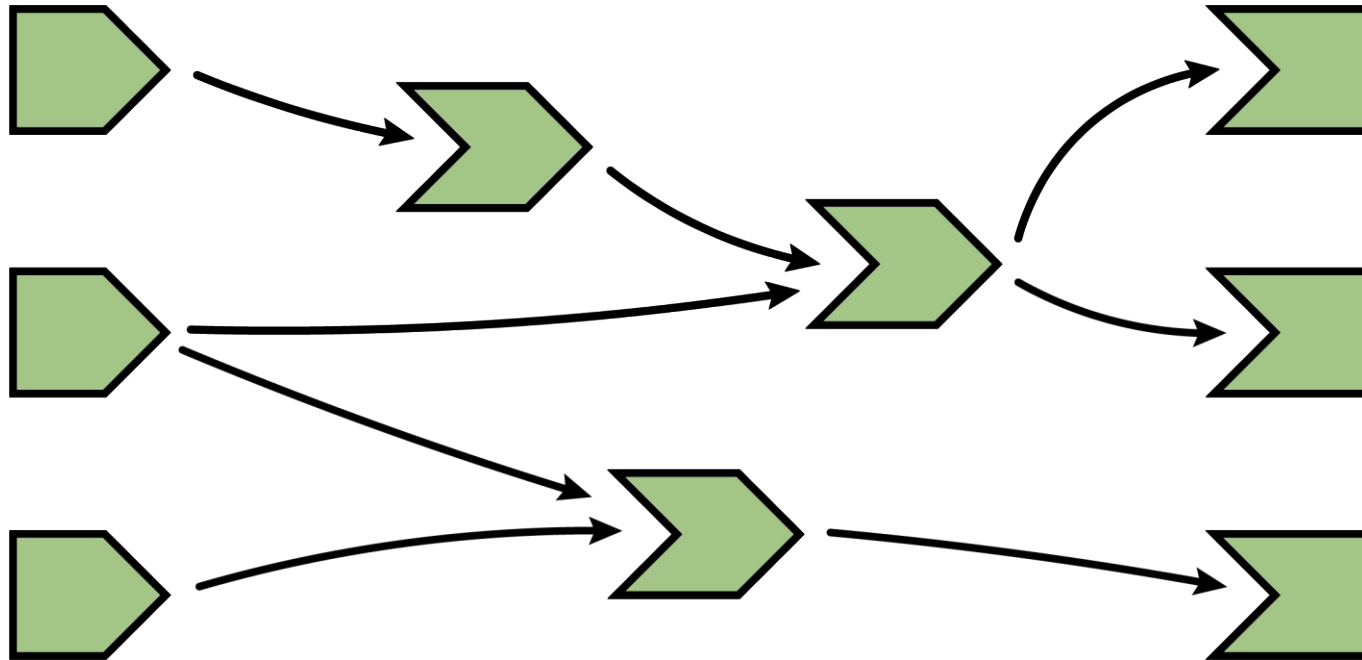
# Observer completes



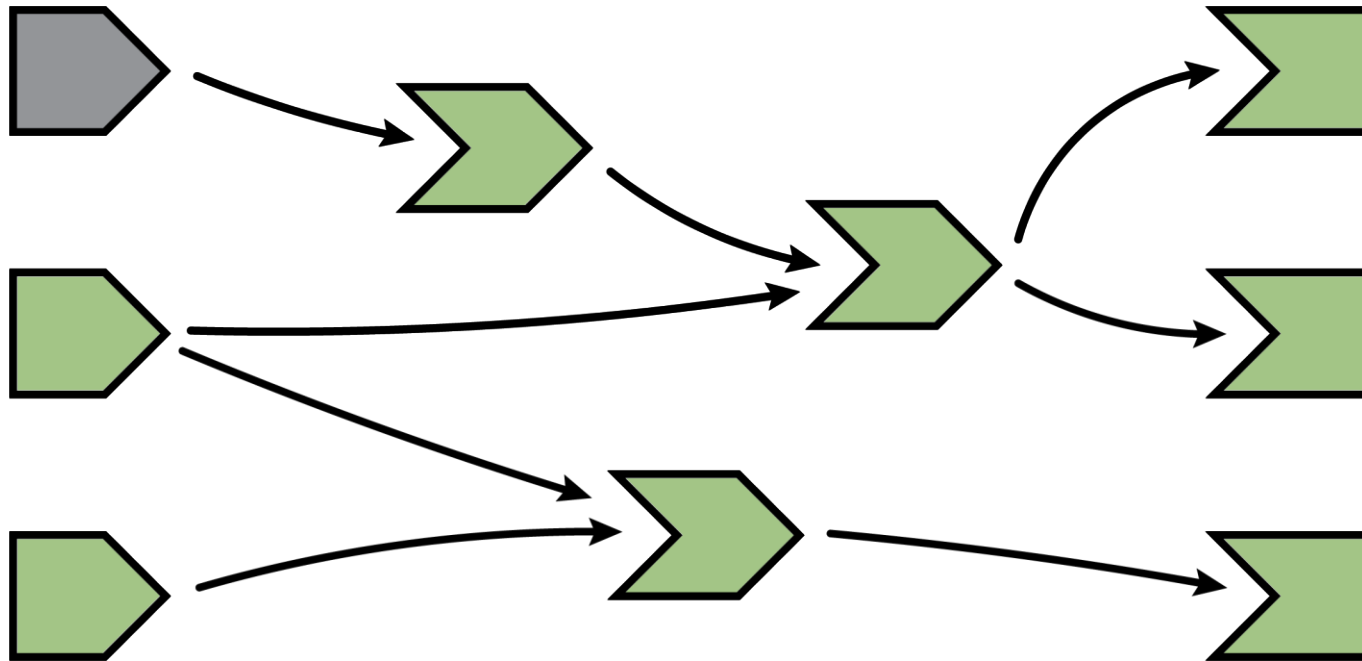
# The next observer executes



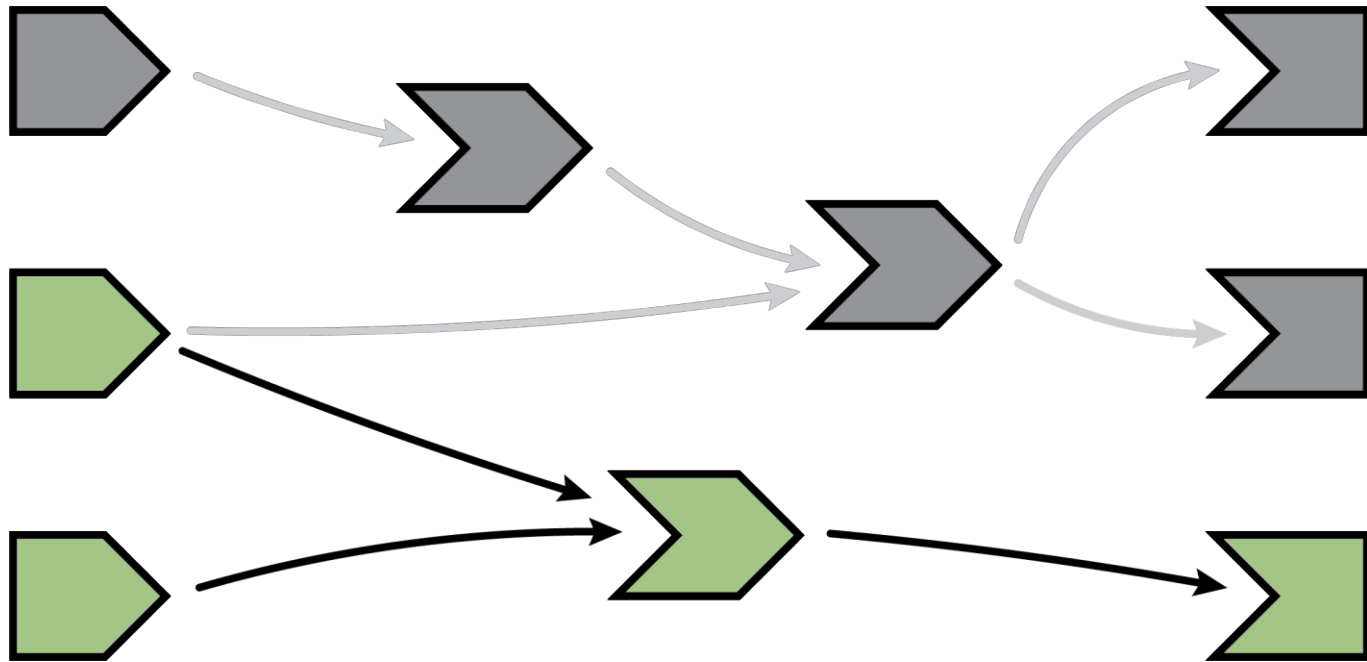
# Execution completes, outputs flushed



# An input changes

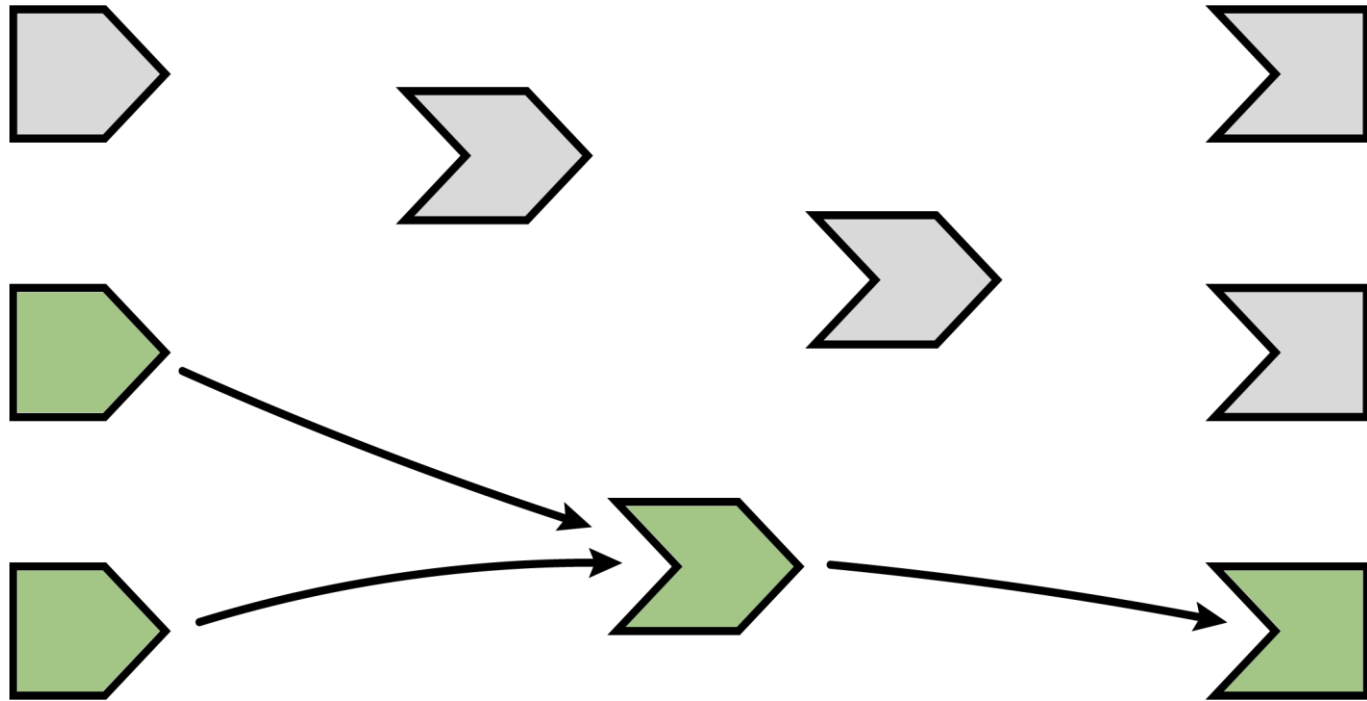


# Notifying dependents

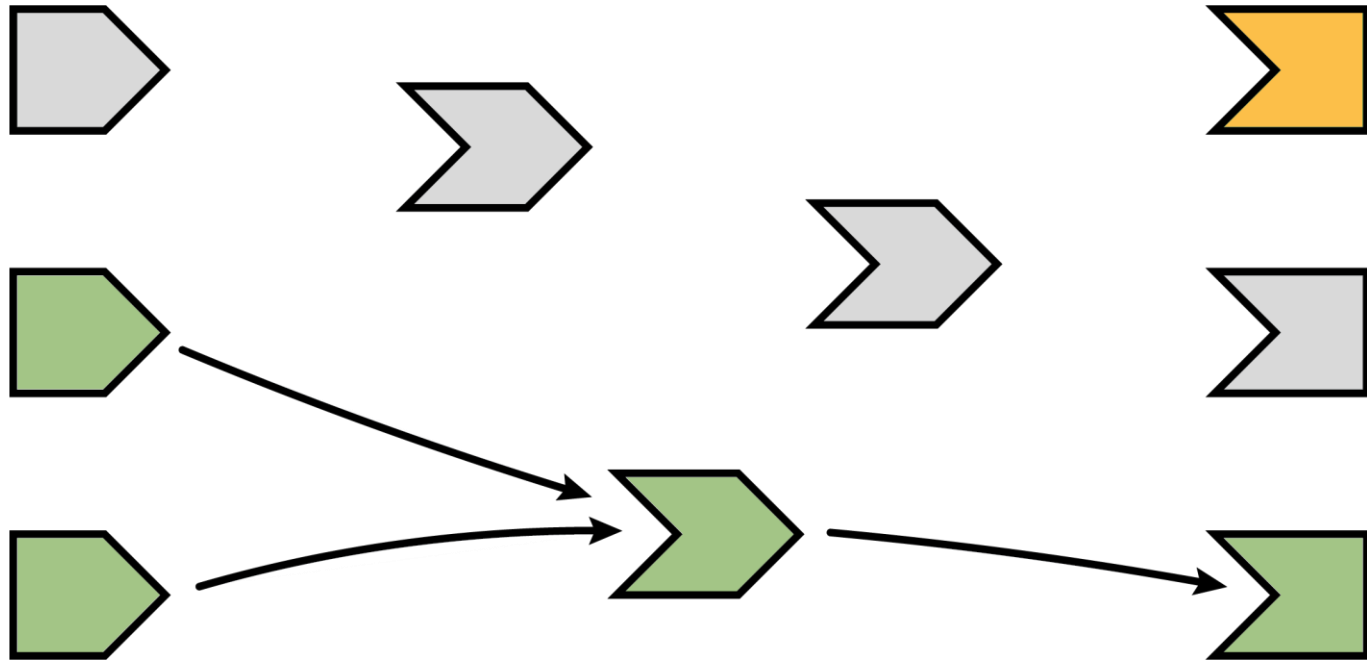




# Removing relationships



# Re-execution



# Share a Shiny app



INRAE

Migale facility  
March 20, 2023 / Sandra Derozier - Mahendra Mariadassou - Cédric Midoux

migale

p. 59

# Share a Shiny app

## Local deployment

- Sharing the code with the user
  - GitHub repository (`shiny::runGitHub` and `shiny::runURL`)
  - R package (build with `golem`) that the user will install locally
- Disadvantage: requires to install R and manage the packages installation

# Share a Shiny app

shinyapps.io

- App is hosted on RStudio servers
  - *Free formula* : 5 Shiny app, 25 actives hours/month
  - *Scalable formula* : since \$9/month to \$300/month
- 1-click deployment
- User-friendly tracking dashboard
- Disadvantage: *Freemium* and RStudio dependency

<https://docs.rstudio.com/shinyapps.io/>



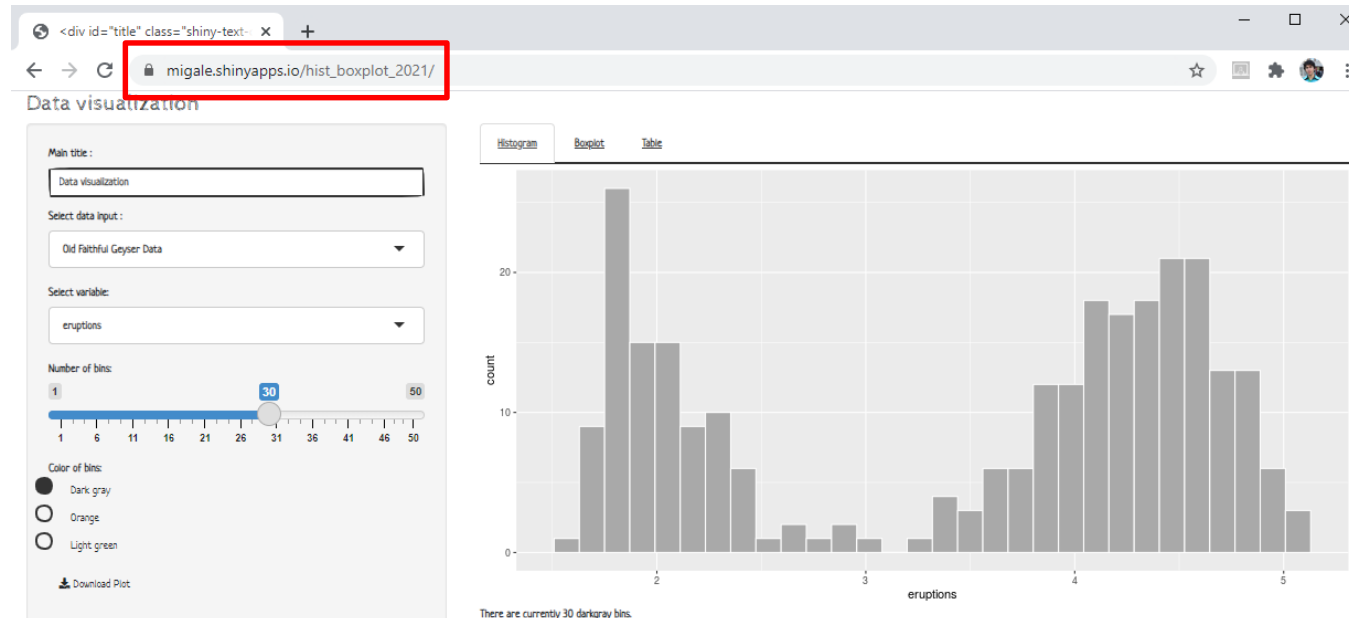
INRAE



# Share a Shiny app

shinyapps.io

→ **TP14:** Deploy an app with shinyapps.io



INRAE

# Building with `{golem}`

- `{golem}` is a package to ease the development and deployment of shiny app (like `{usethis}`)
  - Description, automatic dependencies, unit test, inline documentation, etc
- Install with `install.packages("golem")`
- Create a shiny app as a package with `golem`
  - File > New Project > Package for Shiny App Using `golem`
  - Fill in and execute `dev/01_start.R`
  - Add modules `dev/02_dev.R` and populate `ui / server`
  - Launch app using `dev/run_dev.R`
  - Deploy using `dev/03_deploy.R`



# Host the app on Migale

<https://shiny.migale.inrae.fr/>

- If your application is stable and packaged, we can host it
  - Scientific field of interest (life sciences)
  - Packaged application (like golem)
  - Using docker (we can help you)
  - GitHub / GitLab versioning
  - No huge computations

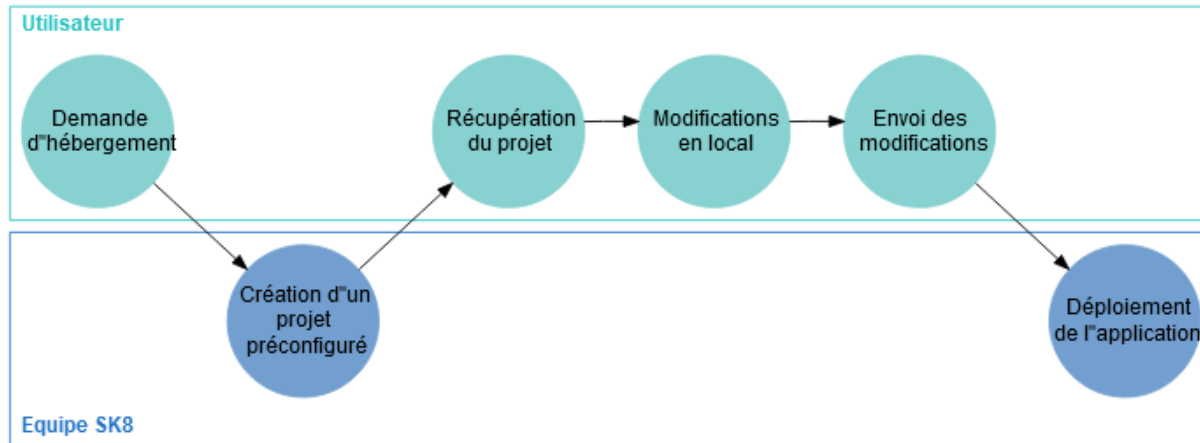




# Host the app on SK8

<https://sk8.inrae.fr/>

- The SK8 project aims at setting up a hosting service for R-Shiny Applications with the tools and infrastructures made available in INRAE.



# Take Home Message



- Shiny allows you to make an R script interactive, easily
- Shiny is the ideal solution for sharing and exploring results
- Several deployment solutions exist and work well, depending what is needed
- Need to optimize your application because latency and memory problems can quickly occur



# Sources

- References :

- **Mastering Shiny** : <https://mastering-shiny.org/>
- Gallery: <https://shiny.rstudio.com/gallery/>
- Gallery widget: <https://shiny.rstudio.com/gallery/widget-gallery.html>
- CheatSheet : <https://shiny.rstudio.com/images/shiny-cheatsheet.pdf>
- CheatSheet VF : <https://thinkr.fr/pdf/shiny-french-cheatsheet.pdf>
- **Engineering Shiny** ( &{golem}): <https://engineering-shiny.org/>

- See also :

- *Building a Wordle App with Shiny!*: [https://youtube.com/playlist?list=PL9HYL-VRX0oQnWleY\\_ydYBdU76iQ-tchU](https://youtube.com/playlist?list=PL9HYL-VRX0oQnWleY_ydYBdU76iQ-tchU)
- Awesome shiny : <https://github.com/nanxstats/awesome-shiny-extensions>
- Docs SK8 (VF, hébergement & Bonnes pratiques shiny) : <https://docs.sk8.inrae.fr>
- Golem : <https://thinkr-open.github.io/golem/>



# Sources

- Tuto :
  - **Official** : <https://shiny.rstudio.com/tutorial/>
  - Introduction à Shiny (VF) : [http://perso.ens-lyon.fr/lise.vaudor/Tuto\\_Shiny/](http://perso.ens-lyon.fr/lise.vaudor/Tuto_Shiny/)
  - Formation IFB (VF) : [https://github.com/IFB-ElixirFr/IFB\\_Shiny\\_training](https://github.com/IFB-ElixirFr/IFB_Shiny_training)
  - UI organisation : <https://speakerdeck.com/jcheng5/styling-shiny>
- Blog :
  - <https://blog.rstudio.com/categories/shiny/>
  - <https://bioinfo-fr.net/rendre-ses-projets-r-plus-accessibles-grace-a-shiny>
  - <https://superstatisticienne.fr/r-sur-le-web-le-package-shiny/>
  - <https://thinkr.fr/a-decouverte-de-shiny/>

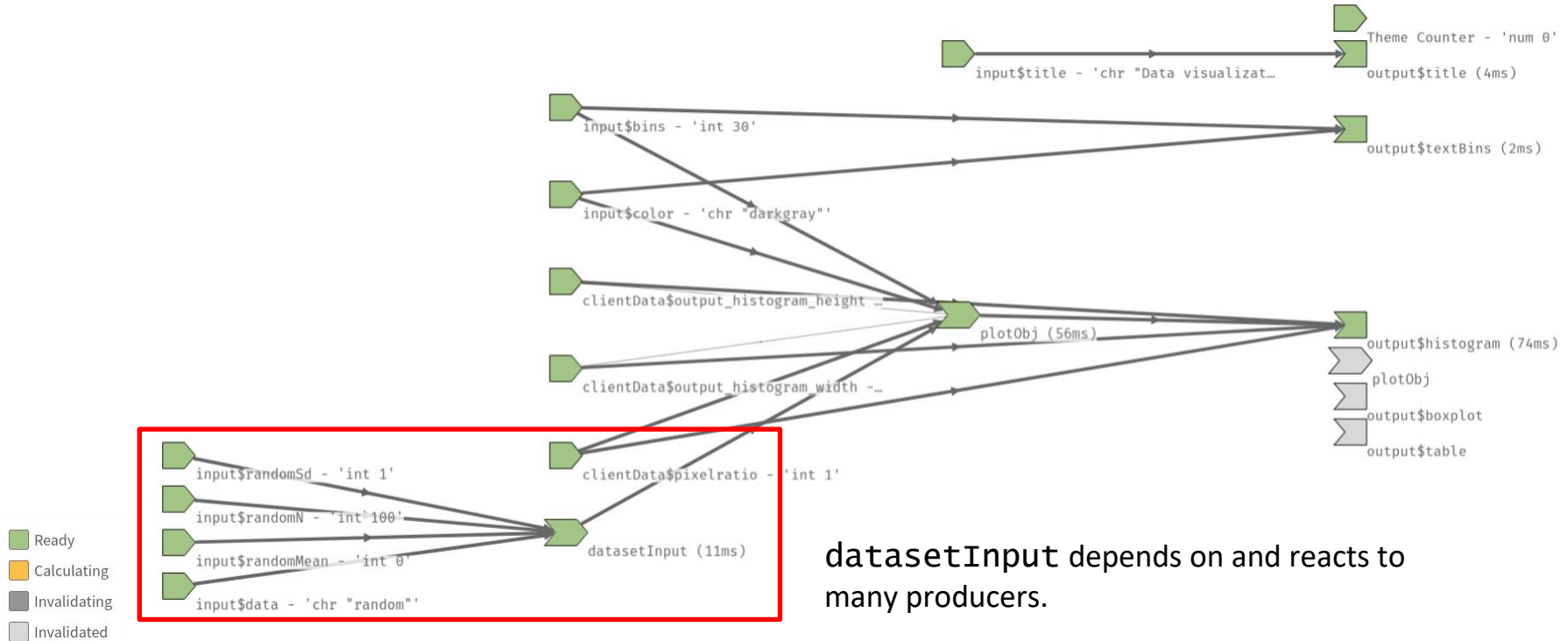
# More about reactlog



INRAE

Migale facility  
March 20, 2023 / Sandra Derozier - Mahendra Mariadassou - Cédric Midoux

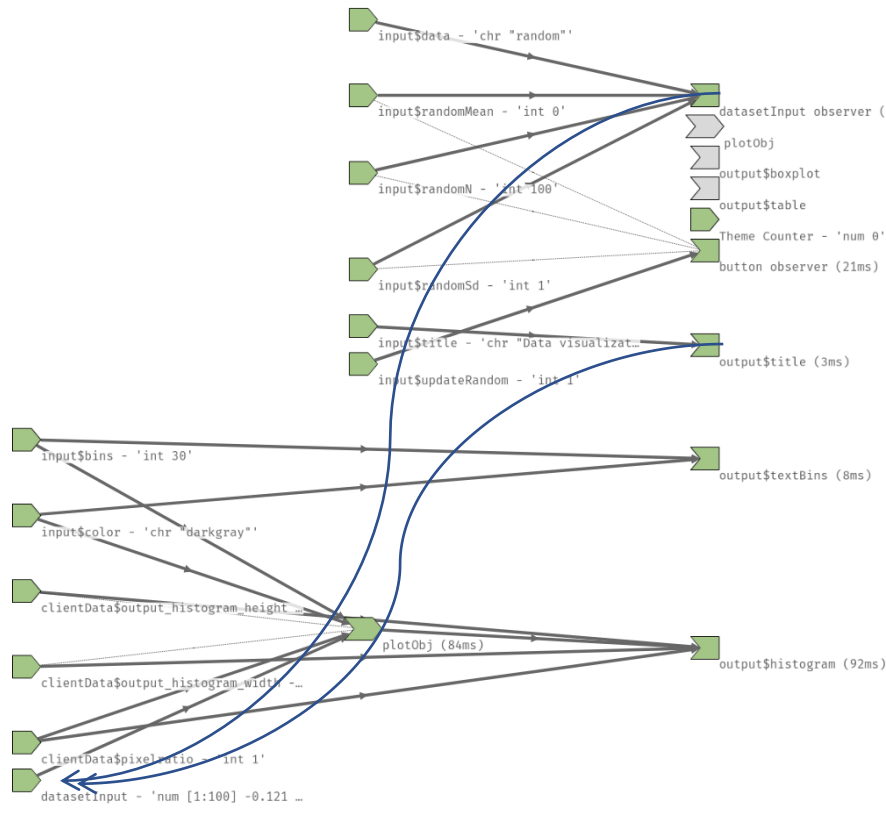
# Exploring reactivity: app9



datasetInput depends on and reacts to many producers.



# App9\_bis: ugly dependency graph



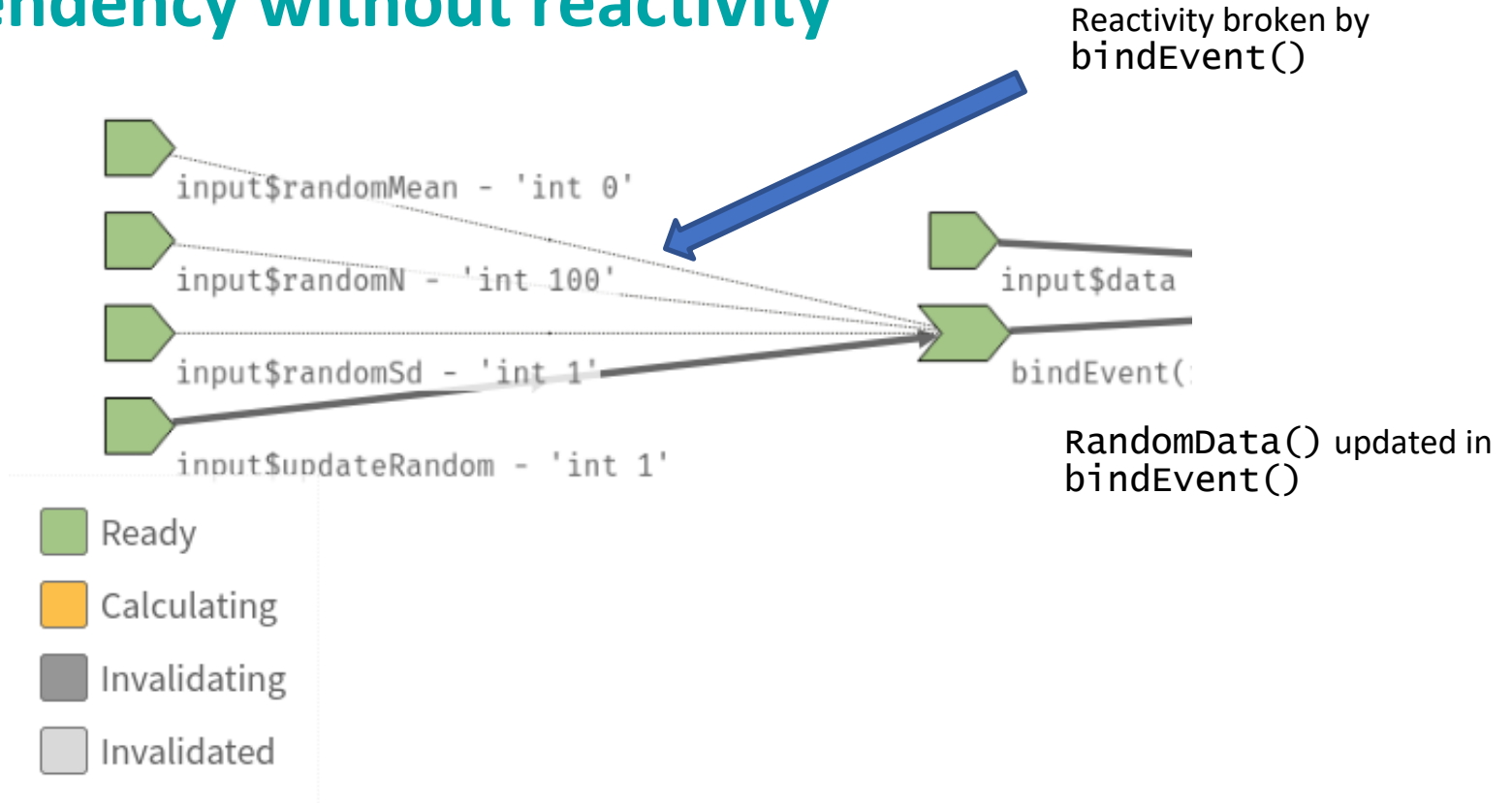
Both observers update datasetInput()

**Note:** observers **break** the natural dependency / reactivity graph

- Ready
- Calculating
- Invalidating
- Invalidated

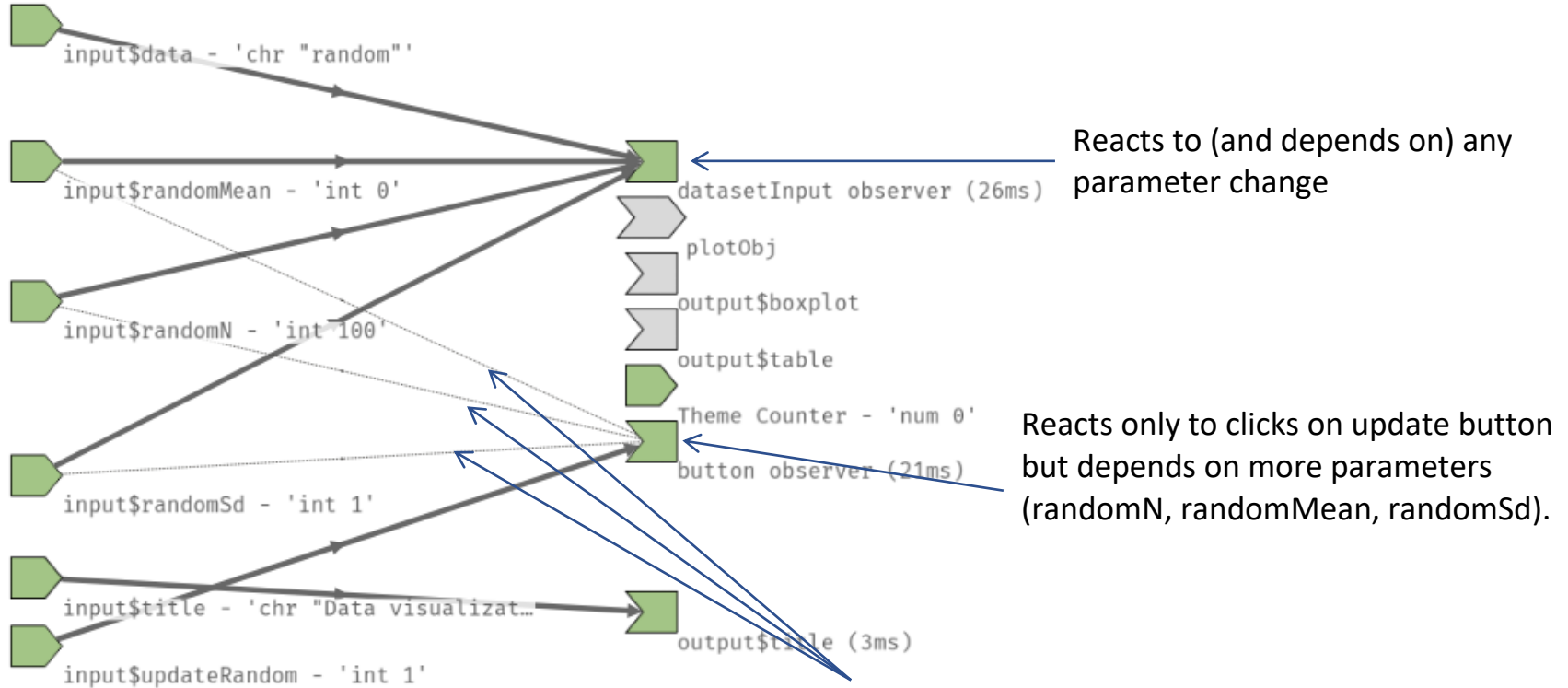


# Dependency without reactivity





# Reactivity versus dependencies



Dependency without reactivity, thanks to `bindEvent()`



# App9\_ter: straightforward dependencies

