

Part 2: Reactivity

This section is for:

- Those who never really understood reactivity
- Don't know what the difference is between `observeEvent()` and `eventReactive()`
- Want to know what `ExtendedTask` does

Why Reactivity?

- Event-driven programming
 - Event: Change in inputs or reactivities
- Only update outputs when necessary (lazy)
- The bad old days: UI polling
- Poke every 100 ms at a UI element
 - Have you changed yet?
 - Have you changed yet?
 - Have you?

Shiny works best when you give it control

- Don't tell Shiny **when** to update, but **how** to update
- Give Shiny control to update as it sees fit
- Don't force order of operations - causes problems
- You need to trust Shiny

Reactivity

- All calculations are dependent on the *reactive graph*
- Depending on how inputs, reactivities, and outputs are connected
- Only recalculate for *visible outputs* that have changes in inputs
- The reactive graph invalidates on changes in inputs / reactivities

Key Players

Input



```
input$select  
reactiveValues()
```

“Producers”

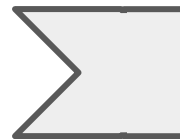
reactive()



```
data <-  
reactive({})
```

“Consumers”/
“Producers”

Output



```
output$plot  
observe()
```

“Consumers”

The Reactive Graph

- Calculates visible outputs (such as plots) based on changes in inputs and reactivities
 - Don't calculate outputs that aren't visible
- Initial calculation leads to an “equilibrium” state of graph
- Changes in inputs invalidate outputs and connected reactivities
- Data flows from Inputs -> reactivities() -> Outputs

`{reactlog}`

Lets you visualize the flow of information through the reactive graph

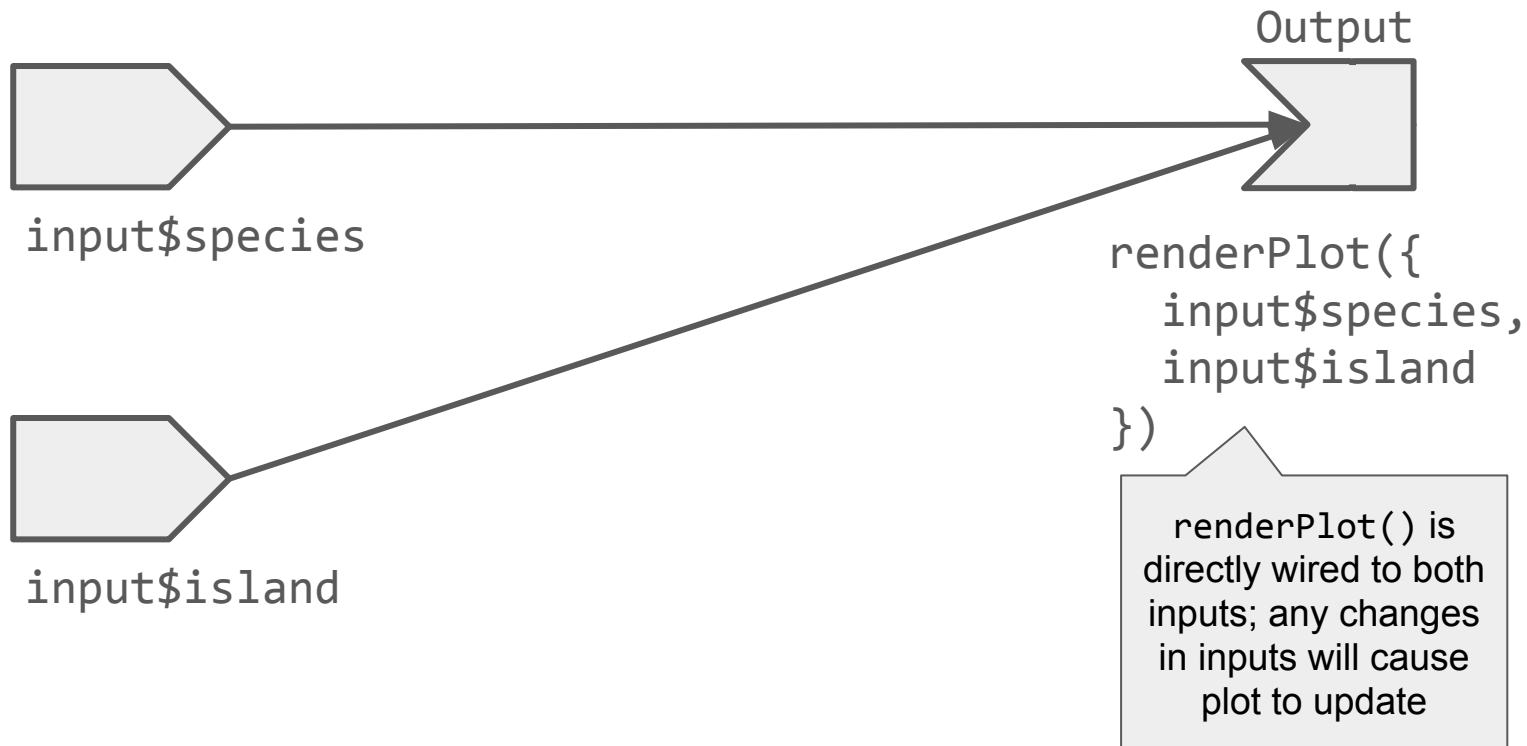
Invaluable for understanding how events trigger recalculation

Does not measure time of calculation

Use `{profvis}` for that

Direct Connections

`runApp("app_direct.R")`



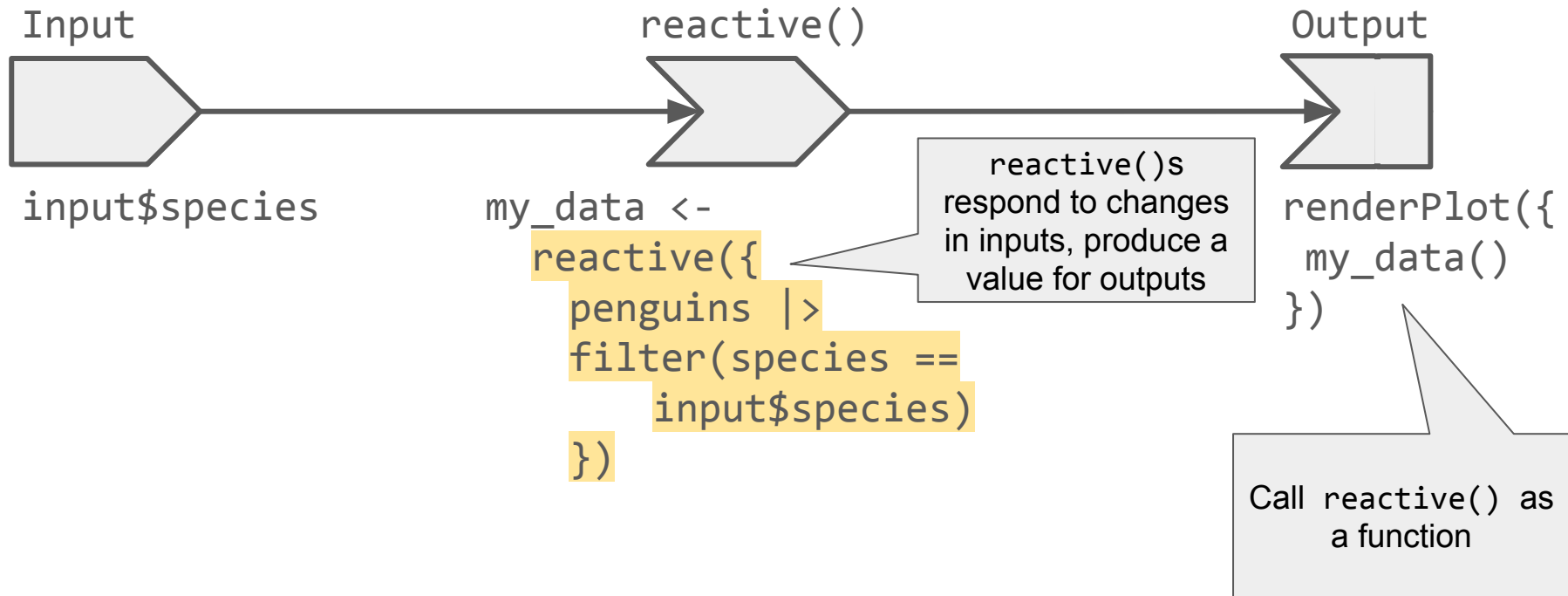
Quick reactlog demo

reactive()

- Dynamic (responds to other reactives or inputs)
- Returns a value (such as a data.frame)
- Decouples inputs from outputs
- Avoids recalculation unless necessary

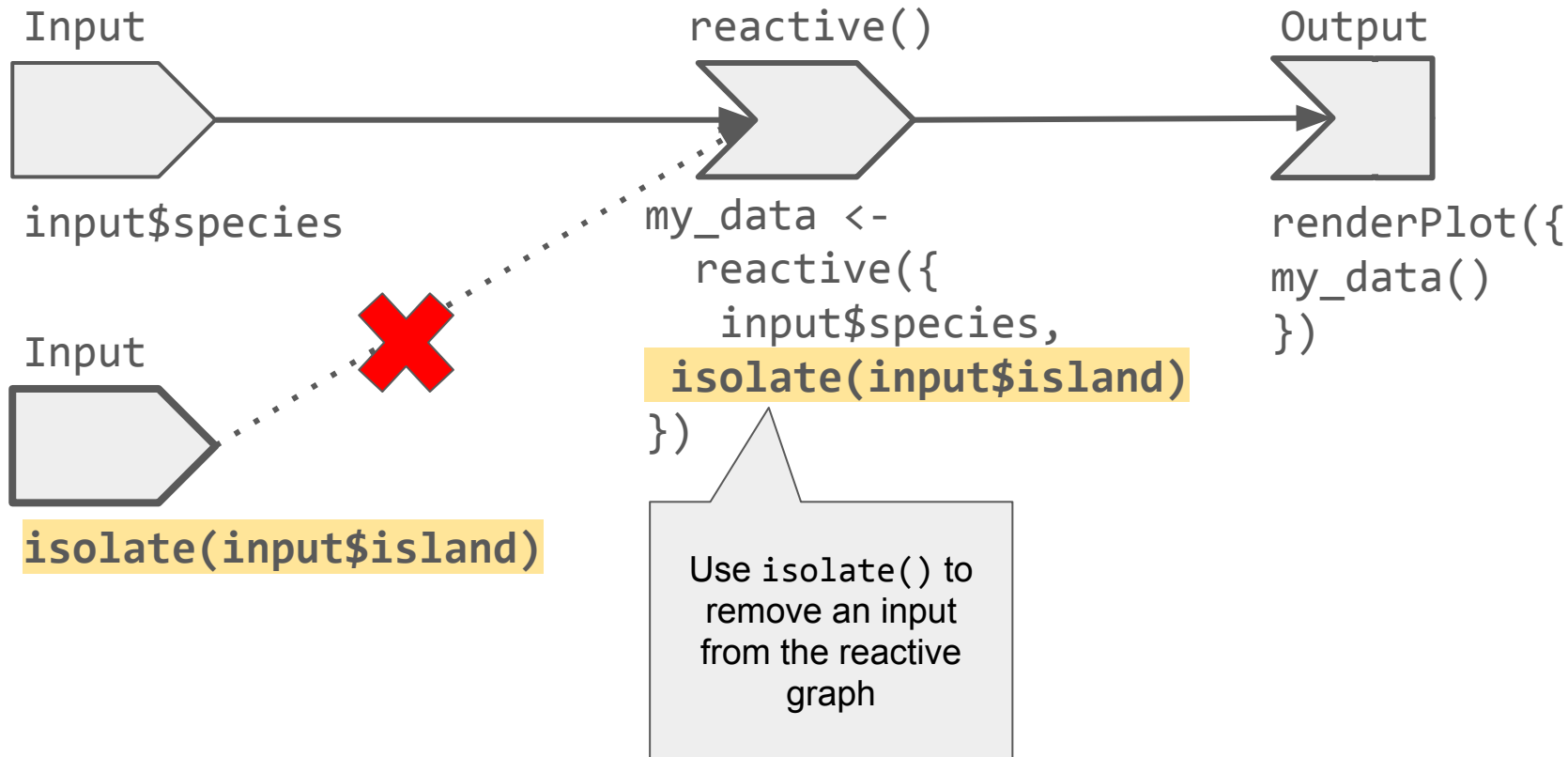
reactive()

runApp("app_reactive.R")



isolate()

runApp("app_isolate.R")



Exercise

- With your neighbor, compare the following two apps with reactlog (run the app, and use CTRL+F3 or CMD+F3)
 - `runApp("reactivity/app_reactive.R")`
 - `runApp("reactivity/app_isolate.R")`
- How do the graphs differ?
- Change the two controls and step through the graph - what are the differences?

Events

- Want app to respond to some sort of event or change
- Usually an `actionButton`
- but could also be a `reactive()`

Pure Functions vs. Side Effects

Pure Functions

Return some sort of **value** that is used downstream triggered by event (reactives)

Side Effects

Used only for **side effects** triggered by event (doesn't return a value)

Pure Functions vs. Side Effects

Pure Functions

Return some sort of **value** that is used downstream triggered by event (reactives)

Examples:

- Reading a file,
- Calculating a new variable,
- Filtering data

Side Effects

Used only for **side effects** triggered by event (doesn't return a value)

Examples:

- Updating a Database,
- Updating UI elements,
- Saving data to a file

Pure Functions vs. Side Effects

Pure Functions

Return some sort of **value** that is used downstream triggered by event (reactives)

Examples:

- Reading a file,
- Calculating a new variable,
- Filtering data

`use eventReactive()` or

`reactive() |> bindEvent()`

lazy execution

Side Effects

Used only for **side effects** triggered by event (doesn't return a value)

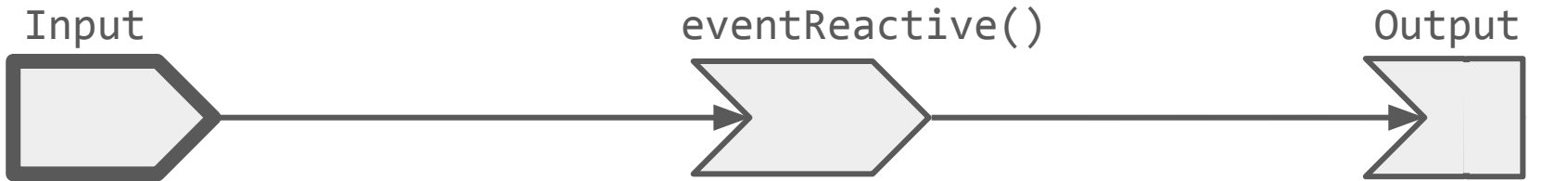
Examples:

- Updating a Database,
- Updating UI elements,
- Saving data to a file

`use observeEvent()` or

`observe() |> bindEvent()`

eager execution

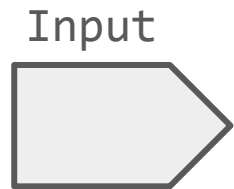


`input$act_button`

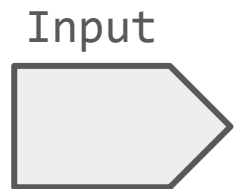
`my_er <-`

```
eventReactive(  
  input$act_button,  
  {input$island,  
    input$species}  
)
```

```
renderPlot({  
  my_er()  
})
```



`input$island`

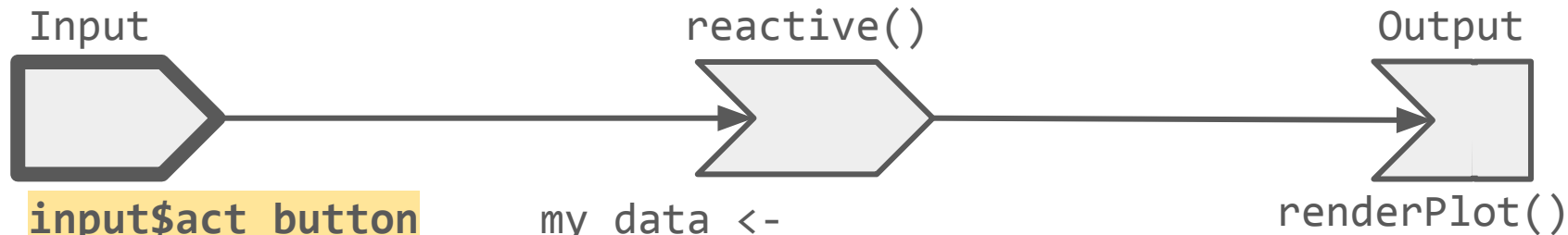


`input$species`

Use `eventReactive()`
to only update a reactive
on an event (button
press, change in another
reactive)

Put reactive code after
event trigger in `{ }`

`runApp("reactivity/app_eventReactive.R")`



Input

`input$island`

Input

`input$species`

```
my_data <-  
  reactive({  
    input$island,  
    input$species}) |>  
  bindEvent(input$act_button)
```

Use `bindEvent()` to
make a `reactive()`
respond only to an
event

Saves some typing /
less error prone than
changing
`reactive()` to
`eventReactive()`

`runApp("reactivity/app_bindEvent.R")`

Why does lazy execution matter?

- Shiny tries to minimize the number of calculations
- Only tries to calculate outputs that are visible
- Reactive graph helps decide when to recalculate
- Invalidation -> calculation -> ready to display

`runApp("app_observeEvent.R")`



`input$act_button`

```
observeEvent(  
  input$act_button, {  
    dbAppendTable()  
  })
```

Input



`input$island`

Input



`input$species`

Use `observeEvent()` only for actions that don't generate a value, such as database updates. It will run right away (eager execution)

```
observe({dbAppendTable()}) |>  
bindEvent(input$act_button)
```

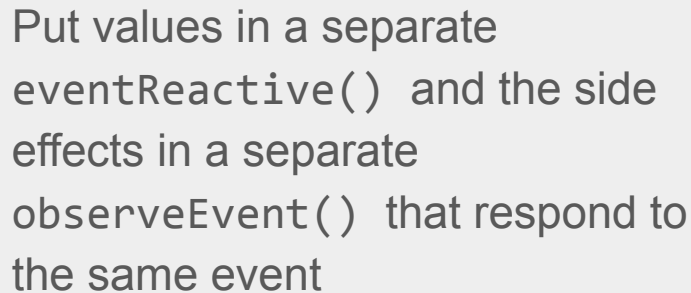
Don't mix values and side effects

Keep your side effects

Outside of your reactives

Or I will kill you.

Joe Cheng



Put values in a separate `eventReactive()` and the side effects in a separate `observeEvent()` that respond to the same event

Good uses of observeEvent()

- Dynamically updating UI based on User Inputs
 - updateSelectInput example
- Writing lines to a database
- Printing to the console
- Saving a File
- Doing R Stuff that doesn't produce any output

observe() + reactiveValues()

- Often an anti-pattern
- Programmers try to use it to force order of execution
- Is very hard to test because of race conditions when multiple observe() statements are updating a reactiveValues() object
 - Example: reactiveValues with bank balance
- Use sparingly
- ExtendedTask is an exception

Exercise

- Do this with your neighbor and discuss
- Compare the reactive graphs between (CTRL+F3 or CMD+F3)
 - reactivity/app_eventReactive.R
 - reactivity/app_observeEvent.R
- How are they wired differently?
- Try triggering the event (pushing the button) and step through the graph

Reactivity: Optimizing

Optimizing Your Shiny App

- Common Sense Stuff
- `reactivePoll()`
- `ExtendedTask`
- `bindCache()`

Common Sense Stuff

- Define “real-time” for your app
- Decouple data pulls from your shiny app if possible
- Scheduled process that pulls every 10 minutes for example
- Precompute as much as possible (memory is cheap, compute time is not)

reactivePoll()

- Use for expensive data updates
- Use when connected to a data source that updates itself
- Only updates the reactive when the data has changed
- You need to write a function that tests for changes in it

`bslib::input_task_button()` for `ExtendedTask`

Won't trigger an event multiple times

```
runApp("app_input_task_button.R")
```

ExtendedTask

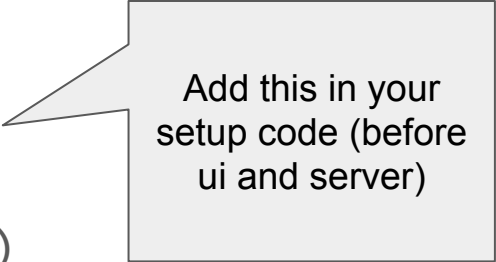
- Lets you run a long running operation in the background (slow API call, etc)
- Non-blocking: you and other users can work with your app without being interrupted by the long task
- Uses `future_promise()` to spin off into its own R-session
- Running models, long calculations, etc
- ExtendedTask is an R6 object
- YouTube Video: <https://www.youtube.com/watch?v=GhX0PcEm3CY>

Setup for ExtendedTask

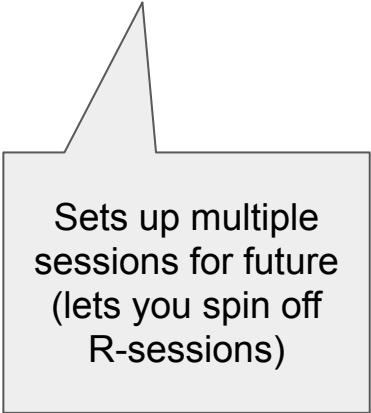
```
library(future)
```

```
library(promises)
```

```
future::plan(multisession)
```

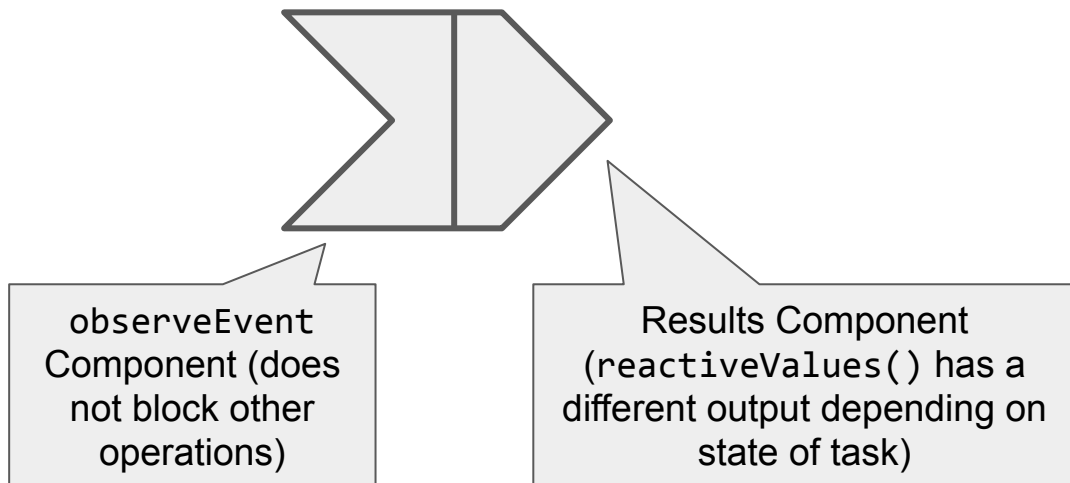


Add this in your
setup code (before
ui and server)



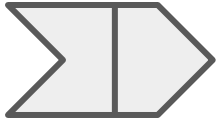
Sets up multiple
sessions for future
(lets you spin off
R-sessions)

ExtendedTask is Two parts



ExtendedTask Methods (use in server)

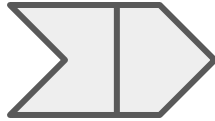
ExtendedTask



```
my_task <-  
ExtendedTask$new(  
  future_promise(  
    function(x){}  
  ))
```

Use `$new` method
to make a new
ExtendedTask

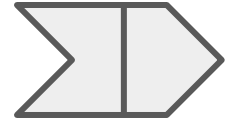
ExtendedTask



```
my_task$invoke(input$x)
```

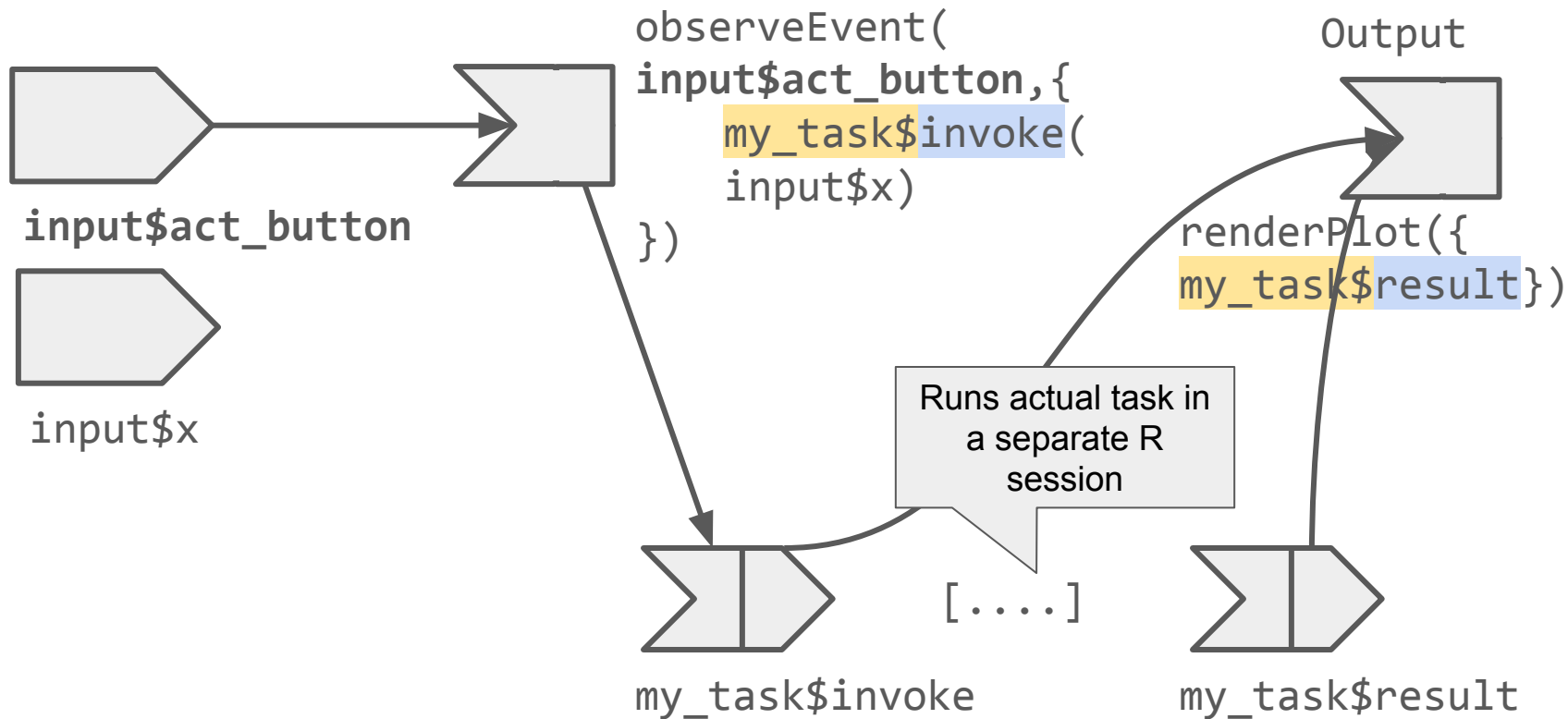
Use `$invoke`
method to start a
task up (runs in a
separate session)

ExtendedTask



```
my_task$result
```

Use `$result`
method to use
results



`runApp("app_ExtendedTask.R")`

Exercise (if we have time)

Examine the Reactive Graph using reactlog for the following application:

- `runApp("app_ExtendedTaskSingle.R")`

Click the button and trace the path.

bindCache()

- Shiny usually only caches the last value
- `bindCache()` lets you cache values based on an event
 - Shared across all users of the app
 - Can change to per-session
- Can be a reactive or a plot
- Saves initial computation to reduce load time