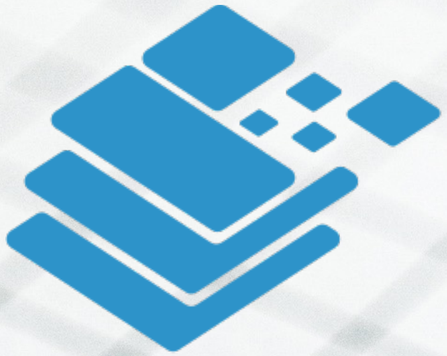
Spatial
Lab
Analytics

partimos pronto



Spatial
Lab
Analytics

Soluciones en análisis de datos

APLICACIONES INTERACTIVAS CON R Y SHINY

Bastián Olea Herrera - baolea@uc.cl

Desarrollo de interfaces web para interactuar con código de R en tiempo real, permitiendo que analistas y usuarios accedan a procesos, resultados y visualizaciones de R mediante su navegador.

Desarrollando apps

Repaso

Repaso de reactividad, y verificar cómo funciona en una aplicación de prueba

Avanzando

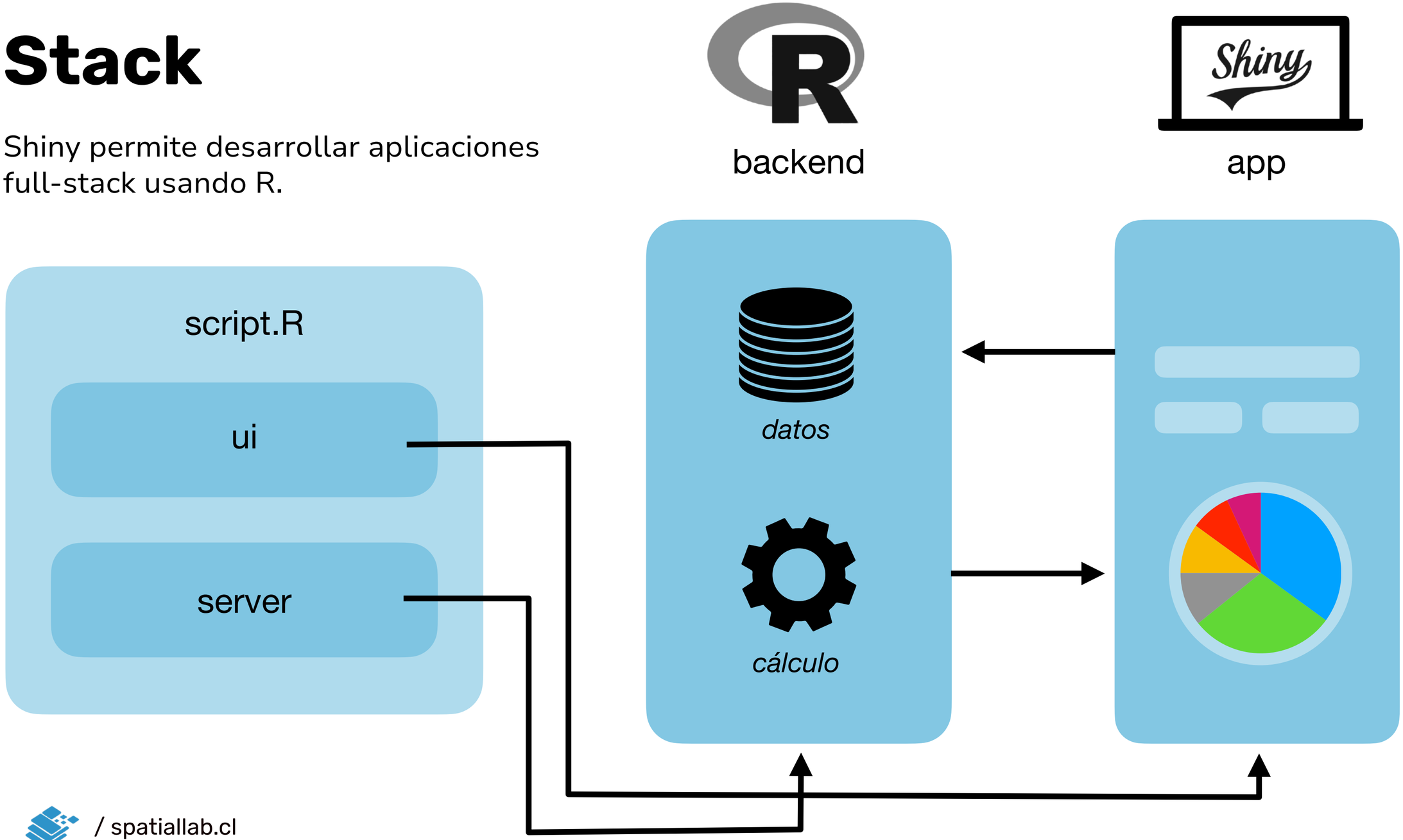
Desarrollo de outputs con mayor complejidad, elementos con condicionales

Otro caso

Selectores dinámicos, varios outputs simultáneos, generación de interfaz dinámica, visualizaciones

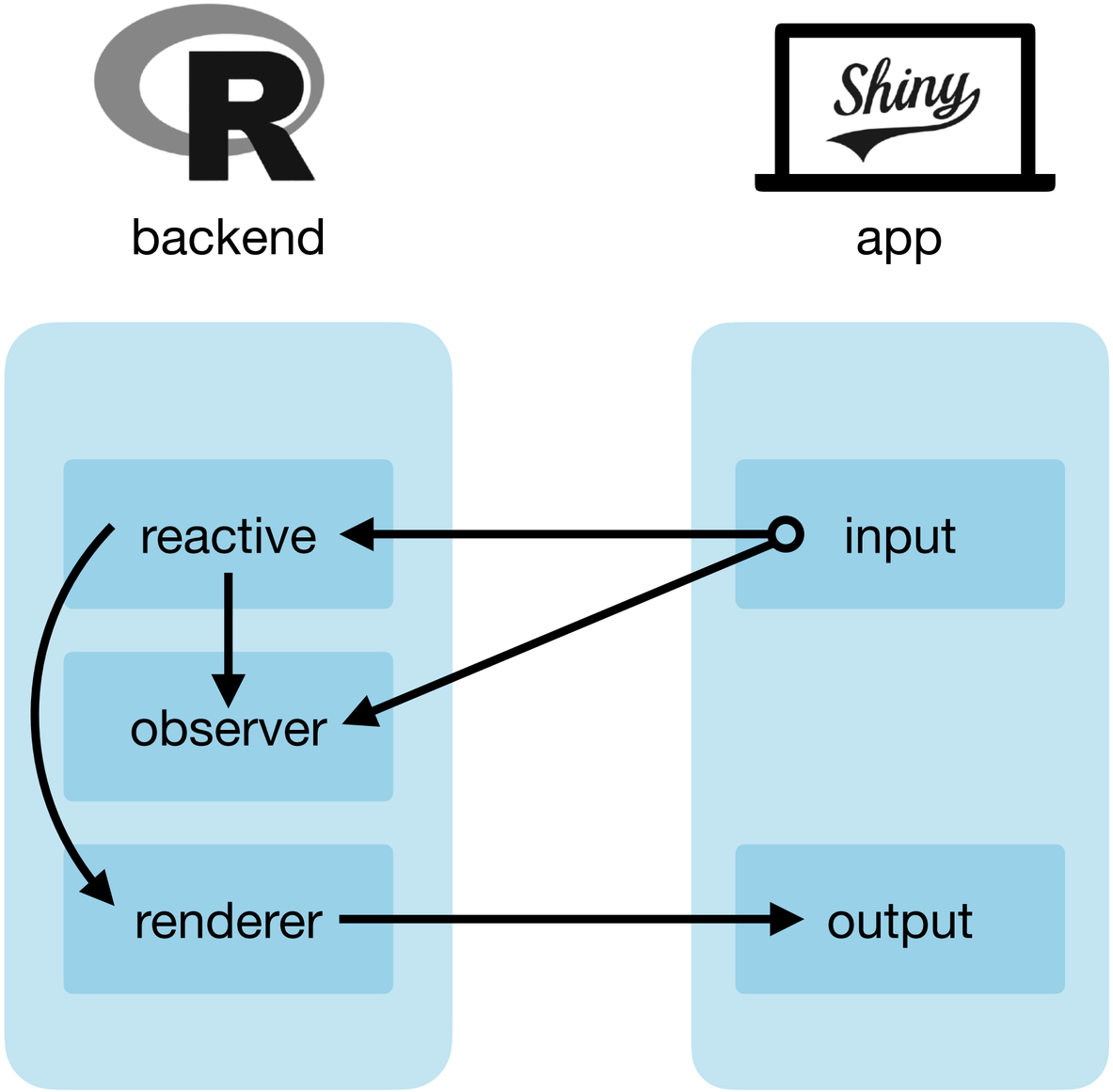
Stack

Shiny permite desarrollar aplicaciones full-stack usando R.



Reactividad

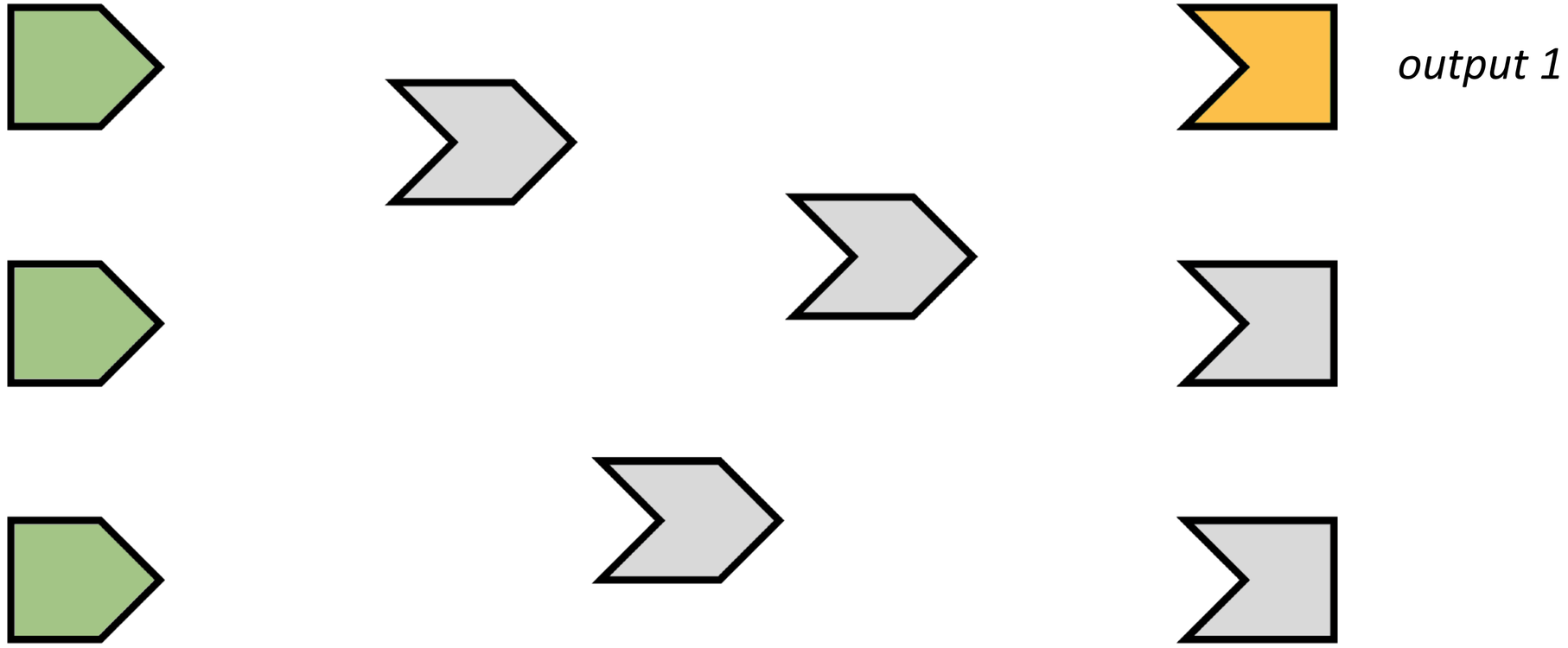
- La reactividad se basa en la existencia de valores (inputs) que pueden cambiar, y en cálculos o acciones (reactives, observers, renderers) que dependen de dichos valores y/o de otros cálculos.
- Las expresiones reactivas llevan a cabo los cálculos, y dependen de inputs o de otros reactivos.
- Las expresiones reactivas se actualizan sólo cuando uno de los elementos de los que depende cambian.
- De este modo, los outputs responden a los inputs.



Reactividad

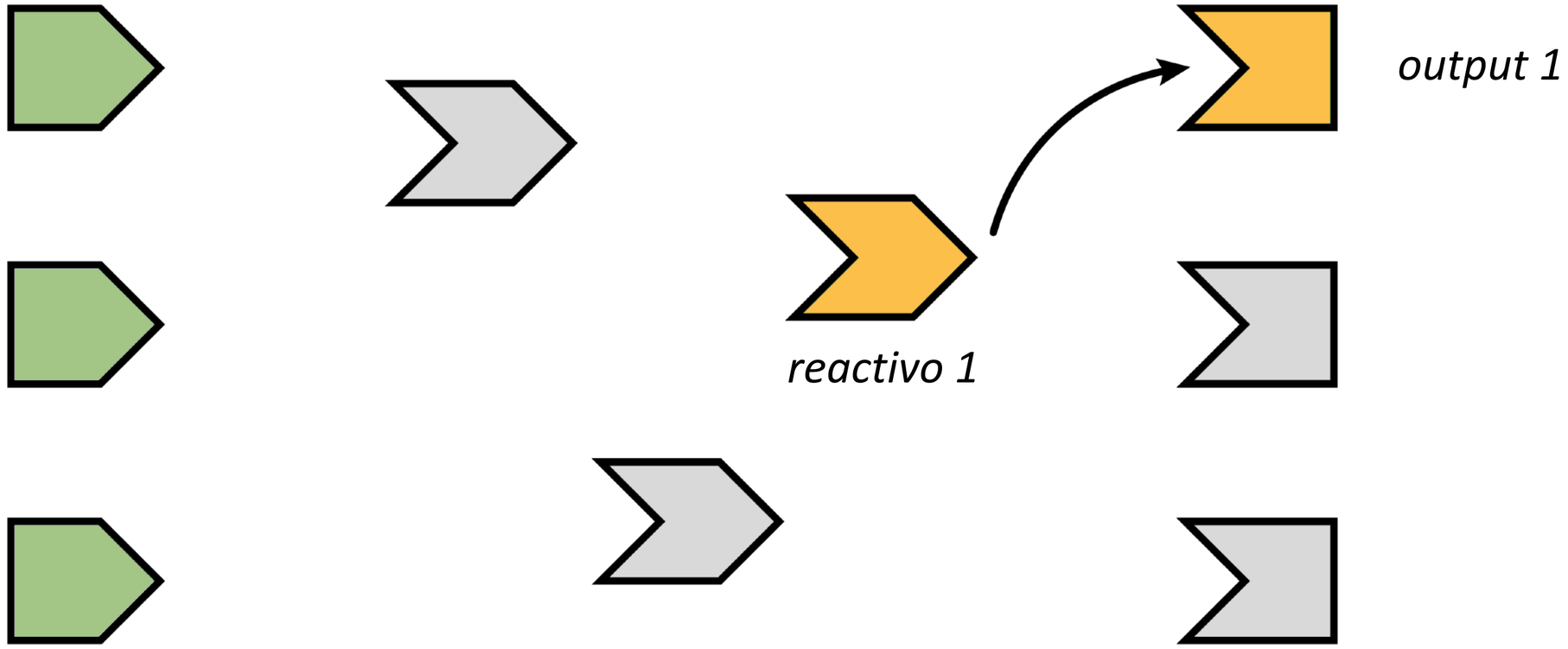
Cada output tiene una cadena de dependencias, mientras que los inputs están disponibles siempre

El proceso inicia por la necesidad de calcular los outputs, uno por uno.



Reactividad

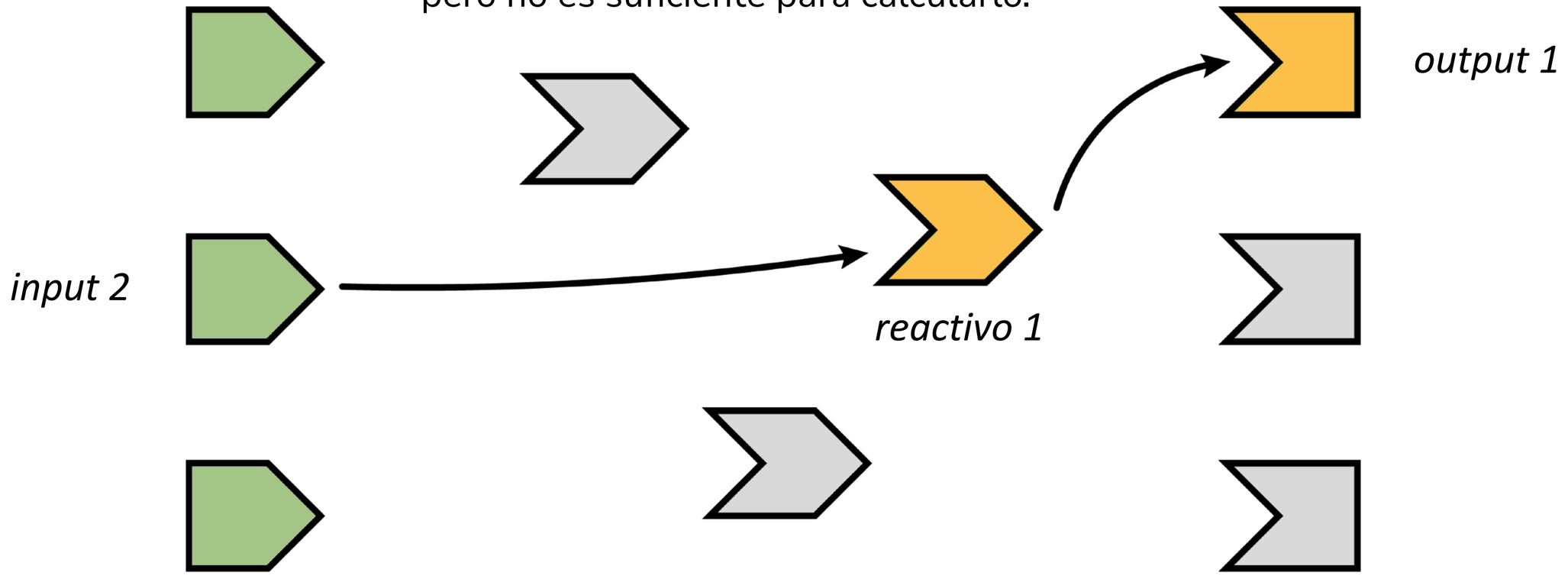
El output 1 depende de un objeto reactivo, por lo que éste también se marca como pendiente a ser calculado.



Como el reactivo 1 también depende de otros objetos, su cálculo queda pendiente.

Reactividad

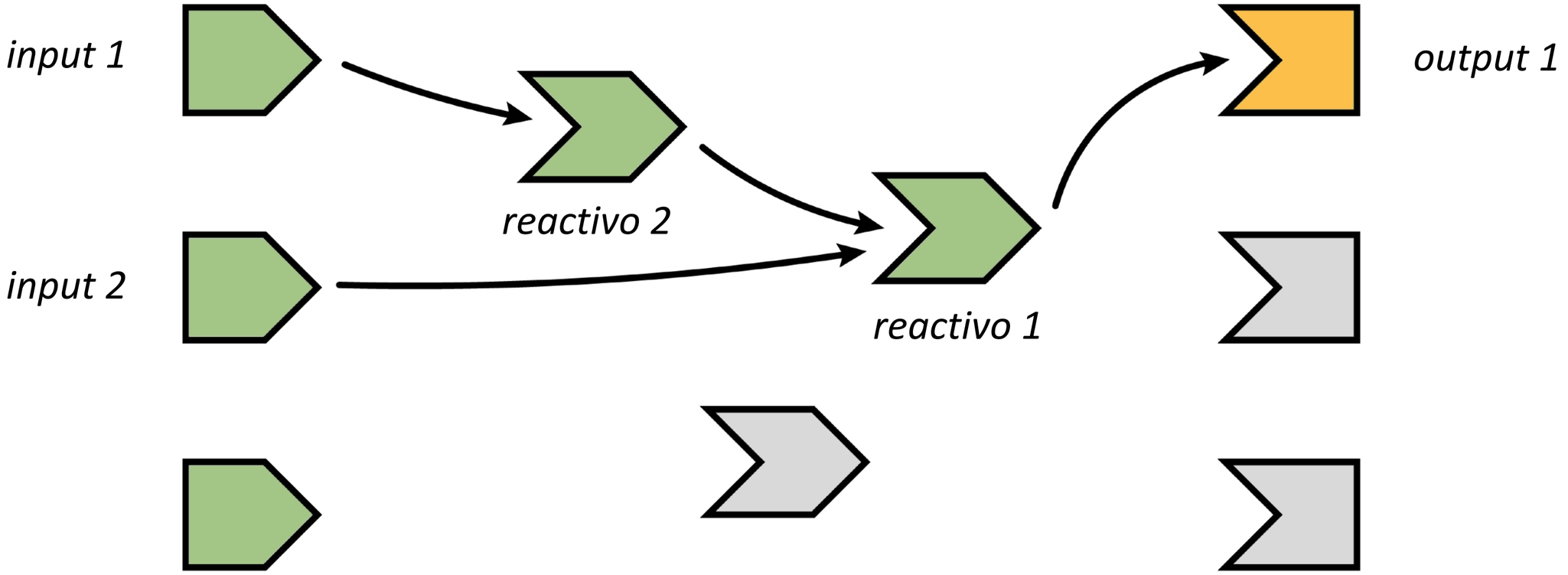
El objeto reactivo 1 además depende de un input, el cual es entregado al objeto reactivo, pero no es suficiente para calcularlo.



Reactividad

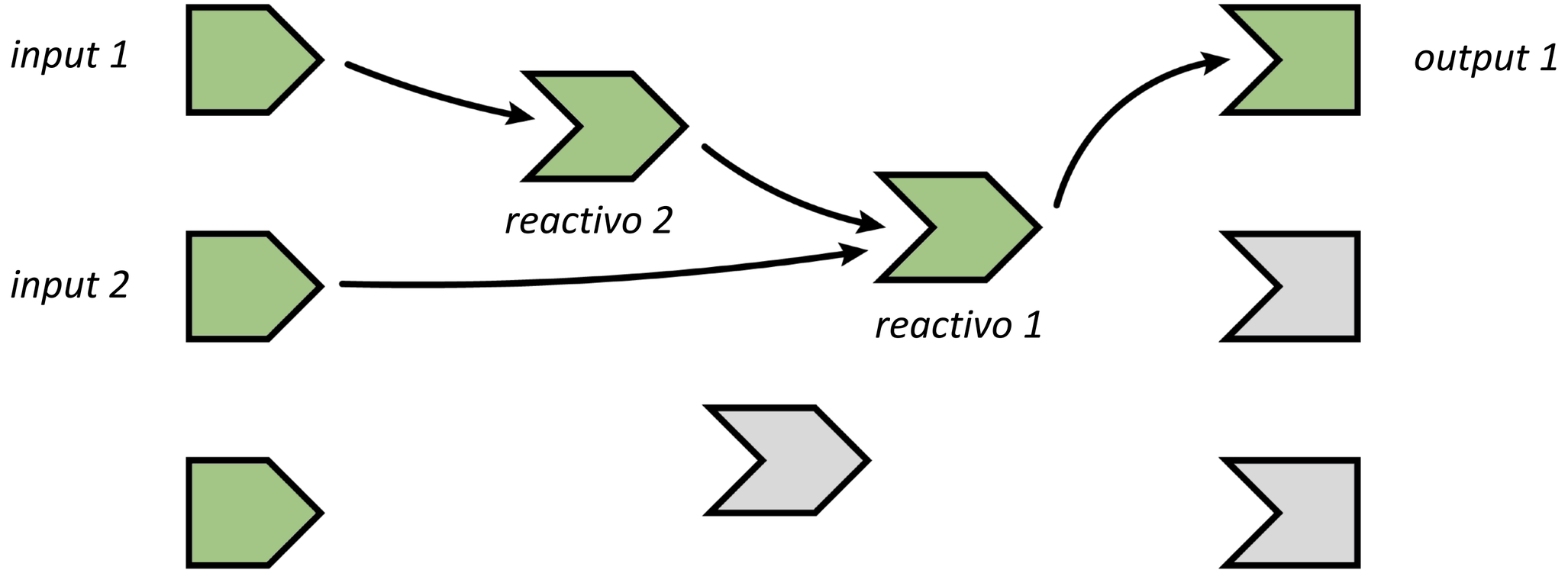
El otro objeto reactivo 2 sólo depende del input 1, por lo que se calcula.

Al calcularse el reactivo 2, el reactivo 1 puede calcularse también.



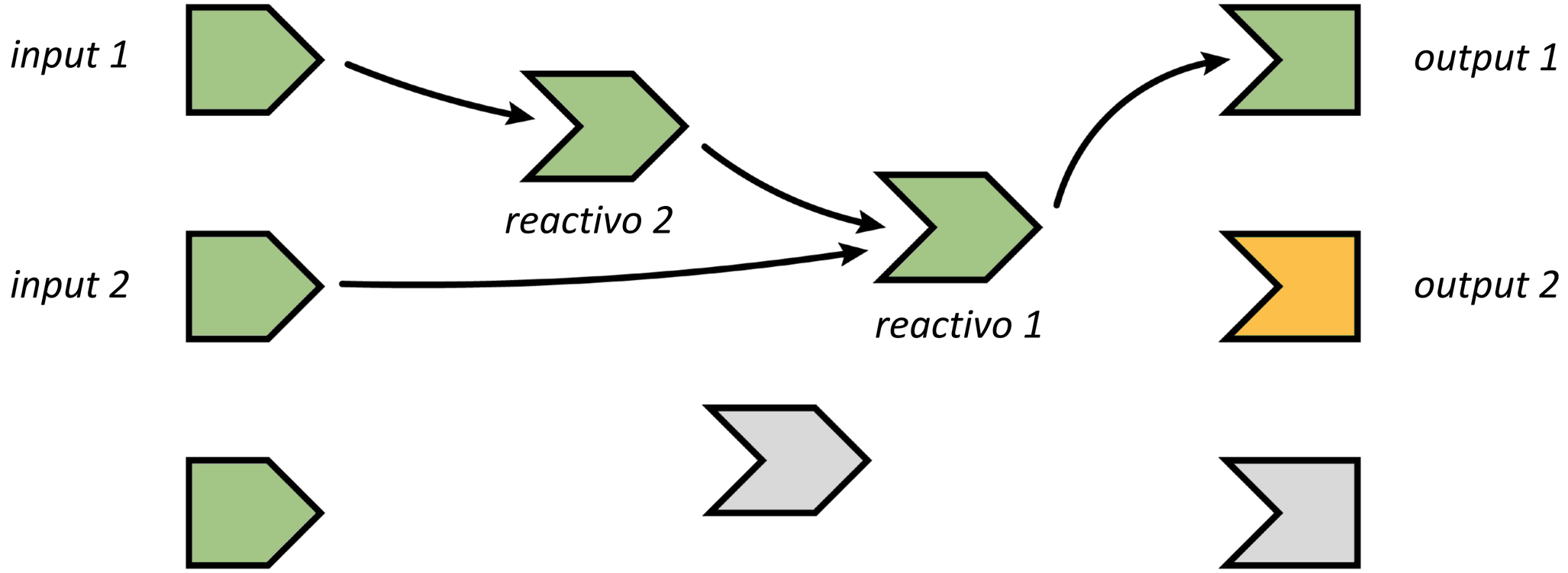
Reactividad

Habiéndose calculado toda la cadena de objetos requeridos por el output 1, éste se calcula y se muestra en la app.

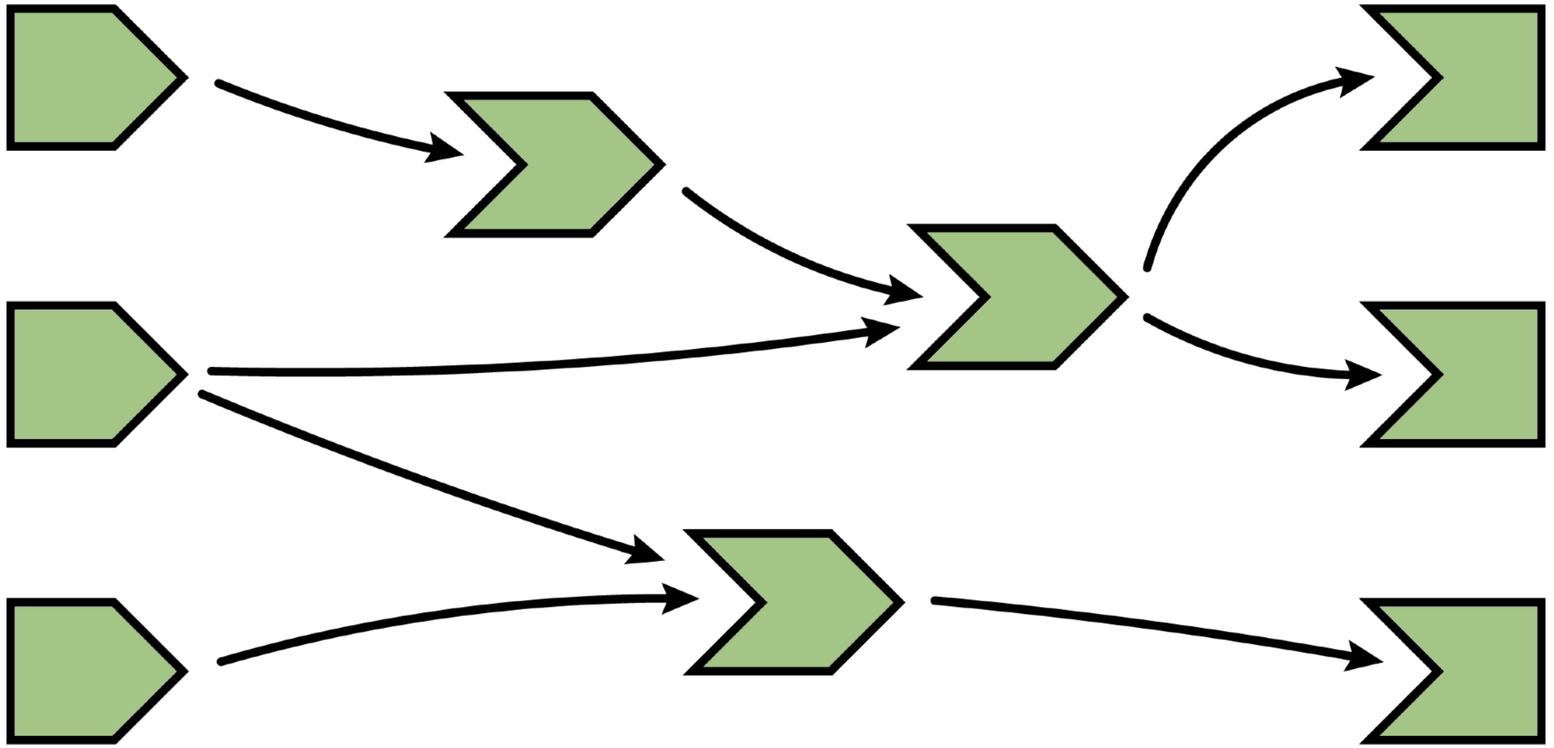


Reactividad

El output 2 también depende del reactivo 1, y como éste ya está calculado, se reutiliza para el output 2.



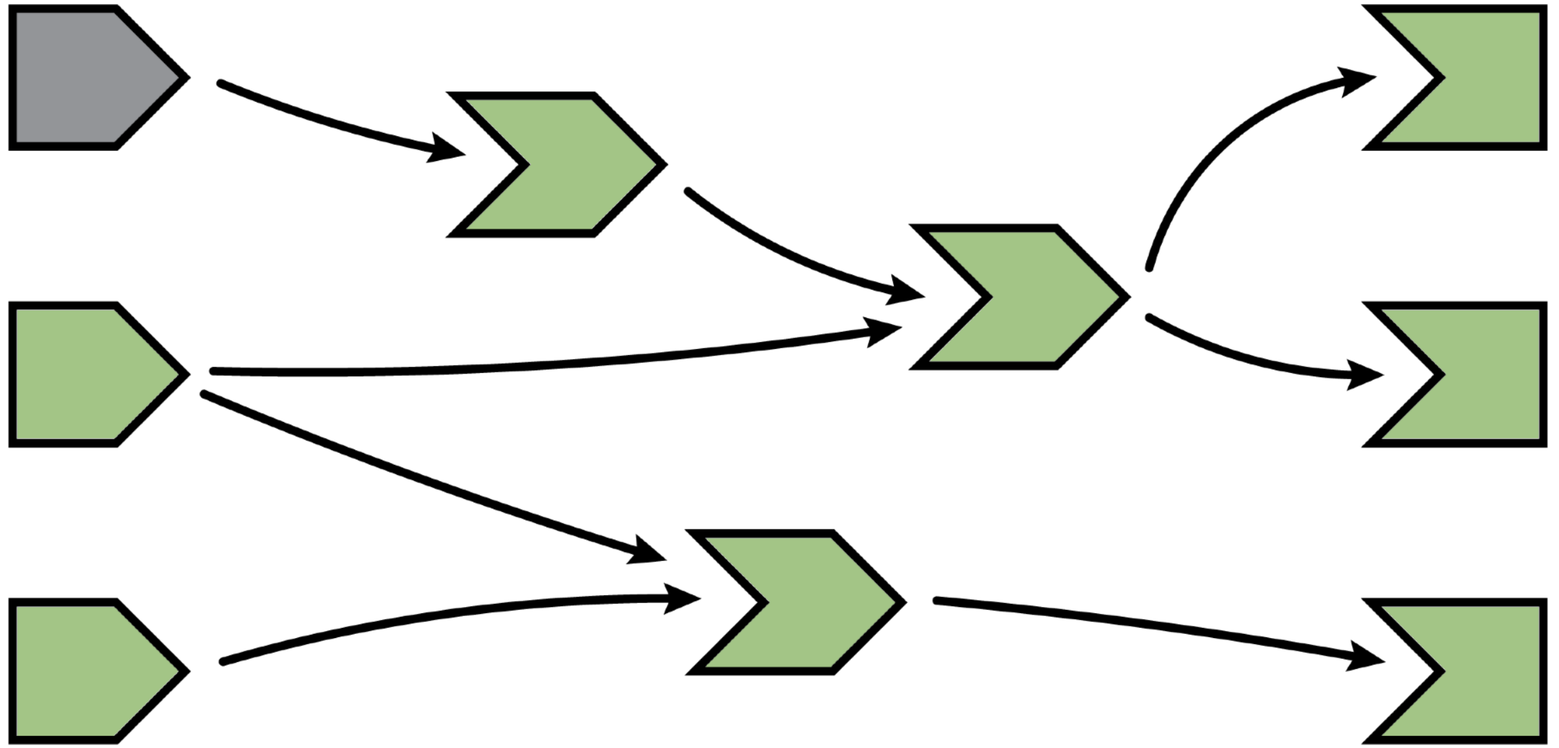
Reactividad



Se aplica la misma lógica para el output restante, y la aplicación queda lista y en estado de espera.

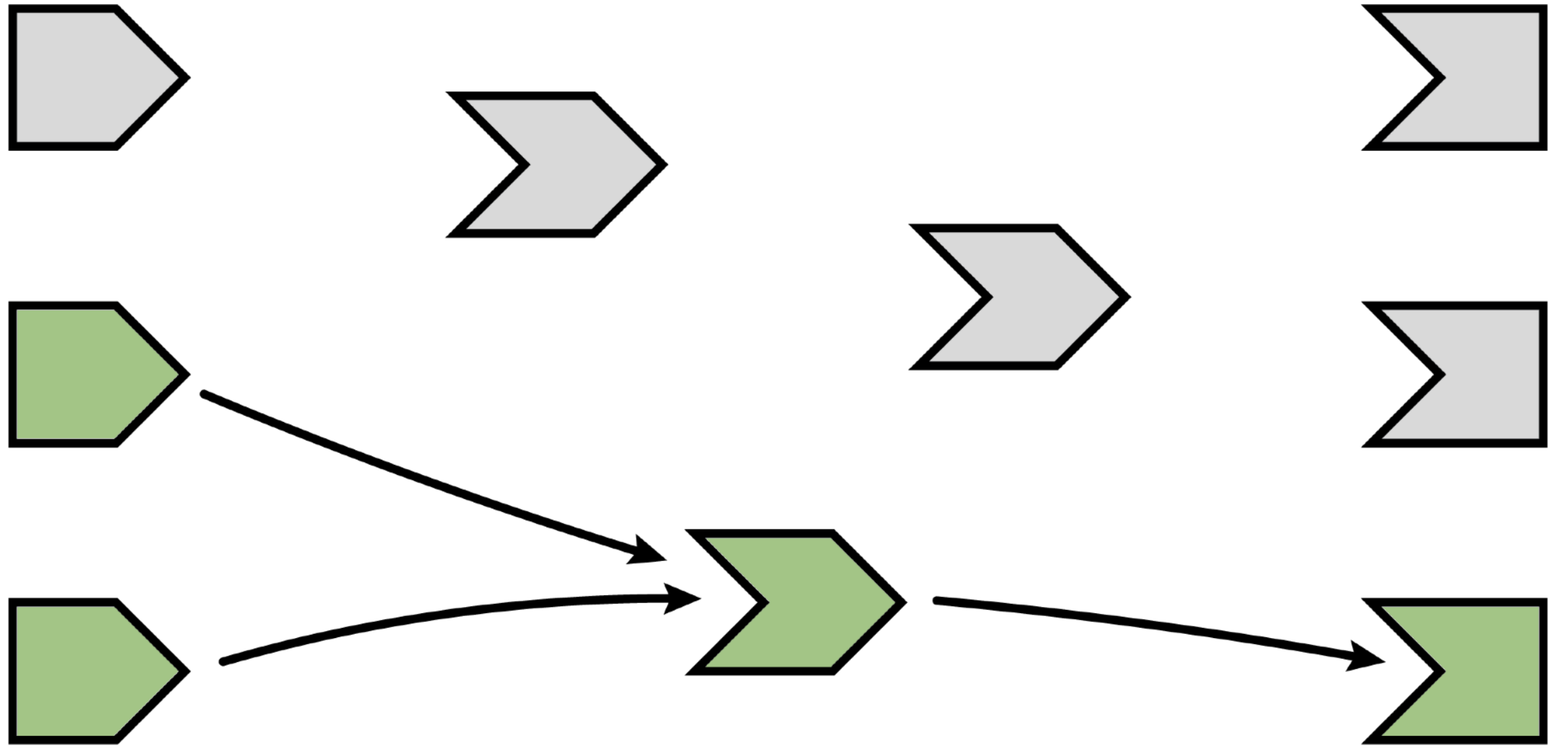
Reactividad

Si el/la usuario/a modifica uno de los inputs, se marca como *invalidado.*



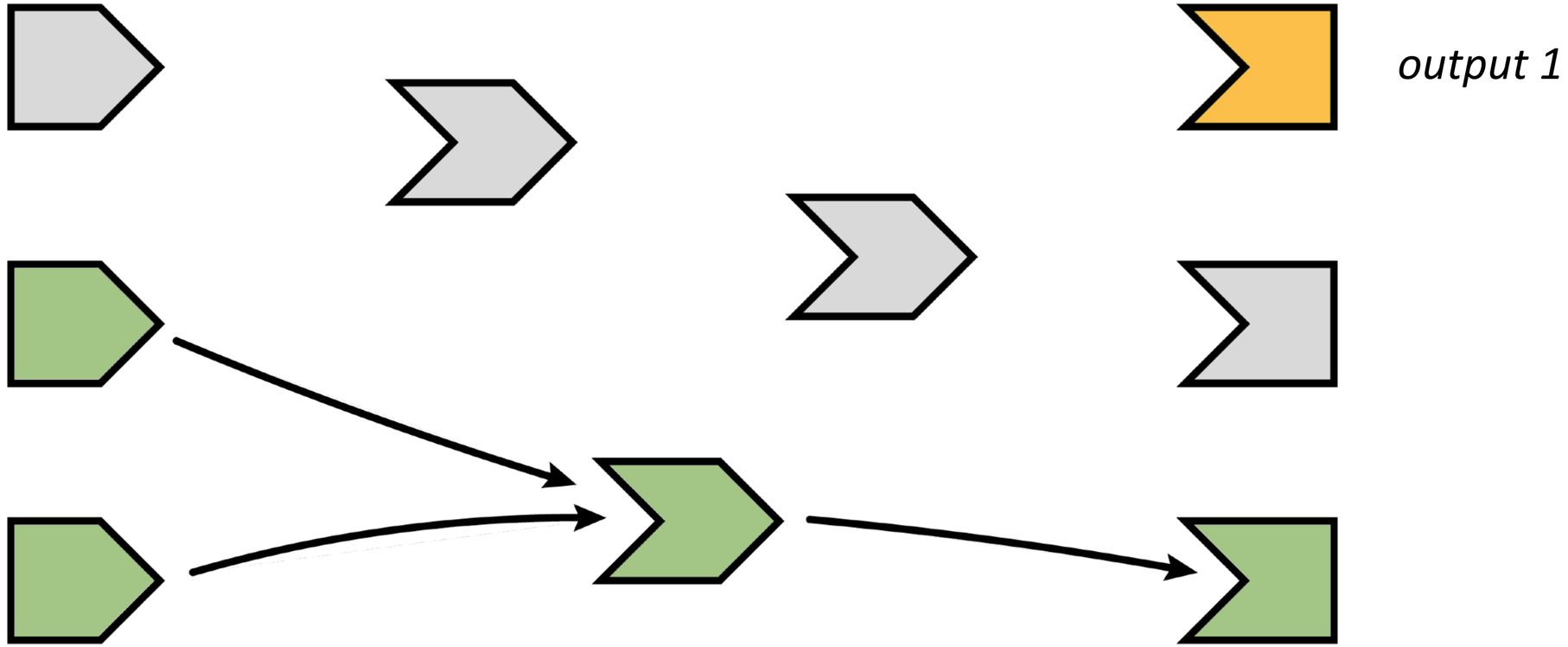
Reactividad

Al invalidarse un input, se invalida también toda la cadena que depende de él, y deben recalcularse en base al nuevo input.



Reactividad

Al haberse invalidado toda la cadena, el output 1 vuelve a necesitar calcularse, y el ciclo de cálculo de las dependencias se repite.



Inputs

- Son elementos de la interfaz de la aplicación que podemos disponer para que nuestros usuarios/as puedan interactuar de la forma que definamos. Sirven para entregar un valor o valores al server por medio del objeto `input`. Cuando el/la usuario/a cambia el input en la aplicación, el objeto `input$x` correspondiente se actualiza, haciendo que se re-ejecuten los reactivos y outputs que lo usen.

selector de elementos definidos en el vector `regiones`, que podrá ser usado en el server como `input$region`

un selector vacío sirve para que sea rellenado/ actualizado desde el server

botón que al ser presionado puede ser detectado por un observador para que se realice una acción

```
selectInput("region",
            "Seleccionar región",
            choices = regiones),

selectInput("comuna",
            "Seleccionar comuna",
            choices = NULL),

actionButton("opciones",
             label = "Más información")
```

Reactivos

- Los objetos reactivos son la forma de hacer un cálculo en R que va a depender de otros elementos. Al ejecutarse, producen otro objeto reactivo que contiene su resultado. Los reactivos se re-ejecutarán si cambia uno de los elementos que se usen en su interior. A su vez, al re-calcularse, se re-calcularán los demás elementos que dependen del objeto actualizado.

recibe el objeto `datos`*, y lo filtra en base al contenido del input* `input$region`, elegido por el/la usuari@

recibe el resultado del reactivo `datos_filtrados_region()`*, y le extrae la columna* `comuna`, y retorna este vector como un nuevo objeto reactivo

```
datos_filtrados_region <- reactive({
    datos |>
      filter(region == input$region)
  })

comunas <- reactive({
    datos_filtrados_region() |>
      pull(comuna) |>
      unique()
  })
```

Renderers

- Son también objetos reactivos, en el sentido que se re-ejecutan si cambia uno de los objetos reactivos en su interior. La diferencia es que los renderers, en vez de retornar objetos normales de R, retornan un output que será visible en la parte gráfica de la aplicación (UI), como pueden ser tablas, textos, gráficos y más. Para ello, escriben su resultado en el objeto `output`.

crea un texto en base al contenido de `comunas()`, *llamado* `texto_comunas`, que se debe llamar en el UI

crea una tabla con los datos de `datos_filtrados_comuna()`, *llamada* `tabla`

```
output$texto_comunas <- renderText({
    comunas()
  })


output$tabla <- render_gt({
    datos_filtrados_comuna() |>
      arrange(desc(hogares)) |>
      gt()
  })
```

Observadores

- Los observadores son reactivos que se ejecutarán si uno de sus elementos interiores cambia. La diferencia con un reactivo es que no producen un resultado (otro objeto reactivo), sino que están solo esperando si algo cambia dentro de sí mismo para ejecutarse. Se usan para ejecutar código con "efectos secundarios" (es decir, que no retorna objetos), como actualizar un selector, mostrar/ocultar elementos, enviar notificaciones, etc.

si cambia `comunas()`, *actualiza el input* `comuna`

si cambia `datos_filtrados_comuna()`, *evalúa la condición, y dependiendo de si es TRUE realiza la acción*

```
observe({
    updateSelectInput(session,
                      "comuna",
                      choices = comunas()
    )
  })

observe({
    if (nrow(datos_filtrados_comuna()) > 10) {
      shinyjs::show("alerta")
    }
  })
```

Observadores

- También hay observadores específicos que sólo se ejecutarán cuando cambie un input o reactivo indicado, sin que necesariamente ese input o reactivo se use en su interior.

cuando cambie `input$opciones` *(por ejemplo, si el usuario/a presiona un botón), se realiza su acción*

```
observeEvent(input$opciones, {
    # mostrar/ocultar elemento
    shinyjs::toggle("texto")
  })
```

Spatial Lab Analytics

Soluciones en análisis de datos

www.spatiallab.cl