

Debug Large Data Apps: shiny + reactlog + sparklyr

Barret Schloerke
RStudio / Shiny Development Team
  @schloerke

slides: <http://bit.ly/adv-r-2019>

R

Overview

- Recap: shiny
- Debugging shiny: shiny + reactlog
- Large Data: sparklyr
- Large Data w/ shiny: shiny + sparklyr
- Combine: shiny + sparklyr + reactlog
- shiny Optimization



Shiny

Shiny

- Website: <https://shiny.rstudio.com/>
- “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.”
 - tl;dr; Make users full stack developers in as few lines as possible
- `shiny::runExample("01_hello")`



Shiny: App Parts

1. ui

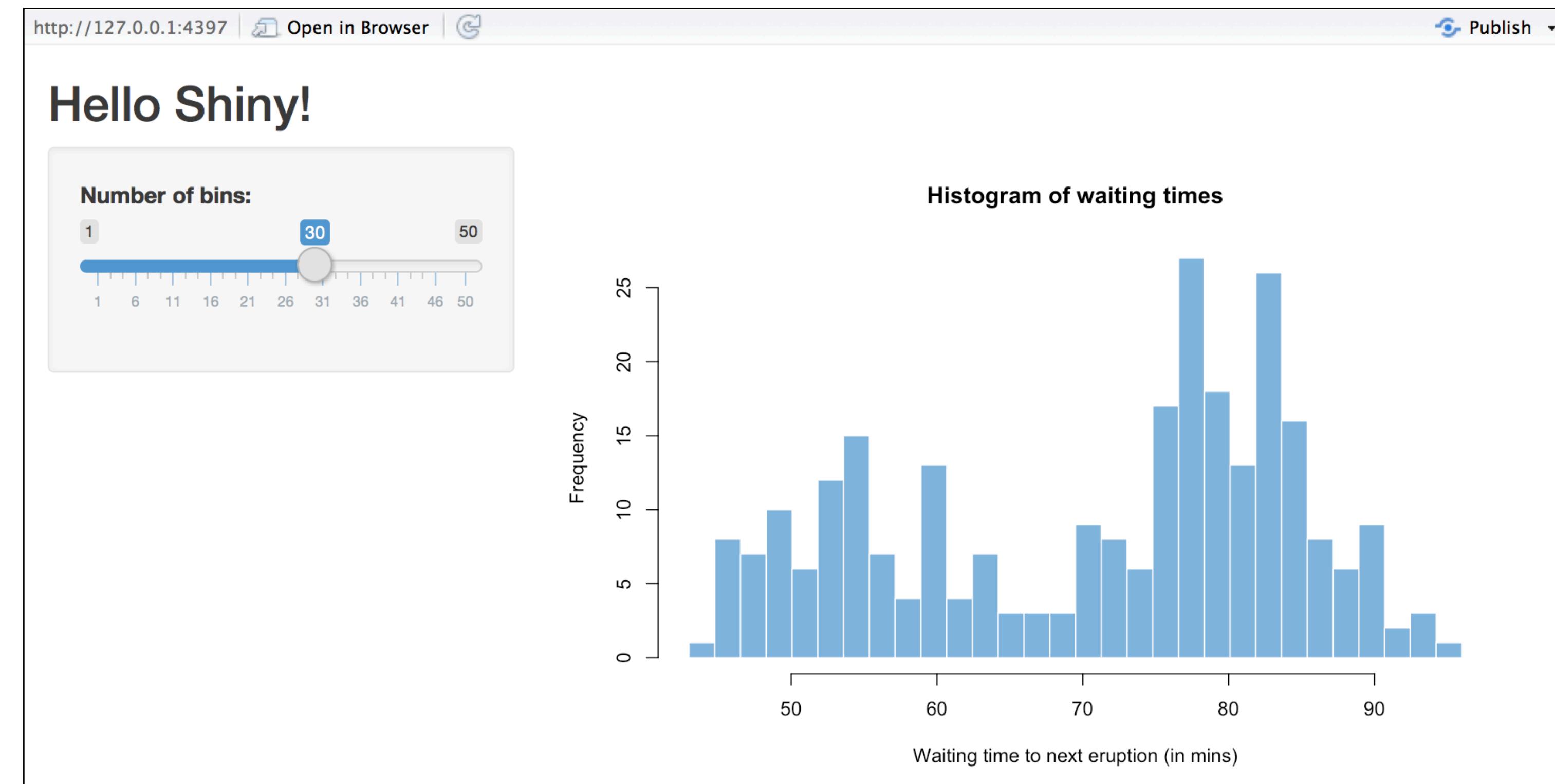
- User Interface
- How it looks

2. server

- How it works

3. shinyApp(ui, server)

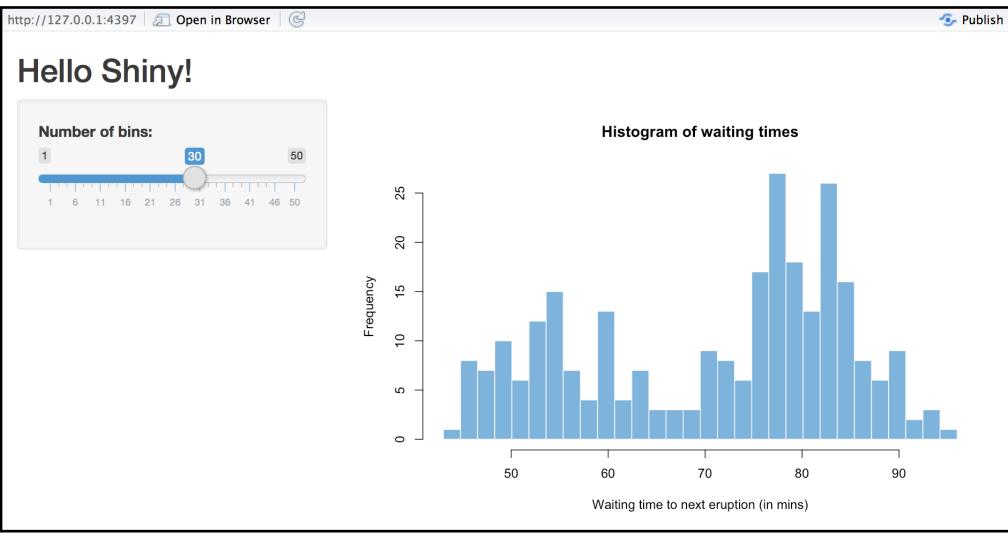
4. In a app.R file



shiny::runExample("01_hello")

```
# Define UI for app that draws a histogram ----  
ui <- fluidPage(  
  
  # App title ----  
  titlePanel("Hello Shiny!"),  
  
  # Sidebar layout with input and output definitions ----  
  sidebarLayout(  
  
    # Sidebar panel for inputs ----  
    sidebarPanel(  
  
      # Input: Slider for the number of bins ----  
      sliderInput(inputId = "bins",  
                  label = "Number of bins:",  
                  min = 1,  
                  max = 50,  
                  value = 30)  
    ),  
  
    # Main panel for displaying outputs ----  
    mainPanel(  
  
      # Output: Histogram ----  
      plotOutput(outputId = "distPlot")  
    )  
)
```

```
# Define server logic required to draw a histogram ----  
server <- function(input, output) {  
  
  # Histogram of the Old Faithful Geyser Data ----  
  # with requested number of bins  
  # This expression that generates a histogram is wrapped in a  
  # call to renderPlot to indicate that:  
  #  
  # 1. It is "reactive" and therefore should be automatically  
  #    re-executed when inputs (input$bins) change  
  # 2. Its output type is a plot  
  output$distPlot <- renderPlot({  
  
    x     <- faithful$waiting  
    bins <- seq(min(x), max(x), length.out = input$bins + 1)  
  
    hist(x, breaks = bins, col = "#75AADB", border = "white",  
          xlab = "Waiting time to next eruption (in mins)",  
          main = "Histogram of waiting times")  
  })  
}
```



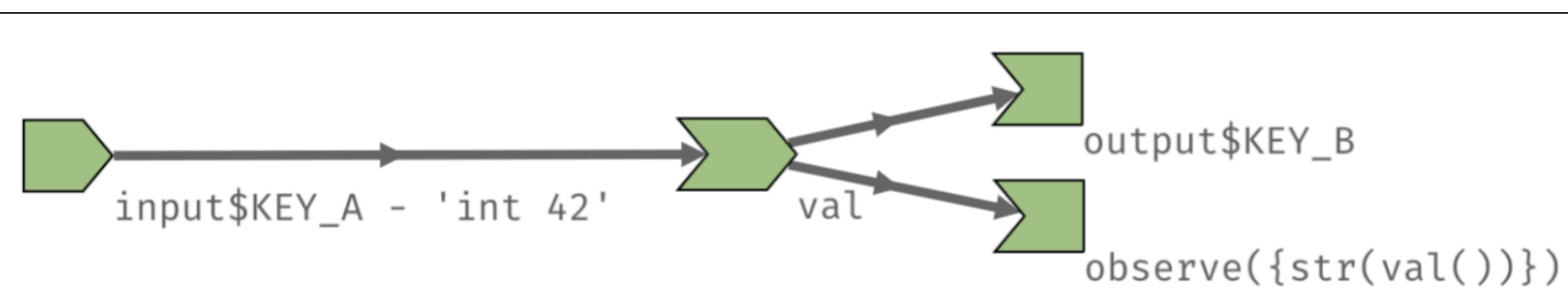
A photograph showing a massive flock of birds, likely cormorants, flying in a dense cluster against a dark sky. They are silhouetted against a lighter area, possibly the horizon or a setting sun. In the foreground, the underside of a bridge arch is visible, and in the background, city buildings are faintly visible across a body of water.

Demo:

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

Reactive Programming

- Reactive Programming -
Paradigm concerned about the propagation of change
- Three types of reactive elements



```
# Sources  
input$KEY_A  
  
# Conductors  
val <- reactive({  
    input$KEY_A + 1  
})  
  
# Endpoints  
output$KEY_B <- renderPrint({  
    val ()  
})  
  
observe({  
    str(val())  
})
```

A photograph of a large flock of birds, likely cormorants, flying in a dense cluster over a bridge at dusk. The sky is a deep blue, and the city skyline is visible in the background.

Demo:

```
shiny::runGitHub(  
  "rstudio/cranwhales",  
  ref = "sync"  
)
```



reactlog: Debugging the State of Shiny

Situation

- Built a Shiny application - 
- Add a new reactive feature to your application - 
- Reactive output does not update - 
- No errors in console - 

Debugging Reactive Code

- Debugging reactive code within a *working* Shiny application is **not a trivial task**
- Must know the reactive state:
 - Value
 - Dependencies / Invalidations
 - ... over time!

Solution:
reactlog !!

What is reactlog?

- Website: <https://rstudio.github.io/reactlog/>
- “A snapshot of the history (“log”) of all reactivity interactions within a shiny application”
- Traverse the reactive log forwards or backwards in time
- Search for defined reactive objects
- Filter to a reactive object’s family tree (dependency tree)
- Built on [cytoscape.js](#)

Setup

- Install

```
• install.packages(  
  "shiny", dependencies = TRUE  
)
```

- Dependency of shiny v1.3.2

- Usage

```
• # enable reactlog  
# before running your app!!  
options(shiny.reactlog = TRUE)
```

```
# run your shiny app  
shiny::runApp()
```

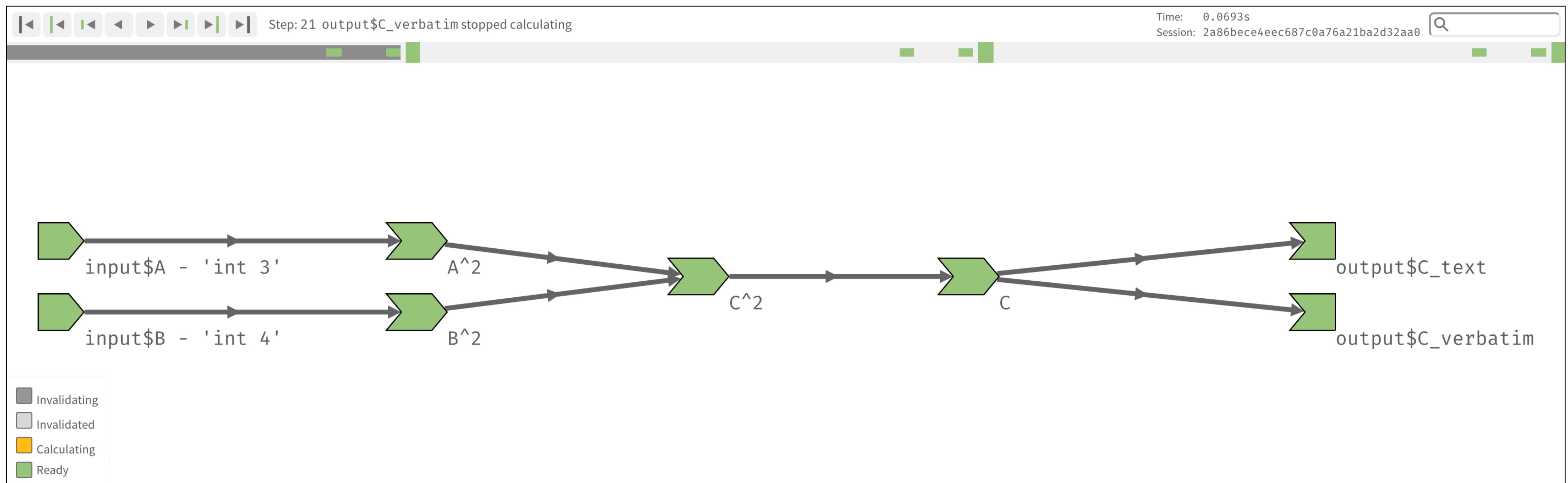
```
# in your app...  
#   Mac: `cmd + F3`  
#   Windows: `ctrl + F3`  
# or...  
shiny::reactlogShow()
```

Debugging Shiny Apps with reactlog



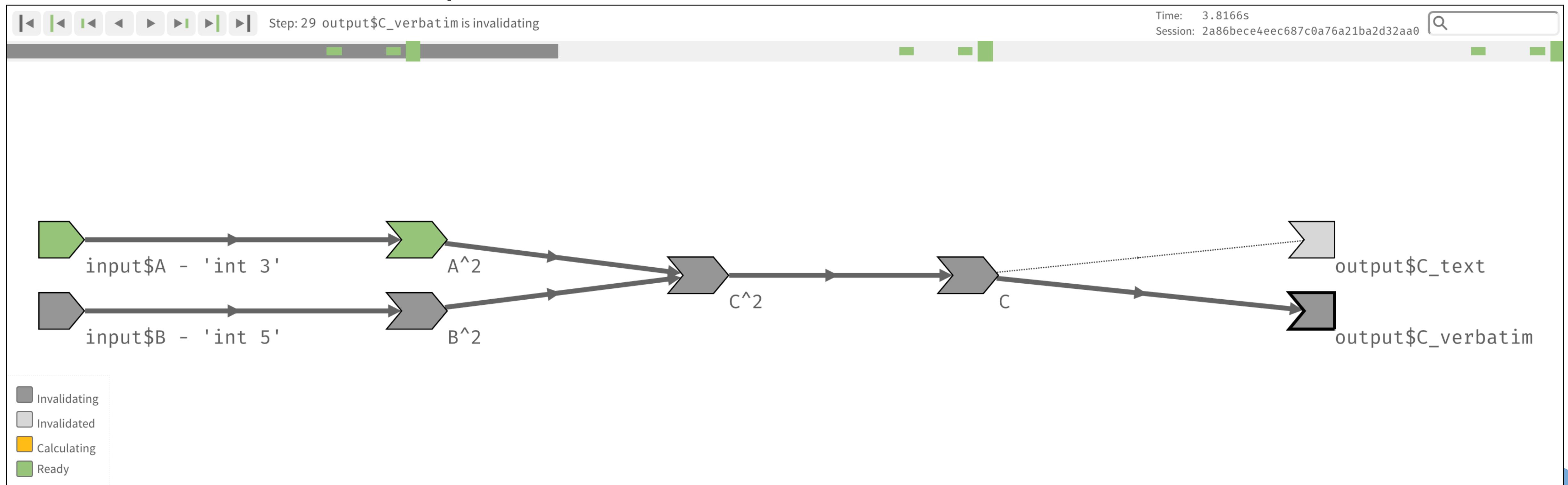
(Working) Pythagoras

- **Demo:** <http://bit.ly/adv-r-2019-cloud>



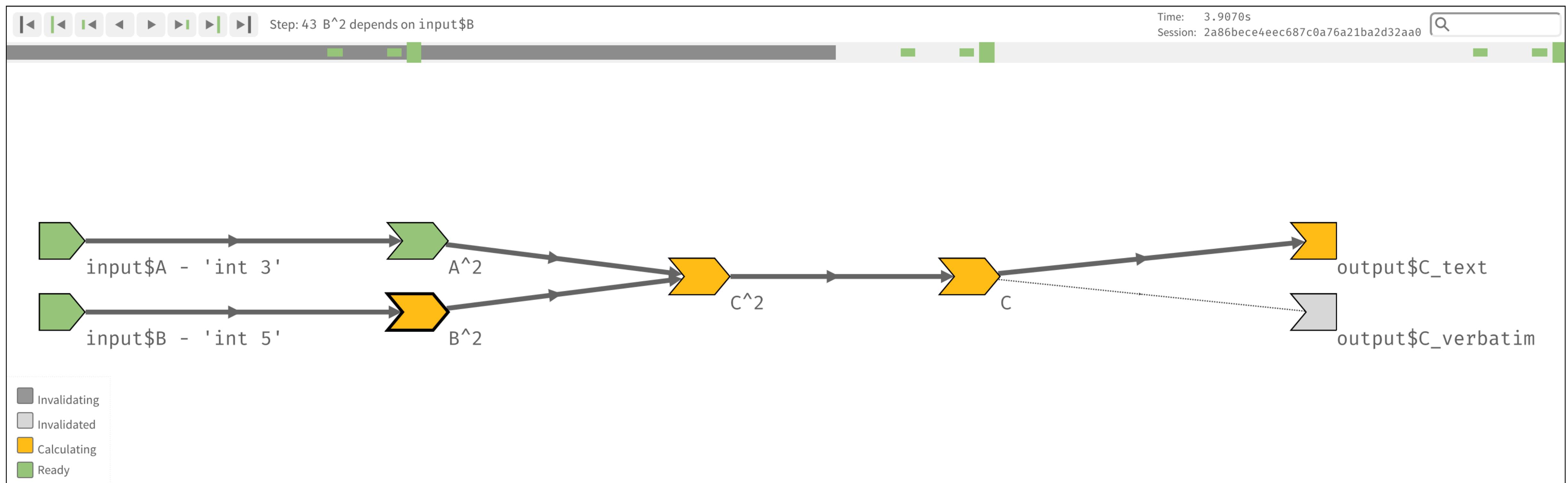
(Working) Pythagoras

- Changed Value of `input$B` to 5
- All downstream dependencies are invalidated



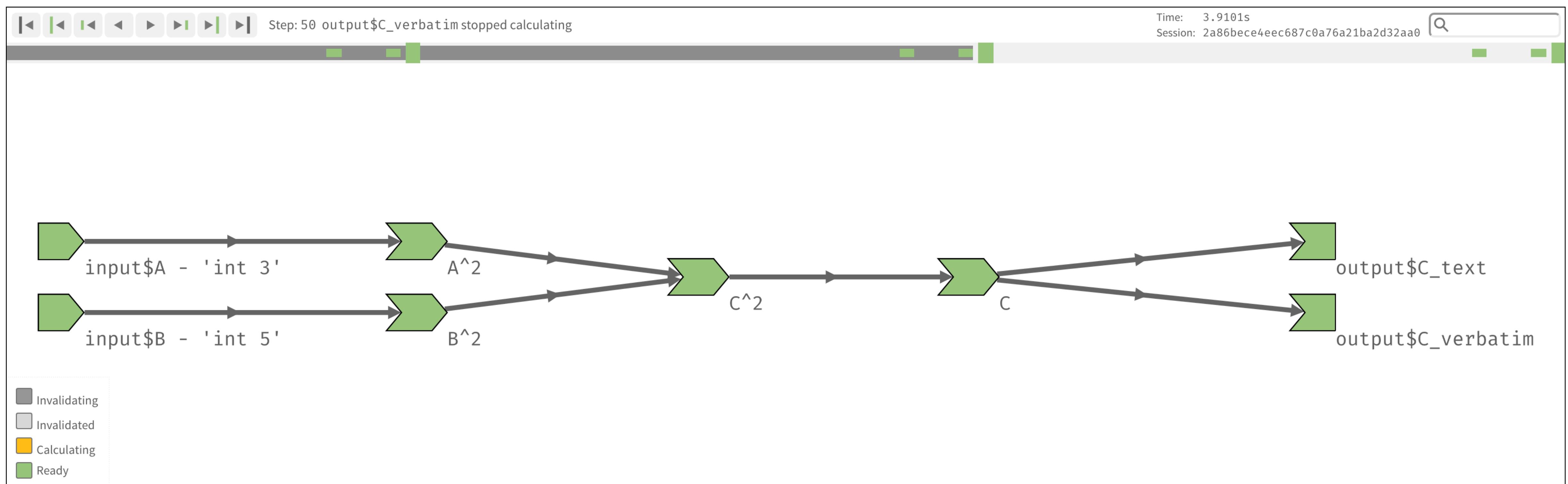
(Working) Pythagoras

- All visible output values are re-calculated



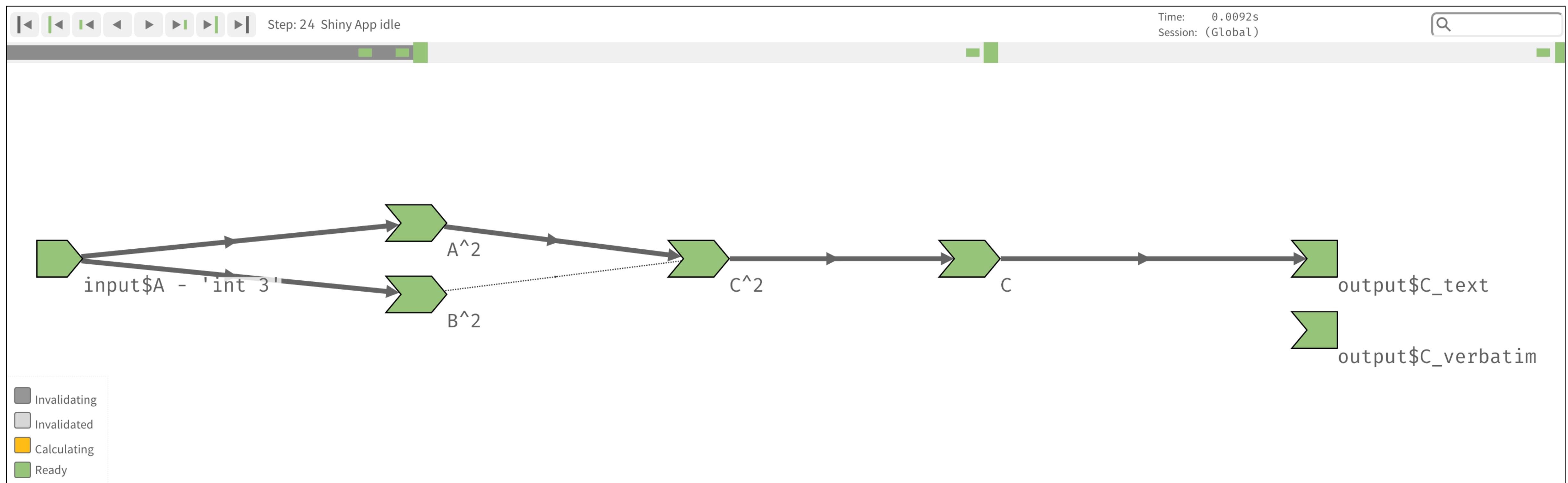
(Working) Pythagoras

- Back to steady state (Shiny is idle)



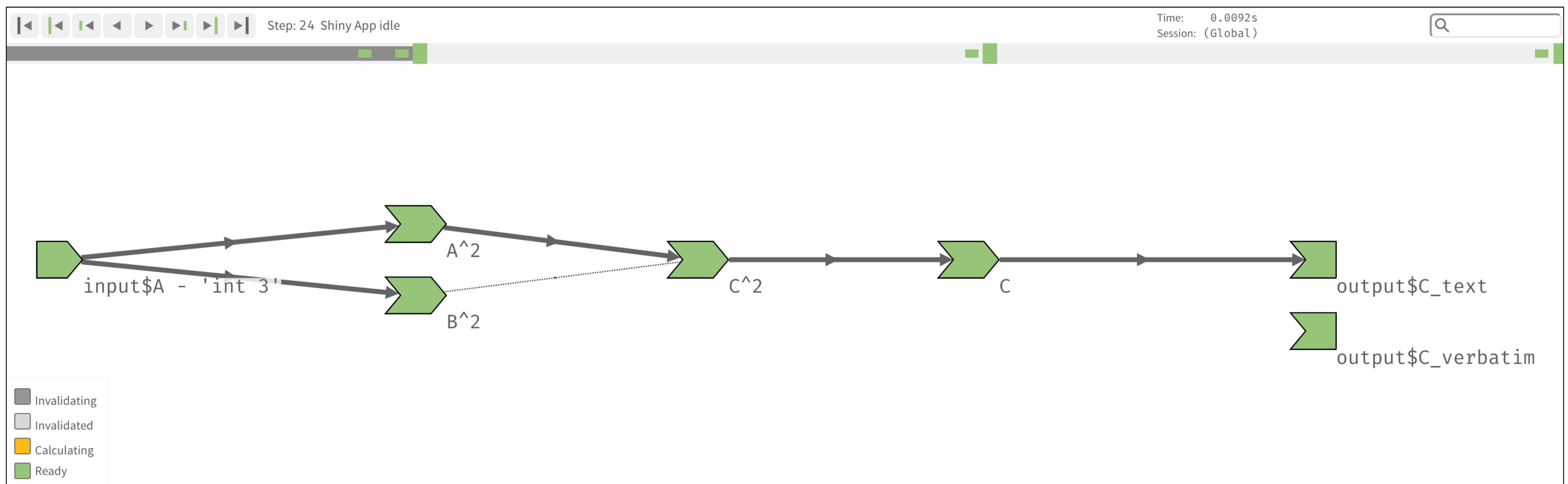
Broken Pythagoras

- **Demo:** <http://bit.ly/adv-r-2019-cloud>



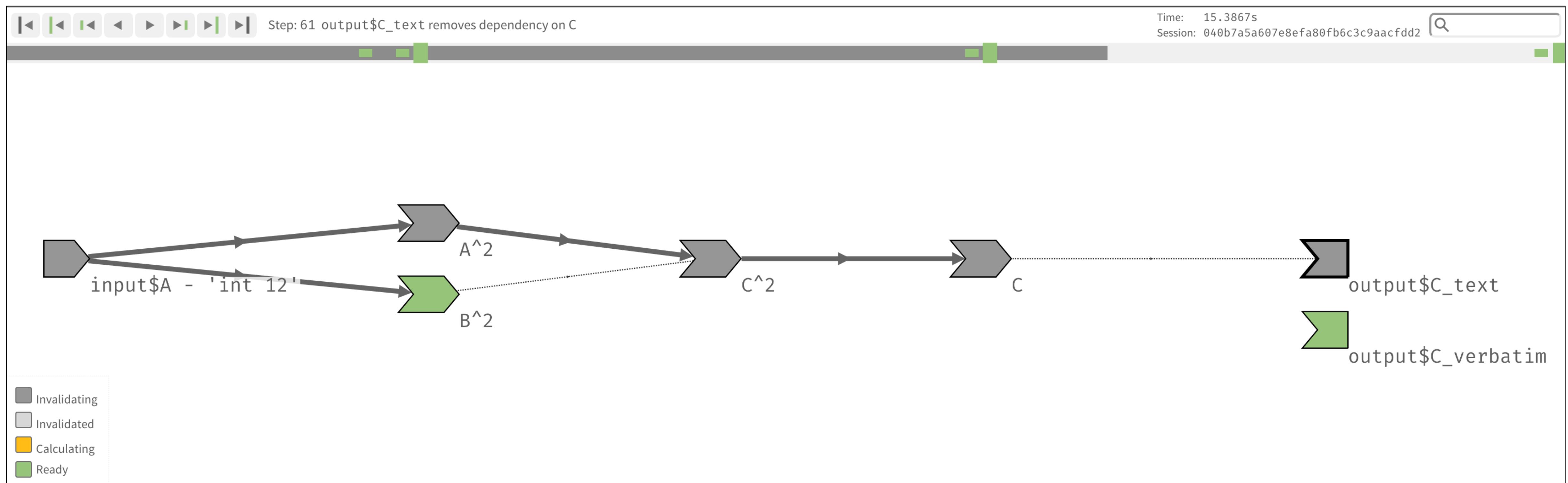
Broken Pythagoras

- Problem: `input$A` is used to calculate B^2



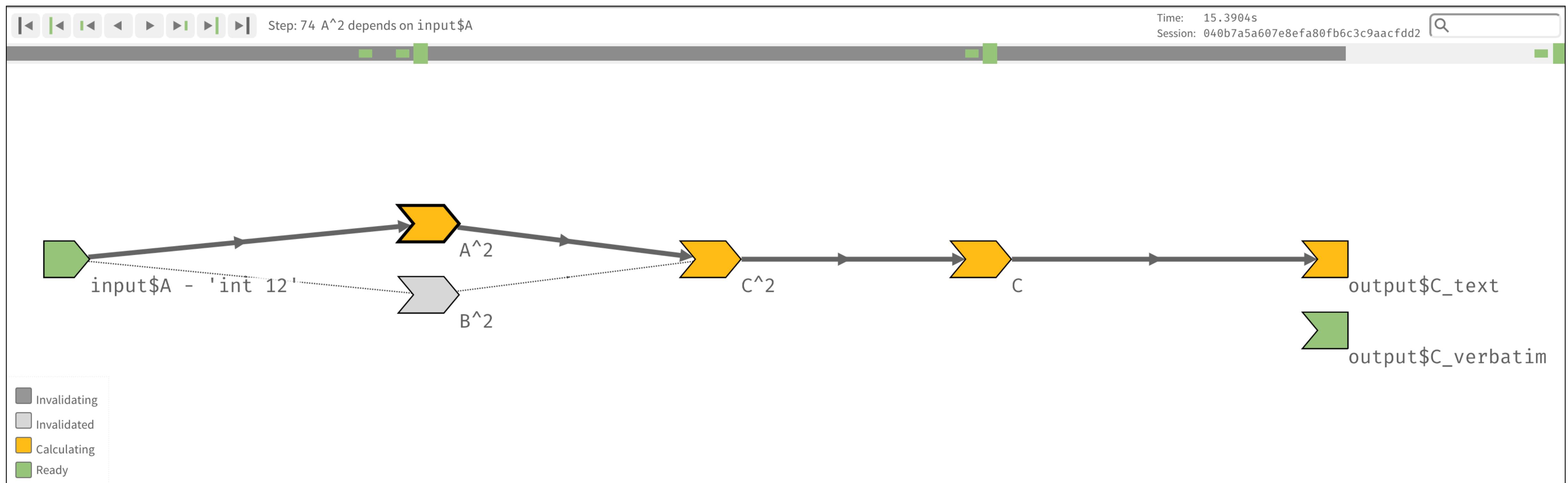
Broken Pythagoras

- Problem: `C_verbatim` is **never** invalidated



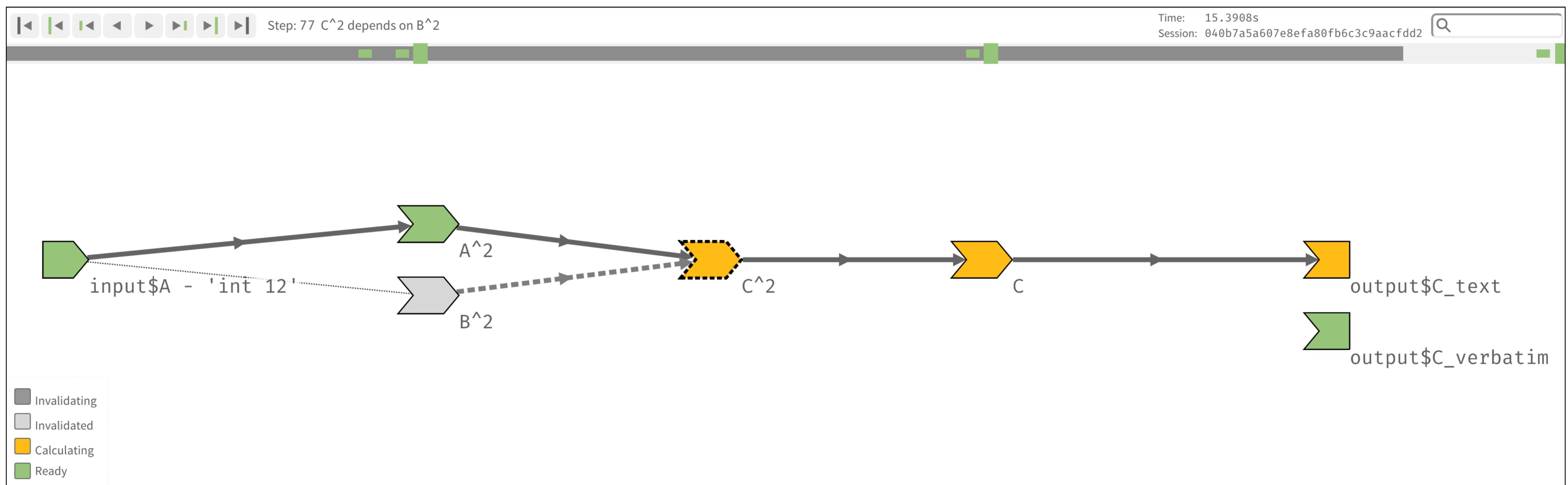
Broken Pythagoras

- Calculation looks correct... so far



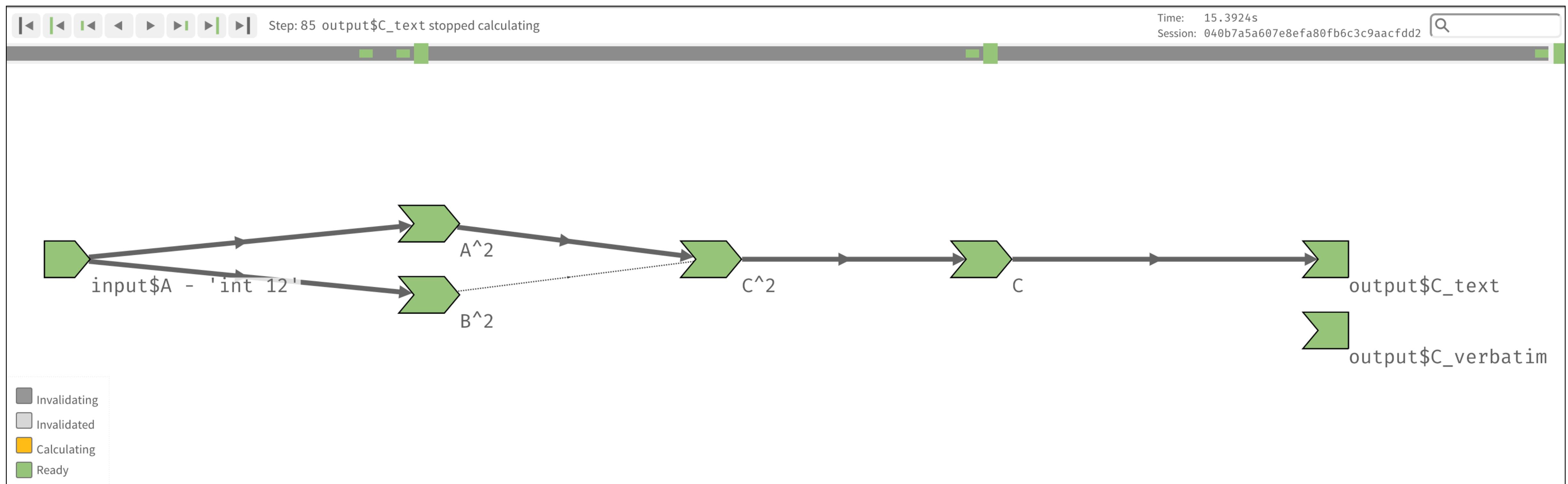
Broken Pythagoras

- Problem: Unwanted `isolate()` call in C^2 to B^2



Broken Pythagoras

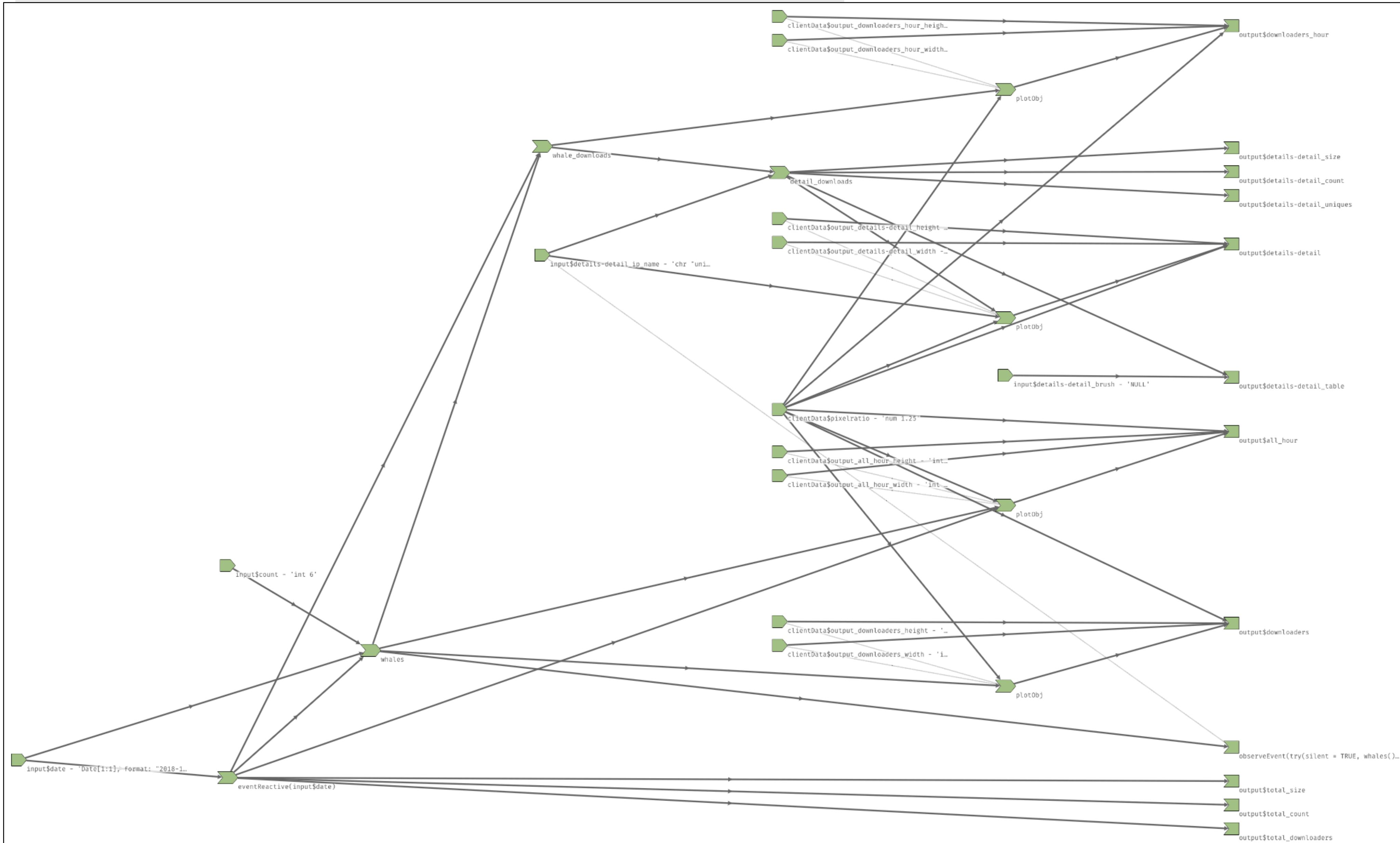
- Back to steady state (Shiny is idle)



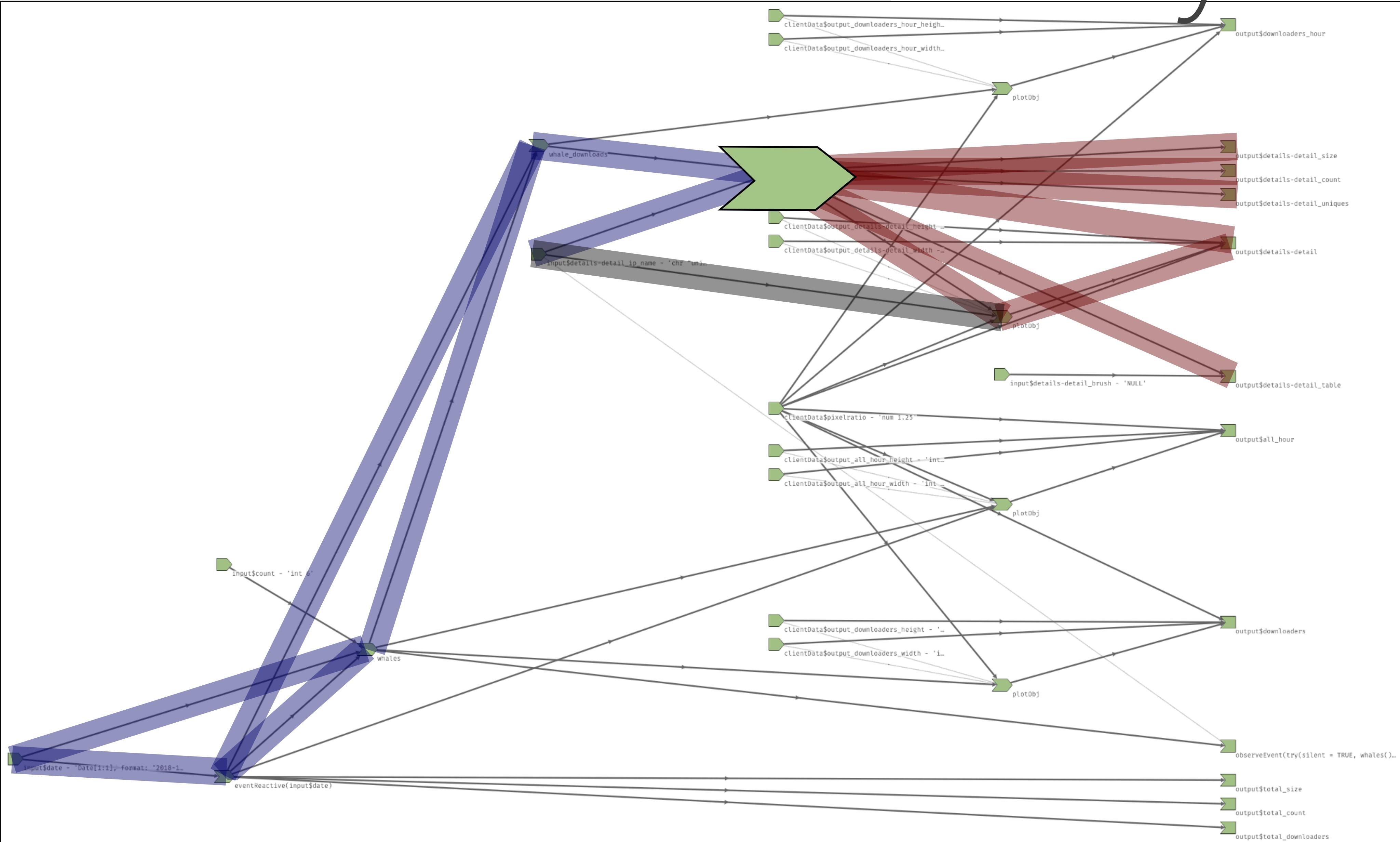
Searching within reactlog

- cranwhales: github.com/rstudio/cranwhales
- Run app:
 - ```
options(shiny.reactlog = TRUE)
shiny::runGitHub("rstudio/cranwhales", ref = "sync")
```
- Very dense application!
- Using the search bar in the top right,  
search for "detail\_downloads" to filter the graph

# cranwhales is dense!

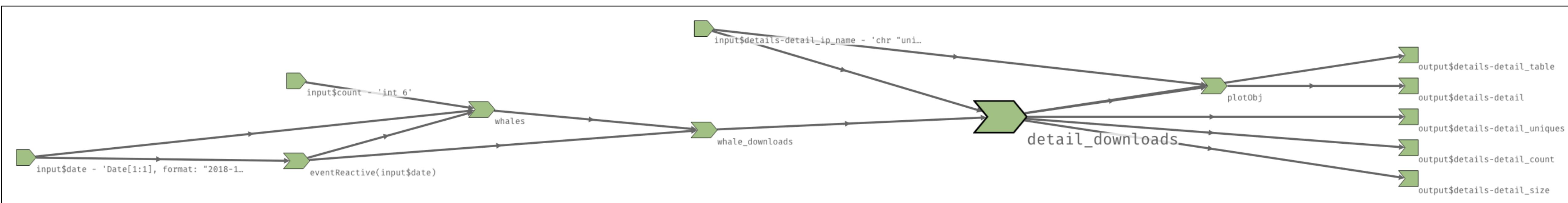


detail\_downloads family tree



R

# detail\_downloads family tree



# Things to Remember

- Shiny outputs that are not visible are not calculated. ....This is on purpose!  
(Effective Reactive Programming - Joe Cheng @ Shiny DevCon 2016 “Part 1”@46:34)
- If new reactive objects are being created for every user interaction,  
you may have coded an “anti-solution”  
(Shiny DevCon 2016 “Part 1”@17:00)
  - Will cause reactlog graph to become **very** large
  - Maybe use a reactive() vs an observe()?
- reactlog **is NOT** a *performance* debugger. reactlog **is** a *reactivity* debugger
  - Use profvis for *performance analysis*
- Do not keep `options(show.reactlog = TRUE)` when deploying to production

# reactlog: Future Ideas

- Visually differentiate separate user sessions
- Display the value of an endpoint (`output`) or conductor (`reactive()`)
- Add expandable / collapsable groups for a set of reactive objects
  - Ex: `renderPlot` is made of 5+ reactive objects...  
*really only* one reactive component for a user
- Remove reactive objects from the `reactlog` graph that have been garbage collected
- Combine `reactlog` and `profvis` to analyze a Shiny application's performance and reactive state simultaneously

reactlog  
Questions?



sparklyr



R

# Data Sizes

- "Small" data
  - In-memory only (typical R)
  - Fast!
- "Medium" data
  - On-disk data sets (MySQL)
  - Disk read time is slower
- "Large" data
  - Distributed (spark)
  - Communication is very slow
  - Parallel computation power is high



# sparklyr

- sparklyr: R interface for Apache Spark
  - GitHub: <https://github.com/rstudio/sparklyr>
  - Website: <https://spark.rstudio.com/>
- Why use sparklyr?
  - (personally) Provides a complete dplyr backend
  - Many natively implementations of Machine Learning methods for spark



A photograph showing a massive flock of birds, likely crows or ravens, flying in a dense, swirling pattern against a darkening sky. In the foreground, a large concrete bridge arch spans across the frame. In the background, city buildings are visible along the horizon.

# Demo:

sparklyr - plot flight delay by distance

~ 5min

The R programming language logo, which consists of a white letter "R" inside a blue circular background.



# Demo:

sparklyr - linear model of flight delay by dist

~ 5 min

sparklyr  
Questions?

A blue-tinted photograph showing the intricate eaves and roofline of a traditional Chinese building, likely a temple or palace. The architecture features multiple layers of curved roofs with decorative ridges.

shiny + sparklyr

A small, light blue circular icon containing a white capital letter 'R', representing the R programming language.



# Demo:

- Accomplished
  - Make plot of delay
  - Make model of delay
- Goals
  - Display plot
  - Display model / model outputs
  - Choose a dist threshold
  - Choose train / test percentages

~15 mins

sparklyr + shiny

Questions?

A photograph showing a massive flock of birds, likely starlings, in flight against a darkening sky. They are silhouetted, creating a dense, swirling pattern. In the foreground, the underside of a bridge arch is visible, and in the background, city buildings are faintly lit.

# Demo:

sparklyr + shiny + reactlog

~5 mins

The R programming language logo, which consists of a white letter 'R' inside a blue circle.

# Shiny Optimization

# Shiny Optimization

## 1. Move work outside of Shiny (very often)

- Preprocess data outside of the server function

## 2. Make code faster (very often)

- Use `profvis::profvis` to find bottle necks

## 3. Use caching (sometimes)

- See #1.
- Use `shiny::renderCachedPlot`

## 4. Use `async` (rare)

- Only after all options have been exhausted

# Questions?

-   @schloerke
- **shiny:** <https://shiny.rstudio.com>
- **reactlog:** <https://rstudio.github.io/reactlog>
- **sparklyr:** <https://spark.rstudio.com>
- **Slides:** <http://bit.ly/adv-r-2019>

