

Reactlog 2.0: Debugging the State of Shiny

Barret Schloerke
Shiny Development Team

  @schloerke

slides: bit.ly/rstudio-conf-2019-reactlog

rstudio::conf
AUSTIN

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!

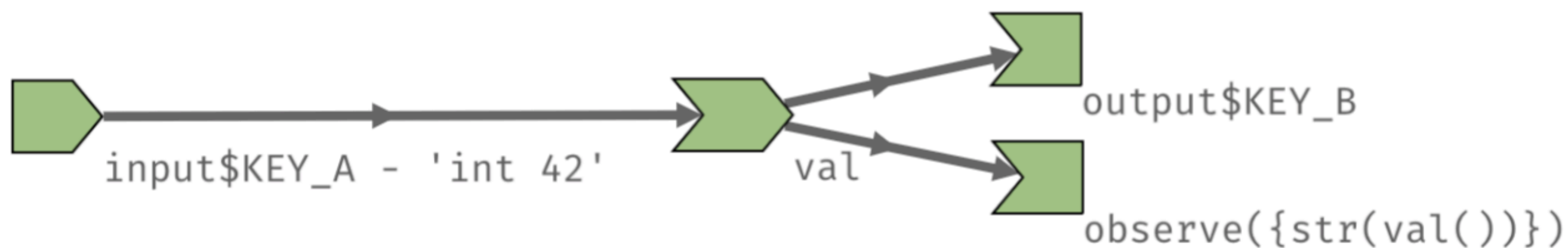
Solution:
reactlog !!

What is `reactlog`?

- `reactlog` - A snapshot of the history (**log**) of all **reactive** interactions within a shiny application
- (Review Reactive Programming)

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())
})
```

What is `reactlog`?

- `reactlog` - A snapshot of the history ("log") of all reactivity interactions within a shiny application
- (Review Reactive Programming)
- 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

- `devtools::install_github("rstudio/reactlog")`

- Will be a natural dependency of Shiny in v1.3.0

- 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...`
`showReactLog()`

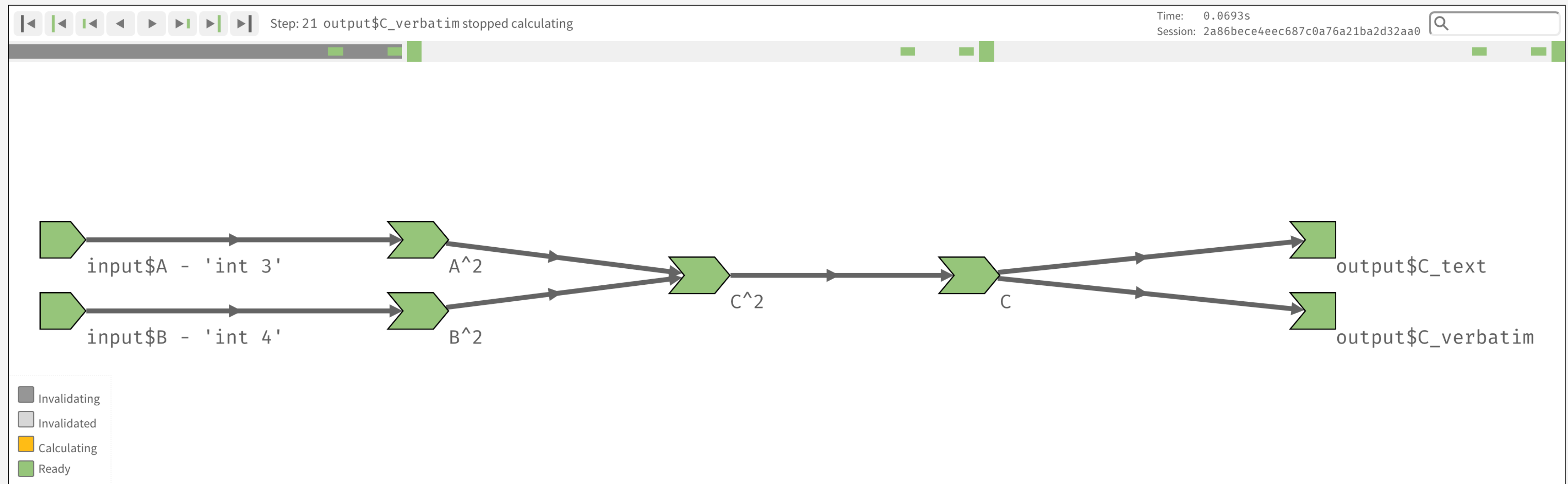
reActivity Being Recorded

- Reactivity
 - Define a reactive object
 - Start/Stop invalidating a reactive
 - Stop/Stop isolating reactive values
 - Add/Remove a reactive dependency
 - Freeze/Thaw a reactive value
- Values
 - Value changes
 - Start/Stop calculations
- Extra
 - User marked time points
 - Shiny is idle

Debugging Shiny Apps with reactlog

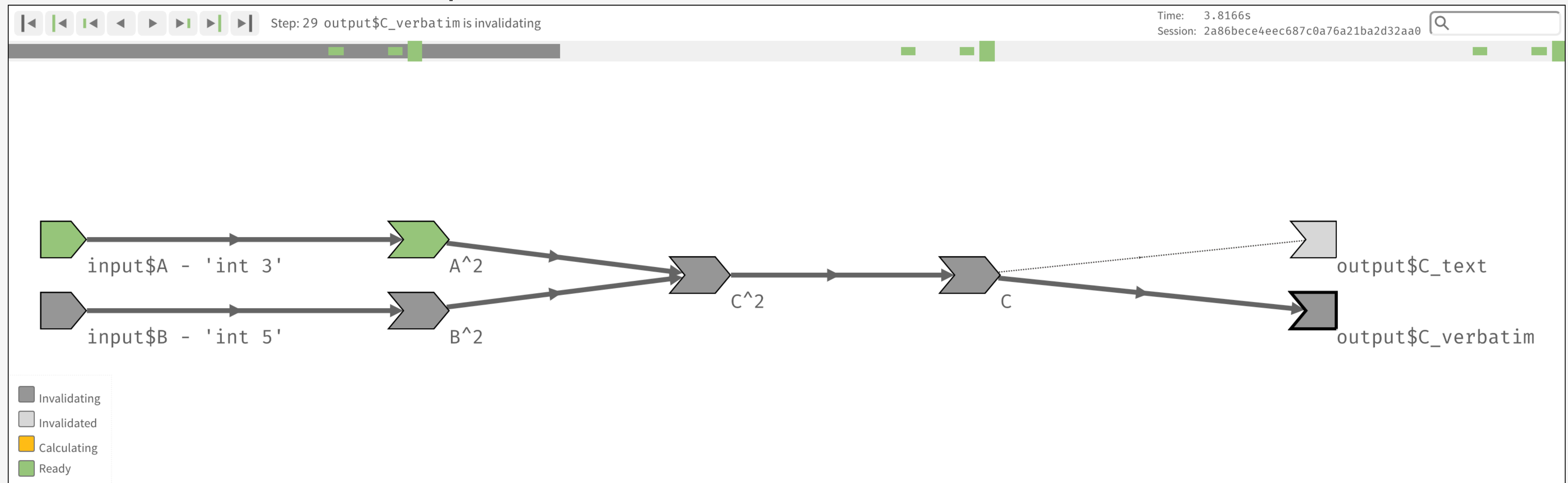
(Working) Pythagoras

- **Demo:** <https://rstudio.cloud/project/170132>



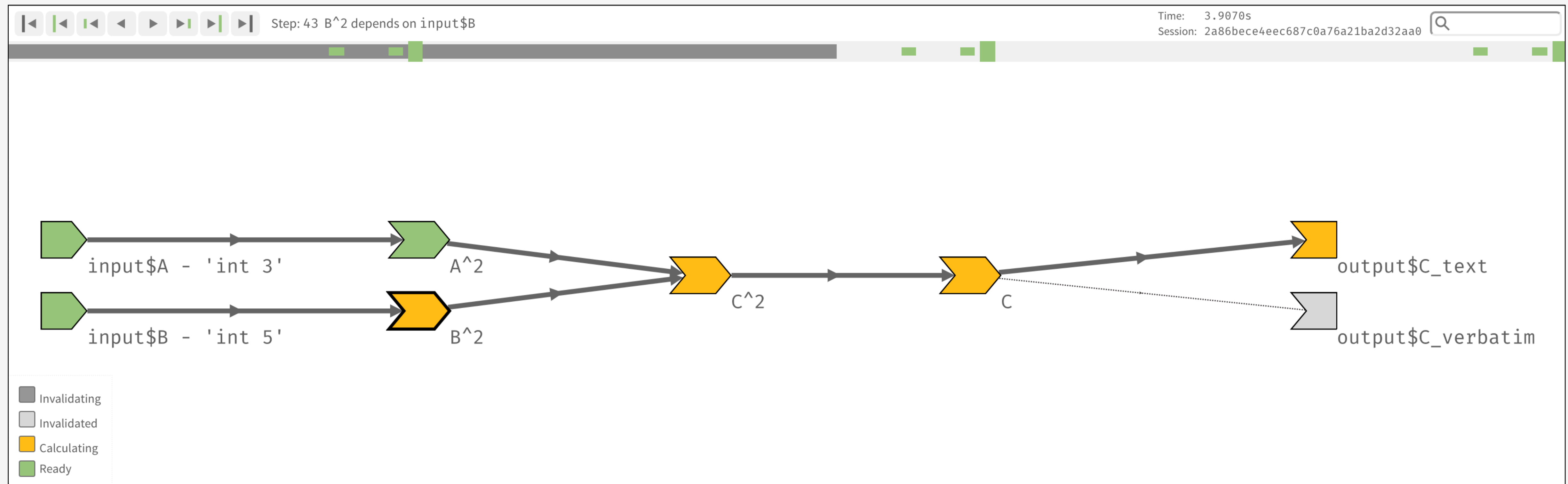
(Working) Pythagoras

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



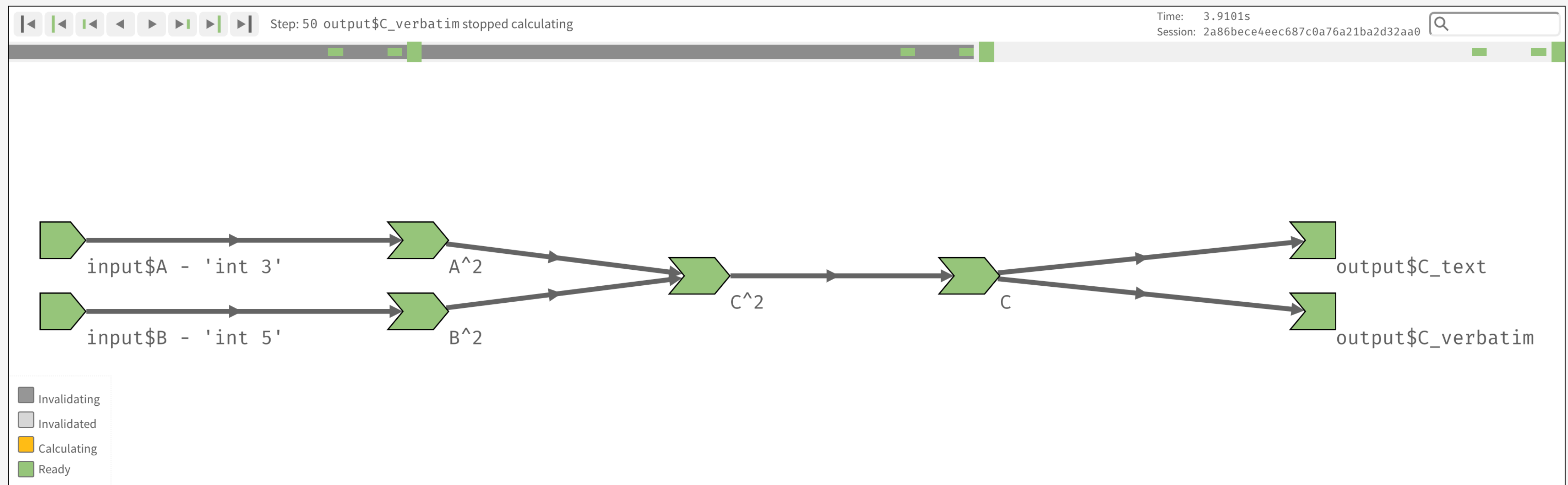
(Working) Pythagoras

- All visible output values are re-calculated



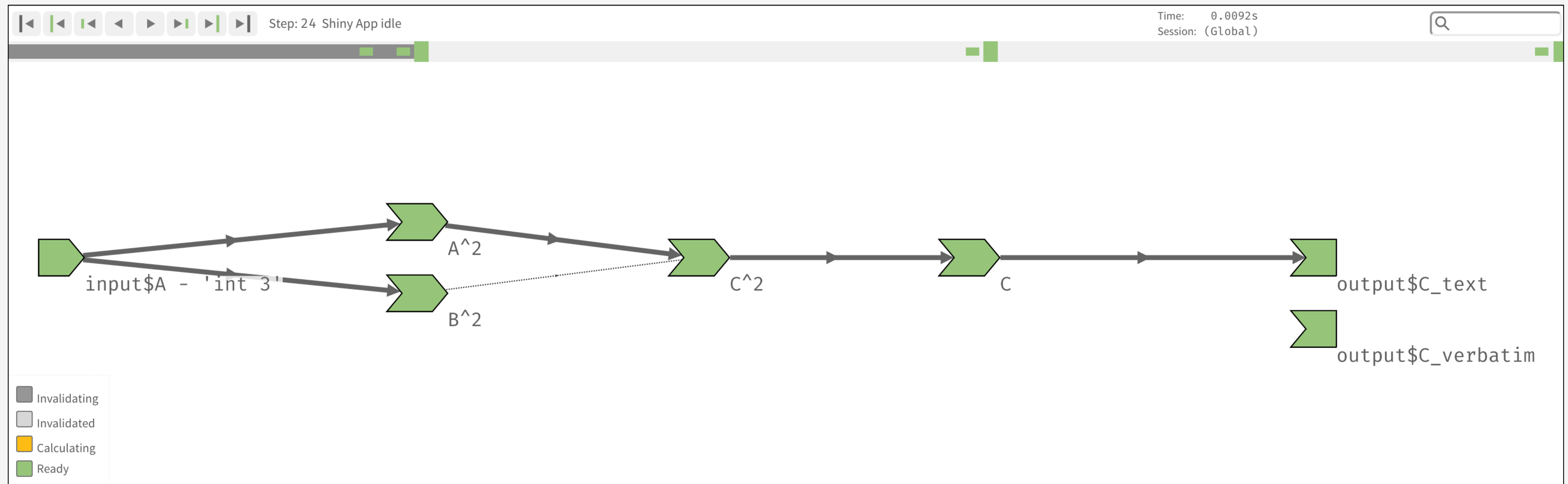
(Working) Pythagoras

- Back to steady state (Shiny is idle)



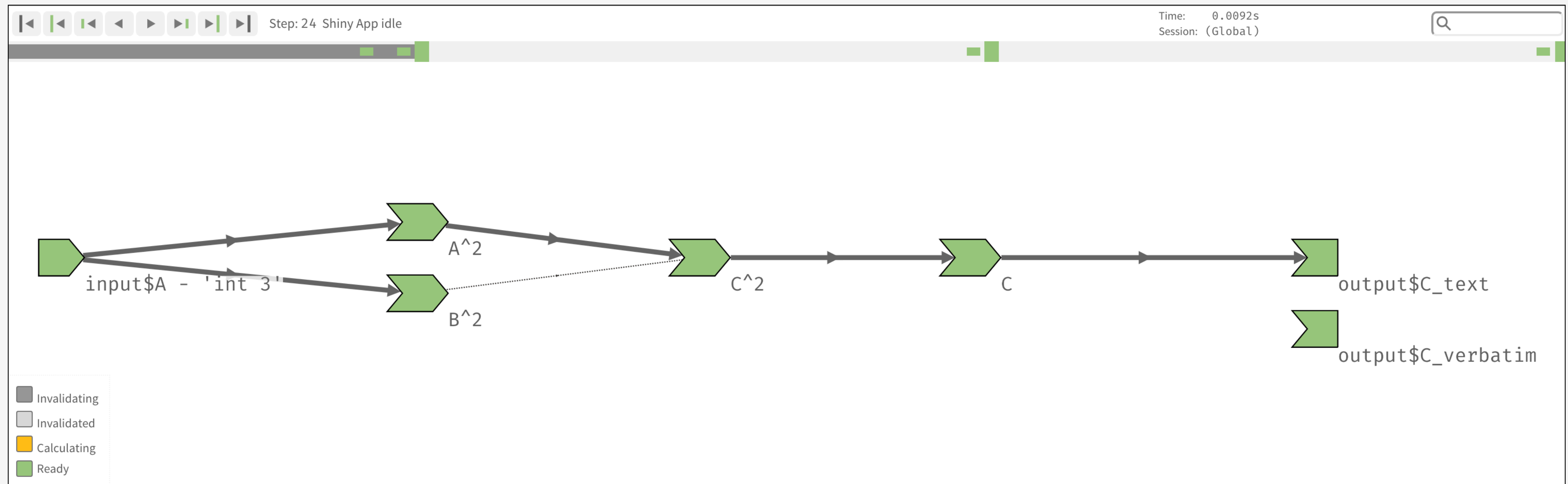
Broken Pythagoras

- **Demo:** <https://rstudio.cloud/project/170132>



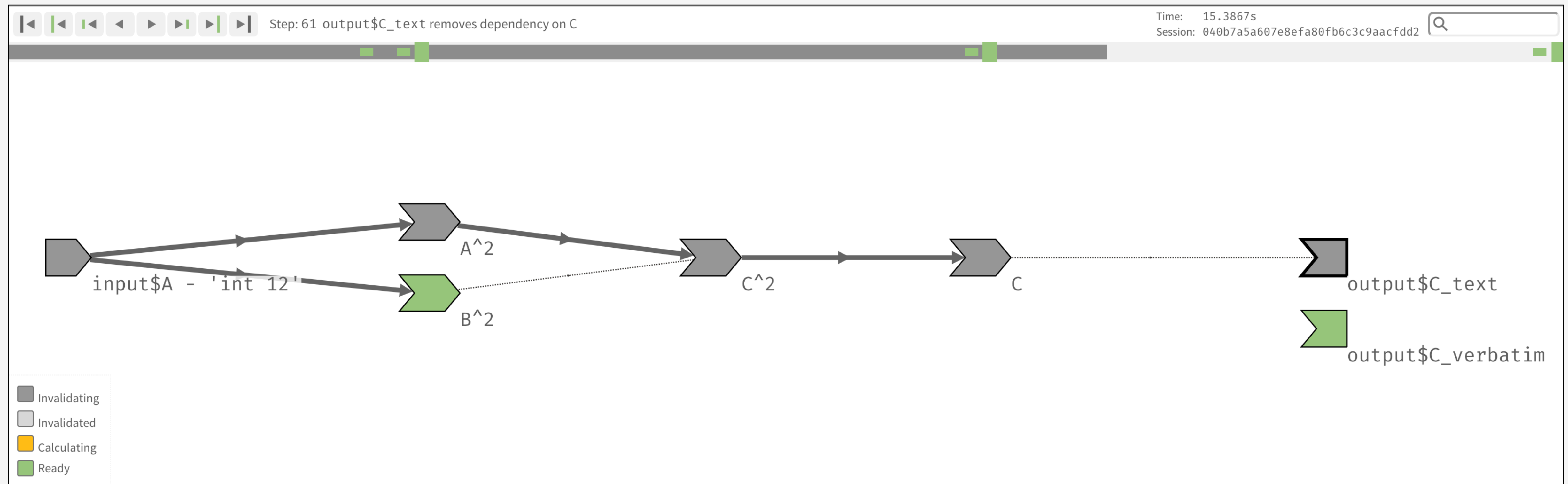
Broken Pythagoras

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



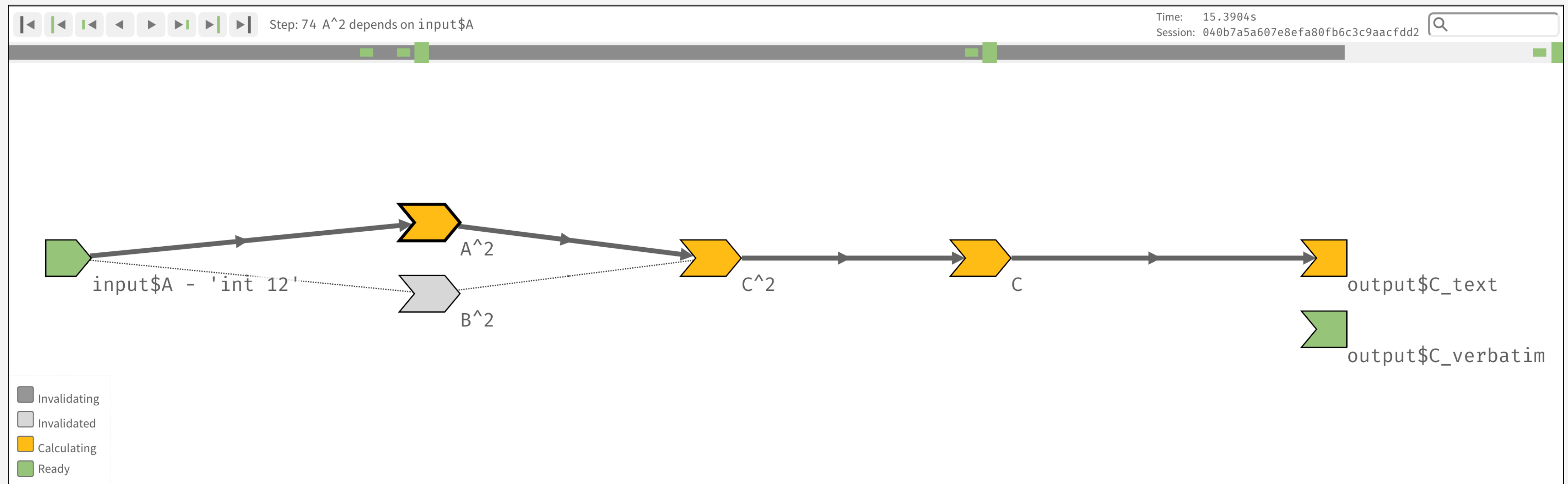
Broken Pythagoras

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



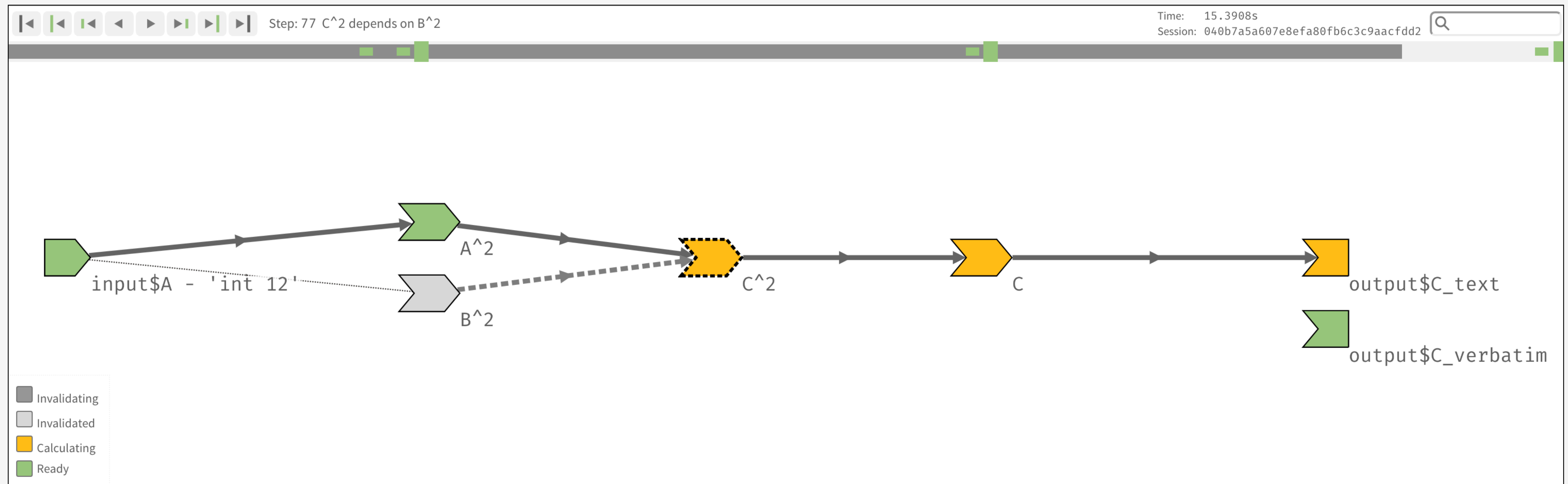
Broken Pythagoras

- Calculation looks correct... so far



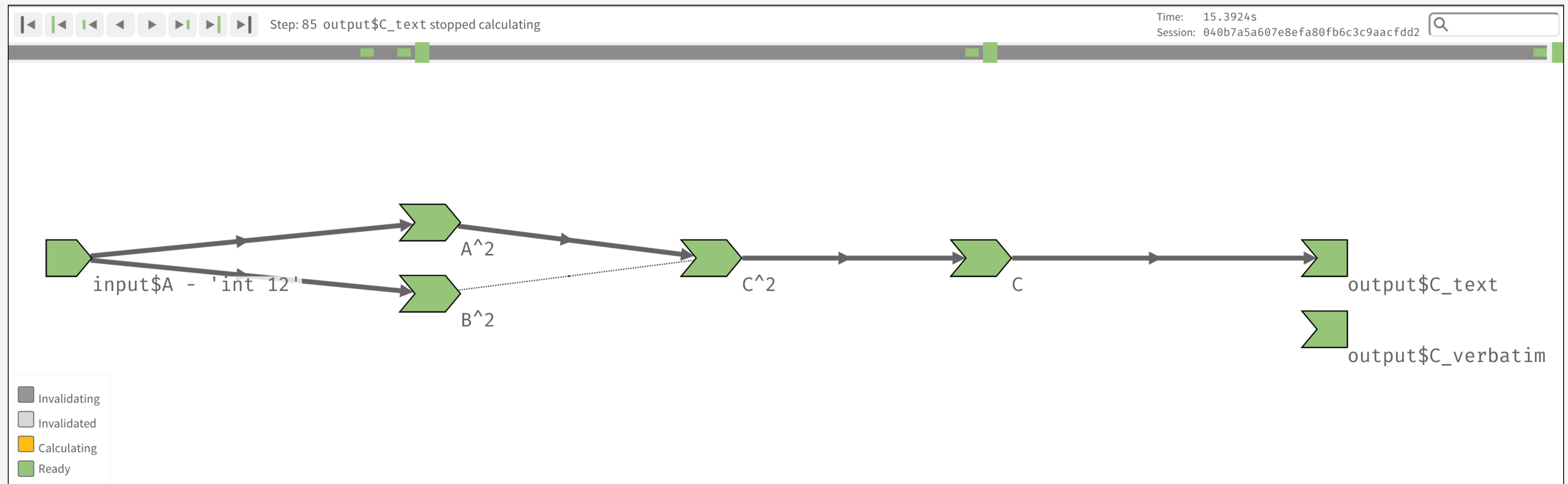
Broken Pythagoras

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



Broken Pythagoras

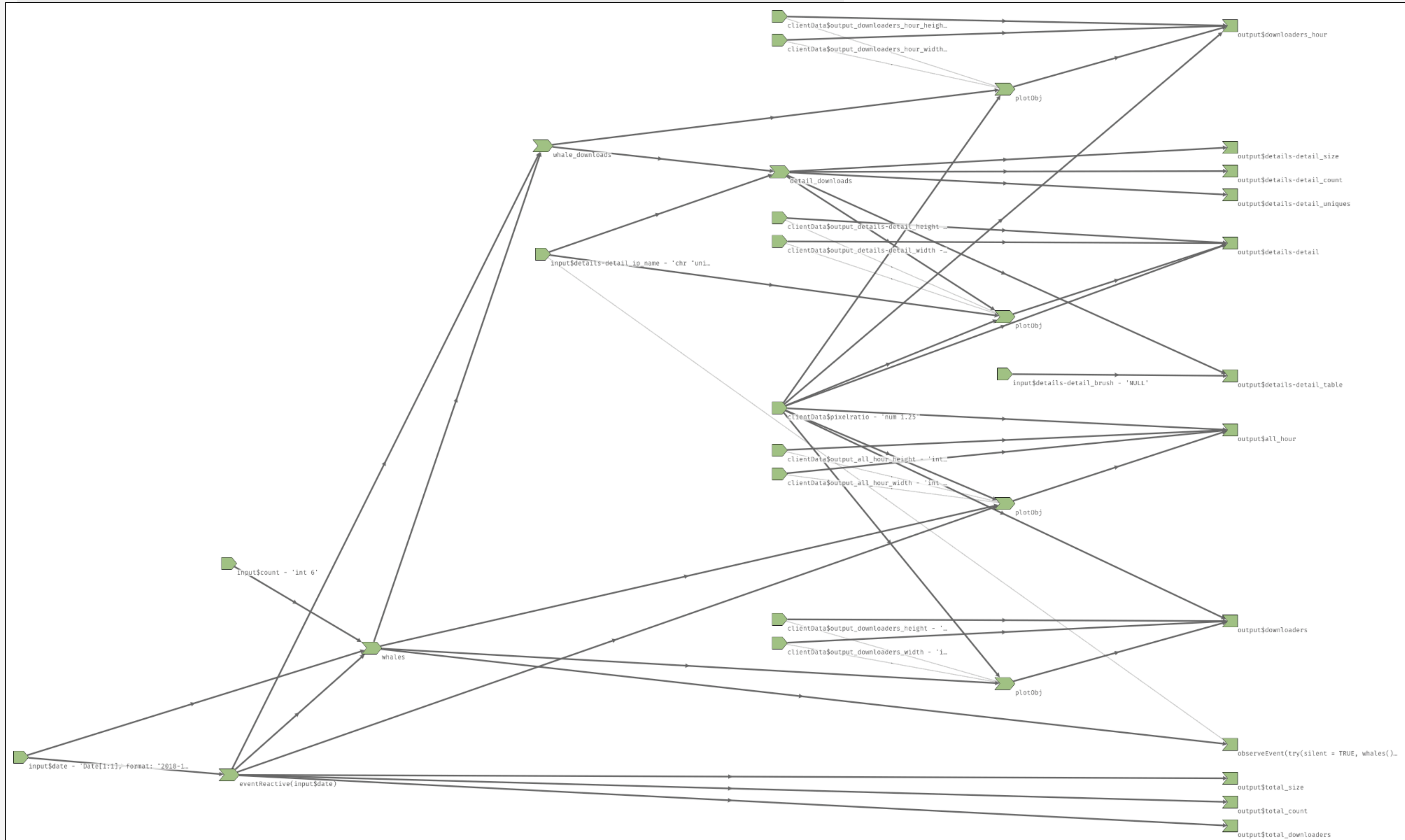
- Back to steady state (Shiny is idle)



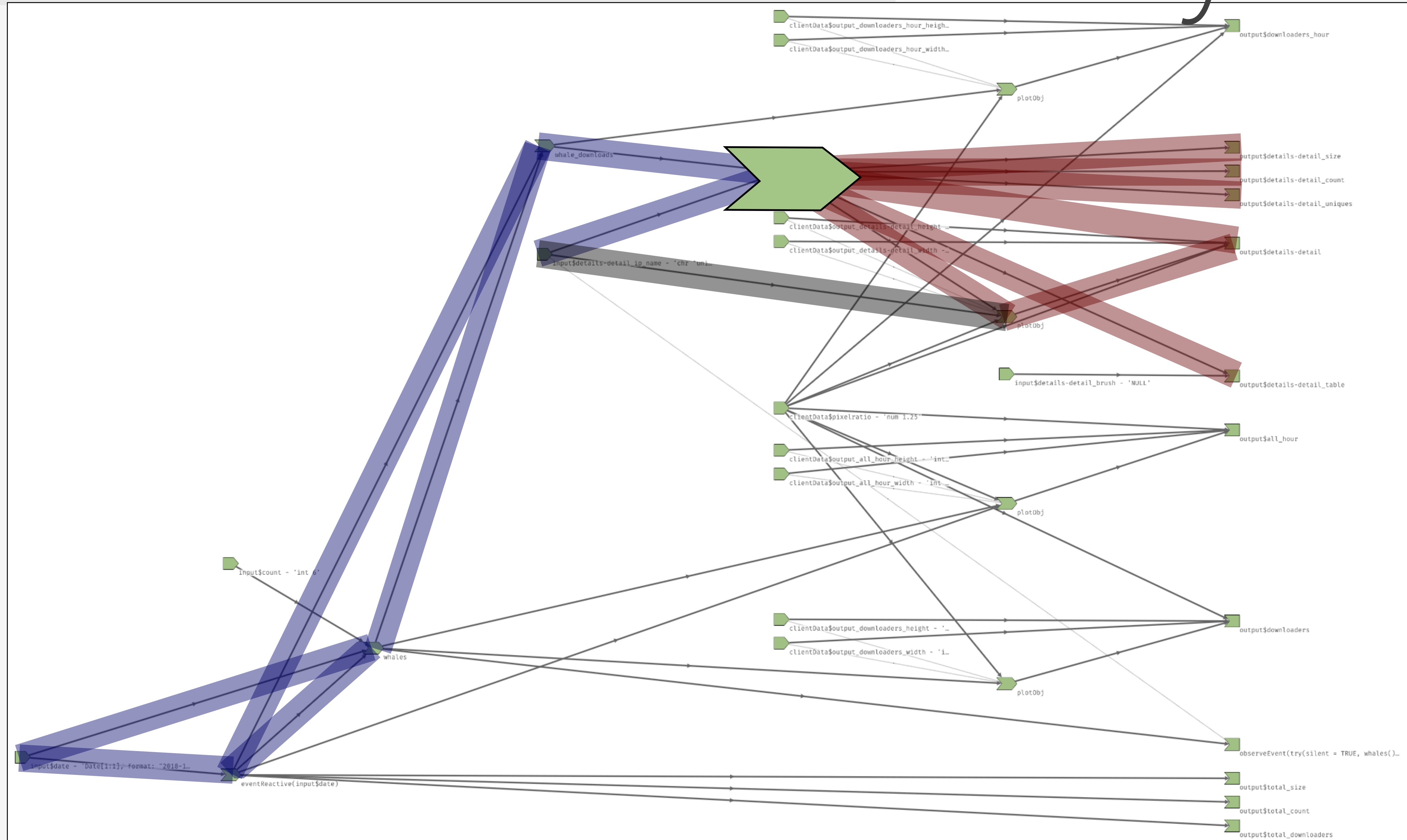
Searching within `reactlog`

- `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

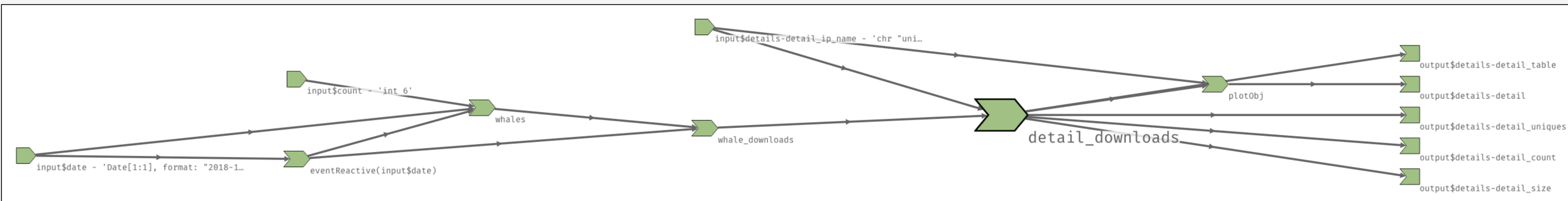
cranwhales is dense!



detail_downloads family tree



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

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 app's performance and reactive state simultaneously

Questions?

- **reactlog:** github.com/rstudio/reactlog
- `options(shiny.reactlog = TRUE); shiny::runApp()`
- **Shiny reactivity:** shiny.rstudio.com/articles/#reactivity
- **Slides:** bit.ly/rstudio-conf-2019-reactlog (GitHub)

