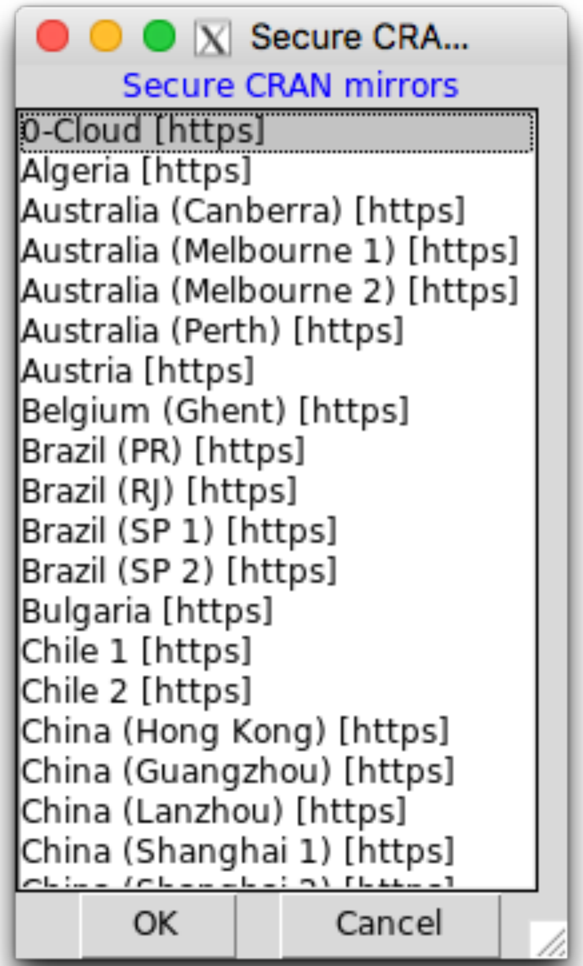
Shiny in production

Principles, practices, and tools

Joe Cheng (@jcheng)

rstudio::conf 2019

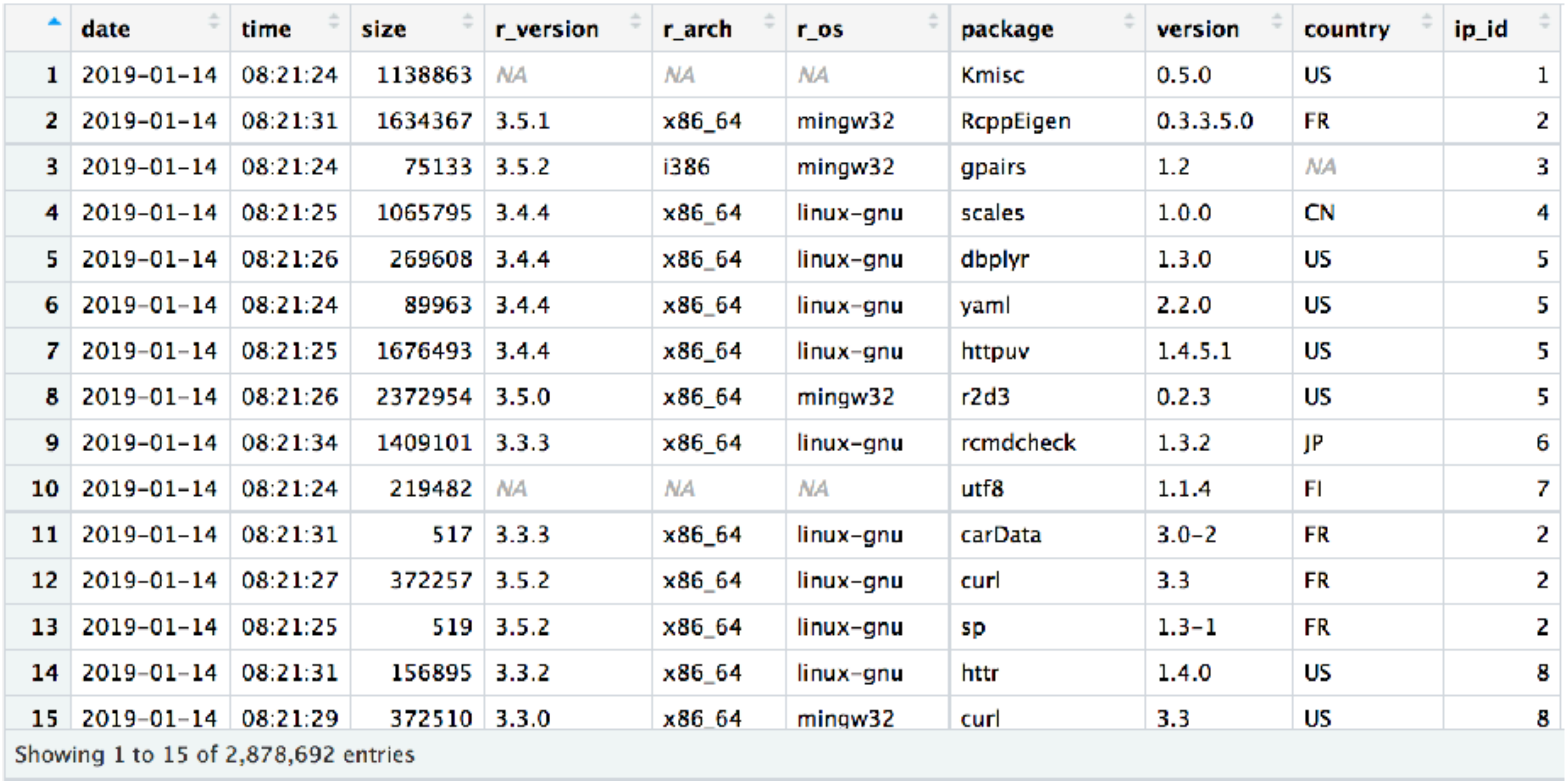


Secure CRAN mirrors

0-Cloud [https]
Algeria [https]
Australia (Canberra) [https]
Australia (Melbourne 1) [https]
Australia (Melbourne 2) [https]
Australia (Perth) [https]
Austria [https]
Belgium (Ghent) [https]
Brazil (PR) [https]
Brazil (RJ) [https]
Brazil (SP 1) [https]
Brazil (SP 2) [https]
Bulgaria [https]
Chile 1 [https]
Chile 2 [https]
China (Hong Kong) [https]
China (Guangzhou) [https]
China (Lanzhou) [https]
China (Shanghai 1) [https]

cloud.r-project.org

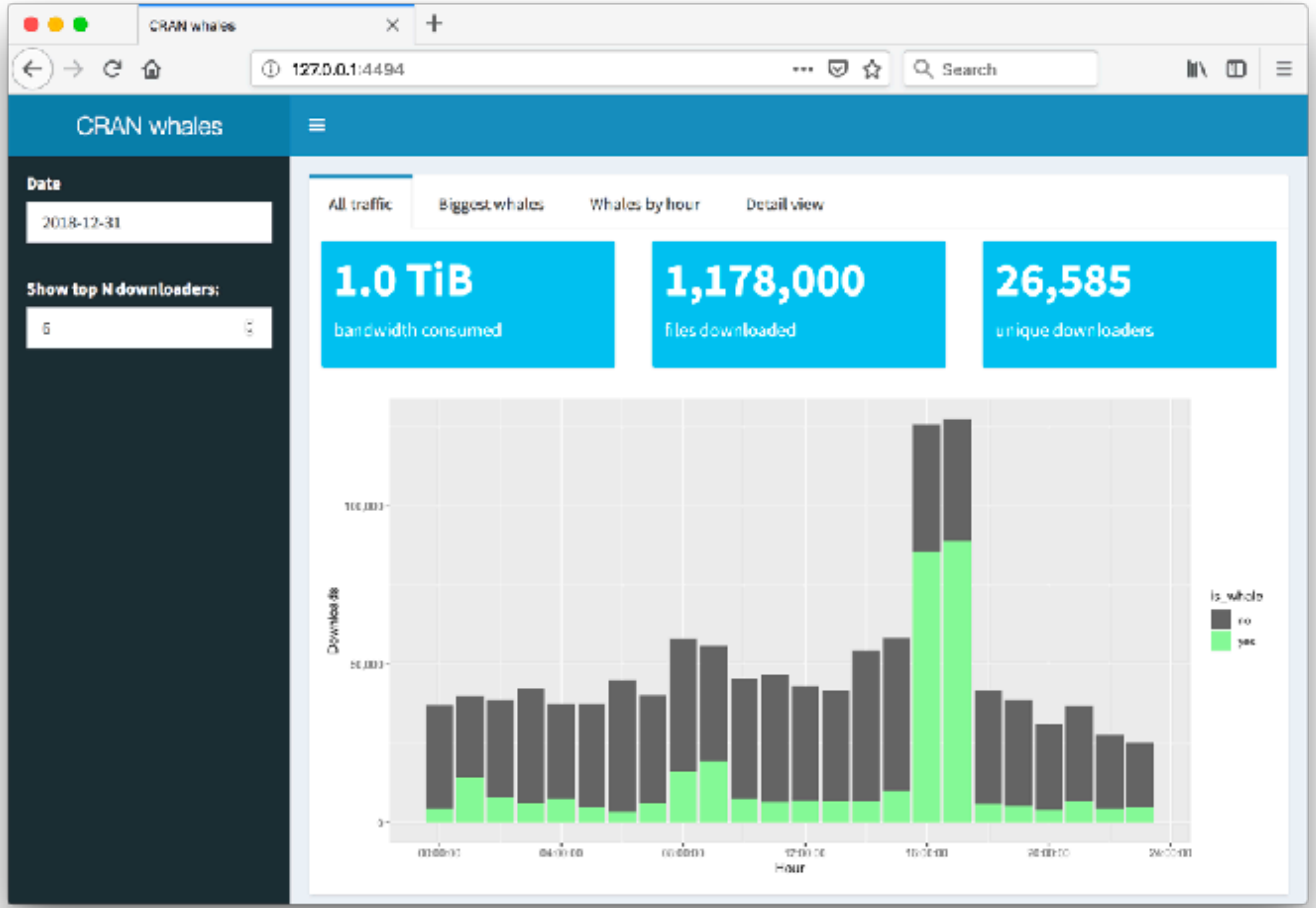
RStudio-maintained, Amazon AWS-hosted CRAN mirror



	date	time	size	r_version	r_arch	r_os	package	version	country	ip_id
1	2019-01-14	08:21:24	1138863	NA	NA	NA	Kmisc	0.5.0	US	1
2	2019-01-14	08:21:31	1634367	3.5.1	x86_64	mingw32	RcppEigen	0.3.3.5.0	FR	2
3	2019-01-14	08:21:24	75133	3.5.2	i386	mingw32	gpairs	1.2	NA	3
4	2019-01-14	08:21:25	1065795	3.4.4	x86_64	linux-gnu	scales	1.0.0	CN	4
5	2019-01-14	08:21:26	269608	3.4.4	x86_64	linux-gnu	dbplyr	1.3.0	US	5
6	2019-01-14	08:21:24	89963	3.4.4	x86_64	linux-gnu	yaml	2.2.0	US	5
7	2019-01-14	08:21:25	1676493	3.4.4	x86_64	linux-gnu	httpuv	1.4.5.1	US	5
8	2019-01-14	08:21:26	2372954	3.5.0	x86_64	mingw32	r2d3	0.2.3	US	5
9	2019-01-14	08:21:34	1409101	3.3.3	x86_64	linux-gnu	rcmdcheck	1.3.2	JP	6
10	2019-01-14	08:21:24	219482	NA	NA	NA	utf8	1.1.4	FI	7
11	2019-01-14	08:21:31	517	3.3.3	x86_64	linux-gnu	carData	3.0-2	FR	2
12	2019-01-14	08:21:27	372257	3.5.2	x86_64	linux-gnu	curl	3.3	FR	2
13	2019-01-14	08:21:25	519	3.5.2	x86_64	linux-gnu	sp	1.3-1	FR	2
14	2019-01-14	08:21:31	156895	3.3.2	x86_64	linux-gnu	httr	1.4.0	US	8
15	2019-01-14	08:21:29	372510	3.3.0	x86_64	mingw32	curl	3.3	US	8

Showing 1 to 15 of 2,878,692 entries

cran-logs.rstudio.com



cranwhales

https://github.com/rstudio/cranwhales

Why cranwhales?

Just an example, but one that feels a lot like a lot of “production” Shiny apps

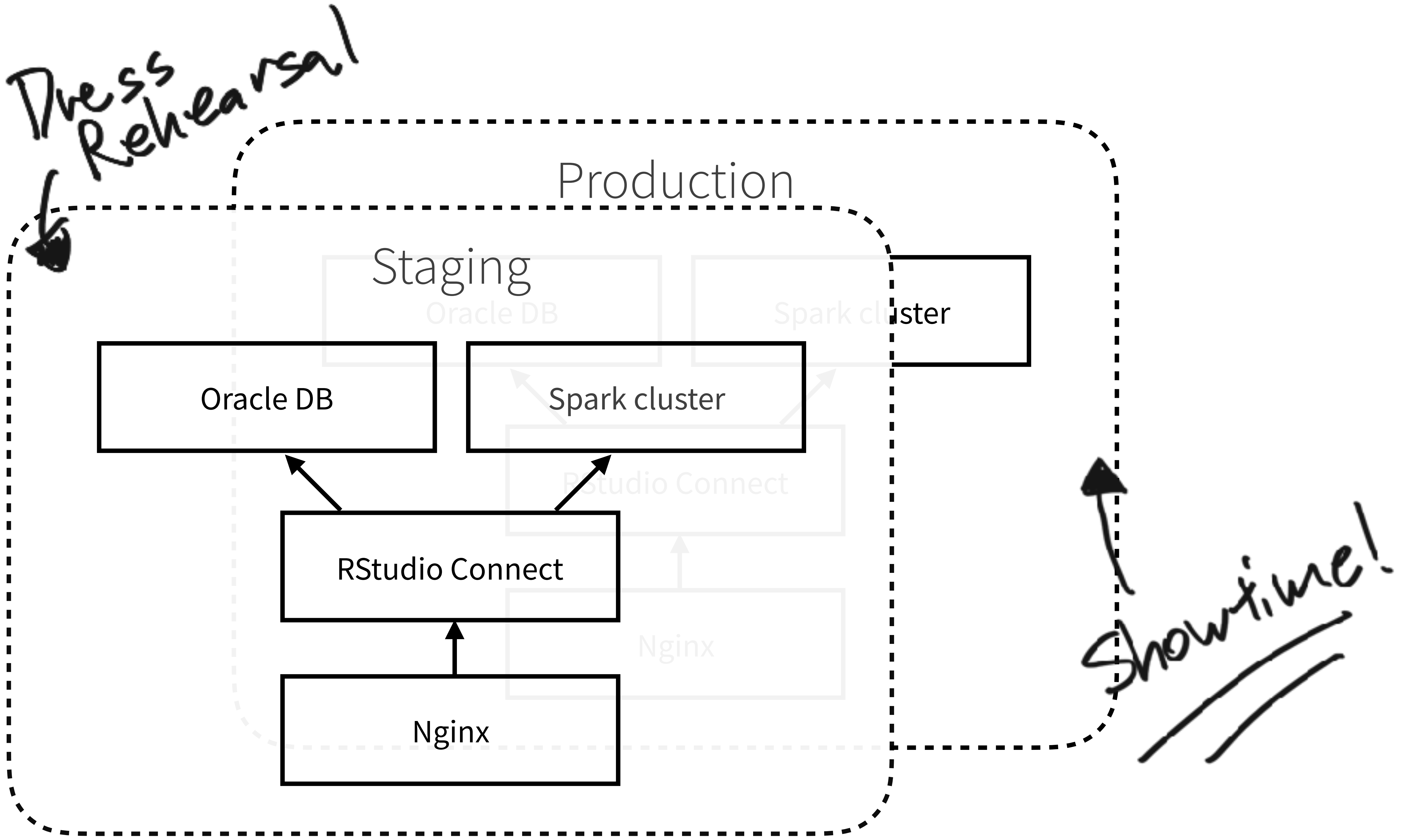
- Very natural to build in Shiny
- Data that’s updated periodically
- Significant computation is happening (loading millions of rows of data, lots of grouping and filtering)
- Mostly aggregated data being shown (with option to drill down)

What is “production”?

Software environments that are used and relied on by real users, with real consequences if things go wrong.

Contrast with:
proof of concept
prototype
sandbox
playground
testing
staging

What is "production"?



What is “production”?

Our goals are to:

- **Keep it up**
  Unplanned outages are rare or nonexistent
- **Keep it safe**
  Data, functionality, and code are all kept safe from unauthorized access
- **Keep it correct**
  Works as intended, provides right answers
- **Keep it snappy**
  Fast response times, ability to predict needed capacity for expected traffic

Can Shiny be used for production?

- 2015 answer: “Yes, it’s quite possible”
- 2019 answer: “Yes, it’s quite easy”

Challenges for Shiny in production

- Cultural
- Organizational
- Technical

Challenges for Shiny in production

Cultural challenges

- Shiny apps are developed by R users, who aren’t necessarily software engineers
    - Don’t realize they’re creating production apps
    - Don’t know best practices for writing and deploying production apps
    - Don’t anticipate/budget for the effort required for production-readiness
    - Don’t come from a culture that worries about runtime efficiency

Challenges for Shiny in production

Organizational challenges

- IT and management can be skeptical
    - The IT function naturally skews towards conservatism
    - Skeptical of data scientists creating production artifacts
    - Skeptical of technologies they haven’t heard of
- Engineering department may not be on your side
    - R is a DSL for statistics, not a Real Programming Language™ (:eyeroll:)

Challenges for Shiny in production

Technical challenges

- Shiny dramatically lowers the effort involved in creating a web app
  - …but it doesn’t lower (or even increases!) the effort involved in: automated testing, load testing, profiling, and deployment
- R can be slow, and is single-threaded

Challenges for Shiny in production

Technical challenges

- Shiny dramatically lowers the effort involved in creating a web app
    - …but it doesn't lower (or even increases!) the effort involved in: automated testing, load testing, profiling, and deployment (until now!)
- R can be slow, and is single-threaded
    - But it's probably a lot less slow than you think

New tools for Shiny in production

- **RStudio Connect** – On-premises Shiny serving with push-button deployment
  https://www.rstudio.com/products/connect/
- **shinytest** – Automated UI testing for Shiny
  https://rstudio.github.io/shinytest/
- **shinyloadtest** – Load testing for Shiny
  https://rstudio.github.io/shinyloadtest/
- **profvis** – Profiler for R (not new but still very important!)
  https://rstudio.github.io/profvis/
- **Plot caching** – Dramatically speed up repeated plots
  http://shiny.rstudio.com/articles/plot-caching.html
- **Async** – Last resort technique for dealing with slow operations
  https://rstudio.github.io/promises/

New tools for Shiny in production

- **RStudio Connect** – On-premises Shiny serving with push-button deployment
  https://www.rstudio.com/products/connect/
- **shinytest** – Automated UI testing for Shiny
  https://rstudio.github.io/shinytest/
- **shinyloadtest** – Load testing for Shiny
  https://rstudio.github.io/shinyloadtest/
- **profvis** – Profiler for R (not new but still very important!)
  https://rstudio.github.io/profvis/
- **Plot caching** – Dramatically speed up repeated plots
  http://shiny.rstudio.com/articles/plot-caching.html
- **Async** – Last resort technique for dealing with slow operations
  https://rstudio.github.io/promises/

Is cranwhales fast enough?

- Estimated peak traffic: 100 concurrent users
- Hardware: Dedicated server with 16-core CPU
- It would be nice to support at least 10 concurrent users per R process, preferably 20—and more is obviously better

Performance workflow

1. Use **shinyloadtest** to see if it's fast enough
2. If not, use **profvis** to see what's making it slow
3. Optimize
    1. Move work out of Shiny (very often)
    2. Make code faster (very often)
    3. Use caching (sometimes)
    4. Use async (occasionally)
4. Repeat!

shinyloadtest

Use **shinyloadtest** to generate large amounts of realistic traffic to your app, then analyze latency.

1. **Run or deploy** your app
2. **Record** an archetypal user session using shinyloadtest
3. **Playback** the recording with your desired level of concurrency using shinycannon
4. **Analyze** the results using shinyloadtest's reporting feature (or perform your own analysis using R)

Record session archetype

In one R session:

```
shiny::runApp(port = 6104)
```

In a second R session:

```
shinyloadtest::record_session("http://127.0.0.1:6104")
```

(Or point record_session to a deployed app on RStudio Connect or Shiny Server Pro)

Playback sessions

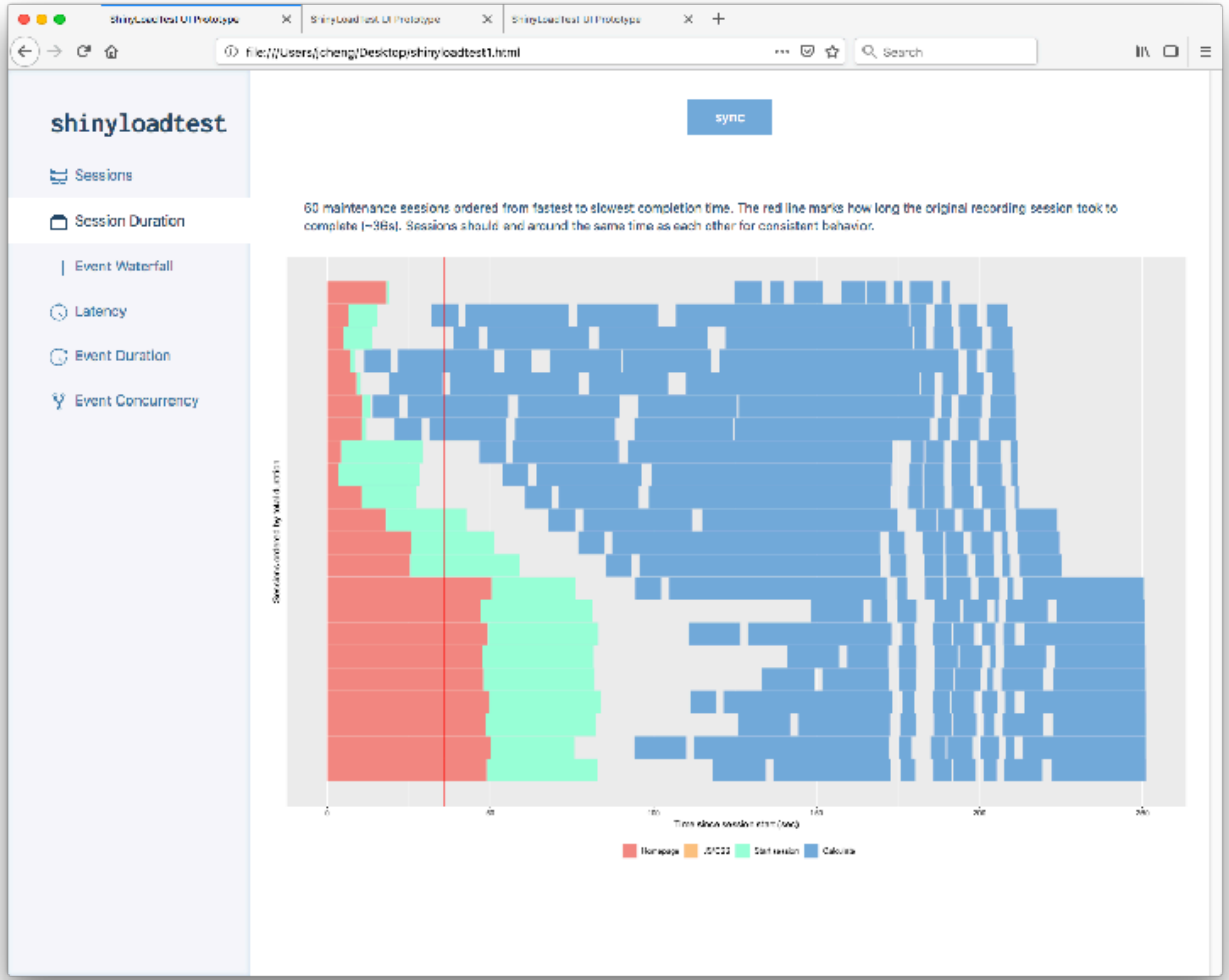
```
$ shinycannon recording.log http://127.0.0.1:6104 --workers=20
    --loaded-duration-minutes=5
```

```
2019-01-06 09:41:29.976 INFO [thread00] - Waiting for warmup to complete
2019-01-06 09:41:29.975 INFO [thread01] - Warming up
2019-01-06 09:41:29.975 INFO [progress] - Running: 0, Failed: 0, Done: 0
2019-01-06 09:41:30.889 INFO [thread02] - Warming up
2019-01-06 09:41:31.808 INFO [thread03] - Warming up
2019-01-06 09:41:32.722 INFO [thread04] - Warming up
2019-01-06 09:41:33.636 INFO [thread05] - Warming up
2019-01-06 09:41:34.551 INFO [thread06] - Warming up
2019-01-06 09:41:34.983 INFO [progress] - Running: 6, Failed: 0, Done: 0
2019-01-06 09:41:35.464 INFO [thread07] - Warming up
2019-01-06 09:41:36.383 INFO [thread08] - Warming up
...
```

Analyze results

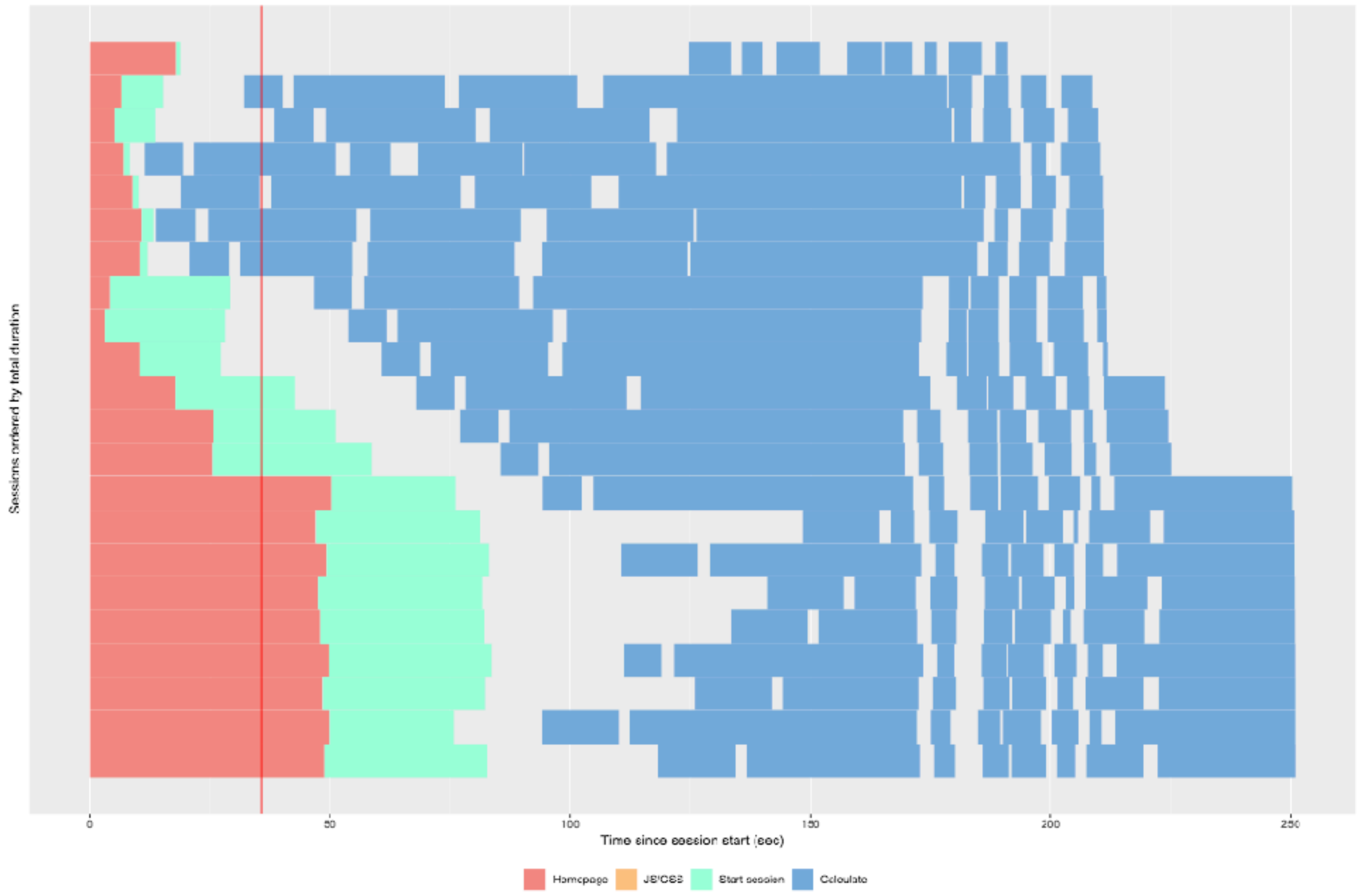
```
library(shinyloadtest)

df <- load_runs("path-to-shinycannon-data")
shinyloadtest_report(df)
```

cranwhales results (n = 20)



60 maintenance sessions ordered from fastest to slowest completion time. The red line marks how long the original recording session took to complete (~36s). Sessions should end around the same time as each other for consistent behavior.

60 maintenance sessions ordered from fastest to slowest completion time. The red line marks how long the original recording session took to complete (~36s). Sessions should end around the same time as each other for consistent behavior.



What does shinyloadtest tell us?

cranwhales is FAR too slow to support 20 users per R process.

- Just throw more R processes at it
- Just throw more servers at it
- Nah, let’s just make it fast!

profvis

Use **profvis** to see what your code is actually spending its time on. *Don’t guess!*

http://rpubs.com/jcheng/cranwhales-sync

- 6.3 seconds in `read_csv`
- `output$all_hour` spends 620ms filtering/aggregating, 280ms plotting

Most of this work should be done before the Shiny app even launches!

Doing work ahead of time

If performance matters, avoid loading raw data into Shiny. Instead, preprocess the data into a form that Shiny can quickly load.

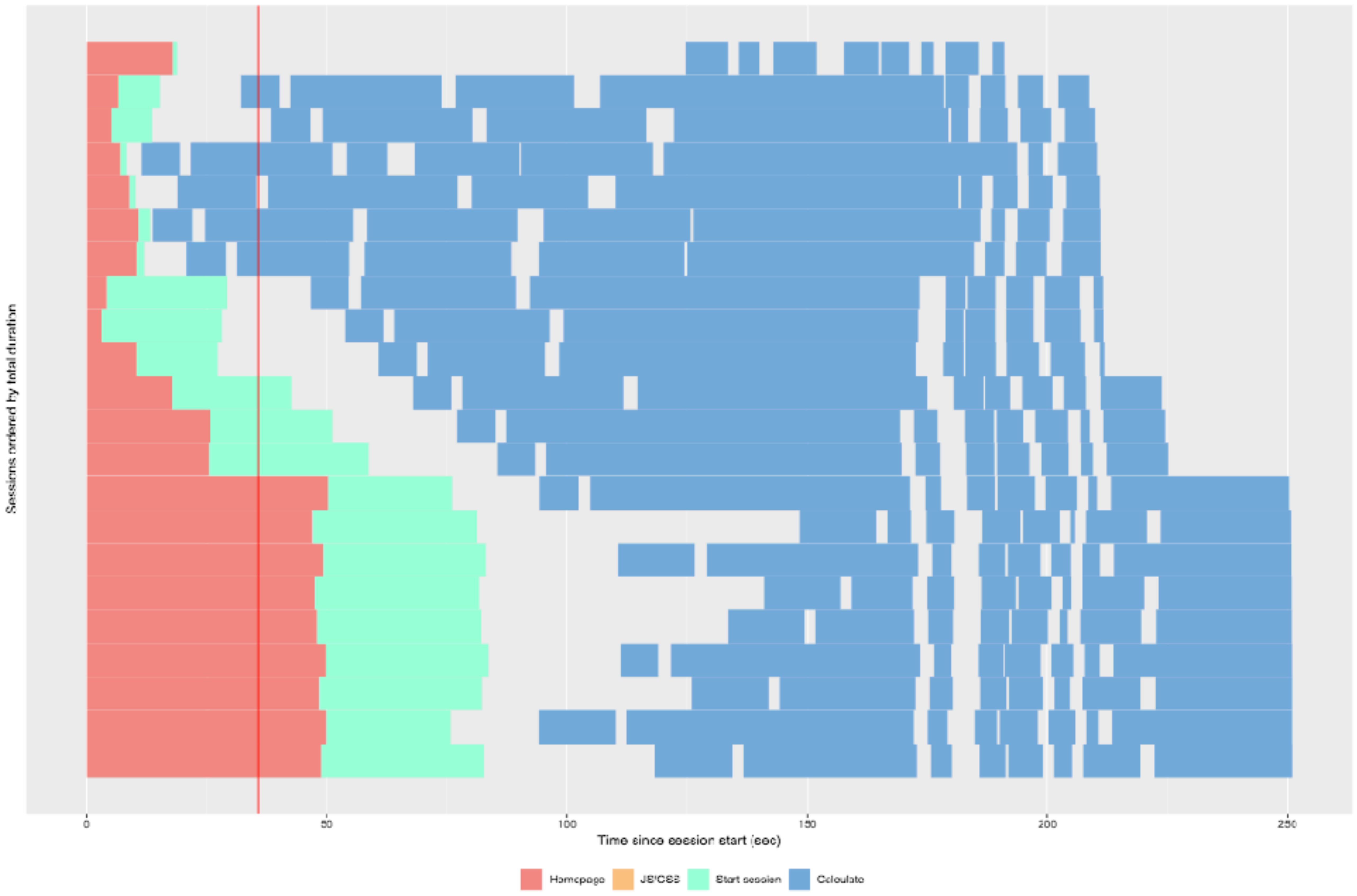
- Perform as much filtering and summarizing as you can
- Save data frames as feather files for (much) faster reading
- If your data source changes over time, schedule a separate job to preprocess the data (RStudio Connect + scheduled R Markdown is a great solution; or if you don’t have Connect, use the Unix utility “cron”)

The cranwhales `etl` branch

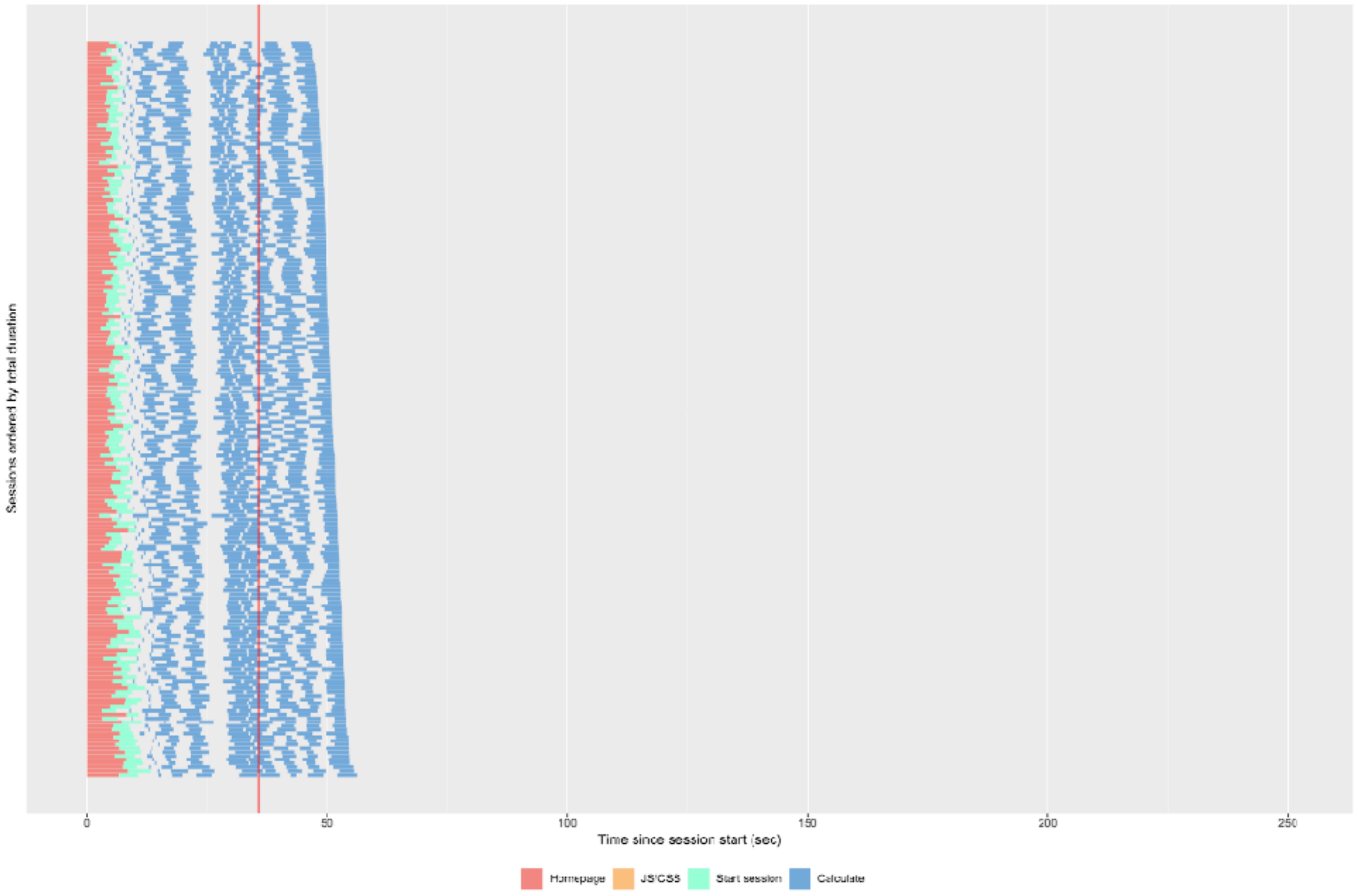
Moves log file data processing into separate script (transform.R), to be run daily

https://github.com/rstudio/cranwhales/tree/etl

60 maintenance sessions ordered from fastest to slowest completion time. The red line marks how long the original recording session took to complete (~36s). Sessions should end around the same time as each other for consistent behavior.



237 maintenance sessions ordered from fastest to slowest completion time. The red line marks how long the original recording session took to complete (~36s). Sessions should end around the same time as each other for consistent behavior.



How did we do?

The `etl` version is dramatically faster!

Median session duration has dropped from 210 seconds to about 50 seconds

Maximum wait time was unspeakable before, now <10 seconds

Can we do better?

Profvis, round 2: http://rpubs.com/jcheng/cranwhales-etl

- Data loading/manipulation time is mostly gone
- Almost all remaining time is being spent in ggplot2

Plot caching

Requires Shiny v1.2 (most recent CRAN release)

A Shiny app is a good candidate for plot caching if:

1. The app has plot outputs that are time-consuming to generate (check—several hundred milliseconds each)
2. These plots are a significant fraction of the total amount of time the app spends thinking (check—not much else left at this point)
3. Most users are likely to request the same few plots (check—probably most people are looking at the last few days)

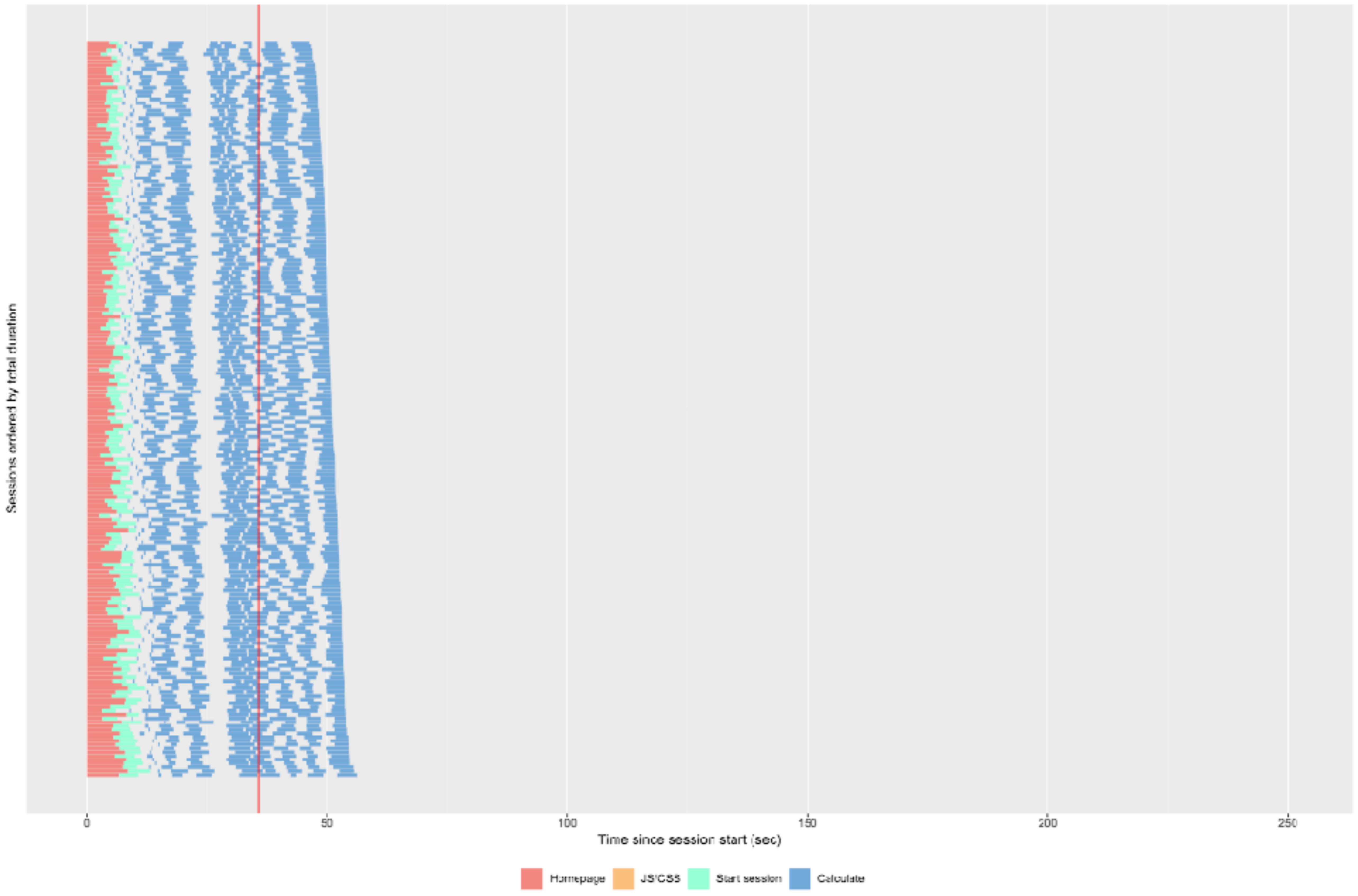
Regular plot

```
output$plot <- renderPlot({
  ggplot(diamonds, aes(
      carat, price, color = !!input$color_by)) +
    geom_point()
})
```

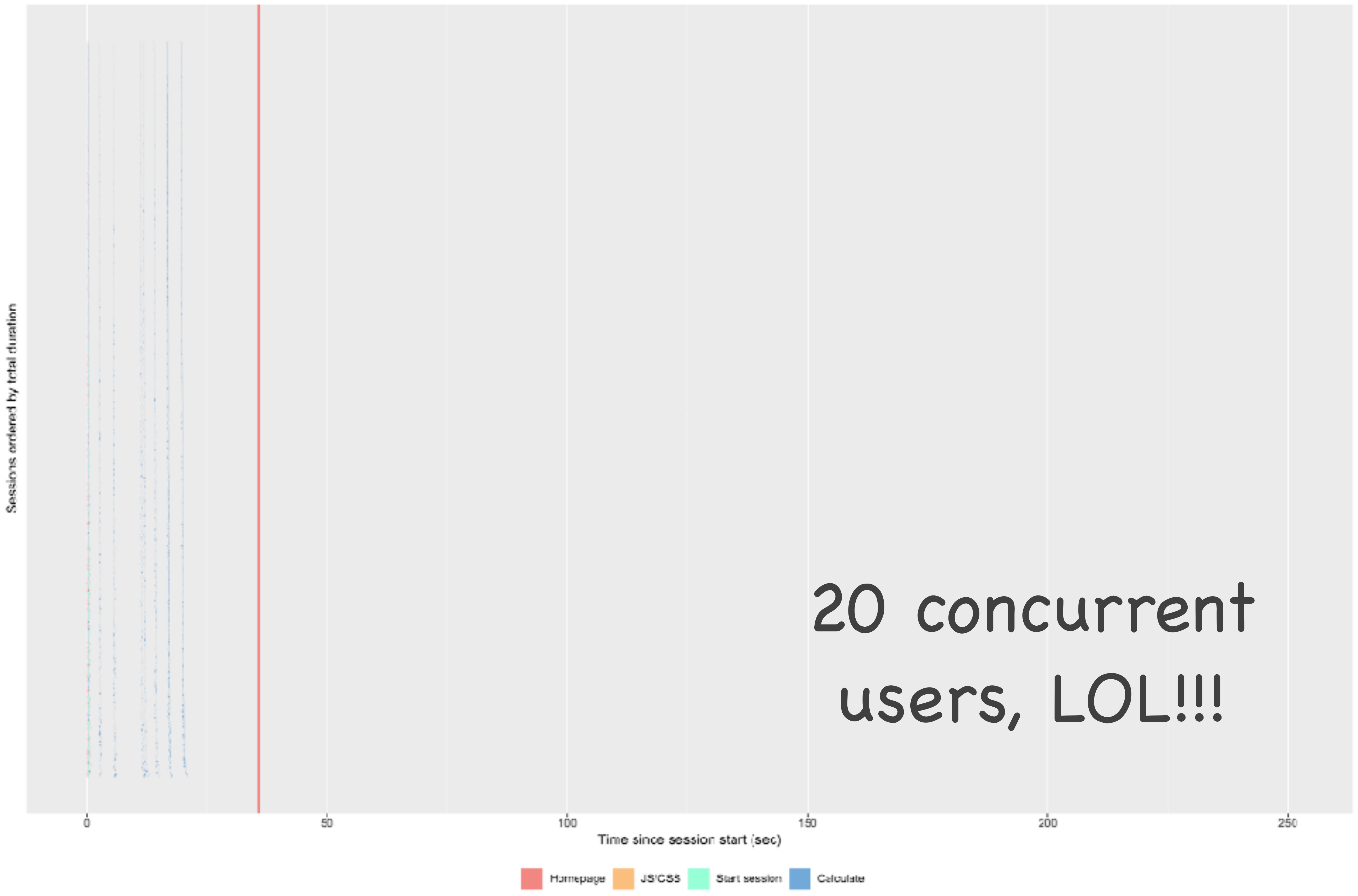
Cached plot

```
output$plot <- renderCachedPlot({
  ggplot(diamonds, aes(
      carat, price, color = !!input$color_by)) +
    geom_point()
}, cacheKeyExpr = { input$color_by })
```

237 maintenance sessions ordered from fastest to slowest completion time. The red line marks how long the original recording session took to complete (~36s). Sessions should end around the same time as each other for consistent behavior.



526 maintenance sessions ordered from fastest to slowest completion time. The red line marks how long the original recording session took to complete (~36s). Sessions should end around the same time as each other for consistent behavior.



20 concurrent users, LOL!!!

New tools for Shiny in production

- **RStudio Connect** – On-premises Shiny serving with push-button deployment
  https://www.rstudio.com/products/connect/
- **shinytest** – Automated UI testing for Shiny
  https://rstudio.github.io/shinytest/
- **shinyloadtest** – Load testing for Shiny
  https://rstudio.github.io/shinyloadtest/
- **profvis** – Profiler for R (not new but still very important!)
  https://rstudio.github.io/profvis/
- **Plot caching** – Dramatically speed up repeated plots
  http://shiny.rstudio.com/articles/plot-caching.html
- **Async** – Last resort technique for dealing with slow operations
  https://rstudio.github.io/promises/

Summary

- With these tools, we’ve made it much easier to run Shiny apps in production
- But the biggest challenges are cultural and organizational
- Deploying production apps successfully requires skill and experience—lean on the expertise of IT/Engineering resources that are available to you

Credits

shinyloadtest
Alan Dipert, Barbara Borges Ribeiro, Tony Santos,
Barret Schloerke, Shalu Tiwari

profvis
Winston Chang, Javier Luraschi

Plot caching
Winston Chang

Thank you!

This slide deck (includes appendix):
https://speakerdeck.com/jcheng5/shiny-in-production

Shiny in Production book by Kelly O'Briant (work in progress):
http://bit.ly/shiny-prod-book

Appendix: Best practices

Keep it up

Goal: Reliability, high availability

- ⭐ Be extremely careful when making even the smallest code or configuration changes ⭐
    - Create a staging environment that closely mirrors production, and test ALL changes there before touching production
    - Many a system has been crashed by a “trivial” code change or minor upgrade of a dependency—be skeptical!
- Eliminate single points of failure
    - Run clustered Connect/Shiny Server instances (load balancing and/or failover)

Keep it safe

Goal: Security, privacy

- ⭐ Use secure HTTP (https) servers (supported directly by Connect and Shiny Server Pro, or use an https proxy) ⭐
- Apply the *Principle of Least Privilege*—almost nobody should be able to log directly into production servers and databases
- Think carefully about how you secure your credentials (e.g. the password to your production database)

Keep it correct

Goal: Accuracy, quality, robustness

- ⭐ Separate your non-Shiny R code from your Shiny R code ⭐
    - Ideally your Shiny code is just the UI- and reactivity-related "glue" between your well-unit-tested analysis functions
    - Non-Shiny R code is easier to unit test and debug
- Use **packrat** to "pin" the versions of your R package dependencies (not needed for RStudio Connect or ShinyApps.io)
- Use **shinytest** (new!) to do high-level testing of your app
- Use **reactlog 2.0** (new!) to debug reactivity issues
- Complement your automated testing with manual testing—do manual testing before each deploy to production

Keep it snappy

Goal: Responsiveness, throughput, scalability

- ⭐ Use **profvis** to determine what is making your app slow—your intuition sucks! ⭐
- ⭐ Pre-process your data if at all possible—it's far better to load pre-summarized/aggregated data than to make the user wait while you summarize/aggregate ⭐
- Plotting can be relatively expensive—use Shiny's built-in feature for **plot caching** (new!)
- If you have optimized as much as possible and still have slow tasks, consider using **async operations** (new!) if scalability is a concern

Keep it snappy

- Use **shinyloadtest** (new!) to see how your app will perform under load
- Spread load across multiple R processes with RStudio Connect, Shiny Server Pro, or ShinyApps.io
- Run multiple servers if necessary