ANALYZING WEB PERFORMANCE DATA WITH JULIA & D3

Fluent 2016

2016-03-08

Philip Tellis
@bluesmoon

http://www.soasta.com/mpulse/

https://github.com/lognormal/boomerang

SOASTA

GET SETUP

1. Visit bit.ly/fluent-julia-d3 and either make a copy or just download the files.

2. Sign in to juliabox.org (This might fail once or twice)

3. In the Sync tab, enter either your copy of the Google Drive folder listed above, or the bit.ly link.

4. This should add a Fluent folder in your JuliaBox.

5. Also available at bit.ly/gh-julia-d3

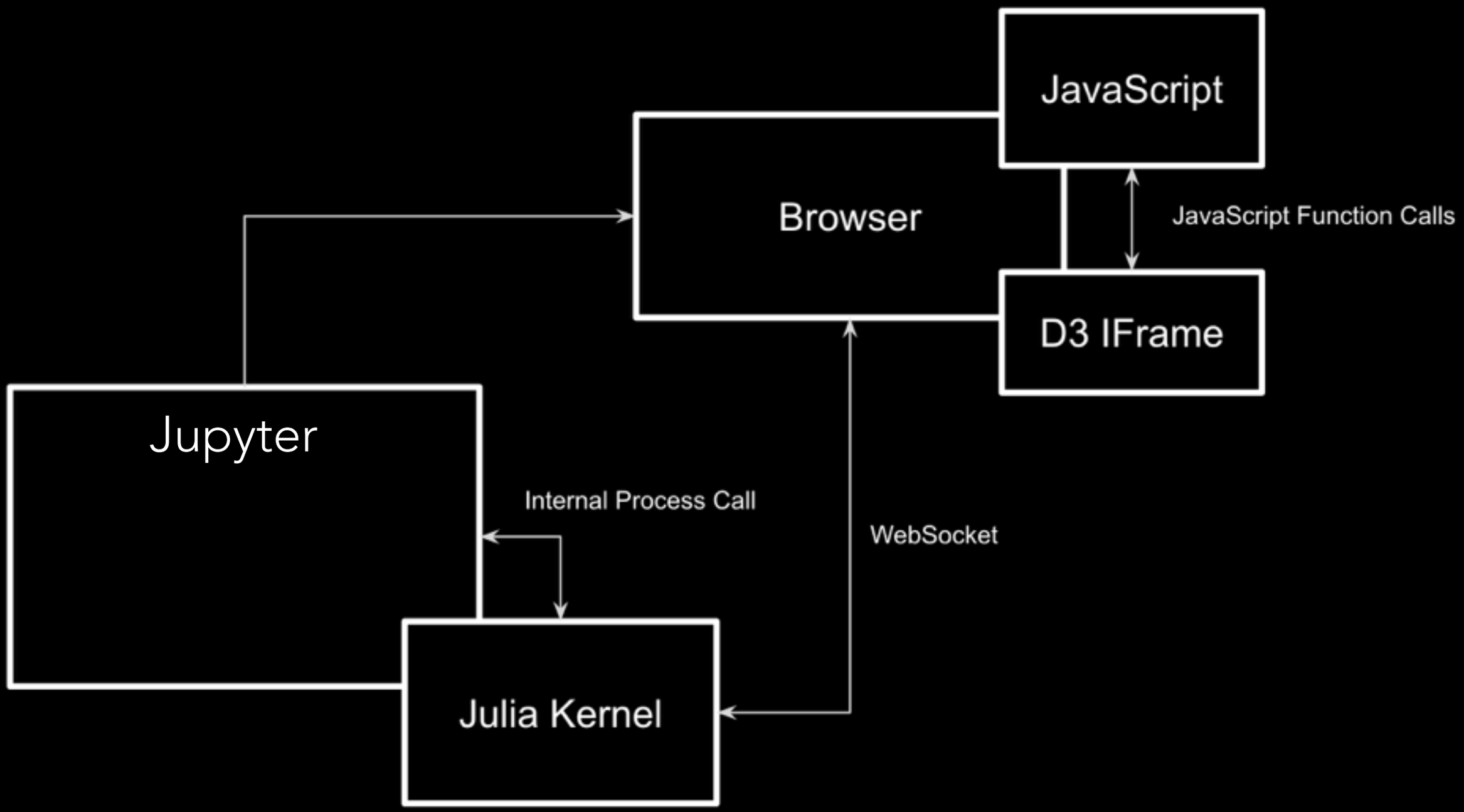
JULIA IS...

- High-level, high-performance dynamic programming language
- Borrows from R, Python, MatLab, etc.
- Performance comparable to C
- Designed for Parallelism & Cloud Computing
- Operates well on Vectors and uses SIMD where possible
- MIT licensed — http://julialang.org/

D3 IS...

- A JavaScript library that maps Data to DOM Nodes
- Extended via layouts & plugins for rich data visualizations
- You still need to write code to draw things
- Fast on its own, but you can easily make it sluggish
- BSD Licensed — http://d3js.org/

JUPYTER + D3


JavaScript
Browser
JavaScript Function Calls
D3 IFrame
Jupyter
Internal Process Call
WebSocket
Julia Kernel

GET STARTED WITH JUPYTER

HTTPS://WWW.JULIABOX.ORG/

BASIC JULIA TUTORIAL

- JSON & JavaScript
- Matrix Operations
- Stats & DataFrames
- JavaScript to update a DOM Node
- Notebooks 01-04 at http://bit.ly/fluent-julia-d3

GET STARTED WITH D3

HTTPS://GITHUB.COM/MBOSTOCK/D3/WIKI/GALLERY

BASIC D3 TUTORIAL

- Adding nodes
- Mapping data to nodes
- Data Driven Documents
- Examples 01-06 at https://soasta.github.io/julia-d3-tutorial/d3/

IT WOULD BE COOL IF JULIA

COULD CALL OUT TO D3

THIS IS A SIMPLE EXTENSION OF WHAT WE ALREADY KNOW

- Create an IFRAME instead of a P
- Assign to its `src` instead of `innerText`

- Example notebook *05 - Include an IFrame*
- Example JavaScript *06-data-driven-bars*

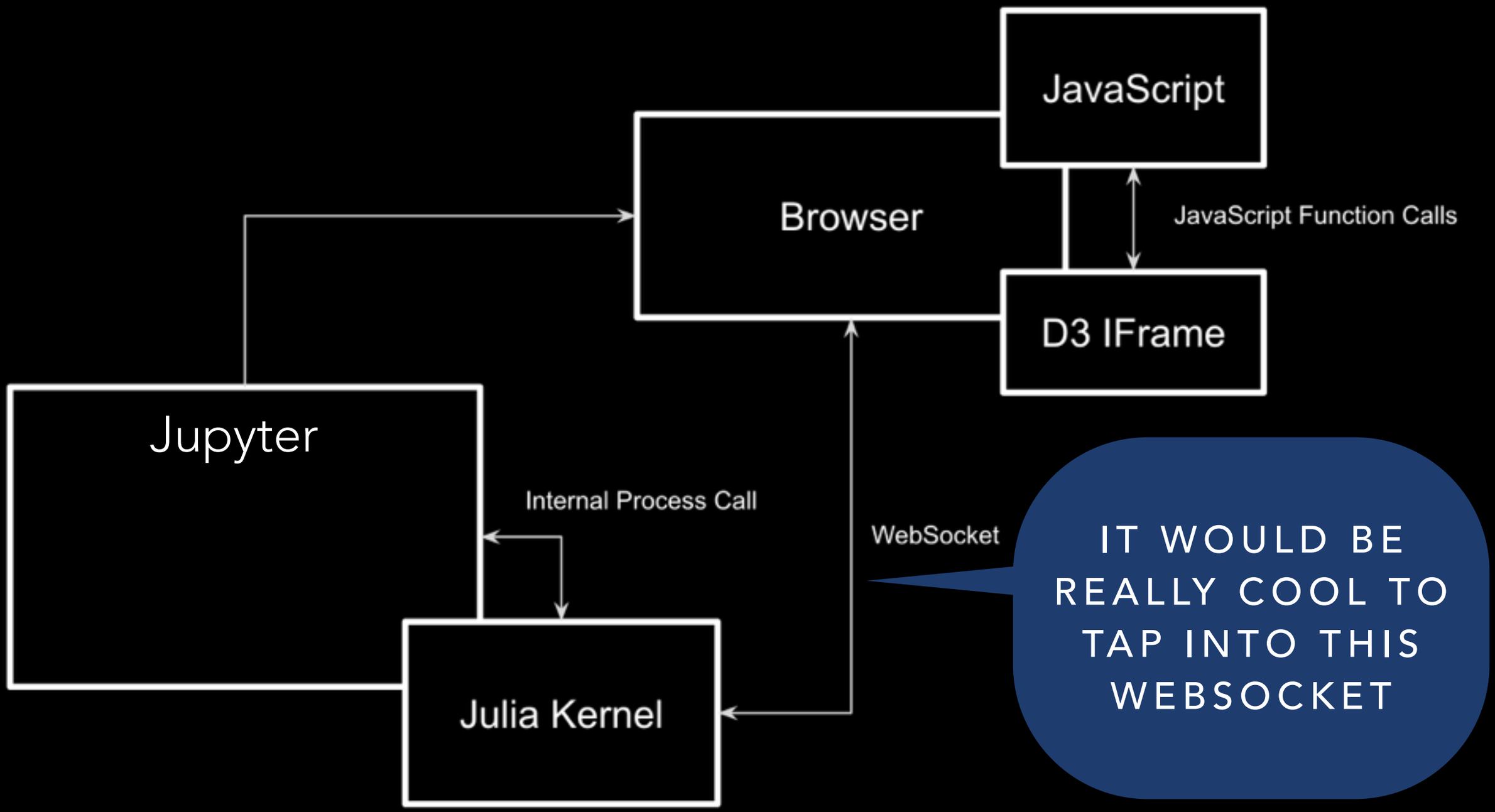
BUT REALLY, JULIA NEEDS TO PASS
DATA TO D3 FOR THIS TO BE USEFUL

WE CAN USE `window.postMessage`

- JS in IFrame looks for data in query string or listens for "message" event
- Julia code sets query string or sends message using `postMessage`
- Example notebook *06 - Pass Data to IFrame*
- Example JavaScript *07-d3-external-data*

AND ONE MORE THING...

THE WEBSOCKET



INTERACT.JL

- Interact.jl taps into the WebSocket to let you build dynamic widgets
- You can interact with your D3 charts and have those events fed back to the Julia Kernel
- https://github.com/JuliaLang/Interact.jl
- It's also available in the tutorial section of your JuliaBox

JUPYTER'S REST API

- Jupyter/IPython also have a REST API
- Any JavaScript widget can use XHR to fetch Notebook contents and kernel details, and then set up the WebSocket
- Then use the WebSocket API to execute Julia code and get results back to JavaScript

THE IPYTHON API IS IDENTICAL BUT HAS BETTER DOCS

HTTP://BIT.LY/IPYTHON-API

TRY A FEW CALLS IN THE BROWSER

- https://juliabox.org/api/sessions
- https://juliabox.org/api/kernels
- https://juliabox.org/api/contents/<notebook-path>
- wss://juliabox.org/api/kernels/<kernel-id>/channels?session_id=<session_id>

TO INTERACT WITH THE SERVER

- Create a new Kernel & Session
- Connect to WebSocket
- Execute Julia Code
- Pass results back to D3
- Use Chrome Web Inspector to study web socket protocol

GET A KERNEL/SESSION FOR A NOTEBOOK

```
POST https://juliabox.org/api/sessions HTTP/1.1
Cookie: <auth-cookies>
Content-type: application/json

{
   "kernel": {"name": "julia-0.4"},
   "notebook": {"path": "<notebook-path>"}
}
```

```
{
 "kernel":
 {
  "id": "688f9d0a-3291-4342-829d-55faf9d81a49",
  "name": "julia-0.4"
 },
 "notebook": {"path": “<notebook-path>"},
 "id": "9f5d7b39-8c32-43e7-b912-1e5b14953524"
}
```

CONNECT TO WEBSOCKET

```
new WebSocket("wss://juliabox.org/api/kernels/" +
        encodeURIComponent(session.kernel.id) +
        "/channels?session_id=" +
        encodeURIComponent(session.id)
    );
```

THE MESSAGE FORMAT

```
{
   "header": {
      "msg_id":msgId,
      "username":"username",
      "session":sessionId,
      "msg_type":"execute_request",
      "version":"5.0"
   },
   "metadata":{},
   "content":{
      "code":code,
      "silent":false,
      "store_history":false,
      "user_expressions":{},
      "allow_stdin":false
   },
   "buffers":[],
   "parent_header":{},
   "channel":"shell"
}
```

PARSE THE RESPONSE

1. JSON.parse(event.data) in sock.onmessage(event)
   This parses the Jupyter payload, not the Julia one

2. Inspect data.msg_type
   This could be "execute_result", "error", "execute_reply", "stream", among others

3. execute_result is the output of display() calls in Julia. JSON.parse this twice.

4. execute_reply and error are the status of Julia calls, inspect data.content and data.content.status

5. stream is the output of print statements or non-suppressed default output, inspect data.content.text

UNFORTUNATELY THERE'S A BUG ON JULIABOX

- Access-Control-Allow-Origin header responds with a *
- But this violates CORS for XHR that has `withCredentials=true`
- https://github.com/JuliaLang/JuliaBox/issues/367

SUMMARY

- Write Julia code to analyze and summarize data
- Convert data to a `Dict()` if necessary, and JSON encode it
- Load your D3 visualization in an iframe
- Use `postMessage` to pass data as JSON between the Julia frame and the D3 frame
- Or use the REST API to execute code via the WebSocket

Thank You