Panel: The dashboard that grew (a bit too much)

A saga about pain, scale, and bad decisions.

PyData Global – December 5th 2024

duarteocarmo.com

PyData Global

/du-art/ - originally from Portugal ☀️

ML/Software/Data/Cloud – or whatever you call it!

Based in Copenhagen, Denmark ☔

Independent contractor – I like difficult problems!

Focus: Data, LLMs, Cloud, Geospatial, Web

Today's talk is about a web app.

One that went a bit too far

This is a *meta* ~~talk~~

Duarte – this sounds more like a rant.

What's this dashboard?

Python on the web (really?)

A primer on Panel

Bad decisions and lessons learned

Final thoughts and conclusion

What's this dashboard?

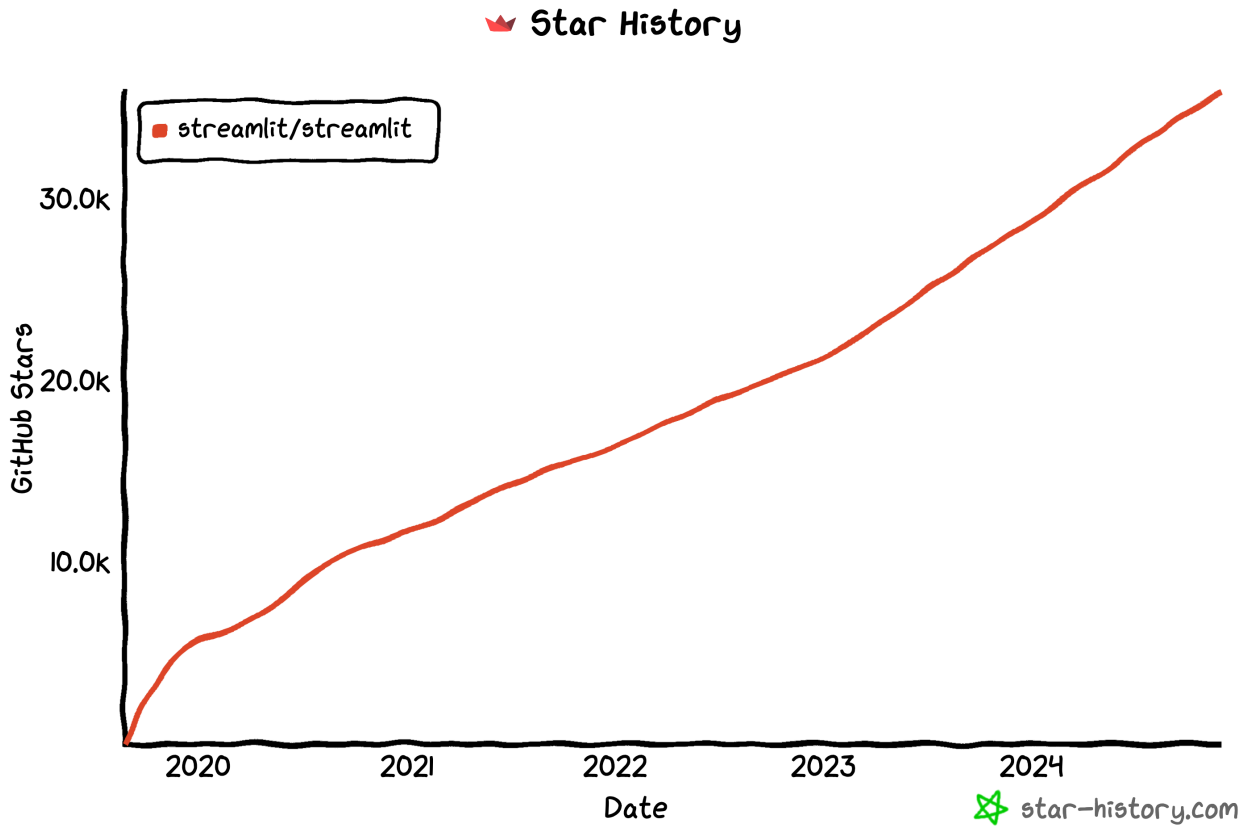
Python on the web (really?)

A primer on Panel

Bad decisions and lessons learned

Final thoughts and conclusion

The “dashboard”.



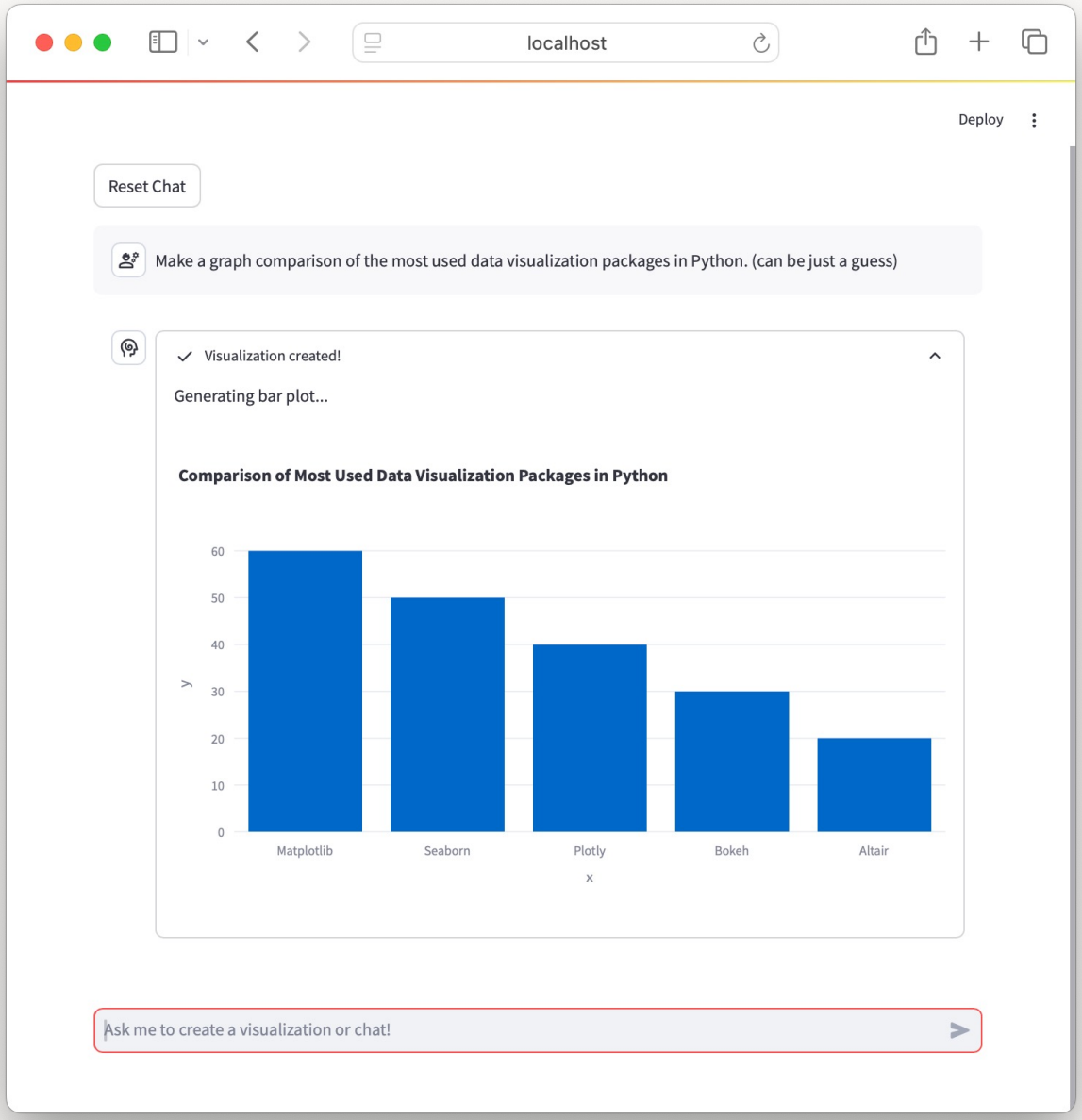
MyApp.py

```python
import streamlit as st
import pandas as pd

st.write("""
# My first app
Hello *world!*
""")

df = pd.read_csv("my_data.csv")
st.line_chart(df)
```

My App • Streamlit

My first app

Hello *world!*



https://tinyurl.com/streamlittools

Streamlit is great!

But Streamlit is not perfect.

But I love you Streamlit thank you for sponsoring ❤️

This just runs from top to bottom?

State and customization

Users? Authentication?

Testing? Web server? Deployment? Scale?

It's great – but it doesn't solve all problems.


It grew. Badly.

What's this dashboard?

Python on the web (really?)

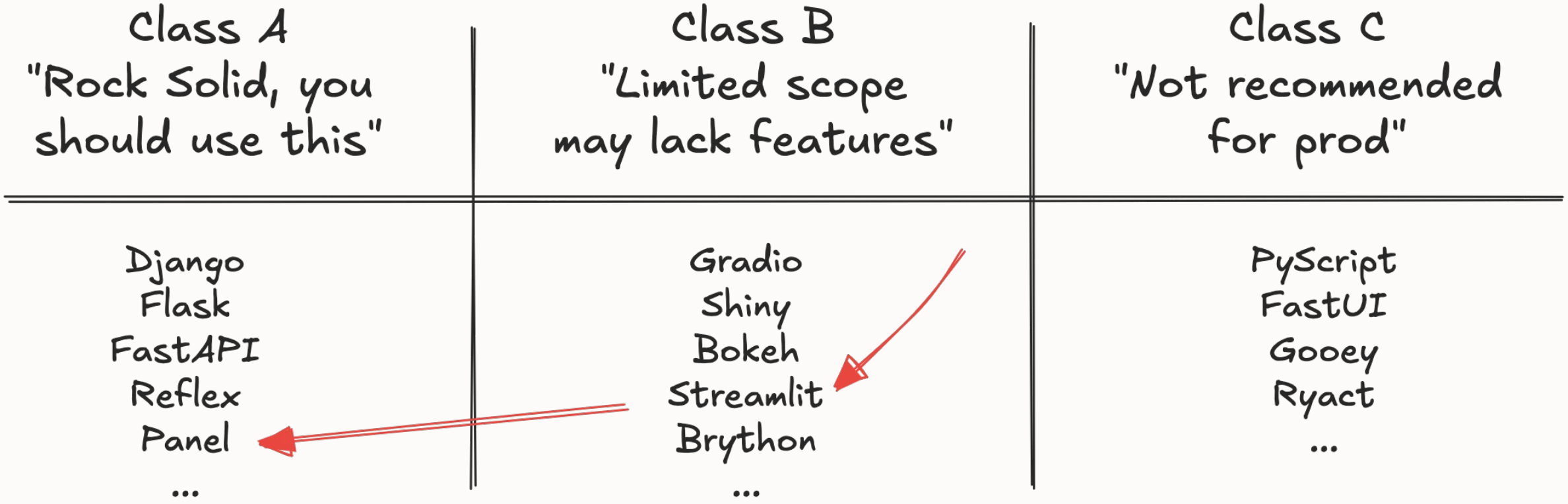
A primer on Panel

Bad decisions and lessons learned

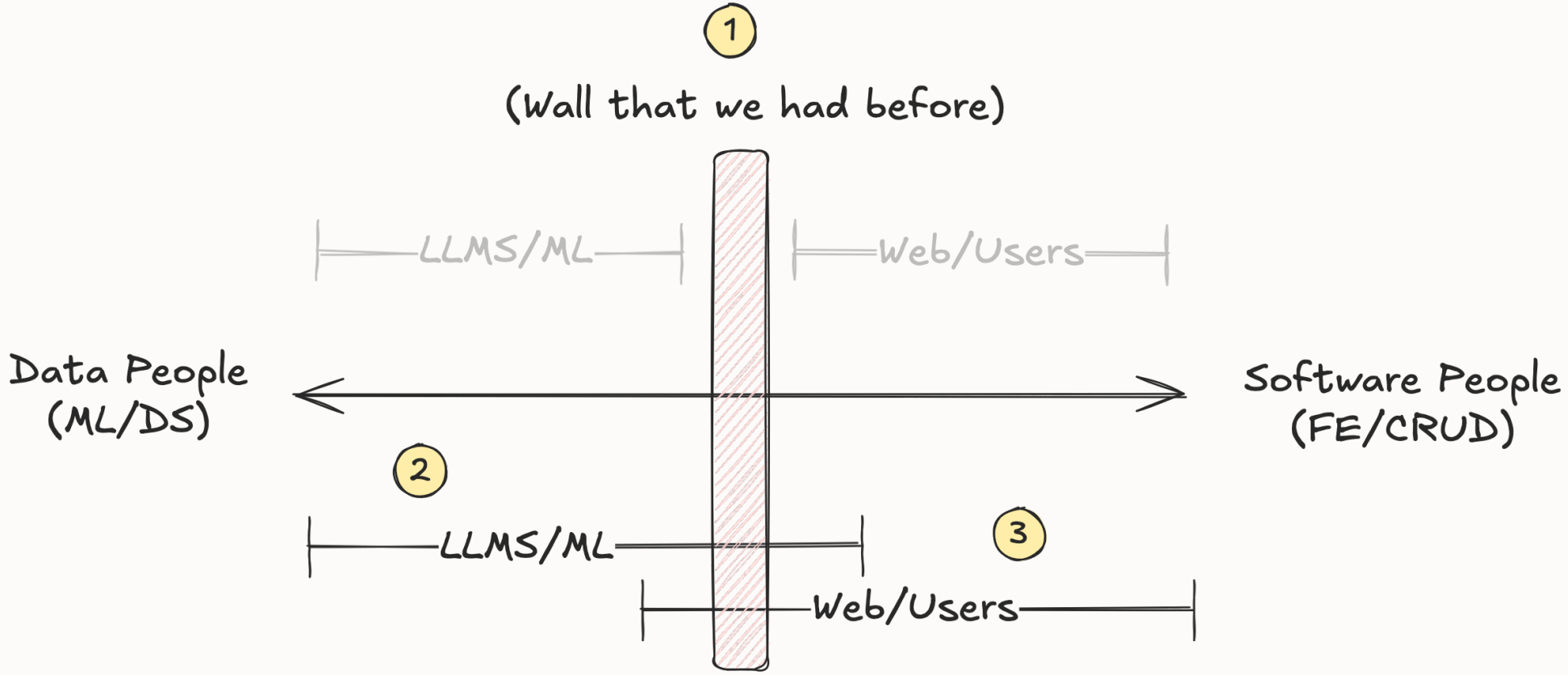
Final thoughts and conclusion

We’re not JS, but we have options

“Pure Python” web development



Source: https://metaperl.github.io/pure-python-web-development



There's a push for us putting things
in front of the users, faster.

(Un) Fortunately, life's not that easy.

What's this dashboard?

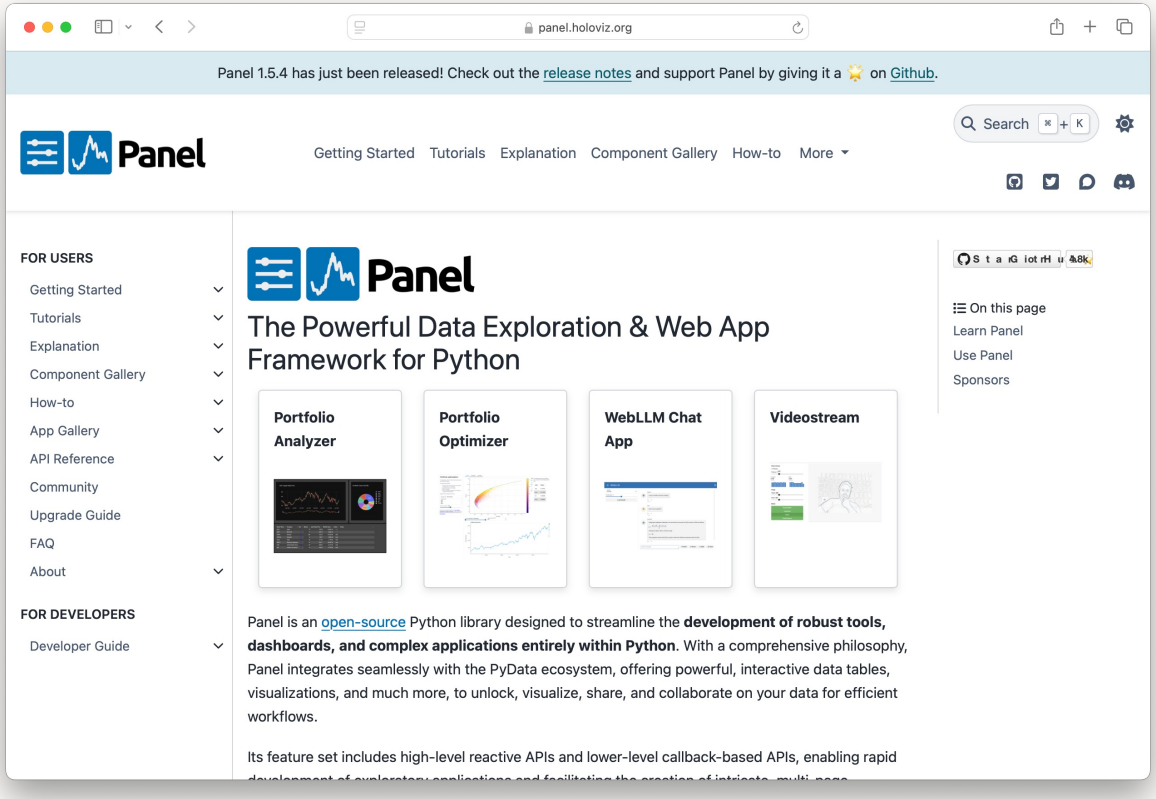
Python on the web (really?)

A primer on Panel

Bad decisions and lessons learned

Final thoughts and conclusion

Great, so what's Panel?



"Full stack" Python dashboarding

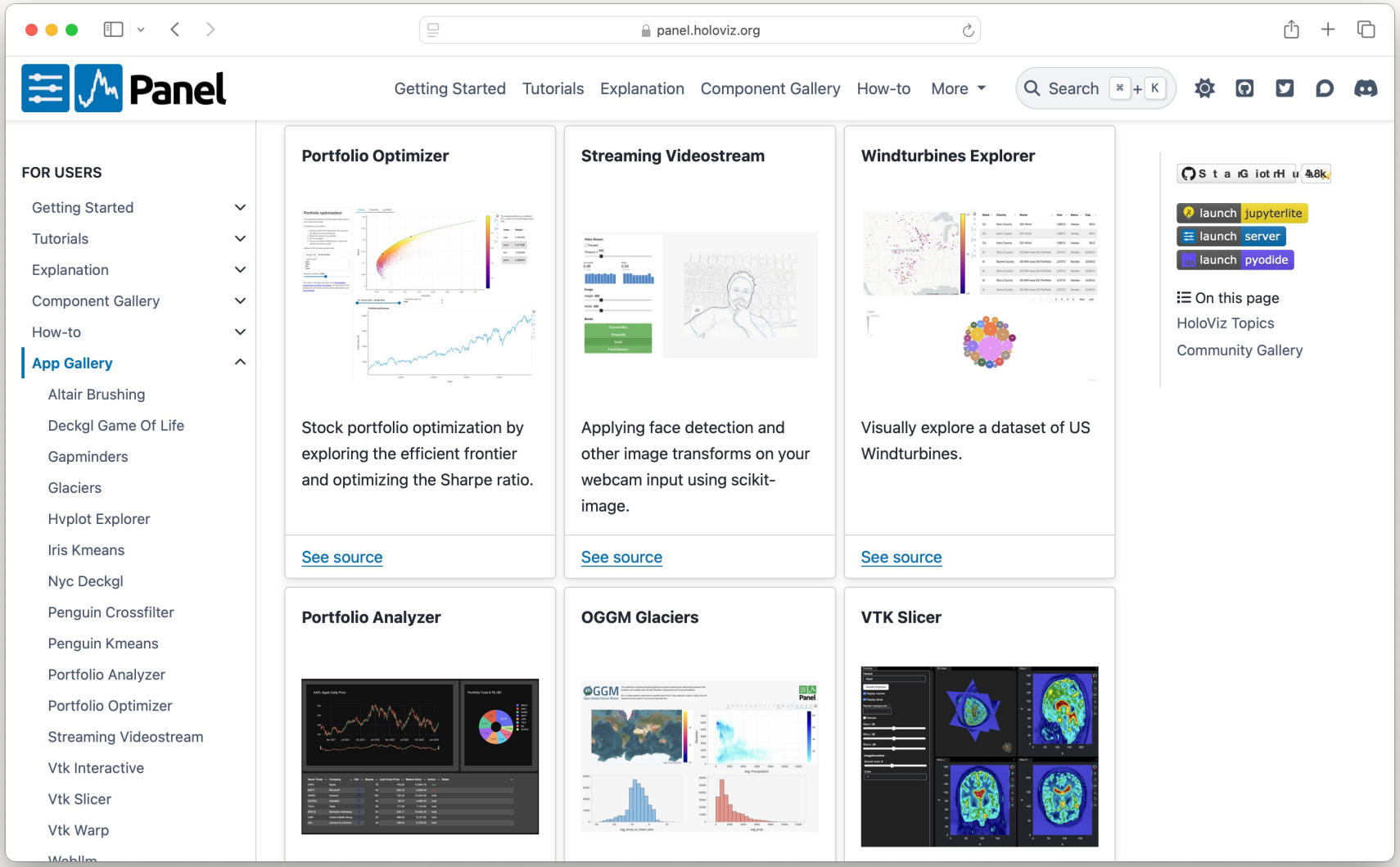
Thousands of widgets and examples

Moves very fast

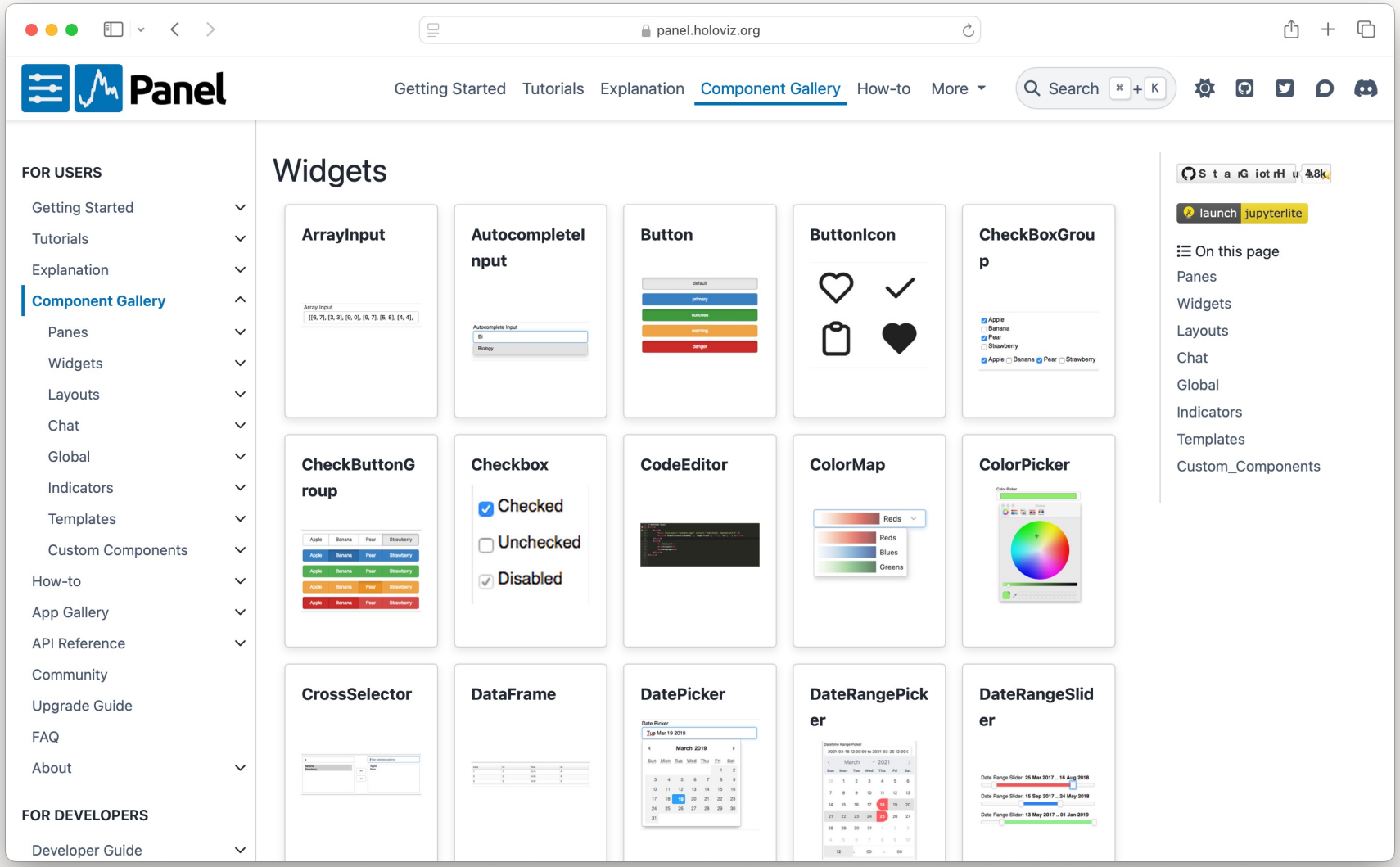
A deep and rich feature set

No fear of JavaScript

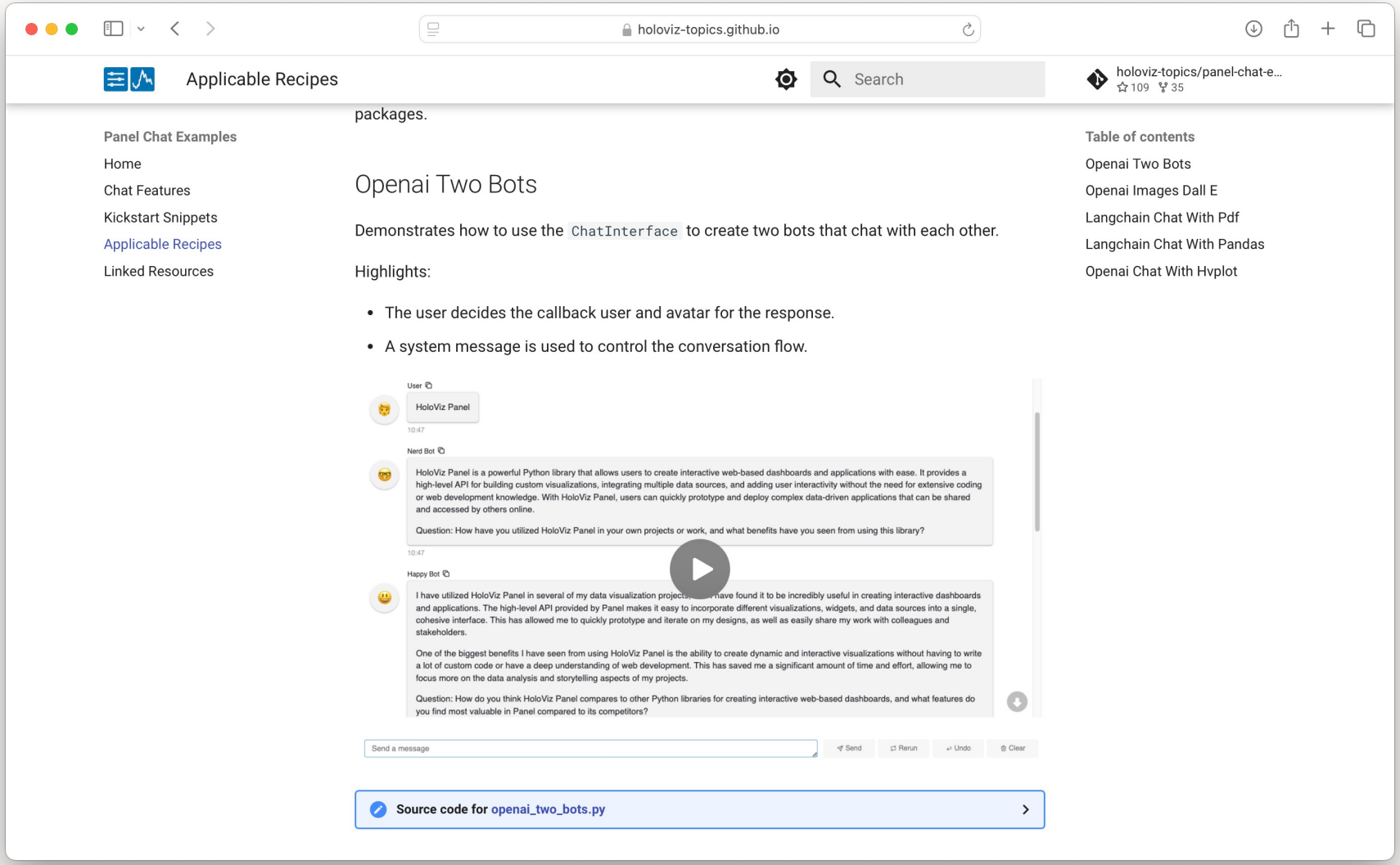
Great (and deep!) documentation



Hundreds of examples



Hundreds of widgets (and custom React)



Examples for particular use cases

Philipp Rudiger

Marc Skov Madsen

Simon Høxbro Hansen

Thank you.

This is not a critic of their great work.

What's this dashboard?

Python on the web (really?)

A primer on Panel

Bad decisions and lessons learned

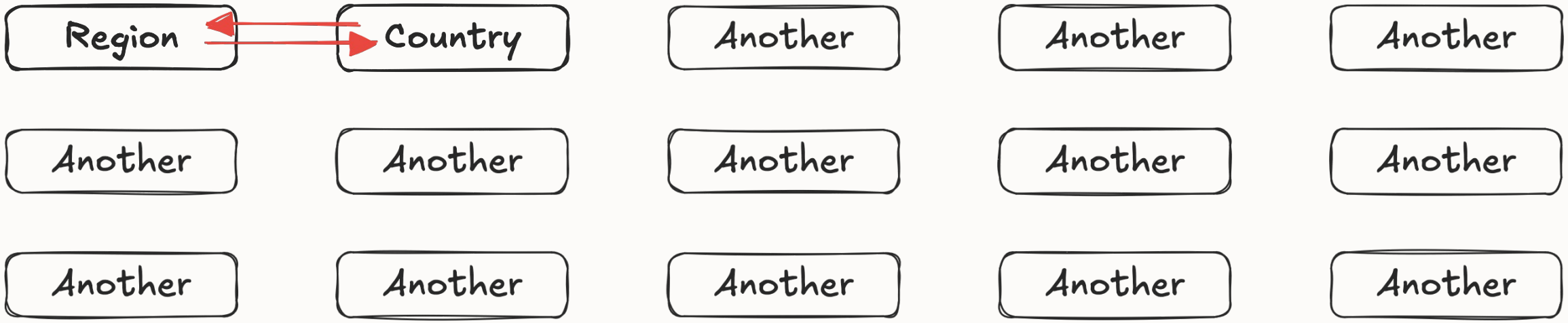
Final thoughts and conclusion

The filter saga.

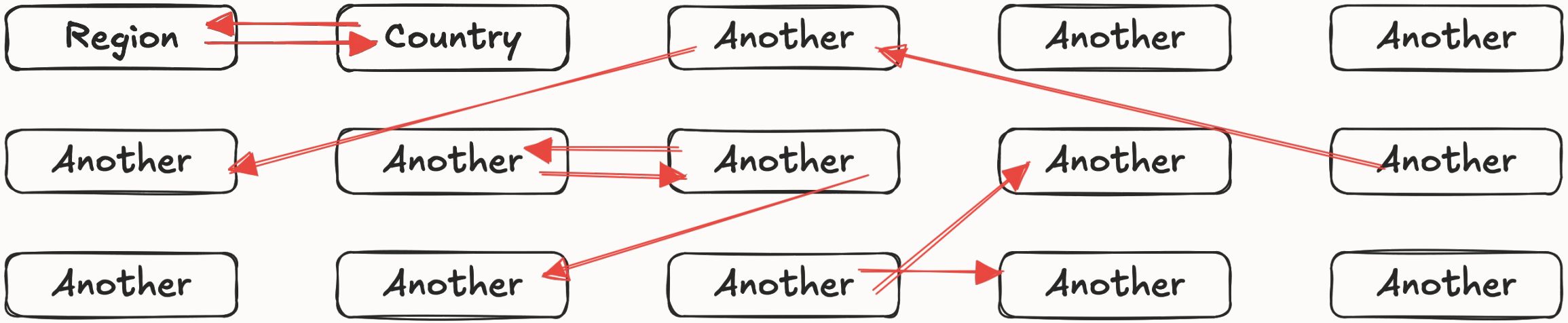
The filter saga.

The filter saga.

The filter saga.



The filter saga.



The filter saga.

Do we really need cascading filters?

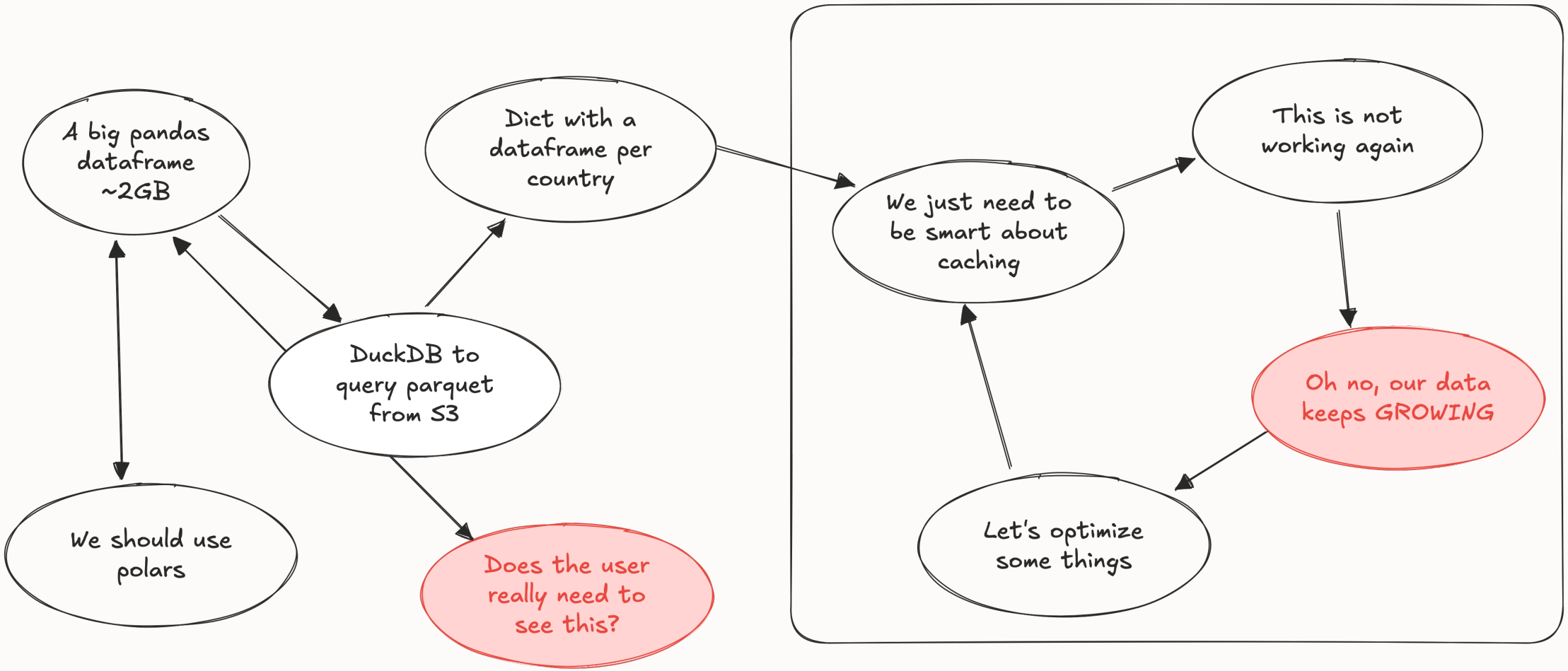
Don’t people use Elastic for this?

JavaScript required?

Should we be spending time here?

The data saga.

The data "loop" of desperation



Being in charge of Data + UI

The data saga.

How big is your data going to be?

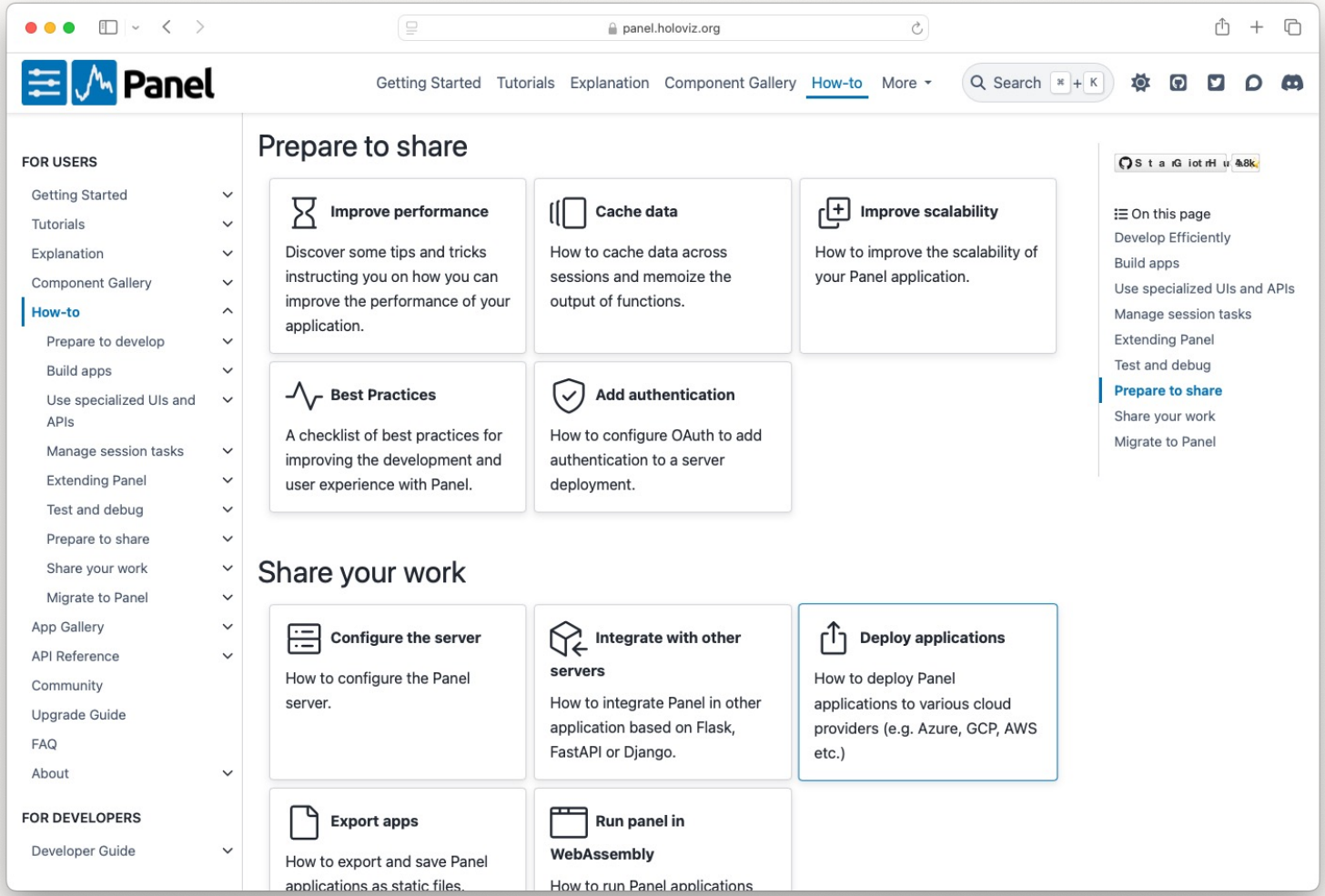
Are you sacrificing the user?

Does this work if the data doubles?

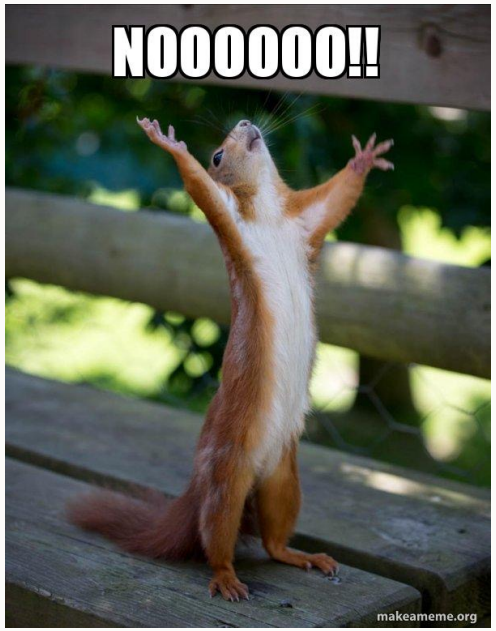
Throwing caching everywhere

It’s all about the caveats.

The docs are great



The ”gas station”



Serving multiple applications

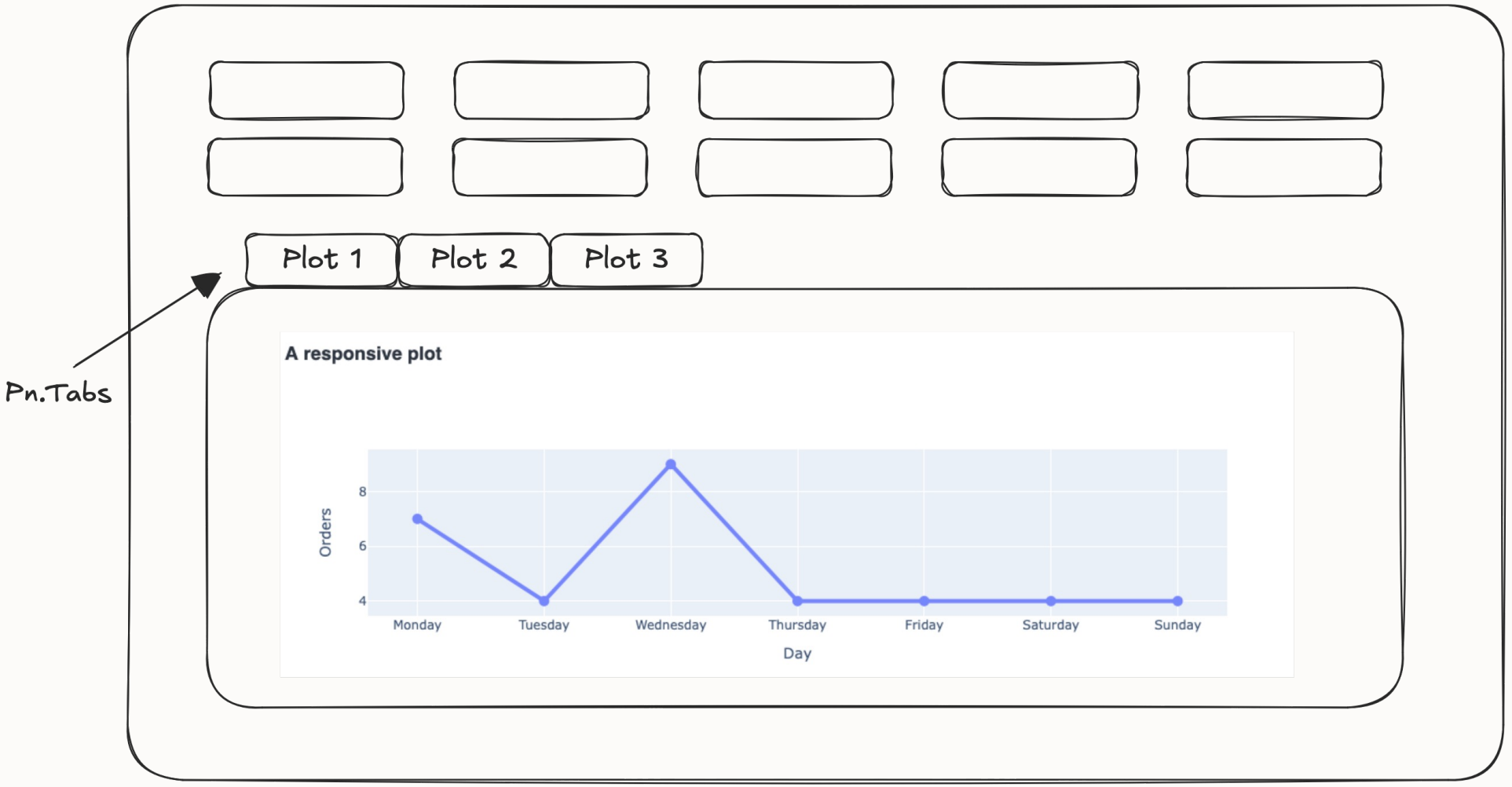
If you want to serve more than one app on a single server you can use the `pn.serve` function. By supplying a dictionary where the keys represent the URL slugs and the values must be either Panel objects or functions returning Panel objects you can easily launch a server with a number of apps, e.g.:

```
import panel as pn
pn.serve({
    'markdown': '# This is a Panel app',
    'json': pn.pane.JSON({'abc': 123})
})
```

Note that when you serve an object directly all sessions will share the same state, i.e. the parameters of all components will be synced across sessions such that the change in a widget by one user will affect all other users. Therefore you will usually want to wrap your app in a function, ensuring that each user gets a new instance of the application:

```
def markdown_app():
    return '# This is a Panel app'

def json_app():
    return pn.pane.JSON({'abc': 123})

pn.serve({
    'markdown': markdown_app,
    'json': json_app
})
```

"The Dashboard"



The caveats saga

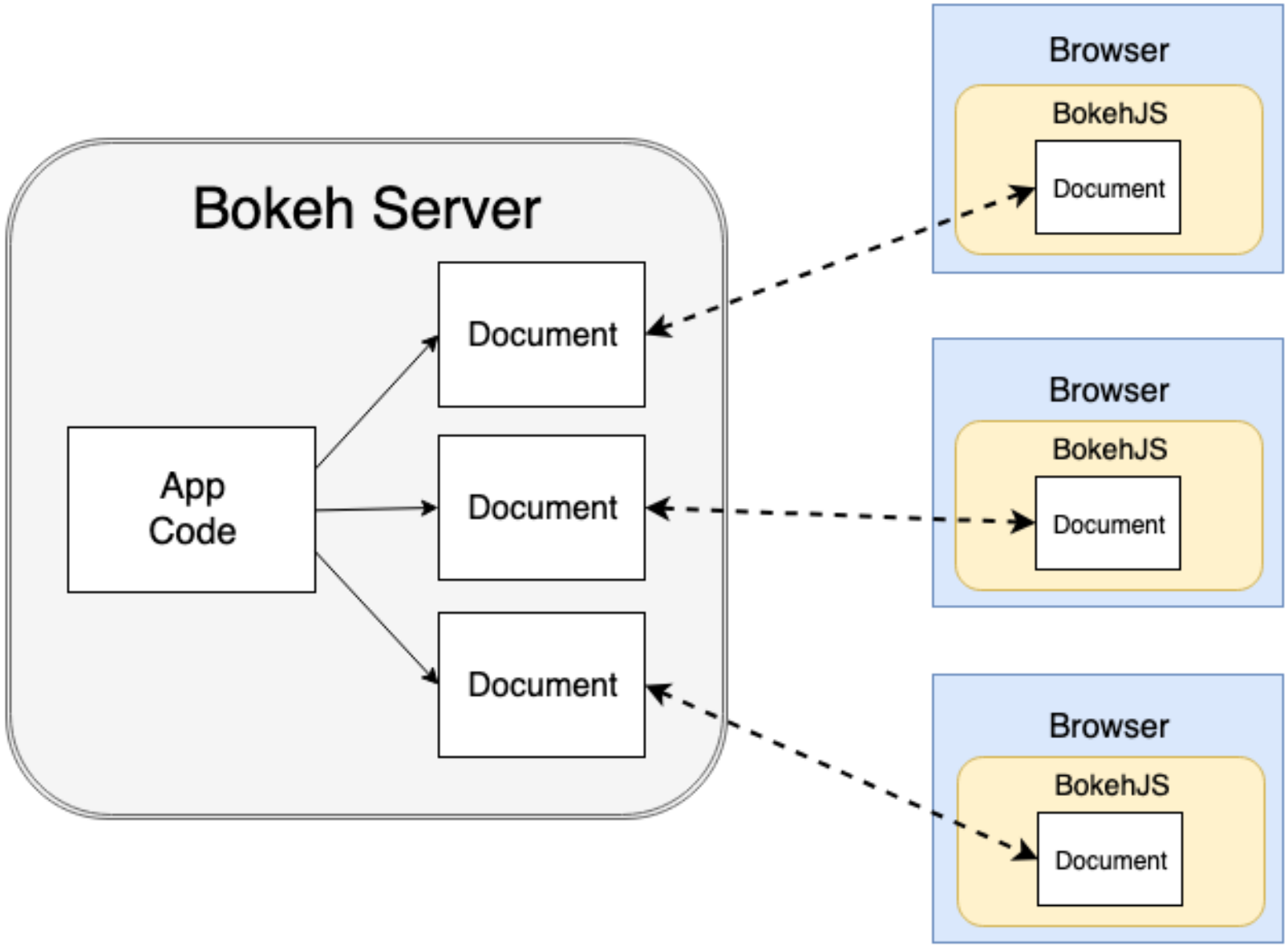
Can you catch errors easily?

How fast are you are rendering?

Can you add a feature easily?

Then things got really slow

The ”Kubernetes”


Bokeh Server
App Code
Document
Document
Document
Browser
BokehJS
Document
Browser
BokehJS
Document
Browser
BokehJS
Document

```
pn.serve({
    'markdown': '# This is a Panel app',
    'json': pn.pane.JSON({'abc': 123})
}, title={'markdown': 'A Markdown App', 'json': 'A JSON App'}
)
```

The `panel serve` command has the following options:

```
positional arguments:
  DIRECTORY-OR-SCRIPT   The app directories or scripts to serve (serve empty document if not spe

options:
  -h, --help            show this help message and exit
  --port PORT           Port to listen on
  --address ADDRESS     Address to listen on
  --unix-socket UNIX-SOCKET
                        Unix socket to bind. Network options such as port, address, ssl options
  --log-level LOG-LEVEL
                        One of: trace, debug, info, warning, error or critical
  --log-format LOG-FORMAT
                        A standard Python logging format string (default: '%(asctime)s %(message
  --log-file LOG-FILE   A filename to write logs to, or None to write to the standard stream (de
  --use-config CONFIG   Use a YAML config file for settings
  --args ...            Command line arguments remaining to passed on to the application handler
  --dev [FILES-TO-WATCH ...]
                        Enable live reloading during app development. By default it watches all
                        --show, must be placed before the directory or script. NOTE: This settin
                        restart the server
  --show                Open server app(s) in a browser
  --allow-websocket-origin HOST[:PORT]
                        Public hostnames which may connect to the Bokeh websocket With unix sock
  --prefix PREFIX       URL prefix for Bokeh server URLs
  --ico-path ICO_PATH   Path to a .ico file to use as the favicon.ico, or 'none' to disable favi
  --keep-alive MILLISECONDS
                        How often to send a keep-alive ping to clients, 0 to disable.
  --check-unused-sessions MILLISECONDS
                        How often to check for unused sessions
  --unused-session-lifetime MILLISECONDS
                        How long unused sessions last
  --stats-log-frequency MILLISECONDS
                        How often to log stats
  --mem-log-frequency MILLISECONDS
                        How often to log memory usage information
  --use-xheaders        Prefer X-headers for IP/protocol information
  --ssl-certfile CERTFILE
                        Absolute path to a certificate file for SSL termination
  --ssl-keyfile KEYFILE
```

There are a lot of options

Improve performance

Design new feature

Something breaks

Improve performance

Design new feature

Something breaks

Improve performance

Design new feature

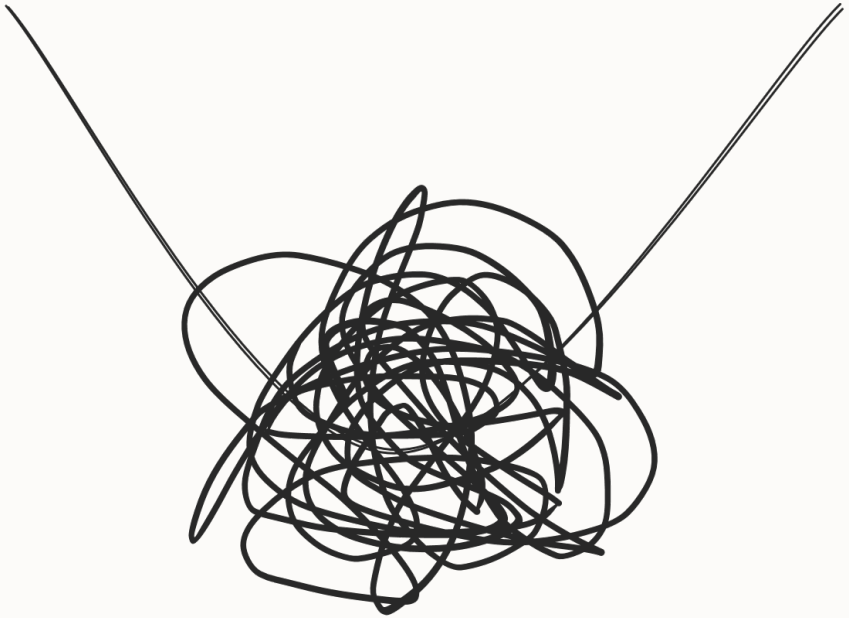
Something breaks

...

You can just test Duarte.

Back end

Front end



Enough ranting. What's the point?

The point is: You can outgrow ~~Panel~~.

Python.

Testing Panel vs. Testing an API

Deploying Bokeh vs. Deploying Uvicorn

Developing a dashboard vs. An API

Focusing on the things you don’t want to.

The front-end belongs in a Front-End

Panel is great, you should use it

Incredible community

Thousands of options

Great for prototyping

Next level docs

But it's hard to predict the future

Should we have an API?

What framework?

Where are you focusing?

The 2x rule

Front end is not Python (yet)

Python is incredible

WASM, PyScript, and more

Eventually you should leave

You are not a FE!

Thank you very much PyData!