

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

A saga about pain, scale, and bad decisions.

/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?)

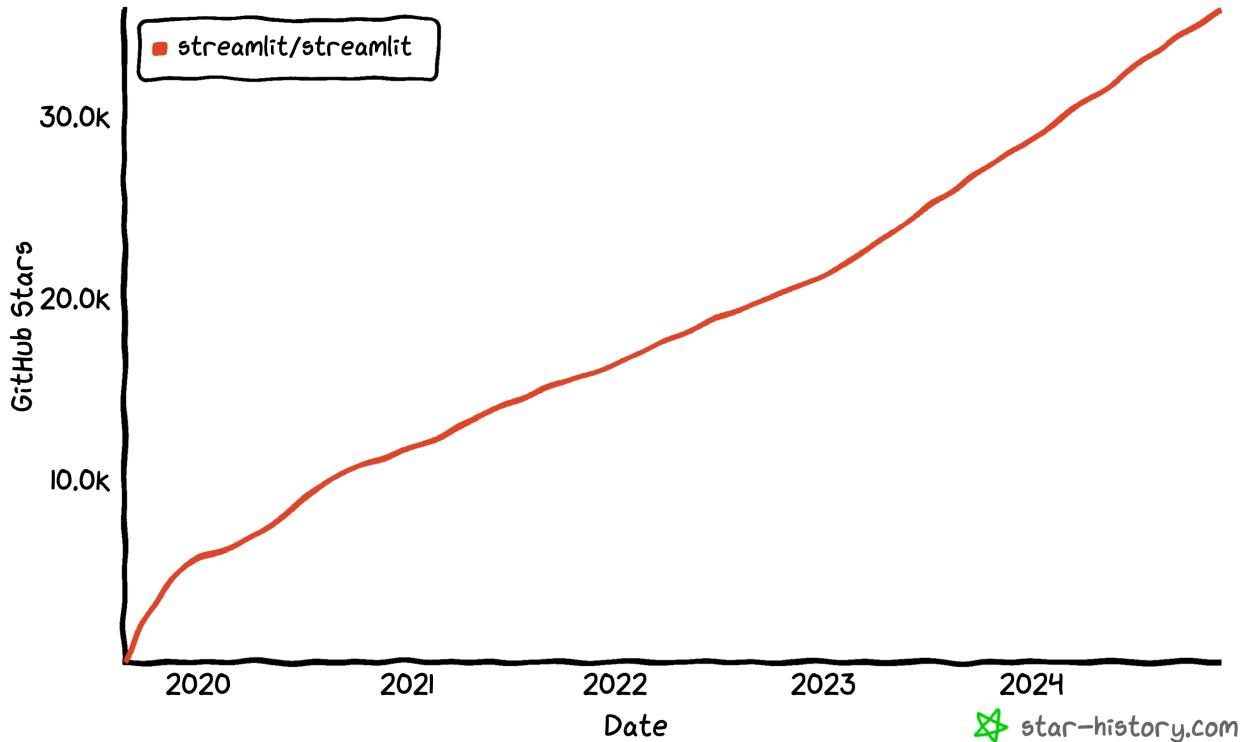
A primer on Panel

Bad decisions and lessons learned

Final thoughts and conclusion

The “dashboard”.

⭐ Star History



My App • Streamlit

```
MyApp.py
```

```
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 first app

Hello world!



localhost

Deploy :

Reset Chat

Make a graph comparison of the most used data visualization packages in Python. (can be just a guess)

✓ Visualization created!
Generating bar plot...

Comparison of Most Used Data Visualization Packages in Python

Package	Value (approx.)
Matplotlib	60
Seaborn	50
Plotly	40
Bokeh	30
Altair	20

Ask me to create a visualization or chat! ▶

<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?)

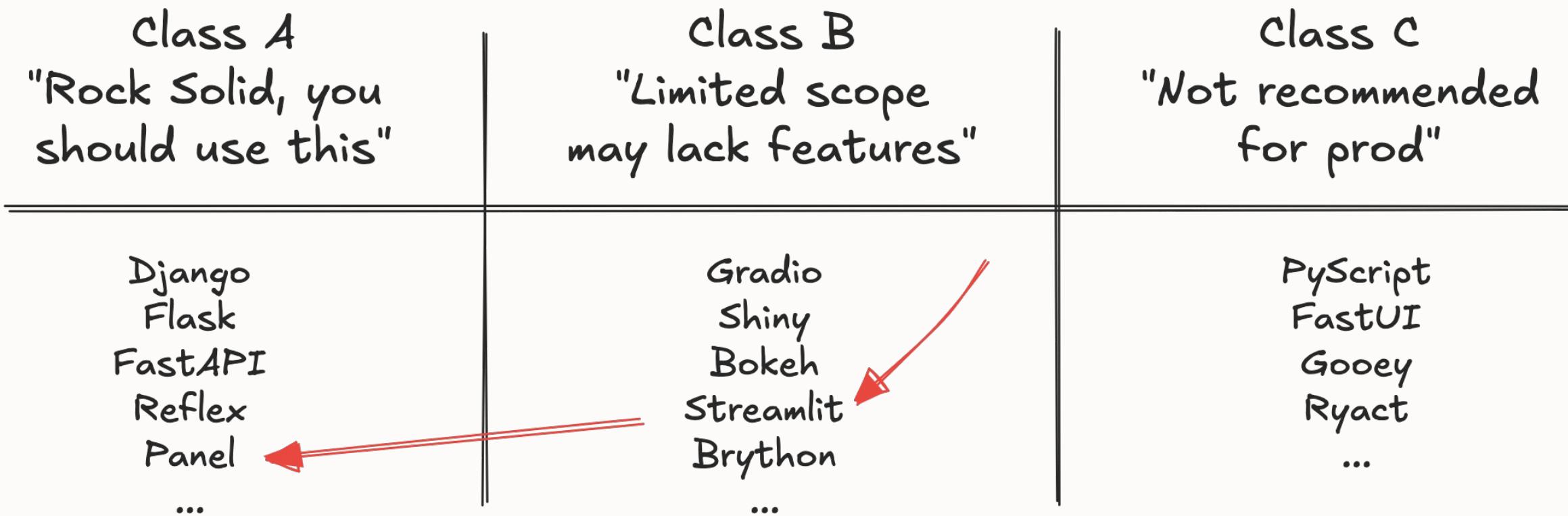
A primer on Panel

Bad decisions and lessons learned

Final thoughts and conclusion

We're not JS, but we have options

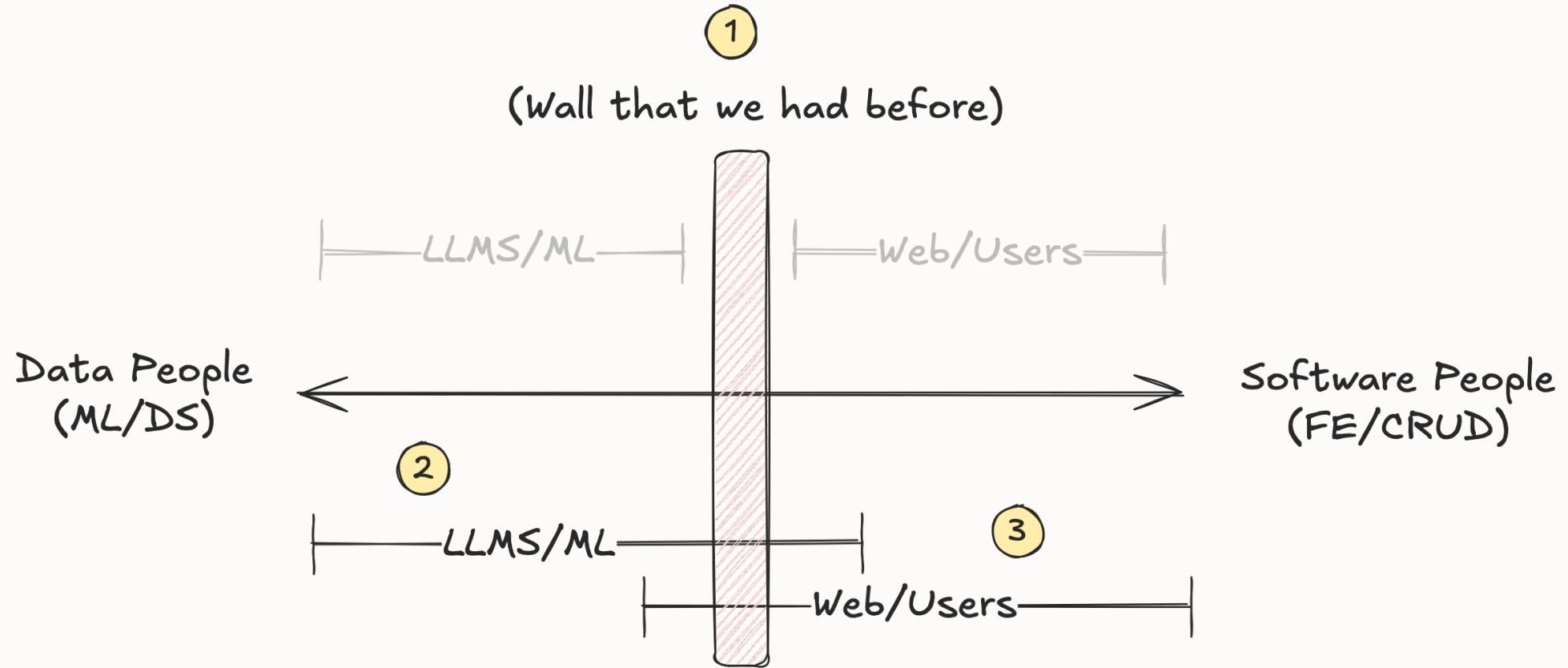
“Pure Python” web development



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

duarteocarmo.com

PyData
Global



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?

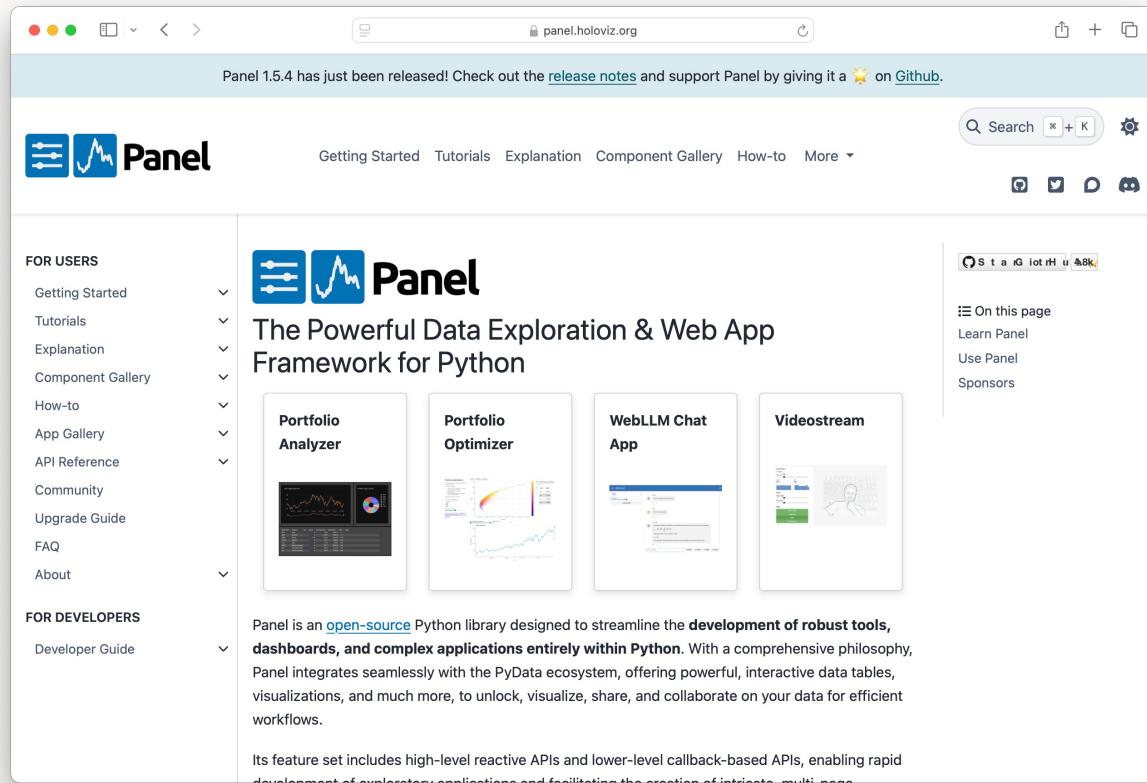
Python on the web (really?)

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

The screenshot shows the Panel app gallery website (panel.holoviz.org) displayed in a web browser. The left sidebar, titled "FOR USERS", contains links for "Getting Started", "Tutorials", "Explanation", "Component Gallery", "How-to", and "App Gallery". The "App Gallery" section is expanded, listing various applications: Altair Brushing, Deckgl Game Of Life, Gapminders, Glaciers, Hvplot Explorer, Iris Kmeans, Nyc Deckgl, Penguin Crossfilter, Penguin Kmeans, Portfolio Analyzer, Portfolio Optimizer, Streaming Videostream, Vtk Interactive, Vtk Slicer, Vtk Warp, and WebIm. The main content area displays seven examples of Panel applications:

- Portfolio Optimizer**: A plot showing the efficient frontier of a stock portfolio.
- Streaming Videostream**: A demo showing face detection and image transforms on webcam input.
- Windturbines Explorer**: A map and data visualization of US wind turbines.
- Portfolio Analyzer**: A dashboard for analyzing stock prices and market data.
- OGGM Glaciers**: A dashboard for visualizing glacier data from OGGM.
- VTK Slicer**: A medical image segmentation application using VTK.

On the right side, there is a sidebar with a "StarHub" icon, a search bar, and social media links. Below the search bar are buttons for "launch jupyterlite", "launch server", and "launch pyodide". The sidebar also includes links for "On this page", "HoloViz Topics", and "Community Gallery".

Hundreds of examples

The screenshot shows the Panel Component Gallery interface. On the left, a sidebar titled "FOR USERS" contains links to "Getting Started", "Tutorials", "Explanation", "Component Gallery" (which is currently selected), "Panes", "Widgets", "Layouts", "Chat", "Global", "Indicators", "Templates", "Custom Components", "How-to", "App Gallery", "API Reference", "Community", "Upgrade Guide", "FAQ", and "About". Below this is another section titled "FOR DEVELOPERS" with a link to "Developer Guide". The main content area is titled "Widgets" and displays a grid of 18 different components. Each component has a thumbnail image and a brief description below it. The components include: ArrayInput, AutocompleteInput, Button, ButtonIcon, CheckBoxGroup, CheckButtonGroup, Checkbox, CodeEditor, ColorMap, ColorPicker, CrossSelector, DataFrame, DatePicker, DateRangePicker, and DateRangeSlider. To the right of the grid is a sidebar titled "On this page" with links to "Panes", "Widgets", "Layouts", "Chat", "Global", "Indicators", "Templates", and "Custom_Components". At the top of the page is a navigation bar with links to "Getting Started", "Tutorials", "Explanation", "Component Gallery", "How-to", "More", a search bar, and various social media and sharing icons.

Hundreds of widgets (and custom React)

The screenshot shows a web browser window with the URL `holoviz-topics.github.io`. The page title is "Applicable Recipes". On the left, there's a sidebar with a navigation menu:

- Panel Chat Examples
- Home
- Chat Features
- Kickstart Snippets
- [Applicable Recipes](#)
- Linked Resources

The main content area is titled "Openai Two Bots" and describes how to use the `ChatInterface` to create two bots that chat with each other. It includes a section for "Highlights" with two bullet points:

- The user decides the callback user and avatar for the response.
- A system message is used to control the conversation flow.

Below this, there's a screenshot of a chat interface between "User" and "HoloViz Panel". The User asks about utilizing HoloViz Panel in projects. The Nerd Bot responds by explaining the library's benefits for creating interactive dashboards and applications. The Happy Bot then adds that the library allows for quick prototyping and iteration on designs. At the bottom, there's a "Send a message" input field and a "Source code for openai_two_bots.py" link.

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.

Region

The filter saga.

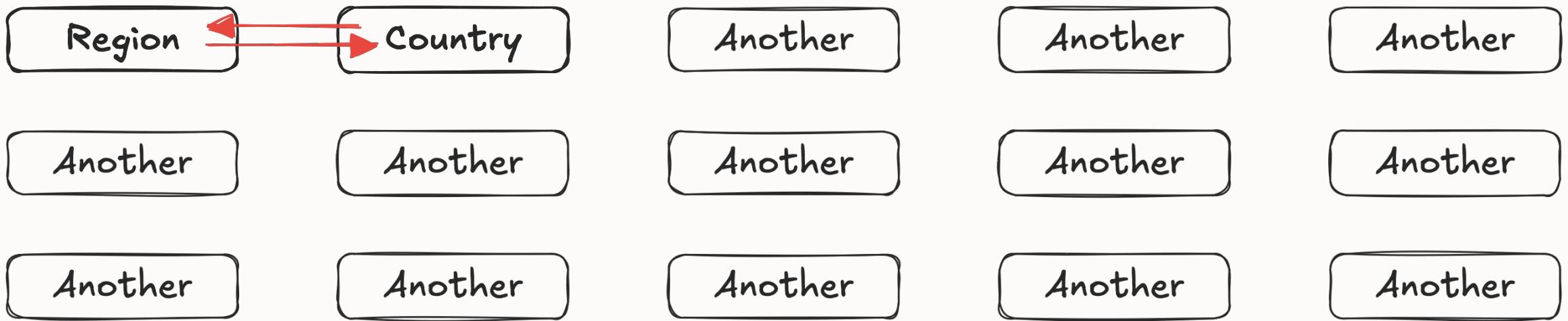
Region

Country

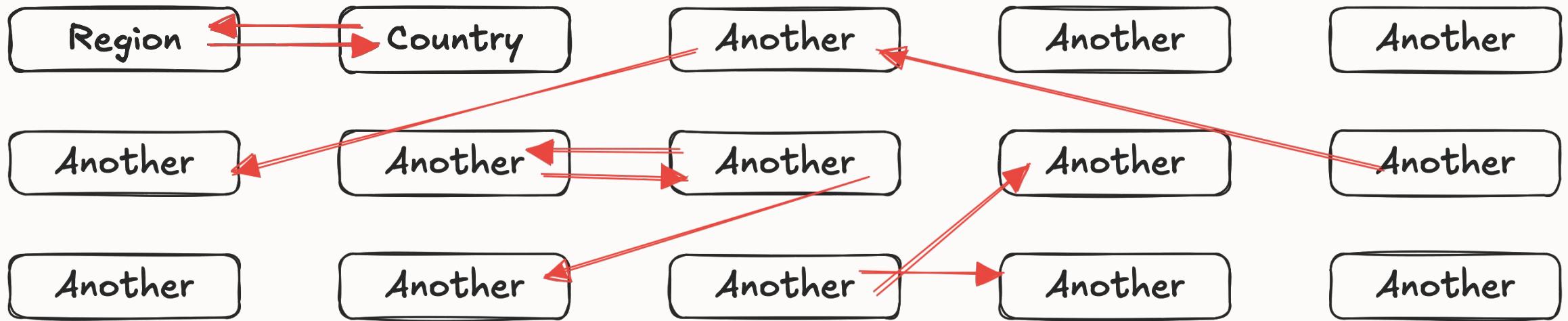
The filter saga.



The filter saga.



The filter saga.

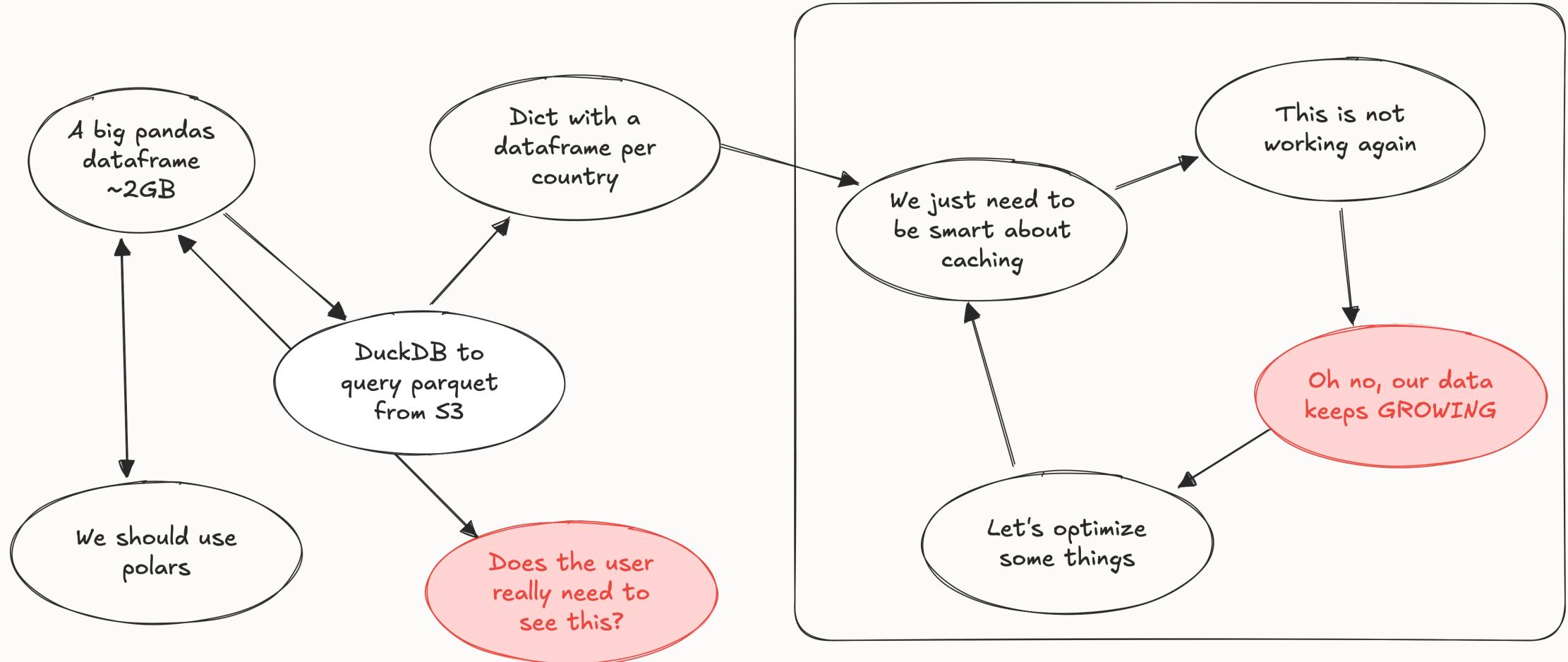


The filter saga.

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 screenshot shows a web browser displaying the [Panel documentation](https://panel.holoviz.org). The URL is visible in the address bar. The page is titled "Panel" with a logo. The navigation bar includes links for "Getting Started", "Tutorials", "Explanation", "Component Gallery", "How-to" (which is underlined, indicating it's the current section), and "More". A search bar and various social media sharing icons are also present.

The main content area is titled "Prepare to share". It features several cards:

- Improve performance**: Discover some tips and tricks instructing you on how you can improve the performance of your application.
- Cache data**: How to cache data across sessions and memoize the output of functions.
- Improve scalability**: How to improve the scalability of your Panel application.
- Best Practices**: A checklist of best practices for improving the development and user experience with Panel.
- Add authentication**: How to configure OAuth to add authentication to a server deployment.

Below this section is another titled "Share your work", containing the following cards:

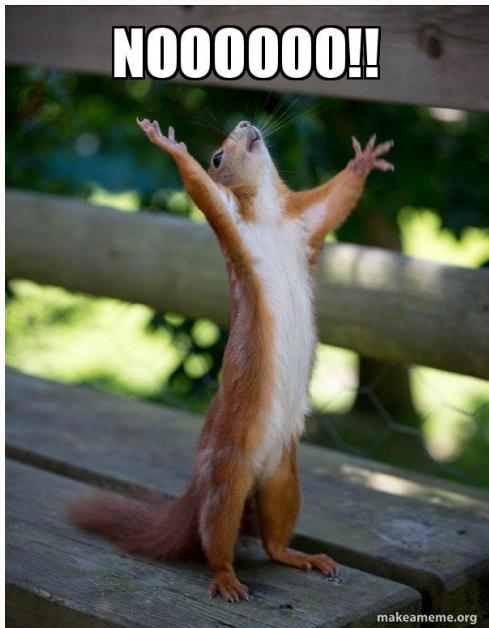
- Configure the server**: How to configure the Panel server.
- Integrate with other servers**: How to integrate Panel in other application based on Flask, FastAPI or Django.
- Deploy applications**: How to deploy Panel applications to various cloud providers (e.g. Azure, GCP, AWS etc.).
- Export apps**: How to export and save Panel applications as static files.
- Run panel in WebAssembly**: How to run Panel applications.

A sidebar on the right lists "On this page" topics such as "Develop Efficiently", "Build apps", "Use specialized UIs and APIs", "Manage session tasks", "Extending Panel", "Test and debug", and "Prepare to share". It also includes links for "Share your work" and "Migrate to Panel".

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})
})
```

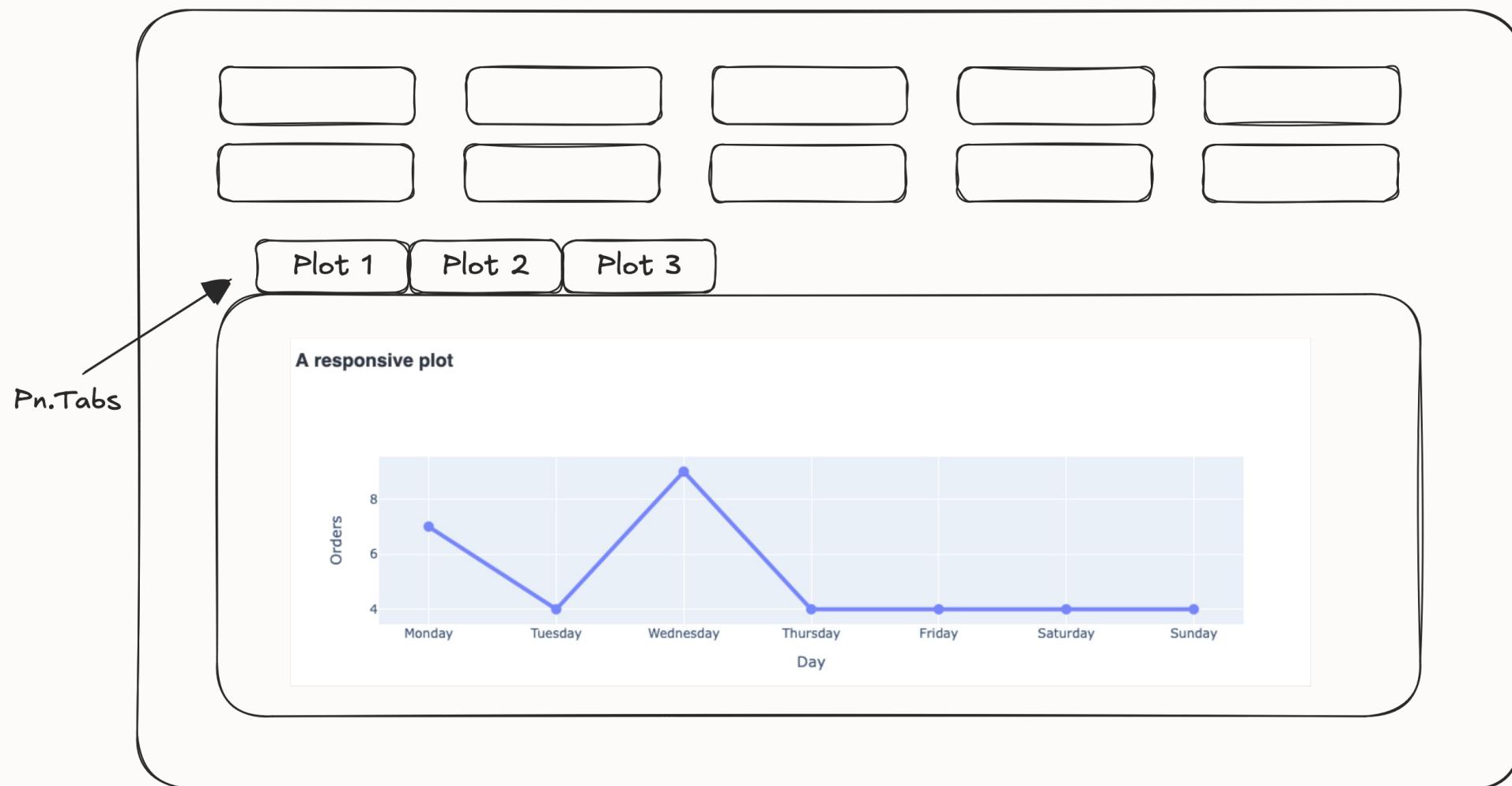
Note that when you serve an object directly all sessions will share the same state, i.e. the parameters of all components will be synced across sessions such that the change in a widget by one user will affect all other users. Therefore you will usually want to wrap your app in a function, ensuring that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

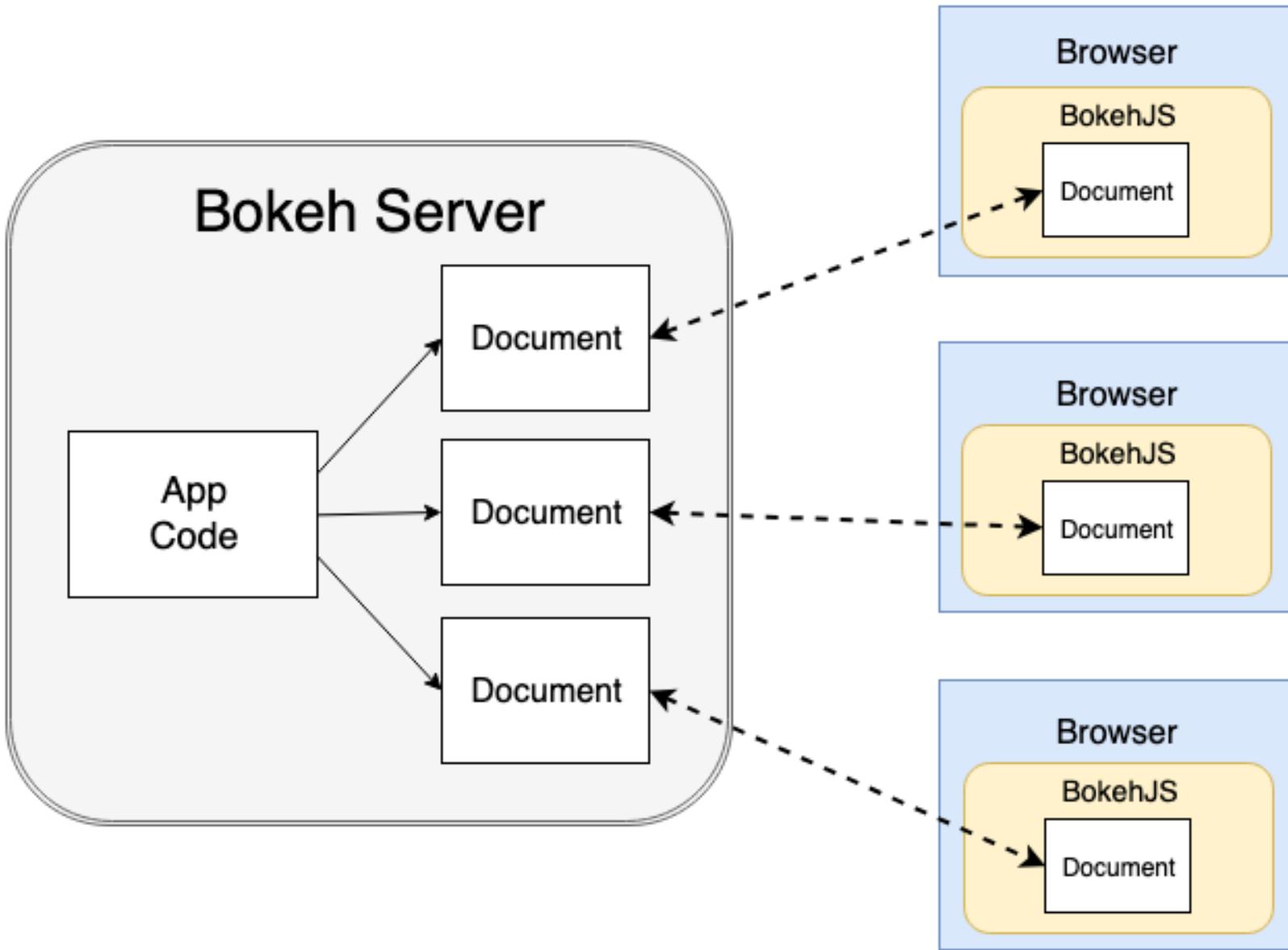
Can you catch errors easily?

How fast are you rendering?

Can you add a feature easily?

Then things got really slow

The "Kubernetes"



```
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 setting  
                      restarts 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  
                      Absolute path to a key file for SSL termination
```

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

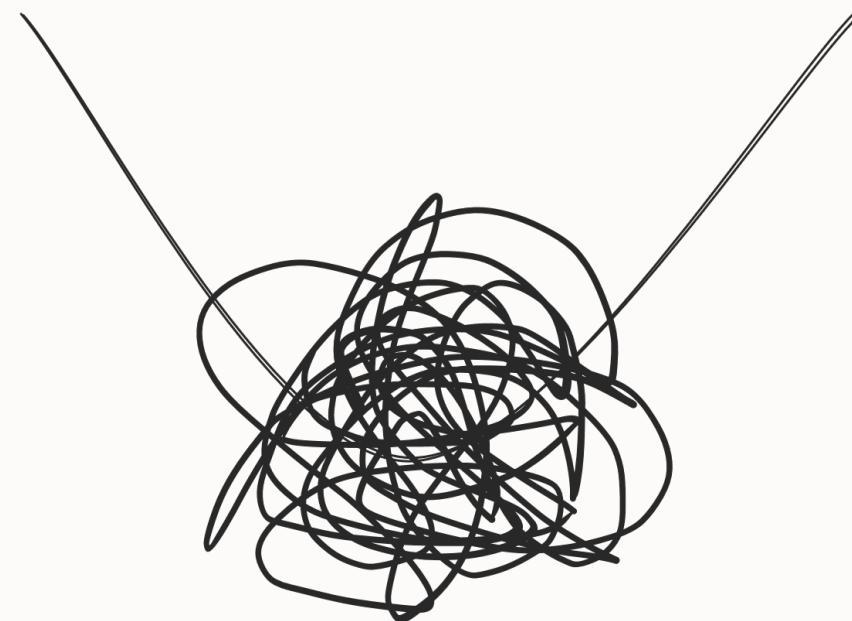
Something breaks

...

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 Unicorn

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!