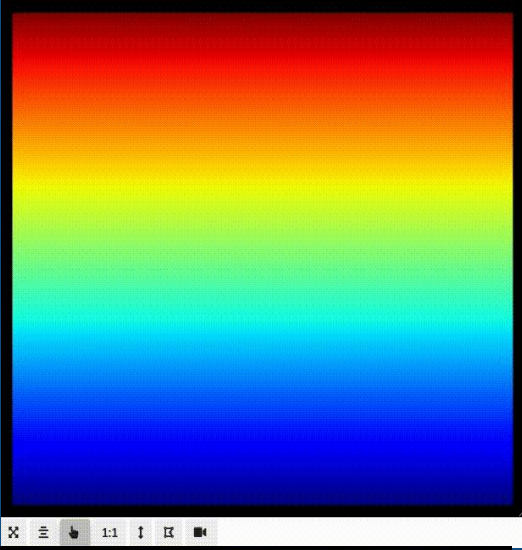
https://github.com/fastplotlib/fastplotlib

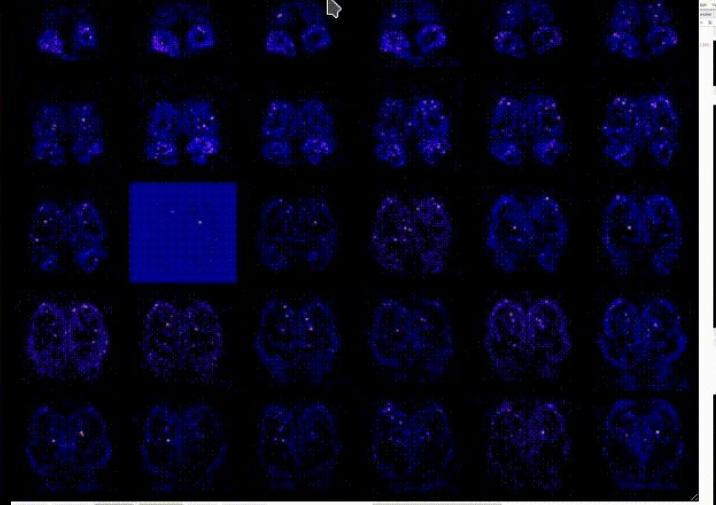
# fastplotlib

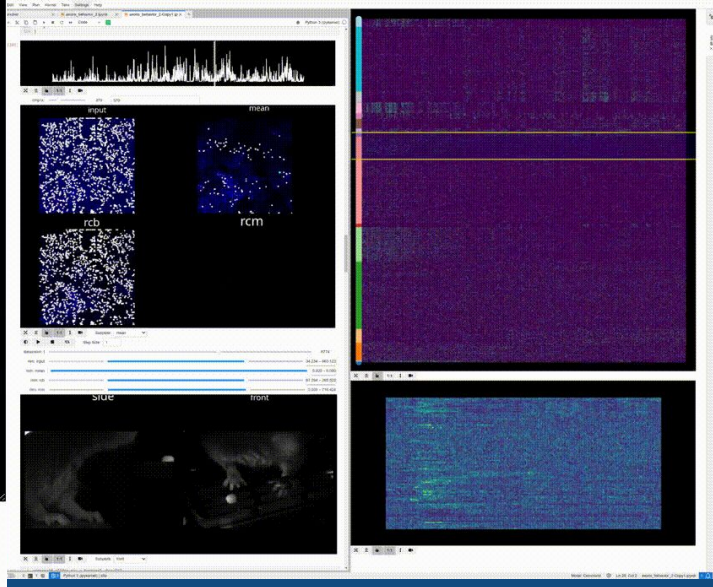
***Ultrafast interactive visualizations***

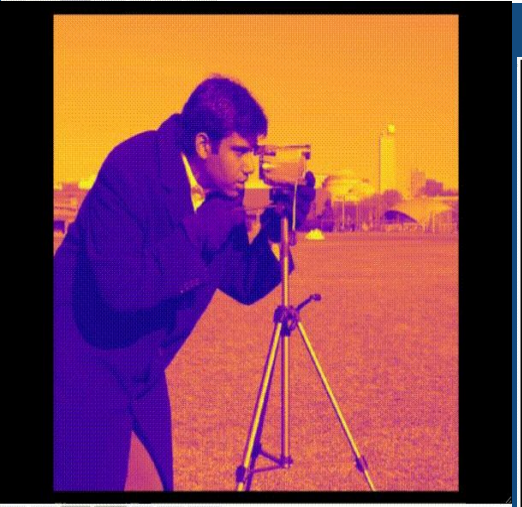

**Caitlin Lewis** **@caitlinllewis** **clewis7**

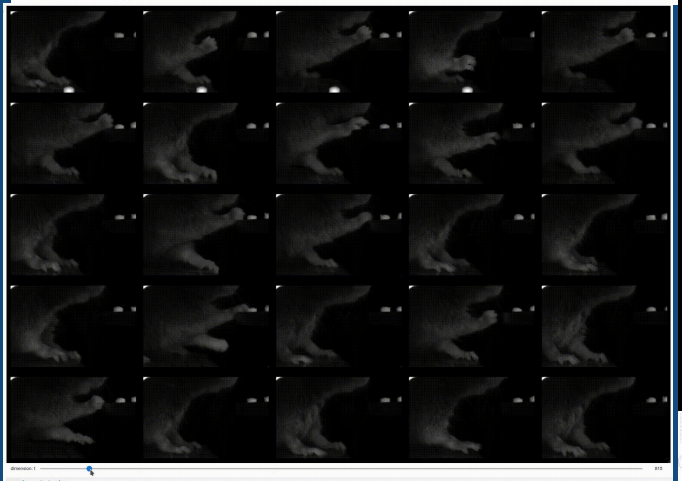
**Kushal Kolar** **@kushalkolar** **kushalkolar**











1:1
Subplots:
(0, 0)
Step Size:
1
Speed
100
dimension: t
0
1:1
1:1
0

# Graphics technologies improvement

Graphics ~20 years ago


Current graphics

# New Graphics APIs

Vulkan / Metal (Mac) / DX12 (Windows)

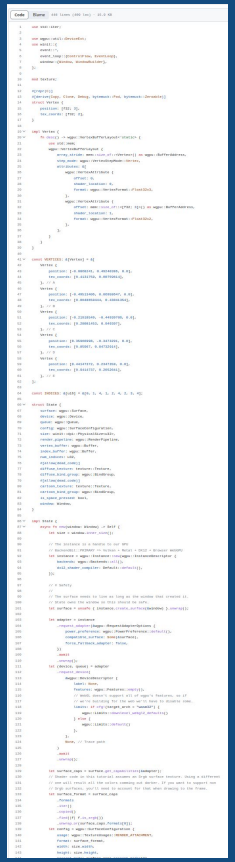
Image

Wikipedia:

> “Vulkan is a low-overhead, **cross-platform API, open standard** for 3D graphics and computing … **higher performance** and **more efficient CPU and GPU** usage **compared to older OpenGL**”

***Basically: New APIs: very fast, efficient, & leverage modern GPU hardware better than OpenGL***

**This is also what newer games use!**



# Abstractions are built

Image

Vulkan / Metal / DX12

wgpu — ~400 lines

wgpu-py — ~400 lines

pygfx — ~15 lines - rendering engine

**fastplotlib** — ~4 lines

# Abstractions are built

Image

Vulkan / Metal / DX12

Open standard:
- Mozilla
- AMD
- Apple
- Microsoft

wgpu ~400 lines

wgpu-py ~400 lines

pygfx ~15 lines - rendering engine

**fastplotlib** ~4 lines

# Abstractions are built

Image

Vulkan / Metal / DX12

wgpu

~400 lines

Open standard:

- Mozilla
- AMD
- Apple
- Microsoft

wgpu-py

~400 lines

Core devs:

- Almar Klein
- Korijn van Golen

Zimmer Biomet
They are awesome!

pygfx

~15 lines - rendering engine

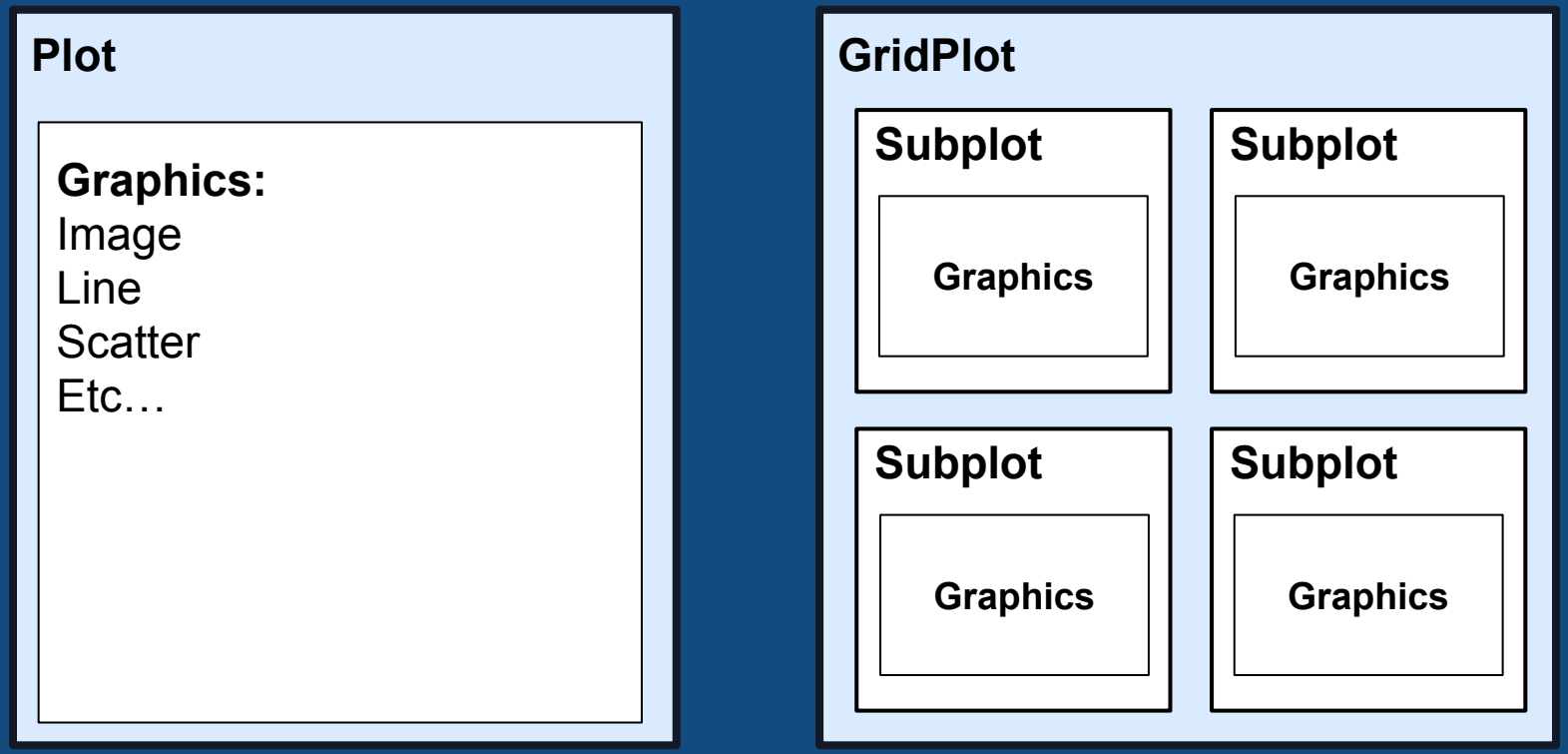
**fastplotlib**

~4 lines

# fastplotlib

- High-level API for scientific plotting - inspiration from *pyqtgraph*, *bokeh*, etc.
- Uses the *pygfx* rendering engine
- Very new - April 2022
- **Interactive in jupyter notebooks - cloud computing, remote infrastructure**
- Goals: fast visualization, **expressive & elegant API -** we'll tell you what this means!
- Core developers:
  - **Kushal Kolar - Flatiron/NYU**
  - **Caitlin Lewis - UNC**
- Possible *fastplotlib* backend for napari in the future :D

# fastplotlib API



# Focus on data, not on rendering!

## pygfx

```
# create a canvas, renderer and scene
canvas = WgpuCanvas()
renderer = gfx.renderers.WgpuRenderer(canvas)
scene = gfx.Scene()

# create a camera, set position to center of image
camera = gfx.OrthographicCamera(512, 512)
camera.position.y = 256 # y center of the scene
camera.scale.y = -1 # flip the y axis
camera.position.x = 256 # x center of the scene

colormap1 = gfx.cm.plasma # define a colormap

# 512x512 array of random data
rand_img_data = np.random.rand(512, 512).astype(np.float32) * 255

# define Geometry with the grid set as a Texture using the image data
image_obj = gfx.Image(
    gfx.Geometry(grid=gfx.Texture(rand_img_data, dim=2)),
    gfx.ImageBasicMaterial(clim=(0, 255), map=colormap1),
)

scene.add(image_obj)
def animate():
    renderer.render(scene, camera)
    canvas.request_draw()
canvas.request_draw(animate)
canvas
```

## fastplotlib

```
plot = Plot() # create a new plot

data = np.random.rand(512, 512) # some data

plot.add_image(data=data) # plot data as an image

plot.show() # show the plot :D
```

Focus on scientific data!

canvas
renderer
camera
viewports
geometry
material
buffers

} fastplotlib manages rendering!

# Gridplots & subplots

## pygfx - gets very tedious

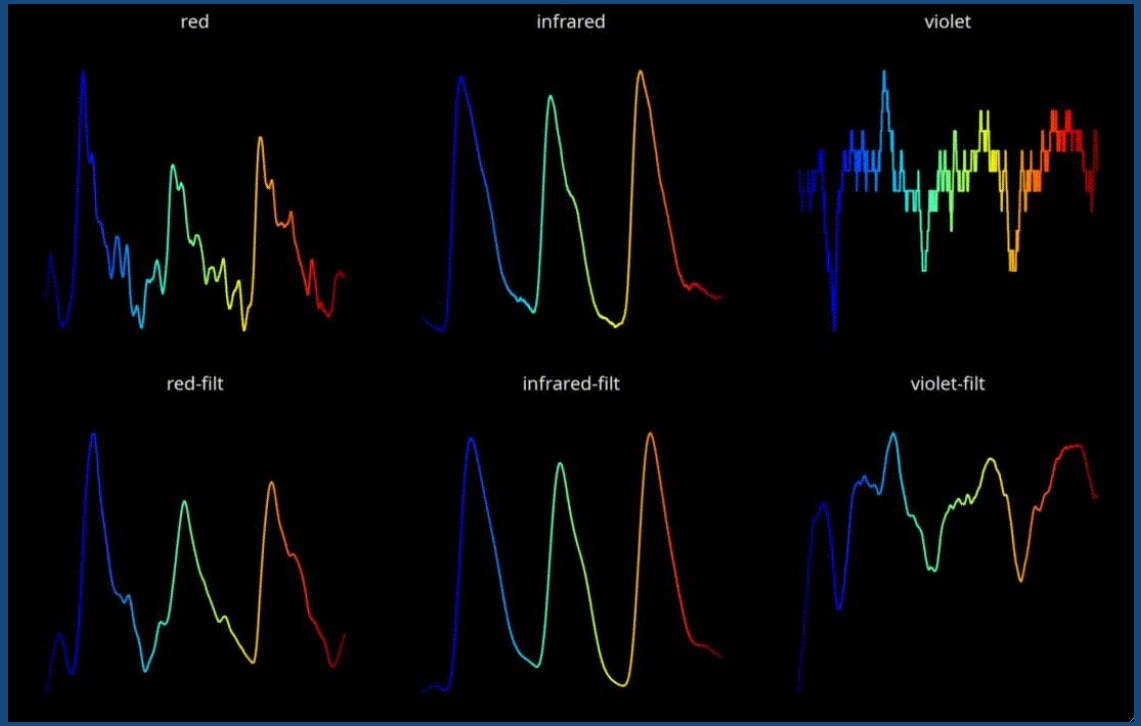
```
canvas = WgpuCanvas()
renderer = gfx.renderers.WgpuRenderer(canvas)
dims = (512, 512)  # image dimensions
center_cam_pos = (256, 256, 0)
cmaps = [gfx.cm.inferno, gfx.cm.plasma, gfx.cm.magma, gfx.cm.viridis]
scenes = list()
cameras = list()
images = list()
controllers = list()
cntl_defaults = list()
viewports = list()
for i in range(4):
    # create scene for this subplot
    scene = gfx.Scene()
    scenes.append(scene)
    # create image WorldObject
    img = gfx.Image(
        gfx.Geometry(           grid=gfx.Texture(np.random.rand(*dims).astype(np.float32) * 255, dim=2)
        ),
        gfx.ImageBasicMaterial(clim=(0, 255), map=cmaps[i]),
    )
    images.append(img)
    scene.add(img)
    camera = gfx.OrthographicCamera(*dims)
    camera.position.set(*center_cam_pos)
    cameras.append(camera)
    viewport = gfx.Viewport(renderer)
    viewports.append(viewport)
    controller = gfx.PanZoomController(camera.position.clone())
controller.add_default_event_handlers(viewport, camera)   controllers.append(controller)
cntl_defaults.append(controller.save_state())
@renderer.add_event_handler("resize")
def layout(event=None):
    """
    Update the viewports when the canvas is resized
    """
    w, h = renderer.logical_size
    w2, h2 = w / 2, h / 2
    viewports[0].rect = 10, 10, w2, h2
    viewports[1].rect = w / 2 + 5, 10, w2, h2
    viewports[2].rect = 10, h / 2 + 5, w2, h2
    viewports[3].rect = w / 2 + 5, h / 2 + 5, w2, h2
def animate():
    for img in images:
        img.geometry.grid.data[:] = np.random.rand(*dims).astype(np.float32) * 255
        img.geometry.grid.update_range((0, 0, 0), img.geometry.grid.size)
    for camera, controller in zip(cameras, controllers):
        controller.update_camera(camera)
    for viewport, s, c in zip(viewports, scenes, cameras):
        viewport.render(s, c)
    renderer.flush()
    canvas.request_draw()
def center_objects(ev):
    for con, cam, img in zip(controllers, cameras, images):
        con.show_object(cam, img)
        con.zoom(1.3)
renderer.add_event_handler(center_objects, "double_click")
layout()
if __name__ == "__main__":
    canvas.request_draw(animate)
    run()
```

## fastplotlib

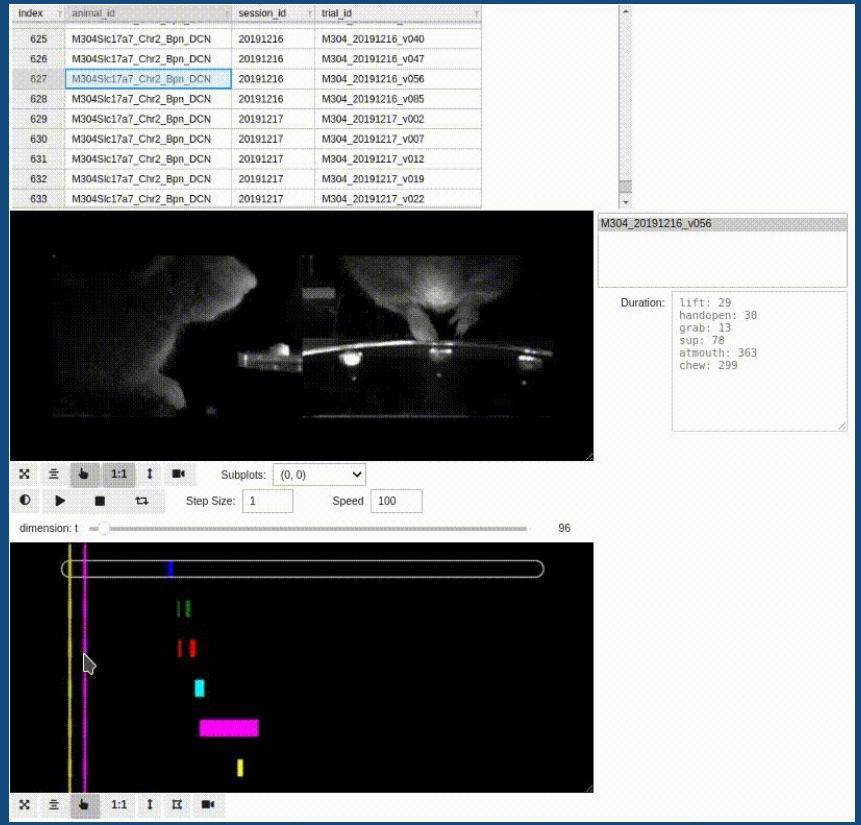
```
# GridPlot of shape 2 x 3
grid_plot = GridPlot(shape=(2, 3))

# Make a random image graphic for each subplot
for subplot in grid_plot:
    img_data = np.random.rand(512, 512)
    # add image to the subplot
    subplot.add_image(img_data)

# update the image graphics with new generated data
def set_random_frame(gp):
    for sp in gp.subplots:
        new_data = np.random.rand(512, 512)
        sp.graphics[0].data = new_data

# add the animation
grid_plot.add_animations(set_random_frame)
grid_plot.show()
```

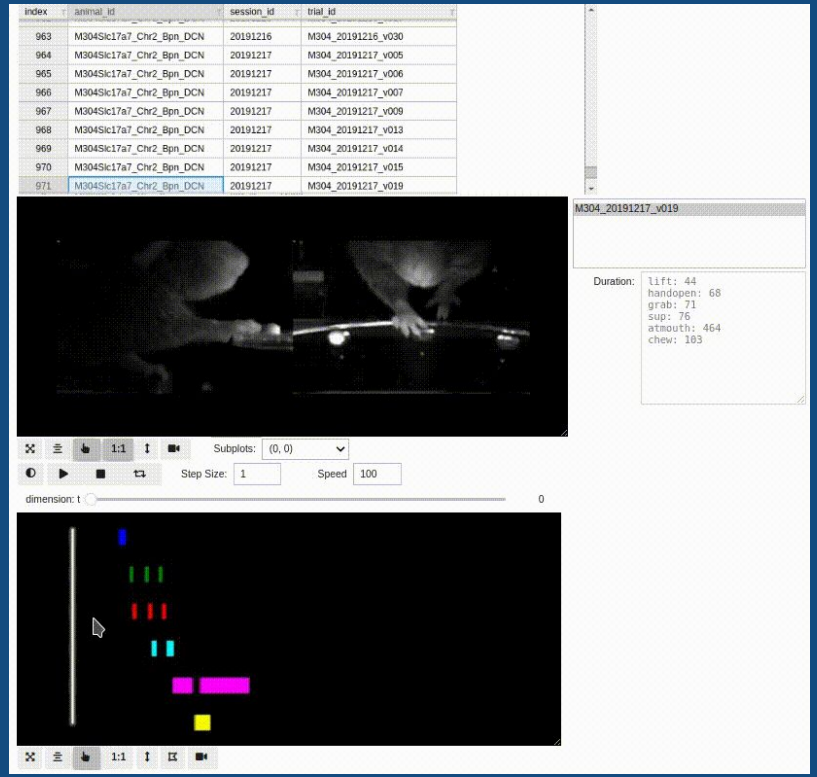
# Example Applications

# Real time sensor data - pulse ox development (Arjun)

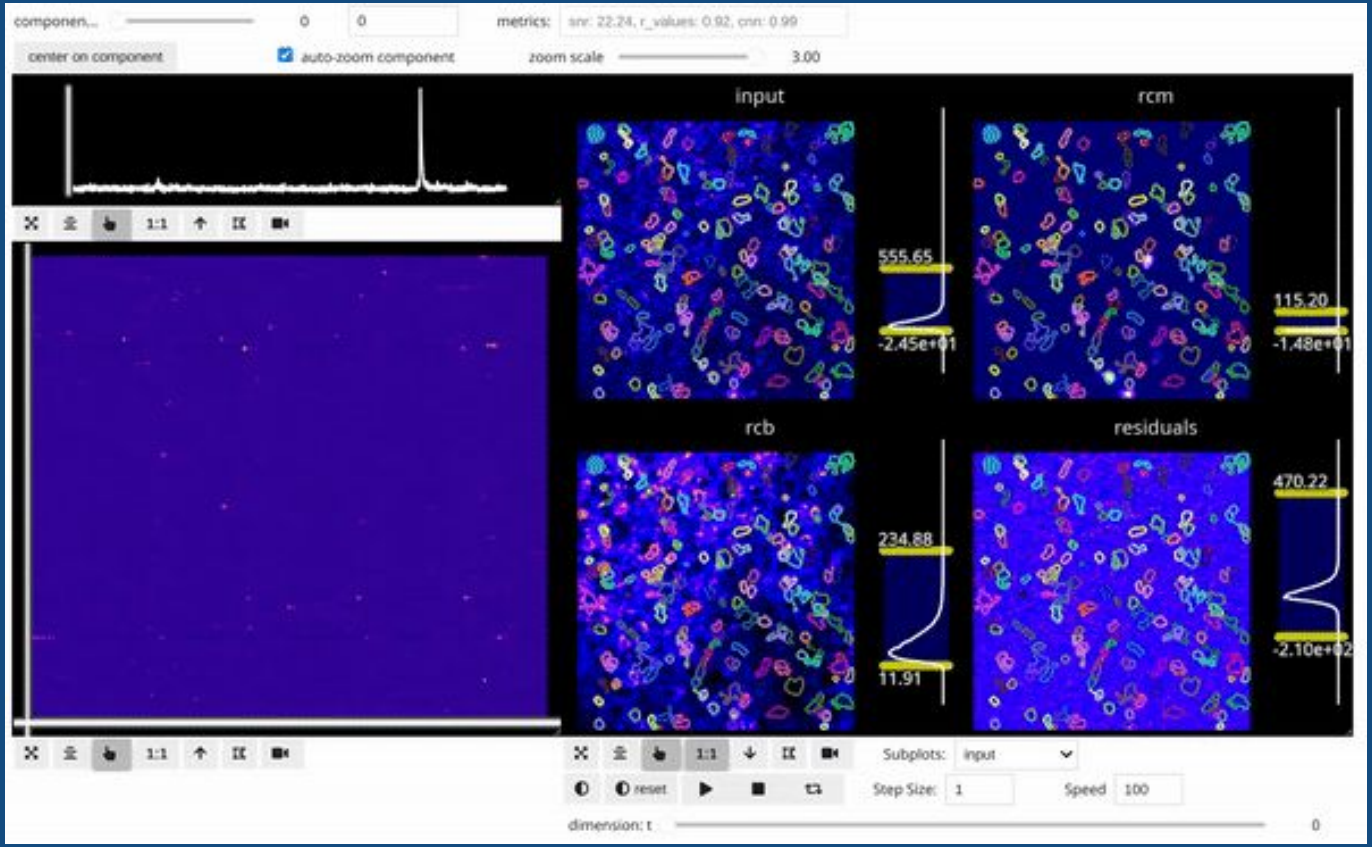


# animal-soup (Caitlin :D )



# mesmerize-viz



# Demo time!

https://github.com/fastplotlib/fastplotlib-sfn2023

# Current state of fastplotlib

- Alpha
  - Evolving API - we are constantly improving things!
  - Don’t hesitate to post an issue or discussion forum post!
- Moderate test coverage
  - ~70%: graphics, graphic features, layouts
  - ~20%: selector tools, toolbars, complex events
- Some basic components are not ready yet
  - Text is recent
  - Axes are primitive
  - No legends yet (great for newcomers to contribute!)

# Roadmap for 2023

Contributions and ideas are welcome from people with all levels of experience! :D

There are several items highlighted with 🟢 that are perfect for newcomers!

https://github.com/kushalkolar/fastplotlib/issues/55

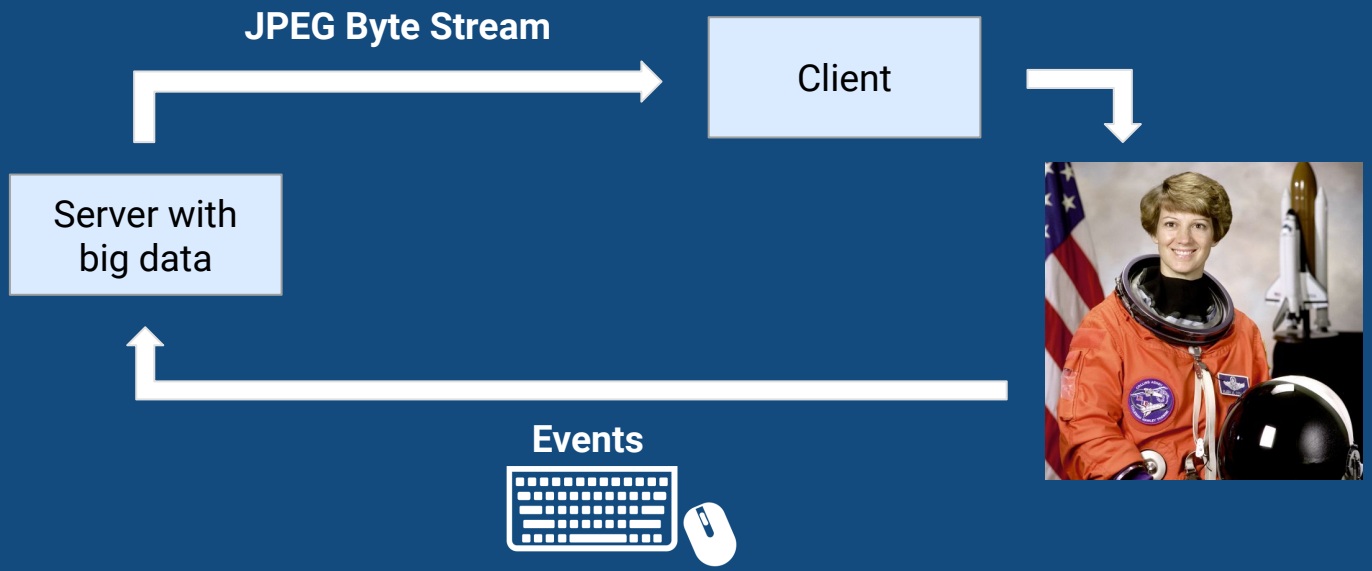
**Come to our sprint tomorrow!**

- **fastplotlib for your use case**
- **Contributions**

# Appendix

# Fastplotlib via remote frame buffer

- **jupyter-rfb**
  - Server-side rendering, client only receives a jpeg byte stream



# Why not just copy matplotlib?

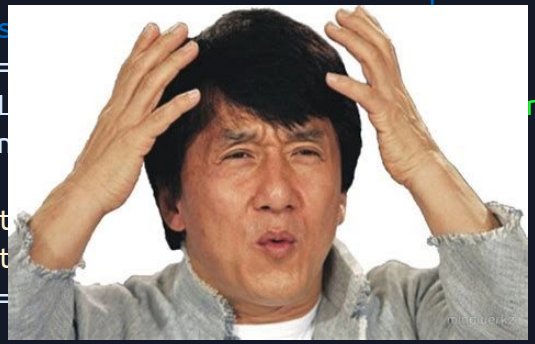
**We are not just re-implementing the matplotlib API - we can do better!**

**Modern Python is so much nicer than when matplotlib was created!**

# Why not just copy matplotlib?

**matplotlib**

```
points = np.array([x, y]).T.reshape(-1, 1, 2)
segments = np.concatenate([points[:-1],
points[1:]], axis=1)

# Create a continuous norm to map from data
points
norm =
lc = L                                    ridis',
norm=n
# Set
lc.set
lc.set
line =
```



**fastplotlib**

```
# cmap_values from an array, so the colors on the
sine line will be based on the sine y-values
sine_graphic = plot.add_line(
    data=sine,
    thickness=10,
    cmap="plasma",
    cmap_values=sine[:, 1]
)

# or make changes! :D
sine_graphic.cmap_values = cosine
sine_graphic.cmap = "jet"
```