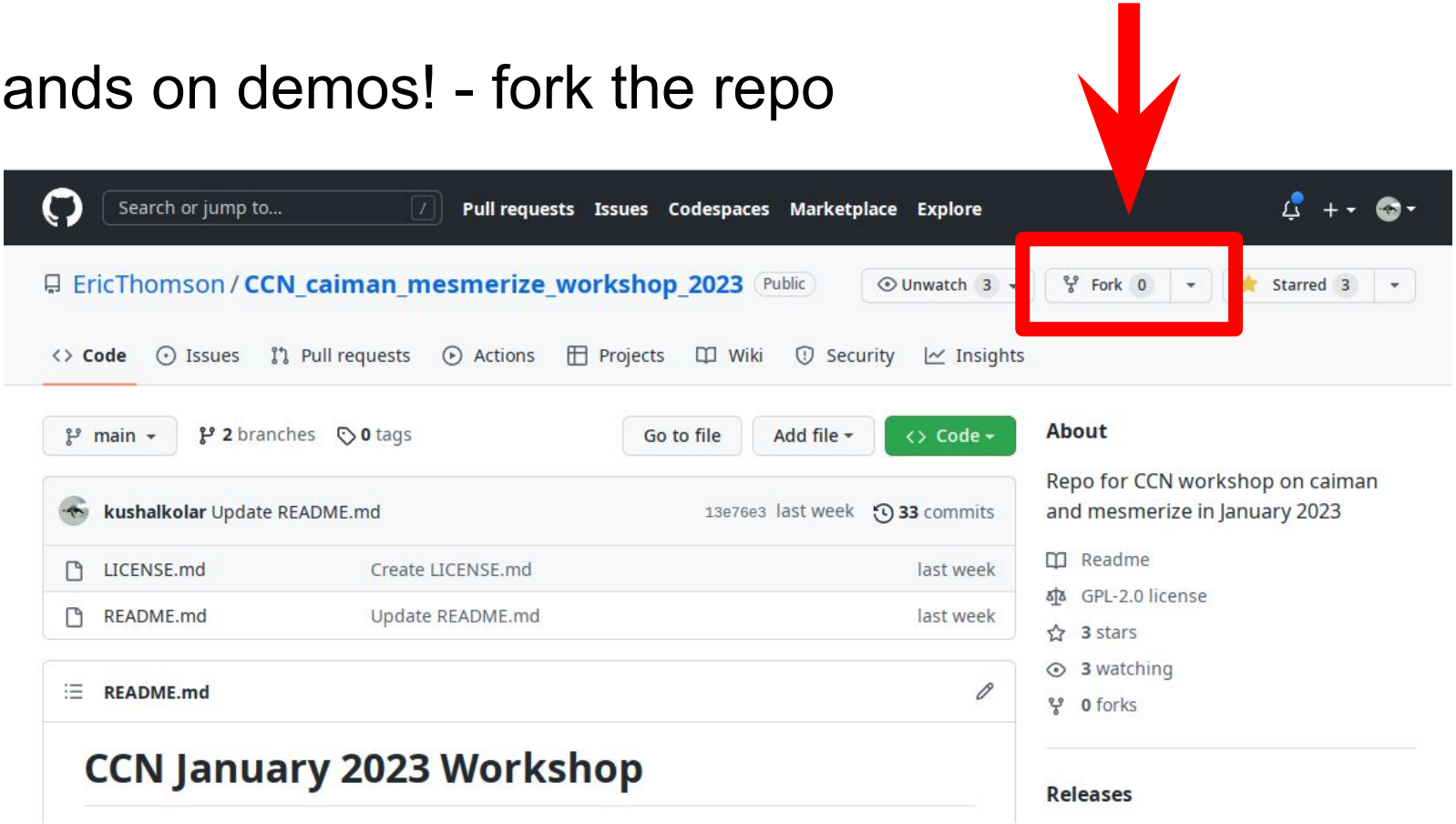
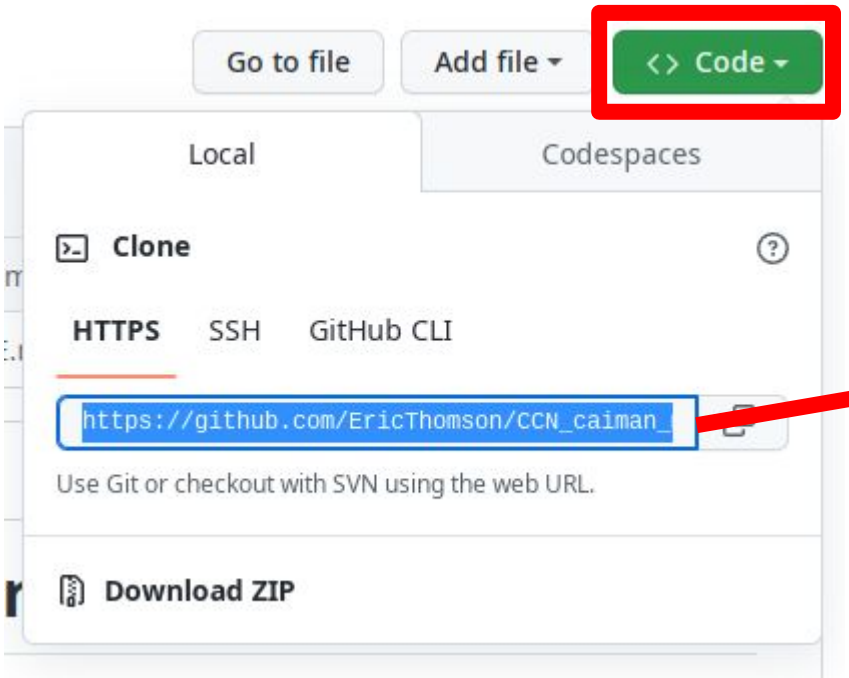
fastplotlib

Hands on demos! - fork the repo



Clone it locally



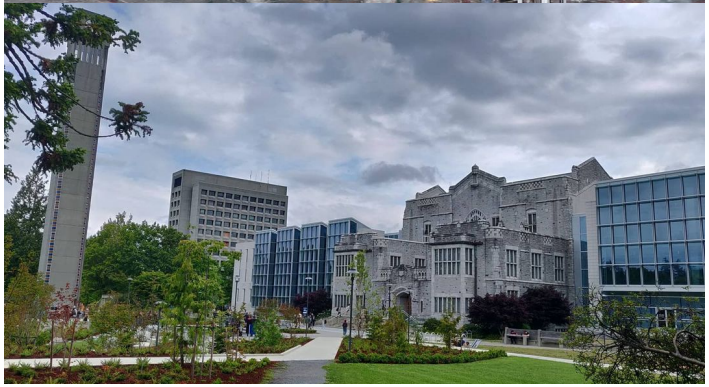
1. make a dir to organize your repos if you don't have one
   a. mkdir repos
   b. cd repos

2. Clone the repo
   a. git clone <url-here>

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

Partner of EMBL

THE UNIVERSITY
OF BRITISH COLUMBIA



What do I do?

- Hantman lab
- Calcium imaging in motor cortex during skilled motor behavior
- Tool development with Caitlin
  - mesmerize-core
  - fastplotlib
  - Realtime online analysis
- Data analysis with Teena
  - Very large calcium imaging dataset

fastplotlib demo

Python plotting ecosystem

- pyqtgraph - 2010
  - Interactive visualizations using Qt
  - limited GPU acceleration

- VisPy - 2012
  - Started by Luke Campagnola (pyqtgraph creator), Almar Klein, and some other people
  - Plotting API based on **OpenGL**
  - Limited high level API

OpenGL is getting old, initial release in 1992

- not fast enough to handle thousands or millions of objects with Python calls
  - Largely due to OpenGL limitations

*There are other libraries but they are not geared towards **fast** interactive visualization (matplotlib, bokeh, etc.)

Graphics technology & GPUs have come a long way since the 90s!

Vulkan® released in 2016, is the successor to OpenGL

“**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**… - Wikipedia

basically: Vulkan is new, very fast, efficient, & leverages modern GPU hardware

From pygfx readme:

- …API feels so much nicer than OpenGL. It's well defined, no global state, we can use compute shaders, use storage buffers (random access), etc.

- … predefining objects and pipelines and then executing these. Almost everything is "prepared" ... big advantage for us Pythoneers: **the amount of code per-draw-per-object is very limited. This means we can have a lot of objects and still be fast**

- Vulkan is too low-level to use directly
  - **we need a rendering engine**
    - in Python, not C++ (so we can make more than 1 plot per year :D)

pygfx is a Python rendering engine that uses Vulkan

- https://github.com/pygfx/pygfx
- Very new, bleeding edge of graphics technology
  - Image shader was added on Jan 21, 2022!
- Open source
  - Almar Klein and other developers at Zimmer Biomet in Netherlands.
  - They are amazing!
  - Already in use for biomedical imaging
- **Same code works as a desktop application and in notebooks!**
- **NON-BLOCKING!**
  - Render a scene in a notebook, interactively write code to modify it

*A few other Vulkan libraries exist, ex. DatoViz, but they are low-level and require using C

pygfx primitives

- World Object
  - Geometry
  - Material
- Renderer
- Canvas
- Camera
- Controller
- Events

** You do not need to know or understand any of this!*

fastplotlib

- High-level Python API for scientific plotting
- Built with pygfx rendering engine
- **Works interactively in jupyter notebooks**
  - **Visualize data on cloud computing and other remote infrastructure**
- Goals: fast visualization, **expressive & elegant API**

https://github.com/kushalkolar/fastplotlib

*very new, April 2022

Reduce rendering engine boilerplate

pygfx

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

# create a scene
scene = gfx.Scene()

# create a camera, set position to center of image
camera = gfx.OrthographicCamera(512, 512)
camera.position.y = 256
camera.scale.y = -1
camera.position.x = 256
colormap1 = gfx.cm.plasma
# 512x512 array of random data
rand_img_data = np.random.rand(512, 512).astype(np.float32) * 255
# create an image graphic
# define Geometry with the grid set as a Texture using the random data
img_graphic = gfx.Image(
    gfx.Geometry(grid=gfx.Texture(rand_img_data, dim=2)),
    gfx.ImageBasicMaterial(clim=(0, 255), map=colormap1),
)
scene.add(img_graphic)
def animate():
    renderer.render(scene, camera)
    canvas.request_draw()
canvas.request_draw(animate)
canvas
```

fastplotlib

```
plot = Plot()

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

plot.add_image(data=data)

plot.show()
```

Reduce rendering engine boilerplate

pygfx

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

# create a scene
scene = gfx.Scene()

# create a camera, set position to center of image
camera = gfx.OrthographicCamera(512, 512)
camera.position.y = 256
camera.scale.y = -1
camera.position.x = 256
colormap1 = gfx.cm.plasma
# 512x512 array of random data
rand_img_data = np.random.rand(512, 512).astype(np.float32) * 255
# create an image graphic
# define Geometry with the grid set as a Texture using the random data
img_graphic = gfx.Image(
    gfx.Geometry(grid=gfx.Texture(rand_img_data, dim=2)),
    gfx.ImageBasicMaterial(clim=(0, 255), map=colormap1),
)
scene.add(img_graphic)
def animate():
    renderer.render(scene, camera)
    canvas.request_draw()
canvas.request_draw(animate)
canvas
```

fastplotlib

```
plot = Plot()

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

plot.add_image(data=data)

plot.show()
```

fastplotlib lets you focus on scientific data!

- ✖ canvas
- ✖ renderer
- ✖ camera
- ✖ viewports
- ✖ geometry
- ✖ material
- ✖ GPU buffers

} fastplotlib manages these for you!

With animations

pygfx

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

# create a scene
scene = gfx.Scene()

# create a camera, set position to center of image
camera = gfx.OrthographicCamera(512, 512)
camera.position.y = 256
camera.scale.y = -1
camera.position.x = 256
colormap1 = gfx.cm.plasma
# 512x512 array of random data
rand_img_data = np.random.rand(512, 512).astype(np.float32) * 255
# create an image graphic
# define Geometry with the grid set as a Texture using the random data
img_graphic = gfx.Image(
   gfx.Geometry(grid=gfx.Texture(rand_img_data, dim=2)),
   gfx.ImageBasicMaterial(clim=(0, 255), map=colormap1),
)
scene.add(img_graphic)
def animate():
   # update with new random image
   img_graphic.geometry.grid.data[:] = np.random.rand(512,
512).astype(np.float32) * 255
   img_graphic.geometry.grid.update_range((0, 0, 0),
img_graphic.geometry.grid.size)
   renderer.render(scene, camera)
   canvas.request_draw()
canvas.request_draw(animate)
canvas
```

fastplotlib

```
plot = Plot()

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

image_graphic = plot.add_image(data=data)

def update_data():
   new_data = np.random.rand(512, 512)
   image_graphic.data = new_data

plot.add_animations(update_data)
plot.show()
```

GridPlot, Subplot interface!

pygfx

```
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(1.2)
renderer.add_event_handler(center_objects, "double_click")
layout()
if __name__ == "__main__":
    canvas.request_draw(animate)
    run()
```

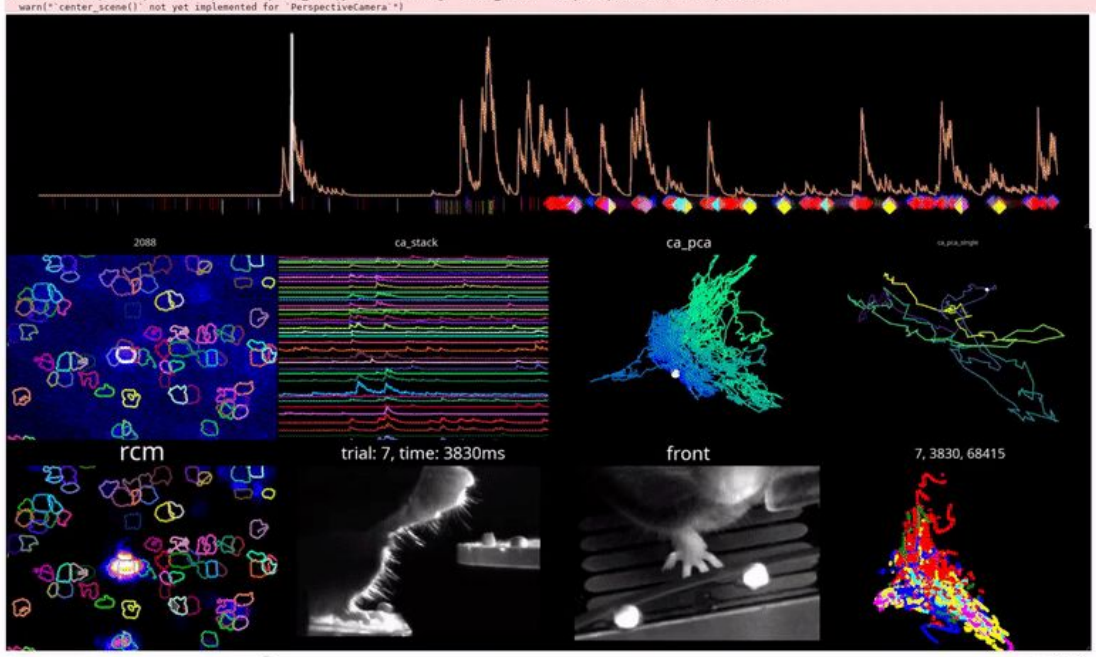
fastplotlib

```
# 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()
```

This would take *thousands* of lines in pygfx
In fastplotlib it's ~150

```
time_slider.observe(slider_changed, "value")
plot_temporal.renderer.add_event_handler(resize_slider, "resize")

VBox([plot_temporal.show(), gp.show(), time_slider])
```

```
/home/kushalk/repos/fastplotlib/fastplotlib/layouts/_base.py:205: UserWarning: `center_scene()` not yet implemented for `PerspectiveCamera`
  warn(f"`center_scene()` not yet implemented for `PerspectiveCamera`")
```




```
plot_temporal["temporal"][:].present = False
gp["ca_movie"]["img"].link("click", target=gp["ca_movie"]["contours"], feature="colors", new_data="w", callback_function=euclidean)
gp["rcm"]["img"].link("click", target=gp["rcm"]["contours"], feature="colors", new_data="w", callback_function=euclidean)
```

fastplotlib API

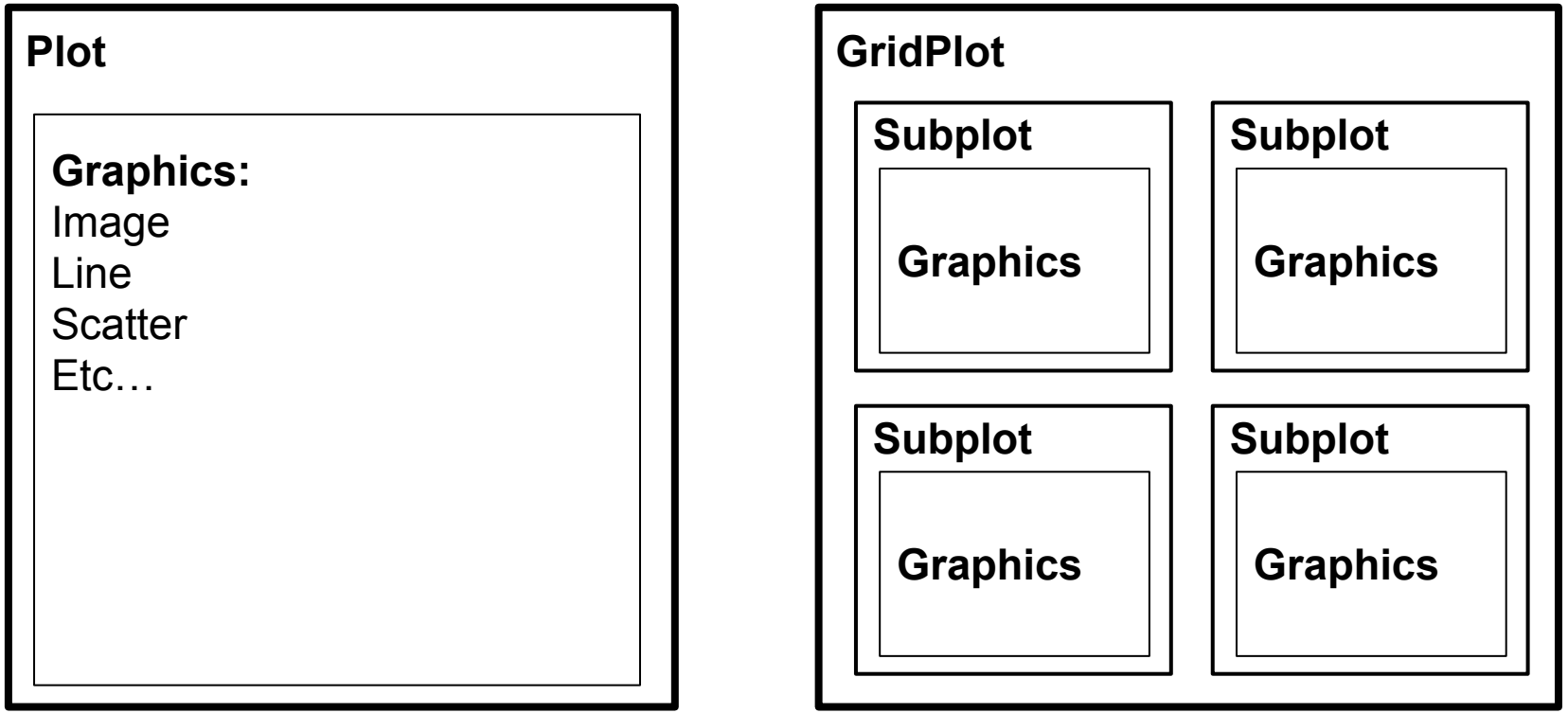
* mostly FYI, you don't need to memorize all this :)

1. Graphics - objects that are drawn
    a. Image, Line, Scatter, Heatmap, Histogram
    b. Collections - LineCollection, LineStack (ex: neural timeseries data)
    c. Interactions

2. Layouts
    a. **Plot** - a single plot area
    b. **GridPlot** - a grid of Subplots

3. Widgets - high level widgets to make repetitive UIs easier
    a. **ImageWidget** - n-dimensional widget for image data
        i. Sliders, support window functions, GridPlot etc.
        ii. You'll be seeing a lot of this!

fastplotlib API



Roadmap for 2023

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

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

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

Talk to any of us during breaks, lunch etc. if you're interested!

- **Caitlin - fastplotlib core developer**
- **Arjun - also familiar with fastplotlib**

Approach us for other visualization questions or help too!