

The background of the slide features the official seal of the University of Texas at Austin. The seal is circular, with a gold border containing the Latin motto "DISCIPLINA PRÆTORIUM CIVITATIS TEXAS" in gold capital letters. Inside the border is a gold shield. At the top of the shield is an open book. In the center of the shield is a five-pointed gold star. A gold laurel wreath encircles the star. The text "THE UNIVERSITY OF TEXAS AT AUSTIN" is written in gold around the inner edge of the seal.

# Integrated Data Acquisition

PyHEP 2020

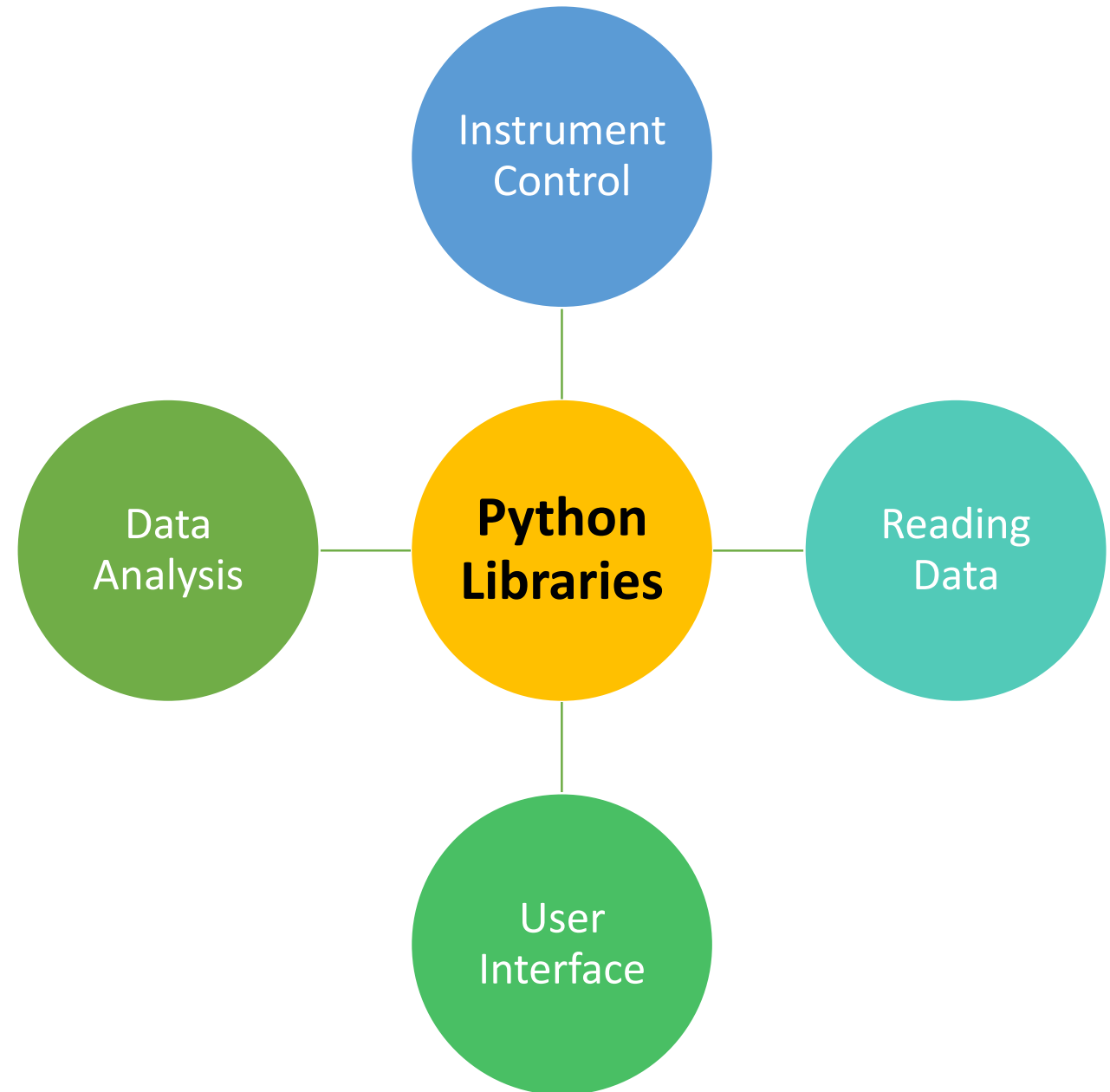
C. D. Burton

University of Texas at Austin

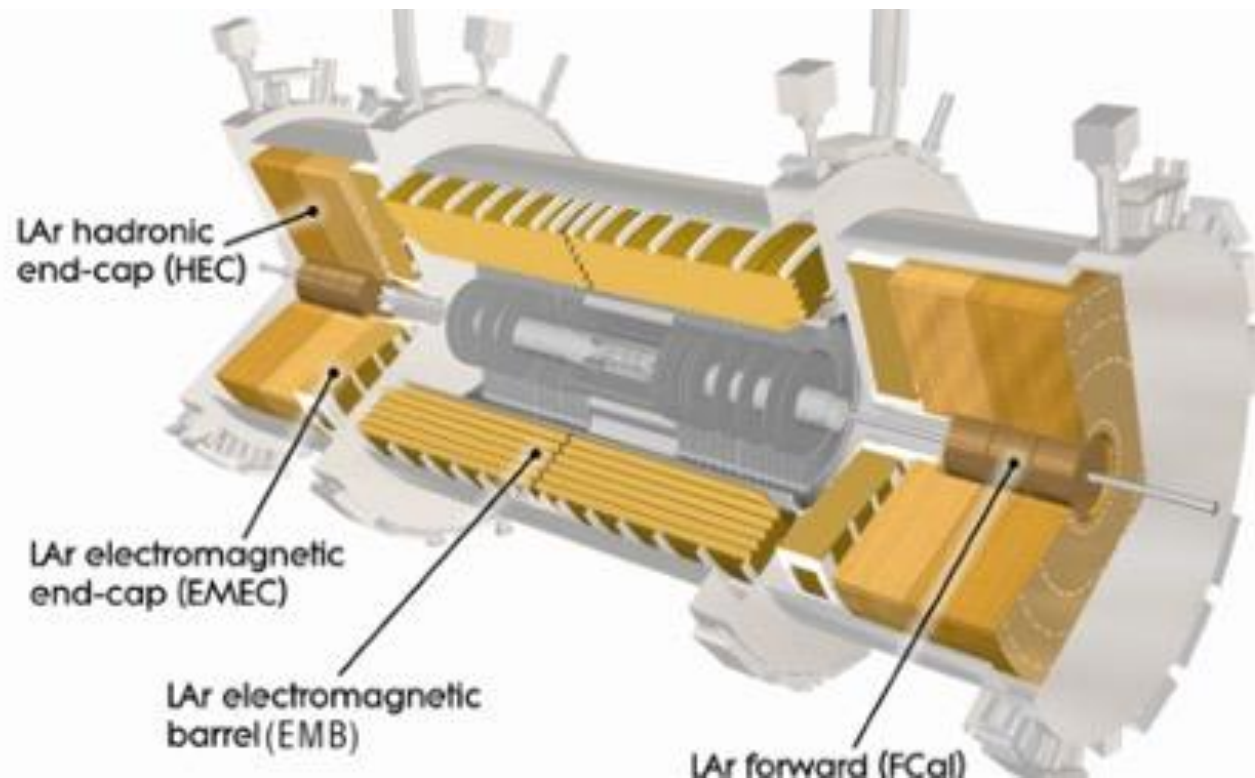
17 July 2020

# Outline

- What do I mean by “integrated”? With the help of stable and simple Python libraries, you can:
  - Control and operate the equipment on your test bench.
  - Read and capture data from your research apparatus.
  - Provide a usable graphical interface for the experimenter.
  - Analyze your data.

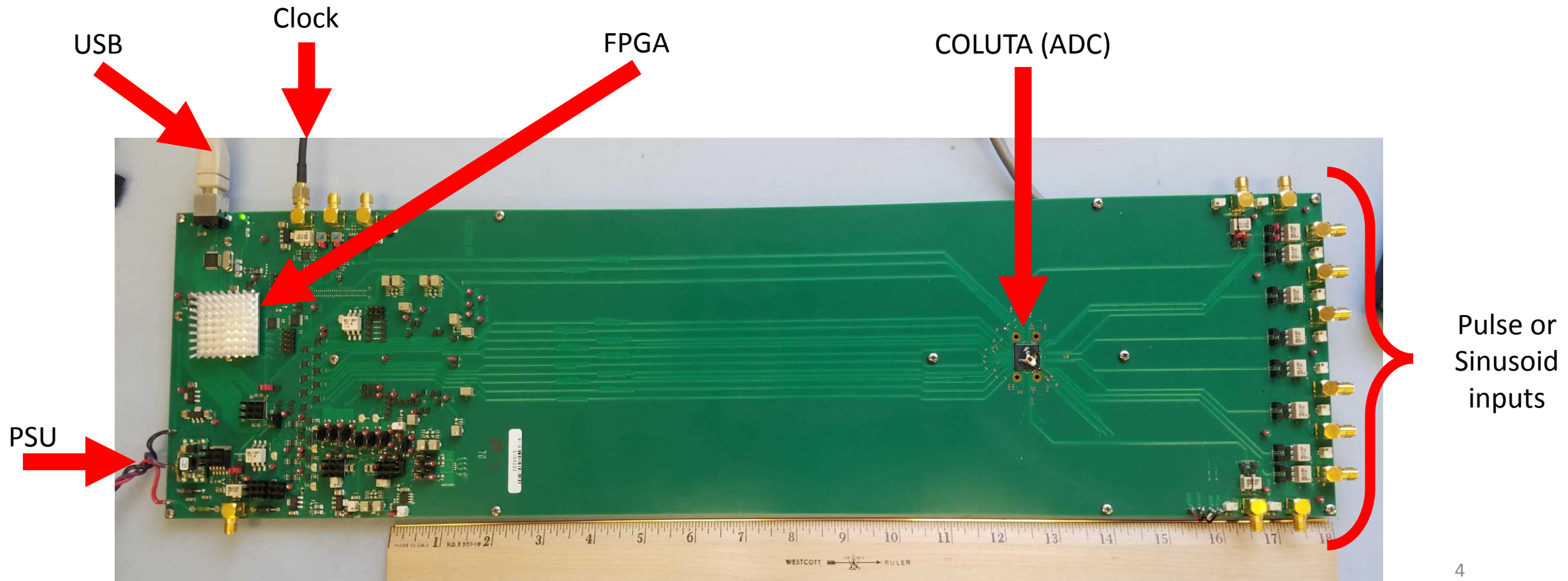


# 60 Seconds of Background: The Plot



- Prototype electronics for the ATLAS Liquid Argon (LAr) Calorimeter in the HL-LHC era are being developed.
- Need a high-speed, wide-range, radiation-hard analog-to-digital converted (ADC) being developed for signals.

# 60 Seconds of Background: The Cast





# 60 Seconds of Background: The Stage

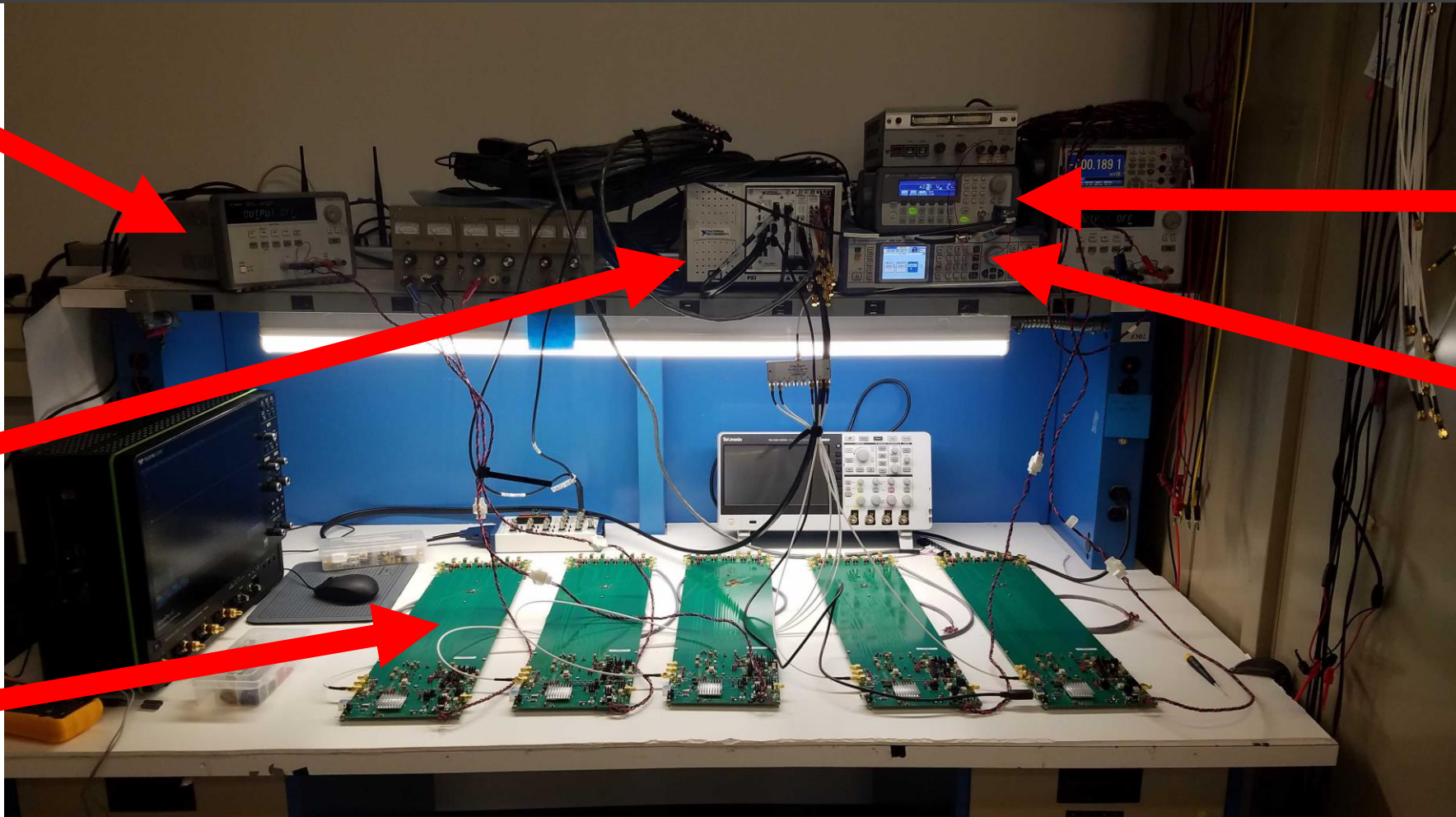
Power Supply

Computer

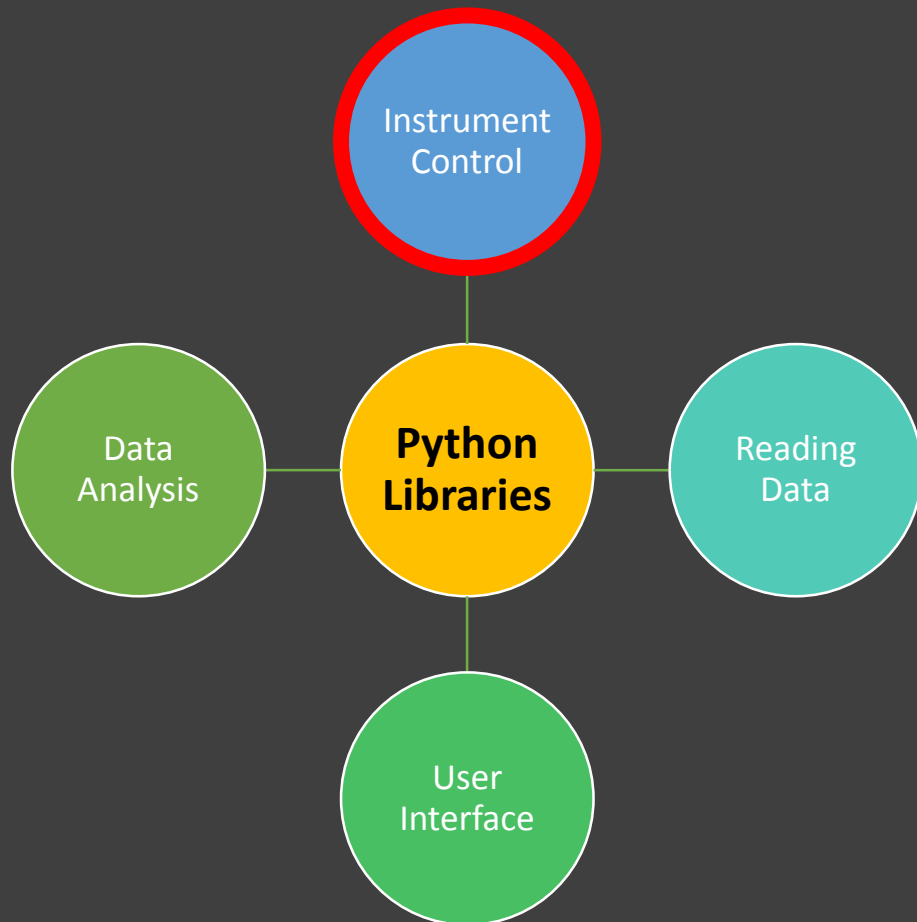
Test Board

Signal  
Generator

Clock  
Generator

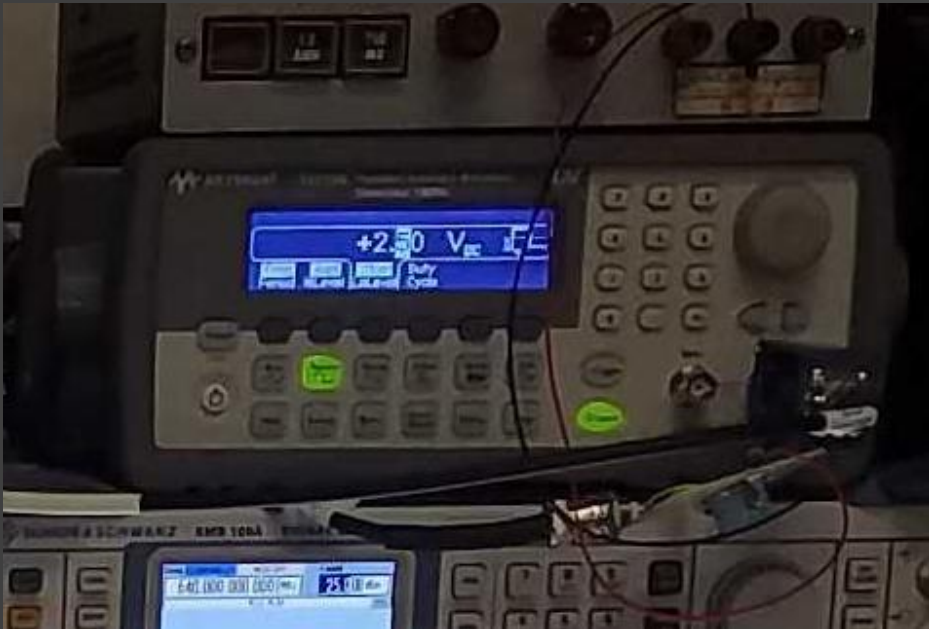


# Controlling the Test Bench



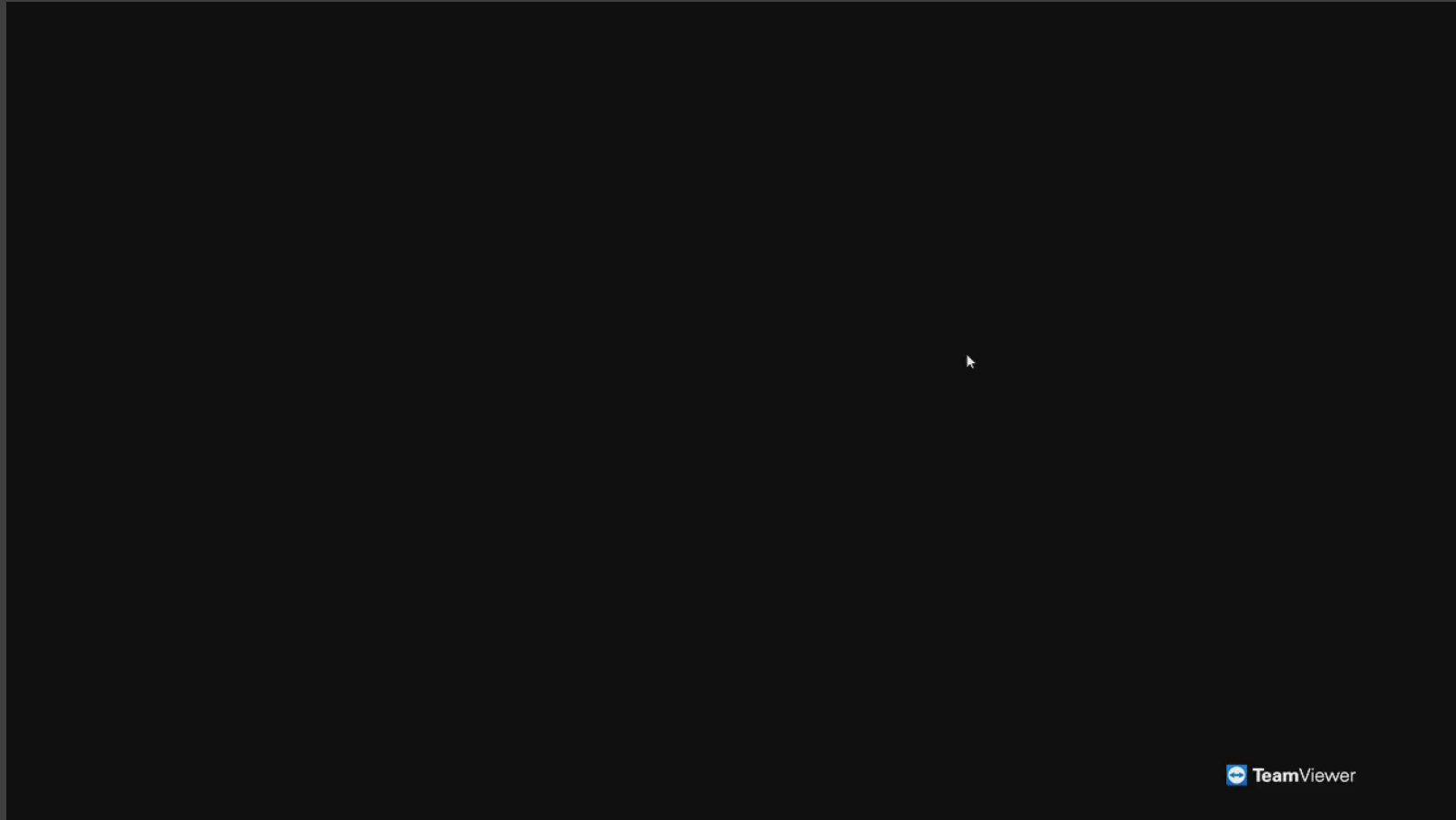
- Nearly every modern lab device on the market offers digital control through GPIB, LAN, and/or some other standard.
- This allows for both automated testing, as well as remote control.
- And with python, this can often be a rather trivial amount of effort.

# Controlling the Test Bench



- First, we will try to control our signal generator with **PyVISA**
- Physically connect the device to the computer with an Ethernet cable.
- Tasks
  1. Connect to device with TCP.
  2. Read the current state.
  3. Change the input voltage and frequency.
  4. Verify the changes.

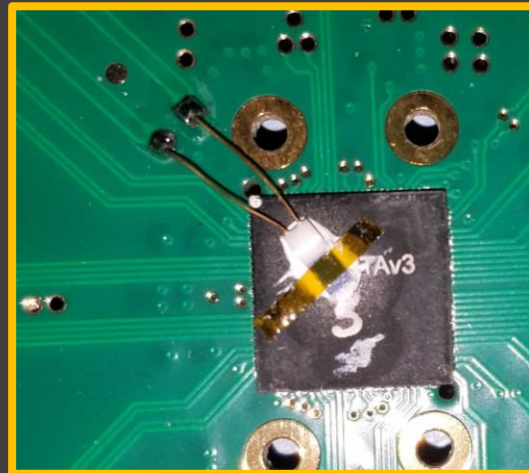
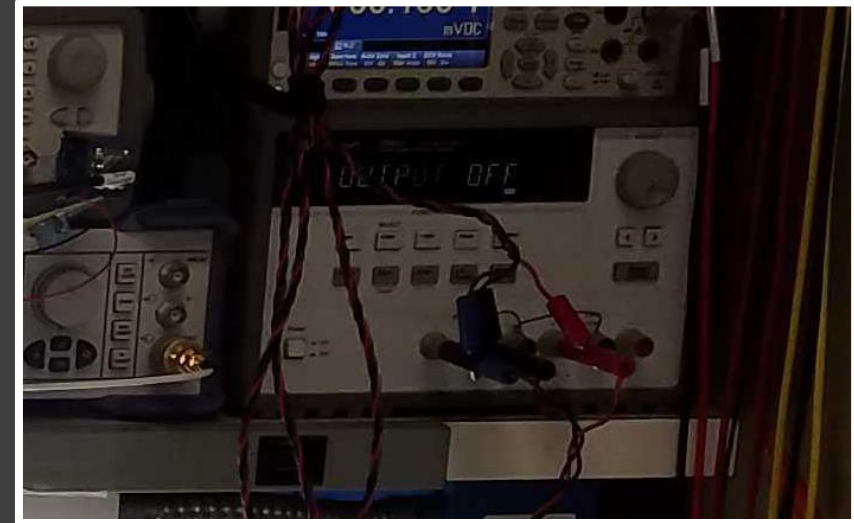
# Controlling the Test Bench





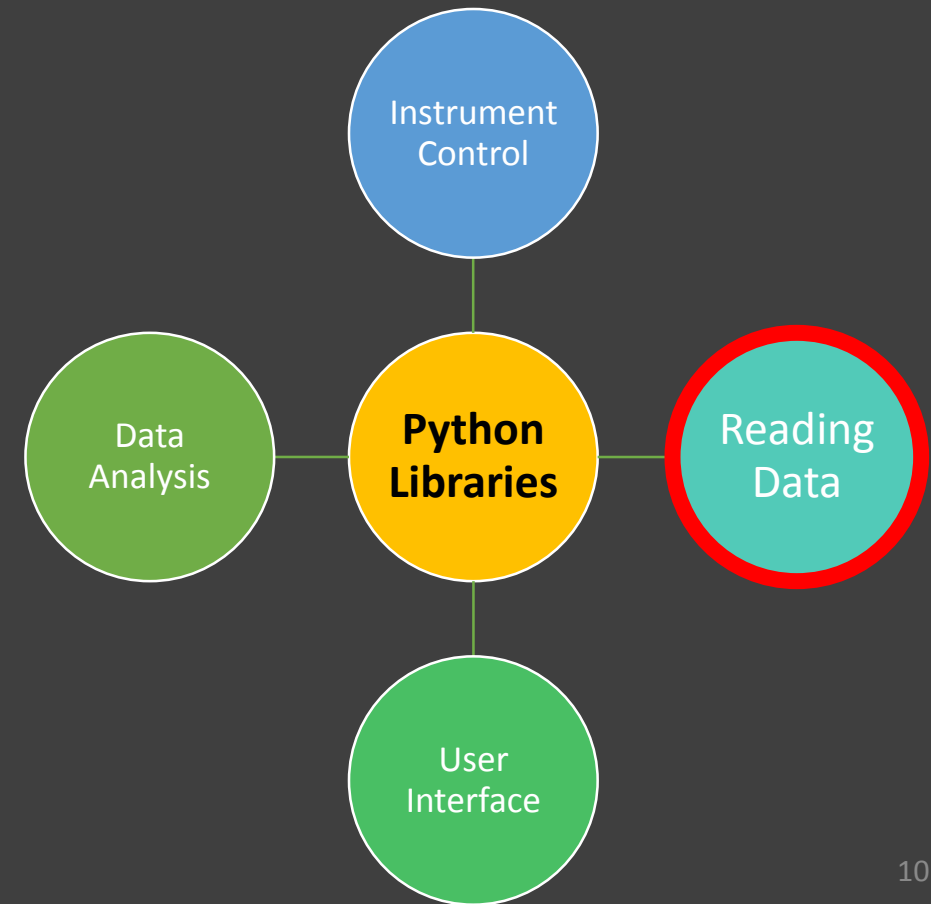
# Controlling the Test Bench

- The same process was done to control the power supply.
- For example, if your system gets into some strange state, and you need to power cycle the system, you can perform this task remotely.
- Additionally, we took readings from the temperature sensor using Keysight's scripts.

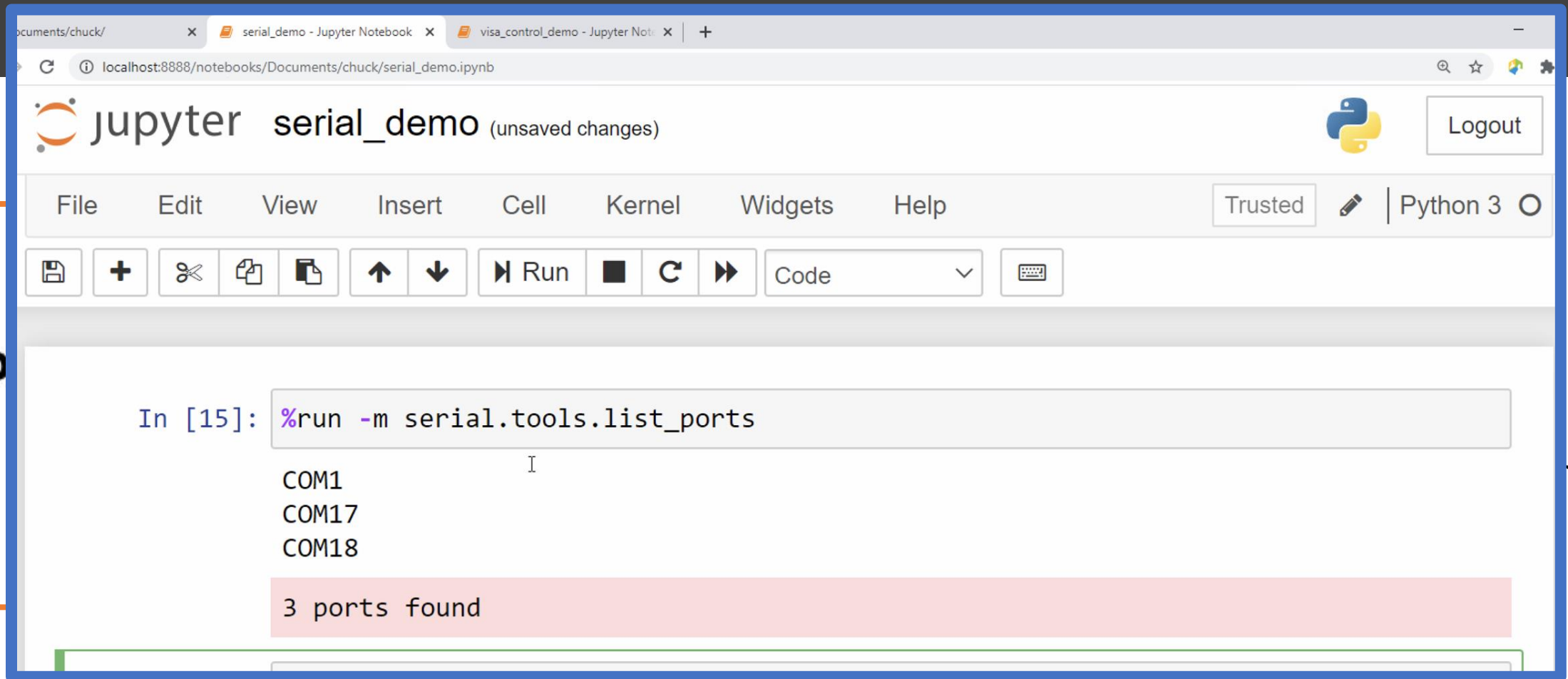


# Reading Data

- Communicating with your USB devices is extremely straightforward with ***pySerial***.
- Encapsulates access for the serial port.
- Includes backends for all operating systems, automatically choosing the appropriate one.
- The developer-facing API remains consistent irrespective of backend.
- Simple file-like access with `read()` and `write()` functions.



# Reading Data



The screenshot shows a Jupyter Notebook interface in a web browser. The browser tabs include 'documents/chuck/', 'serial\_demo - Jupyter Notebook', and 'visa\_control\_demo - Jupyter Notebook'. The address bar shows 'localhost:8888/notebooks/Documents/chuck/serial\_demo.ipynb'. The Jupyter header includes the logo, the name 'serial\_demo (unsaved changes)', a Python logo, and a 'Logout' button. The menu bar contains 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The toolbar includes icons for saving, adding cells, cutting, copying, pasting, undo, redo, running, and a dropdown menu currently set to 'Code'. The code cell contains the command 'In [15]: %run -m serial.tools.list\_ports'. The output of the command is displayed below the code cell, showing 'COM1', 'COM17', and 'COM18' on separate lines, followed by a pink-shaded box containing the text '3 ports found'.

```
In [15]: %run -m serial.tools.list_ports
```

```
COM1
COM17
COM18
```

```
3 ports found
```

# Reading Data

```
# Block an unlocked thread before writing to chip
block = Thread.block()
coluta.logger.addTraceback("{0} <- {1}".format(port, " ".join(messageList)))
fifo.write(BAMessage)
time.sleep(0.01)

# If an unlocked thread was blocked, release the lock
Thread.block(block)
return True
```

- For this set of commands, what does the developer see? Just `write()`.
- Waiting for this task to complete will hold the GUI, read/write operations are performed in a background thread.
- For safety, I have implemented mutex access to the Serial objects.

# Reading Data

- After creating the serial connection, we pass commands that the board's firmware understands.
- Then read back any data in the USB buffer.

```
Anaconda Powershell Prompt (Anaconda3)
Number of samples requested 1024
B <- fe 00 00 00 00 ff
B <- fe 00 00 00 04 ff
B <- fe 1b 00 08 40 ff
A -> ['00', '92', 'ca', '10', 'cc', '10', 'c2', '10', 'b',
      'cb', '10', 'ca', '10', 'c3', '10', 'bd', '10', '00',
      'cc', '10', 'c1', '10', 'bd', '10', '00', '94', 'bf', '4',
      '10', 'be', '10', '00', '95', 'c0', '40', 'b5', '40',
      '10', '00', '96', 'bf', '40', 'b5', '40', 'b4', '40', 'c',
      'bf', '40', 'b4', '40', 'b5', '40', 'd3', '40', '00',
      'b4', '40', 'b2', '40', 'd2', '40', '00', '00', '00', '0',
      '40', 'd2', '40', '00', '00', '00', '00', '00', '00',
      '40', '00', '00', '00', '00', '00', '00', '00', '00', '0',
      '00', '00', '00', '00', '00', '00', '00', '00', '00', '0',
      '00', '00', '00', '00', '00', '00', '00', '9e', 'ca', '10',
      '00', '00', '00', '00', '9f', 'c9', '10', 'cd', '10', 'c
```

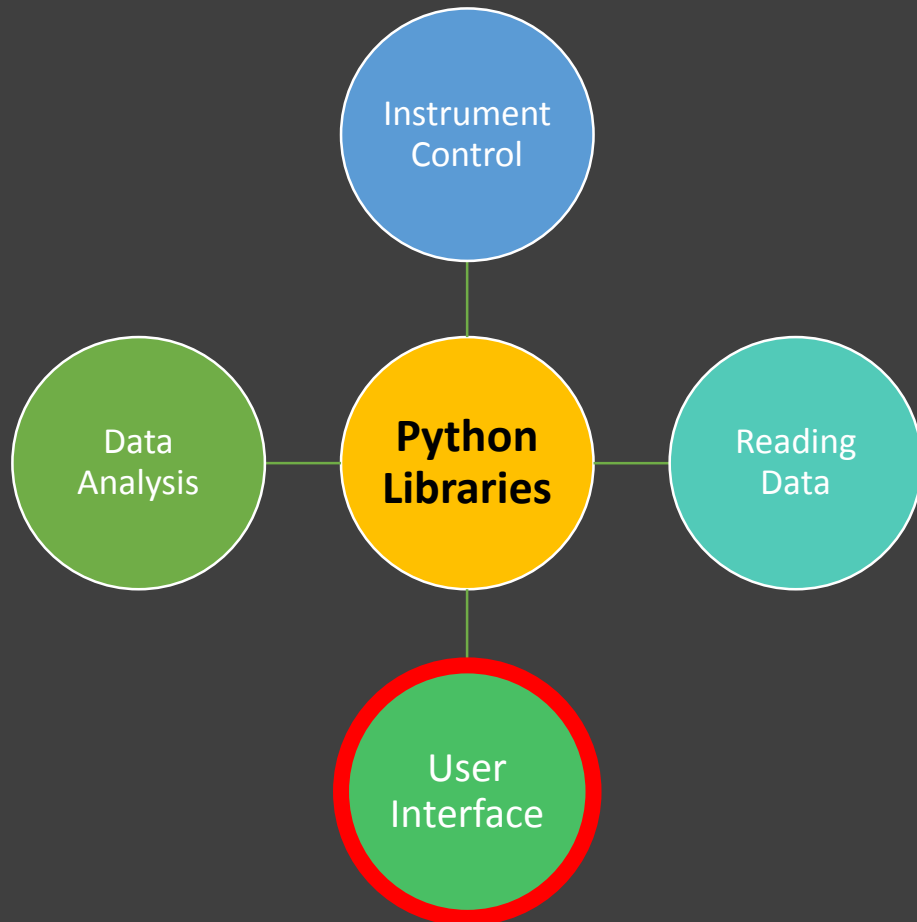


# Reading Data

- Tells the FPGA to listen
- Read 1024 points
- Move data to buffer A
- Read buffer A

```
Anaconda Powershell Prompt (Anaconda3)
Number of samples requested 1024
B <- fe 00 00 00 00 ff
B <- fe 00 00 00 04 ff
B <- fe 1b 00 08 40 ff
A -> ['00', '92', 'ca', '10', 'cc', '10', 'c2', '10', 'b',
      'cb', '10', 'ca', '10', 'c3', '10', 'bd', '10', '00',
      'cc', '10', 'c1', '10', 'bd', '10', '00', '94', 'bf', '4',
      '10', 'be', '10', '00', '95', 'c0', '40', 'b5', '40',
      '10', '00', '96', 'bf', '40', 'b5', '40', 'b4', '40', 'c',
      'bf', '40', 'b4', '40', 'b5', '40', 'd3', '40', '00',
      'b4', '40', 'b2', '40', 'd2', '40', '00', '00', '00', '0',
      '40', 'd2', '40', '00', '00', '00', '00', '00', '00', '0',
      '40', '00', '00', '00', '00', '00', '00', '00', '00', '0',
      '00', '00', '00', '00', '00', '00', '00', '00', '00', '0',
      '00', '00', '00', '00', '00', '00', '00', '00', '00', '9',
      '00', '00', '00', '00', '00', '00', '00', '9e', 'ca', '10',
      '00', '00', '00', '00', '9f', 'c9', '10', 'cd', '10', 'c
```

# User Interface



- In many cases, your users may not be comfortable with doing their own software development (nor should they).
- For our chip, we could not rely on users being able to write scripts.
- Enter, ***PyQt***.

# User Interface

**Page tabs**

**Data-taking**

**Data-storage**

**Immediate Display**

**Serial Connections**

The screenshot displays the COLUTA software interface with several key sections:

- Page tabs:** Located at the top, including Control, Radiation Run, Instrumentation, Global, and Channels 1 through 8.
- Board Connection:** On the left, showing COM17 and COM18 ports as 'Connected'.
- Data Acquisition:** In the center, featuring buttons for 'Take Coluta Samples', 'Take AD9650 Samples', 'Take AD121 Samples', 'Take Dual Samples', 'Take Single Histogram', and 'Take Samples Repeat'. It also includes a 'Test Button' and 'Load Samples' button.
- Data-taking controls:** On the right, with fields for 'Number of Samples' (1024), 'Do FFT' (unchecked), 'Save HDF5' (checked), 'Save Txt' (unchecked), 'Save CSV' (unchecked), 'Plot Channel' (frame1), 'Increment Pulse Delay' (unchecked), 'Reset Pulse Delay' (unchecked), 'Repeat Data Taking' (5), and checkboxes for saving individual channels (FRAME1?, Ch1?, Ch2?, Ch3?, Ch4?, FRAME8?, Ch5?, Ch6?, Ch7?, Ch8?).
- Data Display:** On the far right, showing a 2D plot of ADC Counts (34000 to 40000) versus D [dB] (-75 to 0).

# User Interface

User-friendly checkboxes, drop-down menus, and text entry. All are converted to commands to the chip.

COLUTA

File

Control Radiation Run Instrumentation Global Channel 1 Channel 2 Channel 3 Channel 4 Channel 5 Channel 6 Channel 7 Channel 8 Miscellaneous

Clock Invert 640 ☒

Clock Delay 640 3

BGP Trim Up 5

BGP Trim Down 3

BGP External Start ☐

BGP Select IBIAS BGP

BGP Select Amp Source BGP

BGP Select VREFP RDAC

BGP Select VREFN RDAC

BGP Enable VREFP ☐

BGP Enable VREFN ☒

BGP RDAC CODE VREFP 5

BGP RDAC CODE VCM 23

BGP RDAC CODE VREFN 58

BGP FDBK CODE VREFP 45

TS Disable ☐

TS TP12 Select Global TP

TS TP12 Mux Ch2 or {TP1=BGP\_VCM; TP2=50uA sink}

TS DOUT Select Ch1 or {TP1=GND; TP2=GND}

TS DI Select Ch2 or {TP1=BGP\_VCM; TP2=50uA sink}

TS DI Pass Ch3 or {TP1=GND; TP2=GND}

TS DI Logic 128

TS DI IO 128

TS IOA Swin Front-end

TS IOA FEUG ☐

TS IOA FEG x(16/16)+1

TS IOA BEG x1/1

TS IOA FE Enable ☒

TS IOA BE Enable ☒

TS IOA Cap 8

TS IOA Bias 5

TS XS nRST Reset

TS XS VT 2

TS XS VBOT Ground High-Z

TS XS Thru High-Z

TS XS SEU Select Open

TS XS Reset Mode BITMAN

TS XS Reset Manual 1

TS XS LR TAP1 2

TS XS LR TAP2 8

TS XS LR PS 156.25 kHz

TS XS LR Mode Internal

TS XS GT Mode Internal

TS XS GT Manual Hold Count

TS XS GT Len 0.8388 s

TS XS CMP Pol VIN>VT

TS XS AMP Pol IOA\_VOUTN

TS MADC nRST Active

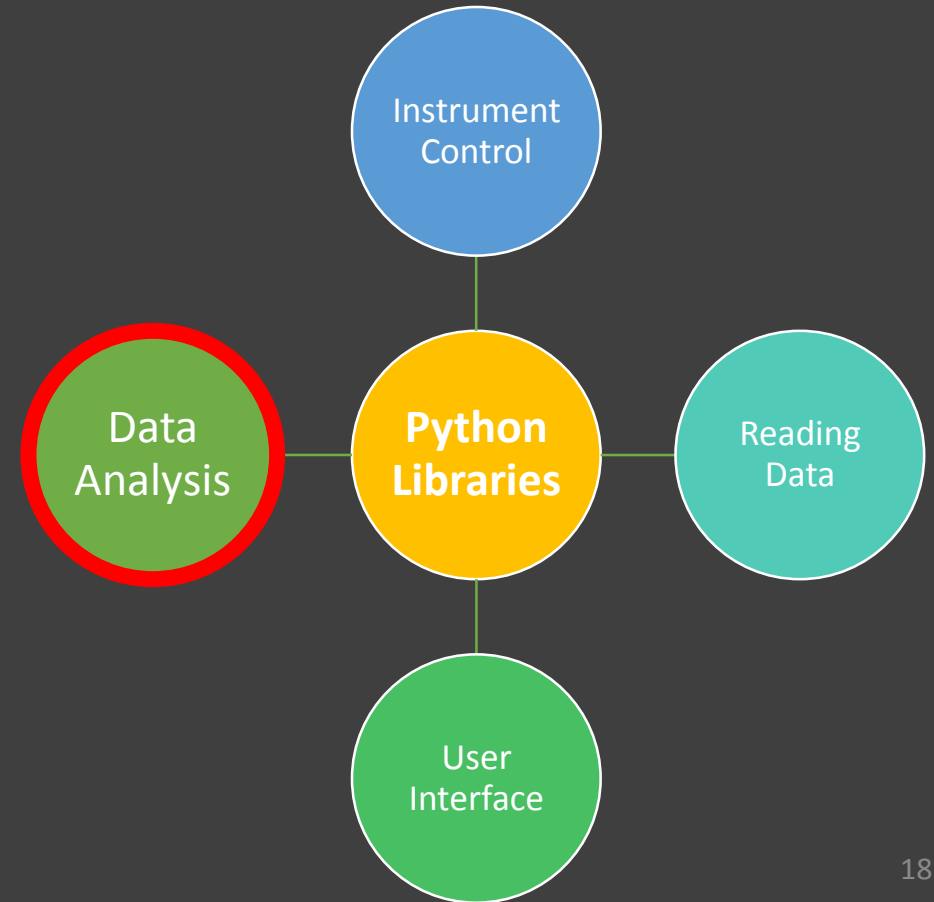
TS MADC CLK PS divide by 1

Configure Global

Ready

# Data Analysis

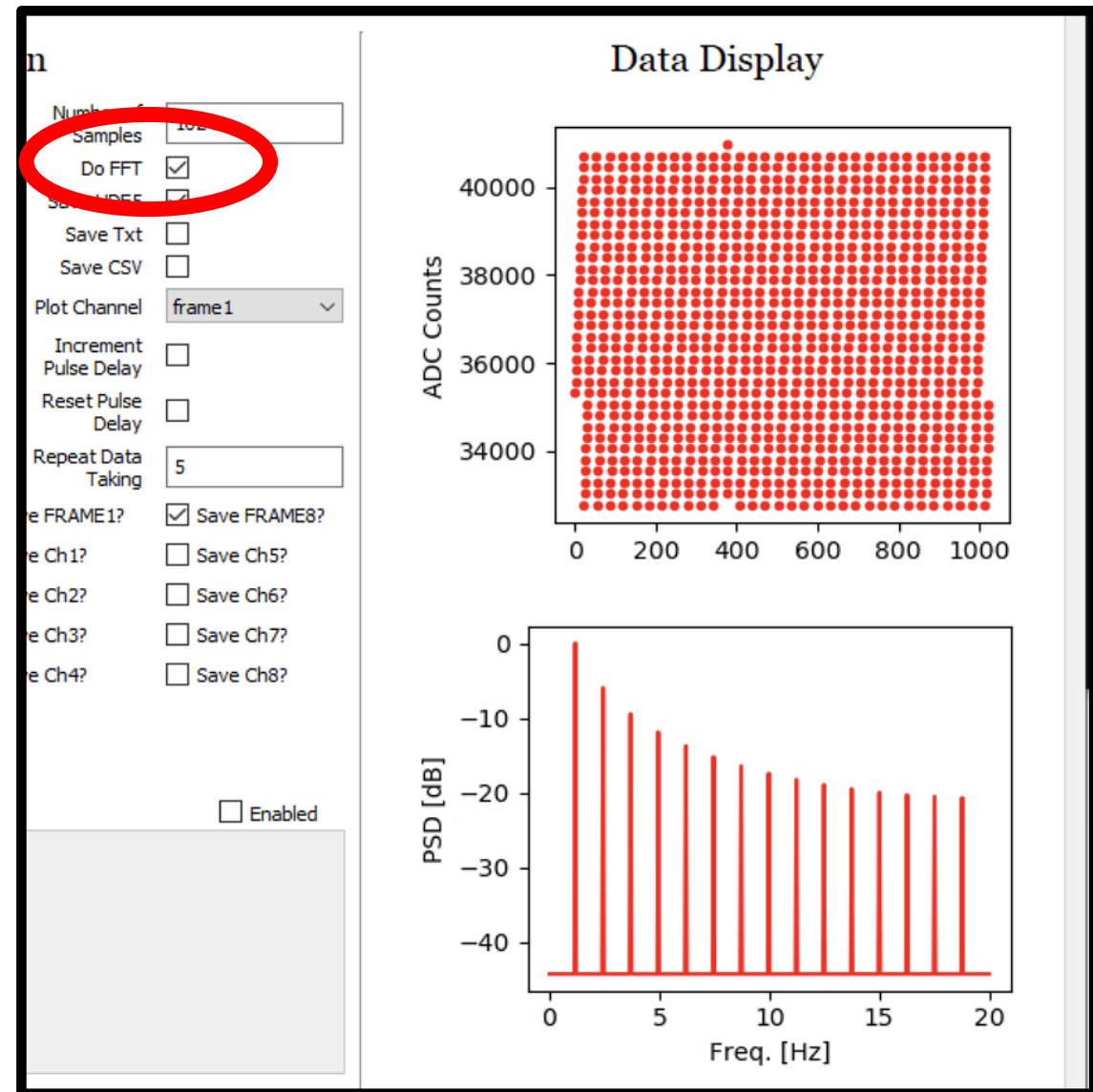
- Analysis should be performed in multiple stages at different levels of detail.
1. Immediate feedback to the user while experimentation is in progress.
  2. Detailed analysis from an organized set of data.





# Data Analysis

- When experimenting, it is ideal—and perhaps even necessary—to have rapid feedback for preliminary analysis.
- Not only can we use matplotlib to show preliminary plots of the data, but we can also perform simple analysis.
- In this case, we use *scipy.fft* to perform the check.



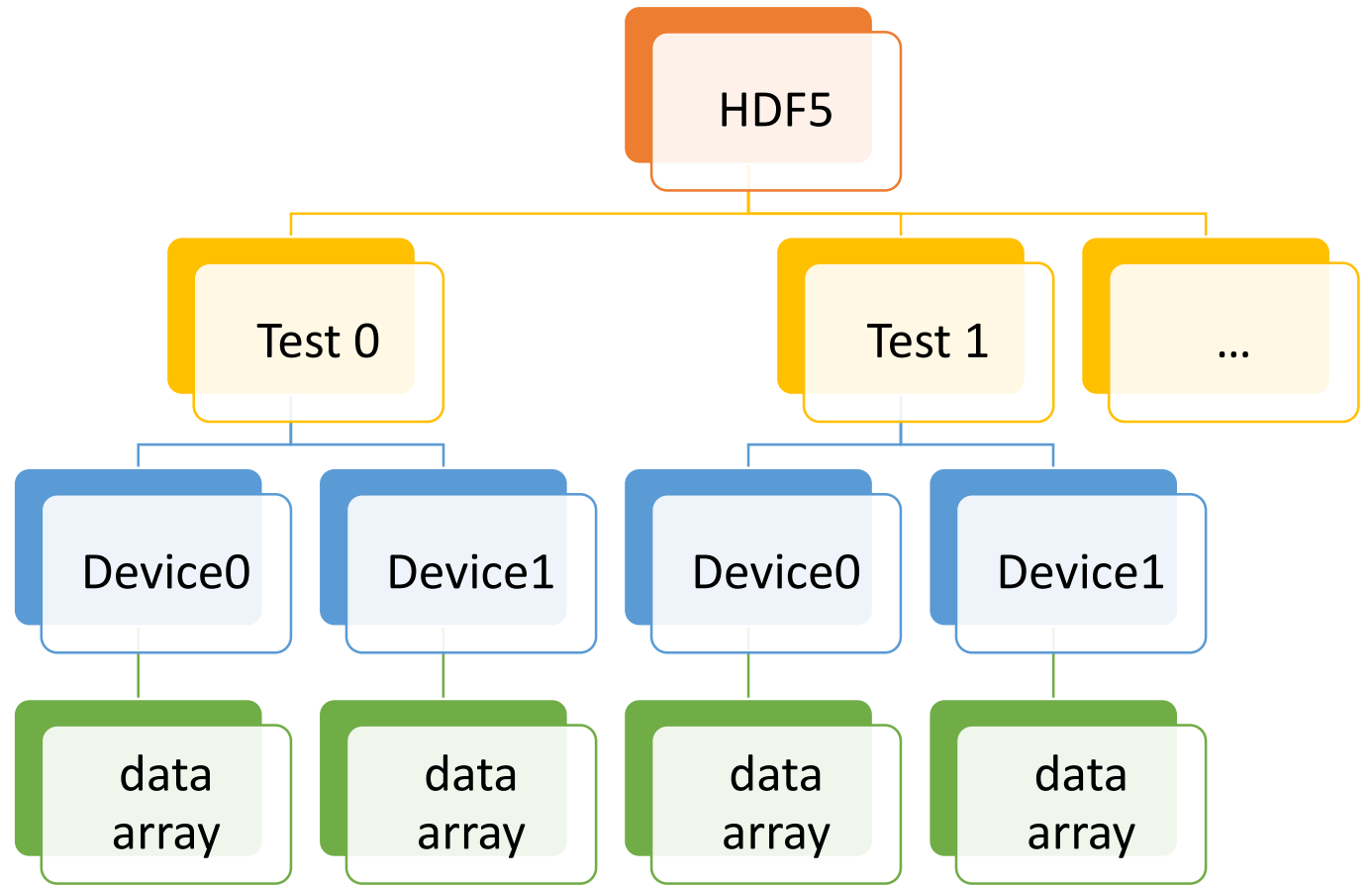
# Data Analysis

- For long periods of data-taking, it makes sense to have continually-updating plots.
- Yet another straightforward [implementation with matplotlib + PyQt for dynamic plots.](#)



# Data Analysis

- Different tasks call for different tools: for complicated collider-physics event data, things like ROOT (and all the surrounding python infrastructure—pyROOT, uproot, etc.) may be necessary.
- What if your data is a small set of single-stream measurements?
- HDF5 may be an answer.



# Data Analysis

- There's a package for *that, too!*
- **H5PY** is a user-friendly tool to read and write files in the HDF5 file format.
- Navigate the data hierarchy in a python-dictionary way.
- Demo: [read hdf5.ipynb](#)

# Summary

With python libraries everything-in-one.

1. Control your equipment with **PyVISA** (or another manufacturer-provided library)
2. Read data over USB with **pySerial**.
3. Build an easy-to-use UX in **PyQt**
4. Analyze data in real-time as well as store data in friendly formats

ColutaGUI.py 109 KB

```
1  """Main class for the COLUTA GUI
2
3  name: ColutaGUI.py
4  author: C. D. Burton
5  email: burton@utexas.edu
6  date: 7 August 2018
7  """
8  # PyQt libraries
9  from PyQt5 import uic,QtWidgets,QtCore,QtGui
10 from PyQt5.QtCore import QThread,QTimer
11 # Python libraries
12 import numpy as np
13 import binascii
14 import time
15 import glob
16 import os,sys
17 # Custom libraries
18 import colutaMod
19 import serialMod
20 import monitoring
21 import chipConfiguration as CC
22 import configparser
23 import dataParser
24 import Thread
25 import status
26 import instrumentControlMod
27 import pxiControl as pc
28 import psuControl
29 from datetime import datetime
30 from logger import Logging
31 from radiationTest import RadiationTest
32 from collections import defaultdict
33 from lxiControl import lxiControl
34
```

Instrument  
Control

Python  
Libraries

Reading  
Data

User  
Interface



*Questions?*

