SlowDash
コントロールとモニタのためのウェブツール

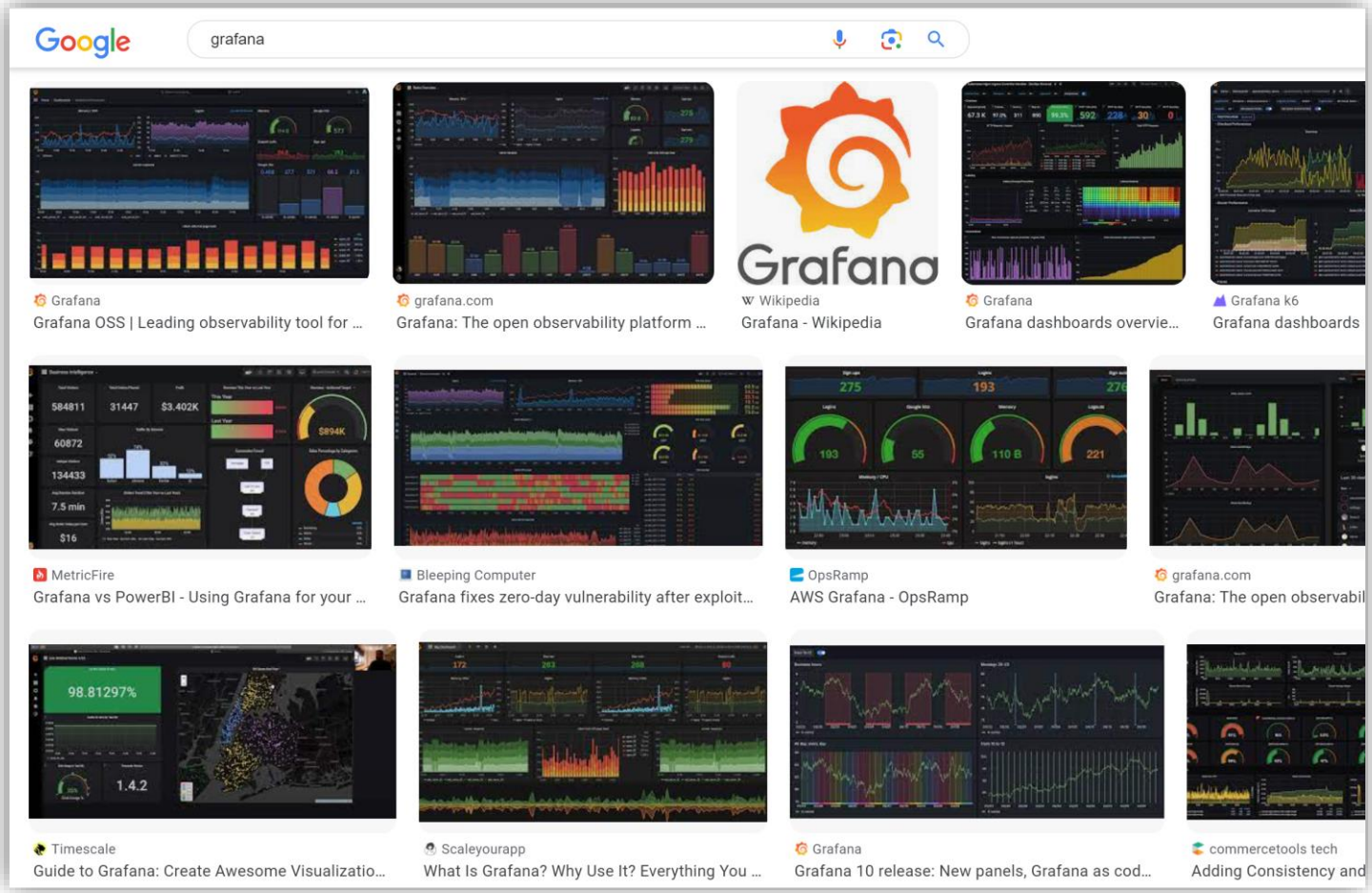
榎本　三四郎
ワシントン大学

計測システム研究会2024, 東京大学, 2024年11月18日

Grafana でかっこいいダッシュボードを作れ と言われた

Slow-Controls 用ダッシュボードを作る.
データを取る部分はすでに稼働していたので, UI だけあれば良かった

Google 検索による Grafana のイメージ

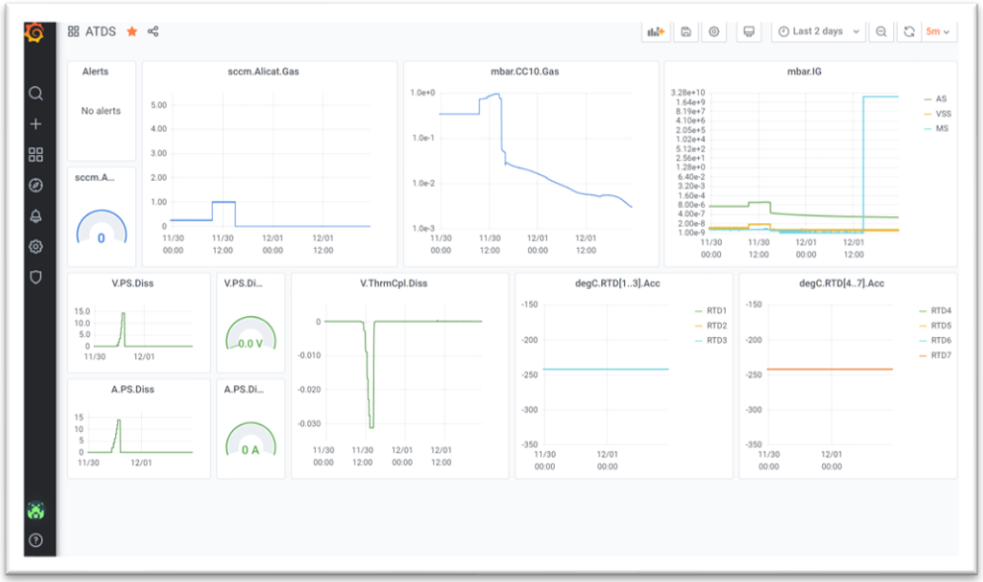


Grafana とは

- データーベース上にあるデータ(主に時系列)を見える化する Web ベースのソフトウェア
- マウスクリックでダッシュボードを作成して共有できる

3週間がんばってみた

- 構成の決まった表示には良い. 対話的なデータブラウジングはほぼ無理
- 科学データ解析用途には作られていないと思う (データサイエンス用途 ☺)



ダークモードはかっこいいけど
絵をスライドに貼れないから
ライトモードにしてみた
⇒ Grafana 台無し…

かっこいいプロットを眺めてるから先に進まない

- 表示を簡単に変えられない. (範囲, 対数軸, ...)
- 一時的プロットを作るのが大変.
- 基本的にデータ値にアクセスできない. ダウンロードもできない.

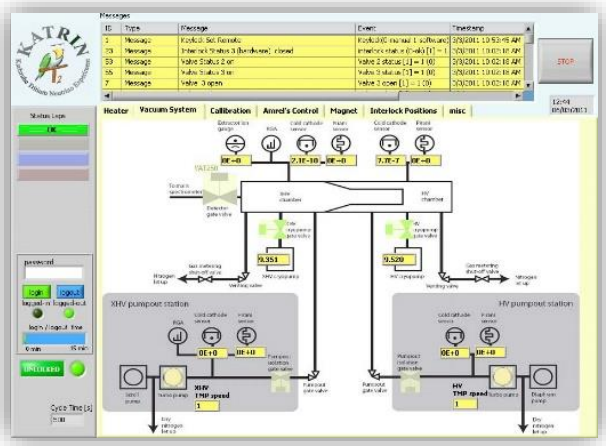
目的に合っていない気がする

- ヒストグラム, 誤差バー付きグラフ, 散布図, etc が素直でない. (地図塗りは簡単)
- 機器制御ができない. コマンドも送れない.

欲しいものリスト

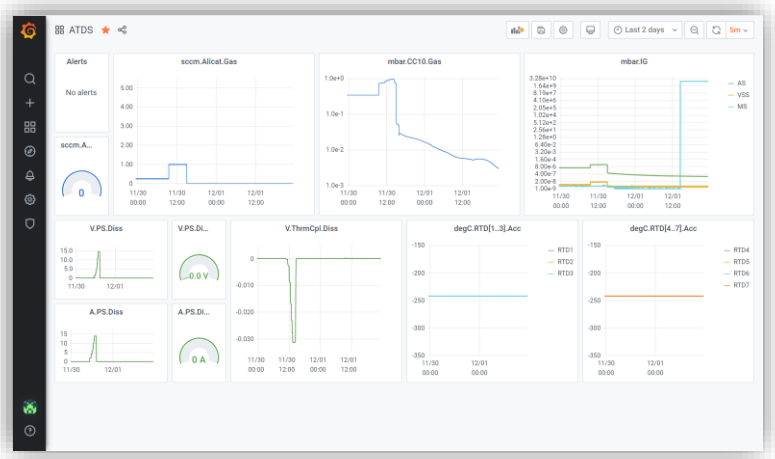
LabVIEW 風のコントロールパネル

（でもグラフィカルプログラミングは嫌）



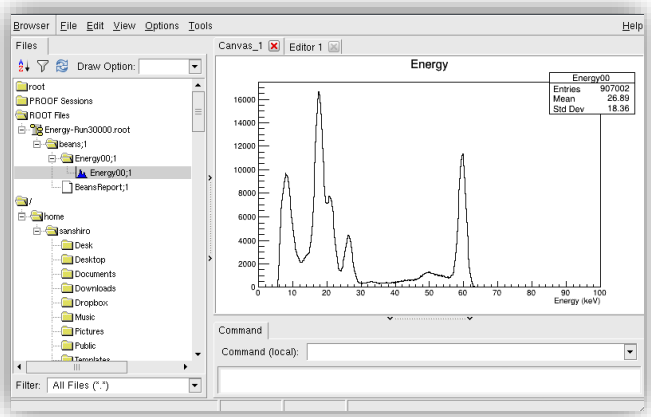
Grafana 風のデータブラウザ

（でも見るだけでは不足）



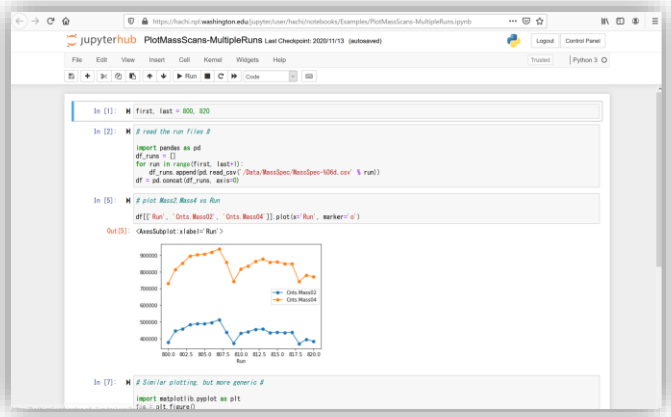
ROOT 風のデータ型

（でも ROOT は嫌い）



Jupyter 風のその場思い付き解析

（でも実時間処理には向かない）



開発経緯・状況

22 年11月 サンクスギビングの連休中に Grafana の置き換えを模索

23 年5月 SPADI-A WG3 に居候開始

24 年1月 GitHub 公開: Version “Nisqually”

- データベース中のデータの Grafana 風表示
- LabVIEW 風のデータ表示（表示だけ）
- いろいろな実験的実装

24年7月 “Snoqualmie”

- コントロールの Python スクリプティング

昨日 “Skykomish”

- 共通コントロール部品の実装
- 複雑なコントロールの実装方法の模索
- Jupyter 接続

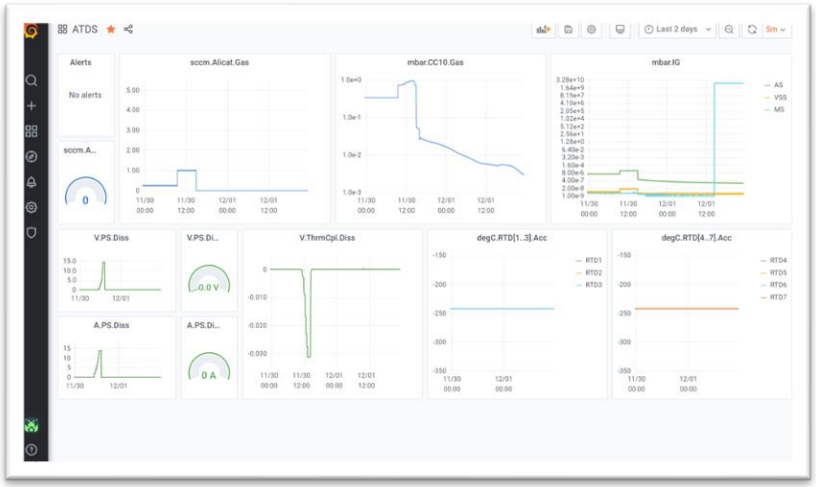
今回の新しい部分

このあと

- 埋め込みデータ加工・データ変換（テーブルからグラフやヒストグラム作成とか）
- かっこいいレイアウトとビジュアル, 数千チャンネルの表示とか
- アラーム検出・通知・認知・マスク・リマインド・履歴管理, 障害予測(?)

Grafana のいいところ

良いと思ったところ



- ダッシュボードをマウス操作だけで作成できる（サーバーにログインしなくて良い）
- 作ったダッシュボードを保存して共有できる
- ダークモードにするとかっこいい

- 基本的に全てプラグインで，それを画面上で組み合わせる感じ
- 表示時間範囲の操作感もいい感じ

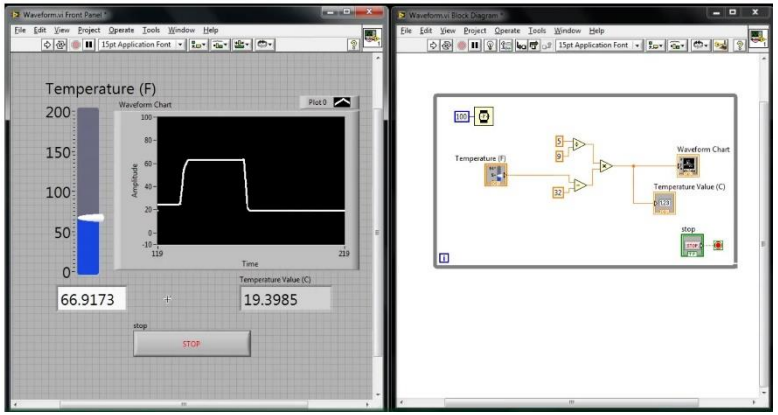
足りないと思ったところ

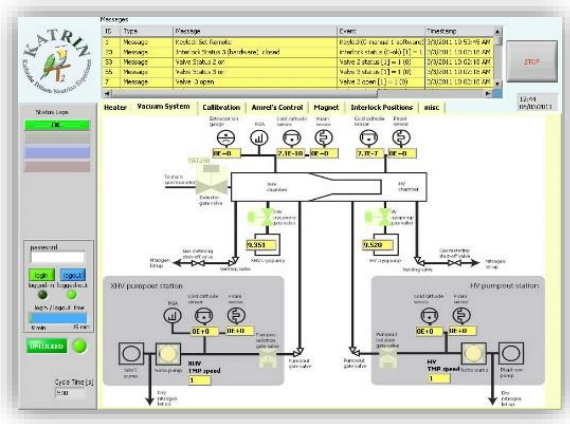
- ダッシュボードなら装置全体の状態を視覚化したい
- 表示した装置は操作したい
- 表示したデータは解析したい

LabVIEW のいいところ

良いと思ったところ

- 表と裏がある
- 裏のロジックもユーザがその場で作成・編集ができる





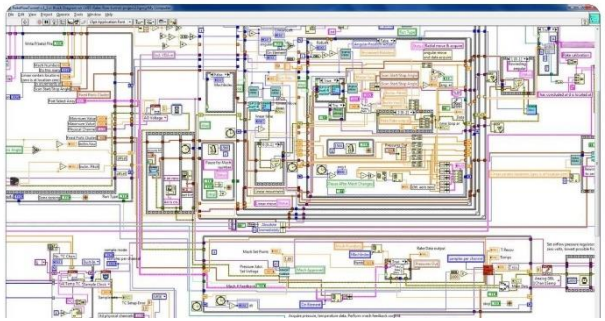
- データ要素の「空間断面」(現在値の平面配置)的な表示
- コントロール部品にロジックが張り付いている感じ

足りないと思ったところ

- ロジックは字で書きたい．図はいずれこうなる ⇒
  (回路を字で書きたいから HDL が普及したんだと思う)

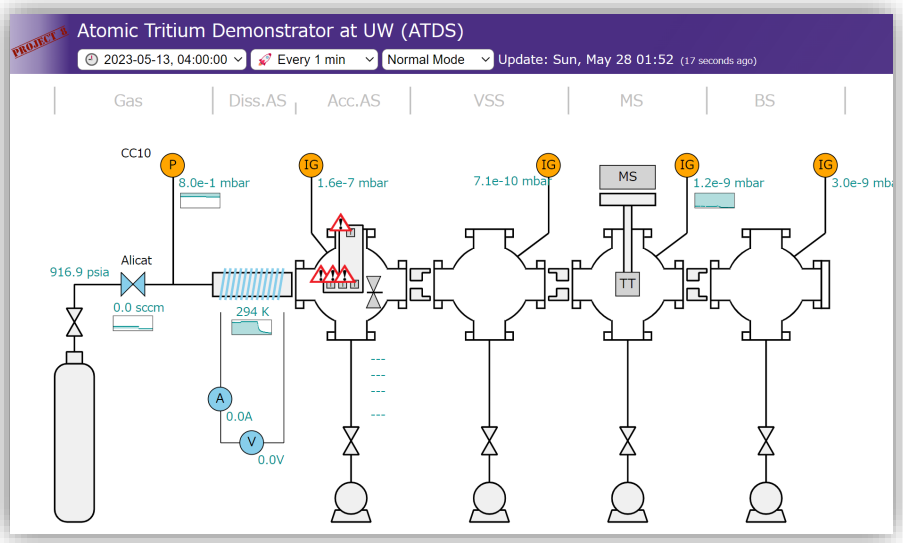
- 発見的データ解析にも向いていない

My worst nightmare

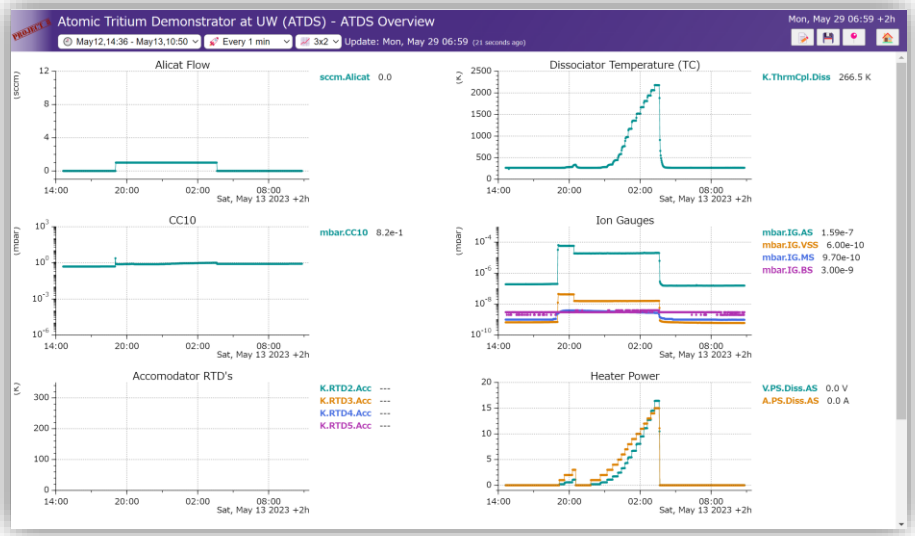


SlowDash：時間断面と空間断面のデュアルビュー

LabVIEW 風のダッシュボード（固定時間）



Grafana 風のプロット（時系列）



- 色で運転状態表示，アイコンで異常状態表示
- クリックすると時系列データを表示（右画面）

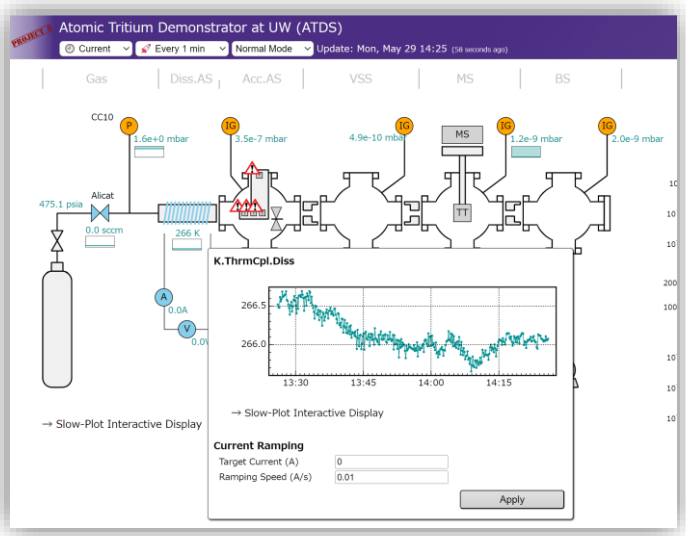
- インタラクティブプロット（ズーム，対数軸，…）
- レイアウトをマウスで作成
- 作成したページを保存，共有可
- データダウンロード

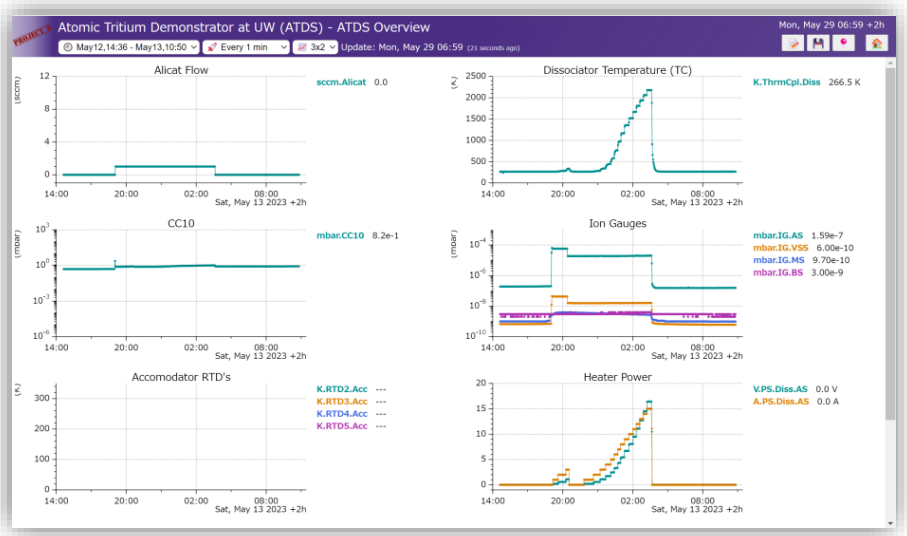
プロットの構成は URL からでもできる．作成した複雑なページを URL にエンコードすることもできる．

hachi.npl.washington.edu/~hachi/SlowDash/ATDS/slowplot.html?channel=V.ThrmCpl&time=2023-05-13,13:00

- いろいろなプロットへのリンクをあちこちに作れる
- Slack や ELOG に投稿できる

SlowDash: GUI とスクリプトのハイブリッド





変数や関数のバインド

コード生成と動的ロード

ユーザの Python
（主に制御）

```
def set_V0(V0, **kwargs):
    ramping = kwargs.get('ramping', 10)
    device.ch(0).ramp(ramping).set(V0)

def set_V1(V1, **kwargs):
    ramping = kwargs.get('ramping', 10)
    device.ch(1).ramp(ramping).set(V1)

def set_V2(V2, **kwargs):
    ramping = kwargs.get('ramping', 10)
```

ユーザの Python
（主に解析）

データベース

制御対象

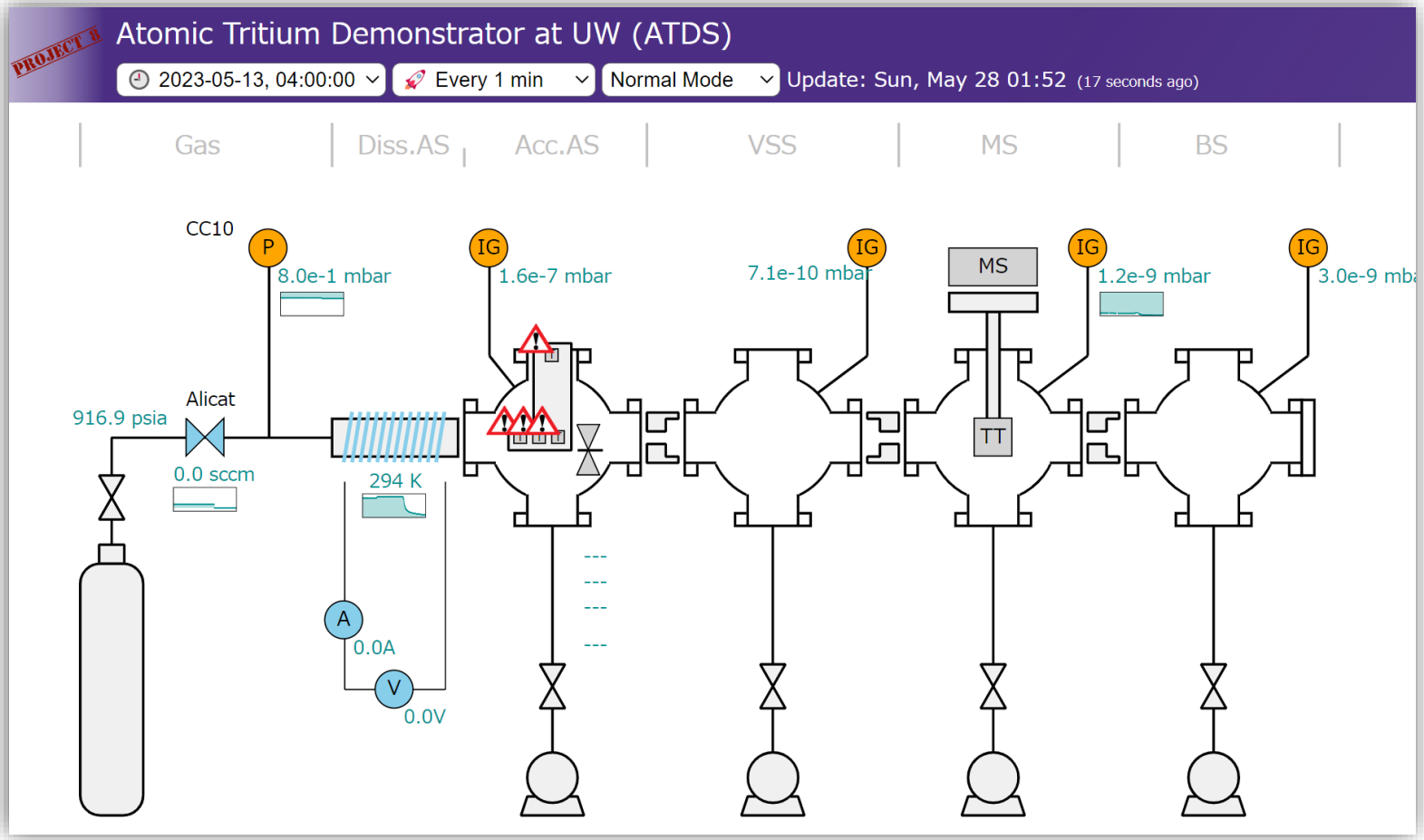
Slow Devices

DAQ Systems

ユーザのシステム

ダッシュボード

- 静止画の上にデータ要素を並べる. YAML ファイルで記述.
- 色で On/Off 状態を表示 (正常/異常ではなく)
- 最新値の数値表示と, ミニトレンドグラフ



ダッシュボード: YAML 設定ファイル例

```
viewBox: 0 0 1600 700

widgets:
  - type: image
    attr: { x: 50, y: 49, height: 503, width: 933, href: "ATDS.png" }

  - type: circle
    attr: { x: 180, y: 95, width: 40, height: 40, label: P }
    data: { channel: "mbar.Gas", "active-below": 500, format: "%.1e mbar" }

  - type: valve
    attr: { x: 130, y: 230, width: 40, height: 40, orientation: h, "data-dx": -10, "data-dy": 60 }
    data: { channel: "SCCM.Gas", "active-above": 0.1, format: "%.1f SCCM" }

  - type: circle
    attr: { x: 255, y: 430, width: 40, height: 40, label: A }
    data: { channel: "A.PS.Diss", "active-above": 0.1, format: "%.1fA" }

  - type: solenoid
    attr: { x: 270, y: 210, width: 110, height: 80, "data-dx": 30, "data-dy": 100 }
    data: { channel: "K.ThermCo.Diss", "active-abo

  - type: circle
```

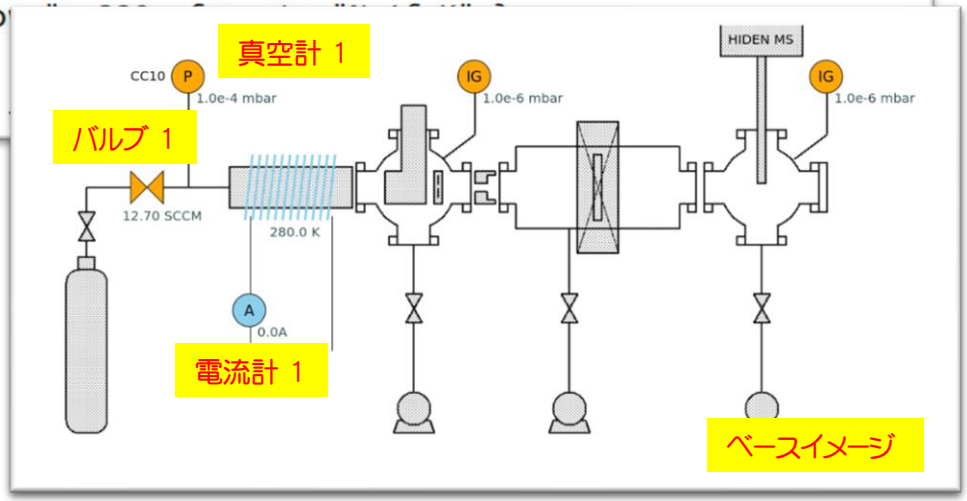
ベースイメージ

真空計1

データストア中のチャンネル名

バルブ 1

電流計 1

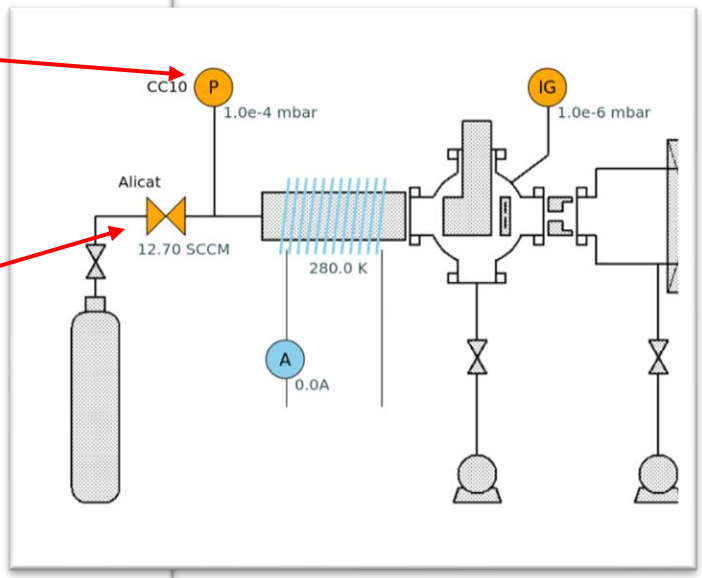


- YAML を書くのはやってみると
  それほど大変ではない
- 作り込みが好きなユーザは意外に多い
- 将来的にはビルドツールを作るかも
  (マウス操作より数値を打つ方が早い説もあるけど)

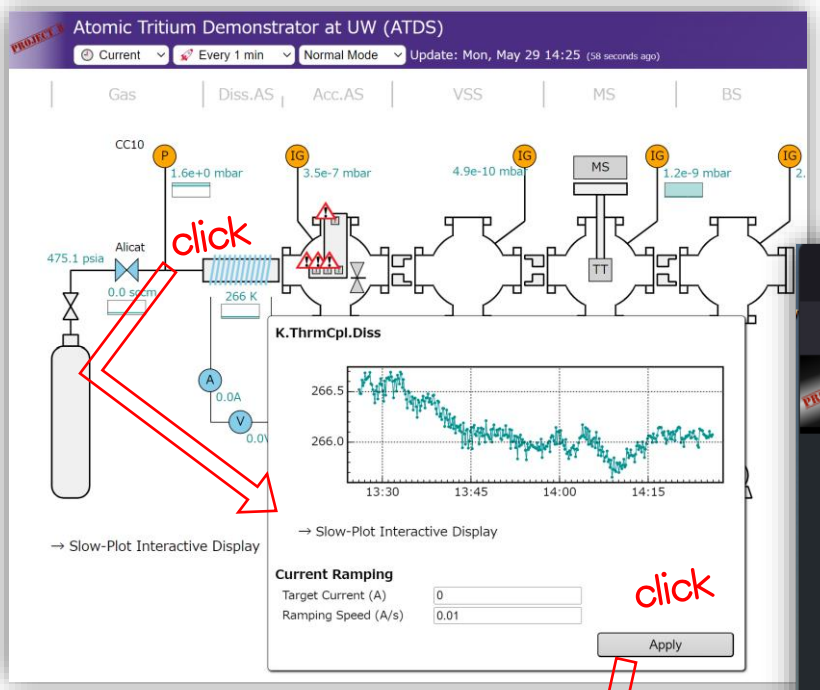
新しい形状の追加も簡単

- JavaScript で 10 行くらい (SVG 要素を作って返すだけ)
- プラグインにできる. コレクションにできる.

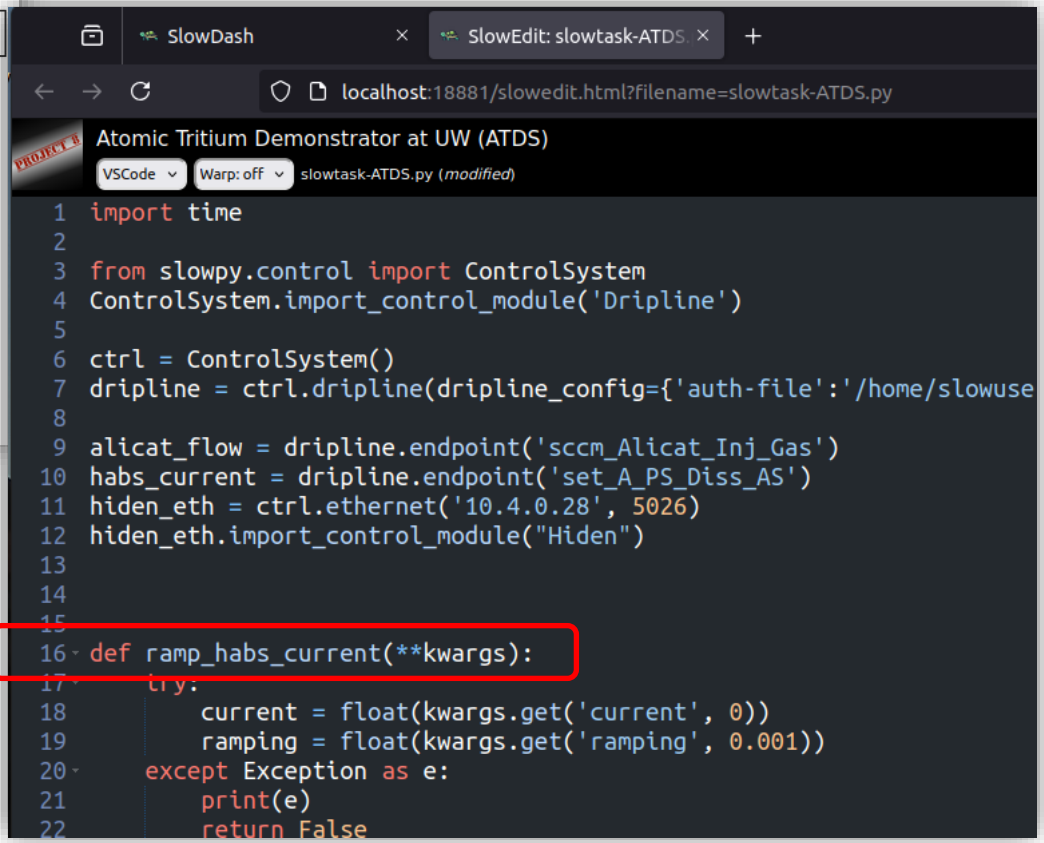
```
class SCCircleWidget extends SCShapeWidget {
    get_defaults() {
        return $.extend({}, super.get_defaults(), {
            width: 20, height: 20
        });
    }
    create_path() {
        let circleAttr = $.extend({}, this.attr, {
            cx: this.attr.x + this.attr.width/2,
            cy: this.attr.y + this.attr.height/2,
            rx: this.attr.width/2,
            ry: this.attr.height/2
        });
        return $('<ellipse>', 'svg').attr(circleAttr);
    }
};

class SCValveWidget extends SCShapeWidget {
    get_defaults() {
        return $.extend({}, super.get_defaults(), {
            width: 20, height: 20, orientation: 'horizontal'
        });
    }
    create_path() {
        let [x0, y0] = [this.attr.x, this.attr.y];
        let [x1, y1] = [x0 + this.attr.width, y0 + this.attr.height];
        let points = (
            (this.attr.orientation[0] == 'v') ?
            `${x0},${y0} ${x1},${y0} ${x0},${y1} ${x1},${y1} ${x0},${y0}` :
            `${x0},${y0} ${x0},${y1} ${x1},${y0} ${x1},${y1} ${x0},${y0}`
        );
        return $('<polyline>', 'svg').attr(this.attr).attr({'points': points});
    }
};
```



コントロールスクリプトのバインド



ユーザが書いた Python スクリプト
(SlowDash ブラウザ上で編集できる)



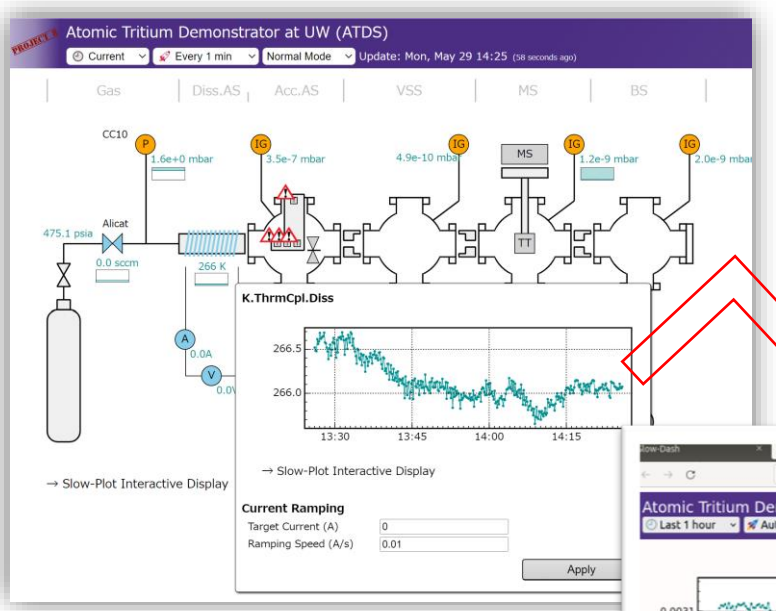
ボタンをクリックすると
ボタンの名前の関数が呼ばれる

入力要素の値が同名の引数になる

スクリプト中の変数を表示要素へ
直接バインドすることもできる.

ただの Python なので Python でできることは全てできる.
SlowDash なしでスクリプトの単独実行もできる.

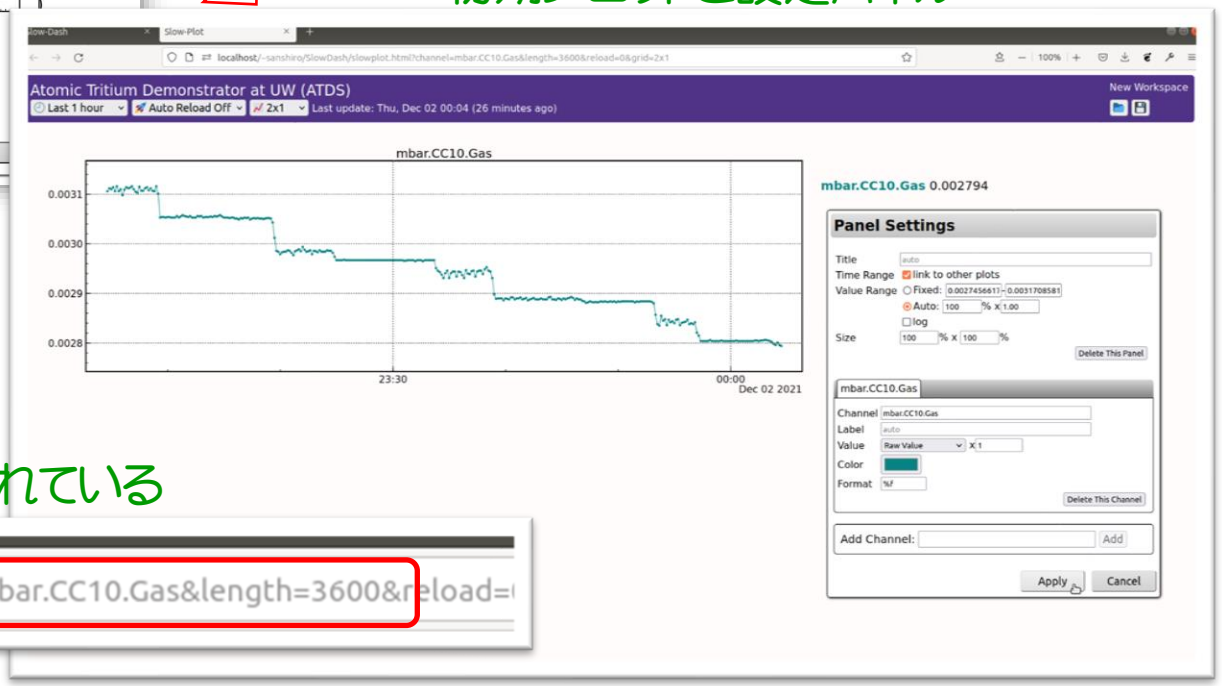
データブラウザへのリンク



click

データ要素をクリックすると時系列プロットを表示

初期プロットと設定パネル



ページは URL から動的に作られている

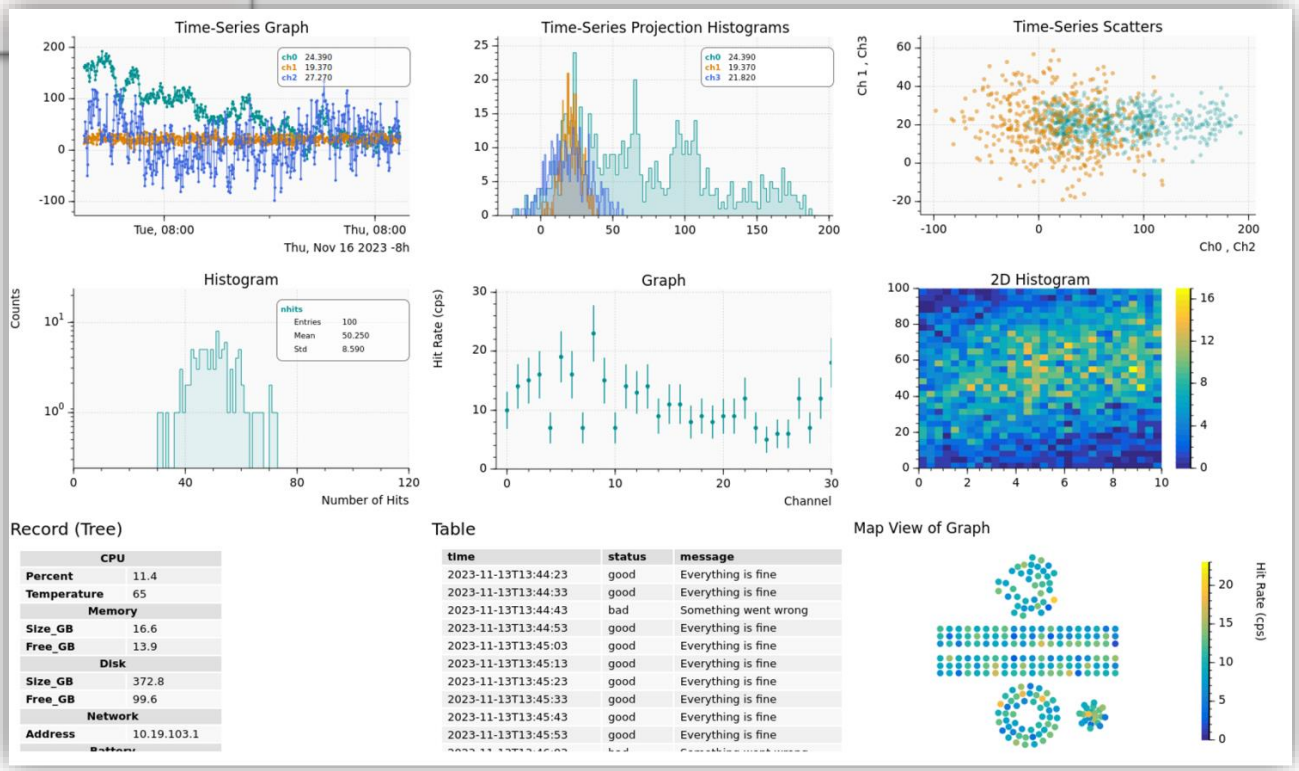
- いちいち「新規作成」→ポチポチ としなくていい
- リンクなので E-log とか Slack とかに貼れる

データブラウザ

ブラウザの中身は動的に構築できる

“新規パネル作成”

ポチポチする

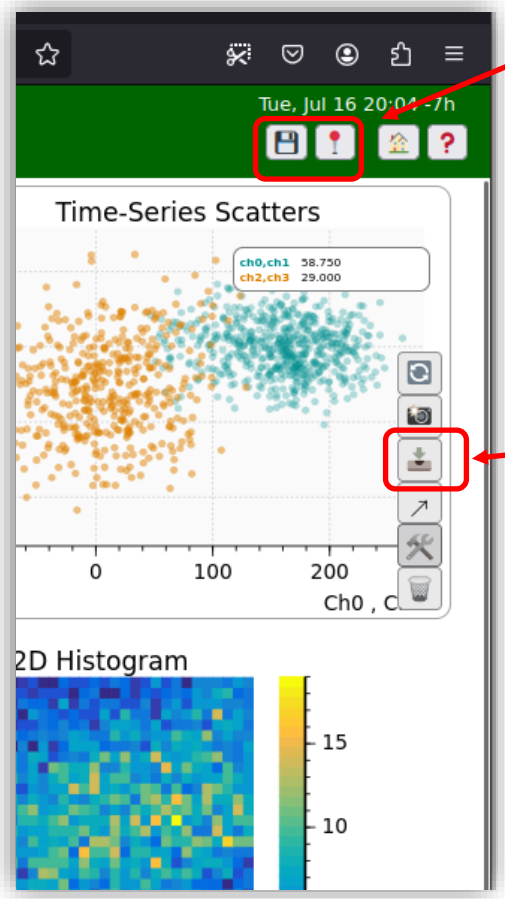


利用可能な表示形式

- 時系列プロット
- ヒストグラム
- エラー付きグラフ
- 2次元ヒストグラム
- テーブル
- ツリー
- マップビュー
- …

全てプラグイン実装

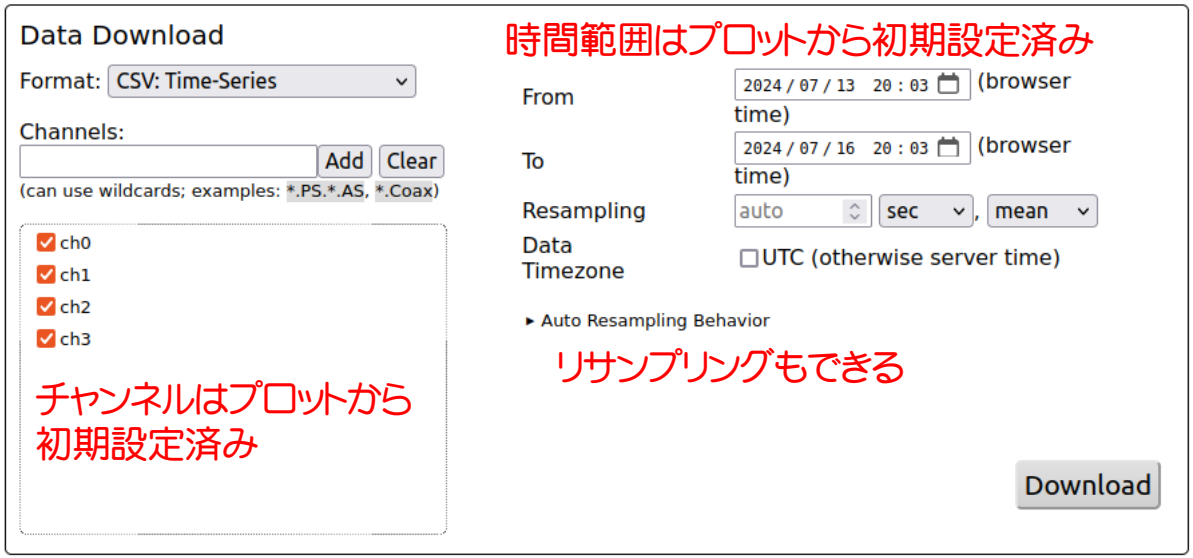
作成ページの共有，データのアクセス



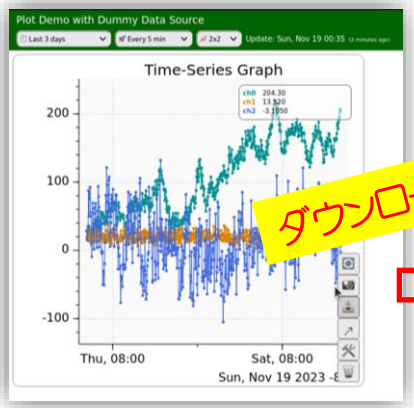
ページの保存，保存なしの URL 作成（5000 文字の URL）

- 保存したページは共有できる
- GUI で作成したページをテキストエディタで編集も可能

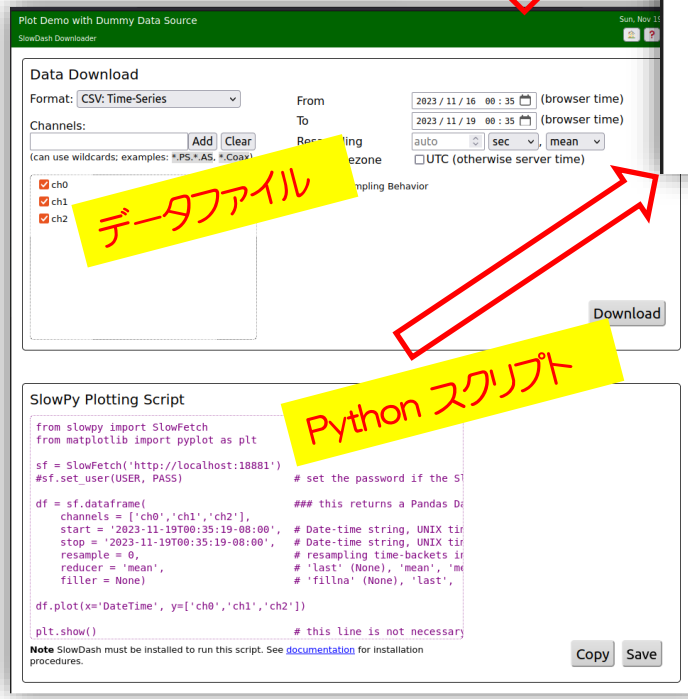
表示データのダウンロード



スクリプト自動生成と Jupyter 引き渡し

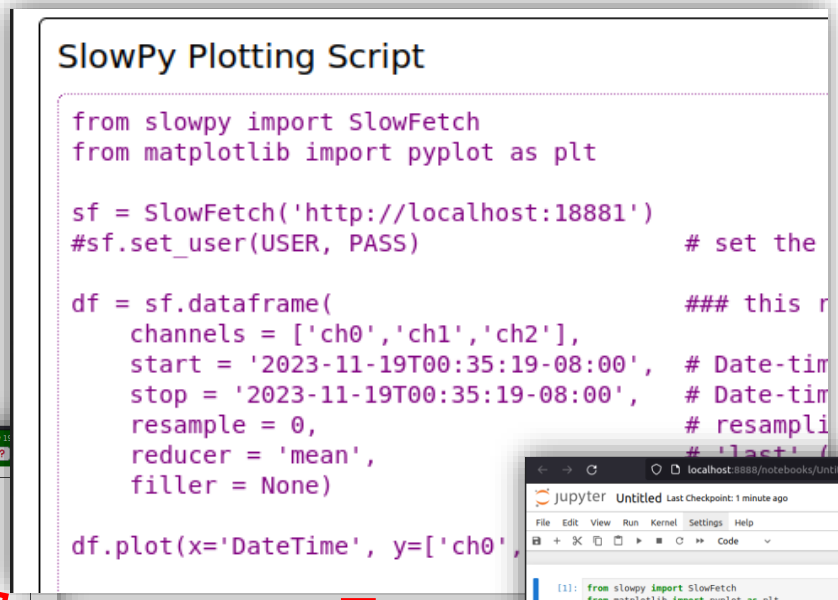


ダウンロードボタン

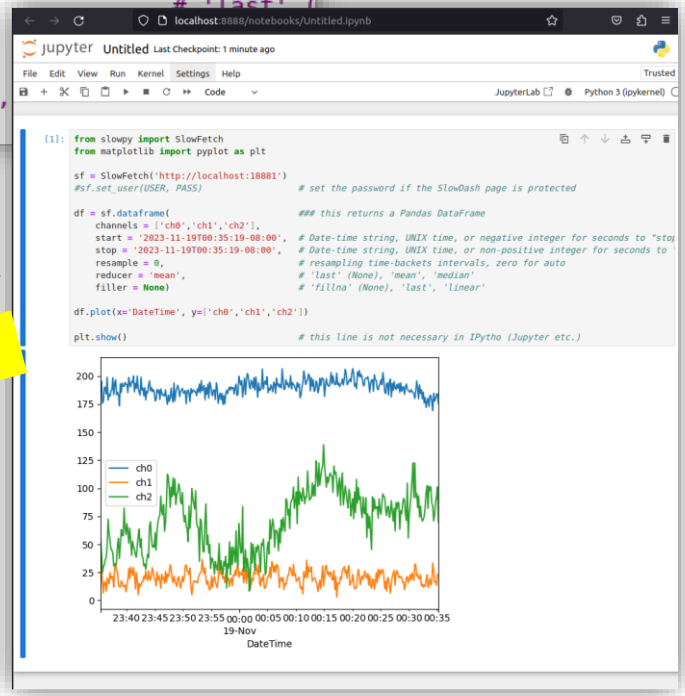


データファイル

Python スクリプト



Jupyter にコピペ



最新バージョンの SlowDash では直接エクスポート

データ元：だいたいの時系列データはそのまま読める

よくある形式その1（ロングフォーマット）

channel	timestamp	value
sccm.Alicat.Inj.Gas	2022-09-15 03:19:25.496212+00	0
V.ThermoCo.Diss.AS	2022-09-15 03:19:27.612427+00	6.605405e-05
mbar.IG.Vac.AS	2022-09-15 03:19:31.490579+00	2.26e-07
mbar.IG	2022-09	45+00	2e-
mbar.IG.Vac.BS	2022-09-15 03:19:31.610188+00	4e-

テーブル

タグ

時刻（ISO）

フィールド

SlowDash のフォーマット記述

```
table [channel]@timestamp = value
```

よくある形式その2（ワイドフォーマット）

RunNumber	TimeStamp	sccm.Alicat.Inj	mbar.CC10.Inj	K.ThrmCpl.Diss	mbar.IG.AS
3098	1664916014	3	1.18467	340.58	5.38333e-05
3097	1664915456	3	1.256	503.275	5.36e-05
3096	1664914833	3	1.36833	745.743	5.38333e-05
3095				1154.09	5.44
3094			1.48933	1501.14	5.46

テーブル

時刻（UNIX）

複数フィールド

```
RunTable@TimeStamp
```

混ぜたもの

metric	set_or_ist	timestamp	value_raw	value_cal
psia.Alicat.Inj.Gas	ist	2022-09-15 03:19:25.419417+00	9.6	9.6
degC.Alicat.Inj.Gas	ist	2022-09-15 03:19:25.458695+00	23.42	23.42
sccm.Alicat.Inj.Gas	ist	2022-09-15 03:19:25.496212+00	0	0
V.ThermoCo		2022-09-15 03:19:27.612427+00	6.6	e-05
V.PS.Diss.AS	ist	2022-09-15 03:19:29.387352+00	0.01	0.01
A.PS.Diss.AS	ist	2		

テーブル

複数タグ

複数フィールド

```
Data[metric, set_or_ist]@timestamp = value_raw, value_cal
```

データ元：いろいろなデータストアを読める

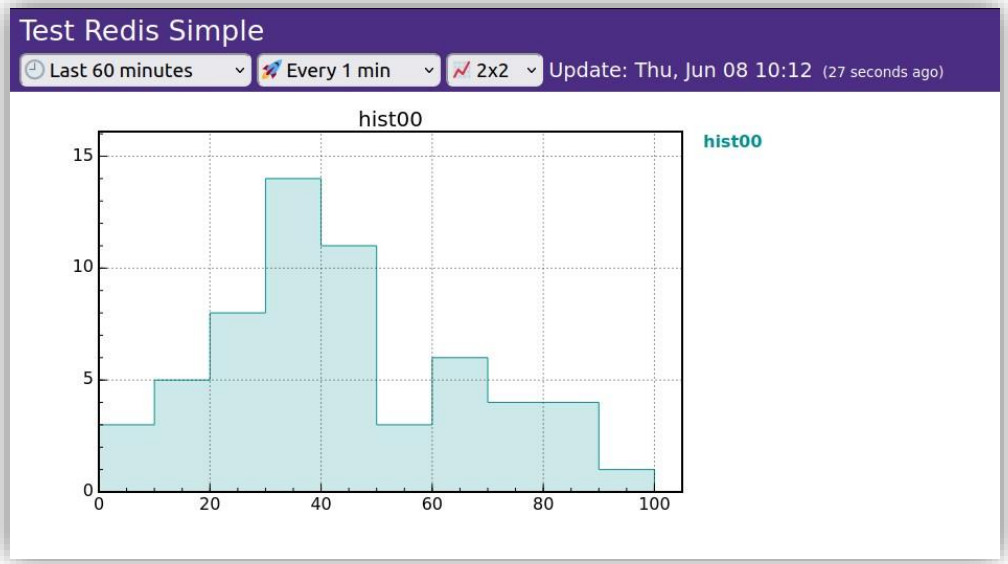
- これらは全てプラグイン実装
- MySQL プラグインは COMET/KEK 大石さんからの寄与

ヒストグラムは JSON 文字列として記録

データ値として JSON の文字列を書く

```
{
  "bins": { "min": 0, "max": 100 },
  "counts": [ 3, 5, 8, 14, 11, 3, 6, 4, 4, 1 ]
}
```

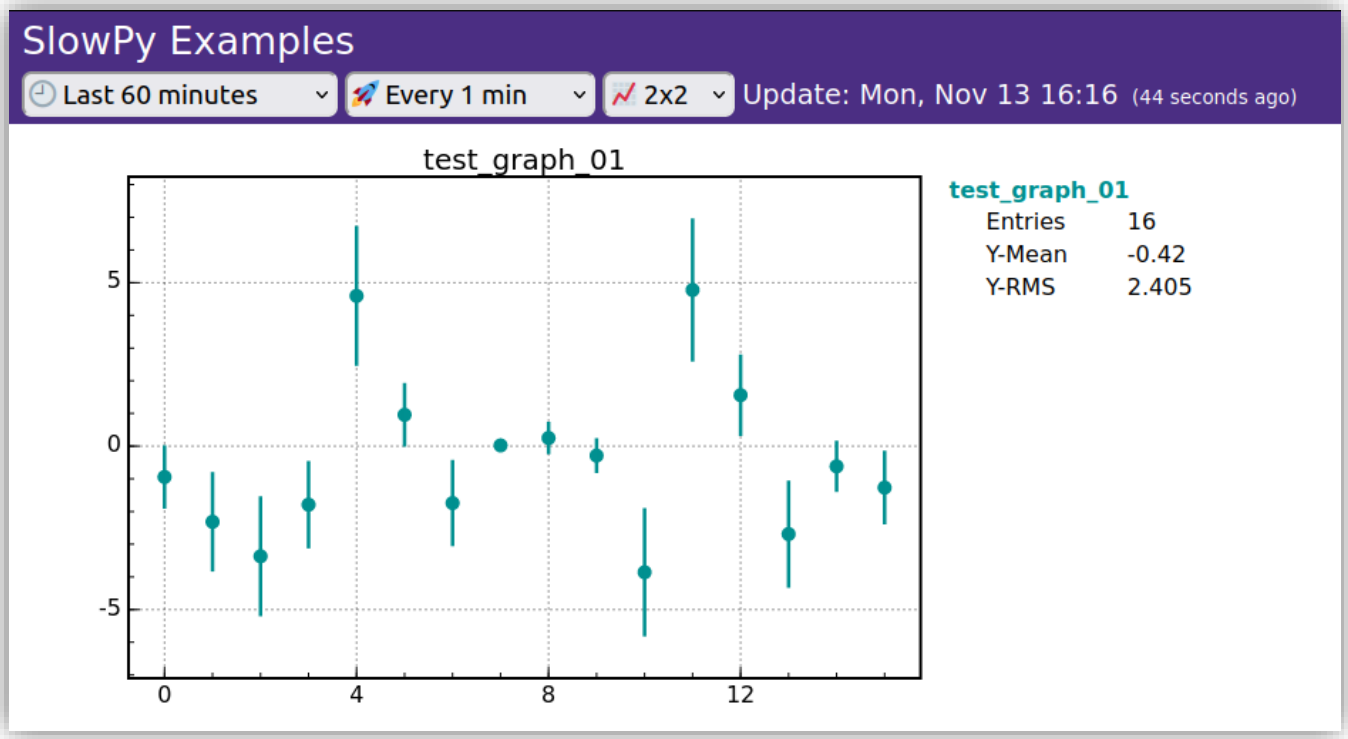
伸びていくヒストグラムの時系列



JSON へのシリアライザライブラリ： SPADI-A による C++ 版と組み込みの Python 版

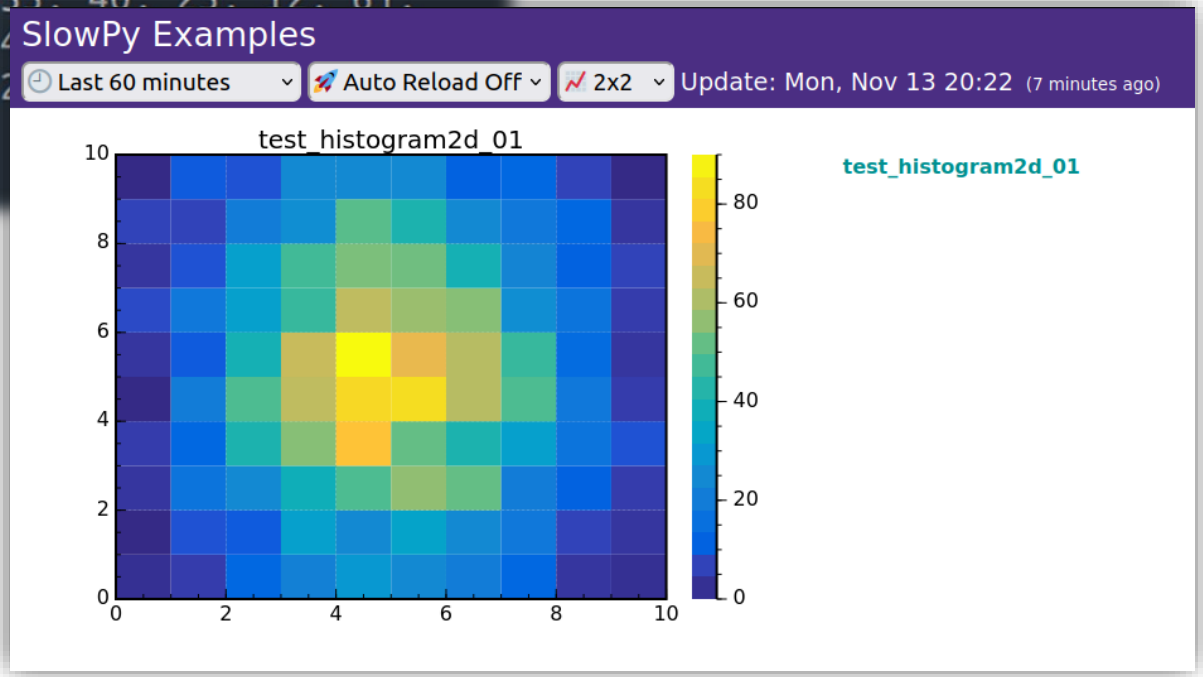
グラフ(エラーバー付き)はこんな感じ


```
{
    "_stat": { "Entries": 16, "Y-Mean": -0.449, "Y-RMS": 2.5 },
    "labels": [ "ch", "value", "error" ],
    "x": [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ],
    "y": [ -5.117, 0.765, -0.467, -2.738, -2.407, 3.652, 0.953, 3.262, 0.327, 0.55
    "y_err": [ 2.2620, 0.8746, 0.6833, 1.6546, 1.5514, 1.9110, 0.9762, 1.8061, 0.5
}
```

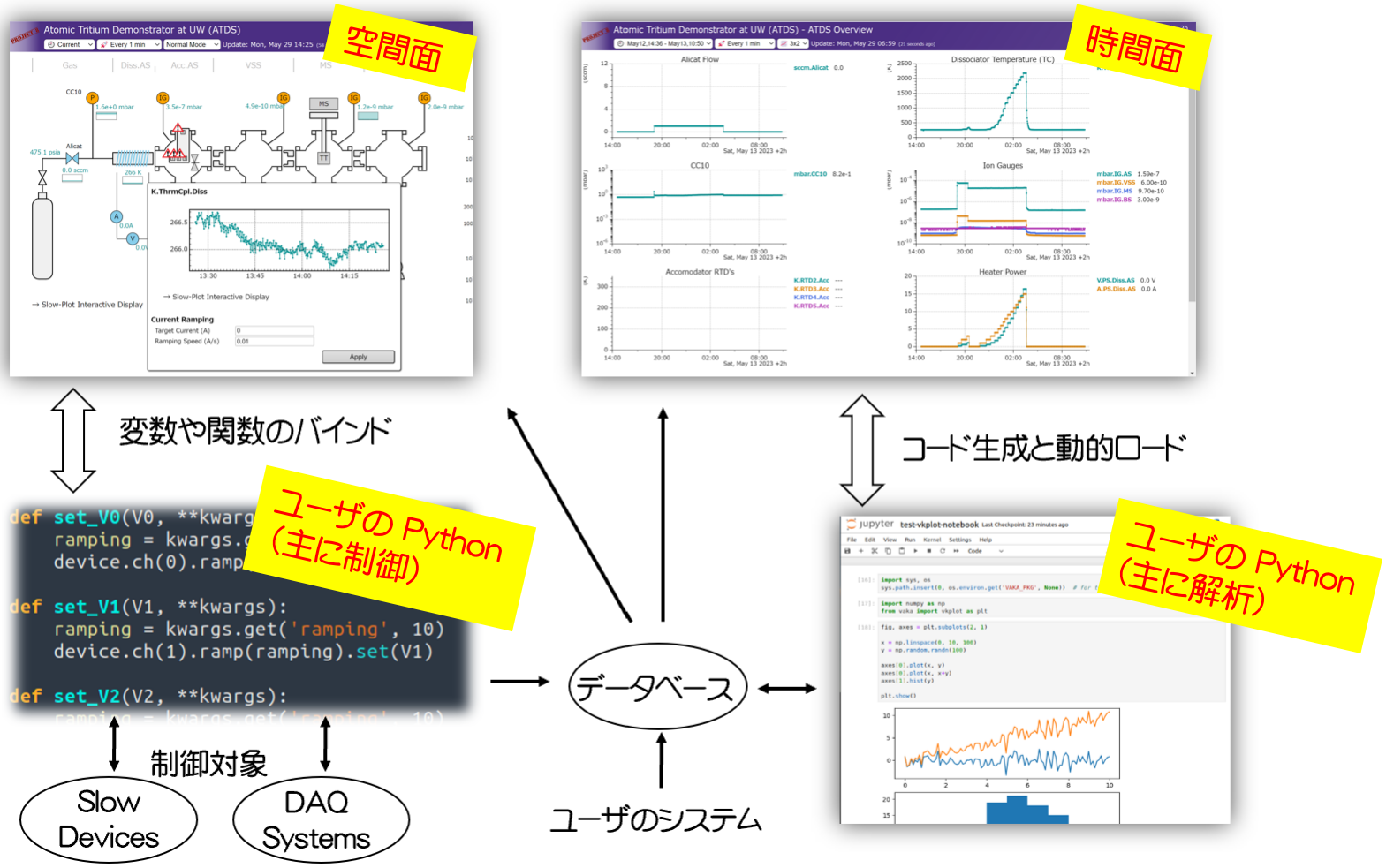
2次元ヒストグラムも


```
{
    "xbins": {"min": 0, "max": 10},
    "ybins": {"min": 0, "max": 10},
    "counts": [
        [2, 5, 14, 22, 29, 25, 20, 13, 3, 2],
        [1, 9, 11, 31, 25, 33, 24, 19, 6, 3],
        [3, 17, 24, 39, 49, 57, 52, 21, 12, 5],
        [5, 13, 42, 55, 76, 51, 42, 31, 17, 9],
        [1, 20, 48, 64, 82, 83, 62, 48, 19, 5],
        [4, 11, 40, 65, 90, 71, 62, 46, 15, 4],
        [8, 19, 31, 45, 64, 58, 55, 26, 17, 5],
        [4, 9, 32, 47, 54, 53, 40, 23, 12, 6],
        [6, 7, 20, 26, 50, 4
        [1, 10, 9, 24, 25, 2
    ]
}
```

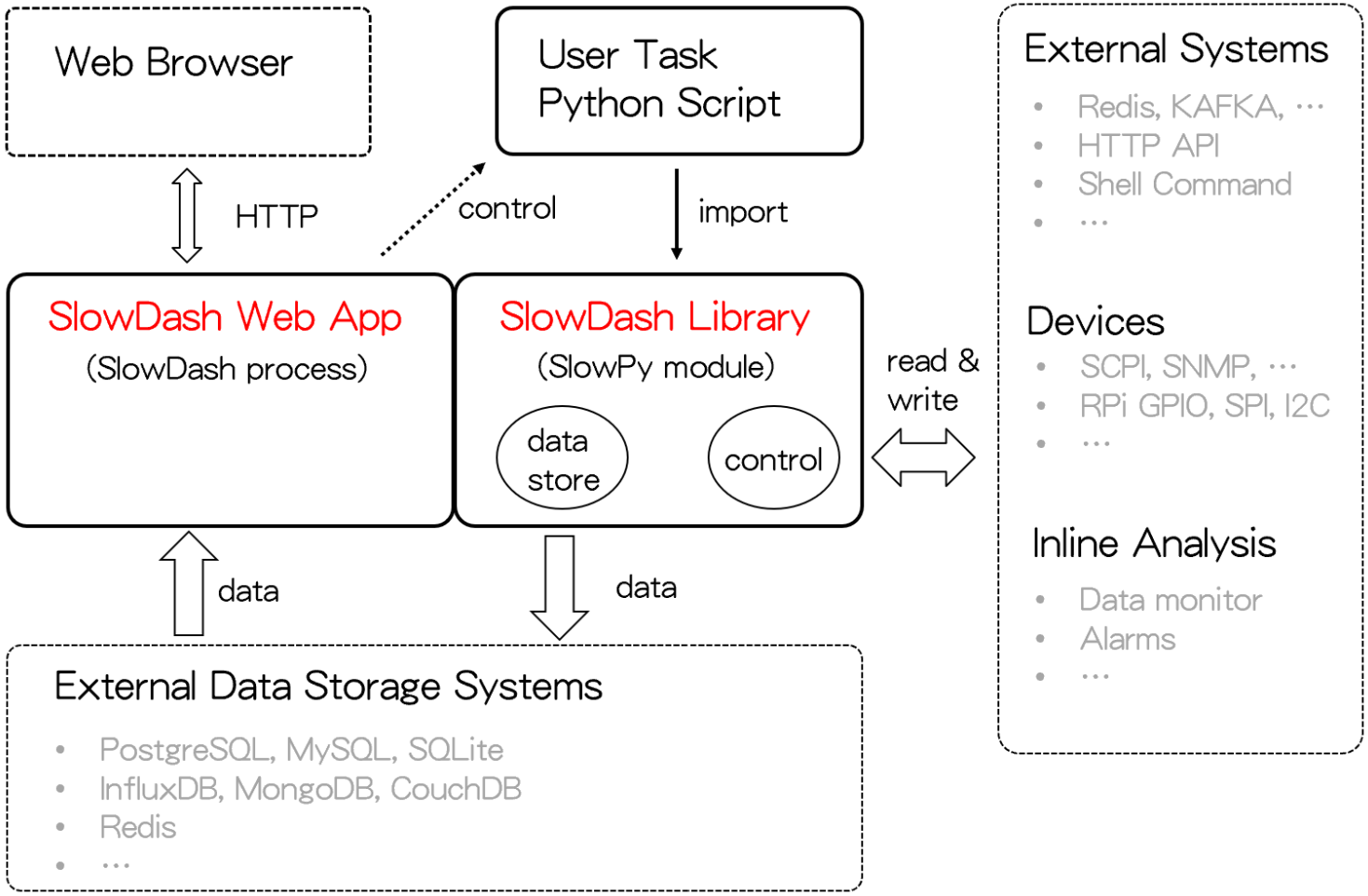
概要まとめ：SlowDash の構造

- 空間面（LabVIEW 風）と時間面（Gfarana 風）のデュアルビュー
- Web アプリと Python スクリプトのハイブリッド
- データベースは既存のものをつかう



SlowDash = App + Lib + Scripts

Web アプリとユーザ用 Python ライブラリ (SlowPy)



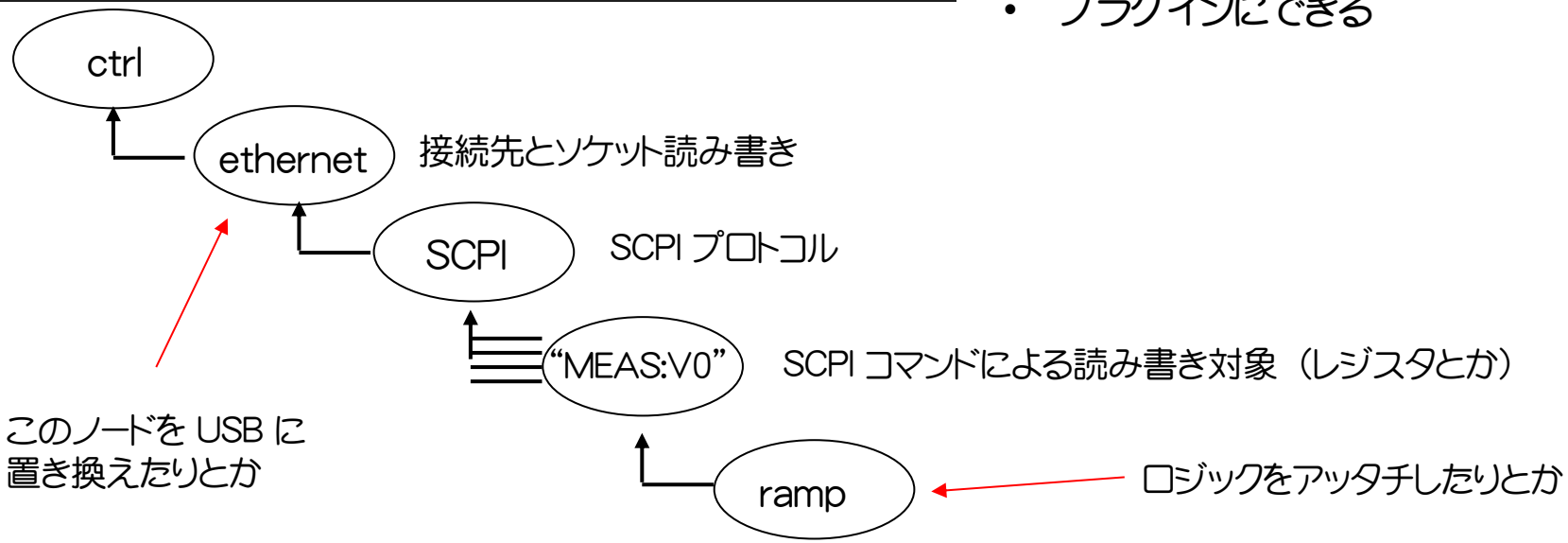
- SlowDash App 自体はどんな Python スクリプトでも組み込める
- SlowPy は機器制御やデータベース記録を簡単にするライブラリ（使わなくてもよい）

SlowPy：読み書きできる制御変数の木構造

ハードウェアの例：イーサーネットのSCPI コマンドで制御する電源ユニット

```python
from slowpy.control import ControlSystem
ctrl = ControlSystem()

# SlowPy Control Variables
device = ctrl.ethernet('192.168.1.43', port=17674).scpi()
device_id = device.command('*IDN')
V0 = device.command('MEAS:V0')

print('ID: %s' % str(device_id.get()))  # read device ID
V0.ramp(ramping=0.1).set(10)  # apply set-point with ramping
while True:
    data = V0.get()     # read voltage value from the device
    ...
```

- 読み書き可能ノードの集合
- データ型は何でもよい：
  - 数値
  - 長いテキストや JSON
  - Python オブジェクトでも
- 下位のノードが上位ノードを利用する
- 接続する上位ノードを選べる
- プラグインにできる

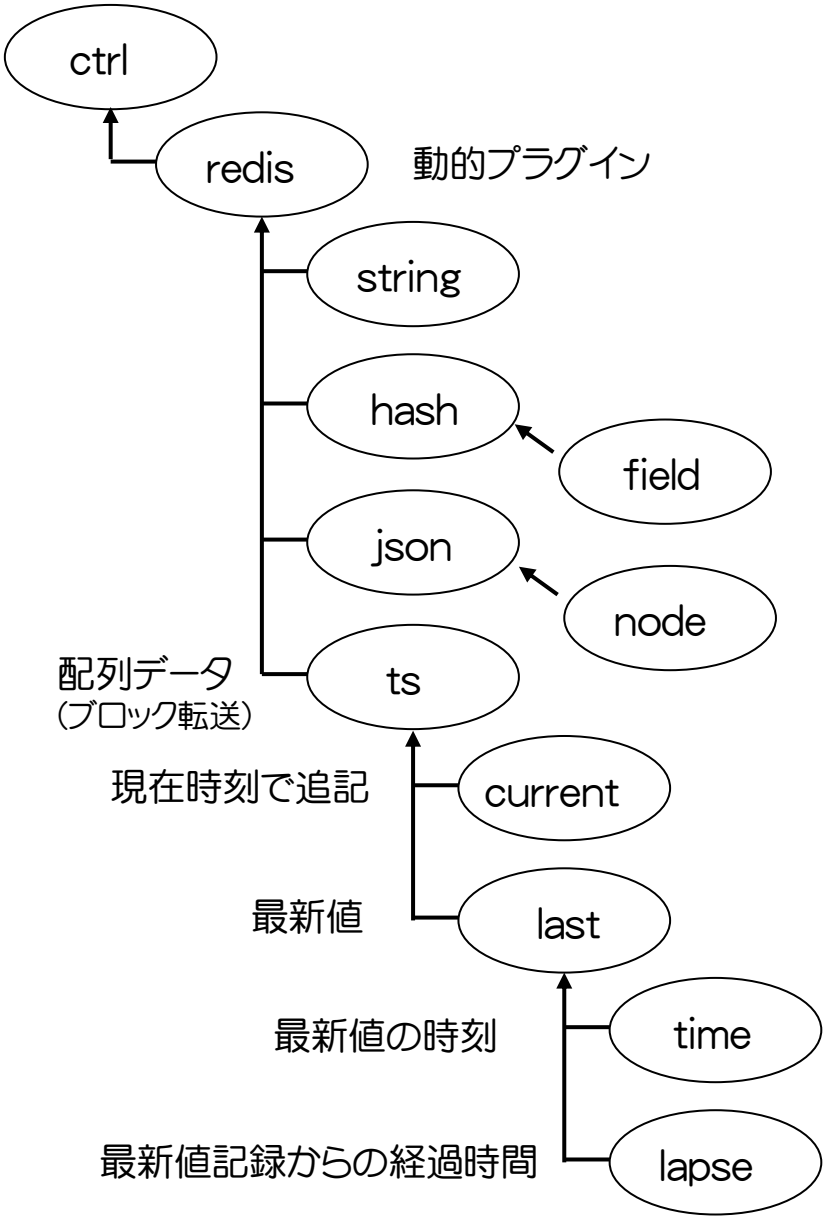


SlowPy：読み書きできる制御変数の木構造

外部システム接続の例：Redis とやりとり

（モジュールはプラグインの動的ロード）

```
from slowpy.control import ControlSystem
ctrl = ControlSystem()
ctrl.load_control_module('Redis')
redis = ctrl.redis('redis://localhost:6379/12')


# string
redis.string('name').set('SlowDash')
print(redis.string('name'))


# hash
print(redis.hash("Status"))
print(redis.hash("Status").field("Count"))
redis.hash("Status").field("Count2").set(10)


# time-series
print(redis.ts('ch00'))
print(redis.ts('ch00').last())
print(redis.ts('ch00').last().time())
print(redis.ts('ch00').last().lapse())

import time
redis.ts('ts00').current().set(123)
redis.ts('ts00').set([(int(1000*(time.time()-100)), 456)])
print(redis.ts('ts00'))
print(redis.ts('ts00').last().lapse())
```



ワークフロー例：電圧計読み出し

BK PRECISION

Model: 5492B, 5492BGPIB

5 ½ Bench Digital Multimeter

USER MANUAL

Chapter 6 **SCPI Command Reference**

This chapter is outlined as follows:

6.1 Command Structure
6.2 Command Syntax
6.3 Command Reference

6.1 Command Structure

The remote commands are divided into two types: Common commands and SCPI commands. The common commands are defined in IEEE std. 488.2-1987, and these commands are common for all devices. Not all commands are supported by the 5492B, and some commands are not supported by the GPIB interface for the 5492BGPIB. Please look through the command syntax thoroughly before programming. The SCPI commands are used to control most of the 5492B functions. They can be represented as a tree structured with three levels deep. (The highest level commands are called the subsystem commands in this manual.) The lower level commands are part of subsystem commands and a colon (:) is used to separate the higher level commands and the lower level commands. See Figure 6-1 as an example.

***IDN?**
Query Syntax:
*IDN?
Query return:
<product>,<version>,<sn number>

Example: 5492B Digital Multimeter, Ver1.0.00.00.01,123A45678

Description: Query the identification of the instrument.

SCPI というテキストコマンド
プロトコルを使っている（業界標準）

コマンドもリプライもソケット経由の
テキストで超簡単

- *IDN? で型番を返す
- *RST? でリセットする
- :READ? で電圧値を返す

Step 1: Socket 経由で SCPI コマンドを送るテスト

SlowPy ライブラリを使うと簡単（使わなくても良い）

```python
from slowpy.control import ControlSystem
ctrl = ControlSystem()

voltmeter = ctrl.ethernet("10.0.0.54", port=1234).scpi()

print(voltmeter.command('*idn?'))
```

```
sanshiro@vs13: ~/SlowVoltmeter
sanshiro@vs13:~/SlowVoltmeter$ python3 slow-voltmeter.py
Ethernet: 10.0.0.54:1234 connected
5492B Digital Multimeter,Ver1.1.13.06.04,343B18165
sanshiro@vs13:~/SlowVoltmeter$
```

スクリプト単体でハードウェアとの通信をチェックできる

Step 2: SCPI コマンドで電圧値を読み出す

```
from slowpy.control import ControlSystem
ctrl = ControlSystem()

voltmeter = ctrl.ethernet("10.0.0.54", port=1234).scpi()

print(voltmeter.command('*idn?'))

print(voltmeter.command(':read?'))
```

```
sanshiro@vs13: ~/SlowVoltmeter
sanshiro@vs13:~/SlowVoltmeter$ python3 slow-voltmeter.py
Ethernet: 10.0.0.54:1234 connected
5492B Digital Multimeter,Ver1.1.13.06.04,343B18165
-8.012122e+000
sanshiro@vs13:~/SlowVoltmeter$
```

読み出し部分完成！

Step 3: データベースに書き込む

```python
from slowpy.control import ControlSystem
ctrl = ControlSystem()

from slowpy.store import DataStore_SQLite
datastore = DataStore_SQLite('sqlite:///SlowTestData.db', table="Test")

voltmeter = ctrl.ethernet("10.0.0.54", port=1234).scpi()

print(voltmeter.command('*idn?'))


for i in range(3):
    V = voltmeter.command(':read?')
    datastore.append(V, tag='Voltage')
```

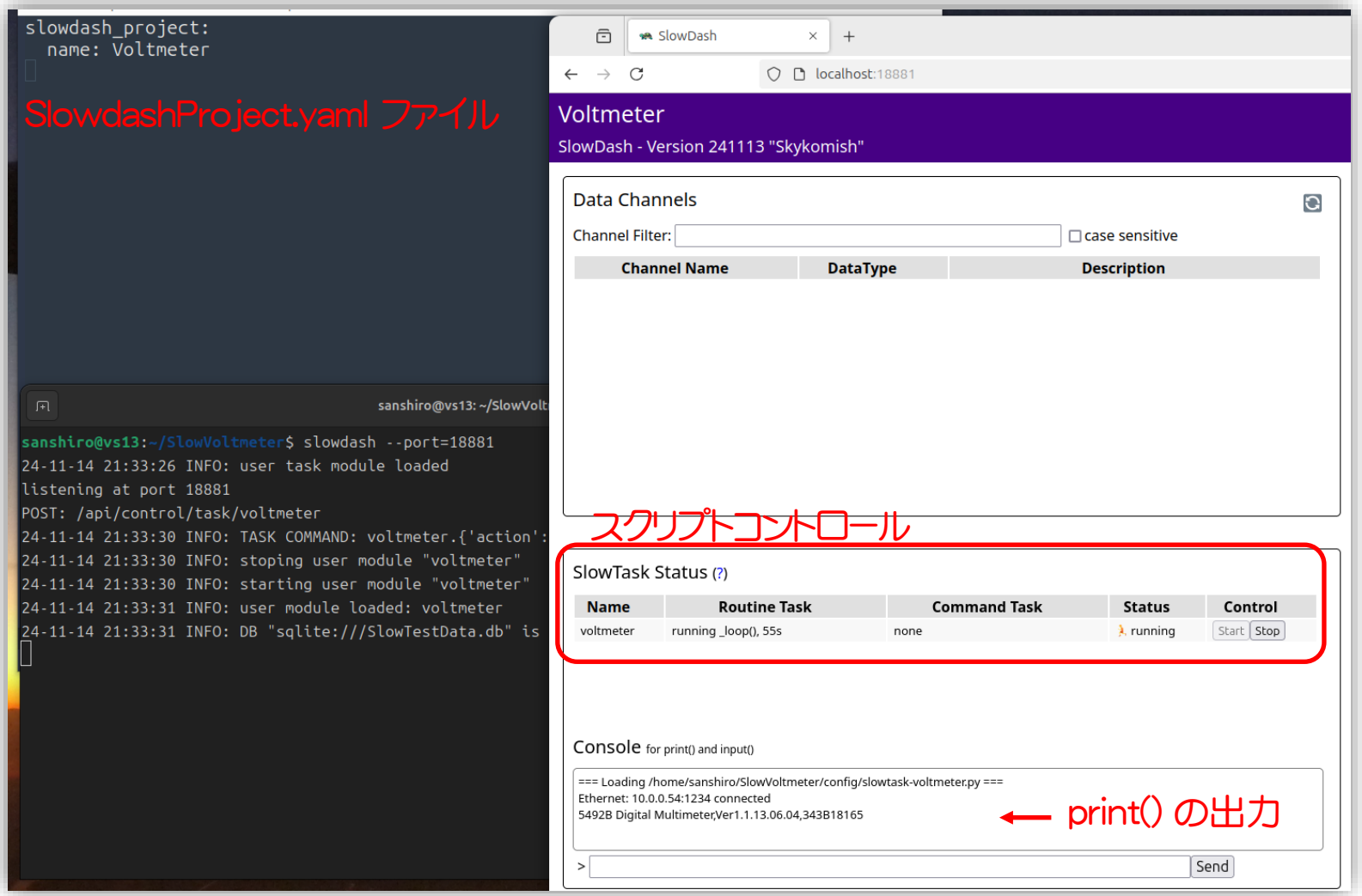
```
sanshiro@vs13: ~/SlowVoltmeter
sanshiro@vs13:~/SlowVoltmeter$ python3 slow-voltmeter.py
Ethernet: 10.0.0.54:1234 connected
5492B Digital Multimeter,Ver1.1.13.06.04,343B18165
sanshiro@vs13:~/SlowVoltmeter$ sqlite3 SlowTestData.db
SQLite version 3.45.1 2024-01-30 16:01:20
Enter ".help" for usage hints.
sqlite> .schema Test
CREATE TABLE Test(timestamp REAL, channel TEXT, value TEXT, PRIMARY KEY(timestamp,channel));
sqlite> select * from Test;
1731648303.498|Voltage|-8.200915e+000
1731648303.656|Voltage|-9.183785e+000
1731648303.81|Voltage|-9.188433e+000
sqlite>
```

データベースの中身を見る

データを読んで記録するプログラムが完成！

Step 4: SlowDash からスクリプトを走らせる

名前だけを指定した SlowDash のプロジェクトファイルを作成　（将来的にはこれも不要にしたい）



- 作ったスクリプトを slowtask-XXX.py という名前で config の下に置くと SlowDash が見つける
- SlowPy を使ってもいいけれど，使わなくてもよい．Python なら何でもよい．

Step 5: SlowDash でデータベースを読む

```
slowdash_project:
  name: Voltmeter

  data_source:
    url: sqlite:///SlowTestData
    parameters:
      time_series:
        schema: Test [channel] @timestamp(unix) = value
```

データベース設定

```
sanshiro@vs13:~/SlowVoltmeter$ slowdash --port=18881
24-11-14 21:44:03 INFO: user task module loaded
listening at port 18881
POST: /api/control/task/voltmeter
24-11-14 21:44:17 INFO: TASK COMMAND: voltmeter.{'action': 'start'}
24-11-14 21:44:17 INFO: stoping user module "voltmeter"
24-11-14 21:44:17 INFO: starting user module "voltmeter"
24-11-14 21:44:17 INFO: user module loaded: voltmeter
24-11-14 21:44:17 INFO: DB "sqlite:///SlowTestData.db" is connnected.
24-11-14 21:44:21 WARNING: Performing a full scan of the SQL table (Test)
24-11-14 21:44:21 WARNING:   This full-scan can be avoided by either:
24-11-14 21:44:21 WARNING:     - manually list the channels, in "data_sou
as an array, or
24-11-14 21:44:21 WARNING:     - provide a SQL that returns the list, in
gs/sql" as a string
```

SlowDash — localhost:18881

Voltmeter

SlowDash - Version 241113 "Skykomish"

Data Channels

Channel Filter: ☐ case sensitive

Channel Name	DataType	Description
Voltage	timeseries	

データチャンネル一覧（自動取得）

SlowTask Status (?)

Name	Routine Task	Command Task	Status	Control
voltmeter	running _loop(), 33s	none	running	Start Stop

Console for print() and input()

```
=== Loading /home/sanshiro/SlowVoltmeter/config/slowtask-voltmeter.py ===
Ethernet: 10.0.0.54:1234 connected
5492B Digital Multimeter,Ver1.1.13.06.04,343B18165
```

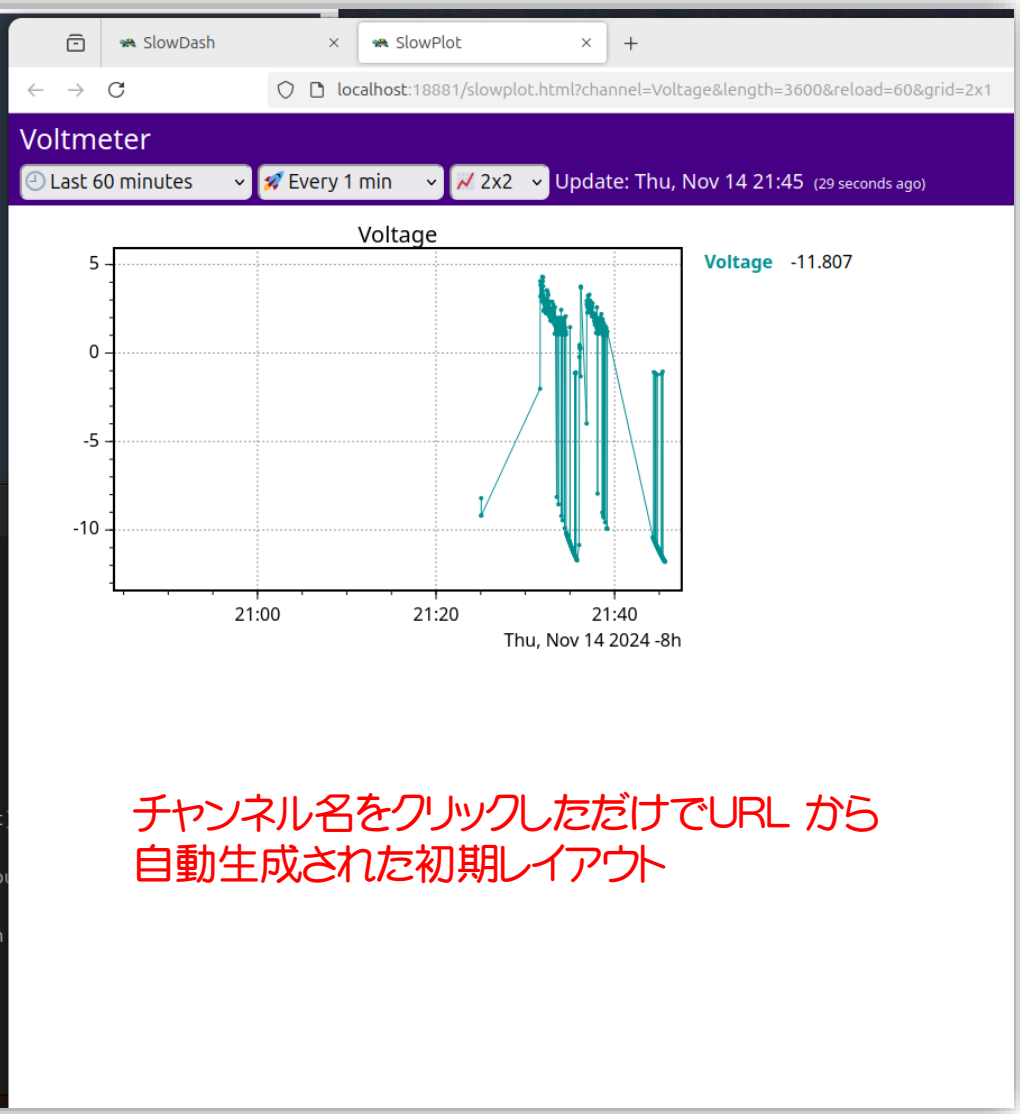
> Send

データベースURLとテーブル名とスキーマを知らせると SlowDash がスキャンしてチャンネル一覧を作る

Step 6: チャンネル名をクリックしてプロットを作成

```
slowdash_project:
  name: Voltmeter

  data_source:
    url: sqlite:///SlowTestData
    parameters:
      time_series:
        schema: Test [channel] @timestamp(unix) = value
```

```
sanshiro@vs13: ~/SlowVoltmeter
sanshiro@vs13:~/SlowVoltmeter$ slowdash --port=18881
24-11-14 21:44:03 INFO: user task module loaded
listening at port 18881
POST: /api/control/task/voltmeter
24-11-14 21:44:17 INFO: TASK COMMAND: voltmeter.{'action': 'start'}
24-11-14 21:44:17 INFO: stoping user module "voltmeter"
24-11-14 21:44:17 INFO: starting user module "voltmeter"
24-11-14 21:44:17 INFO: user module loaded: voltmeter
24-11-14 21:44:17 INFO: DB "sqlite:///SlowTestData.db" is connnected.
24-11-14 21:44:21 WARNING: Performing a full scan of the SQL table (Test
24-11-14 21:44:21 WARNING:   This full-scan can be avoided by either:
24-11-14 21:44:21 WARNING:     - manually list the channels, in "data_so
as an array, or
24-11-14 21:44:21 WARNING:     - provide a SQL that returns the list, in
gs/sql" as a string
```



チャンネル名をクリックしただけでURL から
自動生成された初期レイアウト

ここからレイアウトを作り込んでいって保存しても良い

Step 7: Jupyter エクステンションを有効化

```
slowdash_project:
  name: Voltmeter

  data_source:
    url: sqlite:///SlowTestData
    parameters:
      time_series:
        schema: Test [channel] @time

  extension:
    - name: jupyter
      parameters:
        url: http://localhost:8888
        token: SlowJupyter
```

URL と Token

```
sanshiro@vs13:~/SlowVoltmeter$ slowdash --po
24-11-14 21:53:19 INFO: user task module loa
listening at port 18881
24-11-14 21:53:40 WARNING: Performing a full
24-11-14 21:53:40 WARNING:   This full-scan
24-11-14 21:53:40 WARNING:     - manually li
as an array, or
24-11-14 21:53:40 WARNING:     - provide a S
gs/sql" as a string
```

SlowDash | SlowPlot | SlowDash | Home Page - Select | SlowDash-241114-2

localhost:18881/slowdown.html?channels=Voltage&length=3600&to=1731649723

Voltmeter

SlowDash Downloader

Thu, Nov 14 21:51 -8h

Data Download / Export

Name: SlowDash-241114-215618

Channels:

Add Clear

(can use wildcards; example: adc_ch*)

Voltage

From 2024 / 11 / 14 20 : 48 (browser time)

To 2024 / 11 / 14 21 : 48 (browser time)

Resampling auto sec mean

Data Timezone UTC (otherwise server time)

▸ Auto Resampling Behavior

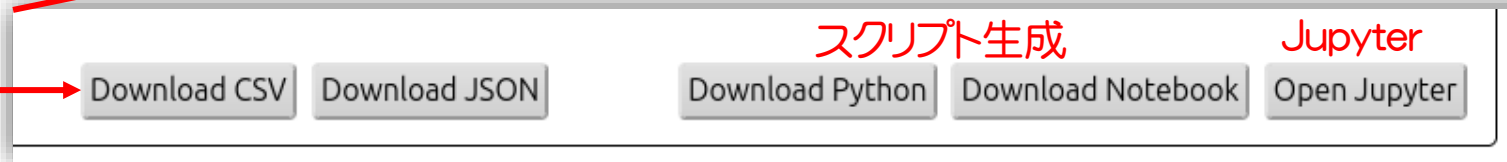
Plotting Script Generation (Experimental)

- To run the script, the SlowPy library must be installed. See documentation for installation procedures, or use the `slowpy-notebook` container.
- This feature is experimental. The generated scripts might not be compatible with future releases of SlowDash.
- Time-axis plots (time-series) and XY plots (histograms, graphs, etc) cannot be mixed.

プロットの「データダウンロード」をクリックして表示されるパネル.
チャンネルや時間範囲など初期設定済み

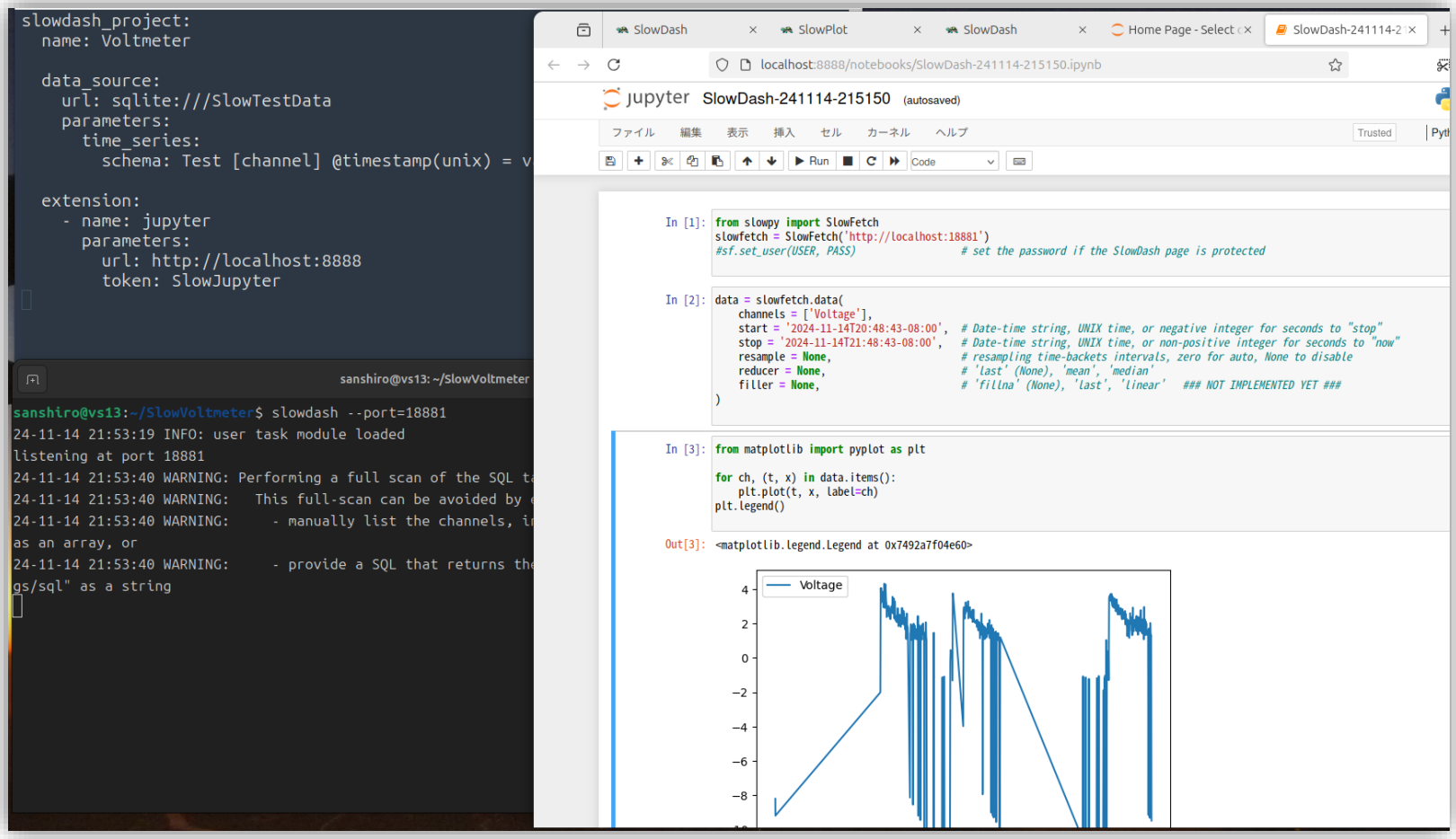
Download CSV | Download JSON | Download Python | Download Notebook | Open Jupyter

Excel 用に
リサンプリングで
四角くする



Step 8: Open Jupyter をクリック

データブラウザと同じプロットを作成するノートブックが生成される

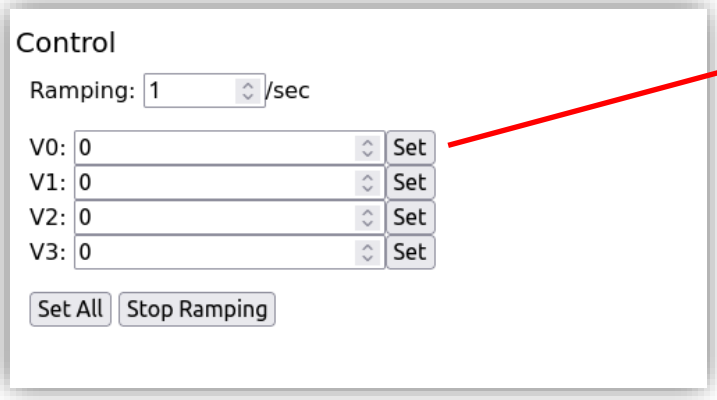


- データベースからデータをダウンロードして NumPy 配列に入れて Matplotlib で描画（SlowPy 使用）
- あとはフィットでもなんでもユーザにおまかせ
- 好きなライブラリを使って解析を続けてられる

さらに先へ: HTML フォームで制御とか

4 ch 電源ユニットを制御する例

```
<form>
  Ramping: <input type="number" name="ramping" value="1" style="width:5em">/sec
  <p>
  V0: <input type="number" name="V0" value="0"><input type="submit" name="async test.set_V0()" value="Set"><br>
  V1: <input type="number" name="V1" value="0"><input type="submit" name="async test.set_V1()" value="Set"><br>
  V2: <input type="number" name="V2" value="0"><input type="submit" name="async test.set_V2()" value="Set"><br>
  V3: <input type="number" name="V3" value="0"><input type="submit" name="async test.set_V3()" value="Set"><br>
  <p>
  <input type="submit" name="test.set_all()" value="Set All">
  <input type="submit" name="async test.stop()" value="Stop Ramping"><br>
</form>
```




```
def set_V0(V0, **kwargs):
    ramping = kwargs.get('ramping', 10)
    device.ch(0).ramp(ramping).set(V0)

def set_V1(V1, **kwargs):
    ramping = kwargs.get('ramping', 10)
    device.ch(1).ramp(ramping).set(V1)

def set_V2(V2, **kwargs):
    ramping = kwargs.get('ramping', 10)
    device.ch(2).ramp(ramping).set(V2)

def set_V3(V3, **kwargs):
    ramping = kwargs.get('ramping', 10)
    device.ch(3).ramp(ramping).set(V3)

def set_all(**kwargs):
    set_V0(**kwargs)
    set_V1(**kwargs)
    set_V2(**kwargs)
    set_V3(**kwargs)
```

ボタンをクリックすると
ユーザスクリプトの関数が呼ばれる

SlowPy 制御変数を HTML エレメントに
直接バインドすることも可能
(変数が変わると表示が変わる)

いろいろな実験的実装

Matplotlib を Slowplotlib で包んでみた

- ユーザが Matplotlib を使って作ったプロットを盗んでデータベースに記録できる
  → SlowDash の自動実行に組み込んでインラインデータ解析
- プロット構造を盗んでユーザスクリプトから SlowDash プロジェクトの自動生成

⇒ このスライドの付録参照

SlowPy の内部変数を SCPI インターフェースで外に出してみた

- SlowPy の読み出しシステムを一つのデバイスとして他のシステムから使える
- SlowDash のマイクロサービスとして分散できる（SCPI を使った SlowDash 版 RPC）
- SlowDash 自体を一つの SCPI デバイスとして他のシステム**から**コントロールできる
  (LabVIEW からでも使える. たぶん.)

⇒ SlowDash ドキュメント参照

制御変数のシーケンスを FPGA 風の文法で記述できるようにしてみた

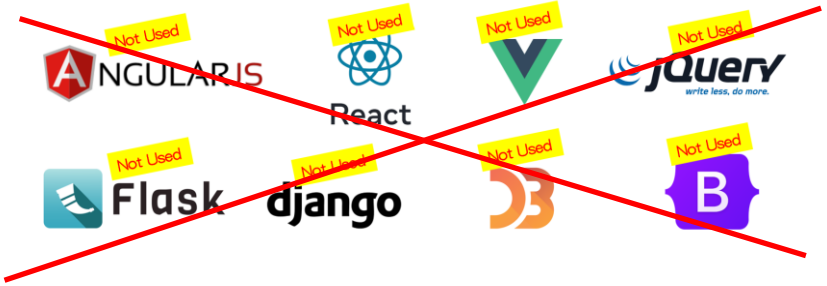
- PLC / ラダーロジック システムを SlowPy へ自動変換できる
- ハードウェアの複雑なシーケンスを記述するのに向いている方法かも???
  (FPGA が HDL で成功しているから SlowPy でもやってみたら良さげだった)

⇒ このスライドの付録参照

長寿命設計（目標25年以上）

プラットフォーム型ライブラリやフレームワークの類は使わない

（jQuery とか使っていたけど排除した）



便利なライブラリはプラグイン経由で使う ⇒ 使える間は使ってあとは捨てる

- データベース：PostgreSQL, InfluxDB, CouchDB, MongoDB, …
- メッセージング：Redis, Kafka, AMQP, …

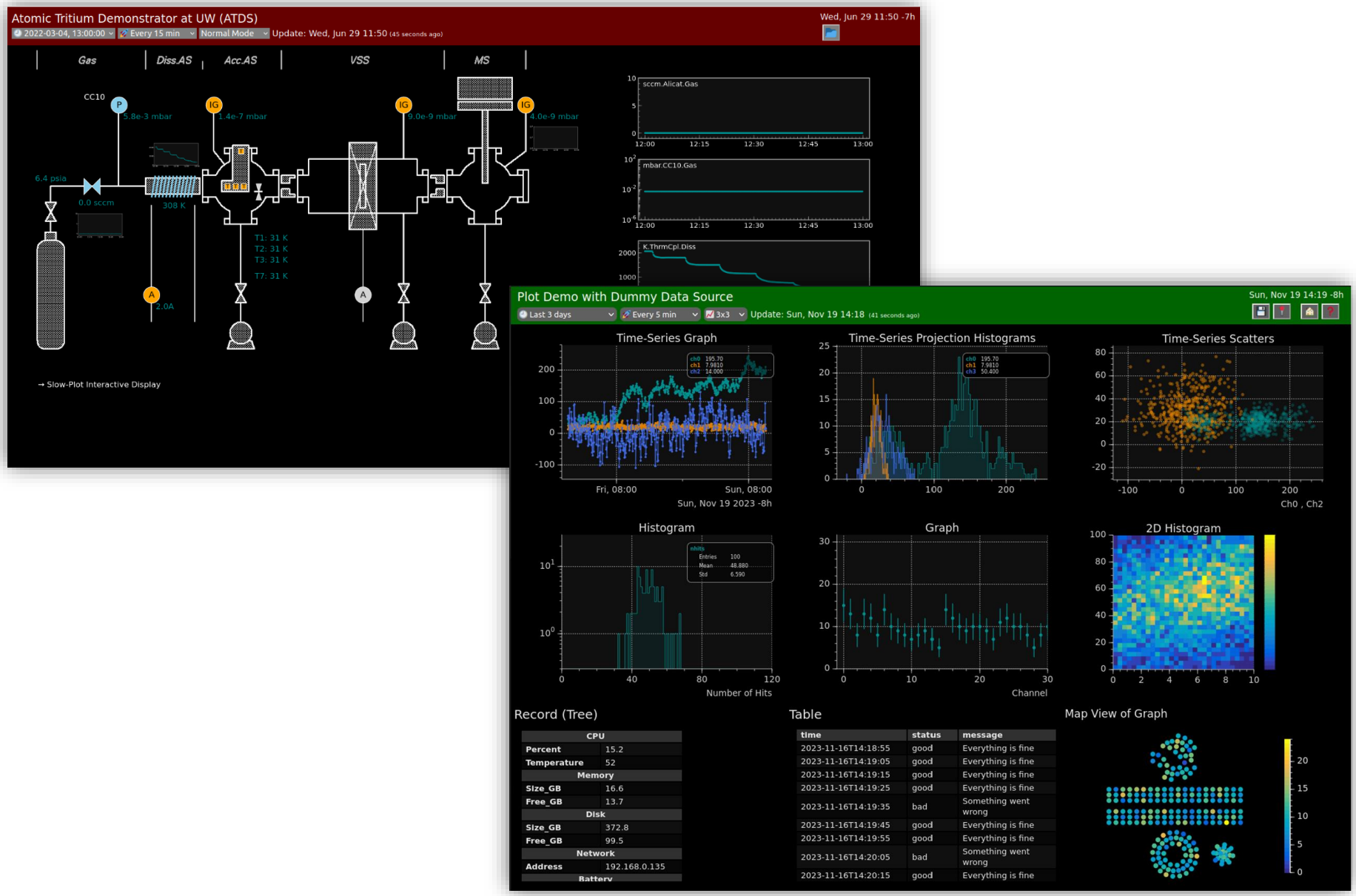
- デバイスを使うためのベンダ供給ライブラリ （SlowDash は Windows でも動きます. たぶん）

- 機械学習とか（scipy, lmfit, scikit-learn, …）
- Jupyter も直接は使っていない（Python を実行する「サービスクラス」の一実装）

コンテナやクラウドに使いやすく作るけれど特化や依存はしない

- 設定ファイルでマイクロサービスにもモノリシックにもできる
  （プロセス分割とスレッド分割）

「Grafana みたいに作れ」と言われたときのための ダークモード



開発状況と今後

24 年1月 “Nisqually”

- データベースに保存されたデータのビジュアライゼーション

24年7月 “Snoqualmie”

- コントロールの Python スクリプティング

昨日 “Skykomish”

- 共通コントロール部品の実装
- 複雑なコントロールの実装方法の模索
- Jupyter 接続

SPADI-A WG3 に居候中
(mattermost WG3-WebDisplay)

要望, アイディア, 共同開発は大歓迎です

このあと：主に解析系

- 埋め込みデータ加工・データ変換（テーブルデータをヒストグラムで表示とか）
- アラーム検出・通知・認知・マスク・リマインド・履歴管理, 障害予測(?)
- WebSocket でリアルタイム表示
- かっこいいレイアウト, 数千チャンネルの表示とか

GitHub: https://github.com/slowproj/slowdash
日本語マニュアル: https://slowproj.github.io/slowdash/#FirstStep-JP

SlowDash から見た

産業標準：PLC とラダーロジック

コントロールロジックの実装方法を考える

進行中プロジェクト：60年前の加速器制御システムを近代化する

これを iPad (SlowDash) に置き換えたい...

(30年前のアップグレードでかなりの部分が VAX と PLC に置き換えられた)

基本的には, たくさんのスイッチ, ノブ, 表示板と, 装置動作の「シーケンス」

今日のテーマ(半分遊び)

全部違う多チャンネル並列システムの「シーケンス」をどうやって記述すればいいか

まずは勉強：産業用コントロールシステム

ふつうは，こういうやつが機器を制御している

これを Web ページ（SlowDash）に
置き換えたい

中身はこんな感じ

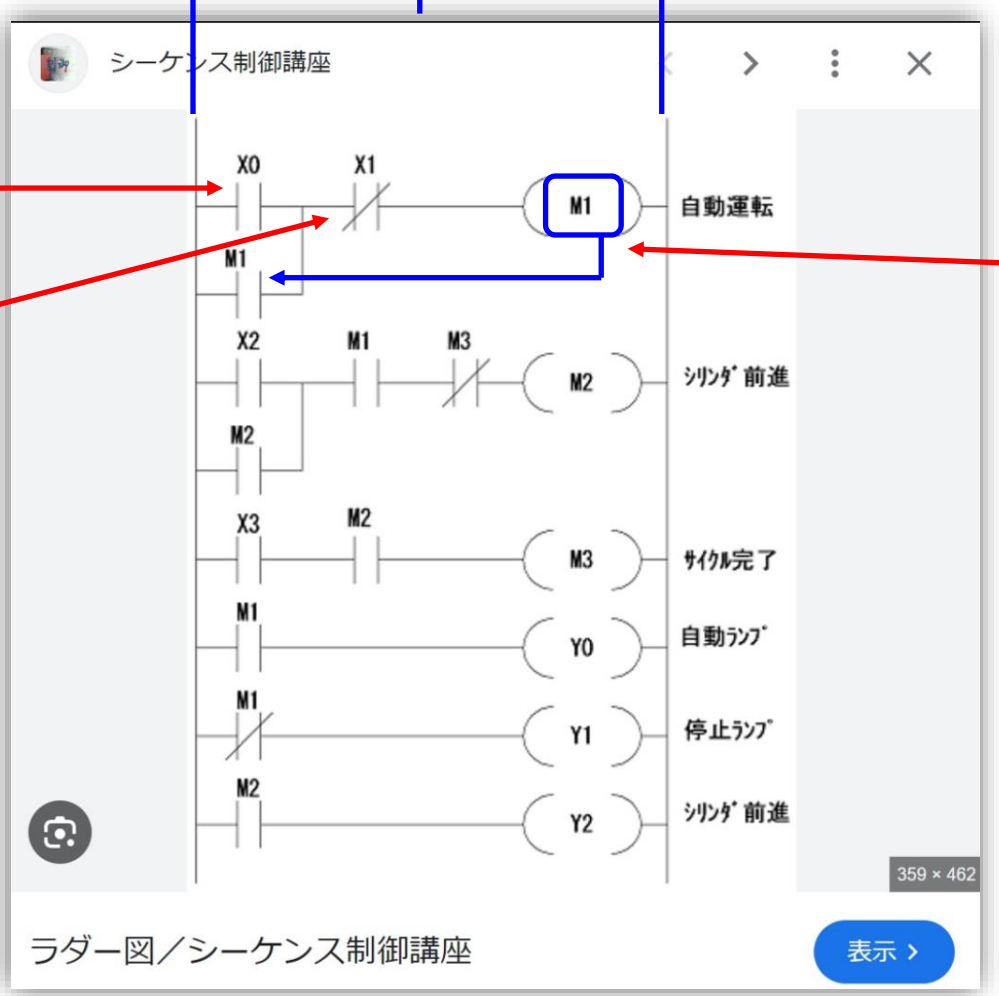
UI

リレースイッチの嵐と
それを制御するもの（PLC）

PLC（Programmable Logic Controller）：ラダーロジックでシーケンスを記述する

ラダーロジック：PLC のシーケンス記述

両端に電源がつながっているイメージ



押すと ON になる
スイッチ（開始）

押すと OFF になる
スイッチ（停止）

直列に並べると AND
並列に並べると OR

一度運転状態になると
「開始」を押し続けるのと
同じ効果

「停止」を押すと止まる

⇩

状態の維持と遷移
（ステートマシン !!!）

つまり

- 梯子の左半分で入力に対する論理演算
- 右端が出力. 出力は次の瞬間の入力にも使える
- 梯子の横棒は（概念的に）並列実行. それをたくさん並べる

} これって聞いたことがある…

ラダーロジックはこれと同じ?

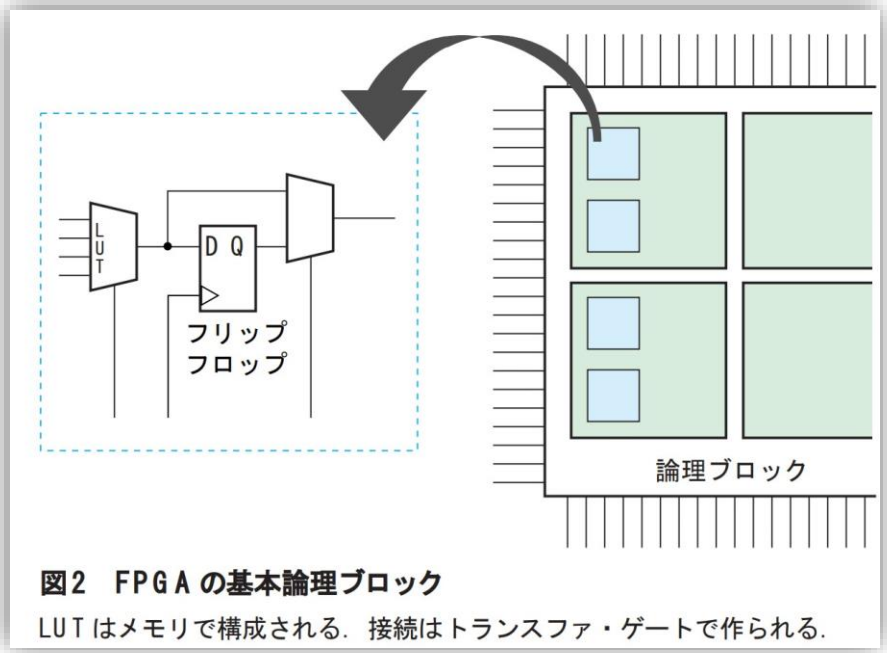


図2　FPGA の基本論理ブロック

LUT はメモリで構成される．接続はトランスファ・ゲートで作られる．

ラダーロジックがいまだに広く使われている
（ソフトウェアを知らなくてもプログラムが簡単らしい）

⇩

←これの集合でロジックを記述するのは
並列同時制御に向いている?

⇩

大規模な「これ」の集合を記述するには
HDL（Verilog とか）がいいと歴史が示している

⇩

SlowDash でも HDL 風にロジックを書きたい
（ループとかダサい）

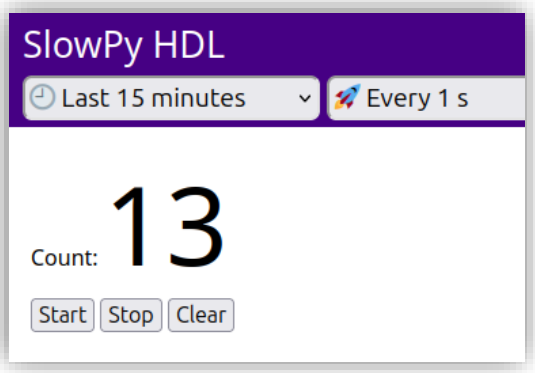
SlowPy-HDL: SlowDash の Python スクリプトを HDL 風に書く（半分遊び）

- シーケンス記述にたぶん向いている
- 既存のラダーロジックをそのまま変換できる

- 無理に使わなくてもよい．普通の Python で書いても全く構わない

SlowPy-HDL 1/2: Verilog だったら

タイマーカウンタを作ってみる



Start を押すと
カウントアップ
していくアプリ

GUI 要素とスクリプト変数の
バインディング
(前回やったとこ)

```
import slowpy.control as spc

ctrl = spc.ControlSystem()
start_btn = ctrl.value(initial_value=False).oneshot()
stop_btn = ctrl.value(initial_value=False).oneshot()
clear_btn = ctrl.value(initial_value=False).oneshot()
display = ctrl.value()

def _export():
    return [
        ('start', start_btn.writeonly()),
        ('stop', stop_btn.writeonly()),
        ('clear', clear_btn.writeonly()),
        ('display', display.readonly())
    ]
```

制御変数間のロジックを記述
Verilog で書くとこんな感じ
(RESET は別で)

```
module Counter(clock, start, stop, clear, count);
    input clock;
    input start;
    input stop;
    input clear;
    output reg[7:0] count;
    reg running;

    always @(posedge clock)
    begin
        if (stop == 1'b1)
            running <= 1'b0;
        else if (start == 1'b1)
            running <= 1'b1;
    end

    always @(posedge clock)
    begin
        if (clear == 1'b1)
            count <= 8'd0;
        else if (running == 1'b1)
            if (count == 8'd59)
                count <= 8'd0;
            else
                count <= count + 8'd1;
    end
endmodule
```

動作状態

動作状態
に基づく
カウンタ

SlowPy-HDL 2/2: Python でやってみる（半分遊び）

SlowPy-HDL によるロジック記述（注：これはソフトウェアです）

```
from slowpy.control.hdl import *

class CounterModule(Module):
    def __init__(self, clock, start, stop, clear, count):
        super().__init__(clock)

        self.start = input_reg(start)
        self.stop = input_reg(stop)
        self.clear = input_reg(clear)
        self.count = output_reg(count)
        self.running = reg()

        self.count <= 0
        self.running <= False

    @always
    def startstop(self):
        if self.stop:
            self.running <= False
        elif self.start:
            self.running <= True

    @always
    def update(self):
        if self.clear:
            self.count <= 0
        elif self.running:
            if self.count == 59:
                self.count <= 0
            else:
                self.count <= int(self.count) + 1
```

入出力に繋がった「レジスタ」

RESET 相当

クロックごとに実行される「プロセス」

Verilog で書いたやつ（前ページ）

```
module Counter(clock, start, stop, clear, count);
    input clock;
    input start;
    input stop;
    input clear;
    output reg[7:0] count;
    reg running;

    always @(posedge clock)
    begin
        if (stop == 1'b1)
            running <= 1'b0;
        else if (start == 1'b1)
            running <= 1'b1;
    end

    always @(posedge clock)
    begin
        if (clear == 1'b1)
            count <= 8'd0;
        else if (running == 1'b1)
            if (count == 8'd59)
                count <= 8'd0;
            else
                count <= count + 8'd1;
    end
endmodule
```

SlowPy-HDL （半分遊び）

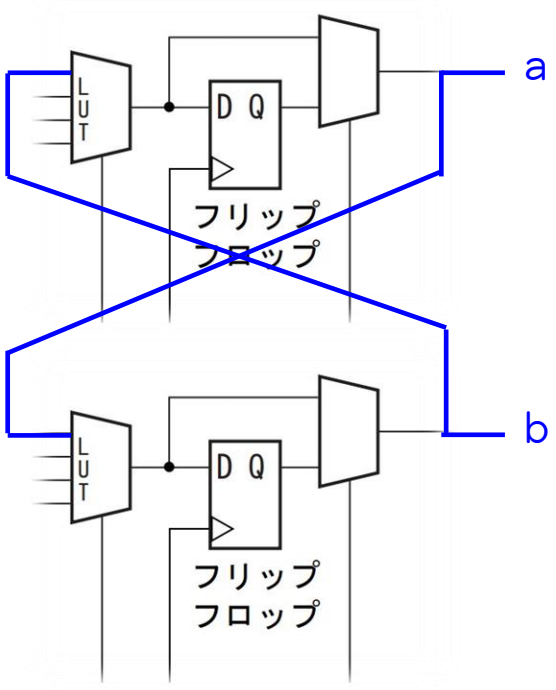
ちゃんと HDL 的にふるまう

こうやって書くと a と b の値を毎回入れ替える

```
class TestModule(Module):
    def __init__(self, clock, a, b):
        super().__init__(clock)
        self.a = output_reg(a)
        self.b = output_reg(b)

        self.a <= 'A'
        self.b <= 'B'

    @always
    def swap_ab(self):
        self.a <= self.b
        self.b <= self.a
```

“ソフトウェア的”なら
両方 B になる

SlowPy の「レジスタ」には
ラッチがエミュレートされている



SlowPy-HDL の実際の動作

1. 入力レジスタ値を更新, 保持（デバイスからの読み出しなど）
2. 全ての「@always プロセス」を実行
3. 出力レジスタ値を更新（デバイスへの書き込みとか）
4. 寝る
5. 1 に戻る

論理的「クロックエッジ」

関数が順番に実行されていて
本当に同時ではない点は
配線遅延だと思えばよい

SlowPy-HDL: コード全体

制御変数バインディング（外部社会とつながる）

```
import slowpy.control as spc

ctrl = spc.ControlSystem()
start_btn = ctrl.value(initial_value=False).oneshot()
stop_btn = ctrl.value(initial_value=False).oneshot()
clear_btn = ctrl.value(initial_value=False).oneshot()
display = ctrl.value()

def _export():
    return [
        ('start', start_btn.writeonly()),
        ('stop', stop_btn.writeonly()),
        ('clear', clear_btn.writeonly()),
        ('display', display.readonly())
    ]
```

ここに入る

main() 相当: インスタンス化と実行

```
clock = Clock(Hz=1)

counter = CounterModule(
    clock,
    start = start_btn,
    stop = stop_btn,
    clear = clear_btn,
    count = display
)

clock.start()
```

制御変数間のロジック（HDL 風）

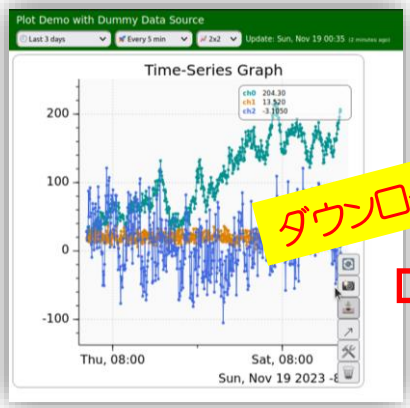
```
from slowpy.control.hdl import *

class CounterModule(Module):
    def __init__(self, clock, start, stop, clear, count):
        super().__init__(clock)

        self.start = input_reg(start)
        self.stop = input_reg(stop)
        self.clear = input_reg(clear)
        self.count = output_reg(count)
        self.running = reg()

        self.count <= 0
        self.running <= False

    @always
    def startstop(self):
        if self.stop:
            self.running <= False
        elif self.start:
            self.running <= True

    @always
    def update(self):
        if self.clear:
            self.count <= 0
        elif self.running:
            if self.count == 59:
                self.count <= 0
            else:
                self.count <= int(self.count) + 1
```

Appendix

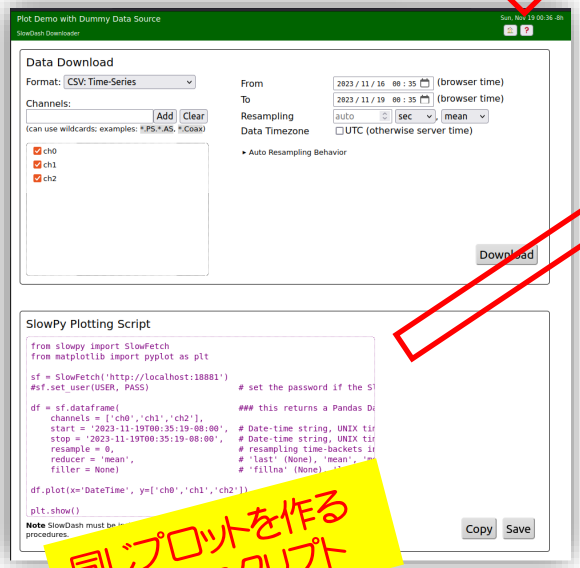
データ解析モジュール（試行錯誤中）

夢：ユーザが Jupyter-Notebook で適当に書いたものを（ほぼ）そのまま走らせたい

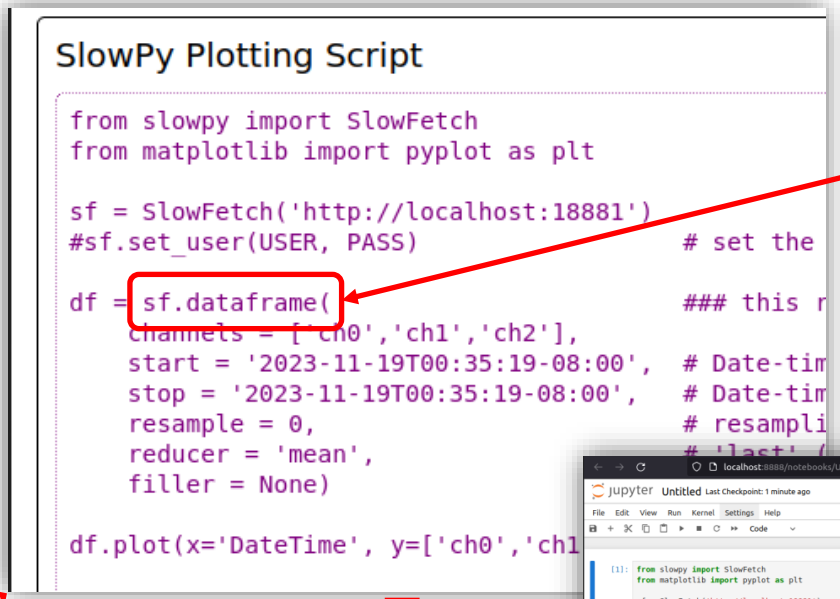
Step 1: データに Python から簡単にアクセスできるようにする



ダウンロードボタン

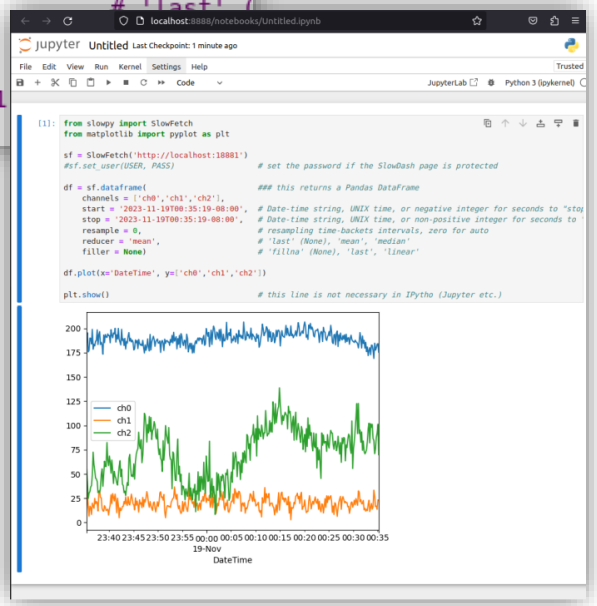


同じプロットを作る Python スクリプト



Pandas の DataFrame を返す

Jupyter にコピペ

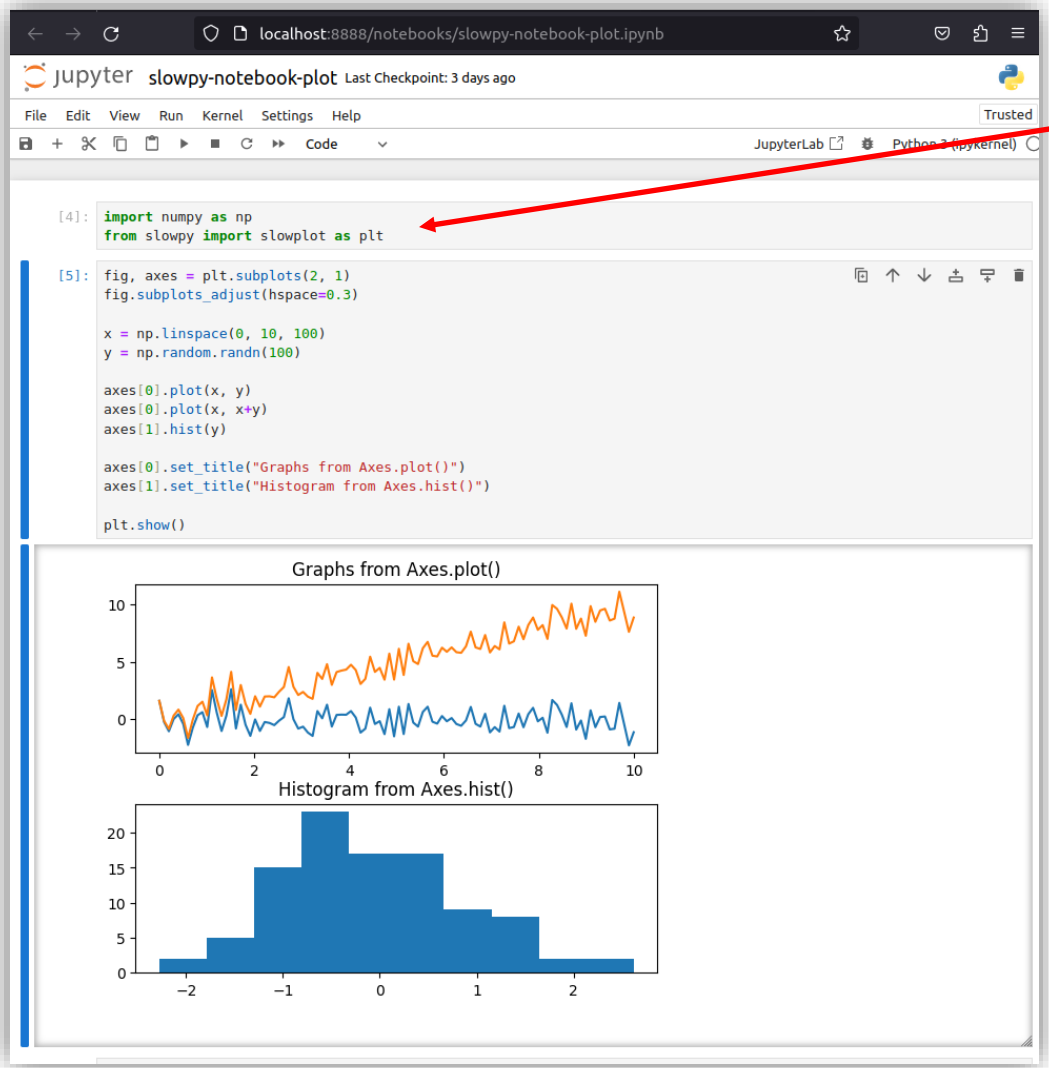


データ解析モジュール（試行錯誤中）

夢：ユーザが Jupyter-Notebook で適当に書いたものを（ほぼ）そのまま走らせたい

Step 2: ユーザに好きなように解析をしてもらう.

NumPy とか SciPy とか Matplotlib とか scikit-learn でも Stan でも



ただし，この行だけちょっと違う

```
from matplotlib import pyplot as plt
```

⇩

```
from slowpy import slowplot as plt
```

そうすると SlowPy が操作を横取り

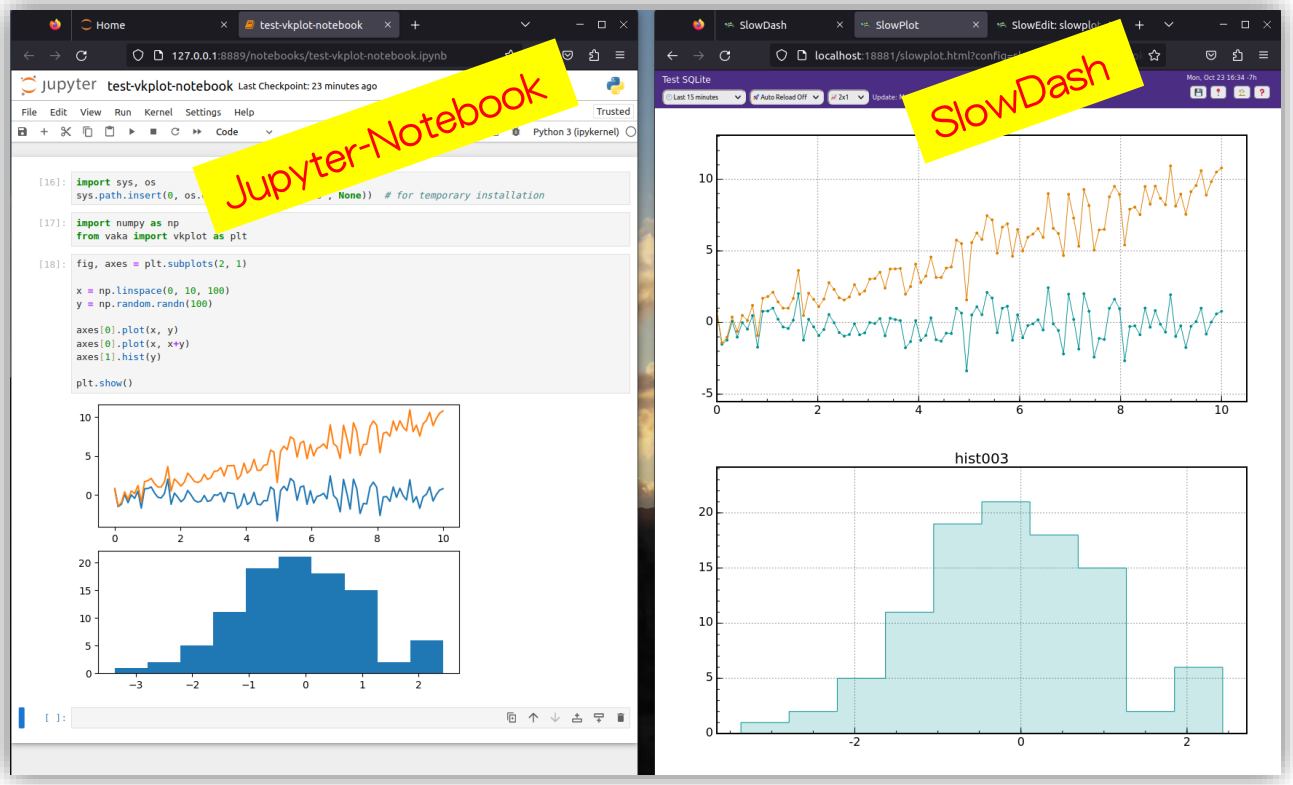
① Matplotlib のプロットと同時に
データをデータベースにも書き込む

② Matplotlib と同じレイアウトの
SlowDash 設定ファイルを生成

データ解析モジュール（試行錯誤中）

夢：ユーザが Jupyter-Notebook で適当に書いたものを（ほぼ）そのまま走らせたい

Step 3: ユーザ解析プログラムを SlowDash モジュールとして（ほぼ）そのまま登録



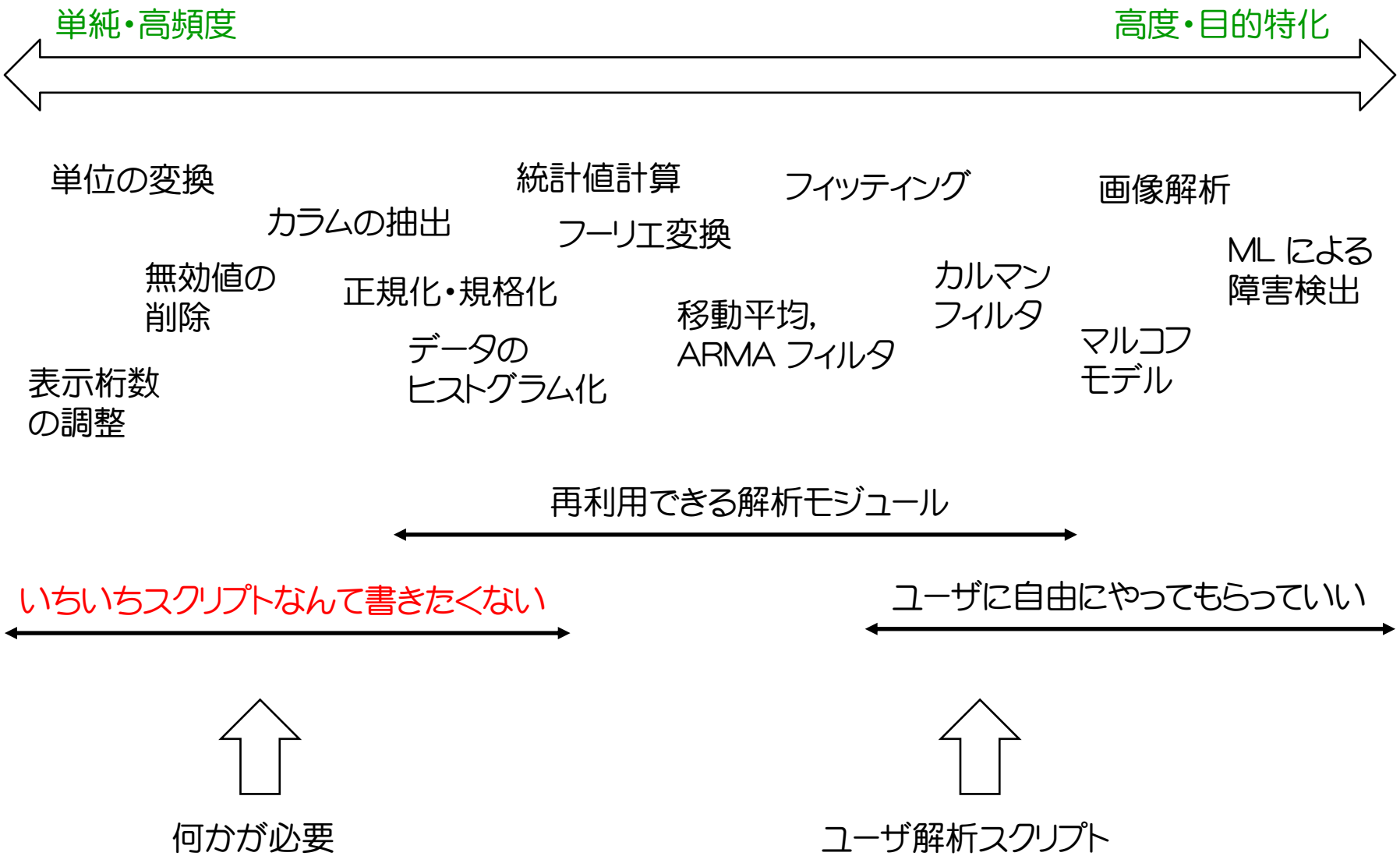
こっちを変えると
（ファイル保存が必要）

こっちも変わる
（リロードが必要）

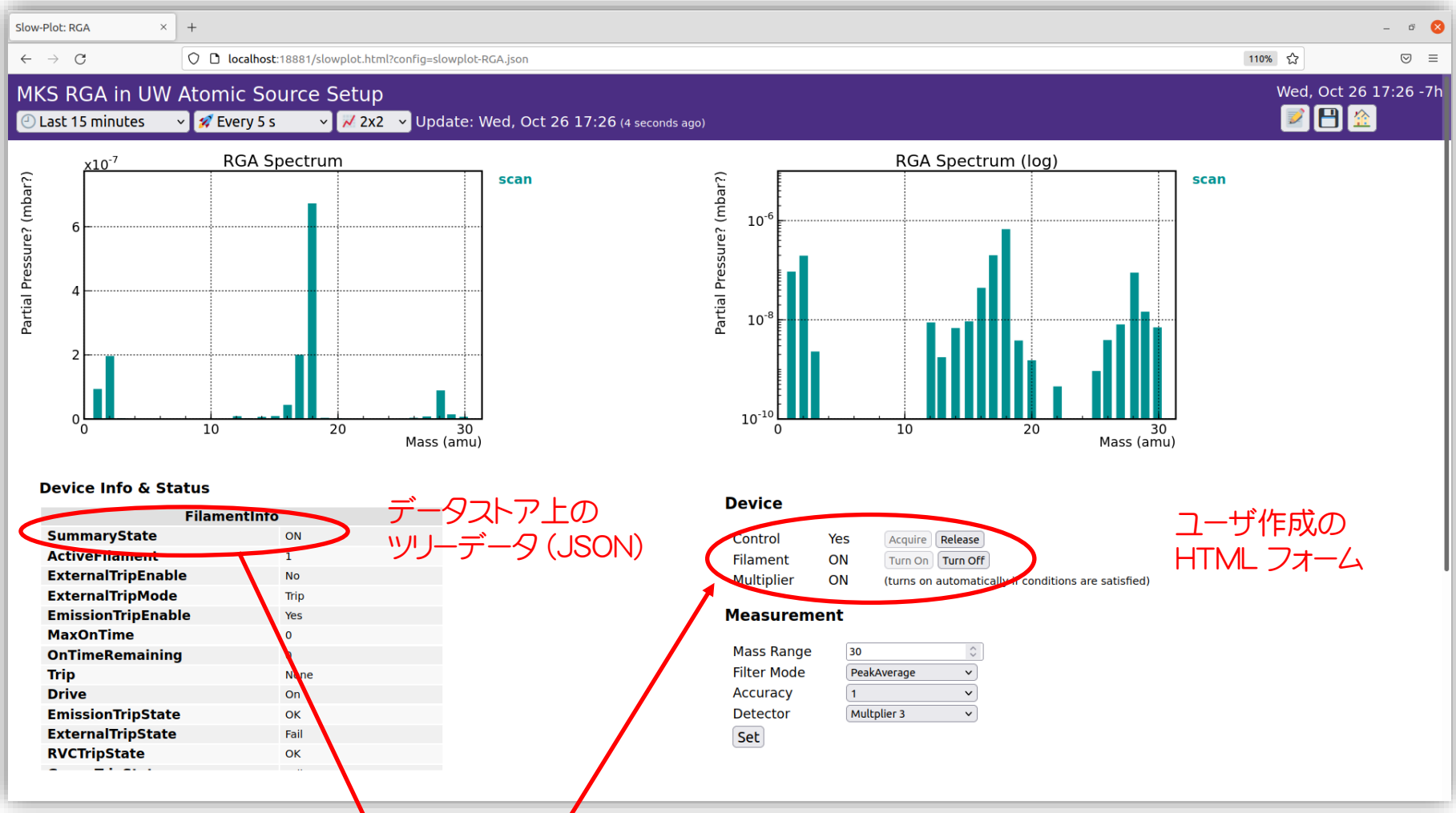
現時点では，繰り返し実行のセルを
関数にしてもらう必要がある
（セルを連打する代わりに SlowDash が関数を呼ぶ）

解析スクリプト：簡単な解析が簡単にできない

解析といってもいろいろある. データストア上のデータはそのまま使えないことが多い



難しい簡単な解析の例



- データ値を抽出して HTML に表示したい
- データ値を抽出して入力フィールドの初期値に使いたい
- データ値によってボタンの Enable / Disable をコントロールしたい

データ解析ファンクタ（試行錯誤中）

汎用ファンクタをチェーンにして必要な機能を実装する


```
"status['FilamentInfo/SummaryState']->match('OFF')
```

データストア上の
JSON オブジェクト
(Tree 型データ)

->get() ファンクタの別名

Tree 型 ⇒ Scalar<string> 型

正規表現ファンクタ

Scalar<string> 型 ⇒ Scalar<bool> 型

- 抽出ファンクタでいろいろなデータ構造を配列やスカラに変換
- 主に配列データにアルゴリズムを実装する（統計値, フーリエ変換, 時系列フィルタ, 等）
- スカラファンクタはデータの整形にも使える（ ->format('%.3f MB/s') とか）

今後追加予定
- 操作対象選別フィルタ
- 複数データ入力

レンダリング時データ加工（Vue.js 風）

Device Info & Status

FilamentInfo	
SummaryState	ON
ActiveFilament	1
ExternalTripEnable	No
ExternalTripMode	Trip
EmissionTripEnable	Yes
MaxOnTime	0
OnTimeRemaining	0
Trip	None
Drive	On
EmissionTripState	OK
ExternalTripState	Fail
RVCTripState	OK

Device

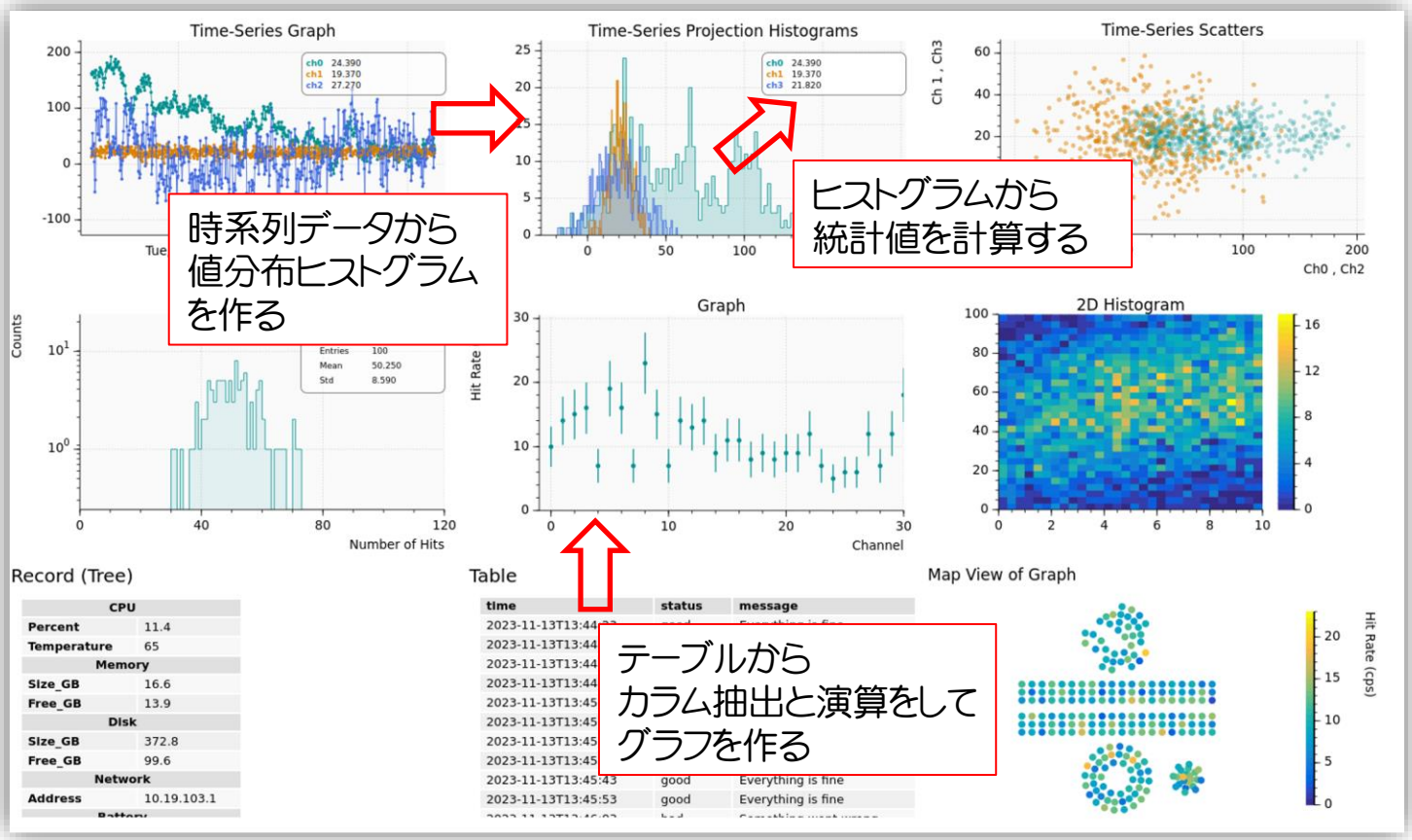
Control Yes Acquire Release
Filament ON ① Turn On Turn Off ②
Multiplier ON ③ (turns on automatically if conditions are satisfied)

Measurement

Mass Range 30
Filter Mode PeakAverage
Accuracy 1
Detector Multplier 3
Set

```
<h3>Device</h3>
<table>
  <tr>
    <td>Control</td>
    <td><span sd-value="status['Control/Controlling']">--</span></td>
    <td>
      <input type="submit" name="AcquireControl" value="Acquire" sd-enabled="status['Control/Controlling']->match('No')">
      <input type="submit" name="ReleaseControl" value="Release" sd-enabled="status['Control/Controlling']->match('Yes')">
    </td>
  </tr>
  <tr>
    <td>Filament</td>
    <td><span sd-value="status['FilamentInfo/SummaryState']">--</span></td>
    <td>
      <input type="submit" name="FilamentOn" value="Turn On" sd-enabled="status['FilamentInfo/SummaryState']->match('OFF')">
      <input type="submit" name="FilamentOff" value="Turn Off" sd-enabled="status['FilamentInfo/SummaryState']->match('ON')">
    </td>
  </tr>
  <tr>
    <td>Multiplier</td>
    <td><span sd-value="status['MultiplierInfo/MultiplierOn']->replace('Yes','ON')->replace('No','OFF')">--</span></td>
    <td><span style="font-size:small">(turns on automatically if conditions are satisfied)</span></td>
  </tr>
</table>

<h3>Measurement</h3>
```

データ解析ファンクタの使い道（構想）



他にも

- 無効値除去・変換
- ビットフラグ値の文字列変換
- 時系列フィルタ（スムージングとか），フーリエスペクトル
- 異常値検出時のアラームアイコン表示

- ヒストグラムフィッティング？

スタック記憶域を持ったデータ解析ファンクタ（構想）

単純な例（セミコロンがスタックプッシュ演算）

二つの時系列データ Ch1 と Ch2 の差の時系列を作る

Ch1; Ch2; -> diff()

Ch1 と Ch2 をリサンプリングしてデータ点を揃えたうえで散布図を作る

Ch1 -> resample(bucket=10); Ch2 -> resample(bucket=10); -> graph()

要検討な例

ヒストグラムを作り，ピークをフィットして，相対ピーク幅（rms/mean）を求める（スタックマシン実装）

Ch1 -> hist() -> peakfit() -> get(“stats”) -> dup() -> get(“rms”); -> exch() -> get(“mean”); -> ratio()

JSON オブジェクトを返す　スタックの先頭データを複製　スタックの上2つを入れ替える

別案：
スタックマシンに外部記憶域（レジスタファイル）を追加し，出力を新しいデータチャネルのように見せる

Ch1 -> hist() -> peakfit() -> get(“stats”) -> Peak; Peak -> get(“rms”); Peak -> get(“mean”); -> ratio()

スタック外の名前付き記憶域（動的に作成された仮想データチャネル）

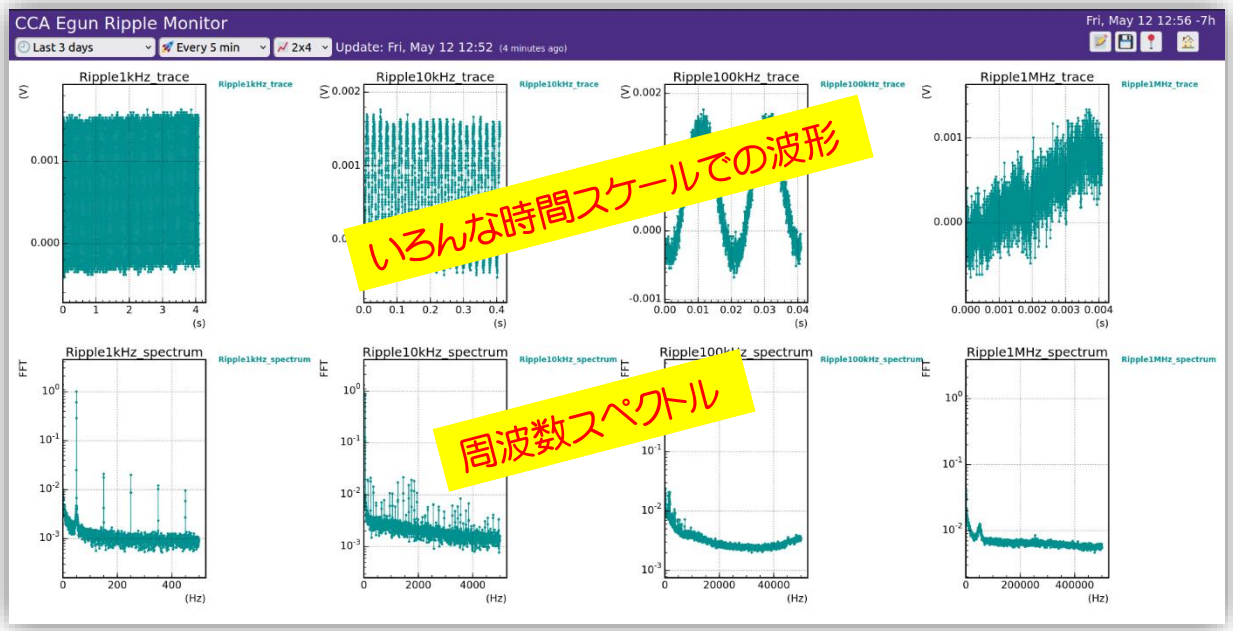
Mini-DAQ 例: オシロによる継続的ノイズモニタ

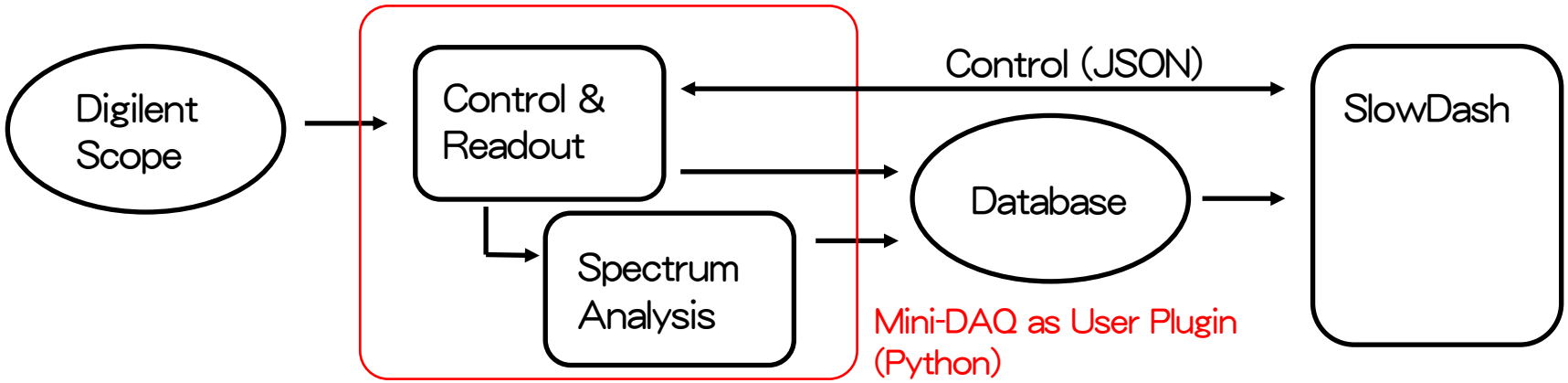
1300ドルの小型オシロスコープが便利に使える

（小型モデルはアカデミックで300ドル）

Analog Discovery Pro 3000 Series: Portable High Resolution Mixed Signal Oscilloscopes

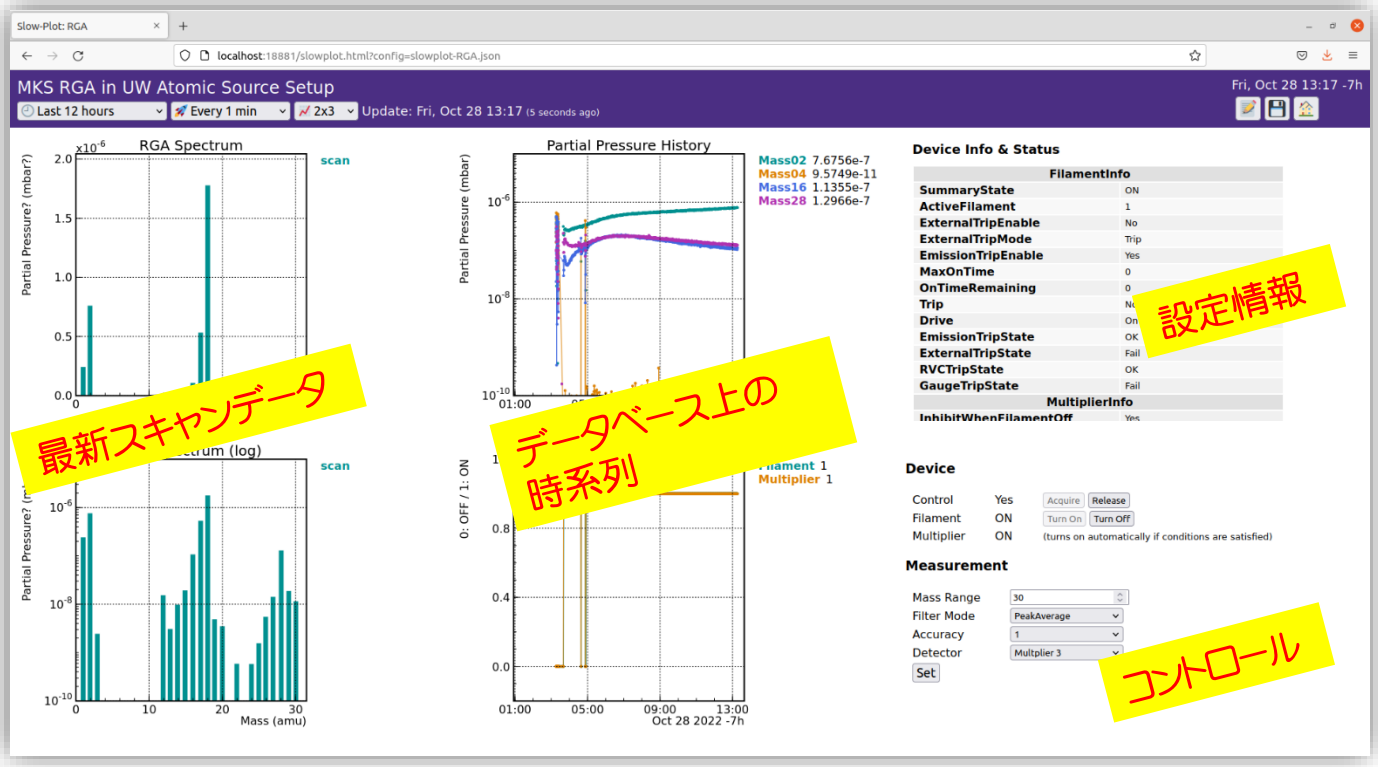
$1,295.00

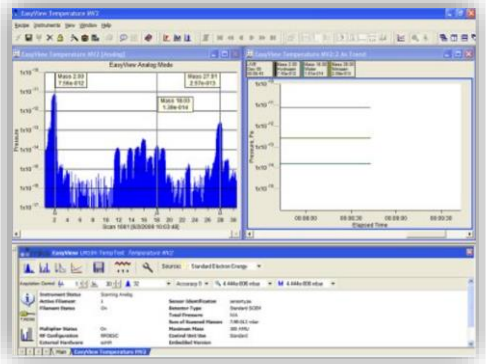




Mini-DAQ 例: 残留ガス分析器

付属の Windows のソフトウェアが使われていて, 使い勝手が悪かった





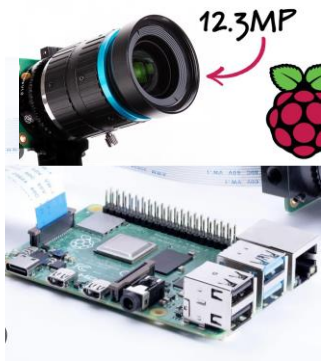
スペクトル分析をして時系列データとしてデータベースに書き込める

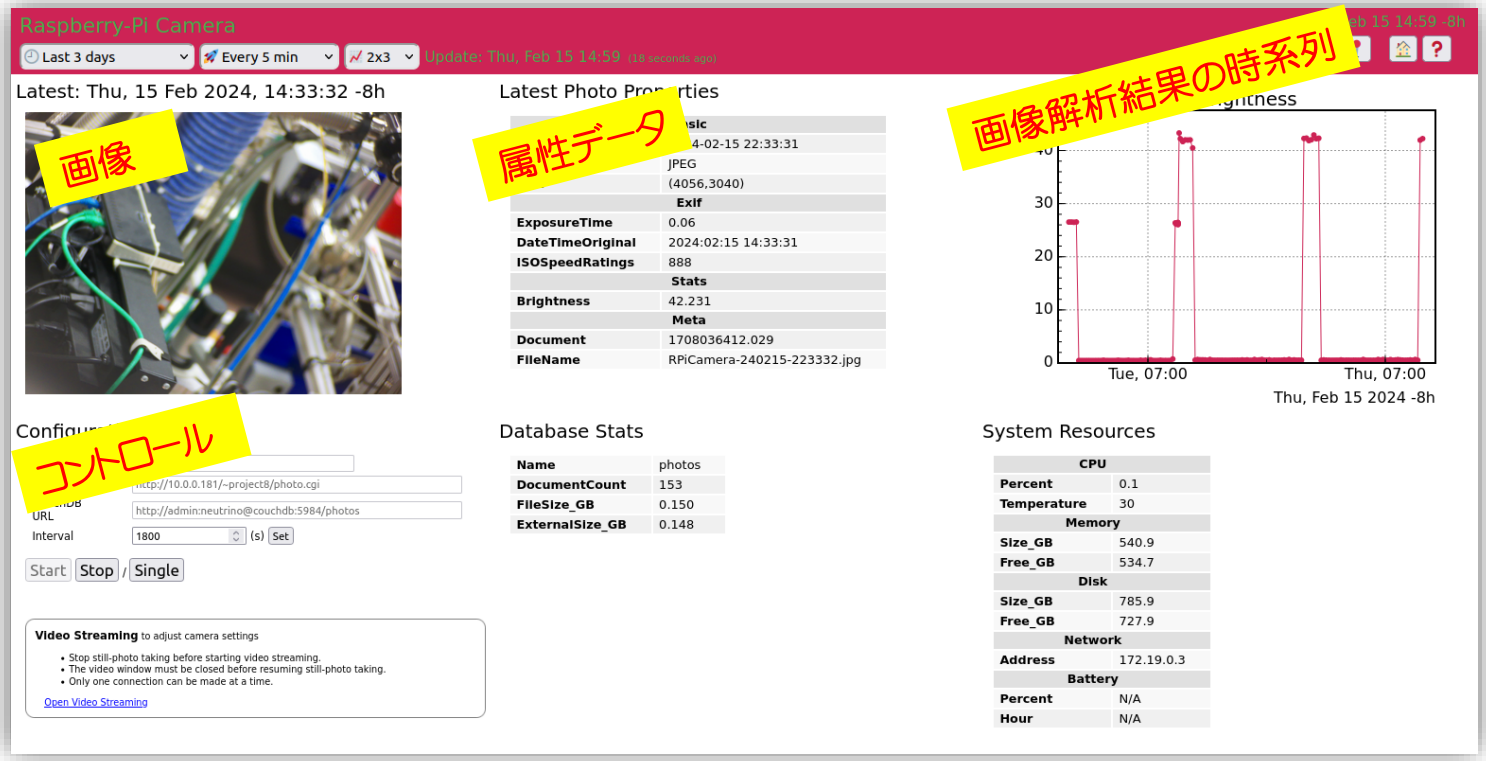
⇐ 製品付属 Windows ソフトウェア：悪くはないけれど, システム統合が難しい

TCP/IP コマンドが公開されているけれどそうすると UI が使えなくなる
Java アプレットも内蔵されているが話にならない

Mini-DAQ 例: Raspberry-Pi カメラと画像解析

画像データ(の時系列)も扱える





Python の画像解析や機械学習のライブラリがそのまま使える

- 変化検出, 異常検知
- パネル表示値認識
- …