# F3RP70-2Lを用いた異常値検知システムの検討

理研仁科センター　内山暁仁

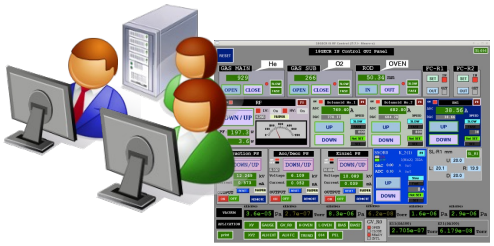
mailto:a-uchi@riken.jp

# 概要

- 加速器制御システムの概要
- RIBF制御系について
- F3RP70-2L
- 加速器パラメータの異常値検知
- 将来的な実装例
- まとめ

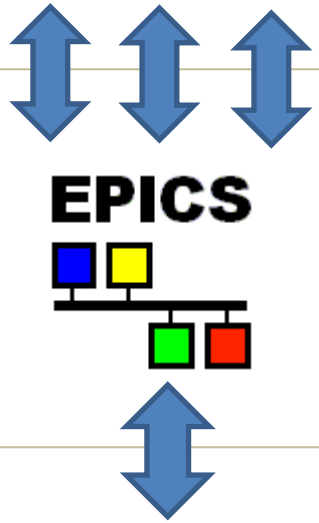
# 加速器制御システムの概要

加速器制御システムの三層モデル



## クライアントシステム(ユーザとのやりとり)

- オペレータインターフェース (OPI)
- データアーカイブシステム




## ミドルウェア(中間層)

- デバイスとクライアント間のプロトコルを統一させる
- EPICS (Experimental Physics and Industrial Control System)

## デバイス (ハードウェアとの相互接続)

- 主にデジタルやアナログで機器に命令を送ったり出力を受け取る.

# 加速器制御システムの概要

## 加速器制御システムの特徴

◎制御対象物が膨大な数ある 例：電磁石電源は1,000台弱

➡ 効率的な開発手法の必要性
プログラム共通部分を共有化

◎システム開発コスト 例：GUIパネルは~500

➡ コラボレーションが容易な仕組みの導入
例えばA施設で開発したシステムをB研究所でそのまま利用

◎高い信頼性の必要性

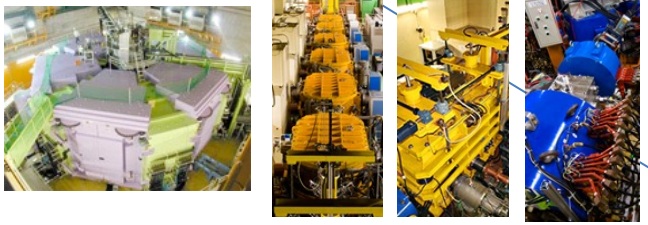
➡ 中間層を持たせコントローラへのアクセス低減
冗長性の仕組みの導入
例えばコントローラへ多数のクライアントがアクセスする

# RIBF制御系について

## EPICSを用いた分散制御システム

RIBF制御システムの範囲

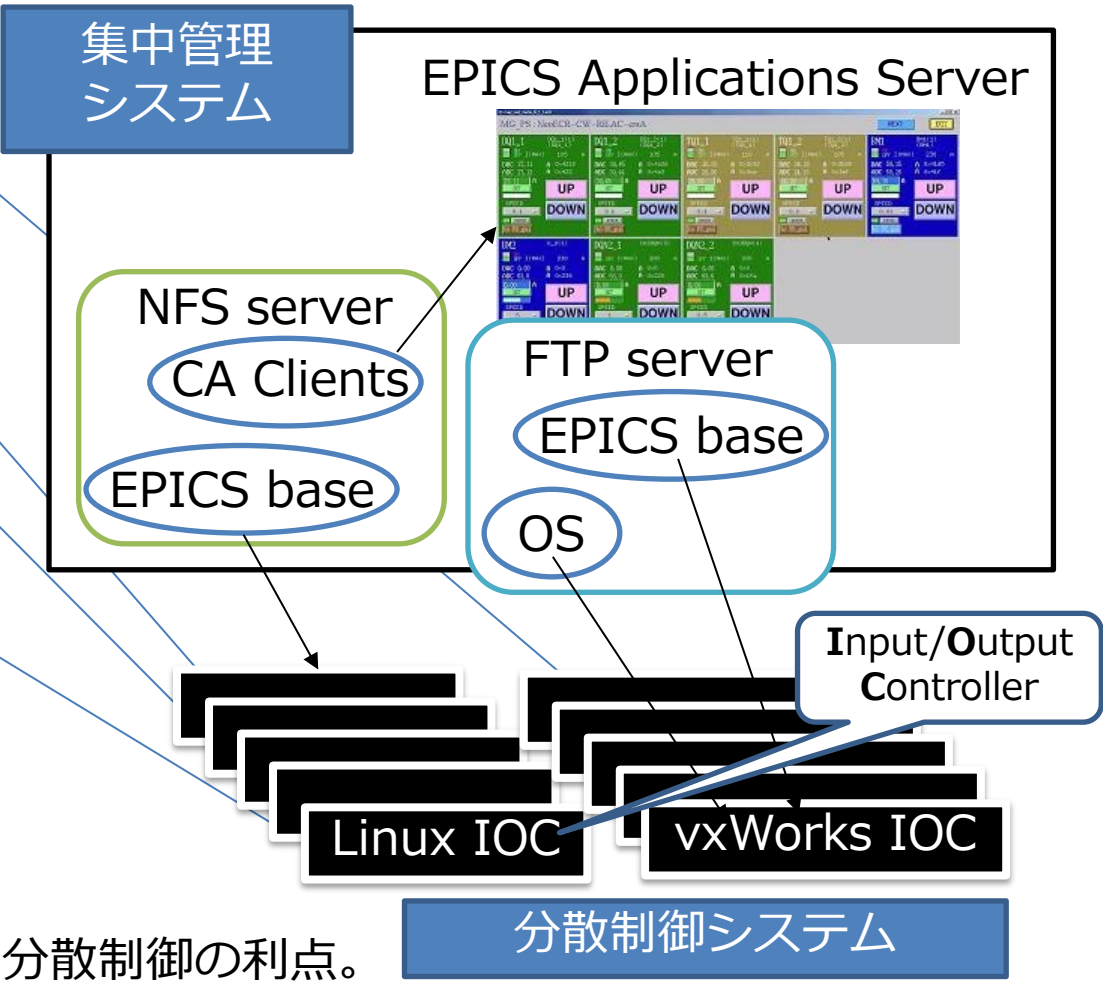
- 電磁石電源制御(<1,000)
- ビーム診断(<300)
- ECRイオン源
- ビーム安全システム
- 真空制御



他サービス

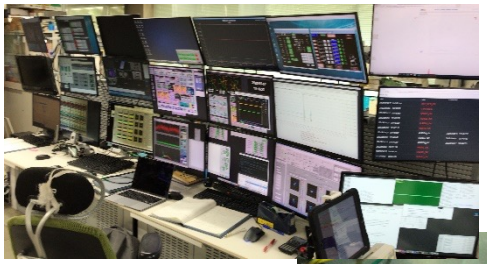
- オペレーショナルログシステム
- データアーカイブシステム
- アラームシステム

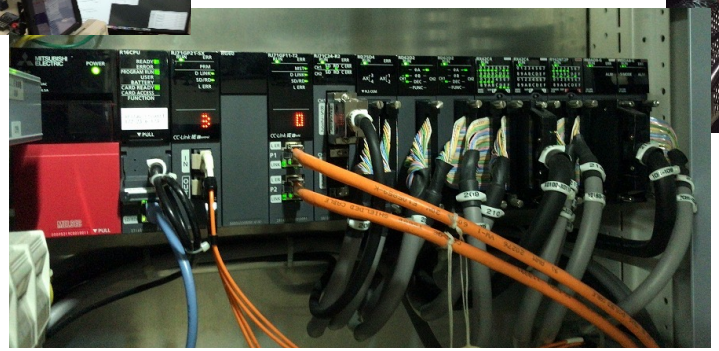
規模としては制御点数20万点以上。分散制御の利点。



# RIBF制御系について

- RIBF制御系ではデバイスとしてProgrammable Logic Controller(PLC)が多用されている。
- PLCはMELSEC, OMRON, 横河FA-M3等が利用されている。
- 通常のCPUモジュールはシーケンスCPUだが、 横河FA-M3はCPUとしてLinuxが走るモジュールが使えるため、多用されている。
- Linuxが走るFA-M3 CPUモジュールはJ-PARC, KEK, 筑波大, RCNP, Spring-8, その他でも使われている。



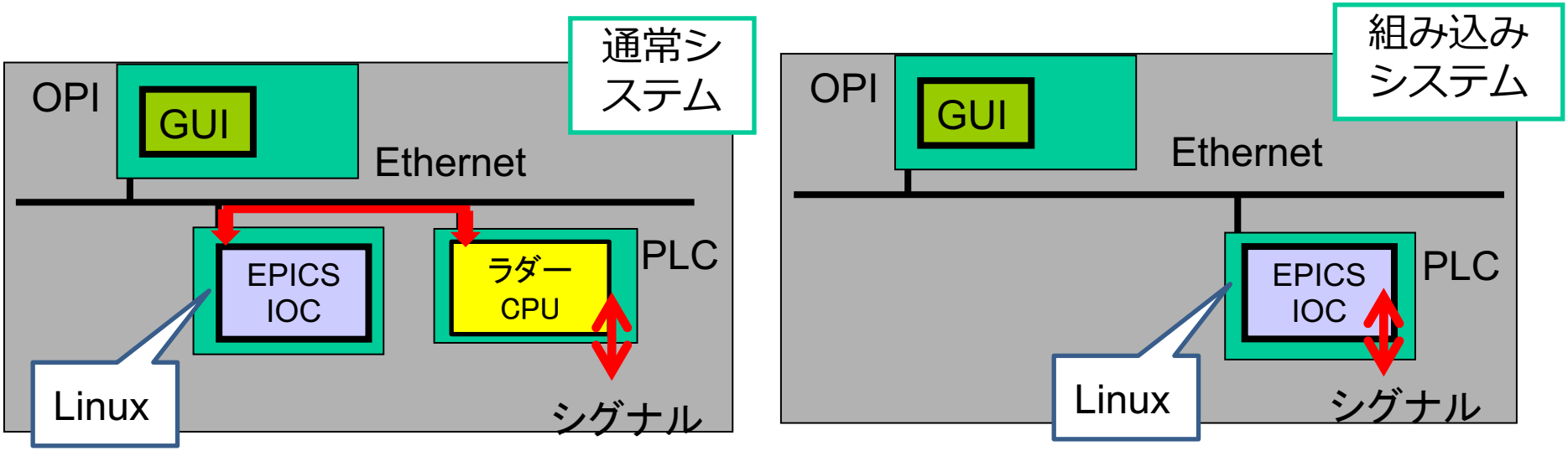


Linuxが走るCPU
モジュール

# RIBF制御系について

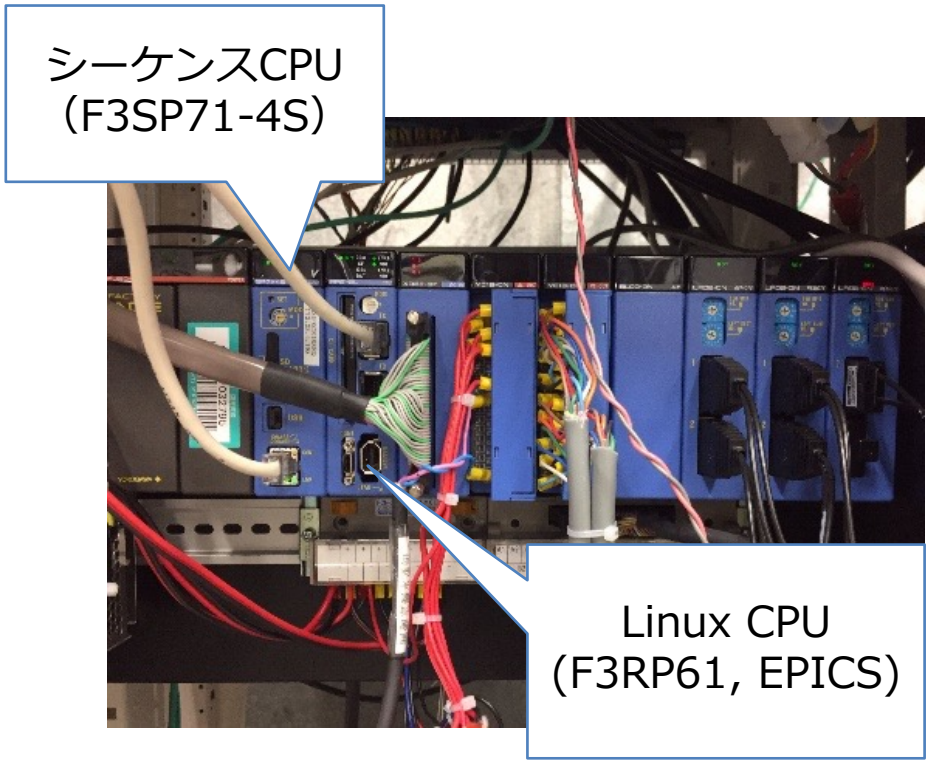
- RIBF制御系ではデバイスとしてProgrammable Logic Controller(PLC)が多用されている。
- PLCはMELSEC, OMRON, 横河FA-M3等が利用されている。
- 通常のCPUモジュールはシーケンスCPUだが、 横河FA-M3はCPUとしてLinuxが走るモジュールが使えるため、多用されている。
- Linuxが走るFA-M3 CPUモジュールはJ-PARC, KEK, 筑波大, RCNP, Spring-8, その他でも使われている。

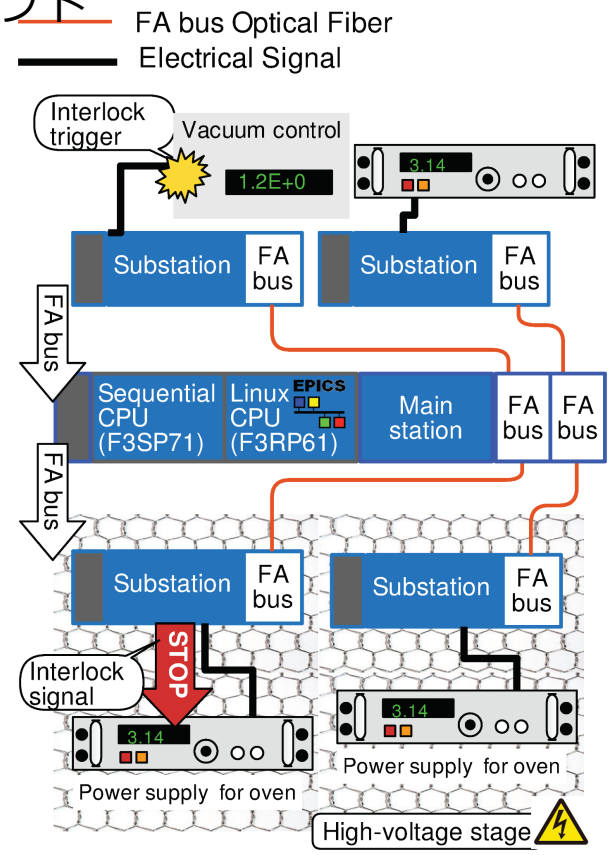
PLC CPUがEPICS IOC（Input/Output Controller）も兼ねることでシンプルでかつ見通しが良いシステムになる



# RIBF制御系について

- 重イオン超伝導線形加速器（SRILAC）制御系では、インターロックはシーケンスCPUとEPICSといったユーザインターフェースはLinux CPUを使っている
  - ◆ インターロックといった信頼性が必要な所にはシーケンスCPU，インターフェース(EPICS)に関してはLinux CPUを使うというコンセプト



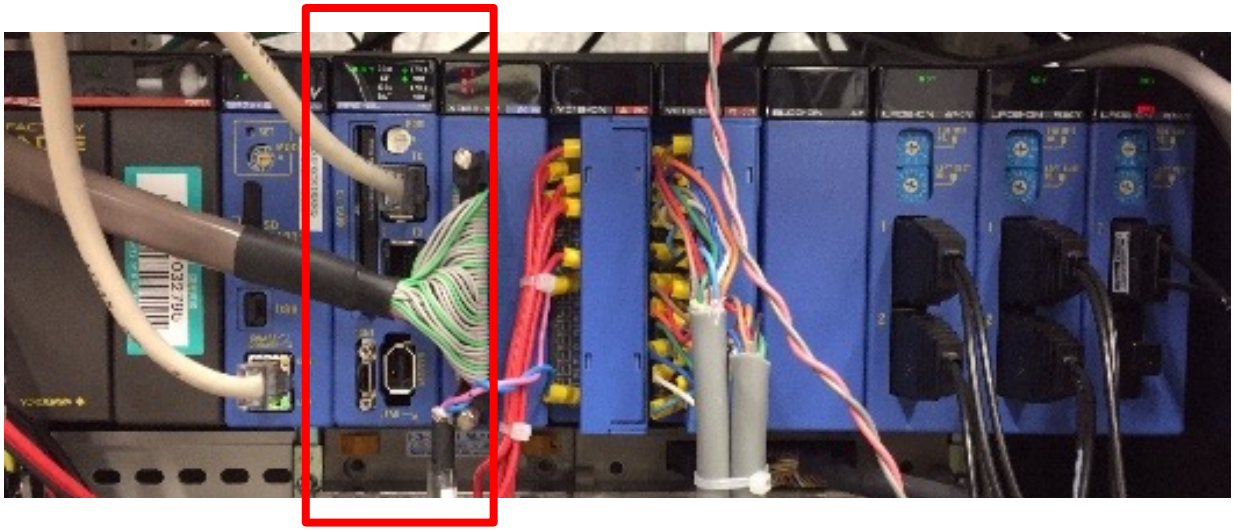
# RIBF制御系について

横河電機、FA-M3用Linux CPUは次のラインナップがあった

- F3PR61-2R/2L
- F3RP71-1R/2L

仕様

- CPU MPC8347E, 533MHz (PowerPC)
- OS Linux (kernel 2.6.26.8+2.6.26.8-rt16 ベース)
- Ethernet 10BASE-T/100BASE-TX (2ch)
- User SRAM 4MB(F3RP61-2Lのみ)



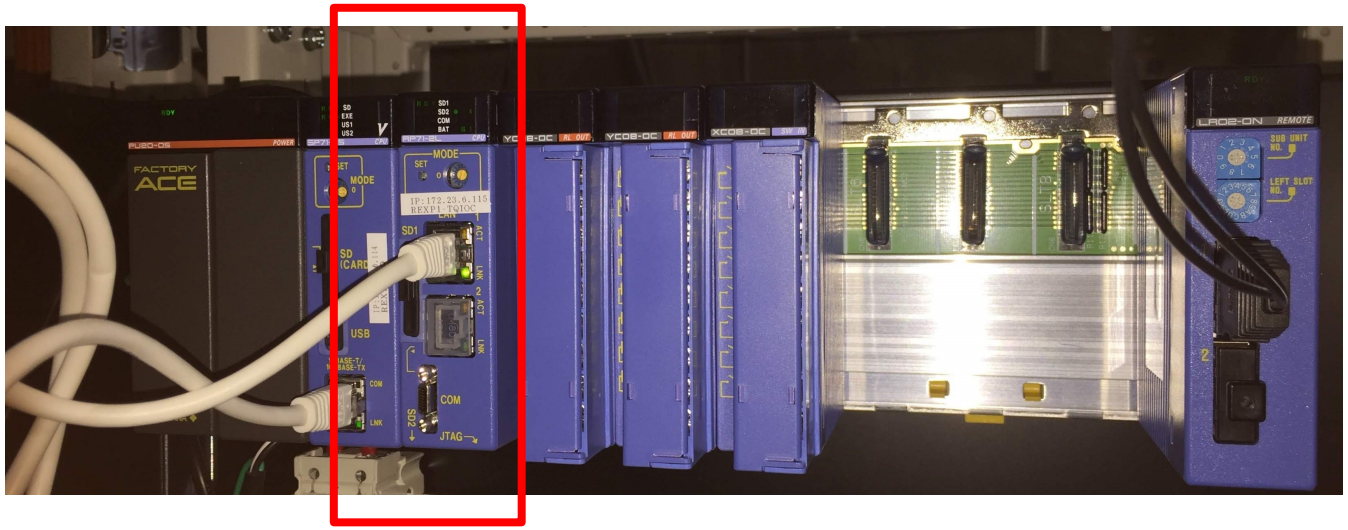
# RIBF制御系について

横河電機、FA-M3用Linux CPUは次のラインナップがあった

- F3PR61-2R/2L
- F3RP71-1R/2L

仕様

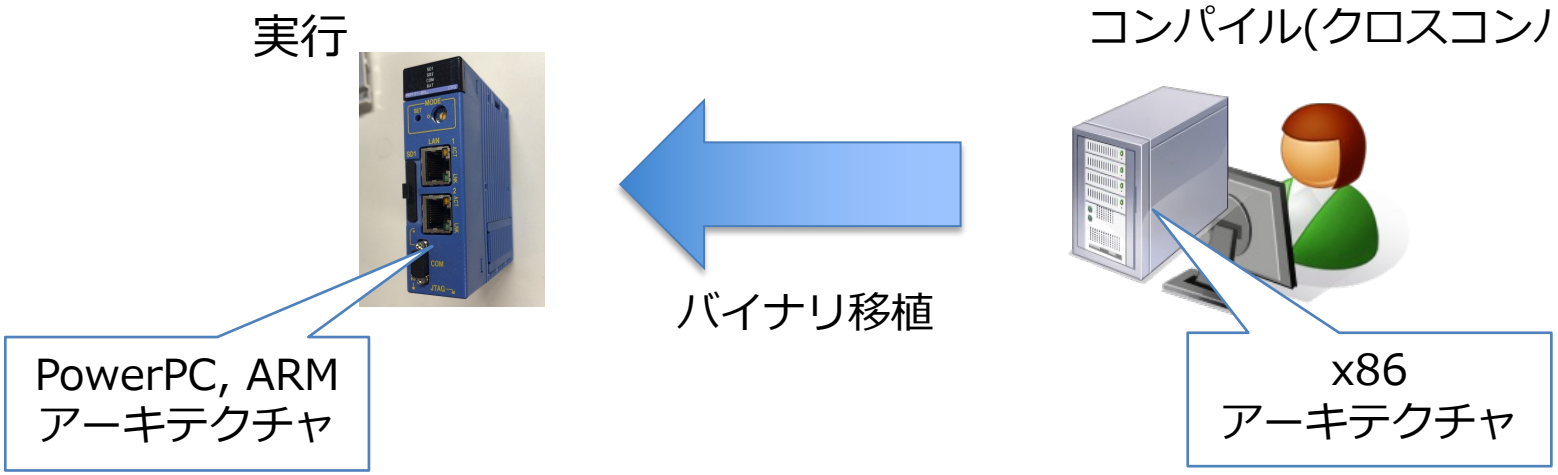
- CPU ARM Cortex-A9 MPCore (Dual Core, 866MHz)
- OS Linux 3.18, PREEMPT_RT patch
- Ethernet 10BASE-T/100BASE-TX/1000Base-T (2ch)
- User SRAM 8MB(F3RP71-2Lのみ)



# RIBF制御系について

横河電機、FA-M3用Linux CPUは次のラインナップがあった
- F3PR61-2R/2L
- F3RP71-1R/2L

F3RP61、F3RP71ともPowerPC、ARMとアーキテクチャは違うもののクロス開発が必要




開発環境と実行環境が異なるシステム
組み込みシステム分野ではオーソドックスな手法

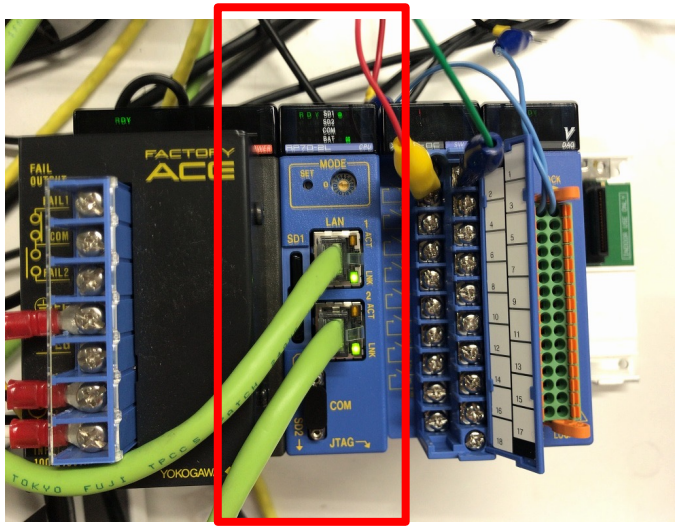
# F3RP70-2L

横河電機、FA-M3用Linux CPUは次のラインナップがあった

- F3PR61-2R/2L
- F3RP71-1R/2L
- F3RP70-2L

仕様

- CPU ARM Cortex-A9 MPCore (Dual Core, 866MHz)
- OS Ubuntu 18.04
- Ethernet 10BASE-T/100BASE-TX/1000Base-T (2ch)
- User SRAM 8MB




ハードウェアの仕様はF3RP71-2Lと同じ

# F3RP70-2L

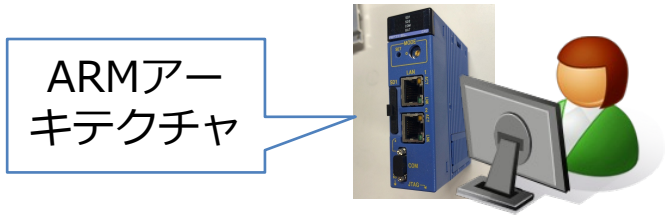
横河電機、FA-M3用Linux CPUは次のラインナップがあった

- F3PR61-2R/2L
- F3RP71-1R/2L
- F3RP70-2L

仕様

- CPU ARM Cortex-A9 MPCore (Dual Core, 866MHz)
- OS Ubuntu 18.04
- Ethernet 10BASE-T/100BASE-TX/1000Base-T (2ch)
- User SRAM 8MB

ネイティブコンパイル

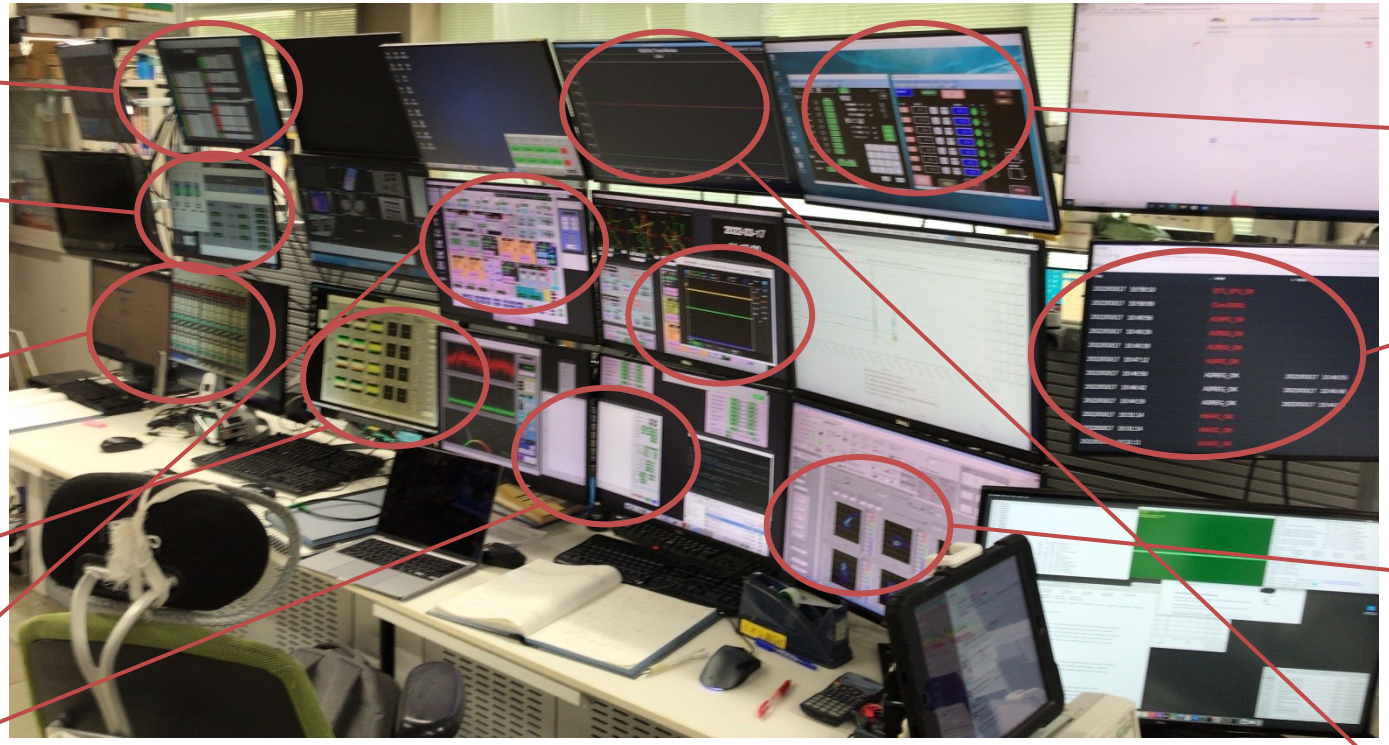



F3RP70-2Lを使うメリット

- オープンソースのライブラリをそのまま利用できるので、クロス開発の必要なし。
- アプリケーションの開発工程を大幅に削減する事ができる。
- Pythonでモジュールにアクセスするapiが色々揃っている。
- Ubuntuなので機械学習のアプリ開発がすぐに始められる。

# 加速器パラメータの異常値検知

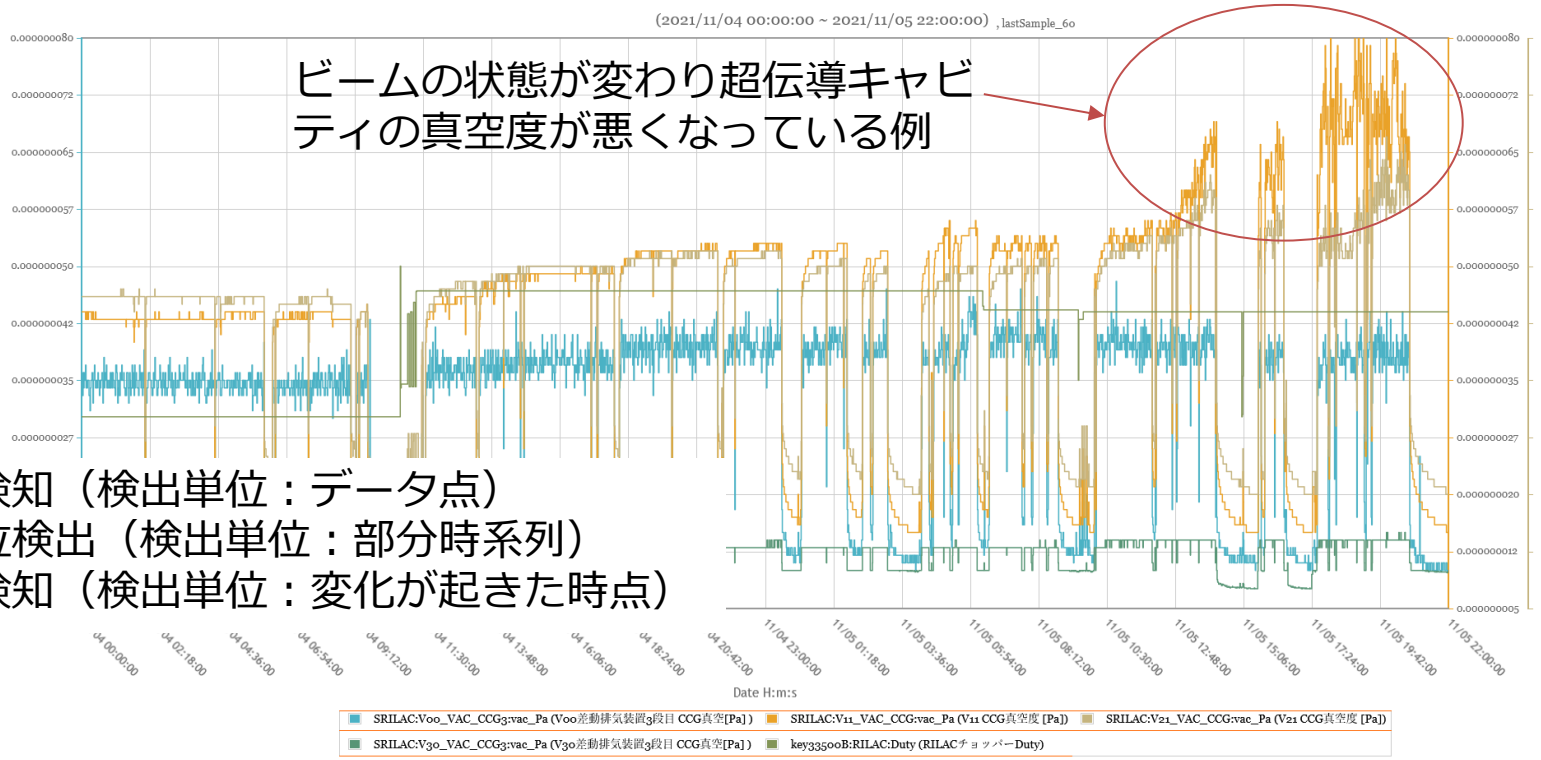
加速器を運転するためには、冷却水温度、電磁石電源の電流値、真空度等、モニタをすべきパラメータは相当数（例えばリニアック関係の温度だけでも300点以上）になる。




制御室のモニタだけでは監視することに限界がある。

# 加速器パラメータの異常値検知

収集したデータから、期待されるパターンとは異なった物体や出来事及び観測結果を（異常値）を検知し、アラームやインターロックを出すことができれば監視やオペレーションに有用。



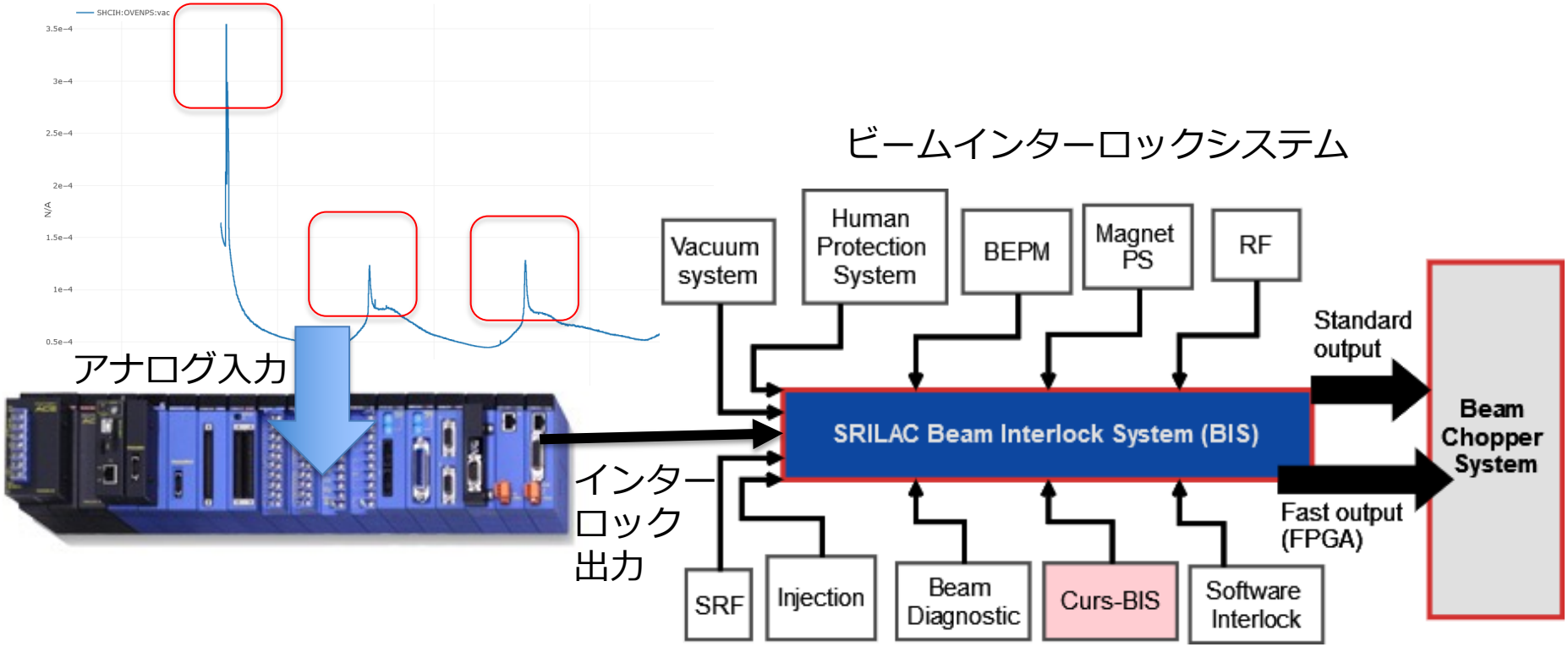

外れ値検知（検出単位：データ点）
異常部位検出（検出単位：部分時系列）
変化点検知（検出単位：変化が起きた時点）

ファーストステップとしてF3RP70-2Lで異常値検知をやってみる

# 加速器パラメータの異常値検知

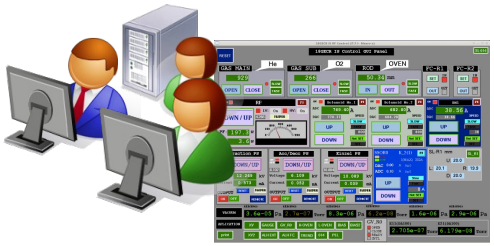
検討している実装例

- F3RP70-2LにEPICSが走らせて、PLCのIOモジュールからデータを取得、Pythonで書かれた機械学習アプリケーションと連携する。
- 異常値検知をトリガーにマシンプロテクションへインターロック出力させる。

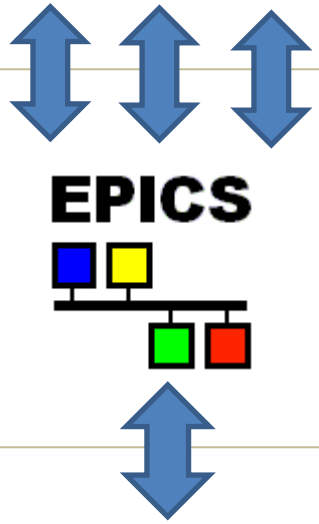


# 加速器パラメータの異常値検知

## 加速器制御システムの三層モデル



### クライアントシステム(ユーザとのやりとり)
- オペレータインターフェース (OPI)
- データアーカイブシステム

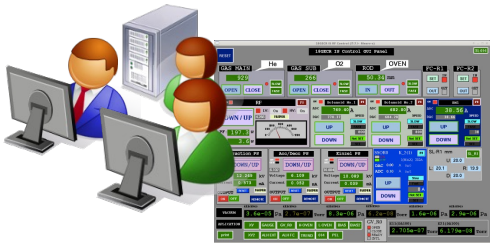



### ミドルウェア(中間層)
- デバイスとクライアント間のプロトコルを統一させる
- EPICS (Experimental Physics and Industrial Control System)

### デバイス (ハードウェアとの相互接続)
- 主にデジタルやアナログで機器に命令を送ったり出力を受け取る.

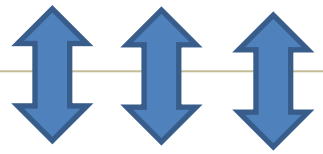
# 加速器パラメータの異常値検知

加速器制御システムの三層モデル



クライアントシステム(ユーザとのやりとり)
- オペ
- デー

ここで機械学習アプリケーションを実装するのが一般的



ミドルウェア(中間層)
- デバイスとクライアント間のプロトコルを統一させる
- EPICS (Experimental Physics and Industrial Control System)

EPICS

デバイス(ハードウェアとの相互接続)
- 主に

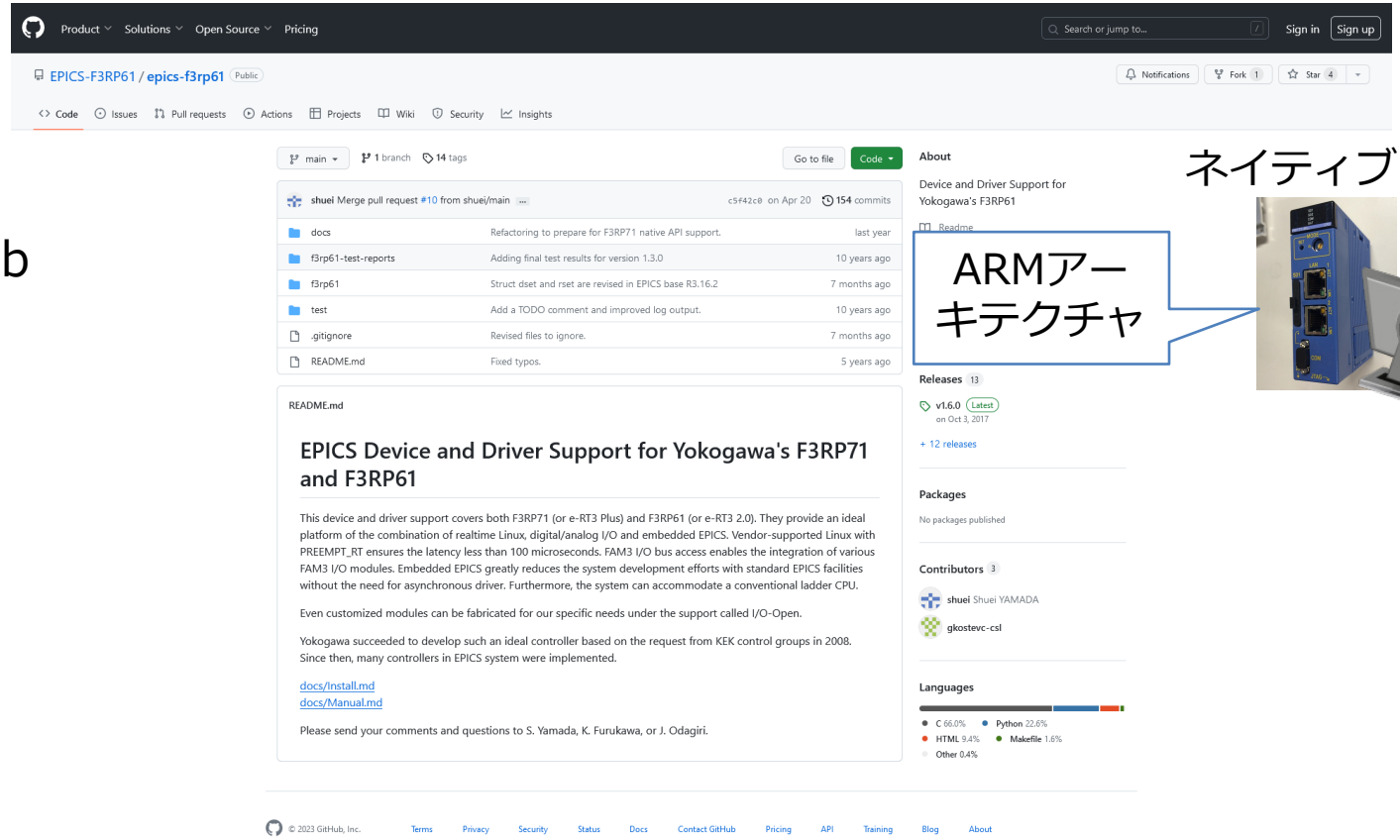
インターロックなので、信頼性の観点からローレベルな所で実装すべき

る

# EPICSビルド

- F3RP70-2L上でARMアーキテクチャ向けにネイティブコンパイルやってみた。
- Pythonでモジュールにアクセスするapiは用意されているが、すでに実績あるデバイスサポートの利用が吉
- デバイスサポートはF3RP61用をMakefileを修正してビルド。そのまま使えそう。

Github




ARMアーキテクチャ

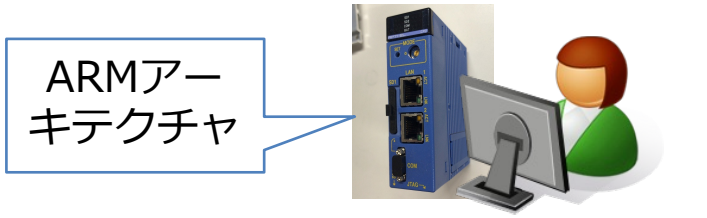
ネイティブコンパイル

# EPICSビルド

- F3RP70-2L上でARMアーキテクチャ向けにネイティブコンパイルやってみた。
- Pythonでモジュールにアクセスするapiは用意されているが、すでに実績あるデバイスサポートの利用が吉
- デバイスサポートはF3RP61用をMakefileを修正してビルド。そのまま使えそう。

```
root@ubuntu:~# cat /root/epics/modules/instrument/epics-f3rp61-
2.0.0pre10/f3rp61/src/Makefile
TOP=../

include $(TOP)/configure/CONFIG
#----------------------------------------
#  ADD MACRO DEFINITIONS AFTER THIS LINE
#=============================

USR_CFLAGS += -std=c99
USR_CFLAGS += -Wall
#USR_CFLAGS += -Werror
USR_CPPFLAGS += -DUSE_TYPED_RSET -DUSE_TYP
USR_INCLUDES += -I/usr/local/include/ert3

#===========================================
# build a support library

ifneq ($(filter $(T_A), linux-f3rp61 linux-f3rp71 linux-arm),)
LIBRARY_IOC += f3rp61

# install devXxxSoft.dbd into <top>/dbd
DBD += f3rp61.dbd

# install header file(s) into <top>/include
INC += drvF3RP61.h
```

ネイティブコンパイル



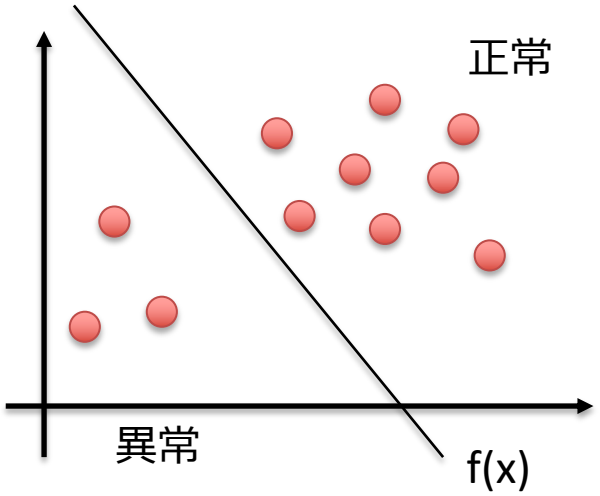

Includeを追加

linux-armを追加

# 異常値検知テスト

- One-Class SVM(One-Class Support Vector Machine)を用いたシステム
- 訓練データに属するクラスのみが与えられている状況で異常検知や外れ値検出を行う。
- 正常な挙動が分かっているデータセットを与えて、それに基づいて異常な挙動を検知。
- 具体的にはPython機械学習ライブラリ Scikit-learnを利用

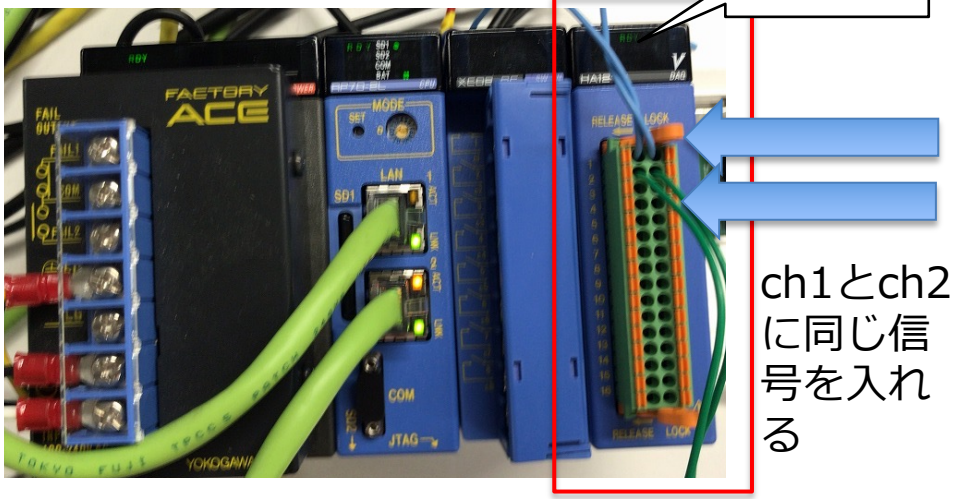
```
def training(rawdata, ch_num):
    #特徴抽出
    feature = feature_extraction(rawdata, ch_num,
DATA_SIZE)

    #学習
    clf = svm.OneClassSVM(nu=0.1, kernel="rbf")
    clf.fit(feature)

    #モデル保存
    pickle.dump(clf, open("model.pickle", 'wb'))
```

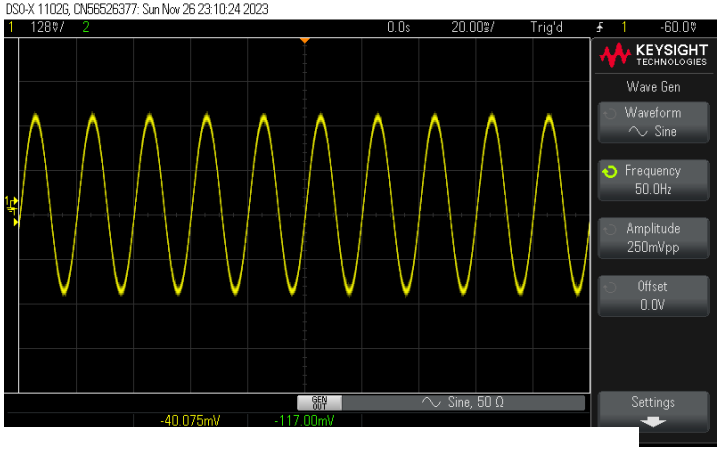


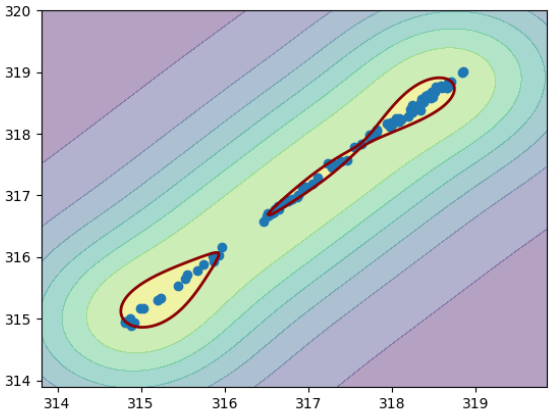

ハイパーパラメータ :0.1
カーネル関数: 放射基底関数（非線形な関係を判定するのに適している）

# 異常値検知テスト

- 高速アナログモジュールF3HA12を利用
- Ch1とCh2に50 Hzのサイン波の信号を入力して、学習させる。
- アナログのサンプリング1msec周期

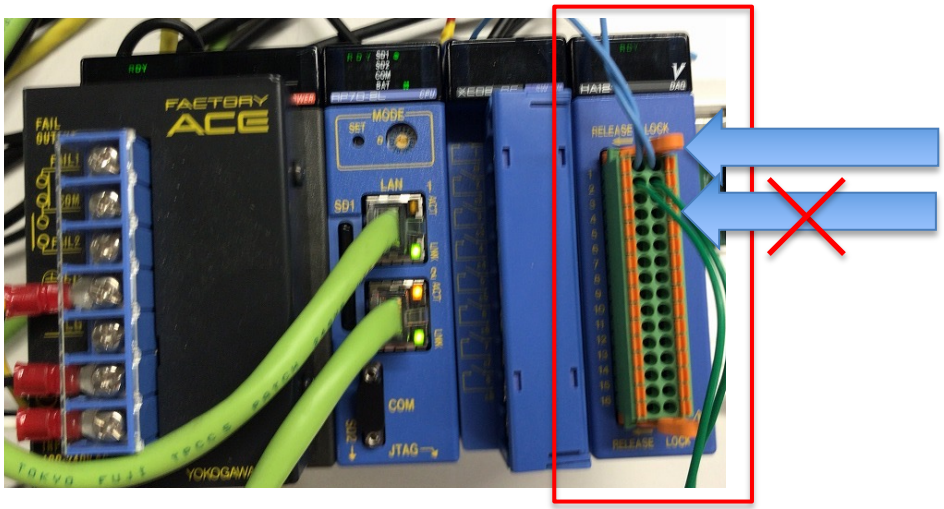


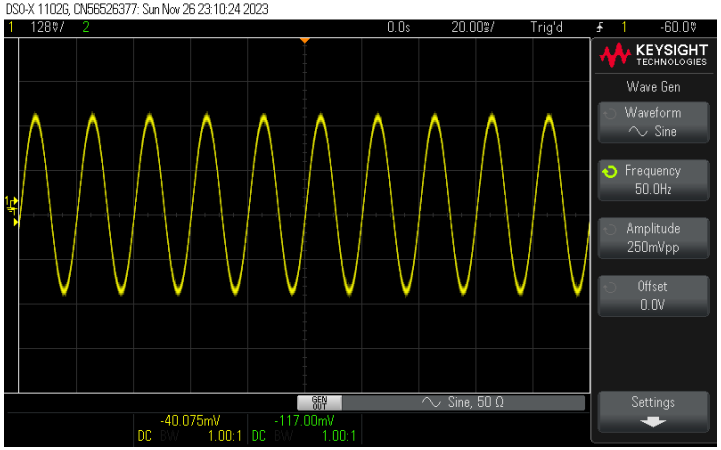



# 異常値検知テスト

- 高速アナログモジュールF3HA12を利用
- Ch1とCh2に50 Hzのサイン波の信号を入力して、学習させる。
- アナログのサンプリング1msec周期

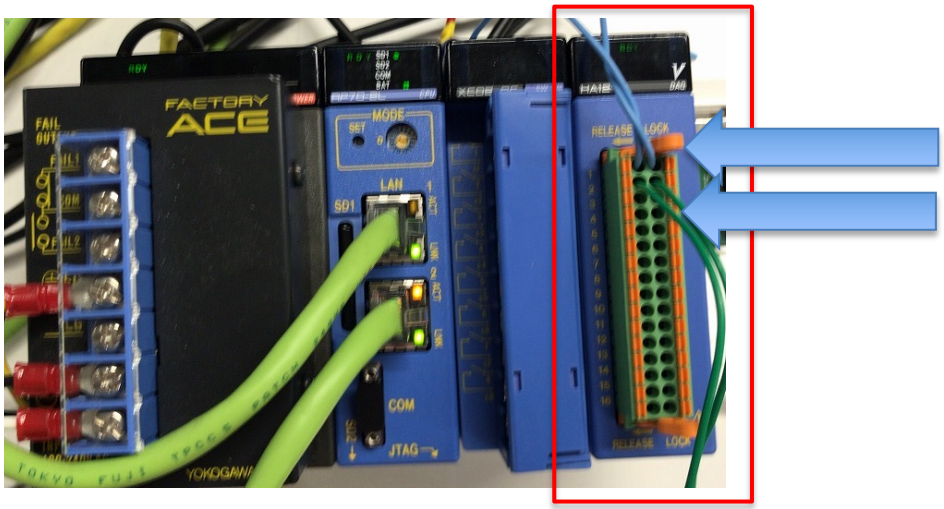


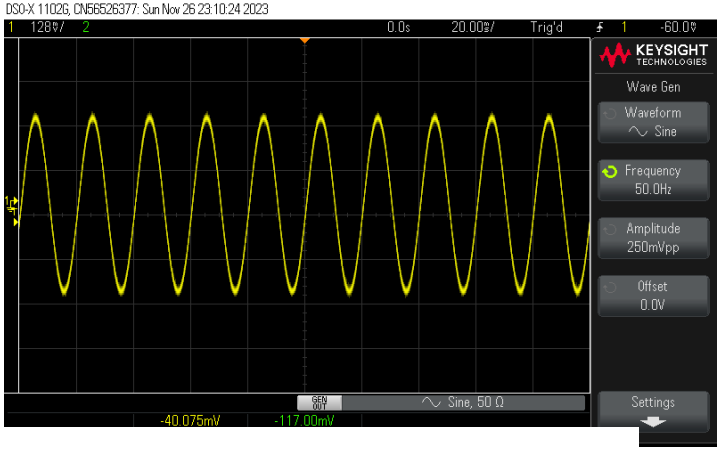



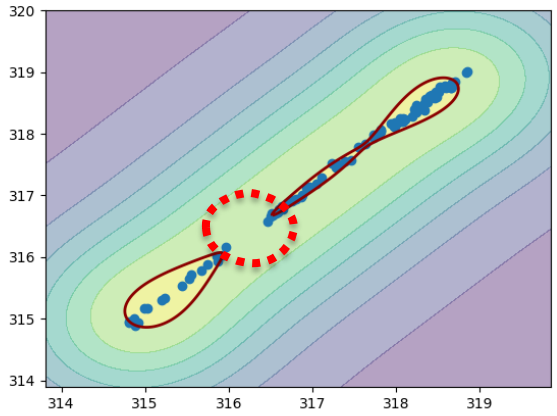

Ch2へ入っている信号をオフさせると、即座に異常判定する挙動をした。
なんとなく動いているみたいだけど、、

# 異常値検知テスト

- 高速アナログモジュールF3HA12を利用
- Ch1とCh2に50 Hzのサイン波の信号を入力して、学習させる。
- アナログのサンプリング1msec周期



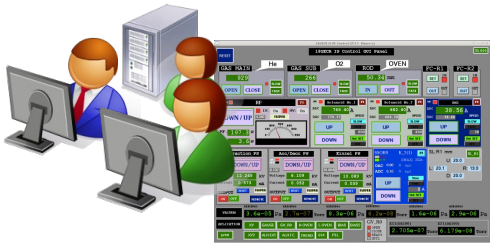




- テスト信号は同じなのに、10%ぐらい異常判定されてしまう、、学習で抜けがある。
- 実際の異常判定させる値（例えばRFの電圧と真空度とか）ではハイパーパラメータチューニング必要
- 異常検出性能向上のためには適切な学習データセットの用意

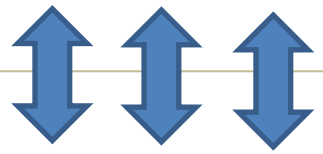
# Pythonインターフェース

加速器制御システムの三層モデル



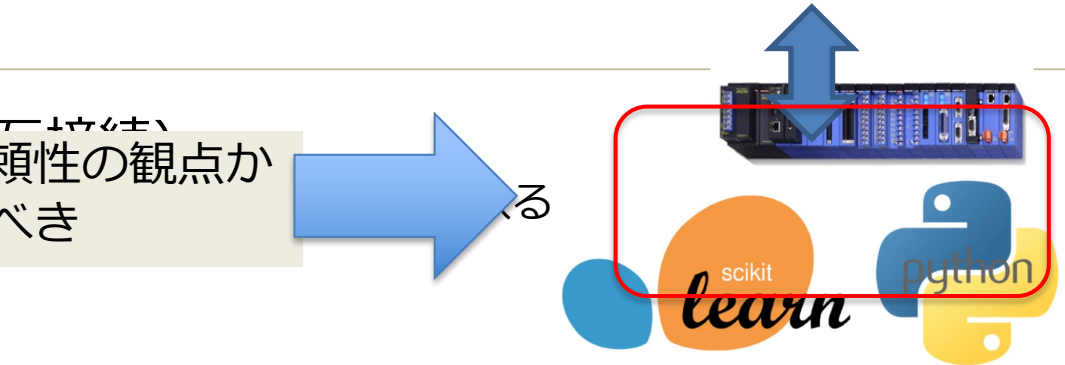
クライアントシステム(ユーザとのやりとり)
- オペ
- デー

ここで機械学習アプリケーションを実装するのが一般的



ミドルウェア(中間層)
- デバイスとクライアント間のプロトコルを統一させる
- EPICS (Experimental Physics and Industrial Control System)

EPICS



デバイス(ハードウェアとの相互接続)
- 主に

インターロックなので、信頼性の観点からローレベルな所で実装すべき

る

# Pythonインターフェース

- 機械学習のプログラムはPythonで作成される事が多くなってきている。
- EPICSとの連携にはPyDeviceを使うと有効なのでは、と考えている。

- PyDeviceではEPICSのデバイスサポートソフトウェアをPythonで記述できる。
- 通常のデバイスサポートはC言語ベースで書く事が一般的だが、PyDeviceを利用すると、Pythonの関数を呼び出して使えるので、コストが下がる。

## PyDeviceの例

配列を返すデバイスサポート例

root@ubuntu:~/epics/python# cat pydevuchi.py

```
def print_array():
    my_array=[1,2,3,4,5,6,7,8,9]
    # 配列の作成
    return my_array
```

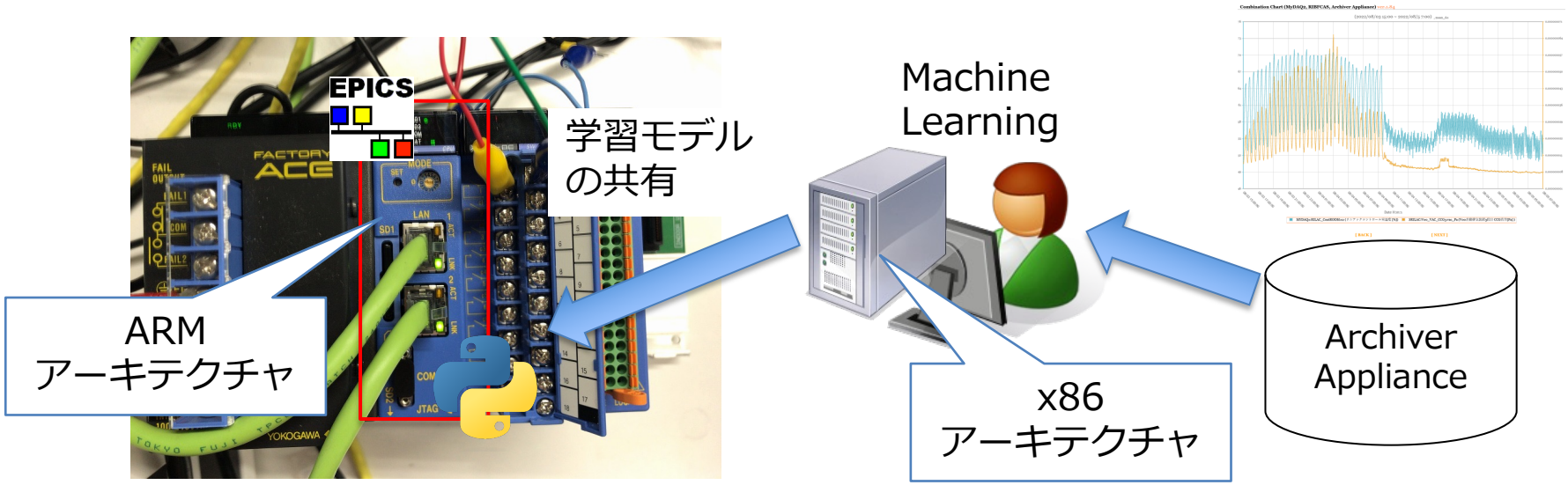
配列を受け取るデータベースの例

```
record(waveform, "PyDev:Array:uchi") {
    field(PINI,"YES")
    field(DTYP, "pydev")
    field(INP,  "@pydevuchi.print_array()")
    field(NELM, "8")
    field(FTVL, "LONG")
}
```

出力1,2,3,4,5,6,7,8,9

# 将来的な実装例

- PyDeviceのビルドはF3RP70-2L上でできた → これを用いてPythonで開発した機械学習プログラムを呼び出してEPICSと連携する予定
- 実際の加速器のパラメータで異常値検知できるかやってみる。
  - 適切なデータセットの準備
- F3RP70-2LのCPUは通常のPCのアーキテクチャに比べると非力なので、実際の運用法方は、別マシンで学習させた後でモデルを共有させる手法を使う。



# まとめ

- 横河電機FA-M3のUbuntuが走るCPUモジュールF3RP70-2Lを触ってみた。
- F3RP61/F3RP71用のEPICSデバイスサポートはMakefileを修正するだけでネイティブコンパイルでき、F3RP70-2Lでも動きそう。
- UbuntuなのでPLC CPUだとしても簡単にオープンソースが使え、特にPythonでの機械学習の開発はすぐに取り掛かることができる。
- 実際にPython機械学習ライブラリ Scikit-learnを利用した異常値検知のテストをして、モジュールからのアナログ入力に対し極端な値の変化では検知できた。
- 将来的にPythonで開発されたアプリケーションとEPICSとの連携をするにはPythonの関数がデバイスサポートとして呼び出せるPyDeviceが有効なのではないかと予想している。
- 他の使い道がないかと模索中。