ストリーミングデータ収集に対応した分散データ収集ソフトウェアの現状

五十嵐 洋一
KEK
2023/11/20

Motivation

- Trigger less DAQ をしたい。
- Software Trigger を行いたい。
- 高速にデータを収集したい。
  - データが流れるバンド幅をあげたい。
    - 複数の計算機にイベントビルドを分散すれば…
    - 複数のストレージに記録すれば…

- 少人数で開発、運用可能であることは外せない。

→ 連続読み出し可能な Front-end Electronics
→ 全体を同期する Clock system
→ 連続的に処理できる分散型の イベントビルド・Software trigger をハンドルできるソフトウェア

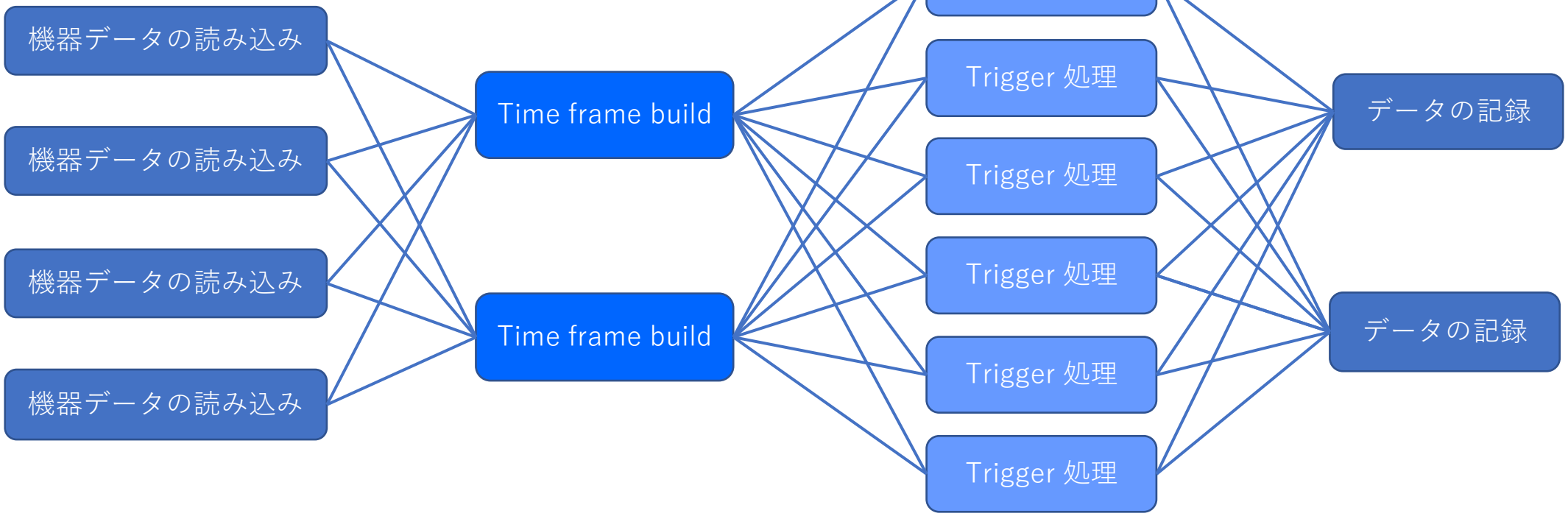
Streaming Oriented DAQ Software

- 全体管理
- 全体制御

ボトルネックのないメッシュ接続
- 機器をまたいだ一対多、多対一 の通信
- バッファリング

負荷分散

複数のデータ記録プロセス



これらを実現するためには?

- TCP/IP, socket より高度な通信手法
- データベース

- ZeroMQ
- redis

分散 DAQ を構成するには

- 通信
  - 一対多、多対一をサポートした メッセージキュー ZeroMQ
- 有限状態遷移機械
  - INITIALZE, IDLE, RUN, POSE, STOP
- 制御・被制御
- 状態把握、Watchdog

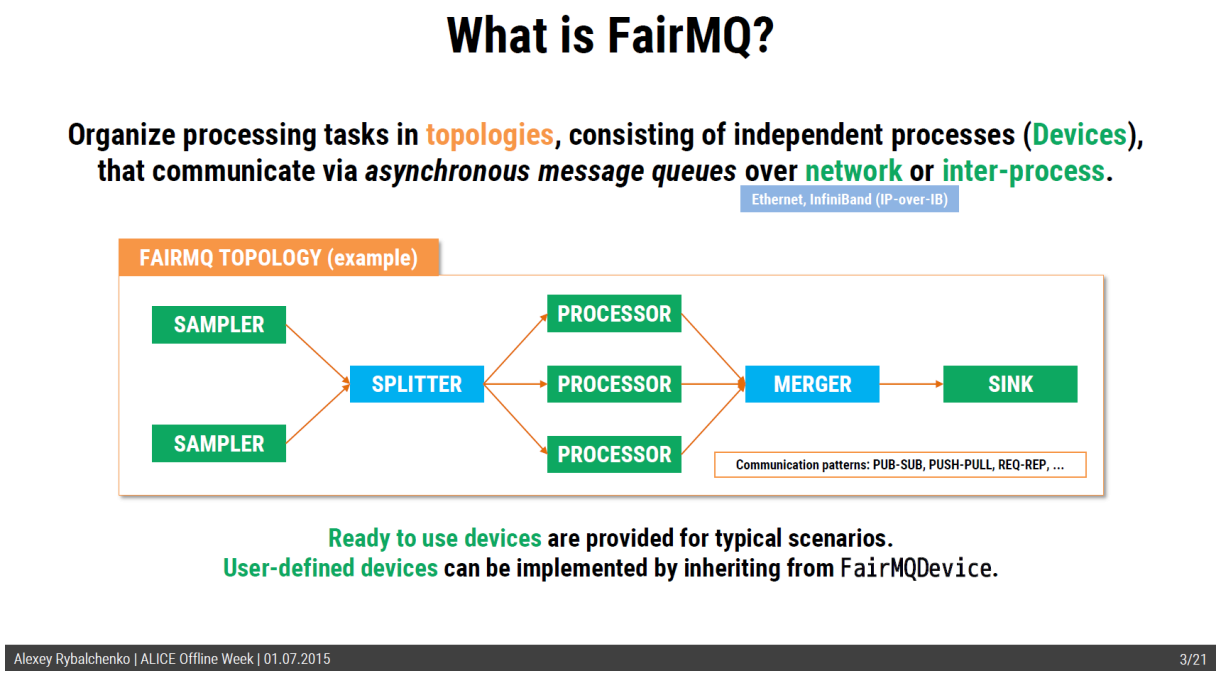
どうする?

→先行研究に何かあった。
→ FairMQ

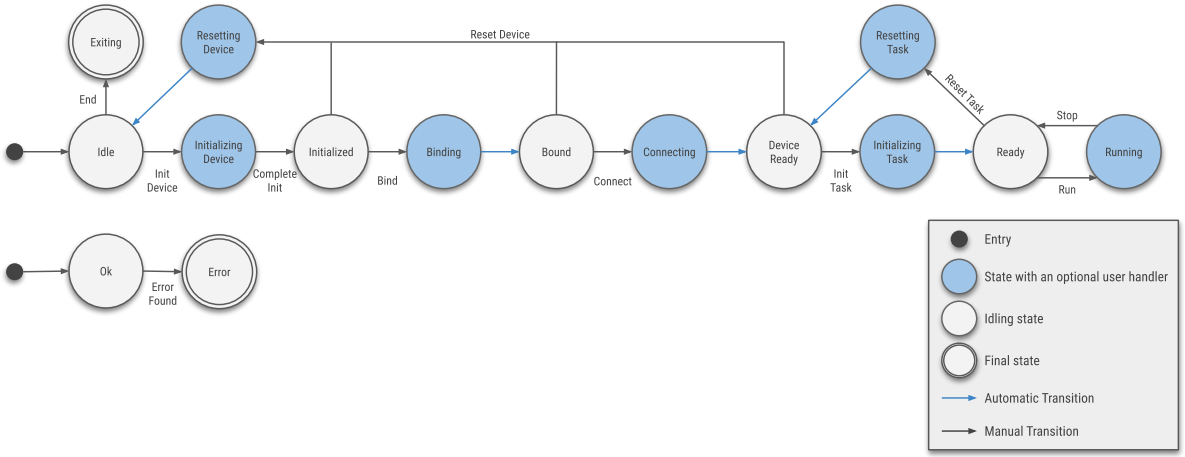
- ハードウェアからのデータの取り込み
  - Socket programing, read(2), device driver
- システムの全体管理
  - データベース redis

FairMQ

- GSI/FAIR のために開発されている (DAQや解析、シミュレーションを含む) フレームワーク
- ZeroMQ + State machine + 制御 plug-in + たくさんの周辺の統合



from Alexey Rybalchenko(GSI)'s slide



FiarMQ state machine

Plug-in による拡張が可能

A streaming oriented DAQ software

- どのように多数のプロセスを管理すればよいのか?
  →Key-Value 型データベース：redis

→ FairMQ と redis を組み合わせたらどうだろう?

FairMQ(core part) + redis
→ NestDAQ (Network based streaming DAQ)

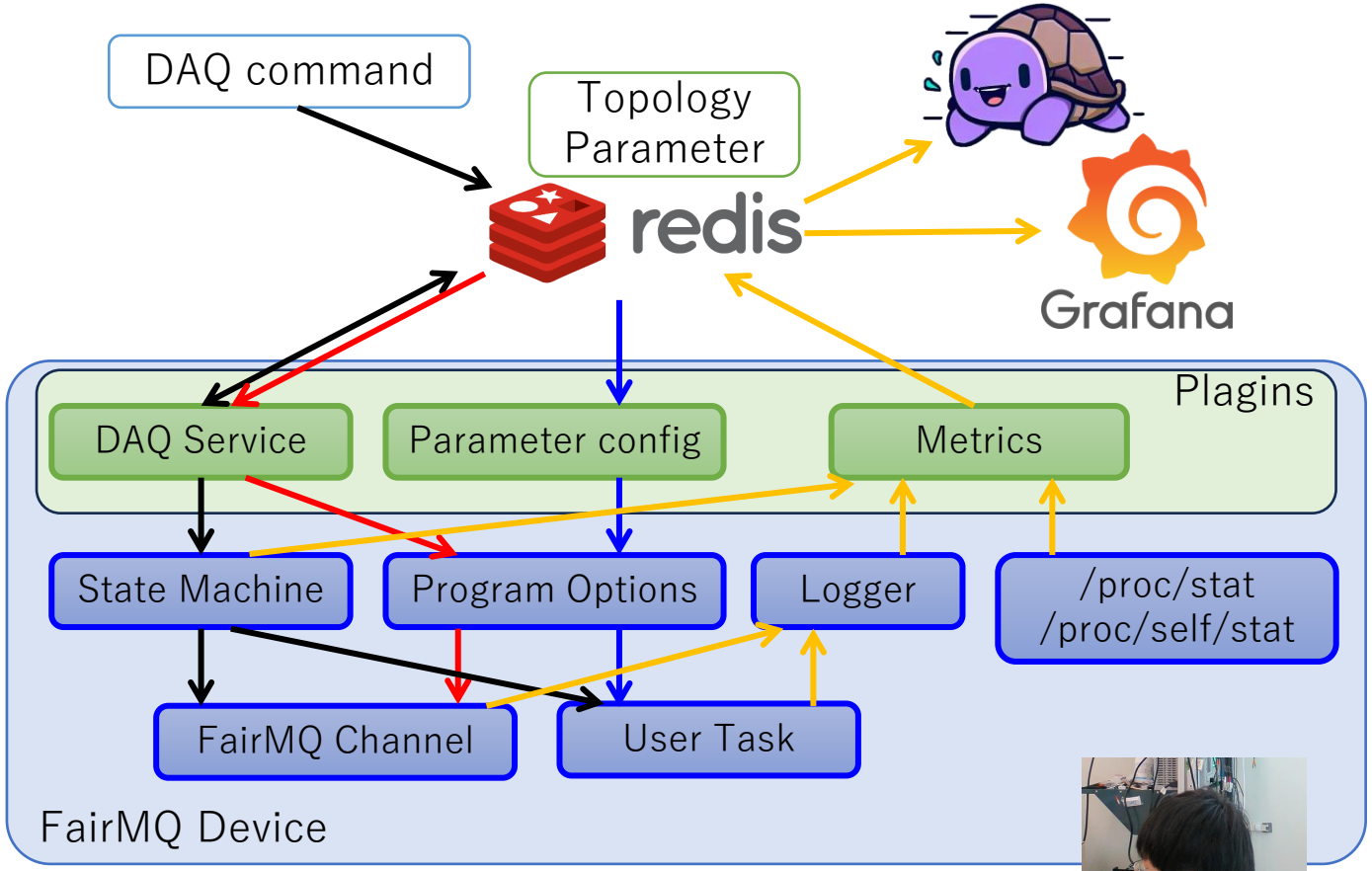
- FairMQ の状態遷移機械、被制御部分を利用。
- 全体管理、制御には redis を使用
  - NoSQL データベース/キーバリュー型
  - メモリ指向/高速
  - キー・スペース通知 → 制御に使える

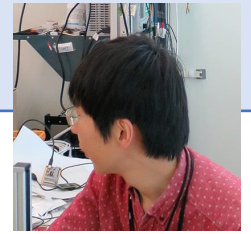
NestDAQ Process 構造

NoSQL/Key-Value 型データベース redis ですべてのプロセスの状態管理と制御をおこなう。

FairMQ Plugin

- DAQ Service Plugin
  - Run control
    - State machine の制御
    - Run number の設定
  - Service discovery
    - 接続設定の半自動化
- Metrics Plugin
  - Process の状態把握
- Parameter config Plugin
  - ハードウェア初期化等のパラメータをデータベースから読み込む





高橋智則氏(RCNP) 中心部分を開発

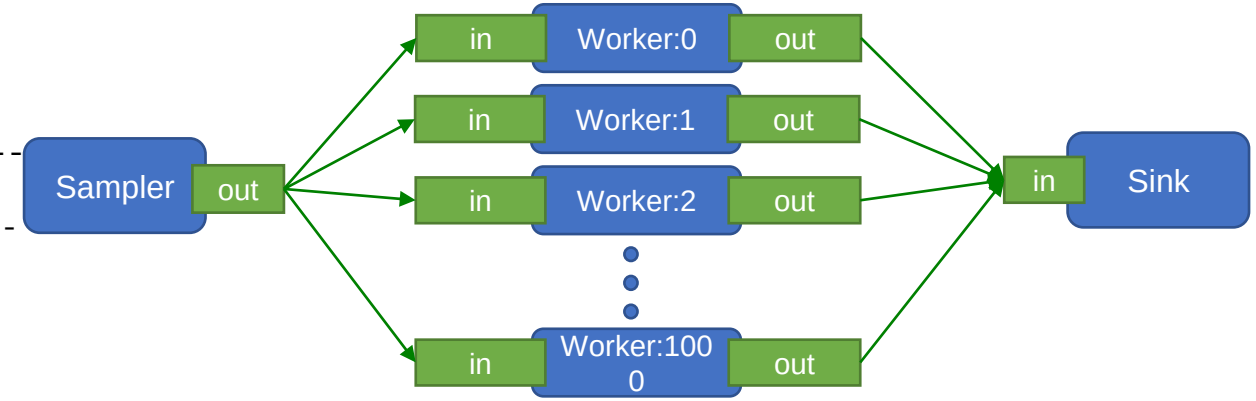
たくさんのものを構成する

DAQ Service：Service discovery

- 1000以上のテーブルは手書きは現実的でない。→ 構成の半自動化
- 機能を論理グループとして分けてヒントを与えて接続情報を自動的に構成する。
  - Interface, Service, Endpoint, Link

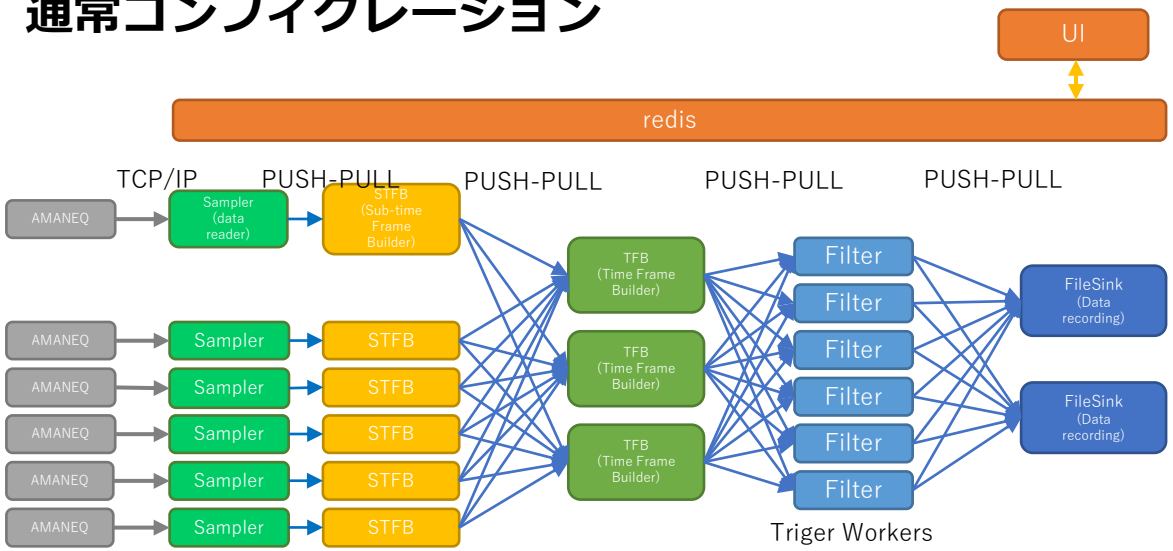
```
#-------------------------------------------------------------------------
#            service        channel         options
#-------------------------------------------------------------------------
endpoint     Sampler         out             type push  method bind
endpoint     Sink            in              type pull  method bind
endpoint     Worker          in              type pull  method connect
endpoint     Worker          out             type push  method connect

#-------------------------------------------------------------------------
#       service1         channel1        service2     channel2
#-------------------------------------------------------------------------
link    Sampler          out             Worker       in
link    Worker           out             Sink         in
```

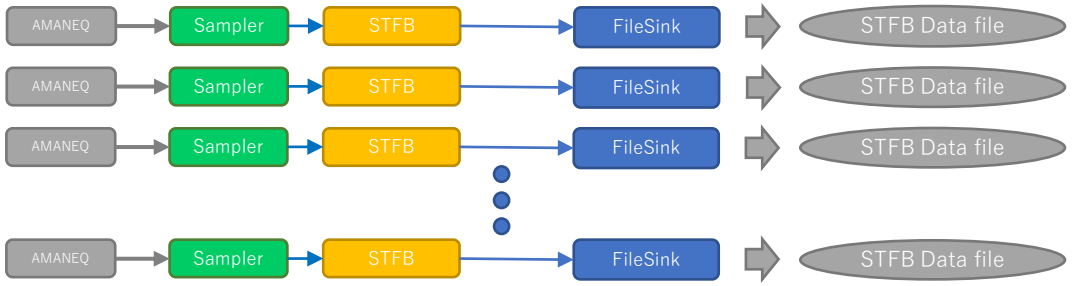


Streaming DAQ process 構成と接続

通常コンフィグレーション



3段N列 並列 コンフィグレーション



- Sampler
  - 連続読み出し FEE のデータを読み出して後段に送る。
  - データは時間を切り分ける HBF (524 us) を持っている。
- Sub-Time Frame Builder
  - Sampler からのデータを HBF 毎に切り出し、いくつかまとめて Sub-Time Frame を作り後段に送る。
- Time Frame Builder
  - 複数の Sub-Time Frame データを同じ時間ごとにまとめて Time Frame を作り後段に送る。
- Filter/Online Trigger
  - Time Frame のなかから Event を見つける。
- Event builder (for Streaming Read Out)
  - Event 付近の時間に含まれるデータを抜き出し後段に送る
- FileSink
  - 受け取ったデータをファイルに書く。

LUT による汎用論理 Online Trigger プロセス

- 汎用 logic filter
  - 初めにLUT デーブルを演算して作る。
  1. Low-resolution TDC / High-resolution TDC → 4ns TDC
  2. Hart-beat Frame (524us) / 4ns の配列にマークを付けていく。
  3. 配列をスキャンして LUT が true になるところを拾い出す。
  4. 変化している部分を vector に詰める。

```
fTrig->SetTimeRegion(1024 * 128); ← 1 HBF
fTrig->ClearEntry();

fTrig->SetMarkLen(10); ← 4ns * 10 (coincidence time width)
fTrig->Entry(0xc0a802a9,  0, 0); //DR
fTrig->Entry(0xc0a802a9,  1, 0); //DL
fTrig->Entry(0xc0a802a9,  2, 0); //DR
fTrig->Entry(0xc0a802a9,  3, 0); //DL
fTrig->Entry(0xc0a802a9,  4, 0); //DR
fTrig->Entry(0xc0a802a9,  5, 0); //DL

fTrig->Entry(0xc0a802aa, 32, 0); //UR
fTrig->Entry(0xc0a802aa, 33, 0); //UL
fTrig->Entry(0xc0a802aa, 34, 0); //UR
fTrig->Entry(0xc0a802aa, 35, 0); //UL

fTrig->MakeTable("0 1 & 2 3 & | 4 5 & | 6 7 & 8 9 & | &");
```

入力する信号数が多くなると、メモリを使いすぎる。

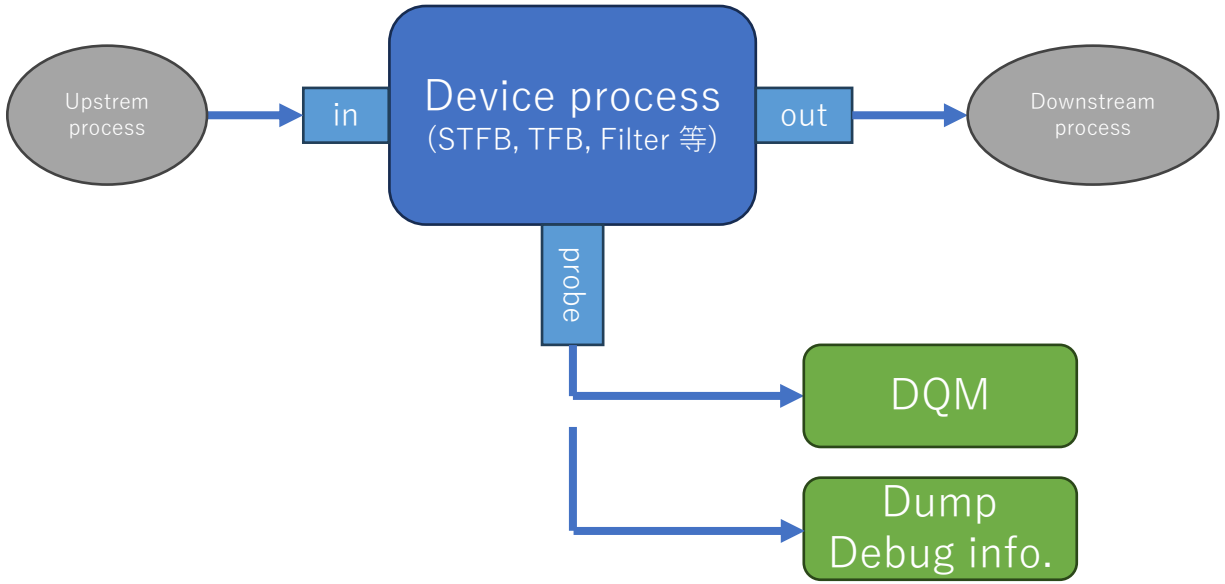
LUT を引いて true になるところを拾い出す。

Module	Ch.	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
AMANEQ1	Ch.02		■	■	■												
AMANEQ1	Ch.04			■	■	■						■	■	■			
AMANEQ1	Ch.06				■	■	■										
AMANEQ1	Ch.08			■	■	■											
	…																
	Ch.31																

ちょっとした工夫

Probe port

- (基本的には)全ての DAQ プロセス (Device program/process) に out port のコピーを出力できる probe out port がついている。
  - デバッグ時/状況を把握したいときはどのプロセスからでもデータを見れる。
  - データを分けて流せる。
  - DQM/Online monitor に使える。

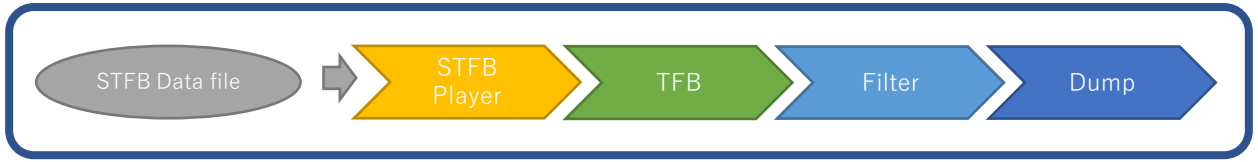


File replayer

- 記録されたファイルを読んで、再度データを流せる

TFB File player, STFB File player による開発

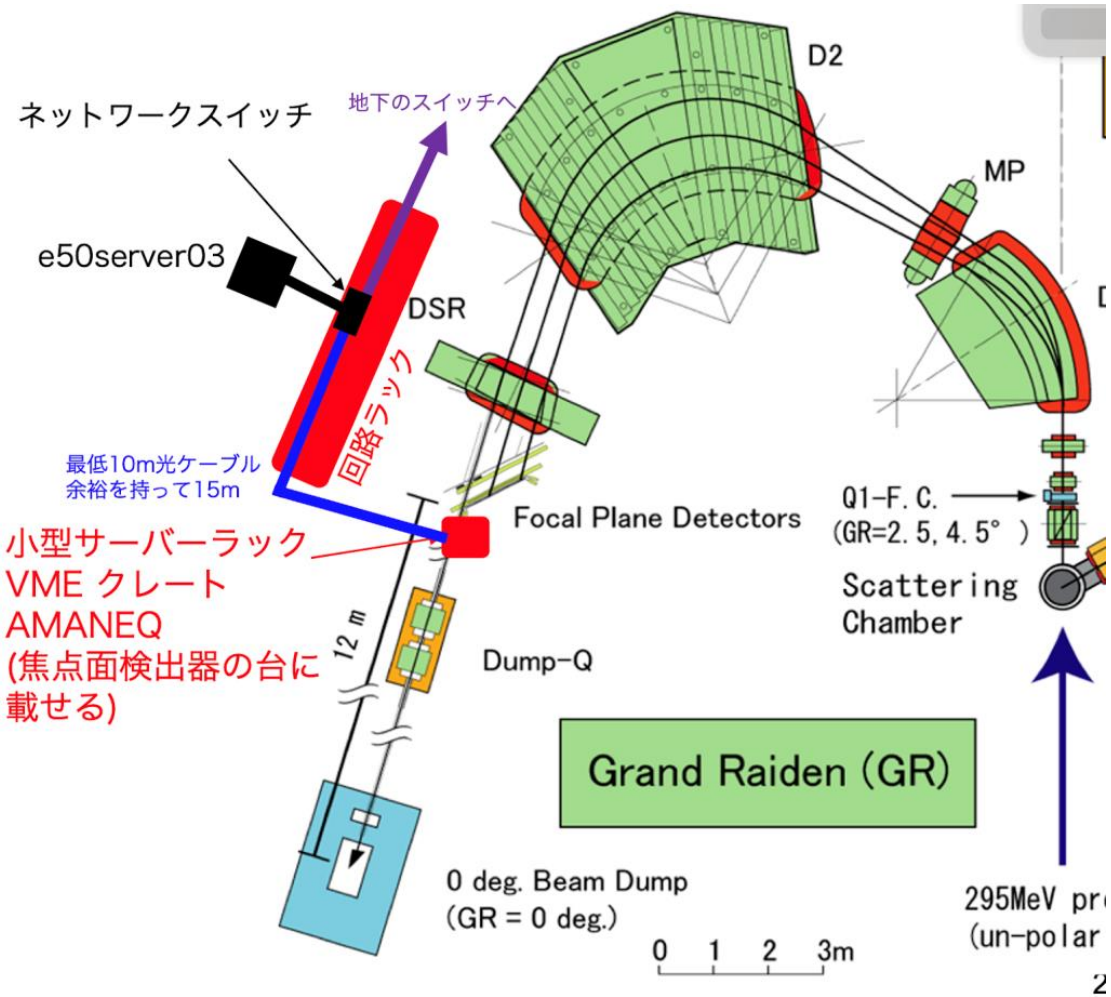
Filter の開発



Time-Frame-Builder、Filter の開発

Streaming DAQ を実際の検出器システムに適用する試み

RCNP Grand Raiden spectrometer

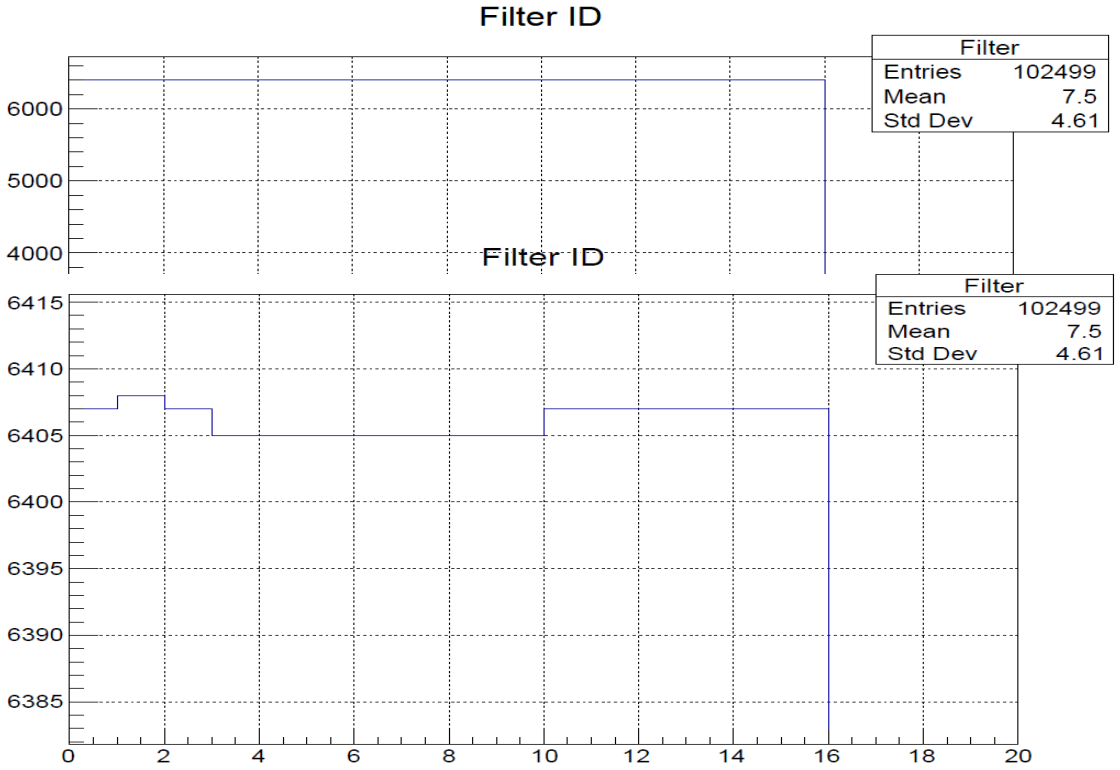


- Plastic scintillation counters
  → FPGA base pipelined HR-TDC with TOT x2
- Drift chambers
  → FPGA base pipelined TDC with TOT x8
- Clock distribution system “MIKUMARI”
- Software trigger process (coincidence + α) “NestDAQ”

→ Streaming DAQ の確立

Filter process のふるまい

負荷分散

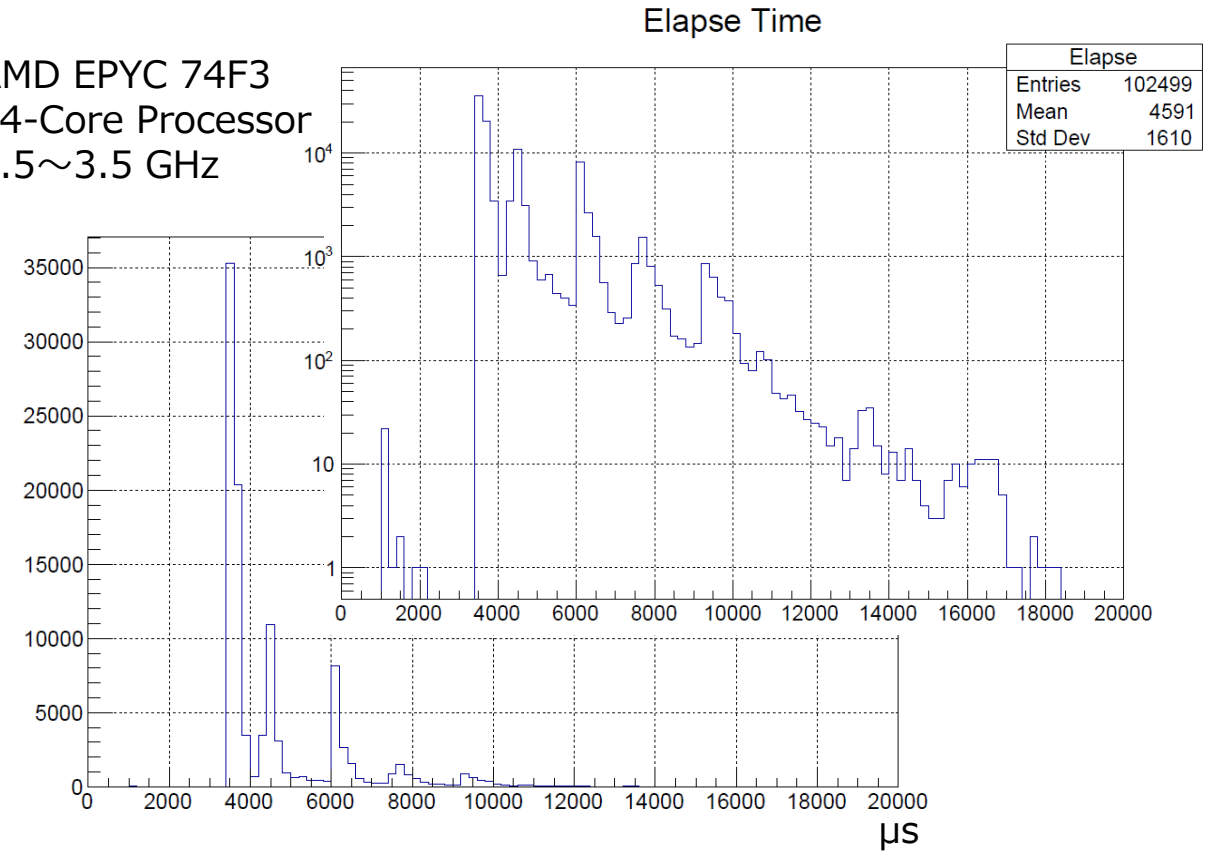


Filter 16 process

Round-Robin + Skip at Queue-Full でほどほどきれいに負荷が分散されている。

5HB 消費時間(データ転送を除く)

AMD EPYC 74F3
24-Core Processor
1.5～3.5 GHz



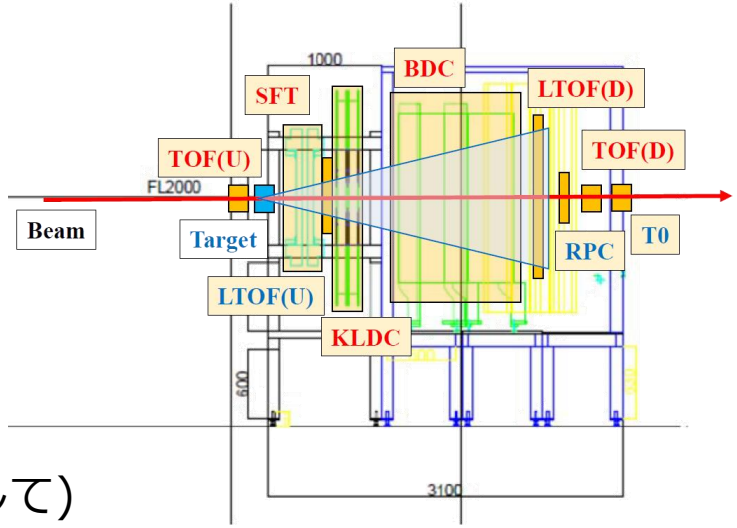
Trigger: Ch1 * Ch2 * Ch3 * Ch4

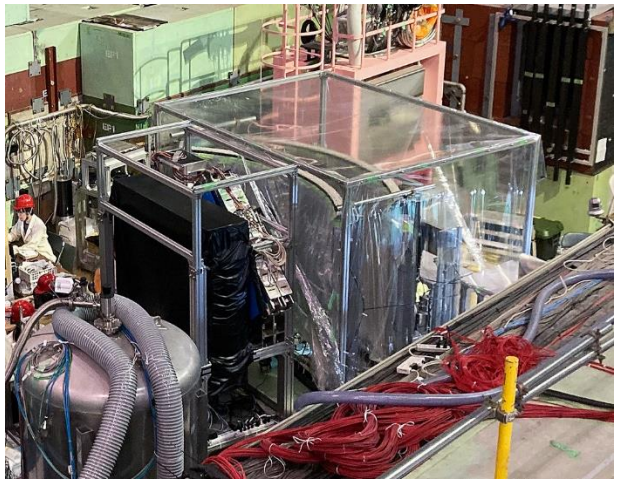
5HB 0.524*5 = 2.62ms で処理するためには?

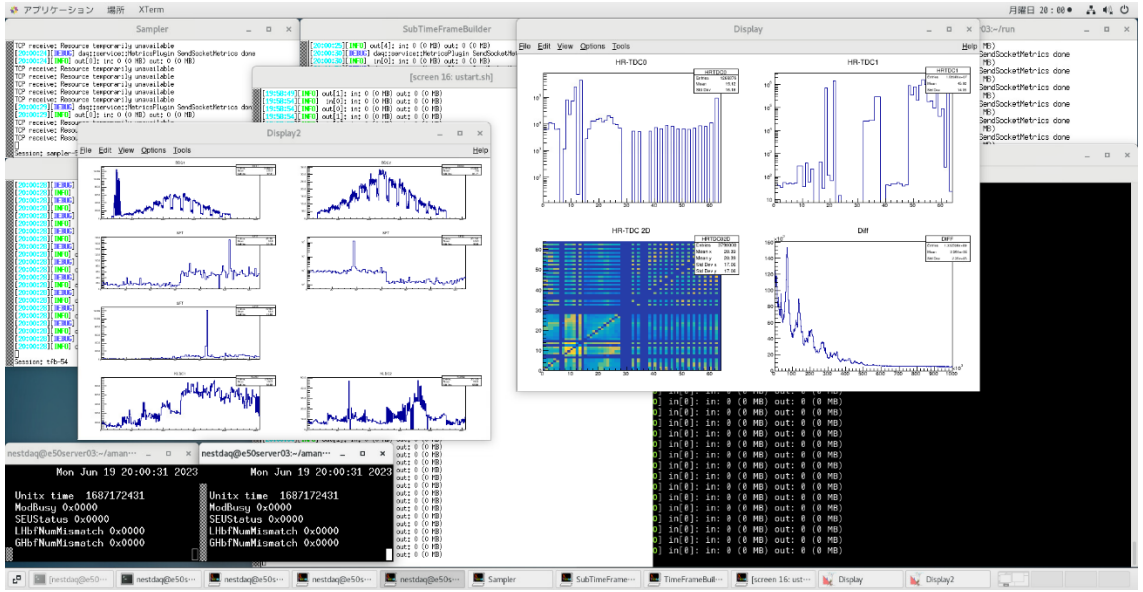
→ 平均 4.5 ms なので、2, 3 プロセスで処理出来る。

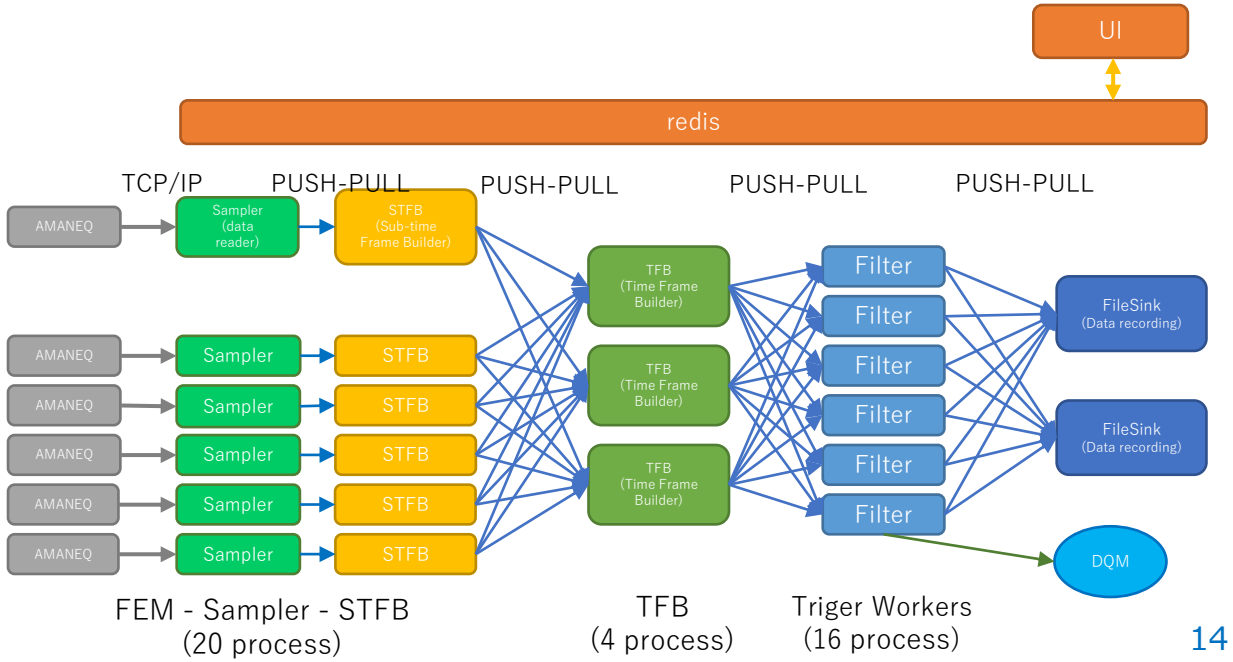
E50 検出器試験 in J-PARC HD K1.8BR

- 読み出し機器、チャンネル数
  - HRTDC x2：128
  - LRTDC x15：1920
  - MIKUMARI x3：64
- LUT による組み合わせ論理による Trigger process
- DQM port からの PUB/SUB による DQM
- 書き込み Data flow：
  - ～180MB/s (Beam 時 FT-ON/OFFを平滑化して)
  - ～240MB/s (FT-ON 時)



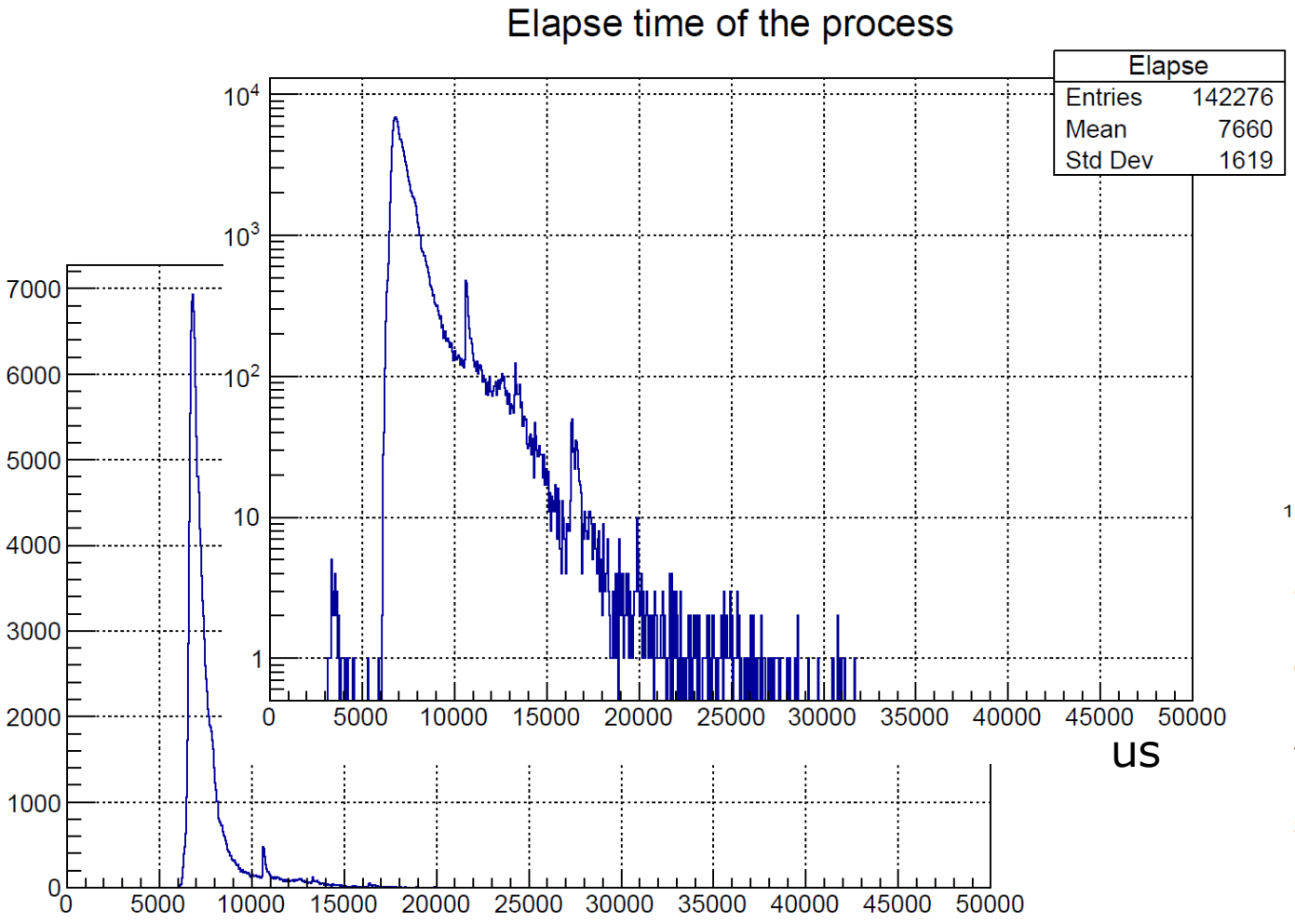


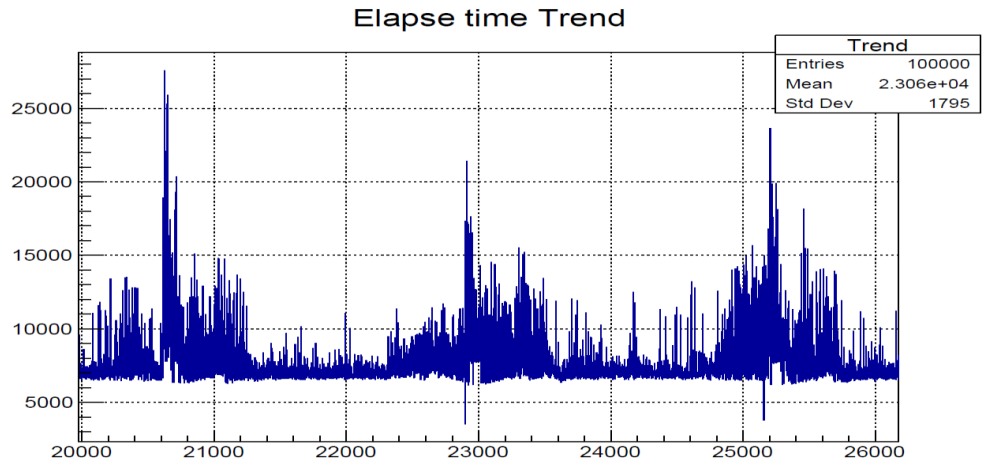


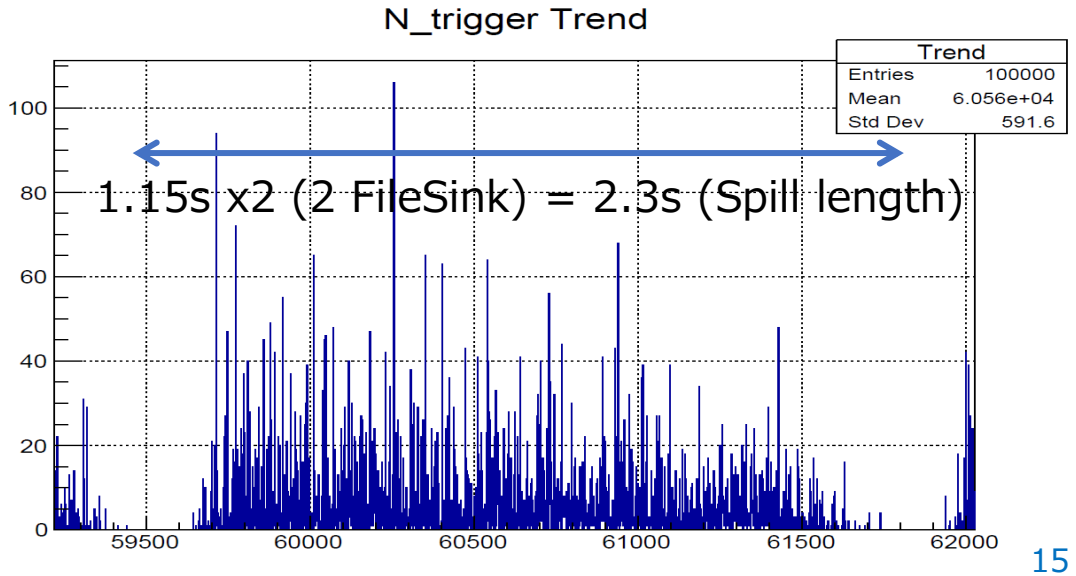


Elapsed process time

- Trigger logic: ((D1L*D1R)+(D2L*D2R)+(D3L*D3R))*((U1L*U1R)+(U2L*U2R))
- 1 HBF を処理するのにかかった時間



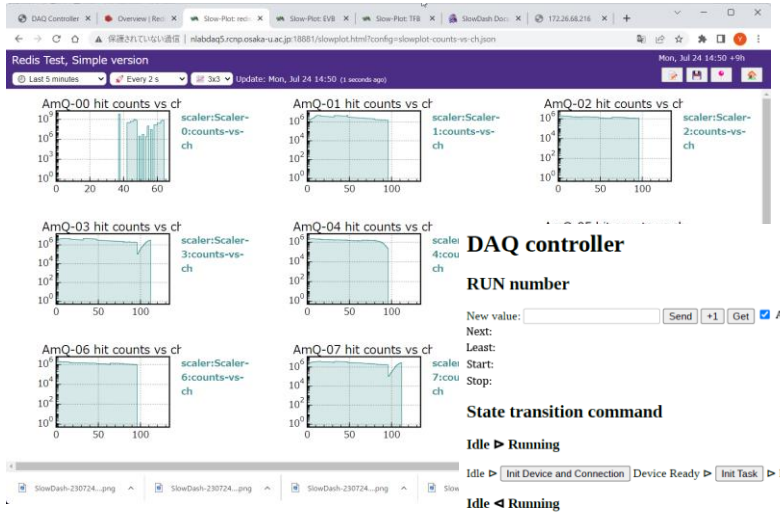


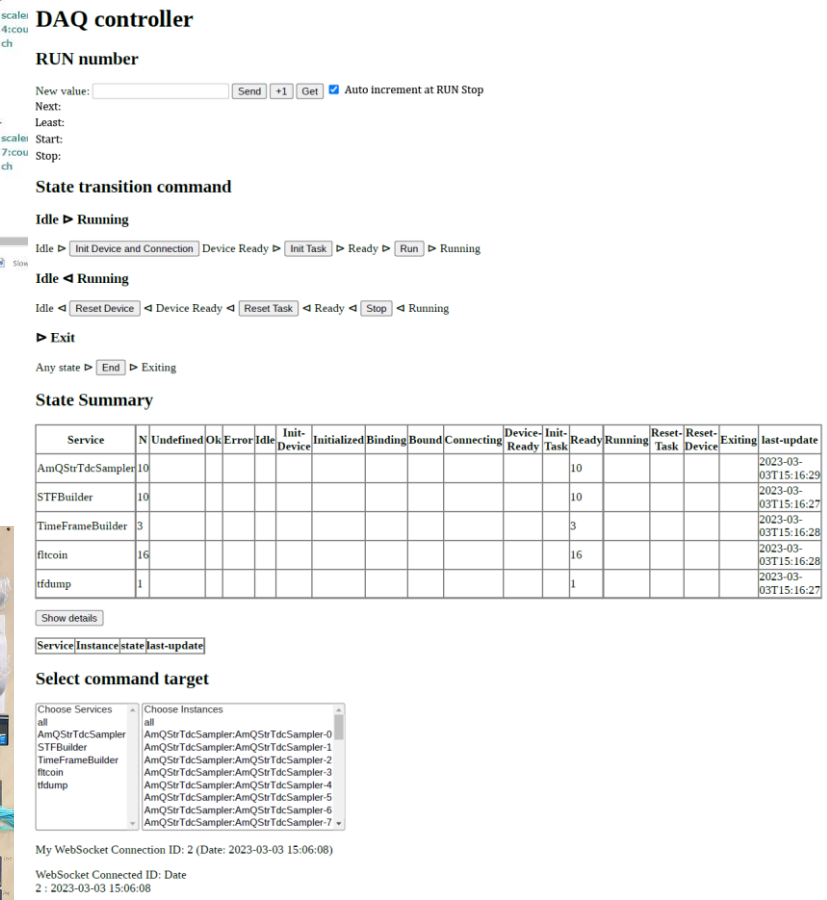


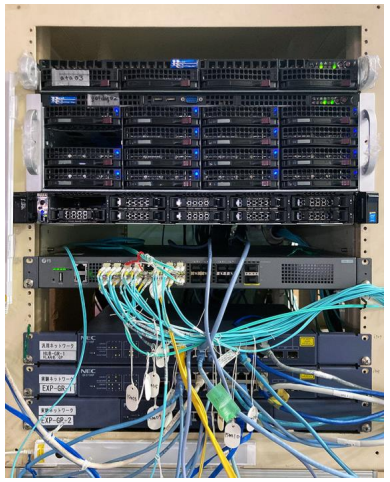
RCNP GR/WS E585

物理実験に投入

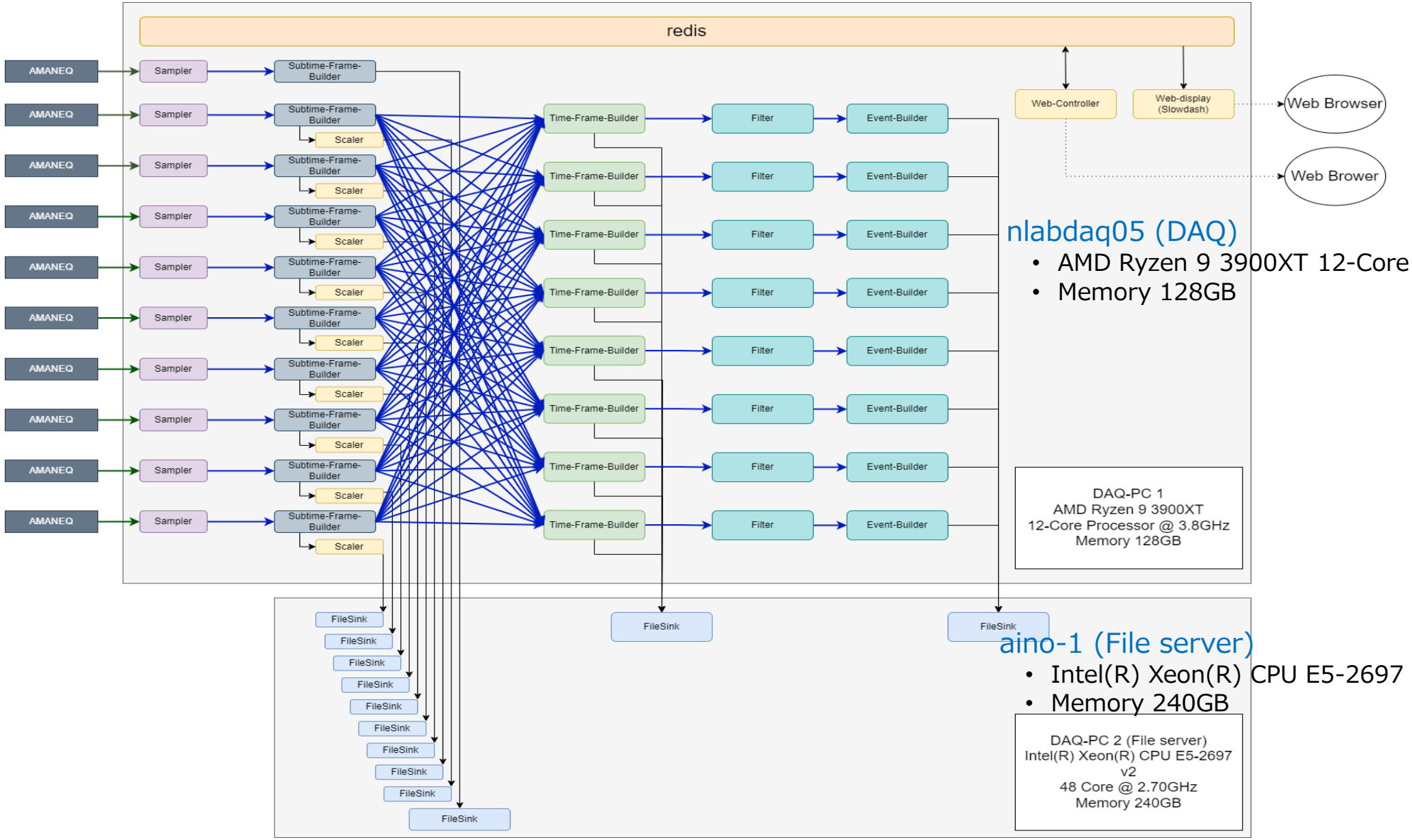
- 任意ロジック Filter の適用
- Event builder (for SRO) の適用
- Pre-scaled unbiased data の記録
- UI が少し便利に
  - Auto inclement Run number
  - Run Start/Stop の前後でScript の実行
- Slowdash 投入
  - Data flow の可視化
  - Software Scaler
  - Issue flag の可視化







RCNP GR/WS E585 software configuration



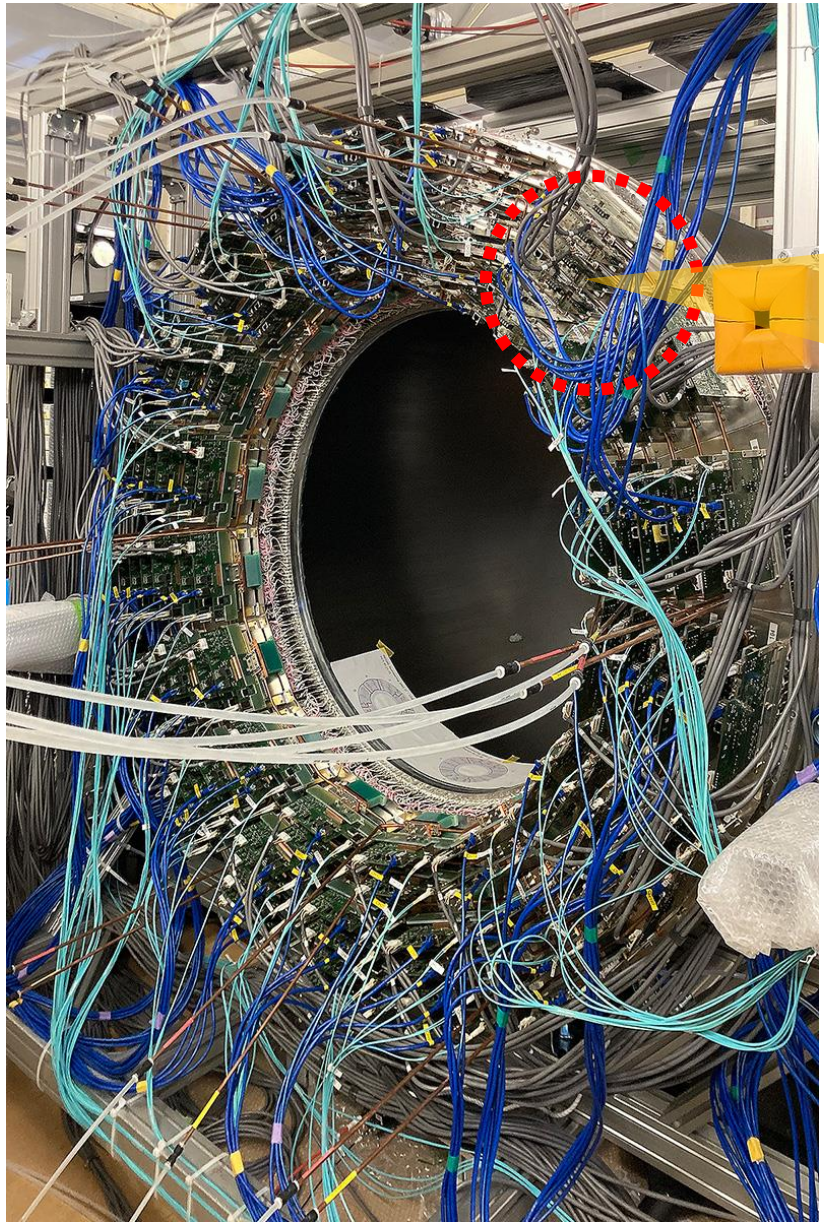
nlabdaq05 (DAQ)

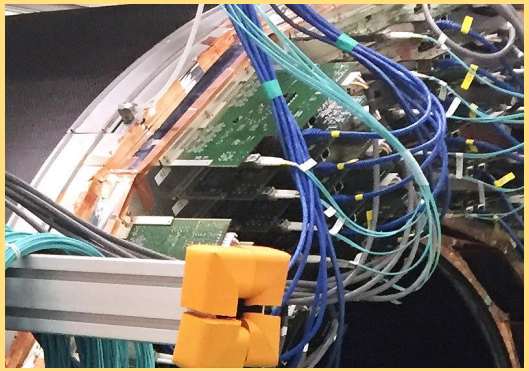
- AMD Ryzen 9 3900XT 12-Core Processor @ 3.8GHz
- Memory 128GB

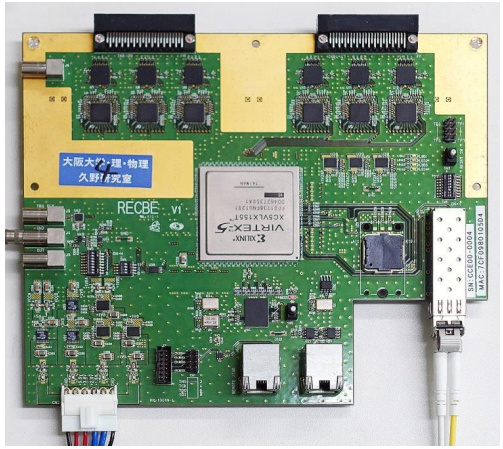
aino-1 (File server)

- Intel(R) Xeon(R) CPU E5-2697 v2 48 Core @ 2.70GHz
- Memory 240GB

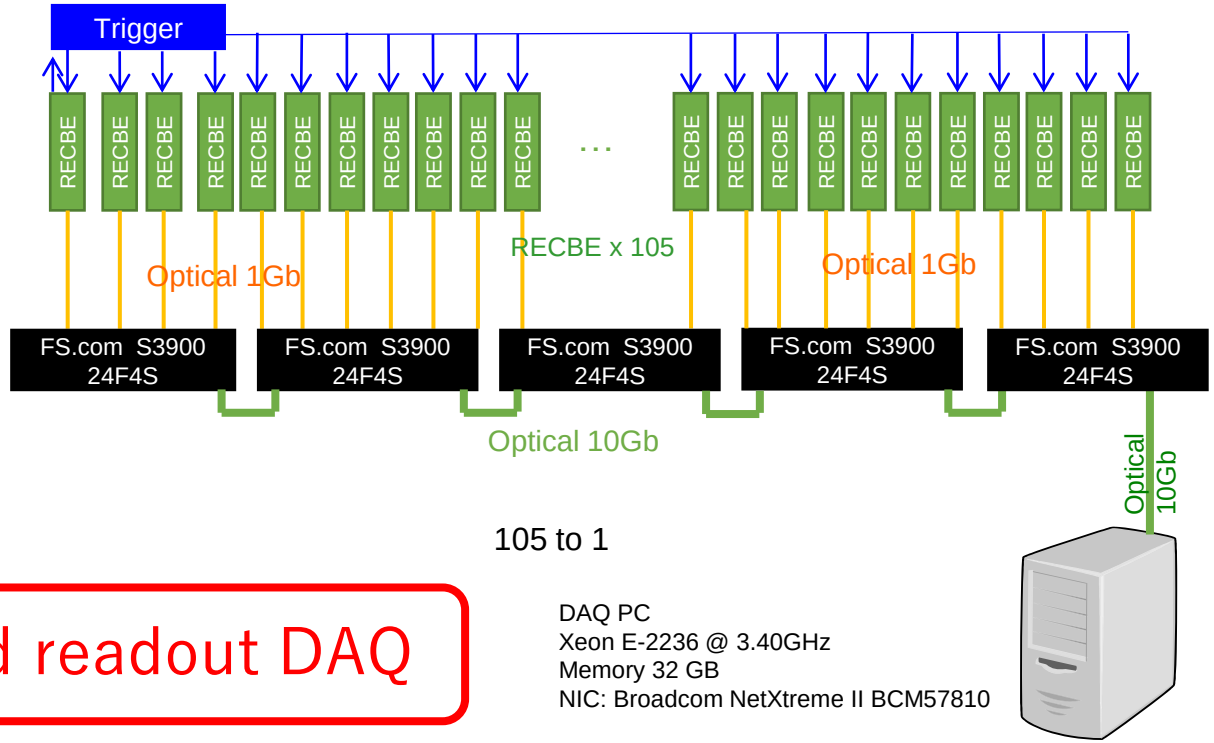
Triggered DAQ への適用(COMET CDC)







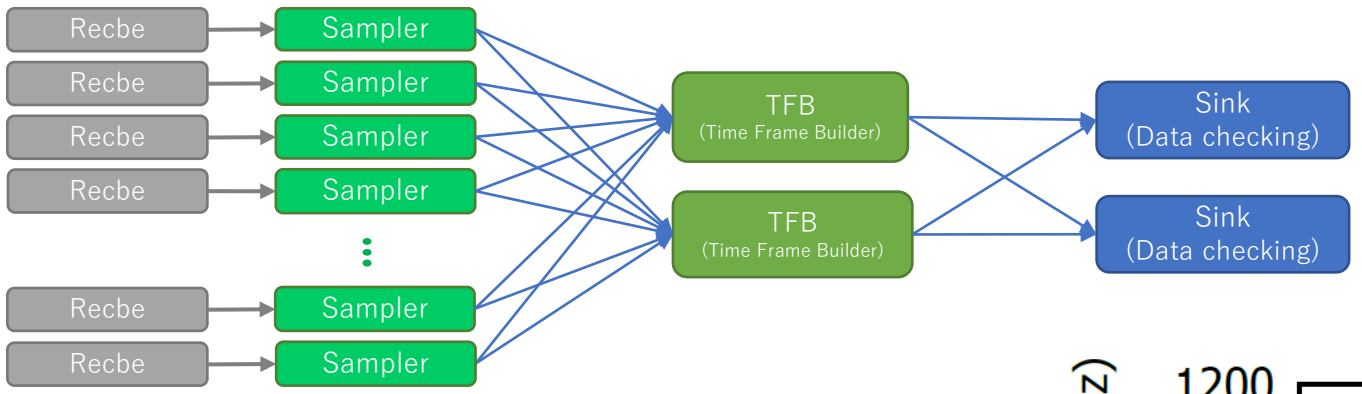
A read-out card for drift chamber “Recbe”



Triggered readout DAQ

DAQ PC
Xeon E-2236 @ 3.40GHz
Memory 32 GB
NIC: Broadcom NetXtreme II BCM57810

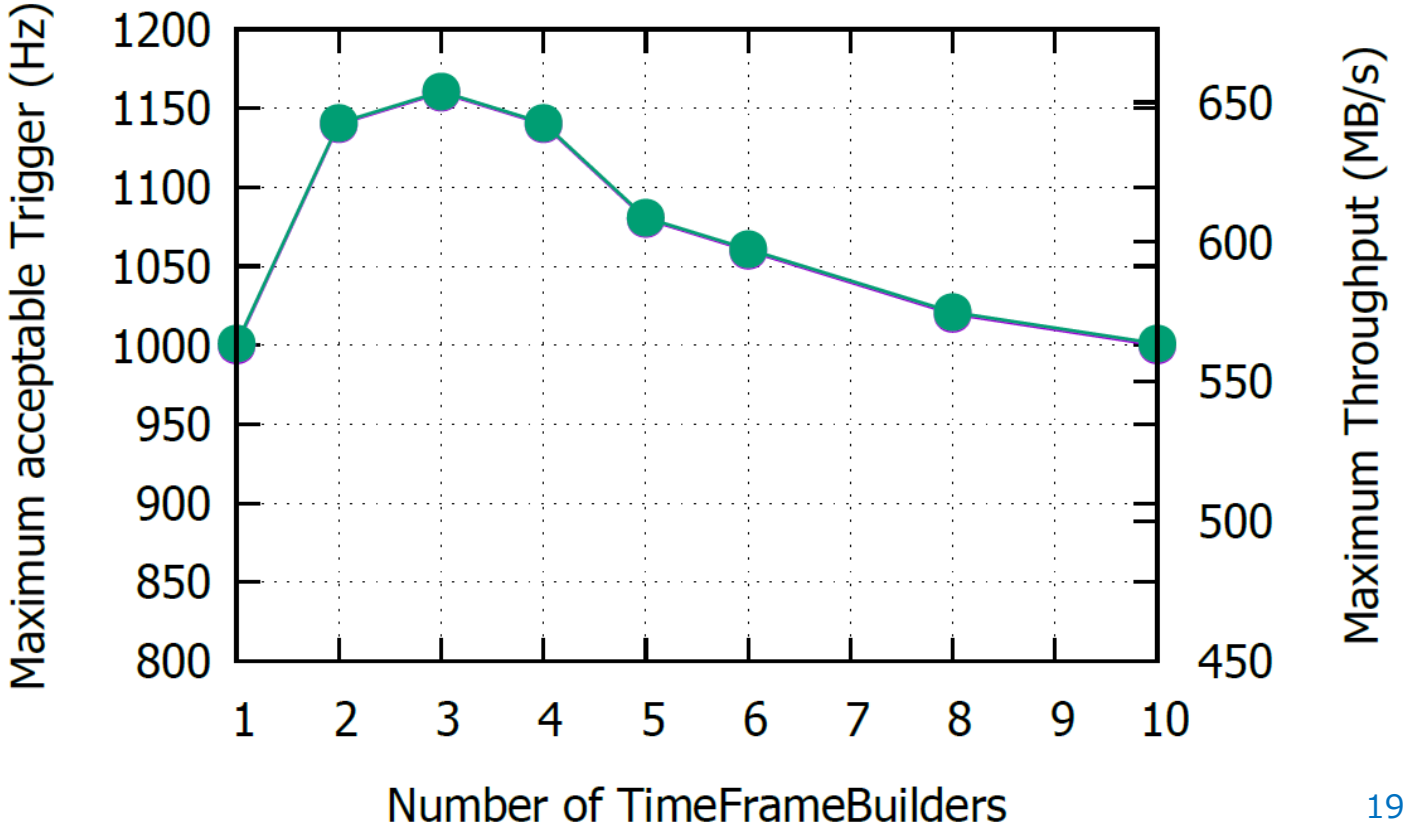
Data flow throughput



Event by event に読んで Event ID を入れた STFB の Header を持ったSampler を作るだけで、何も問題なくそのまま動いた。

Internal process communication に TCP ではなく UDS を使用してみた。

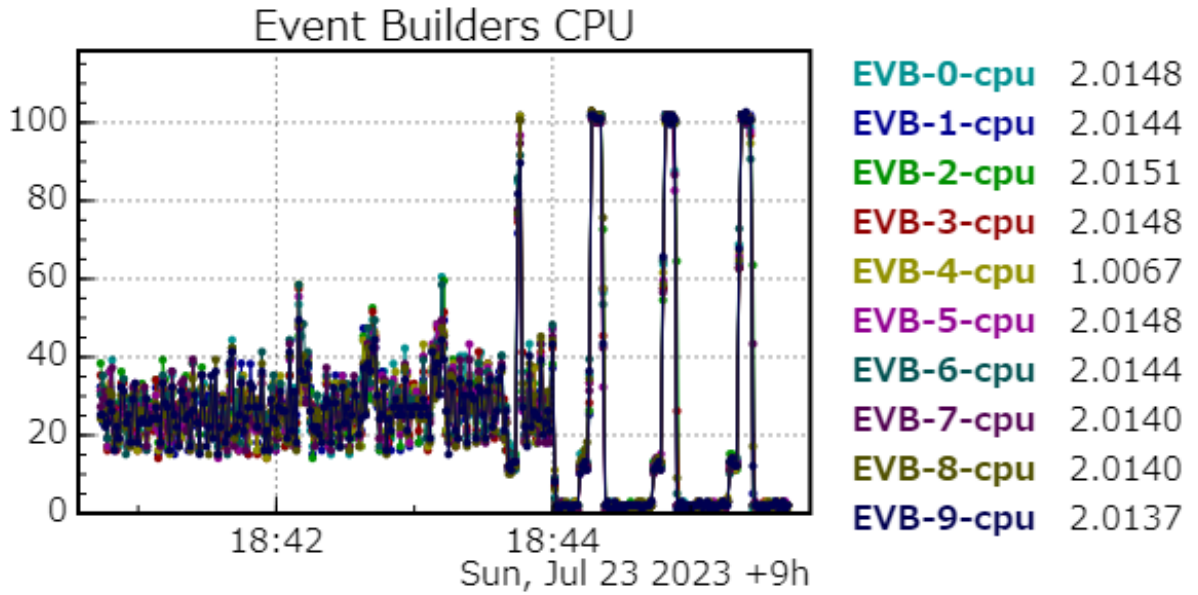
- Event size: 6156B
- Number of Recbes: 96
- DAQ PC
  - Xeon E-2236 @ 3.40GHz 6 Cores
  - Memory 32 GB
  - NIC: Broadcom NetXtreme II BCM57810
- 1G/10G network switch
  - FS.com S3900 24F4S



問題と解決

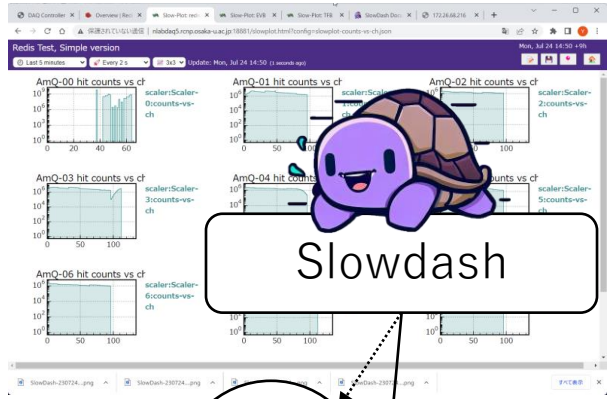
- Data flow の息継ぎ問題
  - GR/ES E585 の時に発現
  - おそらく Event Builder が Multi-part message を作りすぎたためではないかと予想。
    - 今まで 1 multi-part message あたり 100 message 程度だったのが 1000 くらいに増えた。
  - Event builder の出力 message を Event 毎に分けるのをやめた。
    → 一応解決したようだ。

→要追跡調査・理解



Mini booking tool for online display

- TH1, TH2 相当のものを Slowdash に表示させたい。
  - Mini booking tool：uhbook
  - JSON に変換：Slowdashify()
  - redis に書き込み：RedisDataStore




```
Title: Hello
Entry:      100
Over flow:  0
Under flow: 0
0.0e+00:0.0e+00|
6.7e+00:0.0e+00|
1.3e+01:0.0e+00|
2.0e+01:0.0e+00|
2.7e+01:0.0e+00|
3.3e+01:0.0e+00|
4.0e+01:1.0e+00|#
4.7e+01:1.0e+00|#
5.3e+01:1.0e+00|#
6.0e+01:3.0e+00|###
6.7e+01:0.0e+00|
7.3e+01:9.0e+00|#########
8.0e+01:8.0e+00|########
8.7e+01:1.8e+01|##################
9.3e+01:1.3e+01|#############
1.0e+02:1.0e+01|##########
1.1e+02:1.5e+01|###############
1.1e+02:8.0e+00|########
1.2e+02:4.0e+00|####
1.3e+02:6.0e+00|######
1.3e+02:1.0e+00|#
1.4e+02:2.0e+00|##
1.5e+02:0.0e+00|
1.5e+02:0.0e+00|
1.6e+02:0.0e+00|
1.7e+02:0.0e+00|
1.7e+02:0.0e+00|
1.8e+02:0.0e+00|
1.9e+02:0.0e+00|
1.9e+02:0.0e+00|
```

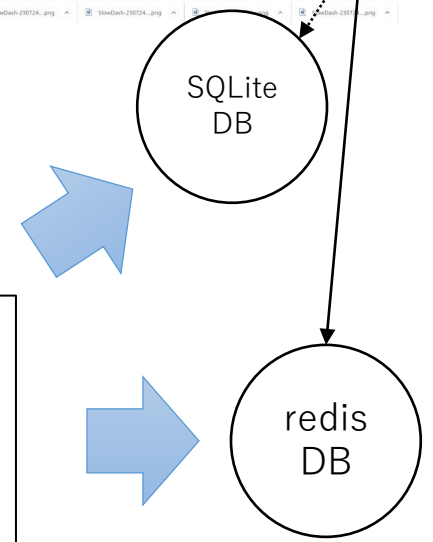
Histogram object

```
Title: Hello 2D
Entry: 10000
Over/Under flow:       0      0      0
                :      0  10000      0
                :      0      0      0

5.0e+01|
5.5e+01|                    . .
6.0e+01|                  .... .. .
6.5e+01|                . ......-...
7.0e+01|                ...-+++----.
7.5e+01|               .---xxx+++--.
8.0e+01|             ..--+x*x@x*+--..
8.5e+01|             ..-++x@@@@x++-..
9.0e+01|            ..-+x*@@##*@xx+-.
9.5e+01|             .--x**/####*++...
1.0e+02|             ..-+*@####@*+-..
1.0e+02|             .--xx@@##@xxx+-
1.1e+02|             .--+x*@@@@*+++.
1.2e+02|              .--+++x*x++x+-..
1.2e+02|             ..--+-+xxx+++..
1.2e+02|               ...--++--...
1.3e+02|                .-...-.....
1.4e+02|                .  ...
1.4e+02|
1.4e+02|
1.5e+02|
```

```
{
        "bins": { "min": 0, "max": 200 },
        "counts": [
           0, 0, 0, 0, 0, 20, 0, 1, 2, 2,
           3, 5, 10, 13, 12, 13, 15, 11, 8, 2,
           1, 0, 0, 2, 0, 0, 0, 0, 0, 0 ]
}
```

JSON



まとめ

- Streaming DAQ software framework
  - FairMQ + redis → NestDAQ
  - データを取るだけならば実用的に使えるようになってきた。
    - 自由に構成が組める。
    - 検出器試験や原子核実験で実績
  - Triggered DAQ にも適用可能
  - DQM/Online monitor 環境も出来てきた。

- 次の課題
  - Log collector
    - たくさんのプロセスの Log や 標準出力を集めて管理する。
    - Fluentbit を評価中
  - Global state の管理と制御
    - 各々のプロセスの状態遷移の順番の制御
  - Data の取り扱い
    - ある程度汎用に使える Data Format
    - 検出器マッピング

情報は SPADI-A mattermost から
https://www.rcnp.osaka-u.ac.jp/mattermost/spadi-alliance
Repository
https://github.com/spadi-alliance