T2K beamline DAQ

坂下健 + 山形 (KEK)

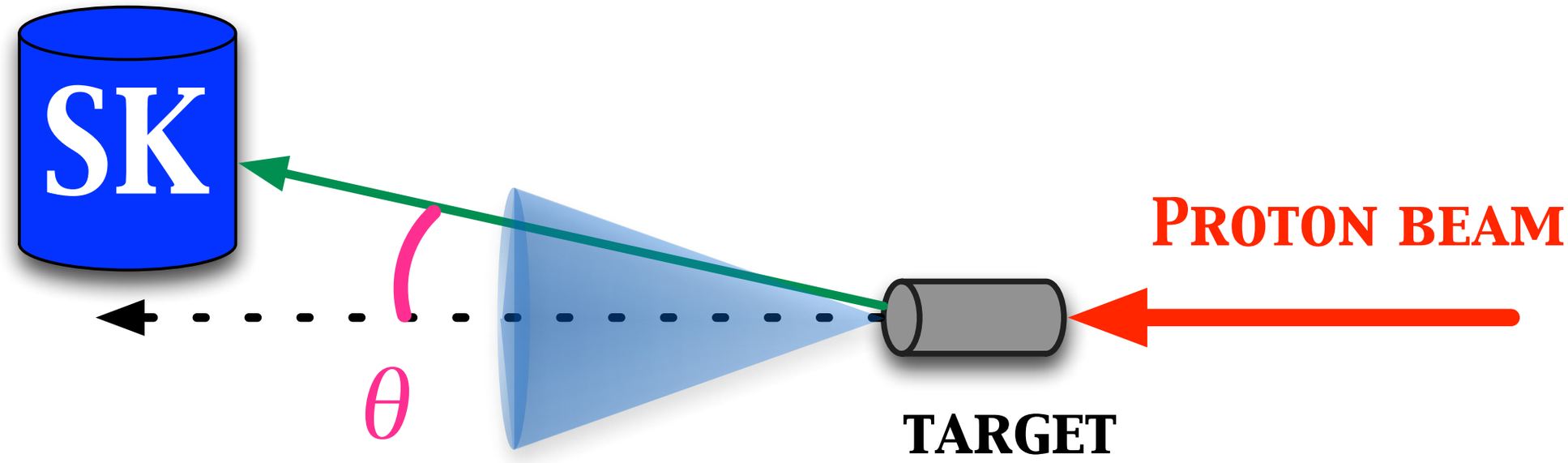
村上明 (京大)

早戸良成 (東大宇宙線研)

T2KのBeamline DAQ

- MRで加速した陽子ビームからニュートリノをつくるところまでをモニターする
  - できたニュートリノは(ほとんど)見えないので、一緒に出来るμも見る
- ビームの形状・通過地点・方向は予定通りか?
  - 予定と違っていたら、次のビーム禁止

T2KのOff-axis beam

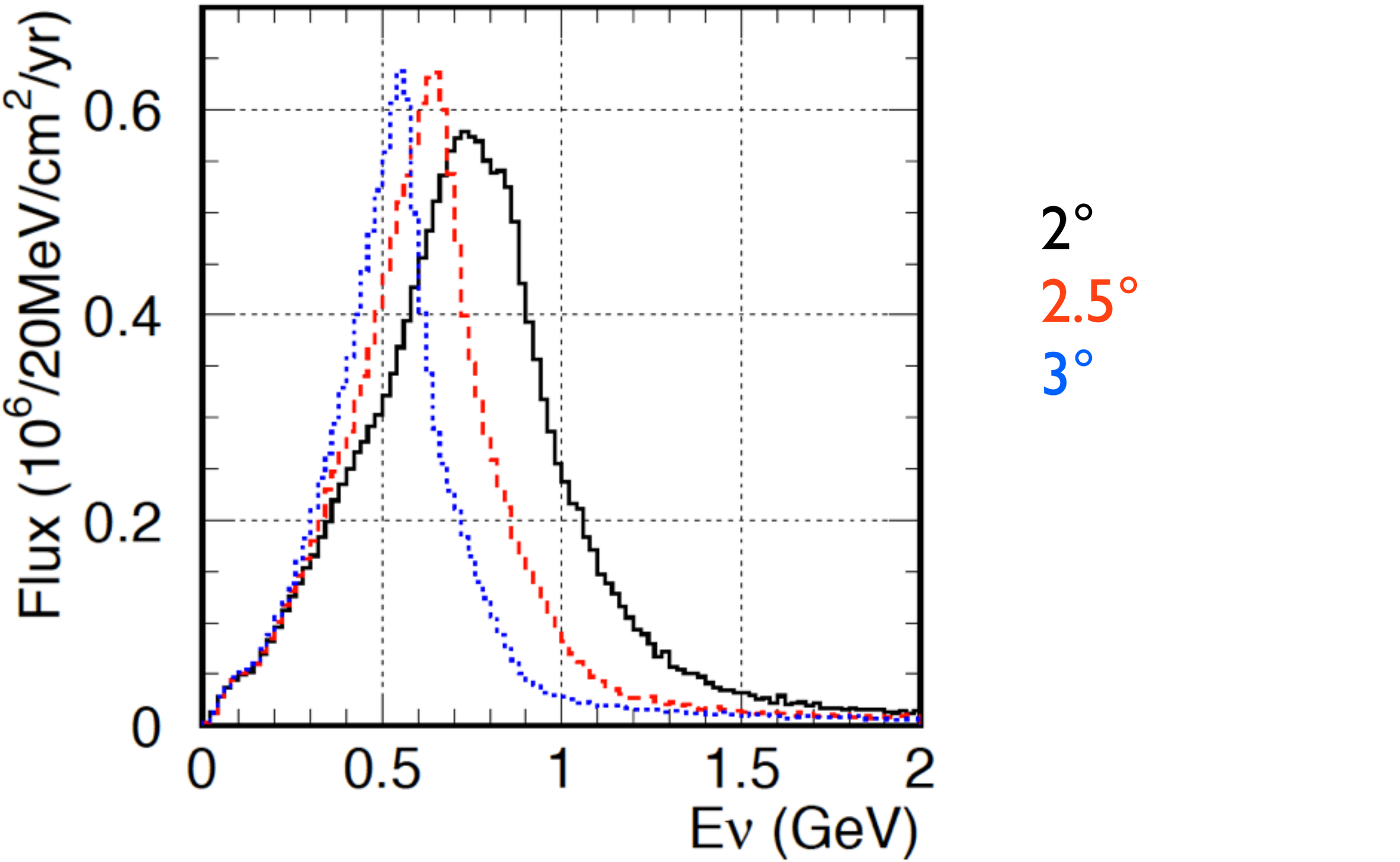


ビーム中心はSKを向いておらず、

中心から角度θの方向にSKがある

(ようにビームを撃つ)

θが変わるとνビームも変わる



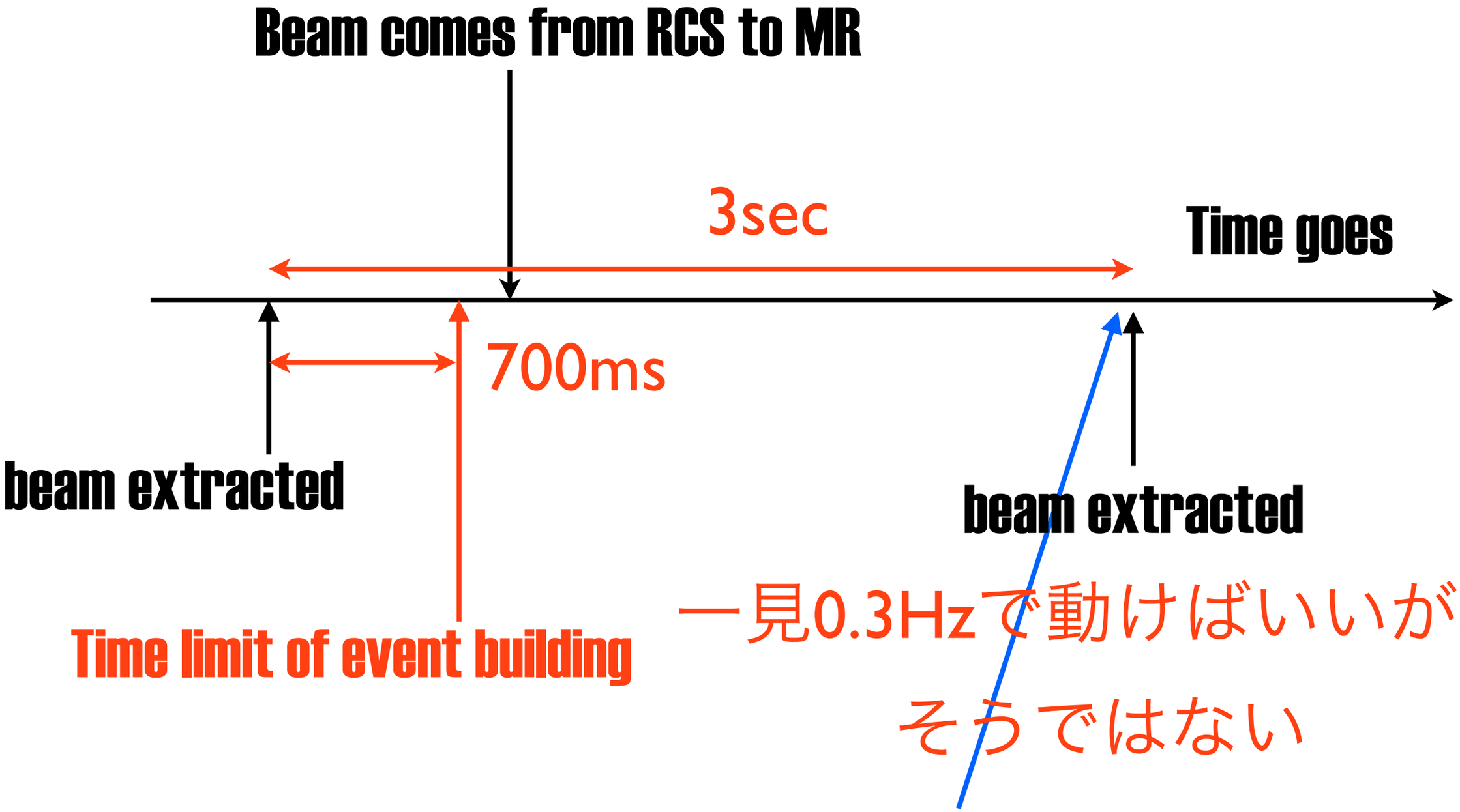
ビームモニタが必要だけど、

- そもそも陽子ビームが直撃するとあちこち心配
- ビーム方向は電磁石で制御しているので、徐々にずれていくことがあるかもしれない
- とにかく全てのビーム状況が見えないとコワイ
- (事後ですが)地震もあったしね

見えなかったら
次のビームは捨てる

- ビーム射出後、すべてのモニタからデータをかき集めてから次のビームをOKにする
- OKにならないと、加速器ごと止める
- 一日一回止めたら「メッ」です
  - Belleとかと異なる

タイムチャート

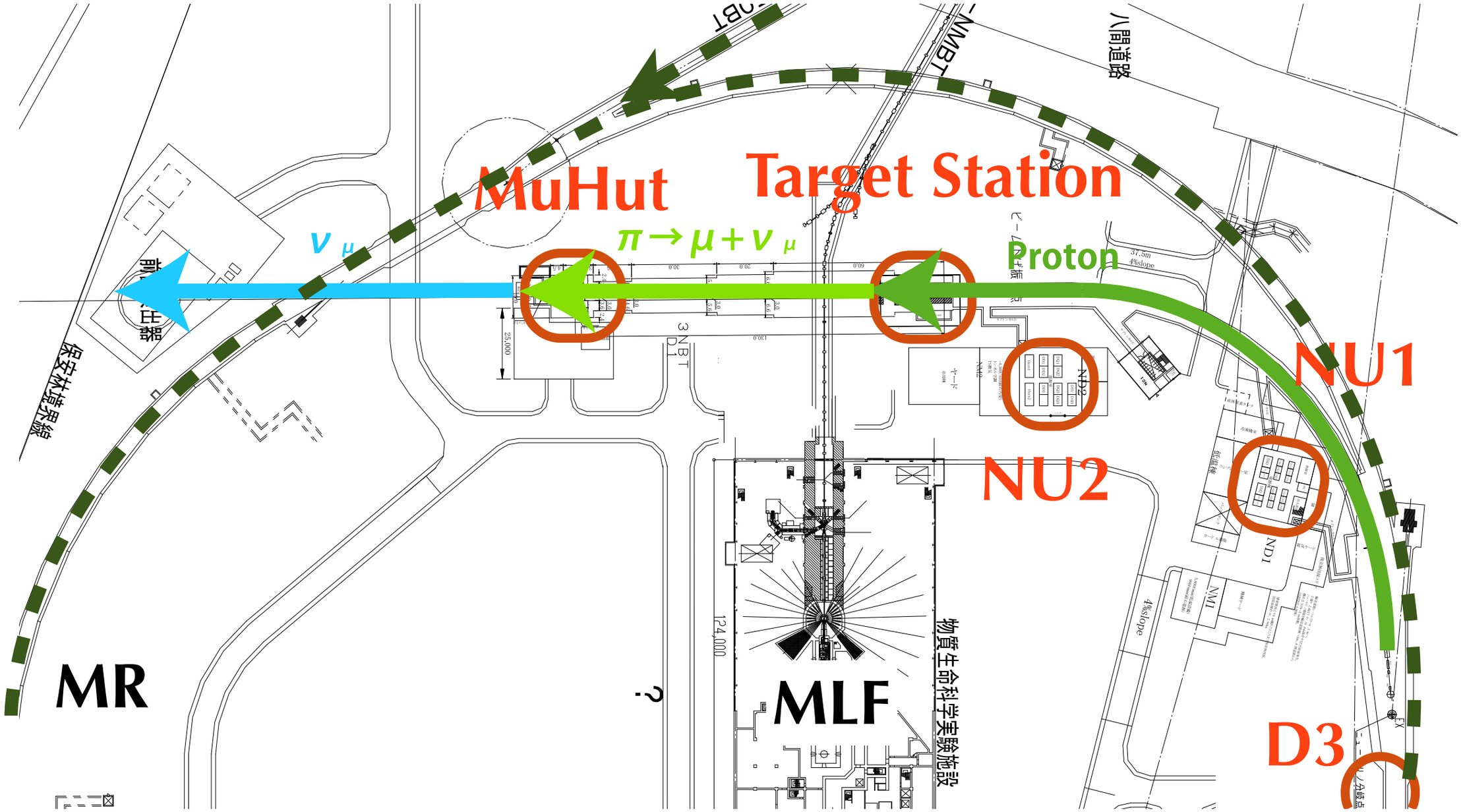


一見0.3Hzで動けばいいが、

そうではない

ビーム許可がらみがなければこの時点までに完了すれば十分

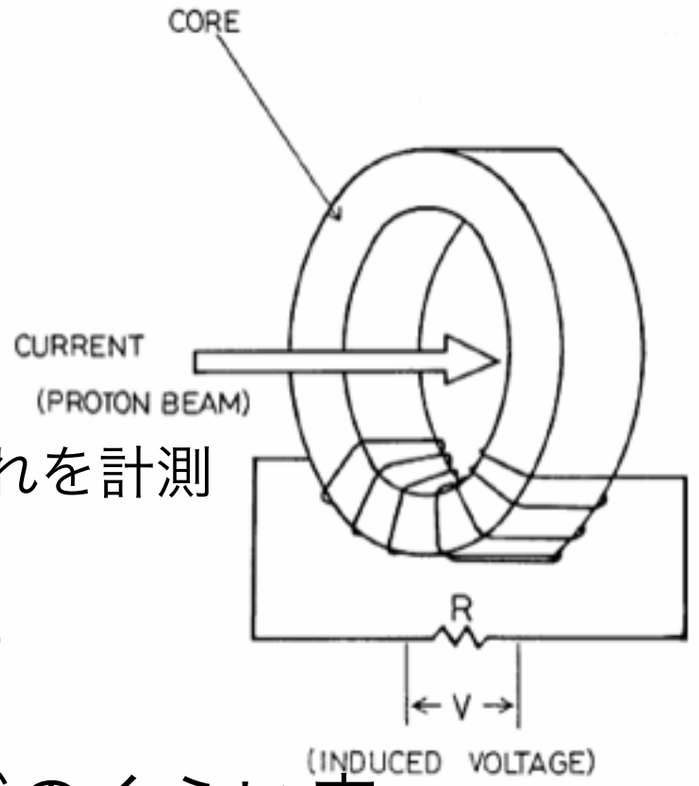
検出器は5つの建物に分散



検出器

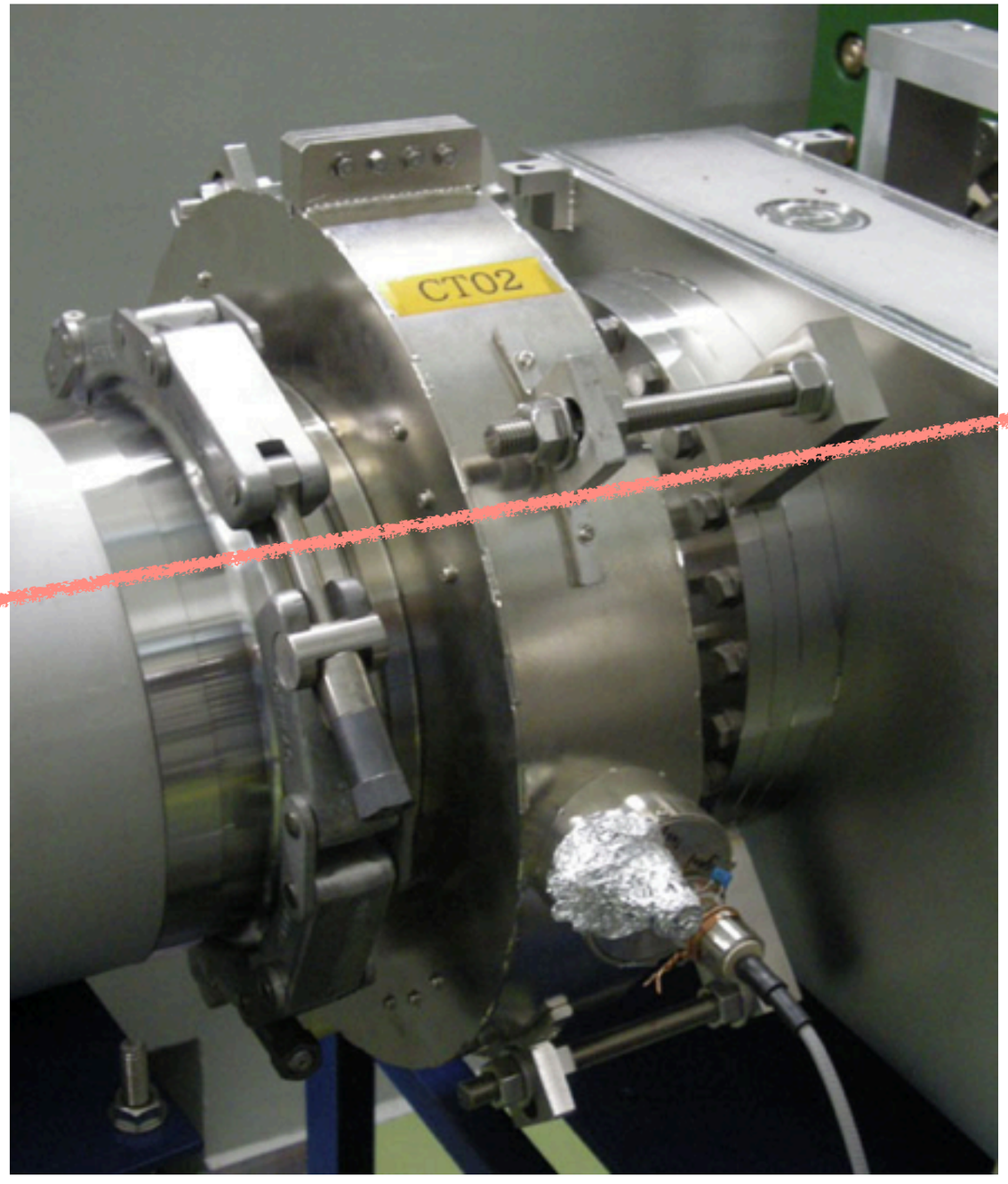
- CT x5
- BLM x50
- SSEM x19
- ESM x20
- OTR x1
- GPS x1 (GPSと原子時計でビーム射出時刻を計る)
- 電磁ホーン x3
  - GPSとOTR以外はすべてFlashADC (2種類)で読む

Current Transform (CT)

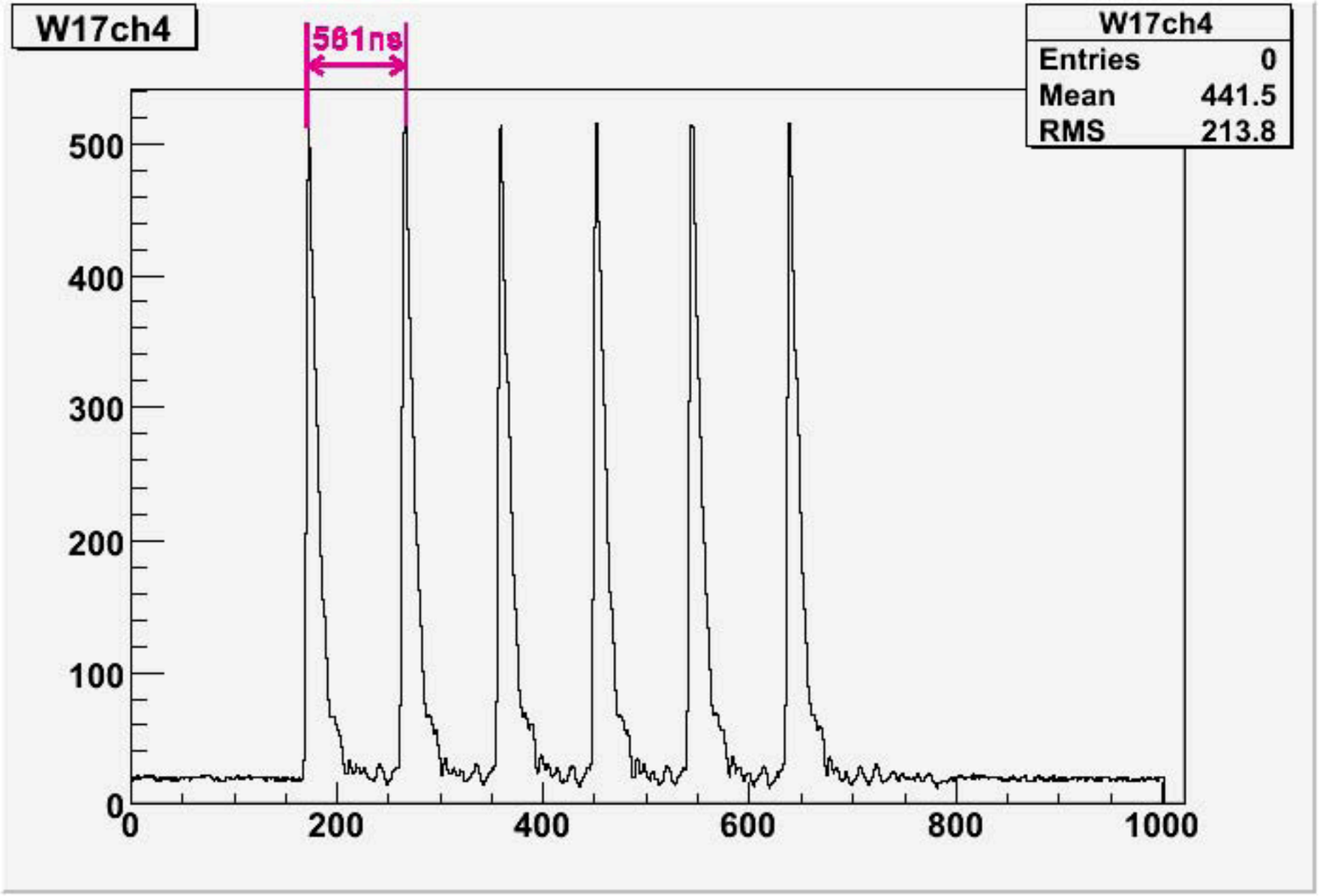


- トロイダルコイルによるトランス
  - 1次電流(ビーム)に応じて2次電流が発生するので、それを計測
- コイルをどのくらいの電荷が通ったかわかる
- スピル全体ではなくてバンチが進行方向にどのくらい広がっているかわかるくらい高速なものを使用
- 経路中のCTで比較することで、どこでビームが減ったかわかる

実際の設置状況



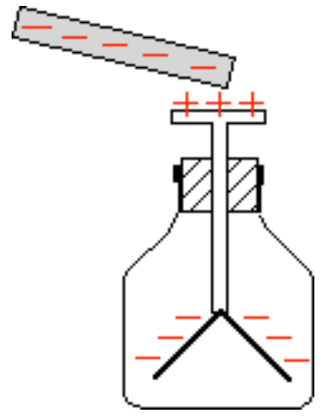
得られる波形



1回の射出で6個バンチが出ていることがわかる

(今は8個になりました)

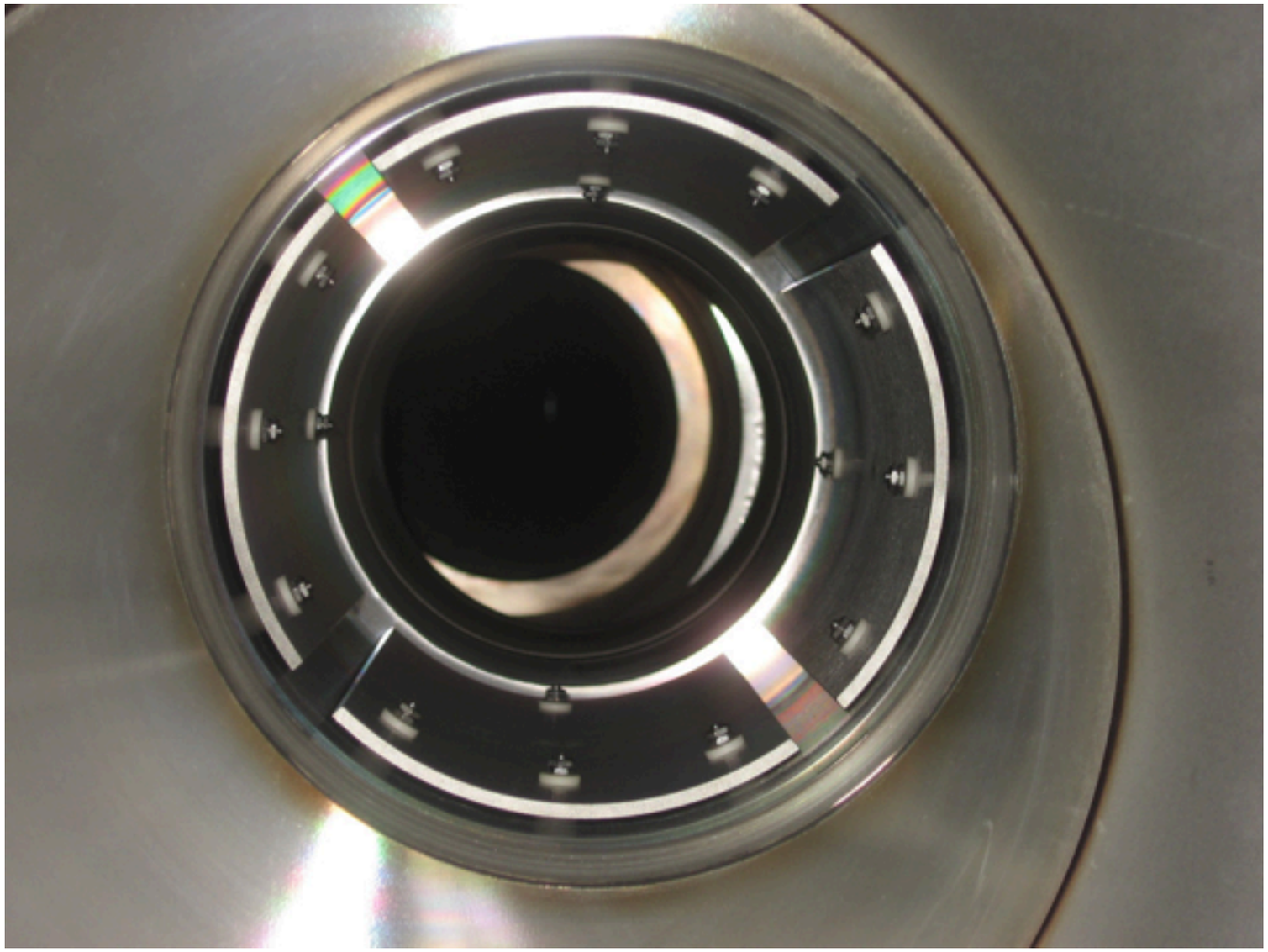
ElectroStaticMonitor (ESM)



- 静電分極を使って測定
- こういうやつありましたよね?(高校物理)
- 金属に帯電している物を近づけると
  近い方と遠い方で分極する
  - 近づき具合が変わると遠い方の電位が変わる
  - 電位の変動は電流として計測できる
  - 電流を測ると近づき具合の微分値が測れる

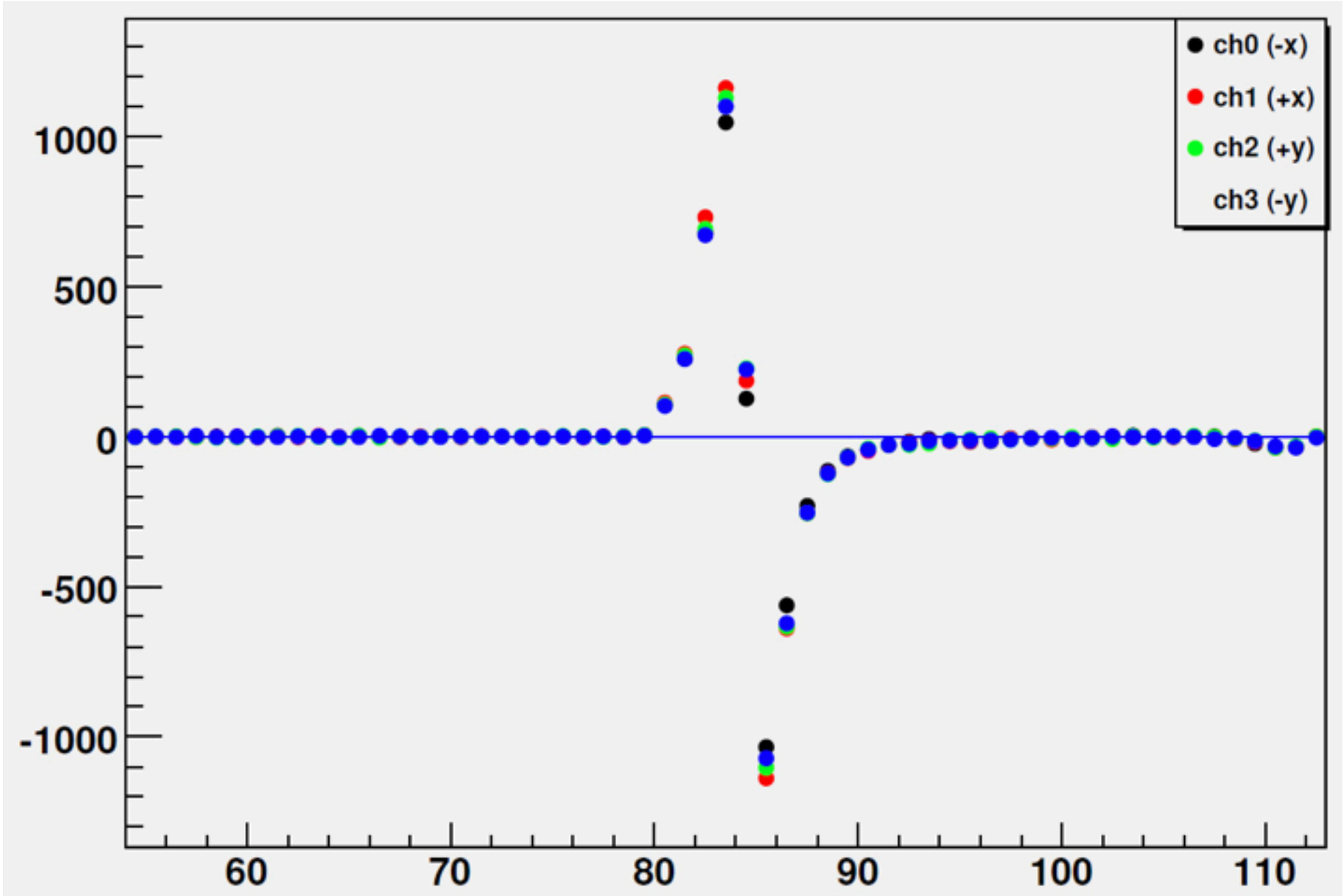
- 近づき具合を複数見て、ビームの中心位置を判定

実際の設置状況



上下左右に4分割

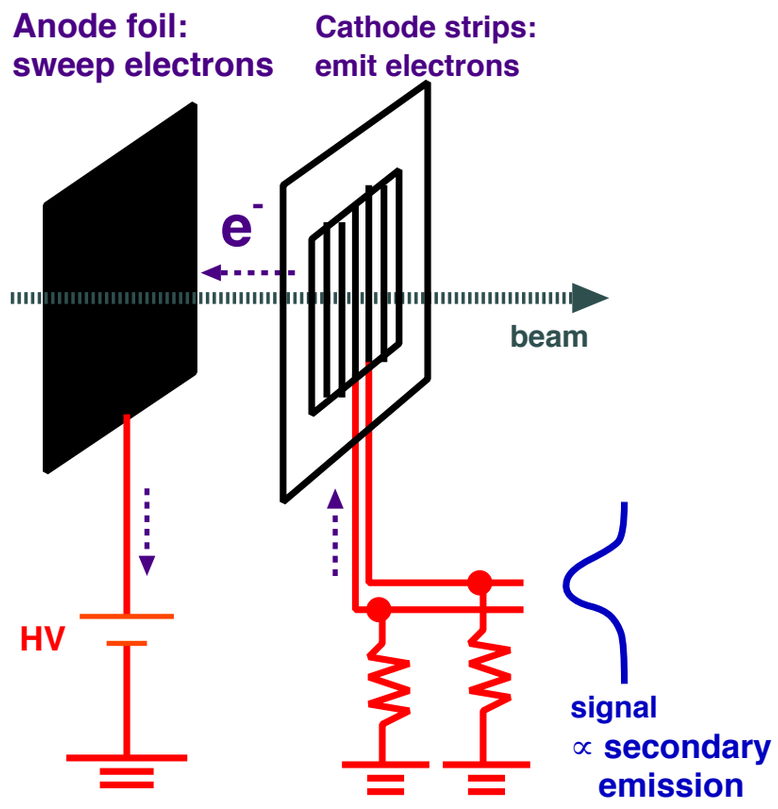
得られる波形



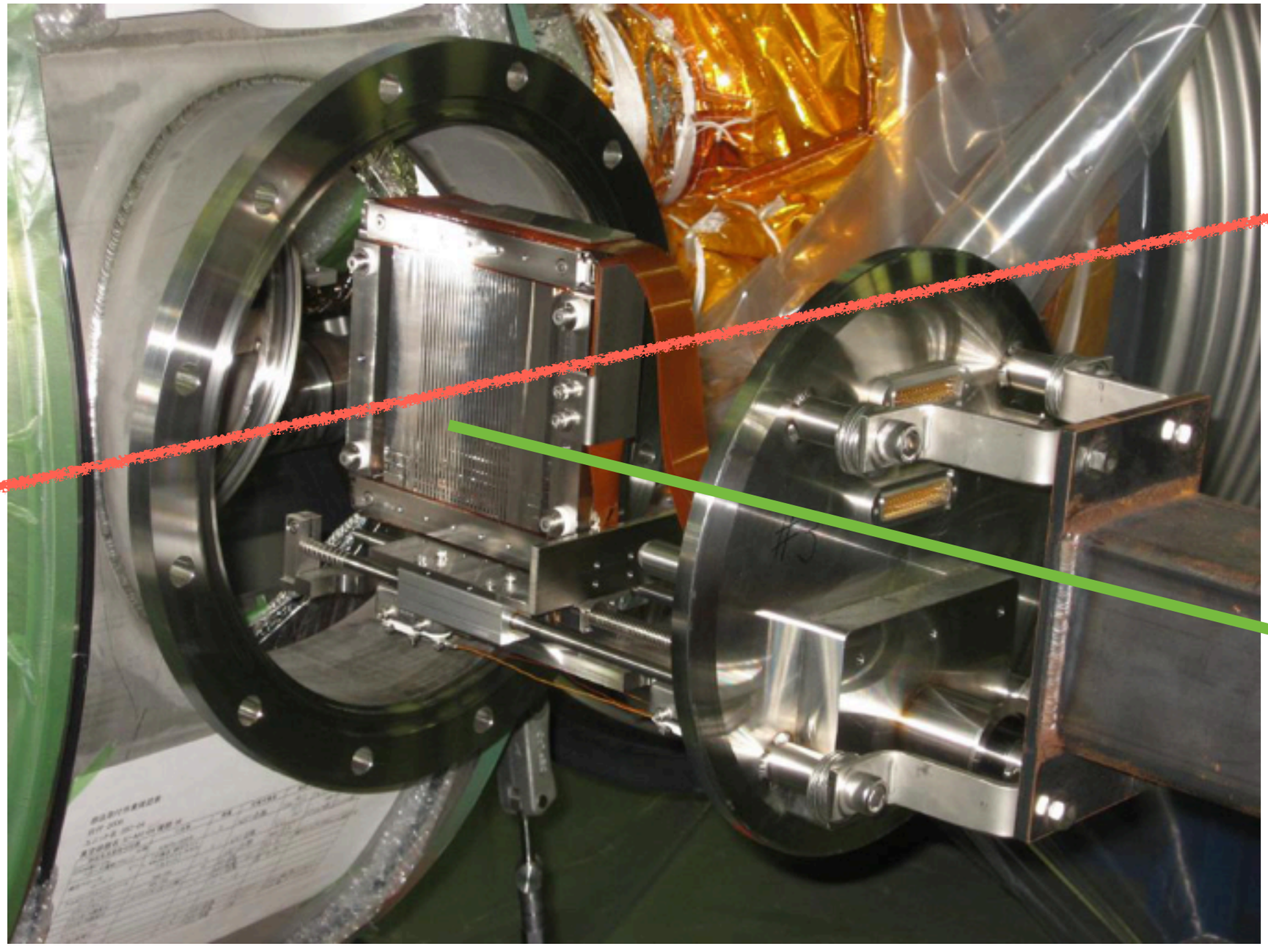
微分値なので変化が速い

Segmented Secondary Emission Monitor (SSEM)

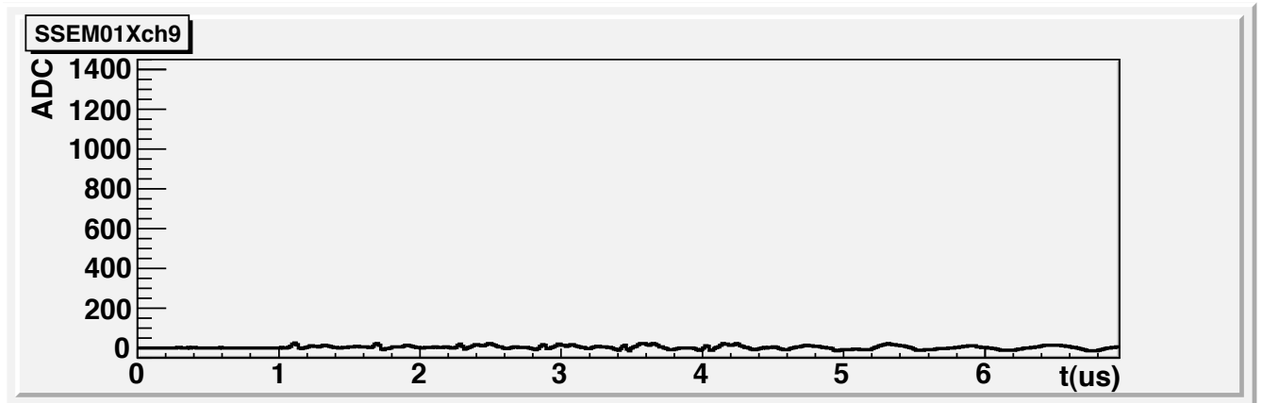
- チタン薄膜にビームを当てて二次電子を計測
- 二次電子の量はビームの電荷に依存している
- 薄膜はすのこ状になっているので、ビームの形状がわかる
- ビームが減るし、薄膜もダメージを受ける
- ビームチューニング時のみビームに当てる
  遠隔待避操作も行う

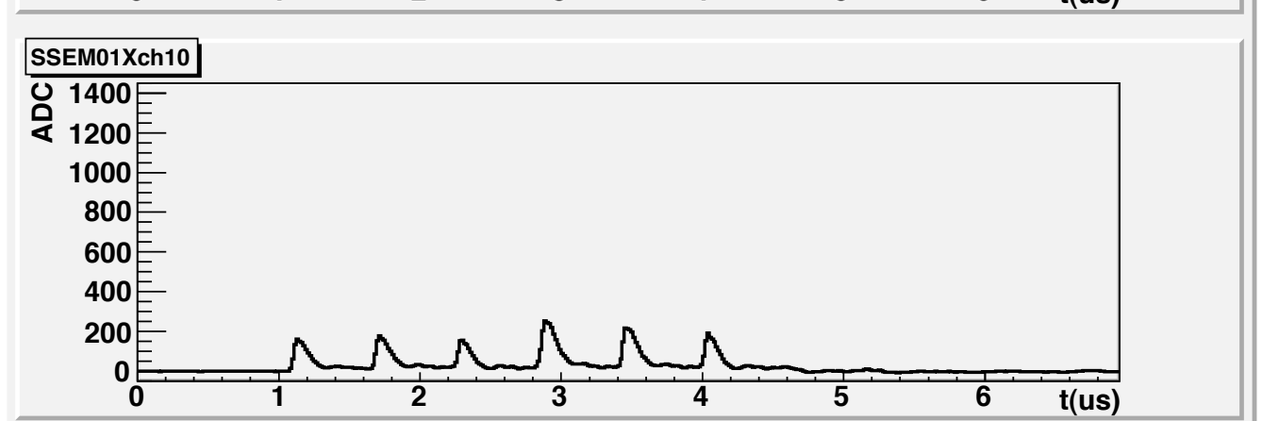


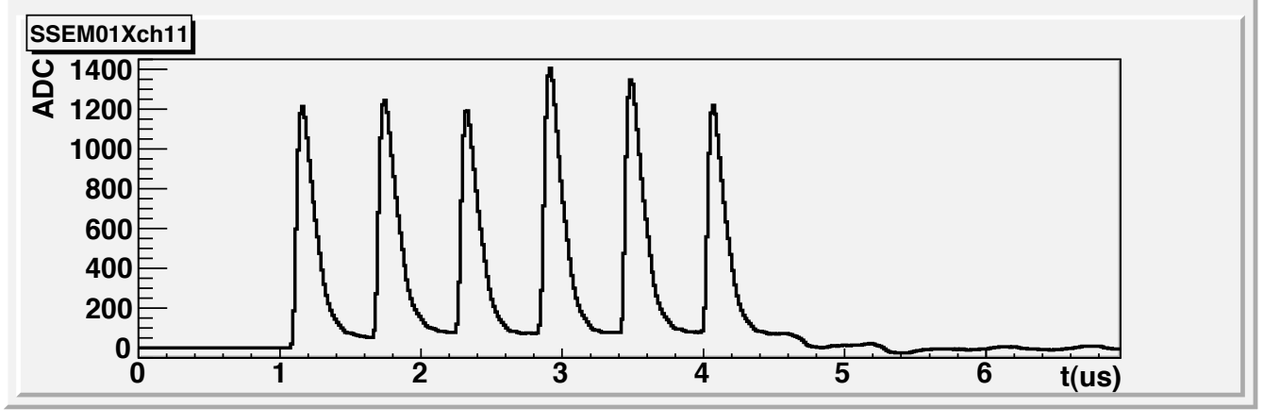
実際の設置状況



得られる波形


SSEM01Xch9
ADC
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5
6
t(us)


SSEM01Xch10
ADC
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5
6
t(us)


SSEM01Xch11
ADC
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5
6
t(us)

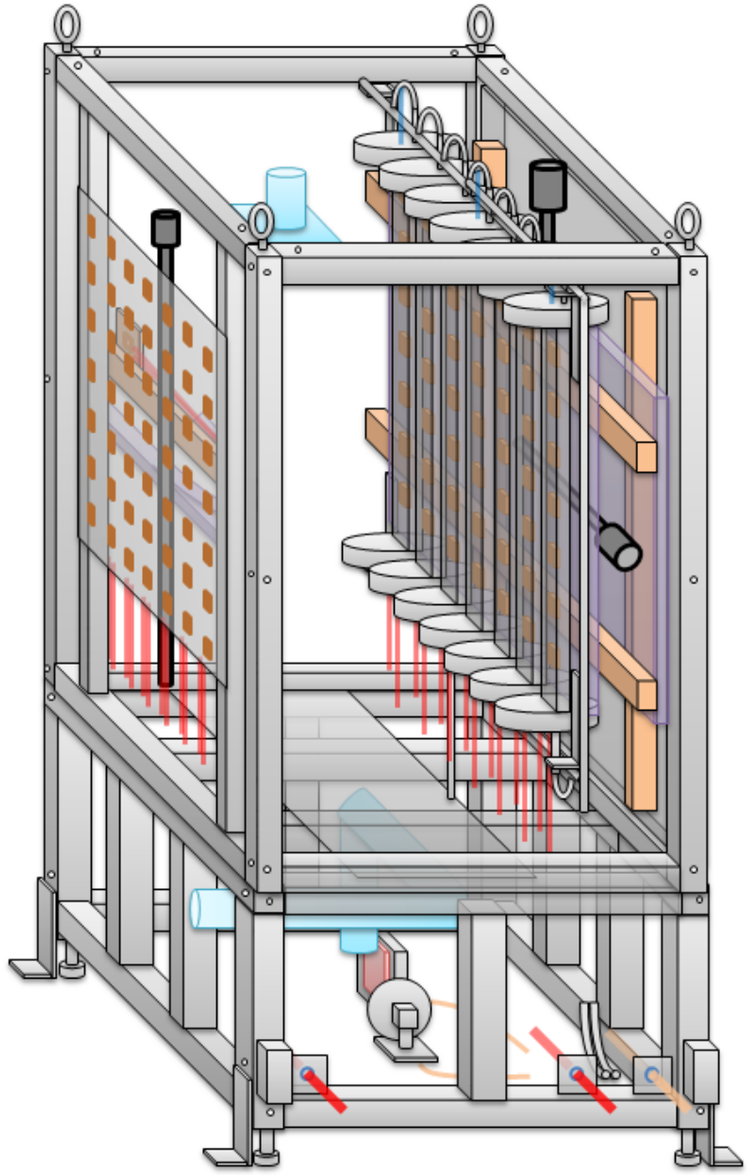
Beam Loss Monitor (BLM)

- アルゴンガスのワイヤープロポーショナルチェンバー
- 説明省略

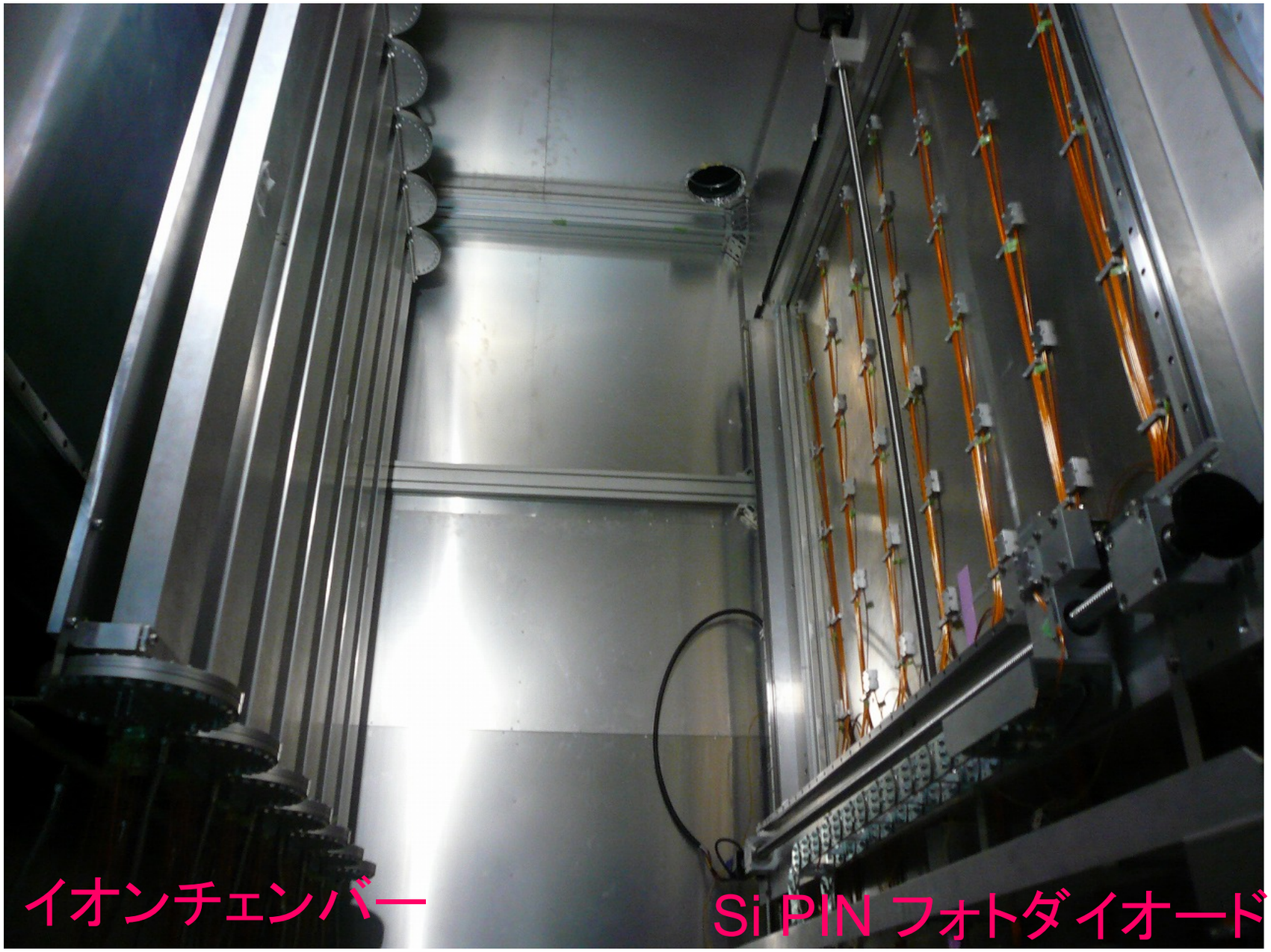
実際の設置状況

MuMon

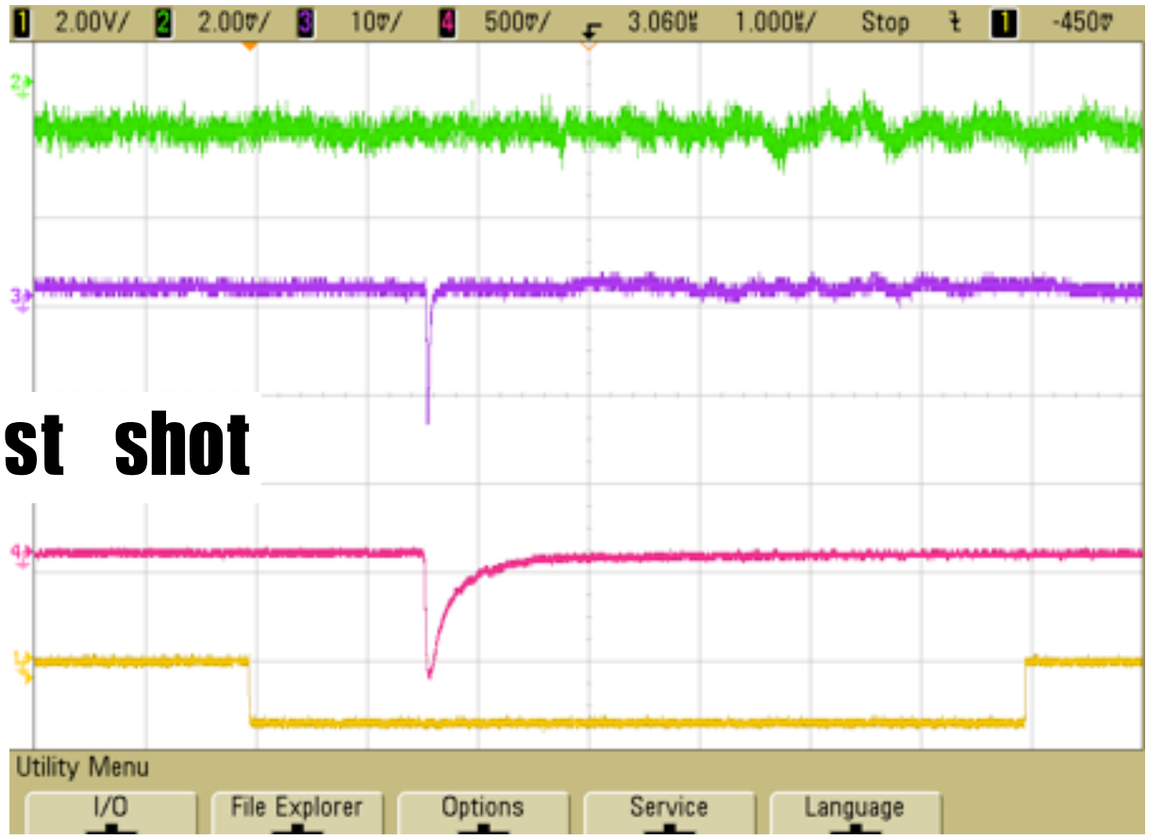
- 1.5m x 1.5m の領域をモニタする
- 検出器は2種類
  - Si PIN PhotoDiode
  - Ion Chamber (Ar + N_2)
  - いずれも7x7
- それぞれの49ch分でビーム強度・角度を測定
- 電磁ホーンでビームが正しく絞れているかもわかる



実際の設置状況



出る信号



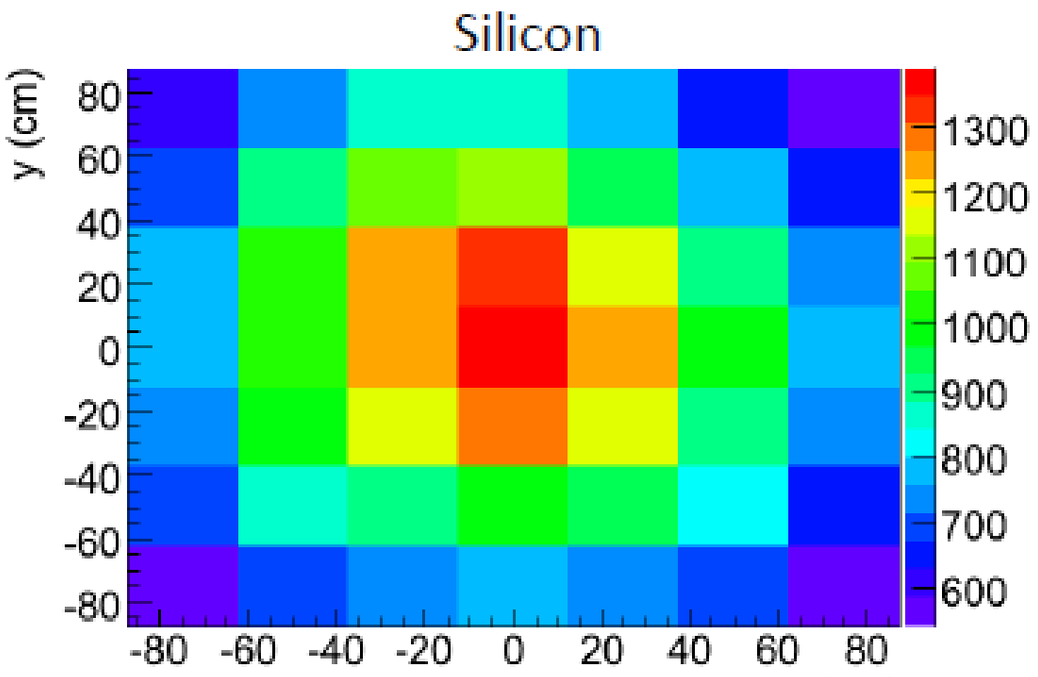
T2K 1st shot

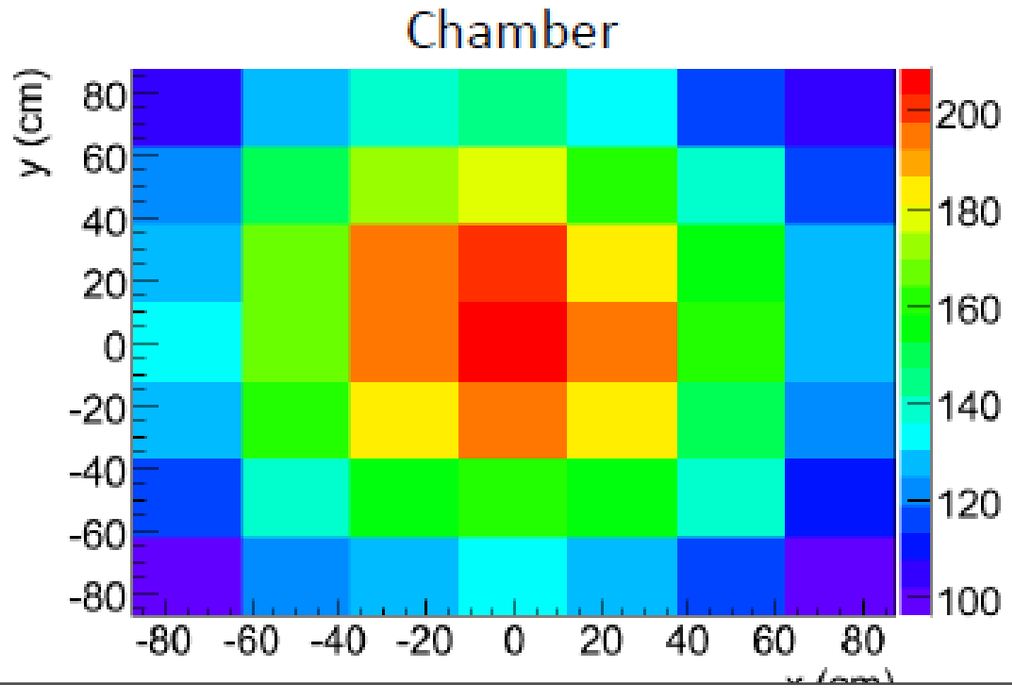
Chamber (まだ小さくて見えない)

Silicon

Scintillator (仮設)

(高エネルギーニュース Vol29No1)

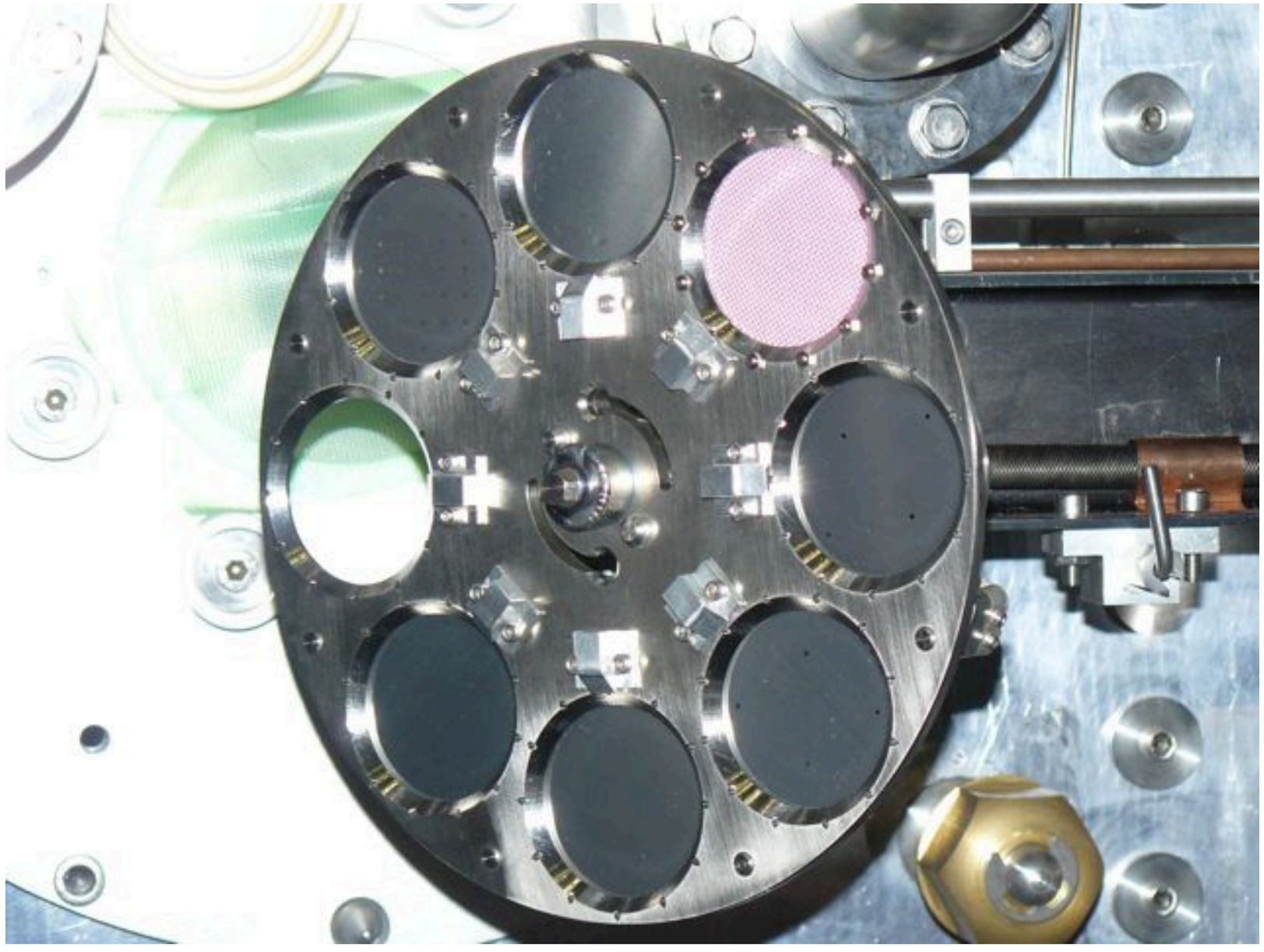




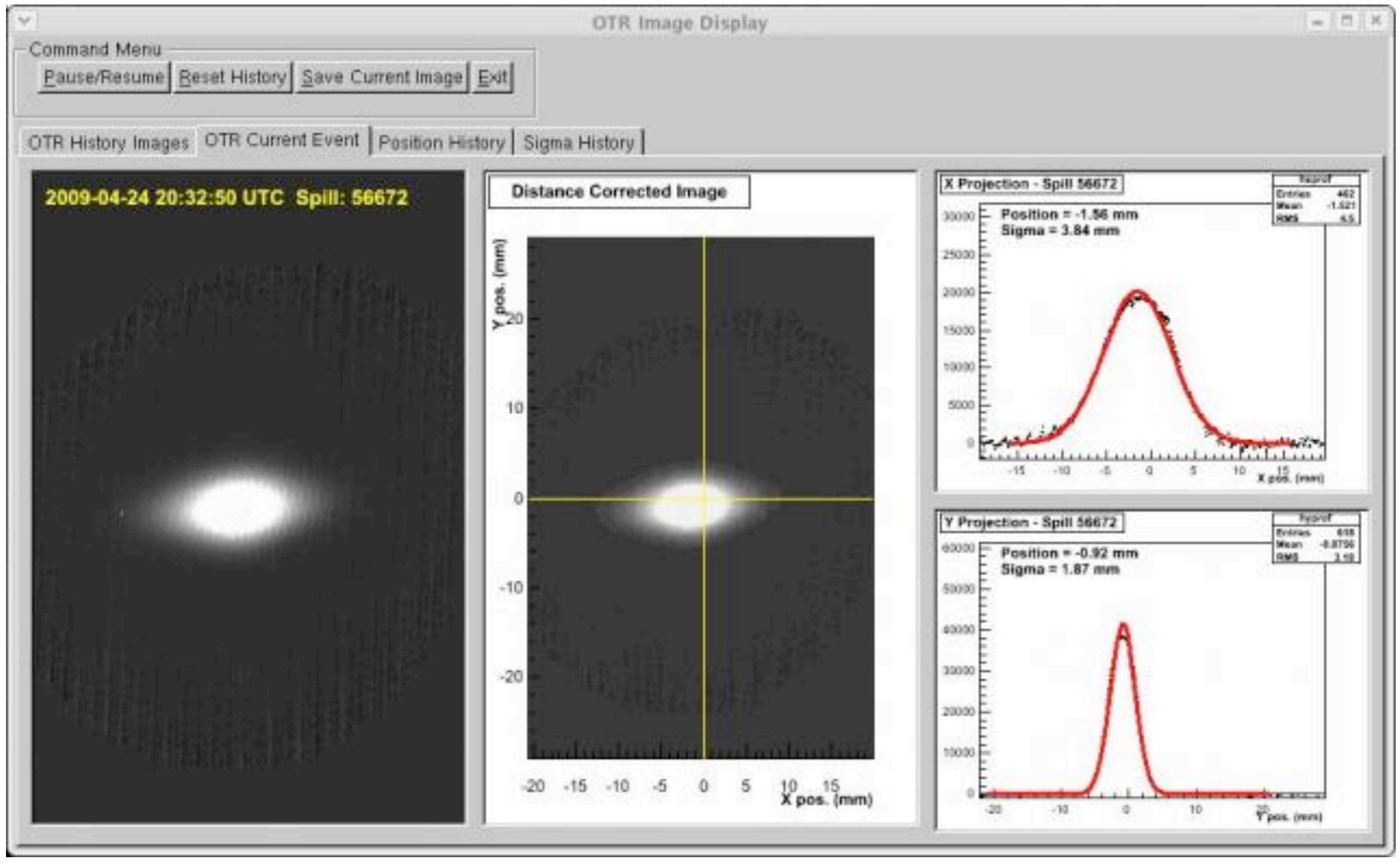
OTR

- Optical Transition Radiation Monitor
- 薄膜にビームを当て、薄膜が出す蛍光をカメラで拾う
- いわゆるイメージングプレート的なもの
- カメラ的撮影 (Windowsで)
- ビーム強度に応じて薄膜の種類を変えるため、リボルバー構造になっている

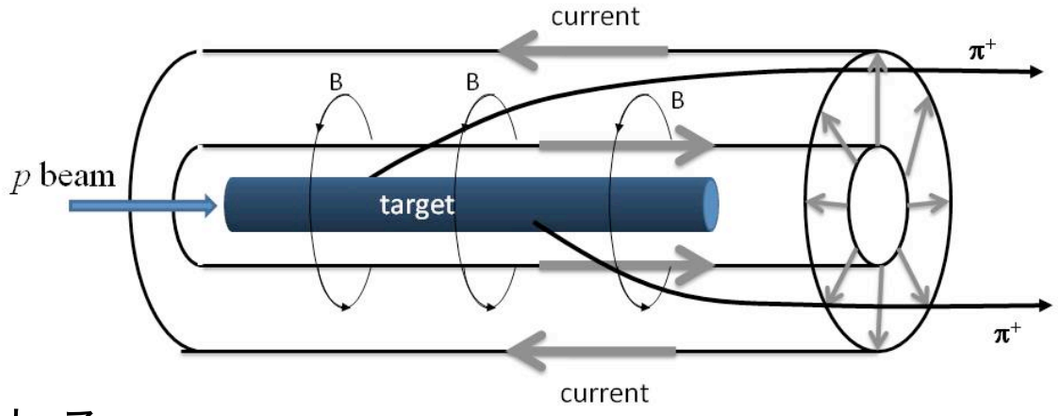
こんなの



測定されるデータ


OTR Image Display
Command Menu
Pause/Resume
Reset History
Save Current Image
Exit
OTR History Images
OTR Current Event
Position History
Sigma History
2009-04-24 20:32:50 UTC Spill: 56672
Distance Corrected Image
Y pos. (mm)
X pos. (mm)
X Projection - Spill 56672
Position = -1.56 mm
Sigma = 3.84 mm
Y Projection - Spill 56672
Position = -0.92 mm
Sigma = 1.87 mm

電磁ホーン



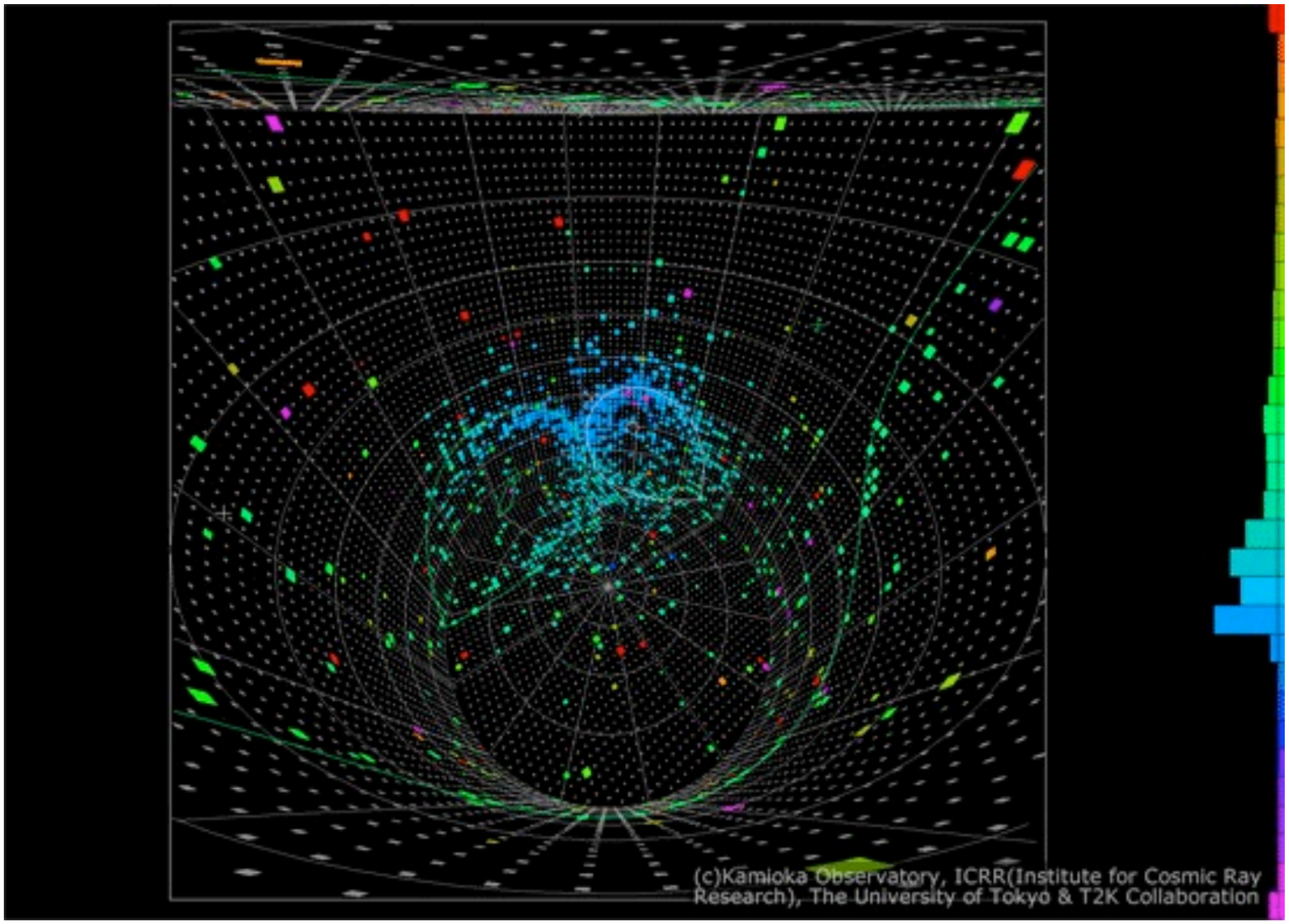
- T2Kでは陽子を炭素標的に当てる
- π中間子が大量生成
- 陽子ビームの運動量によってπは吹き飛ばされる
- 飛んでるうちにμとνμに崩壊するので、ν以外は遮蔽する

$$\text{magnetic fied strength } B\,[T] = \frac{I\,[kA]}{5r\,[mm]}$$

- ほっとくとπは乱雑に散らかり、結果νビームが薄く広範囲に散らかってしまうので、πの段階で電磁石で絞る
- 数百kAとか流して絞る
  - これだけ**常時流すとコイルが熔ける**ので、必要なときだけ電流を流す
  - 高エネニュースの[T2K実験ニュートリノ生成機器]を参照して下さい
- これに正しく流れてるのかもDAQで測る
- 全部で3つある

ビーム運転開始後、

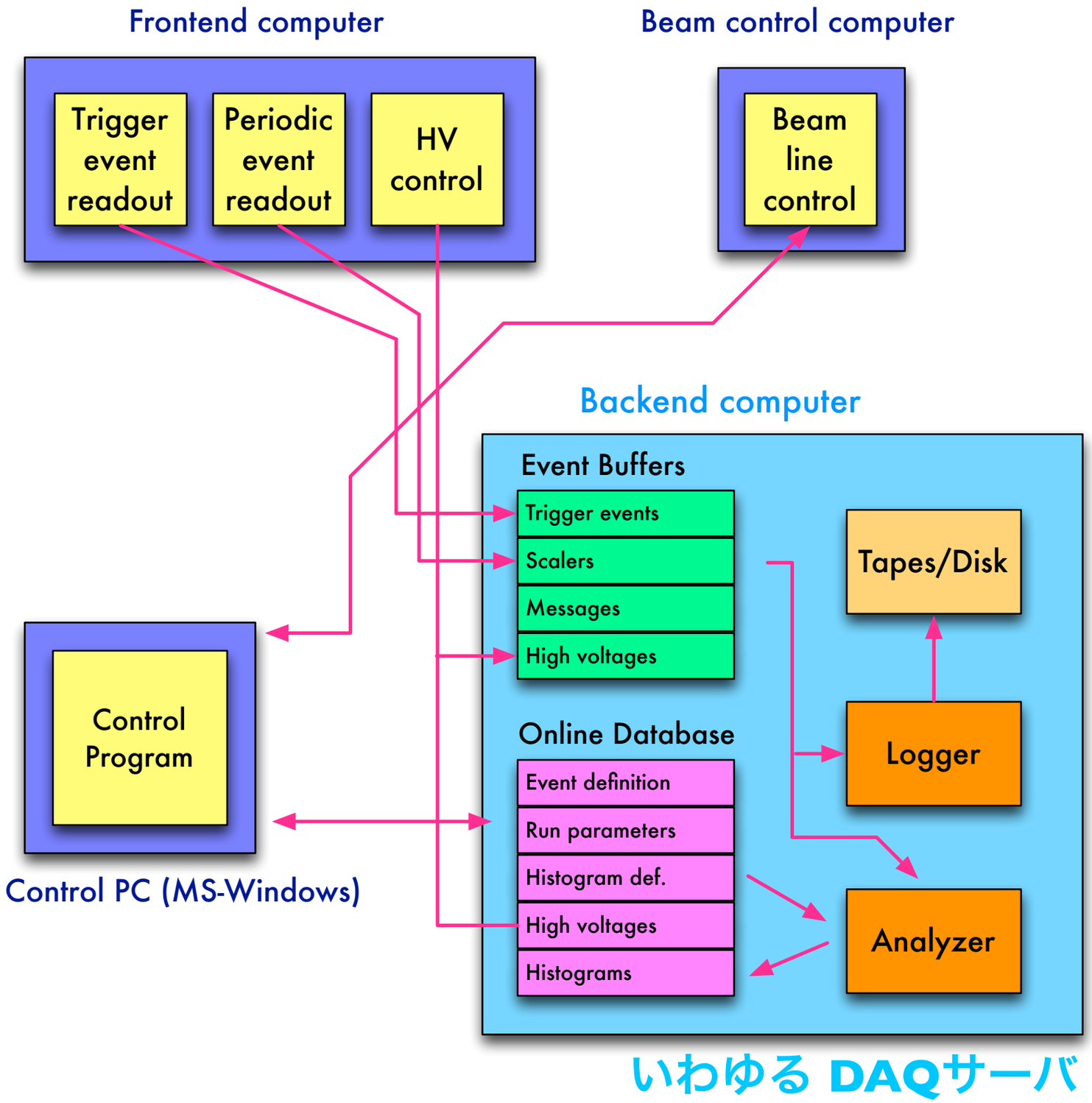
- 無事、神岡でもイベントが見えました



DAQはMIDASで組みました

- そもそもTRIUMFからMIDASの開発者(特上級)が来ている
  - 前置検出器(ND)やINGRID(ビーム中心方向のν検出器)はMIDASで確定
  - 開発者が似た用途で複数のフレームワークを運用するのは不幸
- 全部自分たちでソフトウェアを書く余裕はない
  - 特にOTRは任せっきり
- E391aを耐え抜いたので、弱点がわかっている
  - 迅速立ち上げには大変重要
  - クリアできそうな目論見はある(なければ違うところに行くしか....)
- フレームワークを自分で作ると英会話で説明→めんどくさい
  - 「そういう使い方は想定してないの」を説明するのは大変疲れる
  - MIDASならばコラボレータに「そうなっている」というだけで済む
  - ま、過不足あれば自分で改造すればいいし
  - ソースが小さく、よそのライブラリに依存しておらず、読んでも苦労が少ない

MIDASの構造



- データの授受はシステムが用意する共有メモリバッファを経由する
- バッファ管理はMIDASライブラリが行う

Online Database (ODB)

- MIDASはテキストの設定ファイルを使わない
- ODBに書くのが定石
  - SQLとは関係なく、しかけとしてはXML
  - MIDASは自前超小型XMLライブラリを使用している
  - XMLとしてのAPIは全て隠蔽し、ODBのAPIだけに抽象化
  - レコードの変更待ちが可能
    - 「このへんの数値が変更されたらこの関数を呼んでね!」
    - そして変更はブラウザやCUIから可能!
  - ついでにイベントデータもこれに格納できる
    - 本気でやると遅くなります
    - 温度計とか統計情報などの遅い人達のぶんはOK
- マシンがリブートしたり、ハードウェアの交換をしてもDAQサーバのODBが失われなければ設定情報は蒸発しない
- CUIやGUIから直接ODBの中身を操作できる

Following example creates a record under /Equipment/Trigger/Settings,
opens a hot-link between that record and a local C structure
trigger_settings and registers a callback function trigger_update()
which gets called each time the record is changed.
\code

```
struct {
  INT level1;
  INT level2;
} trigger_settings;
char *trigger_settings_str =
"[Settings]\n\
level1 = INT : 0\n\
level2 = INT : 0";
void trigger_update(INT hDB, INT hkey, void *info)
{
  printf("New levels: %d %d\n",
    trigger_settings.level1,
    trigger_settings.level2);
}
main()
{
  HNDLE hDB, hkey;
  char[128] info;
  ...
  cm_get_experiment_database(&hDB, NULL);
  db_create_record(hDB, 0, "/Equipment/Trigger", trigger_settings_str);
  db_find_key(hDB, 0,"/Equipment/Trigger/Settings", &hkey);
  db_open_record(hDB, hkey, &trigger_settings,
    sizeof(trigger_settings), MODE_READ, trigger_update, info);
  ...
}
```

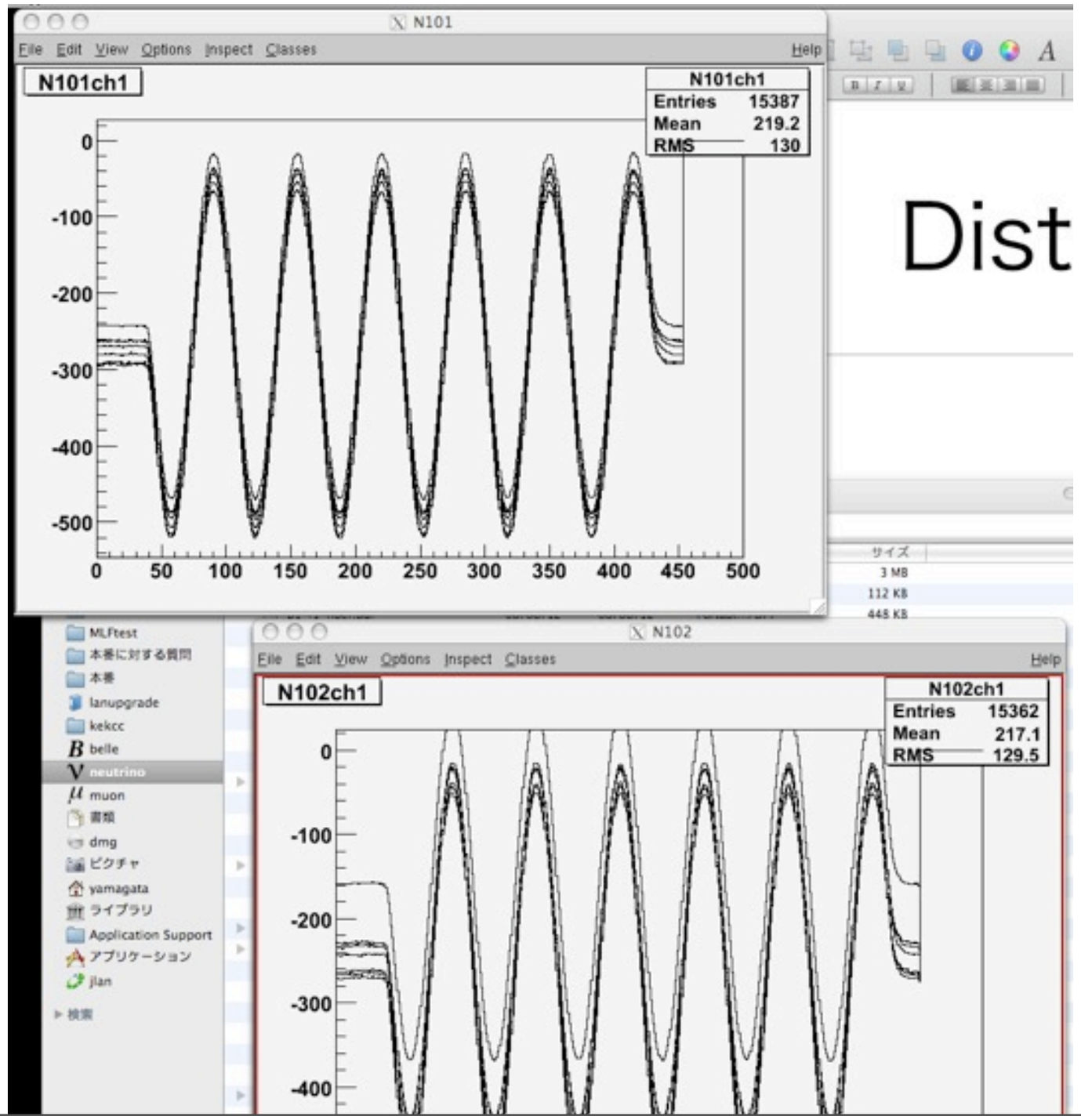
ハードウェア読み出しのために用意するべきコード

- イベントは到着しているか?
  - ブロック待ちしなくてよい。あるかないかわかればよい。
  - ビジーループにならないようにMIDASが面倒見てくれます
  - VMEのレジスタを見るとか
  - NEUNET型のSiTCPならば「データある?」ときいて返事を見る
  - COPPERやSiTCPの垂れ流しならpollとかselect
- 1イベント分を読み出す
  - 固定長整数の羅列として取り出せれば楽
  - 謎バイナリ構造体はデコードしてメンバー毎にバンク割当したほうがよい
- プログラム起動時の初期化
  - 重複起動の防止はシステムではやってくれません
- RUN開始・終了時の再初期化コード

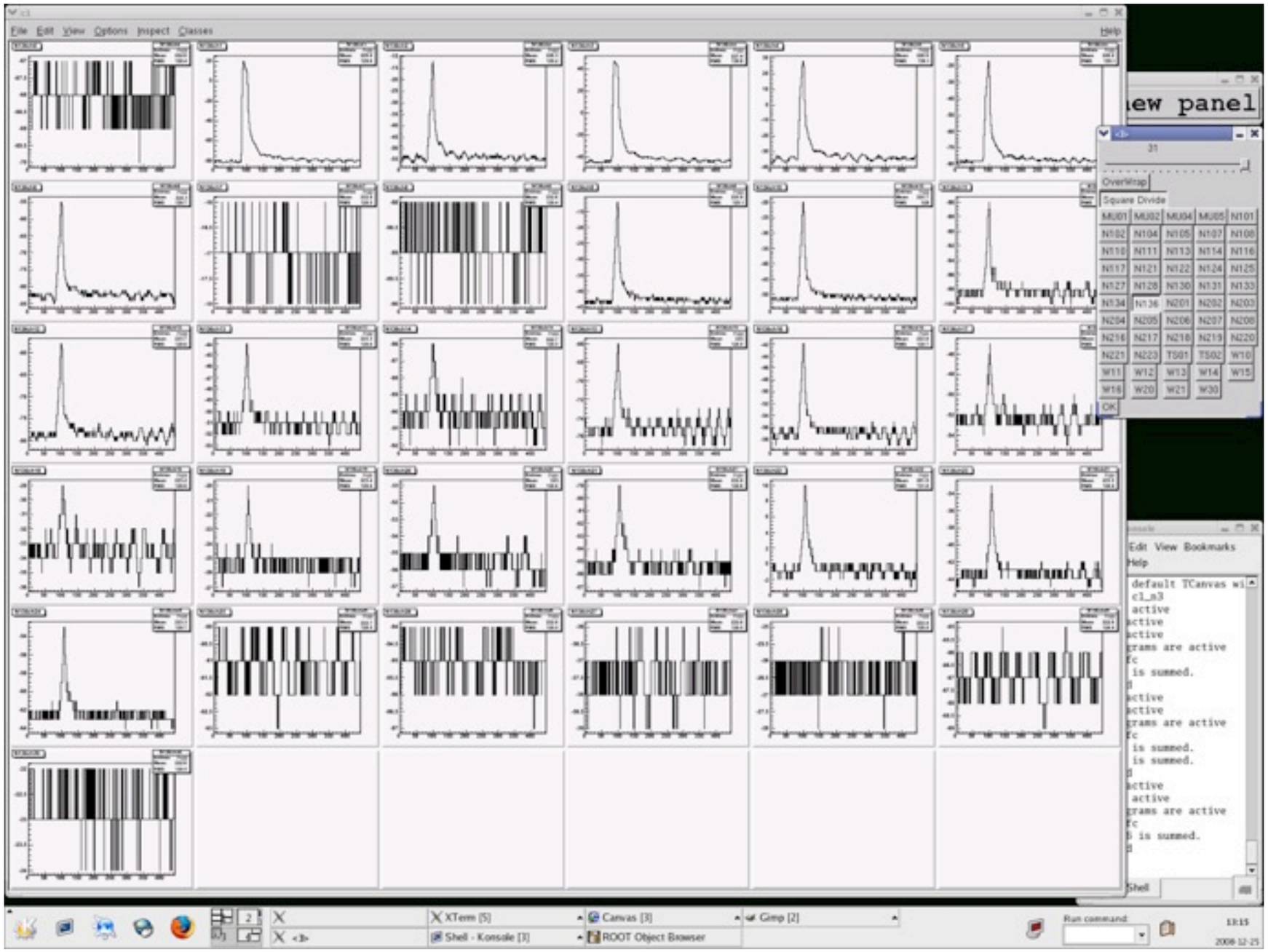
デコーダ

- ADC
  - ほとんど全ての検出器がFlashADCで読まれている
  - すぐに波形が見たい!(よね?)
  - ADC自体は2種類あるので、デコーダも2種類必要
  - 生データからTH1に変換するクラスを用意
    - すぐにDraw()できるようにすれば客もついてくる
      - 測定器の人は忙しいので再利用できないライブラリの勉強をさせるのは悪
      - せめてキャリブレーションにも使える程度には....
    - 両方ともTH1にしてしまえばユーザーはADCの種類を気にしない
    - TH1だと実体も配列なので変換も再利用も楽
    - TGraphにするとデータサイズ倍だし
- GPS
  - バイナリ構造体がやってくるので、仕様をもらって自作
- OTR
  - 完全におまかせ(PNGがやってくるので、手が出ません)

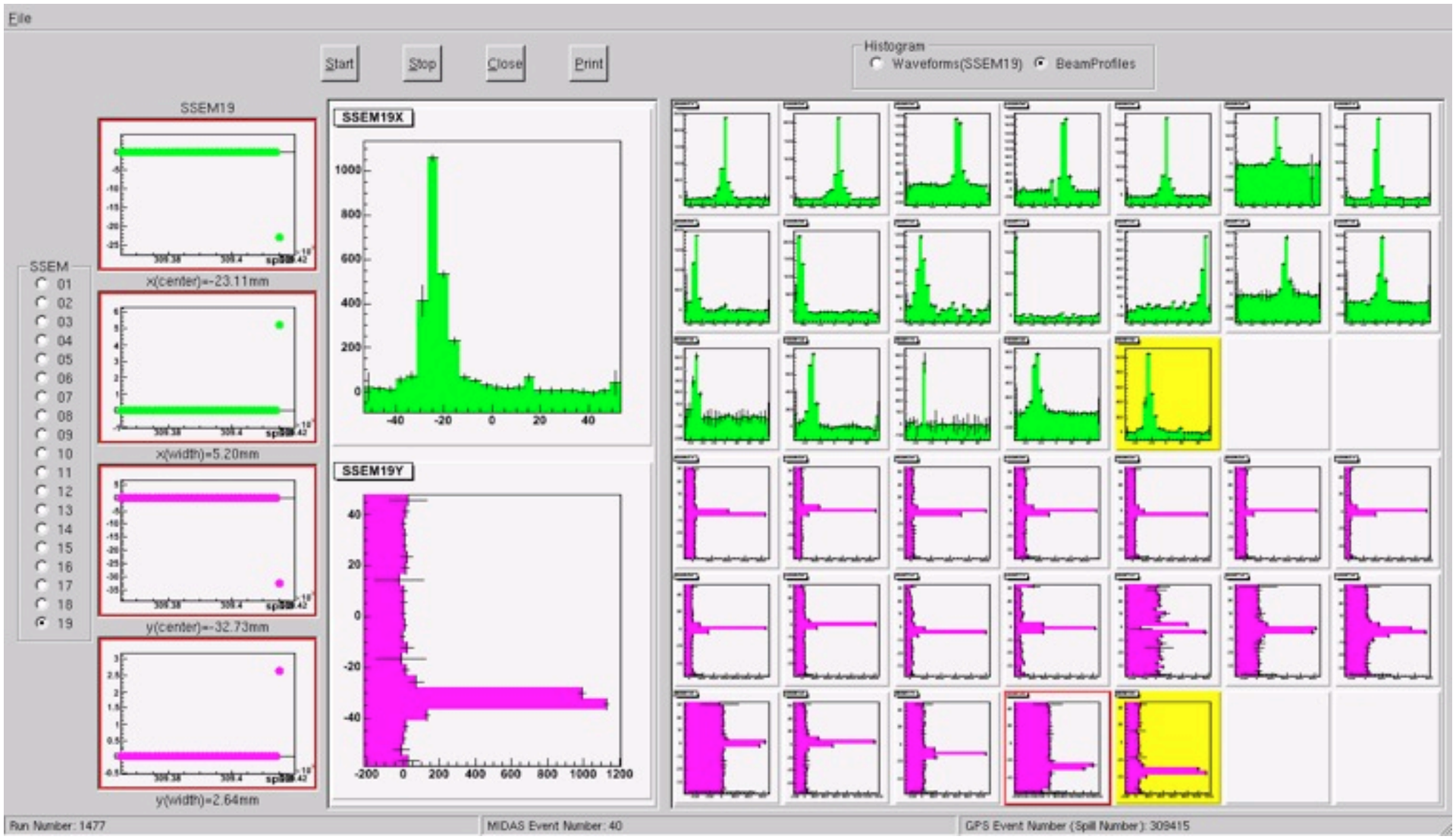
リアルタイム可視化の
サンプル (1)



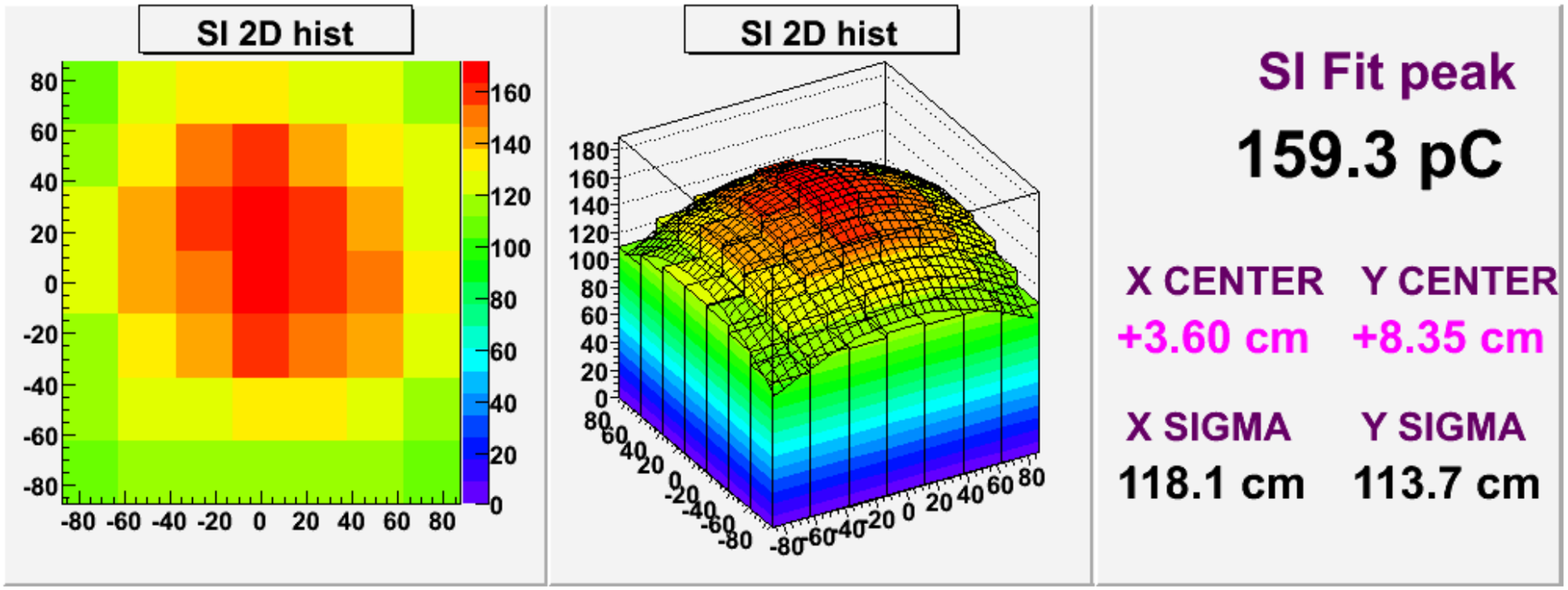
可視化のサンプル(2)


File Edit View Options Inspect Classes
Help
ew panel
Overlap
Square Divide
Edit View Bookmarks
Help
Shell
XTerm [5]
Canvas [3]
Gimp [2]
Shell - Konsole [3]
ROOT Object Browser
Run command:

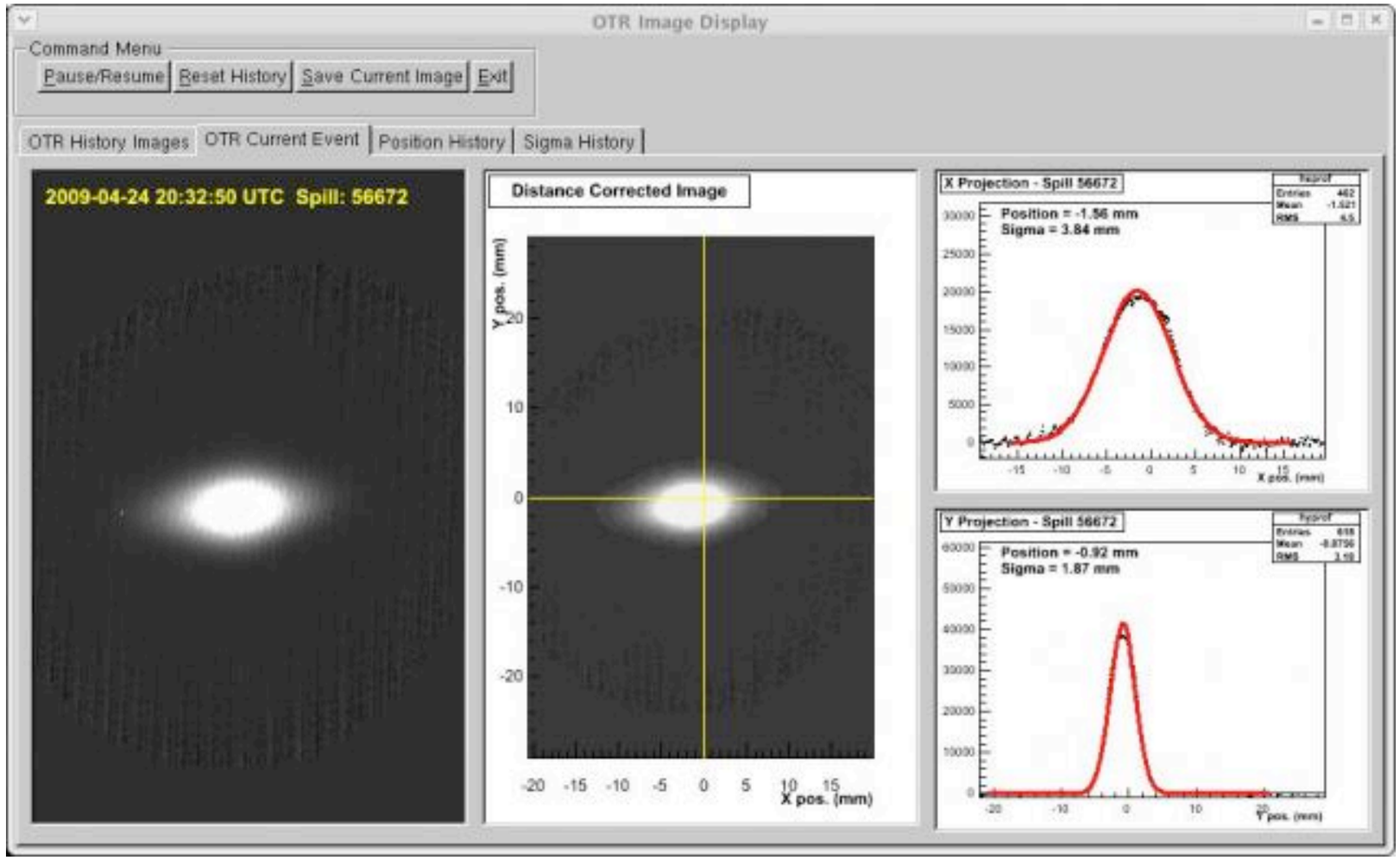
SSEM担当者の作品


File
Start
Stop
Close
Print
Histogram
Waveforms(SSEM19)
BeamProfiles
SSEM19
SSEM19X
SSEM19Y
SSEM
01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
x(center)=-23.11mm
x(width)=5.20mm
y(center)=-32.73mm
y(width)=2.64mm
Run Number: 1477
MIDAS Event Number: 40
GPS Event Number (Spill Number): 309415

MuMon担当者の作品



OTRはMIDAS clientで実装されたものを
山形がEventDistributorの子供として移植



MIDAS本体も

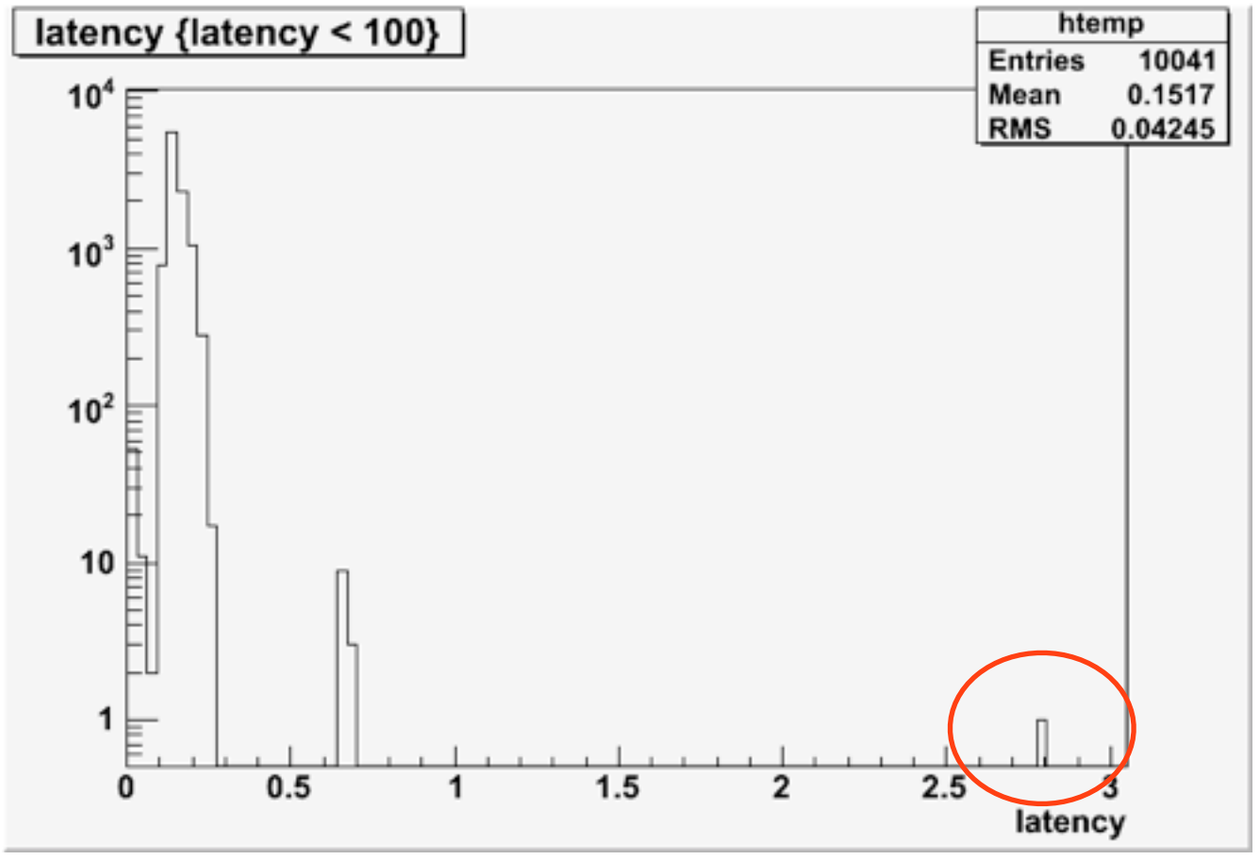
あちこちいじります

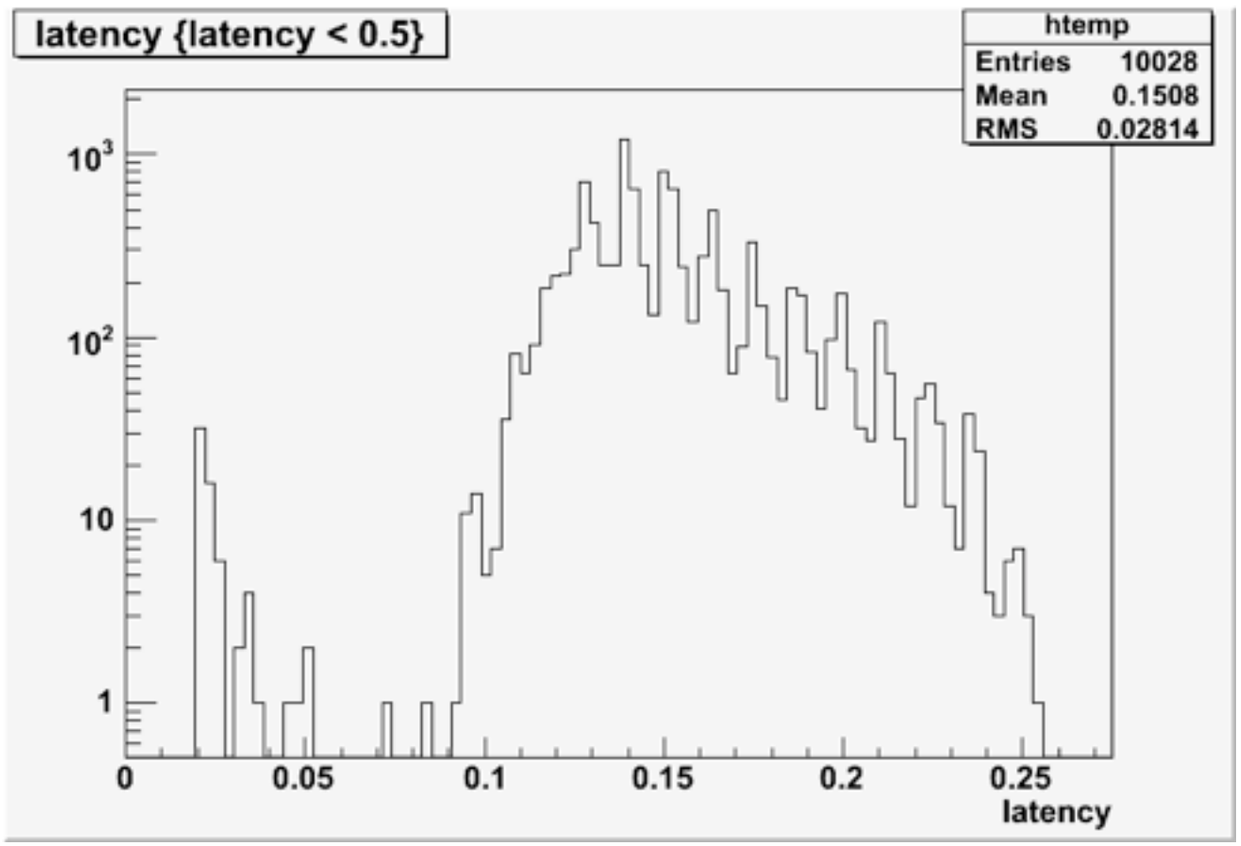
- 改造したもの
  - 全体にCPU負荷を削減するためのウェイトが多いので、ほとんど削除してレイテンシを短くする(busy loopっぽい動作になる)
  - mevb (イベントビルダ)
    - 「このデータソースはいまはずれてるから気にするな」機能を追加
  - mlogger (データロガー)
    - TTreeフォーマット記録時のバグとり
      - ファイルの切替が起きると死ぬなど山積
- 自作
  - COPPER専用サブイベントビルダ
    - データ源の数が多いとmevbでレイテンシが劇的に悪化
      - E391aでの情報により事前に察知
  - Event Distributor
    - TTreeの1Entry分のデータをTObject+TCollectionで配布
    - 各種モニターはオフラインと同じ解析コードが使用できる
    - MIDASの定石に従うとEvent Bufferの排他制御のため、処理速度が悪化する
      - E391aでの情報により事前に察知

イベントビルダ

- MIDASには最初からイベントビルダ(mevb)がついている
  - Event Bufferを監視し、全てのFrontendからデータが到着したらそれをくっつけて別のEvent Bufferに書き込む
- T2Kでは全部で50個くらいのデータ源がある
  - COPPERが40個くらい
  - COPPER上ではLinuxが走るので、直接MIDASの「frontend computer」にもなれる
  - しかし、Frontend computer1個につき1個のプロセスが出現する
  - 50個もプロセスがあるとプロセス切替の時間が無視できないだろう

20 frontend の時、トリガー発生からビルド完了までの時間

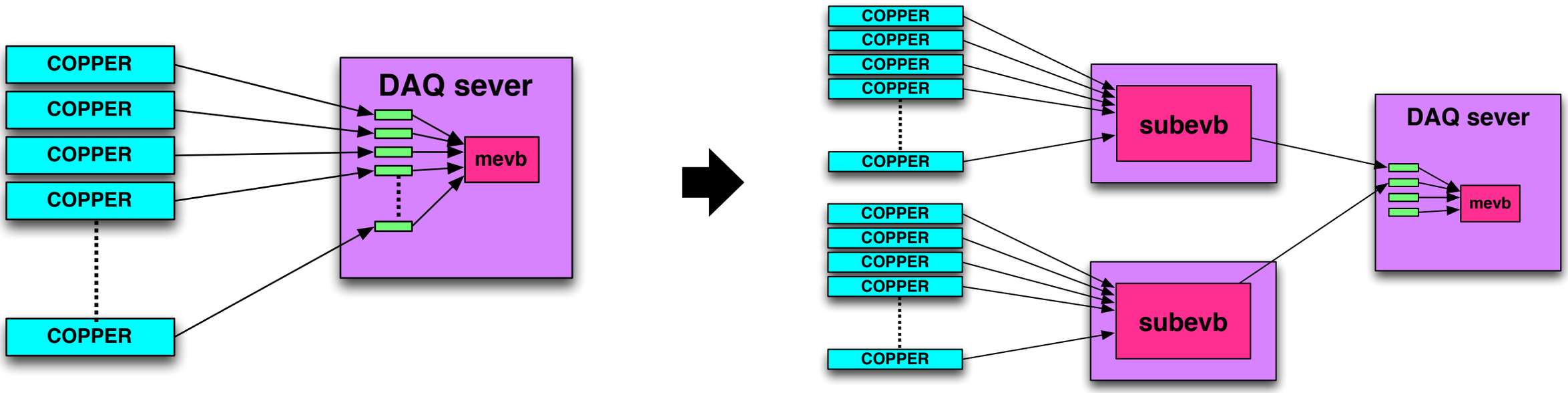




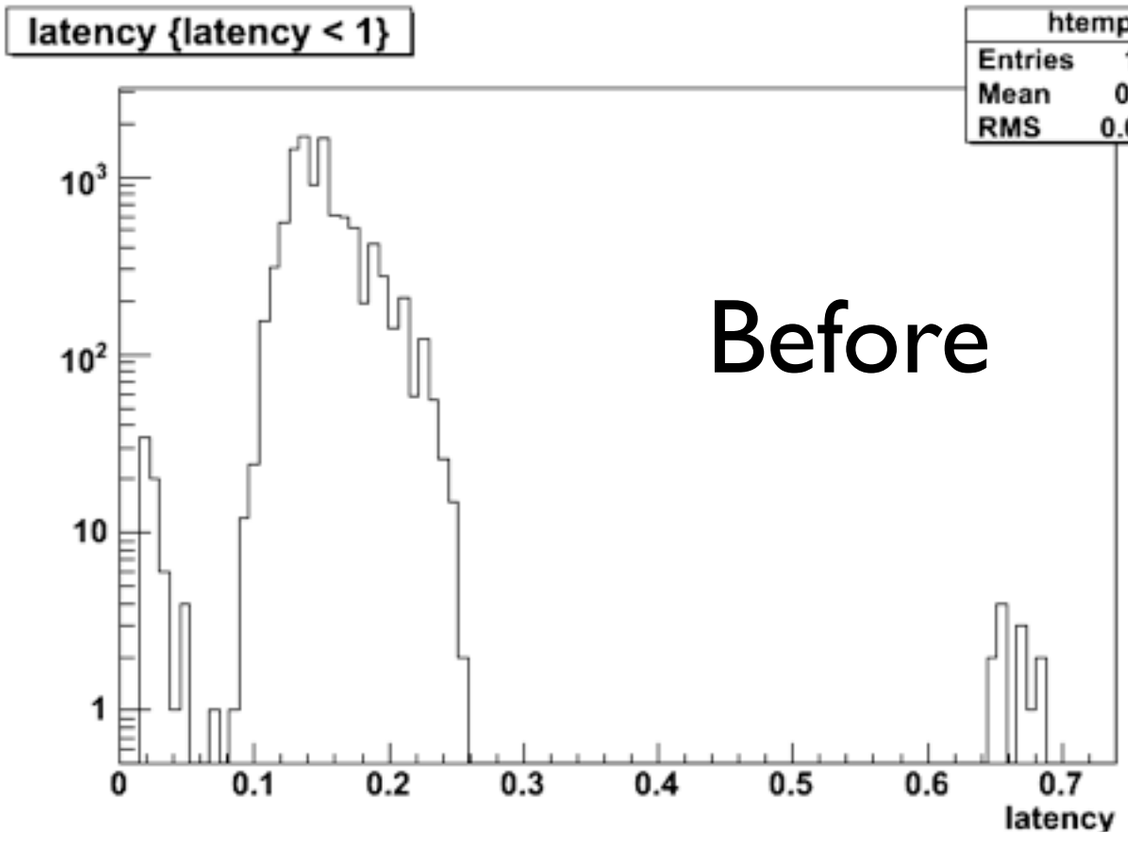
20でこれなら40はアウト

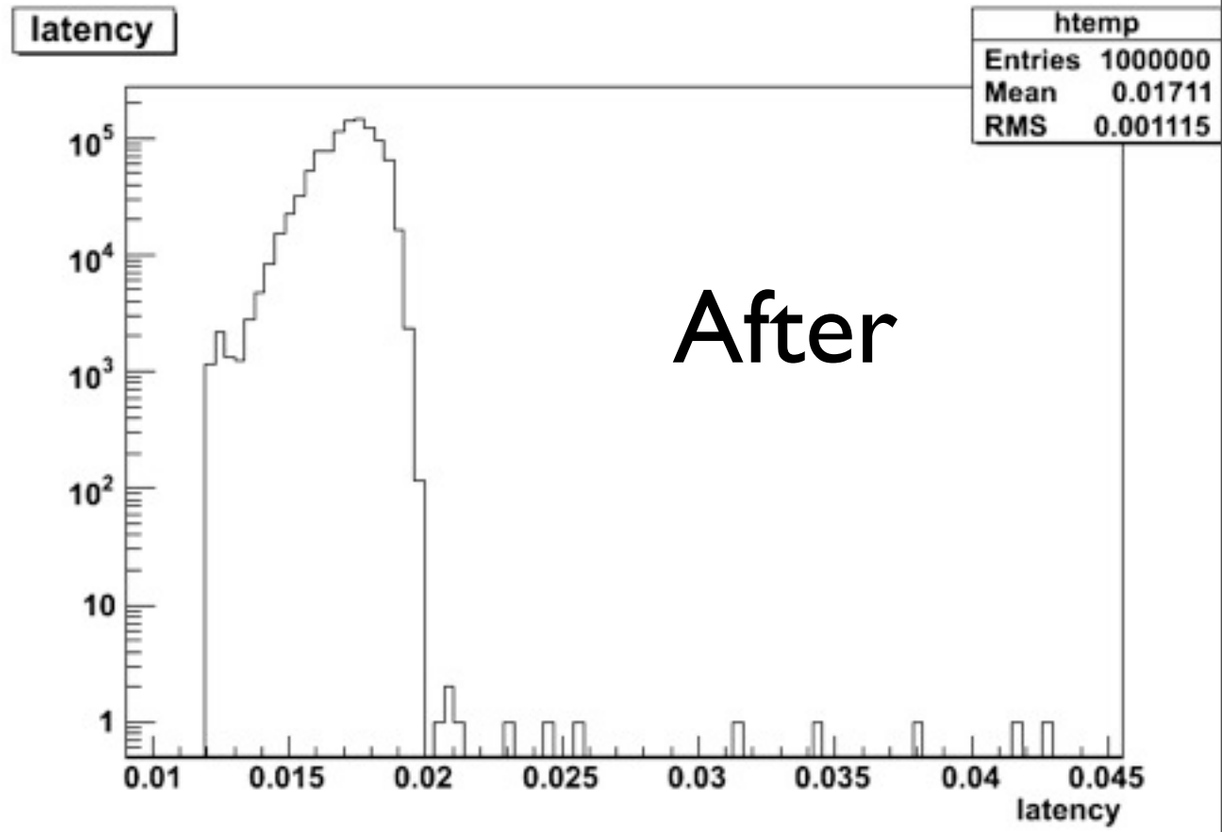
サブイベントビルダ

- DAQサーバが走るPCでプロセスを増やさなければよい
- 各建物のCOPPERからの出力をPC1個で固めて出力し、一つのfrontend PCとして振る舞えばよいのでは?
  - サブイベントビルダを用意する
  - COPPERからサブイベントビルダの間は独自プロトコルで垂れ流し
  - レイテンシ10ms以下



180ms -> 18ms


latency {latency < 1}
htemp
Entries 10040
Mean 0.1514
RMS 0.03325
10^3
10^2
10
1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
latency
Before


latency
htemp
Entries 1000000
Mean 0.01711
RMS 0.001115
10^5
10^4
10^3
10^2
10
1
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
latency
After

些細な問題の例

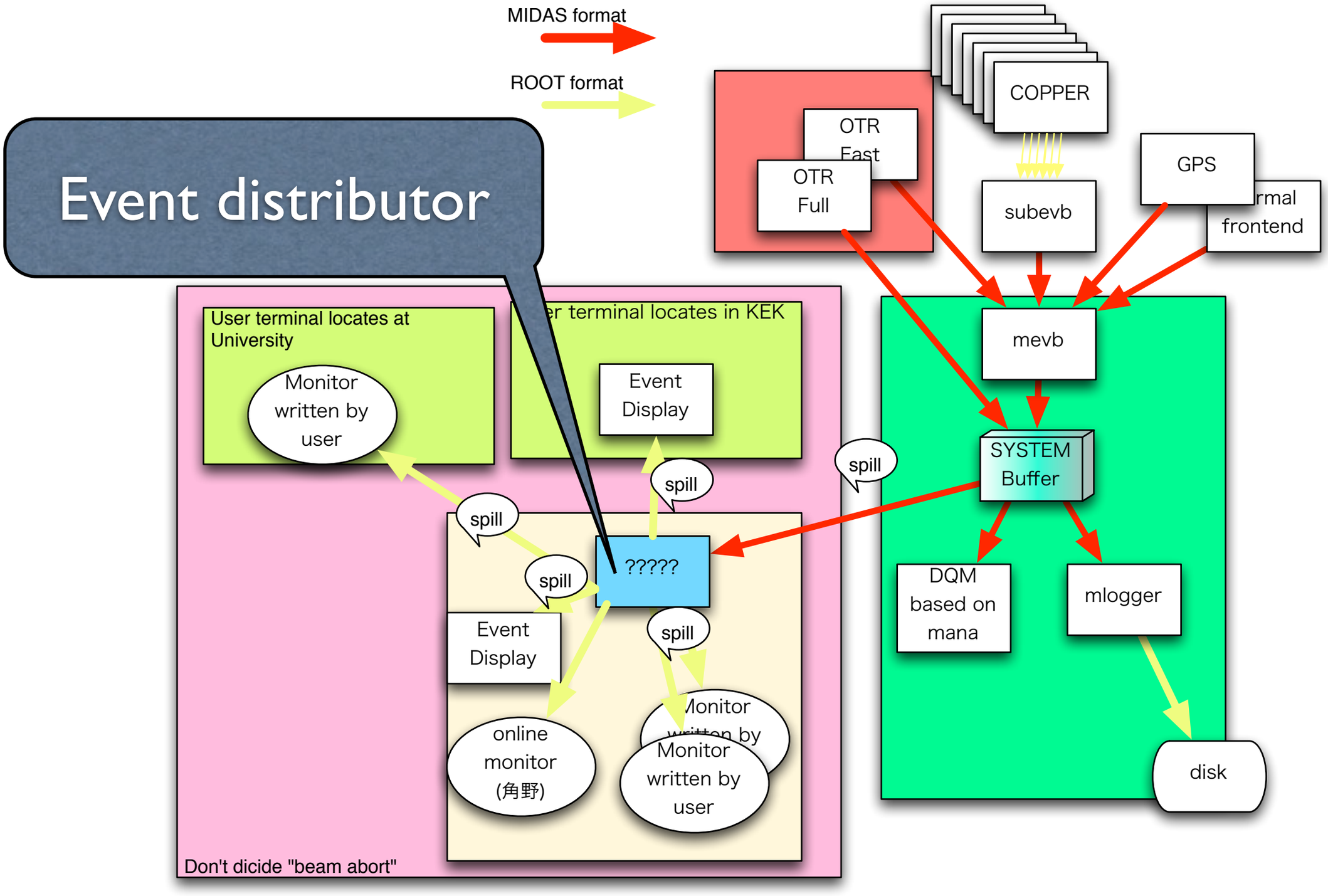
- COPPER 30枚から1個のPCにPlanex経由で送るとときどきパケットロスしてレイテンシ悪化
- QoSでCOPPER1個の速度を60Mbpsくらいに落とす

mlogger の改造

- MIDASの大半のプログラムはMIDASの独自フォーマットで記録する前提になっている
  - しかし解析プログラムは大半ROOTでつくられるだろう
  - だからTTreeで記録したいよね?
  - 説明を見ると一見可能
- が、問題山積
  - Branch構造をODBから拾おうとする
  - ODBにBranch構造を記録するためには全てのデータをODBに記録しなくてはならない→たいへん遅い
  - TTreeのファイルチェンジが発生すると死ぬ

Event Distributor

- 大量のオンラインモニタに対してデータを配布したい
  - T2Kは建物がいっぱいあるので人も分散するかも
- MIDASでの定石はEvent Bufferを監視し、データ到着後の表示処理が完了したら解放する
  - 処理が終わるまでバッファが解放されないので、時間がかかると危険
  - しかしROOTで処理したいとするとフォーマット変換が必要
  - プロセスの数が多いとDAQ本体にもダメージ
- そこで、
  - Event Bufferを監視し、1イベントをコピーしたら即座にバッファを解放
  - TTreeの1Entry分と同じ体裁でTCollectionにつめてTSocketで配る
    - 各自でフォーマット変換しなくてよい
  - 各クライアント毎にバッファを用意
    - 遅いモニタがいても速いモニタの邪魔をしない


Event distributor
MIDAS format
ROOT format
COPPER
OTR
Fast
OTR
Full
subevb
GPS
rmal
frontend
mevb
SYSTEM
Buffer
DQM
based on
mana
mlogger
disk
User terminal locates at
University
er terminal locates in KEK
Monitor
written by
user
Event
Display
spill
spill
spill
spill
spill
?????
Event
Display
online
monitor
(角野)
Monitor
written by
user
Don't dicide "beam abort"

History

- Mar 2007: Scout + Decision to use MIDAS
- Apr 2007: Latency test with COPPER +mevb
- May 2007: sub event builder development
- Sep 2007: design of event distributor
- Oct 2007: readout program is exported and tested by MuSR experiment at RAL

History

- Jan 2008: Whole system test at KEK
- Apr 2008: development of distributor
- Jun 2008: rehouse from KEK to J-PARC
- Jul 2008: μ chamber beam test at Kyoto
- Aug 2008: UW-FADC integration
- Nov 2008: GPS integration

History

- Dec 2, 2008: NU1+NU2+D3 test
- Dec 12, 2008: NU1+NU2+D3+TS test
- Dec 15, 2008: OTR integration
- Dec 23, 2008: MR 30GeV extraction test
- Feb 2009: MuHut integration
- Apr 4, 2009: Final dry run
- Apr 23, 2009: First shot

でたー

プログラム書きについて

- バージョン管理重要
  - HgとかGitでマルチマスタすべき
  - 使用者からのコミットも受け取った方がいいと思いますよ

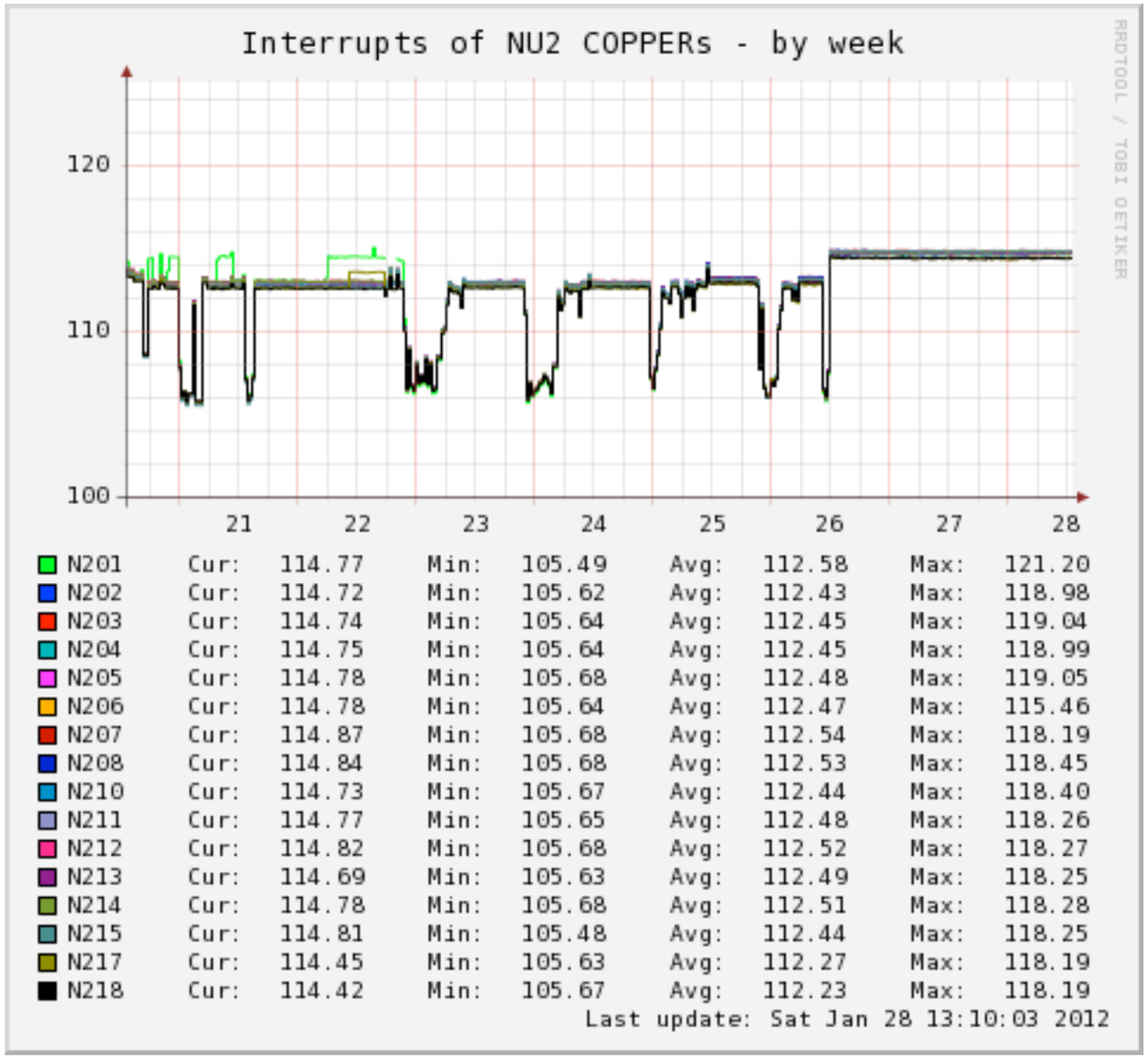
私は何をやっていたか

- ひたすらレイテンシを下げる
  - 最悪値を短くする
- スループットの糊代を削ってビジーループにすげ替える
  - やり過ぎると逆にレイテンシ長くなるので注意
- その程度の調整では無理であればフロントエンドの構造も変える

さらなる向上への障害

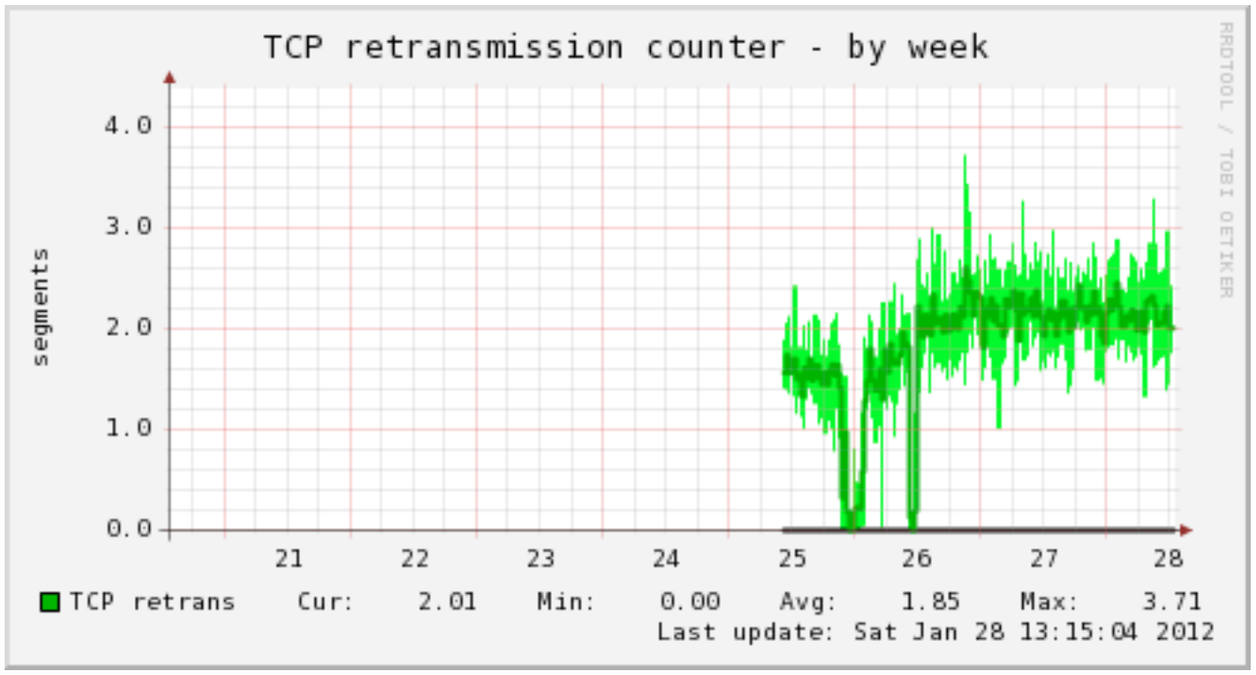
- ESM用ADCにはDMAがないため、遅い(100ms以上かかる)
- Planexのスイッチが途中から特定のポートだけときどきパケットロスしはじめることがあった
  →安定運用に問題
- 途中でNICが「中途半端に」壊れる→安定運用に問題

とりあえずモニター



COPPERの割り込みレート

とりあえずモニター



TCPの再送カウンタ
(とりわけ多い人の例)

総評

- MIDASすげー 超すげー
  - わりと必要なものはなんでもあります(趣味に合うかは別)
  - 地雷をE391aで洗ってあったから
    - まあ改造するけどな
    - GUI/CUIを自分で作らなくていいのはかなり楽
- 客に自分をあわせた方が楽じゃない?
  - 人を説得するのと、プログラムを改造するのとどっちが楽か
  - 説得が得意ならそっちでももちろんよし

ただし

- 最初から用意されているグラフ化機能がプア
  - ODBに値が格納できないものはグラフ化できません
  - つまり1次元グラフしかつくれない
    - そうかといってODBに大量にものを書くと遅くなる
- データレートが高いとアウト
- 読み出し機器の数が多いとアウト
  - frontend PCとDAQ PCが相互に通信が出来ないとだめ
- Belleには使えません