2025/11/18
Shunichi Kashima
RCNP, U-Osaka

計測システム研究会2025

KEK 東海キャンパス 1号館

発表内容

- J-PARC チャームバリオン分光実験(E50)
- DAQの健全性とボトルネック
- DAQソフトウェアのパフォーマンス評価
- 性能評価の方法・用いる指標
- 結果と考察
- 試行錯誤
- まとめ

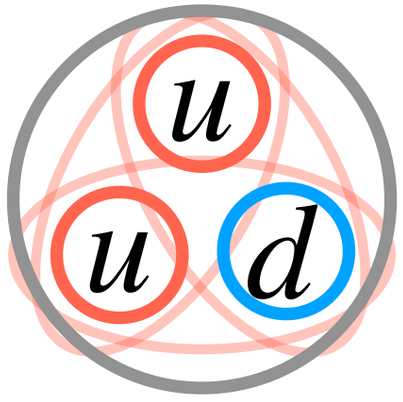
研究の背景

バリオンの内部構造を知る手掛かり

クォークが「強い力」によって閉じ込められた複合粒子

- **クォークからハドロンがどのように形成されたか**
- **励起状態**の調査が重要：ダイクォーク相関が1つのカギ

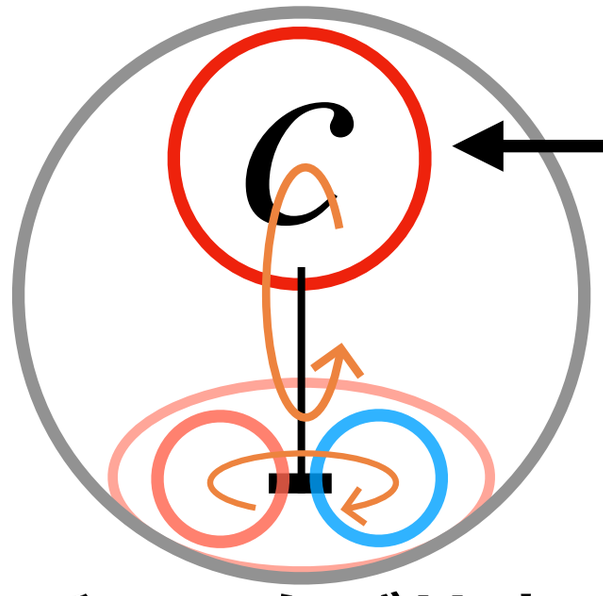
バリオン(クォーク3つ)の内部状態



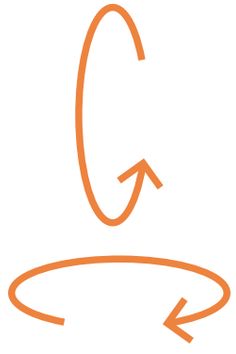
陽子

クォーク間の相関が縮退

クォークを1つ重くすると…



チャームバリオン



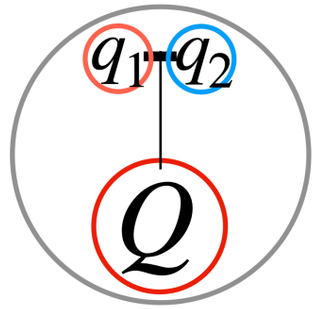
重いクォーク, 軽いクォーク対の間の運動

軽いクォーク間の運動

⇒**励起準位構造、生成率、崩壊分岐比**

研究の背景

バリオンの内部構造を知る手掛かり

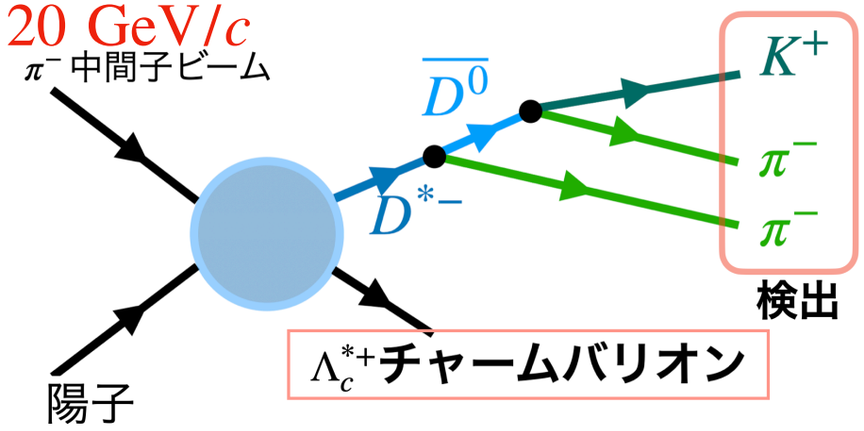


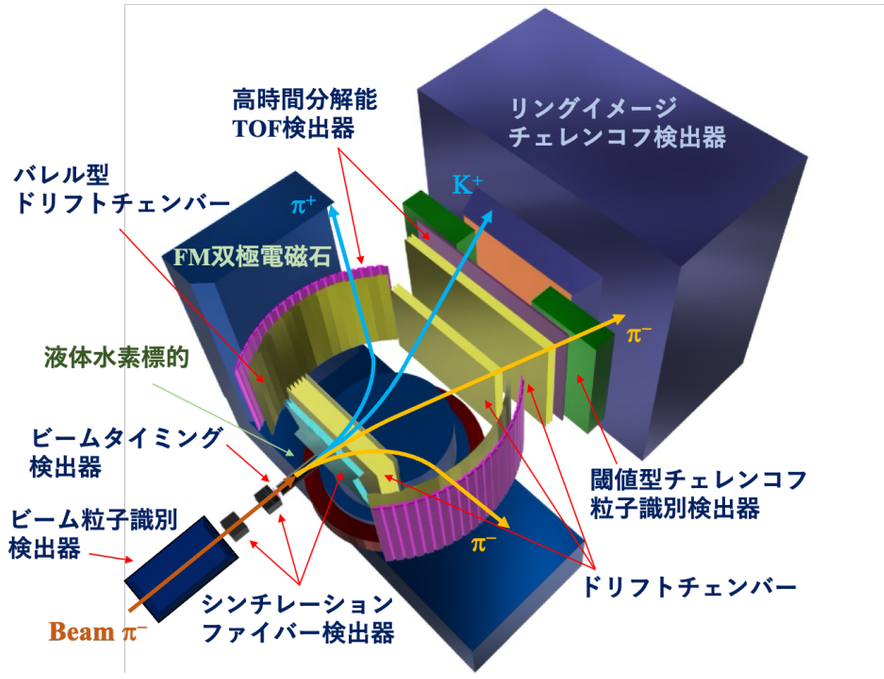
- チャームクォークを含むバリオン：チャームバリオン
- 励起準位構造, 励起状態の生成率, 崩壊分岐比

⇒チャームバリオンの分光実験(E50) @ J-PARC

実験手法

$$\pi^- + p \to D^{*-} + \text{チャームバリオン}$$





- チャームバリオン生成：数nb ほどの反応断面積
- **大強度ビーム(=高反応レート)**
- 複雑なハードウェアトリガー ⇒ **ソフトウェアでやる**

連続読み出し式データ収集システム

連続読み出し式DAQシステム・NestDAQ

連続読み出し・ソフトウェアで事象選別

- **信号データを全て**コンピュータへ送る
- **ビームが標的と反応せずに通過する事象(ビームスルー事象)**(95%)
- **チャームバリオン生成**~ 1-10 nb $\ll \pi^- + p$ 反応 ~ 25 mb
- 興味の薄いイベントが多すぎる
- **チャームバリオン**生成事象の候補を漏らさず**リアルタイム**で事象選別

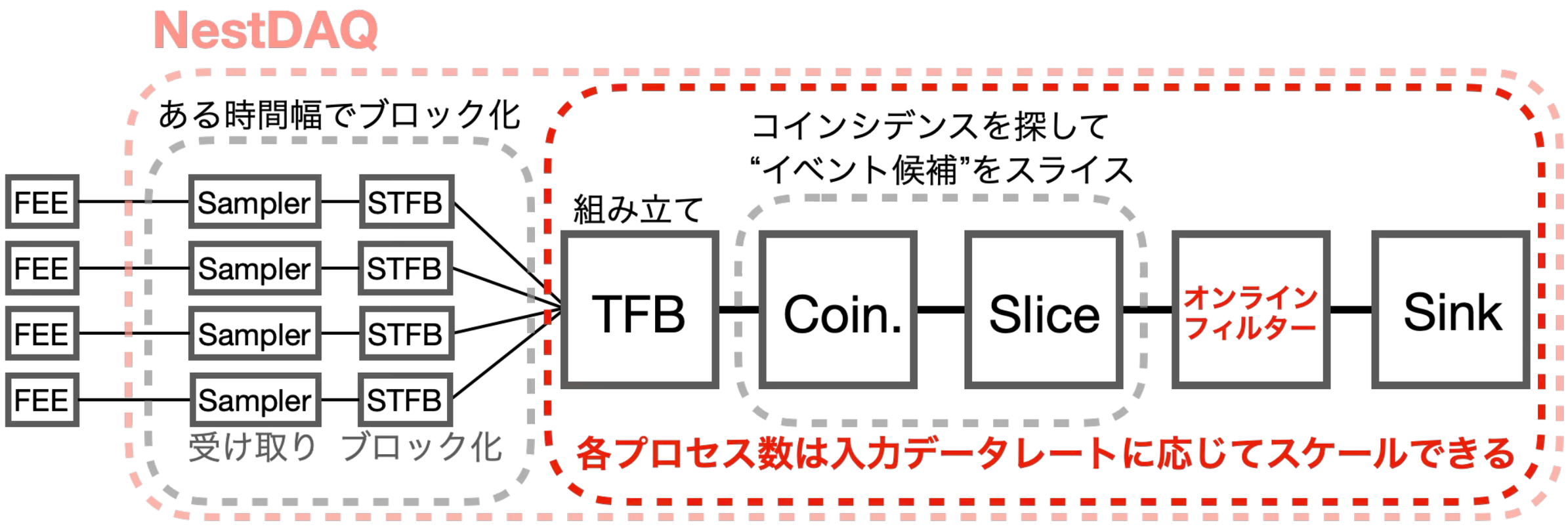
ソフトウェアで

当面の目標

- タイミングカウンターを用いたTOF条件のフィルター
- ビームスルー事象の排除

⇨ **オンラインTOFフィルター**の性能評価

NestDAQ - 分散型データ収集ソフトウェア



スケーラビリティ

- プロセスを増やせば負荷が分散(=システム全体のスループットが増える)
- 負荷に応じた計算資源
- **E50実験ではどれくらいの計算資源が必要か**
- DAQ動作試験のテストベンチのデータを用いてTOFフィルター1プロセスあたりの性能評価を行う

  NestDAQデータリプレイヤーを使いました

性能評価の仕方 — DAQの健全性

DAQの動作限界を決める要素

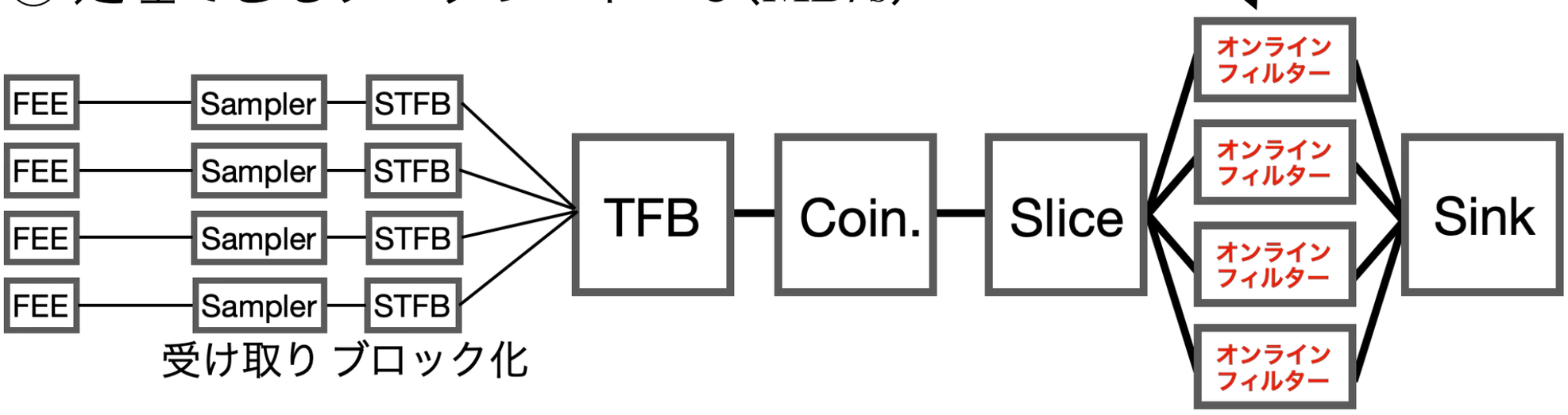
- (①)一定時間で入ってくるデータサイズ < (②)処理できるデータサイズ
  - 少しでも②よりも①が大きければDAQはいずれ止まる
  - 以下では単位時間あたりの量
    - ① 処理すべきデータレート(MB/s)
    - ② 処理できるデータレート(MB/s)

足りなければプロセス増やせば良い

$$\text{必要プロセス数} = \frac{\text{処理すべきデータレート}}{\text{処理できるデータレート}}$$ ⇐ 1プロセスで測定

- ① 処理すべきデータレート = 10 (MB/s)
- ② 処理できるデータレート = 3 (MB/s)

3.3個



TOFフィルターの性能評価 — 測定対象

- 連続読み出し式DAQのテストベンチ(T103)で実装されたTOFフィルターの性能評価 ⇒ 処理にかかった時間(elapsed time)の計測

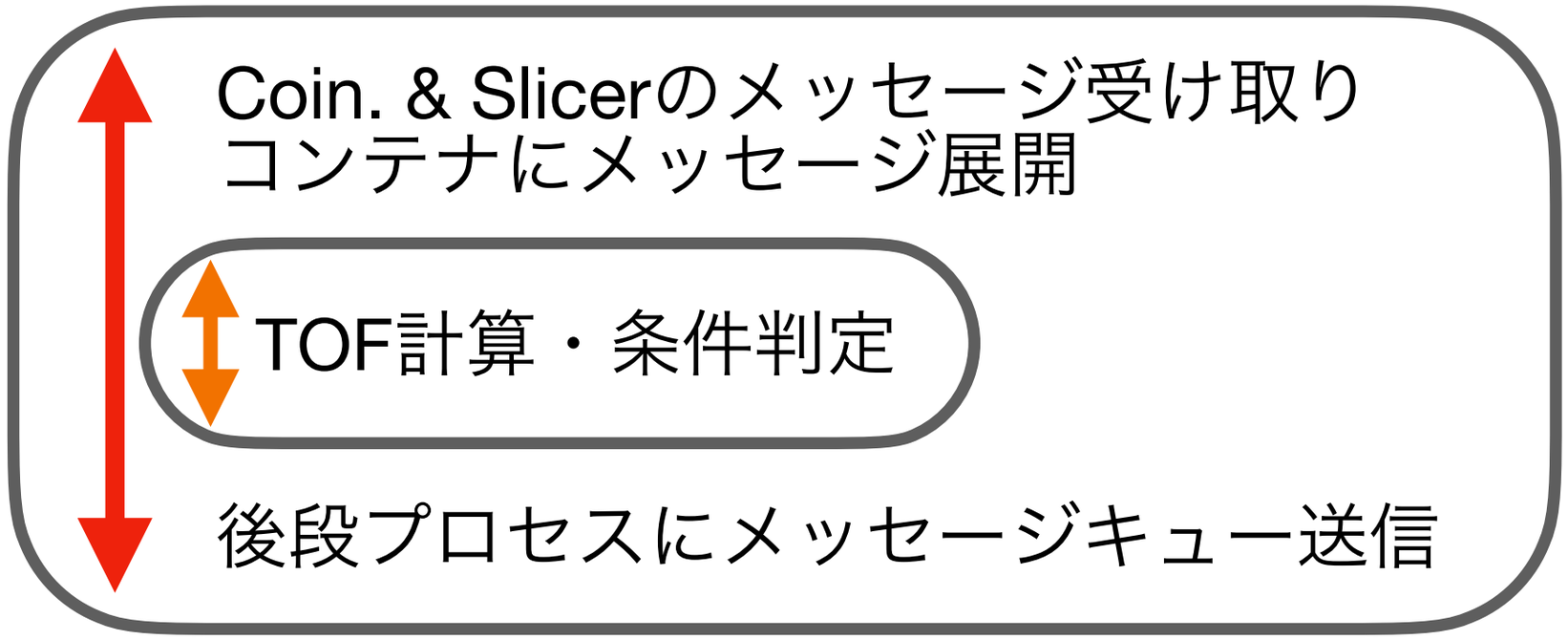
タイマー(1)

純粋なTOF判定時間

タイマー(2)

全体の処理にかかる時間

TOFフィルター処理内容



- 処理したデータ [byte] / タイマー(1) のなどの指標を用いて評価

計測環境

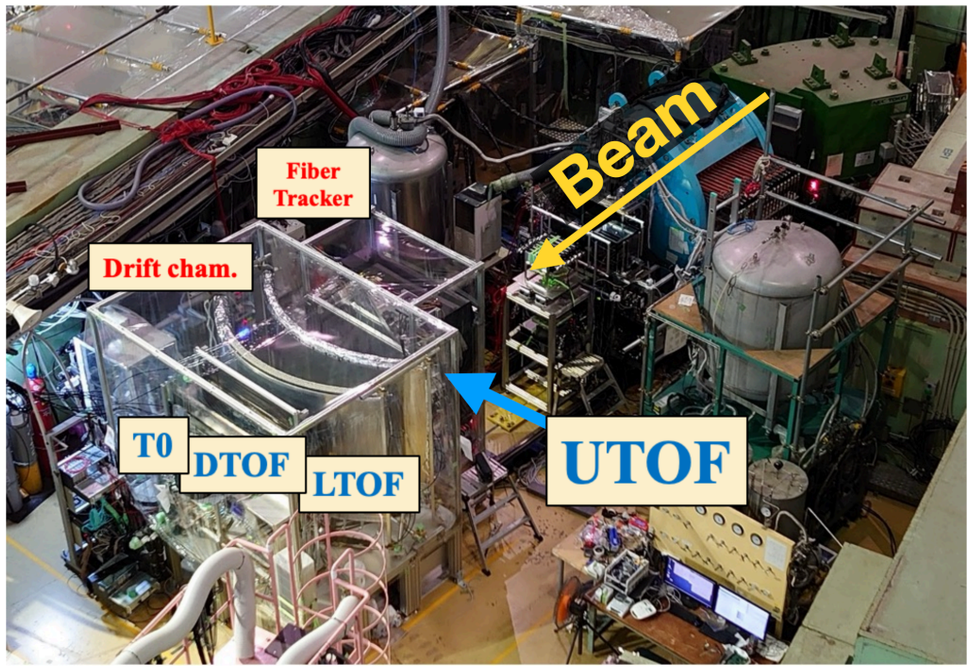
計算機のスペック

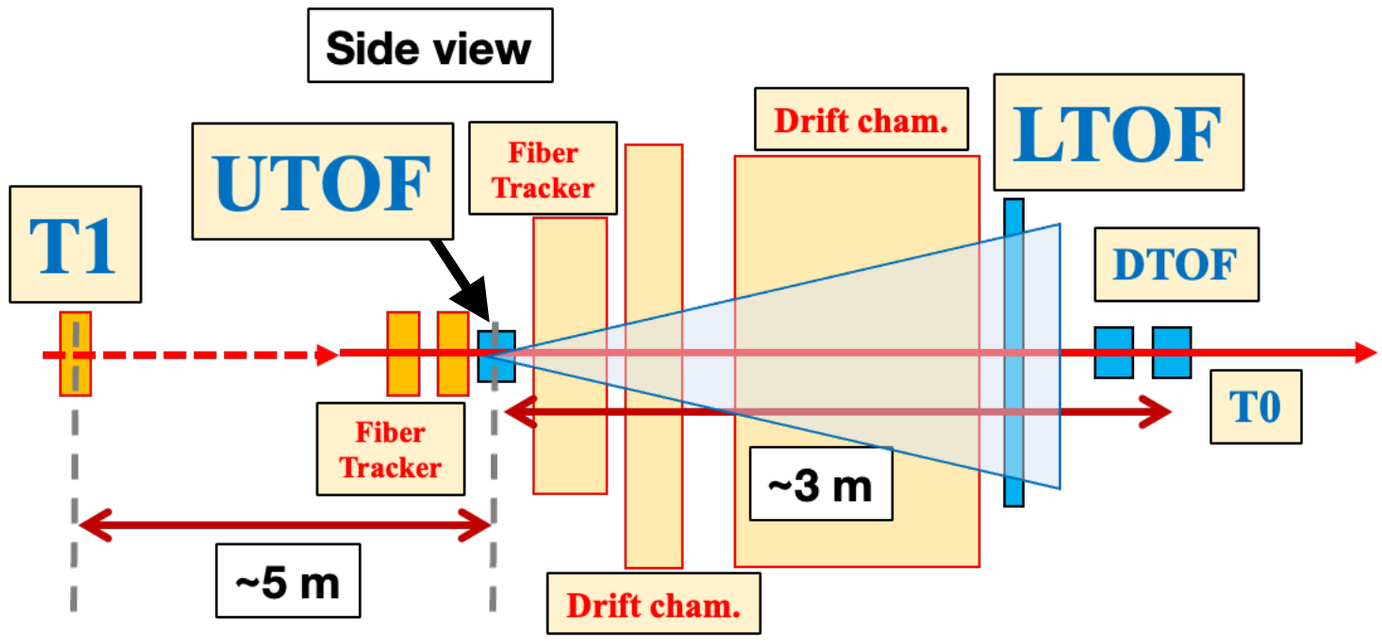
CPU	AMD EPYC 74F3
#Cores	24 コア / 48 スレッド
メモリ	64 GB

テストベンチのセットアップ

T103テストベンチ @ J-PARC K1.8BR

- 連続読み出し式DAQのテストベンチ(2024年4月)
- **タイミングカウンター**と**トラッカー**



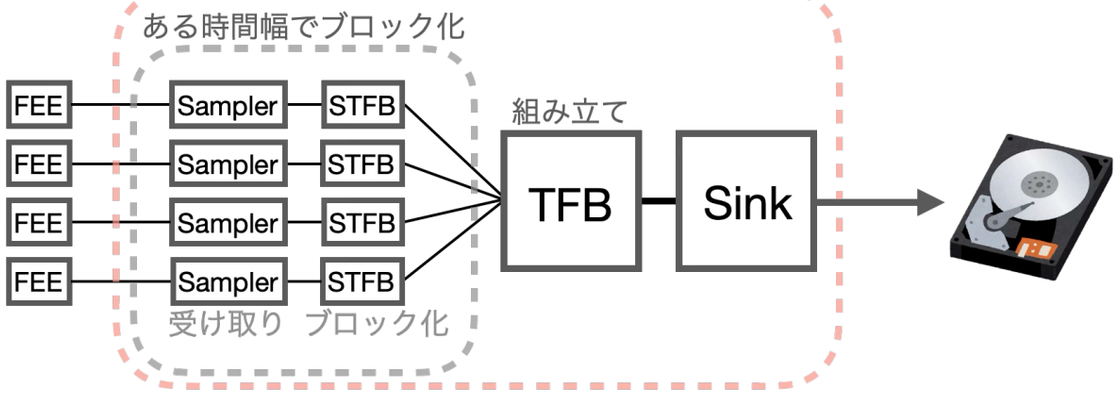


- コインシデンスロジック(Coin.) : **UTOF**と**LTOF**のAND
  - コインシデンス幅は120 ns

  ↓ Coin.から±1000 ns のウィンドウでスライス(Slice)
- **T1**と**UTOF**のTDC値の差がある条件を満たさない場合はSliceを棄てる

リプレイヤーによるオンラインフィルター性能評価

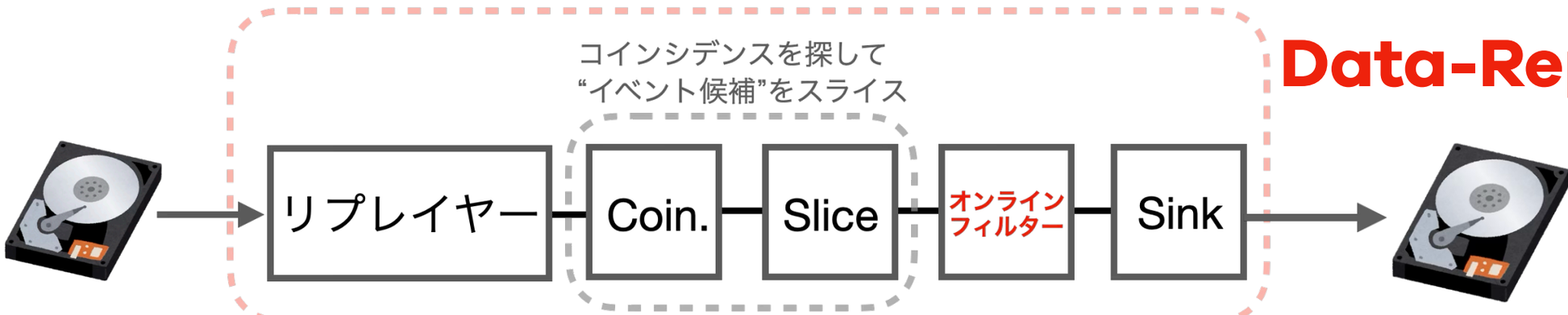
- **T103テストベンチで主に用いたトポロジー**



Trigger-less DAQ

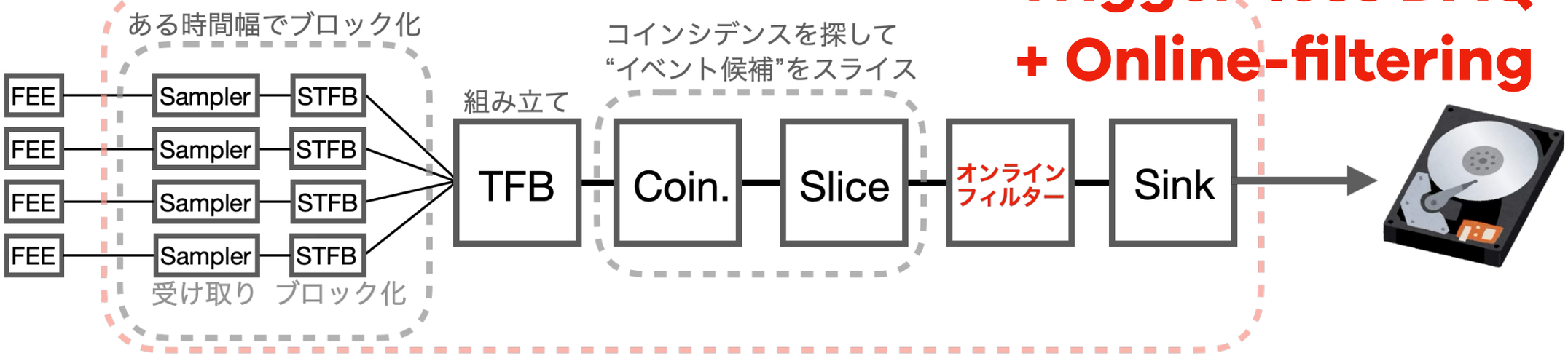
0% loss in Software

- **リプレイヤーによるベンチマークで用いたトポロジー**



Data-Replayer study

- **E50実験で用いる予定のトポロジー**

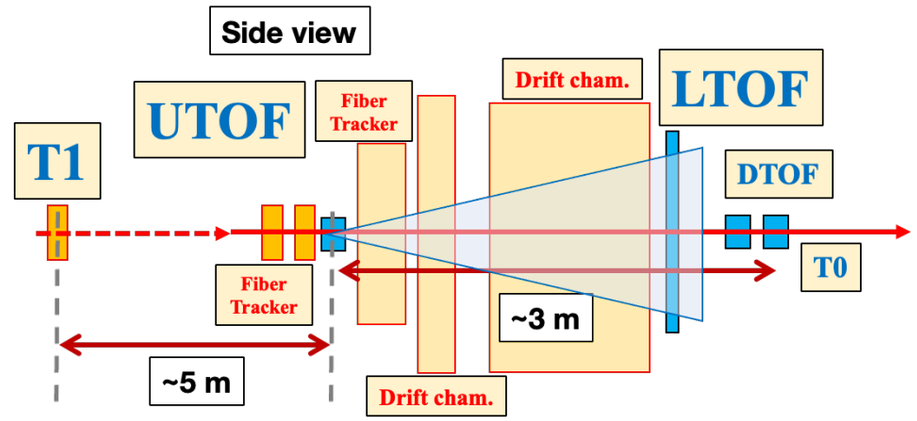


Trigger-less DAQ

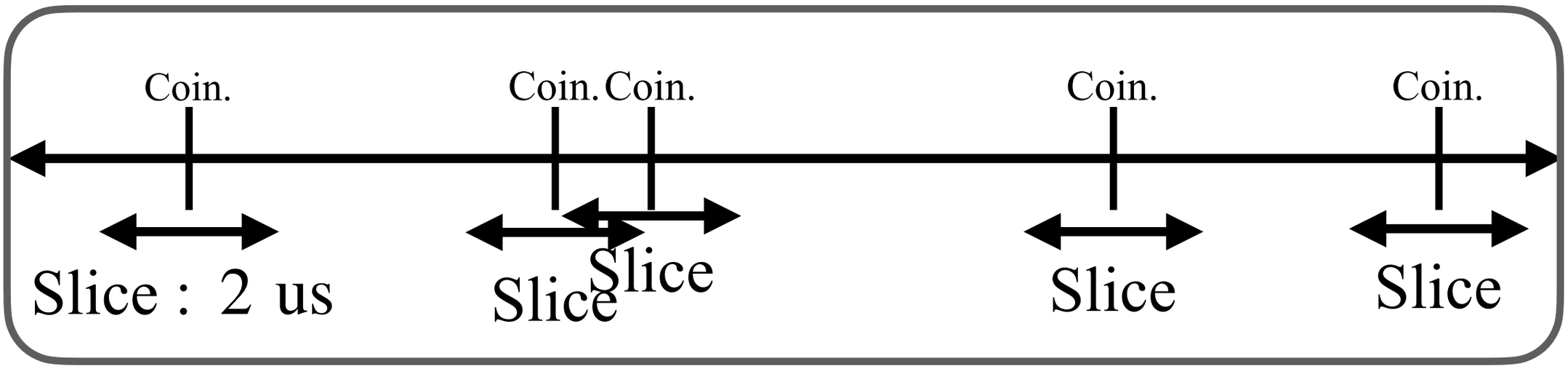
+ Online-filtering

TOFフィルターに送られてくるデータ構造

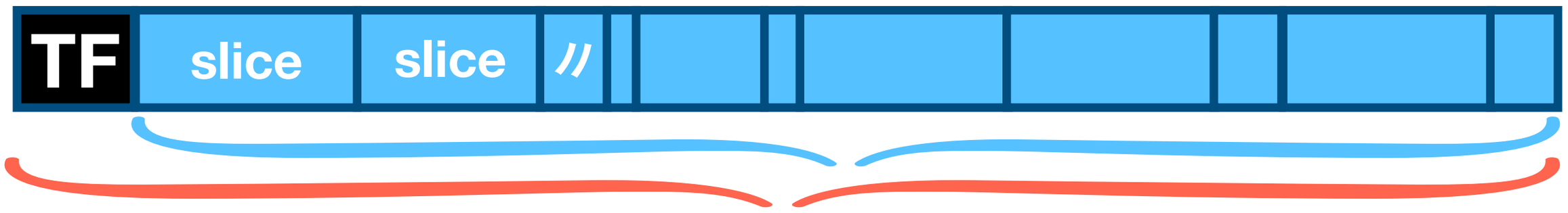
- 一つのTimeFrame(~ 2 ms)
- **コインシデンスの数**だけSliceが付属



TF : ~ 2 ms



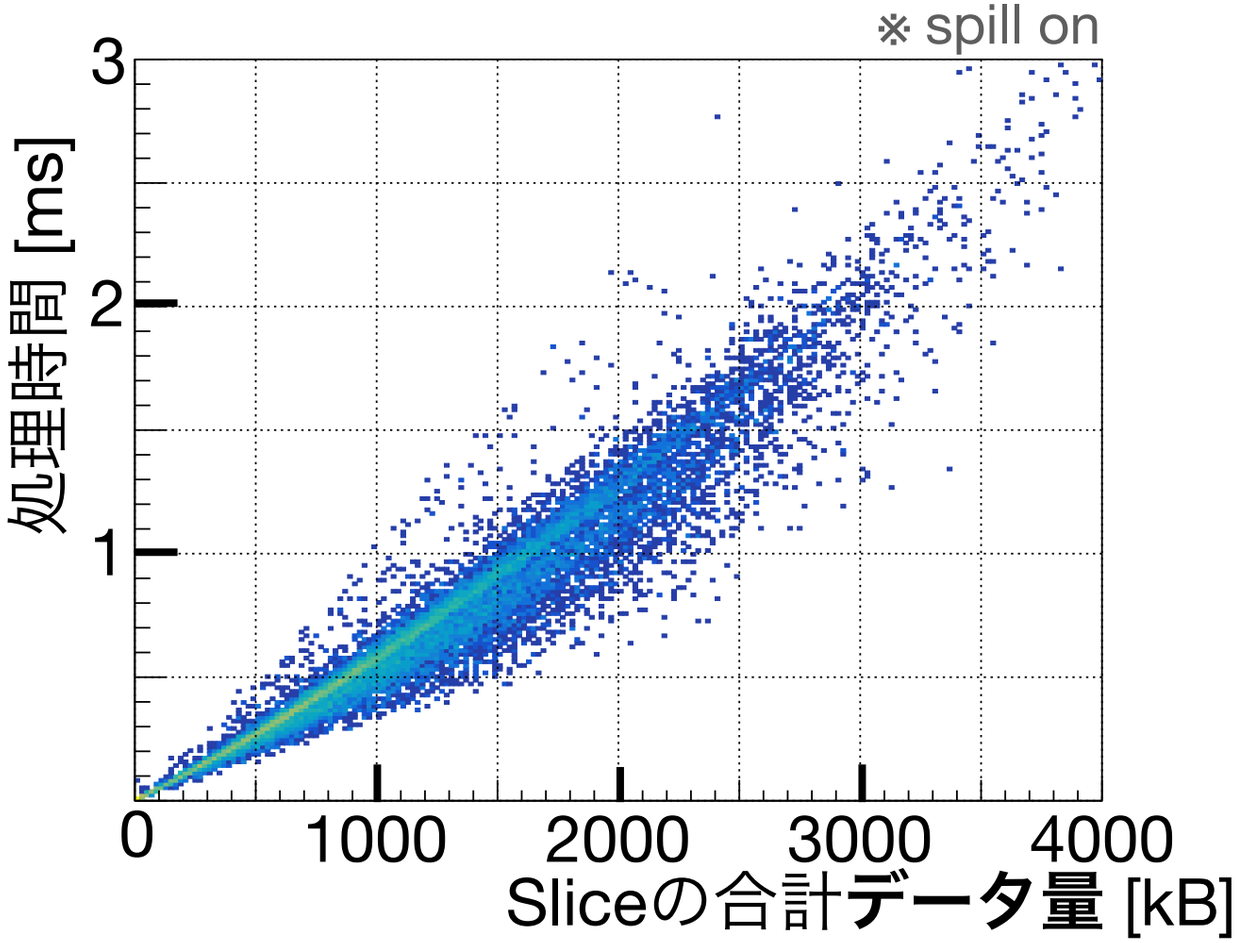
データ構造

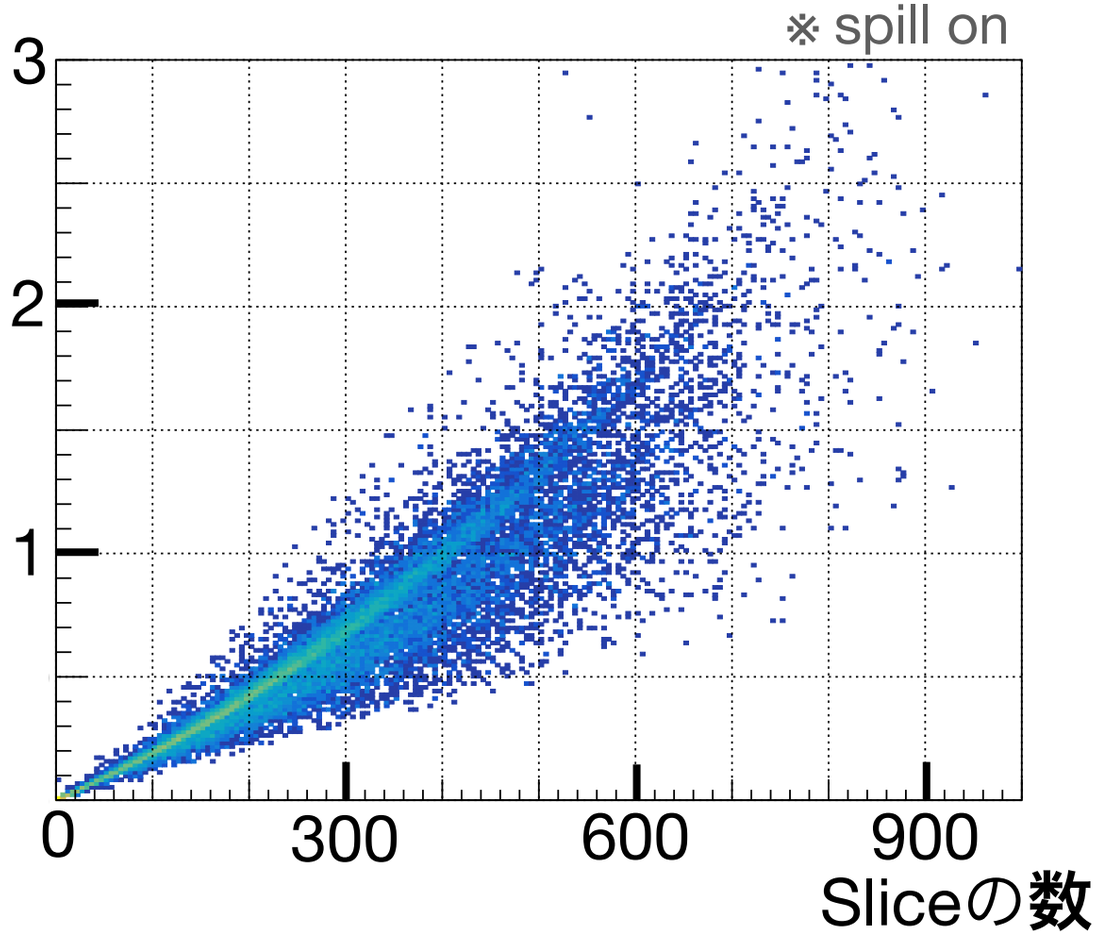


- **総データ量**、**Sliceのデータ量**：スループットの計算
- Sliceの数 = コインシデンスの数 ∝ **ビーム強度**

結果 - TOF判定の処理時間とビーム強度

- タイマー(1)でTOF計算・判定時間を評価



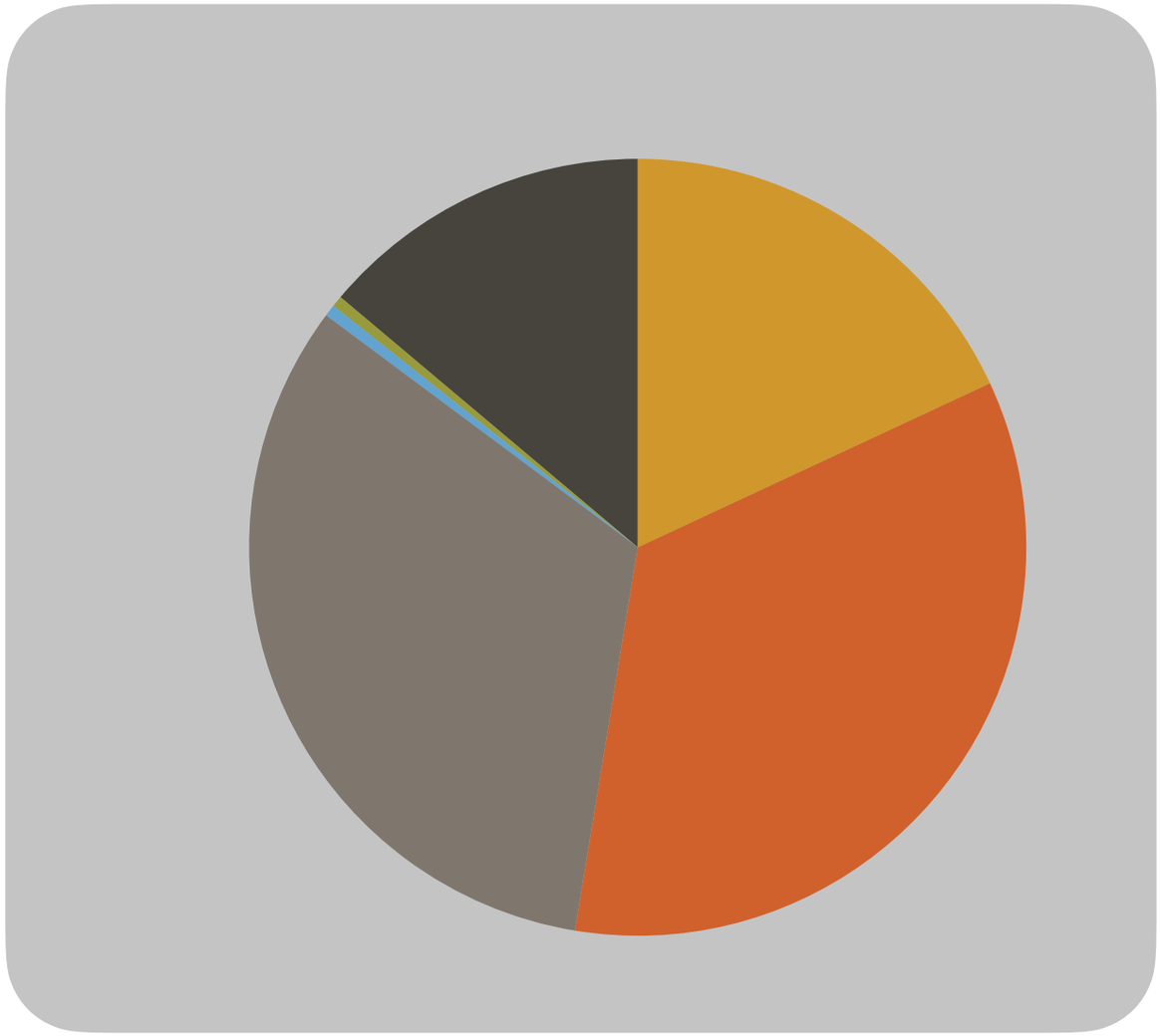


- データ量と処理時間の相関：時間はちゃんと測れてそう
- TOF計算の**組み合わせ**が2次的関数的に増えることを反映した増加傾向
  - T1ヒット数xUTOFヒット数で増えていく
- もっとビーム強度が高いときの処理時間は今後評価が必要
- GPUによる並列TOF計算の活用も視野に

結果 - フィルター全体とTOF判定のスループット

TOF判定処理にはまだ余裕がある

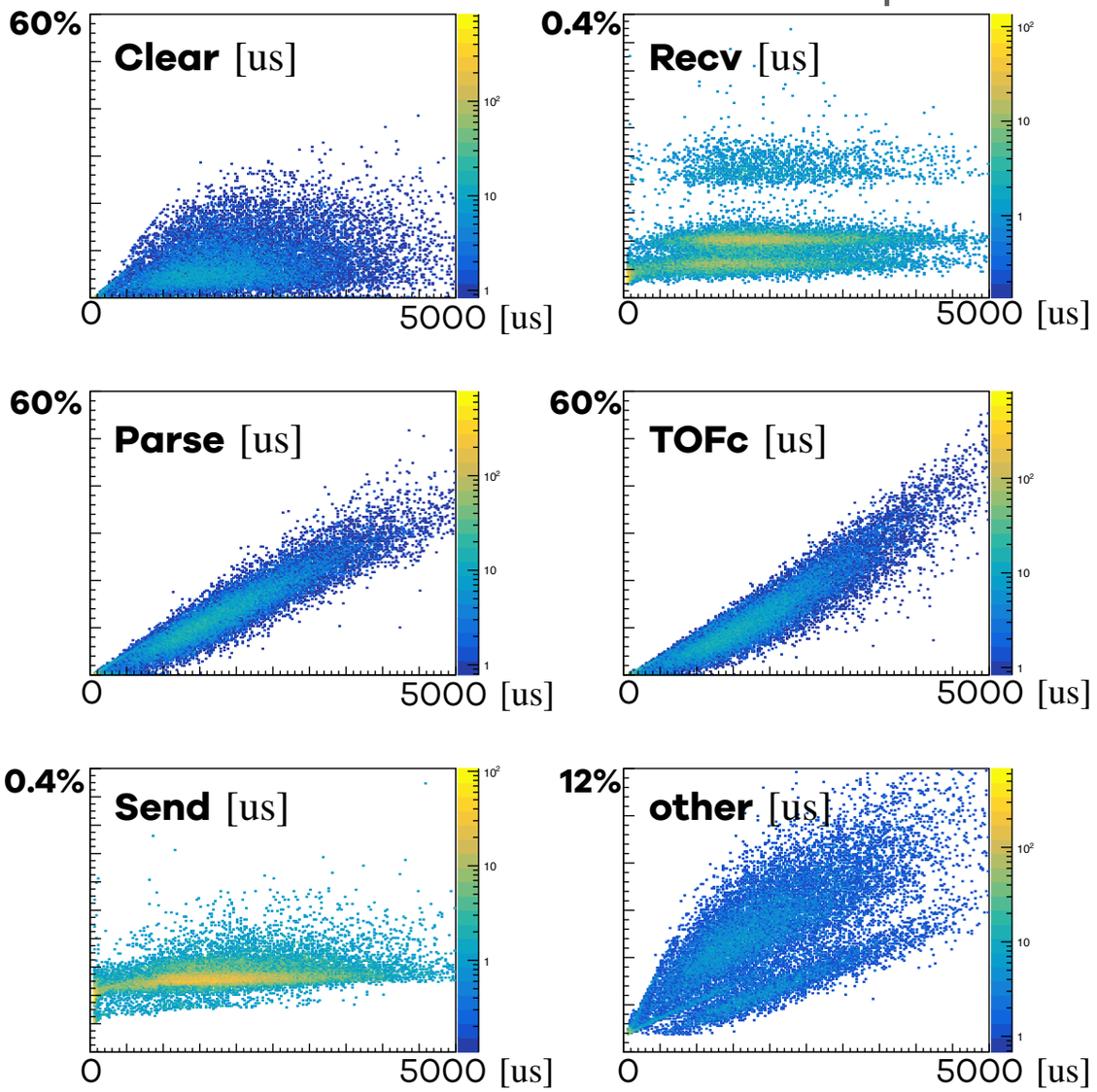
- **TOF calc**以外はオンラインフィルターの動作に必要な処理
- もっと高度な処理を行う余地あり
  - 複数カウンター間のTOFなど



※ spill on

全体の時間との相関

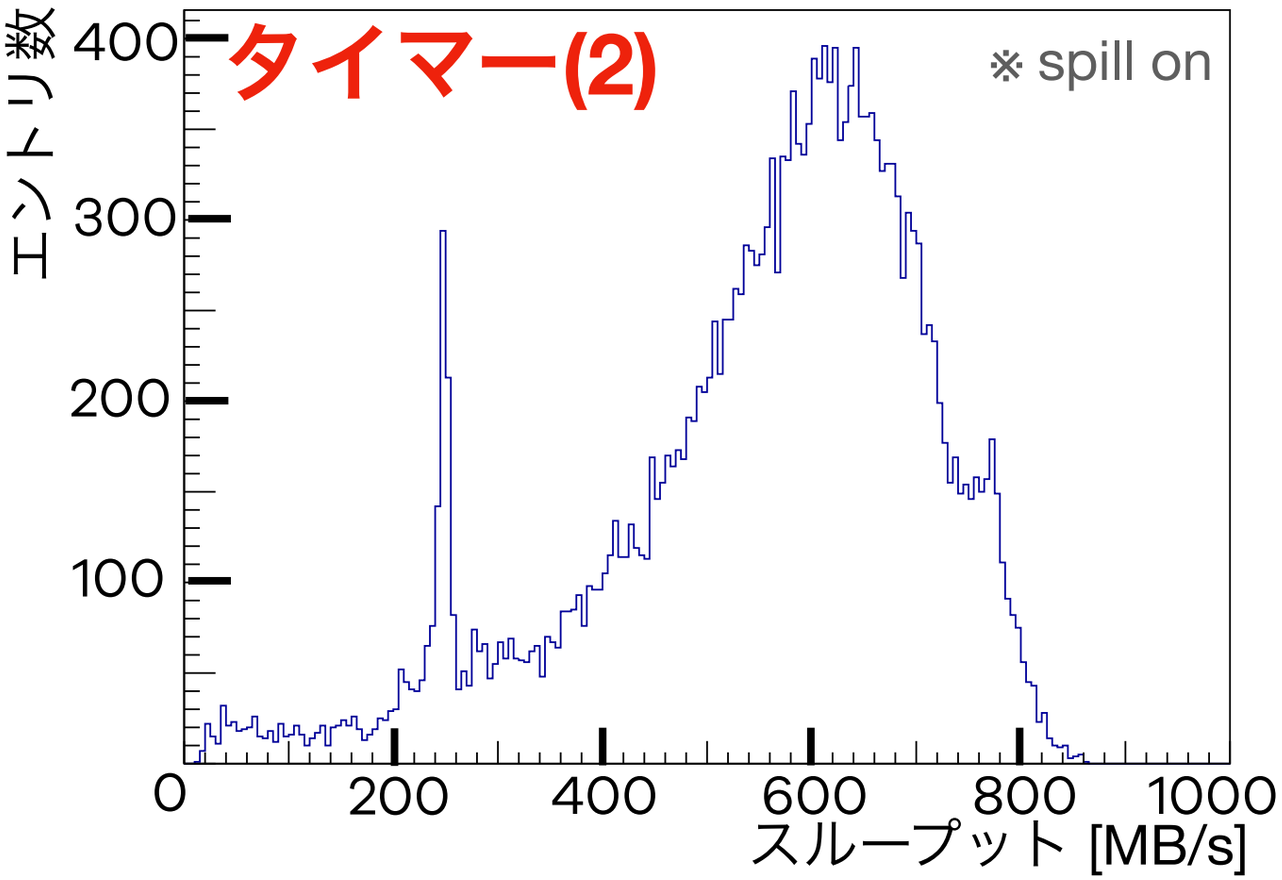
※ spill on



横軸：全体の時間 [us]

結果 - 処理できるデータレート

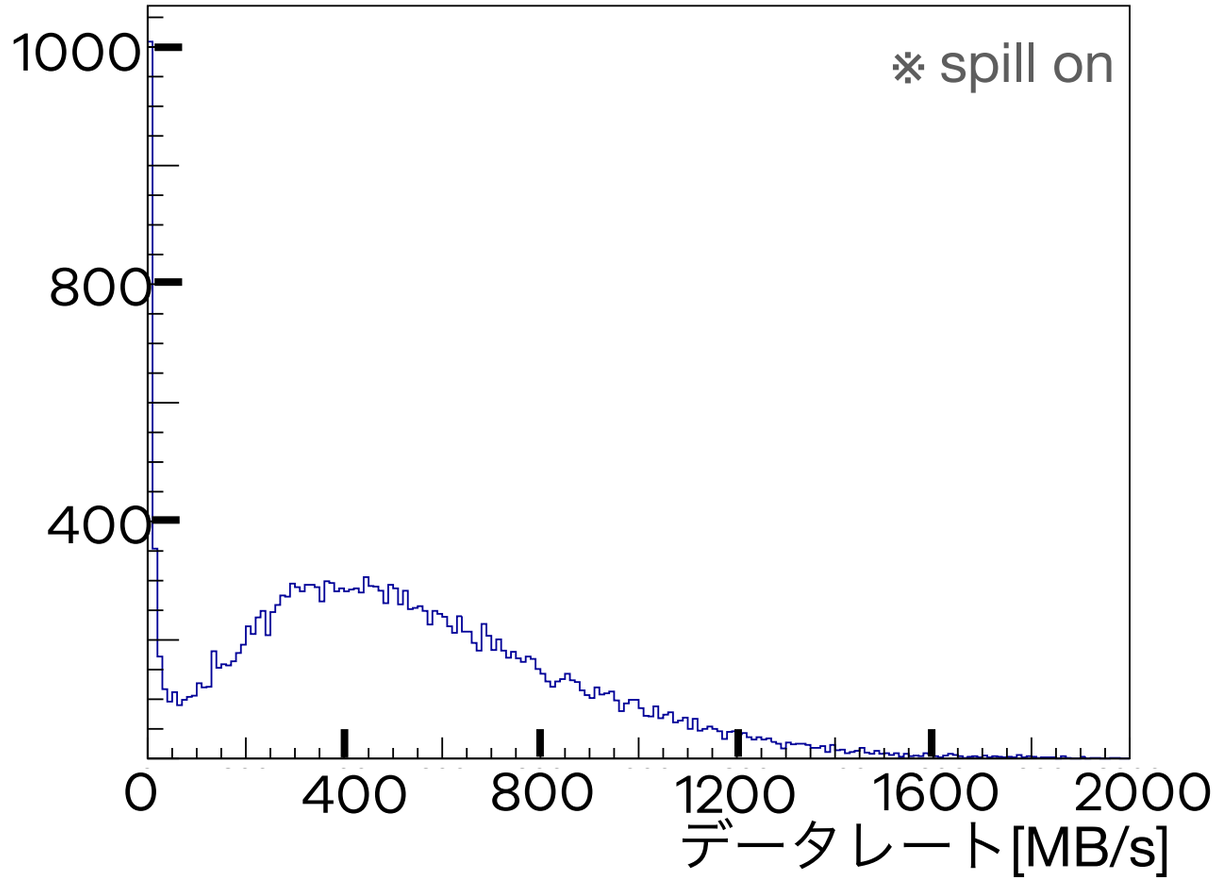
TOFフィルター全体のスループット



avg 594.9 MB/s

処理できるデータレート

オンサイトで送られてくるデータレート



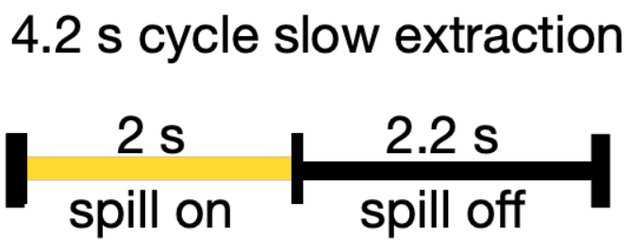
1.08 MB / TF

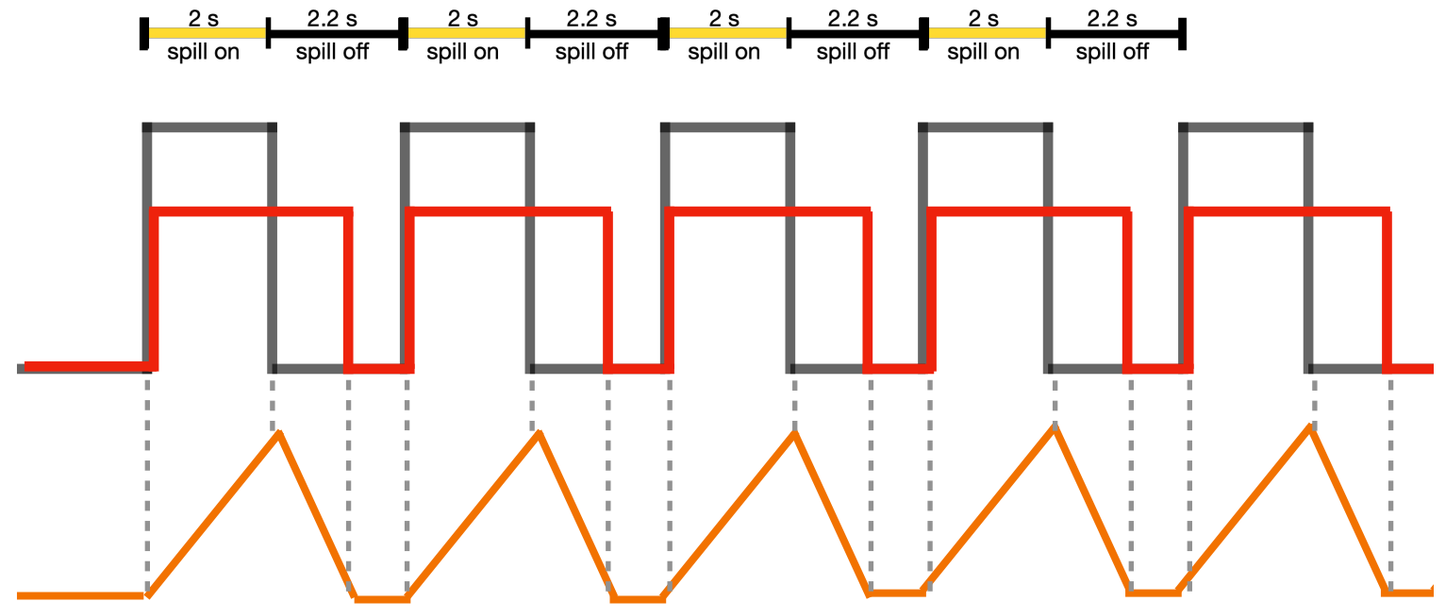
= 515 MB / real sec

処理すべきデータレート

結果 - 必要な計算資源

ビーム構造を加味した必要プロセス数





$$\text{必要プロセス数} = \frac{\text{処理すべきデータレート}}{\text{処理できるデータレート}} \times \text{duty比}$$

- ① 処理すべきデータレート = 515 (MB/s)
- ② 処理できるデータレート = 594 (MB/s)

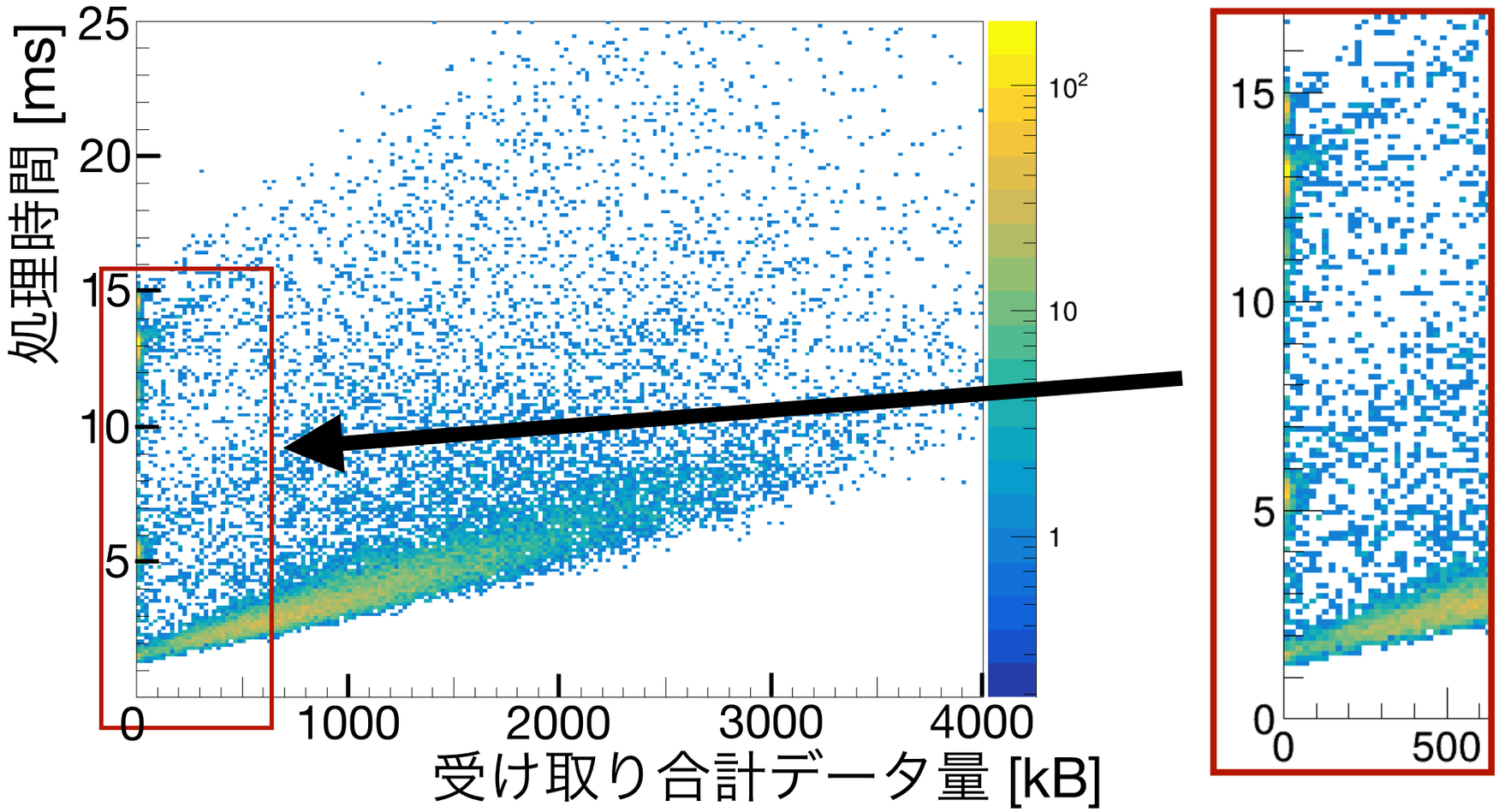
$$\Rightarrow \frac{515}{594} \times \frac{2}{2 + 2.2} = 0.41 \text{ (個)}$$

- 仮にT103セットアップでE50実験のビームレート(60 M/spill)で実験すると、少なくとも27個のTOFフィルタープロセスが要求される
- TOFの計算の組み合わせの影響で増える可能性はある

Trial&Error

結果 (全体の処理時間)　正しくない評価方法

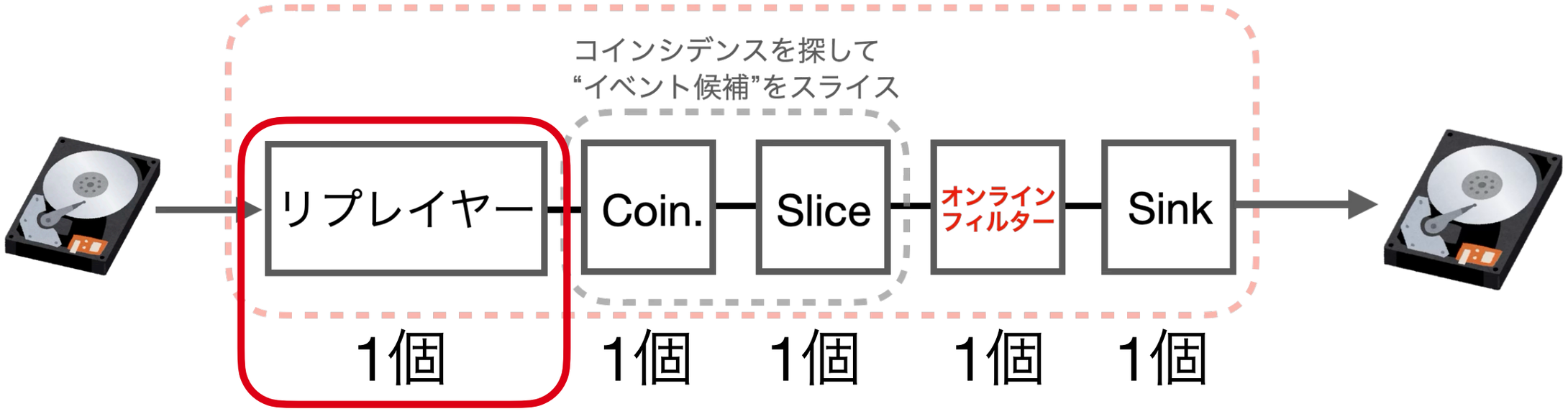
- タイマー(2)でTOFフィルター全体の処理時間を評価



- データ量の小さい時に異常に処理に時間がかかることがある
- TOF判定ではない部分：メッセージ展開時?メッセージ受け取り時?

評価方法の改善 ー キュー待ち

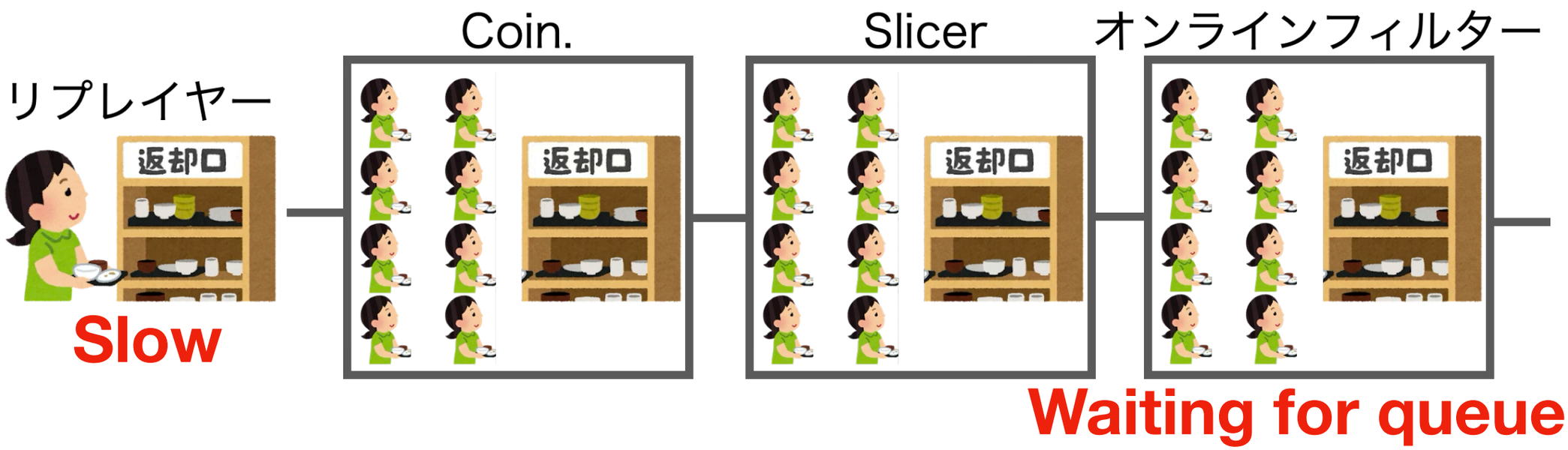
- リプレイヤーによるベンチマークで用いた構成



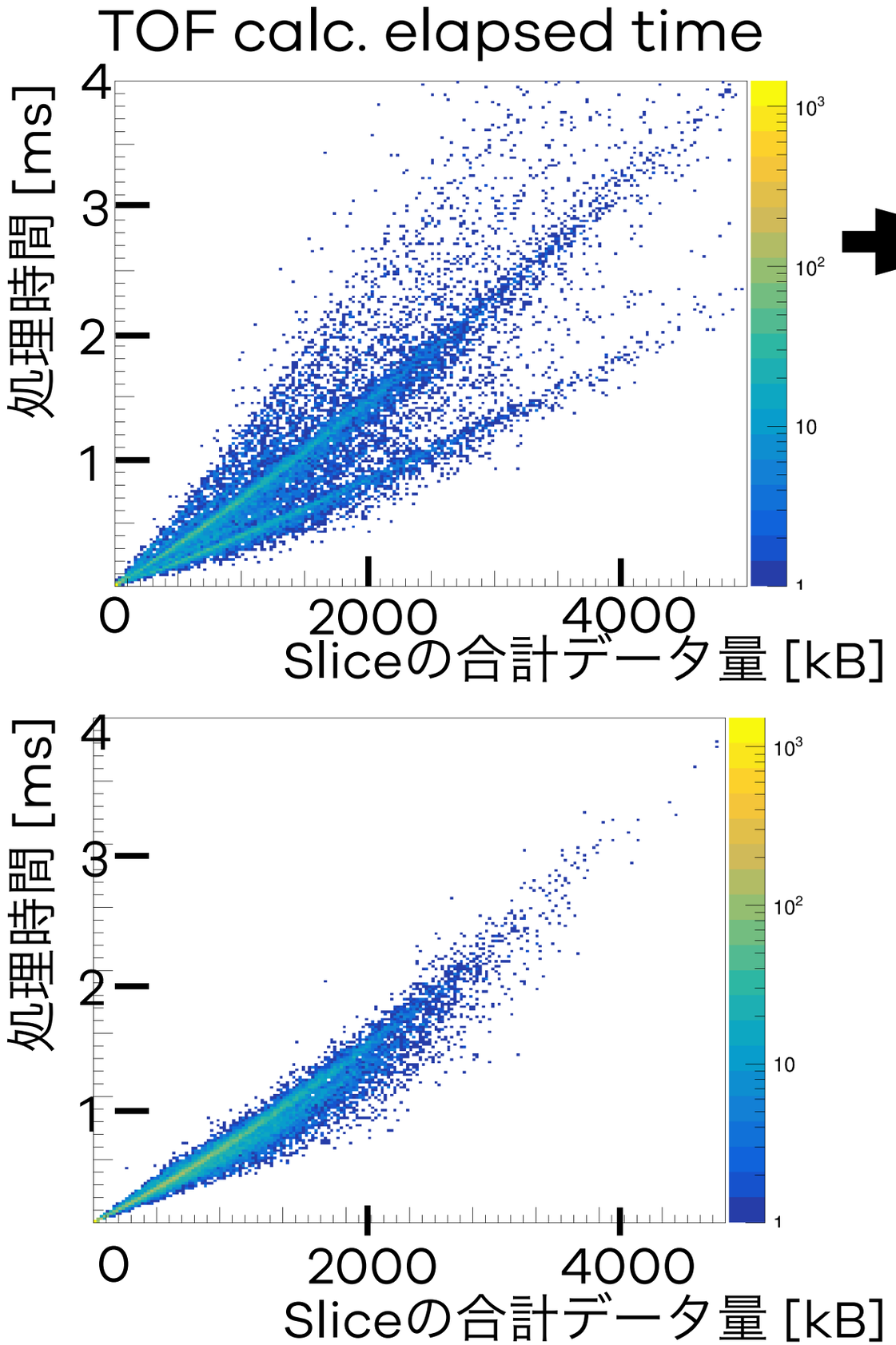
リプレイヤーの読み出しで律速していた

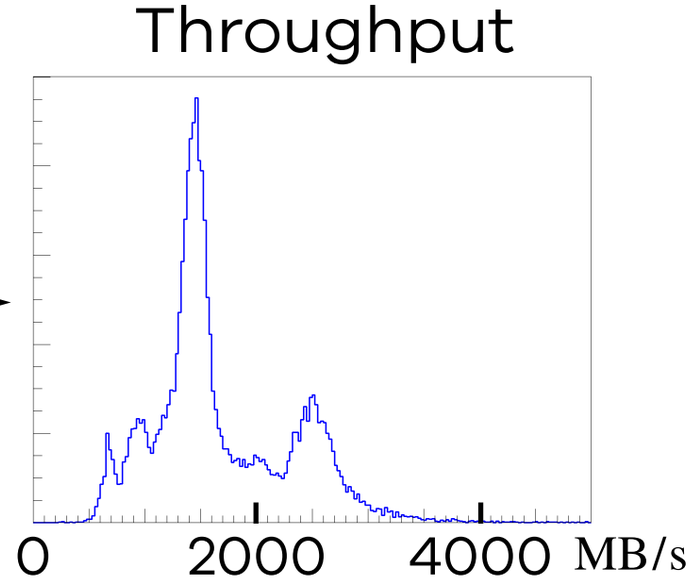
- オンラインフィルターが前段のプロセスがキューを投げてくるのを待つ時間が含まれていた

- ビジーな時の処理時間こそ正しい性能評価の指標である



評価方法の改善 — 実行環境の動作速度の固定





- 処理速度に複数の系列
- CPUの省電力&ブースト機能
- どちらもオフ & 全コア3.2 GHz固定
- ソフトウェアの性能評価に貢献

Discussion

- 適切に負荷がかかっていれば省電力とハイパフォーマンスは両立するか?

まとめと今後の展望

まとめ

- J-PARC チャームバリオン分光実験のための連続読み出し式DAQシステムの構成要素の1つとなる**TOFオンラインフィルター**の性能評価を行なった
- **TOF判定関数だけ**の時間と**TOFフィルター全体**での時間の指標を用いて評価を行なった
- フィルタープロセスの性能の評価手法を確立し、E50実験で必要とされる計算資源の量を見積もった
- NestDAQオンラインフィルターの性能評価のノウハウの共有

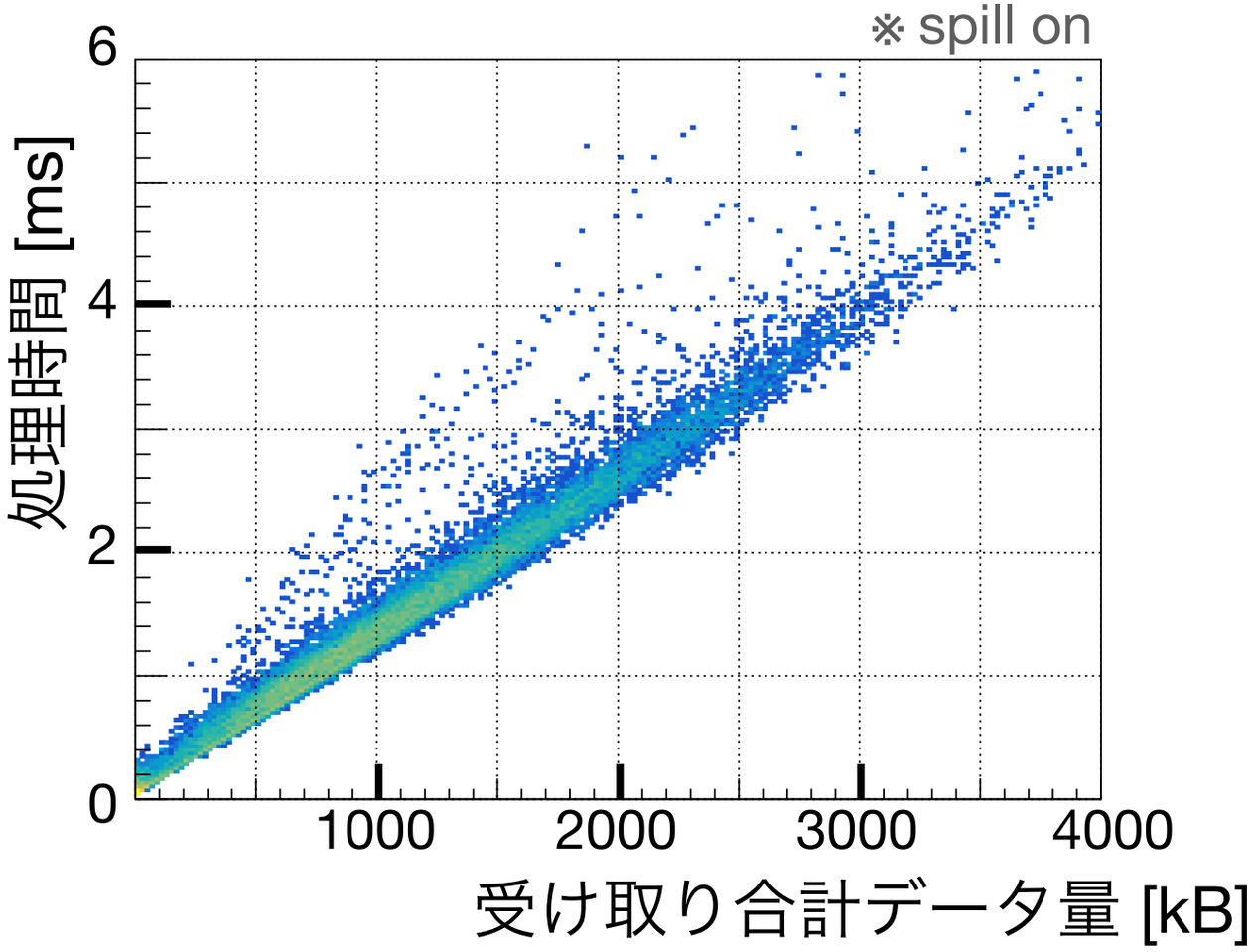
今後の展望

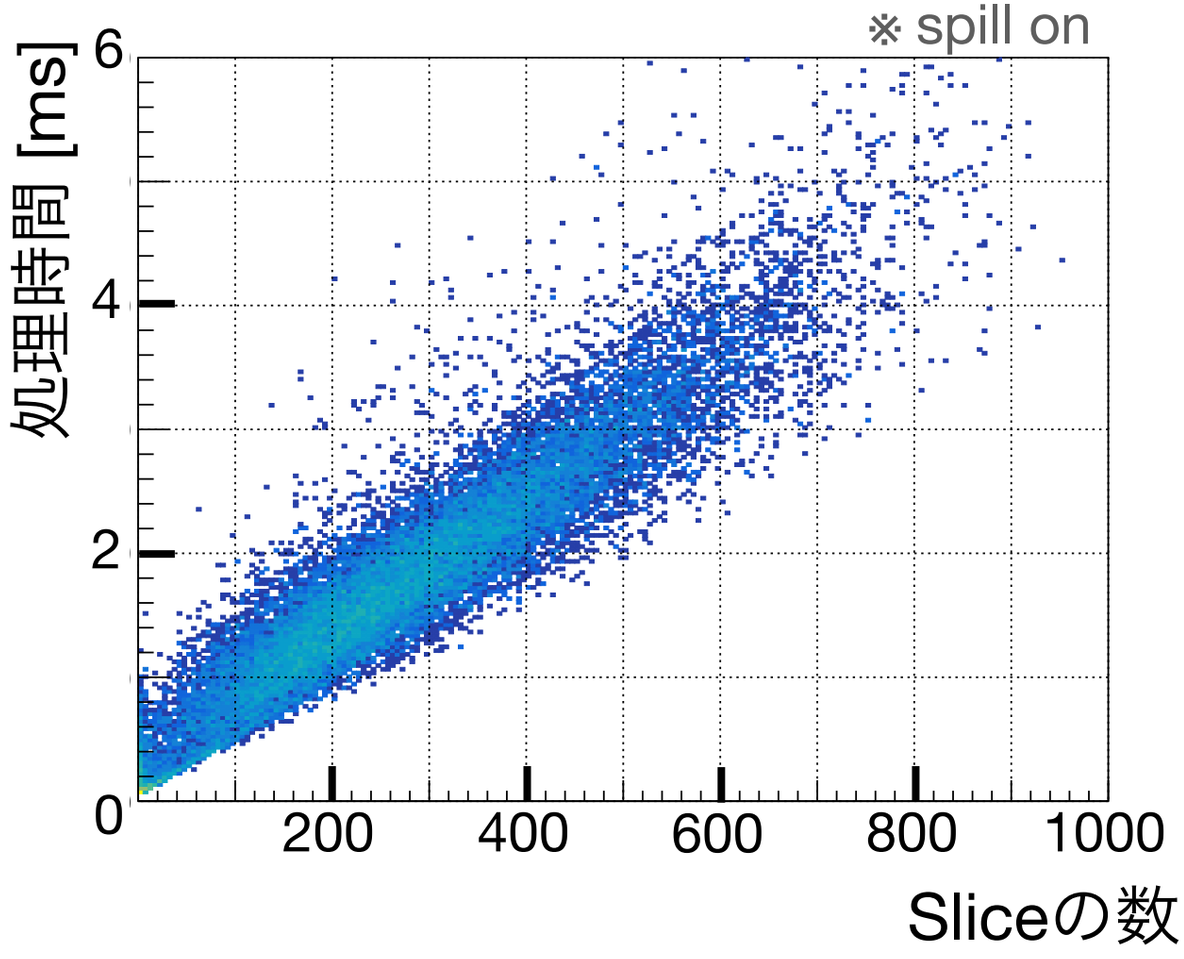
- TOF計算量(∝TDC値の差の組み合わせ)の多いときの性能評価
- GPUによる計算加速の検討
- 他のフィルターの実装(複数TOF、多重度、トラッキング、運動量...)

BACKUP

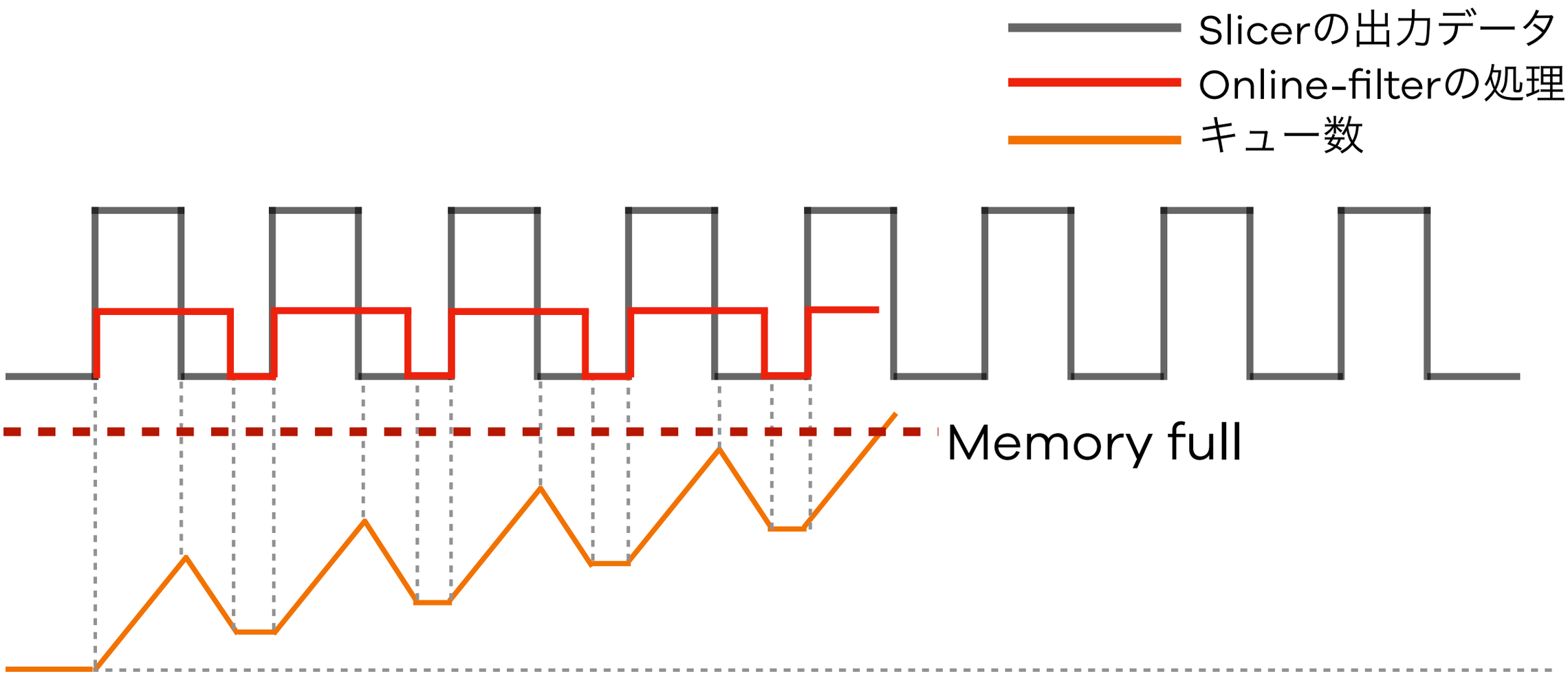
結果 (全体の処理時間)

- タイマー(2)でTOFフィルター全体の処理時間を評価

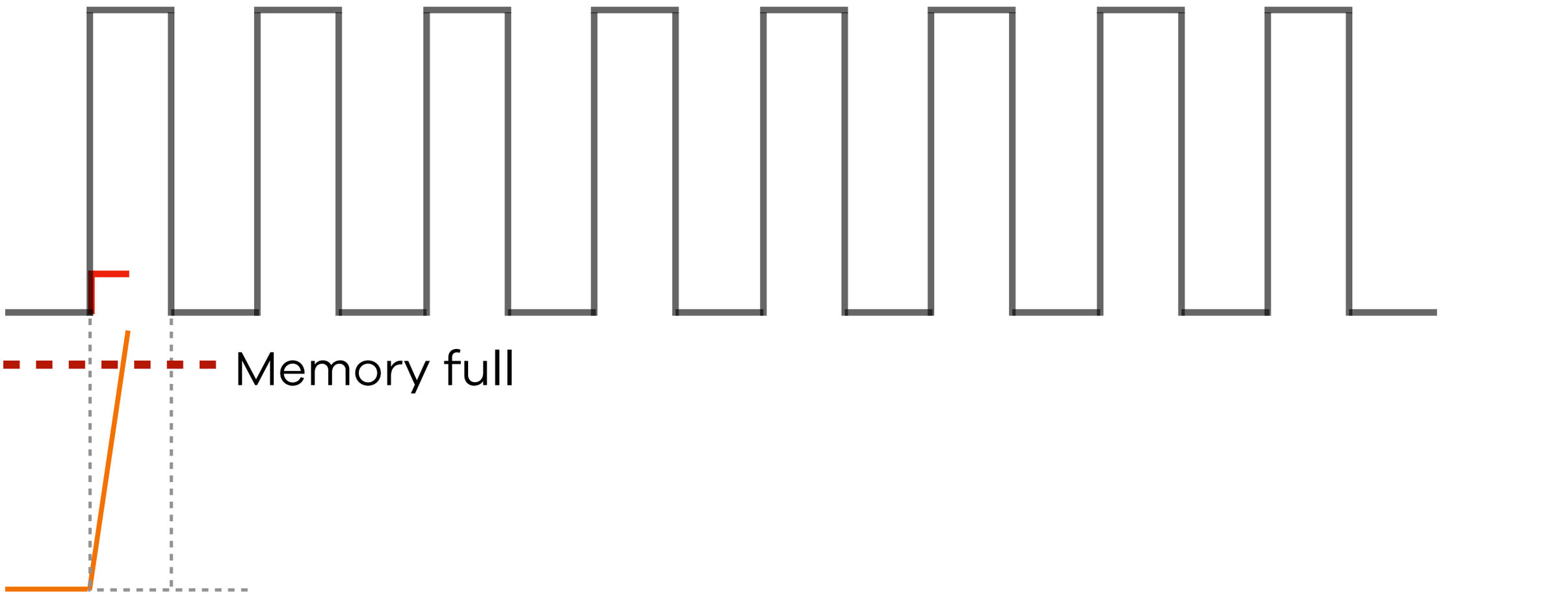




DAQの崩壊(2種類)



DAQの崩壊(2種類)

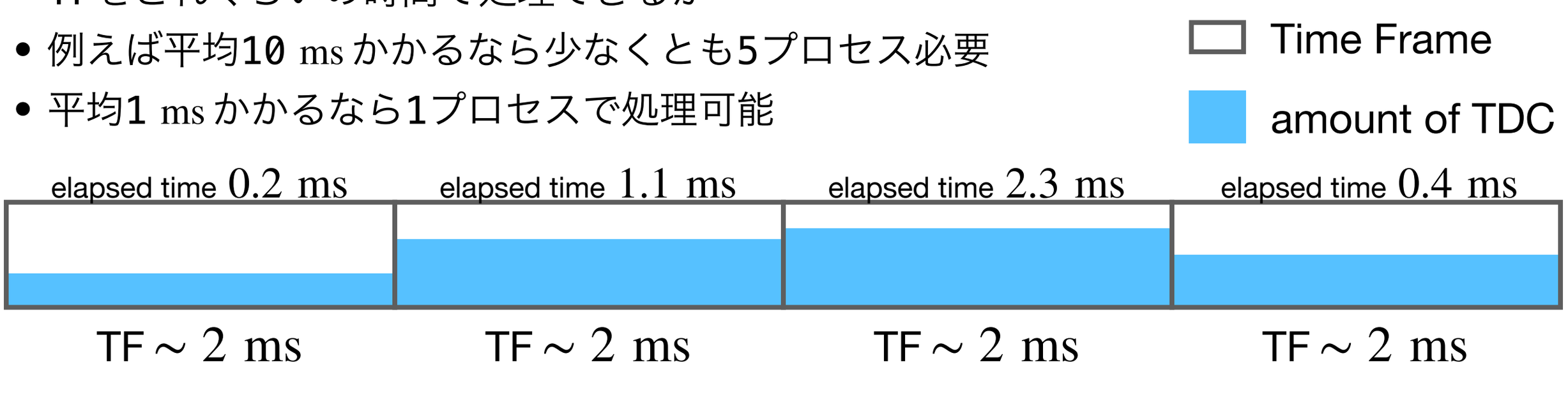


性能評価の仕方 — DAQの健全性(別の表記)

DAQの生死の境界を決める指標

- 比較すべき指標は何か
- **平均的な**処理すべきデータレートとスループット(MB/s)の比較
- **1TFの時間**($4 \times 524.288 \sim 2000$ us)と、**スループット(平均処理時間** us/TF**)**
  - スループットを「1TFを処理するのにかかる平均時間」と定義して1TFの時間と比較する
  - この定義の利点は定義が簡便であること、測定が初心者にも容易であること(データ量を測る必要がなく、NestDAQのデータ構造がコード内でどのように実装されているかを考慮する必要もない。std::chronoで時間を測れば良いだけ)

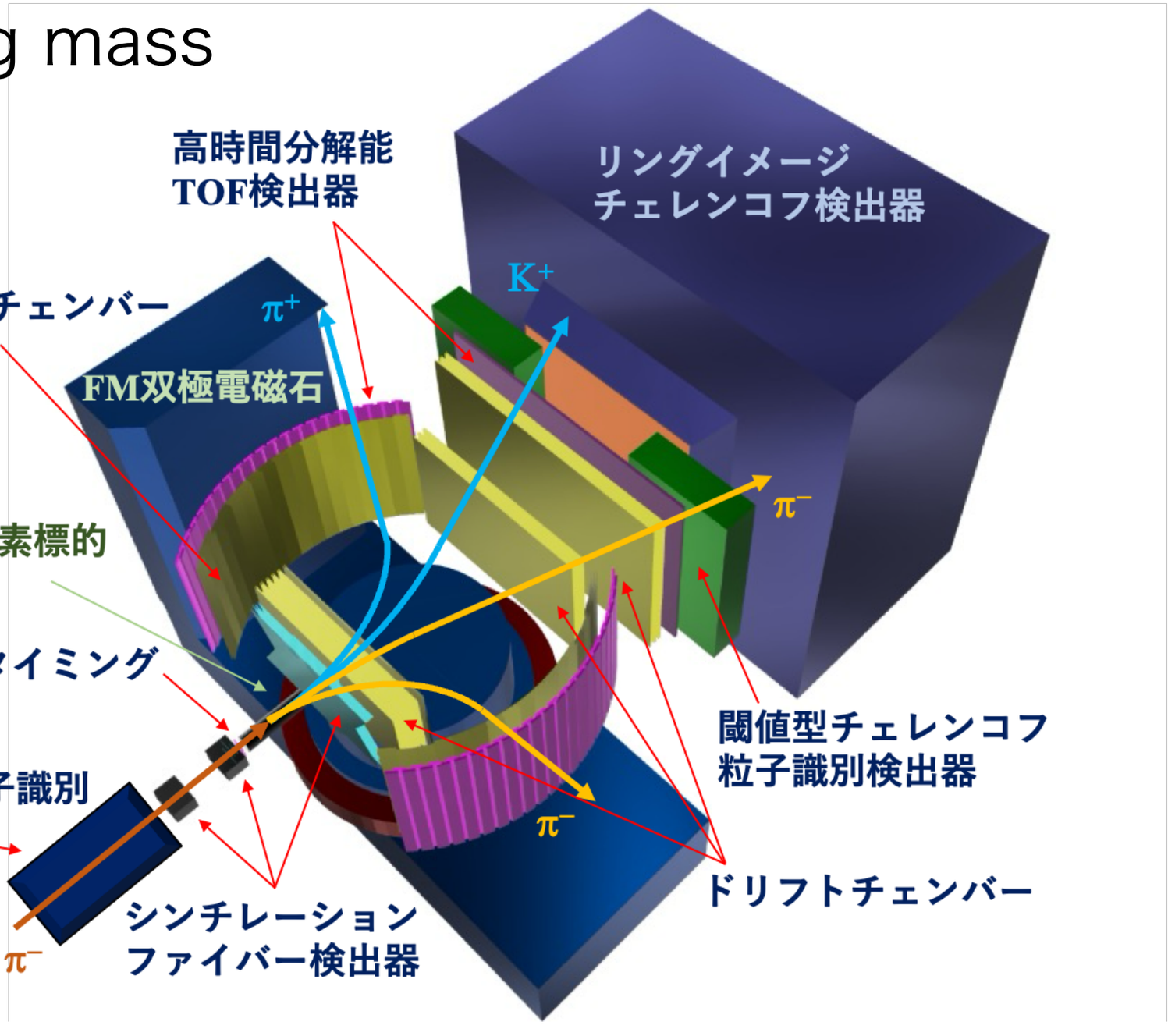
- TF ~ 2 ms ごとのTDCのパケット
- TFをどれくらいの時間で処理できるか
- 例えば平均10 ms かかるなら少なくとも5プロセス必要
- 平均1 ms かかるなら1プロセスで処理可能



(要点)

$$\pi^- + p \rightarrow D^{*-} + \text{チャームバリオン}$$

K^+, π^-, π^-を捉えてmissing mass spectroscopyをしたい



研究の背景 - なぜチャームバリオンか

1/10→merge1

ハドロンとは

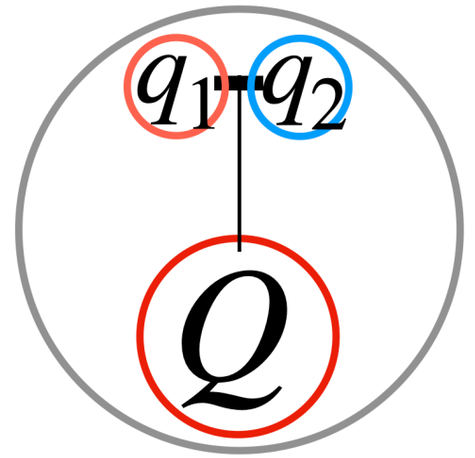
クォークが「強い力」によって閉じ込められた複合粒子

バリオンの内部構造を知る手掛かり

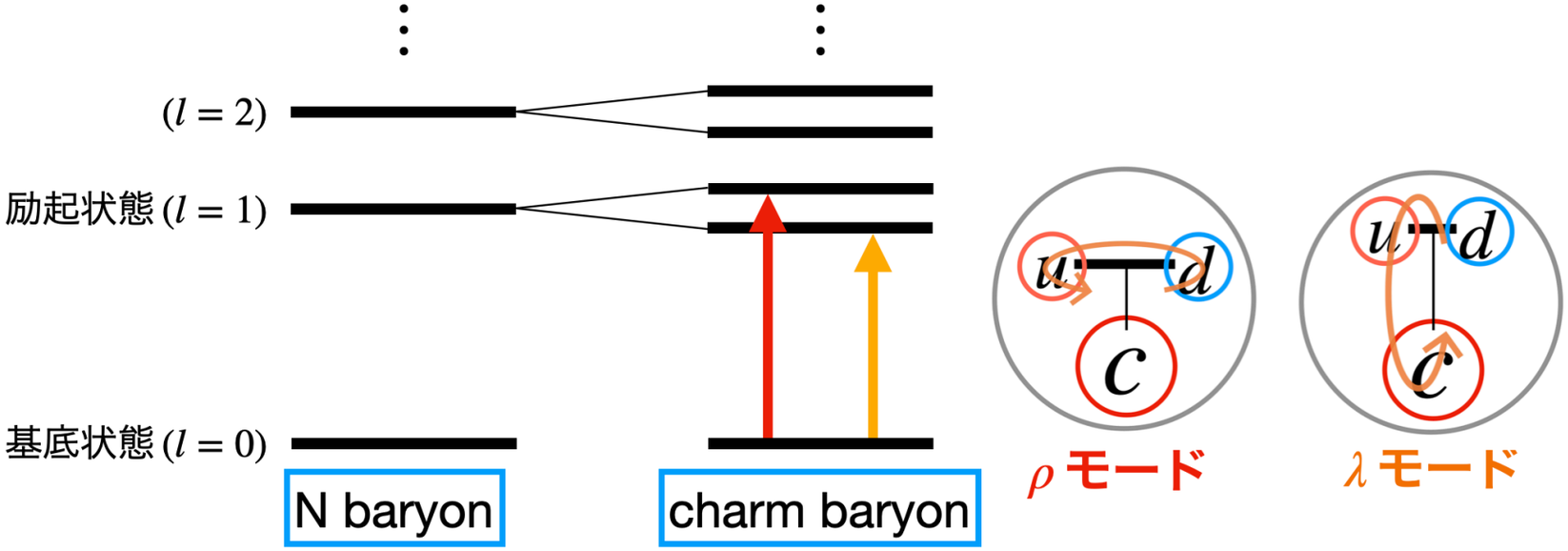
- チャームクォークを含むバリオン：チャームバリオン

① 軽いクォーク2つの相関 $q_1 q_2$ ＞ 軽いクォークと重いクォークの間の相関 $q_1 Q,\quad q_2 Q$

下図の状態が実現しているだろう



② エネルギーの違う2つの軌道励起状態 $(l = 1)$



ρ モード

λ モード

- 励起準位構造, 励起状態の生成率, 崩壊分岐比

⇨チャームバリオンの分光実験

チャームバリオン分光実験 @ J-PARC

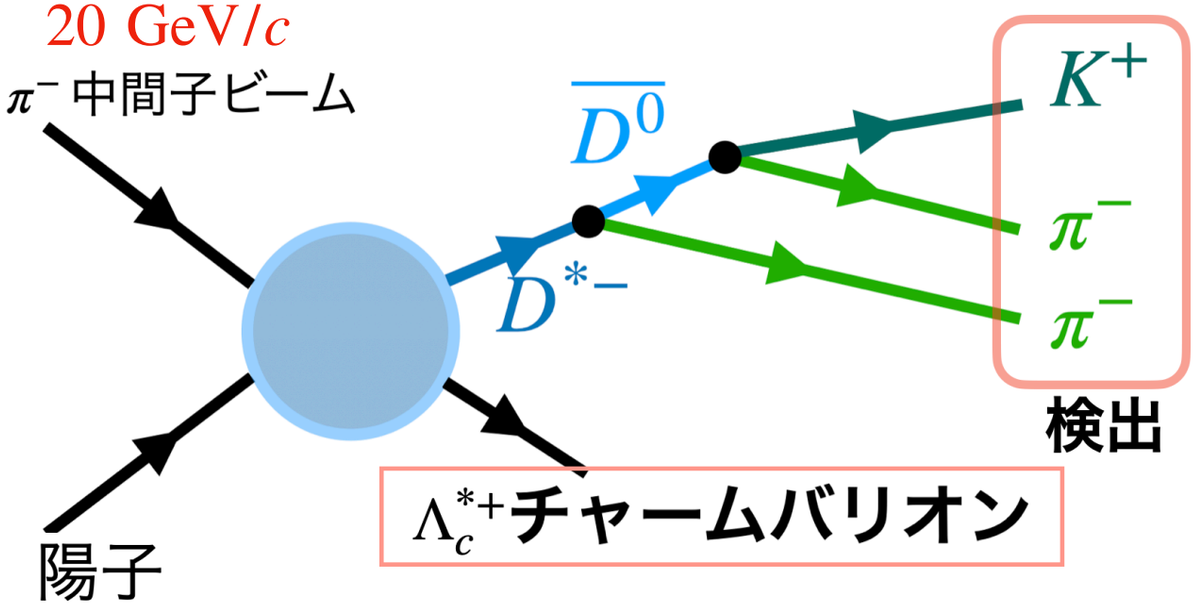
2/10→merge1

実験手法

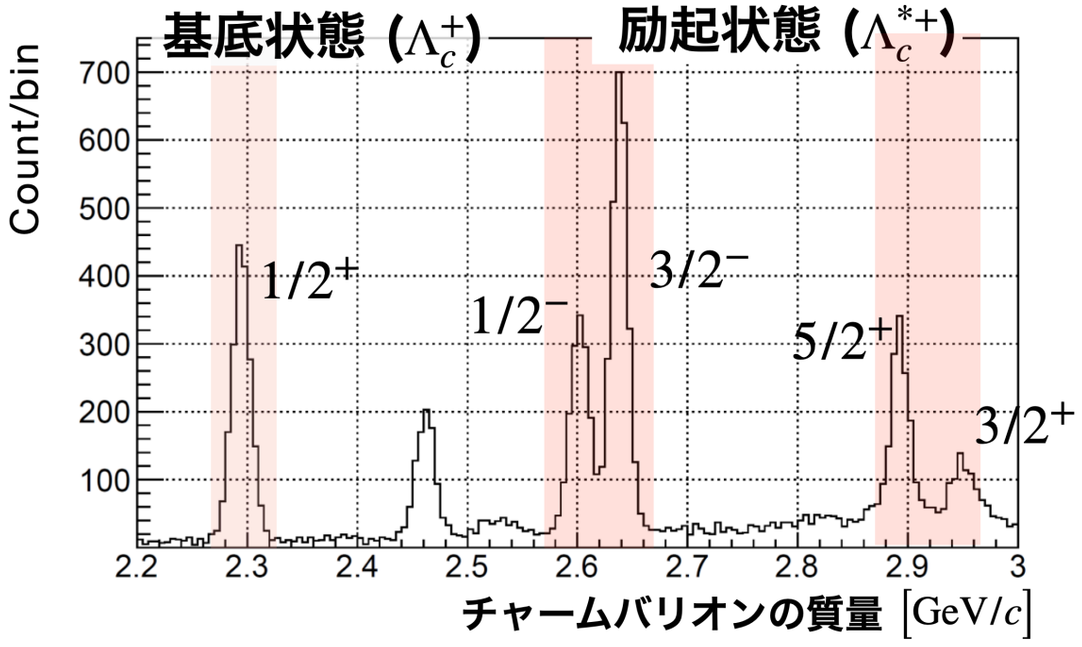
merge to P1

$$\pi^- + p \rightarrow D^{*-} + \text{チャームバリオン}$$

入射 π^- 中間子ビームと散乱 D^{*-} 中間子の運動量



チャームバリオンの質量スペクトルの測定



(⚠シミュレーション)

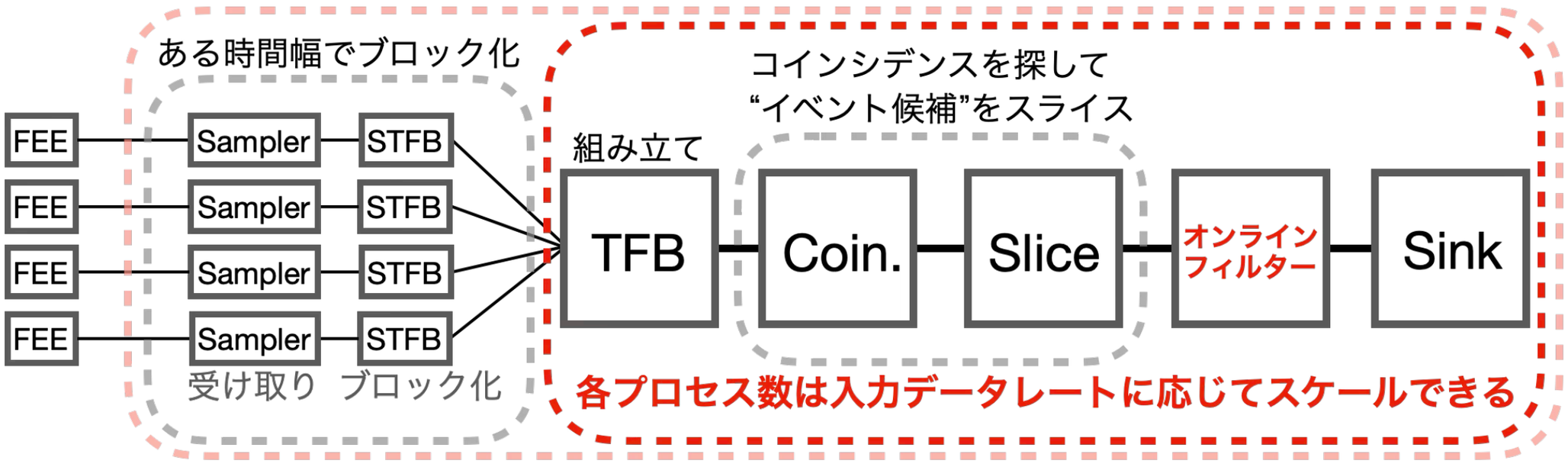
- チャームバリオン生成：nb ほどの反応断面積
- **大強度ビーム(=高反応レート)**
- トリガー回路の**デッドタイム**が課題

⇒**連続読み出し式データ収集システム**

NestDAQデータ構造補足

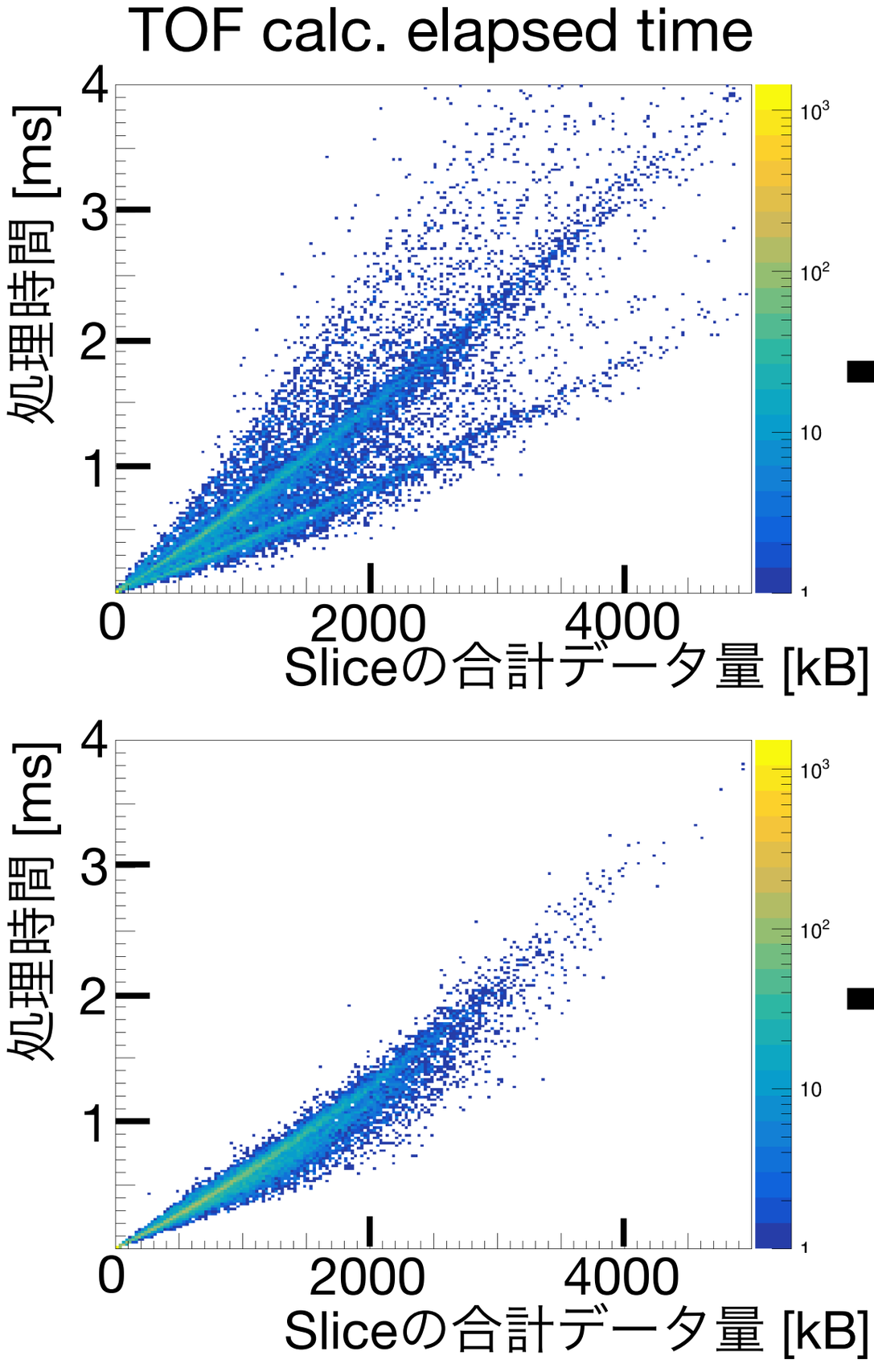
各プロセスの処理すべきデータレート

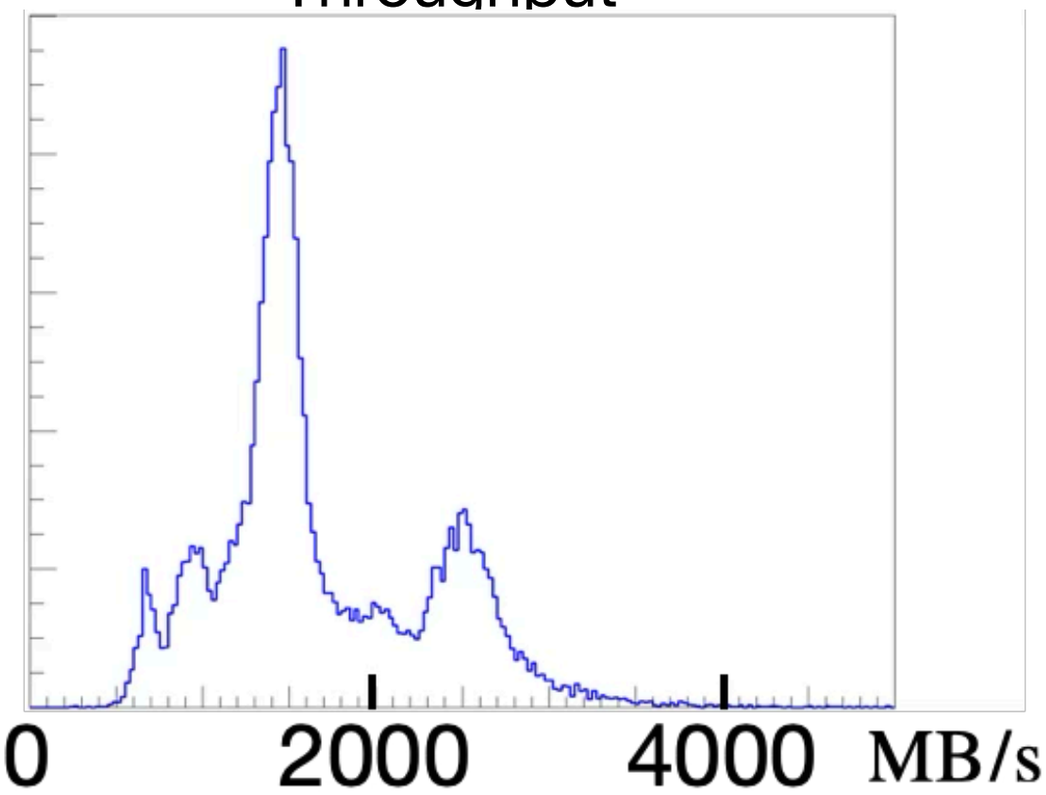
- FEEの吐くデータ構造
  - HBデリミタ(8 byte *2)/524 us
  - TDCデータ(8 byte)/hit
- SamplerでHBヘッダ(16 byte)/524 usを**付与**して後段に送る
- つまりSamplerに入ってくるデータサイズとSTFBに入ってくるデータサイズは**違っている**
- 各プロセス(Sampler, STFB, TFB…)に入ってくるデータレートは**異なる**
- つまり、各プロセスが**処理すべきデータレート**も異なる

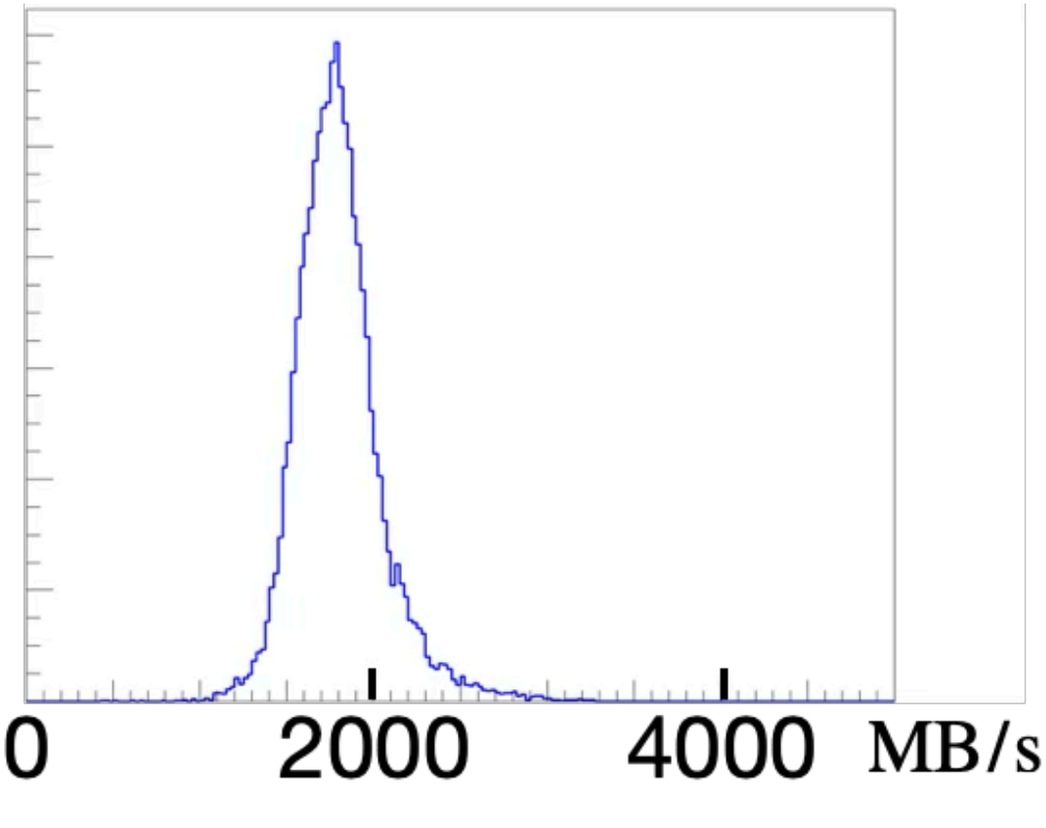


評価方法の改善 — 実行環境の動作速度の固定

trial/error

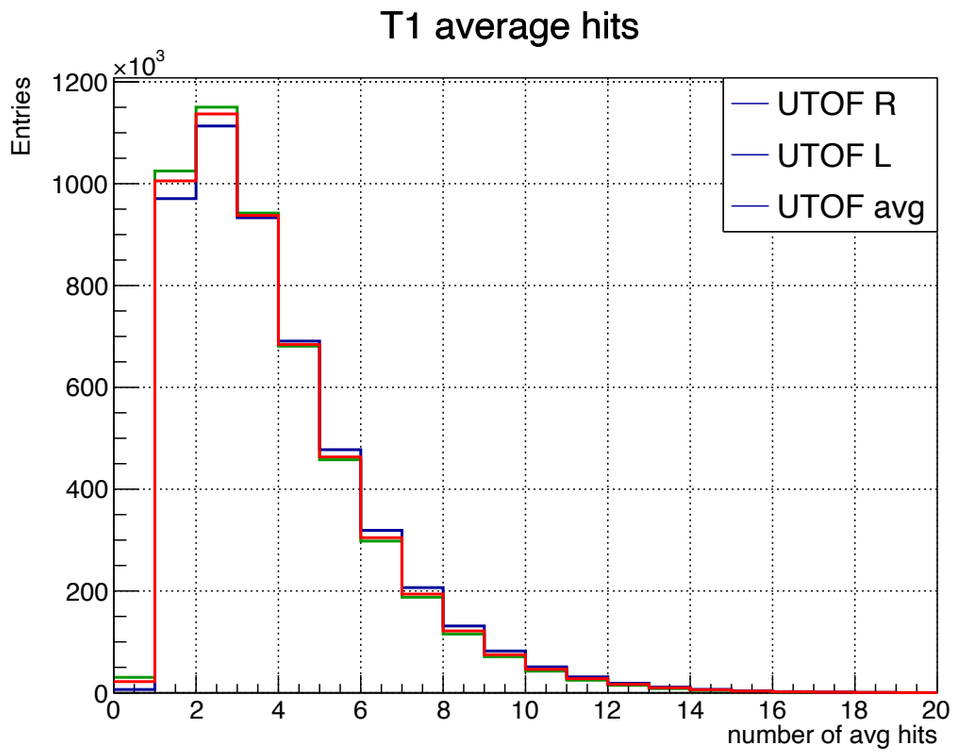






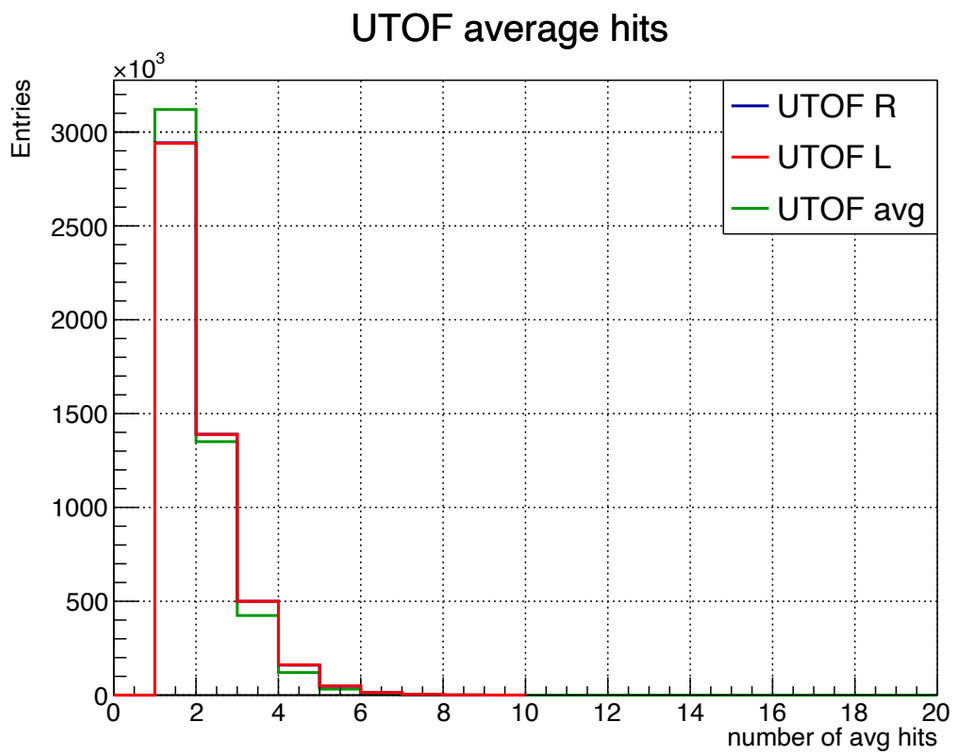
GPUを使った計算加速の検討

2025/11/05




```
Average nhits T1R: 3.5115
Average nhits T1L: 3.41697
Average num avg T1: 3.35392
Average nhits UTOFR: 1.62488
Average nhits UTOFL: 1.6283
Average num avg UTOF: 1.54351
Average num of for loop: 6.20914
```

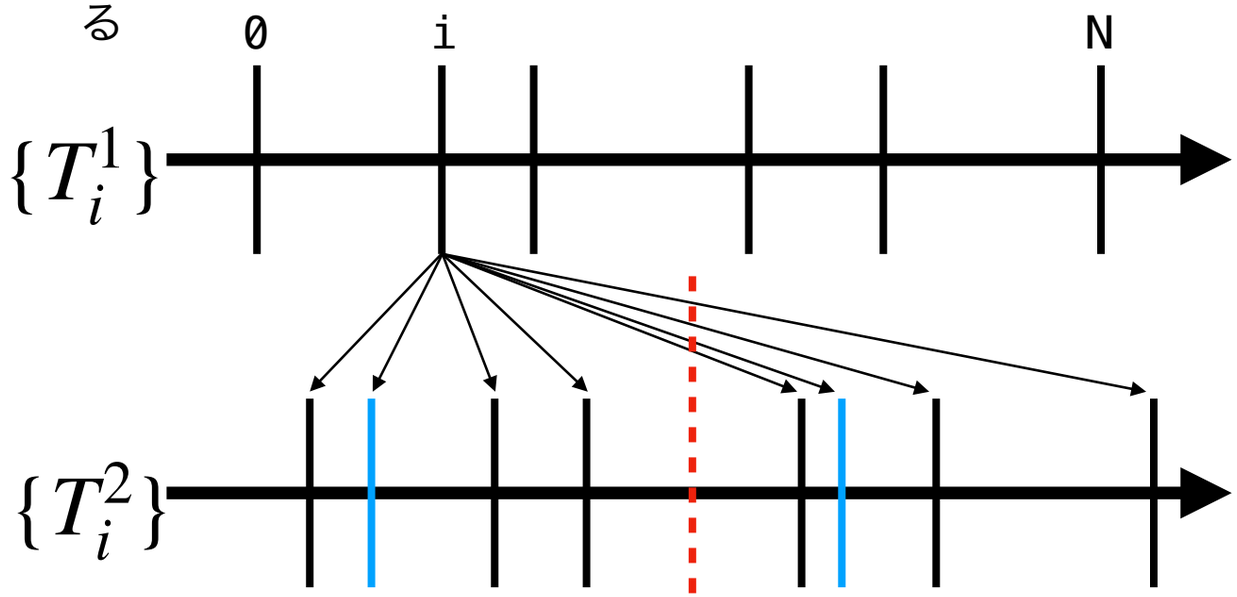
定義 ： LとRから1つづつ選び、その平均値をそのカウンターのヒット時刻とする
ただし、LとRが10ns以上離れている組み合わせは棄却する



TOF総なめは必要か

2025/11/06

- TOFの計算のアルゴリズム
  - 1スライス(LogicFilterのコインシデンス時刻から±1000 ns)に含まれるカウンター①とカウンター②のヒット時刻のベクター①，ベクター②の要素を2重forループで回している



このあたりでforループを打ち切る?

この実装は有効ではない。

なぜなら欲しいTOFを見つけたらその時点で

forループを抜けるようにすでになっている

```
        for (const auto& t1_avg : t1_averages) {
            for (const auto& utof_avg : utof_averages) {
                //最小値 最大値の順番
                if (flt->CheckAllTOFConditions(*t1_avg, *utof_avg, t1tof_r_hits, t1tof_l_hits, utof_r_hits, utof_l_hits, -131584, -127488)) { // 124.5から128.5ns
                    tofCondition = true;
                    //std::cout << "合格tof: " << *utof_avg - *t1_avg << std::endl;
                    //std::cout << "min: -130000" << " " << "max: -125000" << std::endl;
                    break;
                }
            }
            if (tofCondition) break;
        }
```

TOFが欲しい範囲に見つからなかったら、総なめすることになる。切り出してるウィンドウがそんなに大きいわけじゃないし、途中で計算を切りやめる判定にむしろ計算コストが要る。わざわざ実装することが得策であるとは思えない。