Nuclear cross section analysis with ROOT6

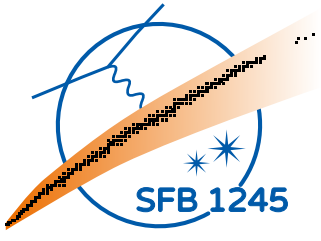
current status

Axel Frotscher

TU Darmstadt
ROOT User's Workshop 2018 Sarajevo

12th September 2018

TECHNISCHE
UNIVERSITÄT
DARMSTADT



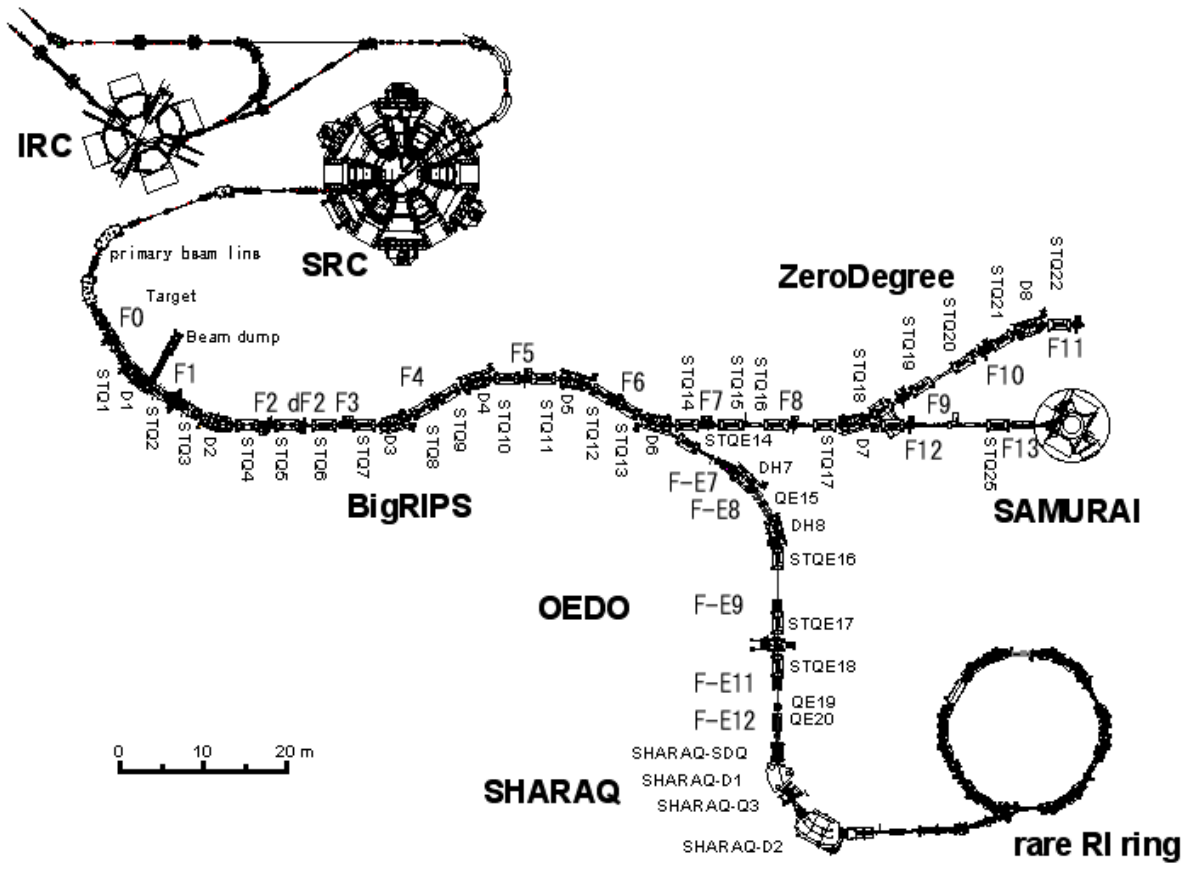
outline

SEASTAR - Campaigns

- SEASTAR = **S**hell **e**volution **a**nd **s**earch for **t**wo-plus energies **a**t the **R**IBF, measurement campaign at RIBF in RIKEN, Japan
- Goal: Deduction of proton-knockout reactions (p,2p), (p,3p) for neutron-rich medium-mass nuclei
- three campaigns: 2014, 2015, 2017
- disentanglement of (p,3p) reaction mechanism possible

Setup

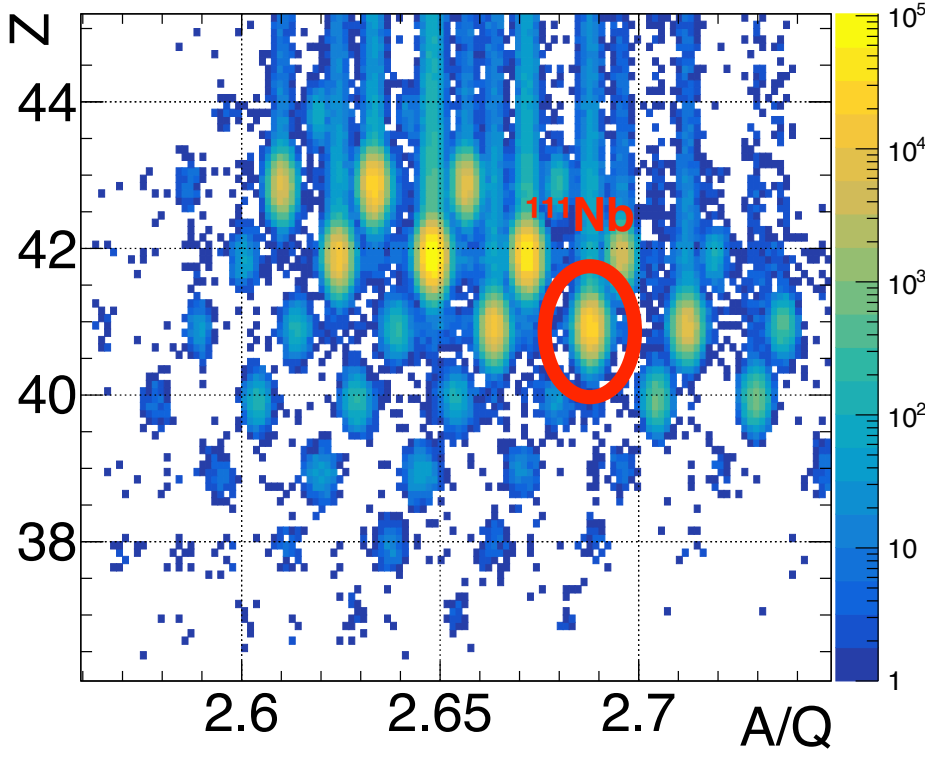
- ^{238}U ablation-fissioned
- fragments guided through beamline, BigRIPS separation
- they hit secondary, outgoing particles separated with ZeroDegree



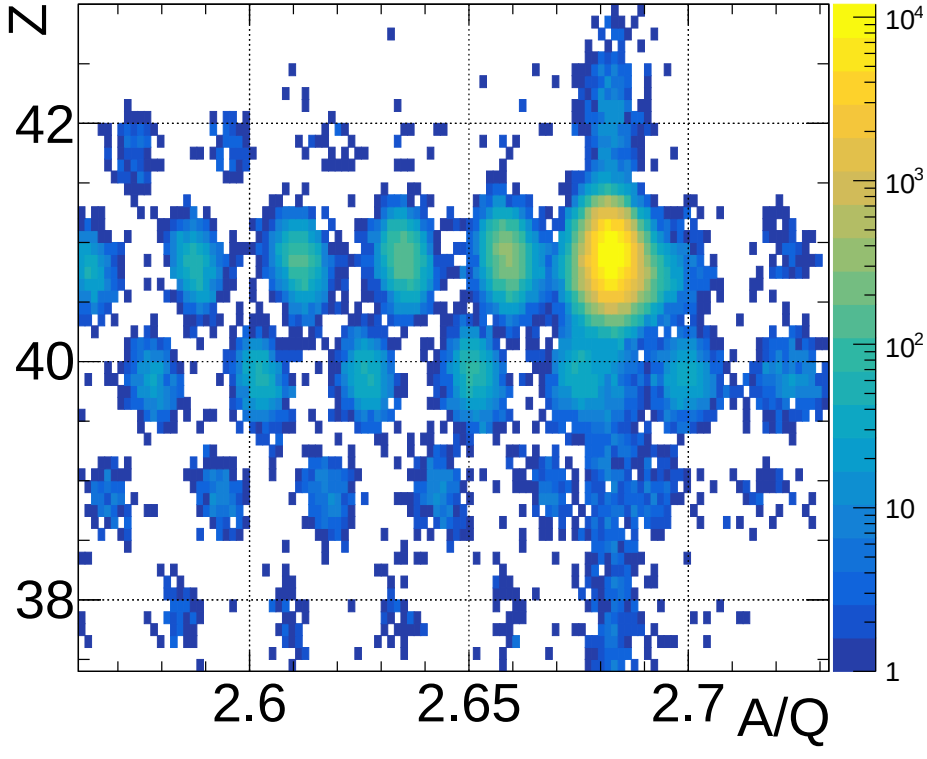
Overview of BigRIPS, [1].

Results- preliminary

- PID spectra, get cross section from ratio's



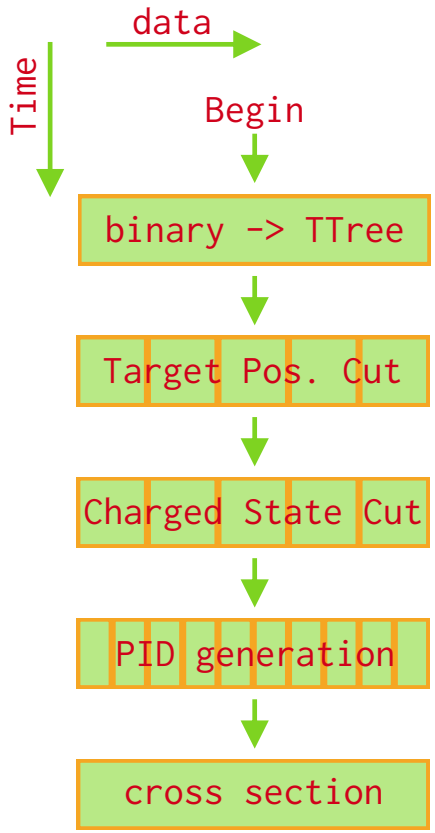
^{111}Nb-Setting, incoming particles.



Outgoing particles with ^{110}Nb cut on incoming particles.

Data analysis

- done with compiled C++ program for performance and maintainability
- automatic conversion to ROOT-Tree with ANAROOT (once)
  - 1st: Applying cuts on data (on position, Energy, particle ID, ...)
  - 2nd: generate particle ID (PID) plots for cross section
  - 3rd: calculate further corrections and cross sections
- parallelization by data, **not** by task



Analysis flow chart.

Reading in a TTree - easy as cake?

- recommended: TTreeReader class → works, but not for 2-dim. array (RDataFrame not available in 6.12 ↔ not tested)
- Solution: use custom ROOT-generated class

```
root file.root
TreeName->MakeClass("inputreader");
```

- gives you new class to access this tree, in program:

```
inputreader reader("file.root");
```

- works with TChain, supports partial branch read, comes with predefined loop method
- tree leaves are public members of this class
- class-objects cannot reset branch reading status after events have been read

my way of multithreading

- Performance: parallelization by data (for large datasets)
- Convenience: parallelization by task
- ROOT has many pseudo-parallelization built in (PROOF, Reading events, IMT, etc. pp.)
  - tedious, hard to understand, none compiled for me (gcc7.3)
- here: **C++11 thread**, basic usage:

```
1 #include <thread>
2
3 std::thread mythread(myfunction, arg1, arg2);
4 mythread.join();
```

- part of C++ core, super simple syntax

Preparing and setting up threads

- analysing method (triggercut.cpp)

```
1  void triggercut::analyse(const vector<string> input){
2    /* constructing the TChain from input */
3
4    vector<treereader*> tree;
5    for(auto *i:chain) tree.emplace_back(new treereader(i));
6
7    vector<string> keys{"EventInfo.fBit"};
8    for(auto &i:tree) i->setloopkeys(keys);
9
10   vector<thread> th;
11   for(uint i=0; i<threads;i++){
12     vector<uint> ranges = {(uint)(i*goodevents.size()/threads),
13                            (uint)((i+1)*goodevents.size()/threads-1)};
14     th.emplace_back(thread(&triggercut::innerloop, this,
15                            tree.at(i),ref(goodevents), ranges));
16   }
17   for(auto &t : th) t.join();
18 }
```

thread content

- and the inner loop (triggercut.cpp)

```
1  void triggercut::innerloop(treereader *tree, vector
       <atomic<bool>> &goodevents, const vector<uint> range) {
2      int i = range.at(0);
3      int threadno = range.at(0)/(range.at(1)-range.at(0));
4
5      while(i<range.at(1)){
6          if(goodevents.at(i)){
7              tree->getevent(i);
8              if(tree->EventInfo_fBit[0] == 6){
9                  goodevents.at(i).exchange(false);
10             }
11         }
12         i++;
13     }
14 }
```

- multithreading doable, even for beginners

Multithreading - words of advice

- don't use it for fun but for performance increase
- prevent data races at all costs
  - don't use shared objects or
  - use objects explicitly marked thread-safe (atomics) or
  - employ locks (compromises performance)
- stay away from TThreadedObject's in std::vector containers
- never assume objects/trivial operations to be thread-safe unless you checked

ROOT Wish-List

- Add Support for n-dim. arrays in TTreeReader/RDataFrame
- Add atomic fill for TH*-Classes
- Add public copy constructor for TH1 (even shallow copy is ok!)

End

thank you for your attention!

Bibliography

[1] N. Fukuda *et. al.* (2013): *Identification and separation of radioactive isotope beams by the BigRIPS separator at the RIKEN RI Beam Factory*
NIM **317**, 323-332

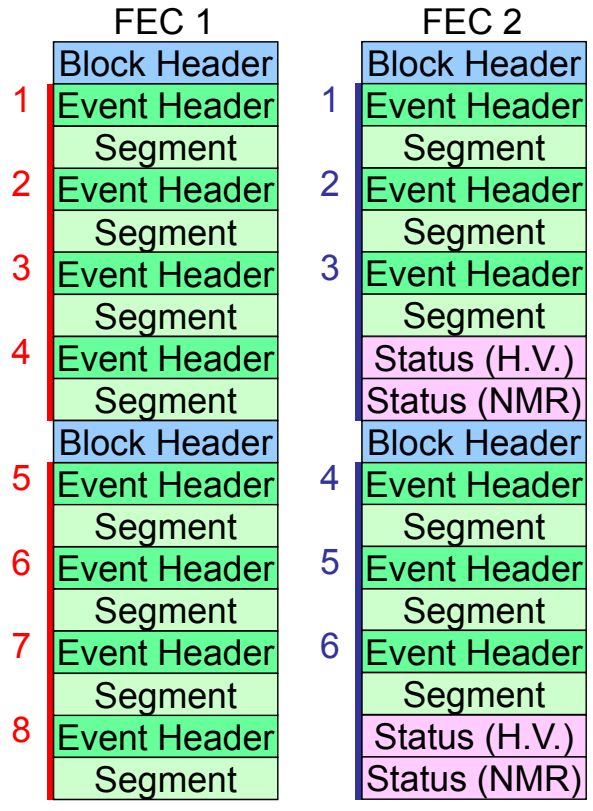
[2] Hidetaba Baba (2010): *RIBF Data Format Version 1.5*
https://ribf.riken.jp/RIBFDAQ/index.php?DAQ%2FManual%2FDataformat

data structure

- 500 GiB of Data
- data is stored in blocks, per event basis
- flexible segment length for variable data
- Event Header identifier (32 bit):

Revision	6 bit
Device	6 bit
Focal Plane	6 bit
Detector	6 bit
Module	8 bit

- followed by data segment



Data Structure, [2].

Multithreading addendum

- How to start?
  - using separate class (.h/.cpp) for each cut
  - one method for initialization, one for loop

```
1  class targetcut{
2  public:
3      void innerloop(arg1);
4      void analyse(arg2);
5      targetcut(arg1,arg2){analyse(input, output);
6      };
7
8      std::vector<std::atomic<bool>> &goodevents;
9  private:
10     const int threads = 7;
11     std::mutex unitemutex;
12     std::vector<TH2D> tarhist;
13 };
```

targetcut.h

Backup 1

- Code at Github:
  `https://github.com/AxelFrotscher/clementine`
- open until 14th September 2018