Lancaster University

The Response in Argon at Pressures of 3750 torr (RAPtorr) event framework

D. Brailsford

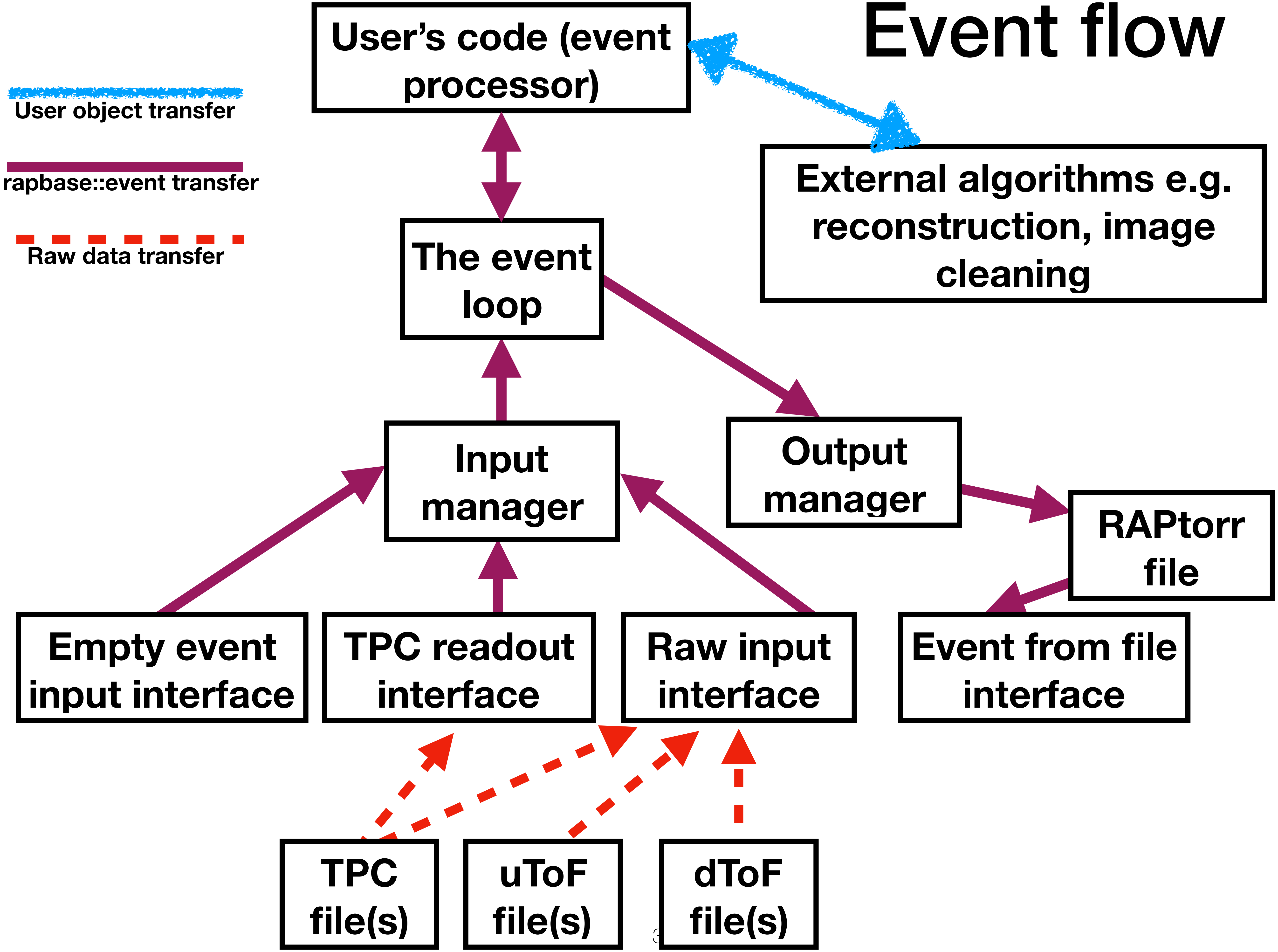
HPTPC RAPtorr ‘hackathon’

10/01/19

RAPtorr key features

- RAPtorr is a lightweight, configuration file-based, event processing framework
- RAPtorr provides the user with an interface to a generic ‘event’ object (a rapbase::Event) for analysis purposes whilst also handling retrieval/ creation and long term storage of said event
- The user is able to store **any** c++ classes they wish inside the rapbase::Event’s container
    - No need for TObject inheritance
- Users are able to form relationships between objects stored in the rapbase::Event
- Users are able to pass parameters via configuration files to their analysis code
    - No need to recompile each time you want to tweak a parameter

Event flow


User object transfer
rapbase::event transfer
Raw data transfer
User's code (event processor)
External algorithms e.g. reconstruction, image cleaning
The event loop
Input manager
Output manager
RAPtorr file
Empty event input interface
TPC readout interface
Raw input interface
Event from file interface
TPC file(s)
uToF file(s)
dToF file(s)

RAPtorr key features

- RAPtorr is a lightweight, configuration file-based, event processing framework
- RAPtorr provides the user with an interface to a generic ‘event’ object (a rapbase::Event) for analysis purposes whilst also handling retrieval/creation and long term storage of said event
- The user is able to store *any* c++ classes they wish inside the rapbase::Event’s container
    - No need for TObject inheritance
- Users are able to pass parameters via configuration files to their analysis code
    - No need to recompile each time you want to tweak a parameter

Usercode/event processor

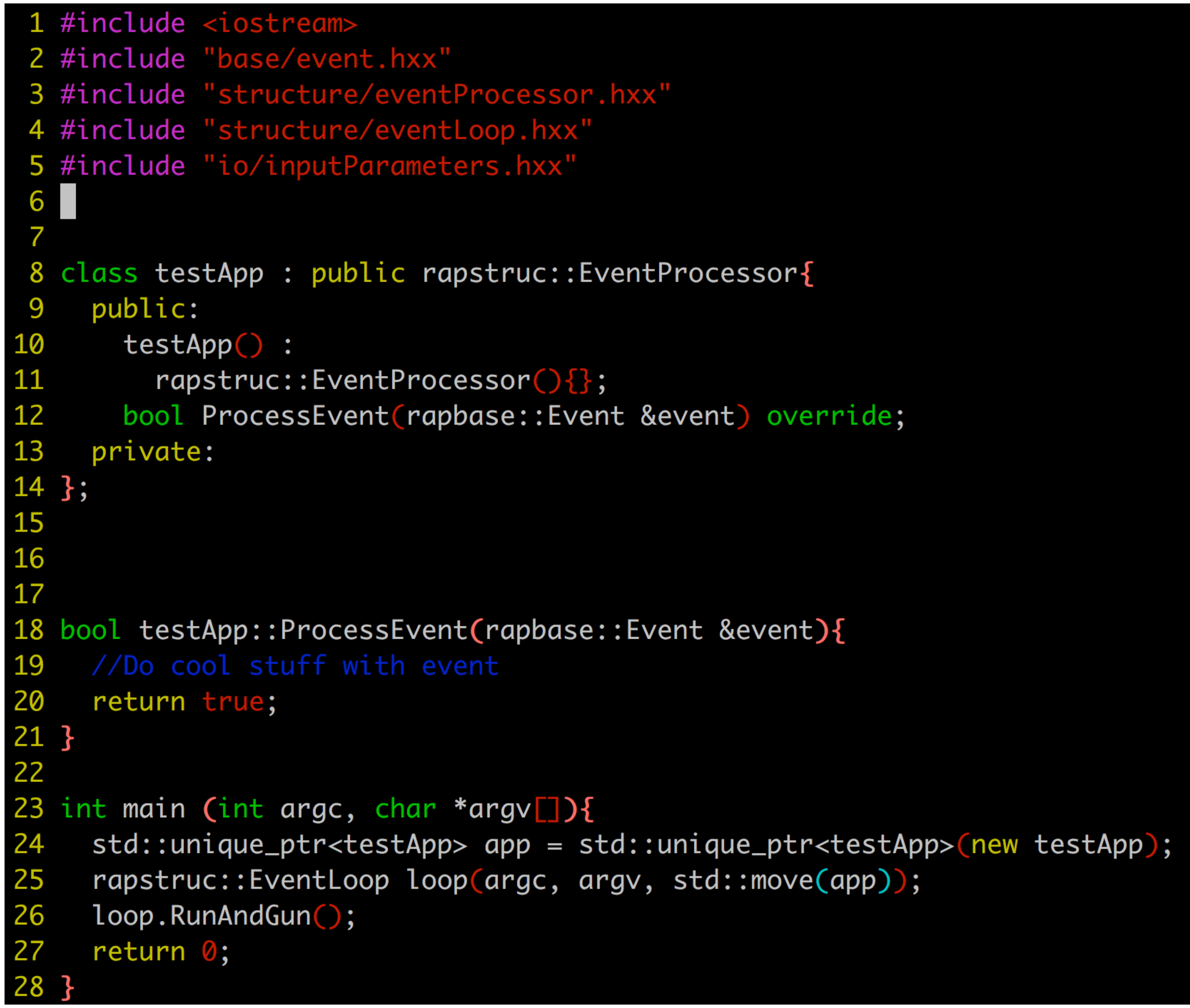
- Users write their own executable class which inherits from the rapstruc::EventProcessor

- The user's class then gets executed at certain points in the event loop

- Most important function is **ProcessEvent**

    - This function provides the interface to each rapbase::Event

- Other functionality also provided (but not shown on the example on the right)



```cpp
#include <iostream>
#include "base/event.hxx"
#include "structure/eventProcessor.hxx"
#include "structure/eventLoop.hxx"
#include "io/inputParameters.hxx"


class testApp : public rapstruc::EventProcessor{
  public:
    testApp() :
      rapstruc::EventProcessor(){};
    bool ProcessEvent(rapbase::Event &event) override;
  private:
};



bool testApp::ProcessEvent(rapbase::Event &event){
  //Do cool stuff with event
  return true;
}

int main (int argc, char *argv[]){
  std::unique_ptr<testApp> app = std::unique_ptr<testApp>(new testApp);
  rapstruc::EventLoop loop(argc, argv, std::move(app));
  loop.RunAndGun();
  return 0;
}
```

Saving information in the rapbase::Event

- Straightforward process
- You need:
    - A label to save your object with
    - A unique_ptr to your object
- Then it’s a one line command to store

```
std::string container_name = "testclasscontainer";
std::unique_ptr<rapobj::TestClass> test_obj = std::unique_ptr<rapobj::Test…
event.AddProduct(container_name, std::move(test_obj));
```

Retrieving information from the event

- Slightly more faffy
- The event hands over a ‘ProductBoxContainer’ (similar to a vector) which hold a collection of ‘ProductBox’es where each ProductBox holds an object you’ve stored in the event
  - The tutorial code can explain to better than I can here
- To get the stored objects, you extract each product box, then extract the object
- Why this way?
  - ProductBox needed to make object storage class agnostic
  - ProductBox holds relationship information
  - System is based on handing over references to the underlying object. The event owns the object but it can loan it back to you

The role of the configuration file

- The usercode/event processor necessary for rapbase::event interaction
- The usercode/event processor has no power over where a rapbase::event comes from or where it goes after you’re done with it
- It’s also very useful to be able to pass parameters to your code at run time
- This is the role of the config file
    - (and why there is the faffy libconfig dependency)

The RAPtorr config file

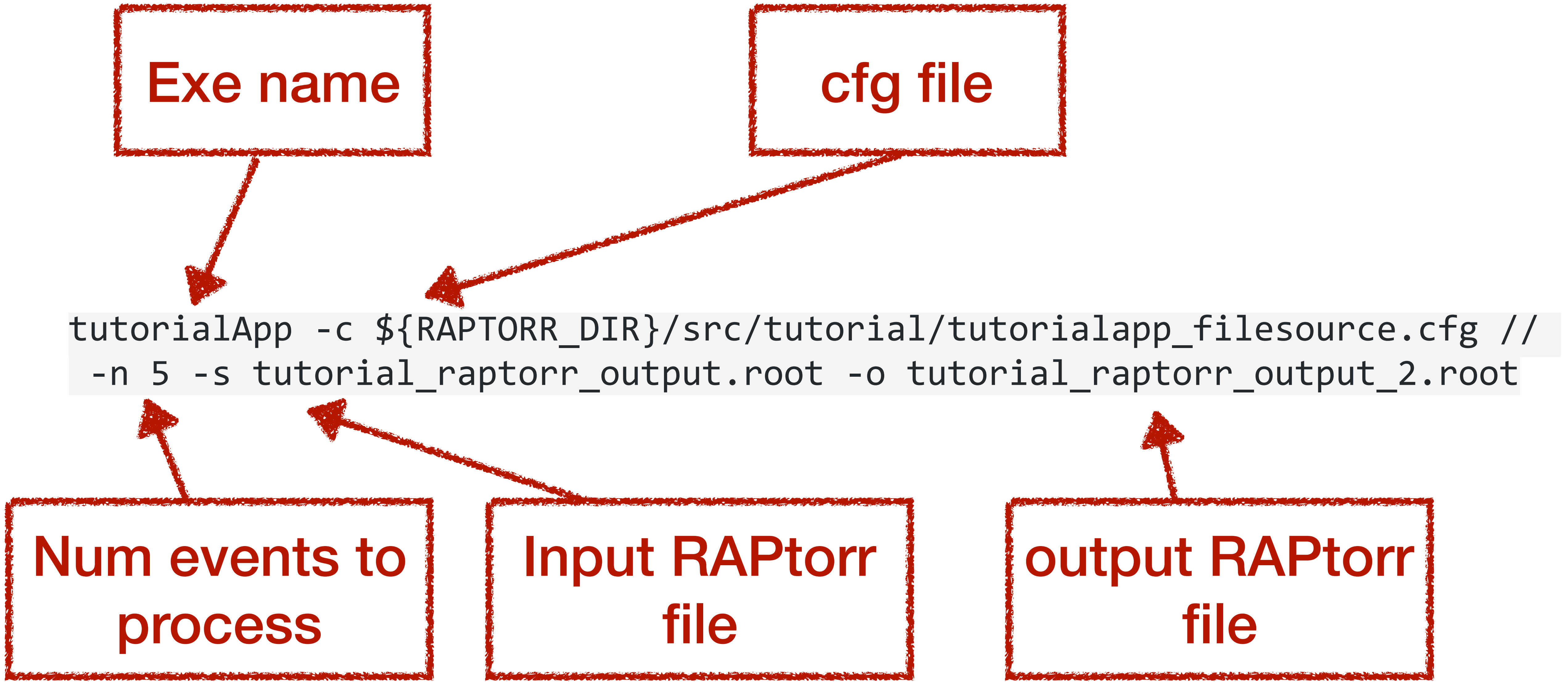
```
source = {
 type = "empty";
 run = 1;
 subrun = 1;
  first_event = 1;
}

output = {
 save_events = true;
 file_name = "tutorialoutput.root";
}

app = {
  tutorialApp = {  #name of your user code class
    #Everything gets passed to your user code
    test_input = "WOW!";    #Put whatever input parameters you want here
  }
}
```

Where the rapbase::event comes from

Where the rapbase::event goes after you're done

Pass parameters to your code

Running a raptorr exe



```
tutorialApp -c ${RAPTORR_DIR}/src/tutorial/tutorialapp_filesource.cfg //
 -n 5 -s tutorial_raptorr_output.root -o tutorial_raptorr_output_2.root
```

Developer tenants

1. Do not use bare pointers unless absolutely necessary
    1. Opt for unique_ptr or references when possible
2. wrap classes, functions inside an appropriate namespace. Unless you are working in an empty directory, look at the namespace neighbouring files use
3. If in doubt, ask!

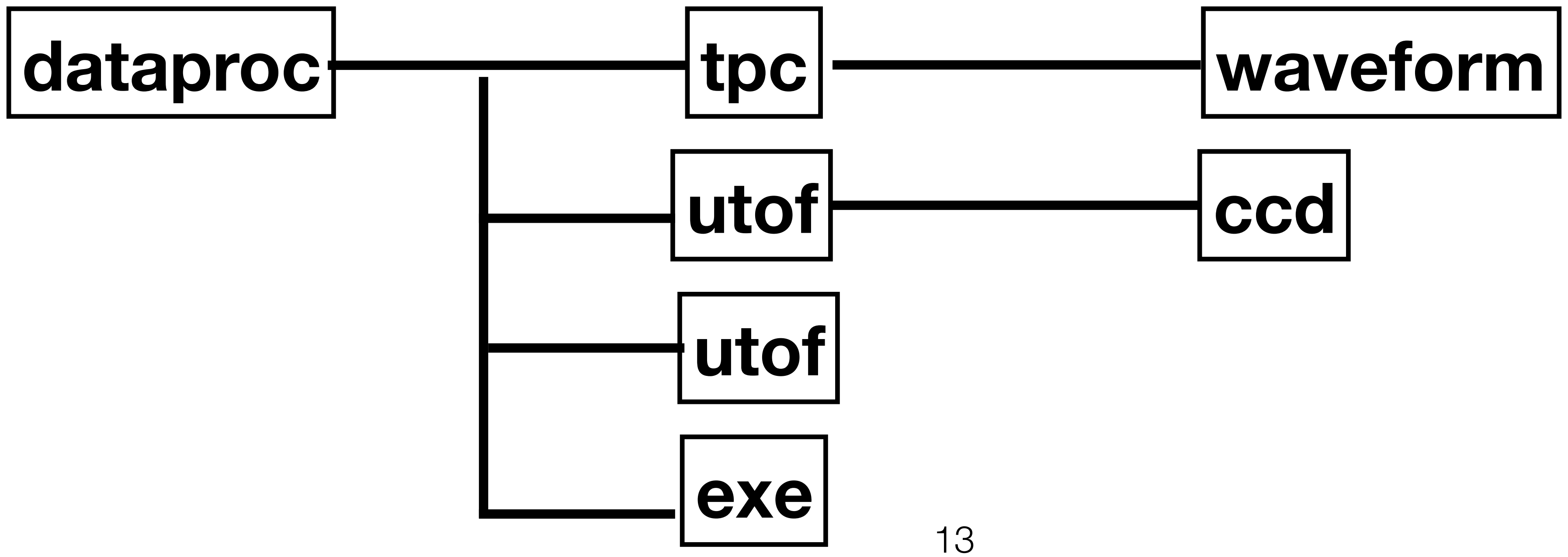
***Rules can be relaxed in executables**

How to create events from a dmtpc file?

- Works on RHUL linappservs only at the moment due to dmtpc dependency
- RAPtorr needs to be compiled with dmtpc. To do this, export an additional environment variable before running cmake
  - export DMTPCSYS=/scratch3/wparker2/hptpc_root6/hptpc-daq/
- Take a look in $RAPTORR_DIR/src/exe/testapp_tpcreadout_dmtpcfile.cfg
  - You need to use the brand new ‘tpcreadout’ source type
  - You also specify the name of the dmtpc file in there
    - No command line overrides yet, sorry!
- The top level class stored in the rapbase::Event is the rapobj::TPCReadout class
  - $RAPTORR_DIR/src/obj/TPCReadout.hxx

Hackathon contribs.

- RAPtorr core stuff is in a usable place so we can focus on physics
- We should build a set of libraries which contain useful classes/ routines for analysis, separated by detector type
- The general idea is that we form functions/classes/routines into libraries and then use those libraries in our executables
- I've produced a skeleton directory structure already



Summary

- RAPtorr is an event processing framework
    - Hides the nasties and lets you mostly focus on physics work
- RAPtorr now interfaces with dmtpc
- We should be in a good place to start coding up physics