ucesb unpacking

Bastian Löher
March 2017

TU Darmstadt

GSI Helmholtzzentrum für Schwerionenforschung GmbH

overview

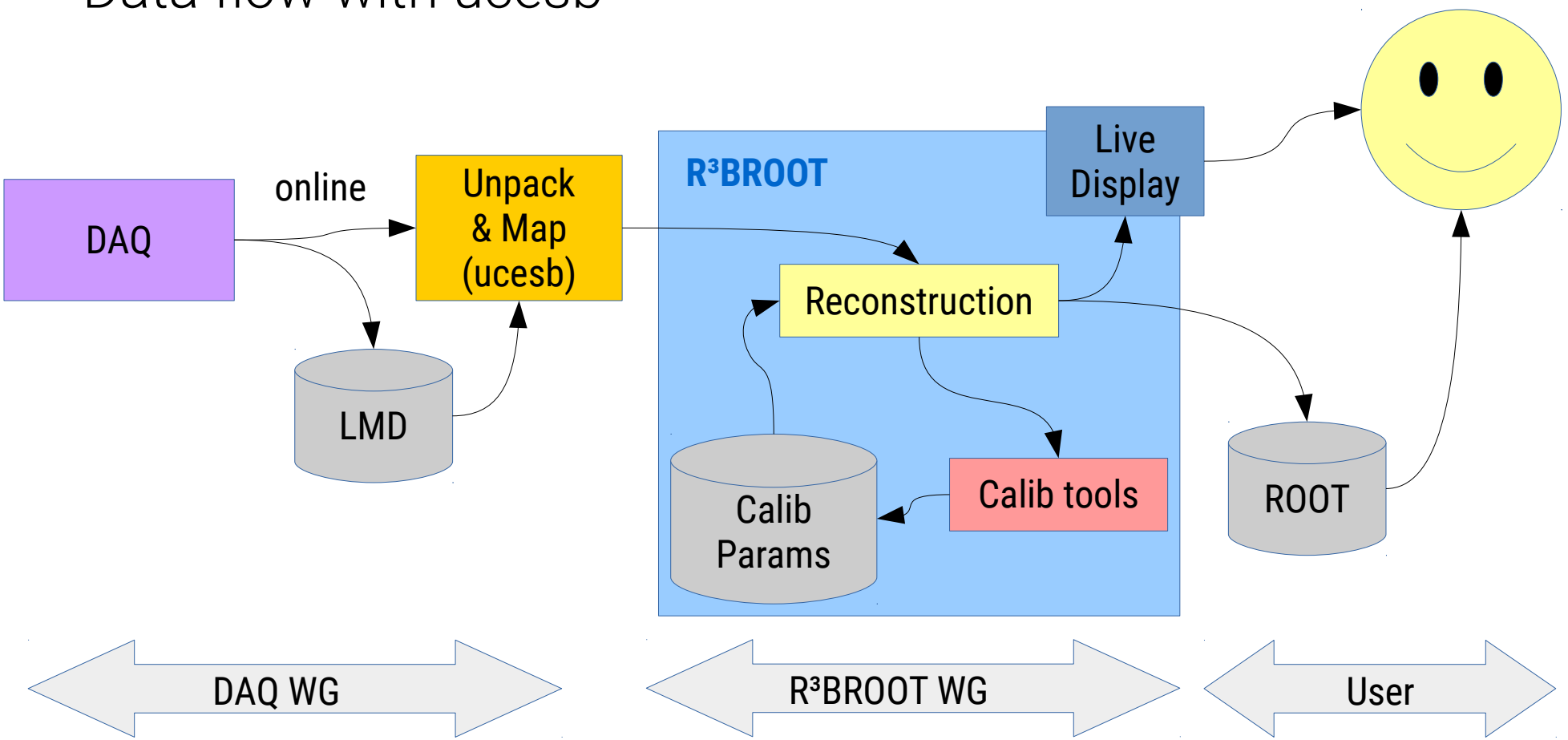
- Introduction to ucesb
- Spec file and mapping for PTOF detector
- Generating external client header file
- Using root_writer / struct_writer
- The R3BUcesbSource class
- Generic R3BReader class
- Specific R3BPtofReader class
- Writing a macro to test this

overview (if there is time)

- Reader class for event header information, i.e. TPAT, Event number, Timestamp, … (right now: R3BUnpackReader)
- Troubleshooting

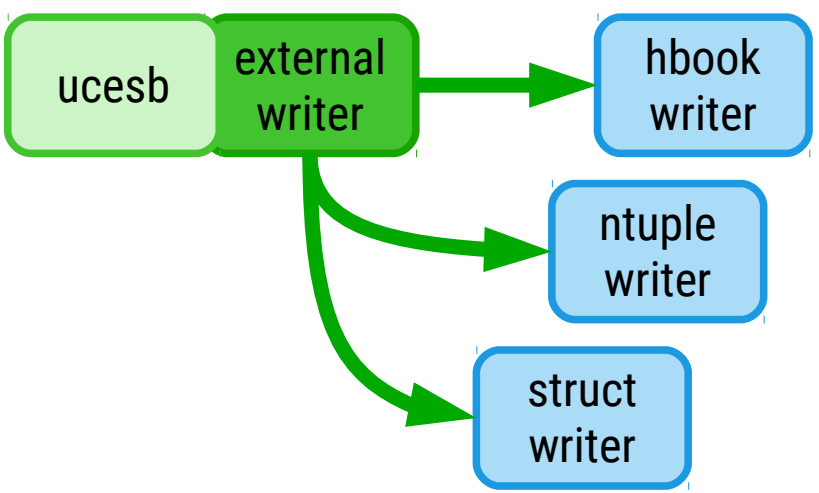
data flow

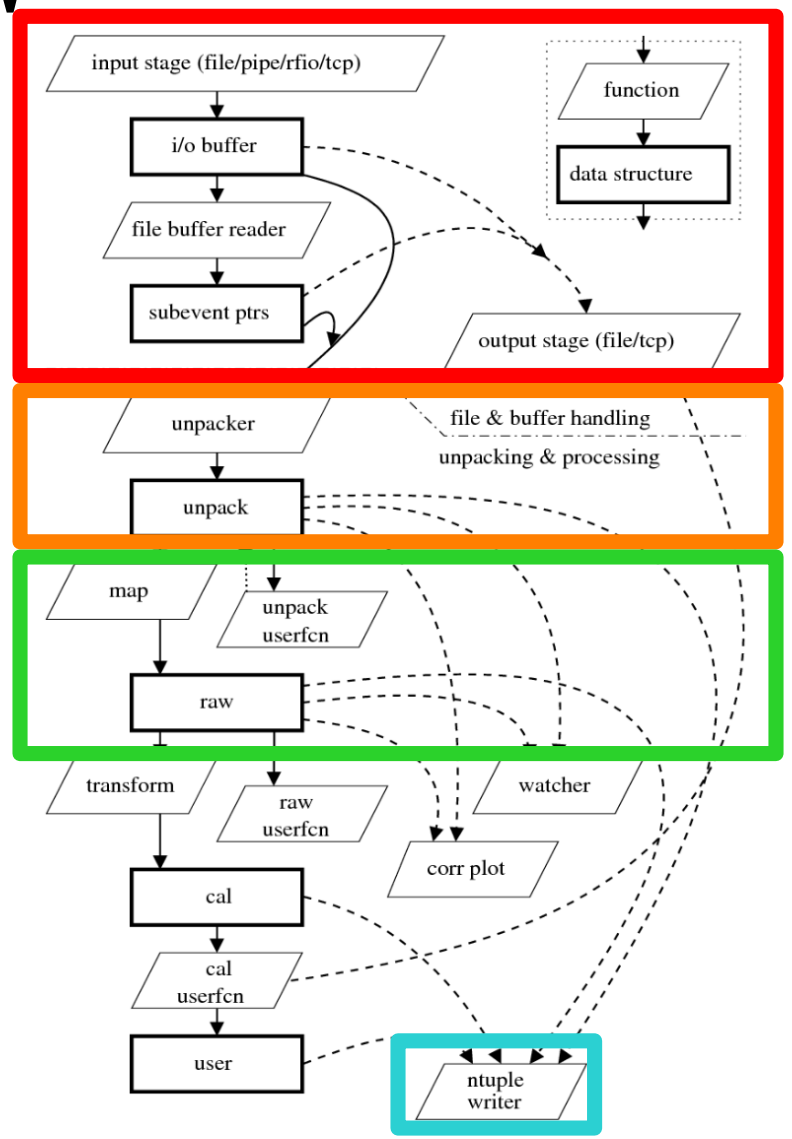
- Data flow with ucesb



ucesb overview

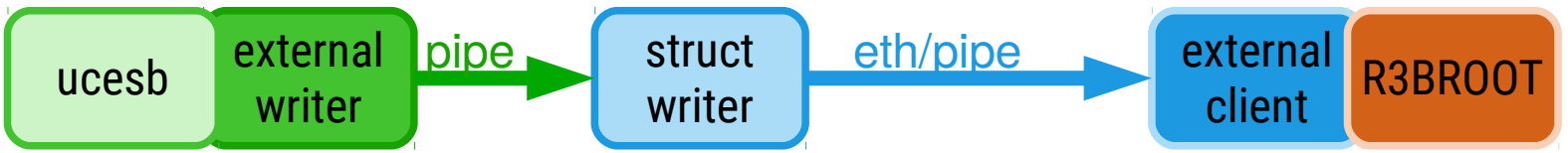
- Internal data flow
- Use input, unpacking and mapping stages
- Write output to file or via network





R3BROOT + ucesb

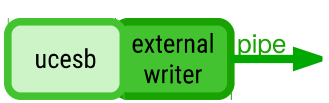
- Data flow



- Three steps:
  - Configure ucesb to unpack and map data
  - Configure external client to read ucesb output
  - Configure R3BRoot to use mapped data

PTOF detector

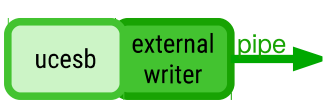
- Two plane scintillation bar detector with 6 bars in each plane
- Each bar is read out with a PMT on each end
- PMT signals are picked up by TAMEX frontend cards, and record the times of leading and trailing edges of each pulse
- Times are composed of a ‚coarse time' from an internal clock signal and a ‚fine time' from an FPGA delay-line TDC



Questions?

Hands-On 1

- Goal: Getting the unpacker ready for use with R3BROOT
- Necessary steps:
  - Update ucesb
  - Check out upexps (experiment specific unpackers)
  - Look at ucesb spec files, raw data and channel mapping
  - Unpack to a ROOT file



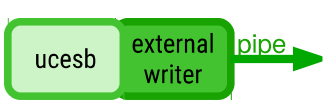
ucesb preparation

- Clone ucesb repository:

```
git clone http://fy.chalmers.se/~f96hajo/ucesb/ucesb.git
```

- OR: Update ‚ucesb‘ to latest revision:

```
cd ucesb
git update
```



ucesb preparation

- Check out ‚upexps‘ to workshop branch:

```
git clone lx-pool.gsi.de:/u/bloeher/git/upexps -b for_r3broot_worksho
mv for_r3broot_workshop upexps
```

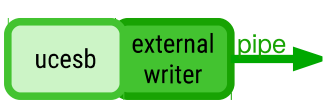
- OR: Fetch into existing upexps repository

```
cd upexps
git remote add bloeher land@lx-pool.gsi.de:/u/bloeher/git/upexps
git fetch bloeher
git checkout -b workshop bloeher/for_r3broot_workshop
```

ucesb preparation

- Make jun16/jun16_ptof unpacker

```
cd upexps/jun16
make -j8 jun16_ptof
```



ucesb help

- Display help output of ucesb:

```
./jun16_ptof
```

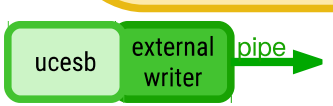
- For some specific options, more help exists:

```
./jun16_ptof --ntuple=help
./jun16_ptof --output=help
./jun16_ptof --server=help
```

ucesb unpacking

- Spec file (specification of data structure)
  - PTOF detector uses TAMEX readout electronics → need to use gsi_tamex.spec file for unpacking
- jun16.spec (main specification file)

```
EVENT {
  tofd_tamex = tofd_tamex_subev(…);
  ...
}
SUBEVENT(tofd_tamex_subev) {
  tamex_1 = TAMEX3_SFP(sfp=2, card=0);
  tamex_2 = TAMEX3_SFP(sfp=2, card=1);
  ...
}
```

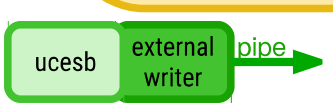


ucesb unpacking

- Spec file (specification of data structure)
  - PTOF detector uses TAMEX readout electronics → need to use gsi_tamex.spec file for unpacking
- jun16.spec (main specification file)

```
EVENT {
   tofd_tamex = tofd_tamex_subev(…);
   ...
}
SUBEVENT(tofd_tamex_subev) {
   tamex_1 = TAMEX3_SFP(sfp=2, card=0);
   tamex_2 = TAMEX3_SFP(sfp=2, card=1);
   ...
}
```

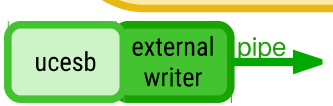
NOTE:
We are in fact using data collected with TOFD detector!
PTOF does not exist yet and will have similar readout.



ucesb unpacking

- LMD Subevents are matched based on 3 numbers
  - type, subtype, and control
  - these are defined in the DAQ during data taking

```
EVENT {
  tofd_tamex = tofd_tamex_subev(type=102,subtype=10200,control=4);
}
```



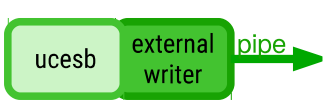
ucesb unpacking

- Check the raw data structure with ucesb

```
./jun16_ptof --print --max-events=1 file.lmd | less
```

- Print the raw data with ucesb

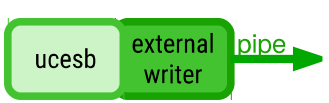
```
./jun16_ptof --print --data --max-events=1 file.lmd | less
```



ucesb unpacking

- Mapping file (jun16/mapping_ptof.hh)

```
// For each bar
SIGNAL(PTOF_P1T1_TFL1, tofd_tamex.tamex_3.time_fine[1], DATA12);
SIGNAL(PTOF_P1T1_TFT1, tofd_tamex.tamex_3.time_fine[2], DATA12);
SIGNAL(PTOF_P1T2_TFL1, tofd_tamex.tamex_3.time_fine[3], DATA12);
SIGNAL(PTOF_P1T2_TFT1, tofd_tamex.tamex_3.time_fine[4], DATA12);

// + coarse times...
```

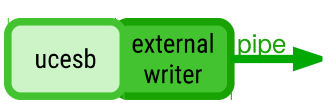


ucesb unpacking

- Mapping file (jun16/mapping_ptof.hh)

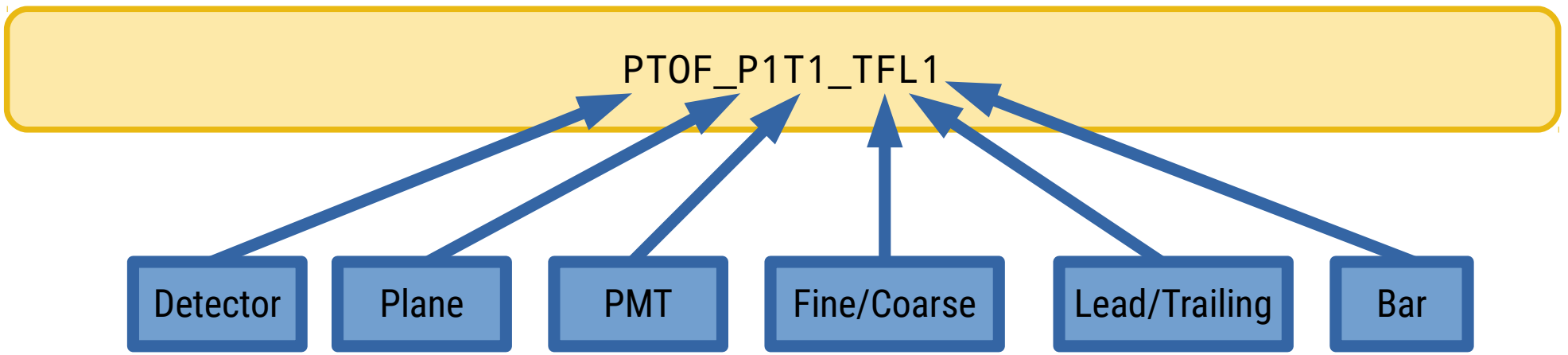
```
// For each bar
SIGNAL(PTOF_P1T1_TFL1, tofd_tamex.tamex
SIGNAL(PTOF_P1T1_TFT1, tofd_tamex.tamex
SIGNAL(PTOF_P1T2_TFL1, tofd_tamex.tamex
SIGNAL(PTOF_P1T2_TFT1, tofd_tamex.tamex

// + coarse times...
```

Your turn!
Implement mapping for plane 2 and the coarse times for PTOF!



ucesb unpacking

- Signal (branch) naming

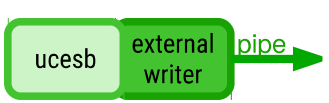


ucesb unpacking

- Defining zero suppression + multi-hit

```
// Put all bars into a zero-suppressed array
SIGNAL(ZERO_SUPPRESS_MULTI(32): PTOF_P1T1_TFL1);
SIGNAL(ZERO_SUPPRESS_MULTI(32): PTOF_P1T1_TFT1);
SIGNAL(ZERO_SUPPRESS_MULTI(32): PTOF_P1T1_TCL1);
SIGNAL(ZERO_SUPPRESS_MULTI(32): PTOF_P1T1_TCT1);
...
```

Always recompile unpacker after changing mapping or spec files!



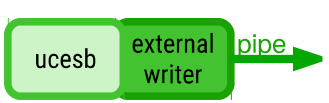
ucesb unpacking

- Defining zero suppression + multi-hit

```
// Put all bars into a zero-suppressed
SIGNAL(ZERO_SUPPRESS_MULTI(32): PTOF_P1
SIGNAL(ZERO_SUPPRESS_MULTI(32): PTOF_P1
SIGNAL(ZERO_SUPPRESS_MULTI(32): PTOF_P1
SIGNAL(ZERO_SUPPRESS_MULTI(32): PTOF_P1
...
```

```
Your turn!
Add zero suppression
for plane 2 and the
coarse times!
```

Always recompile unpacker after changing mapping or spec files!

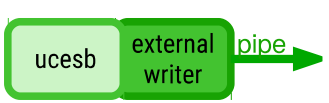


ucesb unpacking

- Checking, what is mapped:

```
./jun16_ptof --show-members
```

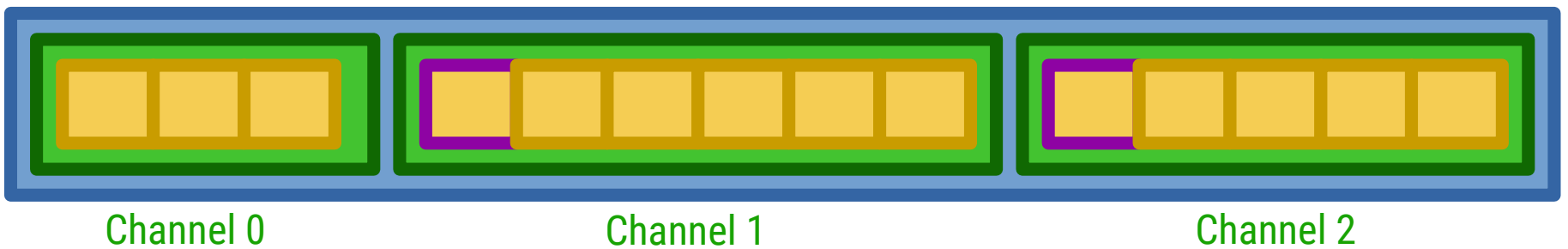
- Only, what is shown here, will end up in the root file!



ucesb unpacking

- ROOT branch names (ZERO_SUPPRESS_MULTI)

```
PTOF_P1T1_TFLM:     Number of channels with hits
PTOF_P1T1_TFLMI[]:  Indices of channels with hits
PTOF_P1T1_TFLME[]:  Index of first hit of next channel
PTOF_P1T1_TFL:      Number of hits in the array
PTOF_P1T1_TFLv[]:   Array of hits
```



ucesb unpacking

- Unpack to a ROOT file

```
./jun16_ptof input_file.lmd --ntuple=RAW,output.root
```

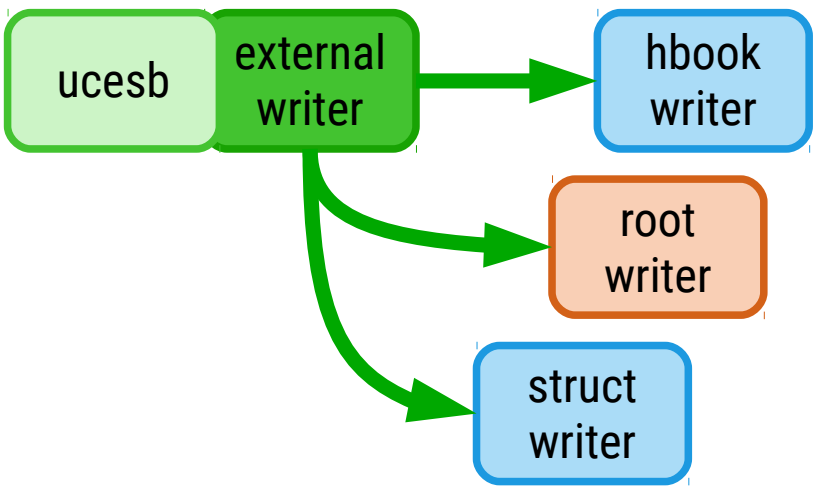
- Input files (jun16, run160 – run169):

```
/d/land2/bloeher/nyx_cache/jun2016/run*.lmd
```

Questions?

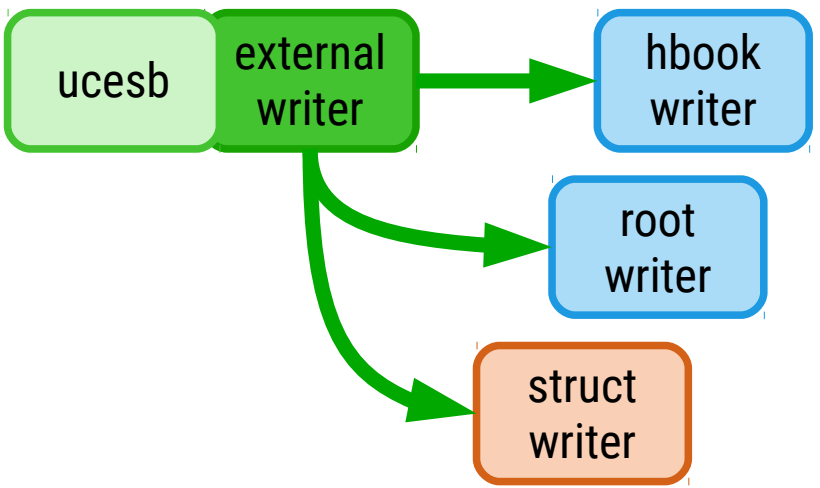
ucesb unpacking

- Ucesb output paths:



ucesb unpacking

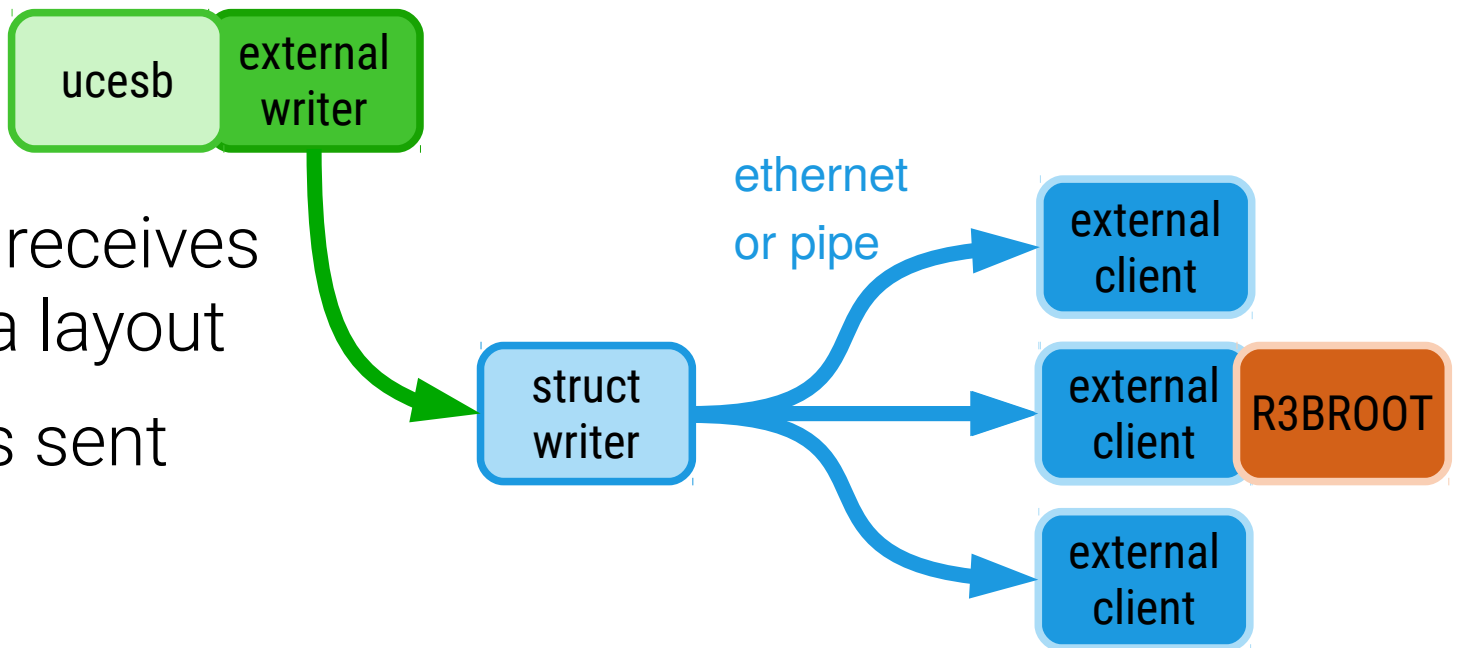
- Ucesb output paths:



see also: http://fy.chalmers.se/~f96hajo/shows/htj_may2015_struct_writer.pdf

ucesb unpacking

- Ucesb struct_writer output:



- Every client receives copy of data layout
- Then data is sent eventwise

ucesb unpacking

- Unpack to a struct_writer data stream and pipe to an external client

```
./jun16_ptof input_file.lmd --ntuple=RAW,STRUCT,- | ext_client
```

- external client needs to know about incoming data structure items → C header file generated from ucesb

ucesb unpacking

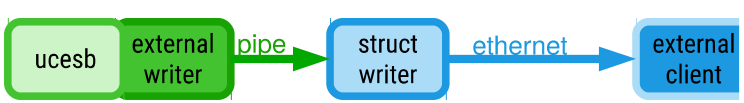
- Generating the structure information from ucesb

```
./jun16_ptof --ntuple=RAW,STRUCT_HH,ext_h101.h
```

- Generated file contains:
  - flat structure of detector data layout (EXT_STR_h101)
  - hierarchical (array) structure (EXT_STR_h101_onion)
  - macro to add data items to the structure information for external client setup, i.e. define what the client uses from the server (EXT_STR_h101_ITEMS_INFO)

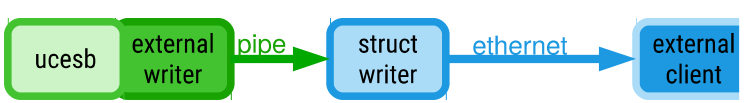
ucesb external client

- C and C++ API:
  - connect() to a struct_writer server on a specific IP:port or pipe
  - setup() the requested data from the server
  - fetch_event() loads the next event into the local buffer
  - close() the connection when done
- More information in (hbook/ext_data_client.h)
- This is what R3BUcesbSource uses!



Hands-On 2

- Goal: Generating ext_h101_ptof.h
- Necessary steps:
  - Generate ext_h101.h via ucesb
  - Modify ext_h101.h to include only what is needed for PTOF
- These steps have to be done every time the spec file or the mapping for a detector changes

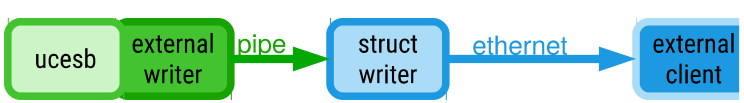


ucesb external client

- Generate the ext_h101.h from ucesb

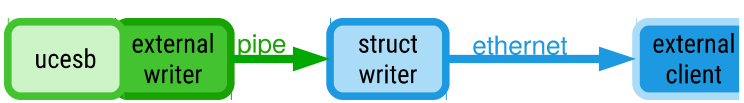
```
./jun16_ptof --ntuple=RAW:PTOF,STRUCT_HH,ext_h101.h
```

- Open the resulting file in an editor



ucesb external client

- Delete unneeded things:
  - Everything below the line
    „For internal use by the network data reader:“
    until the last „#endif“ line
- Rename the structures and the macro:
  - EXT_STR_h101 → EXT_STR_h101_PTOF
  - EXT_STR_h101_onion → EXT_STR_h101_PTOF_onion
  - EXT_STR_h101_ITEMS_INFO → EXT_STR_h101_PTOF_ITEMS_INFO

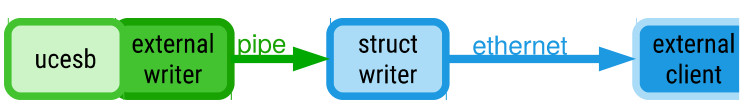


ucesb external client

- **OR:** Generate the ext_h101.h from ucesb

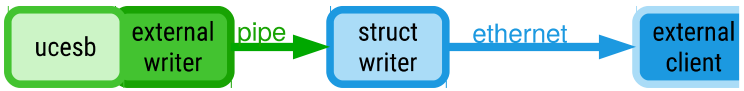
```
./jun16_ptof --ntuple=RAW:PTOF,STRUCT_HH,id=h101_PTOF,ext_h101.h
```

- Takes care of renaming automatically
- Only have to delete the last part



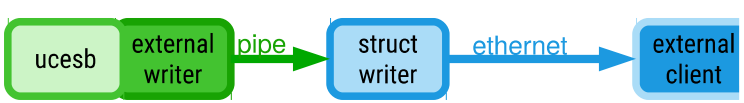
Questions?


Questions?


ucesb
external writer
pipe
struct writer
ethernet
external client

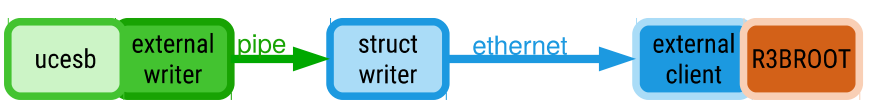
ucesb external client

- The resulting file can then be used by an external client to receive data from the struct_writer
- R3BRoot uses a single R3BUcesbSource class to manage a list of detector specific R3BReaders
- Each R3BReader has its own ext_h101_<det>.h structure information (look inside r3bsource/ for a list)
- An R3BReader should copy all data needed by the detector from the ucesb structure into R3BRoot data containers
- List of R3BReaders used for analysis is specified in the macro file



R3BRoot as external client

- The R3BUcesbSource class
  - Wraps around ext_data_clnt C++ API
- Tasks
  - Start the ucesb unpacker in struct_writer mode (fork)
  - Connect to the unpacker as a client
  - Setup the client to receive requested data items
  - Fetch events and copy data to local buffer
  - Report any errors encountered

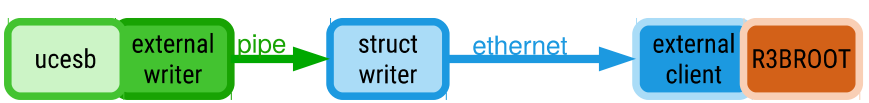


R3BRoot as external client

- R3BUcesbSource Constructor:

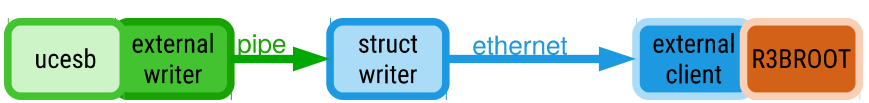
```
R3BUcesbSource(TString filename, TString ntuple_options, TString \
    ucesb_path, EXT_STR_h101 *event, size_t event_size)
```

- filename: Path to the LMD file
- ntuple_options: Additional options to –ntuple option
- ucesb_path: Path to ucesb executable
- event: Pointer to the full event structure
- event_size: Size of the full event structure



R3BRoot as external client

- R3BUcesbSource Init():
  - Forks a ucesb process and opens a pipe using popen() based on the arguments to the constructor
  - Connects to the server in the forked process
- R3BUcesbSource InitUnpackers():
  - Initialise each R3BReader in the list of Readers
- R3BUcesbSource ReadEvent():
  - Fetch data from ucesb
  - Run each detector specific R3BReader

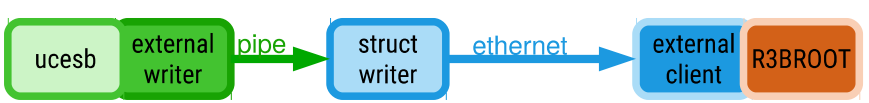


Questions?


ucesb
external writer
pipe
struct writer
ethernet
external client
R3BROOT

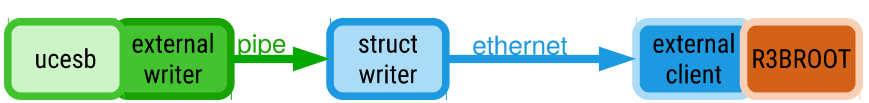
Hands-On 3

- Goal: Implement a reader class for PTOF detector
- Necessary steps:
  - New data container (or reuse an existing one)
  - New class deriving from R3BReader
  - Implement Init() function
  - Implement Read() function



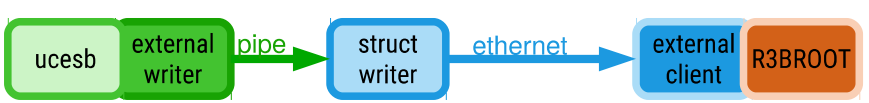
Tamex Mapped Data container

- Modify the existing r3bdata/neulandData/R3BPaddleTamexMappedData.h
- It already (on github R3BRoot/dev) handled data for a single PMT, but needed to be extended for two PMTs attached to the same paddle
- Dima‘s workshop repository already has version for two PMTs → quick walkthrough



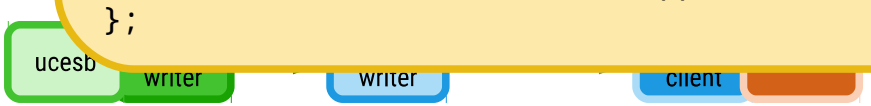
Tamex Mapped Data container

- Tasks
  - Add data members for second PMT
  - Simplify the constructor to only accept plane and bar number and set all members to zero
  - Adjust accessor functions



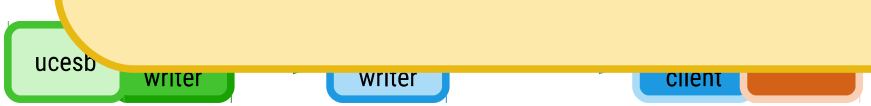
Tamex Mapped Data container - Outline

```
// r3bdata/neulandData/R3BPaddleTamexMappedData.h
#ifndef R3BPADDLETAMEXMAPPEDITEM_H
#define R3BPADDLETAMEXMAPPEDITEM_H

#include "TObject.h"

class R3BPaddleTamexMappedData : public TObject
{
  public:
    /* Default Constructor */
    R3BPaddleTamexMappedData();

    /* Standard Constructor */
    R3BPaddleTamexMappedData(Int_t planeId, Int_t barId);

    // Destructor
    virtual ~R3BPaddleTamexMappedData() {}

  protected:
    Int_t fPlane; //... number of plane 1..n
    Int_t fBar;   //... number of bar   1..n

  public:
    ClassDef(R3BPaddleTamexMappedData, 1)
};
```



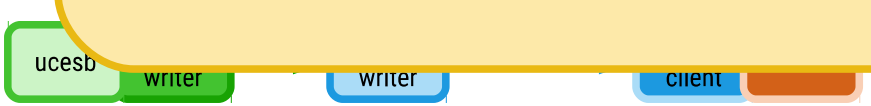
Tamex Mapped Data – Data members

```
// r3bdata/neulandData/R3BPaddleTamexMappedData.h

  protected:
    Int_t fPlane; //... number of plane 1..n
    Int_t fBar;   //... number of bar   1..n

  public:
    // PM1:
    Int_t fCoarseTime1LE;  //... coarse time of leading edge
    Int_t fFineTime1LE;    //... fine time of leading edge
    Int_t fCoarseTime1TE;  //... coarse time of trailing edge
    Int_t fFineTime1TE;    //... fine time of trailing edge
    // PM2:
    Int_t fCoarseTime2LE;  //... coarse time of leading edge
    Int_t fFineTime2LE;    //... fine time of leading edge
    Int_t fCoarseTime2TE;  //... coarse time of trailing edge
    Int_t fFineTime2TE;    //... fine time of trailing edge
```



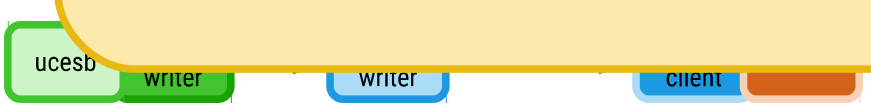
Tamex Mapped Data – Cxx file

```
#include "R3BPaddleTamexMappedData.h"

R3BPaddleTamexMappedData::R3BPaddleTamexMappedData()
    : fPlane(0), fBar(0)
    , fCoarseTime1LE(0), fFineTime1LE(0)
    , fCoarseTime1TE(0), fFineTime1TE(0)
    , fCoarseTime2LE(0), fFineTime2LE(0)
    , fCoarseTime2TE(0), fFineTime2TE(0)
{
}

R3BPaddleTamexMappedData::R3BPaddleTamexMappedData(Int_t planeId, Int_t barId)
    : fPlane(planeId), fBar(barId)
    , fCoarseTime1LE(0), fFineTime1LE(0)
    , fCoarseTime1TE(0), fFineTime1TE(0)
    , fCoarseTime2LE(0), fFineTime2LE(0)
    , fCoarseTime2TE(0), fFineTime2TE(0)
{
}

ClassImp(R3BPaddleTamexMappedData)
```



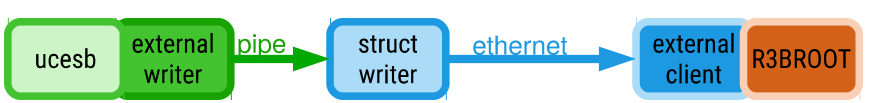
Tamex Mapped Data – Accessor functions

```
// Getters
inline const Int_t& GetPlaneId() const
{
    return fPlane;
}
inline const Int_t& GetBarId() const
{
    return fBar;
}
inline const Int_t& GetCoarseTime1LE() const
{
    return fCoarseTime1LE;
}

...

inline const Int_t& GetFineTime(int t,int e) const
{
    return t ? (e ? fFineTime2TE : fFineTime2LE) :
     (e ? fFineTime1TE : fFineTime1LE);
}
```



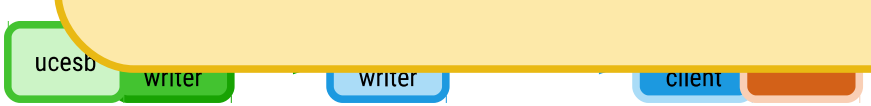
New Reader class

- R3BPtofReader, is initially derived from R3BTofdReader
- Needs to read data from PTOF data structure supplied by ucesb unpacker → Multi-Hit zero-suppressed arrays
- Must check that:
  - hits in both PMTs for the same paddle exist (within 100 ns), otherwise they don‘t belong to the same particle
  - leading and trailing edges for the same hit exist (within 5 us), otherwise the time over threshold is unreasonably high



R3BPtofReader - Outline

```
// r3bsource/R3BPtofReader.h
#ifndef R3BPTOFREADER_H
#define R3BPTOFREADER_H

#include "R3BReader.h"

class R3BPtofReader : public R3BReader
{
     public:
          R3BPtofReader(EXT_STR_h101_PTOF *, UInt_t);
          ~R3BPtofReader();
     private:
          /* Reader specific data structure from ucesb */
          EXT_STR_h101_PTOF* fData;
          /* Data offset */
          UInt_t fOffset;
          /* FairLogger */
          FairLogger*    fLogger;
          /* the structs of type R3BPtofxMappedItem */
          TClonesArray* fArray; /**< Output array. */
     public:
          ClassDef(R3BPtofReader, 0);
};
```



R3BPtofReader – Member functions

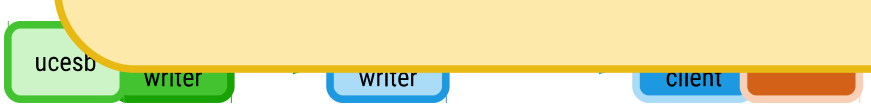
```
class R3BPtofReader : public R3BReader
{
    ...
    public:
          /* These are required by R3BReader */
          Bool_t Init(ext_data_struct_info *);
          Bool_t Read();
          void Reset();
    ...
    private:
          /* These are helper functions to maintain a nice code structure */
          Bool_t ReadLeadingEdges(EXT_STR_h101_PTOF_onion *, int, int);
          Bool_t ReadTrailingEdges(EXT_STR_h101_PTOF_onion *, int, int);
          Bool_t ReadLeadingEdgeChannel(EXT_STR_h101_PTOF_onion *, int,
              int, uint32_t, int);
          Bool_t ReadTrailingEdgeChannel(EXT_STR_h101_PTOF_onion *, int,
              int, uint32_t, int);
    ...
}
```

R3BPtofReader – Constructor / Destructor

```
//r3bsource/R3BPtofReader.cxx:
#include "TClonesArray.h"
#include "FairLogger.h"
#include "FairRootManager.h"
#include "R3BPtofReader.h"
#include "R3BPaddleTamexMappedData.h"

extern "C" {
#include "ext_data_client.h" /* Header describing the external client API from ucesb */
#include "ext_h101_ptof.h" /* ← This is the data structure layout header from ucesb */
}

R3BPtofReader::R3BPtofReader(EXT_STR_h101_PTOF* data, UInt_t offset)
     : R3BReader("R3BPtofReader")
     , fData(data)
     , fOffset(offset)
     , fLogger(FairLogger::GetLogger())
     , fArray(new TClonesArray("R3BPaddleTamexMappedData"))
{
}

R3BPtofReader::~R3BPtofReader()
{}
```

R3BPtofReader – Init() function 1

```
//r3bsource/R3BPtofReader.cxx:
#define MAX_PTOF_PLANES 1       /* Some constants used below */
#define N_TUBES_PER_PADDLE 2

Bool_t R3BPtofReader::Init(ext_data_struct_info *a_struct_info)
{
     /* Setup external client to request all PTOF-related data items */

     int ok;
     EXT_STR_h101_PTOF_ITEMS_INFO(ok, *a_struct_info, fOffset, EXT_STR_h101_PTOF, 0);
     if (!ok) {
          perror("ext_data_struct_info_item");
          fLogger->Error(MESSAGE_ORIGIN, "Failed to setup structure information.");
          return kFALSE;
     }

     return kTRUE;
}
```



R3BPtofReader – Init() function

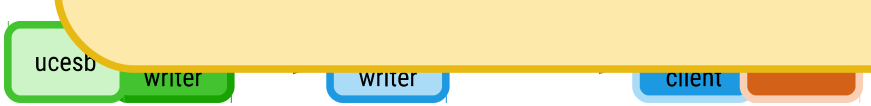
```
//r3bsource/R3BPtofReader.cxx:

Bool_t R3BPtofReader::Init(ext_data_struct_info *a_struct_info)
{
     ...

     // Register output array in tree
     FairRootManager::Instance()->Register("PtofMapped", "Land", fArray, kTRUE);

     // initial clear (set number of hits to 0)
     EXT_STR_h101_PTOF_onion* data = (EXT_STR_h101_PTOF_onion*)fData;
     for (int d = 0; d < MAX_PTOF_PLANES; d++) {
          for (int t = 0; t < N_TUBES_PER_PADDLE; t++) {
               data->PTOF_P[d].T[t].TFLM = 0;
               data->PTOF_P[d].T[t].TFTM = 0;
          }
     }
     return kTRUE;
}
```

R3BPtofReader – Read() function

```
//r3bsource/R3BPtofReader.cxx:

Bool_t R3BPtofReader::Read()
{
     // Convert plain raw data to multi-dimensional array
     EXT_STR_h101_PTOF_onion* data = (EXT_STR_h101_PTOF_onion*)fData;

     // Loop over detector planes and PM tubes
     // Then read the leading edges, and the trailing edges
     for (int d = 0; d < MAX_PTOF_PLANES; d++)
     {
          for (int t = 0; t < N_TUBES_PER_PADDLE; t++)
          {
               ReadLeadingEdges(data, d, t);
               ReadTrailingEdges(data, d, t);
          }
     }
}
```

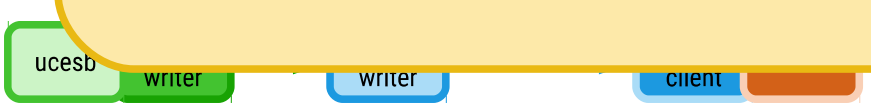
R3BPtofReader – ReadLeadingEdges() function

```
//r3bsource/R3BPtofReader.cxx:
Bool_t R3BPtofReader::ReadLeadingEdges(EXT_STR_h101_PTOF_onion *data, int d, int t)
{
     // # of channels with data. not necessarly number of hits! (b/c multi hit)
     uint32_t numChannels = data->PTOF_P[d].T[t].TFLM;

     // loop over channels, index in v for first item of current channel
     uint32_t curChannelStart = 0;
     for (int i = 0; i < numChannels; i++)
     {
          uint32_t bar;                       // bar number
          uint32_t nextChannelStart;          // index in v for first item of next channel

          bar = data->PTOF_P[d].T[t].TFLMI[i];
          nextChannelStart = data->PTOF_P[d].T[t].TFLME[i];

          for (int j = curChannelStart; j < nextChannelStart; j++) {
               ReadLeadingEdgeChannel(data, d, t, bar, j);
          }

          curChannelStart = nextChannelStart;
     }
     return kTRUE;
}
```



R3BPtofReader – ReadLeadingEdgeChannel()

```
//r3bsource/R3BPtofReader.cxx:
#define MAX_TIME_DIFF_PADDLE_PMT 20 /* 20 * 5 ns = 100 ns */
Bool_t R3BPtofReader::ReadLeadingEdgeChannel(EXT_STR_h101_PTOF_onion *data, int d, int t,
    uint32_t bar, int ch)
{
     R3BPaddleTamexMappedData* mapped = NULL;

     mapped = new ((*fArray)[fArray->GetEntriesFast()])
          R3BPaddleTamexMappedData(d + 1, bar); // plane, bar

     /* Fill leading edge time members */
     if (t == 0)
     {
          // PM1
          mapped->fCoarseTime1LE = data->PTOF_P[d].T[t].TCLv[ch];
          mapped->fFineTime1LE   = data->PTOF_P[d].T[t].TFLv[ch];
     } else {
          // PM2
          mapped->fCoarseTime2LE = data->PTOF_P[d].T[t].TCLv[ch];
          mapped->fFineTime2LE   = data->PTOF_P[d].T[t].TFLv[ch];
     }
     return kTRUE;
}
```



R3BPtofReader – ReadLeadingEdgeChannel()

```
//r3bsource/R3BPtofReader.cxx:
#define MAX_TIME_DIFF_PADDLE_PMT 20 /* 20 * 5 ns = 100 ns */
Bool_t R3BPtofReader::ReadLeadingEdgeChannel(EXT_STR_h101_PTOF_onion *data, int d, int t,
    uint32_t bar, int ch)
{
     R3BPaddleTamexMappedData* mapped = NULL;

     mapped = new ((*fArray)[fArray->GetEntriesFast()])
          R3BPaddleTamexMappedData(d + 1, bar); // plane, b

     /* Fill leading edge time members */
     if (t == 0)
     {
          // PM1
          mapped->fCoarseTime1LE = data->PTOF_P[d].T[t].TC
          mapped->fFineTime1LE   = data->PTOF_P[d].T[t].TF
     } else {
          // PM2
          mapped->fCoarseTime2LE = data->PTOF_P[d].T[t].TCL
          mapped->fFineTime2LE   = data->PTOF_P[d].T[t].TFLv[c
     }
     return kTRUE;
}
```

WAIT!
We would like to join the hits in the same paddle!

R3BPtofReader – ReadLeadingEdgeChannel()

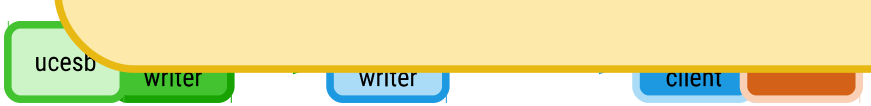
```
//r3bsource/R3BPtofReader.cxx:
Bool_t R3BPtofReader::ReadLeadingEdgeChannel(EXT_STR_h101_PTOF_onion *data, …) {

     if (t == 1) {
          int n = fArray->GetEntriesFast();
          int coarse = data->PTOF_P[d].T[t].TCLv[ch];

          for (int k = 0; k < n; k++) {
               R3BPaddleTamexMappedData* hit = (R3BPaddleTamexMappedData*)fArray->At(k);

               /* if leading time1 within 100ns window and time2 not yet set */
               if ((hit->fCoarseTime2LE == 0)
                   && (abs(hit->fCoarseTime1LE - coarse) <= MAX_TIME_DIFF_PADDLE_PMT)) {
                    mapped = hit;
                    break;
               }
          }
     }

     if (!mapped) mapped = new ((*fArray)[fArray->GetEntriesFast()])
          R3BPaddleTamexMappedData(d + 1, bar); // plane, bar
     /* Otherwise mapped is the hit containing the matching time1 */
     ...
}
```

R3BPtofReader – ReadTrailingEdges() function

```
//r3bsource/R3BPtofReader.cxx:
Bool_t R3BPtofReader::ReadTrailingEdges(EXT_STR_h101_PTOF_onion *data, int d, int t) {
     uint32_t numChannels = data->PTOF_P[d].T[t].TFTM;

     // loop over channels
     // index in v for first item of current channel
     uint32_t curChannelStart = 0;
     for (int i = 0; i < numChannels; i++)
     {
          uint32_t bar;
          uint32_t nextChannelStart;
          nextChannelStart = data->PTOF_P[d].T[t].TFTME[i];
          bar = data->PTOF_P[d].T[t].TFTMI[i];

          for (int j = curChannelStart; j < nextChannelStart; j++)
          {
               ReadTrailingEdgeChannel(data, d, t, bar, j);
          }

          curChannelStart = nextChannelStart;
     }
     return kTRUE;
}
```

R3BPtofReader – ReadTrailingEdgeChannel()

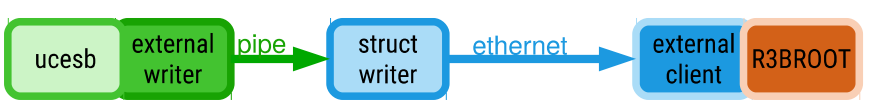
```
//r3bsource/R3BPtofReader.cxx:
#define MAX_TIME_OVER_THRESHOLD 1000 /* 1000 * 5 ns = 5 us */
Bool_t R3BPtofReader::ReadTrailingEdgeChannel(EXT_STR_h101_PTOF_onion *data,
    int d, int t, uint32_t bar, int ch)
{
     R3BPaddleTamexMappedData* mapped = NULL;
     mapped = new ((*fArray)[fArray->GetEntriesFast()])
          R3BPaddleTamexMappedData(d + 1, bar); // plane, bar

     if (t == 0)
     {
          // PM1
          mapped->fCoarseTime1TE= data->PTOF_P[d].T[t].TCTv[ch];
          mapped->fFineTime1TE  = data->PTOF_P[d].T[t].TFTv[ch];
     } else {
          // PM2
          mapped->fCoarseTime2TE= data->PTOF_P[d].T[t].TCTv[ch];
          mapped->fFineTime2TE  = data->PTOF_P[d].T[t].TFTv[ch];
     }
     return kTRUE;
}
```



R3BPtofReader – ReadTrailingEdgeChannel()

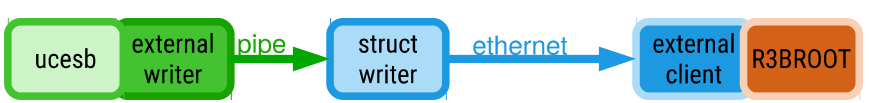
```
//r3bsource/R3BPtofReader.cxx:
Bool_t R3BPtofReader::ReadTrailingEdgeChannel(EXT_STR_h101_PTOF_onion *data, ...)
{
     int n = fArray->GetEntriesFast();
     int coarse = data->PTOF_P[d].T[t].TCTv[ch];

     // distinguish between PM1 and PM2
     if (t == 0) {
          for (int k = 0; k < n; k++)
          {
               R3BPaddleTamexMappedData* hit = (R3BPaddleTamexMappedData*)fArray->At(k);
               int tot = coarse - hit->fCoarseTime1LE;
               if ((tot <= MAX_TIME_OVER_THRESHOLD) && (tot >= 0)
                   && (hit->fCoarseTime1TE == 0)  /* no trailing */
                   && (hit->fCoarseTime1LE != 0)) /* has leading */
               {
                    mapped = hit;
                    break;
               }
          }
     } else { /* PM2 is treated similarly */ }

     if (!mapped) mapped = new ((*fArray)[n]) R3BPaddleTamexMappedData(d + 1, bar);
}
```

Questions?


ucesb
external writer
pipe
struct writer
ethernet
external client
R3BROOT

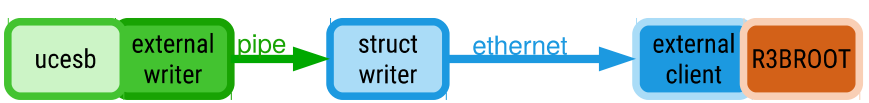
Writing a macro

- We will produce a macro reading the LOS detector and the PTOF detector
- It will:
  - add the R3BUcesbSource as data input
  - run the Reader classes to extract data
  - run the LosMapped2Cal task to produce calibrated Los data
- Check out a template of the macro from, or download:
  - /u/bloeher/git/R3BRoot-macros, branch ‚for_r3broot_workshop‘
  - http://web-docs.gsi.de/~bloeher/data/unpack_ptof_run160.C



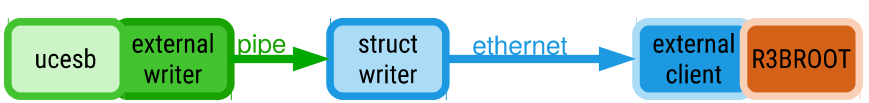
PtofReader Macro

```
/*
 * TODO:
 * Add the PTOF data structure to the full EXT_STR_h101_t structure
 */
struct EXT_STR_h101_t
{
     EXT_STR_h101_unpack_t unpack;
     EXT_STR_h101_LOS_t los;
     EXT_STR_h101_PTOF_t ptof;
};
```



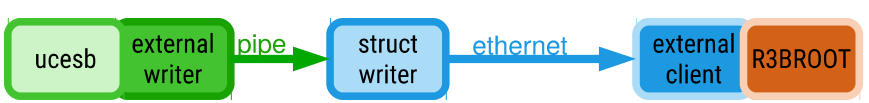
PtofReader Macro

```
/*
 * TODO:
 * Add the R3BPtofReader
 */
source->AddReader(new R3BPtofReader(
    (EXT_STR_h101_PTOF_t *)&ucesb_struct.ptof,
    offsetof(EXT_STR_h101, ptof)));
```



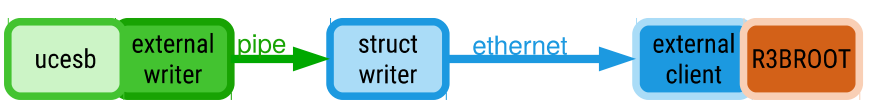
PtofReader Macro

```
/*
 * TODO:
 * Add the ptof_time_params_run165.root file.
 */
parList->Add(new TObjString("parameter/ptof_time_params_run165.root"));
```



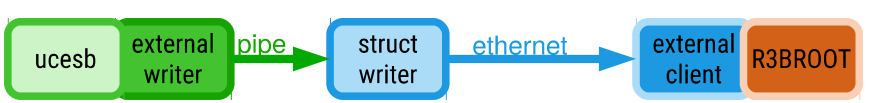
PtofReader Macro

```
/*
 * TODO:
 * Add the PtofTCalPar container
 */
rtdb1->getContainer("PtofTCalPar");
rtdb1->setInputVersion(RunId, (char*)"PtofTCalPar", 1, 1);
```



PtofReader Macro

```
/*
 * TODO:
 * Add the R3BPtofMapped2Cal task
 */
R3BPtofMapped2Cal* ptofMapped2Cal =
    new R3BPtofMapped2Cal("PtofTCalPar", 1);
run->AddTask(ptofMapped2Cal);
```

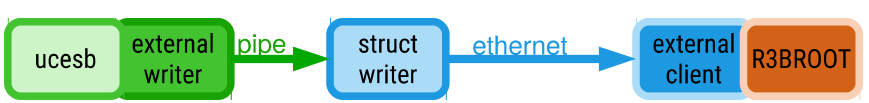


PtofReader Macro

- Run the macro

```
root -l unpack_ptof_run160.C
```

- Done! That‘s it =)



Questions?


KEEP
CALM
AND
ASK
QUESTIONS


ucesb
external writer
pipe
struct writer
ethernet
external client
R3BROOT