PandaX Data Model for Reconstruction

Xun Chen (谌勋)

INPAC, Department of Physics and Astronomy

上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

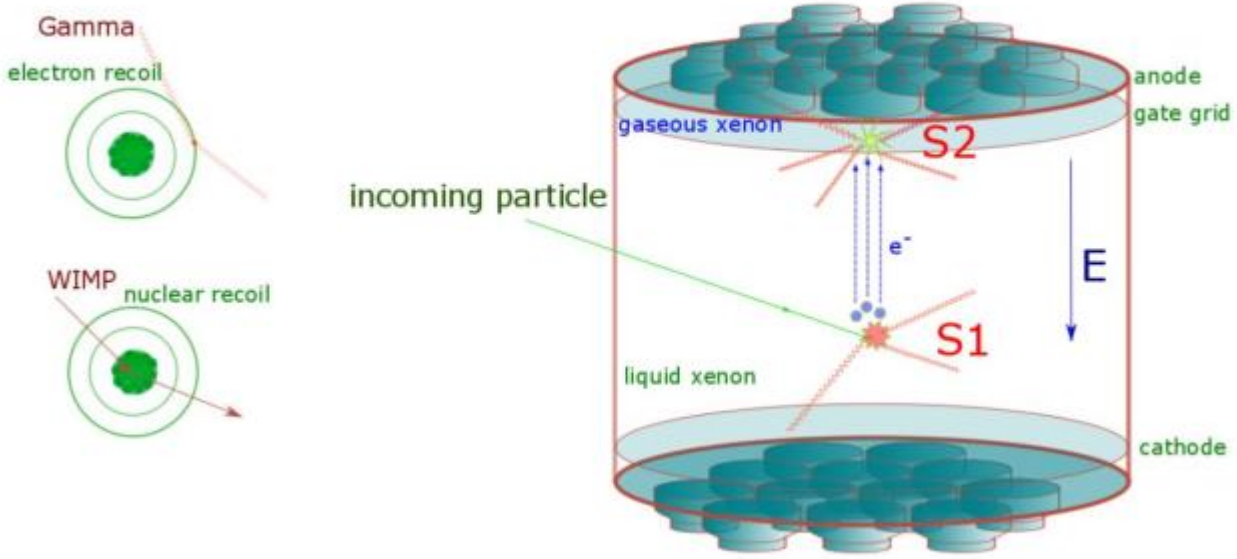
Sep 13, 2016

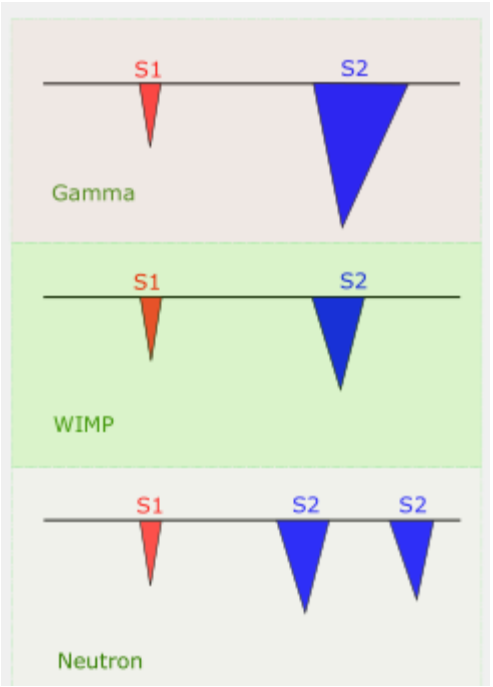
Outline

- Introduction
- Data model and bamboo-shoot2
- Reconstruction overview
- New data model based on bamboo-shoot3
- Summary

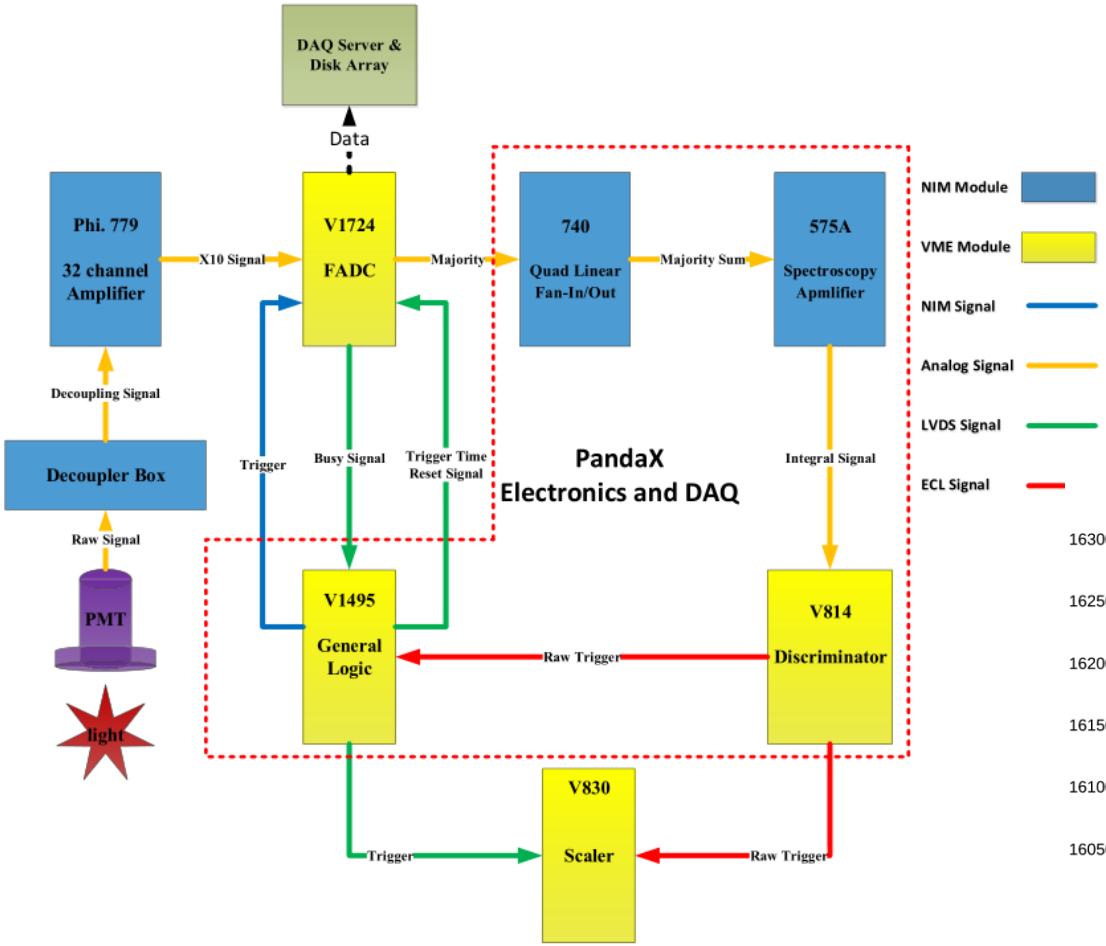
Introduction

PandaX detector detect the primary scintillation **S1** and the proportional scintillation **S2** with two **PMT** arrays.

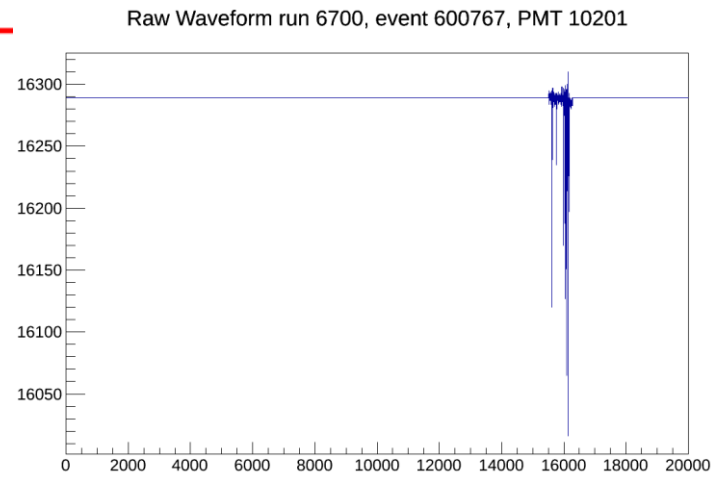




The Raw Data



The raw waveforms from each of the PMT have been recorded in binary format.



Raw Data Rate of PandaX-II

- Small number of channels with large sample number
  - 55x2 3-inch PMTs (R11410) + 24x2 1-inch PMTs (R8520) ➔ 158 PMTs (channels)
  - Sample window width: 1ms, sampling rate: 100MHz ➔ 100,000 samples in each window
- Simple estimation
  - 16bitx100000x158 ➔ 30MB/event
  - Zero-Length-Encoding (ZLE) makes the data much smaller ~ 1MB/event
- Actual Data Rate
  - DM search data: 1GB per 2 minutes, ~ 3.5Hz (decreasing over time due to the decay of ^{127}Xe)
  - Low rate neutron calibration data: 1GB per 1.5 minutes ~ 5Hz
  - CH_3D ER calibration data: 1GB per 2 minutes ~ 3

More than 150T raw data have been collected by PandaX-II so far.

Need to carry data from Jinping back to SJTU.

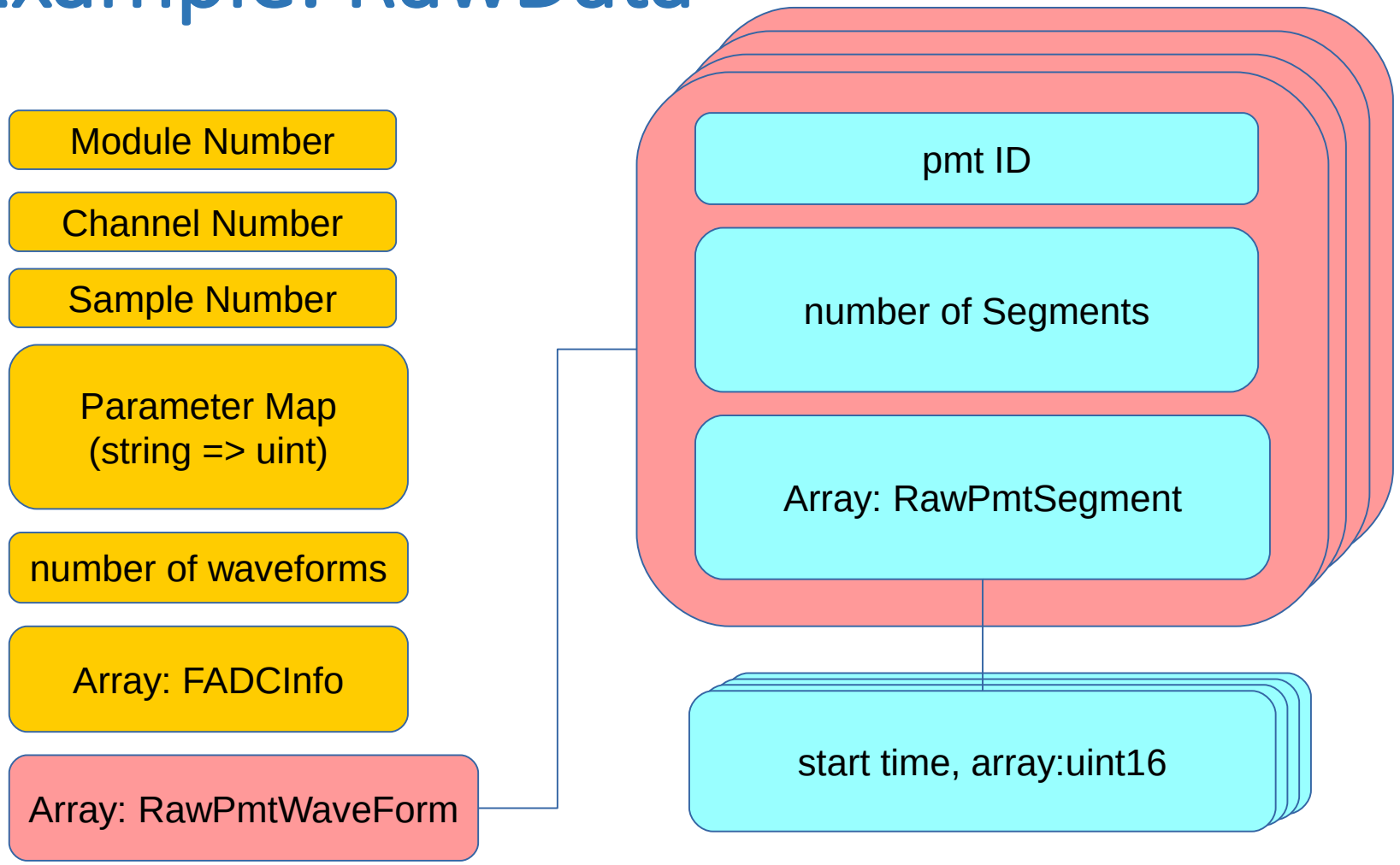
Requirement for Data Model

- Need to define the data contents in each step of the reconstruction.
  - Reconstruction algorithm will be developed based on the data contents in each step
- Need to find a way for the data persistency.
  - Speed.
  - Size.
  - Stability.

bamboo-shoot2

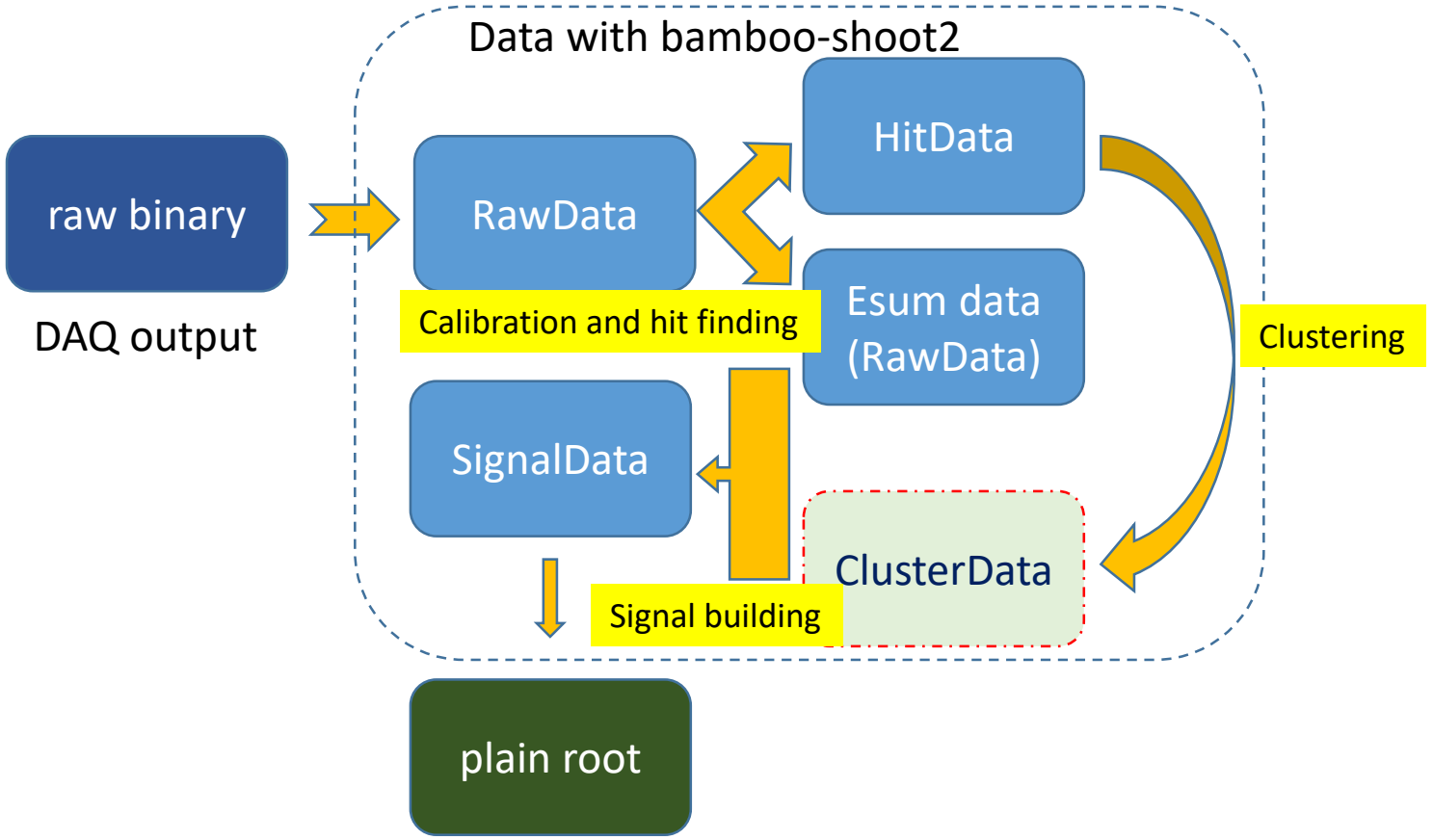
- Current default data model for PandaX-I and PandaX-II
- Based on ROOT TTree - ROOT serves as the persistency layer.
- Mapping Customized TObject to the Branch
- Top level Objects:
  - RawData
  - HitData
  - ClusterData
  - SignalData

Example: RawData



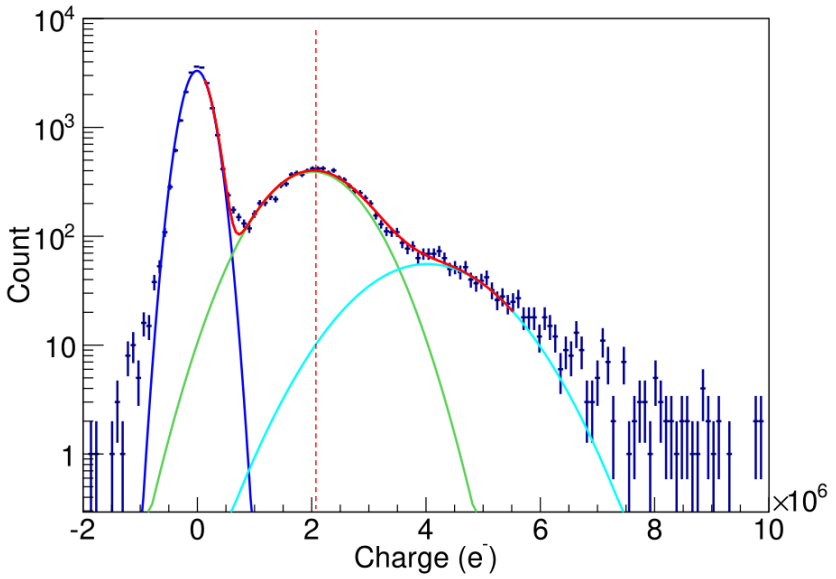
Array = TClonesArray

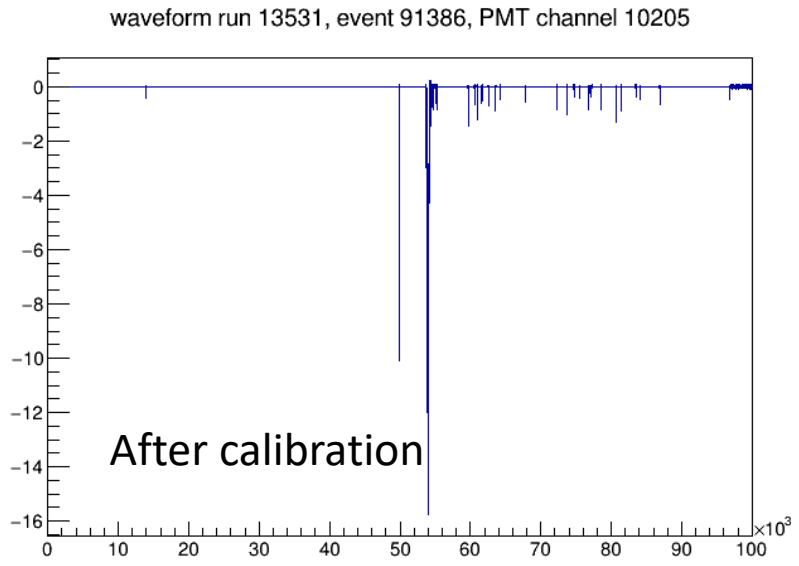
Reconstruction Flow



Calibration

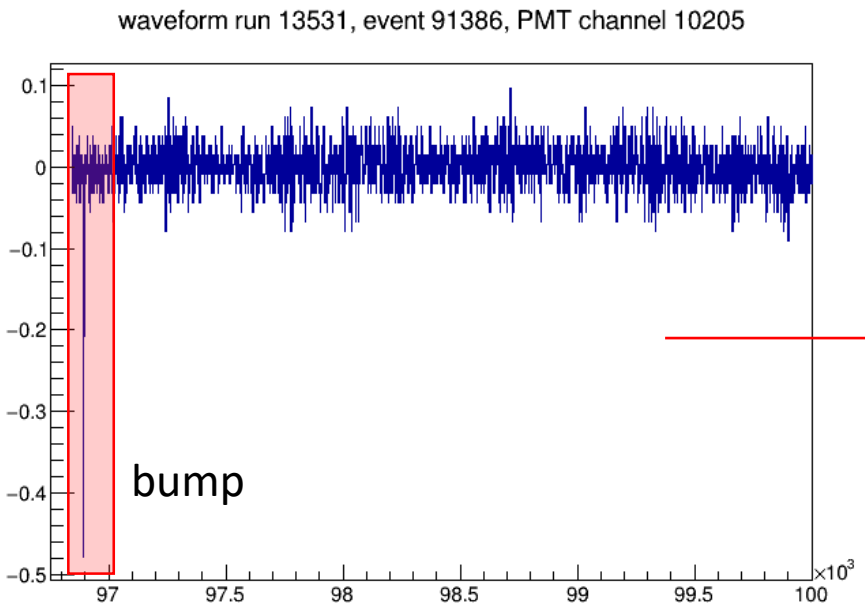
- The PMT signal unit: PE (photoelectron)
- Gain from LED calibration
- Calibrated value = (ADC - baseline)/gain
- Baseline is calculated as the start 40 samples in each segment.

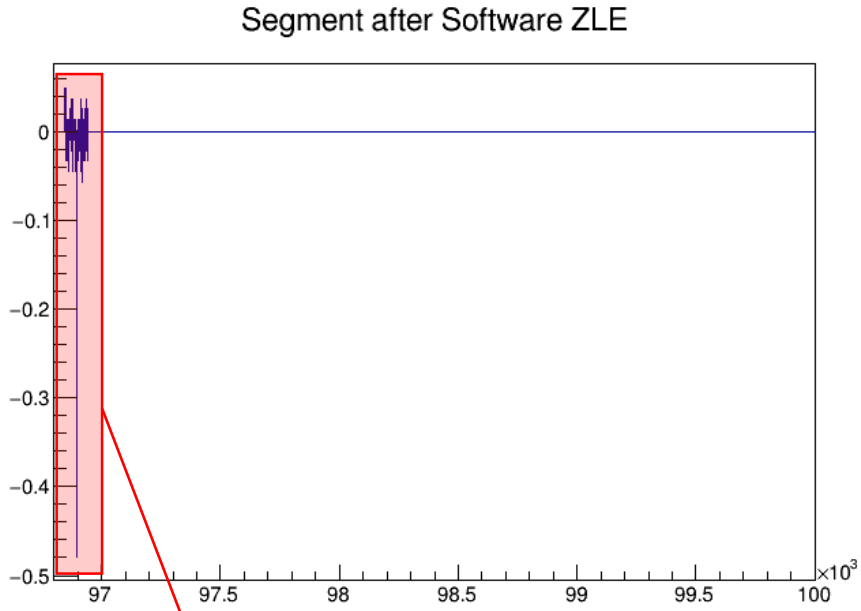


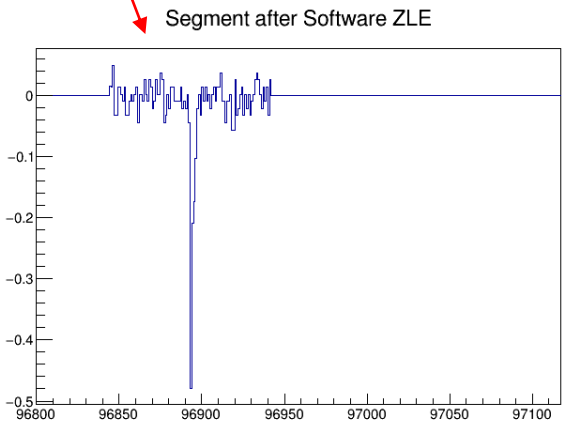


Software ZLE

Sometimes there may be shift of the baseline and the hardware ZLE does not work well



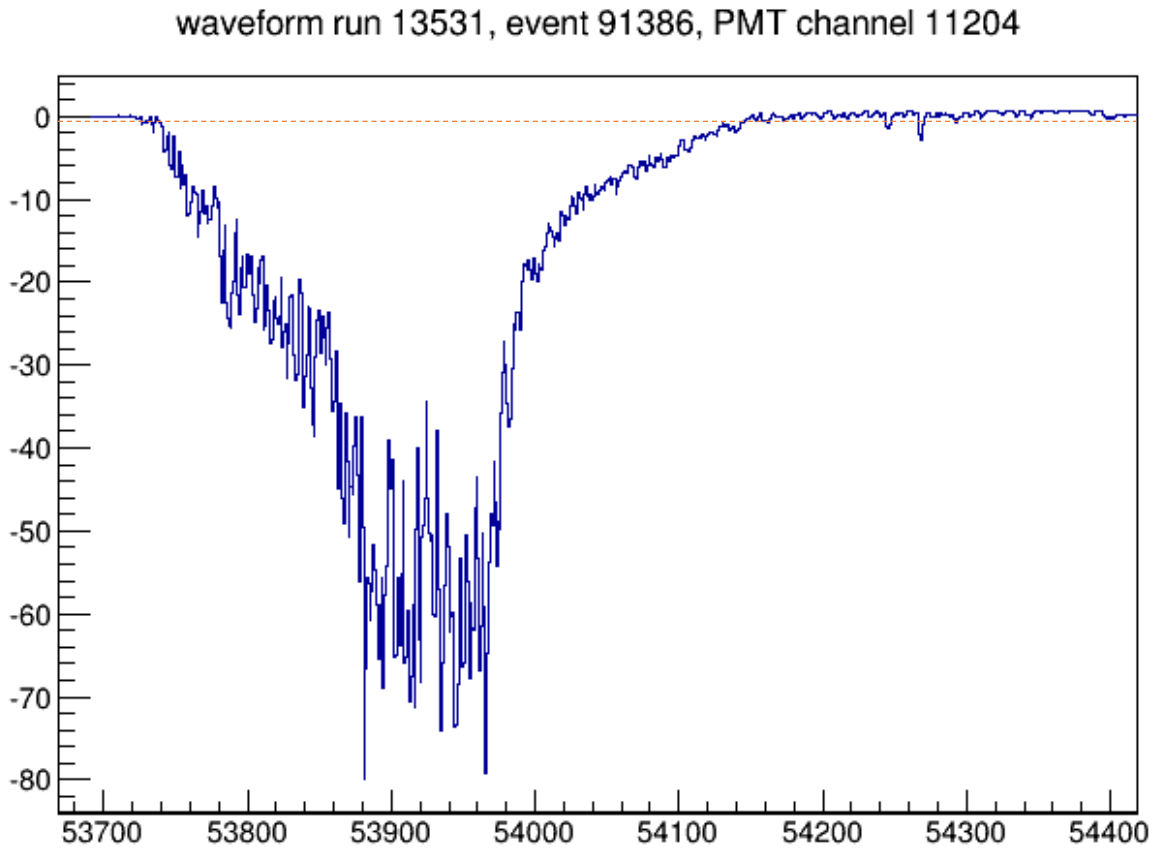




Non-bump samples have been suppressed.

Hit Finding

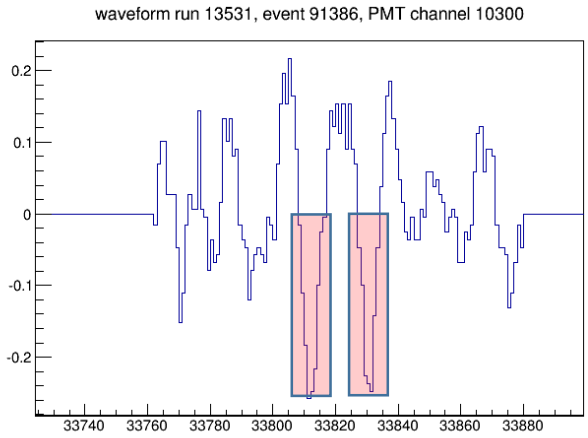
- Find out the bumps over threshold in each channel



Threshold
0.25

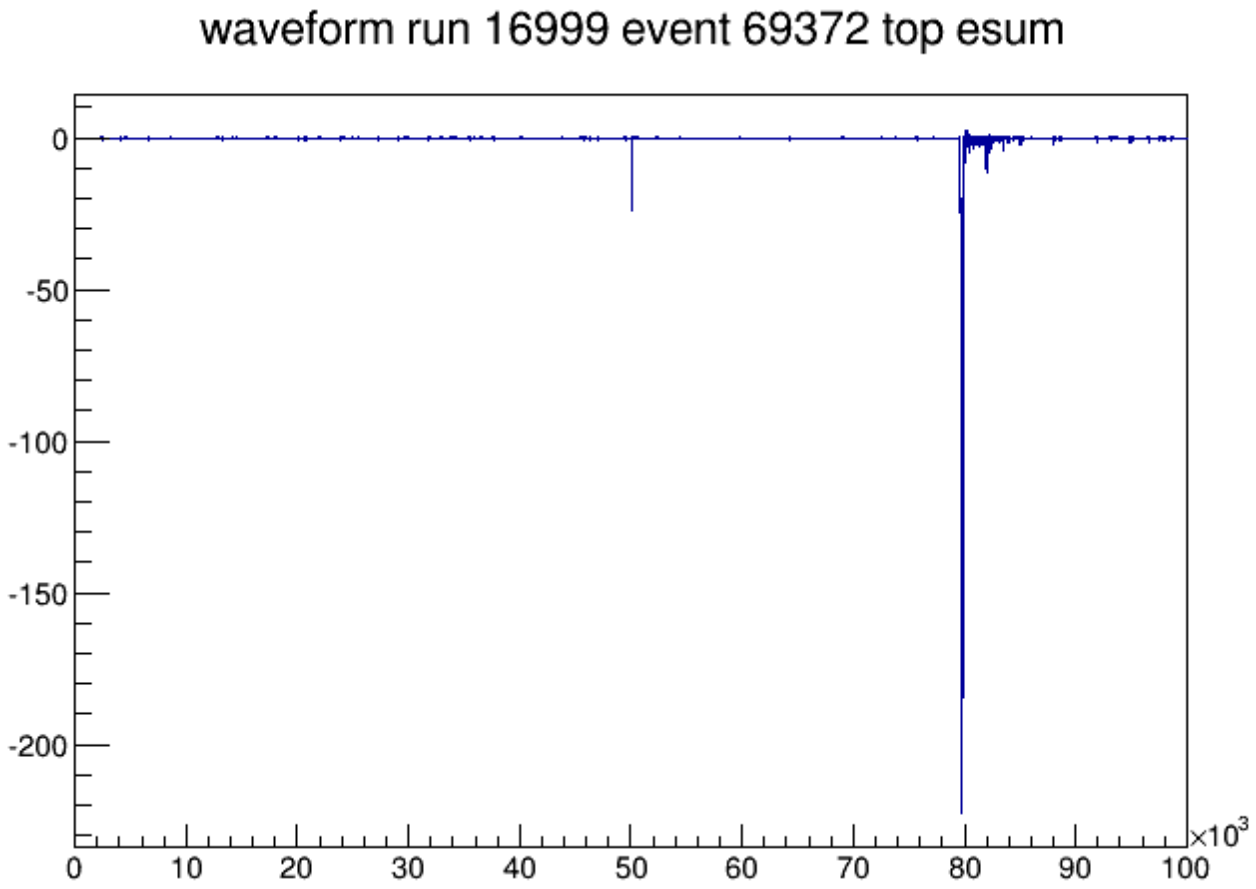
Hit Type Recognition

- Noise Hit: positive sample value exceeding half of the hit height was found in a window 30 samples before/after the hit.
- Saturation Hit: continuous number of samples with maximum value is larger than 3.
- Normal Hit: others.



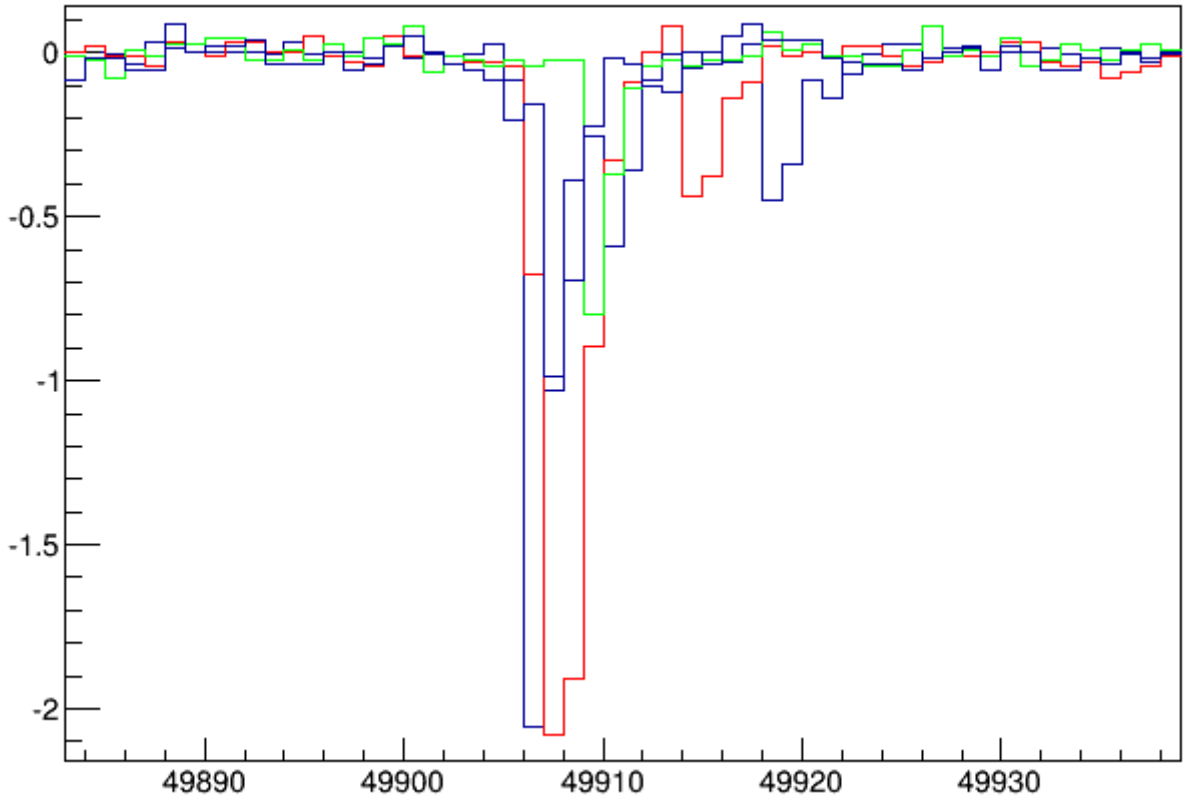
Two noise hits.

Esum - Sum of Waveforms in Different PMTs



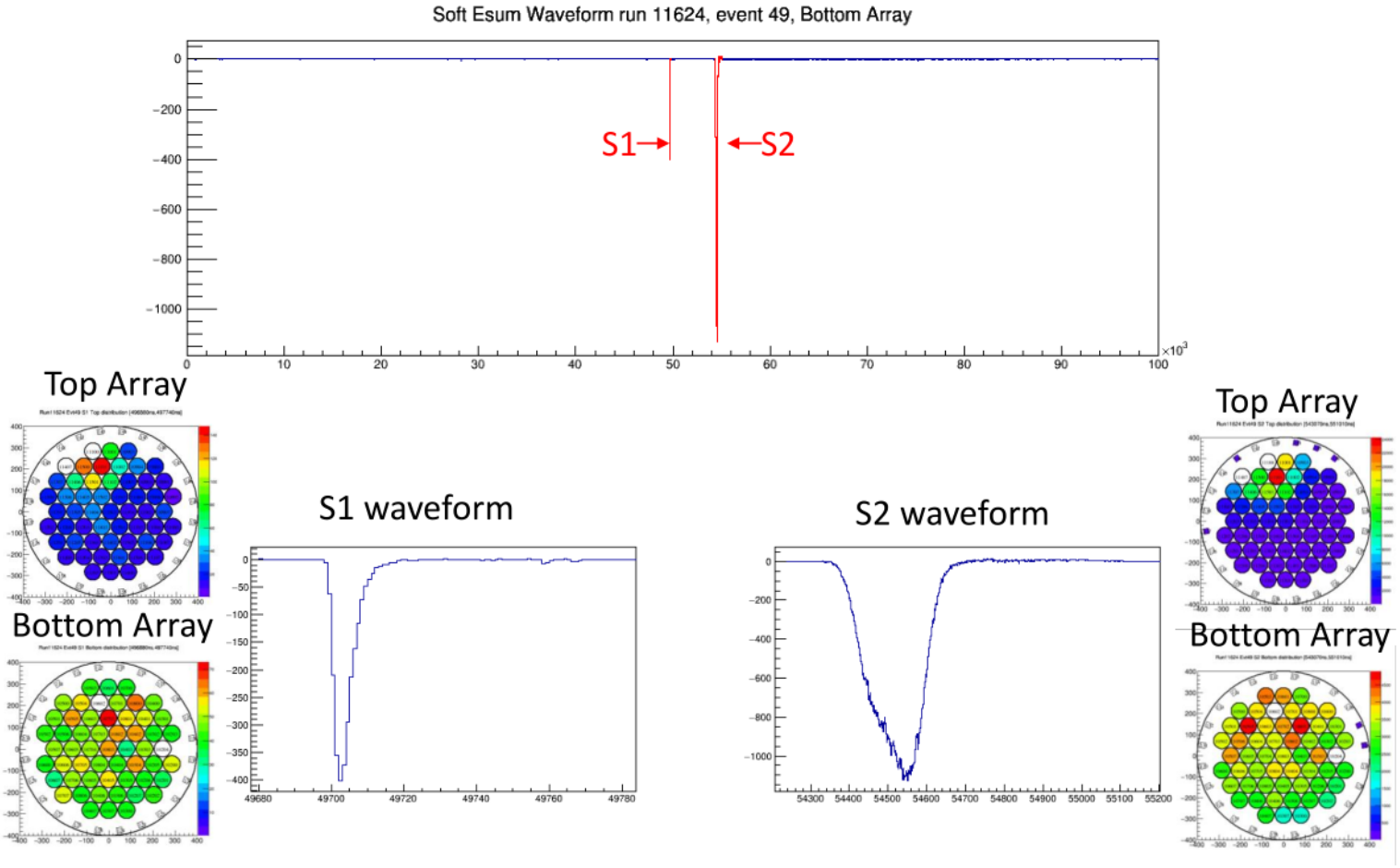
Hit Clustering

Hits in differenct channel in an events are grouped by time to create clusters.
At least 2 hits are required.



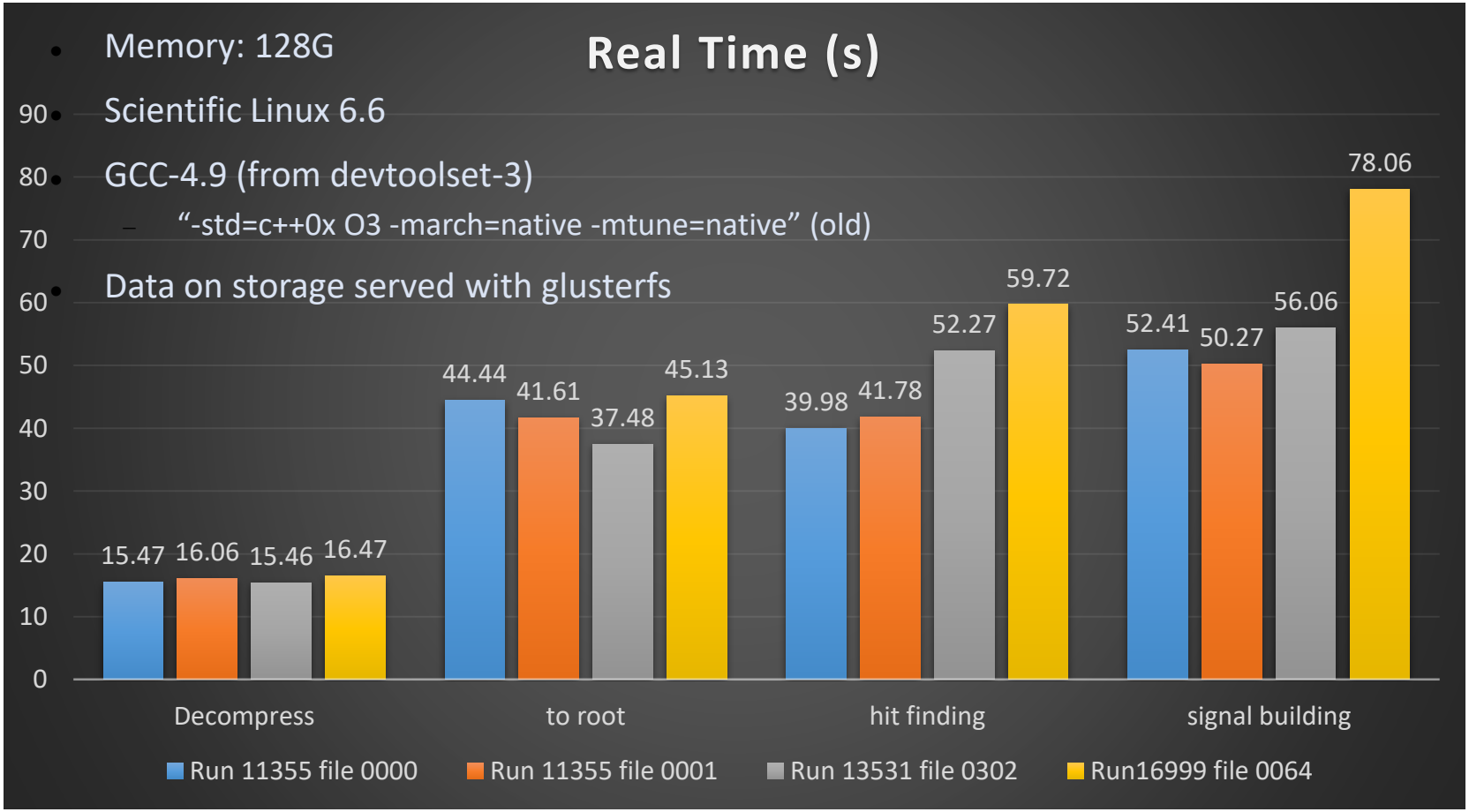
Signal Building

Both cluster and esum information are combined to calculate the signal properties.



Time for Reconstruction

- Dell M630 blade server
- CPU: Intel Xenon E5-2650v3 @ 2.3GHz
- Memory: 128G
- Scientific Linux 6.6
- GCC-4.9 (from devtoolset-3)
  - "-std=c++0x O3 -march=native -mtune=native" (old)
- Data on storage served with glusterfs



Issues of bamboo-shoot2

- The persistency layer depends on ROOT, which is not stable as we expected.
  - Program compiled with the root 5.34/20 crashed when reading the data produced by program compiled with root 5.34/19. It seems that the implementation for the I/O of TClonesArray changed.
  - ROOT 5 is going to the end of its life. We haven't try ROOT 6, but it's will be used as the standard in the near future. We need to consider for the future experiments.
- A library which doesn't depends on ROOT is a better choice.

Consideration to Support C++11

- C++11 have been release for several years with some nice features:
  - auto specifier
  - Type deduction with decltype
  - Lambda functions
  - Rvalue reference
  - …
- Compiler: gcc>=4.9 support all specifications of C++11
- ROOT6, Geant4 10.2 took C++11 as the default compilation option.
- New software need to support C++11.

bamboo-shoot3

- Unlike bamboo-shoot2, which is a data model built on top of ROOT, bamboo-shoot3 is mainly served as the layer of persistency.
- Bamboo-shoot3 contains:
  - A library smaller than 500KB can be statically linked into a program.
  - A tool to generate data model headers based on an input text file.
- Data models can be defined easily in a text file and header (header only) can be generated directly.
- More detailed information in https://github.com/revive/bamboo-shoot3

Event level Checksum and Compression

- Each event has a CRC32 checksum to determine the error during I/O.
- Compression is applied to each event, several fast compression algorithm can be chosen:
  - For higher compression ratio, choose ZSTD (default one)
  - For better performance, choose LZO.

Data Model definition (I)

- Declare the data realm

```
#ifndef PXTYPES_HH
#define PXTYPES_HH

#include <cstdint>
#include <string>
#include <vector>
#include <map>

#include <pxtypes-generated.hh>

PBSF_DECLARE_REALM(
  PandaXIIRealm, 2014,
  PBSF_REGISTER_TYPE(1, RawData),
  PBSF_REGISTER_TYPE(2, CalibData),
  PBSF_REGISTER_TYPE(3, HitData),
  PBSF_REGISTER_TYPE(4, ClusterData),
  PBSF_REGISTER_TYPE(5, SignalData));

#endif /* PXTYPES_HH */
```

This header file is to be generated by the “bsic” tool provided by bamboo-shoot3.

Data Model Definition (II)

- Define the data models in a text file

```
struct RawFadcInfo {
  uint32 eventCounter = 1;
  uint32 triggerTimeTag = 2;
};

struct RawData {
  uint32 runNumber = 1;
  uint32 eventNumber = 2;

  uint32 moduleNumber = 3;
  uint32 channelNumber = 4;
  uint32 sampleNumber = 5;
  uint32 zsCode = 6;
  uint32 inputDac = 7;
  uint32 zsThreshold = 8;

  // originally in rawGeneralParameters
  uint32 triggerTime = 9;
  uint32 v830TriggerCount = 10;
  uint32 v830NoiseCount = 11;
  uint32 v814TriggerCount = 12;
  uint32 nHitsTriggerInMv = 13;
  uint32 sampleHoldOff = 14;

  (uint32 => RawFadcInfo) fadcInfo = 15;
  (uint32 => [RawPmtSegment]) rawWaveforms = 16;
};
```

Type support shall include
primary types,
custom structs,
and STL containers; with features
static typing,
forward/backward compatibility for custom structs,
and NO support for cross referencing.

This is an std::map

Brackets indicate an std::vector

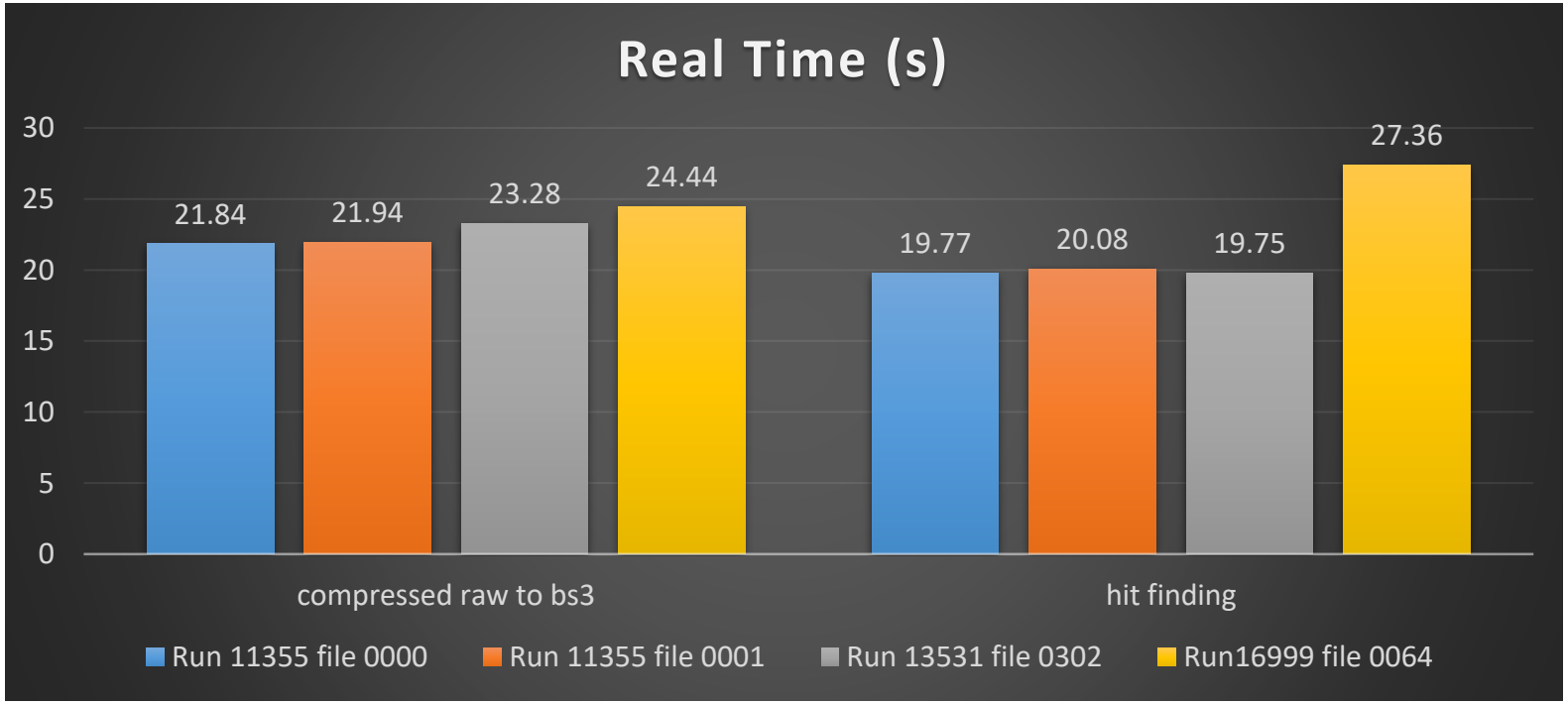
Functional Programming

```
// perform calibration
Calibration cal;
Esum esum;
if (raw_data.begin() != raw_data.end()) {
    auto run = raw_data.begin()->runNumber;
    PandaXDataSource pds;
    cal = { pds, run };
    esum = { pds, run };
}

HitFinder find_hit;
find_hit.threshold = threshold;

auto results = pu::map([&](const RawData &raw) {
        auto cd = cal(raw);
        auto hd = find_hit(cd);
        auto ed = esum(cd, hd.second);
        return std::make_tuple(std::move(hd), std::move(ed));
    }, raw_data);
// write to corresponding output file
pu::copy(results,
         pu::spread_writer(hit_file.write_iterator(),
                           esum_file.write_iterator()));
```

An example of calibration, calculate esum and hit finding.

PandaX data reconstruction migration

- We just started to migrate the reconstruction chain with bamboo-shoot3.
- Only signal building step missing. Processing speed increased by 100%!*
- Smaller file size compared to ROOT. *



* Both claims work for ZSTD compression, LZO have better performance but generate larger files.

Bamboo-shoot3 in collider experiments - an example

```
struct  TrackerHit {
  uint32 hitId = 1;
  uint32 detectorId = 2;
  double energy = 3;
  double x = 4;
  double y = 5;
  double z = 6;
  double t = 7;
  [uint32] trackIds;
}


struct TrackerHitData {
  uint32 runNumber = 1;
  uint32 eventNumber = 2;
  [TrackerHit] hits = 3;
}

struct Track {
  uint32 trackId = 1;
  double rho = 2;
  ...
  [uint32] hitIds = n;
}

struct TrackerData {
  uint32 runNumber = 1;
  uint32 eventNumber = 2;
  [Track] tracks = 3;
}
```

Visit the tracker hits within each track

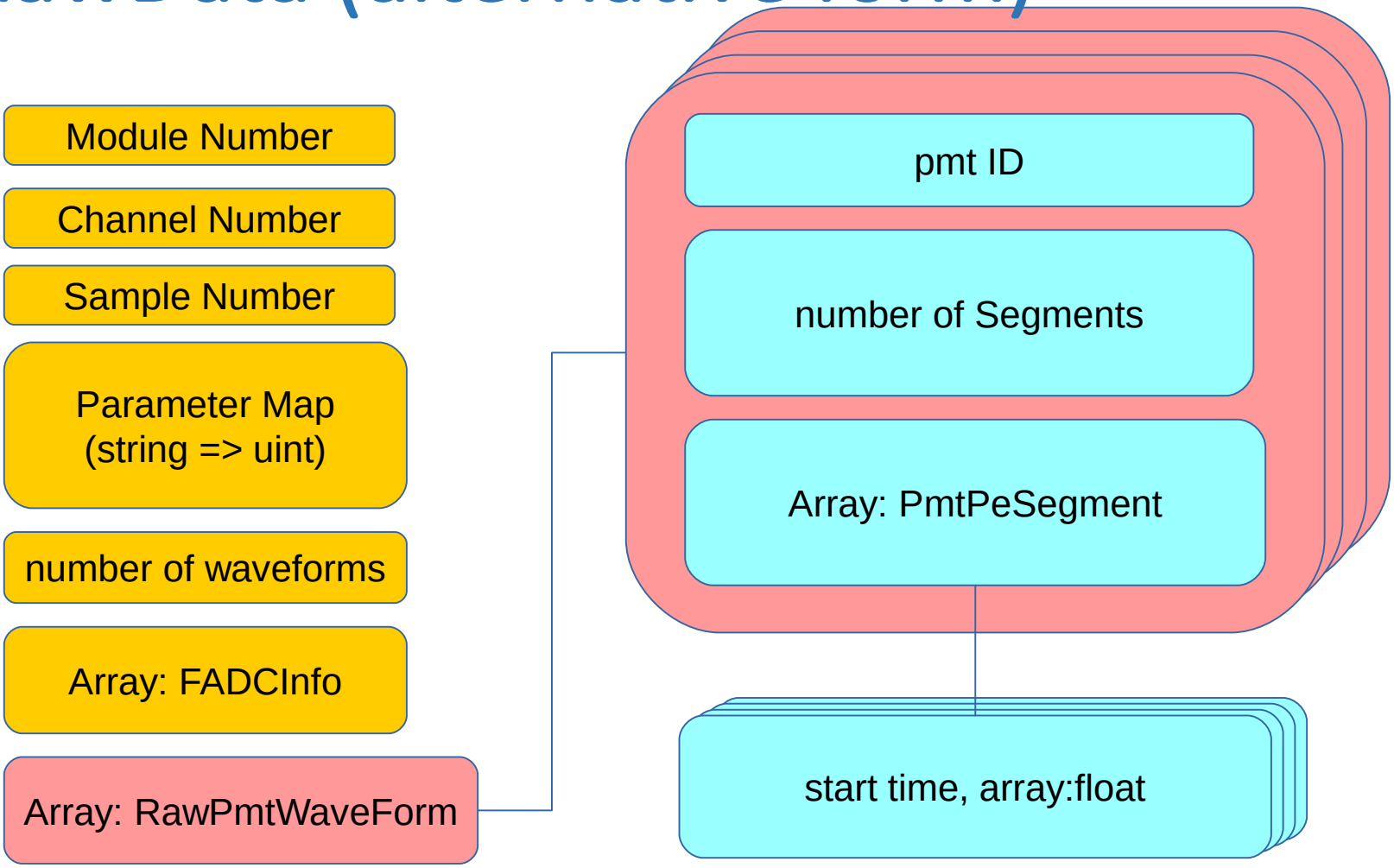
```
{

    TrackerData tracker_data;
    TrackerHitData tracker_hit_data;
    ...;

    for (const auto & track : tracker_data.tracks) {
        const auto & hits = tracker_hit_data.hits;
        for (const auto & hit_id: track.hitIds) {
            auto hit_it = std::find(hits.begin(), hits.end(),
                                    [hit_id](const TrackerHit & h) {
                                        return h.hitId == hit_id;
                                    });
            if (hit_it != hits.end()) {
                const auto & hit = *hit_it;
                // do something with hit
                ...;
            }
        }
    }
}
```

Summary

- A data model based on ROOT has been developed for the PandaX dark matter experiments.
- The model plays an important role in the event reconstruction procedure, and some issues addressed.
- A new data persistency library is developed to service the basis of the data model.

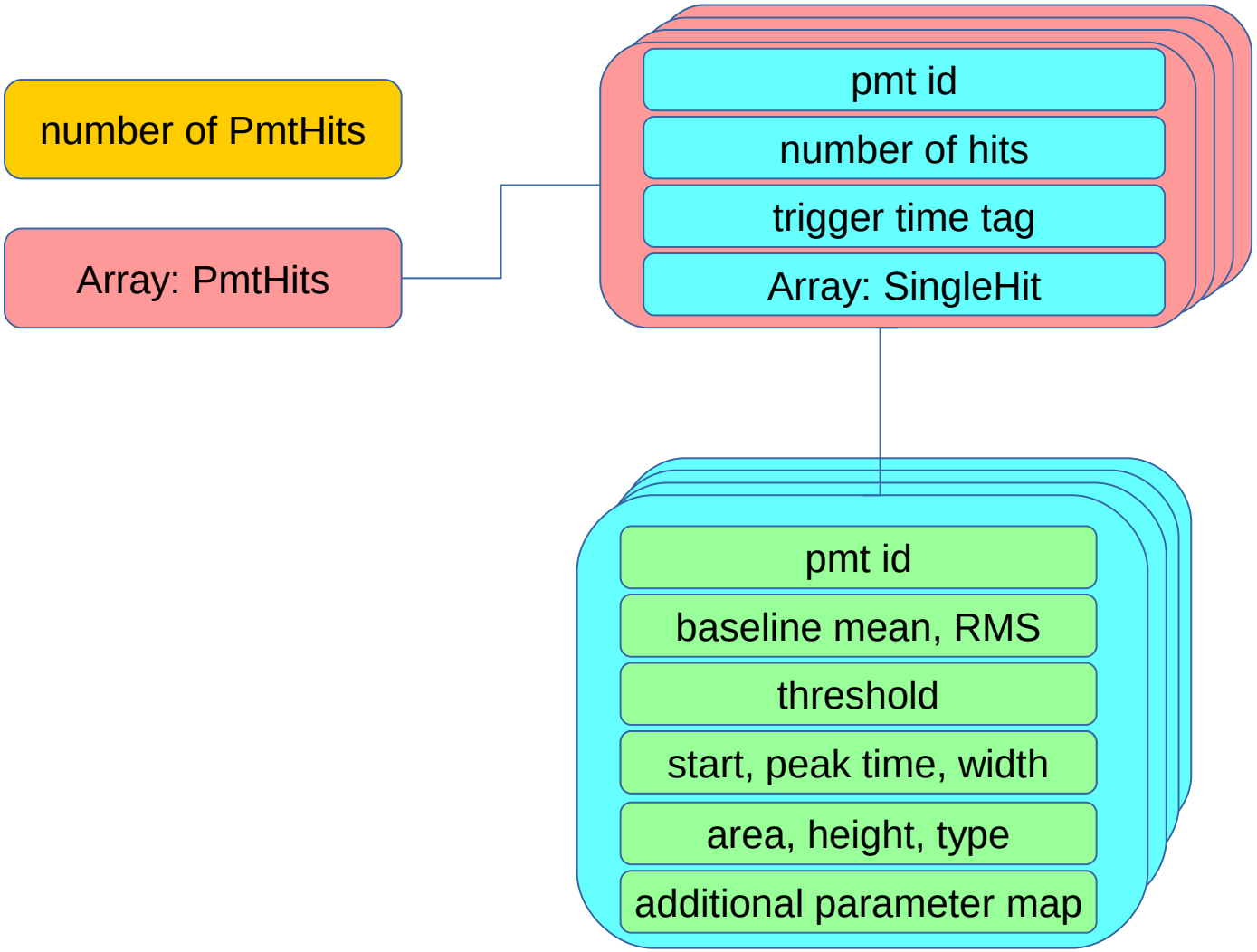
BACKUP SLIDES

RawData (alternative form)

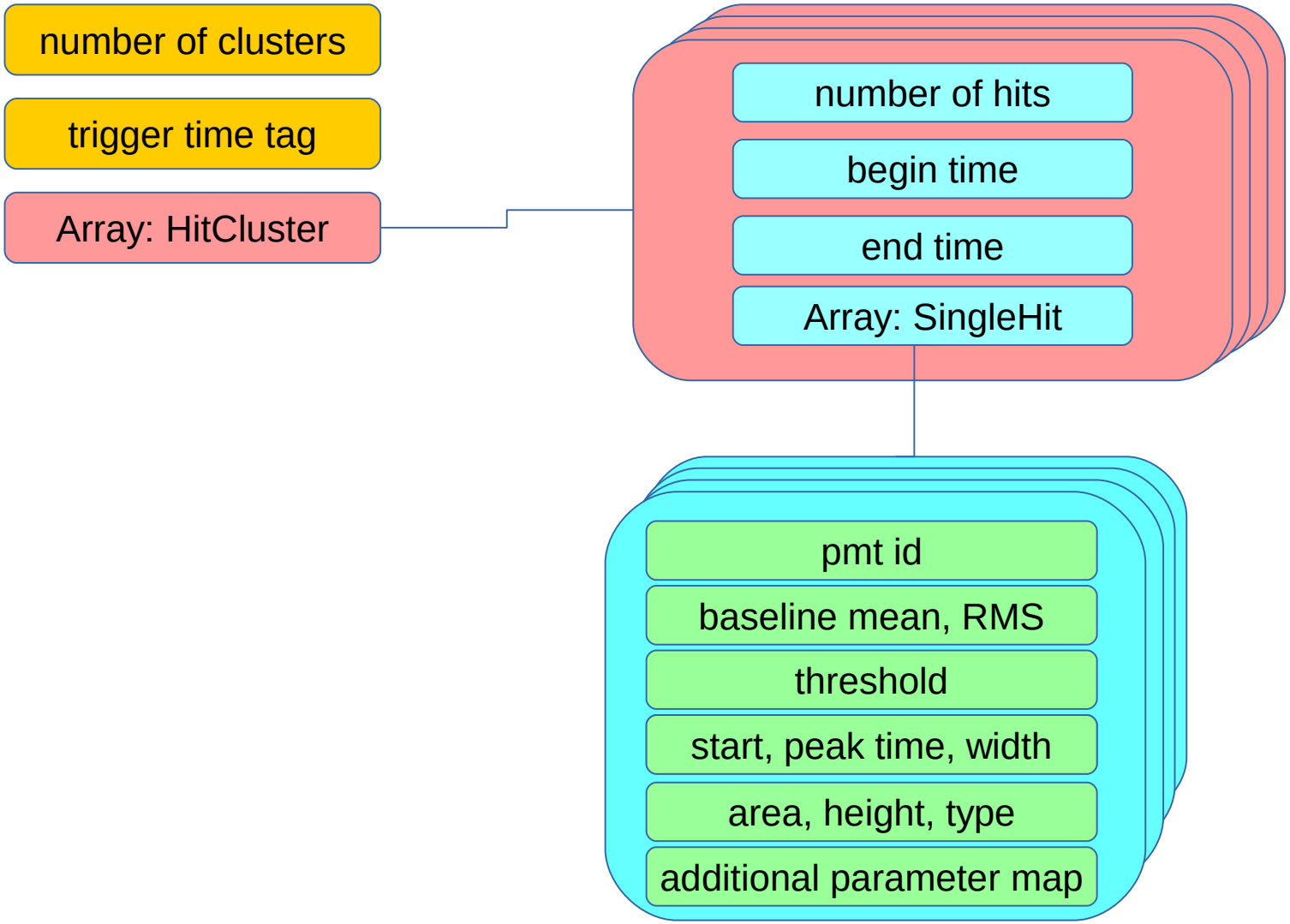


Array = TClonesArray

HitData


number of PmtHits
Array: PmtHits
pmt id
number of hits
trigger time tag
Array: SingleHit
pmt id
baseline mean, RMS
threshold
start, peak time, width
area, height, type
additional parameter map

ClusterData



SignalData

number of signals

trigger time

Array: Signal