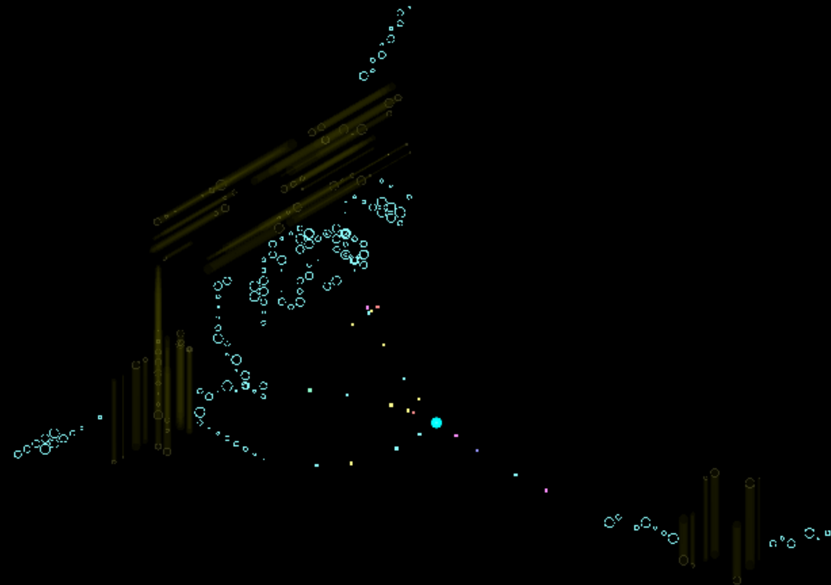
Mitglied der Helmholtz-Gemeinschaft

Time-based simulation

PANDA Computing Workshop - SUT

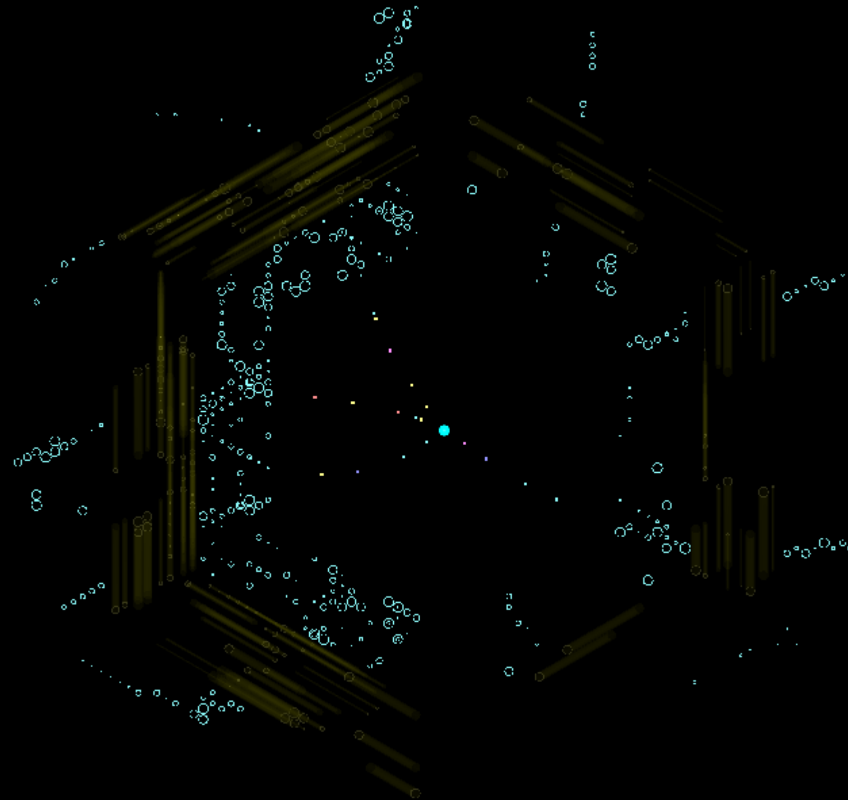
Juli 5, 2017

Time-Based Simulation



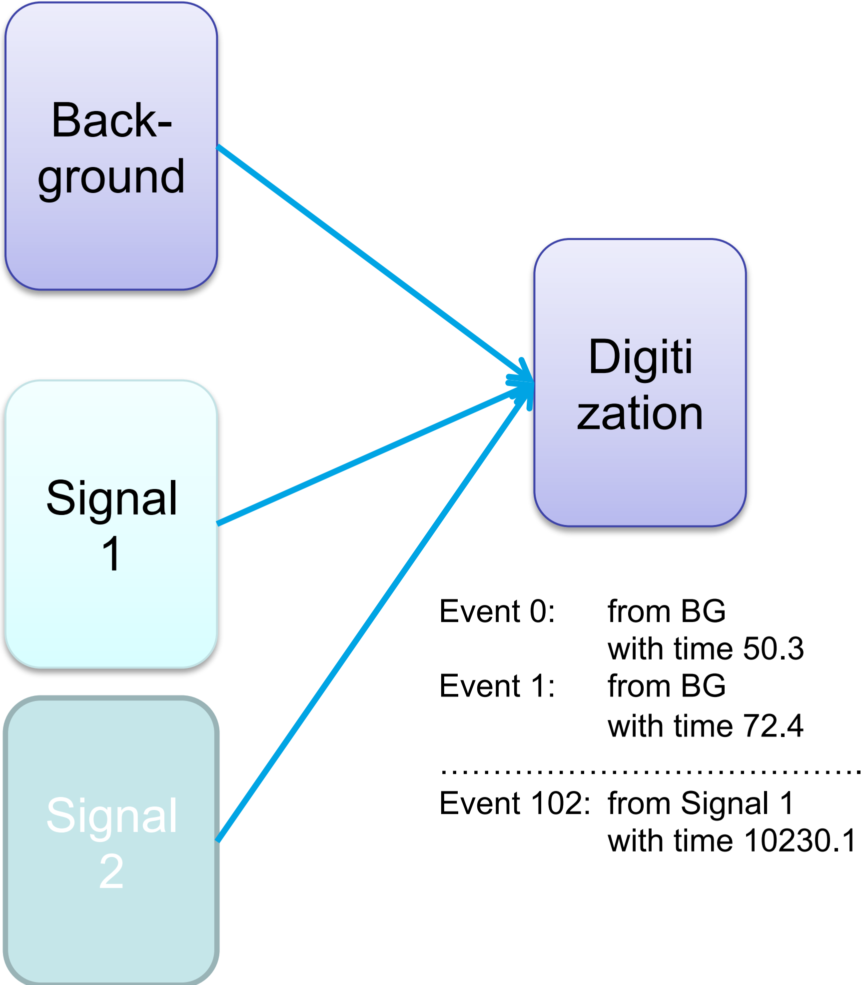
Single Event

Time-Based Simulation



20 MHz overlap

Event Mixing (Mohammad's work)



- MC file does not know anything about time structure
- Time structure is calculated in Digitization stage
- Many different signal files can be added to one background file
- Where the data is coming from is stored in EventHeader
- No overlap (pileup) of events
- `FairMixedFileSource` does the job

FairRoot/examples/simulation/Tutorial2/macros/create_digis_mixed.C

How do events overlap?

JÜLICH
FORSCHUNGSZENTRUM

- Problem: In the (usual) event-based simulation each event can be treated completely independent of each other and time between events does not play a role.
  This is not the reality in experiments:
  - Sensor elements are still blocked from previous hits
  - Electronic is still busy
  - Hits too close in time cannot be distinguished
  - …
- Special problem for PANDA:
  - Continuous beam with Poisson statistics ➔ many events with short time between them
  - No hardware trigger

Event Structure and Detector Response

JÜLICH FORSCHUNGSZENTRUM

ev 1 ev 2 ev 3 time

SciTil

EMC

MVD

STT

.
.
.

Event Structure and Detector Response

STT

.
.
.

Event Structure and Detector Response

Wanted data structure

STT

Data structure in Tree

Event 1

Event 2

Event 3

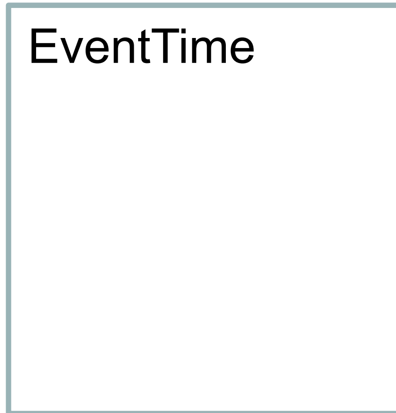
.
.
.

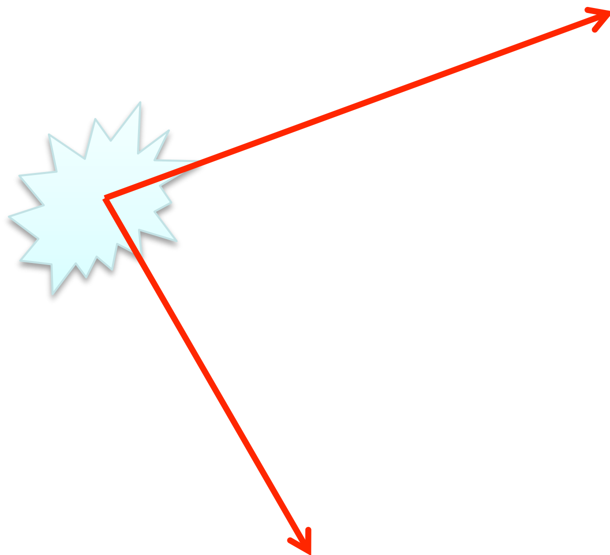
= detector hits from different events

Times to take into account

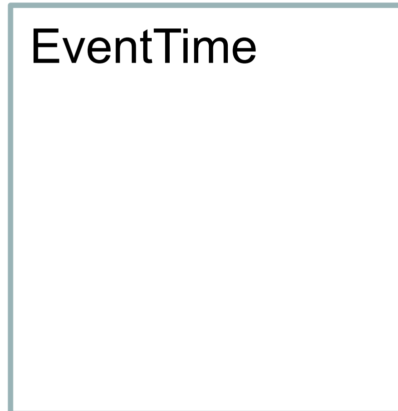
EventTime [ns]

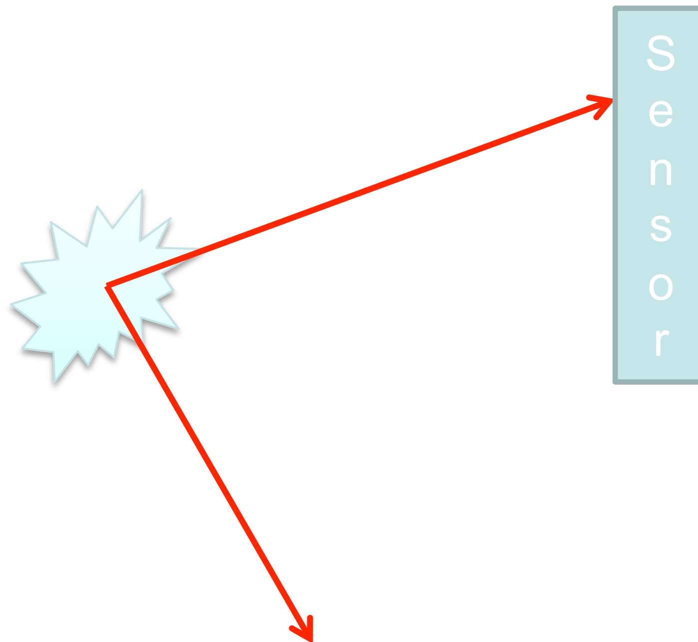
Times to take into account



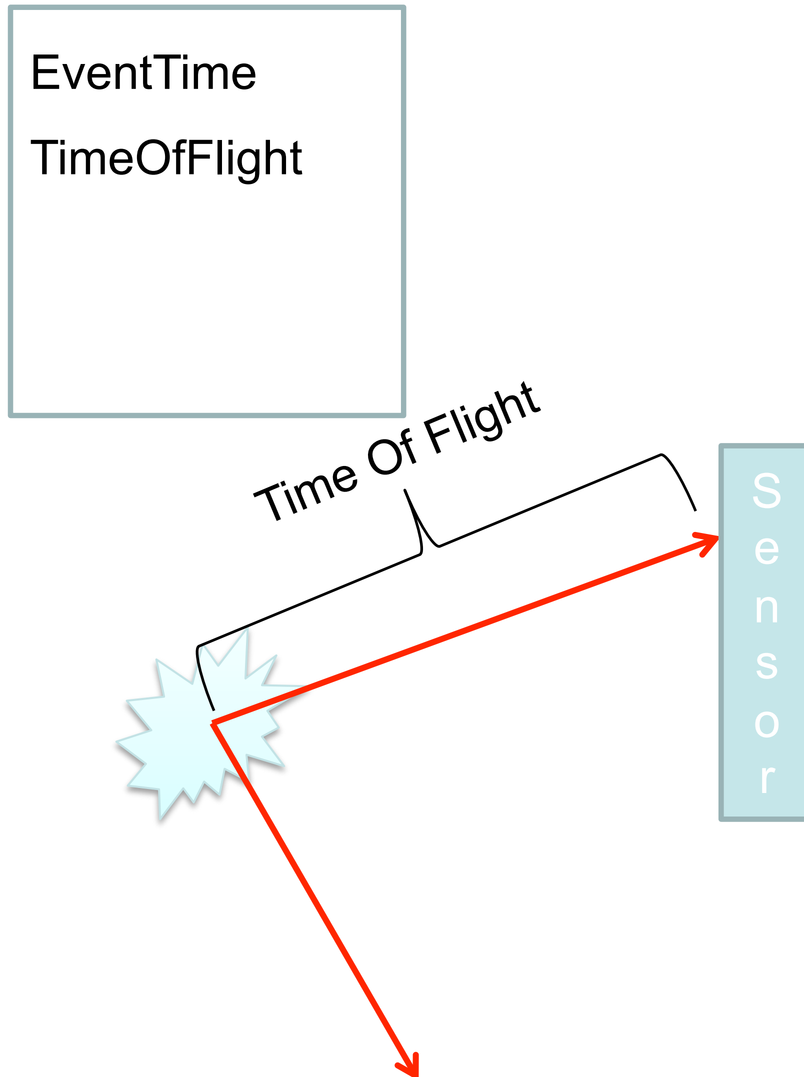


Times to take into account





Times to take into account

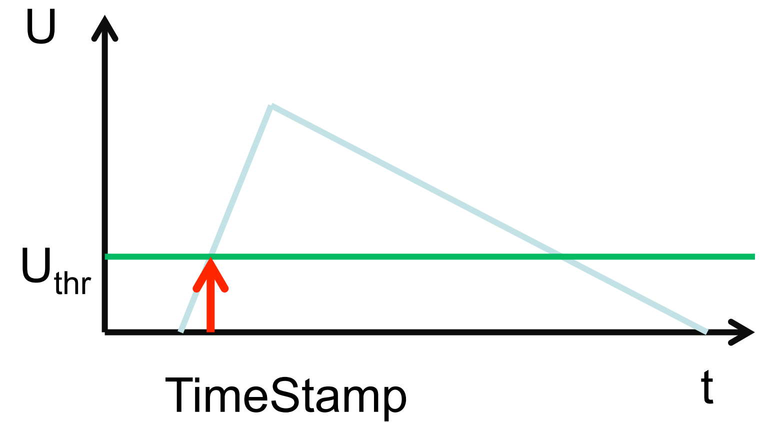


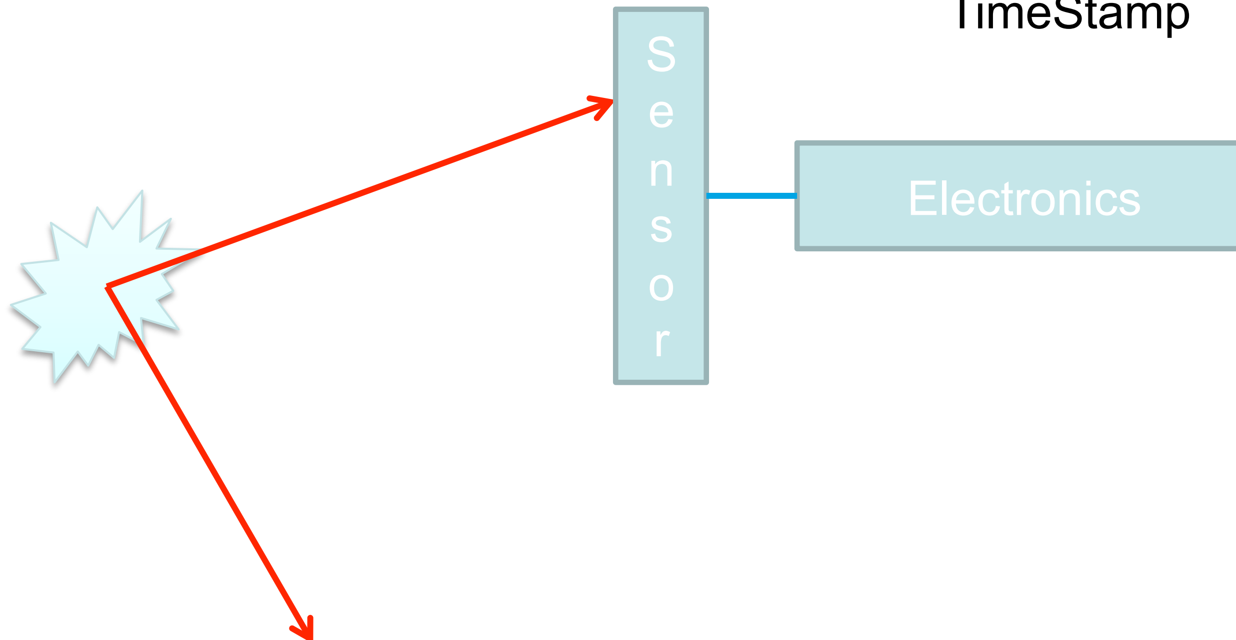
Times to take into account

EventTime

TimeOfFlight

TimeStamp



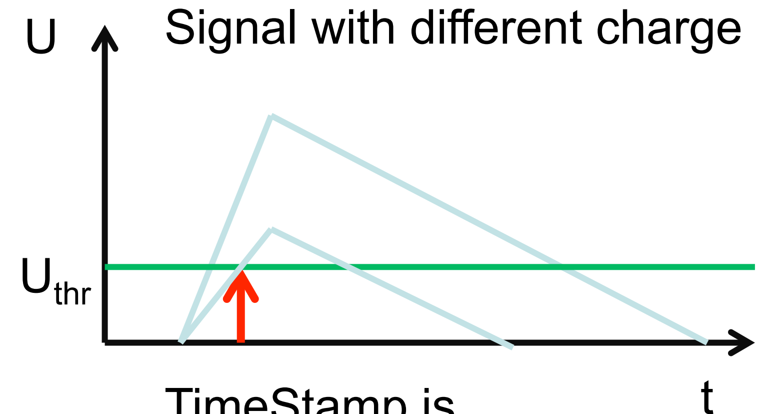


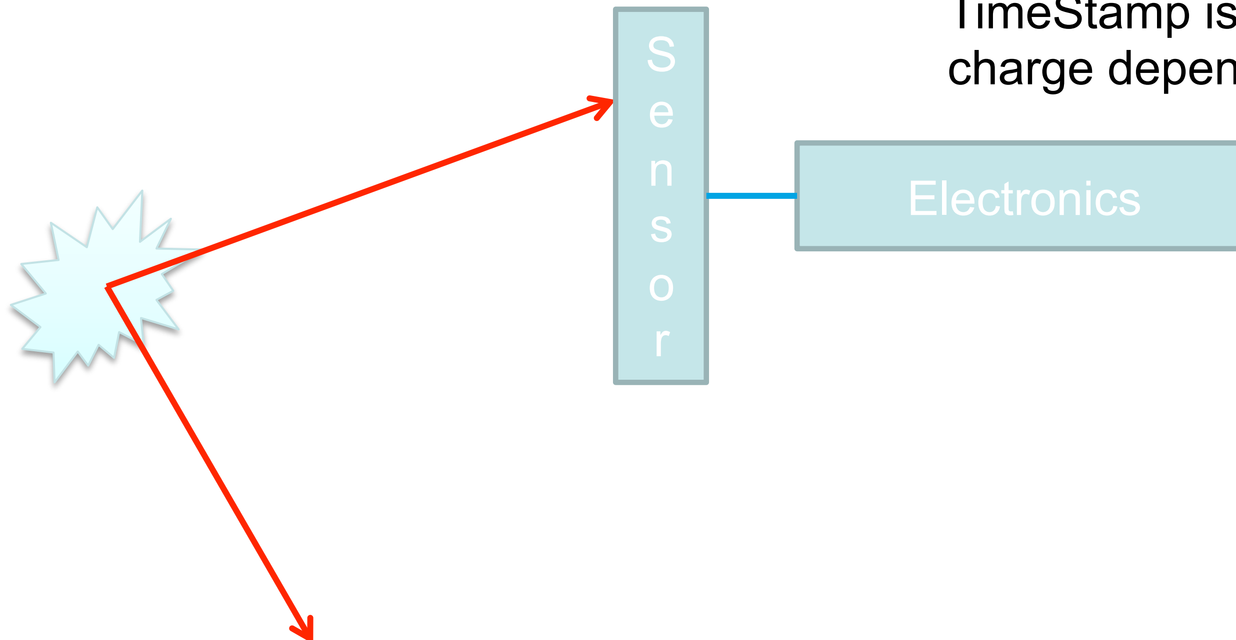
Times to take into account

EventTime

TimeOfFlight

TimeStamp



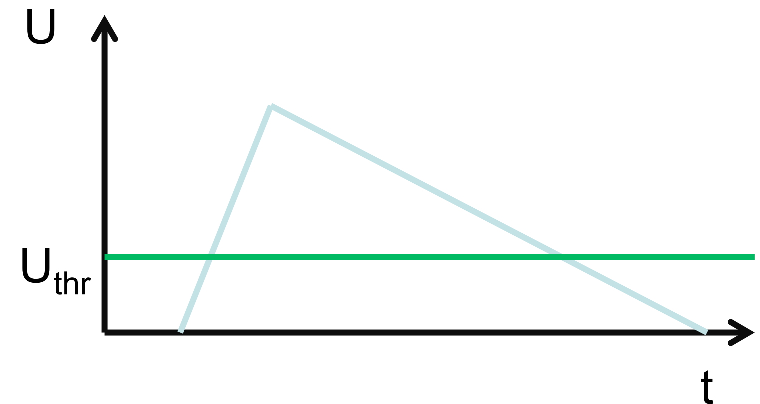


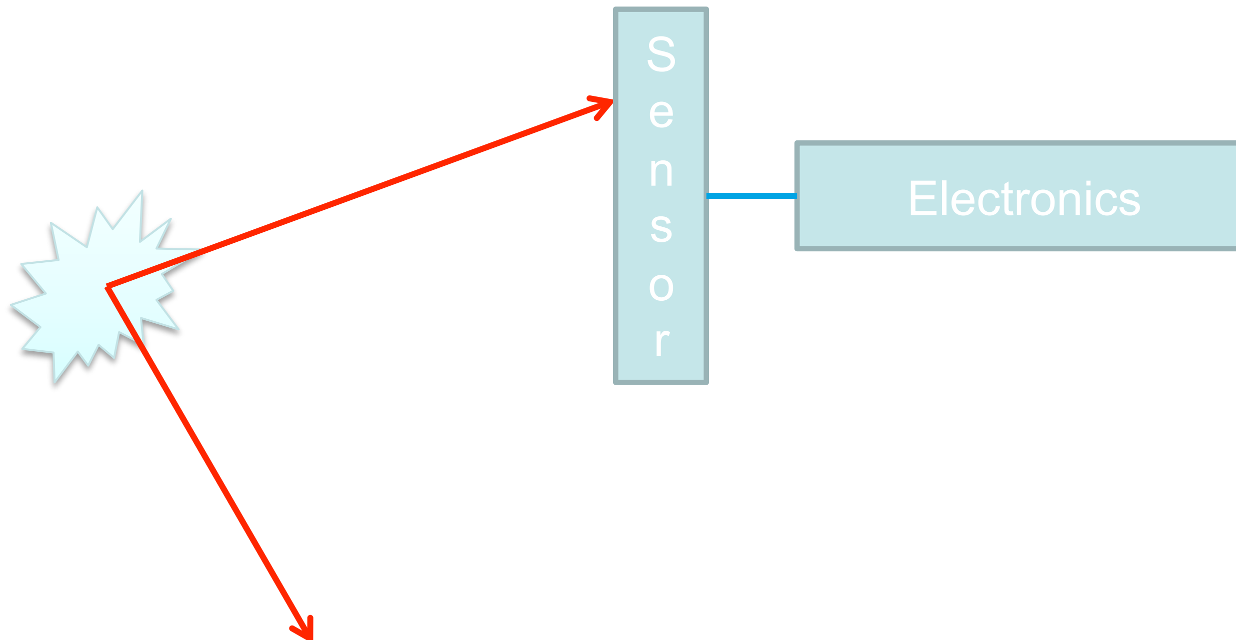
Times to take into account

EventTime

TimeOfFlight

TimeStamp



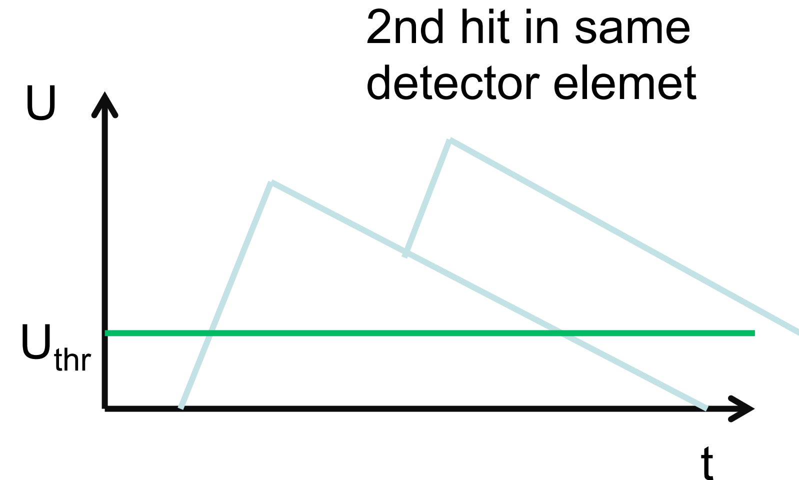


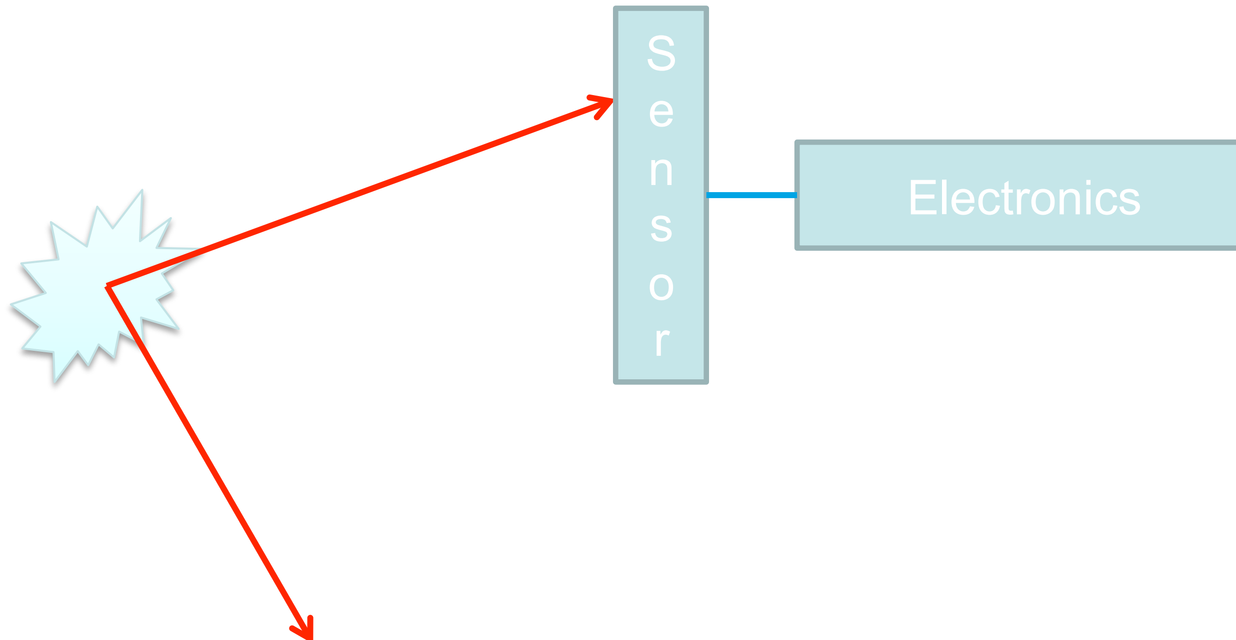
Times to take into account

EventTime

TimeOfFlight

TimeStamp

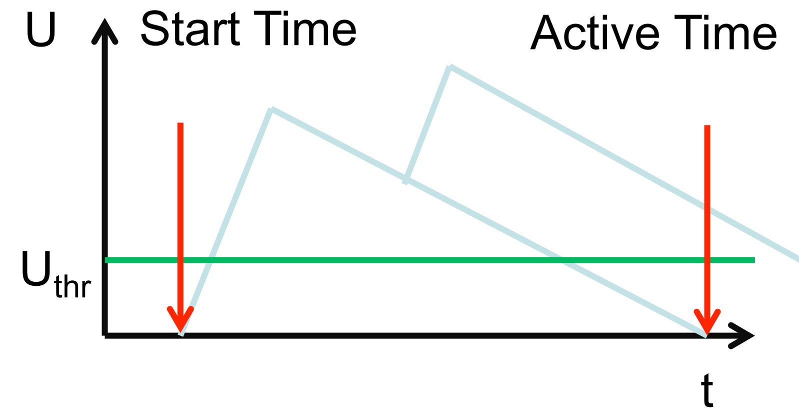


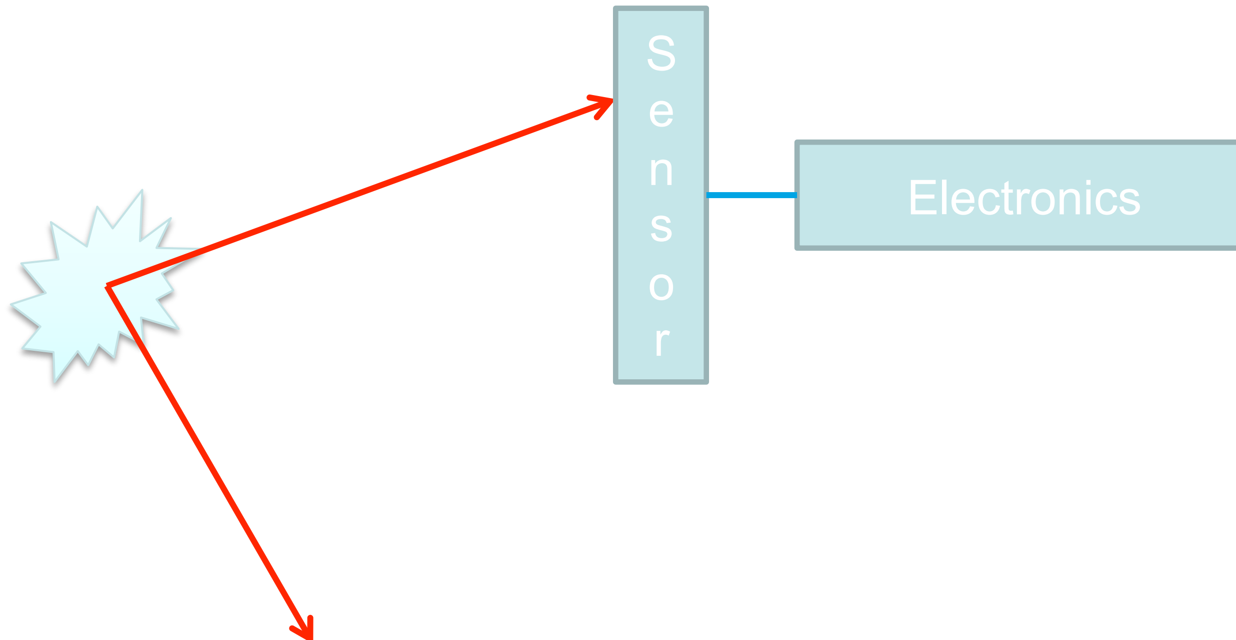


Times to take into account

JÜLICH FORSCHUNGSZENTRUM

- EventTime
- TimeOfFlight
- TimeStamp
- Start/ActiveTime



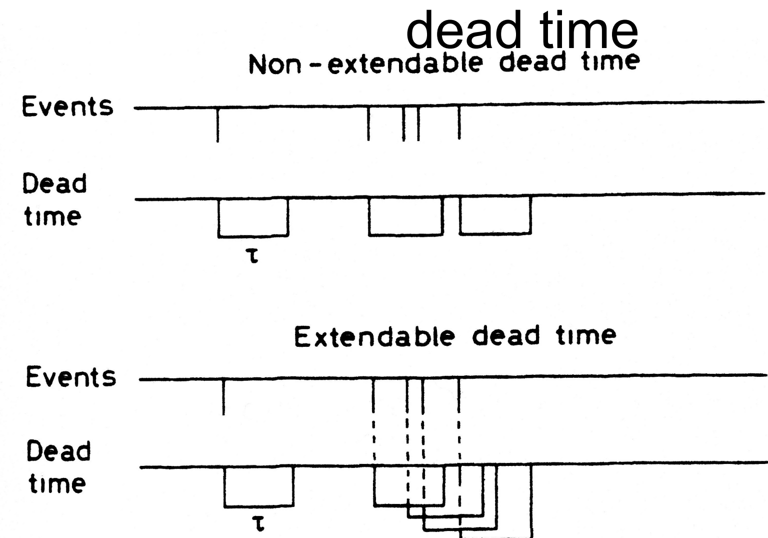


Dead Time

- Finite time required by a detector to process an event, during which no additional signal can be registered

Two cases:

- detector is insensitive → **non-extendable/paralyzable**
- detector stays sensitive → **extendable** or **paralyzable** dead time (e.g. rate of a Geiger-Müller counter drops at at very high dosis)



Necessary Times

JÜLICH FORSCHUNGSZENTRUM

- 4 times necessary:
  - Event Time
    - *Assigned to Events in digitization if* `fRun->SetEventMeanTime(Double_t)` *is set in the digi macro*
    - *Value can be accessed in tasks via:* `FairRootManager::Instance()->GetEventTime()`

panda

Necessary Times

- 4 times necessary:
  - Event Time
  - Time-of-Flight:
    - *Automatically stored in the MC points of each detector*

Necessary Times

JÜLICH FORSCHUNGSZENTRUM

- 4 times necessary:
  - Event Time
  - Time-of-Flight
  - Time Stamp:
    - *Time assigned to each detector hit (MANDATORY for PANDA!)*
    - *Absolute time! (includes Event Time, ToF , Electornics)*
    - *Resolution and offset depends on individuall detector*

Necessary Times

JÜLICH FORSCHUNGSZENTRUM

- 4 times necessary:
  - Event Time
  - Time-of-Flight
  - Time Stamp
  - Start/Active Time:
    - *Time window an event can influence any other event happening in the same detector element*
    - *Strongly detector dependent*
    - *What happens if a second hit occurs during the active time of a previous hit is strongly detector dependent (hit lost, new hit modified, old hit modified, new hits created, …)*
    - *Absolute time!*

panda

FairWriteoutBuffer

JÜLICH FORSCHUNGSZENTRUM

- Special buffer to store detector data between events
- You give the data you want to store an absolute time window this data is active in your detector and can influence later events.
- If the same detector element is hit a second time the data is modified.
- This is an abstract base class where you have to inherit from

SORTING THE DATA

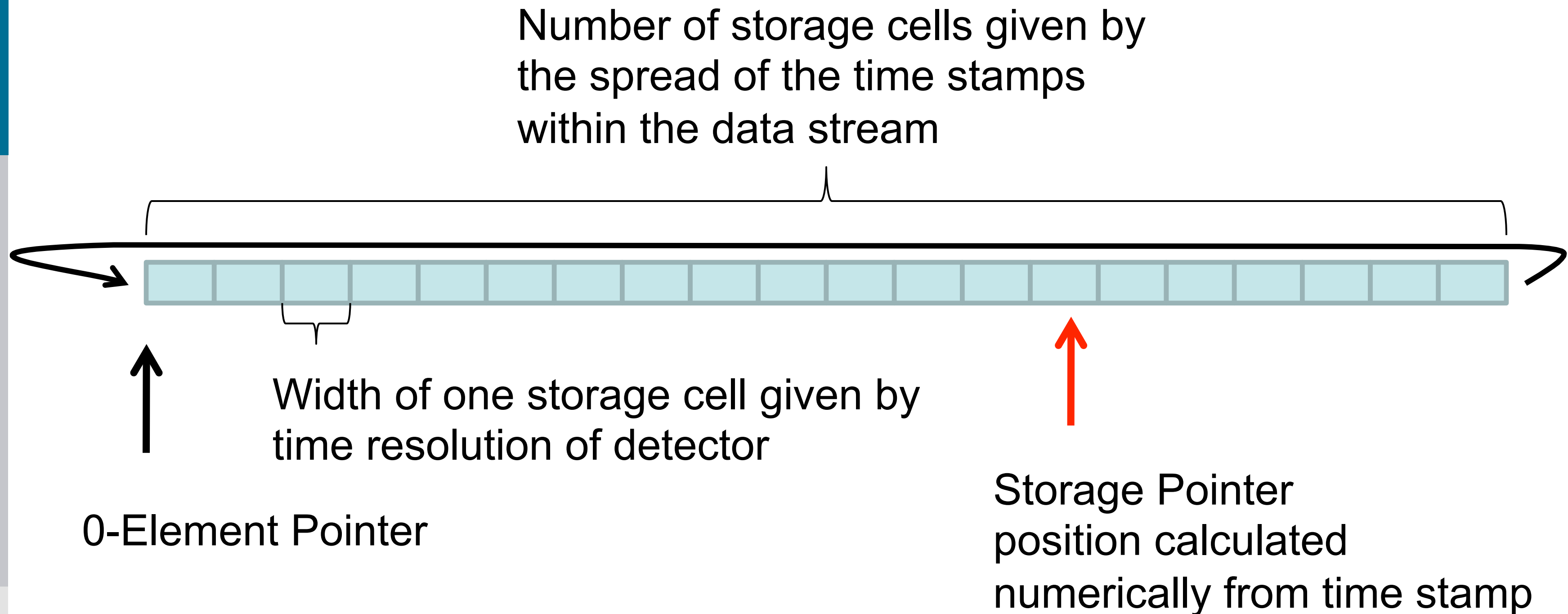
Sorting the data

FORSCHUNGSZENTRUM JÜLICH

- It is essential for the later extraction of the data that it is sorted by its time stamp
- A base class for the sorter (`FairRingSorter`) and a base class for a sorter task (`FairRingSorterTask`) are already implemented in the software
- To use them you have to derive your own sorter classes from them and overwrite some methods

panda

Technical implementation – Ring Sorter



If a storage position is calculated which would override old data, the old data is saved to disk and the storage cell is freed

Digi Data randomized

Digi Data Stream

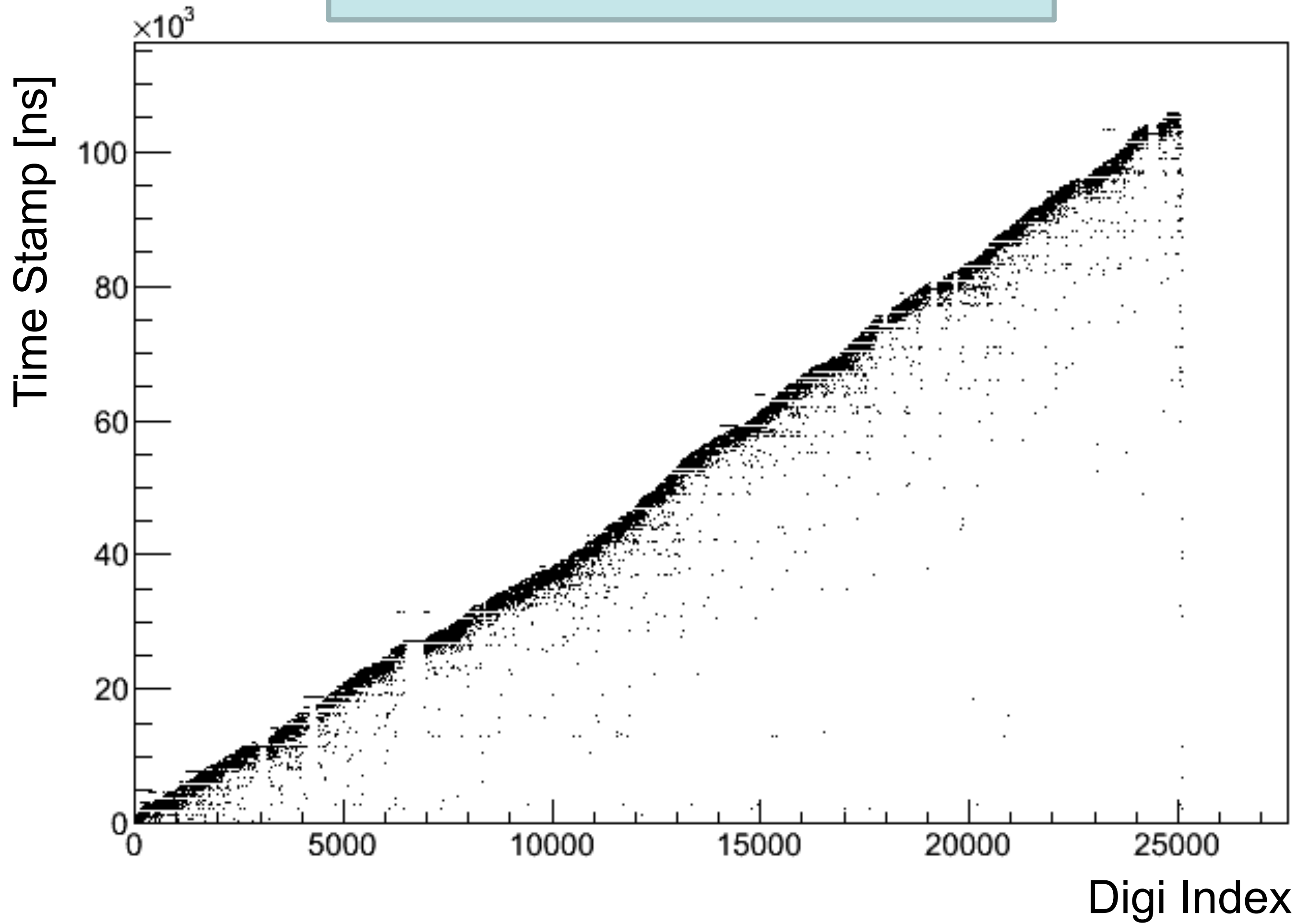
Same color = same event

0, 0.2, 0.4, 0.6, 0.8, 1

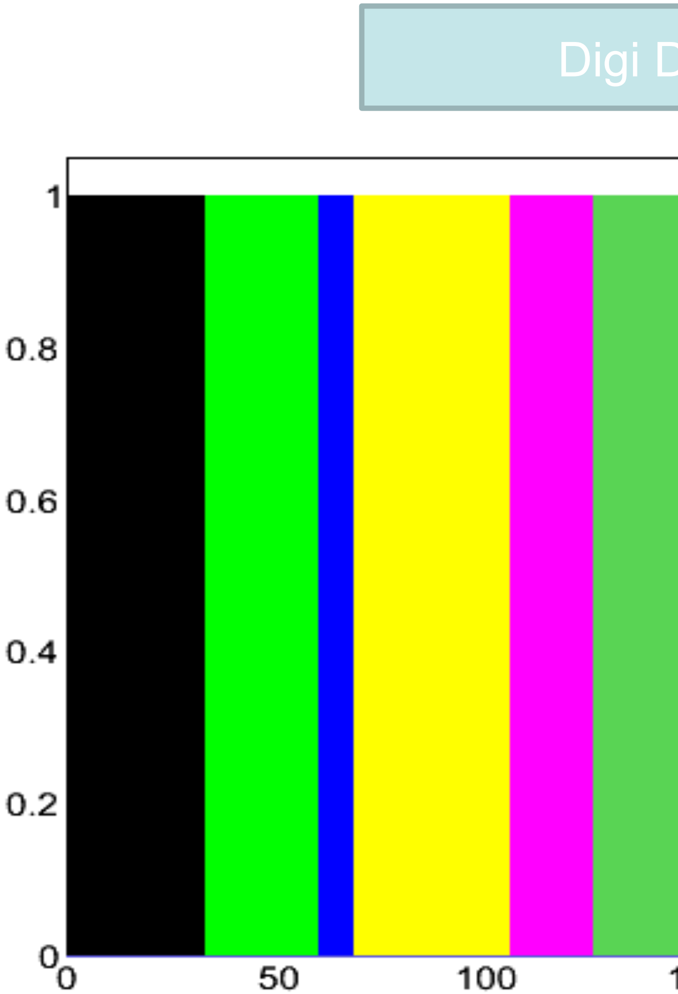
0, 50, 100, 150, 200, 250, 300

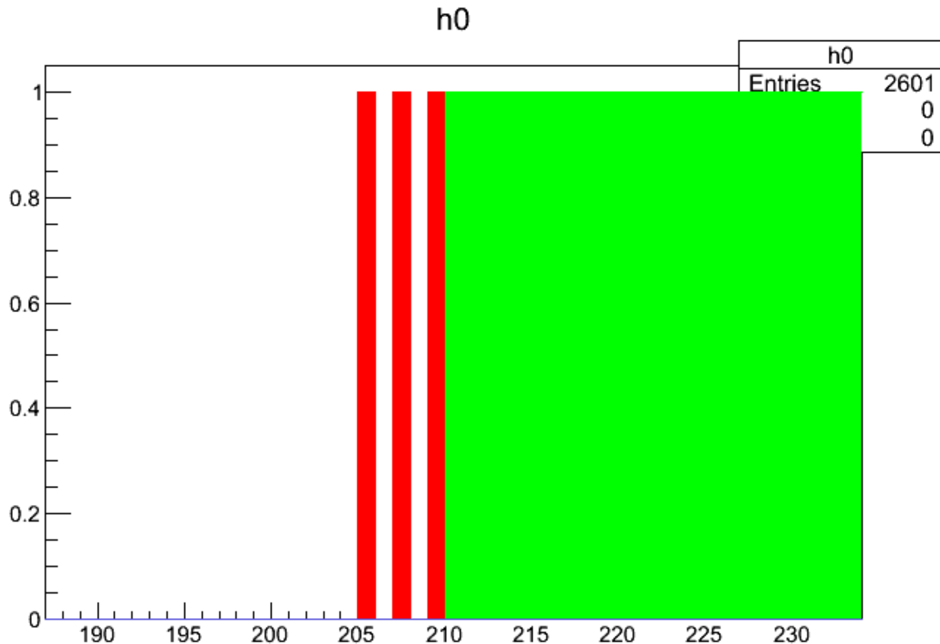
Digi Index

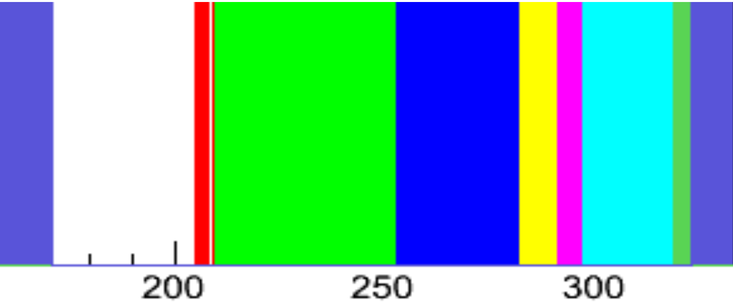
Digi Data randomized



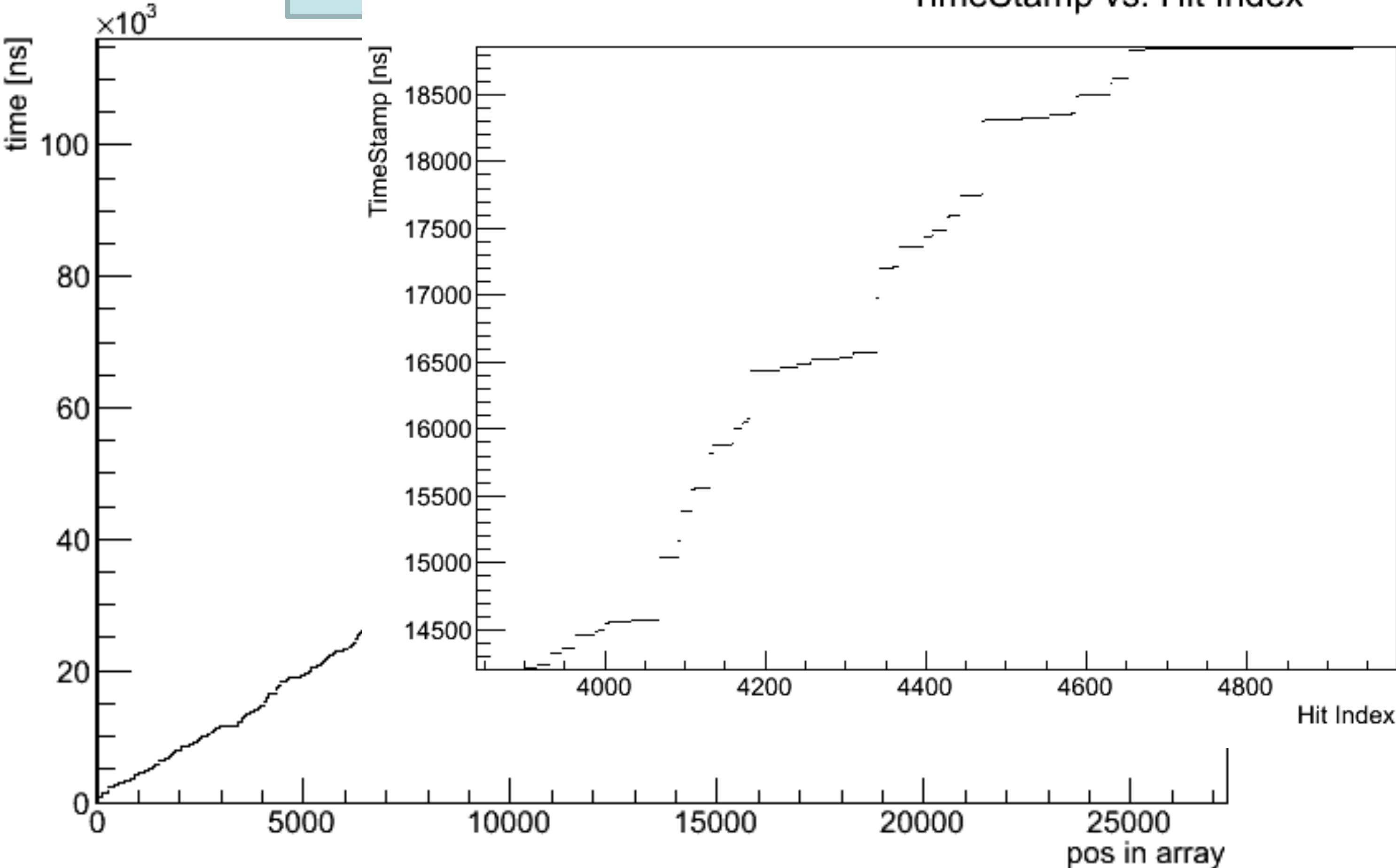
Digi Data Sorted







Digi Data Sorted



READING BACK THE DATA

Reading back data

JÜLICH FORSCHUNGSZENTRUM

- Reading back data is done via the FairRootManager
- Two different methods exists

```
FairRootManager::GetData(BranchName, Functor, Parameter);
FairRootManager::GetData(BranchName, StartFunctor, StartParam.,
                                           StopFunctor, StopParam.);
```

- GetData with one functor/parameter runs always forward in time
  - Data is only read once
- GetData with two sets of functor/parameter is able to get data within a time interval
  - Data can be extracted many times
  - Works only with special functors ➔ next page

panda

What is a Functor?

JÜLICH FORSCHUNGSZENTRUM

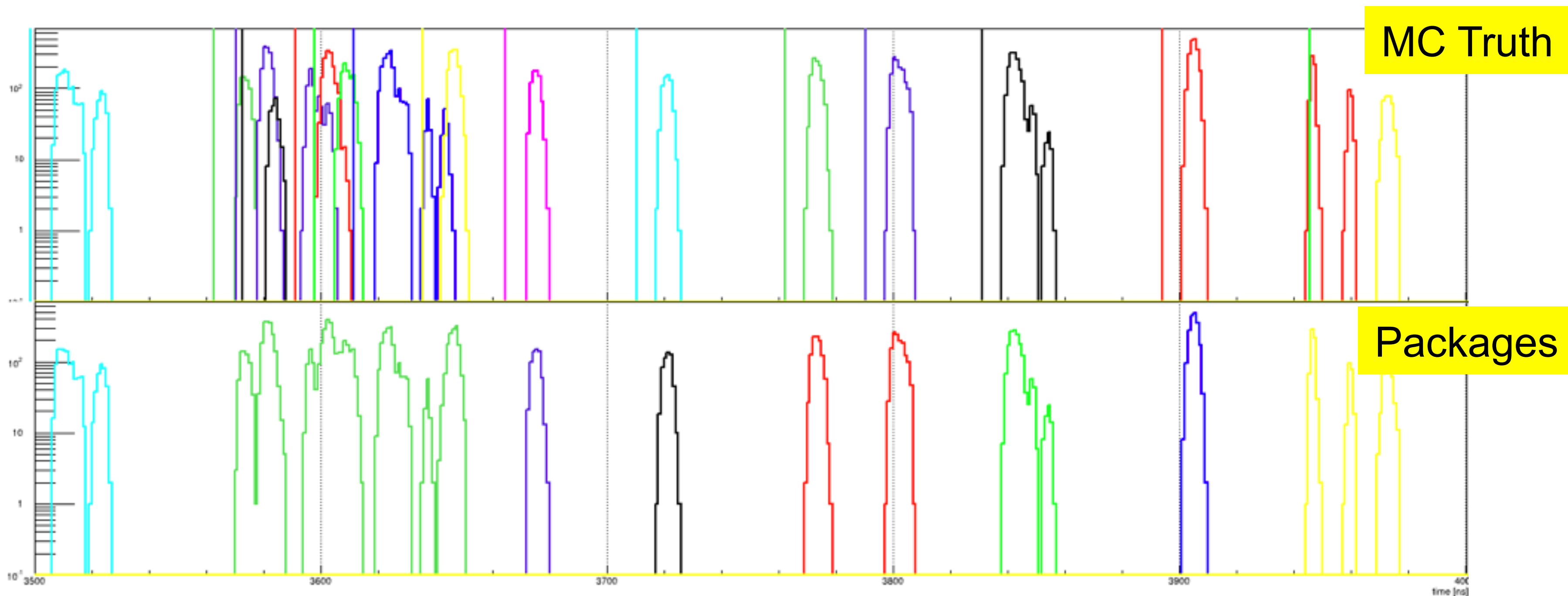
- A (binary)functor is a class with an `operator()` which takes two parameters as an input and has one output.
- In our case the parameters are `FairTimeStamp*`, `double` as input and `bool` as output.
- The functor is true if it gets data which does not fit into the selection criterion
- In this way you can define your own data selectors
- Existing examples:
  - `StopTime`: Returns all data with a TimeStamp less than the given parameter
  - `TimeGap`: Returns all data before a time gap larger than the given parameter
- For the `GetData`-Method with two functors the first has to be the `StopTime` - functor

panda

Data packaging

- Select data packages according to time gaps between digi clusters
- Works very well for detectors with precise time measurement

Example: GEM Digis

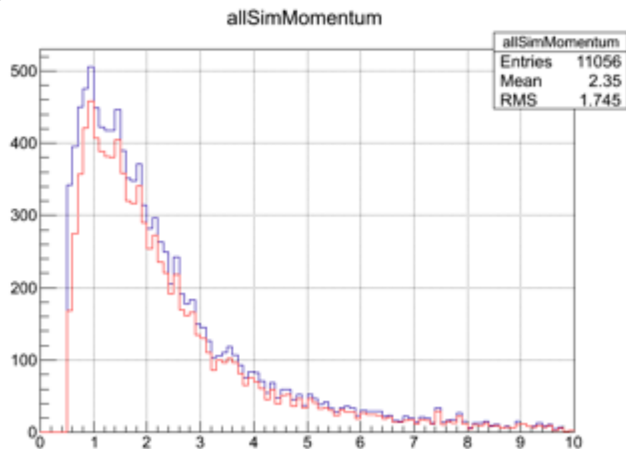


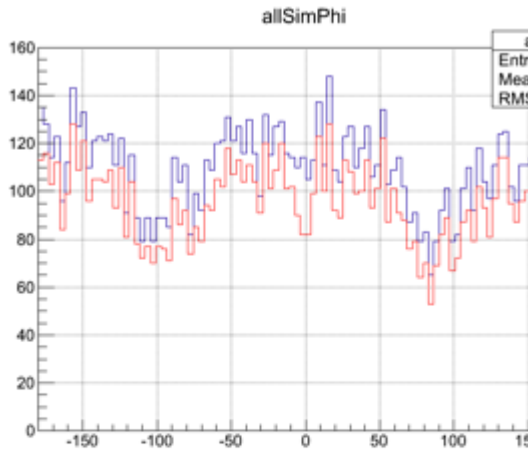
R. Karabowicz

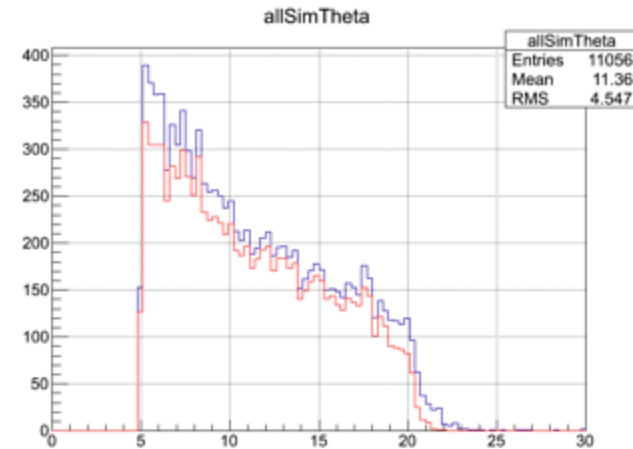
GEM Tracking Efficiency

87% for primaries with |p|>1GeV/c,

compared to ~95% in event-based reconstruction







R. Karabowitcz

GEM Event Building

JÜLICH
FORSCHUNGSZENTRUM

Use the tracks' start-time

- Tracks with start-time closer than 3 ns end up in one event (3ns come from the tracks start-times correlation), with event time set to a mean of constituent tracks' start times
- Even one track can form an event
- Compare reconstructed event times with MC event times, if the difference is smaller than 5 ns the MC event is considered to be reconstructed

R. Karabowicz

Event Builder

Event start time

10^{3}
10^{2}
10
1
10^{-1}

6600 6700 6800 6900 7000 7100 7200 7300

R. Karabowicz

Event Builder

Event start time

Digi distribution

10^3 10^2 10 1 10^{-1}

6600 6700 6800 6900 7000 7100 7200 7300

R. Karabowicz

Event Builder

JÜLICH
FORSCHUNGSZENTRUM

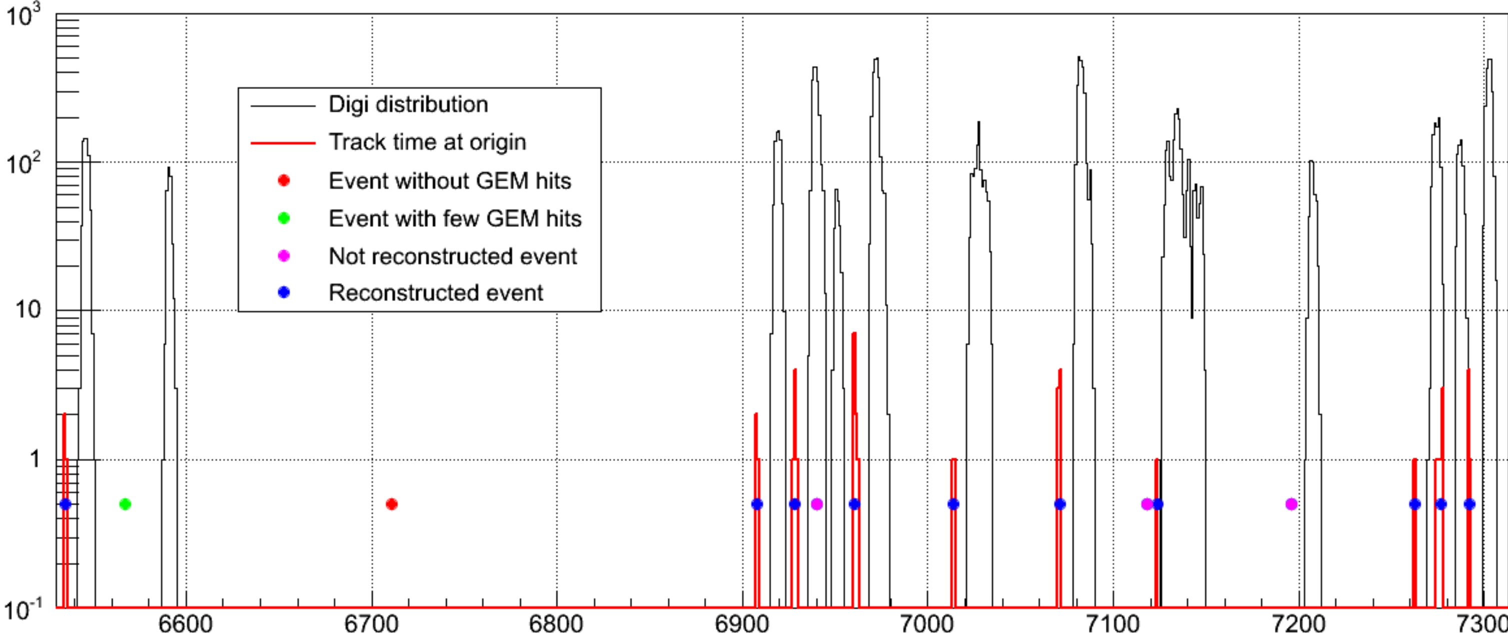
Event start time
Digi distribution
Track time at origin

10^{3}
10^{2}
10
1
10^{-1}

6600 6700 6800 6900 7000 7100 7200 7300

R. Karabowicz

Event Builder

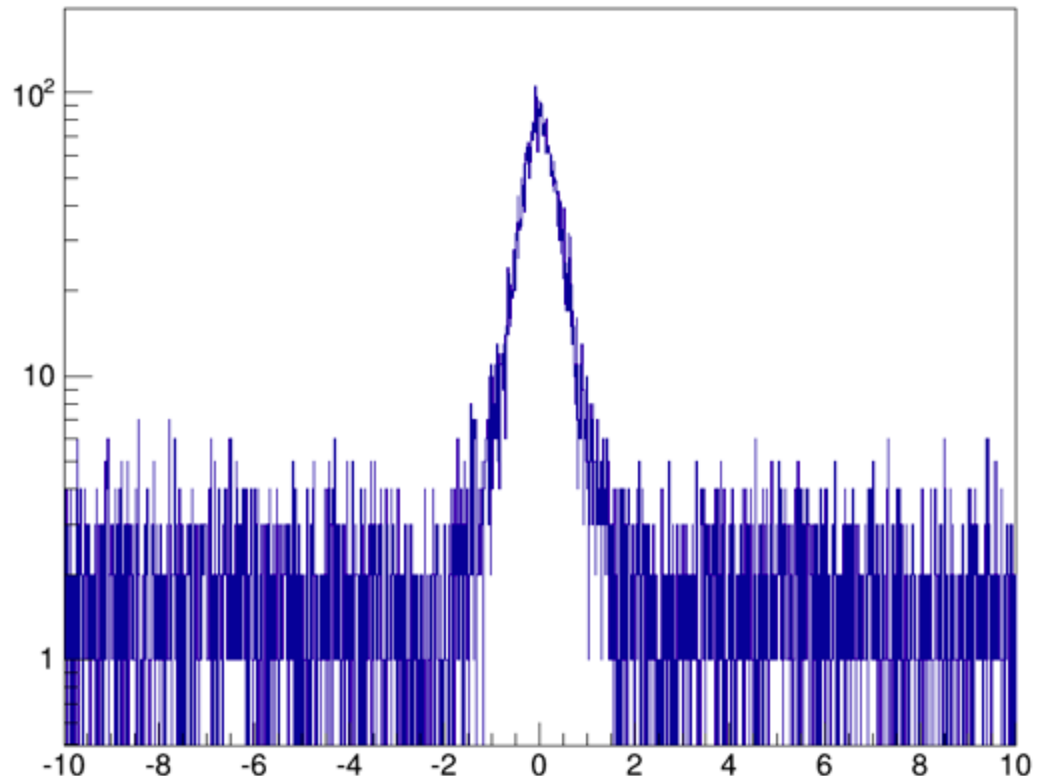


R. Karabowicz

GEM Event Reconstruction

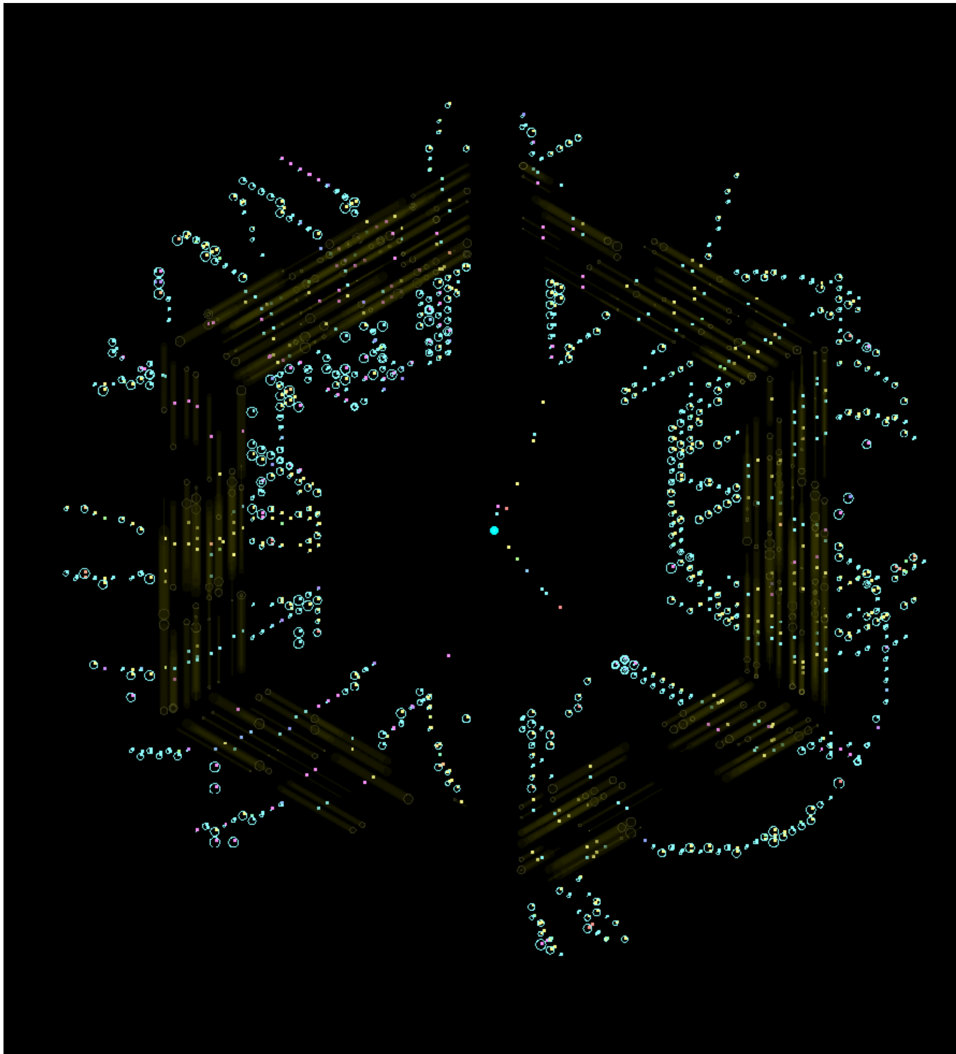
10000 DPM events simulated

- 8165 events with reconstructable track in GEM tracker
- 7536 events Reconstructed (92.3%)
- 139 ghost events (1.8%)



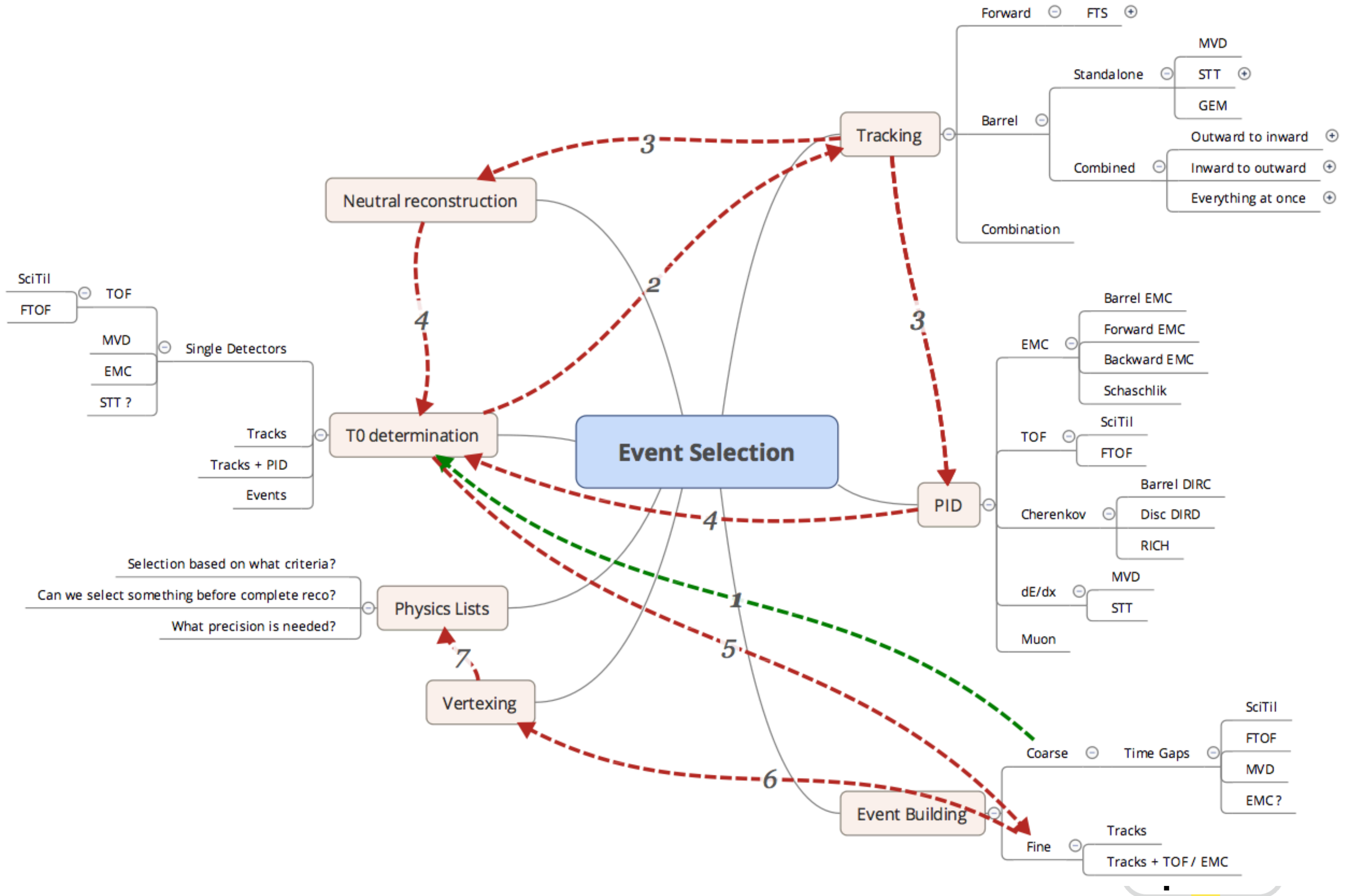
R. Karabowicz

Event Display



- Event selected by MVD via time gap
- STT hits matching to MVD time

Event Building



HOW TO IMPLEMENT IT

JÜLICH
FORSCHUNGSZENTRUM

STORING THE DATA

STEP 1

- Check the data object you want to store:
  - Does it derive form `FairTimeStamp` ?
  - Does it have an `operator<<` ?
  - Does it have an `bool operator< (const yourClass& name) const` ?
  - Does it have an `equal(FairTimeStamp*)` method?
    - *The equal method should return true if two identical detector elements are compared (e.g. the same pixel, pad, strip, straw, …)*
    - *It should not check if the data in the element is identical as well*

- An example can be found in PndSdsDigiPixel

STEP 2

JÜLICH FORSCHUNGSZENTRUM

- Derive your own buffer class from FairWriteoutBuffer
- In the class header you have to add:
  - `std::map<yourDataClass, double> fData_map;`
  - yourWriteoutBuffer::yourWriteoutBuffer(TString branchName, TString folderName, Bool_t persistance): FairWriteoutBuffer(branchName, “yourDataClass", folderName, persistance)
- Implement the pure virtual methods:
  - ```
    void AddNewDataToTClonesArray(FairTimeStamp* data)
    {
      FairRootManager* ioman = FairRootManager::Instance();
      TClonesArray* myArray = ioman-> GetTClonesArray(fBranchName);
      new ((*myArray)[myArray->GetEntries()]) yourclass(*(yourclass*)(data));
    }
    ```
  - ```
    Double FindTimeForData(FairTimeStamp* data)
    {
      std::map<yourclass, double>::itertor it;
      yourclass myData = *(yourclass)data;
      it = fData_map.find(myData);
      if (it == fData_map.end())
           return -1;
      else return it->second;
    }
    ```

panda

STEP 2b

- Implement the pure virtual methods:
  - ```
    void FillDataMap(FairTimeStamp* data, double activeTime)
    {
      yourclass myData = *(yourclass*)data;
      fData_Map[myData] = activeTime;
    }
    ```
  - ```
    void EraseDataFromDataMap(FairTimeStamp* data)
    {
      yourclass myData = *(yourclass*)data;
      if (fData_map.find(myData) != fData_map.end())
              fData_map.erase(fData_map.find(myData));
    }
    ```

Modify

- Overwrite `Modify(…)` if you want to have a different PileUp behavior (optional)
  - `Modify(…)` is called if data should be written into the buffer of an element which already exists in the buffer (e.g. same pixel hit a second time)
  - The standard behavior is that the new data is just ignored
  - Input is the old data already in the buffer and the new data which goes into the buffer
  - Output is a vector of data which will be stored in the buffer plus the new active time of the data

STEP 3

- Open your digi task:
  - Add to the header file:
    - `YourWriteoutBuffer* yourBufferName;`
    - `void RunTimeBased(){fTimeOrderedDigi = kTRUE;}`
    - `Bool_t fTimeOrderedDigi;`
  - Replace in Init() the creation of the TClonesArray for your data class by::

```
yourBufferName = new YourWriteoutBuffer(OutputBranchName, FolderName, Persistance);


yourBufferName = (YourWriteoutBuffer*)ioman->
                    RegisterWriteoutBuffer(OutputBranchName, youBufferName);
yourBufferName->ActivateBuffering(fTimeOrderedDigi);
```

STEP 3

- Replace in Exec() the filling of the TClonesArray of your data object by:

```
yourBufferName->FillNewData(yourData, theAbsoluteStartTimeOfTheData,
theAbsoluteActiveTimeOfTheData);
```

- The writeout buffer replaces the usual TClonesArray used to store the data

- If the variable fTimeOrderedDigi is set to kFALSE the behaviour of the buffer is identical to the standard TClonesArray storage of data

STEP 4

JÜLICH FORSCHUNGSZENTRUM

- Add to your CMakeLists.txt and …LinkDef.h file the new classes
- Make sure that you store a TimeStamp including error with your data!

- Now add in your digi macro the line
  `fRun->SetEventMeanTime(yourValue);`
  to assign a time to the events.
- To use now the buffer you just need to set in your digi macro
  `yourTask->RunTimeBased();`
- This data has to go to a root file before you can continue analysing the digitized data!

- That's it.

STEP 5

- Compile it, run it, test it
- If you look at the output data you should see that the digis are not any longer stored in the same entry of the branch as the MC data but at later entries.
- You should have one entry more in the digi file as in the MC file

STEP 6

- For the FairRingSorter only the method `CreateElement()` has to be overwritten
- This method creates a new element of the object which was passed by a FairTimeStamp pointer to the method

- Here is an example:

```
FairTimeStamp*
PndSdsDigiPixelRingSorter::CreateElement(FairTimeStamp* data) {
        return new PndSdsDigiPixel(*(PndSdsDigiPixel*)data);
}
```

STEP 7

JÜLICH FORSCHUNGSZENTRUM

- For the FairRingSorterTask two methods have to be overwritten: `InitSorter` and `AddNewDataToTClonesArray`
- Here is an example of InitSorter:

```
FairRingSorter* PndSdsDigiPixelSorterTask::InitSorter
         (Int_t numberOfCells, Double_t widthOfCells)
{
   return new PndSdsDigiPixelRingSorter(numberOfCells, widthOfCells);
}
```

- And for AddNewDataToTClonesArray:

```
void PndSdsDigiPixelSorterTask::AddNewDataToTClonesArray
         (FairTimeStamp* data)
{
 FairRootManager* ioman = FairRootManager::Instance();
 TClonesArray* myArray = ioman->GetTClonesArray (fOutputBranch);
 new ((*myArray)[myArray->GetEntries()])PndSdsDigiPixel
         (*(PndSdsDigiPixel*)(data));
}
```

panda

Test it

- Compile it.
- Add to your digi task the new sorter task after your digitization task.
- Run your digi macro.
- Compare the sorted digis with the unsorted digis. In which entry of the Tree are they stored?
- What happens if you change the size of the RingSorter?

STEP 8

JÜLICH FORSCHUNGSZENTRUM

- Add in your reco task header:
  `#include "FairTSBufferFunctional.h"` and
  `BinaryFunctor* fFunctor; //!`

- In `Init()` of your task cxx file add:
  `fFunctor = new StopTime();`

- In `Exec()` of Task:

```
if (FairRunAna::Instance()->IsTimeStamp()){
   fDigiArray = FairRootManager::Instance()->
                GetData(yourDigiBranchName, fFunctor,
                FairRunManager::Instance()->GetEventTime()
                + 10);
}
```

panda

STEP 8

JÜLICH FORSCHUNGSZENTRUM

- At the end of `Exec()` add:
  - `fHitarray->Sort();`
  - `fDigiArray->Delete();`

panda

STEP 9

JÜLICH FORSCHUNGSZENTRUM

- To activate the reading back of the data via the functors you have to add `fRun->RunWithTimeStamps()` in your reco macro.
- If you leave `RunWithTimeStamps()` away `GetData(…)` returns the data as it is stored in the entries of your digi branch
- Run your macro.
  - What happens if you set RunWithTimeStamps or leave it away?
  - What happens if you modify the functor parameters?
  - Test the other functor.

MVD Examples

- Examples of the time ordered simulation can be found in /macro/mvd/TimeOrderedSim

- To check that everything is working you should start a TreeViewer and scan the TimeStamp variable of your data. If it is randomized before your RingSorter and it is sorted afterwards it works.