



Time based simulation

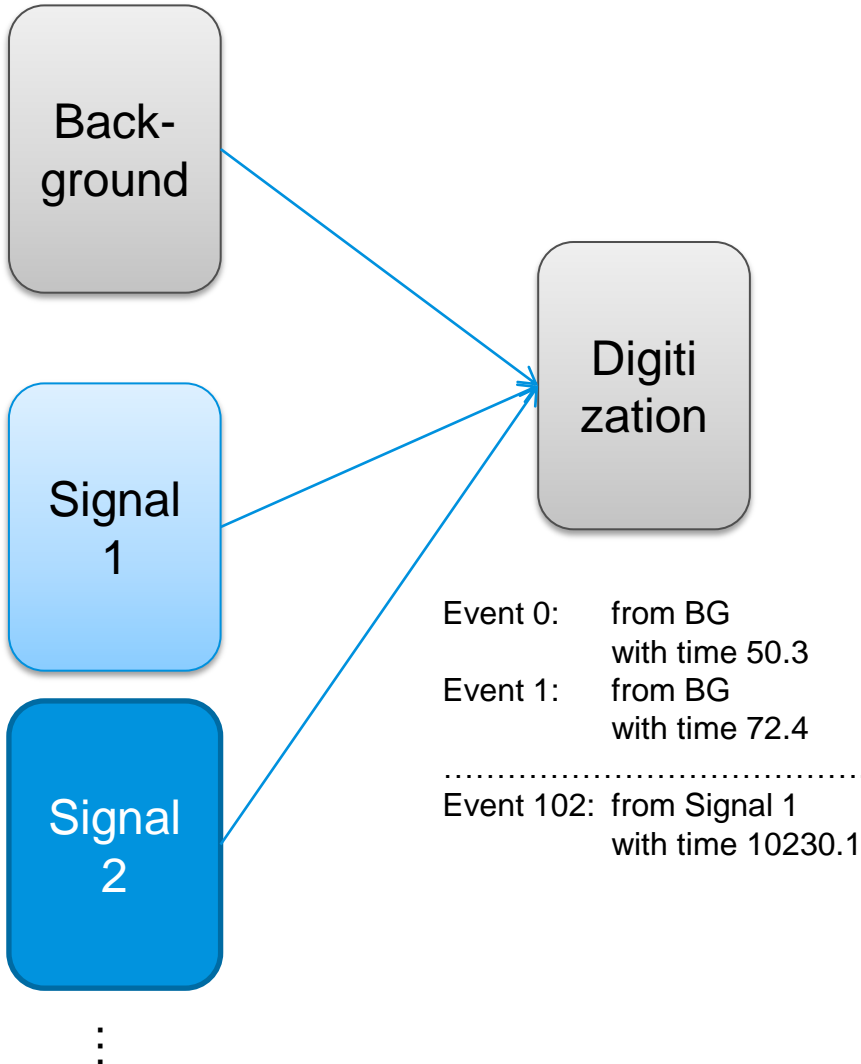
Workshop in Torino 25.07.2012

27. Juli 2012

| Tobias Stockmanns

Event Mixing (Mohammad's work)

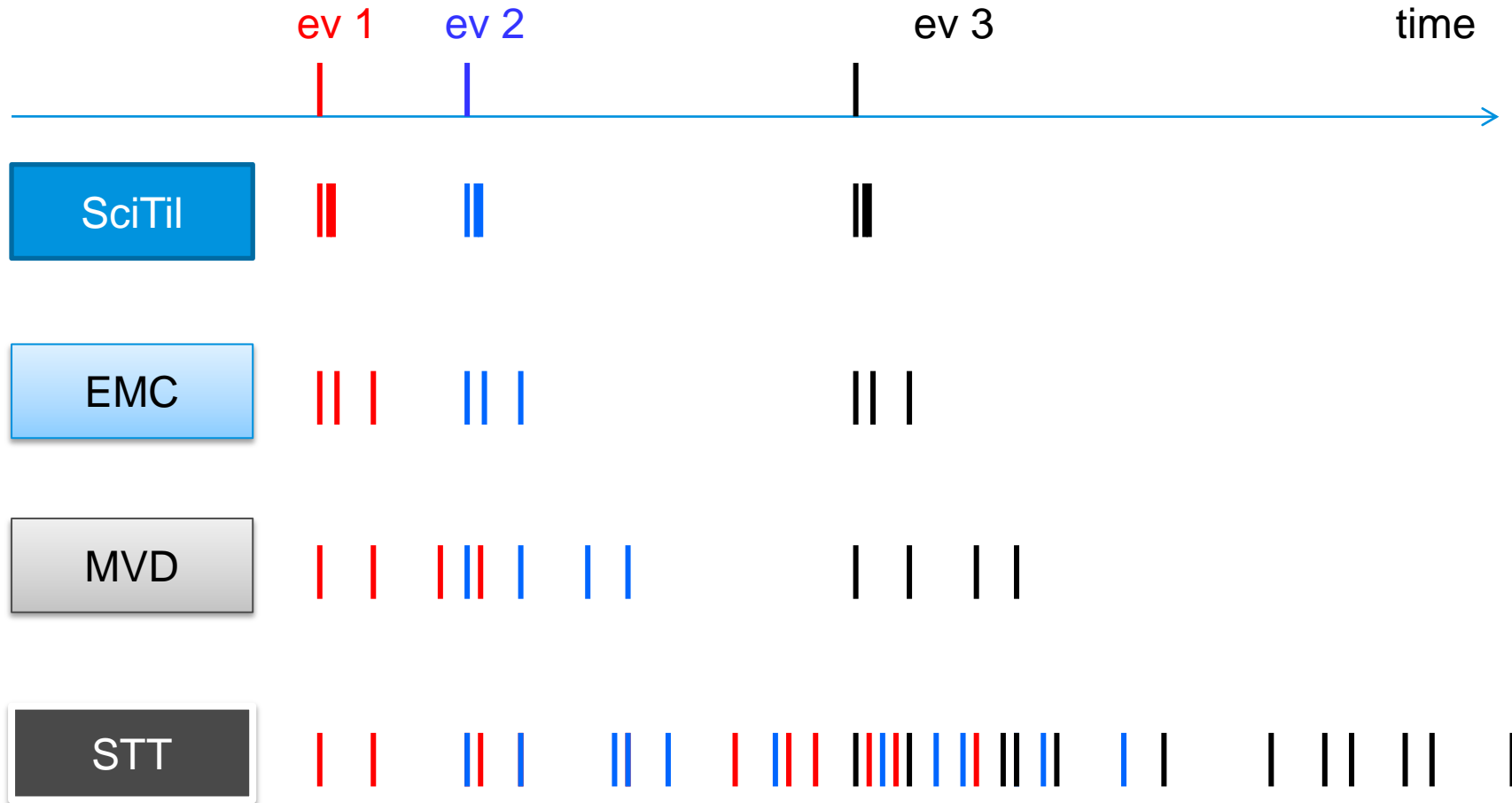
MC Files:



- 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
- See presentation of Mohammad (last EVO Meeting)
- Not topic of this workshop

- Problem: In the current version of the software 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



 = detector hits from different events

Event Structure and Detector Response



STT



 = detector hits from different events

Event Structure and Detector Response



Wanted data structure

STT



Data structure in Tree

Event 1



Event 2



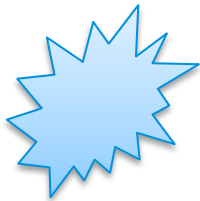
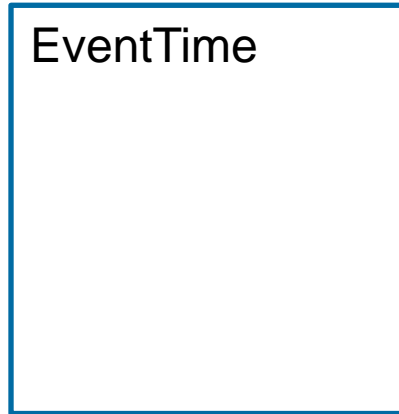
Event 3



·
·
·

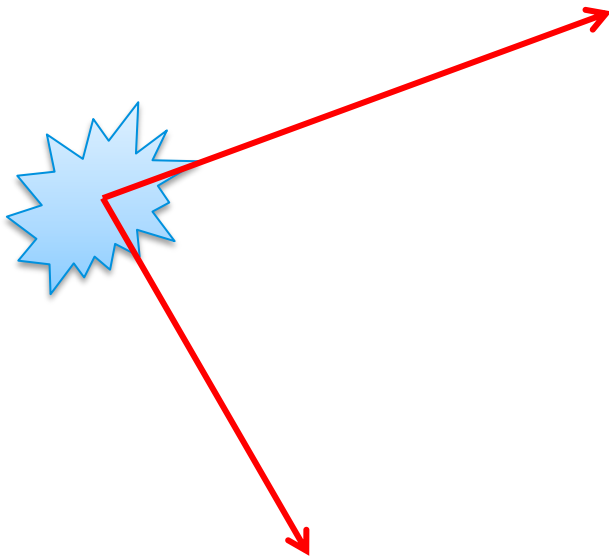
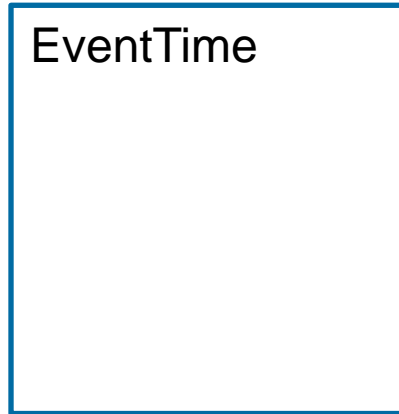
 = detector hits from different events

Times to take into account

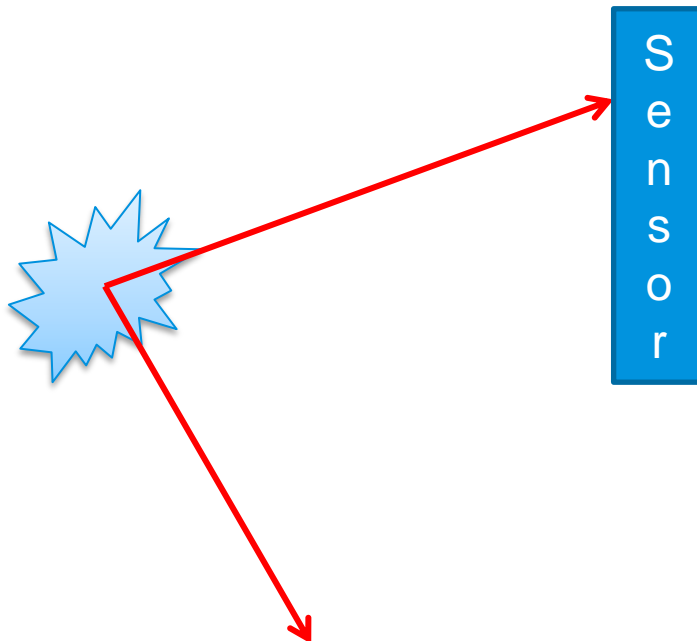
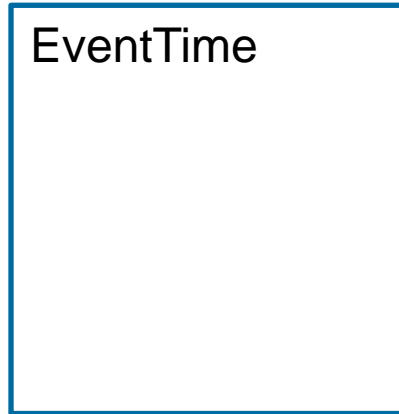


EventTime [ns]

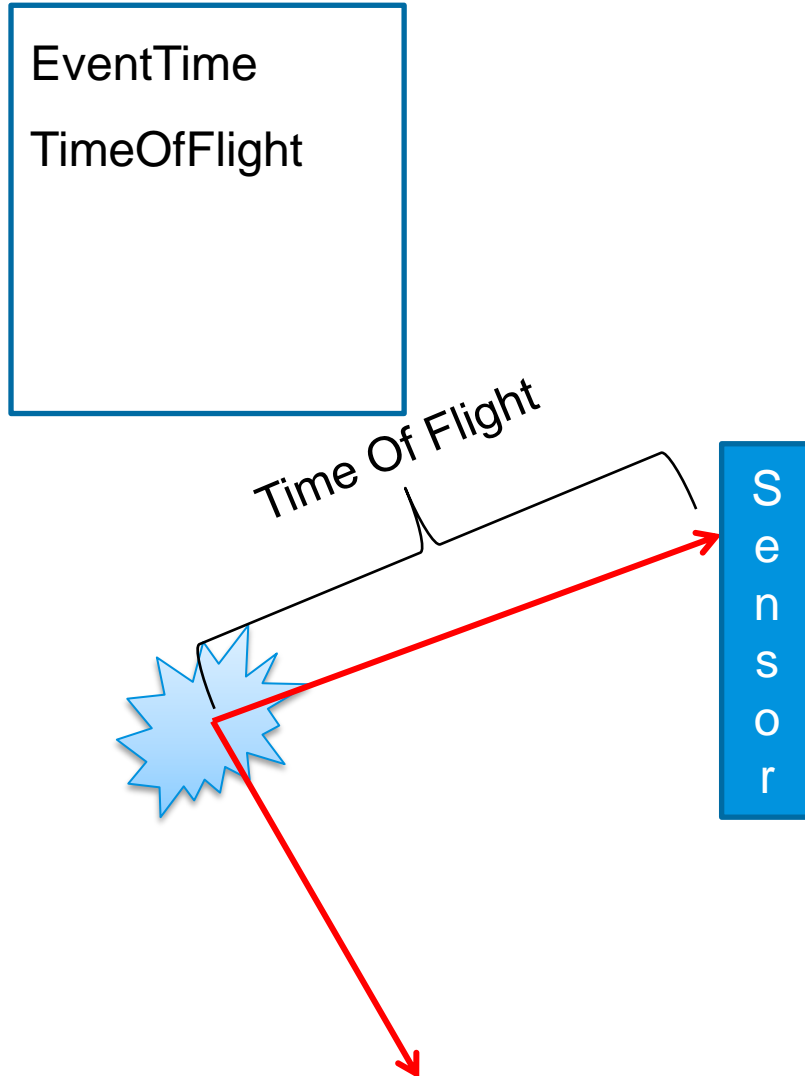
Times to take into account



Times to take into account

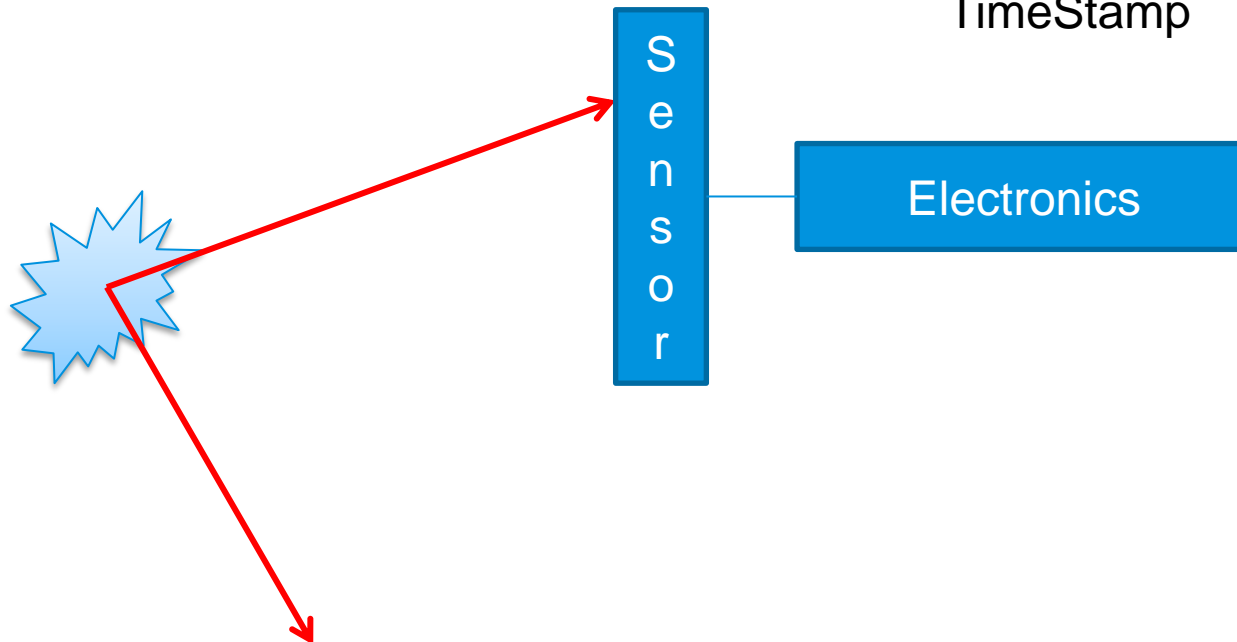
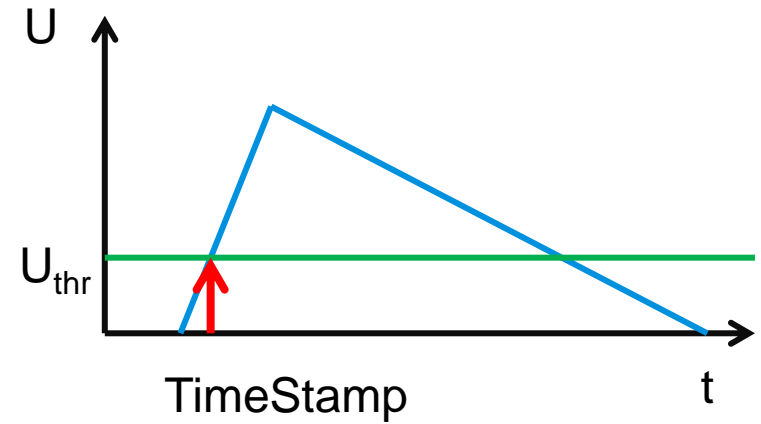


Times to take into account



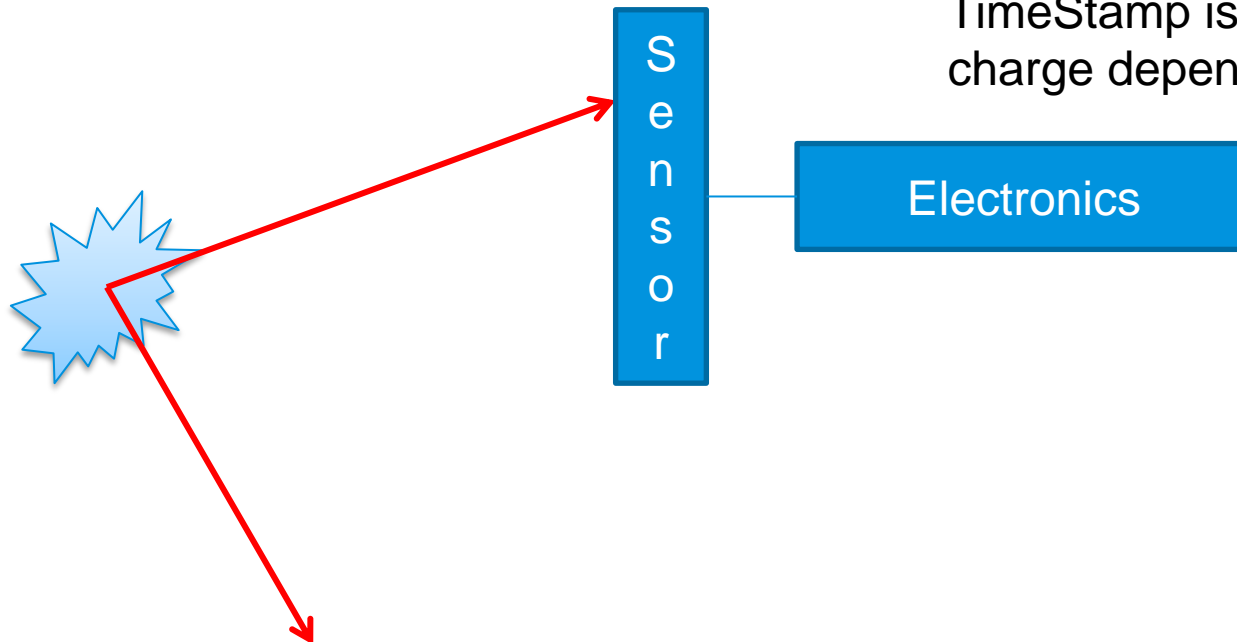
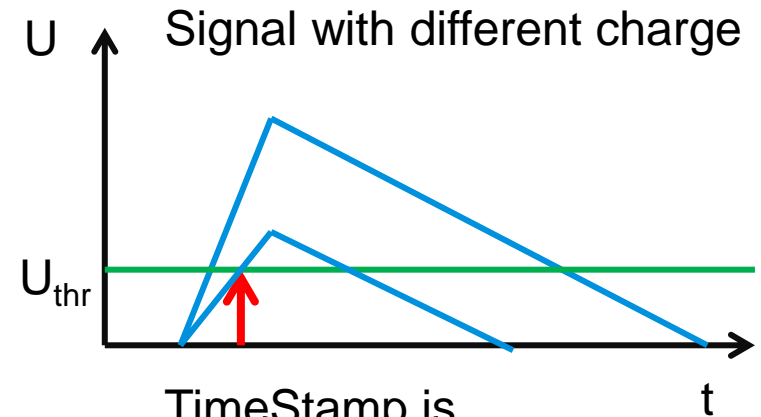
Times to take into account

EventTime
TimeOfFlight
TimeStamp



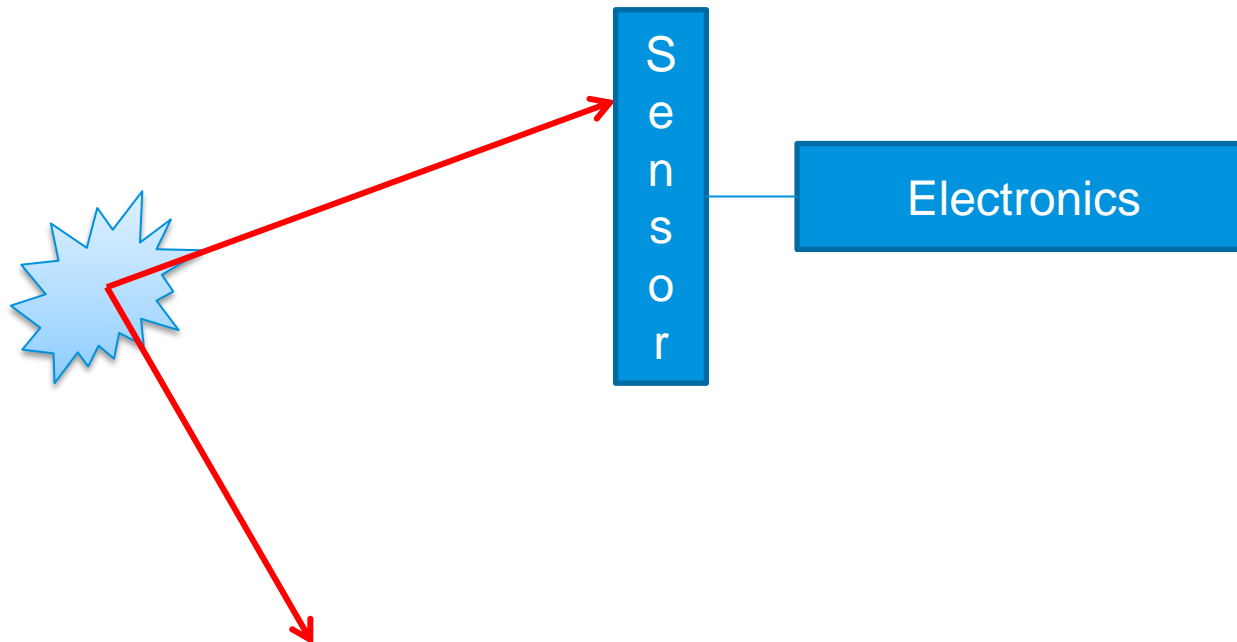
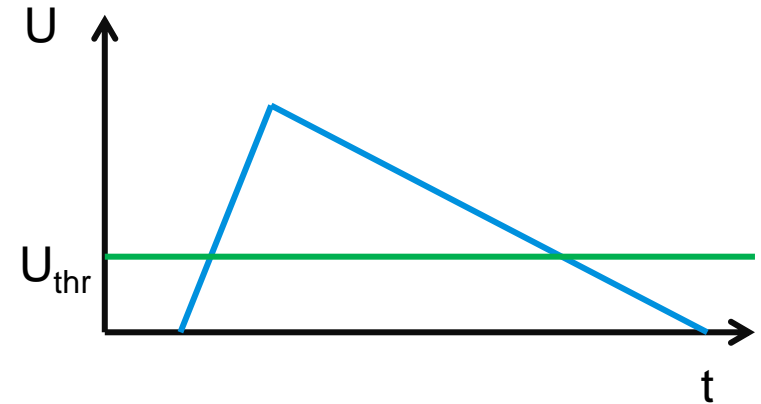
Times to take into account

EventTime
TimeOfFlight
TimeStamp



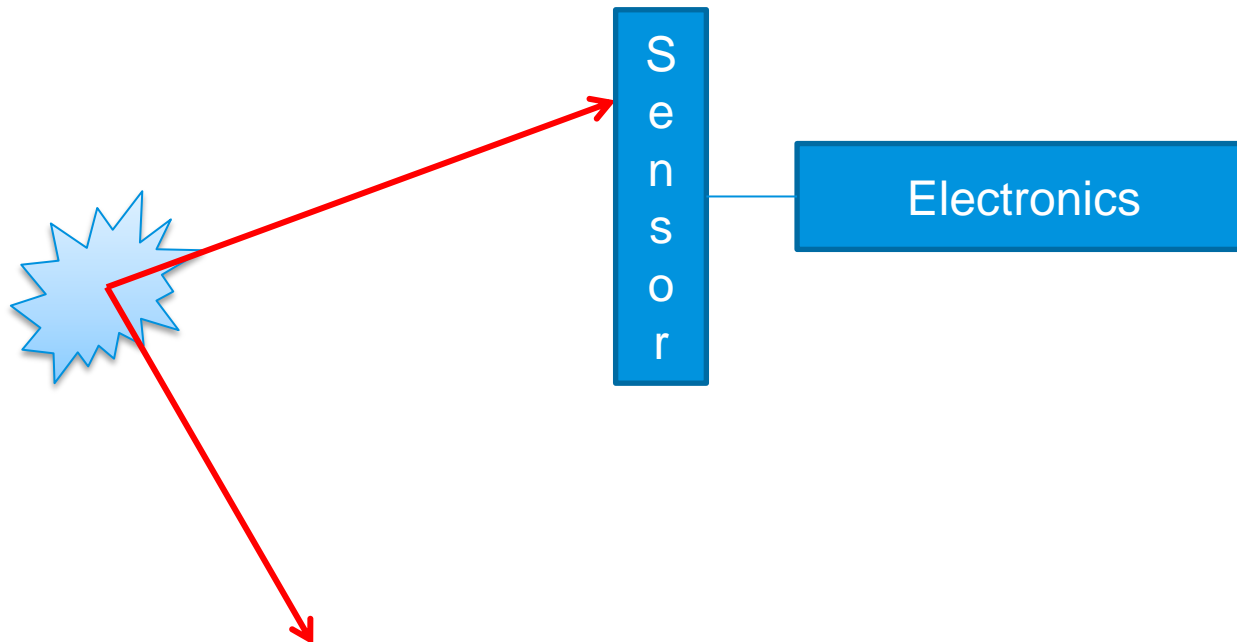
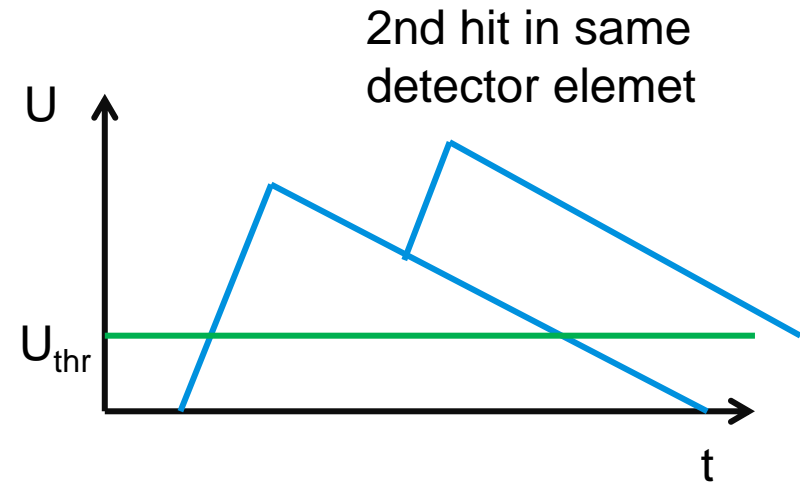
Times to take into account

EventTime
TimeOfFlight
TimeStamp



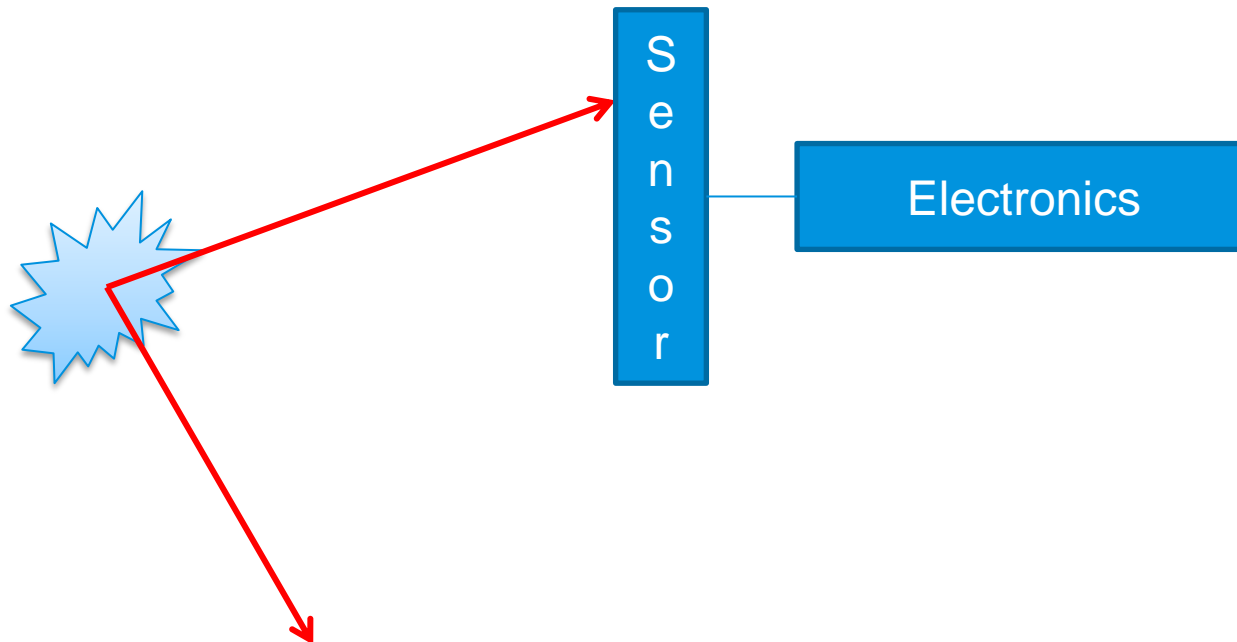
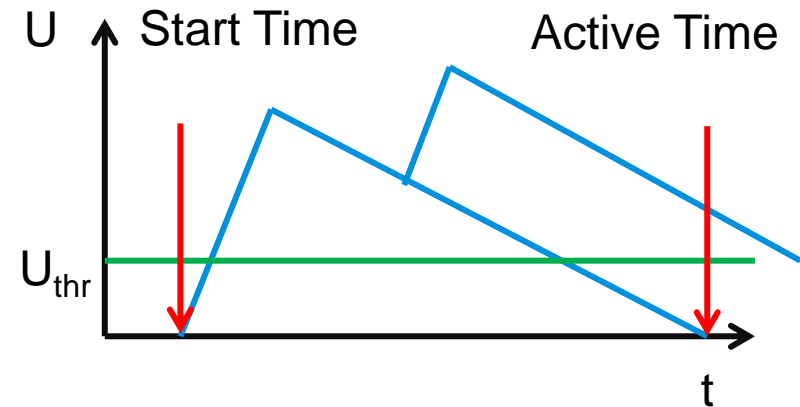
Times to take into account

EventTime
TimeOfFlight
TimeStamp



Times to take into account

EventTime
TimeOfFlight
TimeStamp
Start/ActiveTime



- 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 ()`

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

- 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 , Electronics)*
 - *Resolution and offset depends on individual detector*

- 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 happend during the active time of a previous it is also strongly detector dependent (hit lost, new hit modified, old hit modified, new hits created, ...)*
 - *Absolute time!*

- 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
- Let's start implementing it

STORING THE DATA

- Commit your changes on your torinoDetector into the SVN
- Switch to:
<https://subversion.gsi.de/fairroot/pandaroot/development/tstoc/kmanns/Torino/version4b/torinoDetector> and
<https://subversion.gsi.de/fairroot/pandaroot/development/tstoc/kmanns/Torino/version4b/macro/torinoDetector>
- Compile the code.
- Have a look at the new classes `PndTorinoDetectorDigiTask` and `PndTorinoDetectorRecoTask`.
What are they doing?
- Run the macros `run_sim.C`, `run_digi.C`, `run_reco.C`
- Look at the results. Are they what you expect?

- Check the data object you want to store:
 - Does it derive from `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`

- 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 persistence): FairWriteoutBuffer(branchName, "yourDataClass", folderName, persistence)`
- 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>::iterator it;
    yourclass myData = *(yourclass)data;
    it = fData_map.find(myData);
    if (it == fData_map.end())
        return -1;
    else return it->second;
}
```

- 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));
}
```

- 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

- 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,  
Persistence);
```

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

- 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

- 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!

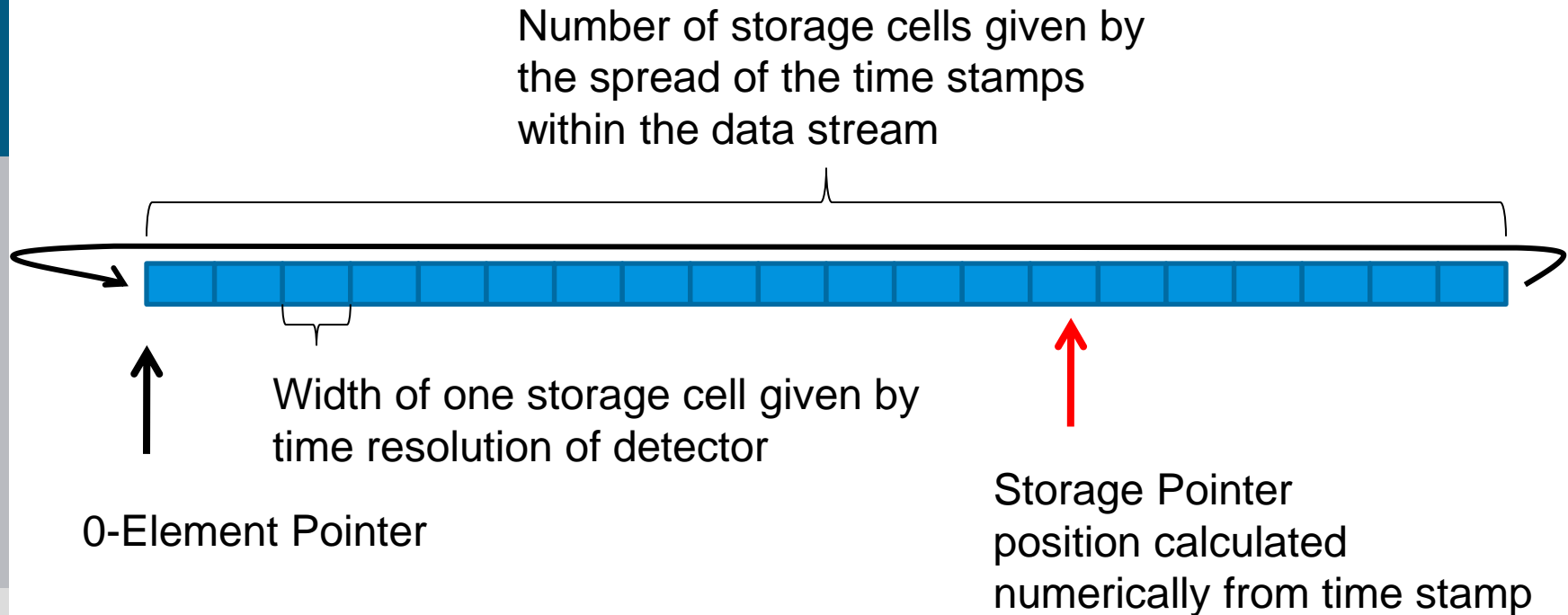
- That's it.

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

SORTING THE DATA

- 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



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

- 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);  
}
```

- 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)));
}
```

- 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?

READING BACK THE DATA

- 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

- 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

- 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);  
}
```

STEP 8

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

- 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.

- 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.