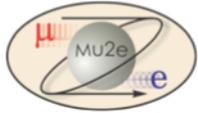


Mu2e-doc-1874-v1



# Event Mixing

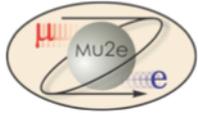
Rob Kutschke, Fermilab  
Software and Simulation Meeting  
October 5, 2011



# The Problem



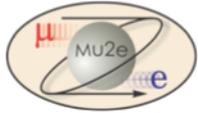
- Per proton pulse ( Mu2e-doc-1174, Mu2e-doc-1865)
  - 35,988 DIO
  - 5,560 ejected protons
  - 10,850,000 beam electrons
  - and lots more
- How do we simulate all of this in time with a single conversion electron?
  - CPU time required is huge, hours per event.
  - Memory limits, especially if we want to retain the full history of leading to all hits.
- The key to the answer:
  - Only a tiny fraction of background particles create any hits.



# The Solution



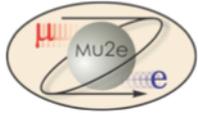
- Generate large samples of each background process
  - One generated particle per event.
  - With one exception, use the full phase space:
    - $t > 500$  ns
  - Send these events through G4.
  - Write out only events with a least one StepPointMC in the detector(s) of interest.
- When simulating signal events
  - Read in or generate one conversion electron
  - Overlay appropriate number of events from each of many background streams.



# DIO Examples



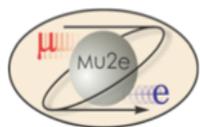
- My jobs: /mu2e/data/outstage/kutschke/72785
  - Single track events generated: 250,000,000
  - Events with >0 hits in tracker: 129,060
  - 0.000516 events with tracker hits per DIO
- From Gianni:
  - 35,988 DIO per proton pulse (Mu2e-doc 1774-v2)
  - 22,456 with  $t > 500$  ns (62.4%)
- $22,456 \times 0.000516 = 11.6$ 
  - Draw **11.6 events** per conversion electron from my DIO background files.



# Ejected Proton examples

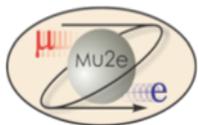


- My jobs: /mu2e/data/outstage/kutschke/72784
  - Single track events generated: 70,000,000
  - Events with >0 hits in tracker: 297,063
  - 0.00424 events with hits in tracker per ejected proton
- From Gianni: Mu2e-doc 1774-v2
  - 5,560 ejected protons per proton pulse
  - 3,513 with  $t > 500$  ns (62.4%)
- $5,560 \times 0.00424 = 14.9$ 
  - Draw **14.9 ejected protons** per conversion electron from my ejected proton background files.

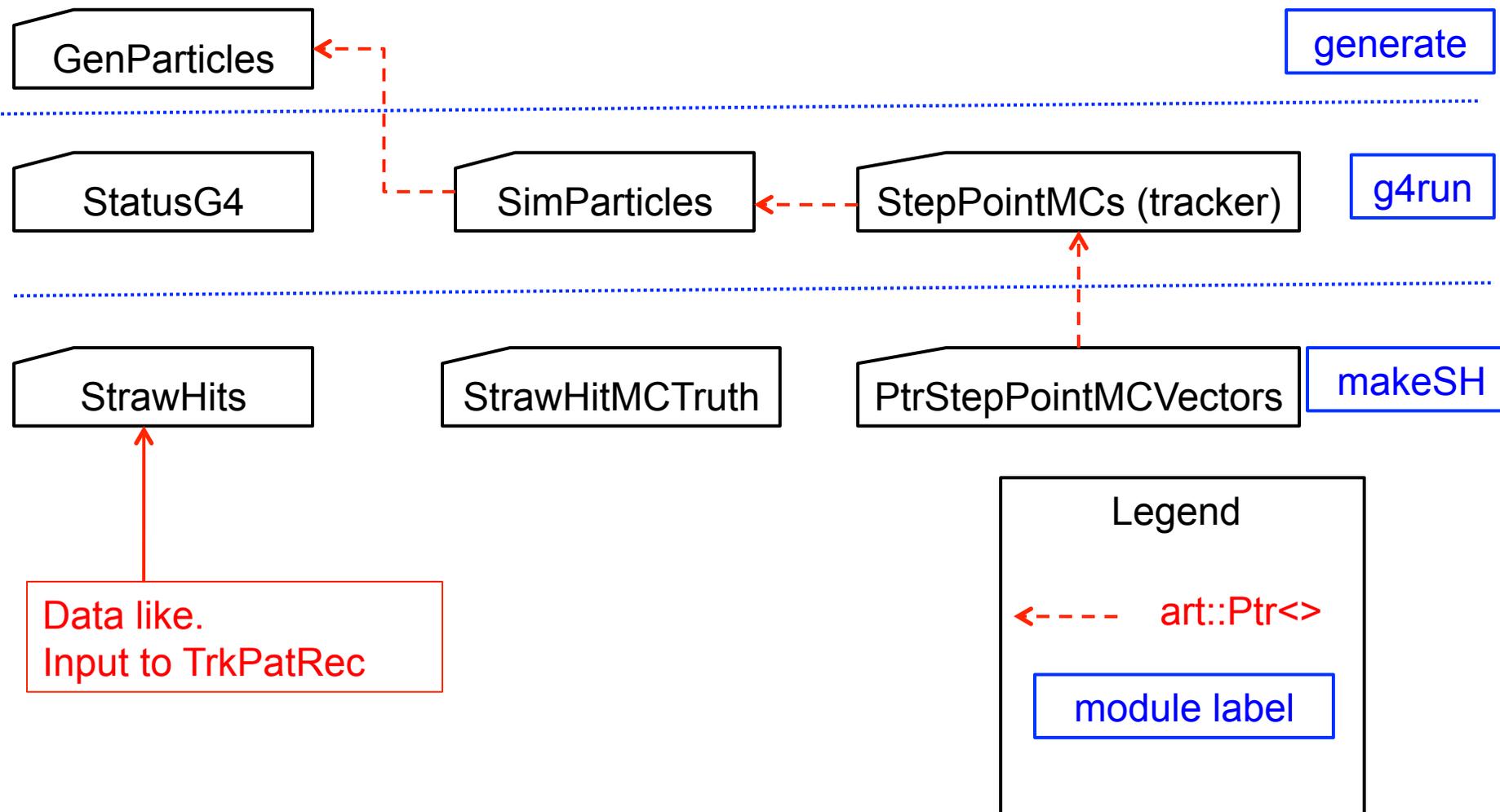


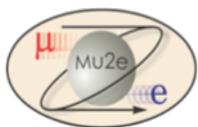
# The Data Product View





# Data Products Related to StrawHits





# Event Mixing



DIO Background File

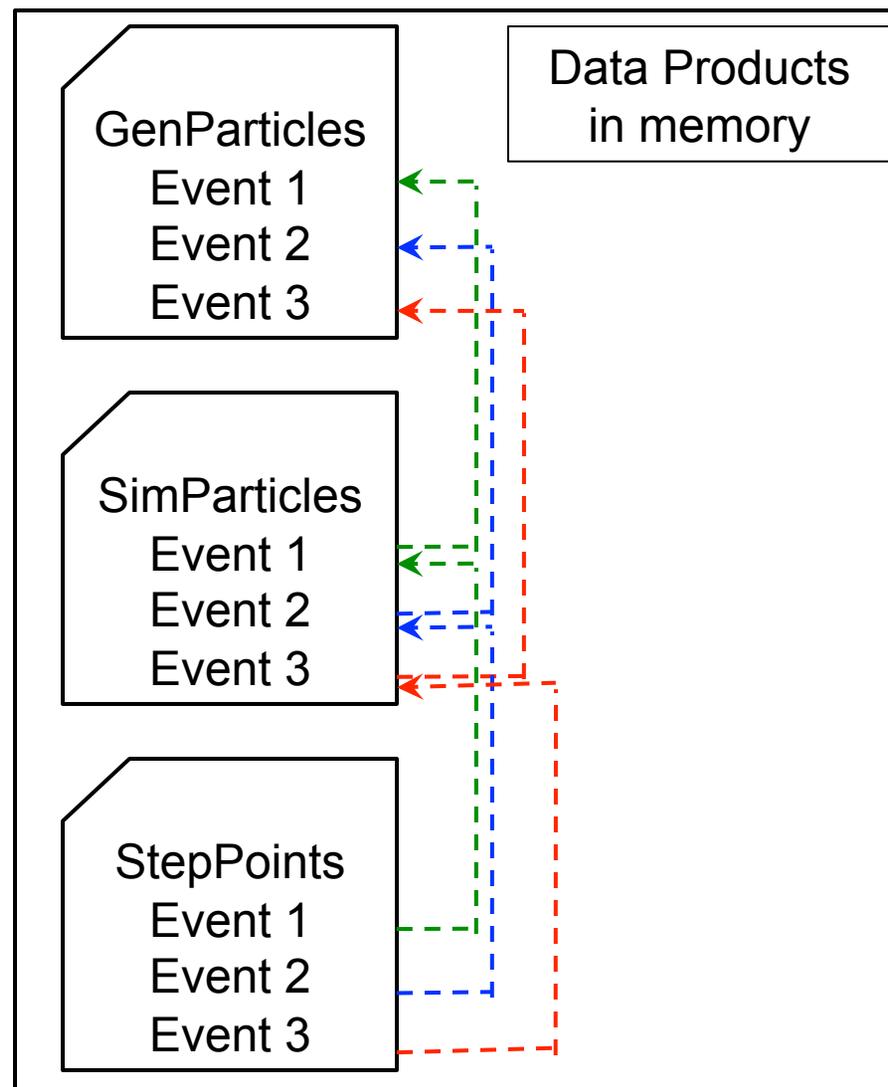
|         | Gens | Sims | Steps | StatusG4 |
|---------|------|------|-------|----------|
| Event 1 |      |      |       |          |
| Event 2 |      |      |       |          |
| Event 3 |      |      |       |          |
| Event 4 |      |      |       |          |

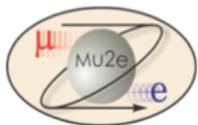
For each signal event, choose N events from the DIO background file, mix the N GenParticleCollections in the file into 1 in memory.

And so on for other data products.

Renumbers items for events 2, 3 ...

Therefore: reseal all art::Ptr objects.





# Example: StepPointMCs



DIO Background File

|         | Gens | Sims | Steps | StatusG4 |
|---------|------|------|-------|----------|
| Event 1 |      |      | 3     |          |
| Event 2 |      |      | 2     |          |
| Event 3 |      |      | 4     |          |

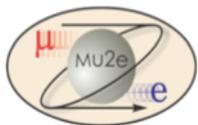
Size of StepPointMCCollection



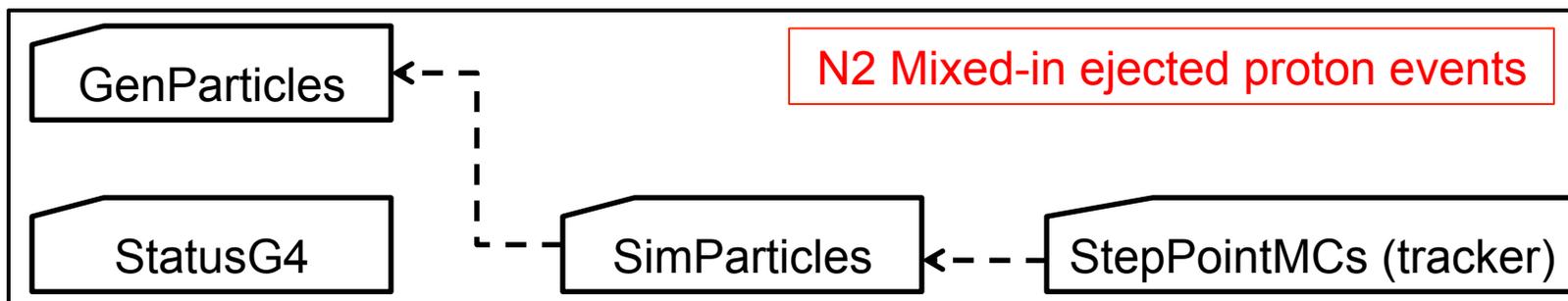
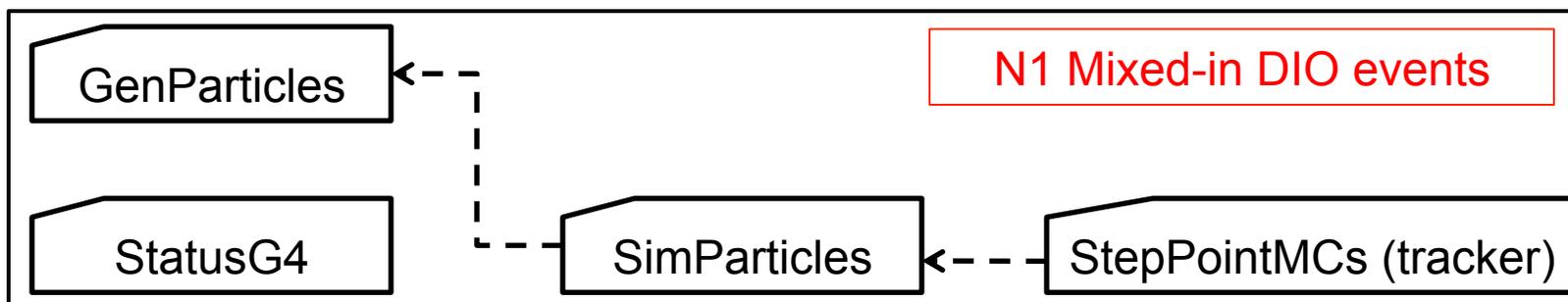
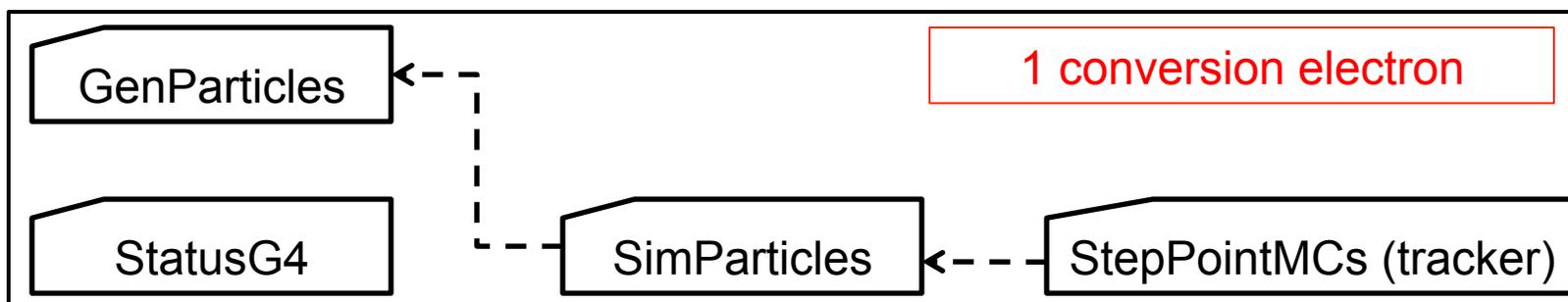
StepPointMCCollection in Memory

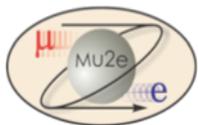
| StepPointMCs |         |
|--------------|---------|
| 0            | – (1,0) |
| 1            | – (1,1) |
| 2            | – (1,2) |
| 3            | – (2,0) |
| 4            | – (2,1) |
| 5            | – (3,0) |
| 6            | – (3,1) |
| 7            | – (3,2) |
| 8            | – (3,3) |

$(n,m) =$   
( event # in bg file, index into the collection )

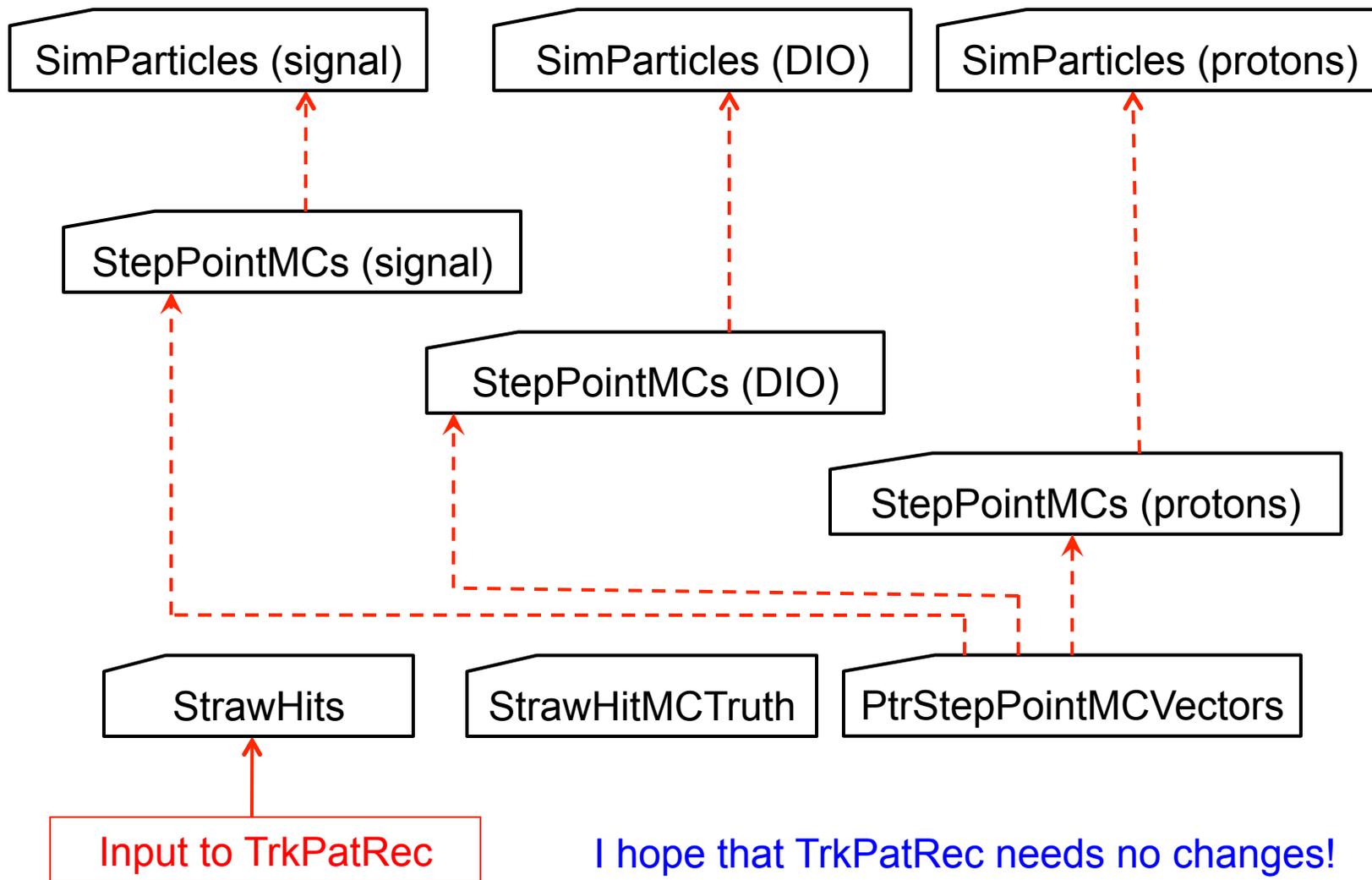


# Mixing: Signal, DIO, Ejected Protons

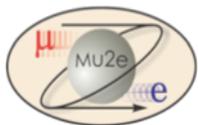




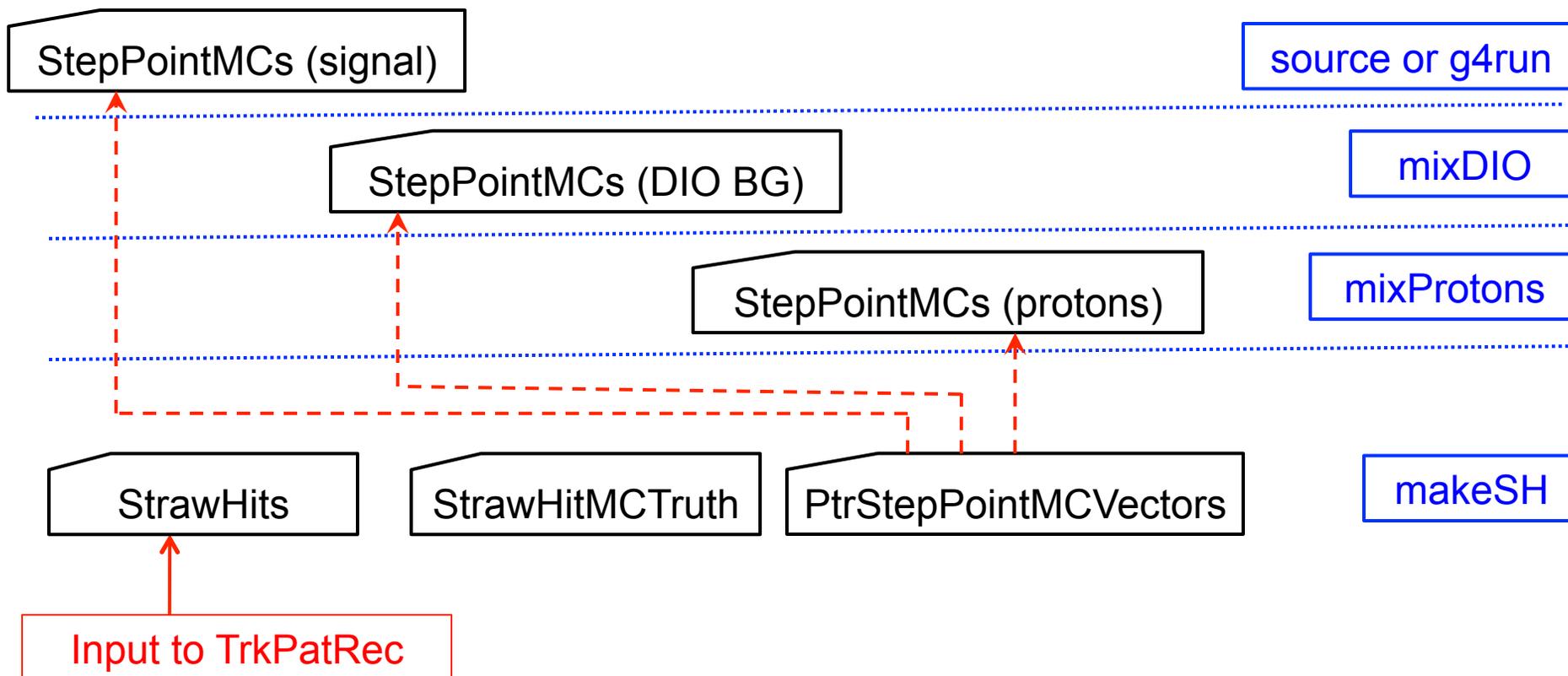
# StrawHits From Multiple StepPointMCs

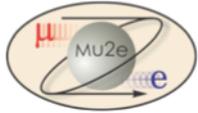


I hope that TrkPatRec needs no changes!

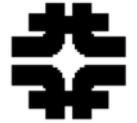


# Another View



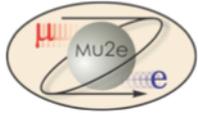


# Configuring a Mixing Module



```
dioMixer: {  
  module_type : MixMCEvents  
  mean        : 11.6  
  fileNames   : [ "dioFile1.root", "dioFile2.root", "dioFile3.root" ]  
  readMode    : sequential  
  seed        : [13579]  
}
```

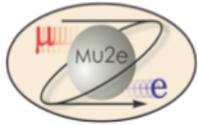
- Mean
  - If positive, specifies mean of a Poisson distribution.
  - If negative, mix in exactly |mean| events per primary event.
- fileNames
  - Arbitrary number allowed.
  - The job shutdown gracefully if inputs run out ( via a throw).



# Configuring a Mixing Module



- readMode:
  - sequential/random
  - I have not tried random.
- seed
  - Used to seed the Poisson distribution.
  - 99% sure it is optional.



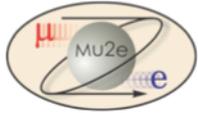
# Example .fcl File Fragment



```
physics :{
  producers:{ makeSH : @local::makeSH }

  filters: {
    dioMixer: {
      module_type : MixMCEvents
      mean         : 11.6
      fileNames    : [ "dioFile1.root", "dioFile2.root", "dioFile3.root" ]
      readMode     : sequential
      seed         : [13579]
    }
    protonMixer: {
      module_type : MixMCEvents
      mean         : 14.9
      fileNames    : [ "protonFile1.root", "protonFile2.root" ]
      readMode     : sequential
      seed         : [24680]
    }
  }
  p1 : [ dioMixer, protonMixer, makeSH ]   trigger_paths : [p1]
  e1 : [ outfile ]                       end_paths     : [e1]
}
```

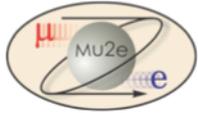
Suppose that source is a file of pure conversion electron events



# Comments on Previous Slides



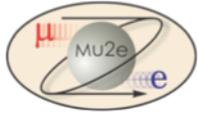
- Can mix-in arbitrary number of streams.
- Signal events can be:
  - Read from source
  - Generated in same job
  - Just another mixing stream ( with mean : -1 ).
  - No signal event, to get a pure BG sample.



## Why 2 Instances of MixMCEvents?



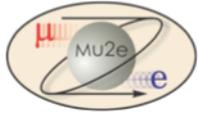
- Why not just one instance of a mixing module that can read multiple input streams?
- The existing solution has a clean separation of responsibilities:
  - Code related to persistency (ROOT IO) that can be written by computer professionals who are unaware of the internal details of Mu2e data product classes.
  - Mu2e specific pieces that can be written by us. We can be entirely unaware of the details of persistency.
- Other solutions, which allowed multiple input streams within one mixing module, compromised this separation of responsibilities.



# Separating Responsibilities



- See: `EventMixing/src/MixMCEvents_module.cc`
- Philosophy:
  - We write callback functions to do the Mu2e specific parts
  - We register the callbacks with an art-written module that does all of the art parts.
  - art does the rest of the work and calls our code as needed.
  - art also provides a toolkit to help us do our work.
- Our class: `mu2e::MixMCEventsDetail`
  - The callback functions are methods of this class.
- The mixing module class is:
  - `art::MixFilter<mu2e::MixMCEventsDetail>`

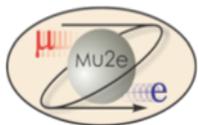


# Configuring a Mixing Module



```
dioMixer: {  
  module_type : MixMCEvents  
  mean        : 11.6  
  fileNames   : [ "dioFile1.root", "dioFile2.root", "dioFile3.root" ]  
  readMode    : sequential  
  seed        : [13579]  
}
```

- mean
  - Processed by mu2e code
- fileNames, readMode, seed
  - Processed by the art::MixFilter template
- I may add more arguments processed by our code.



# A Mixing Method

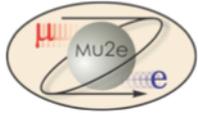


```
class MixMCEventsDetail {
    ....
    bool mixGenParticles ( .... );
    ....
private:
    ....
    std::vector<size_t> genOffsets_;
}

bool mu2e::MixMCEventsDetail::
mixGenParticles( std::vector< mu2e::GenParticleCollection const *> const& in,
                 mu2e::GenParticleCollection& out,
                 art::PtrRemapper const & ){

    art::flattenCollections(in, out, genOffsets_);

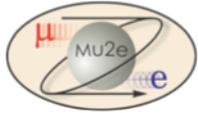
    return true;
}
```



## Comments on Previous Slide



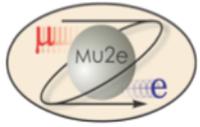
- `genOffsets_` records the boundaries, within the output collection, at which the new events begin.
  - Used in downstream code to update `art::Ptr<GenParticle>`.



# Summary and Conclusions

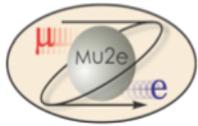


- Event mixing is working but still has a few rough edges.
  - Requires the art v0\_7\_16
  - art v0\_7\_16 is available for mu2egpvm02 but not the SLF4 machines.
- Expect it to be ready for use in a few days.



# Backup Slides

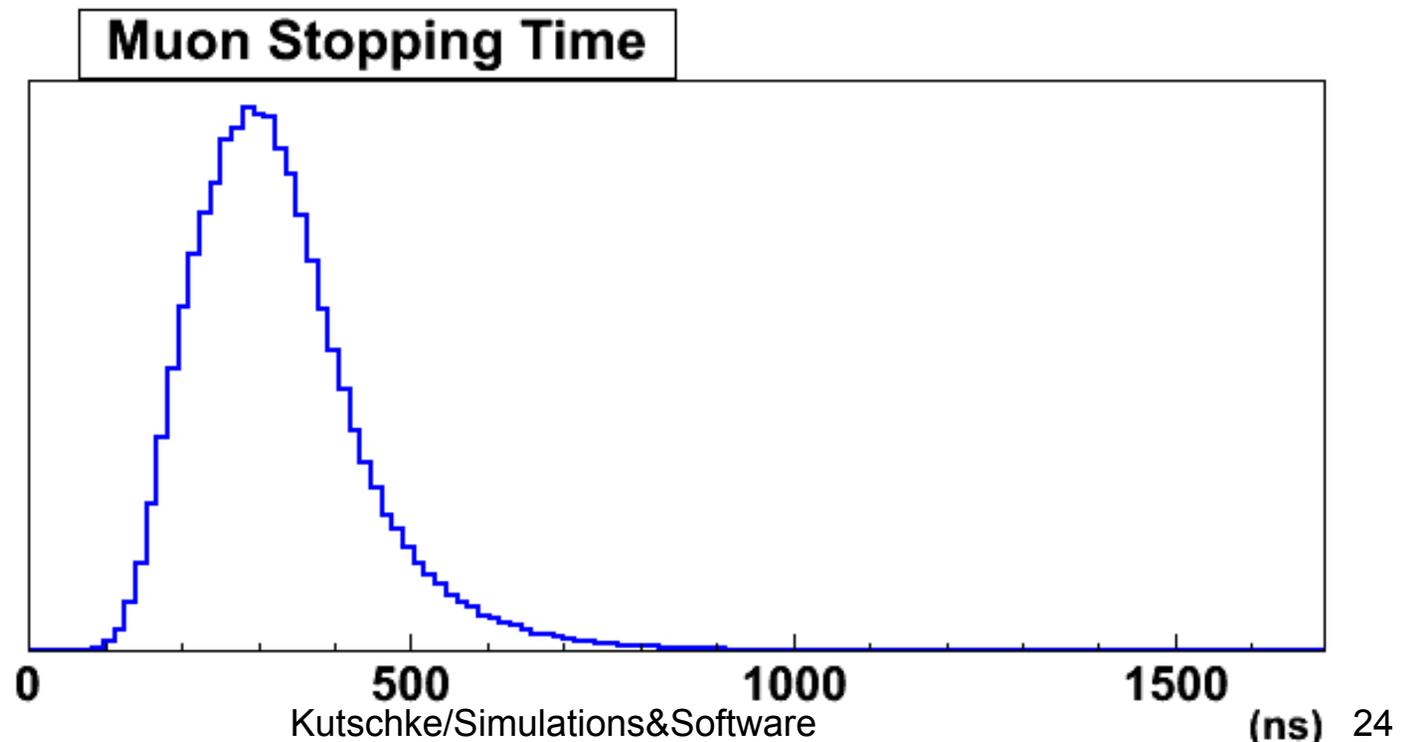


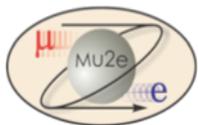


## Steps 1 and 2

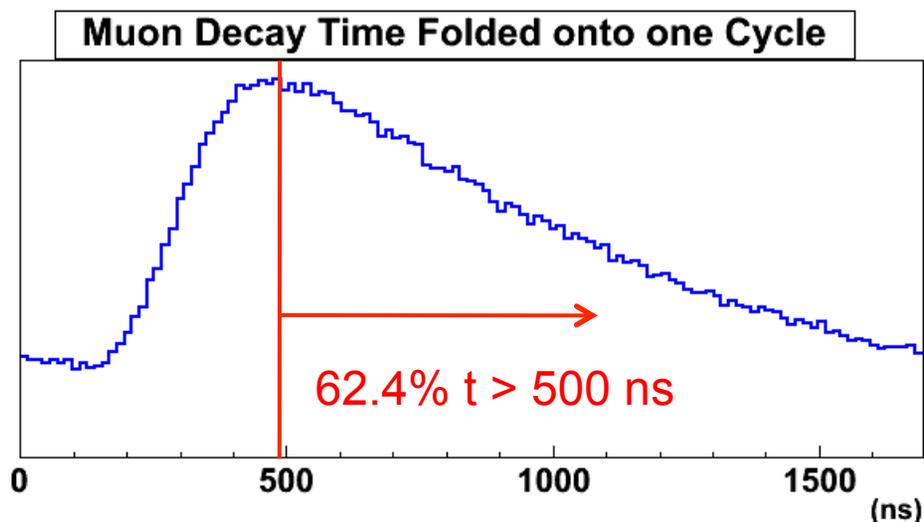
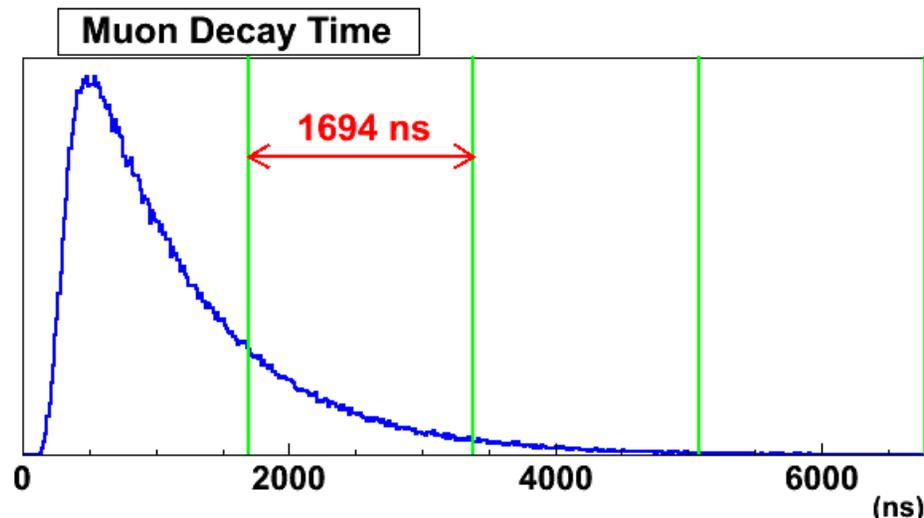


- Step 1: protons on production target
  - Save muons entering DS
- Step 2: output of step 1.
  - Save time and position of muons stopping in foils





# Generated Time Distribution



- Muon stopping time convoluted with an exponential decay.
- ... and with proton pulse shape if needed.
- Fold time into the first cycle of the muon beamline.
- Restart from the beginning if  $t < 500$  ns.