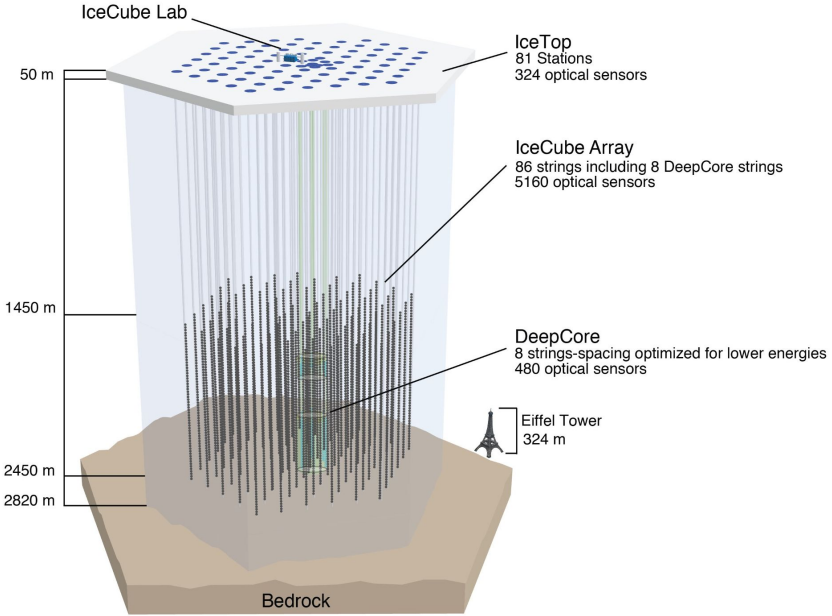
IceTray

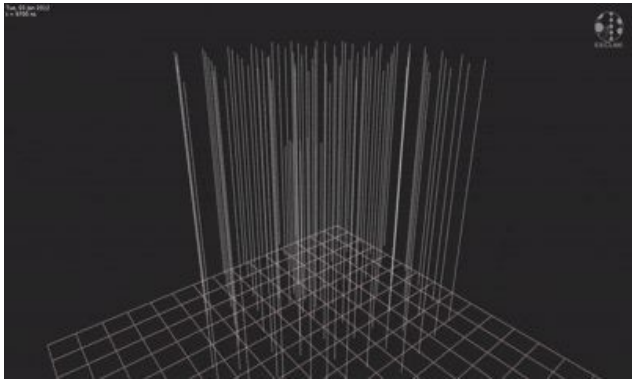
Hannah Erpenbeck, Maxwell Nakos

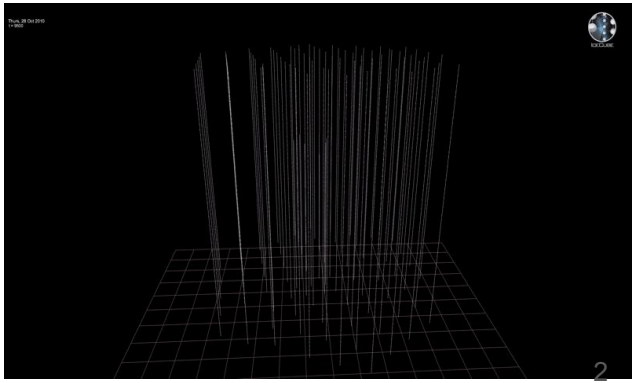
Presentation heavily adapted from Kevin Meagher

From IceCube the detector to neutrino events

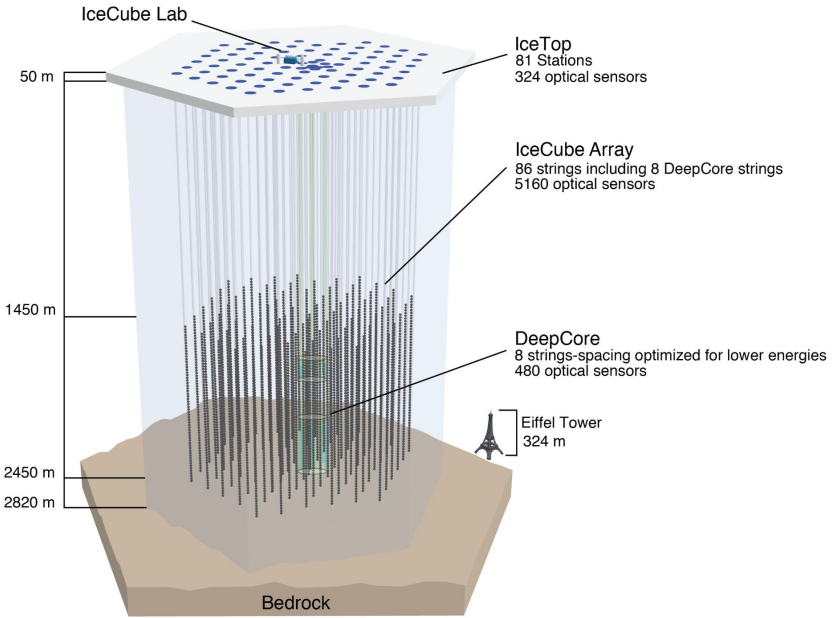


?





IceTray - A Swiss Army knife for IceCube analysis.



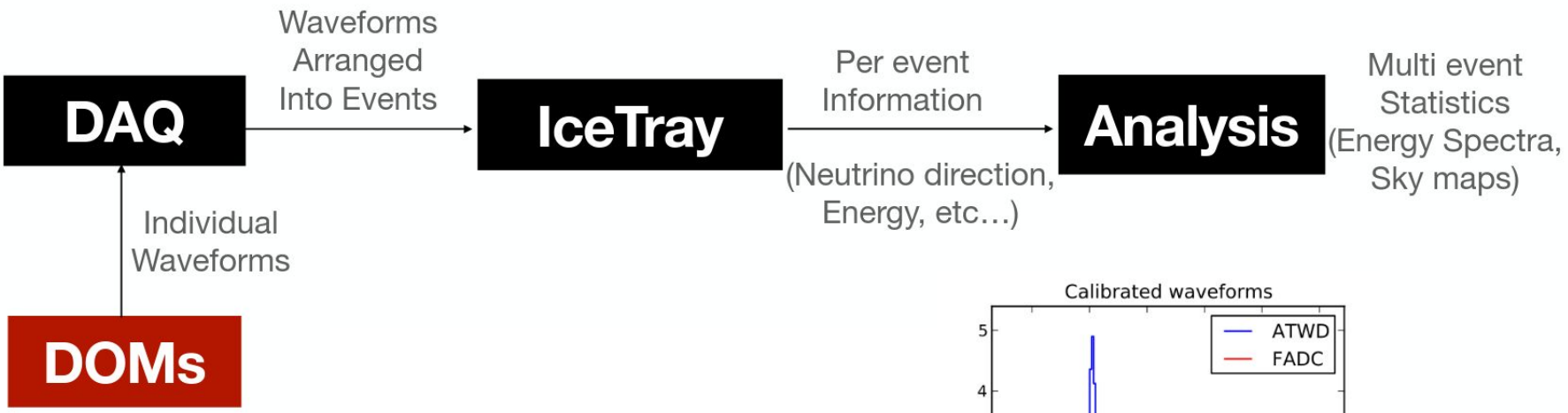
IceTray is IceCube's Framework for serial processing of IceCube Data.

- 5160 DOMs send information
- One event ~ 10 μs

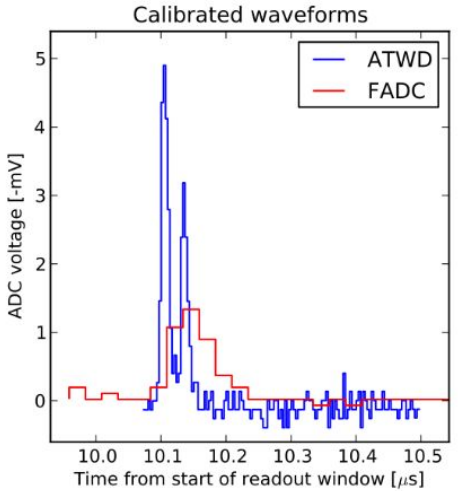
Reconstruct the event including energy and direction

https://docs.icecube.aq/icetray/main/info/quickstart.html

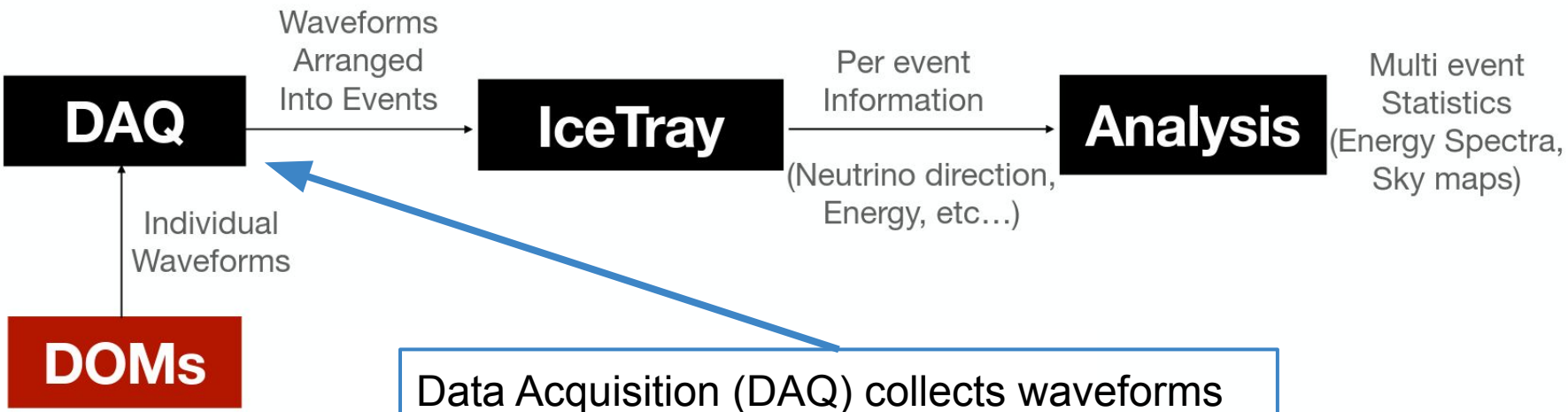
Overview of IceCube Processing Pipelines



DOMs record light from the PMT and save a time series as a waveform

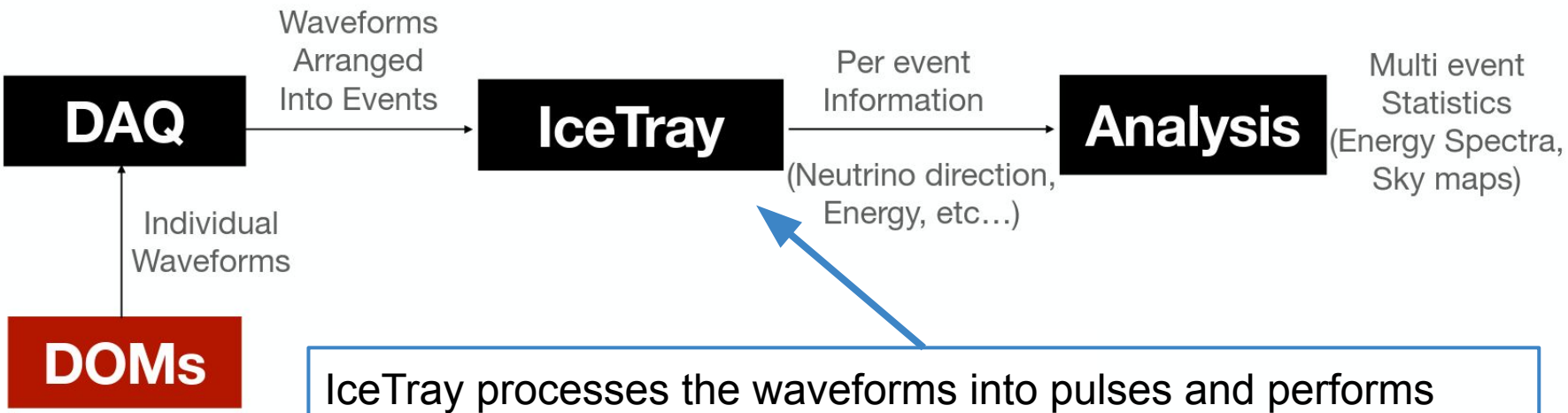


Overview of IceCube Processing Pipelines



Data Acquisition (DAQ) collects waveforms from DOMs and arranges them into events that last ~10μs

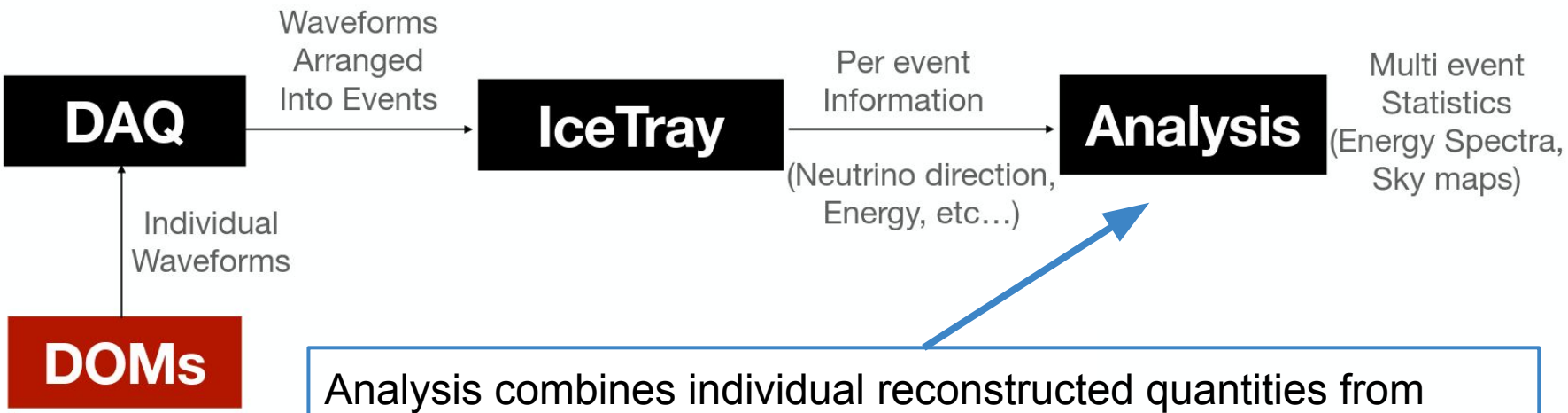
Overview of IceCube Processing Pipelines



IceTray processes the waveforms into pulses and performs reconstructions on those pulses to record information about the specific event.

Examples include: Zenith, azimuth, total charge, muon energy

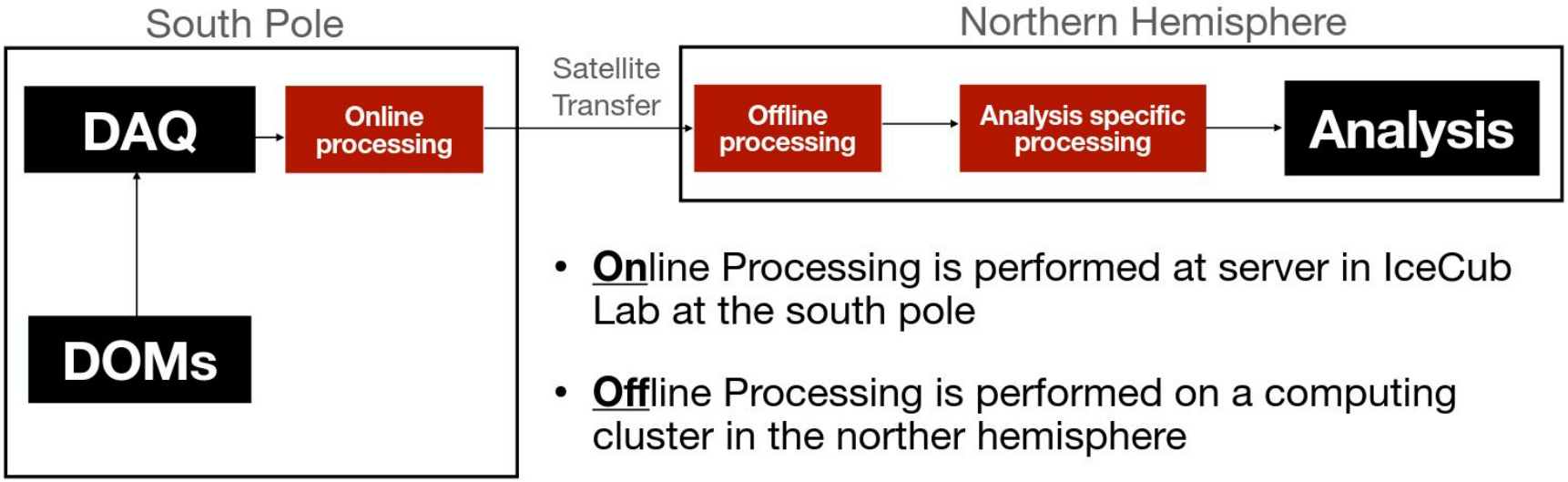
Overview of IceCube Processing Pipelines



Analysis combines individual reconstructed quantities from multiple events and combines them into a statistic

Example: Zenith and azimuth from all the events to form a skymap or energy from each event to form an energy spectrum

More detailed view of processing



- **On**line Processing is performed at server in IceCub Lab at the south pole
- **Off**line Processing is performed on a computing cluster in the norther hemisphere
- Most analyses require additional processing beyond what is provided by offline processing, usually handled by working groups

Getting Help

IceTray is a framework that has grown a lot over time

- Works great for IceCube
- Very specifically designed for IceCube
- Many people have used it for more than 10 years and still need help
- There are experts for every part of it

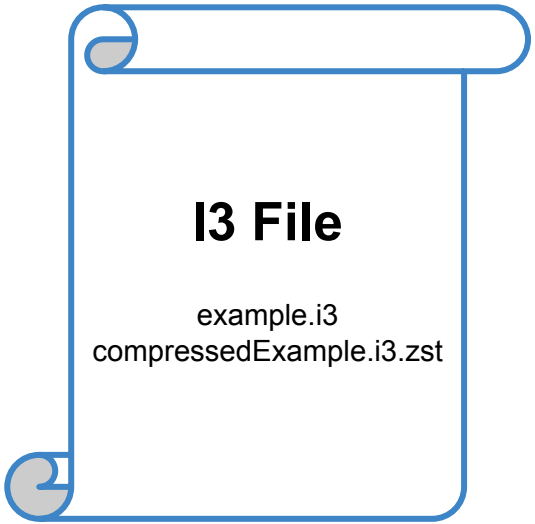
IceTray Documentation : https://docs.icecube.aq/icetray/main/

Ask for help on slack: **#software**

If documentation is missing or unclear or incorrect please file an issue on github

Detailed Look at IceTray

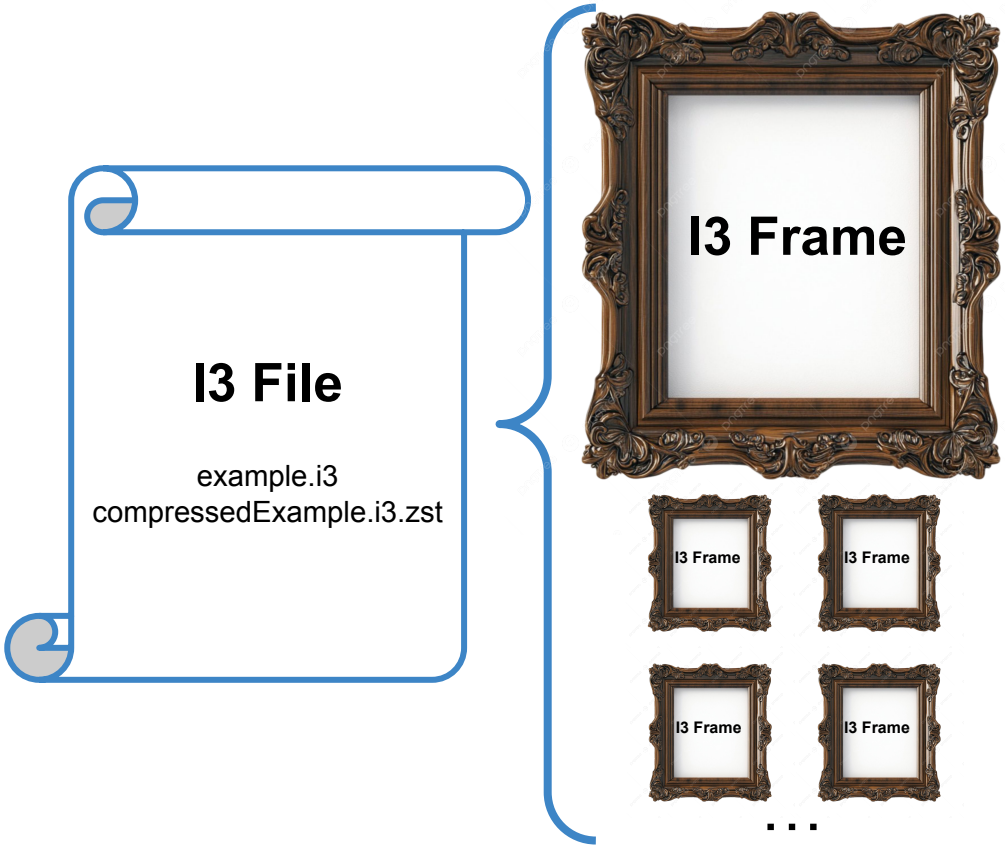
Building Blocks of IceTray



- Standard file format for IceCube data
- Used for both experimental data and simulation
- I3 files are used for serial processing of data
- The frame is the fundamental unit of an i3 file

https://docs.icecube.aq/icetray/main/projects/dataio/portable_binary_archive.html#i3file

Building Blocks of IceTray



- I3Frames are a data container that stores all information about a particular event (~10μs)
  - Raw waveforms, processed pulses, and reconstruction results
- Any data structure that IceTray supports can be put into a frame
- Every object in the frame has a name or key
- I3Frames are what is written to disk (in I3Files) to save data

Building Blocks of IceTray

I3 File

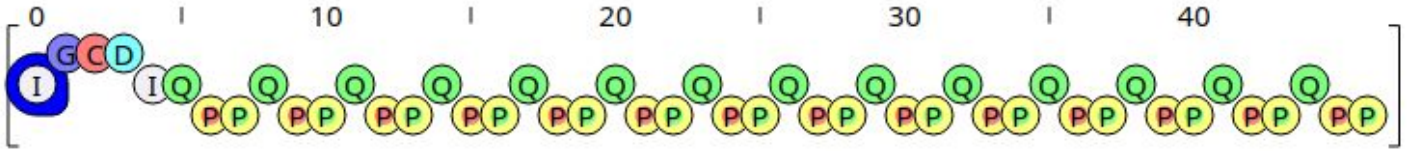
example.i3
compressedExample.i3.zst

I3Frame Types

- I I3Frame::TrayInfo
- G I3Frame::Geometry
- C I3Frame::Calibration
- D I3Frame::DetectorStatus
- P I3Frame::Physics
- Q I3Frame::DAQ

and others (special)

I3 Frames



Metadata (usually once per file):

"I" (TrayInfo): Information on how the file was previously processed

Detector and Run specific information:

"G" (Geometry): Geometric coordinates of each DOM

"C" (Calibration): Calibration constants relating to the photomultiplier tubes

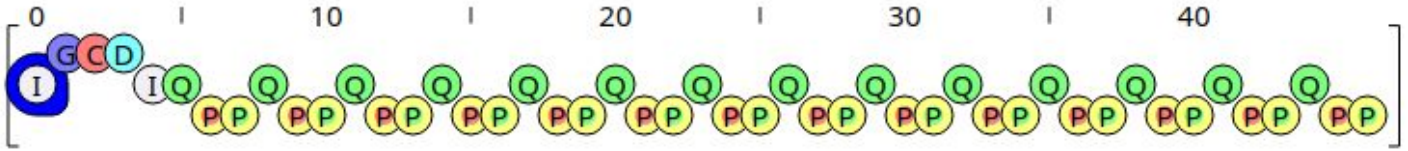
"D" (Detector Status): How the data acquisition system was configured for this particular (8 hour) run. E.g. which DOMs were on.

Event Info: (~10 us of data):

"Q" (DAQ): Waveforms recorded for this event

"P" (Physics): High level pulse information and reconstructions

GCD File



There is a GCD file for every run or simulation set.

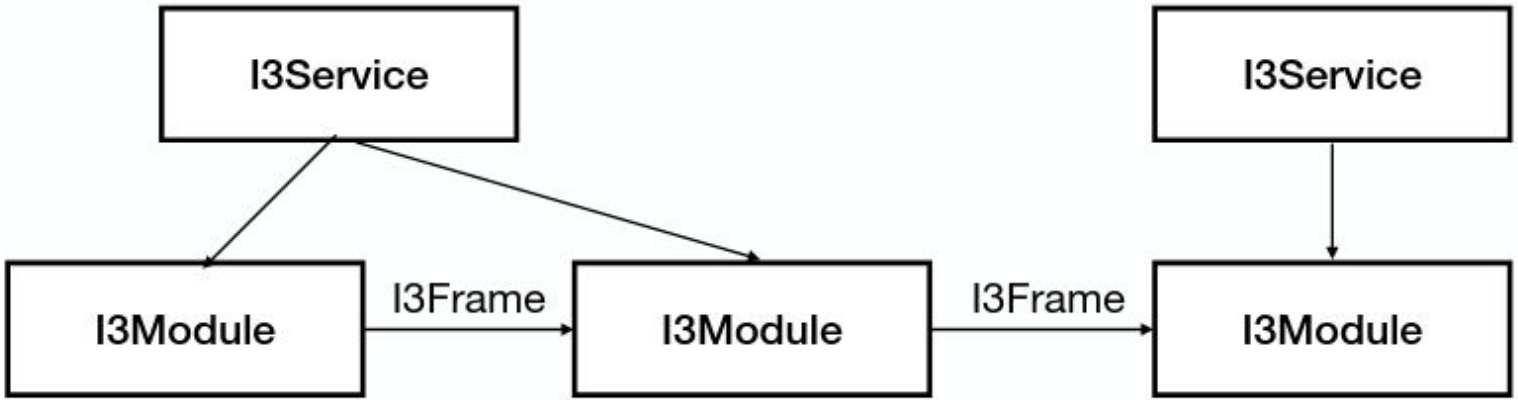
It’s very important to have the correct GCD file for a run/simulation file!

I3Module

- I3Modules take data from I3Frames and process them and add more data to the frame
- Modules are arranged in a “Tray” which passes frames from one module to the next
- Each frame is processed serially — Every module will process a particular frame before the tray moves on to the next event (frame)

I3 Service

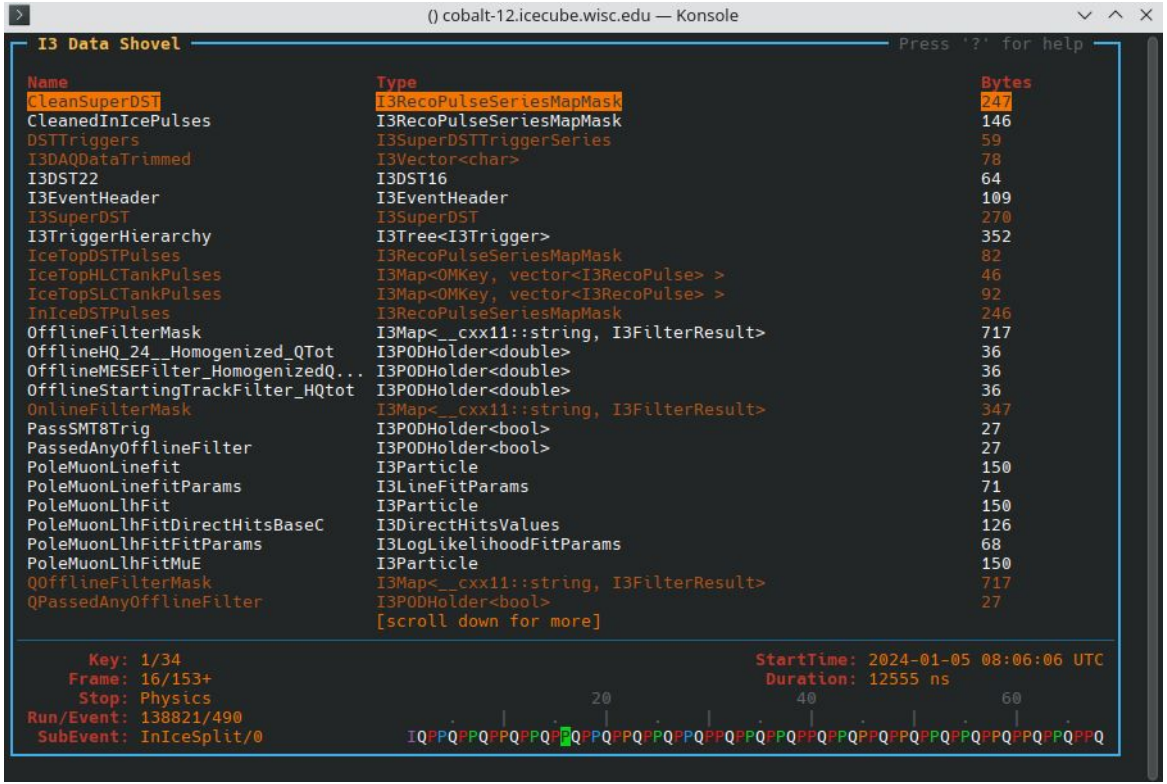
- Services provide code to multiple modules
- Modules can access services such as a random number generator



Using IceTray

Visualization: Dataio-shovel

- Tool for viewing I3File in terminal
- Can be run on remote machines (such as cobalts)



To use:

1. Enter the IceTray environment */build/env-shell.sh
2. Open the file with dataio-shovel dataio-shovel */example.i3.zst

Visualization: - Dataio-shovel

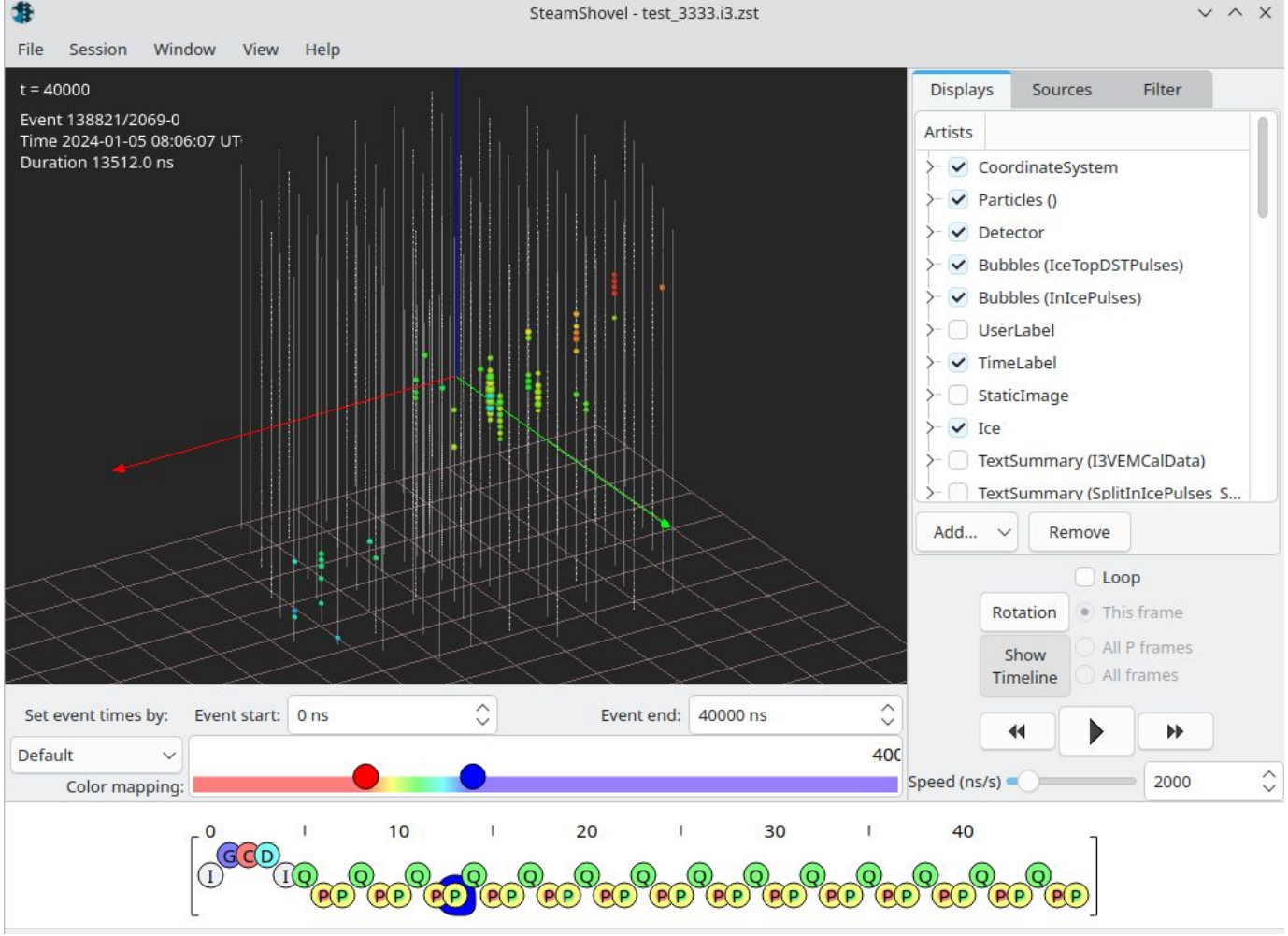
Useful commands:

- “enter”: open an object in a cleaned human readable format
- “x”: open a module in XML format
- “q”: exit an object or the file
- “e”: go to an event number
- “g”: go to a frame number
- “}”: go to last frame
- “{“: go to first frame
- “i”: enter a python shell
- “L”: import a library
- “?”: show all available commands

Dataio-shovel Demo

Visualization: Steamshovel

Tool for 3D visualization of IceCube events in a GUI



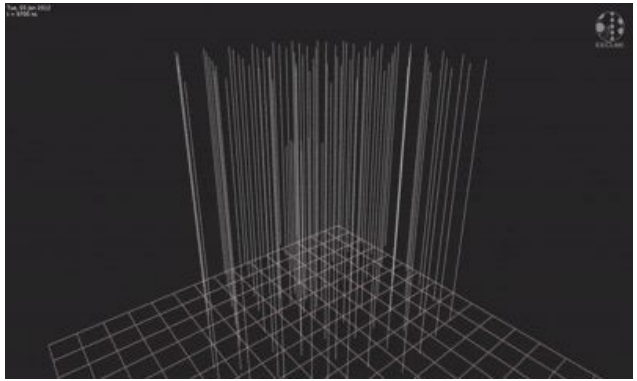
Visualization: - Steamshovel

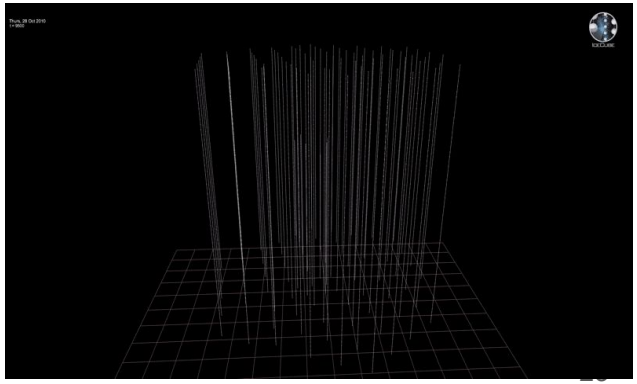
To use:

1. Navigate to the build folder of your IceTray
2. Enter the IceTray environment
   ./env-shell.sh
3. Open steamshovel steamshovel
4. Open 1) the GCD file and then 2) an event I3 file in steamshovel

Documentation:

https://docs.icecube.aq/icetray/main/projects/steamshovel/index.html





Tutorial

Connect to the cobalts

ssh *user*@pub.icecube.wisc.edu

ssh *user*@cobalt.icecube.wisc.edu

Information:

https://wiki.icecube.wisc.edu/index.php/Computing_Services

https://wiki.icecube.wisc.edu/index.php/SSH#config

Or setup .ssh/config with:

```
Host pub*.icecube.wisc.edu
   User *****
   Compression yes
   GSSAPIAuthentication yes
   GSSAPIDelegateCredentials yes

Host cobalt.icecube.wisc.edu
   User *****
   ControlMaster auto
   ForwardAgent yes
   KeepAlive yes
   GSSAPIAuthentication yes
   GSSAPIDelegateCredentials yes
   LocalForward 5555 localhost:5555
   ProxyJump pub.icecube.wisc.edu
```

Load IceTray and try dataio-shovel

1. In your terminal connect to the cobalts
2. eval `/cvmfs/icecube.opensciencegrid.org/py3-v4.4.0/setup.sh`
3. /data/user/herpenbeck/bootcamp/icetray/build/env-shell.sh

Then i.e. either run a python script (that imports icetray) or use dataio-shovel:

4. dataio-shovel /data/user/herpenbeck/bootcamp/i3_files/Level2_IC86.2023_data_Run00138821_84_781_GCD_withSLCcal2.i3.zst /data/user/herpenbeck/bootcamp/i3_files/example_Run00138821_Subrun00000000_00000000.i3.zst

An example of a Simple Tray

Tray:

```python
# Import icetray and dataio
from icecube import icetray, dataio

# Create a new Tray
tray = icetray.I3Tray()

# Add a module that produces an
# infinite number of empty frames
tray.Add("I3InfiniteSource")

# Add a module that prints the
# contents of each frame
tray.Add("Dump")

# Start the execution of the tray
# But only do 10 frames
tray.Execute(10)
```

Output:

```
---------------------------- This is frame number 1 ----------------------------
[ I3Frame  (DAQ):
]
---------------------------- This is frame number 2 ----------------------------
[ I3Frame  (DAQ):
]
---------------------------- This is frame number 3 ----------------------------
[ I3Frame  (DAQ):
]
---------------------------- This is frame number 4 ----------------------------
[ I3Frame  (DAQ):
]
---------------------------- This is frame number 5 ----------------------------
[ I3Frame  (DAQ):
]
---------------------------- This is frame number 6 ----------------------------
[ I3Frame  (DAQ):
]
---------------------------- This is frame number 7 ----------------------------
[ I3Frame  (DAQ):
]
---------------------------- This is frame number 8 ----------------------------
[ I3Frame  (DAQ):
]
---------------------------- This is frame number 9 ----------------------------
[ I3Frame  (DAQ):
]
---------------------------- This is frame number 10 ---------------------------
[ I3Frame  (DAQ):
]
NOTICE (I3Tray): I3Tray finishing... (I3Tray.cxx:525 in void I3Tray::Execute(bool, unsigned int))
```

Add an I3MCTree to the frame

Modules written Python

Modules written In C++

```python
# Import icetray and dataio
from icecube import icetray, dataio, dataclasses

def generator(frame):
    # Add tree containing Monte Carlo particles
    # to the frame
    frame["tree"] = dataclasses.I3MCTree()

# Create a new Tray
tray = icetray.I3Tray()

# Add a module that produces an
# infinite number of empty frames
tray.Add("I3InfiniteSource")

# add generator to the
tray.Add(generator,streams=[icetray.I3Frame.DAQ])

# Add a module that prints the
# contents of each frame
tray.Add("Dump")

# Start the execution of the tray
# But only do 10 frames
tray.Execute(10)
```

```
---------------------------- This is frame number 1 ----------------------------
[ I3Frame  (DAQ):
  'tree' [DAQ] ==> TreeBase::Tree<I3Particle, I3ParticleID, i3hash<I3ParticleID>> (unk)
]
---------------------------- This is frame number 2 ----------------------------
[ I3Frame  (DAQ):
  'tree' [DAQ] ==> TreeBase::Tree<I3Particle, I3ParticleID, i3hash<I3ParticleID>> (unk)
]
---------------------------- This is frame number 3 ----------------------------
[ I3Frame  (DAQ):
  'tree' [DAQ] ==> TreeBase::Tree<I3Particle, I3ParticleID, i3hash<I3ParticleID>> (unk)
]
---------------------------- This is frame number 4 ----------------------------
[ I3Frame  (DAQ):
  'tree' [DAQ] ==> TreeBase::Tree<I3Particle, I3ParticleID, i3hash<I3ParticleID>> (unk)
]
---------------------------- This is frame number 5 ----------------------------
[ I3Frame  (DAQ):
  'tree' [DAQ] ==> TreeBase::Tree<I3Particle, I3ParticleID, i3hash<I3ParticleID>> (unk)
]
---------------------------- This is frame number 6 ----------------------------
[ I3Frame  (DAQ):
  'tree' [DAQ] ==> TreeBase::Tree<I3Particle, I3ParticleID, i3hash<I3ParticleID>> (unk)
]
---------------------------- This is frame number 7 ----------------------------
[ I3Frame  (DAQ):
  'tree' [DAQ] ==> TreeBase::Tree<I3Particle, I3ParticleID, i3hash<I3ParticleID>> (unk)
]
---------------------------- This is frame number 8 ----------------------------
[ I3Frame  (DAQ):
  'tree' [DAQ] ==> TreeBase::Tree<I3Particle, I3ParticleID, i3hash<I3ParticleID>> (unk)
]
---------------------------- This is frame number 9 ----------------------------
[ I3Frame  (DAQ):
  'tree' [DAQ] ==> TreeBase::Tree<I3Particle, I3ParticleID, i3hash<I3ParticleID>> (unk)
]
---------------------------- This is frame number 10 ----------------------------
[ I3Frame  (DAQ):
  'tree' [DAQ] ==> TreeBase::Tree<I3Particle, I3ParticleID, i3hash<I3ParticleID>> (unk)
]
NOTICE (I3Tray): I3Tray finishing... (I3Tray.cxx:525 in void I3Tray::Execute(bool, unsigned int))
```

Use a random service

```python
# Import everything
from icecube import icetray, dataio, phys_services

# Module that gets a random number and prints it
class PrintRandom(icetray.I3Module):
    def __init__(self,context):
        icetray.I3Module.__init__(self,context)
    def DAQ(self,frame):
        #get a random number from the random number service
        rnd = self.context["I3RandomService"].uniform(1)
        #print that number
        print(rnd)

# Create a new Tray
tray = icetray.I3Tray()

# add a random number service to the context with seed = 42
tray.context["I3RandomService"] = phys_services.I3GSLRandomService(42)

# Add a module that produces an
# infinite number of empty frames
tray.Add("I3InfiniteSource")

# add the module we defined above to the tray
tray.Add(PrintRandom)

# Start the execution of the tray
# But only do 10 frames
tray.Execute(10)
```

```
0.37454011430963874
0.7965429842006415
0.9507143115624785
0.18343478767201304
0.7319939383305609
0.7796909974422306
0.5986584862694144
0.5968501614406705
0.1560186385177076
0.4458327575121075
NOTICE (I3Tray): I3Tray finishing... (I3Tray.cxx:525 in void I3Tray::Execute(bool, unsigned int))
```

Add a random number to the frame

```python
# Import everything
from icecube import icetray, dataio, dataclasses, phys_services

# Module that gets a random number and prints it
class AddRandomToFrame(icetray.I3Module):
    def __init__(self,context):
        icetray.I3Module.__init__(self,context)
    def DAQ(self,frame):
        #get a random number from the random number service
        rnd = self.context["I3RandomService"].uniform(1)
        #add that number to the frame as an I3Double
        frame["random_number"] = dataclasses.I3Double(rnd)
        # You need to pass the frame on to the next module
        self.PushFrame(frame)

# Create a new Tray
tray = icetray.I3Tray()

# add a random number service to the context with seed = 42
tray.context["I3RandomService"] = phys_services.I3GSLRandomService(42)

# Add a module that produces an
# infinite number of empty frames
tray.Add("I3InfiniteSource")

# add the module we defined above to the
tray.Add(AddRandomToFrame)

# add module to print each frame
tray.Add("Dump")

# Start the execution of the tray
# But only do 10 frames
tray.Execute(10)
```

```
---------------------- This is frame number 1 ----------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
---------------------- This is frame number 2 ----------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
---------------------- This is frame number 3 ----------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
---------------------- This is frame number 4 ----------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
---------------------- This is frame number 5 ----------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
---------------------- This is frame number 6 ----------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
---------------------- This is frame number 7 ----------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
---------------------- This is frame number 8 ----------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
---------------------- This is frame number 9 ----------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
---------------------- This is frame number 10 ----------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
```

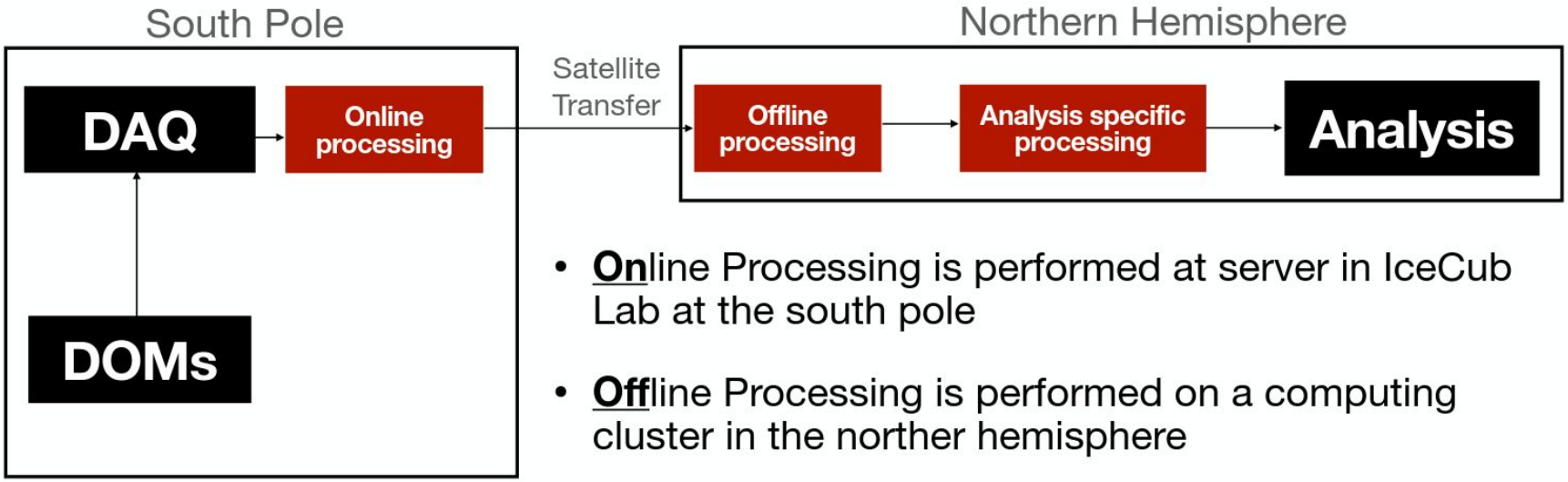
```python
# Import everything
from icecube import icetray, dataio, dataclasses, phys_services

# Module that gets a random number and prints it
class AddRandomToFrame(icetray.I3Module):
    def __init__(self,context):
        icetray.I3Module.__init__(self,context)
    def DAQ(self,frame):
        #get a random number from the random number service
        rnd = self.context["I3RandomService"].uniform(1)
        #add that number to the frame as an I3Double
        frame["random_number"] = dataclasses.I3Double(rnd)
        # You need to pass the frame on to the next module
        self.PushFrame(frame)

# define filter that removes half of the events
def filter(frame):
    return frame['random_number']<0.5

# Create a new Tray
tray = icetray.I3Tray()

# add a random number service to the context with seed = 42
tray.context["I3RandomService"] = phys_services.I3GSLRandomService(42)

# Add a module that produces an
# infinite number of empty frames
tray.Add("I3InfiniteSource")

# add the module we defined above to the frame
tray.Add(AddRandomToFrame)

#add filter to the tray
tray.Add(filter,streams = [icetray.I3Frame.DAQ])

# add module to print each frame
tray.Add("Dump")

# Start the execution of the tray
# But only do 10 frames
tray.Execute(10)
```

Use a filter to remove Events based on the contents of the frame

```
------------------------- This is frame number 1 -------------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
------------------------- This is frame number 2 -------------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
------------------------- This is frame number 3 -------------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
------------------------- This is frame number 4 -------------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
NOTICE (I3Tray): I3Tray finishing... (I3Tray.cxx:525 in void I3Tray::Execute(bool, unsigned int))
```

Offline Filter example

More detailed view of processing



- **On**line Processing is performed at server in IceCub Lab at the south pole
- **Off**line Processing is performed on a computing cluster in the norther hemisphere
- Most analyses require additional processing beyond what is provided by offline processing, usually handled by working groups

Read in a file and then use existing information

```python
#The traysegment for the charge filter
@icetray.traysegment
def ChargeFilter(tray, name):

    #It is common to apply a filter to only specific frames
    #In this case, we want to apply the filter only to InIceSplit frames
    #also we want to use the Online Filtering information
    #So we check for two online filters.
    def online_filters_passed(frame):
        """Return True if frame passed any of the specified filters."""
        if not which_split("InIceSplit")(frame):
            return False

        filter_names = ["OnlineL2Filter_23", "SoftwareSMT12Filt_23"]
        return any(frame["OnlineFilterMask"][filter_name].prescale_passed for filter_name in filter_names)


    # apply HomogenizedQTot to the SplitInIcePulses
    # it calculates the charge of an event using the given pulse
    #The output is an I3Double named HomogenizedQTot with the total charge in photoelectrons
    from icecube import VHESelfVeto
    tray.AddModule("HomogenizedQTot", name+"_qtot_causal",
        Pulses="SplitInIcePulses",
        Output="HomogenizedQTot",
        If = online_filters_passed)

    # Cut on the newly calculated charge of the event
    # Save the result of the cut in a boolean I3Bool named Charge_Cut_Bool
    def cut(frame):
        cut_passed = frame["HomogenizedQTot"].value>100.
        if cut_passed:
            print(f"Cut passed for event {frame['I3EventHeader'].event_id} with charge {frame['HomogenizedQTot'].value:.2f} PE")
        frame["Charge_Cut_Bool"] = icetray.I3Bool(cut_passed)

    tray.AddModule(cut, "cut", Streams=[icetray.I3Frame.Physics], If=online_filters_passed)
```

Reading in a file, writing new keys and saving them in another file

```
#Now open the i3 file and apply the filter
# Create a new Tray
tray = icetray.I3Tray()

# a gcd file and an i3 file with data
files = ["/data/user/herpenbeck/bootcamp/i3_files/Level2_IC86.2023_data_Run00138821_84_781_G

#load i3 file
tray.Add(dataio.I3Reader, "reader",
                filenamelist=files)

# Add filter to the tray
tray.AddSegment(ChargeFilter, "ChargeFilter")

# save the output to a new i3 file
output_file = "ChargeFilter_output.i3.zst"
# Write the physics and DAQ frames
# Also write TrayInfo frame which lists modules and services, along with their configuration
tray.AddModule("I3Writer", "EventWriter",
               filename=output_file,
               Streams=[icetray.I3Frame.TrayInfo,
                        icetray.I3Frame.DAQ,
                        icetray.I3Frame.Physics,
                        ])

tray.Execute(100)

# Stop the tray
tray.Finish()

# Print the output file name
print(f"Output written to {output_file}")
```

Build your own icetray on the cobalts

https://docs.icecube.aq/icetray/main/info/quickstart.html = Instructions

Our changes: git clone git@github.com:icecube/icetray.git src

Or use a cvmfs version, example:

1. eval `/cvmfs/icecube.opensciencegrid.org/py3-v4.4.0/setup.sh`
2. /cvmfs/icecube.opensciencegrid.org/py3-v4.4.0/RHEL_7_x86_64_v2/metaprojects/icetray/v1.13.0/bin/icetray-shell