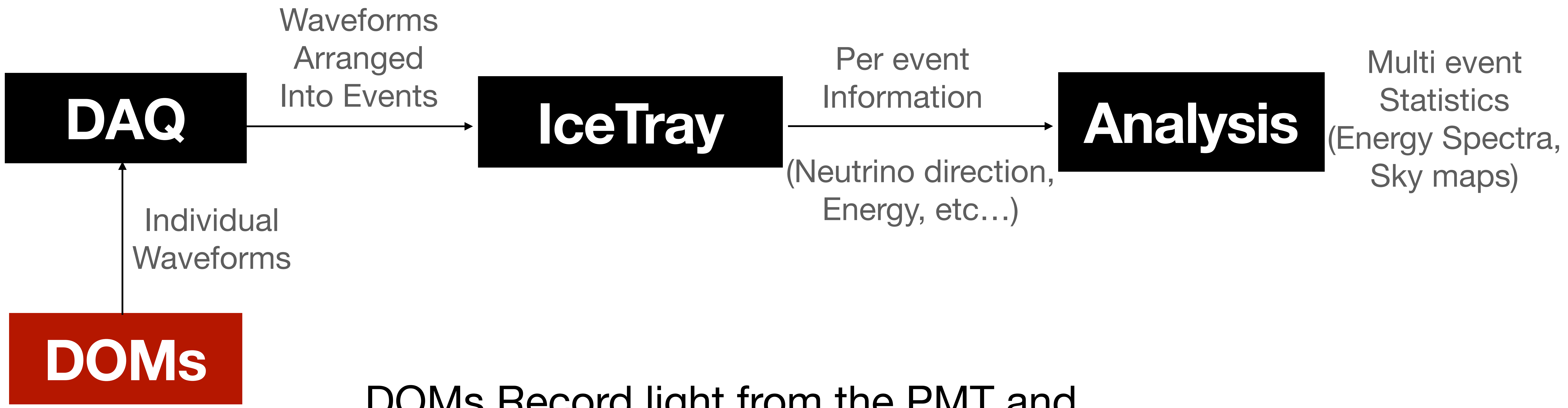
IceTray Tutorial

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

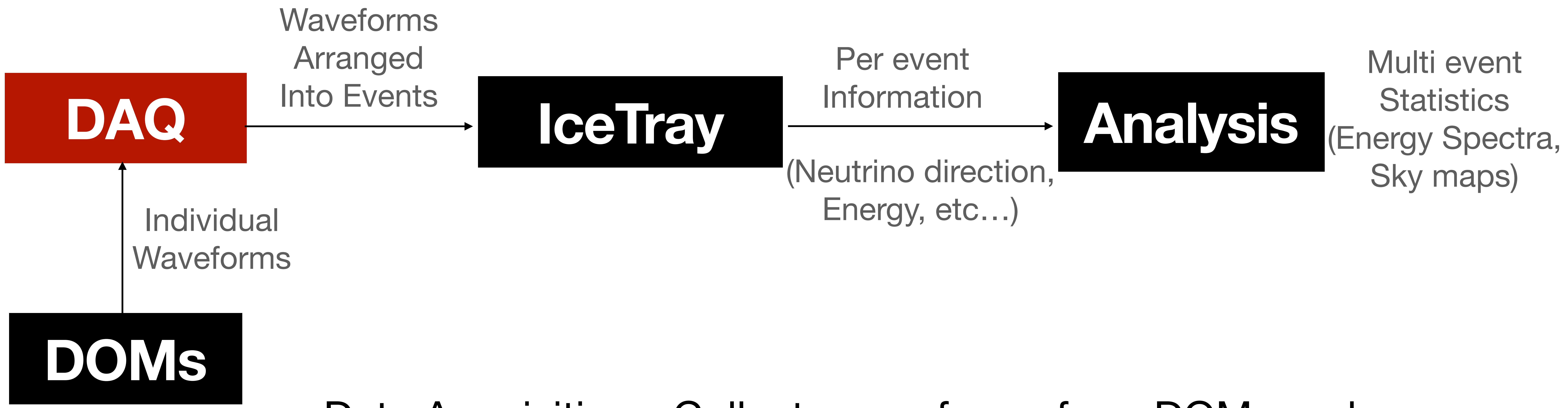
Kevin Meagher
IceCube Summer School
June 2024

Overview of IceCube Processing Pipelines


DAQ
Waveforms Arranged Into Events
IceTray
Per event Information
(Neutrino direction, Energy, etc…)
Analysis
Multi event Statistics (Energy Spectra, Sky maps)
Individual Waveforms
DOMs

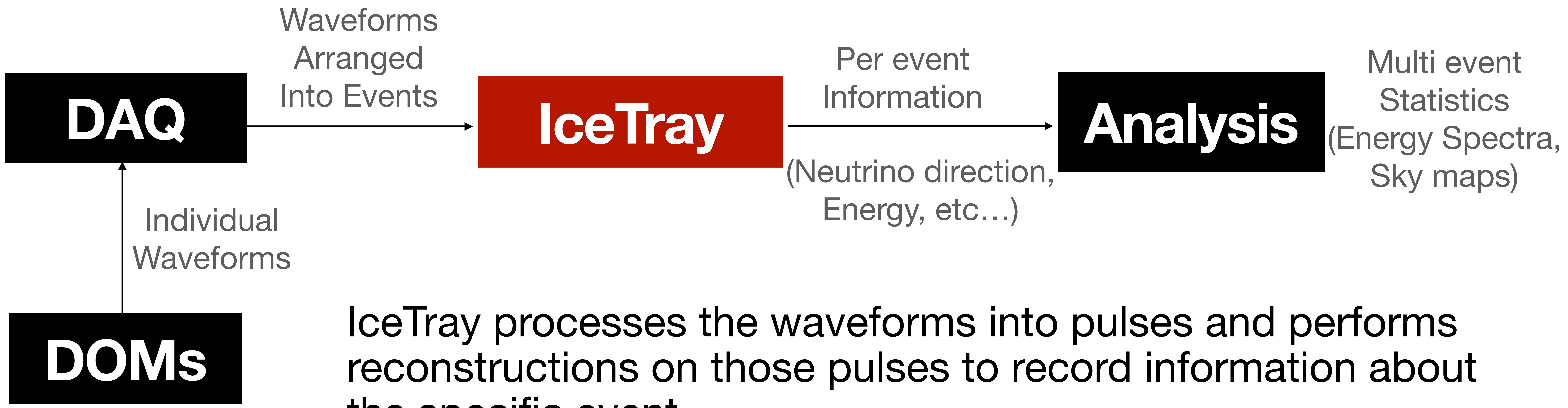
DOMs Record light from the PMT and save a time series as a waveform

Overview of IceCube Processing Pipelines


DAQ
Waveforms Arranged Into Events
IceTray
Per event Information
(Neutrino direction, Energy, etc…)
Analysis
Multi event Statistics (Energy Spectra, Sky maps)
Individual Waveforms
DOMs

Data Acquisition - Collects waveforms from DOMs and arranges them into events that last ~10µs

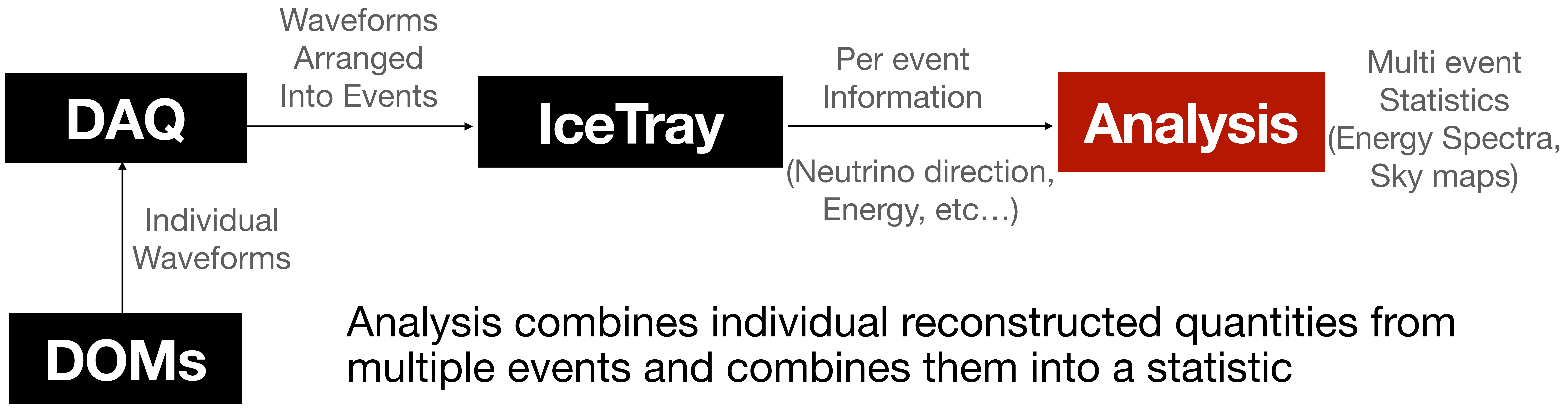
Overview of IceCube Processing Pipelines


DAQ
Waveforms Arranged Into Events
IceTray
Per event Information
(Neutrino direction, Energy, etc…)
Analysis
Multi event Statistics (Energy Spectra, Sky maps)
Individual Waveforms
DOMs

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

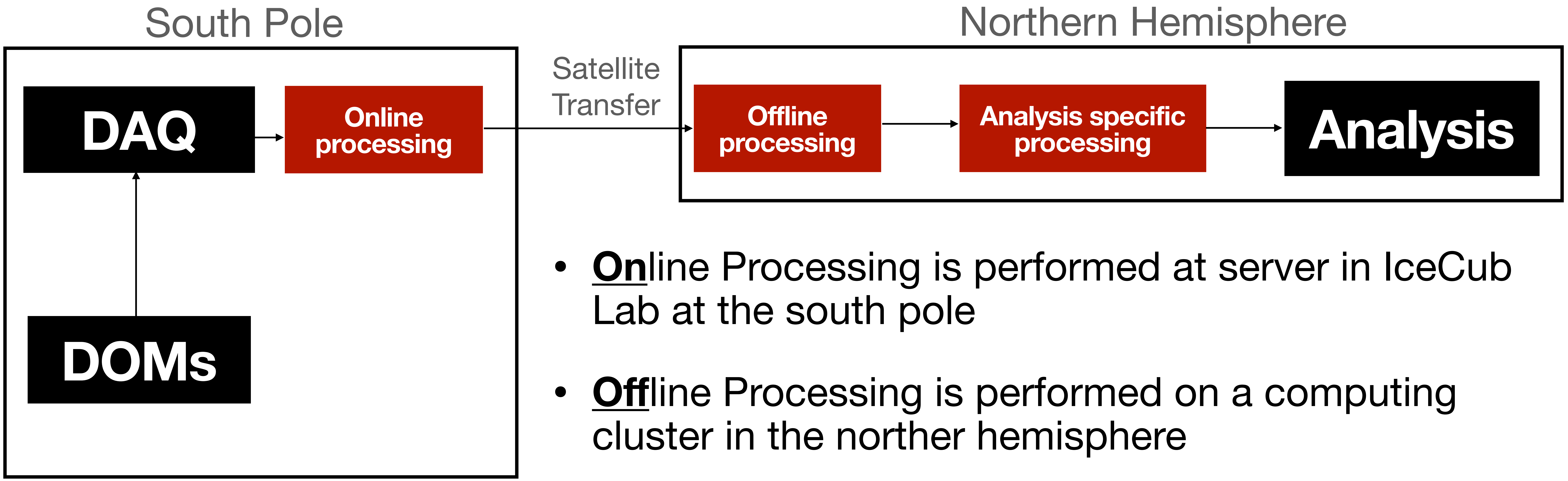
Overview of IceCube Processing Pipelines


DAQ
Waveforms Arranged Into Events
IceTray
Per event Information
(Neutrino direction, Energy, etc…)
Analysis
Multi event Statistics (Energy Spectra, Sky maps)
Individual Waveforms
DOMs

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



- **<u>On</u>**line Processing is performed at server in IceCub Lab at the south pole

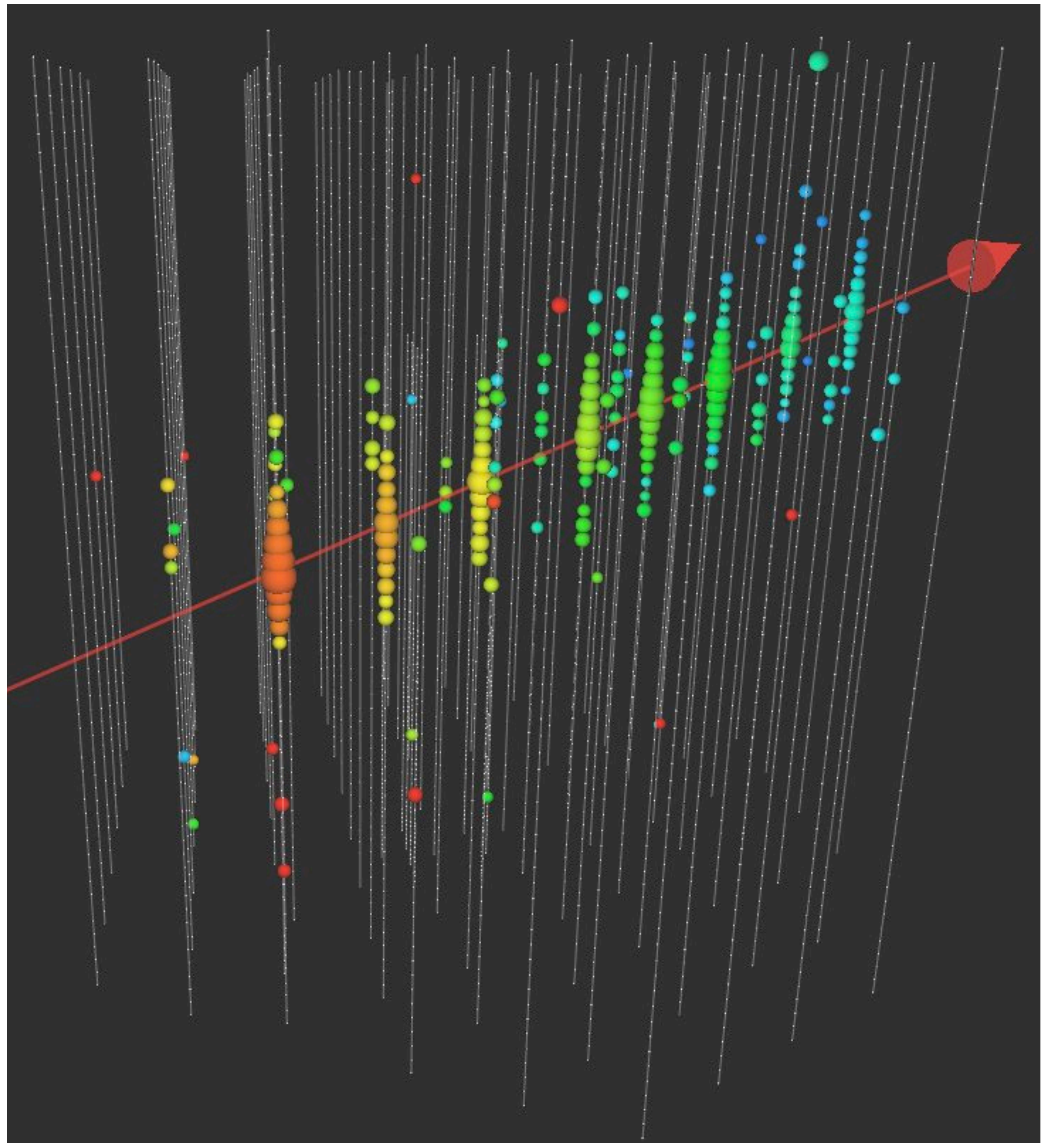
- **<u>Off</u>**line Processing is performed on a computing cluster in the norther hemisphere

- Most analyses require additional processing beyond what is provided by offline processing, usually handled by working groups

Getting Help

- IceTray 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

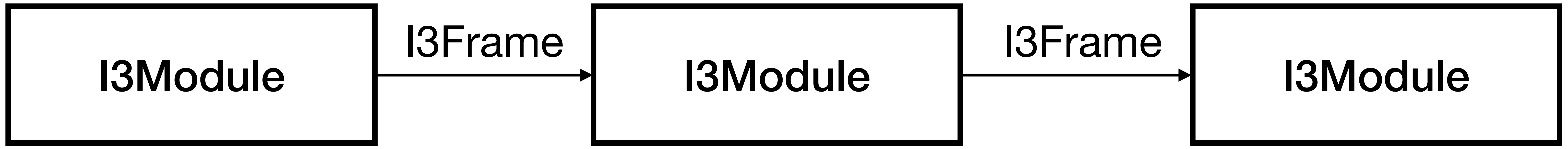
I3Frame is the building block of IceTray



- 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 to save data

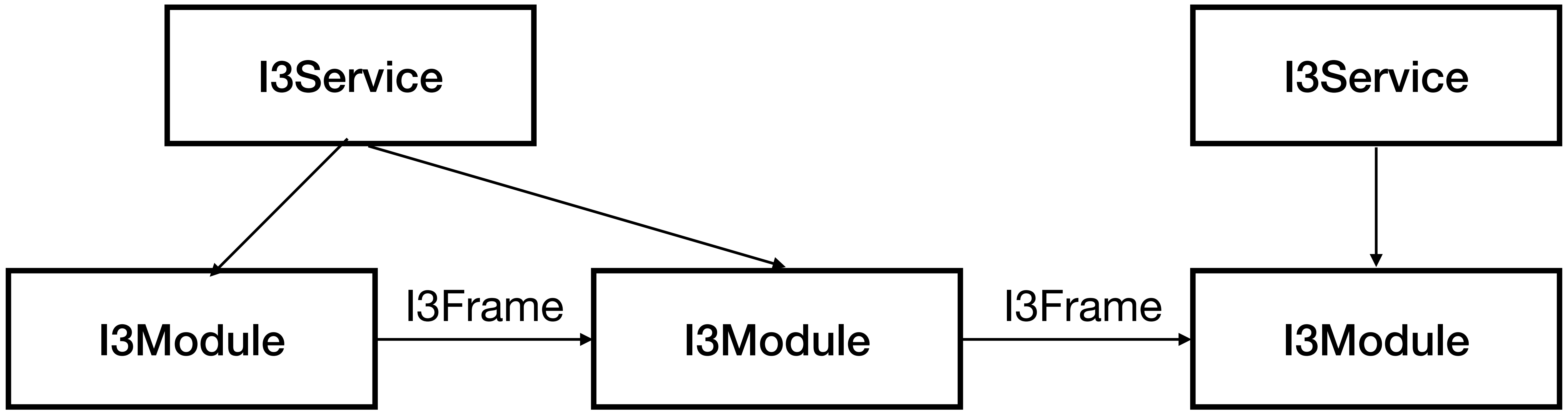
I3Modules process the data in the frame

- 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)



Services provide code to multiple modules

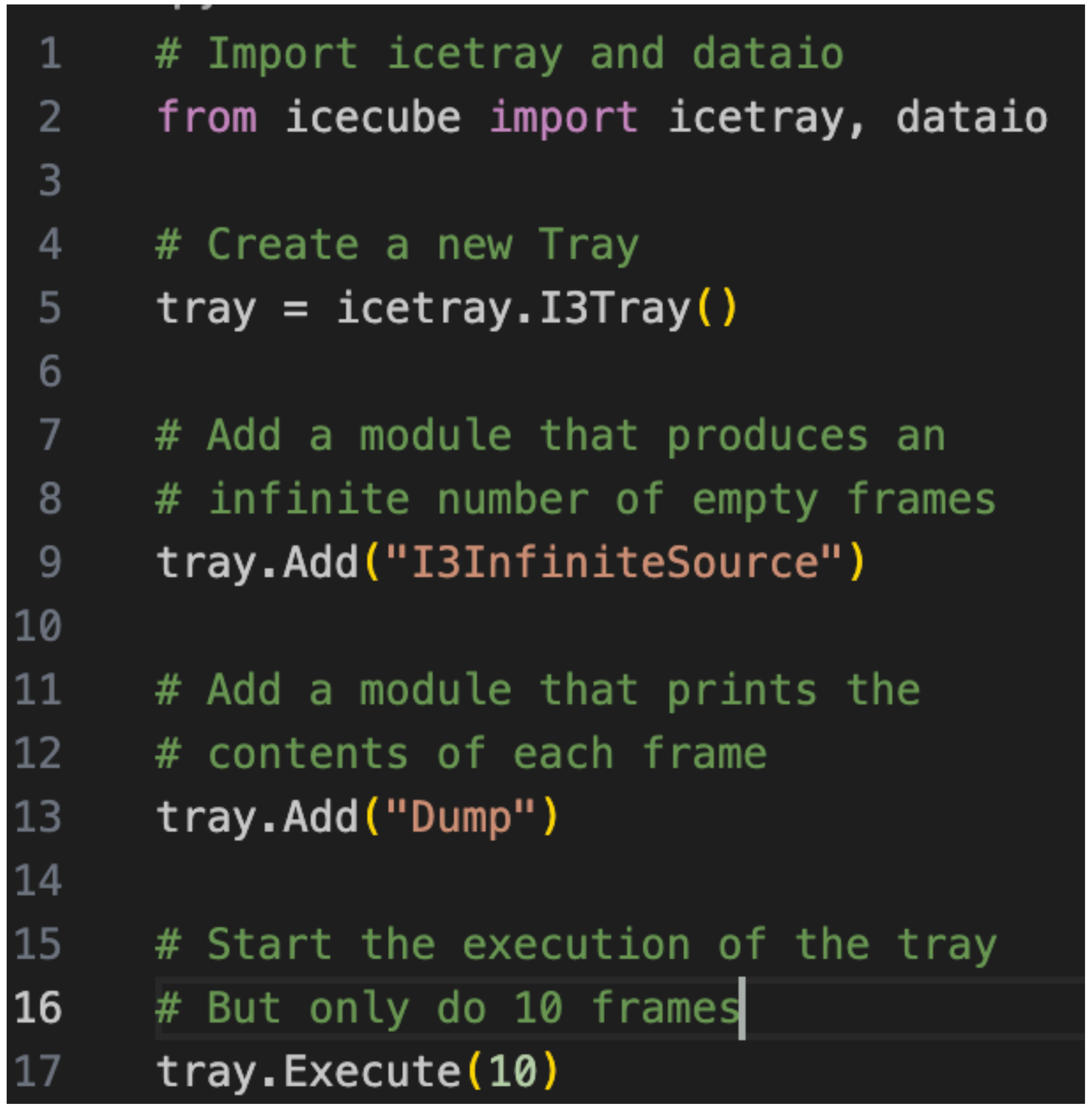
Modules can access services such as a random number generator



Interactive Tutorials

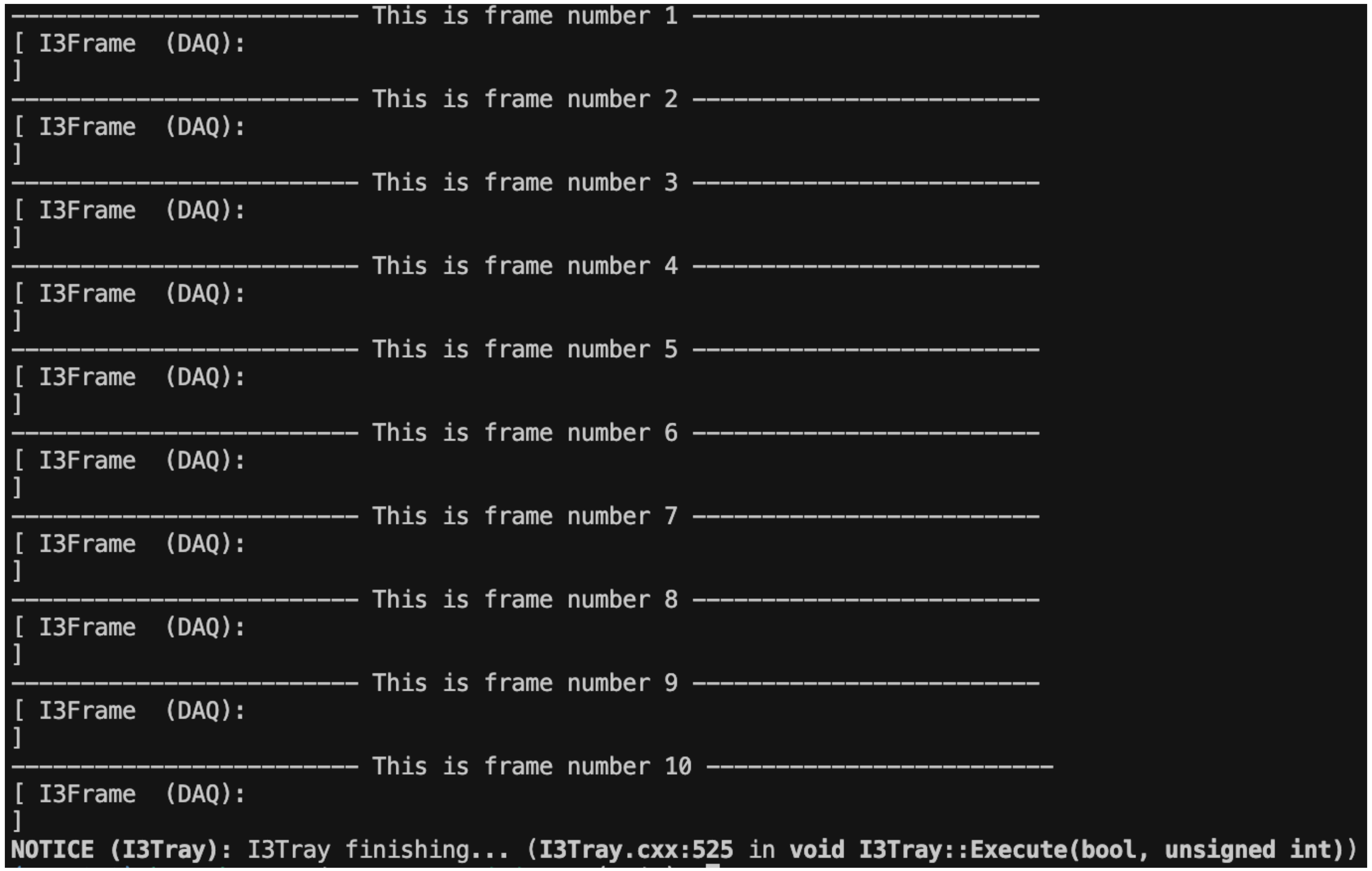
An example of a Simple Tray

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:

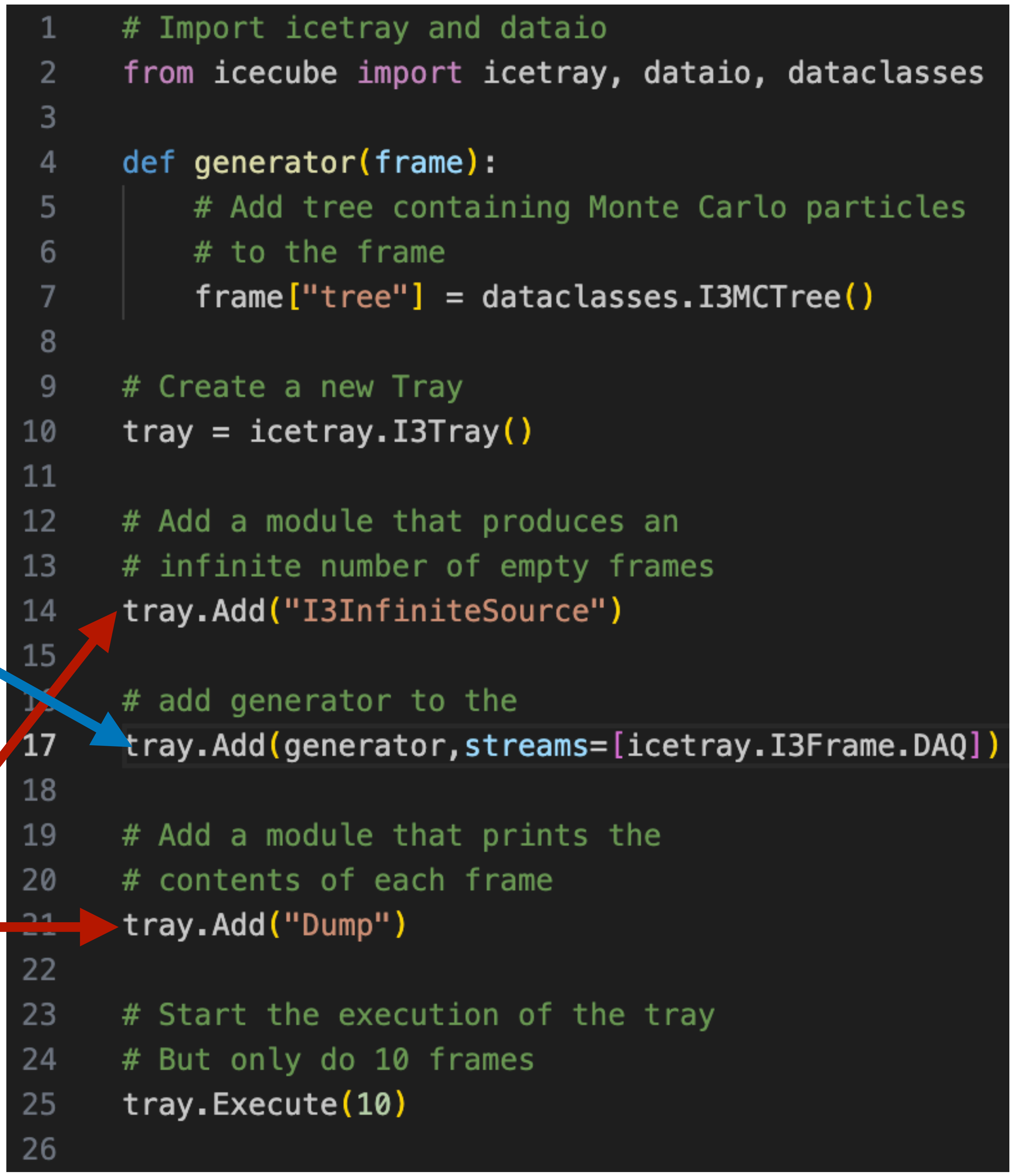


```
-------------------------- This is frame number 1 --------------------------
[ I3Frame  (DAQ):
]
-------------------------- This is frame number 2 --------------------------
[ I3Frame  (DAQ):
]
-------------------------- This is frame number 3 --------------------------
[ I3Frame  (DAQ):
]
-------------------------- 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))
```

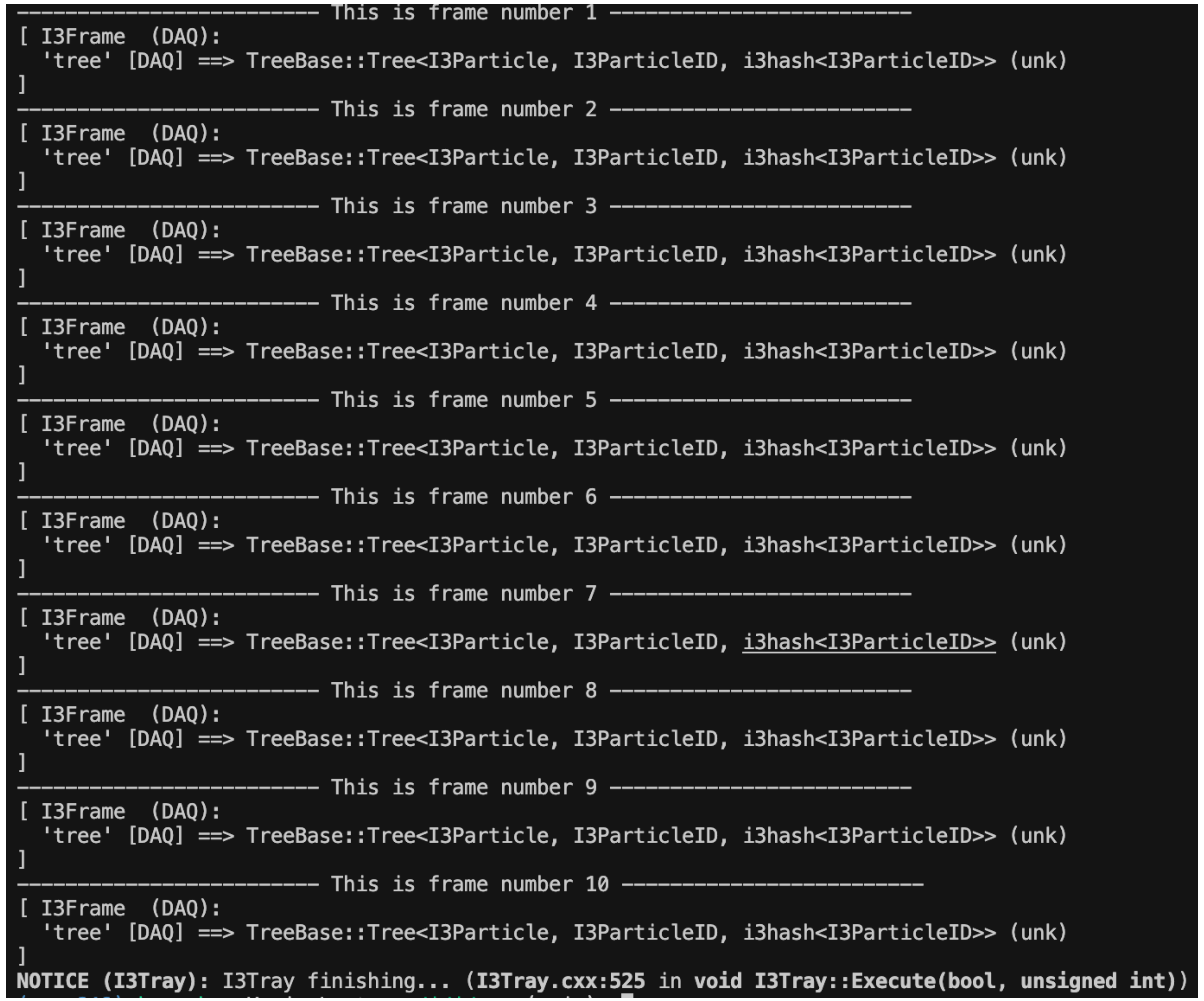
Add an I3MCTree to the frame

Modules written Python

Modules written In C++

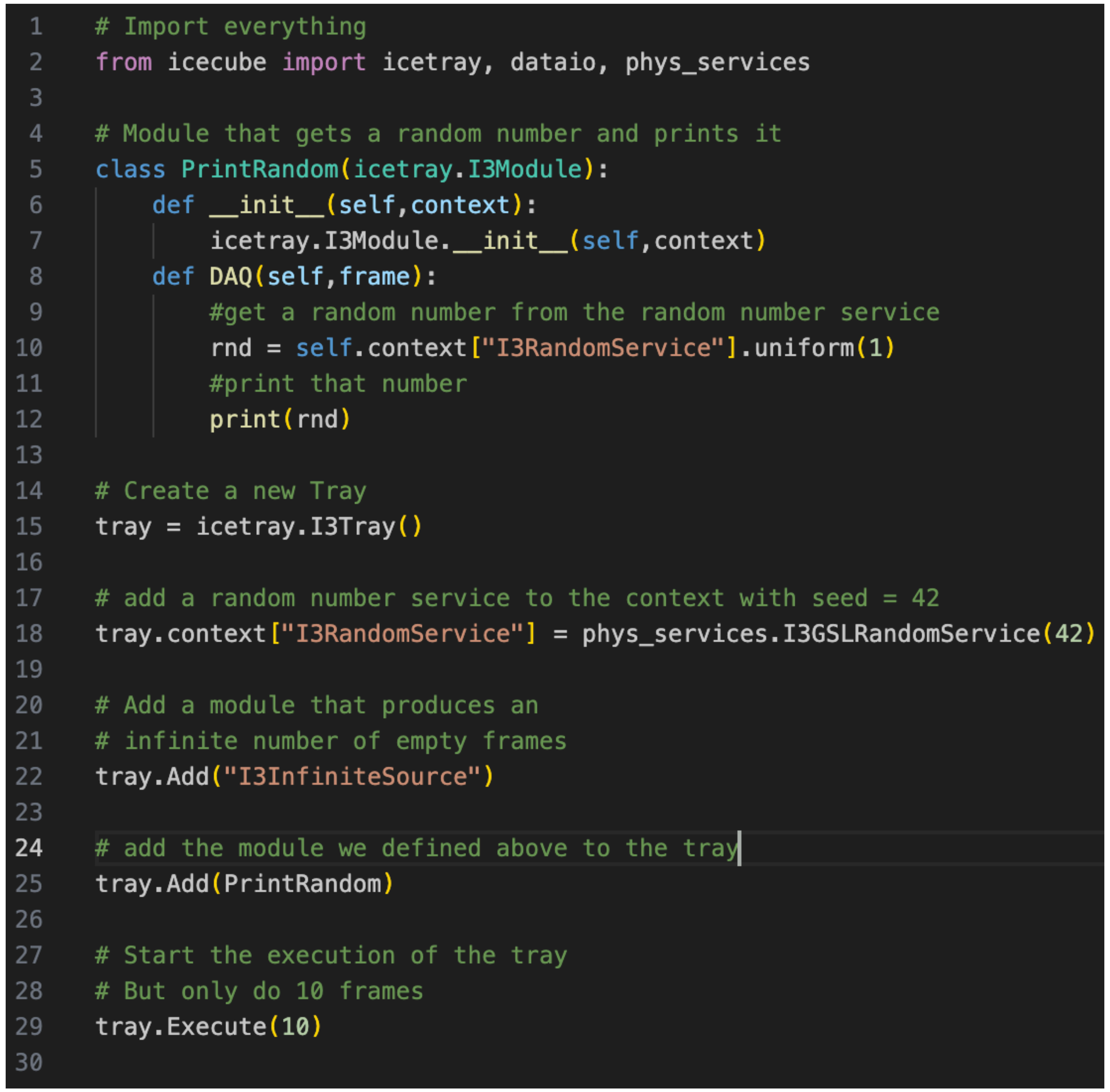


```python
# Import icetray and dataio
from icecube import icetray, dataio, dataclasses

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)

```

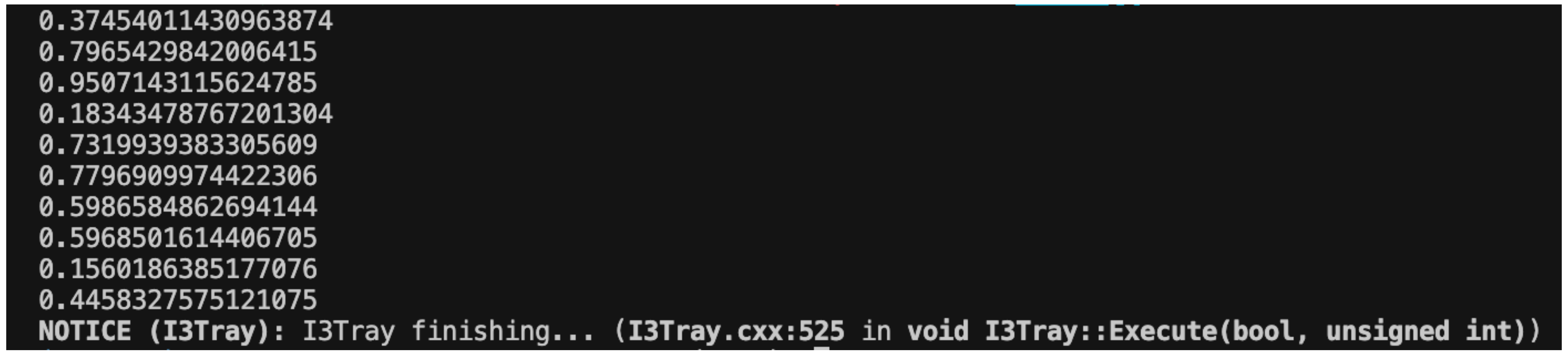


```
------------------------- This is frame number 1 -------------------------
[ I3Frame  (DAQ):
  'tree' [DAQ] ==> TreeBase::Tree<I3Particle, I3ParticleID, i3hash<I3ParticleID>> (unk)
]
------------------------- This is frame number 2 -------------------------
[ I3Frame  (DAQ):
  'tree' [DAQ] ==> TreeBase::Tree<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

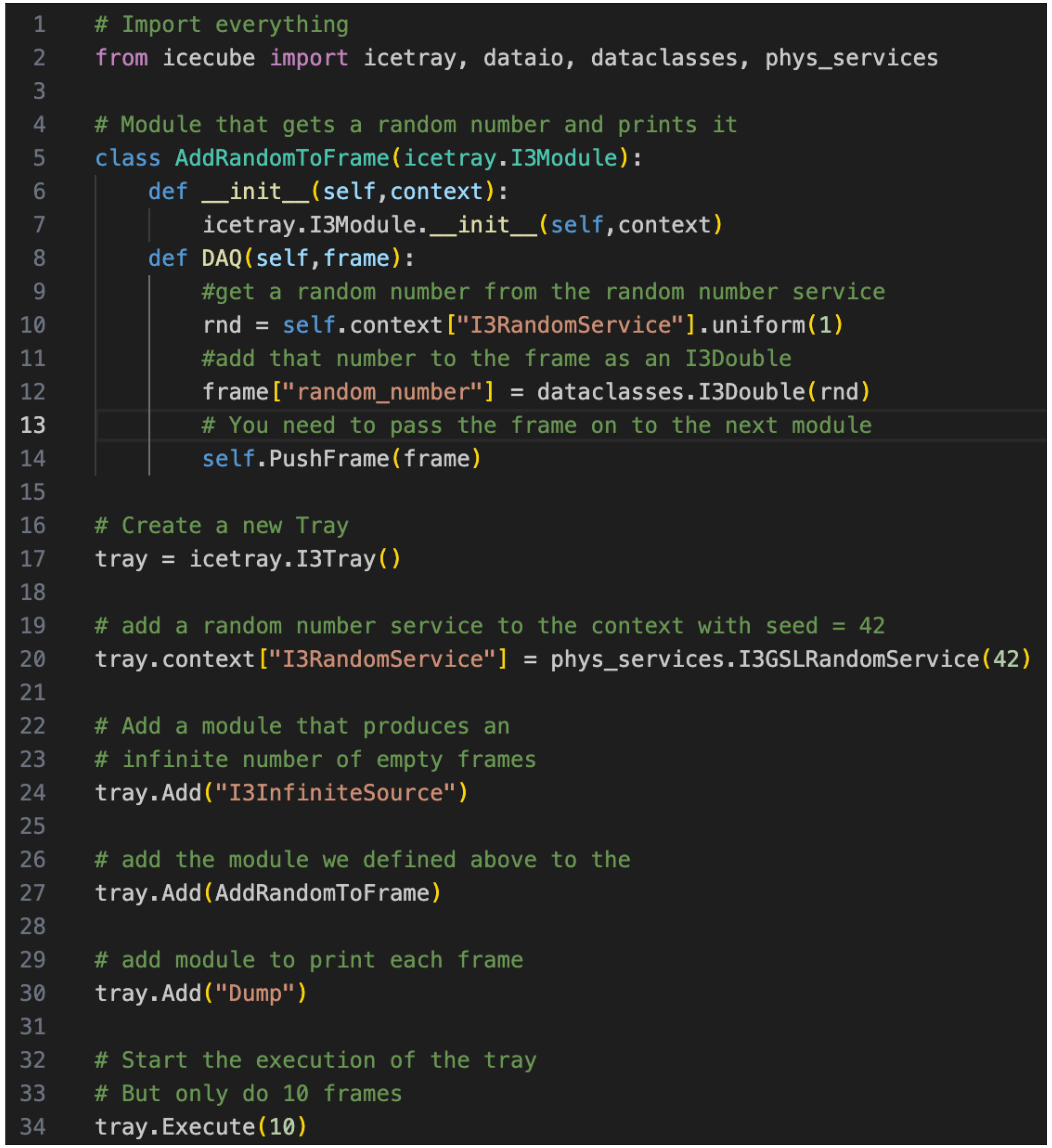


```python
# Import everything
from icecube import icetray, dataio, phys_services

# 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)

```

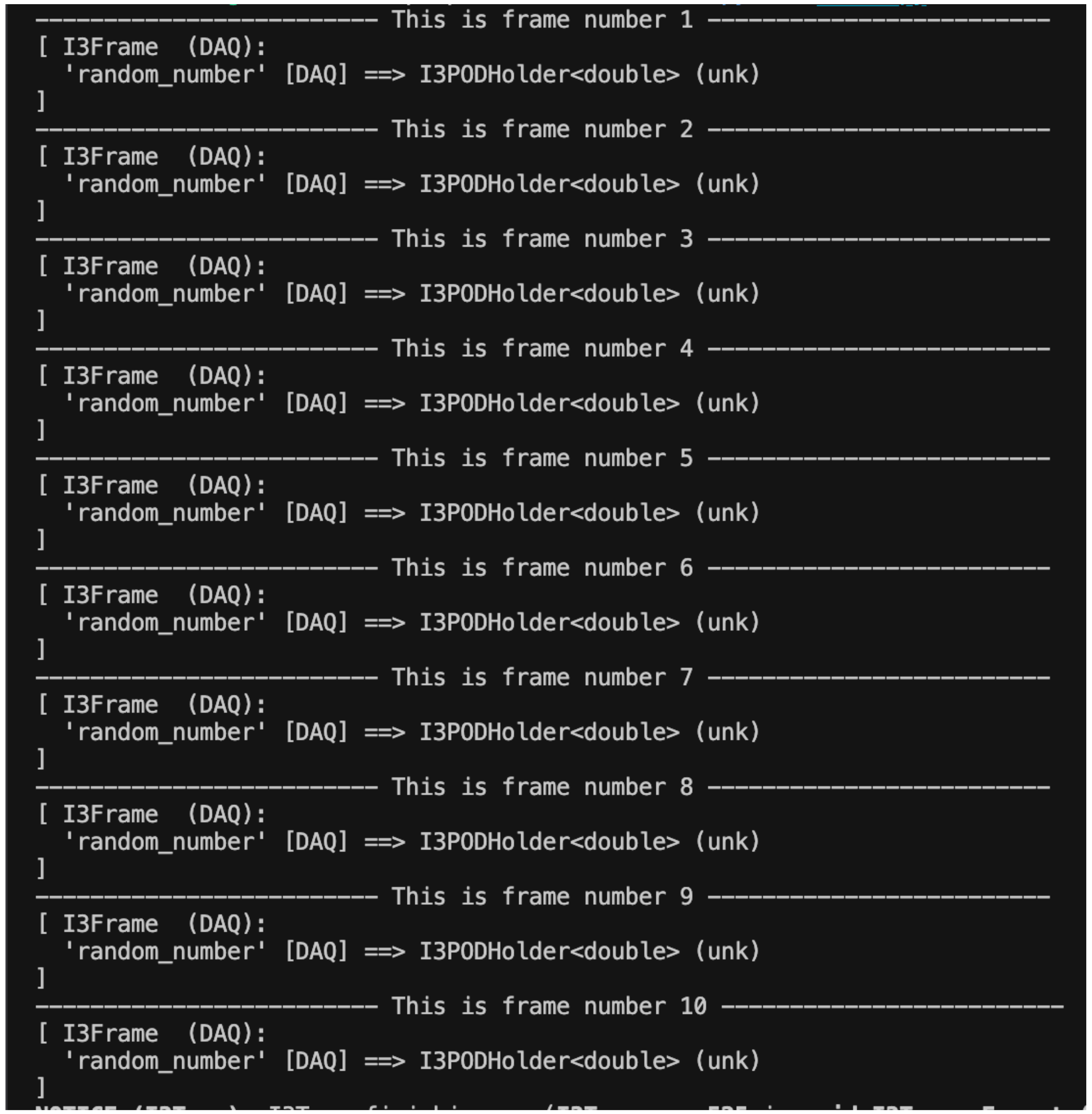


```
0.37454011430963874
0.7965429842006415
0.9507143115624785
0.18343478767201304
0.7319939383305609
0.7796909974422306
0.5986584862694144
0.5968501614406705
0.1560186385177076
0.4458327575121075
NOTICE (I3Tray): I3Tray finishing... (I3Tray.cxx:525 in void I3Tray::Execute(bool, unsigned int))
```

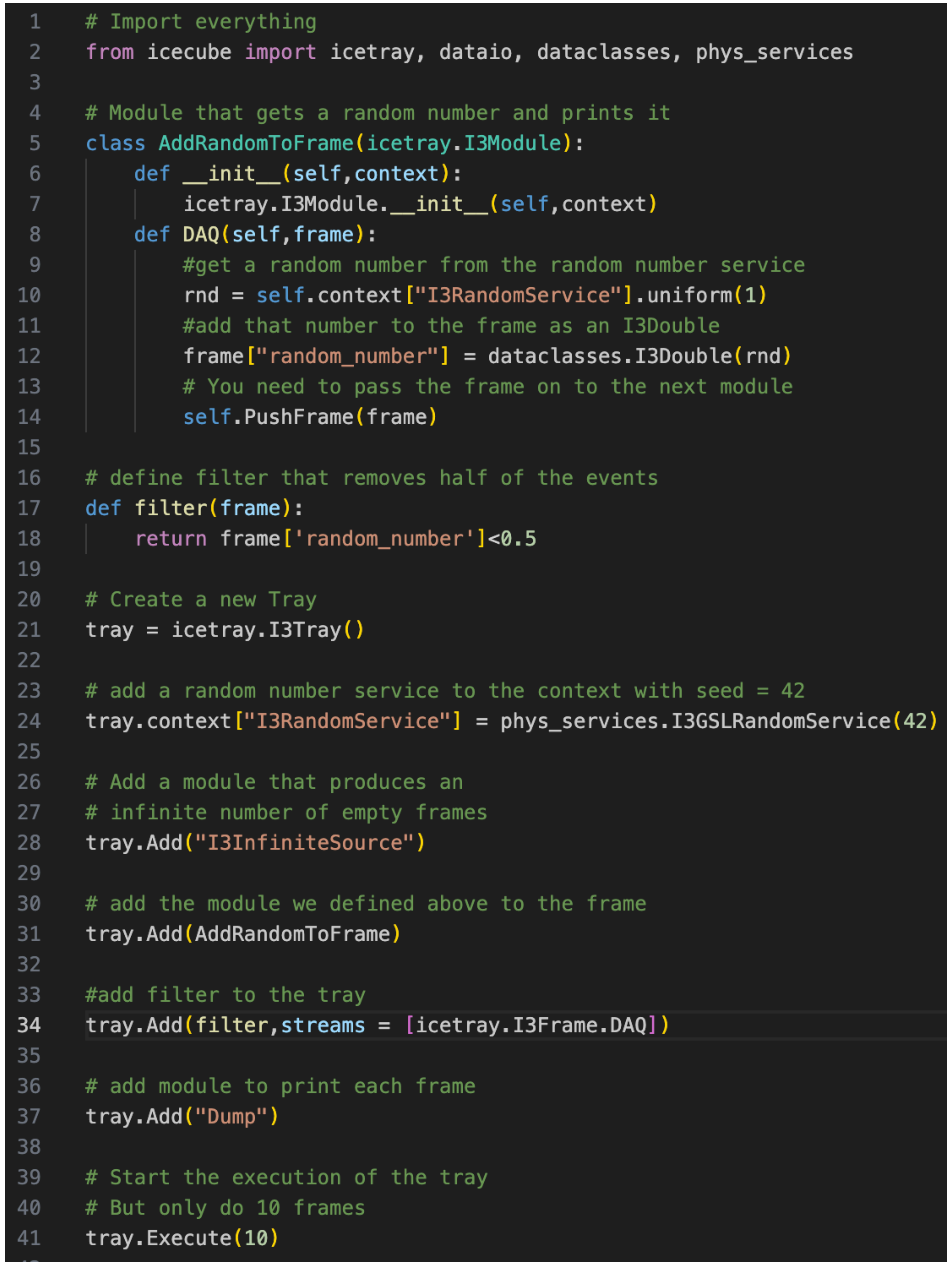
Add a random number to the frame



```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)
```

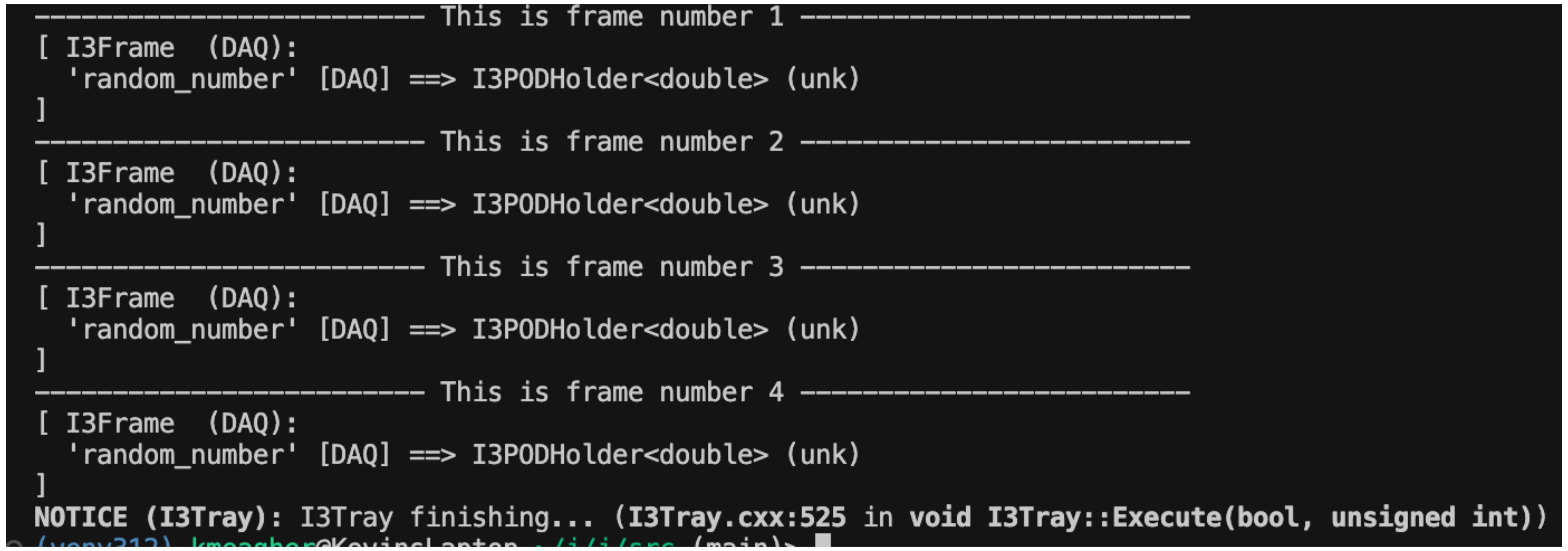


```
------------------------- This is frame number 1 -------------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
------------------------- This is frame number 2 -------------------------
[ I3Frame  (DAQ):
  'random_number' [DAQ] ==> I3PODHolder<double> (unk)
]
------------------------- 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))
```

- **In class assignment:** Add a neutrino to the I3MCTree with a random energy and write a filter to cut on its energy

- **Homework:** Add an secondary muon to the I3MCTree and write a filter to cut on its energy using a lambda

Getting Help

- 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