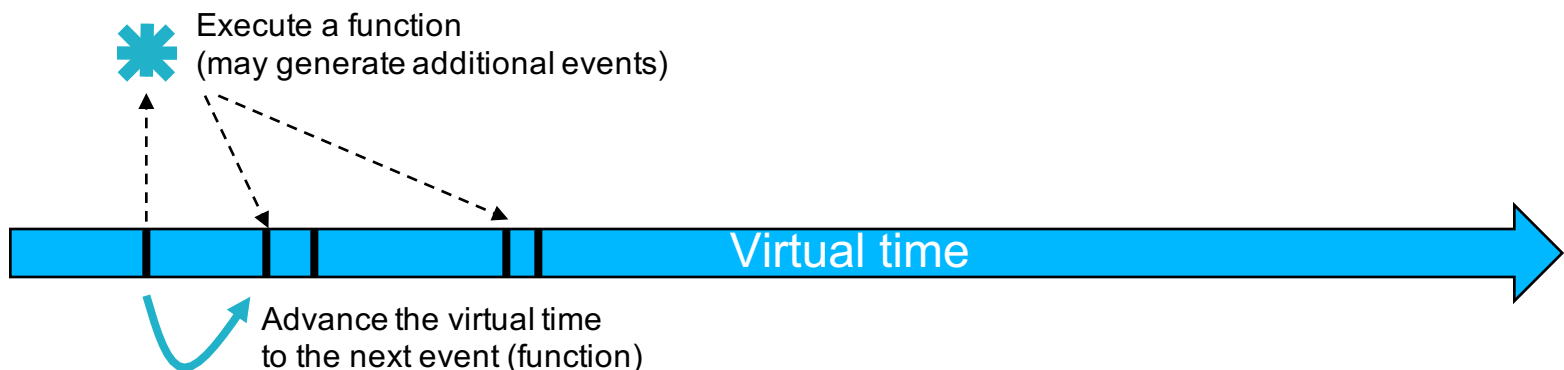

ns-3 Training

ns-3 training, June 2017

Simulator core

- Simulation time
- Events
- Simulator and Scheduler
- Command line arguments
- Random variables



Simulator example

```
#include <iostream>
#include "ns3/simulator.h"
#include "ns3/nstime.h"
#include "ns3/command-line.h"
#include "ns3/double.h"
#include "ns3/random-variable-stream.h"

using namespace ns3;
```

```
int main (int argc, char *argv[])
{
    CommandLine cmd;
    cmd.Parse (argc, argv);

    MyModel model;
    Ptr<UniformRandomVariable> v = CreateObject<UniformRandomVariable> ();
    v->SetAttribute ("Min", DoubleValue (10));
    v->SetAttribute ("Max", DoubleValue (20));

    Simulator::Schedule (Seconds (10.0), &ExampleFunction, &model);

    Simulator::Schedule (Seconds (v->GetValue ()), &RandomFunction);

    EventId id = Simulator::Schedule (Seconds (30.0), &CancelledEvent);
    Simulator::Cancel (id);

    Simulator::Run ();

    Simulator::Destroy ();
}
```

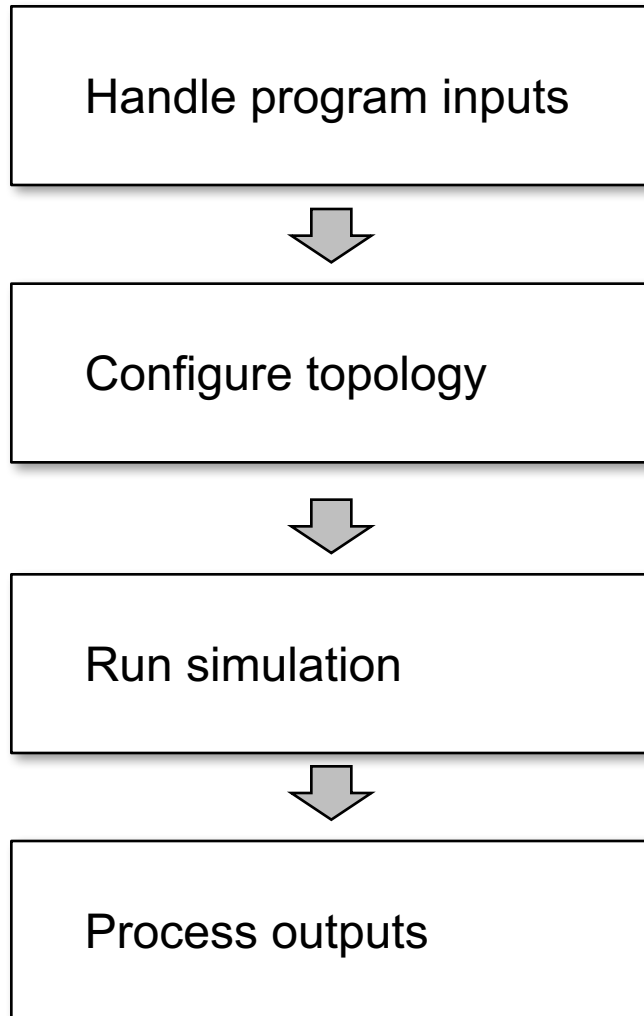
Simulator example (in Python)

```
# Python version of sample-simulator.cc
```

```
import ns.core
```

```
def main(dummy_argv):  
  
    model = MyModel()  
    v = ns.core.UniformRandomVariable()  
    v.SetAttribute("Min", ns.core.DoubleValue(10))  
    v.SetAttribute("Max", ns.core.DoubleValue(20))  
  
    ns.core.Simulator.Schedule(ns.core.Seconds(10.0), ExampleFunction, model)  
  
    ns.core.Simulator.Schedule(ns.core.Seconds(v.GetValue()), RandomFunction, model)  
  
    id = ns.core.Simulator.Schedule(ns.core.Seconds(30.0), CancelledEvent)  
    ns.core.Simulator.Cancel(id)  
  
    ns.core.Simulator.Run()  
  
    ns.core.Simulator.Destroy()  
  
if __name__ == '__main__':  
    import sys  
    main(sys.argv)
```

Simulation program flow



Command-line arguments

- Add CommandLine to your program if you want command-line argument parsing

```
int main (int argc, char *argv[])  
{  
    CommandLine cmd;  
    cmd.Parse (argc, argv);  
}
```

- Passing --PrintHelp to programs will display command line options, if CommandLine is enabled

```
./waf --run "sample-simulator --PrintHelp"
```

```
--PrintHelp: Print this help message.  
--PrintGroups: Print the list of groups.  
--PrintTypeIds: Print all TypeIds.  
--PrintGroup=[group]: Print all TypeIds of group.  
--PrintAttributes=[typeid]: Print all attributes of typeid.  
--PrintGlobals: Print the list of globals.
```

Time in ns-3

- Time is stored as a large integer in ns-3
 - Minimize floating point discrepancies across platforms
- Special Time classes are provided to manipulate time (such as standard operators)
- Default time resolution is nanoseconds, but can be set to other resolutions
 - Note: Changing resolution is not well used/tested
- Time objects can be set by floating-point values and can export floating-point values

```
double timeDouble = t.GetSeconds();
```

- Best practice is to avoid floating point conversions where possible

Events in ns-3

- Events are just function calls that execute at a simulated time
 - i.e. callbacks
 - this is another difference compared to other simulators, which often use special "event handlers" in each model
- Events have IDs to allow them to be cancelled or to test their status

Simulator and Schedulers

- The Simulator class holds a scheduler, and provides the API to schedule events, start, stop, and cleanup memory
- Several scheduler data structures (calendar, heap, list, map) are possible
- "RealTime" simulation implementation aligns the simulation time to wall-clock time
 - two policies (hard and soft limit) available when the simulation and real time diverge

Random Variables

from src/core/examples/sample-rng-plot.py

- Currently implemented distributions
 - Uniform: values uniformly distributed in an interval
 - Constant: value is always the same (not really random)
 - Sequential: return a sequential list of predefined values
 - Exponential: exponential distribution (poisson process)
 - Normal (gaussian), Log-Normal, Pareto, Weibull, triangular

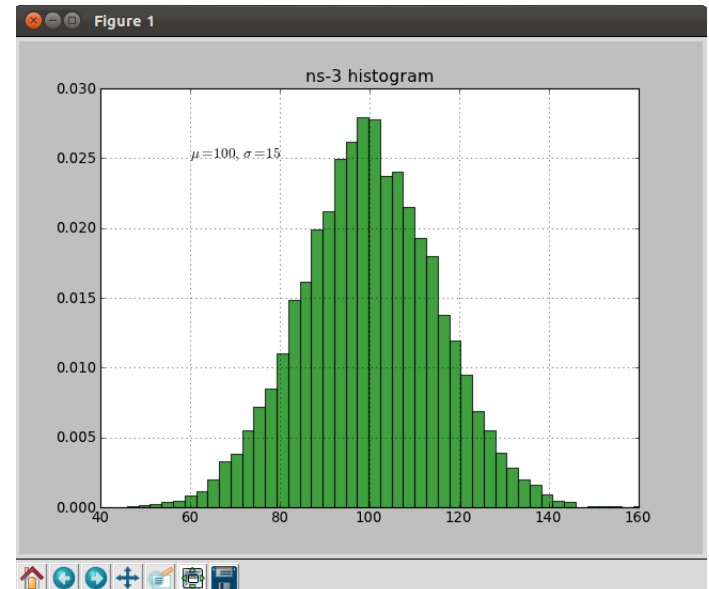
```
# Demonstrate use of ns-3 as a random number generator integrated with
# plotting tools; adapted from Gustavo Carneiro's ns-3 tutorial

import numpy as np
import matplotlib.pyplot as plt
import ns.core

# mu, var = 100, 225
rng = ns.core.NormalVariable(100.0, 225.0)
x = [rng.GetValue() for t in range(10000)]

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)

plt.title('ns-3 histogram')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```



Random variables and independent replications

- Many simulation uses involve running a number of *independent replications* of the same scenario
- In ns-3, this is typically performed by incrementing the simulation *run number* – *not by changing seeds*

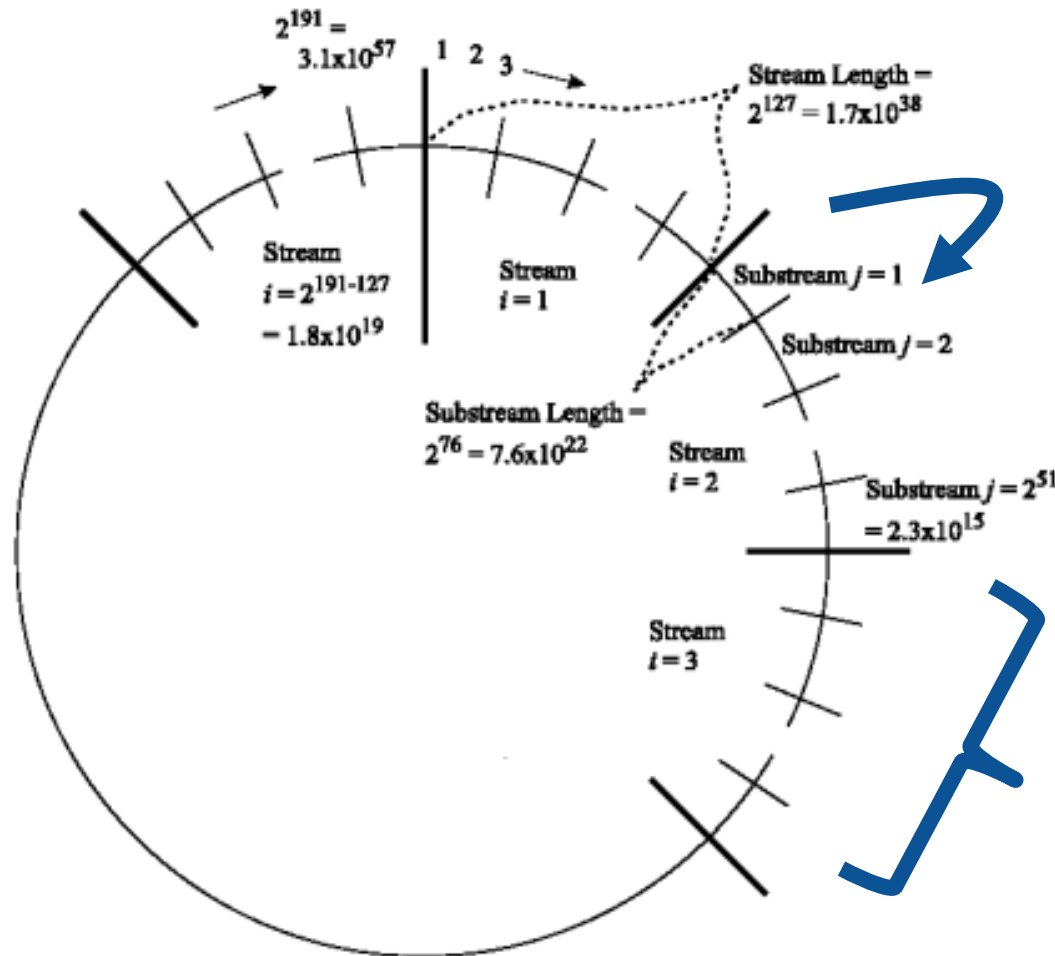
ns-3 random number generator

- Uses the MRG32k3a generator from Pierre L'Ecuyer
 - <http://www.iro.umontreal.ca/~lecuyer/myftp/papers/streams00.pdf>
 - Period of PRNG is 3.1×10^{57}
- Partitions a pseudo-random number generator into uncorrelated *streams* and *substreams*
 - Each RandomVariableStream gets its own stream
 - This stream partitioned into substreams

Key Terminology

- **Seed:** A set of values that generates an entirely new PRNG sequence
- **Stream:** The PRNG sequence is divided into non-overlapping intervals called streams
- **Run Number (substream):** Each stream is further divided to substreams, indexed by a variable called the run number.

Streams and Substreams



Incrementing the Run Number will move **all** streams to a new substream

Each ns-3 RandomVariableStream object is assigned to a stream (by default, randomly)

Figure source: Pierre L'Ecuyer, Richard Simard, E. Jack Chen, and W. David Kelton.
An object-oriented random number package with many long streams and substreams. Operations Research, 2001.

Run number vs. seed

- If you increment the seed of the PRNG, the streams of random variable objects across different runs are not guaranteed to be uncorrelated
- If you fix the seed, but increment the run number, you will get uncorrelated streams

Setting the stream number

- The ns-3 implementation provides access to 2^{64} streams
- 2^{63} are placed in a pool for automatic assignment, and 2^{63} are reserved for fixed assignment

```
<----->
^               ^^               ^
|               ||               |
stream 0        stream ( $2^{63} - 1$ )  stream  $2^{63}$         stream ( $2^{64} - 1$ )
<- automatically assigned -----><- assigned by user ----->
```

- Users may optionally assign a stream number index to a random variable using the `SetStream ()` method.
 - This allows better control over selected random variables
 - Many helpers have `AssignStreams ()` methods to do this across many such random variables

Putting it together

- Example of scheduled event

```
static void
RandomFunction (void)
{
    std::cout << "RandomFunction received event at "
               << Simulator::Now ().GetSeconds () << "s" << std::endl;
}
```

```
int main (int argc, char *argv[])
{
    CommandLine cmd;
    cmd.Parse (argc, argv);

    MyModel model;
    Ptr<UniformRandomVariable> v = CreateObject<UniformRandomVariable> ();
    v->SetAttribute ("Min", DoubleValue (10));
    v->SetAttribute ("Max", DoubleValue (20));

    Simulator::Schedule (Seconds (10.0), &ExampleFunction, &model);

    Simulator::Schedule (Seconds (v->GetValue ()), &RandomFunction);
}
```

Demo real-time, command-line, random variables...