ns-3 training

Tom Henderson

ns-3 annual meeting 2019

June 17-21, Florence, Italy

UNIVERSITY *of* WASHINGTON

Agenda and Instructors

> Monday: ns-3 overview (Tom Henderson)
  - software overview
  - sample program and experiments (M/M/1 queue)

> Tuesday AM: TCP (Tom Henderson)

> Tuesday AM: Wi-Fi (Sebastien Deronne)

> Tuesday PM: LTE (Zoraze Ali)

> Tuesday PM: sensor networks (Tommaso Pecorella and Davide Magrin)

Wiki: https://www.nsnam.org/wiki/AnnualTraining2019

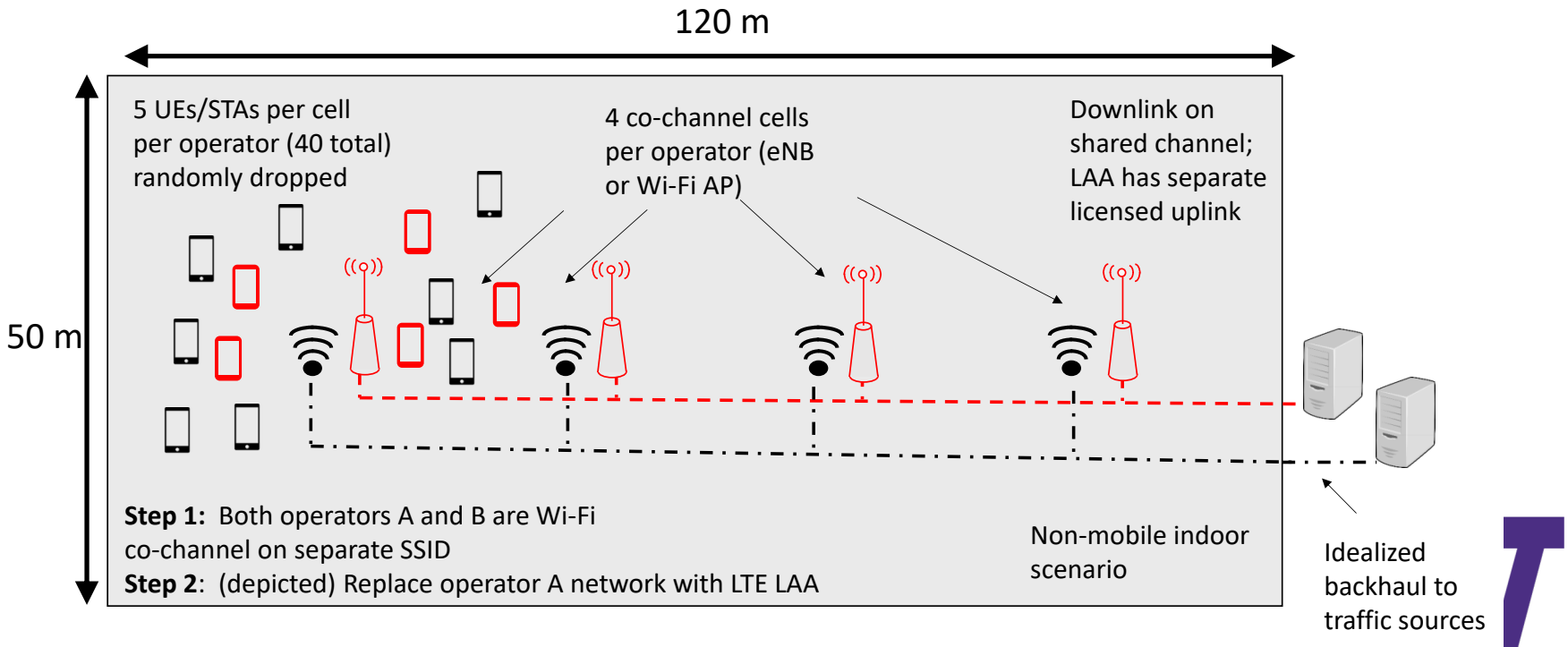
Working with training code

- See the wiki page for instructions on how to get and update the code used in this training
- https://www.nsnam.org/wiki/AnnualTraining2019

What is ns-3?

> Software to build models of computer networks, to conduct performance evaluation studies

Question: Can LTE safely co-exist with Wi-Fi?



Network Performance Analysis Fundamentals

- Studies are conducted to try to answer questions
- "Can LTE safely co-exist with Wi-Fi?"
  - Question is too broad; need to sharpen its focus
- **Guideline 1:** Clearly state the goals of the study and define the scope
- **Guideline 2:** Select performance metrics

- ***Refined question:*** "Can a specific unlicensed variant of LTE (LAA) operate in the same spectrum as a Wi-Fi network, without impacting Wi-Fi system throughput and latency more than another co-located Wi-Fi network would impact it?"

Network Performance Analysis Fundamentals (cont.)

- What do you mean by “throughput” and “latency”?
  - How measured? (precise definition)
  - What statistics? (average throughput, 99%th percentile, worst-case, etc.)?

- **Guideline 3:** Select system and experimental parameters

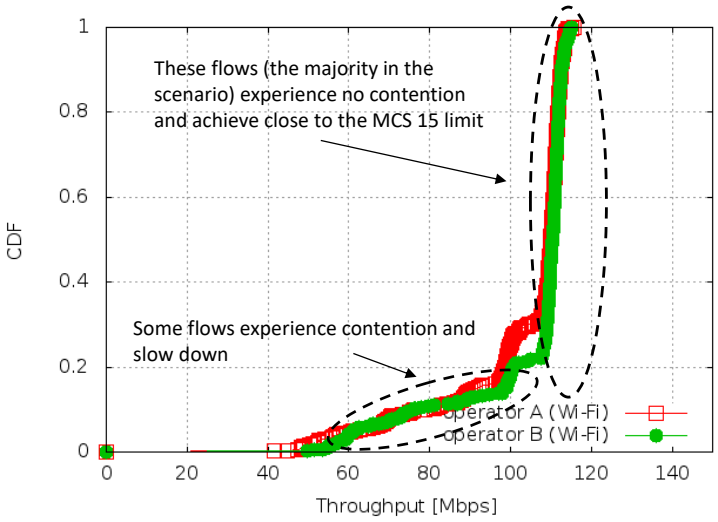
Network Performance Analysis Fundamentals (cont.)

Unlicensed channel model	3GPP TR 36.889	ns-3 implementation
Network Layout	Indoor scenario	Indoor scenario
System bandwidth	20 MHz	20 MHz
Carrier frequency	5 GHz	5 GHz (channel 36, tunable)
Number of carriers	1, 4 (to be shared between two operators) 1 for evaluations with DL+UL Wi-Fi coexisting with DL-only LAA	1 for evaluations with DL+UL Wi-Fi coexisting with DL-only LAA
Total Base Station (BS) transmission power	18/24 dBm	18/24 dBm Simulations herein consider 18 dBm
Total User equipment (UE) transmission power	18 dBm for unlicensed spectrum	18 dBm
Distance dependent path loss, shadowing and fading	ITU InH	802.11ax indoor model
Antenna pattern	2D Omni-directional	2D Omni-directional
Antenna height	6 m	6 m (LAA, not modelled for Wi-Fi)
UE antenna height	1.5 m	1.5 m (LAA, not modelled for Wi-Fi)
Antenna gain	5 dBi	5 dBi
UE antenna gain	0 dBi	0 dBi
Number of UEs	10 UEs per unlicensed band carrier per operator for DL-only 10 UEs per unlicensed band carrier per operator for DL-only for four unlicensed carriers. 20 UEs per unlicensed band carrier per operator for DL+UL for single unlicensed carrier. 20 UEs per unlicensed band carrier per operator for DL+UL Wi-Fi coexisting with DL-only LAA	Supports all the configurations in TR 36.889. Simulations herein consider the case of 20 UEs per unlicensed band carrier per operator for DL LAA coexistence evaluations for single unlicensed carrier.
UE Dropping	All UEs should be randomly dropped and be within coverage of the small cell in the unlicensed band.	Randomly dropped and within small cell coverage.
Traffic Model	FTP Model 1 and 3 based on TR 36.814 FTP model file size: 0.5 Mbytes. Optional: VoIP model based on TR36.889	FTP Model 1 as in TR36.814. FTP model file size: 0.5 Mbytes Voice model: DL only
UE noise figure	9 dB	9 dB
Cell selection	For LAA UEs, cell selection is based on RSRP (Reference Signal Received Power. For Wi-Fi stations (STAs), cell selection is based on RSS (Received signal power strength) of WiFi Access Points (APs). RSS threshold is -82 dBm.	RSRP for LAA UEs and RSS for Wi-Fi STAs
Network synchronization	For the same operator, the network can be synchronized. Small cells of different operators are not synchronized.	Small cells are synchronized, different operators are not synchronized.

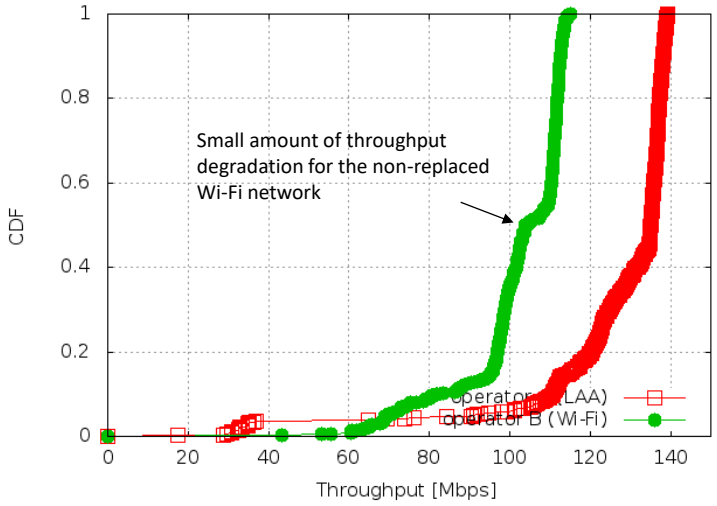
Figure from: http://arxiv.org/abs/1604.06826

Network Performance Analysis Fundamentals (cont.)

- **Guideline 4:** Design experiments
  - Select evaluation techniques
  - Select factors and their values
- Example: Place two Wi-Fi networks in same region, fully load the system, and plot a CDF of observed throughputs per station. Repeat by replacing one Wi-Fi network with LAA.



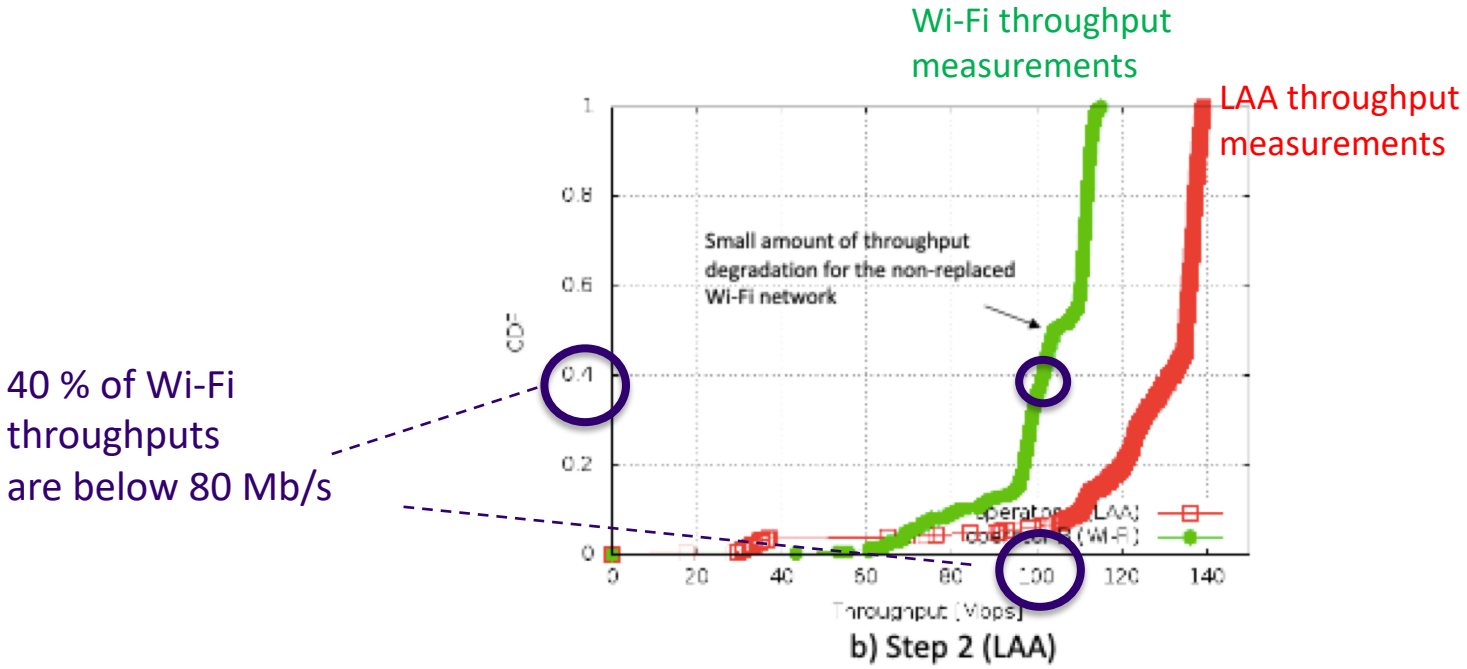
a) Step 1 (Wi-Fi)



b) Step 2 (LAA)

Aside: What is a 'CDF'?

> A cumulative distribution function measures the probability that samples fall below a specified value: For random variable X, F(x) = P(X <= x)



Why interesting?

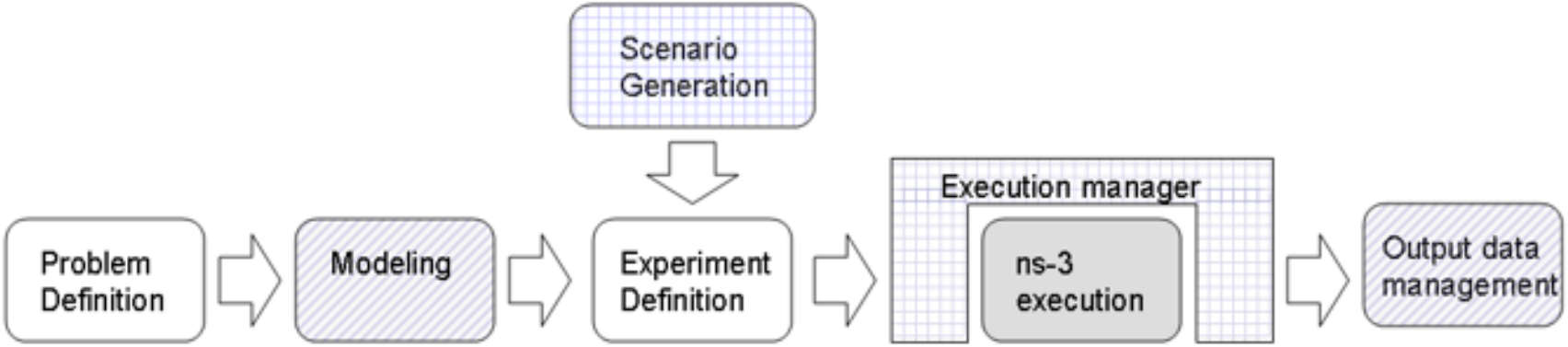
- Many networked systems are designed with concerns about worst-case behavior
- A CDF provides a sense of the spread of the data samples

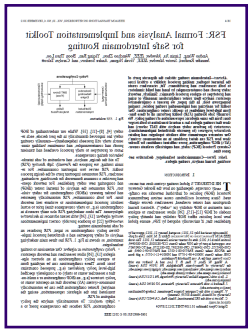
Network Performance Analysis Fundamentals (cont.)

- **Guideline 5**: Analyze and interpret data, and iterate
  - Almost never a one-shot process
  - Often need to dig deeper into model or scenario, to mine it for fine-grained detail
- **Guideline 6**: Make your results easy to reproduce
  - For others, and by yourself (at a later date)

What is ns-3?

> We have just hinted at a workflow:



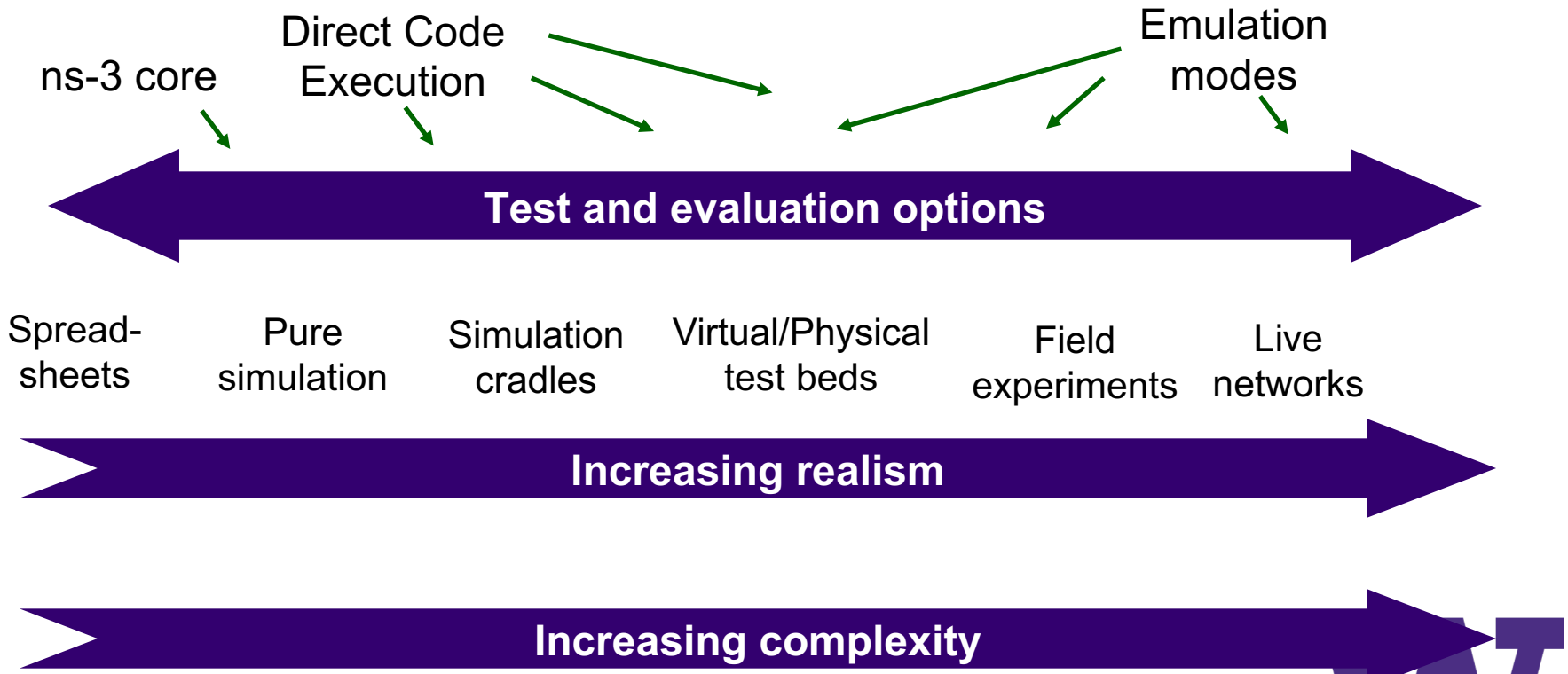


Performance evaluation alternatives

- Mathematical analysis
- Numerical computing packages (e.g., MATLAB)
- Packet-level simulators
- System-level simulators
- Testbeds, prototypes
- Field trials

What is ns-3? (cont.)

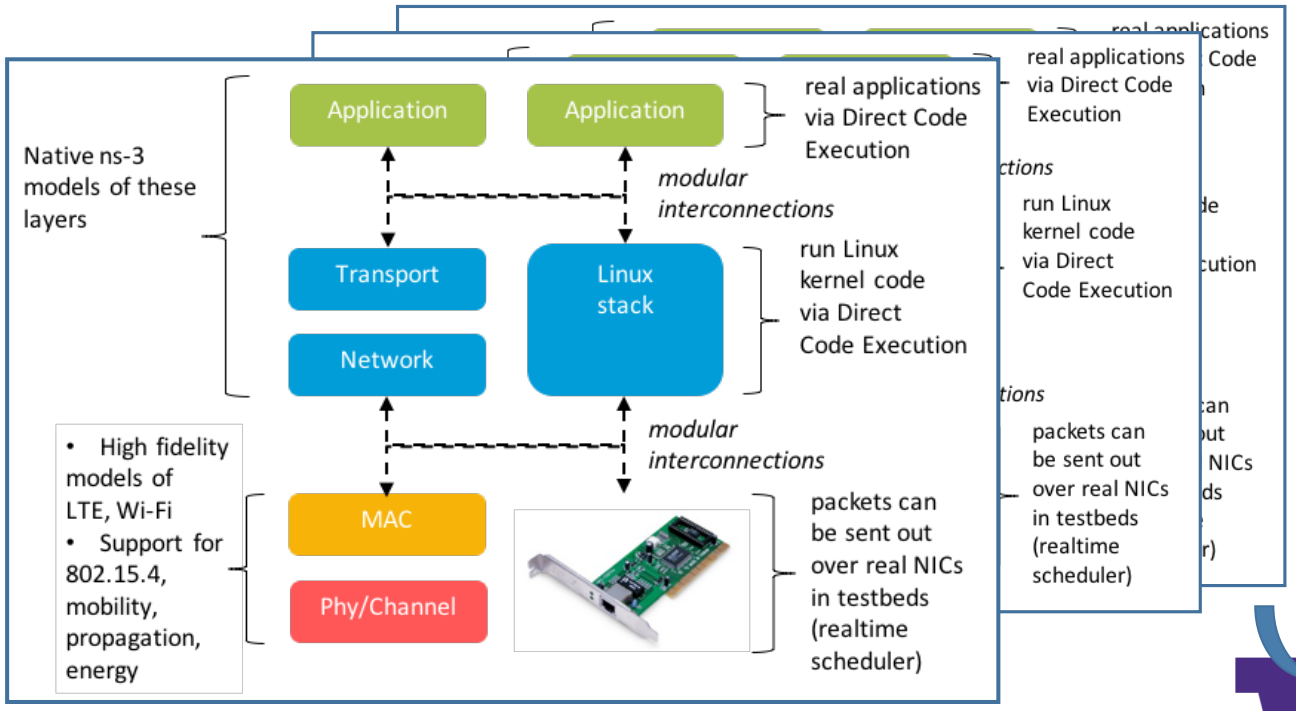
> ns-3 also has modes of operation that allows it to interact with real-world software and networks



What is ns-3? (cont.)

- ns-3 is a leading open source, **packet-level network simulator** oriented towards network research, featuring a **high-performance core** enabling **parallelization across a cluster** (for large scenarios), **ability to run real code**, and **interaction with testbeds**

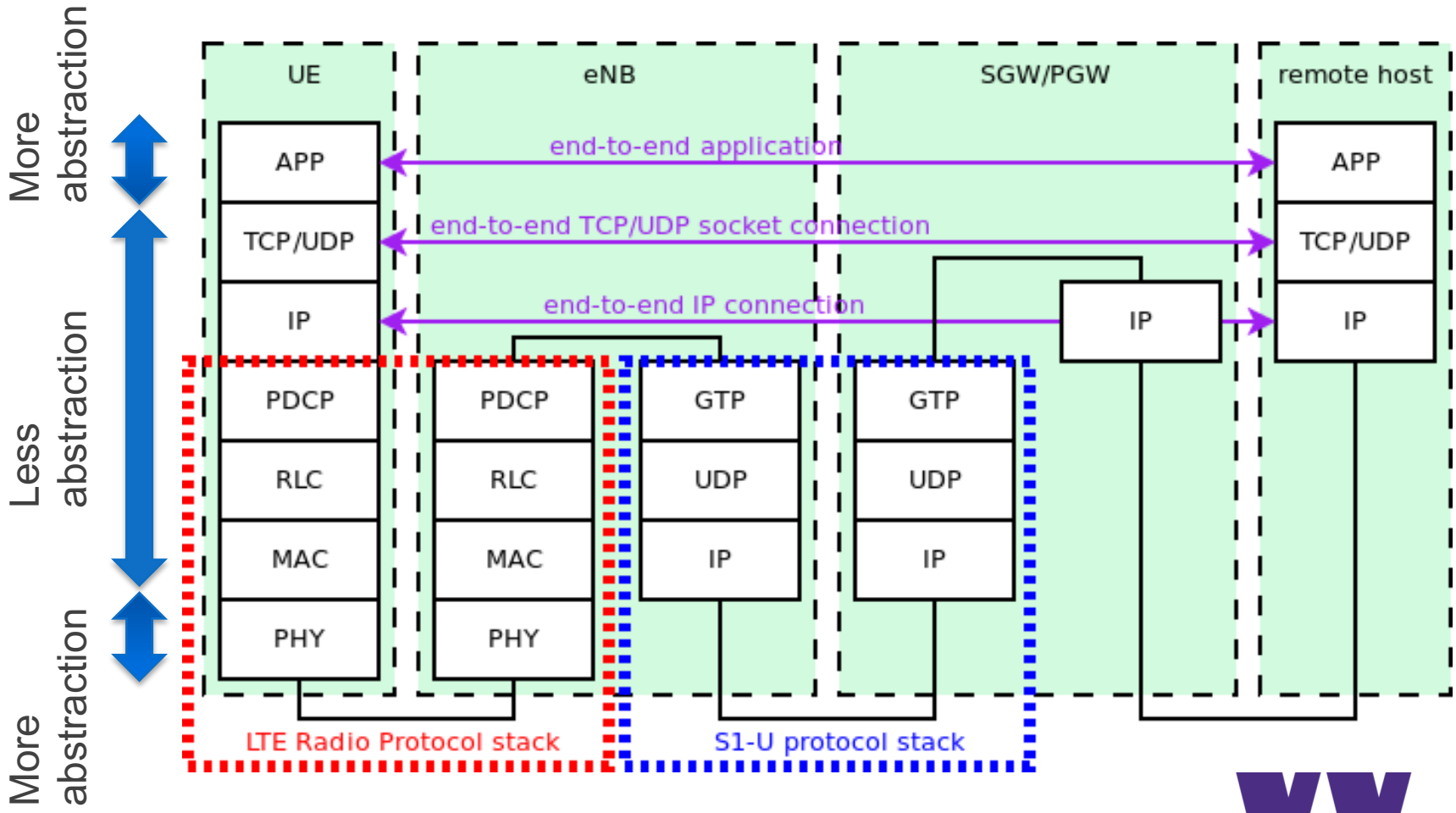
Runs on a single machine



or partitioned across a cluster

What is ns-3? (cont.)

> **Packet-level network simulation:** The main unit of modeling is the *packet* and entities that exchange packets.



ns-3 from the ground up

- ns-3 is written in C++
  - most code conforms to C++98 standard (starting to use C++11), and makes use of the STL (standard template library)
  - ns-3 makes use of a collection of C++ design patterns and enhancements with applicability to network simulation
  - ns-3 experiments can be written in Python (more on that later)
- ns-3 programs make use of standard C++, ns-3 libraries written in C++, and (optionally) third-party C++ libraries
- ns-3's build system requires a working Python (soon to require Python 3)
- Various other tools can be used to handle output data
  - We'll focus on Python matplotlib and Gnuplot

Example ns-3 program

> Located in scratch/ns3-hello-world.cc

This is basic C++, except:

1) we are using methods defined in an 'ns3' namespace

2) The object 'cmd' is an instance of the CommandLine C++ class.

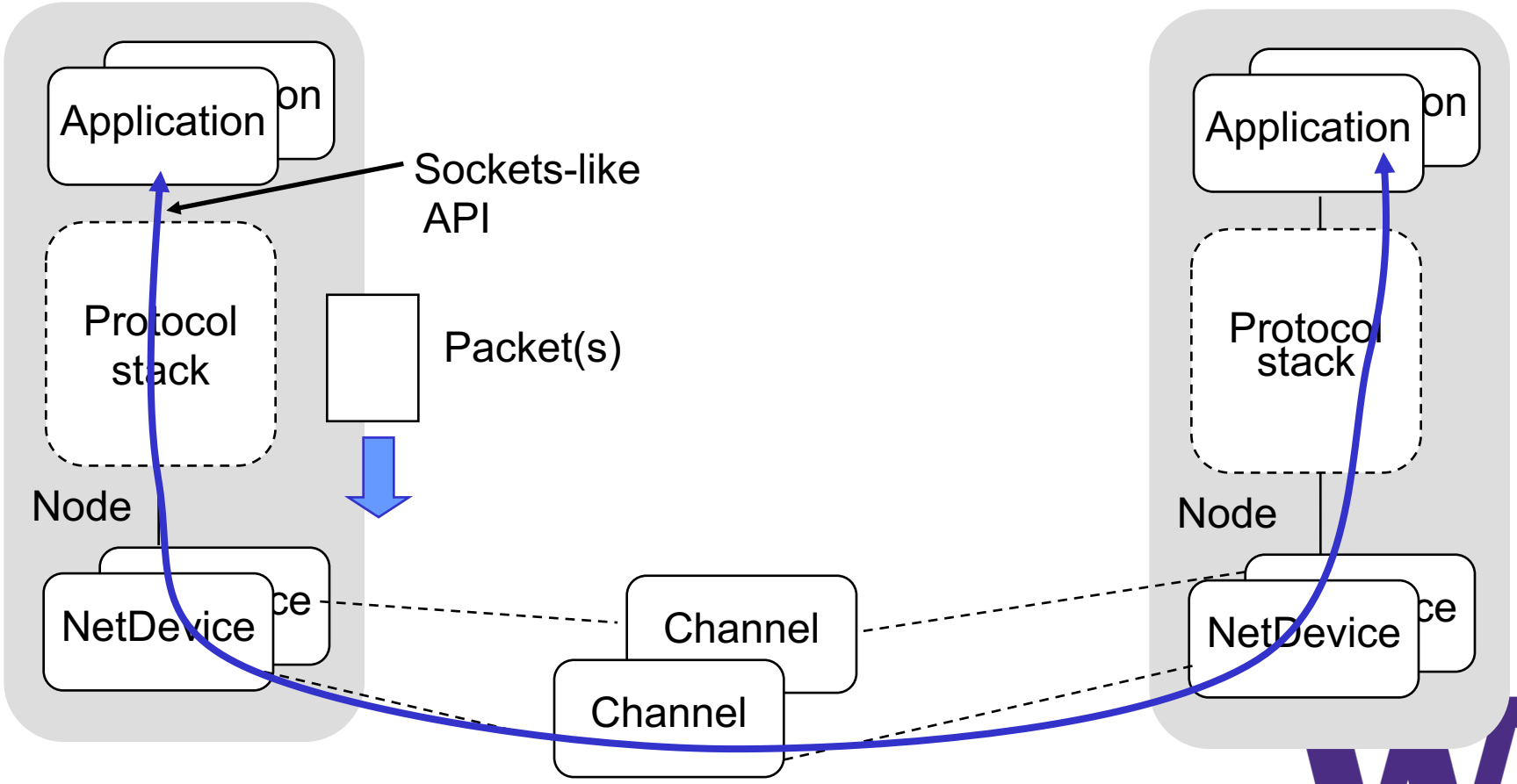
CommandLine exists in C++ namespace 'ns3'.

CommandLine objects process command-line arguments.

```
#include <iostream>
#include "ns3/command-line.h"

using namespace ns3;

int main (int argc, char *argv[])
{
  std::string language = "English";
  std::string phrase;

  CommandLine cmd;
  cmd.AddValue ("language", "Specify language", language);
  cmd.Parse (argc, argv);

  if (language == "English")
    {
      phrase = "Hello world";
    }
  else if (language == "Italian")
    {
      phrase = "Ciao mundo";
    }
  else
    {
      phrase = "That language is not spoken here";
    }

  std::cout << phrase << std::endl;
}
```

ns-3 from the top down

> Rather than (just) CommandLine objects, ns-3 combines objects like 'Packets', 'Nodes', 'Applications', etc.



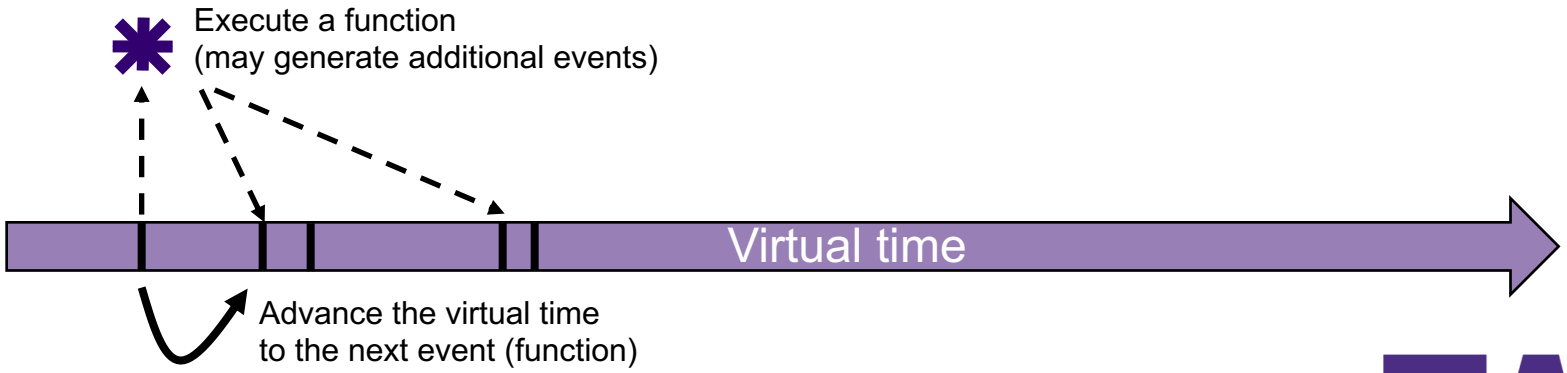
Discrete-event simulation basics

We are trying to represent the operation of a network within a single C++ program

- We need a notion of ***virtual time*** and of ***events*** that occur at specified (virtual) times
- We need a data structure (***scheduler***) to hold all of these events in temporal order
- We need an object (***simulator***) to walk the list of events and execute them at the correct virtual time
- We can choose to ignore things that conceptually might occur between our events of interest, focusing only on the (***discrete***) times with interesting events

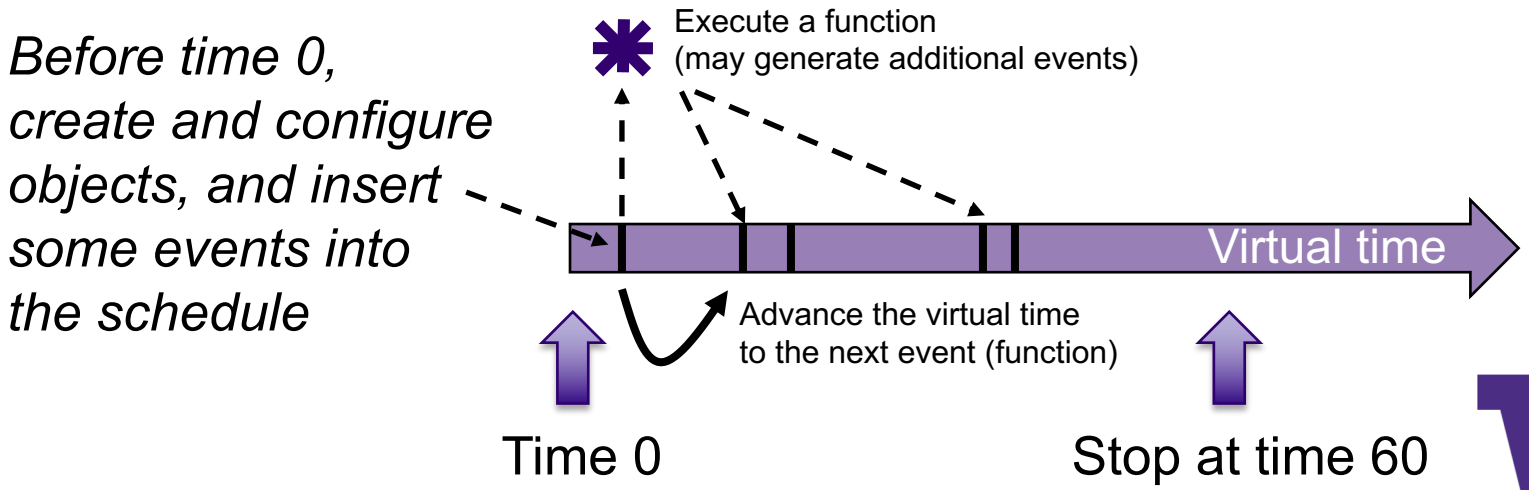
Discrete-event simulation basics (cont.)

- Simulation time moves in discrete jumps from event to event
- C++ functions schedule events to occur at specific simulation times
- A simulation scheduler orders the event execution
- Simulation::Run() executes a single-threaded event list
- Simulation stops at specified time or when events end



ns-3 simulation basics and terminology

- A simulation 'run' or 'replication' usually consists of the following workflow
  1. Before the notional 'time 0', create the scenario objects and pre-populate the scheduler with some initial events
  2. Define stopping criteria; either a specific future virtual time, or when certain criteria are met
  3. Start the simulation (which initializes objects, at 'time 0')



Virtual 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 and use Time arithmetic operators

Key building blocks: Callback and function pointer

> C++ methods are often invoked directly on objects

```
#include <iostream>
#include "ns3/command-line.h"

using namespace ns3;

int main (int argc, char *argv[])
{
  std::string language = "English";
  std::string phrase;

  CommandLine cmd;
  cmd.AddValue ("language", "Specify language", language);
  cmd.Parse (argc, argv);

  if (language == "English")
    {
      phrase = "Hello world";
    }
  else if (language == "Italian")
    {
      phrase = "Ciao mundo";
    }
  else
    {
      phrase = "That language is not spoken here";
    }

  std::cout << phrase << std::endl;
}
```

Unlike CommandLine.AddValue(), we more generally need to call functions at some future (virtual) time.

Some program element could assign a function pointer, and a (later) program statement could call (execute) the method

Events in ns-3

- Events are just functions (callbacks) that execute at a simulated time
  - nothing is special about functions or class methods that can be used as events
- Events have IDs to allow them to be cancelled or to test their status

Simulator and Scheduler

- 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

Simulator core

- Simulation time (✓)
- Events (✓)
- Simulator and Scheduler (✓)
- Command line arguments
- Random variables
- Example program walkthrough

CommandLine 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 "ns3-hello-world --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.
```

Random Variables and Run Number

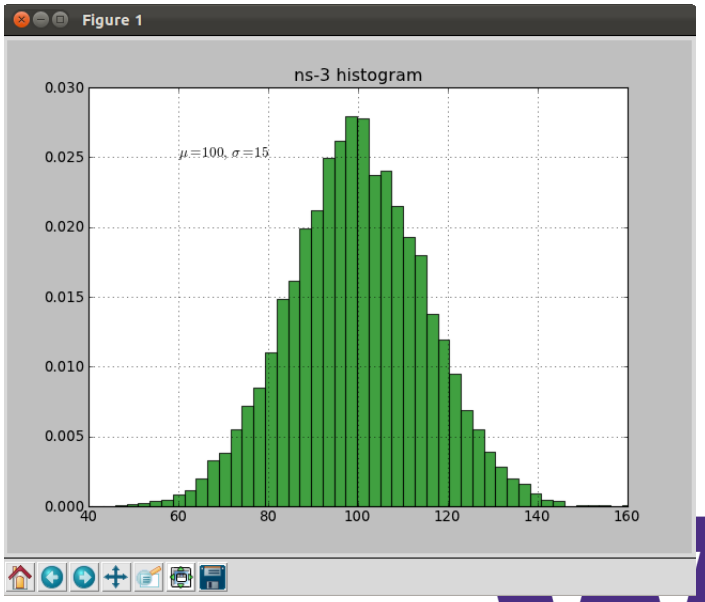
- Many ns-3 objects use random variables to model random behavior of a model, or to force randomness in a protocol
  - e.g. random placement of nodes in a topology

- Many simulation uses involve running a number of ***independent replications*** of the same scenario, by changing the random variable streams in use
  - In ns-3, this is typically performed by incrementing the simulation ***run number***

Random Variables

- 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, Zipf, Zeta, Deterministic, Empirical

```
# 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()
```

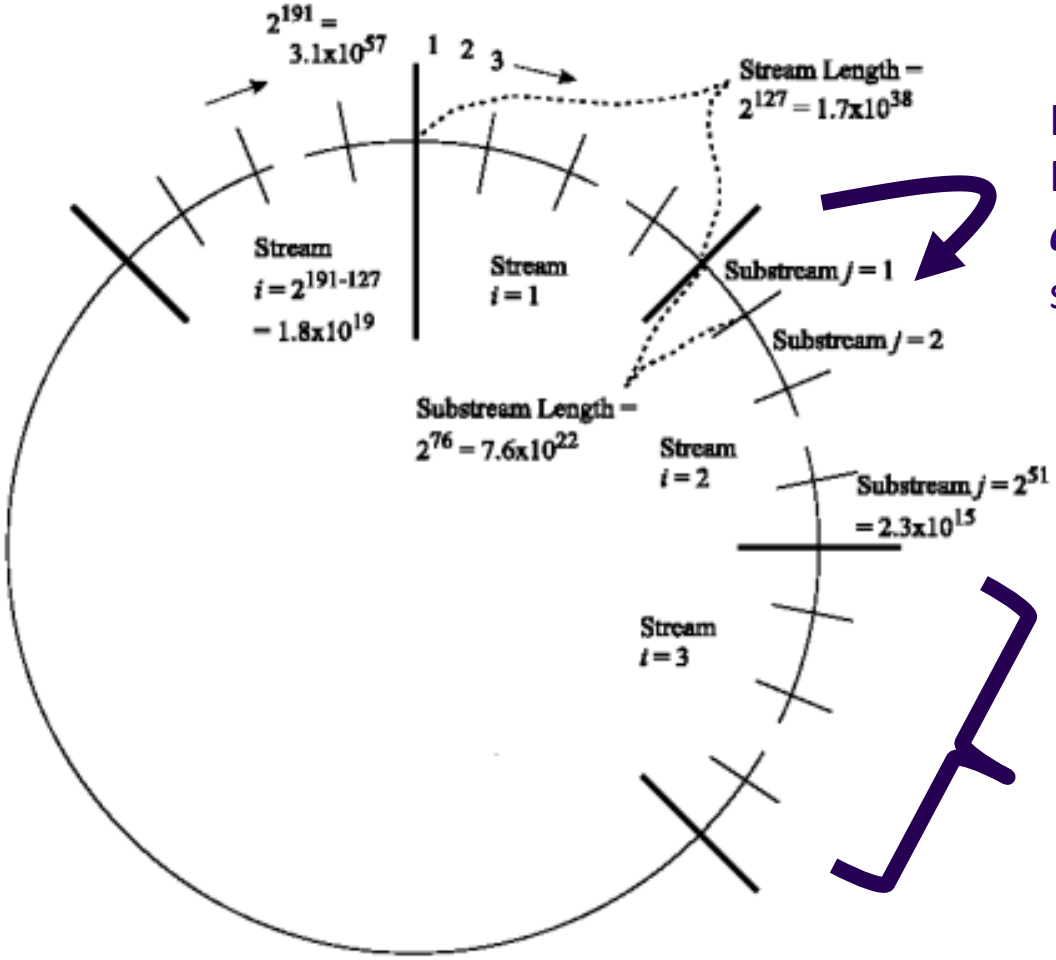


from src/core/examples/sample-rng-plot.py

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



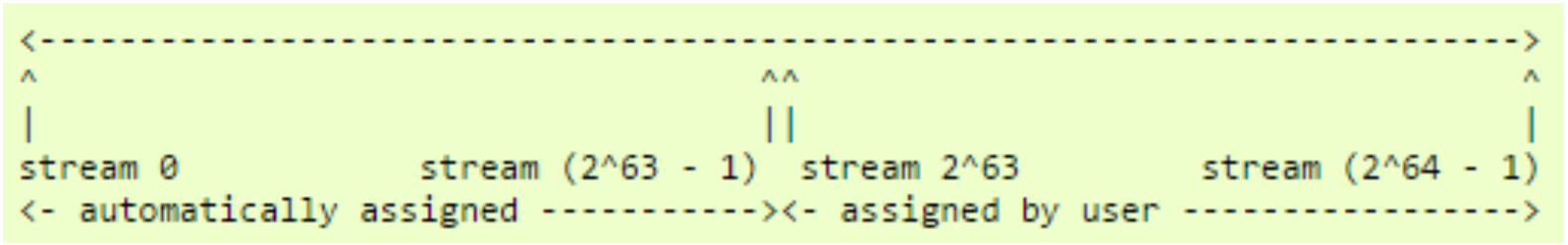
Incrementing the Run Number will move ***all*** streams to a new substream

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

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



- 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

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

Set RngRun, not RngSeed!

Example walk-through

> Example program: `src/core/examples/sample-simulator.cc`

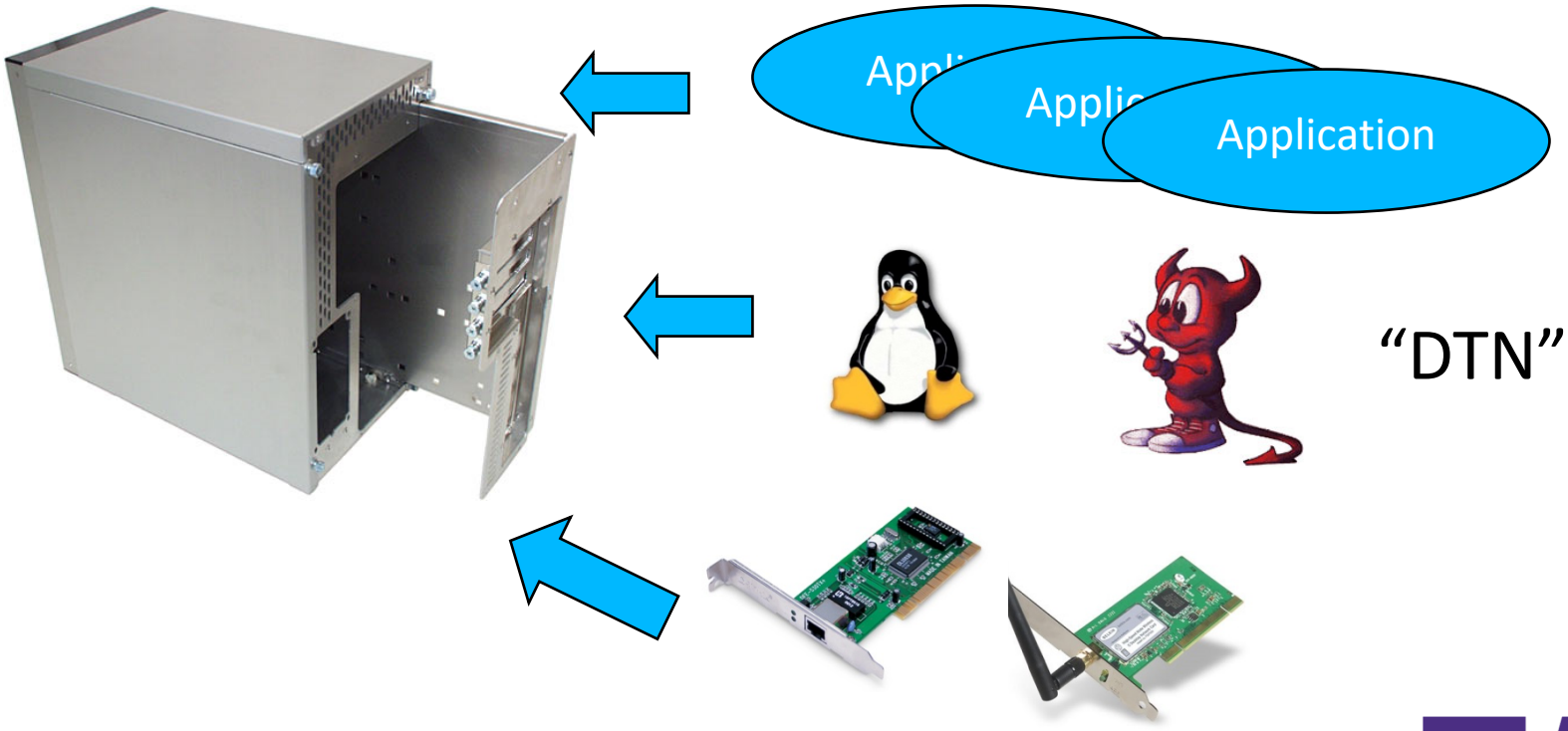
```
static void
RandomFunction (void)
{
  std::cout << "RandomFunction received event at "
            << Simulator::Now ().GetSeconds () << "s" << std::endl;
}
```

```
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 command line usage, event scheduling, random variables

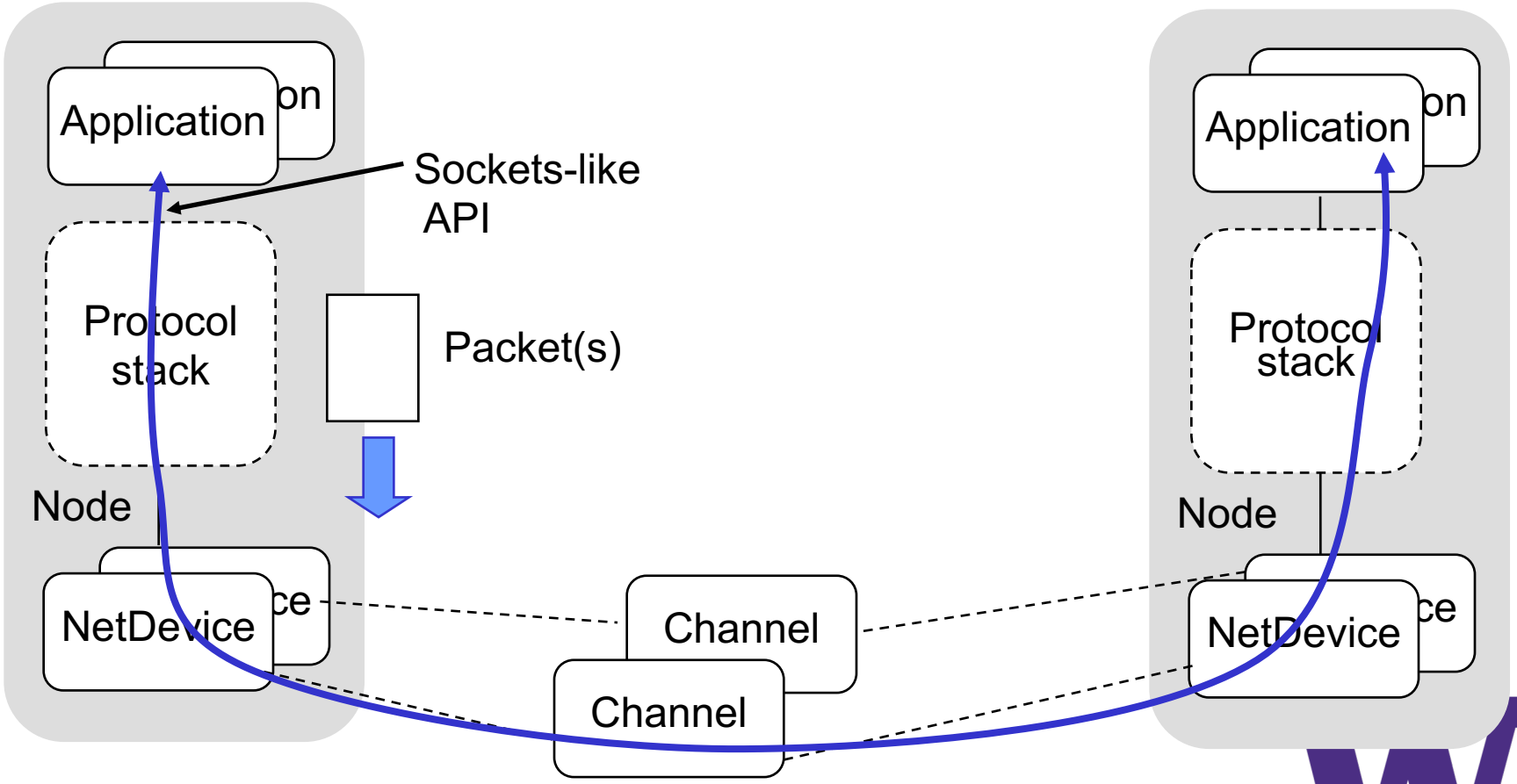
Node basics

> An ns-3 Node is a shell of a computer, to which applications, protocol stacks, and NetDevices are added



ns-3 example scenario

> Most simulations involve packet exchanges such as depicted below



Simulation setup

- We have already explained the operation of the start of the program (configuring default values and command line arguments)

```
int
main (int argc, char *argv[])
{
  DataRate dataRate = DataRate ("100Mbps");
  ...

  CommandLine cmd;
  cmd.AddValue("distance","the distance between the two
nodes",distance);
 ...
  cmd.Parse (argc, argv);
```

Nodes and ns-3 Objects

> The next statements create the scenario, usually starting with the Node objects:

```
Ptr<Node> senderNode = CreateObject<Node> ();
Ptr<Node> receiverNode = CreateObject<Node> ();
NodeContainer nodes;
nodes.Add (senderNode);
nodes.Add (receiverNode);
```

What is CreateObject<Node> ()?
What is a NodeContainer?

class ns3::Object

- ns-3 is, at heart, a C++ object system
- ns-3 objects that inherit from base class ns3::Object get several additional features
  - smart-pointer memory management (Class Ptr)
  - dynamic run-time object aggregation
  - an attribute system

Instead of:

```
Node* senderNode = new Node;
```

in ns-3, we write

```
Ptr<Node> senderNode = CreateObject<Node> ();
```

Create a new Node object on the heap

Create a smart pointer to hold objects of type Node

ns-3 attributes

- Attributes are special member variables that the Object system exposes in a way to facilitate configuration

- An Attribute can be connected to an underlying variable or function
  - e.g. TcpSocket::m_cwnd;
  - or a trace source

Helper API

- The ns-3 “helper API” provides a set of classes and methods that make common operations easier than using the low-level API
- Consists of:
  - container objects
  - helper classes
- The helper API is implemented using the low-level API
- Each function applies a single operation on a ''set of same objects”
  - A typical operation is "Install()"

Containers

- Containers are part of the ns-3 “helper API”
- Containers group similar objects, for convenience
  - They are often implemented using C++ std containers
- Container objects also are intended to provide more basic (typical) API

Helper API examples

- NodeContainer: vector of Ptr<Node>
- NetDeviceContainer: vector of Ptr<NetDevice>
- InternetStackHelper
- WifiHelper
- MobilityHelper
- OlsrHelper
- ... many ns-3 models provide a helper class

Installation onto containers

> Installing models into containers, and handling containers, is a key API theme

```
NodeContainer c;
c.Create (numNodes);
...
mobility.Install (c);
...
internet.Install (c);
...
```

Native IP models

- IPv4 stack with ARP, ICMP, UDP, and TCP
- IPv6 with ND, ICMPv6, IPv6 extension headers, TCP, UDP
- IPv4 routing: RIPv2, static, global, NixVector, OLSR, AODV, DSR, DSDV
- IPv6 routing: RIPng, static

IP address configuration

- An Ipv4 (or Ipv6) address helper can assign addresses to devices in a NetDevice container

```
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
csmaInterfaces = ipv4.Assign (csmaDevices);

...

ipv4.NewNetwork ();  // bumps network to 10.1.2.0
otherCsmaInterfaces = ipv4.Assign (otherCsmaDevices);
```

Applications and sockets

- In general, applications in ns-3 derive from the ns3::Application base class
  - A list of applications is stored in the ns3::Node
  - Applications are like processes
- Applications make use of a sockets-like API
  - Application::Start () may call ns3::Socket::SendMsg() at a lower layer

Sockets API

Plain C sockets

```
int sk;
sk = socket(PF_INET, SOCK_DGRAM, 0);
```

```
struct sockaddr_in src;
inet_pton(AF_INET,"0.0.0.0",&src.sin_ad
    dr);
src.sin_port = htons(80);
bind(sk, (struct sockaddr *) &src,
    sizeof(src));
```

```
struct sockaddr_in dest;
inet_pton(AF_INET,"10.0.0.1",&dest.sin_
    addr);
dest.sin_port = htons(80);
sendto(sk, "hello", 6, 0, (struct
    sockaddr *) &dest, sizeof(dest));
```

```
char buf[6];
recv(sk, buf, 6, 0);
}
```

ns-3 sockets

```
Ptr<Socket> sk =
udpFactory->CreateSocket ();
```

```
sk->Bind (InetSocketAddress (80));
```

```
sk->SendTo (InetSocketAddress (Ipv4Address
    ("10.0.0.1"), 80), Create<Packet>
    ("hello", 6));
```

```
sk->SetReceiveCallback (MakeCallback
    (MySocketReceive));
• […] (Simulator::Run ())

void MySocketReceive (Ptr<Socket> sk,
    Ptr<Packet> packet)
{
...
}
```

Mobility and position

- The MobilityHelper combines a **mobility model** and **position allocator**.
- Position Allocators setup initial position of nodes (only used when simulation starts):
  - **List:** allocate positions from a deterministic list specified by the user;
  - **Grid:** allocate positions on a rectangular 2D grid (row first or column first);
  - **Random position allocators:** allocate random positions within a selected form (rectangle, circle, …).
- Mobility models specify how nodes will move during the simulation:
  - **Constant:** position, velocity or acceleration;
  - **Waypoint:** specify the location for a given time (time-position pairs);
  - **Trace-file based:** parse files and convert into ns-3 mobility events, support mobility tools such as SUMO, BonnMotion (using NS2 format) , TraNS

Propagation

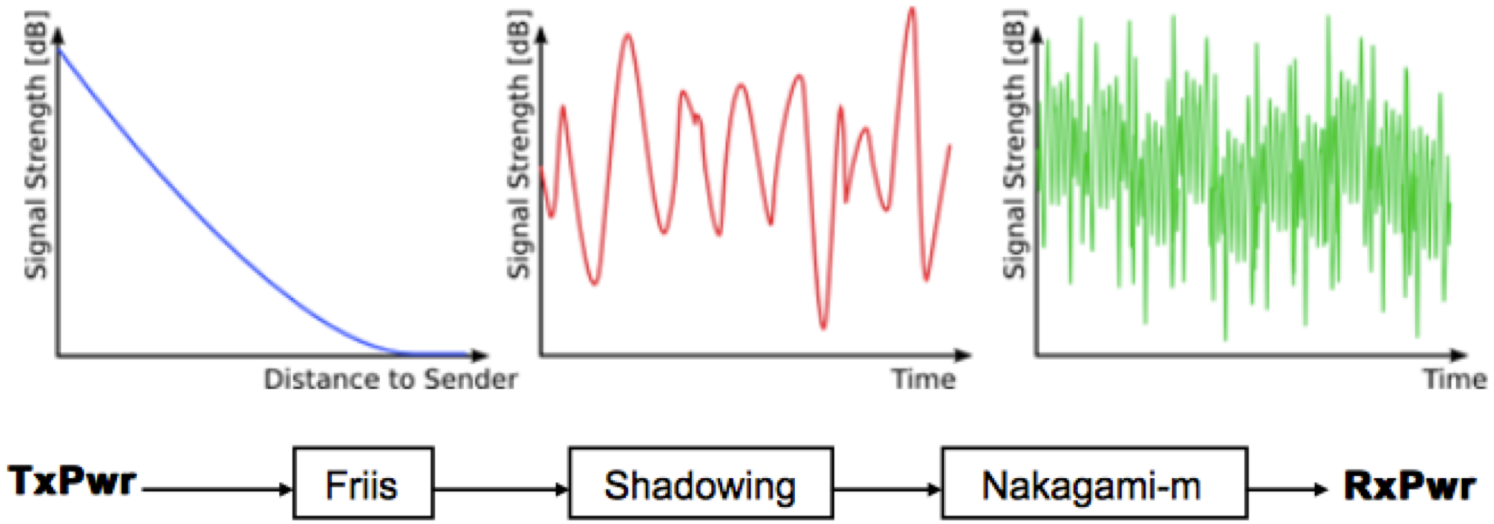
- Propagation module defines:
  - Propagation loss models:
    Calculate the Rx signal power considering the Tx signal power and the respective Rx and Tx antennas positions.
  - Propagation delay models:
    Calculate the time for signals to travel from the TX antennas to RX antennas.
- Propagation delay models almost always set to:
  - ConstantSpeedPropagationDelayModel: In this model, the signal travels with constant speed (defaulting to speed of light in vacuum)

Propagation (cont.)

- Propagation loss models:
  - Many propagation loss models are implemented:
    - ✓ Abstract propagation loss models:
      FixedRss, Range, Random, Matrix, …
    - ✓ Deterministic path loss models:
      Friis, LogDistance, ThreeLogDistance, TwoRayGround, …
    - ✓ Stochastic fading models:
      Nakagami, Jakes, …

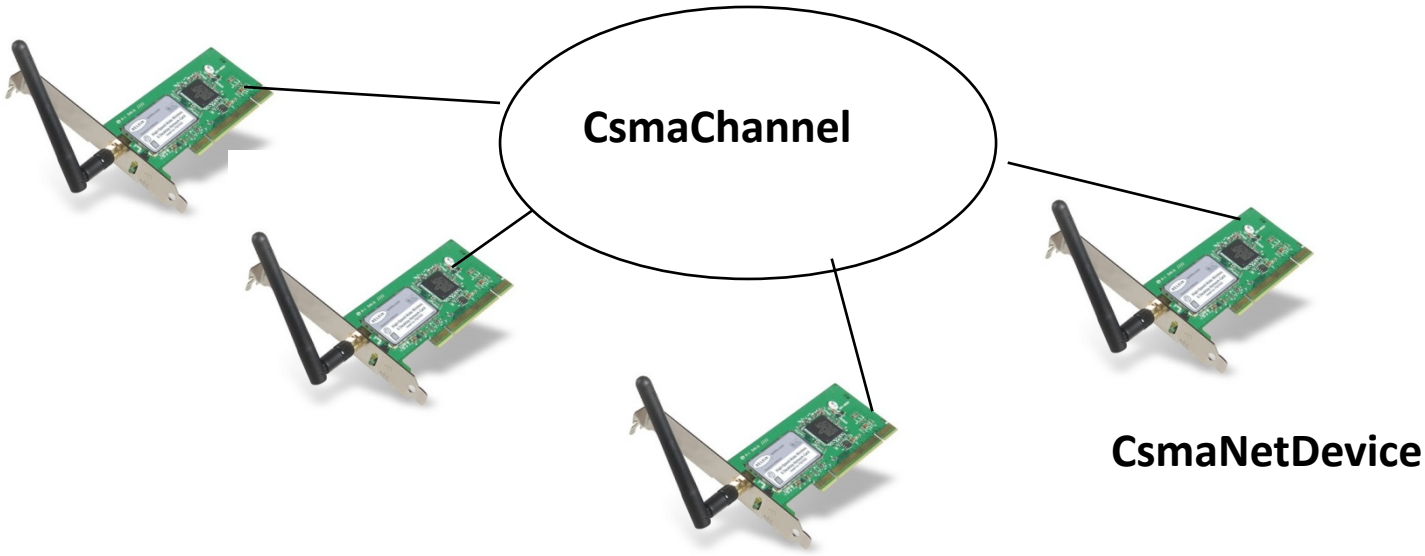
Propagation (cont.)

– A propagation loss model can be “chained” to another one, making a list. The final Rx power takes into account all the chained models.
Example: path loss model + shadowing model + fading model



NetDevices and Channels

Some types of NetDevices are strongly bound to Channels of a matching type

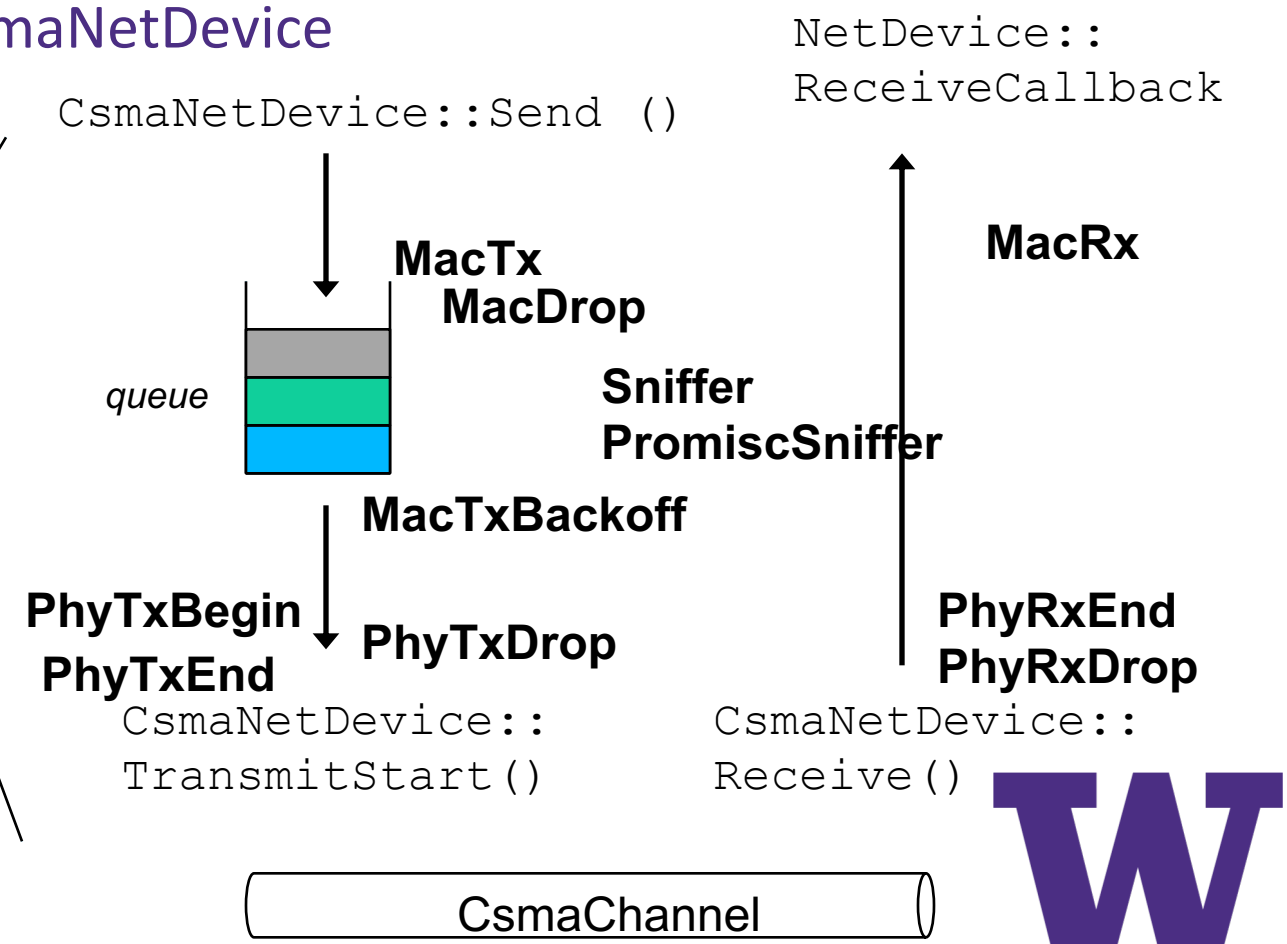


More recently, NetDevices use a channel allowing multiple signal types to coexist

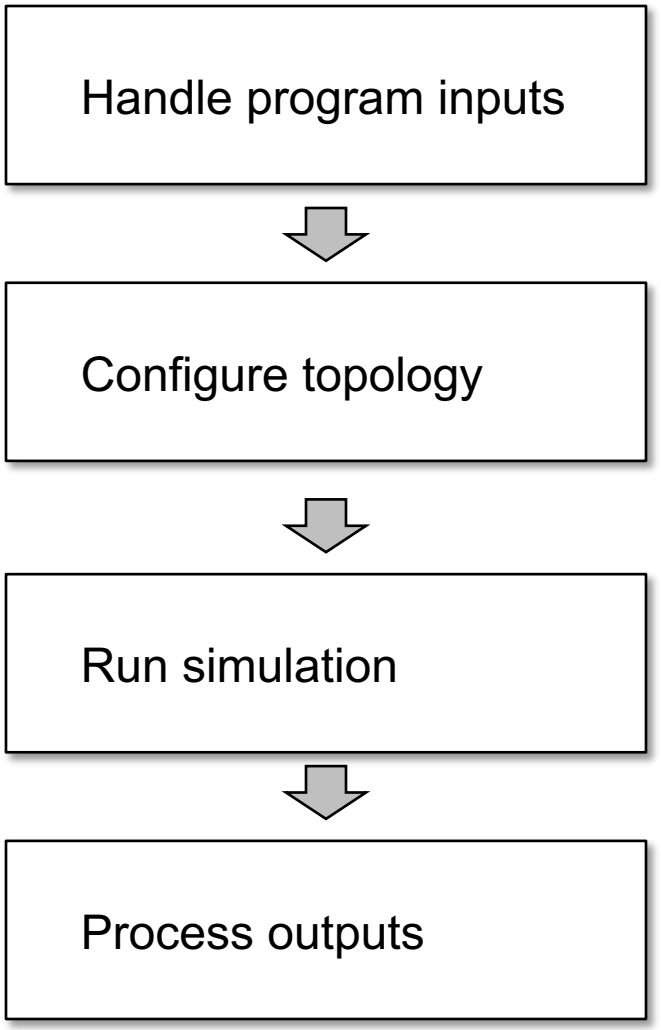
> SpectrumChannel

NetDevices and traces

- ns-3 TraceSource objects are callbacks that may be hooked to obtain trace data from the simulator
- Example: CsmaNetDevice



ns-3 program structure



Placeholder

> Review examples/tutorial/first.cc

Next steps

- Code organization and build system
- Documentation system
- Packet objects and queues
- Walkthrough of ‘mm1-queue.cc’ example
  - Simple experiment management
  - Objects, attributes, tracing
  - Logging and debugging