Computer Networks (2023-2024)
Lab6 introduction

Andrei Belogaev
andrei.belogaev@uantwerpen.be

University of Antwerp
Faculty of Science

Table of contents

Discrete-event simulation

Simulation vs emulation

- Emulators (e.g., Mininet):
  - Replicate the behavior of a specific real system or real hardware
  - Provide a high level of accuracy and detail
  - Work in real time
  - Usually are used for debugging and testing

- Simulators (e.g., NS-3)
  - Model the behavior of a real system or hardware without following the exact procedures
  - Involve implementation of a simplified representation of real system, e.g., using mathematical equations
  - Works in model time
  - Usually is used for experimentation, optimization and analysis

Events queue

Each **event** is described by:

- Identifier
- Moment in model time when the event occurs
- Sequence of actions (a function) that should be executed when the event occurs

During the execution of function, new events can be created, or some existing events can be deleted.

Event queue: $t_1 \le t_2 \le t_3 \le \ldots \le t_N$

$(ID_1, t_1, func_1)$	$(ID_2, t_2, func_2)$	$(ID_3, t_3, func_3)$	**…**	$(ID_N, t_N, func_N)$

```
func1 (args) { ...
  if <condition1> {
    Create event (IDi, ti, funci)
    Delete event (IDj)
  } ...
```

Data structure?

University of Antwerp
Faculty of Science

Events queue

Each **event** is described by:

- Identifier
- Moment in model time when the event occurs
- Sequence of actions (a function) that should be executed when the event occurs

During the execution of function, new events can be created, or some existing events can be deleted.

Event queue: $t_1 \leq t_2 \leq t_3 \leq \ldots \leq t_N$

$(ID_1, t_1, func_1)$	$(ID_2, t_2, func_2)$	$(ID_3, t_3, func_3)$	**...**	$(ID_N, t_N, func_N)$

$func_1$ (args) { ...
  if <condition1> {
    Create event $(ID_i, t_i, func_i)$
    Delete event (ID_j)
  } ...

Data structure?

std::map<t, event>

Terminating event

Terminating event is the last event. It finishes the simulation.

After this event:

- All remaining events in the queue are deleted
- Memory allocated to various objects is cleaned
- The data collected during the experiment is processed

Event queue: $t_1 \leq t_2 \leq t_3 \leq \ldots \leq t_N$

$(ID_1, t_1, func_1)$	$(ID_2, t_2, func_2)$	$(ID_3, t_3, func_3)$	…	$(ID_N, t_N, func_N)$

Simulation time

Scheduling events

To add an event to the event queue, we **schedule** it to be executed at a certain moment of time.

ID_k = Schedule (t_k, $event_k$)

(ID_k, t_k, $func_k$)

Event queue: $t_1 \leq t_2 \leq t_3 \leq \ldots \leq t_i \leq t_k \leq t_j \leq t_N$

(ID_1, t_1, $func_1$) | (ID_2, t_2, $func_2$) ••• (ID_i, t_i, $func_i$) | (ID_j, t_j, $func_j$) ••• (ID_N, t_N, $func_N$)

Callbacks - I

Desired behavior: a server sends one packet every *gap* seconds.

class **Server**

Methods:

Send () {
<send packet>;
Simulator::Schedule (gap, Send);
}

Fields:

Time gap

class **Simulator**

Methods:

static ID Schedule (t, func)

Fields:

std::map<t, func> queue

(ID_1, t_1, $func_1$)	(ID_2, t_2, $func_2$)	(ID_3, t_3, Server::Send)	…	(ID_i, t_3+gap, Server::Send)	…

Class **Simulator** does not have direct access to class Server. **How to call function Send then?**

Callbacks - II

Desired behavior: a server sends one packet every *gap* seconds.

class **Simulator**

Methods:
static ID Schedule (t, func)

Fields:
std::map<t, func> queue

class **Server**

Methods:
Send () {
 <send packet>;
 Simulator::Schedule (gap, Send);
}

Fields:
Time gap

Pure C++ style:

std::function<void ()> func = std::bind (&Server::Send, this, *packet*);
Simulator::Schedule (gap, func);

Callback

<Object*, function*, args>

Callbacks - III

Desired behavior: a server sends one packet every *gap* seconds.

class **Simulator**

Methods:
static ID Schedule (t, func)

Fields:
std::map<t, func> queue

class **Server**

Methods:
Send () {
<send packet>;
Simulator::Schedule (gap, Send);
}

Fields:
Time gap

NS-3 style:

Simulator::Schedule (gap, &Server::Send, this);
//if args present: Simulator::Schedule (gap, &Server::Send, this, args);

Callback (created internally)

<Object*, function*, args>

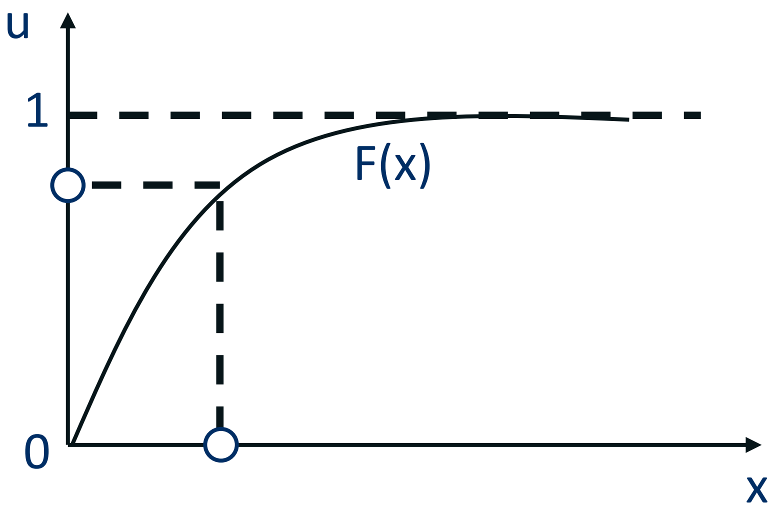
Random variables

An important property of simulation is the *experiment repeatability,* i.e., two or more runs of the simulation model with the same parameters will produce precisely the same results.

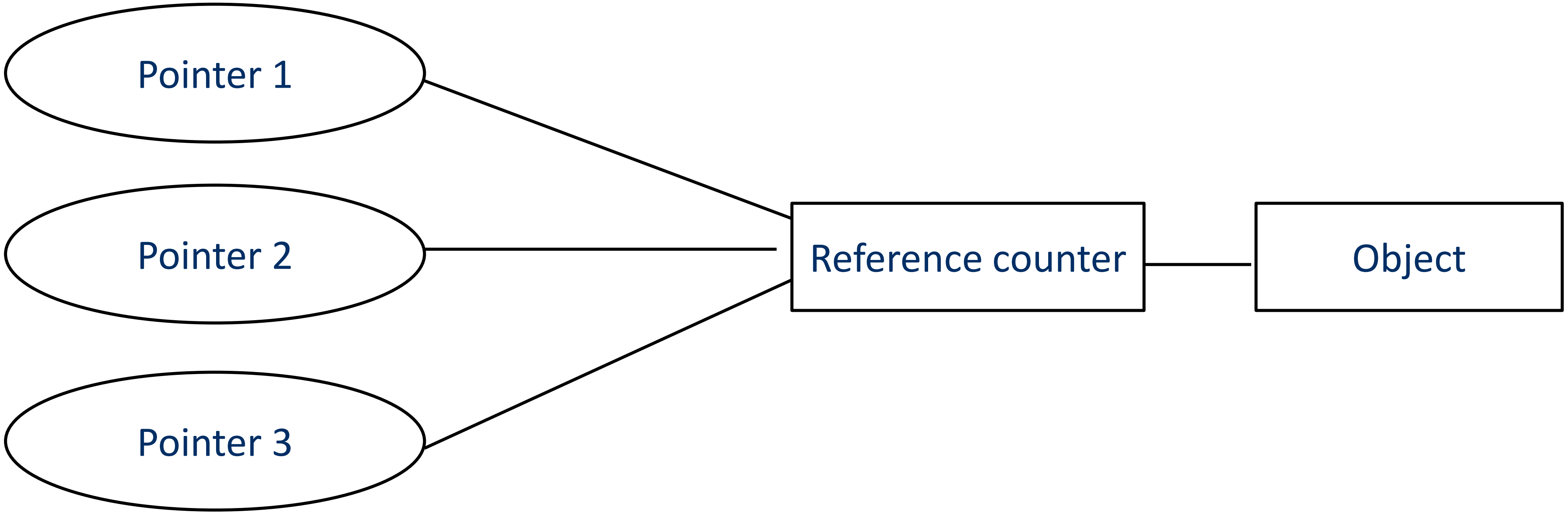
To model stochastic processes, where events occur with a certain probability, the *pseudorandom number generator* is used. This is an algorithm for generating a sequence of numbers whose properties approximate the properties of uniformly distributed random numbers. The sequences are determined by an initial value called *seed*.

To get the random variable with cumulative distribution F from uniformly distributed random variable U[0, 1], the *inverse CDF method* can be used:

$$\Pr(F^{-1}(U) \leq x) : \{u: F^{-1}(u) \leq x\} = \{u: u \leq F(x)\}$$
$$= \Pr\big(U \leq F(x)\big) : \Pr(U \leq u) = u \text{ when } U \text{ is uniform } [0, 1]$$
$$= F(x)$$



Smart pointers



When the reference counter becomes 0, the object is deleted, and the memory is freed.

Ptr<A> a = CreateObject<A> (); *//pure C++ style is* std::shared_ptr<A> obj (new A ());

Ptr<A> b = a; //+1 reference

a = 0; //-1 reference

b = 0; //-1 reference, the object is deleted

Typical source code of an experiment

```
int main () {
  <reading arguments from command line>
  …
  Simulator::SetSeed (seed); // seed of pseudorandom number generator
  …
  Simulator::Schedule (<first events>);
  Ptr<Object> some_object = CreateObject<Object> (); // can also create events, e.g., in constructor
  …
  Simulator::Schedule (<terminating event>);
  Simulator::Run (); //start iteration over the event queue
  ...
  <analysis of collected statistics>
}
```

NS-3 introduction

Simulator core

What is NS-3

Network simulator-3 (NS-3) – discrete-event simulation platform aimed at investigating various network protocols.

Open source project, license GNU GPLv2, i.e., allows modifying code for personal purposes, including for commercial organizations. The NS-3 based products shall be distributed with the same license.

Repository: https://gitlab.com/nsnam/ns-3-dev

How to participate in the project:

- Clone repo into personal gitlab
- Find bag Sor feature-request: https://gitlab.com/nsnam/ns-3-dev/-/issues
- Solve a problem, create merge request and select a reviewer (NS-3 maintainer)
- Participate in discussion https://gitlab.com/nsnam/ns-3-dev/-/merge_requests
- Optional: Google Summer of Code https://www.nsnam.org/wiki/Summer_Projects (stipend from Google $2700 for medium-size and $5400 for large-size project).

NS-3 applications

Maintained external repositories, that implement features not available in NS-3 mainline.
https://www.nsnam.org/docs/contributing/html/external.html
App store: https://apps.nsnam.org/

Examples:

- Direct code execution (DCE) – framework to run real applications and Linux kernel code within NS-3
- NS-3-ai – extension module enabling interaction between NS-3 and Python-based AI frameworks, such as TensorFlow and PyTorch
- NS-3-gym – a framework that integrates OpenAI Gym and NS-3
- NetSimulyzer – renders the scenario's topology in 3D, provides configurable charts and logging

NS-3 documentation

Tutorial: https://www.nsnam.org/docs/tutorial/html/

Manual: https://www.nsnam.org/docs/manual/html/index.html

Doxygen: https://www.nsnam.org/doxygen/

Other resources: https://www.nsnam.org/documentation/

Event queue

Class **Simulator**

- Scheduling:
  Simulator::Schedule (Time delay, MEM func_ptr, OBJ obj_ptr, args)
  For example:
  EventId event = Simulator::Schedule (Seconds (12), &SocketWriter::Write, socketWriter, 10)

- Cancel event:
  Simulator::Cancel (EventId event) or *EventId::Cancel ()*

- Terminating event, start and stop simulation
  Simulator::Stop (Time simtime), Simulator::Run (), Simulator::Destroy ()

Model time

Class **Time**

- Represented by a 64-bytes integer variable (uint64_t);
- The time resolution can be set before we call Simulator::Run
  Time::SetResolution (unit), by default nanoseconds;
- Special functions to create objects of type Time and units convesion:
  Seconds (1), MilliSeconds (100), t.GetSeconds (), t.GetMilliSeconds ();
- Arithmetic operations with Time objects:
  *operator +, -, +=, -=, * (by a number), <, >, <=, >=, ==*

Base class object

All child classes get:

- Smart pointers support } Base class SimpleRefCount
- Object aggregation
- System of attributes

} Base class ObjectBase

<u>Example:</u>

Class A : public Object {…};

Создать объект:

Ptr<A> a = CreateObject<A> ();

The direct call of operator new should never be used!

Object aggregation

Allows objects storing pointers to other objects.

An object can store at most one pointer to an object of a particular type.

<u>Example:</u>

Ptr<Node> node = CreateObject<Node> ();

Ptr<MobilityModel> mobility = CreateObject<MobilityModel> ();

node **-> AggregateObject** (device);

…

Ptr<MobilityModel> mobility = node **-> GetObject<MobilityModel>** ();

Vector position = mobility -> GetPosition ();

System of attributes - I

Example:

```
TypeId DropTailQueue::GetTypeId (void) {
static TypeId tid = TypeId ("ns3::DropTailQueue") // string identifier
.SetParent<Queue> () // base class (used for conversion)
.AddConstructor<DropTailQueue> () // constructor (used by object factories)
.AddAttribute ("MaxPackets", // string attribute identifier
"The maximum number of packets accepted by this DropTailQueue.", // text description
UintegerValue (100), // default value
MakeUintegerAccessor (&DropTailQueue::m_maxPackets), // reference to the field of the class
MakeUintegerChecker<uint32_t> ()); // checker for the attribute value
return tid;
```

System of attributes - II

Creation of new objects and setting their attributes:

- Using SetAttribute:

```
ptr = CreateObject<DropTailQueue> ();
ptr-> SetAttribute(" MaxPackets ", UintegerValue (60));
```

- Using object factory:

```
ObjectFactory factory;
factory.SetTypeId ("ns3::DropTailQueuel")
factory. Set ("MaxPackets ", UintegerValue (60));
ptr = factory.Create ();
```

- Setting attribute default value:

```
Config::SetDefault ("ns3::DropTailQueue::MaxPackets", UintegerValue (60));
```

Random variables - I

Pseudorandom number generator creates sequence $X(seed) = \{x_1, x_2, \ldots, x_N\}$, N ~ 3.1×10^57.
Different *seed* values do not guarantee that sequences $X(seed_1)$ и $X(seed_2)$ do not overlap.

To conduct statistically independent experiments:

- fixed seed
- different runs (./ns3 –run “experiment –RngRun=…”)

$$x_1\ x_2 \ldots x_K \quad x_{K+1}\ x_{K+2} \ldots x_{2K} \quad \ldots \quad x_{(M-1)*K+1}\ x_{(M-1)*K+2} \ldots x_N$$

M ~ 2^64 independent ***streams***

$$x_1\ x_2 \ldots x_L \quad x_{L+1}\ x_{L+2} \ldots x_{2L} \quad \ldots \quad x_{(P-1)*L+1}\ x_{(P-1)*L+2} \ldots x_K$$

P ~ 2.3×10^15 ***runs***, each contains L ~ 7.6×10^22 numbers

University of Antwerp
Faculty of Science

Random variables - II

For each random variable, NS-3 assigns streams sequentially.
Rus is selected according to global variable g_rngRun.

<u>Example:</u> exponential random variable, CDF $F(x) = 1 - e^{-\lambda x}$

```
Ptr<ExponentialRandomVariable> x = CreateObject<ExponentialRandomVariable> ();
x->SetAttribute (“Mean”, DoubleValue (mean)); // mean = 1 / λ
double gap = x->GetValue (); // -mean * ln(U[0, 1])
```

Main abstractions

- *Node:* basic computing device, is located at a certain position and can have a mobility pattern.

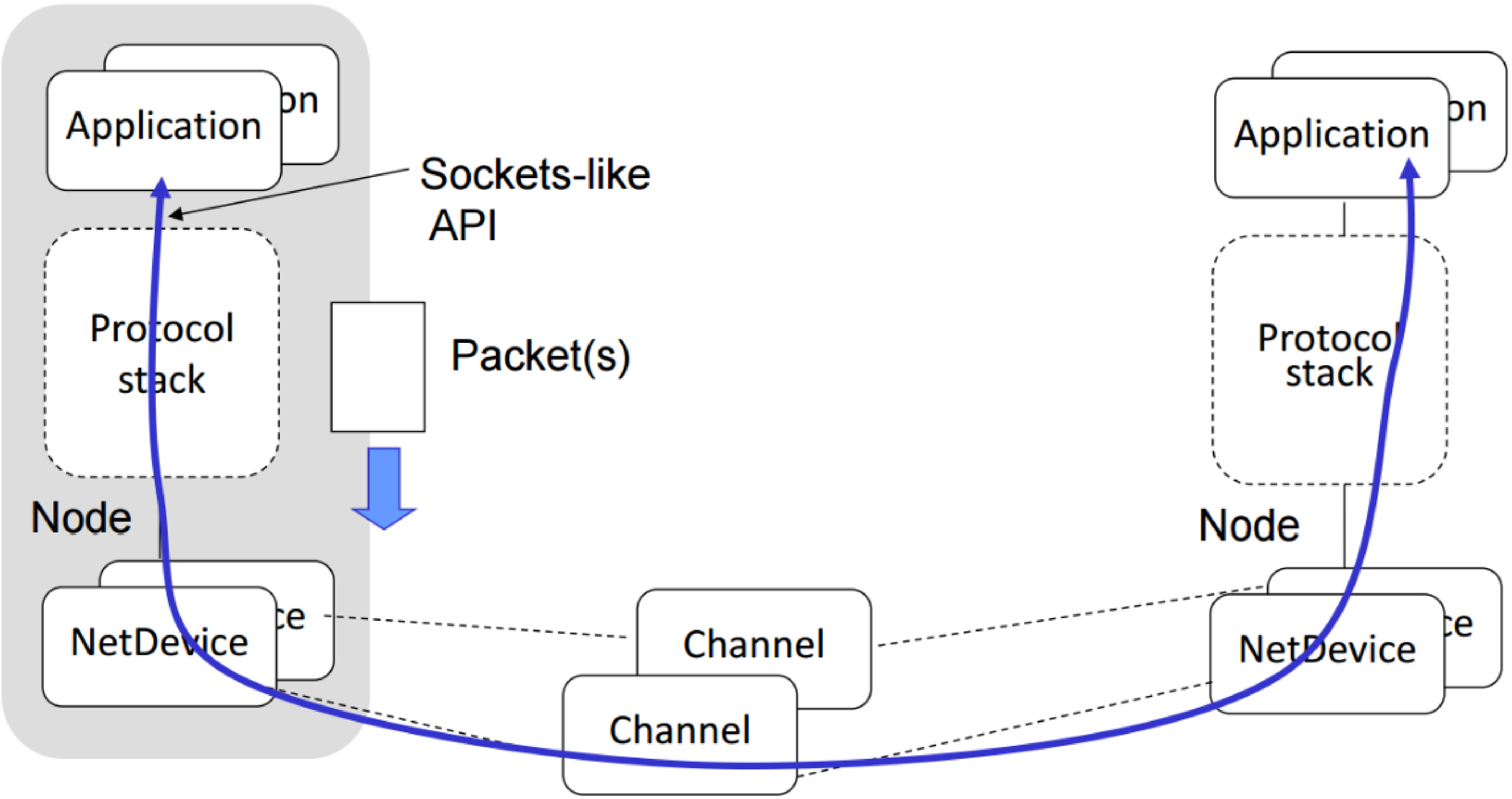
- *NetDevice:* mimics the network interface card (NIC), covers both hardware and software driver.

- *Channel:* models the communication channel. For example, 2 nodes with installed WifiNetDevice can communicate via wireless WifiChannel.

- *Application:* responsible for traffic generation.

- *Packet:* a piece of data, at any level of the protocol stack we can add or remove headers.

Protocol stack

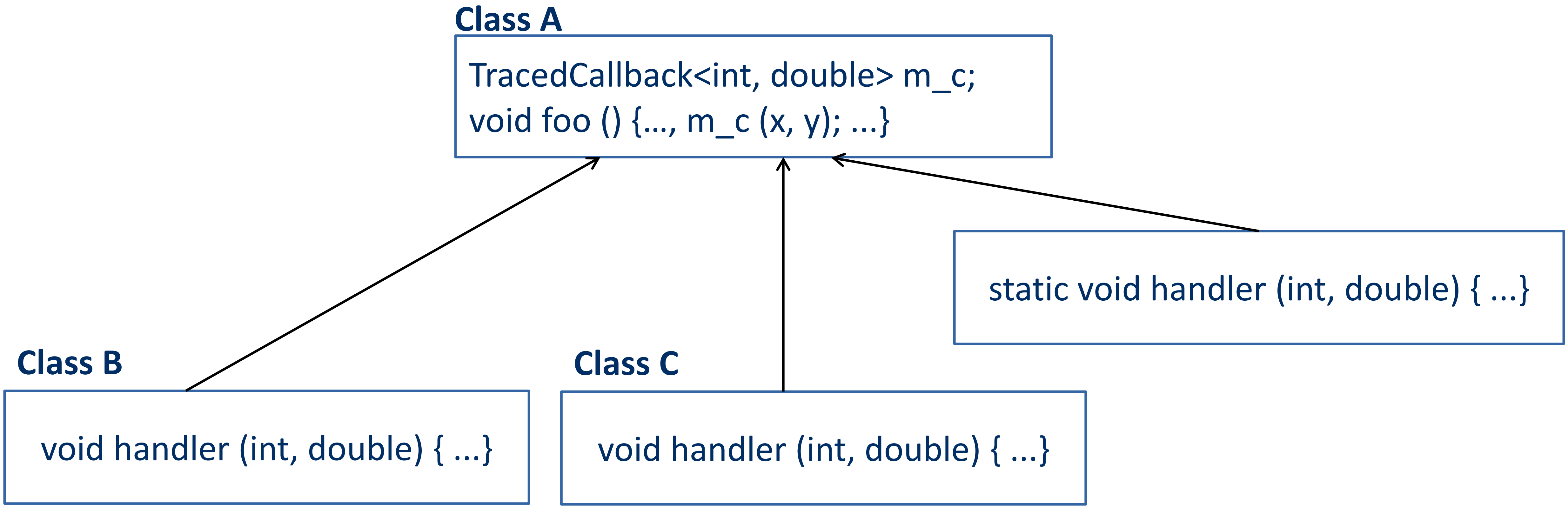


NS-3 modules

- *core*: event queue, model time, random variables, attributes, etc.
- *network*: node, NetDevice, packet, channel, address
- *bridge*: switch, bridge
- *internet*: IP, TCP, UDP, ARP, ICMP, other internet protocols
- *internet-apps:* ping, traceroute, DHCP, radvd
- *mobility*: nodes' positions allocations, mobility models
- *propagation*: signal propagation (attenuation, delays)
- *applications*: various applications
- Various technologies: *wifi*, *lte*, *csma* (Ethernet), etc.

Other modules (e.g., some routing protocols)

Tracing - I




```
Ptr<A> a = CreateObject<A> ();
Ptr<B> b = CreateObject<B> ();
a ->TraceConnectWithoutContext (“MyTrace”, MakeCallback(&handler));
a ->TraceConnectWithoutContext (“MyTrace”, MakeCallback(&B::handler, b));
```

Tracing - II

packet-sink.cc

```
TypeId
PacketSink::GetTypeId (void) {
  static TypeId tid = TypeId ("ns3::PacketSink")
    .SetParent<Application> ()
    .AddConstructor<PacketSink> ()
    ...
    .AddTraceSource ("Rx", "A packet has been received",
              MakeTraceSourceAccessor (&PacketSink::m_rxTrace))
  ;
  return tid;
}

TracedCallback<Ptr<const Packet>, const Address &> m_rxTrace;

void PacketSink::HandleRead (Ptr<Socket> socket) {
  Ptr<Packet> packet;
  while ((packet = socket->RecvFrom (from))) { ... m_rxTrace (packet, from); ...}
```

scenario.cc

```
void
ReceiveTrace (Ptr<const Packet>,
const Address&) {
  RcvPktCount++;
}
```

Tracing - III

scenario.cc

```
void
ReceiveTrace (Ptr<const Packet> pkt, const Address & addr) {
  RcvPktCount++;
}

int main (int argc, char *argv[]) {
  Ptr<PacketSink> myObject = CreateObject<PacketSink> ();
  myObject->TraceConnectWithoutContext ("Rx", MakeCallback(&ReceiveTrace));
}
```

Going through Lab 6