# Developing Simple Simulations with ndnSIM

**NDN Tutorial – ACM ICN 2016**
September 26, 2016, Kyoto, Japan

Alex Afanasyev
University of California, Los Angeles

https://named-data.net/icn2016-tutorial

# Outline

- **Introduction**
  - Current status and usage
  - Internal structure

- Tutorial with Hands On
  - Getting started
  - Prepare environment
  - Writing scenarios
  - More

# Introduction

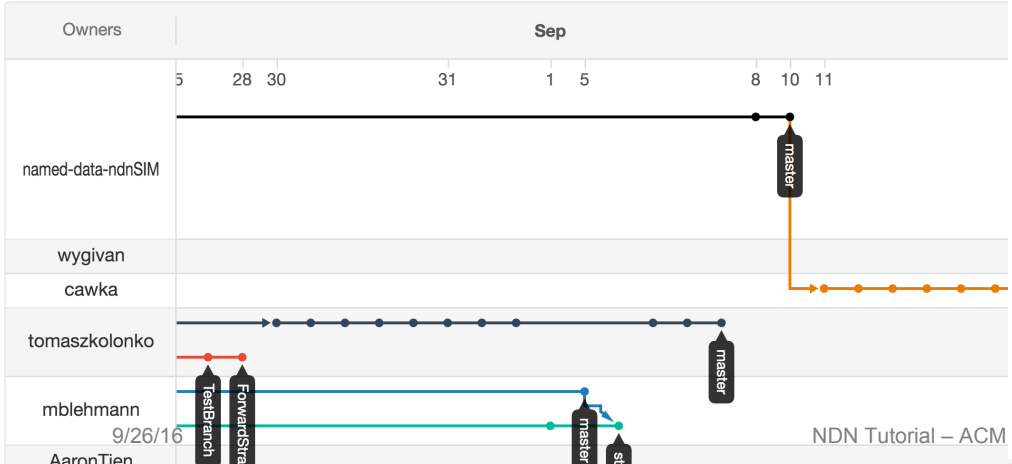
- Based on the NS-3 network simulator framework
  - C++, highly modular, actively maintained

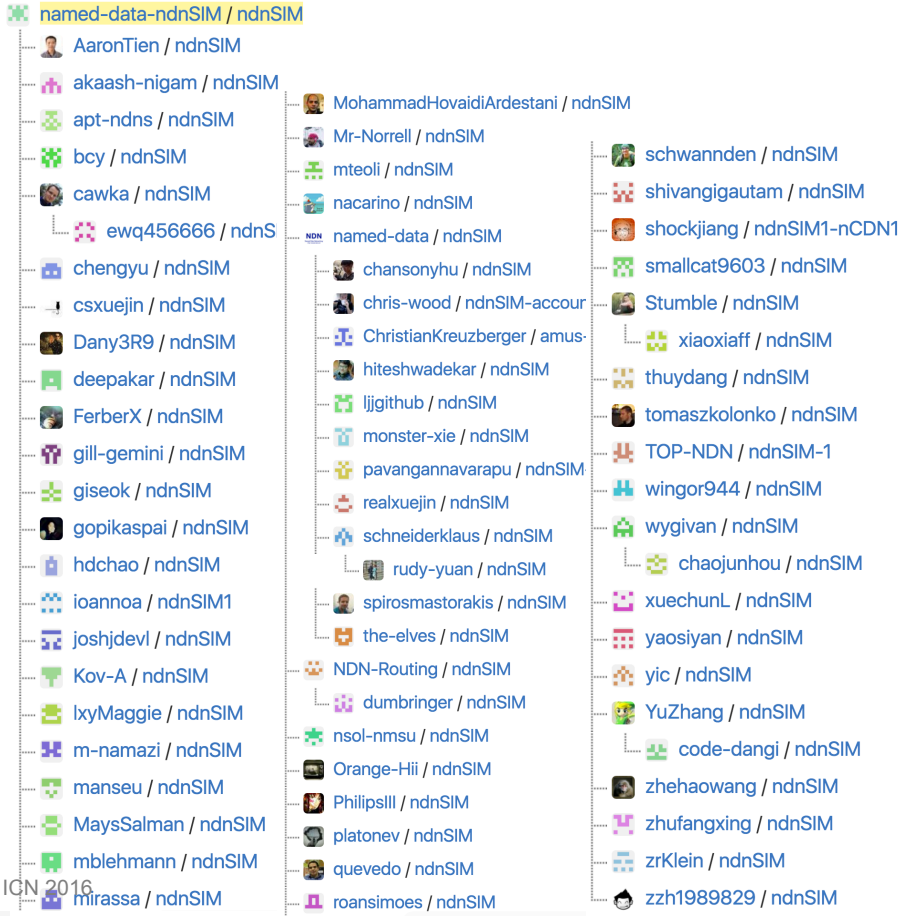
- An open-source common framework to perform NDN-related simulation-based experiments
  - Extensively documented
  - Actively supported

- Matches the latest advancements of NDN research
  - re-uses existing codebases (ndn-cxx, NFD) in ndnSIM
  - allows porting of ndnSIM code into real implementations

# Current Status of ndnSIM 1.x + 2.x

- 70 known forks on GitHub
- Active development
  - Release 2.2 this month
  - Next release soon
- Active mailing list with 400 members
- >280 published papers

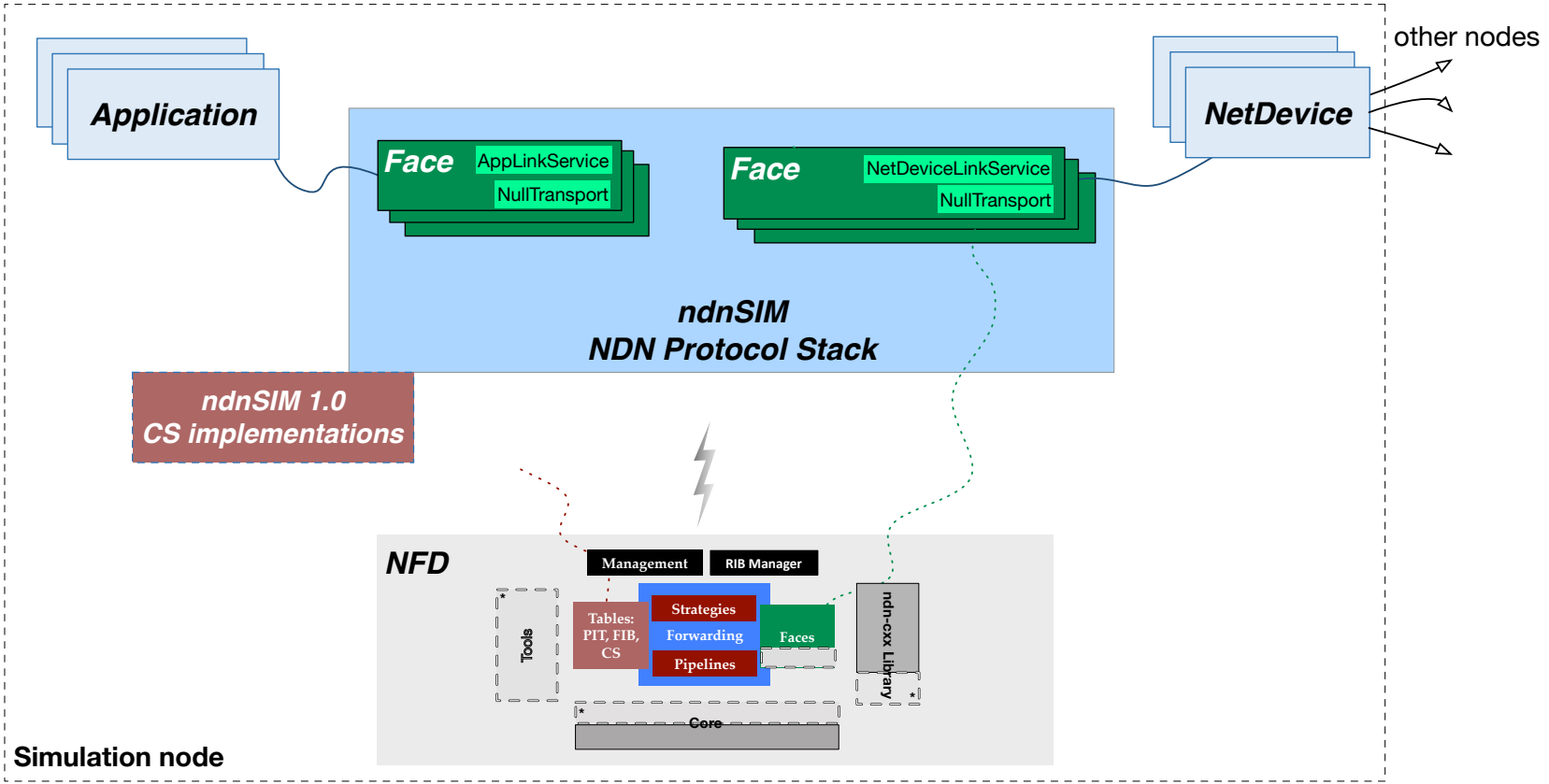


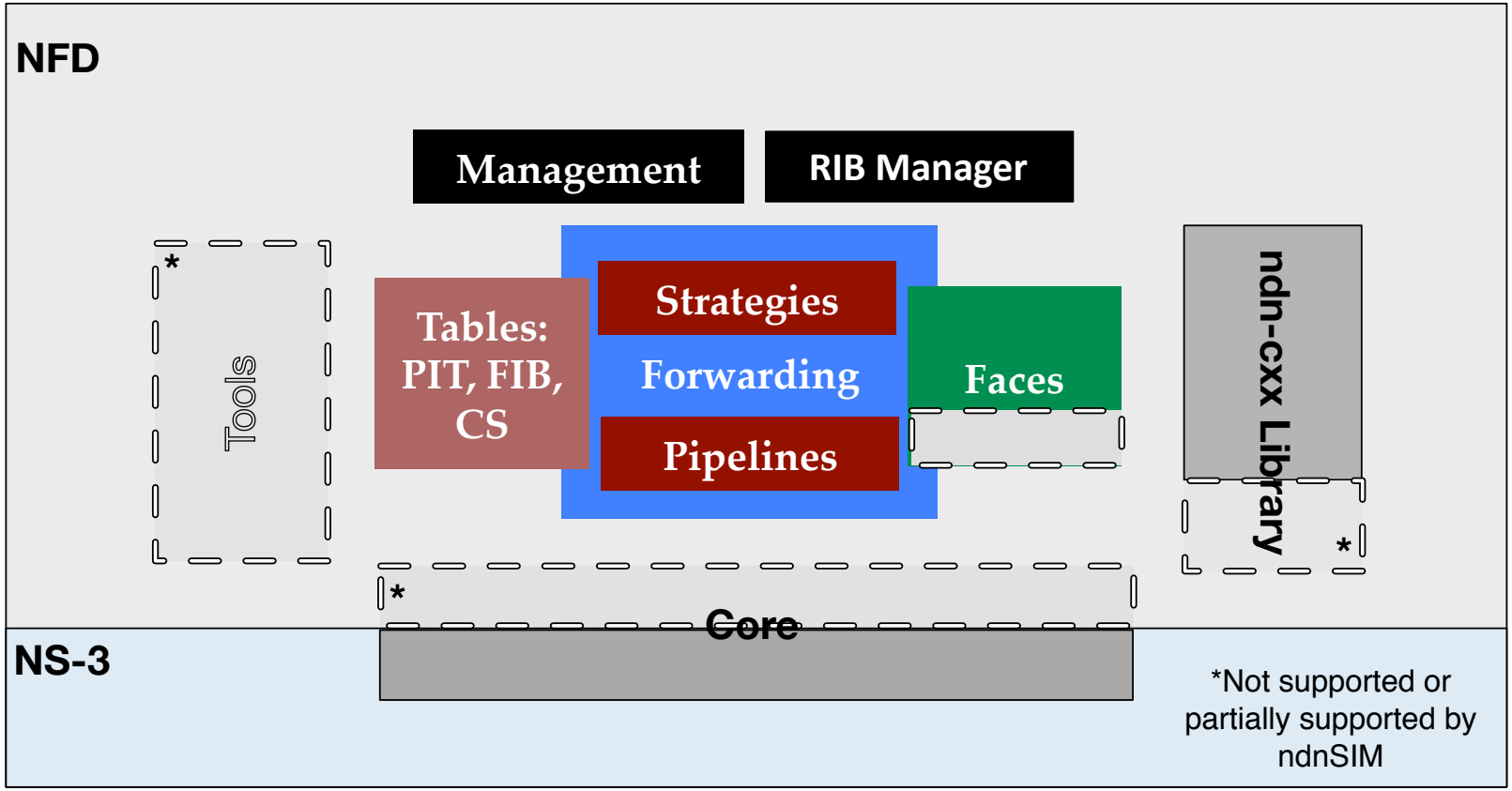
# ndnSIM Can...

- Run large-scale experiments in a simple way
  - Initial topology and link parameters can be defined in a file
  - Stack helpers can adjust parameters of individual or multiple nodes at a time
  - Run emulation-like simulation experiments
    - http://ndnsim.net/guide-to-simulate-real-apps.html
- Collect detailed traces of NDN traffic flow and behavior of each forwarding component

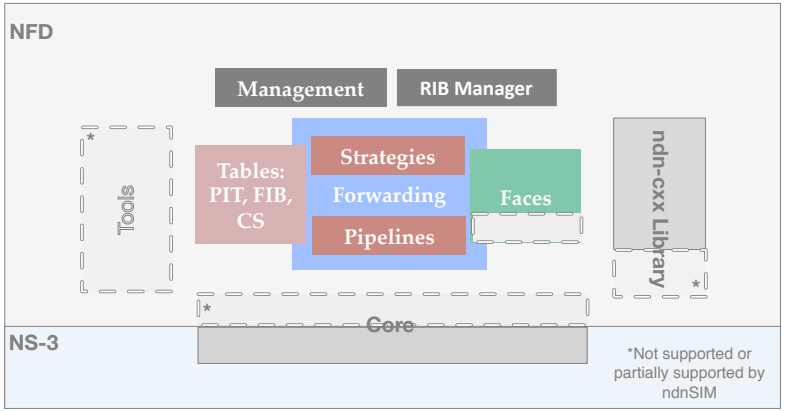
# ndnSIM 2.2 Structure Overview



# ndnSIM-NFD Integration



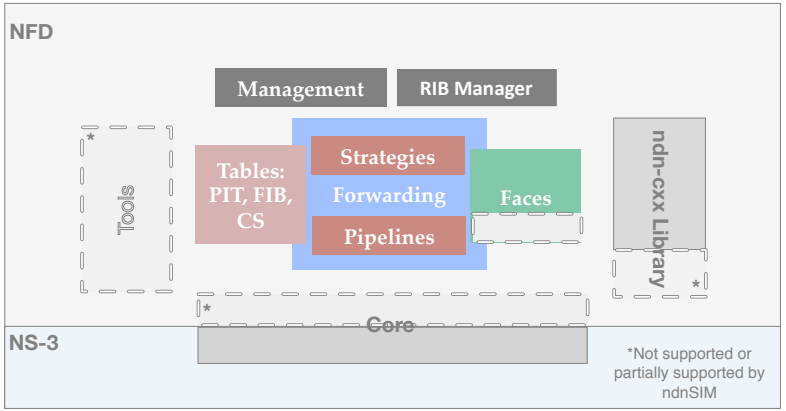
# ndnSIM-NFD Integration: ndn-cxx Features




- **ndn-cxx**
  - NDN and NFD abstractions
  - wire format
  - security library
  - (partially) utils
  - (partially) ndn::Face

Can simulate real applications

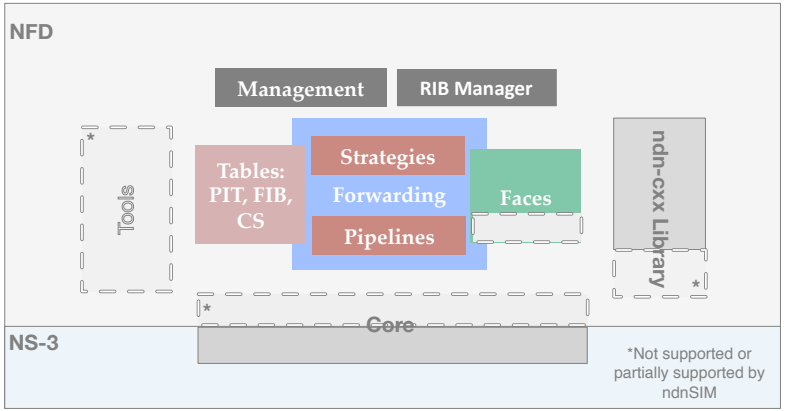
# ndnSIM-NFD Integration: NFD Features




- **NFD**
  - All from Forwarding
    - Built-in forwarding strategies
    - Forwarding pipelines
  - All from Tables
    - PIT, FIB, CS, measurements, strategy choice
  - Management
  - Partial face system
    - native NDN over NetDevices

# ndnSIM-NFD Integration

- **ndnSIM**
  - Faces
    - Face with NetDevice “transport”
    - Face with ndn::App “transport”
    - ndn::Face for real app simulation
  - Tables
    - All available in NFD
    - ndnSIM 1.0 CS implementations
  - Helpers
    - NDN stack helper, Global routing helper, FIB helper, Strategy choice helper, App helper, Link control helper, Scenario Helper, etc.
  - Tracers
    - NDN packet tracer
    - CS tracer
    - App-level tracer



# Additional Documentation

- Technical Report
  - S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, “ndnSIM 2.0: A new version of the NDN simulator for NS-3,” NDN, Technical Report NDN-0028, 2015
    - http://named-data.net/techreport/ndn-0028-1-ndnsim-v2.pdf

- Detailed documentation, with pointers to source code, download instructions, and examples can be found on the ndnSIM website:
  - http://ndnsim.net/

# Outline

- Introduction
  - Current status and usage
  - Internal structure

- **Tutorial with Hands On**
  - Getting started
  - Prepare environment
  - Writing scenarios
  - More

Code from the tutorial can be cloned from GitHub:
https://github.com/cawka/ndnSIM-tutorial

# Getting started

- Works in Linux, OS X, FreeBSD
  - Requires boost libraries >= 1.53
  - Python and various python bindings (for visualizer)
- Download

```
mkdir ndnSIM
cd ndnSIM
git clone git://github.com/named-data-ndnSIM/ns-3-dev-ndnSIM.git ns-3
git clone git://github.com/named-data-ndnSIM/pybindgen.git pybindgen
git clone --recurse git://github.com/named-data-ndnSIM/ndnSIM.git ns-3/src/ndnSIM
```

- Build

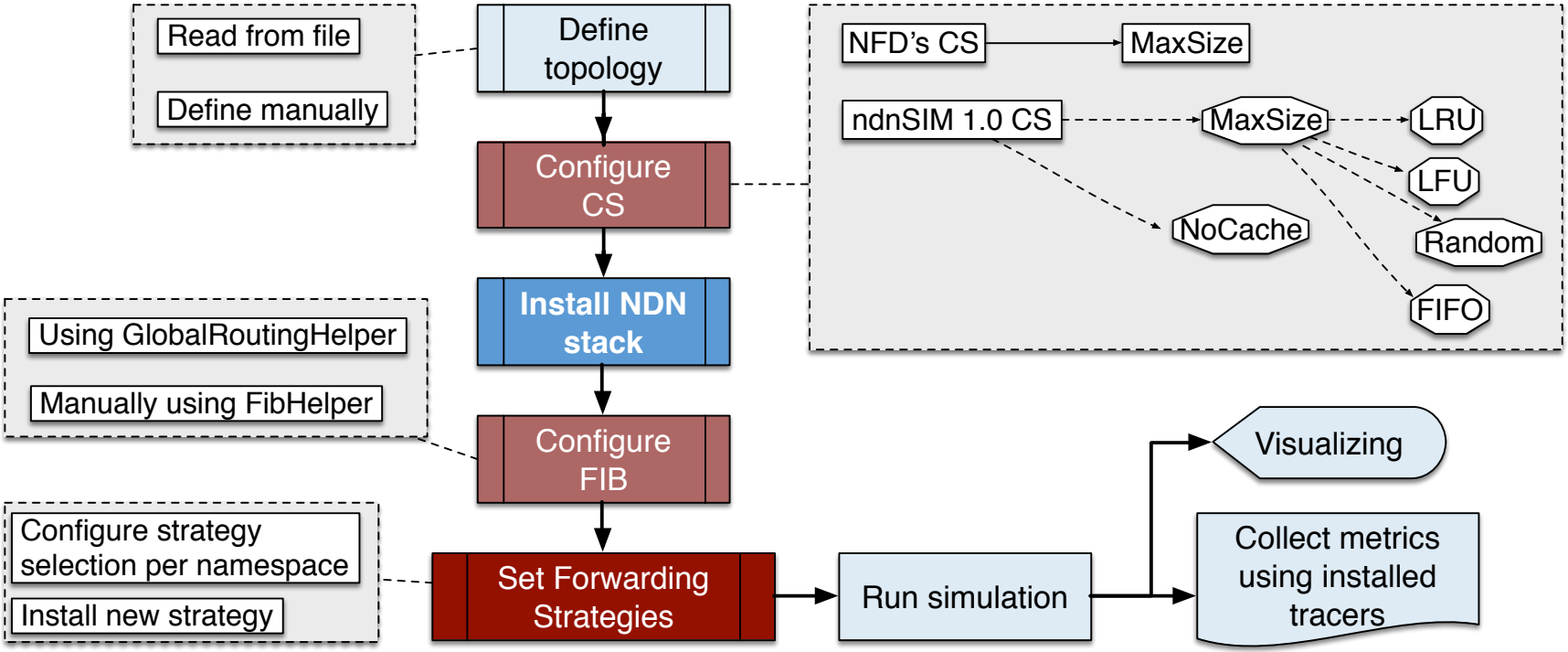
```
./waf configure --enable-examples
./waf
```

- Run examples

```
./waf --run=ndn-grid
./waf --run=ndn-grid --vis
```

http://ndnsim.net/getting-started.html

# Typical Workflow with ndnSIM



# Simulating with ndnSIM: NS-3 Scratch Folder

**NOT recommended**

```
cd ndnSIM/ns-3
vim scratch/my-scenario.cc
# edit
./waf
./waf --run=my-scenario
```

- Cons and pros
  - cons
    - compilation of the scenario can be very slow
    - hard to separate simulation code from the simulator code
  - pros
    - works out-of-the box

# Simulating with ndnSIM: Separate Repository

**RECOMMENDED**

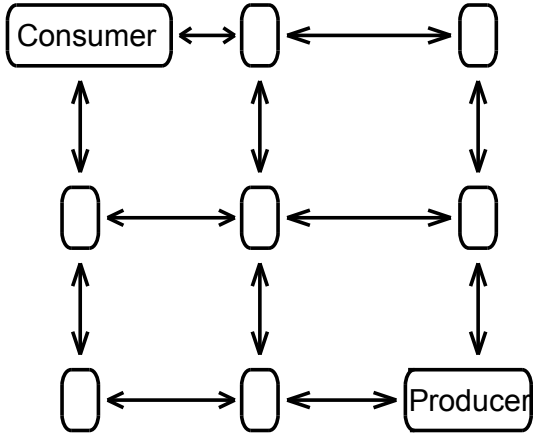
```
cd ndnSIM/ns-3
sudo ./waf install   # install ndnSIM and NS-3

git clone https://github.com/named-data-ndnSIM/scenario-template ../my-scenario
cd ../my-scenario
# create extensions (any .cpp|.hpp files) in extensions/
# create scenarios in scenarios/
vim scenarios/my-scenario.cpp
# edit
./waf configure --debug
./waf --run=my-scenario
```

- Cons and pros
  - cons
    - may need certain configuration tricks (refer to README.md)
  - pros
    - fast compilation
    - clear separation of the simulator code from the extensions and scenarios
    - easy to make code available for others to reproduce scenarios

# Writing a Basic Scenario

- Simple simulation
  - filename
    - scenarios/example1.cc (C++)
    - scenarios/example1.py (Python)
  - Topology
    - 3x3 grid topology
    - 10Mbps links / 10ms delays
    - One consumer, one producer
  - NDN parameters
    - Forwarding Strategy for interests: **BestRoute**
    - FIB is computed automatically using global routing controller
    - Cache: **LRU** with 100 items on each node (default)




10 Mbps / 10 ms delay

# NS-3 101: Simulation Scenario (C++)

**Necessary includes**

```
#include ”ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/point-to-point-grid.h"
#include "ns3/ndnSIM-module.h"
```

**Standard main function**

```
int main(int argc, char* argv[])
{
  // scenario “meat”
  return 0;
}
```

**Set necessary defaults and allow setting different defaults at run time**

```
Config::SetDefault("ns3::PointToPointNetDevice::DataRate", StringValue("10Mbps"));
Config::SetDefault("ns3::PointToPointChannel::Delay", StringValue("10ms"));
Config::SetDefault("ns3::DropTailQueue::MaxPackets", StringValue("20"));
CommandLine cmd; cmd.Parse (argc, argv);
```

**Create topology**

```
PointToPointHelper p2p;
PointToPointGridHelper grid (3, 3, p2p);
grid.BoundingBox(100,100,200,200);
```

Define what to simulate

**“Run” the simulation**

```
Simulator::Stop(Seconds (20.0));
Simulator::Run();
Simulator::Destroy();
```

# ndnSIM 101: Defining the Simulation (C++)

**Install NDN stack**

```
ndn::StackHelper ndnHelper;
ndnHelper.InstallAll();
```

**Install consumer app(s)**

```
NodeContainer consumerNodes;
consumerNodes.Add(grid.GetNode (0,0));
ndn::AppHelper cHelper("ns3::ndn::ConsumerCbr");
cHelper.SetPrefix("/prefix");
cHelper.SetAttribute("Frequency",
                     StringValue("10"));
cHelper.Install(consumerNodes);
```

**Install producer app(s)**

```
Ptr<Node> producer = grid.GetNode(2, 2);
ndn::AppHelper pHelper("ns3::ndn::Producer");
pHelper.SetPrefix("/prefix");
pHelper.SetAttribute("PayloadSize",
                     StringValue("1024"));
pHelper.Install(producer);
```

**Configure FIB (manually or like here using the helper)**

```
ndn::GlobalRoutingHelper ndnGlobalRoutingHelper;
ndnGlobalRoutingHelper.InstallAll();
ndnGlobalRoutingHelper.AddOrigins("/prefix", producer);
ndnGlobalRoutingHelper.CalculateRoutes();
```

# Same using NS-3 Python Bindings

- Pros
  - Whole scenario defined as a simple python script
  - No need to do any compilation
- Cons
  - Limited set of features of NS-3 and ndnSIM

# NS-3 101: Simulation Scenario (Python)

**Necessary includes**

```
from ns.core import *
from ns.network import *
from ns.point_to_point import *
from ns.point_to_point_layout import *
from ns.ndnSIM import *
```

**Rest of the Python script**

```
# scenario “meat”
```

**Set necessary defaults and allow setting different defaults at run time**

```
Config.SetDefault ("ns3::PointToPointNetDevice::DataRate", StringValue ("10Mbps"))
Config.SetDefault ("ns3::PointToPointChannel::Delay", StringValue ("10ms"))
Config.SetDefault ("ns3::DropTailQueue::MaxPackets", StringValue ("20"))
import sys; cmd = CommandLine(); cmd.Parse(sys.argv);
```

**Create topology**

```
p2p = PointToPointHelper()
grid = PointToPointGridHelper(3,3,p2p)
grid.BoundingBox(100,100,200,200)
```

Define what to simulate

**“Run” the simulation**

```
Simulator.Stop(Seconds (20.0))
Simulator.Run()
Simulator.Destroy()
```

# ndnSIM 101: Defining the Simulation (Python)

**Install NDN stack**

```
ndnHelper = ndn.StackHelper()
ndnHelper.InstallAll();
```

**Install consumer app(s)**

```
consumerNodes = NodeContainer()
consumerNodes.Add(grid.GetNode (0,0))
cHelper = ndn.AppHelper("ns3::ndn::ConsumerCbr”)
cHelper.SetPrefix ("/prefix”)
cHelper.SetAttribute("Frequency",
                     StringValue ("10"))
cHelper.Install(consumerNodes)
```

**Install producer app(s)**

```
producer = grid.GetNode(2, 2)
pHelper = ndn.AppHelper("ns3::ndn::Producer”)
pHelper.SetPrefix("/prefix”)
pHelper.SetAttribute("PayloadSize",
                          StringValue("1024"));
pHelper.Install(producer)
```

**Configure FIB (manually or like here using the helper)**

```
ndnGlobalRoutingHelper = ndn.GlobalRoutingHelper()
ndnGlobalRoutingHelper.InstallAll()
ndnGlobalRoutingHelper.AddOrigins(“/prefix”, producer)
ndnGlobalRoutingHelper.CalculateRoutes()
```

# Running the simulation (C++)

- Run C++ scenario

```
./waf --run example1
# or ./waf && ./build/example1
# or ./waf --run example1 --vis
```

- Run Python scenario

```
python scenarios/example1.py
```

- If in debug mode

```
NS_LOG=ndn.Consumer ./waf --run example1
# or NS_LOG=ndn.Consumer python scenarios/example1.py
```

Result if you followed the steps

Hint: using right click on a node in visualizer, it is possible to check FIB, PIT, and CS contents on the node during the active simulation

Same example is on http://ndnsim.net

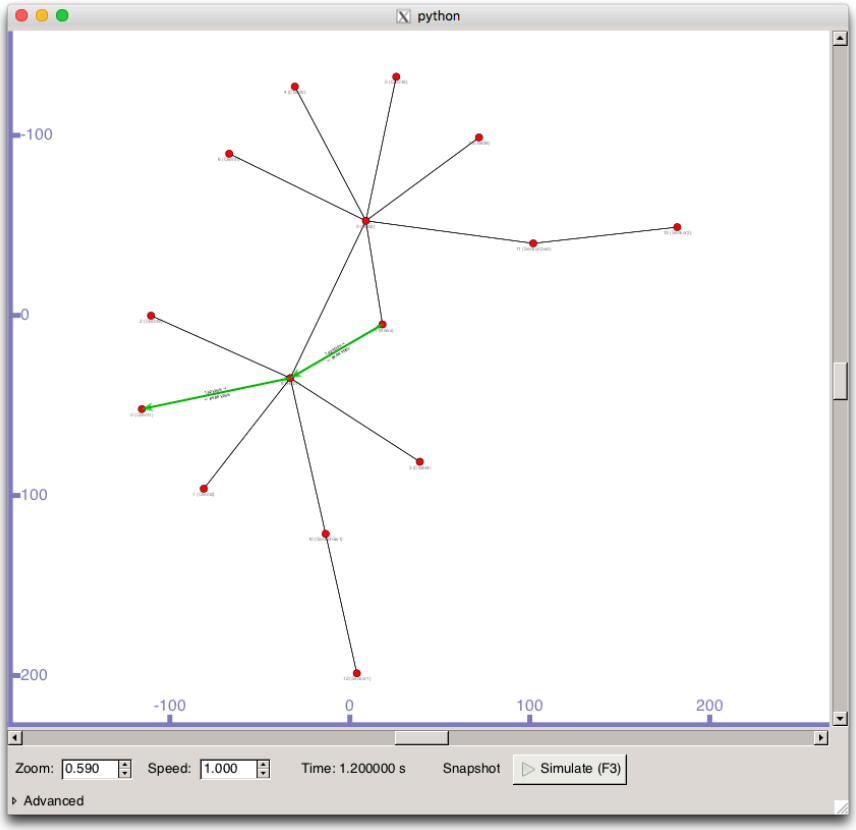
# Writing a Custom App

- Two approaches
  - Use ns3::ndn::Producer/ns3::ndn::Consumer based implementation
    - Very simple and limited producer/consumers with a set of assumptions
    - Useful when investigating forwarding plane behavior (new strategy, new cache policy, etc.)
  - Use ndn::Face
    - Can write full-featured application (or use the existing one)

- Simple Producer
  - Let's use the first approach

- Simple Consumer
  - Let's use the second approach

# Let's Code


python
-100
0
100
200
-100
0
100
200
Zoom: 0.590
Speed: 1.000
Time: 1.200000 s
Snapshot
Simulate (F3)
Advanced

# Hints for Writing/Running Simulations

- Logging
- Getting metrics
- Processing metrics
- Customizing
  - Strategy
  - Content store policy
- Writing your own (not covered today)
  - Strategy
  - Content store policy

# Logging in debug mode (cont.)

- Selecting several several loggings

```
NS_LOG=ndn.fw:ndn.fw.BestRoute:ndn.Consumer ./waf --run=example1
```

- Select all loggings (including from the NS-3)

```
NS_LOG=* ./waf --run=example1
```

- **DO NOT USE LOGGING TO GET METRICS**
  - Use existing tracing helpers or write your own

# Getting Metrics

* For now, supported only in C++

- L3RateTracer

```
L3RateTracer::InstallAll("rate-trace.txt", Seconds(1.0));
```

- AppDelayTracer

```
AppDelayTracer::InstallAll("app-delays-trace.txt");
```

- CsTracer

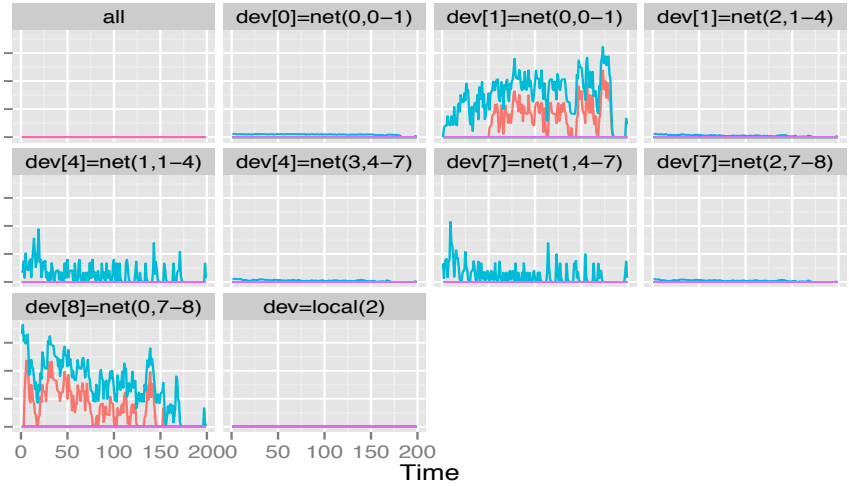
```
CsTracer::InstallAll("cs-trace.txt", Seconds(1));
```

- Write your own tracer (see existing implementation for hints)

http://ndnsim.net/metric.html

# Processing metrics

- Resulting .txt files can be processed
  - R
  - gnuplot
  - python graph library and others
- Example with R
  - Same scenario, but with small modifications
    - Config::SetDefault ("ns3::PointToPointNetDevice::DataRate", StringValue ("**20Kbps**"));
    - ndn::AppHelper cHelper ("ns3::ndn::**ConsumerZipfMandelbrot**");
  - very basic rate-trace.txt procesing
    - library (ggplot2)
    - data = read.table ("results/rate-trace.txt", header=T)
    - ggplot(data, aes(x=Time, y=Kilobytes, color=Type)) + geom_line () + facet_wrap(~ FaceDescr)



# Customizing Forwarding Strategy

```
StrategyChoiceHelper::Install(nodes, prefix, strategyName);
// or StrategyChoiceHelper::InstallAll(prefix, strategyName);
```

- Available strategies and defaults

| Namespace | Strategy Class | Strategy Name |
|---|---|---|
| / | fw::BestRouteStrategy | /localhost/nfd/strategy/best-route |
| /localhost | fw::MulticastStrategy | /localhost/nfd/strategy/multicast |

# Customizing Content Store Policy

```
ndn::StackHelper ndnHelper;
ndnHelper.SetContentStore (“ns3::ndn::cs::Lru”, “MaxSize”, “100”);
```

- Available content stores
  - ns3::ndn::cs::Lru
  - ns3::ndn::cs::Random
  - ns3::ndn::cs::Fifo
  - ns3::ndn::cs::Lfu
  - ns3::ndn::cs::Nocache

  - ns3::ndn::cs::Lru::Freshness
  - ns3::ndn::cs::Random::Freshness
  - ns3::ndn::cs::Fifo::Freshness
  - ns3::ndn::cs::Lfu::Freshness

  - ns3::ndn::cs::Lru::LifetimeStats
  - ns3::ndn::cs::Random::LifetimeStats
  - ns3::ndn::cs::Fifo::LifetimeStats
  - ns3::ndn::cs::Lfu::LifetimeStats

* By default, a FIFO policy (new-style NFD)

# Writing a custom forwarding strategy

- Tutorial
  - http://ndnsim.net/fw.html#writing-your-own-custom-strategy
- Example
  - http://ndnsim.net/fw.html#example
    - https://github.com/named-data-ndnSIM/ndnSIM/blob/master/examples/ndn-load-balancer.cpp
    - https://github.com/named-data-ndnSIM/ndnSIM/blob/master/examples/ndn-load-balancer/random-load-balancer-strategy.hpp
    - https://github.com/named-data-ndnSIM/ndnSIM/blob/master/examples/ndn-load-balancer/random-load-balancer-strategy.cpp

# Write Your Own Cache Policy (ndnSIM 1.x Style)

- Option A:
  - create a class derived from ndn::ContentStore, implementing all interface functions
  - example
    - ndn::cs::NoCache
- Option B:
  - use C++ templates of ndnSIM
    - define “policy traits” (example utils/trie/lru-policy)
      - defines what to do
        - on insert (e.g., put in front)
        - on update (e.g., promote to front)
        - on delete (e.g., remove)
        - on lookup (e.g., promote to front)
    - instantiate cache class with new policy:
      - template class ContentStoreImpl<lru_policy_traits>;
    - see examples in model/cs/content-store-impl.cc

# Writing a custom cache policy

- ExamplePolicy
  - only every other data packet will be cached
  - “promote” Data packet if it is accessed more that twice
  - apply LRU cache replacement strategy

- Write “policy traits”
  - `extensions/example-policy.hpp`
  - use one of the existing policy traits as a base
    - `utils/trie/lru-policy.hpp`

- “Instantiate” content store with the policy
  - create `extensions/cs-with-example-policy.cpp`

# Feedback

- Try out ndnSIM and let us know your thought/comments/bug reports/new feature requests!
- Join our mailing list
  - http://www.lists.cs.ucla.edu/mailman/listinfo/ndnsim
- Contribute
  - issues on Redmine
    - http://redmine.named-data.net/projects/ndnsim/issues
  - submit code reviews to Gerrit
    - http://gerrit.named-data.net/

**http://ndnsim.net**

# Thanks

- Questions?

**http://ndnsim.net**