Network Simulator (NS-2)

Internet Technologies

60-375

Agenda

- Overview of NS-2
- History
- Current Status
- Platforms Supported
- Discrete Event Simulator
- NS-2 Environment
- NS-2 Hierarchy
- NS-2 Architecture
  - Node Architecture
  - Multicast Node Architecture
  - Packet Structure
  - Links
  - Traffic Flow
- Example

Overview of NS-2

- Discrete Event Simulator
- Packet level
- Modeling Network protocols
  - ✓ Collection of Various protocols at multiple layers
    - ✓ TCP(reno, tahoe, vegas, sack)
    - ✓ MAC(802.11, 802.3, TDMA)
    - ✓ Ad-hoc Routing (DSDV, DSR, AODV, TORA)
    - ✓ Sensor Network (diffusion, gaf)
    - ✓ Multicast protocols, Satellite protocols, and many others

Overview of NS-2

- Maintained through VINT project
- NS2 :collaborative simulation environment
  - ✓ Freely distributed and open source
  - ✓ Supports NT research and education
    - ✓ Protocol design , traffic analysis etc.
  - ✓ Provides common reference

History

- 1995 : Developed by LBL through support of DARPA
- 1996: NS was extended and distributed by VINT project
- 1997: Satellite models added @ UCB
- 1999: Wireless models added @ CMU
- Recent incorporation of emulation

Current status

- Releases:
  - ✓ Periodic releases (currently 2.27, Jan 2004)
  - ✓ Daily snapshots (probably compiles and works, but “unstable”)
  - ✓ Available from: USC/ISI, UC Berkeley, UK mirror
  - ✓ More than 10k users from hundreds of univs

Platforms supported

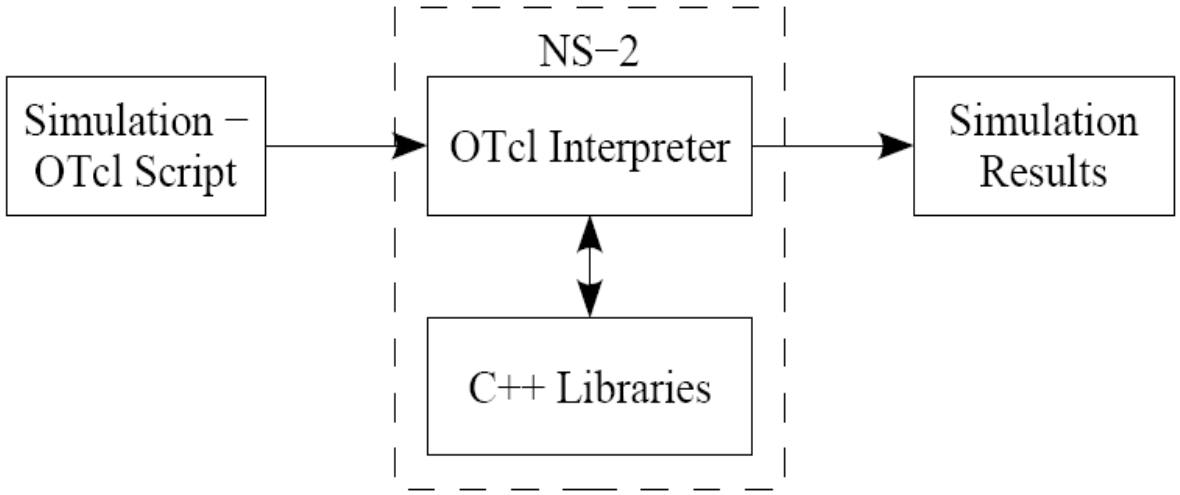
- Most UNIX and UNIX-like systems
  - ✓ FreeBSD
  - ✓ Linux
  - ✓ Solaris
- Windows 98/2000/2003/XP
  - Cygwin required
  - Some work , some doesnt

NS-2 : Components

- NS – Simulator
- NAM – Network AniMator
  - visual demonstration of NS output
- Preprocessing
  - Handwritten TCL or
  - Topology generator
- Post analysis
  - Trace analysis using Perl/TCL/AWK/MATLAB

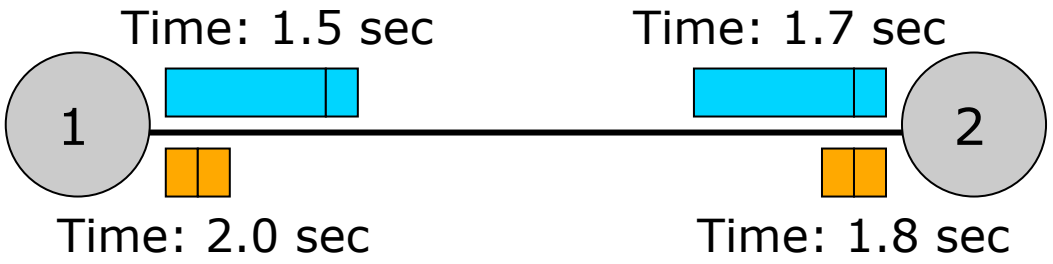
User's Perspective

- From the user's perspective, NS−2 is an OTcl interpreter that takes an OTcl script as input and produces a trace file as output.

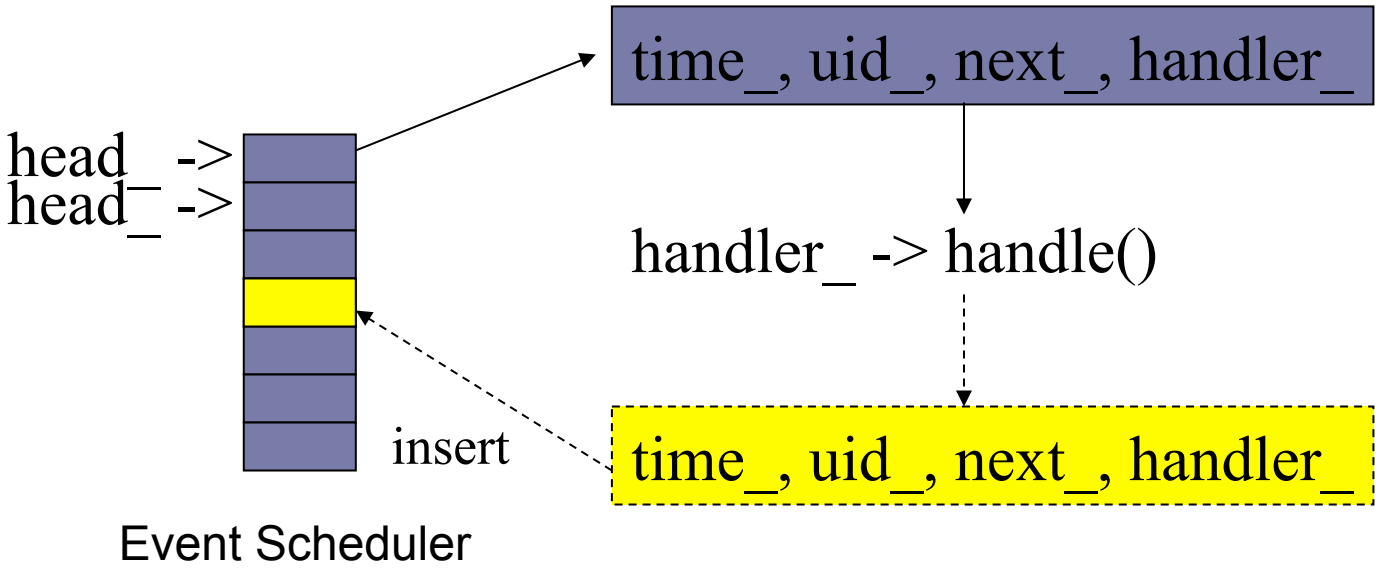


Discrete event simulator

- ns-2 is an discrete event driven simulation
  - Physical activities are translated to events
  - Events are queued and processed in the order of their scheduled occurrences
  - Time progresses as the events are processed



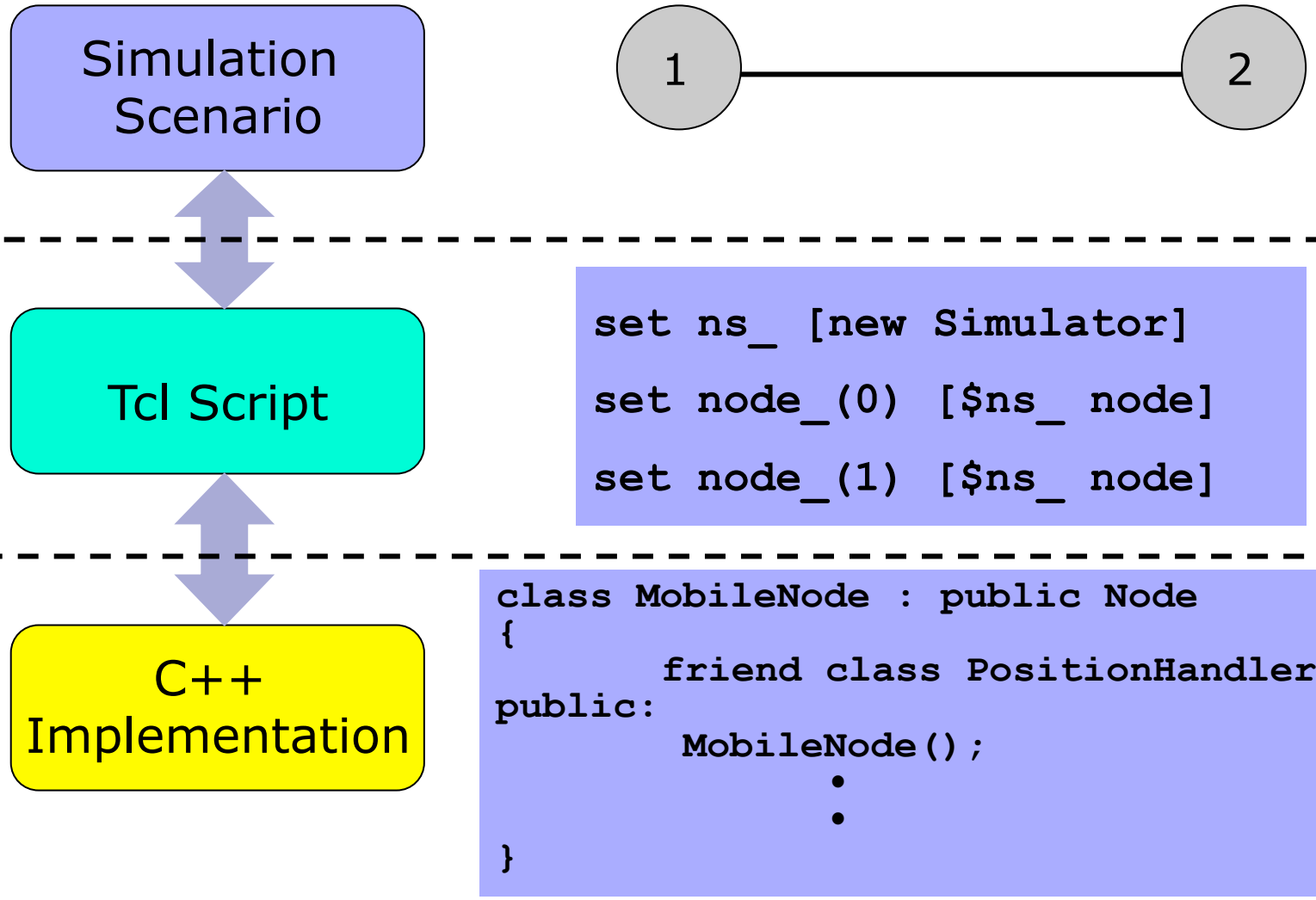
Discrete Event Scheduler



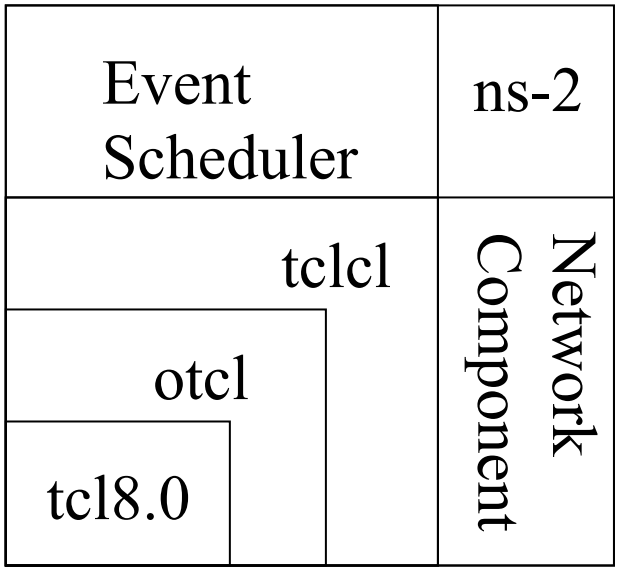
Event Scheduler

- Non-Real time schedulers
  - Implementations : List , Heap , Calender
  - Calender is default
- Real time schedulers
  - Used for emulation for direct interaction with real NT.
- Basic use of an event scheduler:
  - schedule simulation events, such as when to start an FTP application, when to finish a simulation, or for simulation scenario generation prior to a simulation run.

NS-2 Environment

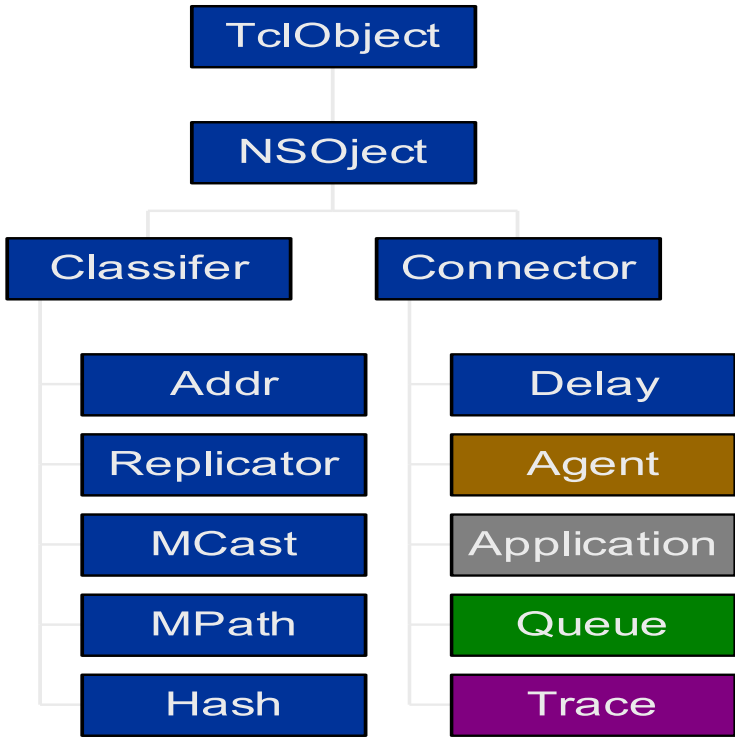


tcl Interpreter With Extents

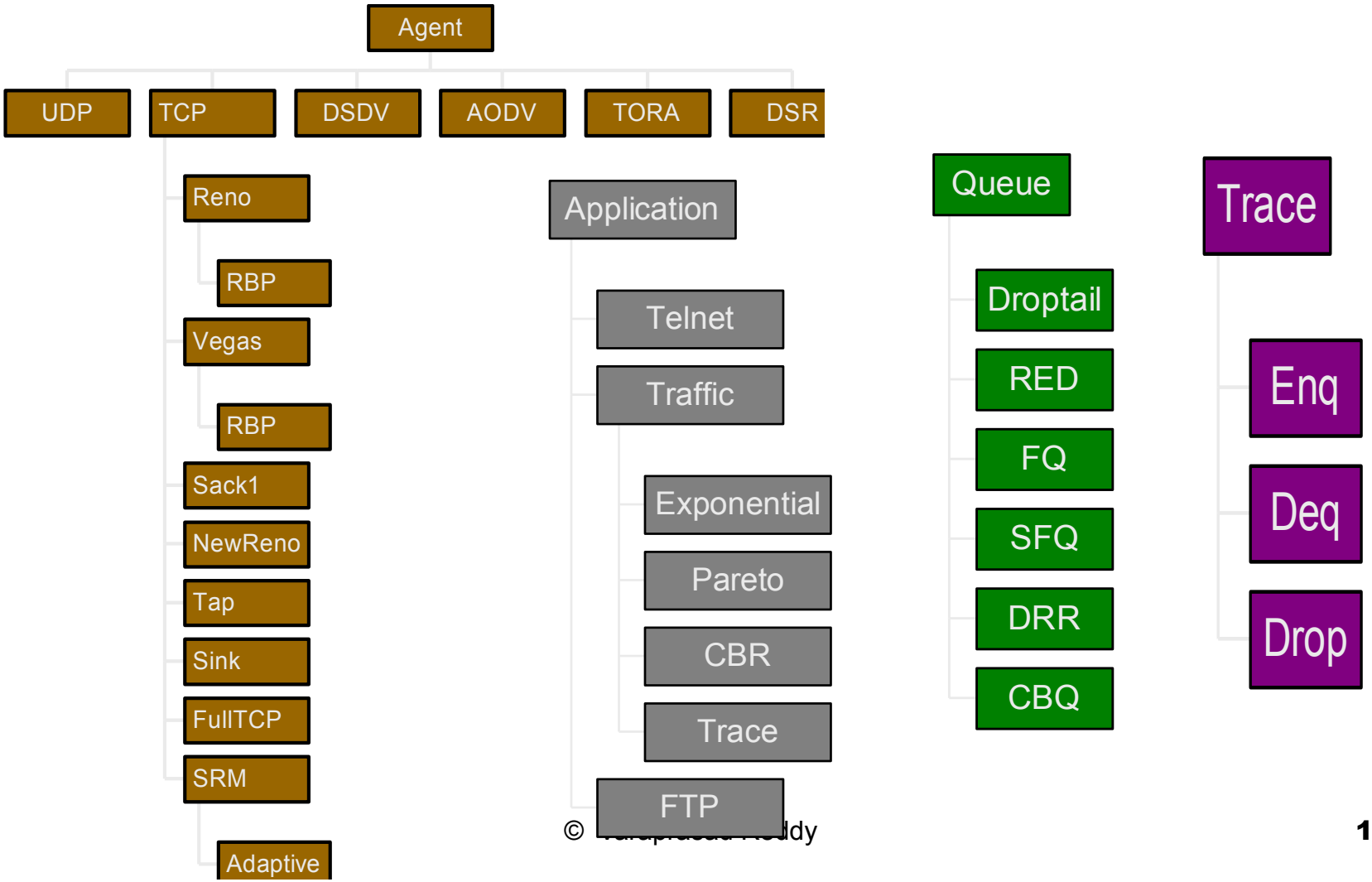


- otcl: Object-oriented support
- tclcl: C++ and otcl linkage
- Discrete event scheduler
- Data network (the Internet) components

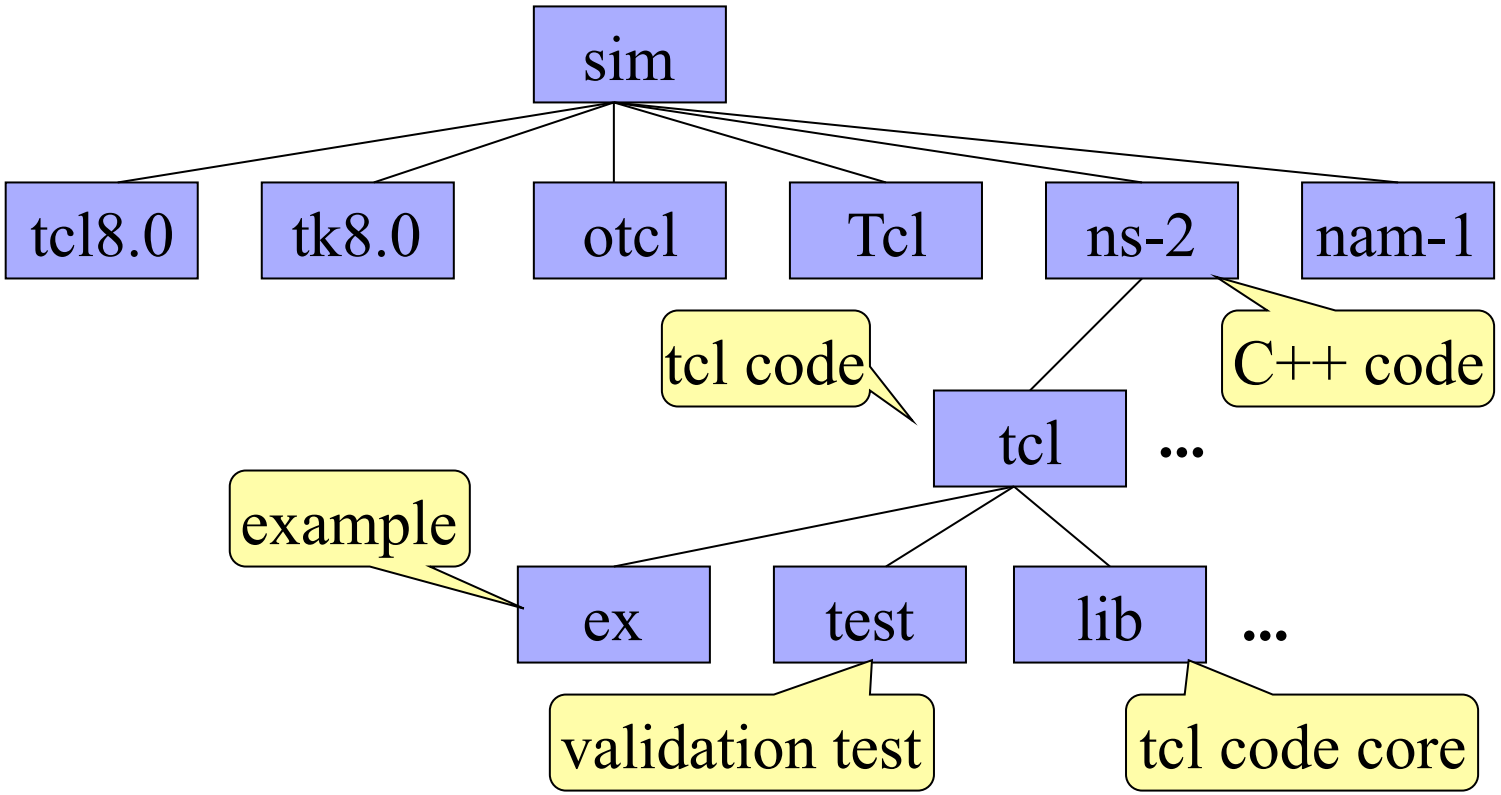
NS-2 Hierarchy



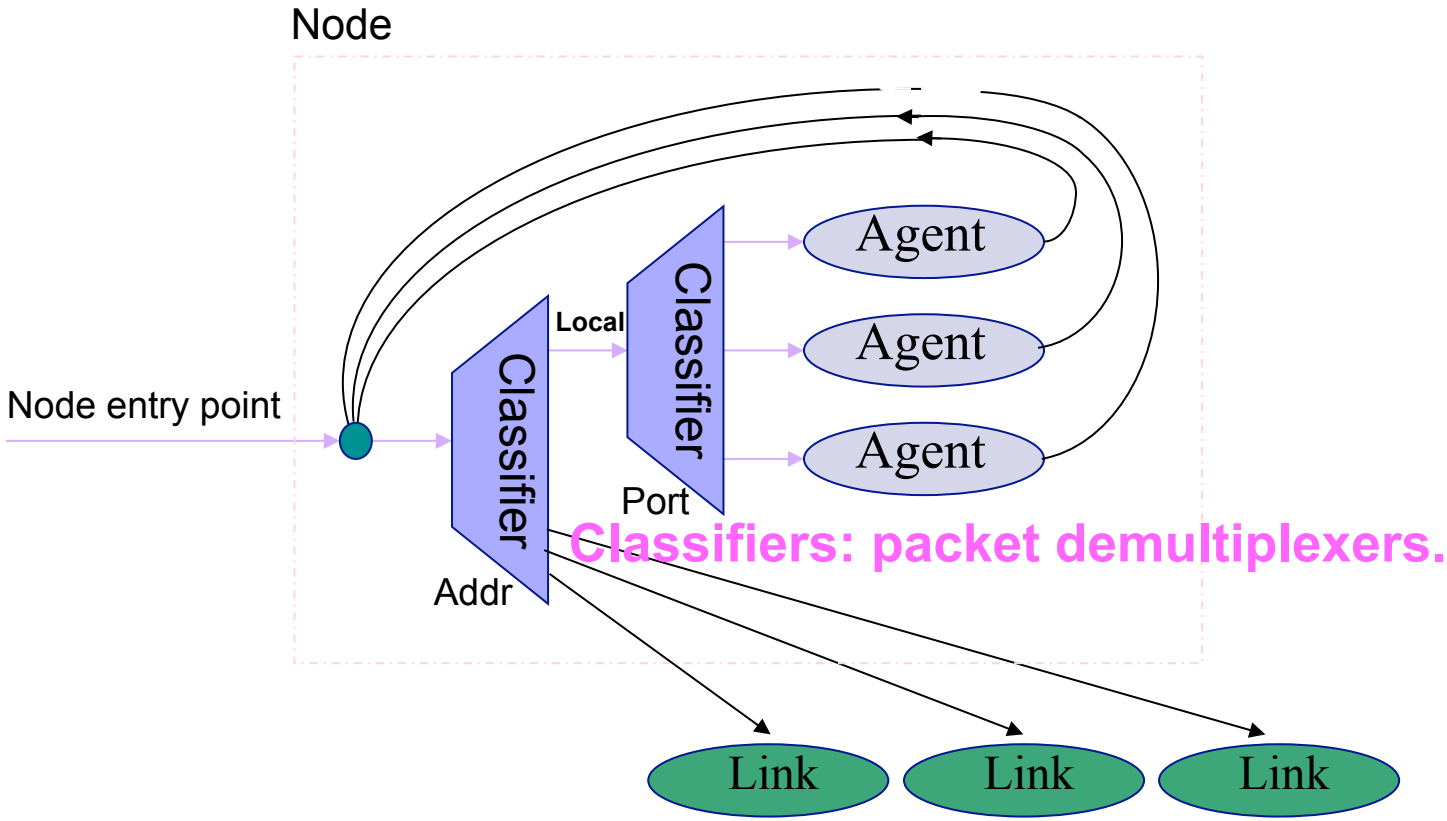
NS-2 Hierarchy



NS-2 Directory Structure



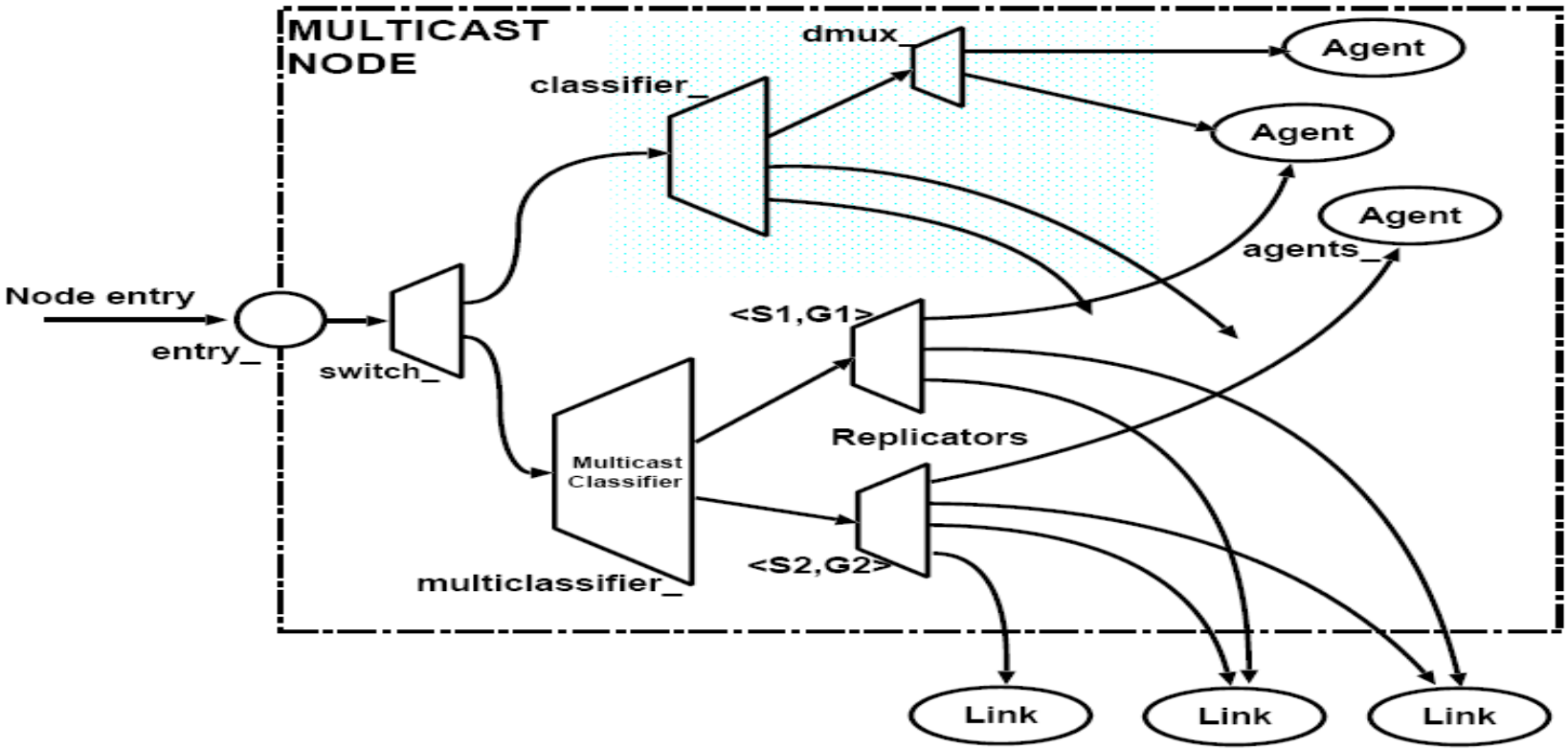
Node Architecture



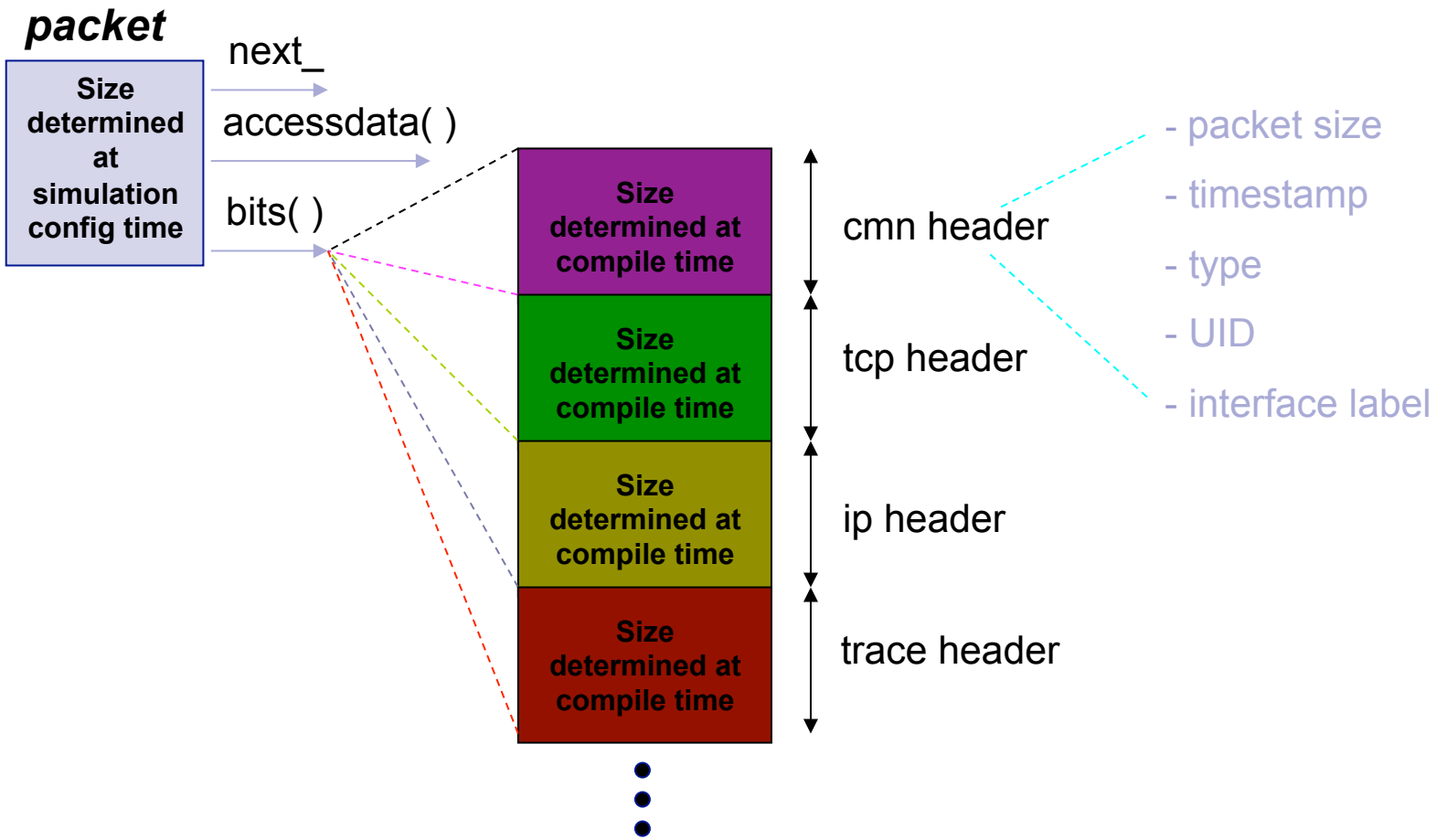
Classifiers: packet demultiplexers.

Agents are either protocol endpoints or related objects that generate/fill-in packet fields.

Multicast Node architecture

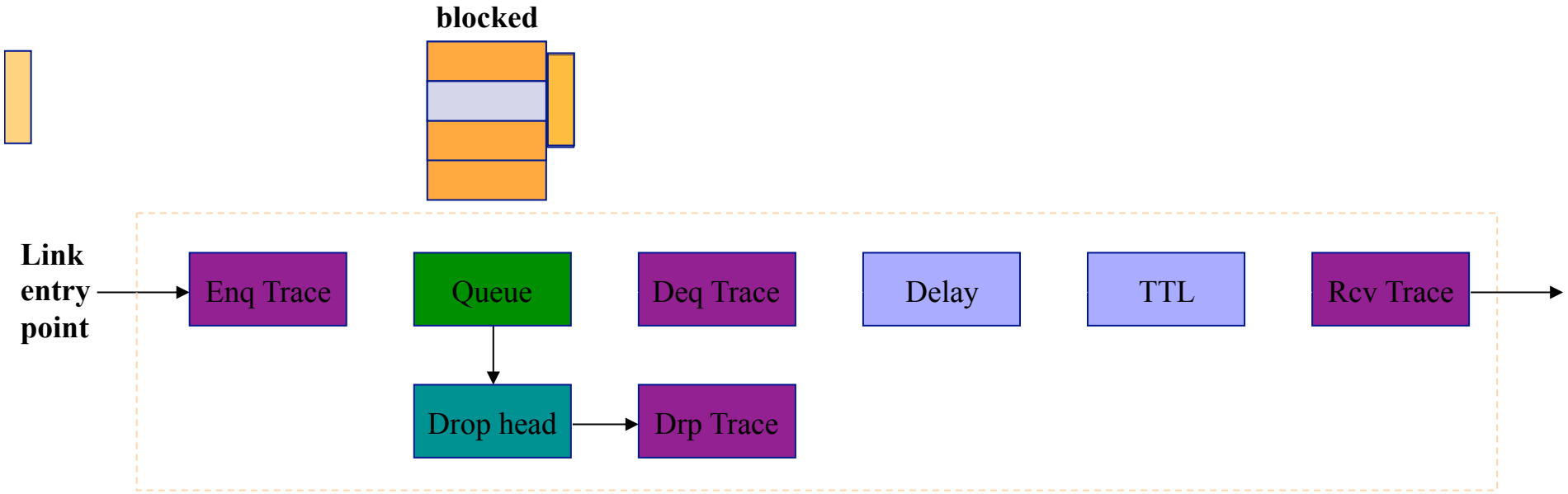


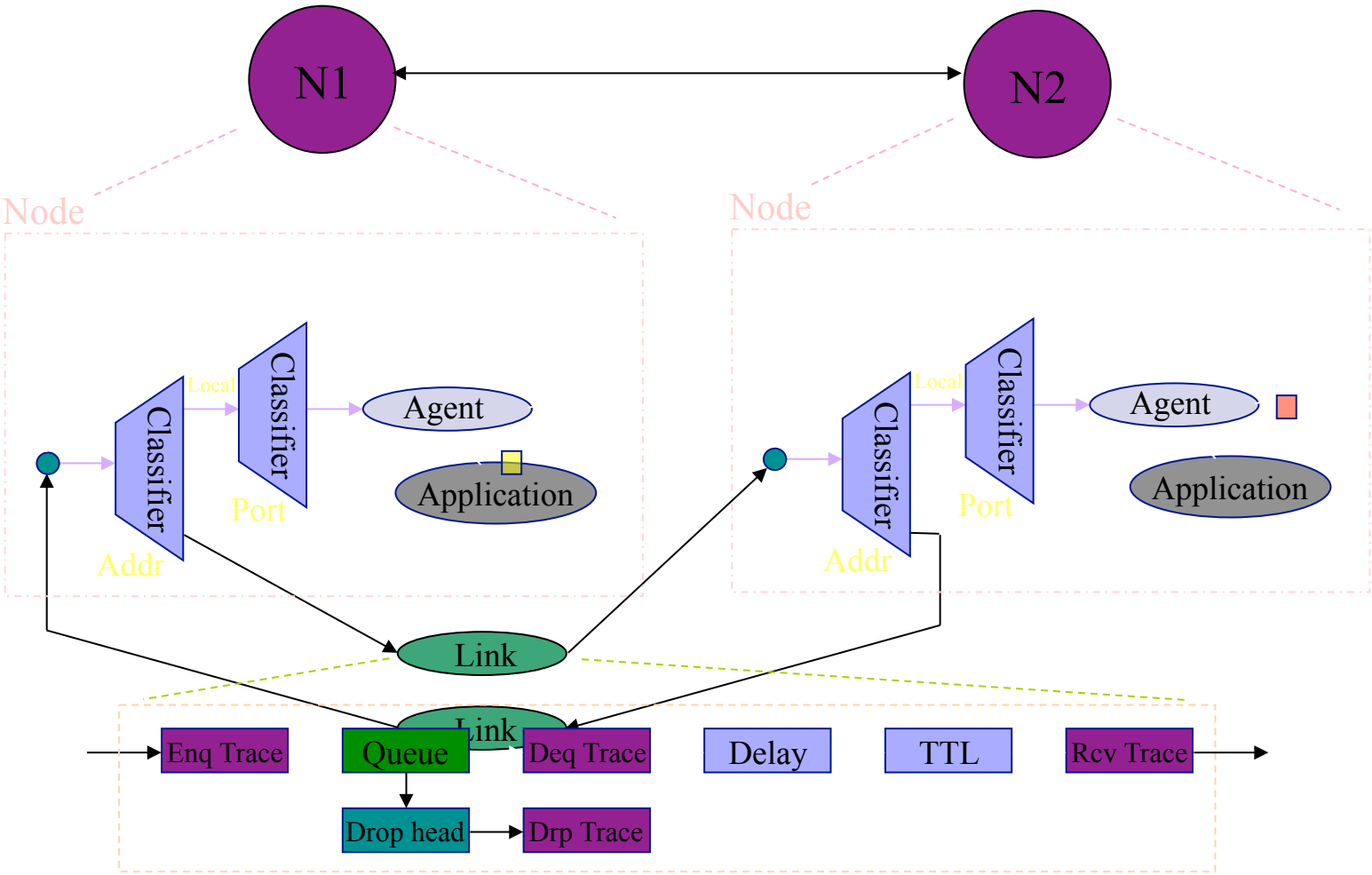
Packet Structure



Links

Links: keeps track of “from” and “to” *node* objects.




N1
N2
Node
Node
Classifier
Local
Classifier
Agent
Application
Port
Addr
Classifier
Local
Classifier
Agent
Application
Port
Addr
Link
Link
Enq Trace
Queue
Deq Trace
Delay
TTL
Rcv Trace
Drop head
Drp Trace

NS-2 : C++ / OTCL

- NS-2 Code contains two sets of languages, namely C++ and OTcl.
- C++ is used for the creation of objects because of speed and efficiency.
- OTcl is used as a front-end to setup the simulator, configure objects and schedule events because of its ease of use.

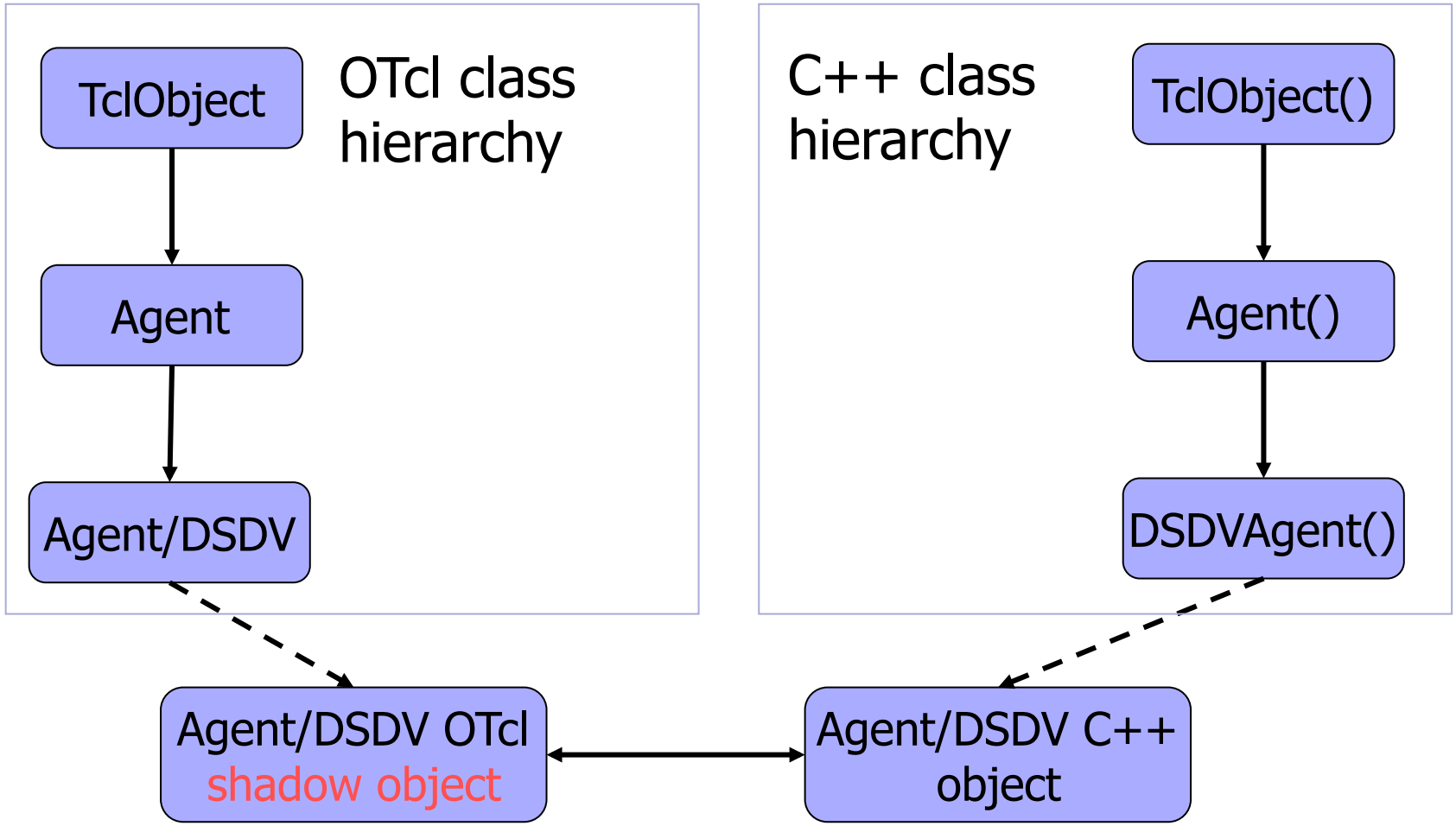
Why two languages? (Tcl & C++)

- C++: Detailed protocol simulations require systems programming language
  - byte manipulation, packet processing, algorithm implementation
  - Run time speed is important
  - Turn around time (run simulation, find bug, fix bug, recompile, re-run) is slower
- Tcl: Simulation of slightly varying parameters or configurations
  - quickly exploring a number of scenarios
  - iteration time (change the model and re-run) is more important

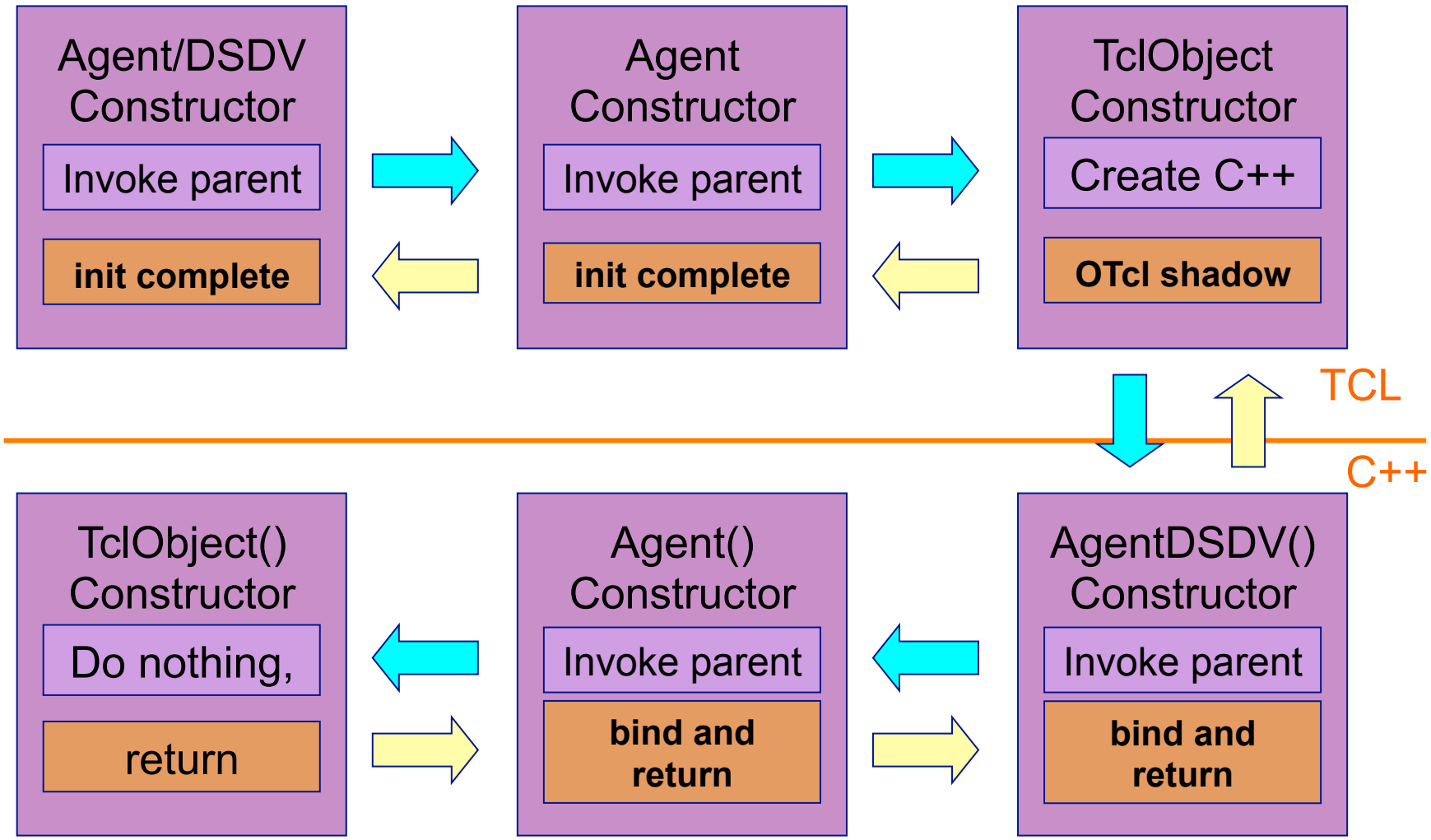
Tcl or C++?

- Tcl
  - Simple Configuration, Setup, Scenario
  - If it's something that can be done without modifying existing Tcl module.
- C++
  - Anything that requires processing each packet
  - Needs to change behavior of existing module

Shadowing

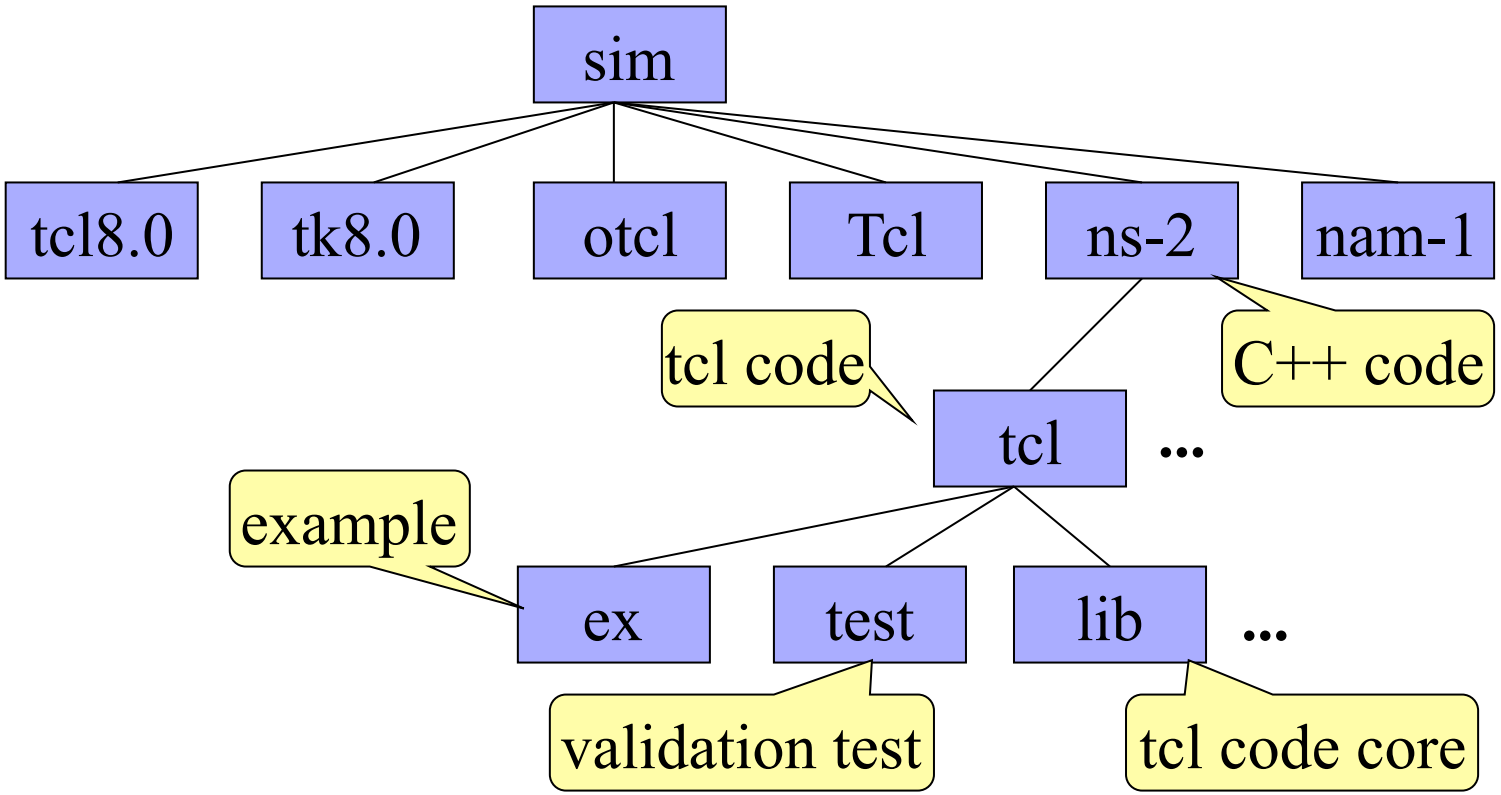

TclObject
OTcl class hierarchy
Agent
Agent/DSDV
C++ class hierarchy
TclObject()
Agent()
DSDVAgent()
Agent/DSDV OTcl shadow object
Agent/DSDV C++ object

Object Correspondence



Outline

NS-2 Directory Structure



Making Changes in C++ Space

- Existing code
  - recompile
- Addition
  - change Makefile and recompile

Making Changes in otcl Space

- Existing code
  - recompile
  - source
- Addition
  - source
  - change Makefile (NS_TCL_LIB), tcl/ns-lib.tcl (source) and recompile

Installation

- Unix variants
  - Download NS-allinone-2.27 package
  - Contains
    - TCL/TK 8.4.5
    - oTCL 1.8
    - Tclcl 1.15
    - Ns2
    - Nam -1

Installation

- sudo apt-get install ns2
- sudo apt-get purge nam
- sudo dpkg -i nam_1.14_amd64.deb
- sudo apt-mark hold nam
- RESTART the machine

Code for simple topology

- Creating a Simulator Object
  - set ns [new Simulator]
- Setting up files for trace & NAM
  - set trace_nam [open out.nam w]
  - set trace_all [open all.tr w]
- Tracing files using their commands
  - $ns namtrace-all $trace_nam
  - $ns trace-all $trace_all

Code for simple topology

- Closing trace file and starting NAM
  - proc finish { } {
    - global ns trace_nam trace_all
    - $ns flush-trace
    - close $trace_nam
    - close $trace_all
    - exec nam out.nam &
    - exit 0 }

Code for simple topology

- Creating LINK & NODE topology
  - Creating NODES
    - set n1 [$ns node]
    - set n2 [$ns node]
    - set n3 [$ns node]
    - set n4 [$ns node]
    - set r1 [$ns node]
    - set r2 [$ns node]

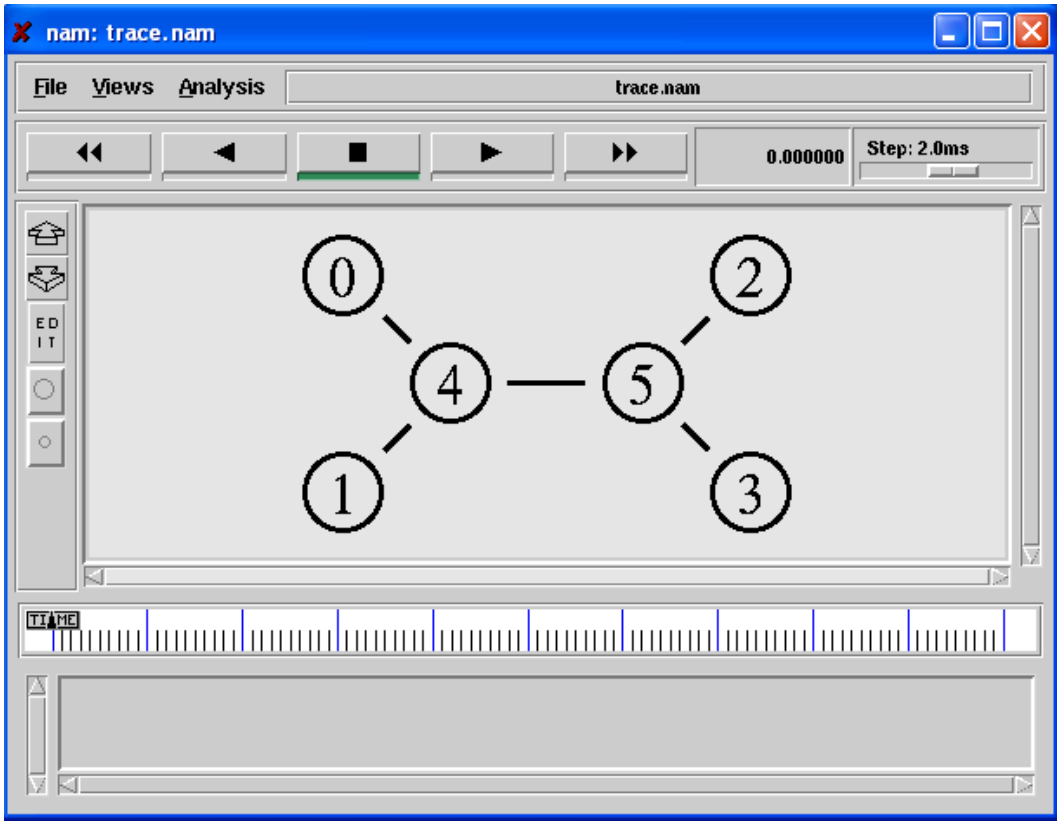
Code for simple topology

- Creating LINKS
  - $ns duplex-link $N1 $R1 2Mb 5ms DropTail
  - set DuplexLink0 [$ns link $N1 $R1]
  - $ns duplex-link $N2 $R1 2Mb 5ms DropTail
  - set DuplexLink1 [$ns link $N2 $R1]
  - $ns duplex-link $R1 $R2 1Mb 10ms DropTail
  - set DuplexLink2 [$ns link $R1 $R2]
  - $ns duplex-link $R2 $N3 2Mb 5ms DropTail
  - set DuplexLink3 [$ns link $R2 $N3]
  - $ns duplex-link $R2 $N4 2Mb 5ms DropTail
  - set DuplexLink4 [$ns link $R2 $N4]

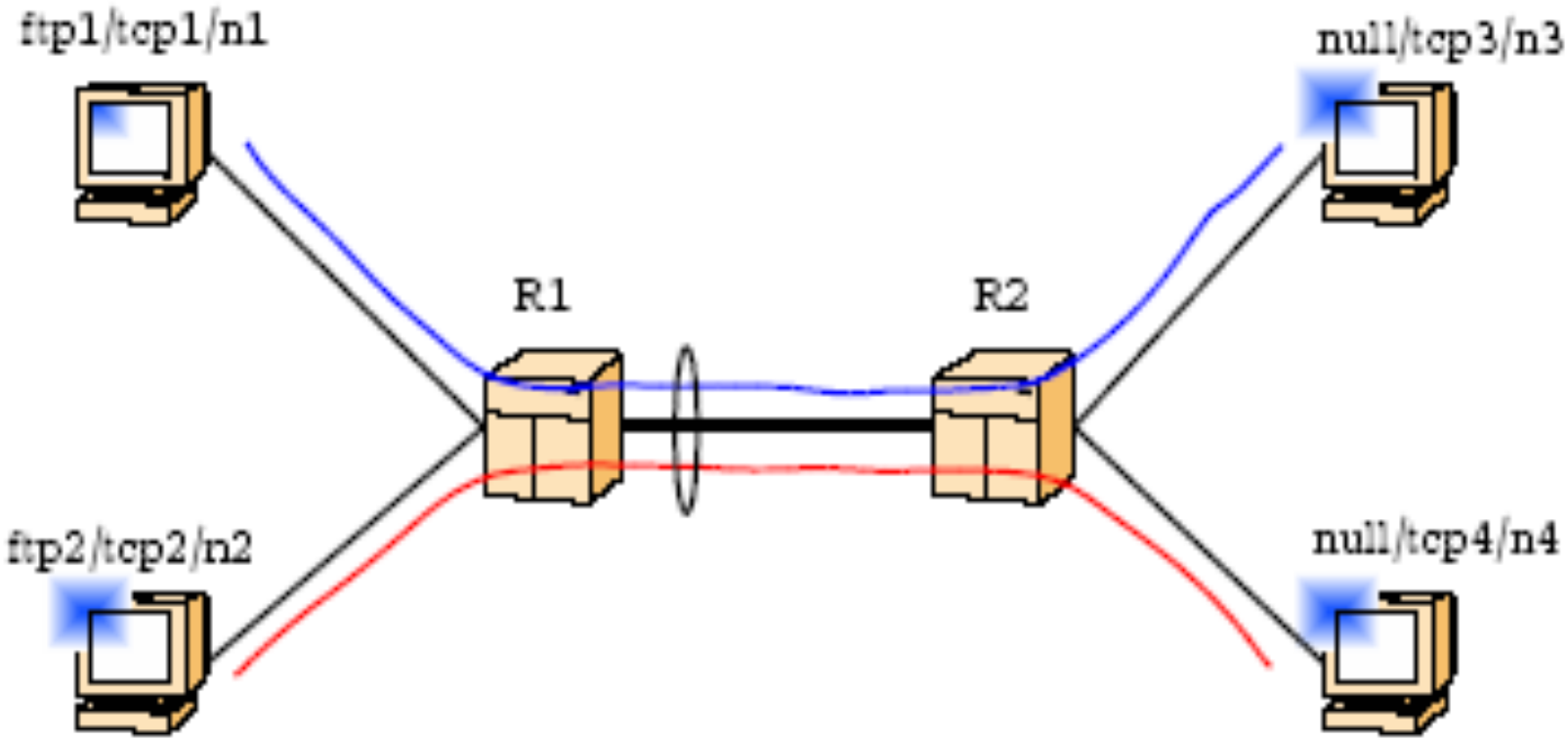
Code for simple topology

- Orientation of links
  - $ns duplex-link-op $N1 $R1 orient right-down
  - $ns duplex-link-op $N2 $R1 orient right-up
  - $ns duplex-link-op $R1 $R2 orient right
  - $ns duplex-link-op $R2 $N3 orient right-up
  - $ns duplex-link-op $R2 $N4 orient right-down

Final topology Generated



Traffic topology aimed at



Generating Traffic

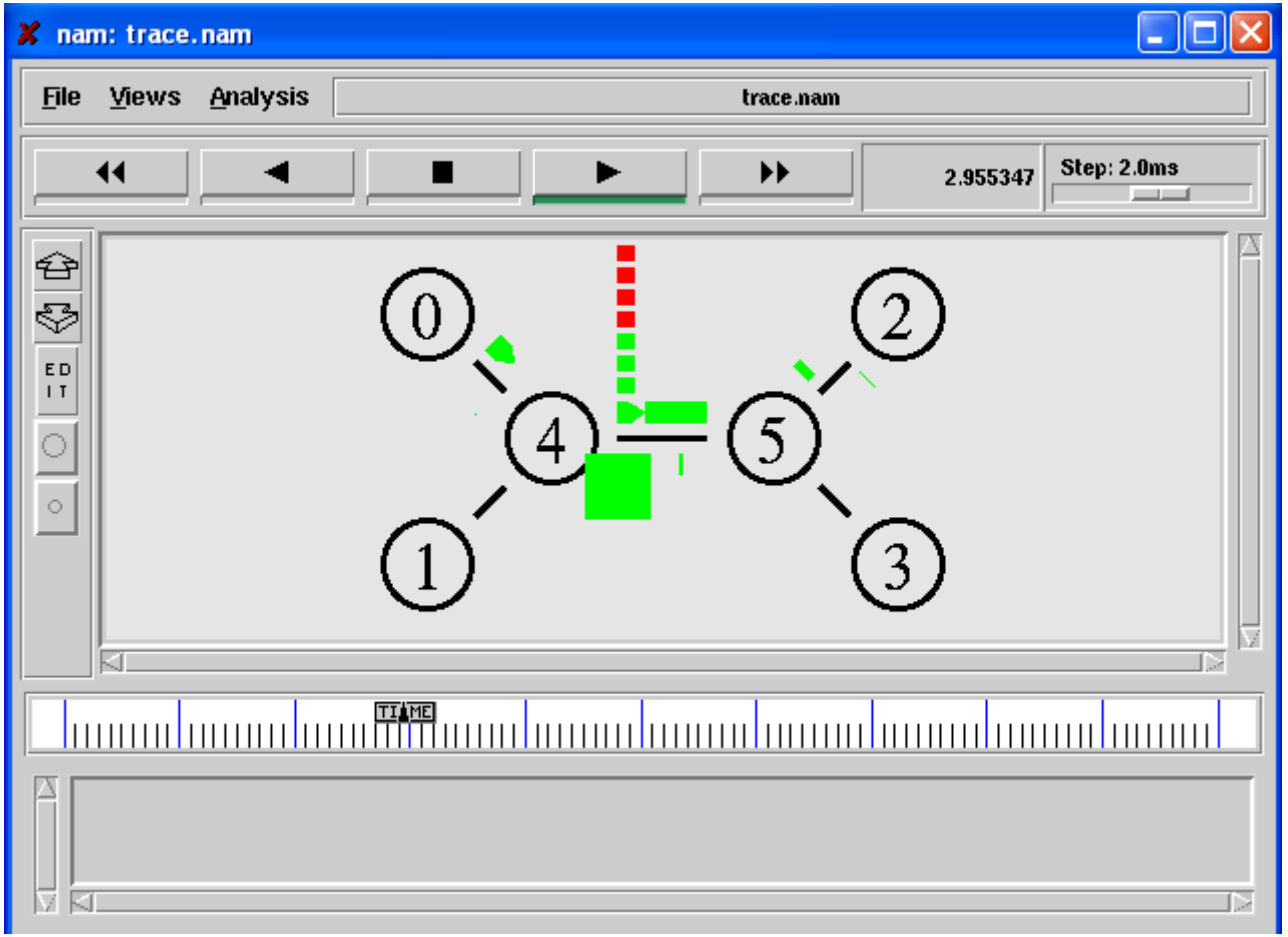
- Attaching AGENT TCP to NODE 1
  - set TCP1 [new Agent/TCP]
  - $ns attach-agent $N1 $TCP1
- Attaching AGENT TCP to NODE 2
  - set TCP2 [new Agent/TCP]
  - $ns attach-agent $N2 $TCP2
- Attaching AGENT TCP to NODE 3
  - set TCP3 [new Agent/TCPSink]
  - $ns attach-agent $N2 $TCP3
- Attaching AGENT TCP to NODE 4
  - set TCP4 [new Agent/TCPSink]
  - $ns attach-agent $N2 $TCP4

Generating Traffic

- Attaching Application (FTP)
  - set FTP0 [new Application/FTP]
  - set FTP1 [new Application/FTP]
  - $FTP0 attach-agent $TCP0
  - $FTP1 attach-agent $TCP1

Setting simulation times

- $ns at 0.5 "$FTP0 start"
- $ns at 0.5 "$FTP1 start"
- $ns at 10.0 "$FTP0 stop"
- $ns at 10.0 "$FTP1 stop“
- $ns at 10.0 “finish”
- Making NS run
  - $ns run


nam: trace.nam
File
Views
Analysis
trace.nam
2.955347
Step: 2.0ms
ED
IT
0
1
2
3
4
5
TIME