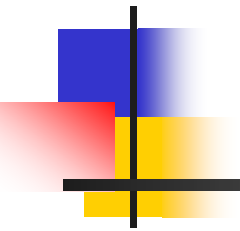


NS-2 Tutorial

Presenter: Qing (Kenny) Shao (SFU/CNL)

Author: Polly Huang (AT&T Labs Research)

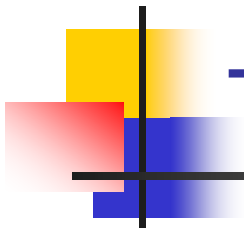
Padmaparna Haldar (USC/ISI)

Xuan Chen (USC/ISI)

Communication Networks Laboratory

http://www.ensc.sfu.ca/research/cnl

Simon Fraser University



Tutorial Schedule

- Introduction
- Ns fundamentals
- Ns programming internal
- Extending ns-2 Simulator

Introduction

- 1989: REAL network simulator
- 1995: DARPA VINT project at LBL, Xerox PARC, UCB, and USC/ISI
- Present: DARPA SAMAN project and NSF CONSER project
  - Collaboration with other researchers including CIRI

Ns Status

- Periodical release (ns-2.1b9a, July 2002)
  - ~200K LOC in C++ and Otcl,
  - ~100 test suites and 100+ examples
  - 371 pages of ns manual
  - Daily snapshot (with auto-validation)
- Stability validation
  - http://www.isi.edu/nsnam/ns/ns-tests.html
- Platform support
  - FreeBSD, Linux, Solaris, Windows and Mac
- User base
  - > 1k institutes (50 countries), >10k users
  - About 300 posts to ns-users@isi.edu every month

Ns functionalities

- Wired world
  - Routing DV, LS, PIM-SM
  - Transportation: TCP and UDP
  - Traffic sources:web, ftp, telnet, cbr, stochastic
  - Queuing disciplines:drop-tail, RED, FQ, SFQ, DRR
  - QoS: IntServ and Diffserv
  - Emulation
- Wireless
  - Ad hoc routing and mobile IP
  - Directed diffusion, sensor-MAC
- Tracing, visualization, various utilities

"Ns" Components

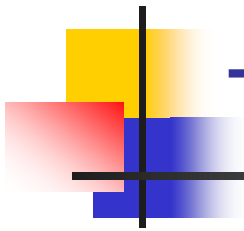
- Ns, the simulator itself
- Nam, the network animator
  - Visualize *ns* (or other) output
  - Nam editor: GUI interface to generate ns scripts
- Pre-processing:
  - Traffic and topology generators
- Post-processing:
  - Simple trace analysis, often in Awk, Perl, or Tcl

Ns Models

- Traffic models and applications:
  - Web, FTP, telnet, constant-bit rate, real audio
- Transport protocols:
  - unicast: TCP (Reno, Vegas, etc.), UDP
  - Multicast: SRM
- Routing and queuing:
  - Wired routing, ad hoc rtg and directed diffusion
  - queuing protocols: RED, drop-tail, etc
- Physical media:
  - Wired (point-to-point, LANs), wireless (multiple propagation models), satellite

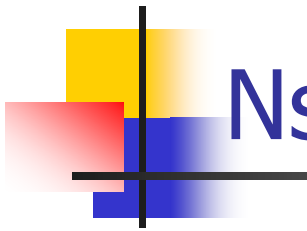
Installation

- Getting the pieces
  - Tcl/TK 8.x (8.3.2 preferred):
    http://resource.tcl.tk/resource/software/tcltk/
  - Otcl and TclCL:
    http://otcl-tclcl.sourceforge.net
  - ns-2 and nam-1:
    http://www.isi.edu/nsnam/dist
- Other utilities
  - http://www.isi.edu/nsnam/ns/ns-build.html
  - Tcl-debug, GT-ITM, xgraph, …



Tutorial Schedule

- Introduction
- Ns fundamentals
- Ns programming internal
- Extending ns-2 Simulator



Ns-2, the Network Simulator

- A *discrete event simulator*
  - Simple model
- Focused on *modeling network protocols*
  - Wired, wireless, satellite
  - TCP, UDP, multicast, unicast
  - Web, telnet, ftp
  - Ad hoc routing, sensor networks
  - Infrastructure: stats, tracing, error models, etc

Ns Architecture

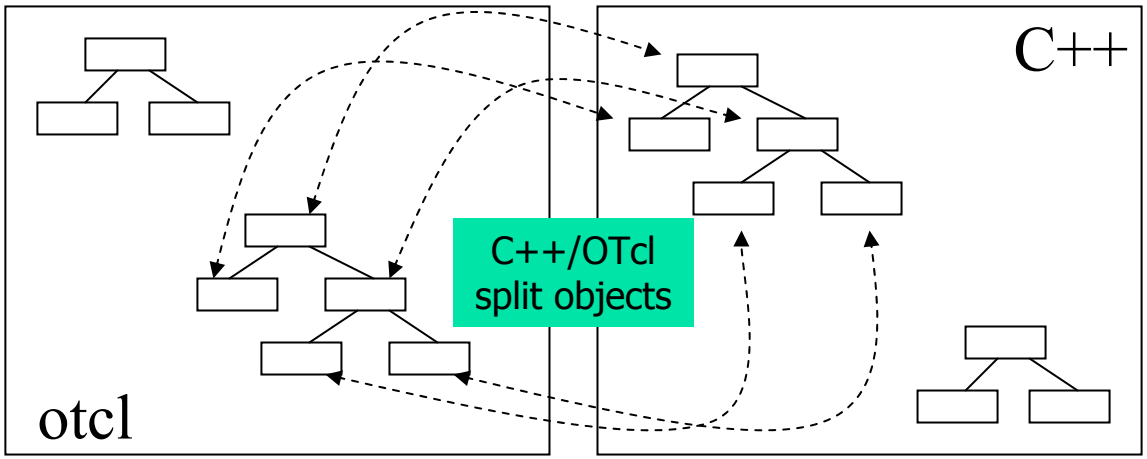
- Object-oriented (C++, OTcl)
- Modular approach
  - Fine-grained object composition

+ Reusability
+ Maintenance
– Performance (speed and memory)
– Careful planning of modularity

C++ and OTcl Separation

- "data" / control separation
  - C++ for "data":
    - per packet processing, core of *ns*
    - fast to run, detailed, complete control
  - OTcl for control:
    - Simulation scenario configurations
    - Periodic or triggered action
    - Manipulating existing C++ objects
    - fast to write and change
- \+ running vs. writing speed
- – Learning and debugging (two languages)

Otcl and C++: The Duality



- OTcl (object variant of Tcl) and C++ share class hierarchy
- TclCL is glue library that makes it easy to share functions, variables, etc

Basic otcl

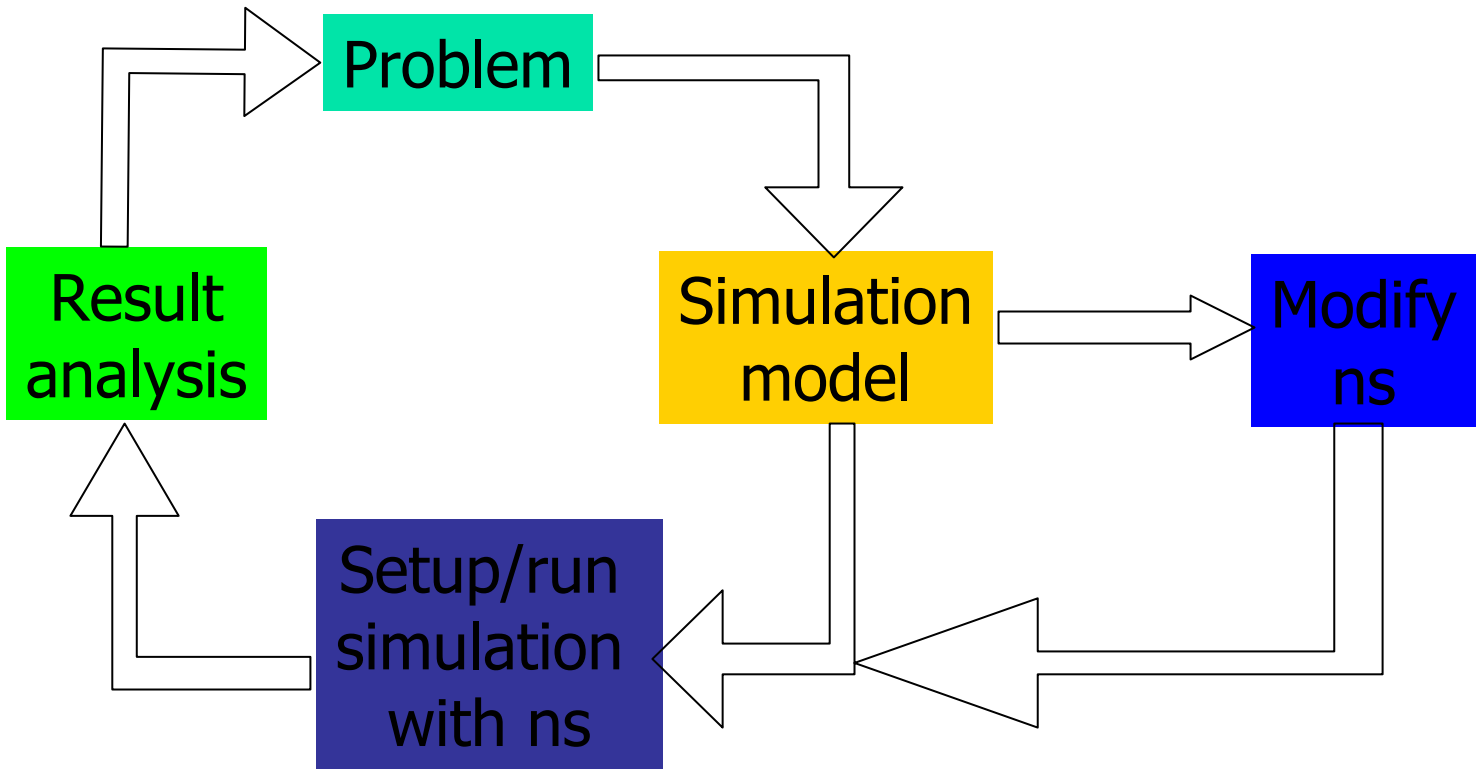
```
Class Person
# constructor:
Person instproc init {age} {
    $self instvar age_
    set age_ $age
}
# method:
Person instproc greet {} {
    $self instvar age_
    puts "$age_ years old: How are you doing?"
}
```

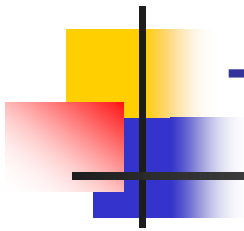
```
# subclass:
Class Kid -superclass Person
Kid instproc greet {} {
    $self instvar age_
    puts "$age_ years old kid: What's up, dude?"
}

set a [new Person 45]
set b [new Kid 15]
$a greet
$b greet
```

=> can easily make variations of existing things (TCP, TCP/Reno)

Using *ns*


Problem
Simulation model
Modify ns
Setup/run simulation with ns
Result analysis



Tutorial Schedule

- Introduction
- Ns fundamentals
- Ns programming internal
- Extending ns-2 Simulator

Ns programming internal

- Create the event scheduler
- Create network
- Turn on tracing
- Setup routing
- Create transport connection
- Create traffic
- Transmit application-level data

Creating Event Scheduler

- Create event scheduler

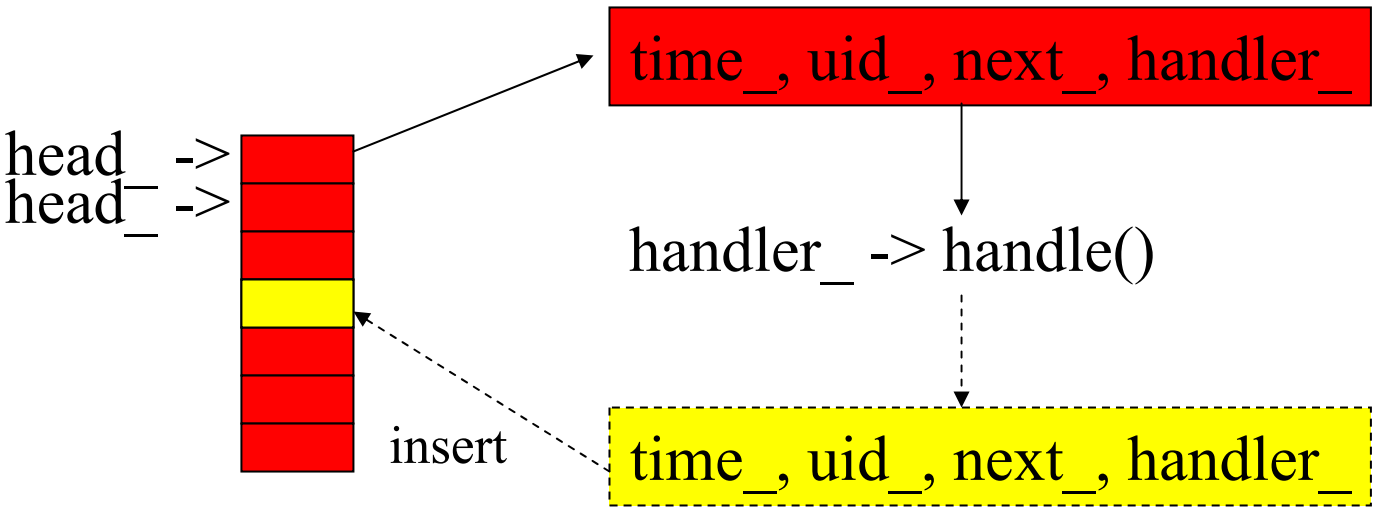
  set ns [new Simulator]
- Schedule events

  $ns at <time> <event>
  - <event>: any legitimate ns/tcl commands

  $ns at 5.0 “finish”
- Start scheduler

  $ns run

Discrete Event Scheduler



Hello World - Interactive Mode

Interactive mode:

```
swallow 71% ns
% set ns [new Simulator]
_o3
% $ns at 1 "puts \"Hello
   World!\""
1
% $ns at 1.5 "exit"
2
% $ns run
Hello World!
swallow 72%
```

Batch mode:

simple.tcl

```
    set ns [new Simulator]
    $ns at 1 "puts \"Hello
      World!\""
    $ns at 1.5 "exit"
    $ns run
```

```
swallow 74% ns simple.tcl
Hello World!
swallow 75%
```

Ns programming internal

- Create the event scheduler
- Create network
- Turn on tracing
- Setup routing
- Create transport connection
- Create traffic
- Transmit application-level data

Creating Network

- Nodes

  set n0 [$ns node]

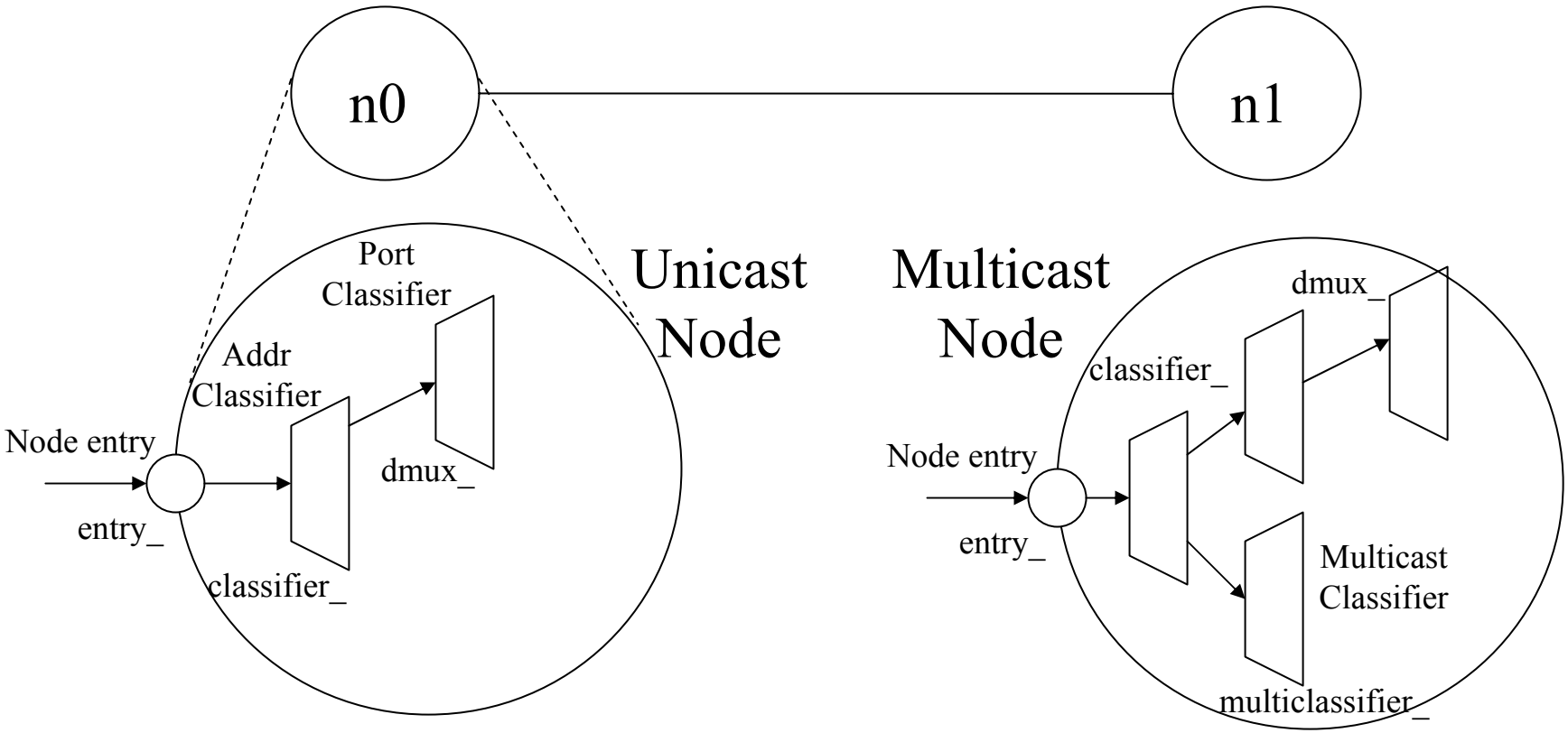
  set n1 [$ns node]

- Links and queuing

  $ns <link_type> $n0 $n1 <bandwidth> <delay> <queue_type>

  - <link_type>: duplex-link, simplex-link
  - <queue_type>: DropTail, RED, CBQ, FQ, SFQ, DRR, diffserv RED queues

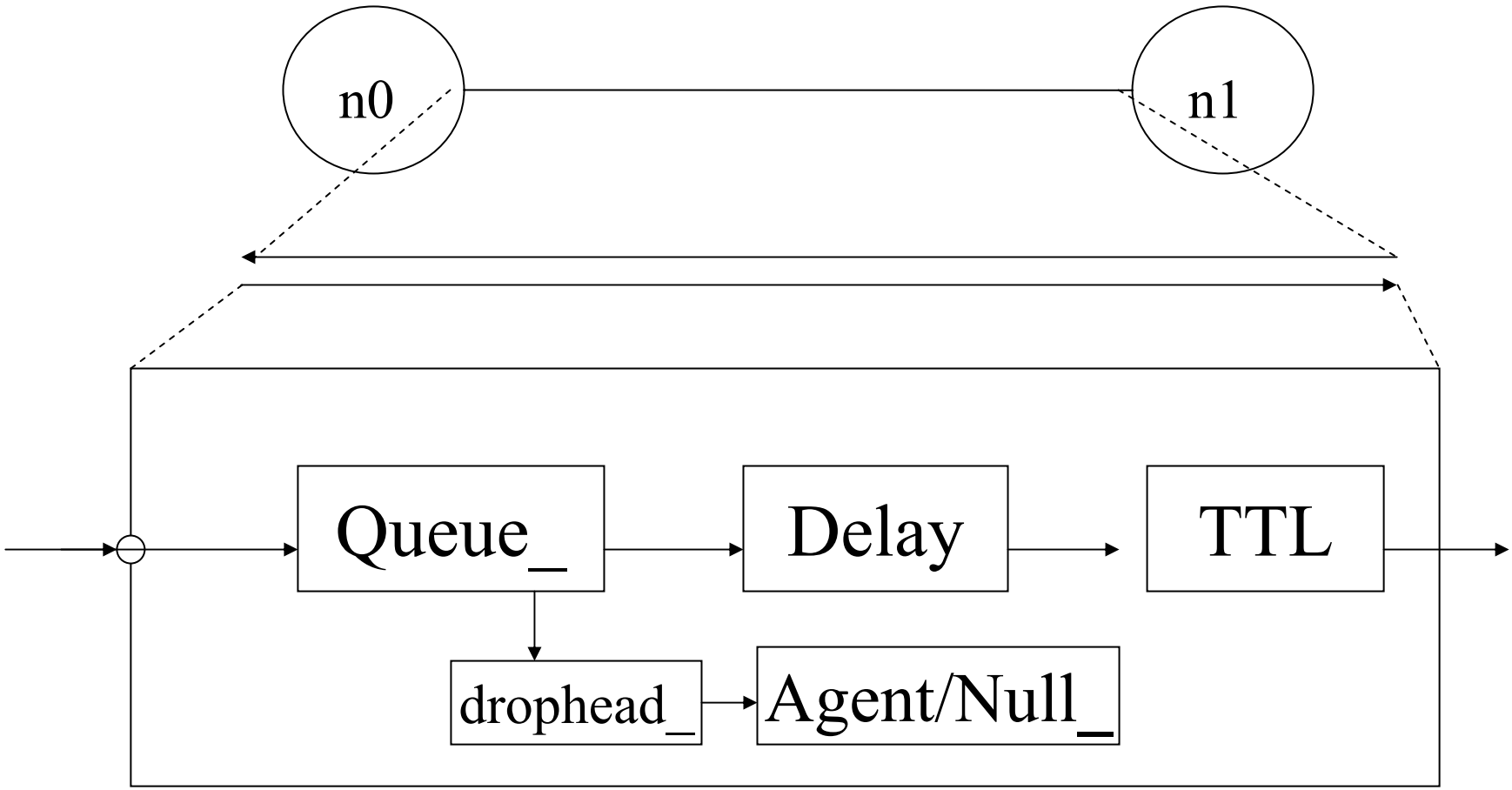
Creating network - Node



set n0 [ns_ node]

Set ns_ [new Simulator –multicast on]
Set n1 [ns_ node]

Creating network - Link



Ns programming internal

- Create the event scheduler
- Create network
- Turn on tracing
- Setup routing
- Create transport connection
- Create traffic
- Transmit application-level data

Tracing and Monitoring

- Packet tracing:
  - On all links: $ns trace-all [open out.tr w]
  - On one specific link: $ns trace-queue $n0 $n1$tr

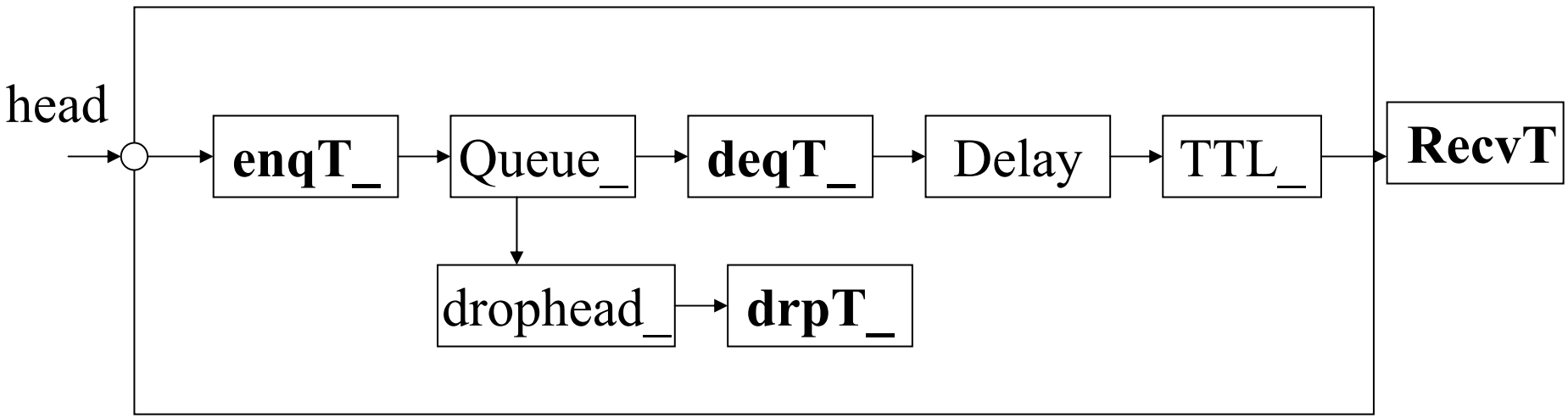
```
<Event> <time> <from> <to> <pkt> <size> -- <fid> <src> <dst> <seq> <attr>
    + 1 0 2 cbr 210 ------- 0 0.0 3.1 0 0
    - 1 0 2 cbr 210 ------- 0 0.0 3.1 0 0
    r 1.00234 0 2 cbr 210 ------- 0 0.0 3.1 0 0
```

- Event tracing (support TCP right now)
  - Record "event" in trace file: $ns eventtrace-all

```
E 2.267203 0 4 TCP slow_start 0 210 1
```

Tracing and Monitoring

$ns trace-all filename or $ns namtrace-all filename



Inserting trace object

Tracing and Monitoring

- Queue monitor

set qmon [$ns monitor-queue $n0 $n1 $q_f $sample_interval]

  - Get statistics for a queue

    $qmon set pdrops_

  - Record to trace file as an optional

```
29.000000000000142 0 1 0.0 0.0 4 4 0 1160 1160 0
```

- Flow monitor

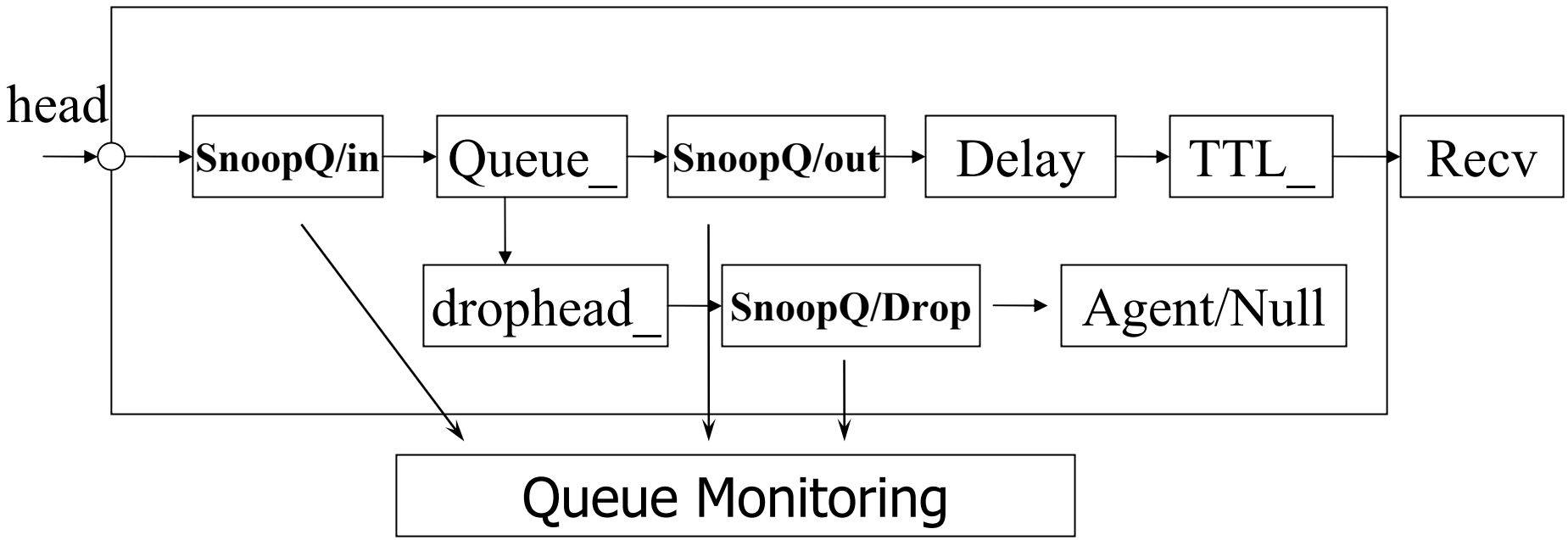
set fmon [$ns_ makeflowmon Fid]

$ns_ attach-fmon $slink $fmon

$fmon set pdrops_

Tracing and Monitoring

$ns monitor-queue node1 node2
$ns at 0.0 qmon trace $filename



Ns programming internal

- Create the event scheduler
- Create network
- Turn on tracing
- Setup routing
- Create transport connection
- Create traffic
- Transmit application-level data

Setup Routing

- Unicast

  $ns rtproto <type>

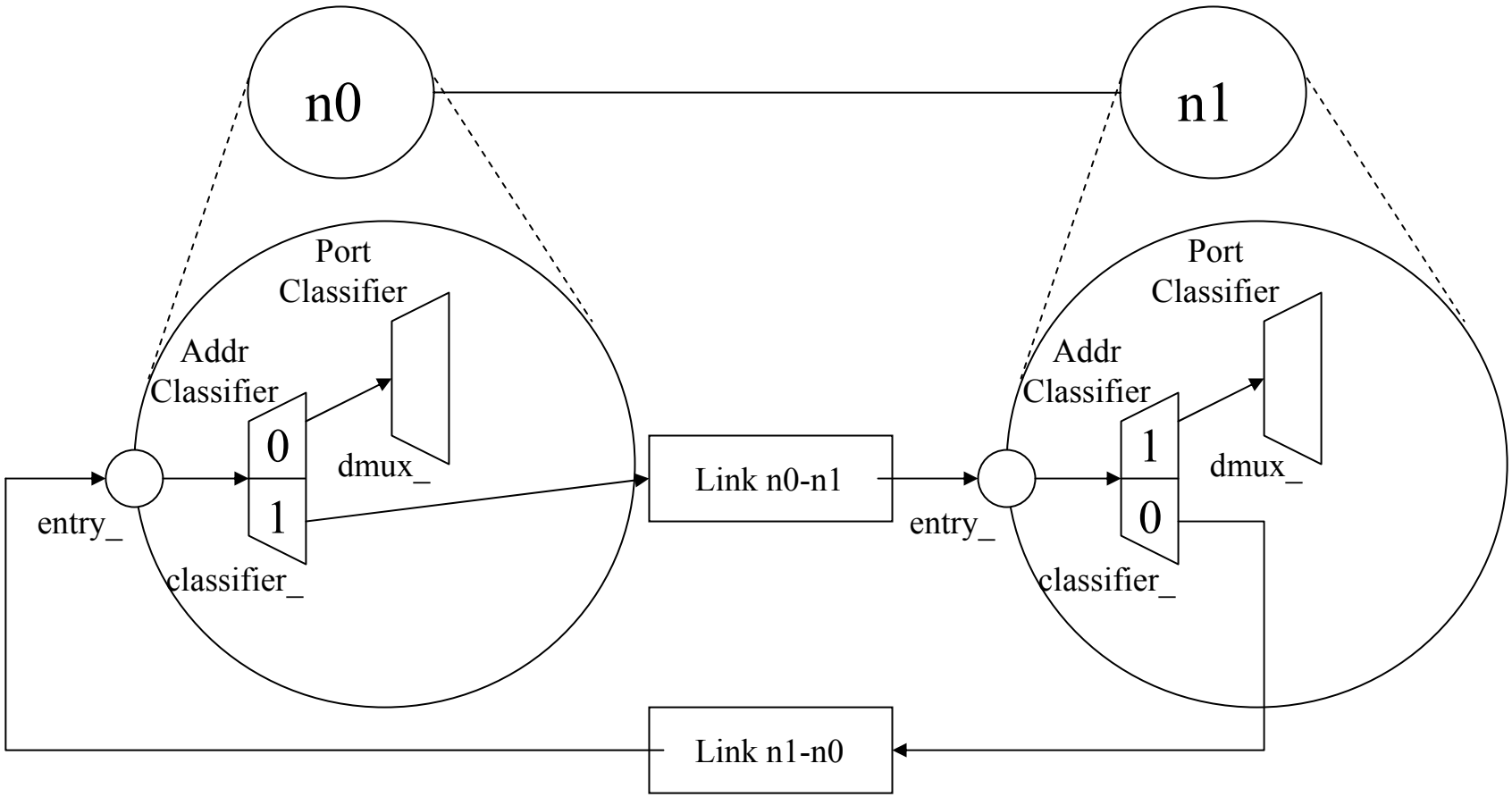
  <type>: Static, Session, DV, cost, multi-path
- Multicast

  $ns multicast (right after [new Simulator])

  $ns mrtproto <type>

  <type>: CtrMcast, DM, ST, BST
- Other types of routing supported: source routing, hierarchical routing

Setup routing



Creating Connection and Traffic

- UDP

```
set udp [new Agent/UDP]
set null [new Agent/Null]
$ns attach-agent $n0 $udp
$ns attach-agent $n1 $null
$ns connect $udp $null
```

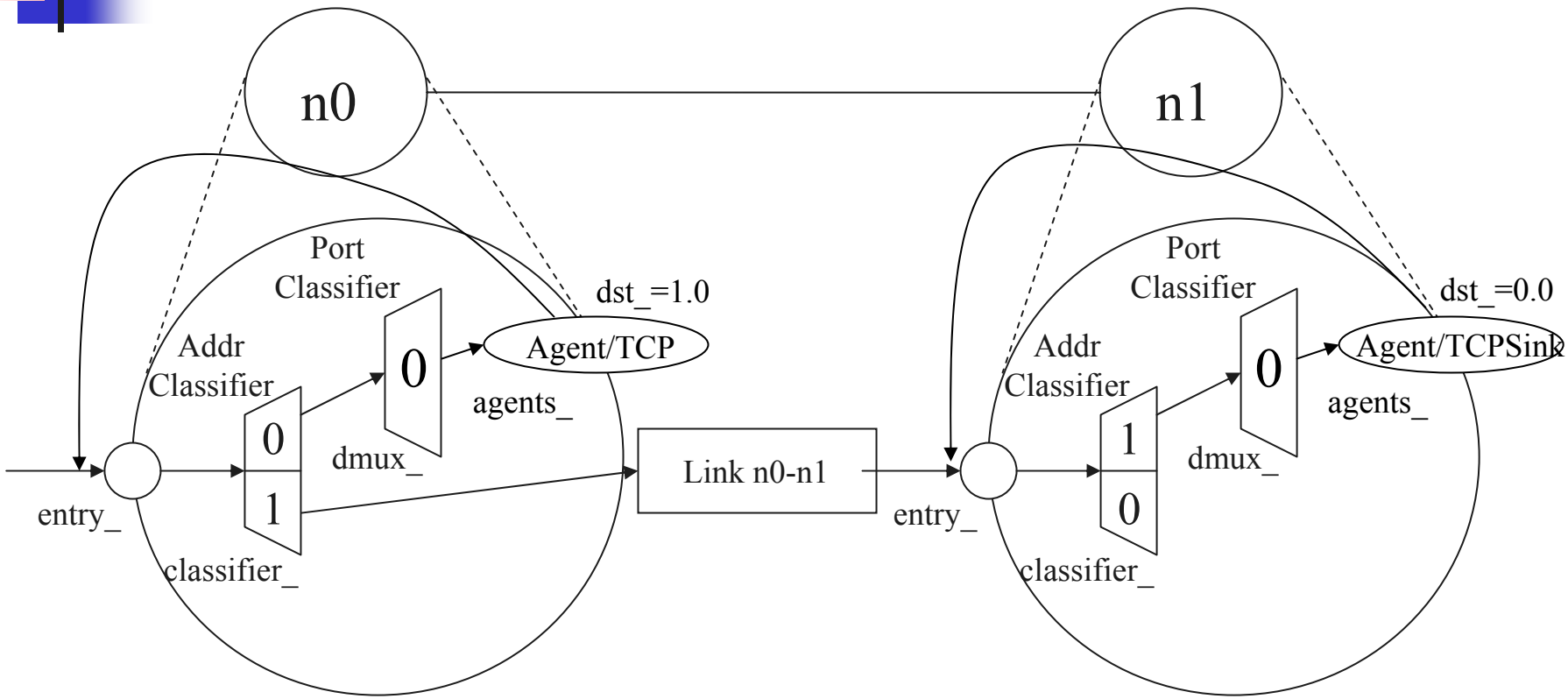
- CBR

```
set src [new Application/Traffic/CBR]
```

- Exponential or Pareto on-off

```
set src [new
    Application/Traffic/Exponential]
set src [new Application/Traffic/Pareto]
```

Creating Connection and Traffic



set tcp [new Agent/TCP]
ns_ attach-agent $n0 $tcp

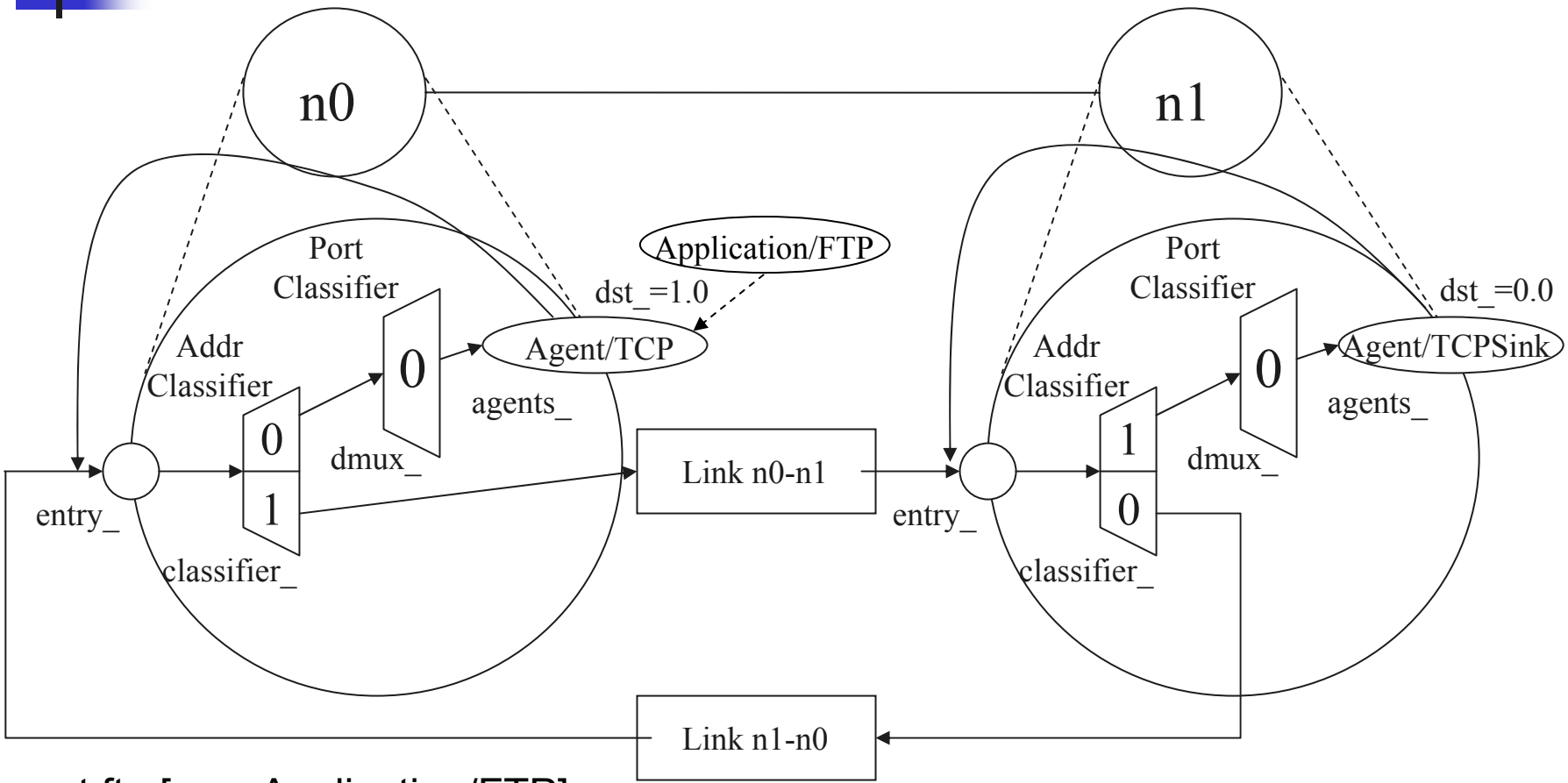
set tcpsink [new Agent/TCPSink]
ns_ attach-agent $n1 $tcpsink

ns_ connect $tcp $tcpsink

Application-Level Simulation

- Features
  - Build on top of existing transport protocol
  - Transmit user data, e.g., HTTP header
- Two different solutions
  - TCP: Application/TcpApp
  - UDP: Agent/Message

Application-Level Simulation




```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.2 “$ftp start”
```

Creating Traffic: Trace Driven

- Trace driven

  set tfile [new Tracefile]

  $tfile filename <file>

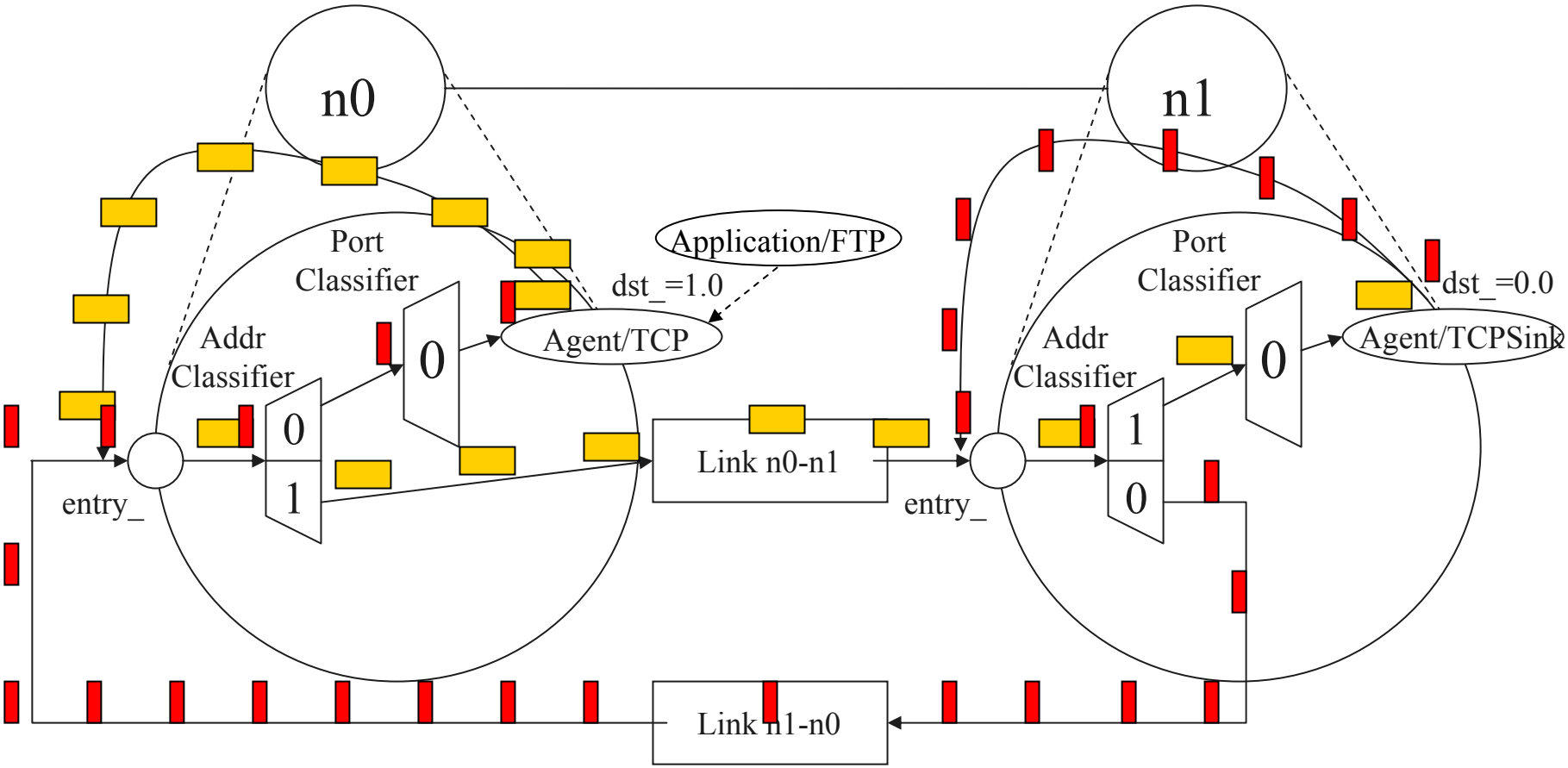
  set src [new Application/Traffic/Trace]

  $src attach-tracefile $tfile

  <file>:

  - Binary format (native!)
  - inter-packet time (msec) and packet size (byte)

Packet Flow


n0
n1
Port
Classifier
Addr
Classifier
0
1
0
Application/FTP
dst_=1.0
Agent/TCP
Link n0-n1
entry_
Port
Classifier
Addr
Classifier
1
0
0
dst_=0.0
Agent/TCPSink
entry_
Link n1-n0

Compare to Real World

- More abstract (much simpler):
  - No addresses, just global variables
  - Connect them rather than name lookup/bind/listen/accept
- Easy to change implementation

  Set tsrc2 [new agent/TCP/Newreno]

  Set tsrc3 [new agent/TCP/Vegas]

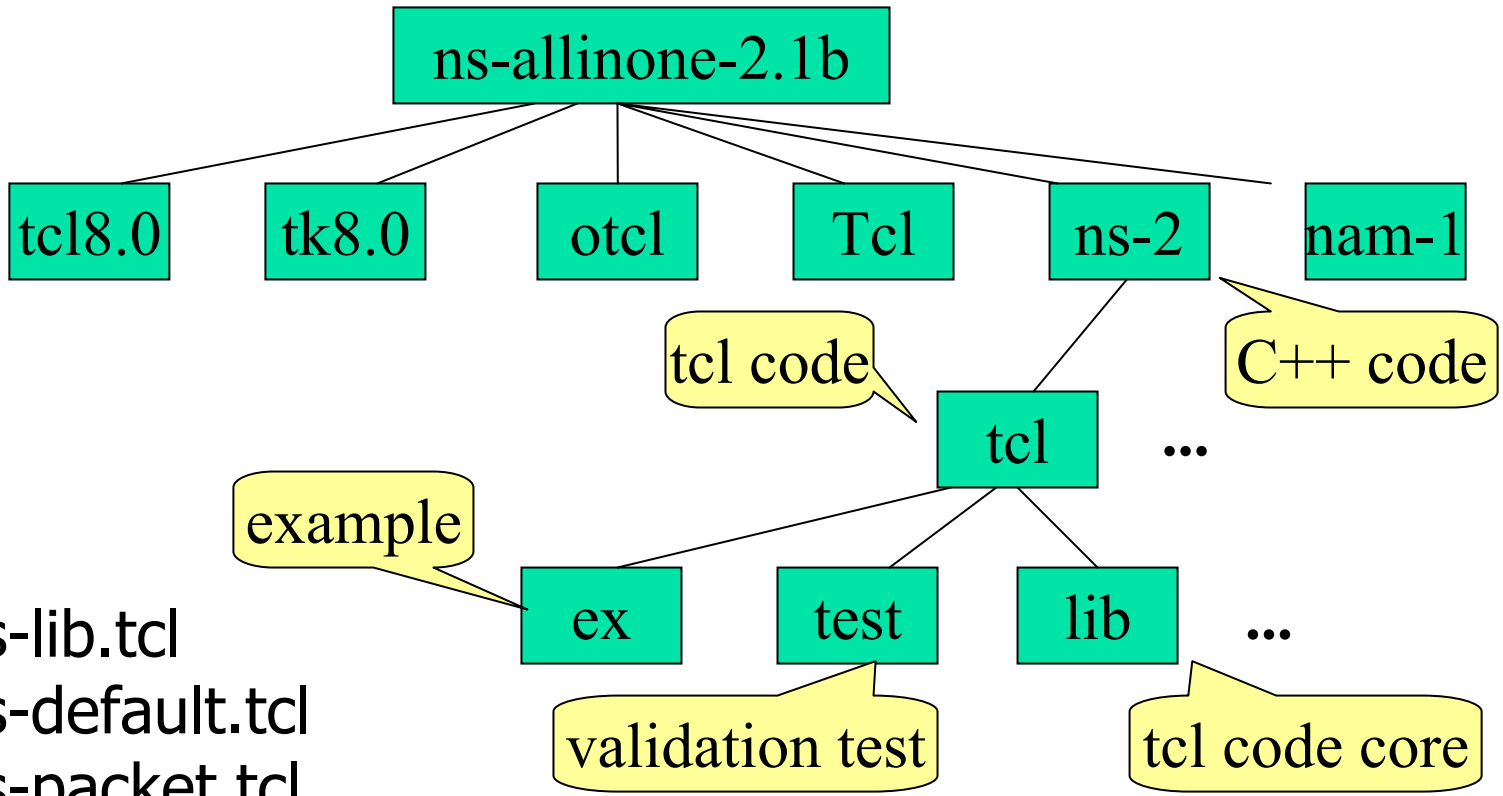
Summary: Generic Script Structure

```
set ns [new Simulator]
# [Turn on tracing]
# Create topology
# Setup packet loss, link dynamics
# Create routing agents
# Create:
#   - multicast groups
#   - protocol agents
#   - application and/or setup traffic sources
# Post-processing procs
# Start simulation
```

Tutorial Schedule

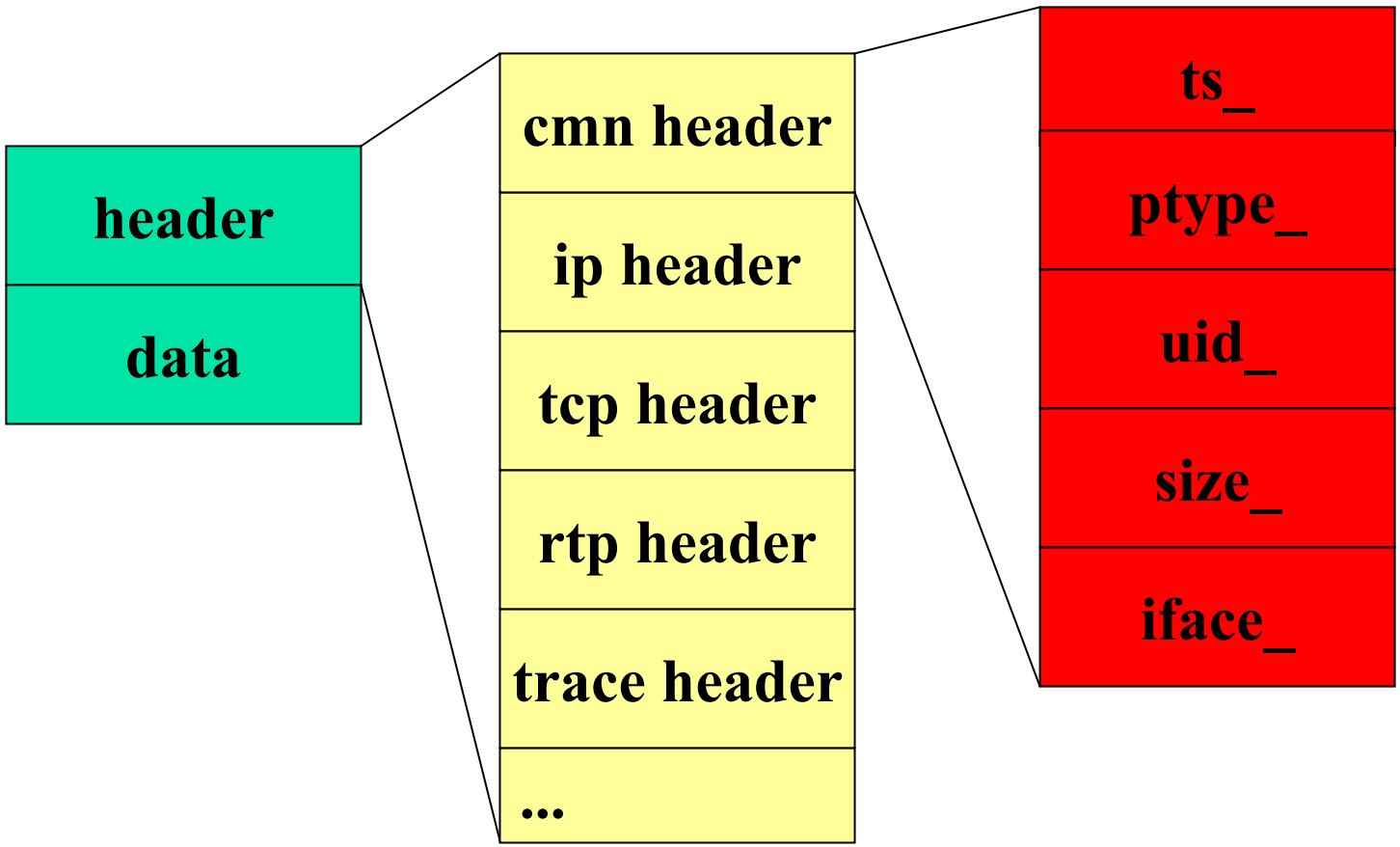
- Basic introduction
- Ns fundamentals
- Ns programming internal
- Extending ns-2 Simulator

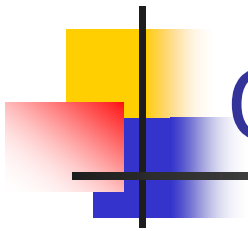
ns-2 Directory Structure



- ns-lib.tcl
- ns-default.tcl
- ns-packet.tcl

Packet Format

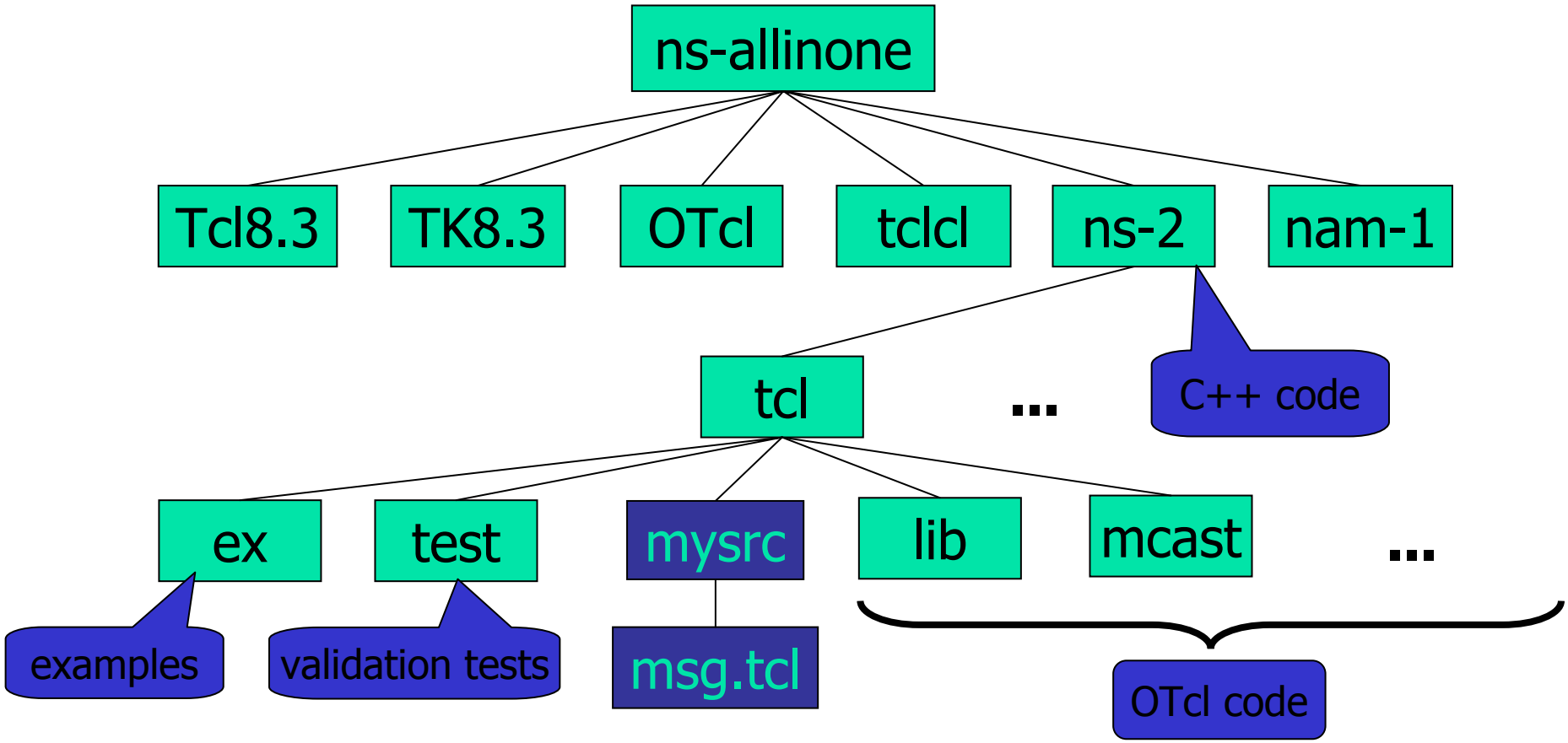




Outline

- Extending ns
  - In OTcl
  - In C++
    - New components

Add your tcl changes into ns



Add your tcl change into ns

- tcl/lib/ns-lib.tcl

```
Class Simulator
…
source ../mysrc/msg.tcl
```

- Makefile

```
NS_TCL_LIB = \
  tcl/mysrc/msg.tcl \
  …
```

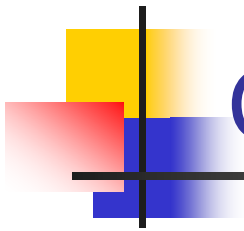
  - Or: change Makefile.in, `make distclean`, then `./configure --enable-debug` , `make depend` and `make`

Extending ns in C++

- Modifying code
  - make depend
  - Recompile
- Adding code in new files
  - Change Makefile
  - make depend
  - recompile

Creating New Components

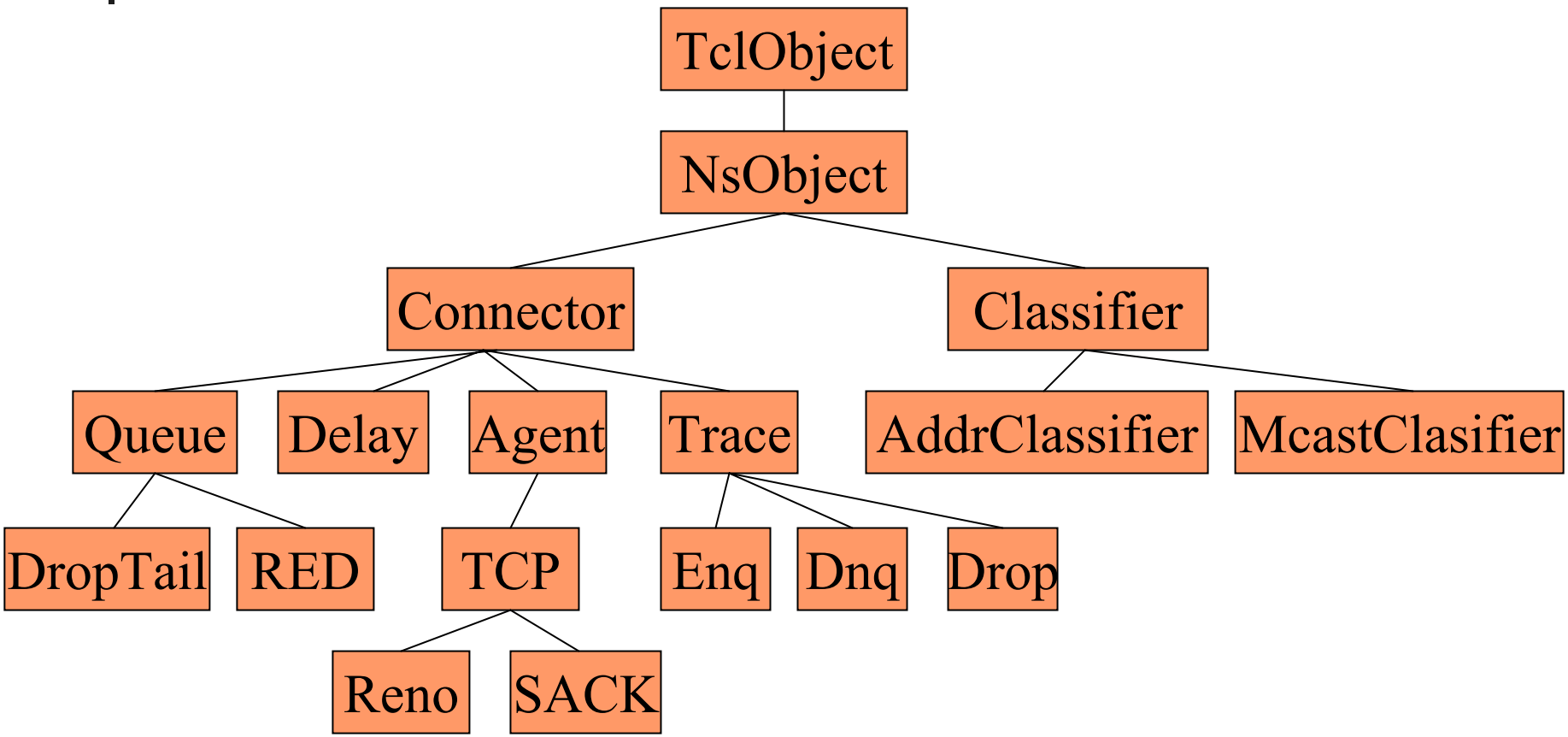
- Guidelines
- Inheritance Hierarchy
- C++ and otcl Interface
- Debugging

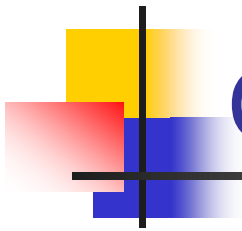


Guidelines

- Decide its inheritance structure
- Create the class and fill in the virtual functions
- Define otcl linkage functions
- Write the necessary otcl code to access your agent

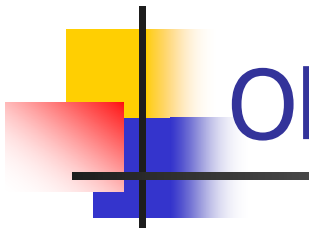
Class Hierarchy (Partial)





C++ and otcl Linkage

- TclClass
- TclObject: bind() method
- TclObject: command() method

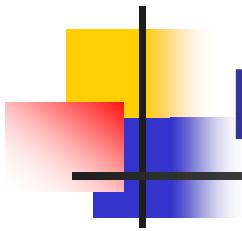


Object Granularity Tips

- Functionality
  - Per-packet processing → C++
  - Hooks, frequently changing code → OTcl
- Data management
  - Complex/large data structure → C++
  - One-time configuration variables → OTcl

Memory Conservation Tips

- Remove unused packet headers
- Avoid `trace-all`
- Use arrays for a sequence of variables
  - Instead of `n$i`, say `n($i)`
- Avoid OTcl temporary variables
- Use dynamic binding
  - `delay_bind()` instead of `bind()`
  - See object.{h,cc}
- See tips for running large sim in ns at `www.isi.edu/ns/nsnam/ns-largesim.html`



Debugging

- printf() and puts “”
- gdb
- tcl debugger
  - http://expect.nist.gov/tcl-debug/
  - place debug 1 at the appropriate location
  - trap to debugger from the script
  - single stepping through lines of codes
  - examine data and code using Tcl-ish commands

ns→nam Interface

- Color
- Node manipulation
- Link manipulation
- Topology layout
- Protocol state
- Misc

nam Interface: Color

- Color mapping

```
$ns color 40 red
$ns color 41 blue
$ns color 42 chocolate
```

- Color ↔ flow id association

```
$tcp0 set fid_ 40 ;# red packets
$tcp1 set fid_ 41 ;# blue packets
```

nam Interface: Nodes

- Color

```
$node color red
```

- Shape (can't be changed after sim starts)

```
$node shape box  ;# circle, box, hexagon
```

- Marks (concentric "shapes")

```
$ns at 1.0 "$n0 add-mark m0 blue box"
$ns at 2.0 "$n0 delete-mark m0"
```

- Label (single string)

```
$ns at 1.1 "$n0 label \"web cache 0\""
```

nam Interfaces: Links

- Color

```
$ns duplex-link-op $n0 $n1 color "green"
```

- Label

```
$ns duplex-link-op $n0 $n1 label "abced"
```

- Dynamics (automatically handled)

```
$ns rtmodel Deterministic {2.0 0.9 0.1} $n0 $n1
```

- Asymmetric links not allowed

nam Interface: Topo Layout

- "Manual" layout: specify everything

```
$ns duplex-link-op $n(0) $n(1) orient right
$ns duplex-link-op $n(1) $n(2) orient right
$ns duplex-link-op $n(2) $n(3) orient right
$ns duplex-link-op $n(3) $n(4) orient 60deg
```

- If anything missing → automatic layout

nam Interface: Misc

- Annotation
  - Add textual explanation to your simulation

```
$ns at 3.5 "$ns trace-annotate \"packet
    drop\""
```

- Set animation rate

```
$ns at 0.0 "$ns set-animation-rate
    0.1ms"
```

Help and Resources

- Ns and nam build questions
  - http://www.isi.edu/nsnam/ns/ns-build.html
- Ns mailing list: ns-users@isi.edu
- Ns manual and tutorial (in distribution)
- TCL: http://dev.scriptics.com/scripting
- Otcl tutorial (in distribution): ftp://ftp.tns.lcs.mit.edu/pub/otcl/doc/tutorial.html