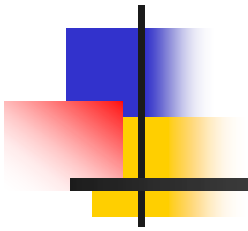


ns-2 Tutorial

Presenter: Savio Lau (SFU/CNL)
Communication Networks Laboratory

compiled from slides:

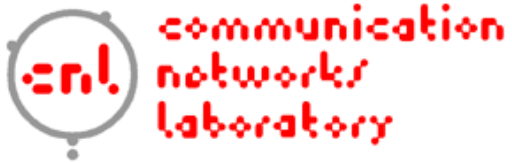
John Heidemann (USC/ISI)
Polly Huang (ETH Zurich)

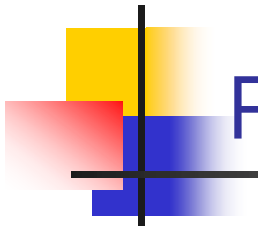
UCLA/IPAM presentation, Mar. 2002

and

Padmaparna Haldar (USC/ISI)
Xuan Chen (USC/ISI)

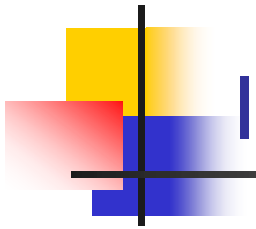
ISI ns-2 Tutorial 2002, Nov. 2002





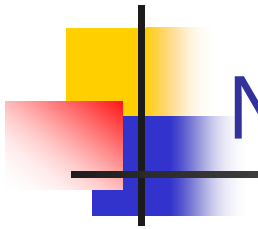
Roadmap

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



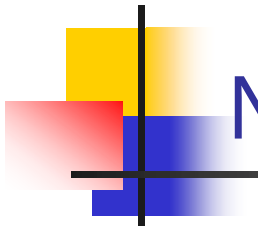
Introduction

- 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 ICIR (formerly ACIRI)



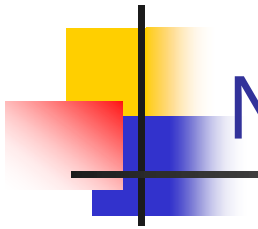
Ns status

- Periodical release (ns-2.29, Oct. 2005)
  - ~200k lines of code in C++ and OTcl,
  - ~100 test suites and 100+ examples
  - 371 pages of ns-2 manual
  - Daily snapshot (with auto-validation)
- Stability validation
  - http://www.isi.edu/nsnam/ns/ns-tests.html



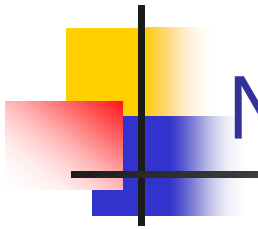
Ns status

- 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



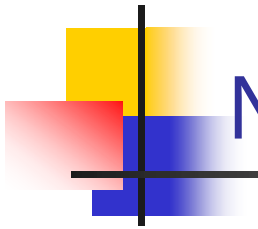
Ns functionalities

- Wired world
  - Routing: distance vector (DV), link state (LS), multicast
  - Transport protocols: TCP, UDP, RTP and SCTP
  - Traffic sources: web, ftp, telnet, cbr, stochastic
  - Queuing disciplines: drop-tail, RED, FQ, SFQ, DRR
  - QoS: IntServ and Diffserv
  - Emulation
- Wireless
  - Ad hoc routing (AODV, DSDV) and mobile IP
  - Directed diffusion, sensor-MAC
- Tracing, visualization, various utilities



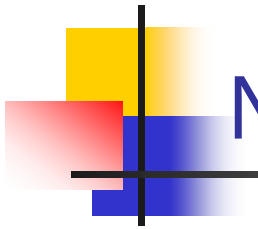
Ns components

- Ns, the simulator
- Nam, the network animator:
  - visualize ns (or other) outputs
  - Nam editor: GUI interface to generate ns scripts
- Pre-processing:
  - traffic and topology generators
- Post-processing:
  - trace analysis with Unix or GNU/Linux tools like awk, Perl, or Tcl
  - graphical visualization with xgraph



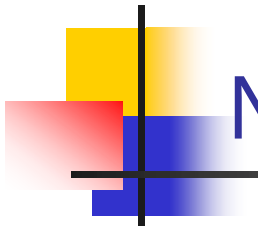
Ns components

- Main components of ns-2
  - Tcl/TK 8.x (8.4.5 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, ...



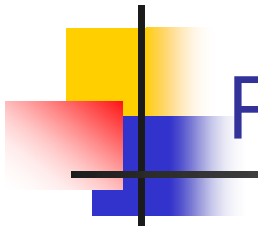
Ns installation notes

- If the GNU/Linux distribution comes with Tcl/TK:
  - install each individual packages separately (ns, nam, xgraph, GT-ITM) to avoid conflicts and to save space
- If you are unfamiliar with the UNIX or GNU/Linux environment:
  - install ns-allinone
- Ns is available for Windows 9x/2000/XP under Cygwin:
  - not widely supported and problematic – avoid it



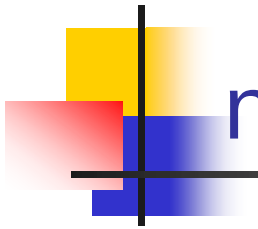
Ns models

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



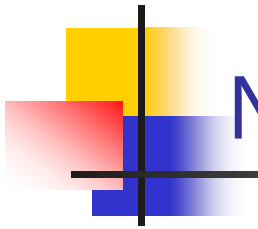
Roadmap

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



ns-2, the Network Simulator

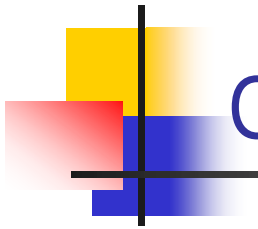
- A discrete event simulator
- Focused on modeling network protocols:
  - wired, wireless, satellite
  - TCP, UDP, multicast, unicast
  - web, telnet, ftp
  - ad hoc routing, sensor networks
  - stats, tracing, error models, etc.



Ns architecture

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

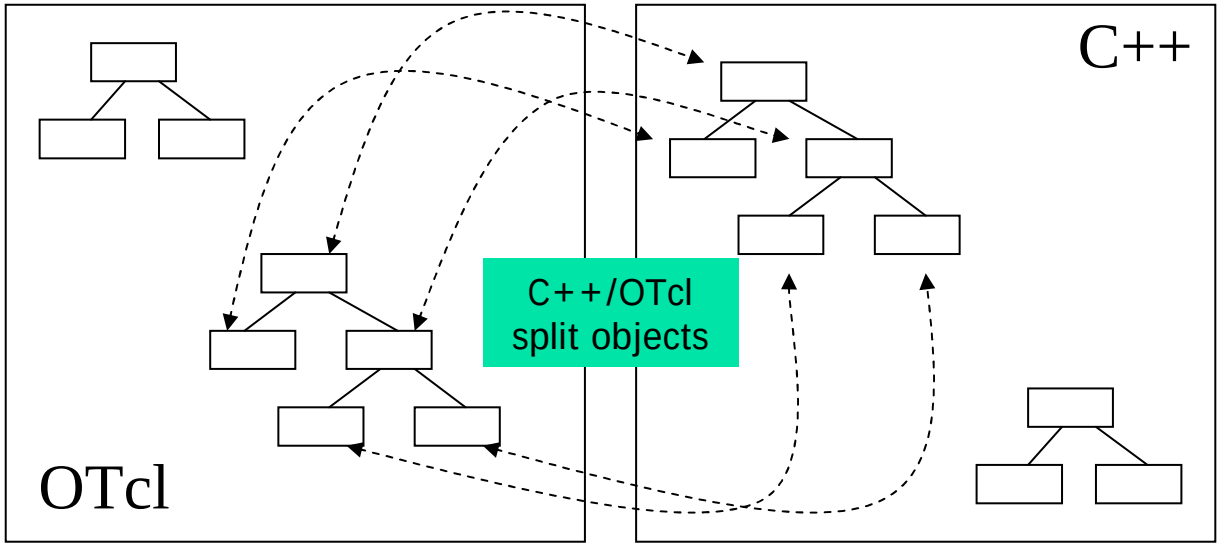
+ Reusability
+ Maintenance
– Performance (speed and memory)
– Careful planning of modularity



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

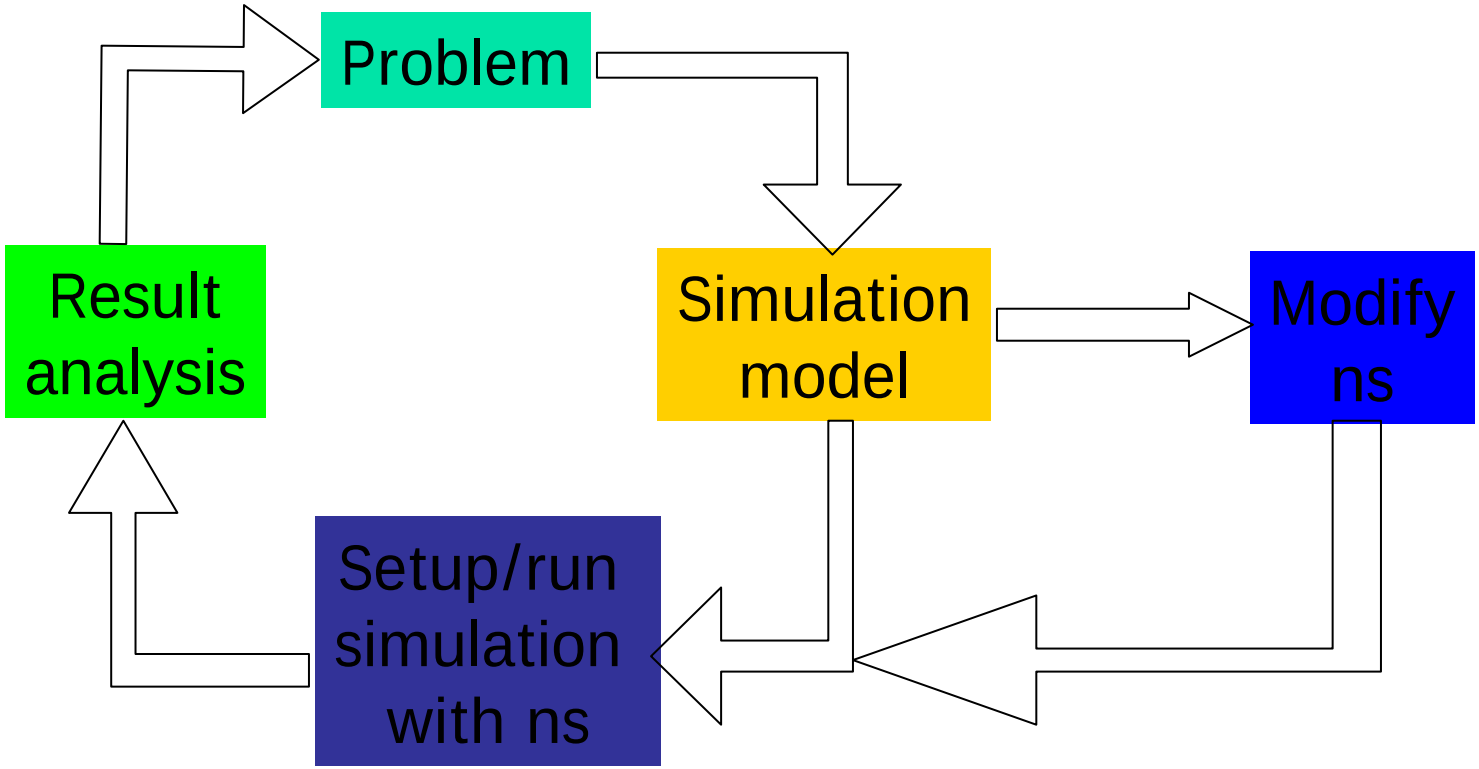
Basic OTcl

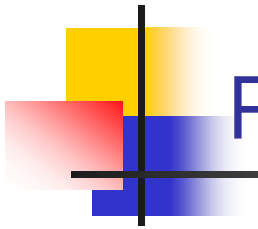
```
Class Person
# constructor:
Person instproc init {age} {
    $self instvar age_
    set age_ $age
}
# method:
Person instproc greet {} {
    $self instvar age_
    puts "$age_ years old: How are
    you doing?"
}
```

```
# 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
```

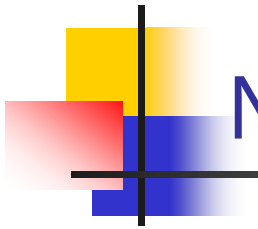
Using ns-2


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



Roadmap

- 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 connection and traffic
- Transmit application-level data

Creating event scheduler

- Create event scheduler

```
set ns [new Simulator]
```

- Schedule events

```
$ns at <time> <event>
```

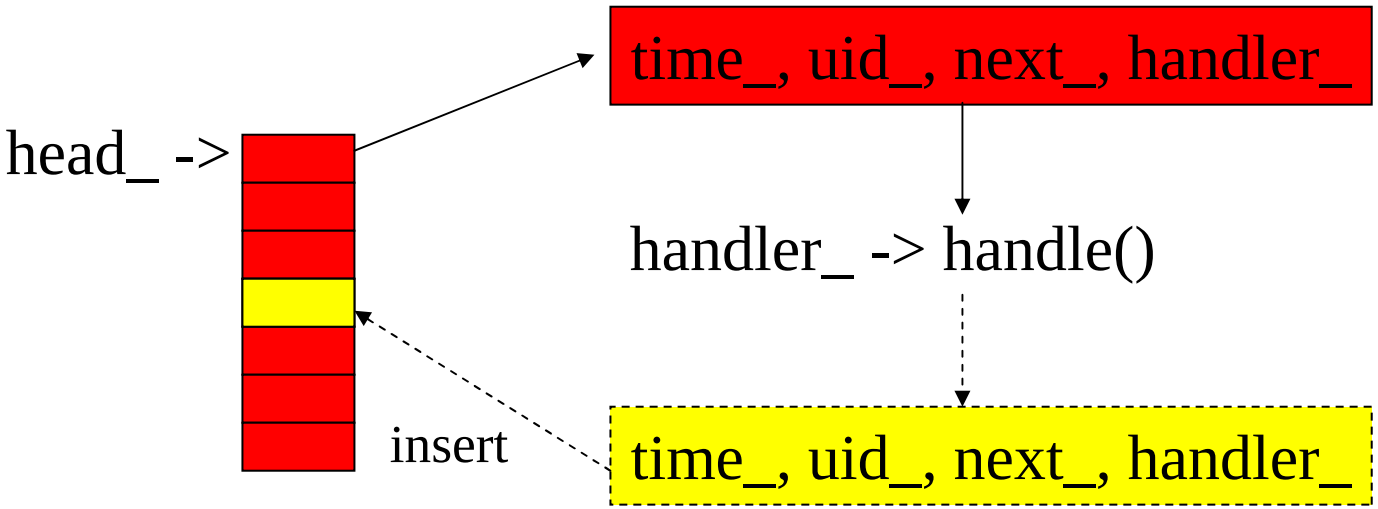
  - `<event>`: any legitimate ns/tcl commands

```
$ns at 5.0 “finish”
```

- Start scheduler

```
$ns run
```

Discrete event scheduler



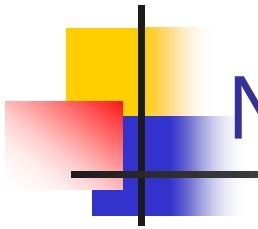
Hello world - interactive mode

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%
```

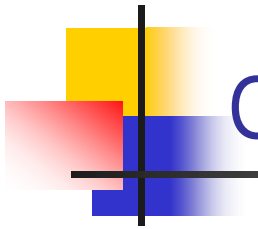
Batch mode:

```
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 connection and traffic
- Transmit application-level data



Creating network

- Nodes

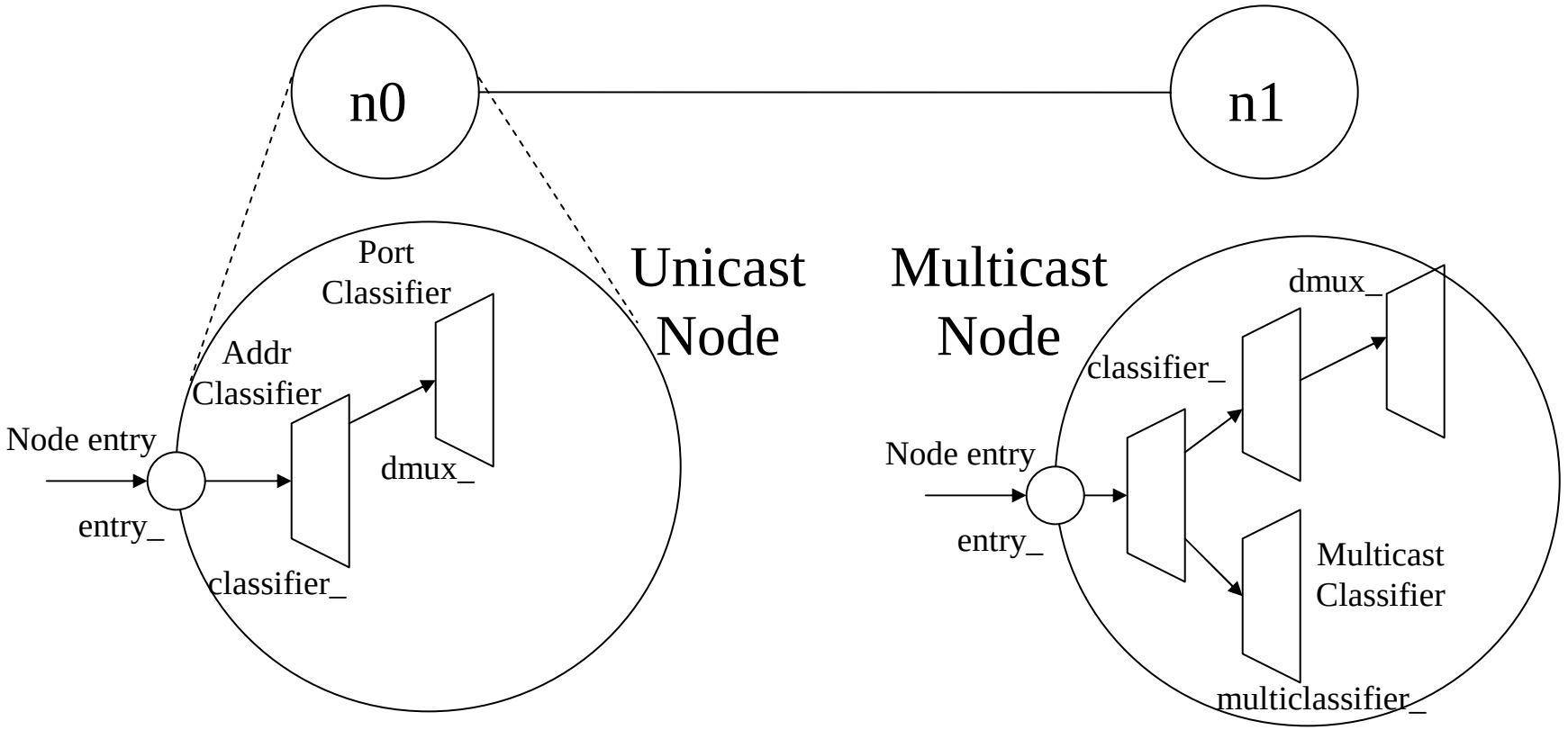
```
set n0 [$ns node]
set n1 [$ns node]
```

- 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

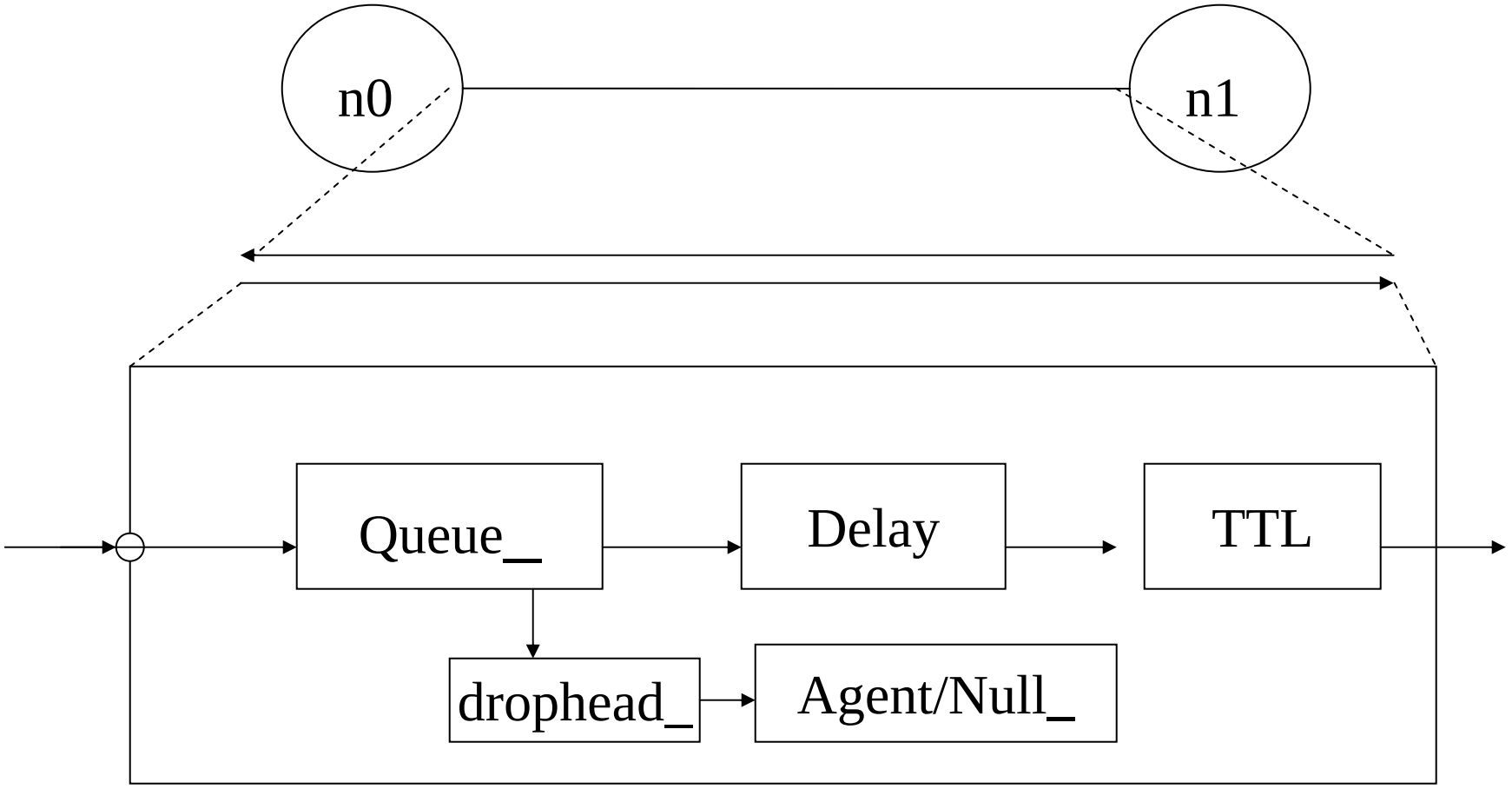
Creating network - node

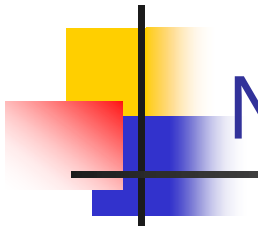



```
set n0 [ns_ node]
```

```
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 connection and 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`

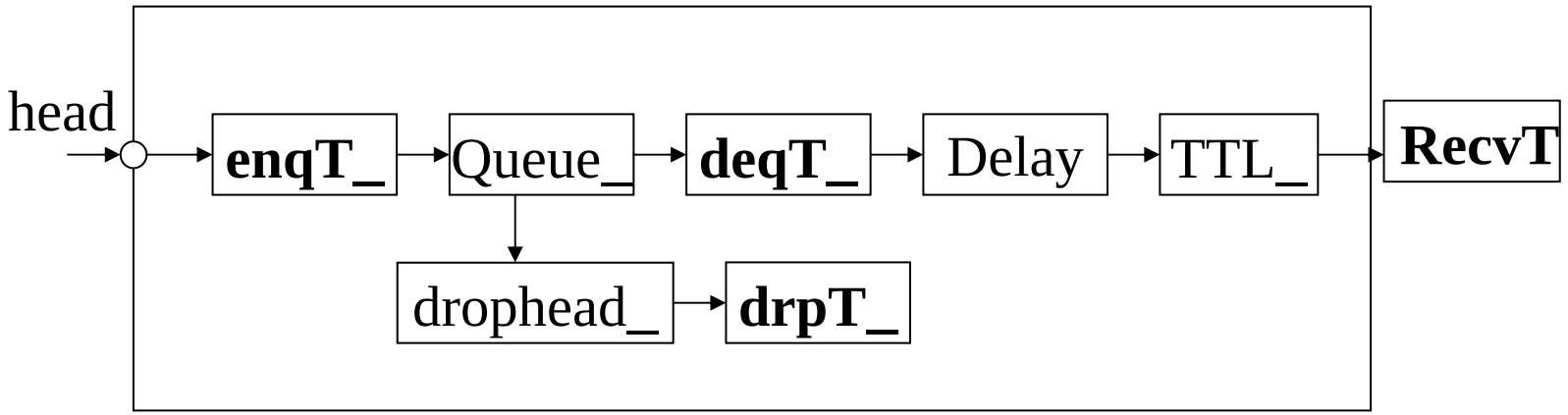
```
<Event> <time> <from> <to> <pkt> <size> -- <fid> <src> <dst> <seq> <attr>
    + 1 0 2 cbr 210 ------- 0 0.0 3.1 0 0
    - 1 0 2 cbr 210 ------- 0 0.0 3.1 0 0
    r 1.00234 0 2 cbr 210 ------- 0 0.0 3.1 0 0
```

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

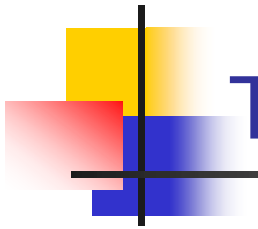
```
E 2.267203 0 4 TCP slow_start 0 210 1
```

Tracing and monitoring

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



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 statistics to trace file as an option

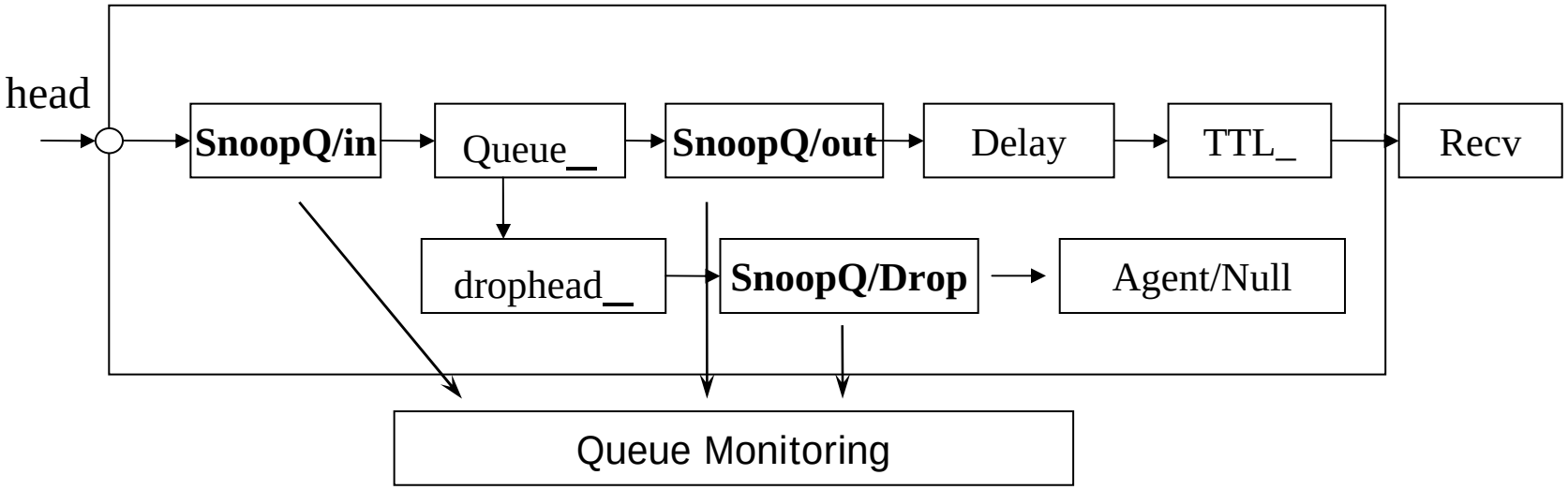
```
29.000000000000142 0 1 0.0 0.0 4 4 0 1160 1160 0
```

- Flow monitor

```
set fmon [$ns makeflowmon Fid]
$ns attach-fmon $slink $fmon
$fmon set pdrops_
```

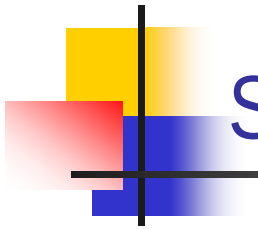
Tracing and monitoring

```
$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 connection and traffic
- Transmit application-level data



Setup routing

- Unicast

  `$ns rtproto <type>`

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

- Multicast

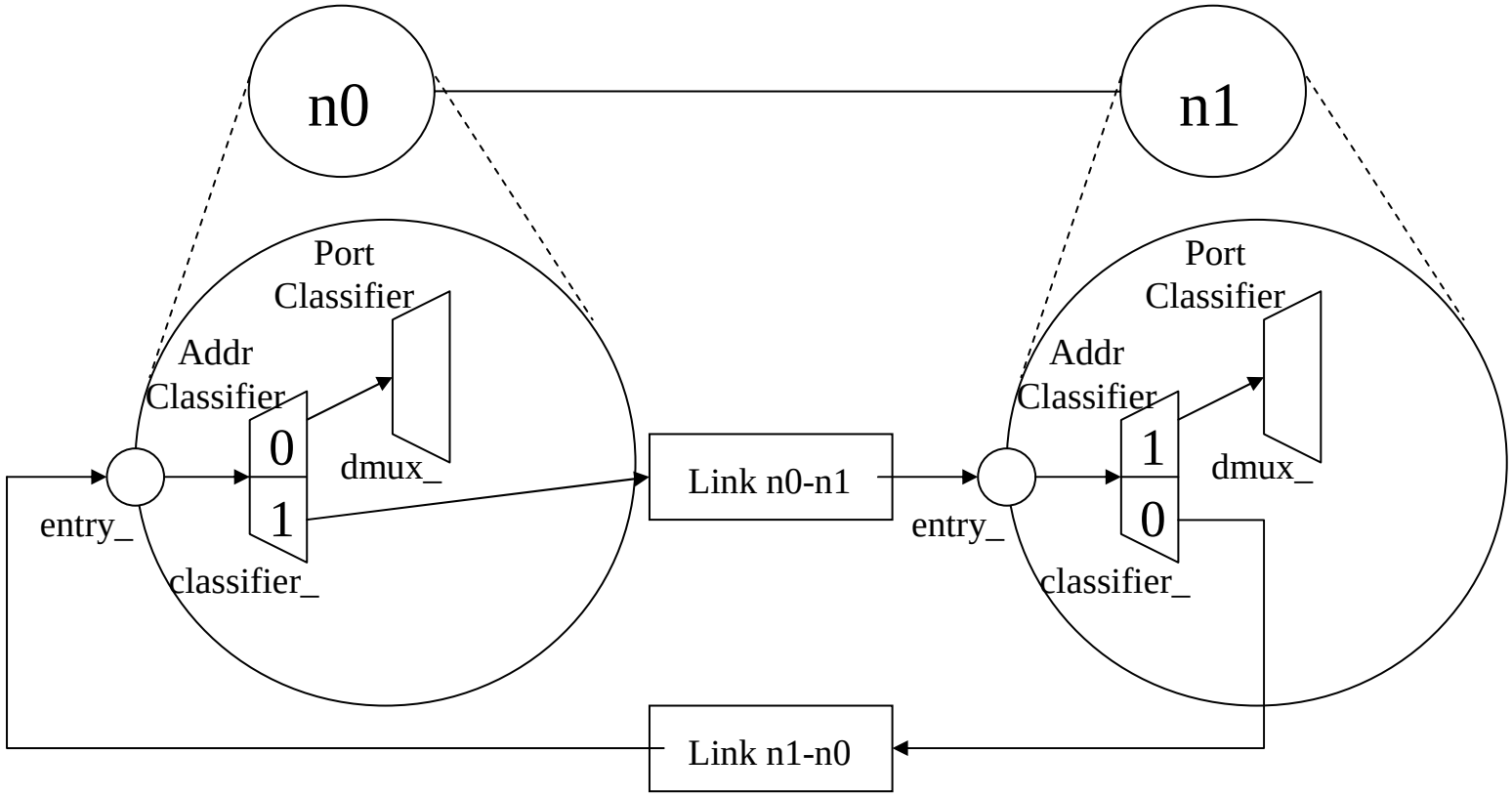
  `$ns multicast` (right after `[new Simulator]` call)

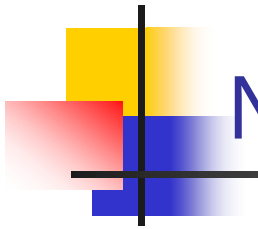
  `$ns mrtproto <type>`

  `<type>`: CtrMcast, DM, ST, BST

- Other types of routing supported: source routing, hierarchical routing

Setup routing





Ns programming internal

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

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]
```

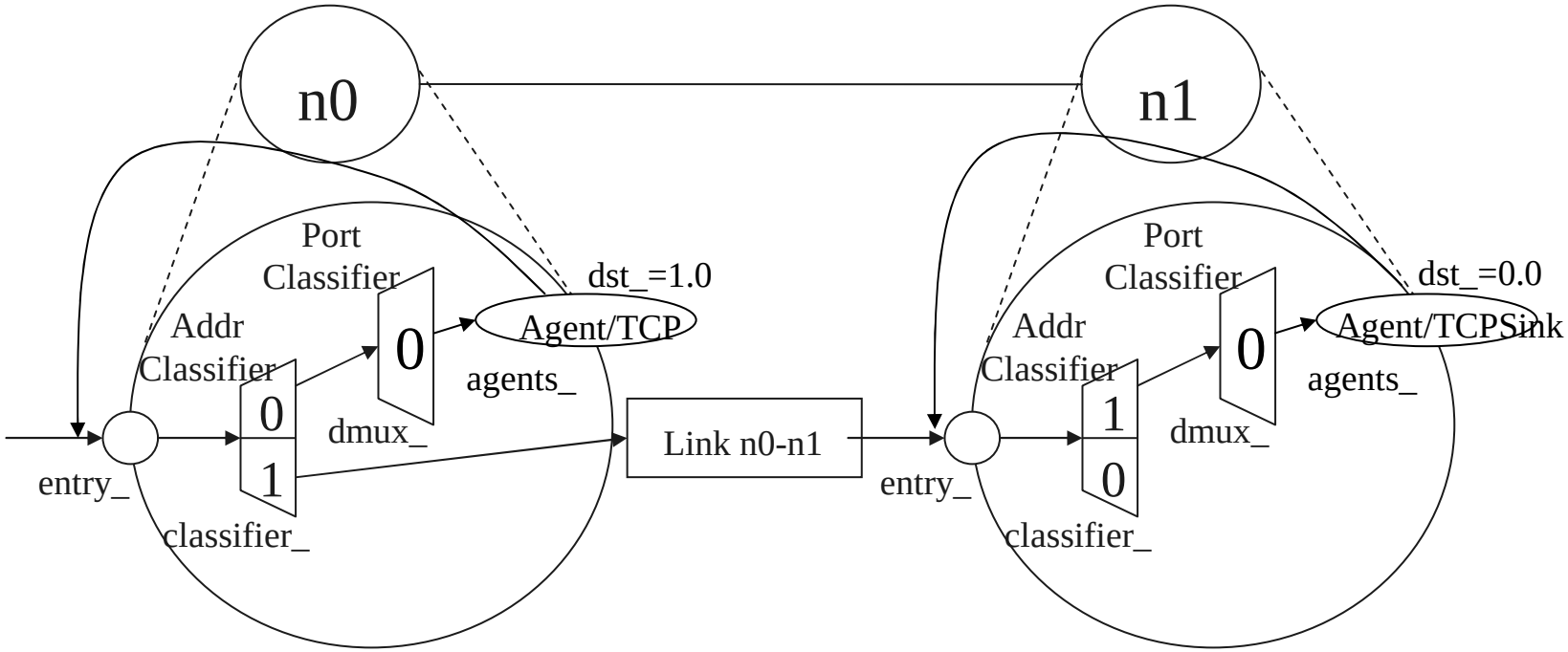
- Exponential

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

- Pareto on-off

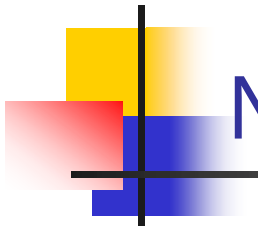
```
set src [new Application/
   Traffic/Pareto]
```

Creating connection and traffic



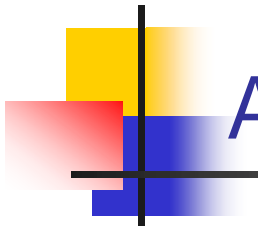

```
set tcp [new Agent/TCP]      set tcpsink [new Agent/TCPSink]
$ns attach-agent $n0 $tcp    $ns attach-agent $n1 $tcpsink

                $ns connect $tcp $tcpsink
```



Ns programming internal

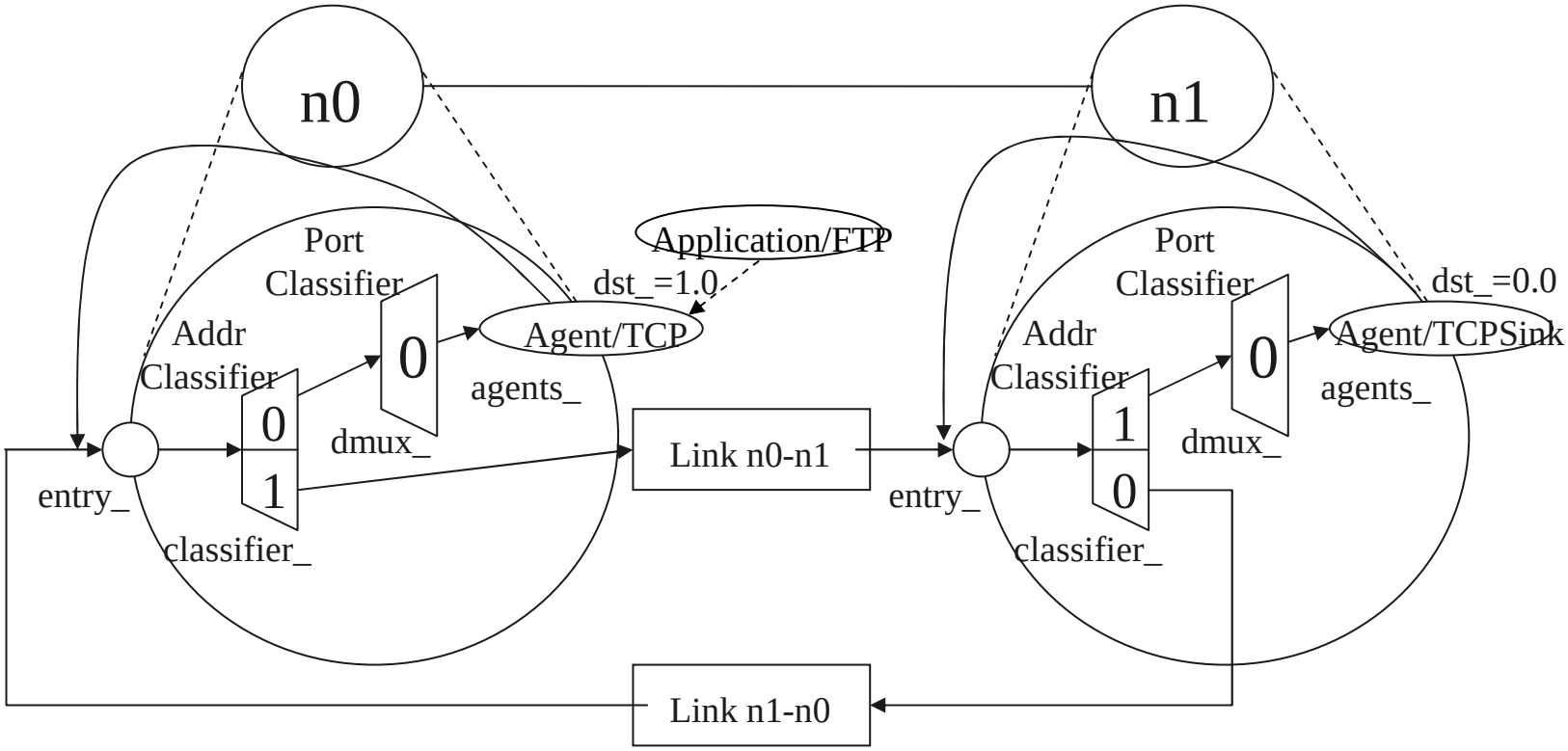
- Create the event scheduler
- Create network
- Turn on tracing
- Setup routing
- Create connection and traffic
- Transmit application-level data



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”
```

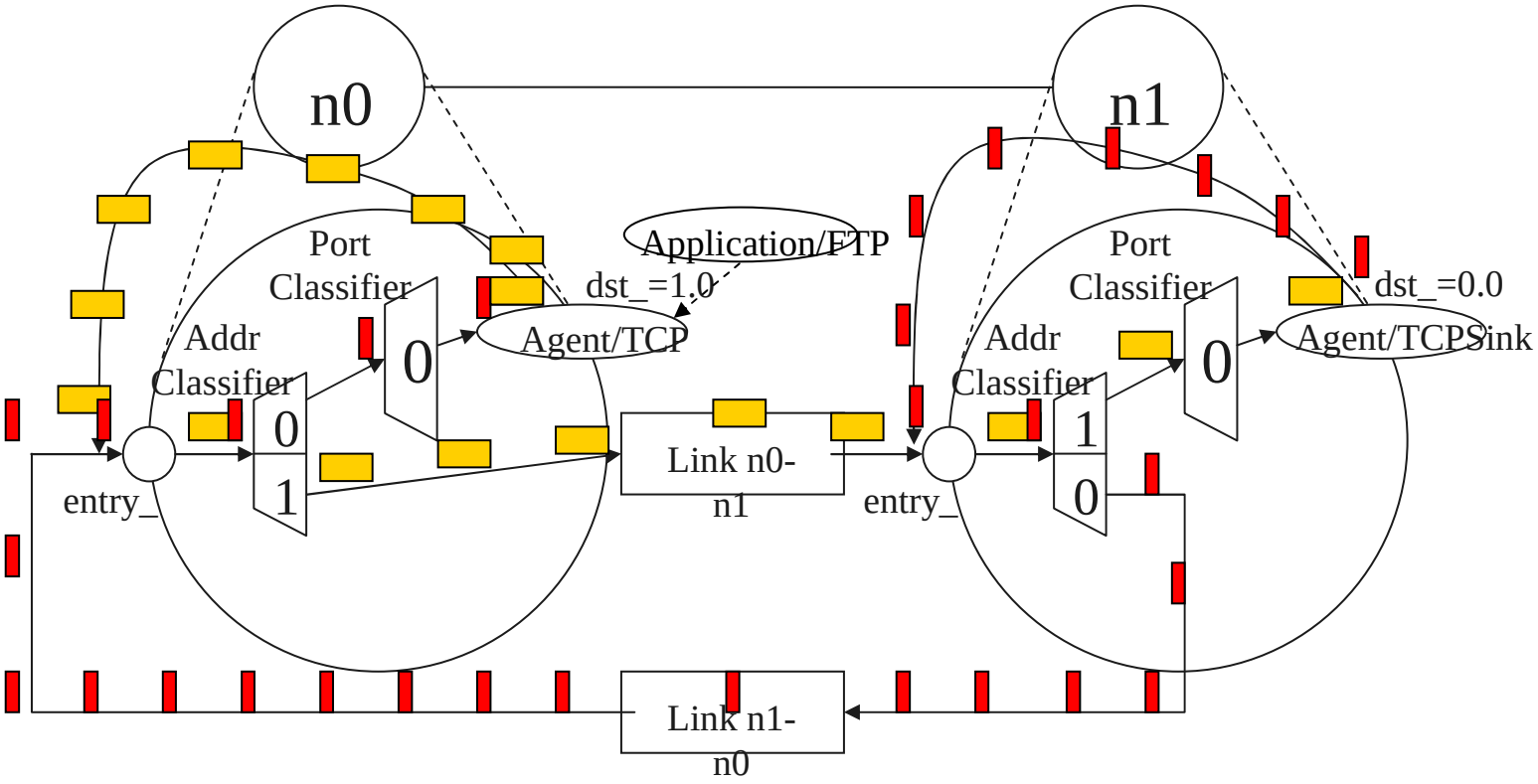
Creating traffic: trace driven

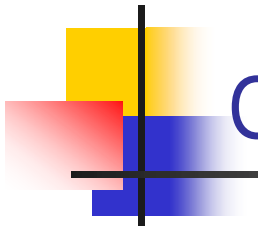
- Trace driven

```
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

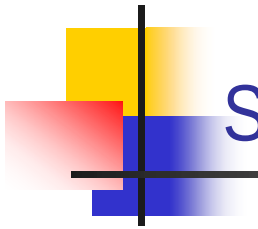

n0
n1
Port
Classifier
Addr
Classifier
0
1
0
Application/FTP
dst_=1.0
Agent/TCP
Link n0-
n1
entry_
Port
Classifier
Addr
Classifier
1
0
0
dst_=0.0
Agent/TCPSink
entry_
Link n1-
n0



Compare to real world

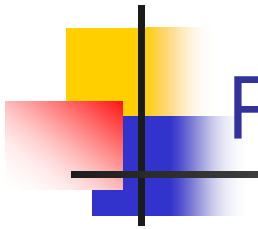
- More abstract (much simpler):
  - No IP addresses used, global variables is used
  - Nodes are connected directly rather than using name lookup/bind/listen/accept
- Easy to change implementation

```
Set tsrc2 [new agent/TCP/Newreno]
Set tsrc3 [new agent/TCP/Vegas]
```



Summary: generic script structure

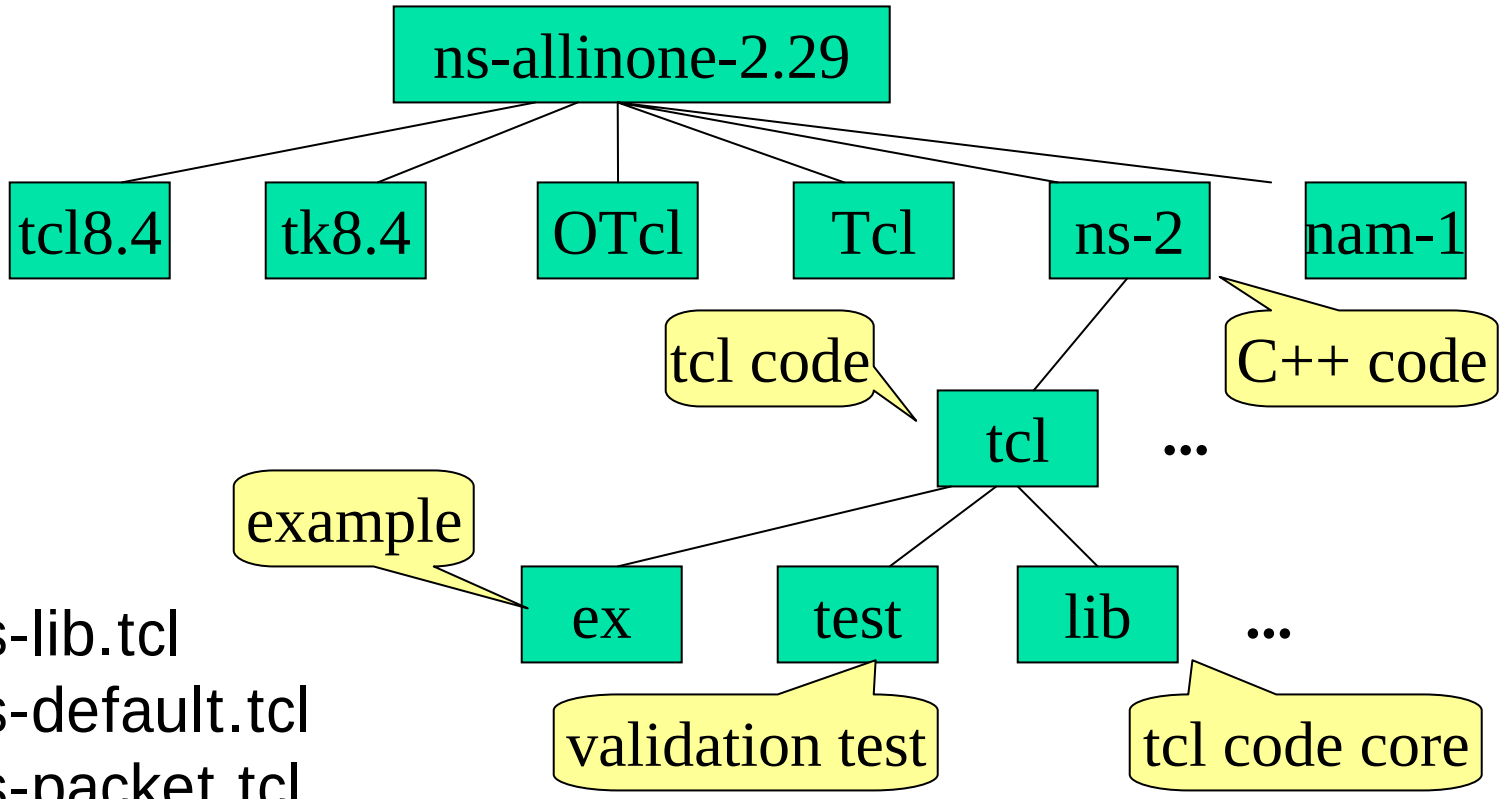
```
set ns [new Simulator]
# [Turn on tracing]
# Create topology
# Setup packet loss, link dynamics
# Create routing agents
# Create:
#   - multicast groups
#   - protocol agents
#   - application and/or setup traffic sources
# Post-processing procs
# Start simulation
```



Roadmap

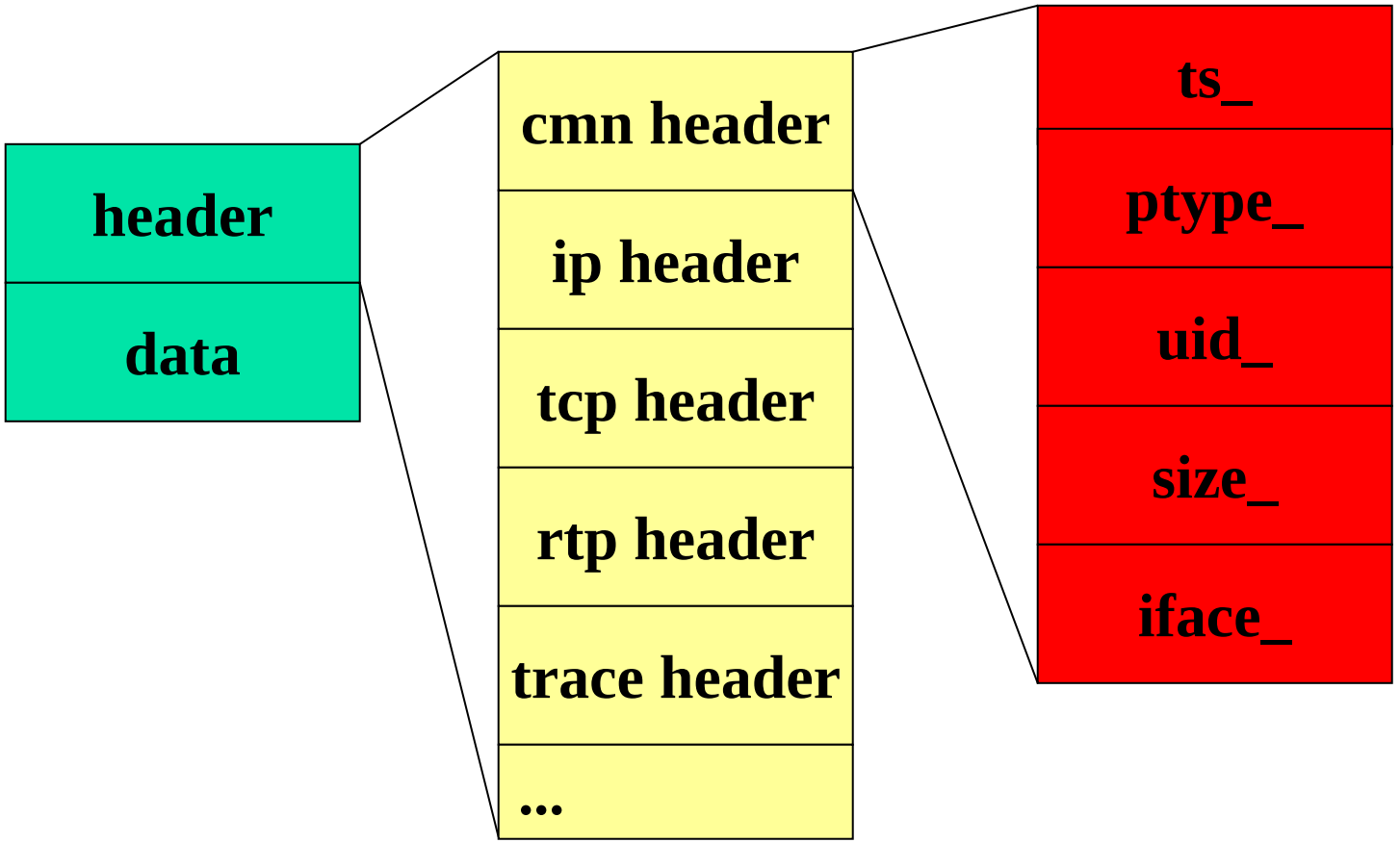
- Basic introduction
- Ns fundamentals
- Ns programming internal
- Extending ns-2 Simulator

ns-2 directory structure

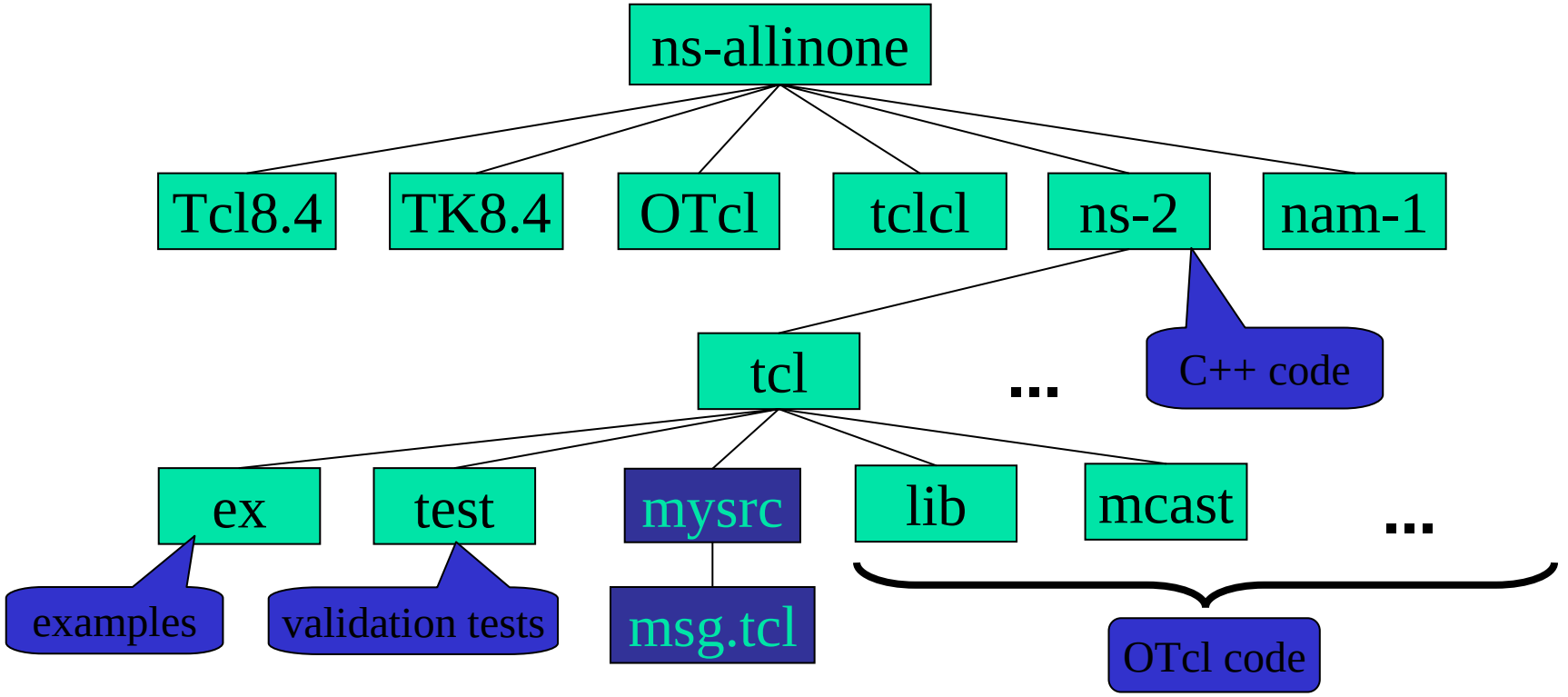


- ns-lib.tcl
- ns-default.tcl
- ns-packet.tcl

Packet format



Add your tcl changes into ns



Add your tcl changes into ns

- tcl/lib/ns-lib.tcl

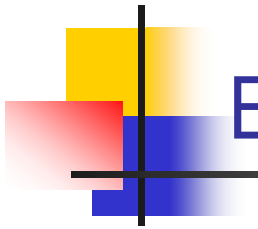
```
Class Simulator
…
source ../mysrc/msg.tcl
```

- Makefile

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

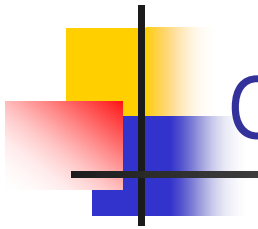
  - Or: change Makefile.in, `make distclean`, then `./configure --enable-debug`,

```
make depend && make
```



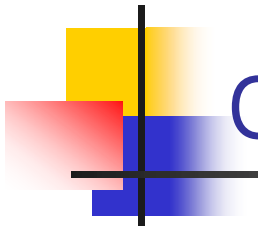
Extending ns in C++

- Modifying code
  - make depend
  - recompile
- Adding code in new files
  - change Makefile
  - make depend
  - recompile



Creating new components

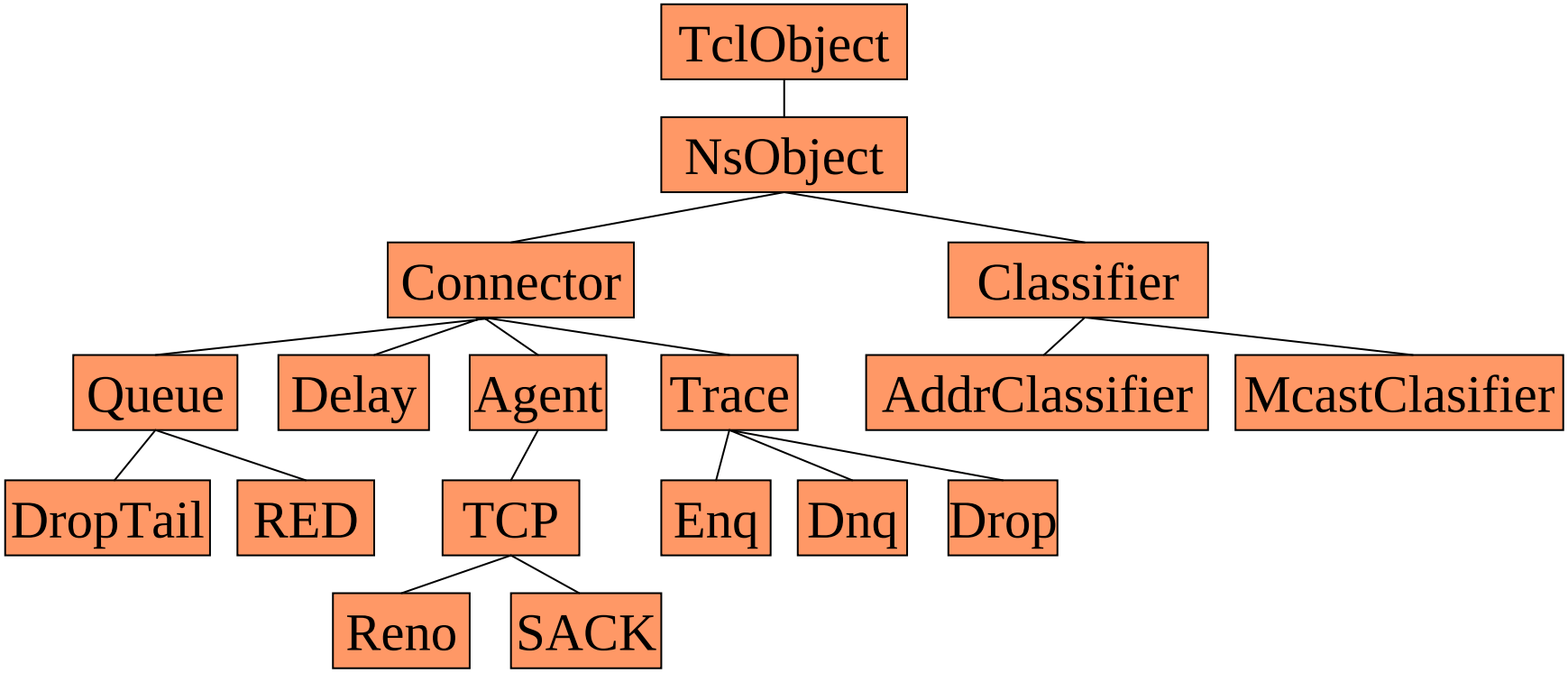
- Guidelines
- Inheritance Hierarchy
- C++ and OTcl Interface
- Debugging

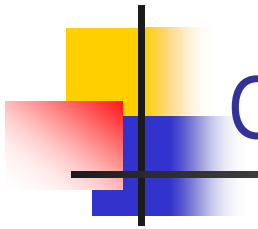


Guidelines

- 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

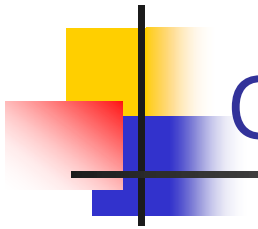
Class hierarchy (partial)





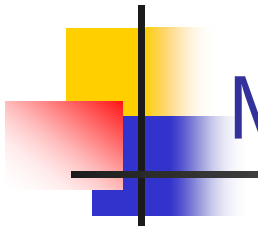
C++ and OTcl linkage

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



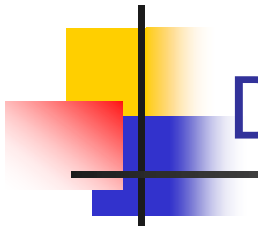
Object granularity tips

- Functionality
  - per-packet processing → C++
  - hooks, frequently changing code → OTcl
- Data management
  - complex/large data structure → C++
  - runtime configuration variables → OTcl



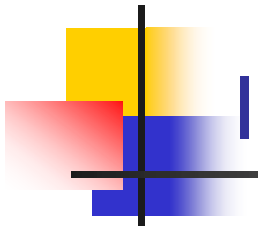
Memory conservation tips

- 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 tips for running large sim in ns at www.isi.edu/ns/nsnam/ns-largesim.html
- Not necessary until >100 nodes with complex models are used



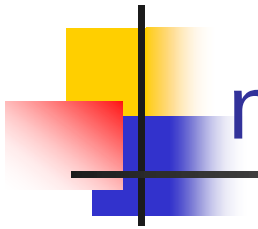
Debugging

- `printf()` in `C++` and `puts “”` in `Tcl`
- 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-like commands



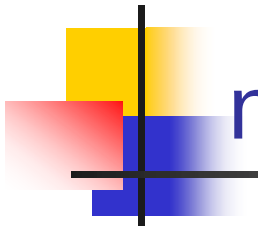
Implementation tips

- ns-2 TCP model in ns-2 does not allow payload in packets:
  - number of bytes are specified
  - a workaround is available at “NS by example”
- ns-2 UDP model allows payload inside the packet header:
  - UDP model is a good starting point for most protocol improvement projects
- Look for similar projects’ or modules’ code and modify:
  - starting from scratch is difficult and prone to errors
  - be very careful with pointers and dynamic arrays:
    - faults are hard to debug due to C++/OTcl duality



ns→nam interface

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



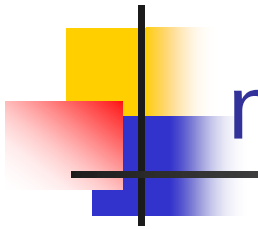
nam interface: color

- 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 interface: links

- Color

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

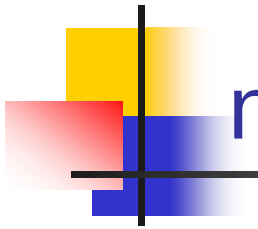
- Label

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

- Dynamics (automatically handled)

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

- Asymmetric links not allowed

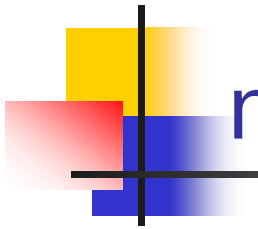


nam interface: topology

- “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 nodes are overlapped → use automatic layout



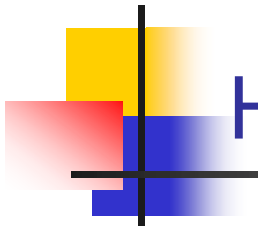
nam interface: miscellaneous

- 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
- NS by example: http://nile.wpi.edu/NS
- NS Simulator for beginners http://www-sop.inria.fr/maestro/personnel/Eltan.Altman/COURS-NS/n3.pdf