How to use NS2

Jae-Pil Yoo , Konkuk university
willow@konkuk.ac.kr

Contents

Part I : NS2 basics
- NS2 Overview
- Tcl, OTcl Overview
- Simple Simulation
- Post Simulation

Part II : NS2
- OTcl linkage
- OTcl linkage

Part III : Extending NS2
- Extending ns2 with OTcl
- Extending ns2 with C++

Part IV : NS2 component
- NS2 component

Part V : Wireless Simulation
- Ad hoc simulation
- Mobile IP simulation

Part I : NS2 Basics

1. NS2 overview
2. Tcl, OTcl overview
3. Starting up simple simulation

NS2 Overview

NS2

- Discrete event driven Object Oriented network simulator
  - Not a real time - serial sequence of events
- Focused on modeling network protocols
  - Wired, wireless, TCP, UDP, adhoc, web, …
  - Packet based
- Academic Project for over 10 years
  - freely distributed, open source
  - *"de facto"* standard in networking research

Usage?

- To evaluate the performance of existing network protocols
- To evaluate new network protocols before use
- To run large scale experiments not possible in real experiments
- To simulate a variety of IP networks

Pros & Cons

- Pros
  - Free & Open source
  - There are many contribution codes
- Cons
  - Tough for beginner
  - User-Interface is not friendly
  - Poor documents

History

- 1989: REAL by Keshav
- 1995: ns by Floyd, McCanne at LBL
- 1997: ns-2 by the VINT project (Virtual InterNetwork Testbed) at LBL, Xerox PARC, UCB, USC/ISI
  - Multi state collaboration

- Now: ns-2.31 maintained at USC/ISI
  - http://www.isi.edu/nsnam/ns/
  - Open source based, now in progress

Installation

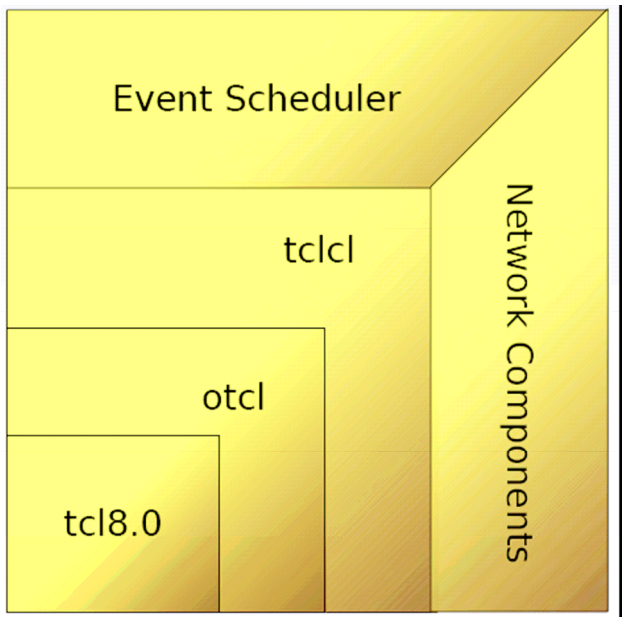
- http://www.isi.edu/nsnam/ns/ns-build.html
  - Getting the pieces
    - (tcl/tk8.0), otcl, tclcl, ns-2, (and nam-1)
  - or Getting all at once

- Work on most OS
  - Unix like system(Linux, BSD, Solaris…)
  - Windows 2000/XP/… via cygwin
    - http://wwwcygwin.com

- Mailing list: ns-users@isi.edu
  - "subscribe ns-users" in body

NS2 – Software Architecture

- NS2
  - OOP based discrete event driven network simulator written *in C++ & OTCL*
- Backend - C++
  - Protocols & Framework
- Frontend - Otcl
  - Configuration

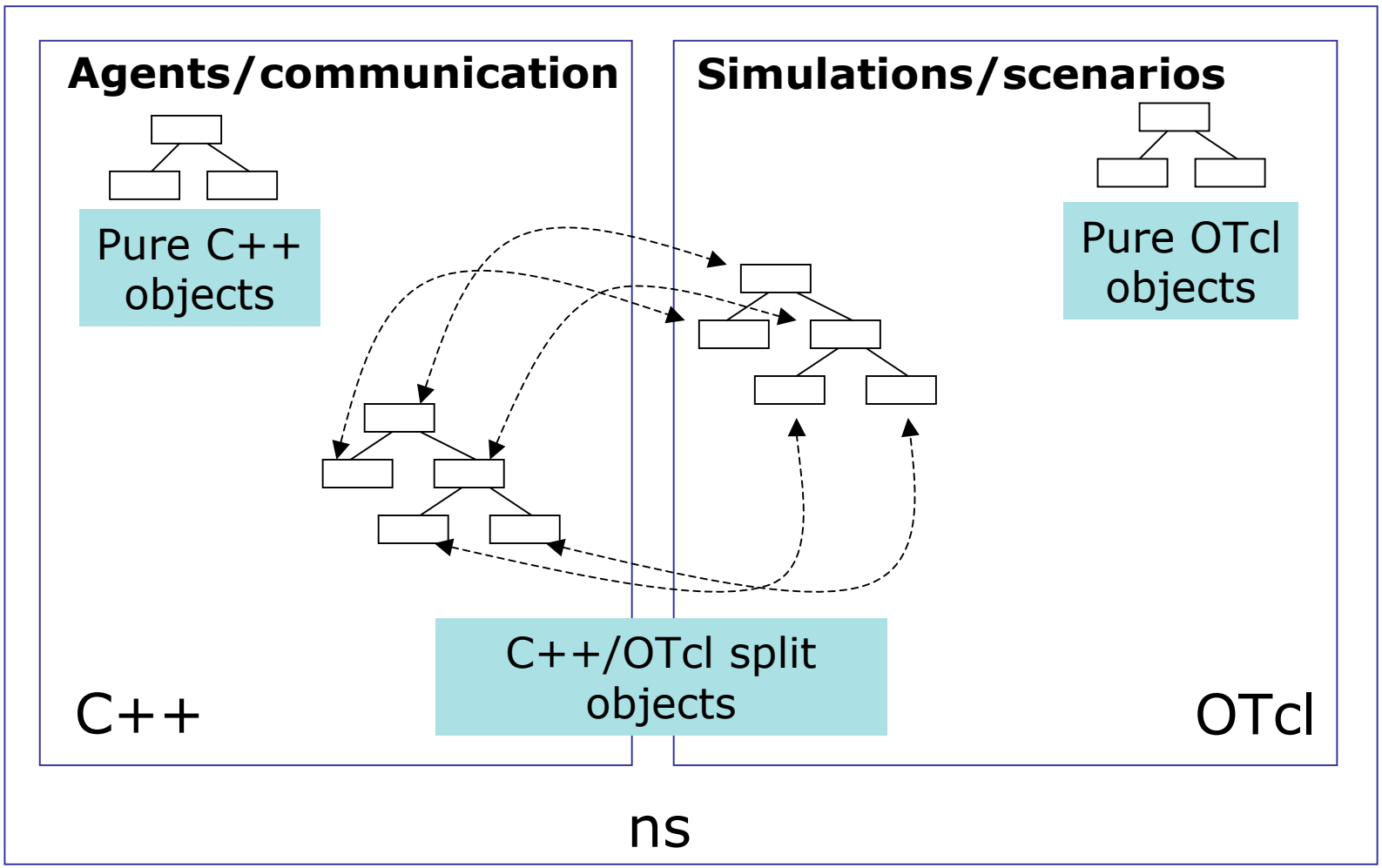


Why two languages?

- C++ for performance
  - algorithm, packet processing…
  - Fast to run, slower to change
  - Once made, less modified

- OTcl for network configuration
  - scenario configuration
  - fast to change, slow to run
  - Frequently modified

NS2 duality : C++ vs OTcl

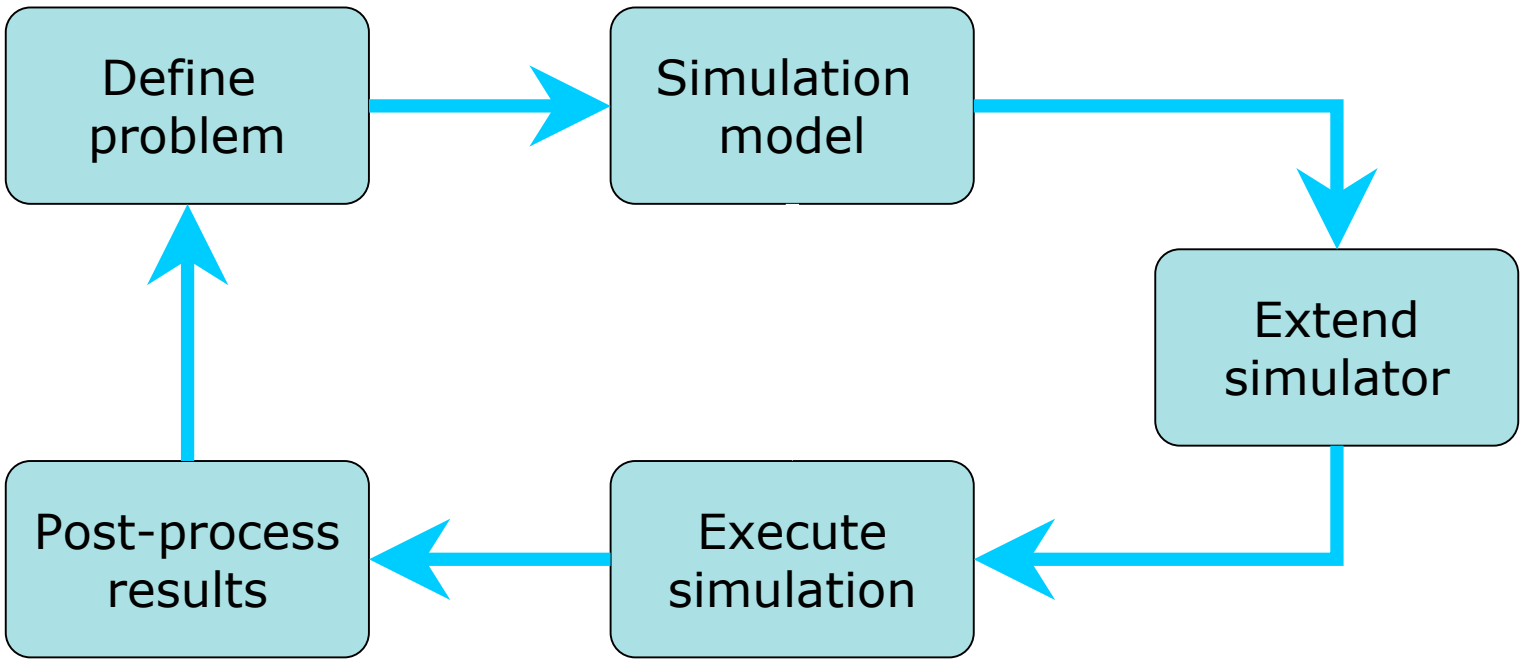


NS2 - applications

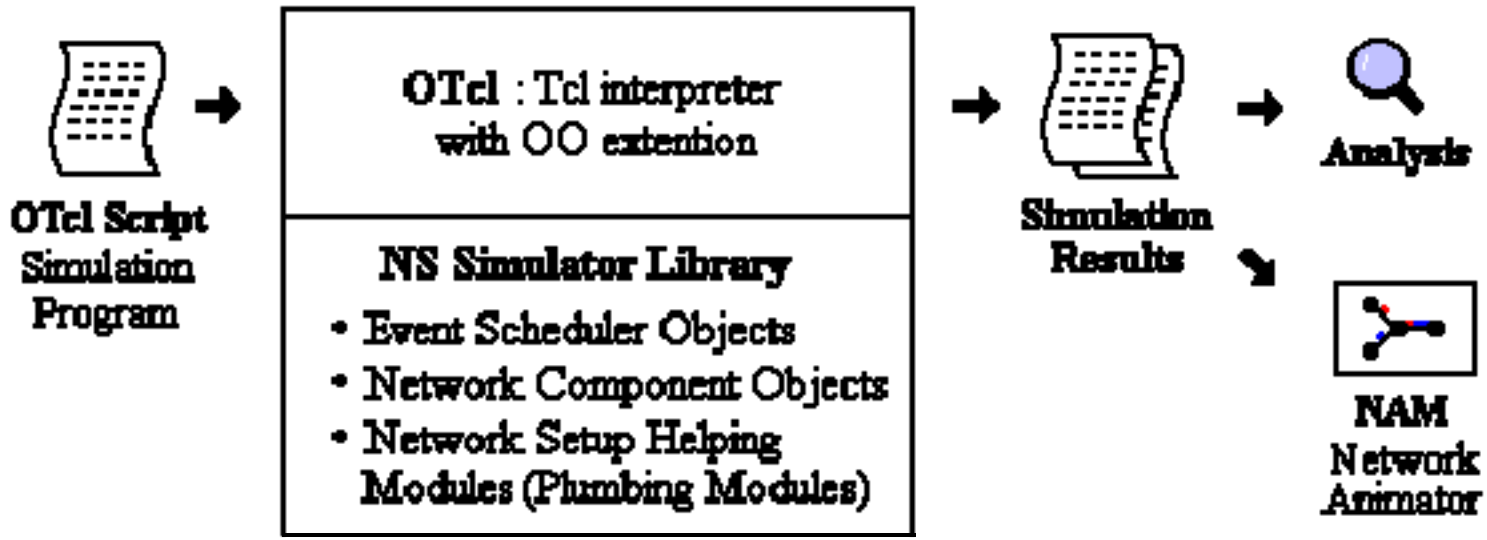
- Lots of TCP (and related) implementations
  - Mobility
  - Satellite nodes
  - Internet routing protocols
  - Various link loss models
  - Various traffic generators (e.g., web, telnet, ..)
  - Network emulation capability (real traffic traverses simulated net)
- Adding in progress
    - *Open source : contribution codes*

Steps when simulating

- Conception flow



Simplified User's View of NS



How to set up a simulation

- Write a OTcl code
  - Init a event scheduler
  - Set up network topology
  - Flow traffics from sender to receiver using scheduler by discrete time event
- Are you ready to 'OTcl'?
  - Do not need to understand deeply

Result of simulation ?

- NS-2 produces one or more text-based output files
  - Output files contain detailed simulation data
    - Trace full link information
- Post-processing
  - Results can be used in analysis
    - xgraph
  - Results also can be graphically displayed
    - Graphical simulation display tool called Network Animator (NAM)

TCL, OTCL Overview

Tcl Overview

- Tcl: Tool command language
  - in 1988 made by John K. Ousterhout
  - simple '*string based scripting language*' for controlling and extending applications
  - generic programming facilities
    - variables, loops, and procedures
- Tk: Graphical Toolkit extension of Tcl

- *Tcl/TK are excellent tools for* ***Rapid prototyping***

Basic command format

- # command arg1 arg2 arg3
  - one command in a line
    - Multiple commands in a line -> ‘;’
    - Continue at the next line -> ‘\
  - very similar with ‘csh”

```
% set a 5; puts “a = $a”
a = 5
% set \
a 5
5
```

Comment

- '#' - at the start of line
- ;#' – at the last of line

```
set a 22; set b 33          <- OK
# this is a comment         <- OK
set a 22  # same thing?     <- Wrong!
set a 22 ;# same thing      <- OK
```

Variable in Tcl

- No predefined type, default String
- Declaration - ‘set’ command
  - assign a value to a variable

```
% set a 2     ;# declaration int type var
2
% set a 1.2   ;# declaration float type var
1.2
% set a “This is string” ;# declaration string var
This is string

% set a       ;# check var value
This is string
       % unset a     ;# remove var
```

Data type in Tcl

- Basic data type – string
- Other data types
  - Array
    - Use string index ( )
  - List
    - Strings separated by spaces

List

- Zero or more elements separated by white space
- Braces for grouping
- Frequently used in ns

```
set list { apple orange banana }
=>apple orange banana

lappend list melon
=>apple orange banana melon

lindex $list 2
=>banana

lsort $list
=>apple banana melon orange
```

List - commands

- List-related commands
  - concat lindex llength lsearch
  - foreach linsert lrange lsort
  - lappend list lreplace

- Examples:

  lindex {a b {c d e} f} 2 c d e

  lsort {red green blue} blue green red
  - *Specially useful with eval and foreach.*

Array

- Format
  - %set color(a) red
  - %set color(b) blue
- Operation
  - %parray [arr_name]
  - %array exists [arr_name]
  - %array get [arr_name]
  - % array size [arr_name]
- Multi-dimensional array
  - name(index1, index2)

Math expression

- ‘expr’ & ‘incr’

```
set b 5                          5
incr b -1                        4
expr ($b*4) - 3                  17
expr $b <= 2                     0
expr $a * cos(2*$b)              -5.03443
expr {$b * [fac 4]}              120
set a Bill                       Bill
expr {$a < "Anne"}               0
```

Math expression

- -, ~, !: Unary minus, bitwise NOT, logical NOT
- * / % + -
- <<, >> : Left or Right shift
- < > <= >=
- == !=
- &, ^, |, &&, ||
- x?y:z
- acos(x): Arc-cosin of x
- asin(x): Arc-sine of x
- atan(x): Arc-tangent of x
- atan2(y, x): Rectangular (x, y) to polar (r, th). atan 2 gives th.
- ceil(x): Ceiling of x
- cos(x): Cosine of x
- cosh(x): Hyperbolic cosine of x
- exp(x): Exponential
- floor(x): Floor of x
- fmod(x, y): Floating point remainder of x/y
- hypot(x, y): sqrt(x*x + y*y)
- log(x): Natural log of x
- log10(x): Log base 10 of x
- pow(x, y): x to the y power
- sin(x): Sine of x
- sinh(x): Hyperbolic sine of x
- sqrt(x): Square root of x
- tan(x): Tangent of x
- tanh(x): Hyperbolic tangent of x
- abs(x): Absolute value of x
- double(x): x의 실수형 값
- int(x): x의 정수형 값
- round(x): x를 내림한 정수값

Substitutions and Quoting

- Substitutions
  - Evaluate script, substitute result.

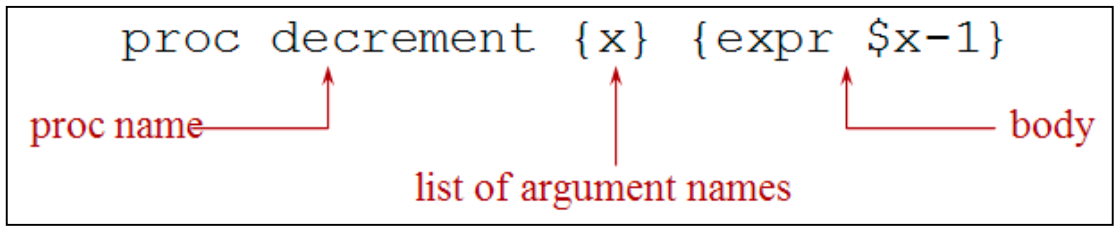
```
set a 47                          47
set b $a                    47
set b [expr $a+10]          57
```

- Quoting
  - Grouping elements

```
set b "a is $a"             a is 47
set b {[expr $a+10]}        [expr $a+10]
```

Procedure

- User can define procedure




```
set foo 3
=>3
proc add {a b} {
   global foo
   return [expr $a + $b + $foo]
}
add 1 2
=>6
```

  - declare first, use later

Display

```
% set variable 255

% puts "The number $variable"
The number 255

% puts [format "The number %d is equal to
0x%02X" \
      $variable $variable]
The number 255 is equal to 0xFF
```

- d: decimal int
- o: unsigned octal
- s: string
- u: unsigned int
- x or X: unsigned hex
- f: float
- i: int
- c: ASCII
- e E: exponential

Control statement-IF

- Format

```
if { $x == 0 } {
    puts “Zero Value”
} elseif { $x > 0 } {
    puts “Plus Value”
} else {
    puts “Nonzero Value”
}
```

- Operators
  - Same with ‘C’ language
    - !, <, >, <=, >=,!=,==,||,&&

Control statement-FOR

- 'For' example with 'continue' & 'break'

```
for {set i 1} {$i <= 3} {incr i} {
   puts $i
}
for {set i 1} {$i <= 5} {incr i} {
   if {$i < 3} { continue }
   puts $i
}
for {set i 1} {$i <= 5} {incr i} {
   if {$i > 3} { break }
   puts $i
}
```

Control statement-WHILE

```
set i 3

while {$i != 0} {
   puts stdout $i
   incr i -1
}
```

- break and continue
- No ‘goto’ keyword

Control statement-FOREACH

- Iteration of list values
  - Effective for args processing in NS

```
foreach i {A B C} {
   puts $i
}
=>A
=>B
=>C
foreach {i j} {A B C D E F} {
   puts "$i $j"
}
=> A B
=> C D
=> E F
```

Variable Arguments

- args – list of arguments separated by

```
proc sum {args} {
   set s 0
   foreach i $args {
      set s [expr $s + $i]
   }
   return $s
}
sum 1 2
=>3
sum 1 2 3
=> 6
```

String

- Commands
  - string compare str1 str2
  - string first str1 str2
  - string index str I
  - string last str1 str2
  - string length str
  - string match pattern str
  - string range str i j
  - string tolower str
  - string toupper str
  - string trim str ?chars?
  - string trimleft str ?chars?
  - string trimright str ?chars?
  - string wordend str ix
  - string wordstart str ix

File IO – OPEN

- Format
  - #set fd [open filename r]
    - - r: read
    - - r+: read & write.
    - - w: write(overwrite).
    - - w+: read & write
    - - a: write, append
    - - a+: read, write, append

File IO – WRITE

flush : instant writing

```
set fd [open MyFileName w]
puts $fd “This line will be printed in MyFileName”
flush $fd
close $fd
puts “This line will be printed in screen”
```

File IO - READ

- Format
  - #gets $fd line
    - read a single line, store to $line
  - #read $fd 5
    - Read 5 bytes and return

```
set input [open /etc/passwd r]
while { [gets $input line] >= 0 } {
    puts $line
}
close $input

set input [open /etc/passwd r]
puts [read $input]
close $input
```

Eval

- Concatenate arguments and evaluate them as a command

```
puts stdout "Hello, World!"

set cmd {puts stdout "Hello, World!"}
=> puts stdout "Hello, World!"

# sometime later...
eval $cmd
=> Hello, World!
```

Special Commands

- exec – execute external command
  - #exec grep -n hoge *.c | wc >& log.txt &
- exit – exit Tcl environment

OTcl

- MIT Object TCL
  - Object oriented extension to Tcl
- Documents
  - ftp://ftp.tns.lcs.mit.edu/pub/otcl/doc/tutorial.html
- OTcl in ns
  - Pumbling network components
  - Front-end for manipulating C++ objects

Comparison with C++

C++	OTcl
Single class decl	Attaches method to object or class
Constructor/destructor	init{}, destroy{}
This pointer	$self
Shadowed methods called explicitly with scope operator	Combiled implicitly with '$self next'

Creating a Class

- Creating a Class

```
Class mom
mom instproc greet {} {
      $self instvar age_
      puts "$age_ years old mom: How are
you doing?"
}
```

  - 'instproc'
    - declares instance procedure of the class
  - 'instvar'
    - declares instance variable of the class (only after object '$self')

Creating a Class

- Creating a child class
  - using `-superclass'

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

Override
mom::greet{}

Getting an instance

```
# Create a mom and a kid object set each age
set a [new mom]
$a set age_ 45
set b [new kid]
$b set age_ 15

# Calling member function "greet" of each object

$a greet
45 age_ years old mom: How are you doing

$b greet
15 age_ years old kid: What's up, dude
```

Constructor

- init {} procedure

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

Kid instproc init {} {
      $self next
      $self set age_ 15
}
```

- detroy {} procedure

Starting up simple simulation

Interactive mode

```
# ns
% set ns [new Simulator]
_o3
% $ns at 1 “puts \“Hello World!\””
1
% $ns at 1.5 “exit”
2
% $ns run
Hello World!
#
```

Batch mode

simple.tcl

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

```
# ns simple.tcl
Hello World!
```

Generic Script Structure

1. Creating the event scheduler
2. Creating network (topology)
   - Creating node, link, connection
3. Creating agent & traffic
   - Creating agents, traffic source
   - Inserting errors
4. Tracing

Plumbing

- Modular Approach
  - Fine-grain object decomposition
  - Positives
    - Composible, re-usable
  - Negatives
    - Must ‘Plumb’ in OTcl
    - Developer must be comfortable with both environments, tools
  - Fairly steep learning curve

Creating Event Scheduler

- Create scheduler
  - set ns [new Simulator]
- Schedule event
  - $ns at <time> <event>
- Start scheduler
  - $ns run

  - simulator has list of events ordered by time
  - process: first scheduled, first dispatched

Creating Network

- Nodes
  - "hardware entities" in a network
    - set n0 [$ns node]
    - set n1 [$ns node]
- Links & Queuing
  - Link : connects the nodes in the network
    - $ns duplex-link $n0 $n1 <bw> <delay> <q_type>
    - <q_type>: DropTail, RED, CBQ, FQ, SFQ, DRR

Creating Connection/UDP

- Agent
  - “Software entities” which are on these nodes
  - End-point where network layer packets are constructed or consumed
    - UDP , TCP
- 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

Creating Connection/TCP

- TCP
  - # originator of the packets
  - set tcp [new Agent/TCP]

  - # destination of the packets
  - set tcpsink [new Agent/TCPSink]

  - $ns attach-agent $n0 $tcp
  - $ns attach-agent $n1 $tcpsink
  - $ns connect $tcp $tcpsink

Creating Traffic/TCP

- FTP
  - set ftp [new Application/FTP]
  - $ftp attach-agent $tcp
- Telnet
  - set telnet [new Application/Telnet]
  - $telnet attach-agent $tcp

Creating Traffic/UDP

- CBR
  - set src [new Application/Traffic/CBR]
- Exponential or Pareto on-off
  - set src [new Application/Traffic/Exponential]
  - set src [new Application/Traffic/Pareto]

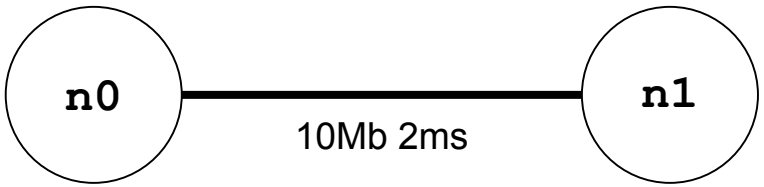
Create Scheduler

scheduler

```
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 10Mb 2ms DropTail
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
$ns connect $udp0 $null0
$ns at 1.0 "$cbr0 start"
$ns run
```

Create network topology

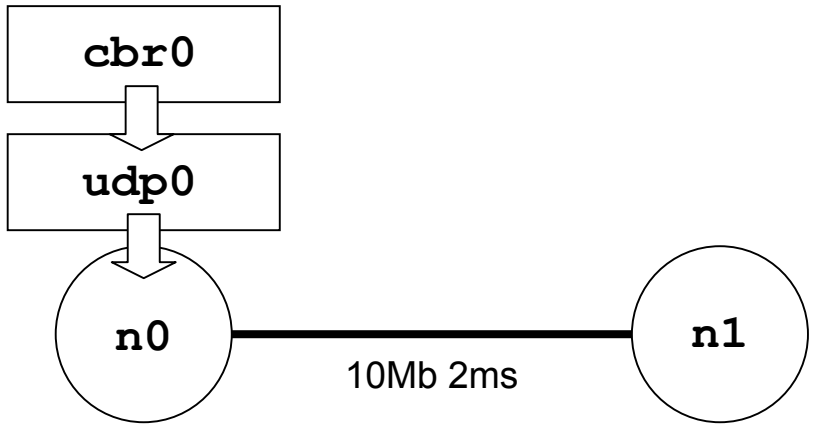
scheduler




```
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 10Mb 2ms DropTail
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
$ns connect $udp0 $null0
$ns at 1.0 "$cbr0 start"
$ns run
```

Stack up Agent and App. for sender

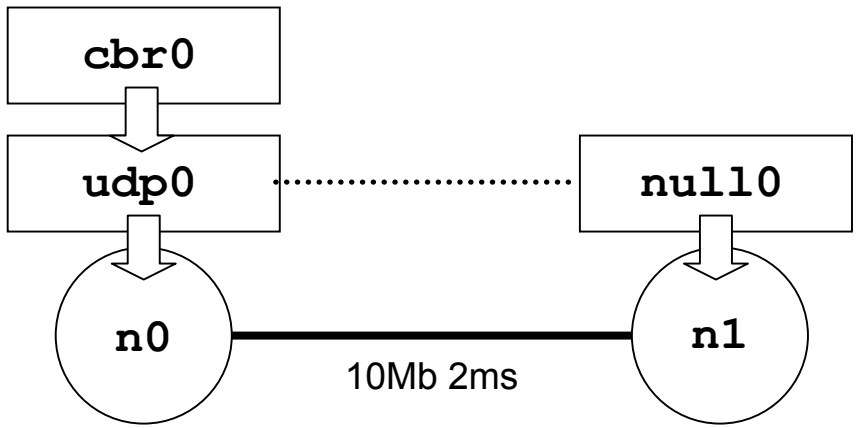
scheduler




```
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 10Mb 2ms DropTail
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
$ns connect $udp0 $null0
$ns at 1.0 "$cbr0 start"
$ns run
```

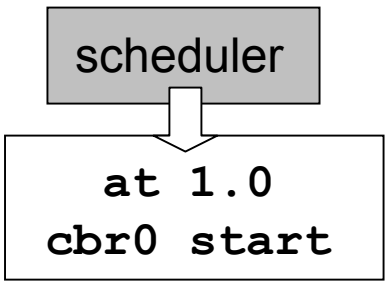
Stack up Agent for receiver and make a connection

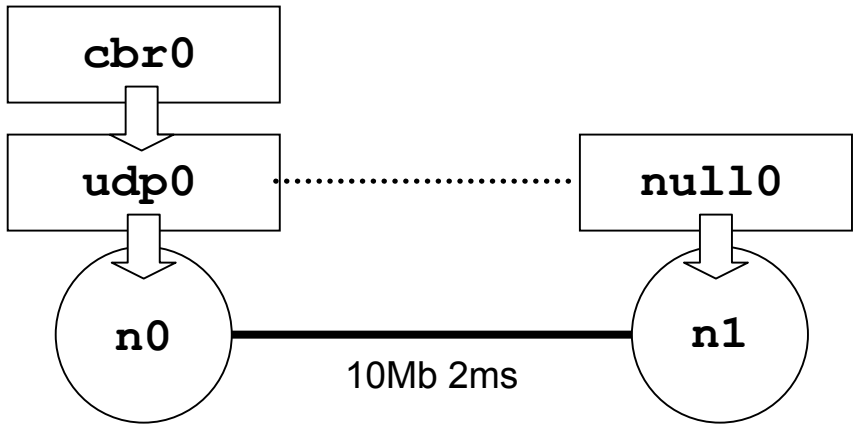
scheduler




```
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 10Mb 2ms DropTail
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
$ns connect $udp0 $null0
$ns at 1.0 "$cbr0 start"
$ns run
```

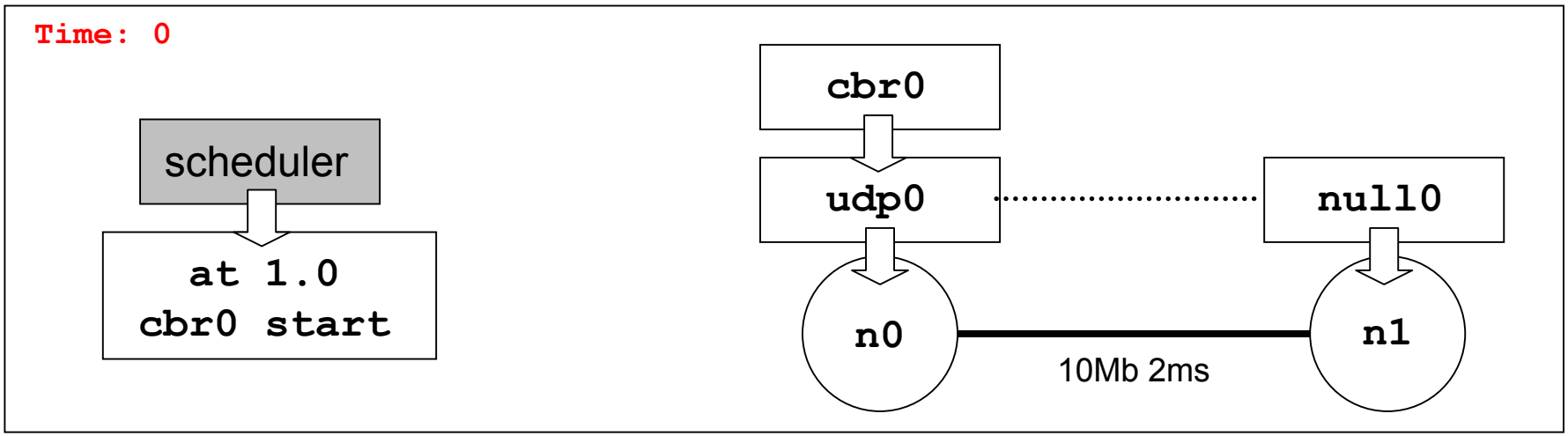
Schedule event and run



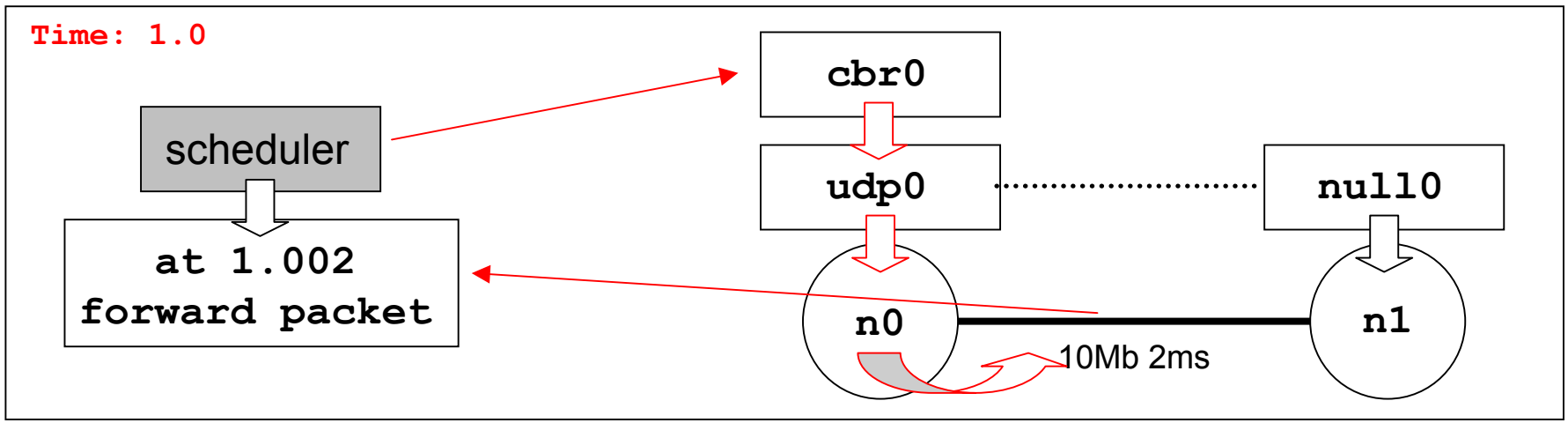


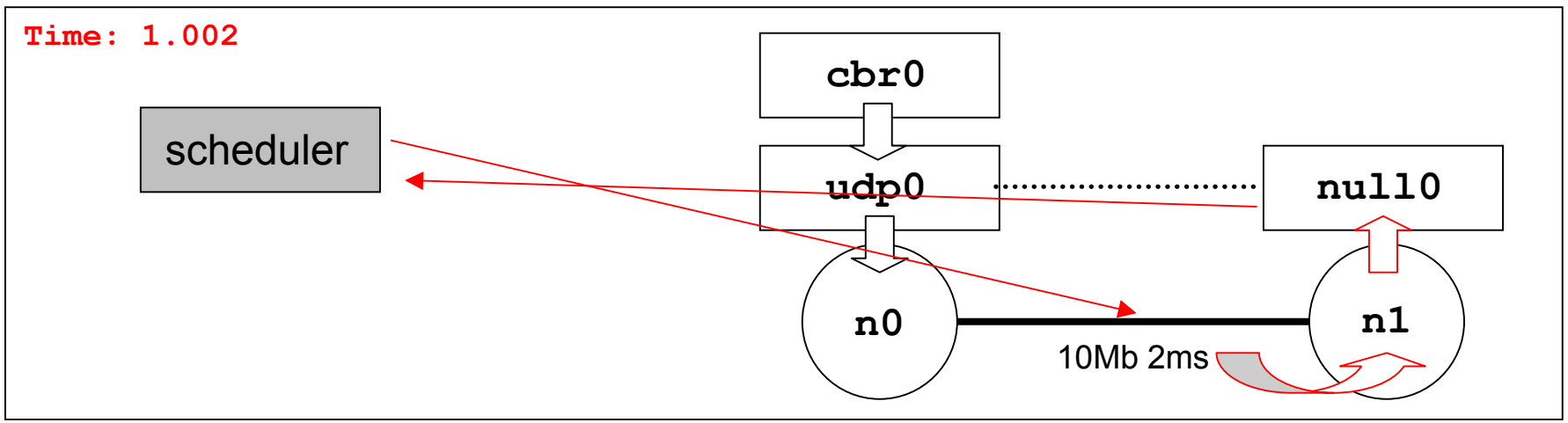

```
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 10Mb 2ms DropTail
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
$ns connect $udp0 $null0
$ns at 1.0 "$cbr0 start"
$ns run
```

Time Event : Starting sim

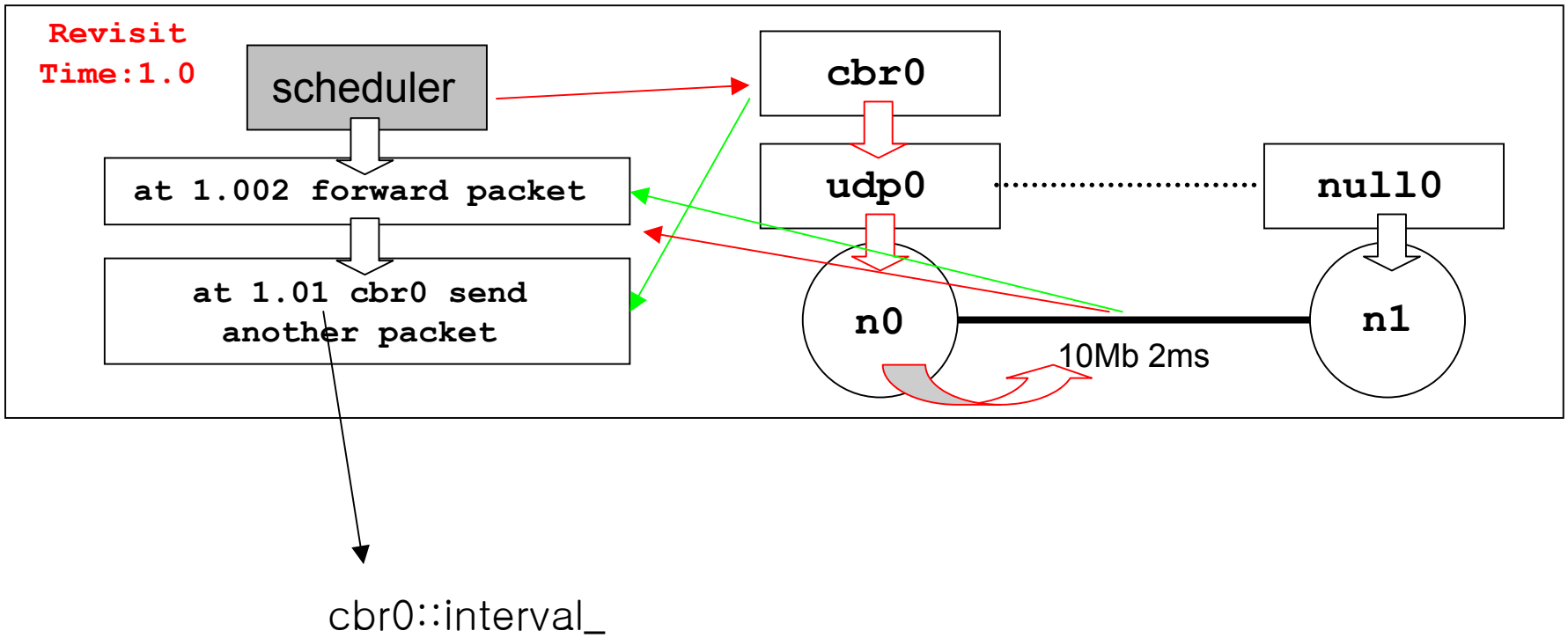


Time Event : forwarding packet





Time Event


Revisit
Time:1.0
scheduler
at 1.002 forward packet
at 1.01 cbr0 send
another packet
cbr0
udp0
null0
n0
n1
10Mb 2ms
cbr0::interval_

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
```

Post Simulation

All packets trace

- Trace *packets on all links*
  - $ns trace-all [open test.out w]

```
<event> <time> <from> <to> <pkt> <size>--<flowid> <src> <dst> <seqno> <aseqno>
    + 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
```

- Trace packets on all links in nam animator format
  - $ns namtrace-all [open test.nam w]

Var. trace-Periodic Probing

- Printing 'trace var' periodically
  - Self calling in OTcl procedure
  - must be OTcl reachable variable

```
proc probe {} {
        global ns tcp0
        set now [$ns now]
        set cwnd [$tcp0 set cwnd_ ]
        puts "congestion window = $cwnd (time = $now)"
        $ns at [expr $now + 1] "probe"
}

n $ns at 1.001 "probe"
```

Var. trace-class Tracer

- Automatically record whenever the value of traced variable changes
  - must be OTcl reachable variables
  - Variable should be registered to trace class

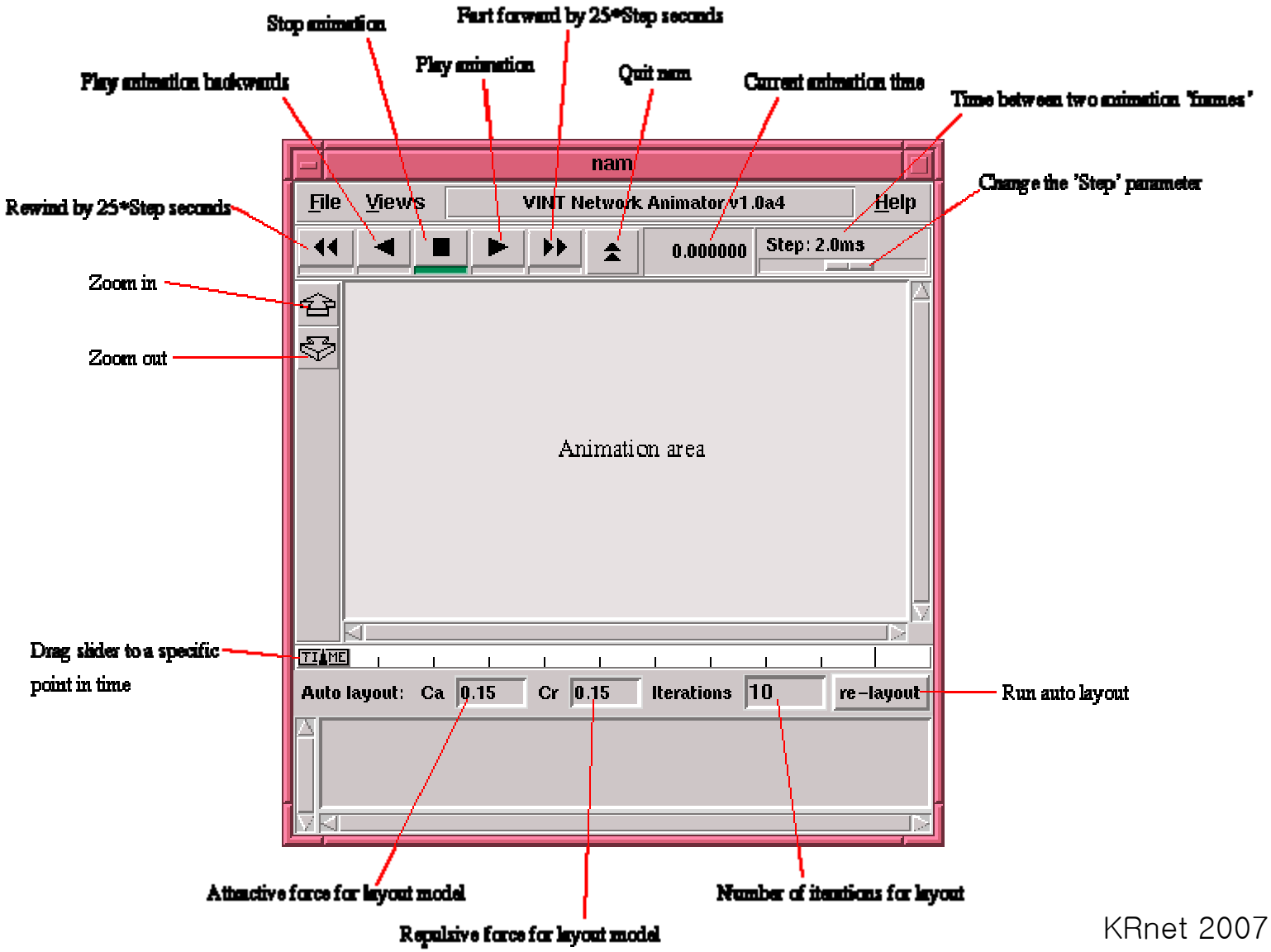
```
set tf [open "cwnd.tr" w]
set tracer [new Trace/Var]
$tracer attach $tf
$tcp trace cwnd_ $tracer

% tail cwnd.tr
… …
f t4.9912 a_o28 ncwnd_ v9.78154
f t4.99952 a_o28 ncwnd_ v9.88378
```

Post Simulation-Trace Analyze

- Xgraph
  - Display simulation result
  - General purpose x-y data plotter
  - expects data in an x y format
- raw2xg
  - Convert ns trace to xgraph format
- NAM (Network Animator)
  - Display simulation on-going progress
  - Packet-level animation
  - Well supported by ns

Xgraph

NAM animator



OTcl Script for NAM

- Tcl scripts for NAM
  - Color
  - Node manipulation
  - Link manipulation
  - Topology layout
  - Protocol state
  - Misc
- Do not effect simulation result!

NAM Interface: Color

- Used for distinguishing each traffic
  - 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
  - $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"

NS Demos

- Routing
- TCP
- RTP
- Multicast
- LAN
- Web

Part II : OTcl linkage

OTcl linkage

OTcl linkage

Split language architecture

- NS2 is OOP simulator
  - Written in C++ with Otcl interpreter as a frontend
- Dual Class Hierarchy closely related to each other
  - Compiled hierarchy(C++ space)
  - Interpreted hierarchy(OTcl space)
- The root of the hierarchy
  - TclObject class
- *ns* provides glue(via tclcl)
  - make objects and variables appear on both languages

Which language for what?

- Basic advice
  - OTcl
    - configuration, setup, and "one-time" stuff
    - Some works done by manipulating existing C++ objects
  - C++
    - packet processing
    - changing an existing C++ class

Creating pure OTcl class

- Add a new OTcl component
  - Script a <new_stuff.tcl> component
- Let the NS binary to understand the new OTcl class whenever it starts
  - Source <new_stuff.tcl> file in $ns/tcl/lib/ns-lib.tcl
  - Add <new_stuff.tcl> in NS_TCL_LIB in $ns/Makefile
  - recompile

Scripting a OTcl class

- %cat pure_otcl.tcl

```
Class PureOtcl

PureOtcl instproc init {} {
   puts "pure Otcl class is created"
}

PureOtcl instproc add { a b } {
   $self instvar result
   set result [expr $a+$b]
   puts "add $a + $b = $result"
}
```

Compile procedure

- Sourcing the script
  - Move pure_otcl.tcl to $ns/tcl/lib
  - ‘source pure_otcl.tcl’ in $ns/tcl/lib/ns-lib.tcl
- Compile
  - ‘NS_TCL_LIB= \ tcl/lib/pure_otcl.tcl’ in $ns/Makefile
  - #cd $ns; make

Running example

- #set a [new PureOtcl]
  pure Otcl class is created

- #$a add 1 6
  add 1 + 5 = 6

Creating C++ class

- C++ for performance
  - Fast to run, slower to change
  - Once made, less modified
    - algorithm, packet processing
- Frontend is Otcl environment
  - Some connection methods are needed between C++ and OTcl
  - Otcl linkage

How to connect ?

- OTcl space

```
set cbr0 [new Application/Traffic/CBR]
…
$cbr0 set rate 1000000
```

- C++ space

```
class CBR_Traffic : public TrafficGenerator{
public:
   CBR_Traffic();
protected:
   double rate;
   …
}
```

C++ component structure

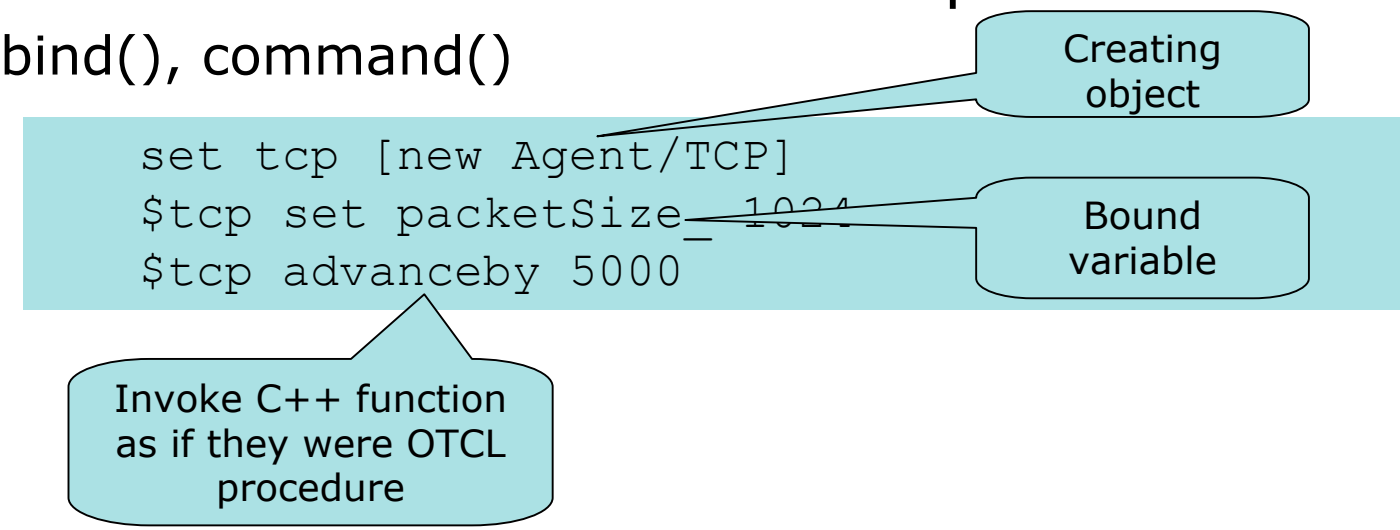
```
[Class definition] - TclObject
```

```
[Define TclClass] - TclClass
     ; create 'class TclObject'!
```

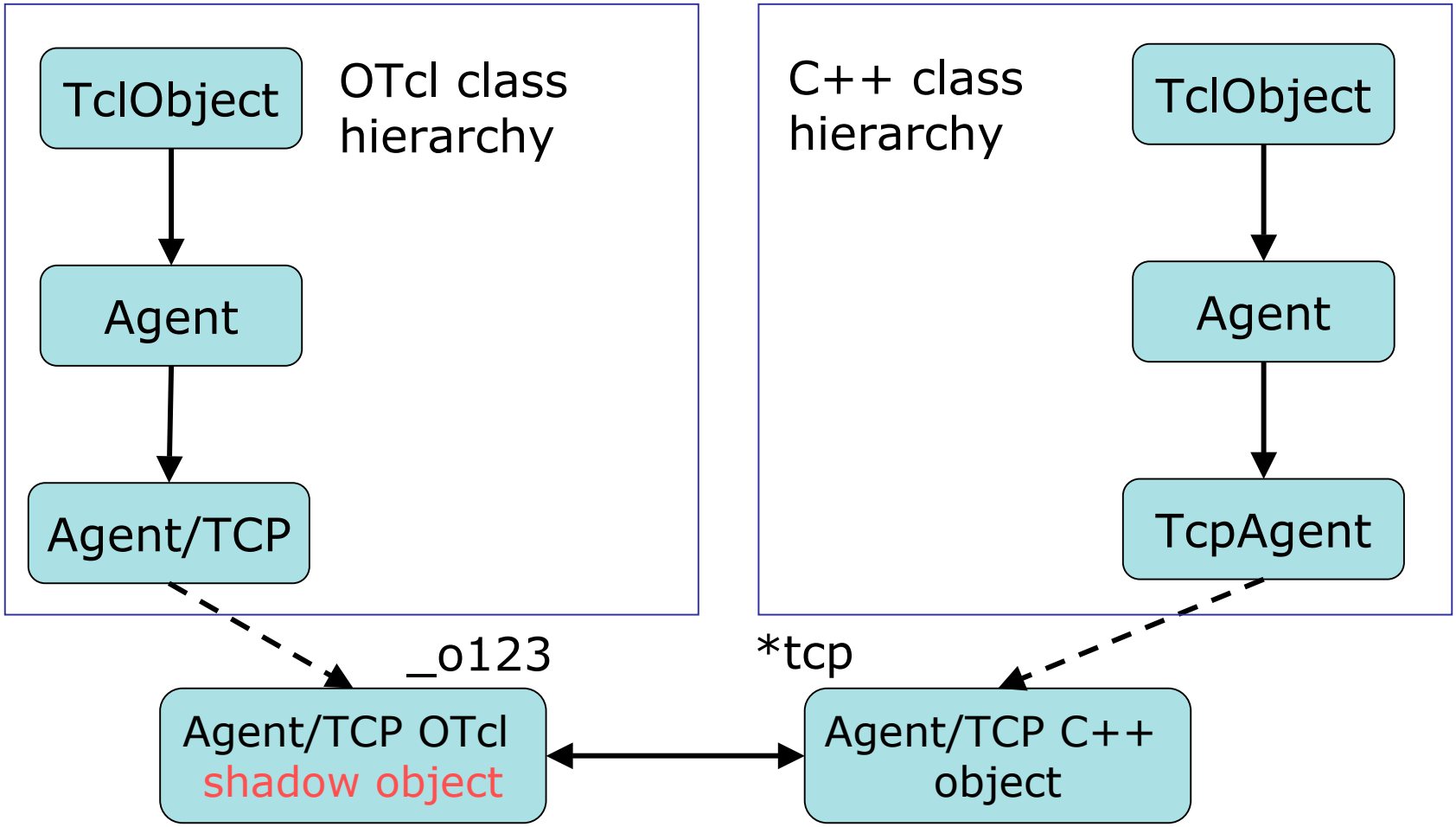
```
[Class body]
     bind() ; link otcl-c++ variables
     command() ; link otcl-C++ functions
```

Class TclObject

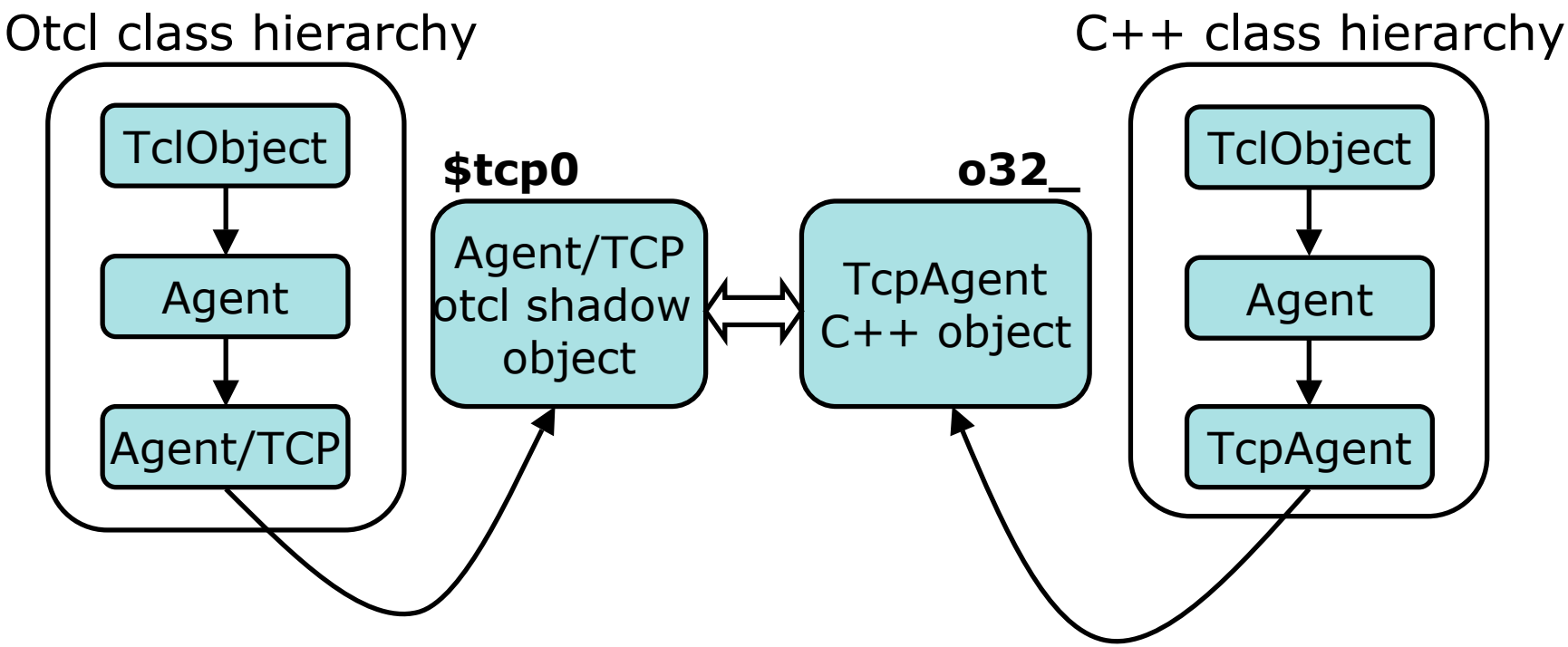
- Unified OTcl and C++ class hierarchies
  - base class in the interpreted & compiled hierarchies
- Mirrored in both C++ and Otcl
  - 'TclClass' contains the mechanisms that perform this shadowing.
- Seamless access between two space
  - bind(), command()



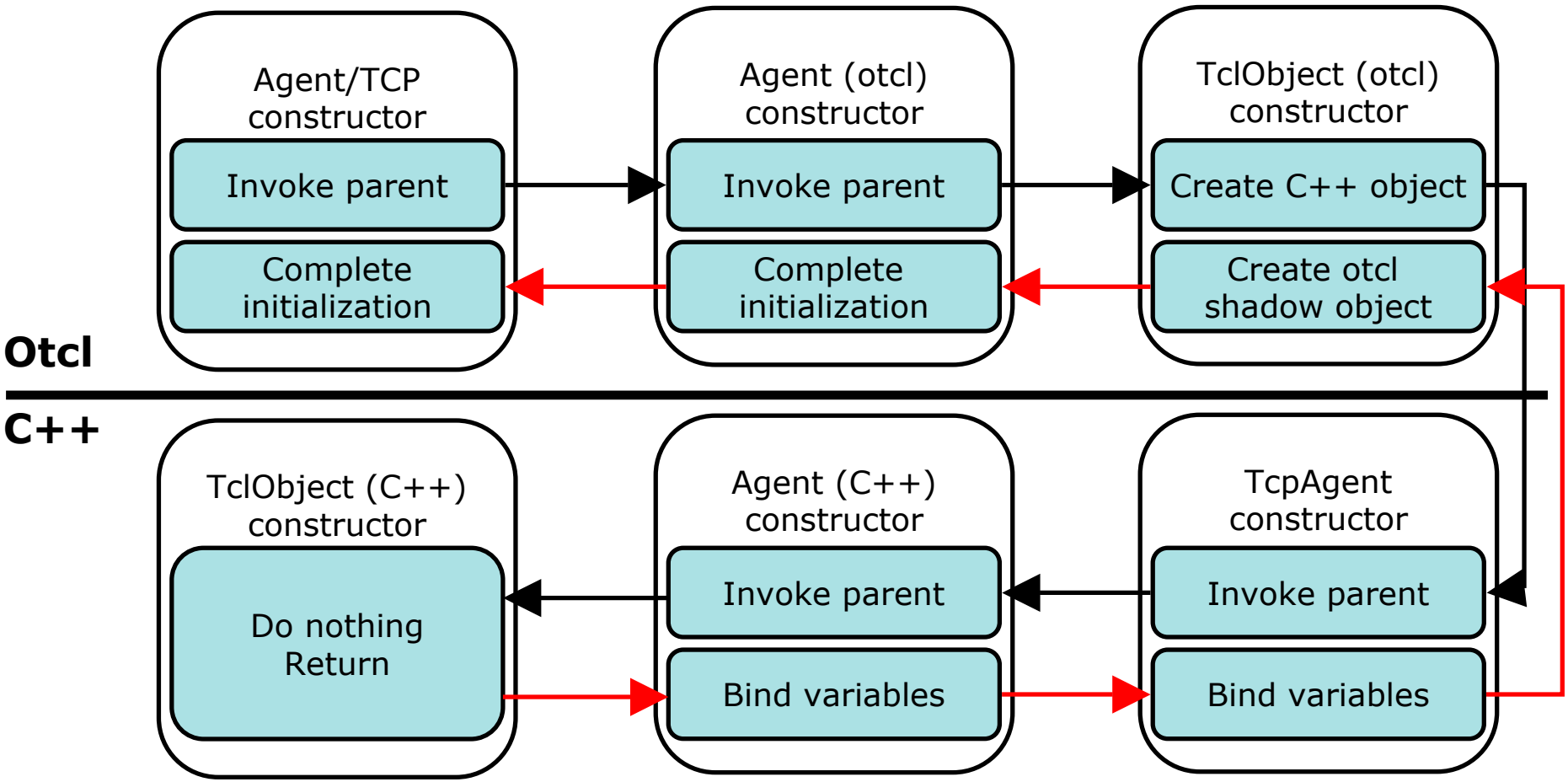
Class TclObject



Object shadowing



Object shadowing



Class TclClass

- Pure virtual class
- Two main functions
  - construct the interpreted class hierarchy to mirror the compiled class hierarchy
  - provide methods to instantiate new TclObjects

```
static class RenoTcpClass: public TclClass {
public:
    RenoTcpClass() : TclClass("Agent/TCP/Reno") {}
    TclObject* create(int argc, const char*const*
argv) {
            return (new RenoTcpAgent());
    }
} class_reno;
```

Object name in Otcl space

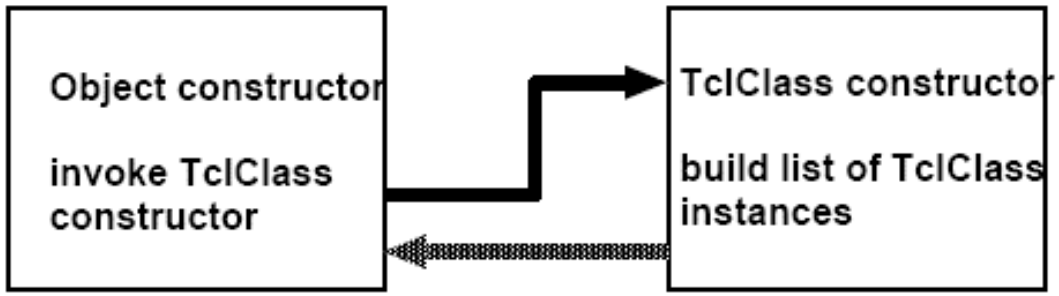
Invoked by 'new Agent/TCP/Reno" in Otcl"

Initial TclObject

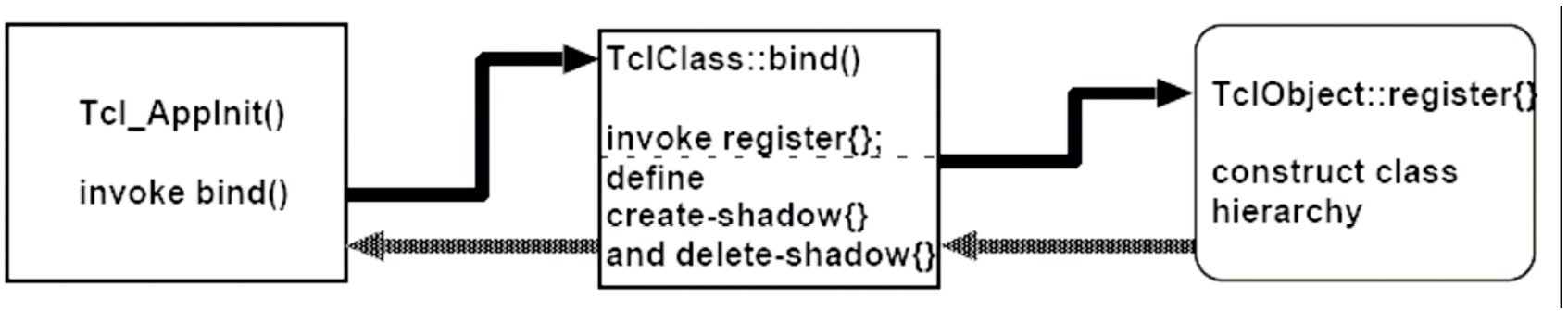
Invoke the constructor

Class TclClass

- static initialization by compiler



- run time activation at startup



Creating C++ class

- Procedure
  - Create <(new_stuff.h,) new_stuff.cc>
  - Add <new_stuff.cc> to OBJ_CC in $ns/Makefile
  - Recompile
    - #cd $ns ; make depend ; make

Description of C++ Class

- Open the same interface to the other space with 'TclObject'

Provide same interface to otcl space and C++

```
class NewCpp : public TclObject {
public:
    NewCpp();
};

NewCpp::NewCpp() {
    printf("Hello, new C++ class is
created.\n");
}
```

Mirroring the both space

- 'TclClass' is used to create mirrored compiled instance

```
class NewCpp : public TclObject {
public:
    …
};

static class NewCppClass : public TclClass
{
public:
    NewCppClass() : TclClass("NewCpp") {}
    TclObject* create(int, const
char*const*) {
            return(new NewCpp());
    }
} class_newcpp;
```

Invoked when 'new NewCpp' is called in otcl interpreter

Compile Procedure

- Move new_cpp.cc to $ns
- Modify the Makefile in $ns
  - OBJ_CC = \ new_cpp.o
- Recompile

Running Example but…

- % set a [new NewCpp]

  Hello, new C++ class is created

- Does not mean you can access to 'NewCpp member'
  - % $a set var 1

    can't read "var" : no such a variables
  - % $a add 1 5

    Error when calling class NewCpp : add

Export C++ to OTCL

- We know how to create component in both spaces
  - interpreted Object, Instance
  - Compiled Object, Instance
- But, we do not know how to communicate between both spaces
  - Set a variable from OTCL to C++ component
  - Get a variable from OTCL to C++ component
  - Execute a function from OTCL to C++ component

Exporting interface class

- **TclObject** provides two virtual functions
  - All the component which is accessible from OTCL should inherit 'TclObject'
  - bind()
    - Export(set, get) C++ variable to OTCL
  - command()
    - Export C++ function to OTCL

Binding Example

- Link C++ member variables to OTcl object variables
- Override 'bind()' of TclObject in its constructor

```
class NewCpp : public TclObject {
public:
    NewCpp();
    int store_int_;
};

NewCpp::NewCpp() {
    printf("Hello, new C++ class is
created.\n");
    bind("store_", &store_int_);
}
```

Variable in OTL

Variable in C++

Function ‘bind’

- For integer and real number:
  - bind(“varotcl”,&var);
- For time:
  - bind_time(“timeotcl”,&time);
  - in tcl script the default units is second.
- For bandwith:
  - bind_bw(“bwotcl”,&bw);
  - // can set bwotcl 1(m: megabits, mb: megabytes, k:kilobits, kb: kilobytes).

Accessing C++ variable from OTCL

- Recompile NewCpp, then

```
%set a [new NewCpp]
Hello, new C++ class is created
Warning: no class variable NewCpp::store_

%$a set store_ 10
10
%$a set store_
10
```

Repairing warning

- Warning: no class variable NewCpp::store_
- Define the default value in $ns/tcl/lib/ns-default.tcl
  - NewCpp set store_ 1
- Recompile

Exporting C++ func. to OTcl

- Implement OTcl methods in C++
- TclObejct::command()
  - Command Dispatcher
  - Link C++ functions to Otcl procedures
- Override ‘command()’ of TclObject

Command Dispatcher

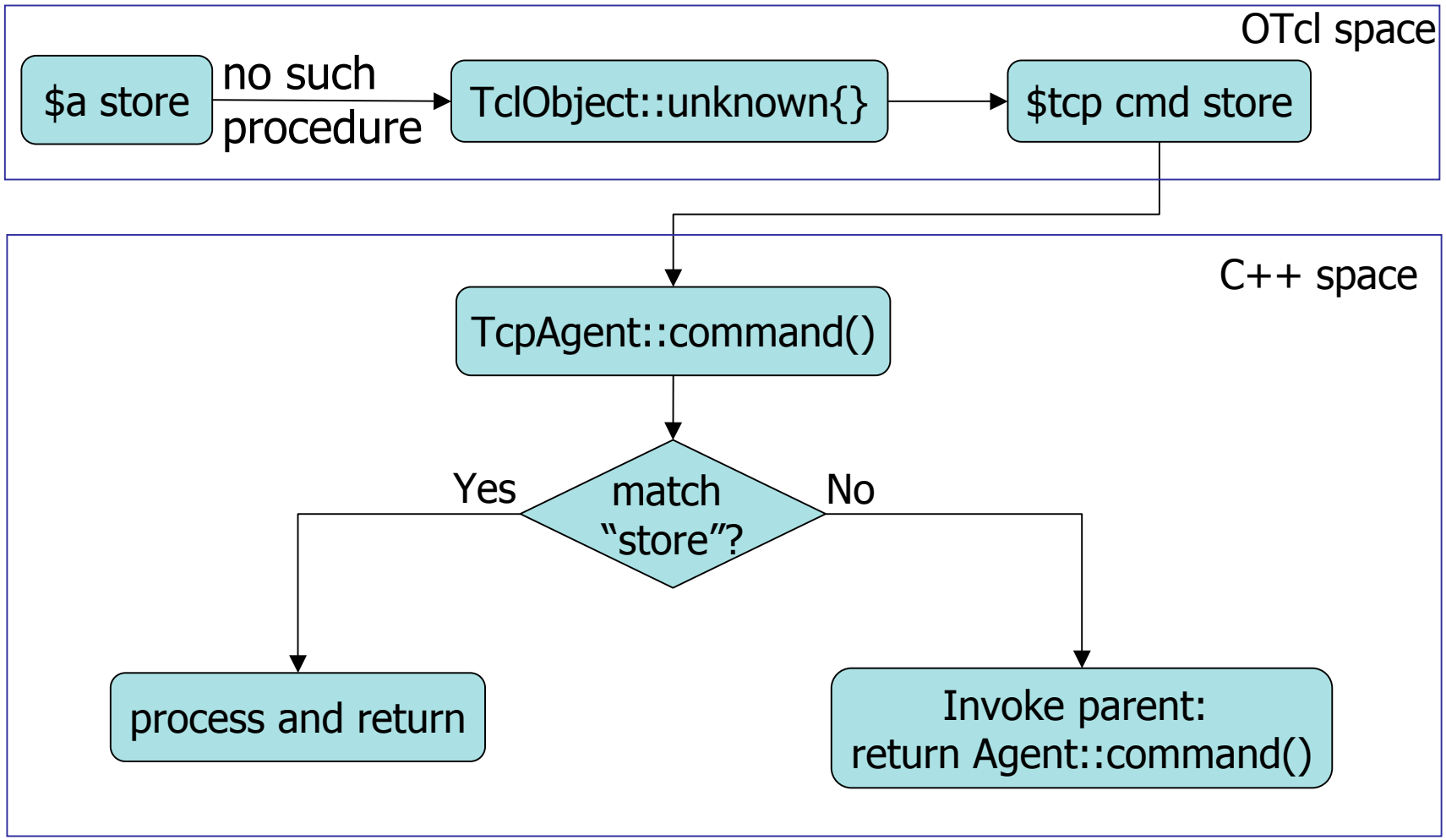
```
class NewCpp : public TclObject {
public:
  NewCpp();
  int store_int_;
  int command(int argc, const char*const* argv);
};
NewCpp::NewCpp() {
  printf("Hello, new C++ class is created.\n");
  bind("store_", &store_int_);
}
NewCpp::command (int argc, const char*const* argv) {
  if(argc==3) {
       if(strcmp(argv[1],"store")==0{
              store_int=atoi(argv[2];)
              reture(TCL_OK);
       }
  }
  return(TclObject::command(argc, argv));
}
```

Running with bind, command

- Recompile, then

```
%set a [new NewCpp]
Hello, new C++ class is created

% $a store 10
% $a set store_
10
```

‘command’ call tracing



Important linkage Classes

class TclObject	Root of basic object hierarchy in *ns* Support bind(), command()
class TclClass	C++ class to set up the TclObject hierarchy
class Tcl	C++ methods to access the OTcl interpreter
class TclCommand	Standalone global commands
class EmbeddedTcl	Ns script initialization Container for Tcl scripts that are pre-loaded at startup
class InstVar	internal class to bind C++ member variables to OTcl instance variables

Accessing OTcl from C++ - Tcl class

- Class with a handle to Tcl Interpreter
  - Invoke an OTcl command from C++
- Obtain a reference to the 'Tcl' instance
  - Tcl& tcl = Tcl::instance();
- invoke an OTcl command

```
tcl.eval("puts \"Print out from C++\"");
tcl.evalf("puts \"My_var1 = %d\"", var1);
tcl.evalf("puts \"My_var1 = %f\"", var2);
```

Class Tcl

- Passing Results to the Interpreter
  - tcl.result(const char* s)
    - Pass the result string s back to the interpreter.
  - tcl.resultf(const char* fmt, . . .)
- Passing Results from the Interpreter
  - tcl.result(void)
    - used to retrieve the result.
    - result is a string!

```
tcl.evalc("Simulator set
NumberInterfaces_");
    char* ni = tcl.result();
    if (atoi(ni) != 1)
    tcl.evalc("Simulator set
NumberInterfaces_ 1");
```

Class TclCommand

- provide the mechanism to export simple commands (top level commands) to the interpreter
  - % hi this is ns [ns-version]
  - hello world, this is ns 2.0a12
- Defined in $ns/common/misc.cc

TclCommand example

```
class say_hello : public TclCommand {
public:
    say_hello():TclCommand("hi") {}
    int command(int argc, const char*const* argv)
{
            printf("Hello World\n"); return TCL_OK;
    }
};
void init_misc(void) {
    new say_hello;
}
```

- Compile, then
  %hi
  Hello, World

Class InstVar

- One object per bound variable
- Created by TclObject::bind() call
- Constructor activity
  - Point to C++ member variable
  - Create instance variable for interpreted object
  - Enable trap read/writes to instance variable using **Tcl_TraceVar()**

Otcl linker summary

- Class TclObject
  - root object for mirrored hierarchies
    - Unifies interpreted and compiled hierarchy
    - Seamless access to *ns* objects in two spaces
- Class TclClass
  - class that sets up the interpreted hierarchy
    - establish interpreted hierarchy
    - shadowing methods
- Class Tcl
  - primitives to access the interpreter

Part III : Extending NS2

1. Extending ns2 with OTcl
2. Extending ns2 with C++

Extending NS2

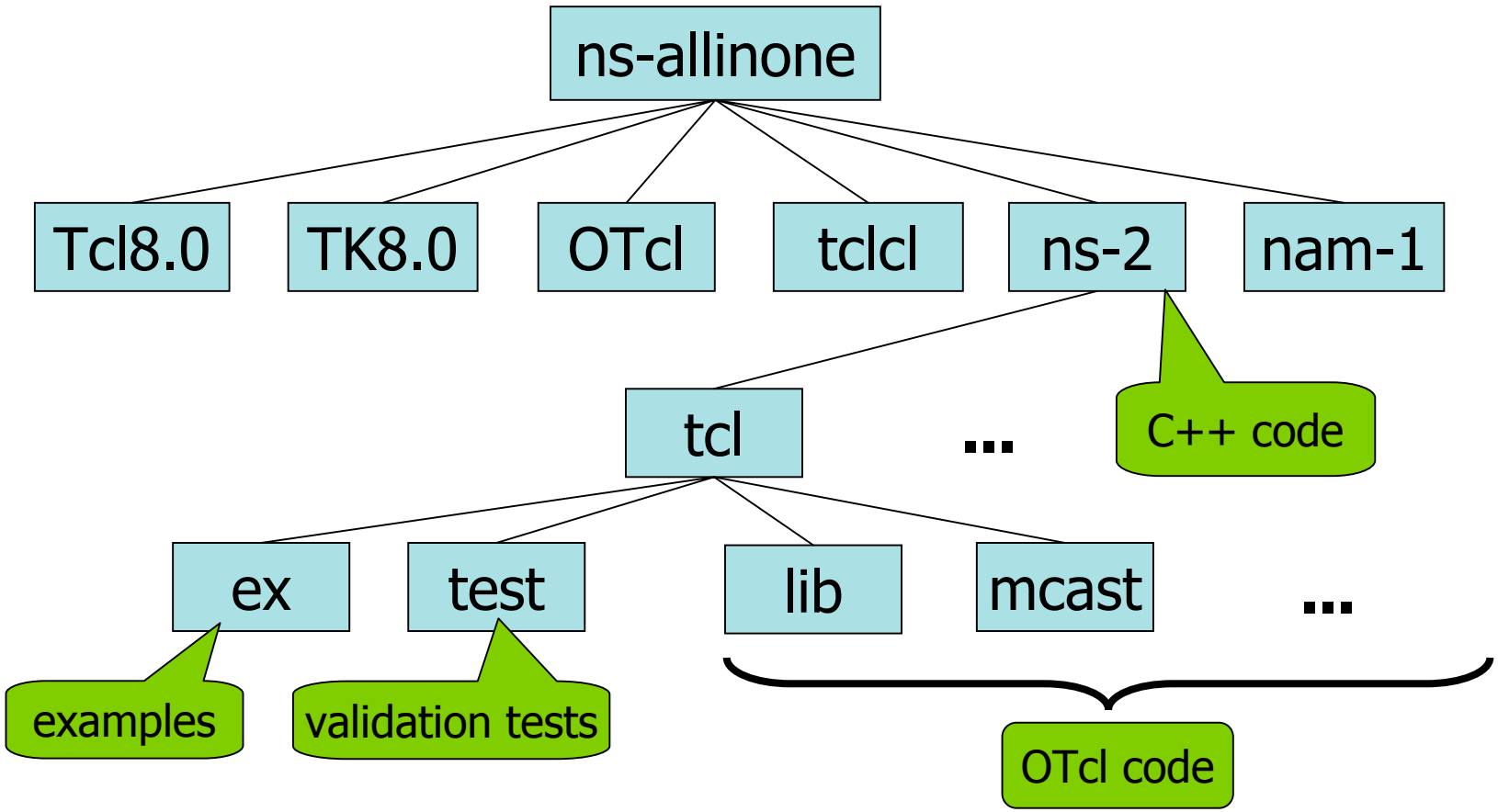
Why extend ns?

- Not all the components exist in NS
  - Contribution codes
    - http://nsnam.isi.edu/nsnam/index.php/Contributed_Code
- If not exist.
  - You should make your own codes
  - In OTcl
  - In C++

Requirement

- C++ language
- OTcl language
- C++ - Otcl linkage
- +Packet processing

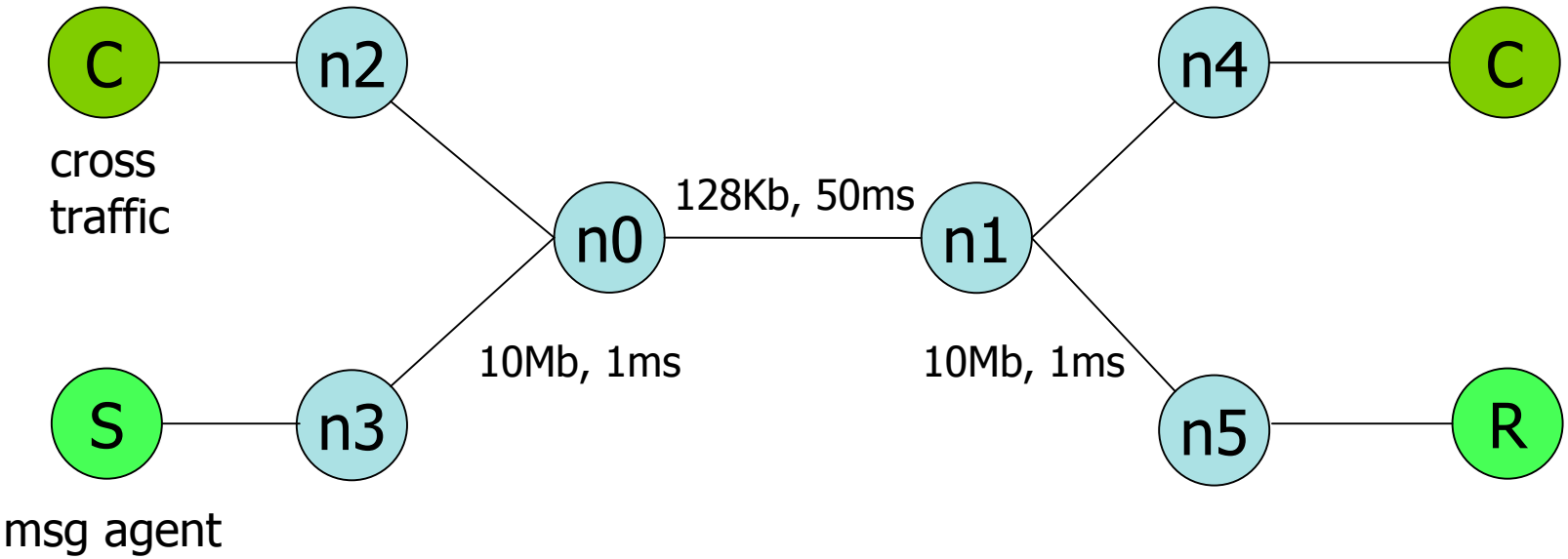
NS directory structure



Extending ns2 with OTcl

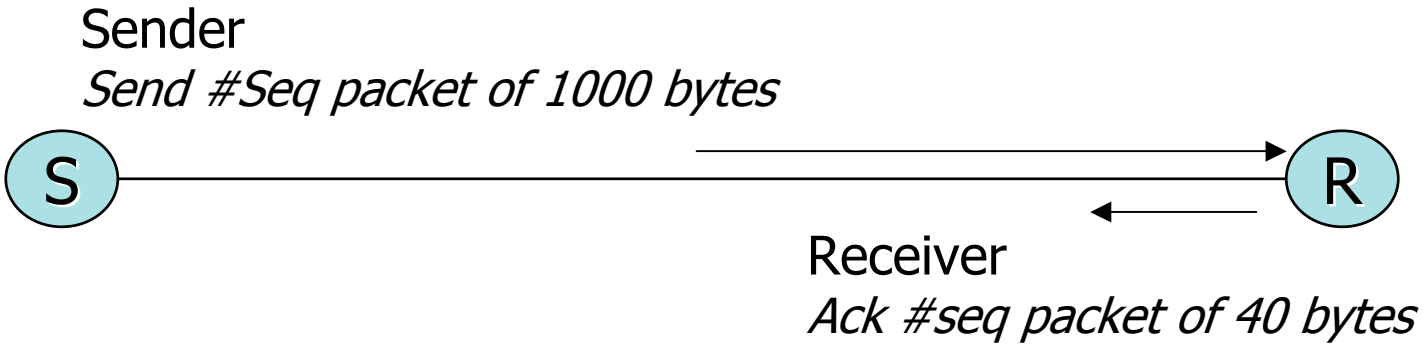
- Reuse?
  - Modify existing code
    - Recompile
  - Adding code
    - Change $ns/Makefile(NS_TCL_LIB)
    - Update tcl/lib/ns-lib.tcl (ns-default.tcl)
    - Recompile
- No reuse
  - Source
    - Sourcing your changes in your simulation scripts
    - do not need compiling

Example:Agent/Message



Example scenario from Haobo Yu & Nader Salehi, USC/ISI

Example:Agent/Message



- Scenario
  - Send/receive seq packets between two message agent
- Good for fast prototyping a simple idea
- Extending ns functionality with pure OTcl

Define Sender

```
Class Sender -superclass Agent/Message

    # Message format: "Addr Op SeqNo"
Sender instproc send-next {} {
    $self instvar seq_
    $self instvar agent_addr_
    $self send "$agent_addr_ send $seq_"
    incr seq_
    global ns
    $ns at [expr [$ns now]+0.1] "$self send-next"
}
Sender instproc recv {msg} {
          $self instvar agent_addr_
          set sdr [lindex $msg 0]
          set seq [lindex $msg 2]
          puts "Sender gets ack $seq from $sdr"
}
```

Define Receiver

```
Class Receiver -superclass Agent/Message
    # Message format: “Addr Op SeqNo”
    Receiver instproc recv {msg} {
          $self instvar agent_addr_
          set sdr [lindex $msg 0]
          set seq [lindex $msg 2]
          puts "Receiver gets seq $seq from $sdr"
          $self send "$agent_addr_ ack $seq"
    }
```

Scheduler and tracing

```
# Create scheduler
set ns [new Simulator]

set fd [open message.nam w]
$ns namtrace-all $fd

proc finish {} {
  global ns fd
  $ns flush-trace
  close $fd
  exec nam message.nam &
  exit 0
}
```

Setting up a topology

```
for {set i 0} {$i < 6} {incr i} {
        set n($i) [$ns node]
    }
    $ns duplex-link $n(0) $n(1) 128kb 50ms DropTail
    $ns duplex-link $n(1) $n(4) 10Mb 1ms DropTail
    $ns duplex-link $n(1) $n(5) 10Mb 1ms DropTail
    $ns duplex-link $n(0) $n(2) 10Mb 1ms DropTail
    $ns duplex-link $n(0) $n(3) 10Mb 1ms DropTail
    $ns queue-limit $n(0) $n(1) 5
    $ns queue-limit $n(1) $n(0) 5

$ns rtproto DV
```

Cross Traffic

```
set udp0 [new Agent/UDP]
$ns attach-agent $n(2) $udp0
set null0 [new Agent/Null]
$ns attach-agent $n(4) $null0
$ns connect $udp0 $null0


set exp0 [new Application/Traffic/Exponential]
$exp0 set rate_ 128k
$exp0 attach-agent $udp0
$ns at 1.0 "$exp0 start"
$ns at 2.0 "$exp0 stop"
```

Message Traffic

```
set sdr [new Sender]
$sdr set packetSize_ 1000
$sdr set seq_ 0

set rcvr [new Receiver]
$rcvr set packetSize_ 40

$ns attach-agent $n(3) $sdr
$ns attach-agent $n(5) $rcvr
$ns connect $sdr $rcvr
$ns connect $rcvr $sdr
$ns at 1.1 "$sdr send-next"

$ns at 2.0 finish

$ns run
```

Example output

```
$ ns message.tcl
Receiver gets seq 0 from 3
Sender gets ack 0 from 5
Receiver gets seq 1 from 3
Sender gets ack 1 from 5
Receiver gets seq 2 from 3
Sender gets ack 2 from 5
Receiver gets seq 4 from 3
Sender gets ack 4 from 5
Receiver gets seq 7 from 3

% nam message.nam &
```

Extending ns2 with C++

- Existing code
  - recompile
- Add new code
  - change Makefile and recompile

Creating a new component

- Basic task
  - Inheritance point
  - Class & API virtual functions def.
  - Otcl linkage functions def.
  - Write the necessary OTcl code to access your component

Creating a new component

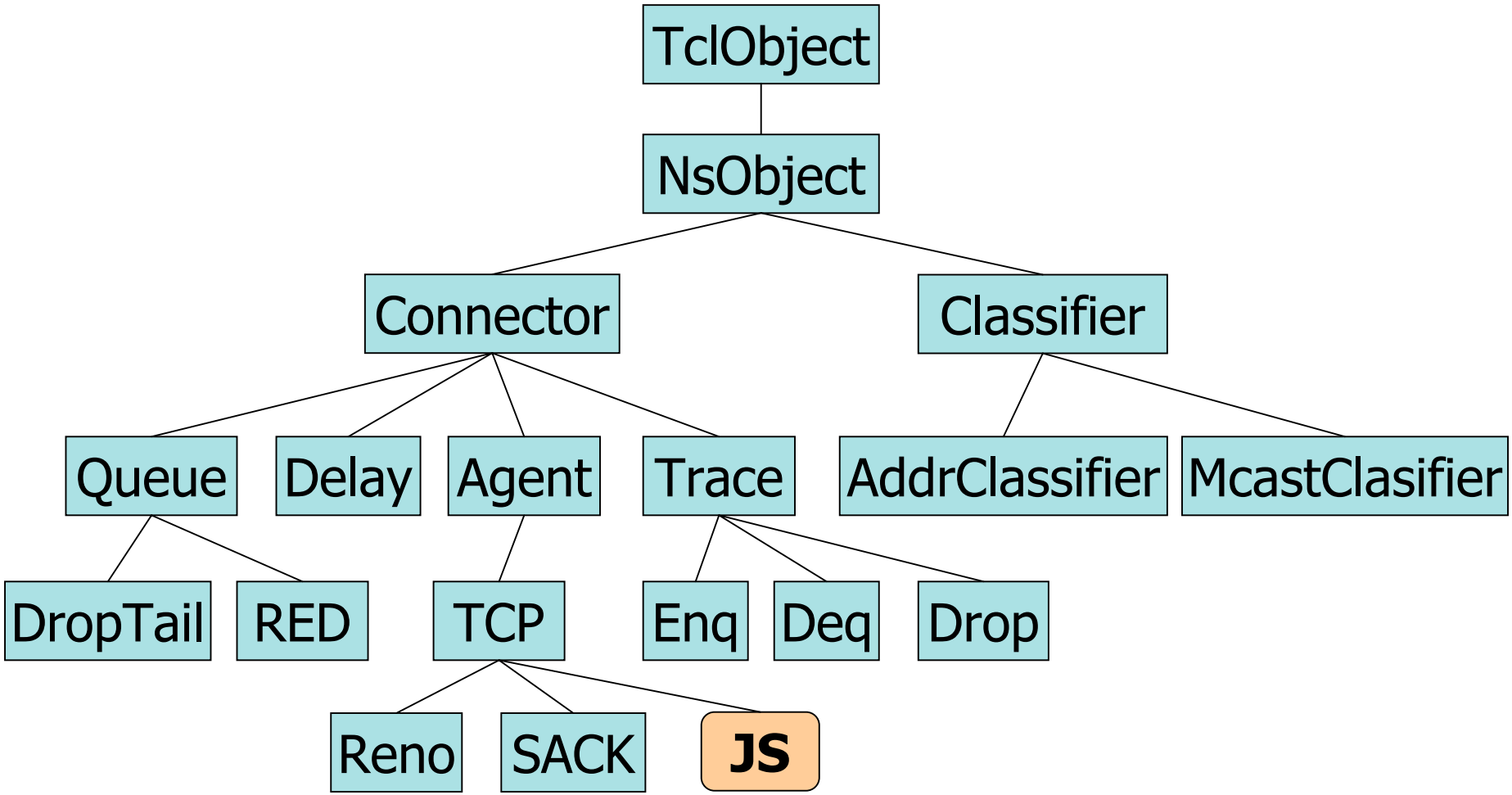
- New Agent/Old Header
  - TCP Jump Start Example
    - Wide-open transmission window at the beginning
    - From `cwnd_ += 1` To `cwnd_ = MAXWIN_`
    - example from Padma Haldar's presentation

- New Agent/New Header
  - Ping Agent Example
    - Ping request & Ping response
    - Example from ns2 tutorial

Creating a new c++ component

New agent / old header

TCP Jump Start - Inheritance



TCP Jump Start – Class def.

- Class & API virtual functions def.
  - ~ns/tcp/tcp-js.cc

```
#include "tcp.h"
/* Declaration */
class JSTcpAgent : public TcpAgent {
public:
JSTcpAgent();
    virtual void set_initial_window();
private:
    int maxwin_;
};
```

Must override function
TcpAgent::set_initial_window()

TCP Jump Start – Otcl linkage

- Otcl linkage functions def.

```
/* Class TclClass */
static class JSTcpClass : public
TclClass {
public:
   JSTcpClass() :
TclClass("Agent/TCP/JS"){}
   TclObject* create(int, const char*
const*) {
          return(new JSTcpAgent());
   }
} class_tcpjs;
```

TCP Jump Start – Class body

- Implement JSTcpAgent body

```
/* variable binding - constructor */
JSTcpAgent::JSTcpAgent() {
   bind("MAXWIN_", &maxwin_);
}

/* override "set_initial_window" */
void JSTcpAgent::set_initial_window()
{
   cwnd_ = maxwin_;
}
```

TcpAgent::set_initial_window()
{ cwnd_ = 1.0; }

TCP Jump Start – default value

```
% cat $ns/tcl/lib/ns-default.tcl
   # To remove warns
   # Agent/TCP/JS
     Agent/TCP/JS set MAXWIN_ 1
```

TCP Jump Start - Compiling

- % cat $ns/Makefile
  - …
  - tcp/tcp.o **tcp/tcp-js.o** tcp/tcp-sink.o
- % cd $ns ;make

TCP Jump Start–OTcl Scenario

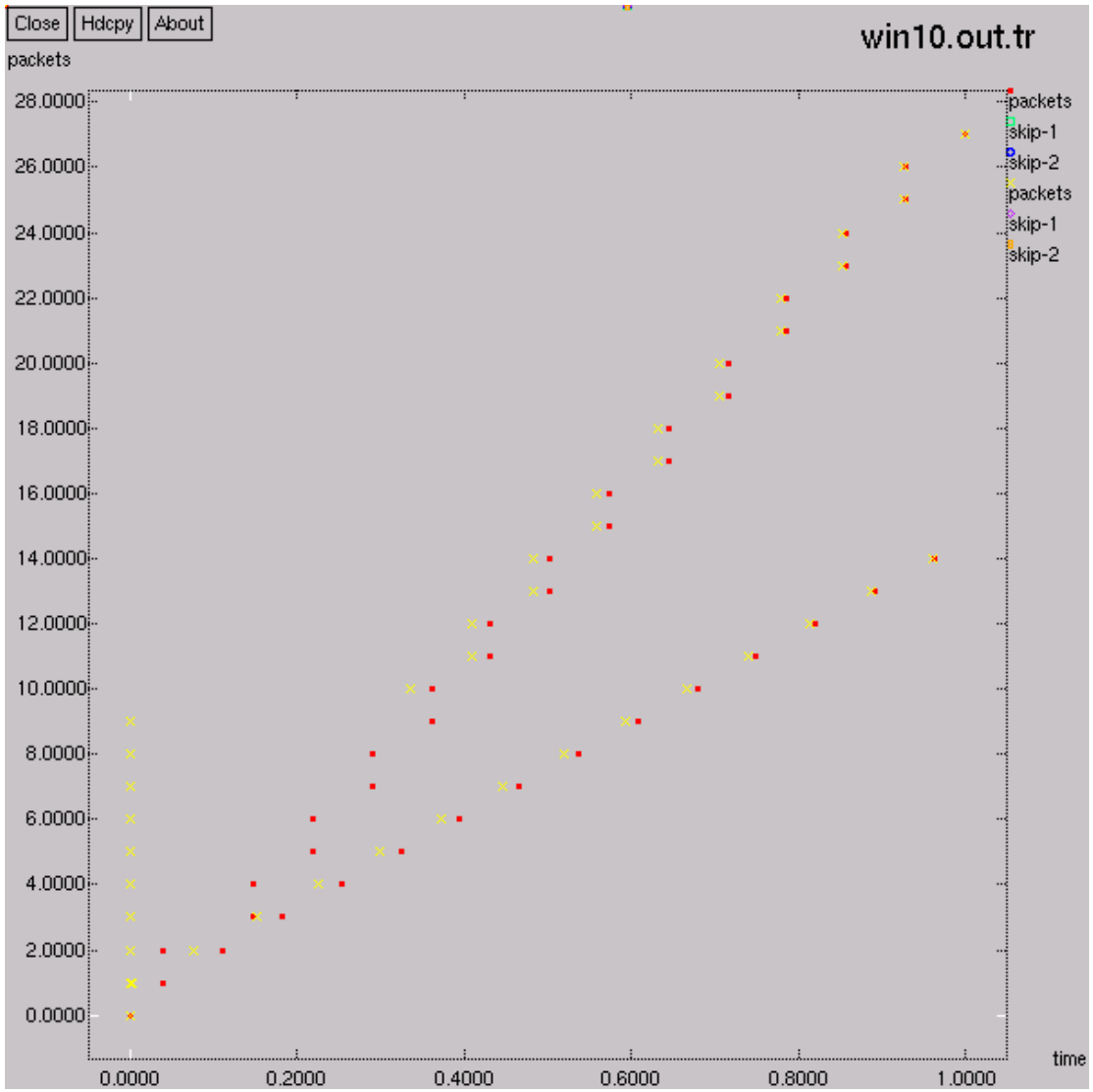
```
set ns [new Simulator]

set init_win [lindex $argv [expr [lsearch $argv "-win"] + 1]]
if {$argc != 2} {
    puts "usage: ns basic_tcpjs_trace.tcl -win #"
}

set tracefile [open out.tr w]
$ns trace-all $tracefile

set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 1Mb 2ms DropTail
```

TCP Jump Start–OTcl Scenario

```
set tcp0 [new Agent/TCP/JS]
$tcp0 set MAXWIN_ $init_win
$ns attach-agent $n0 $tcp0
set ftp0 [new Source/FTP]
$ftp0 set agent_ $tcp0

set sink0 [new Agent/TCPSink]
$ns attach-agent $n1 $sink0
$ns connect $tcp0 $sink0

$ns at 1.0 "$ftp0 start"
$ns at 5.0 "exit"
$ns run
```

```
% ns basic_tcpjs_trace.tcl -win 10
% ns basic_tcpjs_trace.tcl -win 1
```

TCP Jump Start-Result



Creating a new c++ component

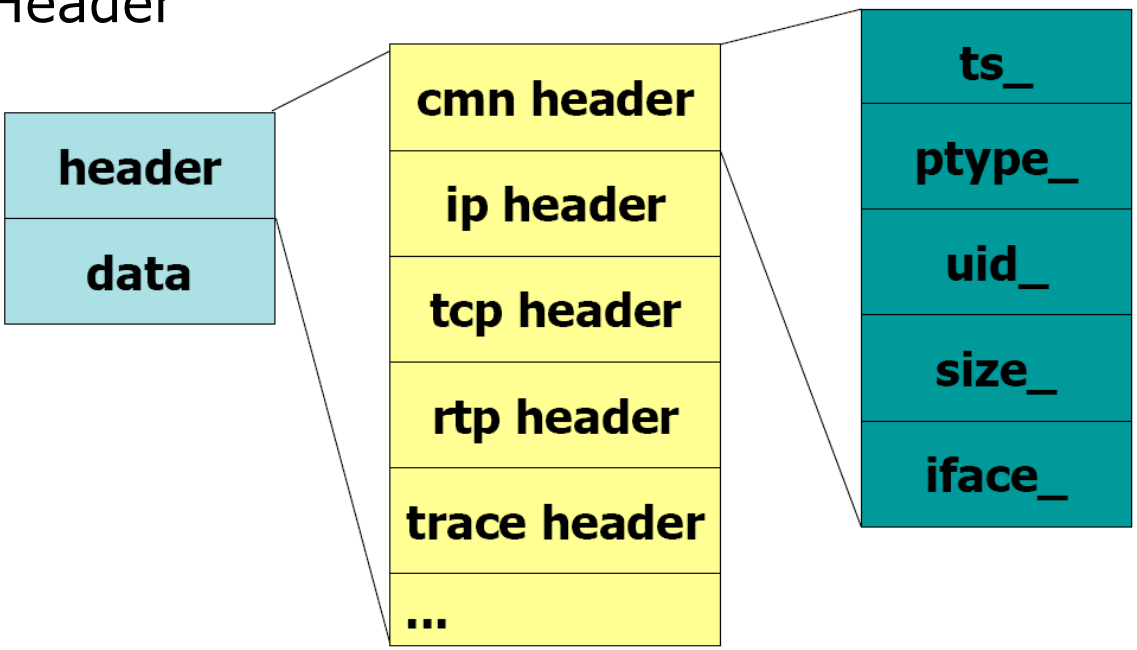
New agent / New header

Packet

- Basic object for data exchange
  - Allocated by an agent
  - Use ‘send()’, receiver()’
- Packets are implemented in C++ space
  - Performance
  - Compiled packet are enabled in OTcl before start of simulation via the class “PacketHeaderManager”
- Each packet contains all enabled packet headers
  - Accessing any header in the stack of a packet can be done by using ‘offset’ value

Packet Header

- Composed of a stack of headers
  - +*optional* data space
    - Meaningless to non-real-time simulation Packet Header



Packet

- Packet header
  - Per-protocol information
  - New protocols may define their own header or may extend existing headers
    - Should create or modify agent handling new header
- Packet header linkage
  - ‘class PacketHeaderManager’
    - Subclass of ‘class TclClass’

New Packet Header

- Create new header structure
- Enable tracing support of new header
- Create static class for Otcl-linkage (packet.h)
- Enable new header in OTcl(tcl/lib/ns-packet.tcl)

Ping Agent Example

- Scenario
  - Ping request/ Ping response between two nodes
- New Agent-SPing/Agent
  - To evade collision with pre-exist Ping/Agent
- New Header
  - PacketHeader/Sping

- Example from ‘ns tutorial’ of Macr Greis
  - Slightly modified for presentation

SPing.h

```
struct hdr_sping {
      char ret; // '0' for ping request, '1' for ping reply
      double send_time;

      // Header access methods
      static int offset_; // required by PacketHeaderManager
      inline static hdr_sping* access(const Packet* p) {
             return (hdr_sping*) p->access(offset_);
      }
};
class SPingAgent : public Agent {
public:
      SPingAgent();
      virtual int command(int argc, const char*const* argv);
      virtual void recv(Packet*, Handler*);
};
```

SPing.cc-OTcl linkage

```
int hdr_sping::offset_;
static class PingHeaderClass : public PacketHeaderClass {
public:
        PingHeaderClass() : PacketHeaderClass("PacketHeader/SPing",
                                              sizeof(hdr_sping)) {
                bind_offset(&hdr_sping::offset_);
        }
} class_spinghdr;

static class SPingClass : public TclClass {
public:
        SPingClass() : TclClass("Agent/SPing") {}
        TclObject* create(int, const char*const*) {
                return (new SPingAgent());
        }
} class_sping;

SPingAgent::SPingAgent() : Agent(PT_SPING)
{
        bind("packetSize_", &size_);
}
```

PT_SPING
Otcl linkage

Agent/Sping
Otcl linkage

SPing.cc-Ping request

```
int SPingAgent::command(int argc, const char*const* argv)
{
  if (argc == 2) {
    if (strcmp(argv[1], "send") == 0) {
      // Create a new packet
      Packet* pkt = allocpkt();
      // Access the Ping header for the new packet:
      hdr_sping* hdr = hdr_sping::access(pkt);
      hdr->ret = 0; // ping request
      hdr->send_time = Scheduler::instance().clock();
      send(pkt, 0);
      return (TCL_OK);
    }
  }
return (Agent::command(argc, argv));
}
```

SPing.cc-Handling ping request

```
void SPingAgent::recv(Packet* pkt, Handler*)
{
  // Access the IP header for the received packet:
  hdr_ip* hdrip = hdr_ip::access(pkt);
  // Access the SPing header for the received packet:
  hdr_sping* hdr = hdr_sping::access(pkt);
  if (hdr->ret == 0) { // case 'ping request'
     // Handling 'ping request'
     double stime = hdr->send_time;
     Packet::free(pkt);
     // Preparing 'ping reply'
     Packet* pktret = allocpkt();
     hdr_sping* hdrret = hdr_sping::access(pktret);
     hdrret->ret = 1;
     hdrret->send_time = stime;
     send(pktret, 0);
```

SPing.cc-Handling ping reply

```
} else {
// case ‘ping reply’
// In the Tcl code, a procedure 'Agent/Ping recv {from
// rtt}’ has to be defined which allows the user to
react // to the ping result.

      char out[100];
      sprintf(out, "%s recv %d %3.1f", name(),
   hdrip->src_.addr_ >>Address::instance().NodeShift_[1],
(Scheduler::instance().clock()-hdr->send_time)*1000);
      Tcl& tcl = Tcl::instance();
      tcl.eval(out);
      Packet::free(pkt);
  }
}
```

sping.tcl-simulation scenario

```
#Define a 'recv' function for the class 'Agent/Ping'
Agent/SPing instproc recv {from rtt} {
      $self instvar node_
      puts "node [$node_ id] received ping answer from \
                $from with round-trip-time $rtt ms."
}
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam &
        exit 0
}
set n0 [$ns node]
set n1 [$ns node]
```

sping.tcl-simulation scenario

```
$ns duplex-link $n0 $n1 1Mb 10ms DropTail

set p0 [new Agent/SPing]
$ns attach-agent $n0 $p0
set p1 [new Agent/SPing]
$ns attach-agent $n1 $p1

$ns connect $p0 $p1

$ns at 0.2 "$p0 send"
$ns at 0.4 "$p1 send"
$ns at 0.6 "$p0 send"
$ns at 0.6 "$p1 send"
$ns at 1.0 "finish"

$ns run
```

Necessary Changes

- Enable Tracking
  - $ns/common/packet.h
  - Used as a “glue” to bind numeric packet type values with their symbolic names

```
enum packet_t {
        PT_TCP,
        ......
        PT_SPING,//numeric code for sping
                 //packet protocol ID for sping-agent
};
class p_info {
public:
        p_info() {
                 name_[PT_TCP]= "tcp";
                 ...........
                 name_[PT_SPING]=“SPing"; //symbolic code for sping-agent
        }
        .....
 }
```

Necessary Changes

- $ns/tcl/lib/ns-default.tcl

  Agent/SPing set packetSize_ 64

- Register New Header
  - $ns/tcl/lib/ns-packet.tcl

```
//The packet header format initialization implementation
        foreach prot {
                AODV
                ARP
                aSRM
        ……
                SPing
        } {
                add-packet-header $prot
        }
```

Compiling and running

- Makefile

```
srm-topo.o \
ping.o \
$(LIB_DIR)dmalloc_support.o \
```

- Recompile and run

```
make depend; make
ns  ping.tcl
```

Timer

- Steps : Add timer to the NewAgent
  - In newTimerClass
    - define the timer object inherit ‘class TimerHandler’
    - add expire() implementation of ‘TimerHandler’
      - Invoked by TimerHandler::sched() in newagent
  - In NewAgent class
    - add timer instance to new agent
    - schedule with ‘time’ value

Class TimerHandler

- Base class 'TimerHandler'
  - void sched(time) :
    - Schedule a timer to expire after 'time' seconds
  - void resched(time)
    - Reschedule a timer
    - similar to sched(), but timer may be pending
  - void cancel()
    - Cancle a pending timer

Class TimerHandler

```
#include "timer-handler.h"

class NewAgent;

class RtxTimer : public TimerHandler {
public:
      RtxTimer(NewAgent *a) : TimerHandler() { a_ = a; }
protected:
      NewAgent *a_;
      virtual void expire(Event *e);
};

void RtxTimer::expire {
      // do an action through the pointer a_
      a_->timeout();
}
```

invoked by TimerHandler::sched(timevalue)

Class NewAgent with timer

```
class NewAgent : {
        …
        void timeout();
        RtxTimer timer_;
        NewAgent::NewAgent : timer_(this) {
                printf("Hello, new C++ class is created.\n");
                timer_.sched(2.0);
        }

        void timeout() {
                double now = Scheduler::instance().clock();
                printf("Timeout occurs at %f.\n", now"
        }
}
```

Invoke RtxTimer::expire()

invoked by RtxTimer::expire()
a_->timeout();

Running Example

- After Recompiling

  % set ns [new Simulator]

  % set a [new NewAgent]

  Hello, a new C++ class is created.

  % $ns run

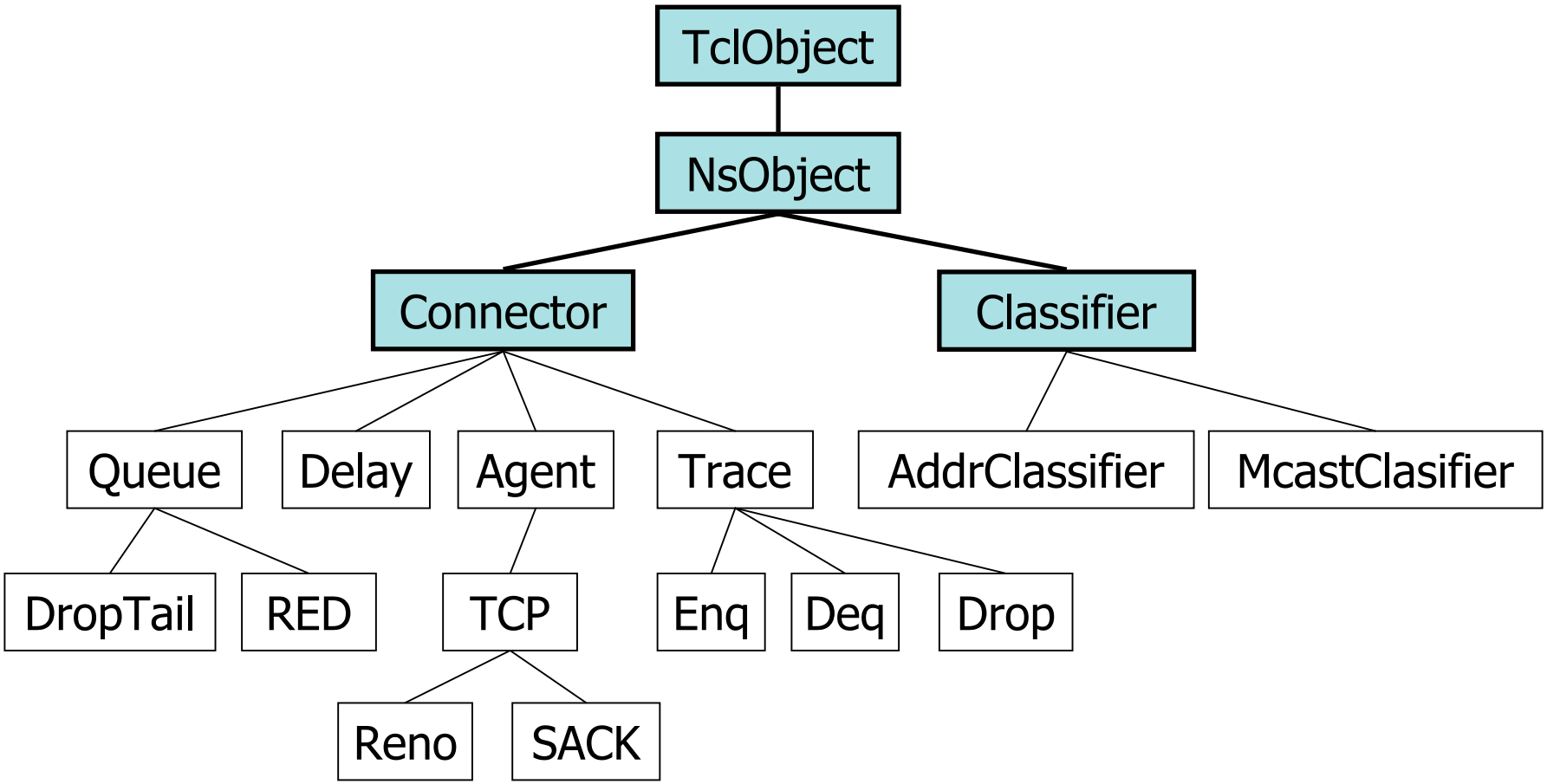
  Timeout occurs at 2.000000

NS2 component

1. NS2 component
2. Tracing

NS component

- Basic Component



TclObject / NsObject

- TclObject
  - Root class of all the compiled object and interpreted object
  - *command, trace, bind*
- NsObject
  - Subclass to TclObejct but
  - Super class of all the compiled obejct
  - Common function to simulator
  - *recv, handle, reset*

Connector

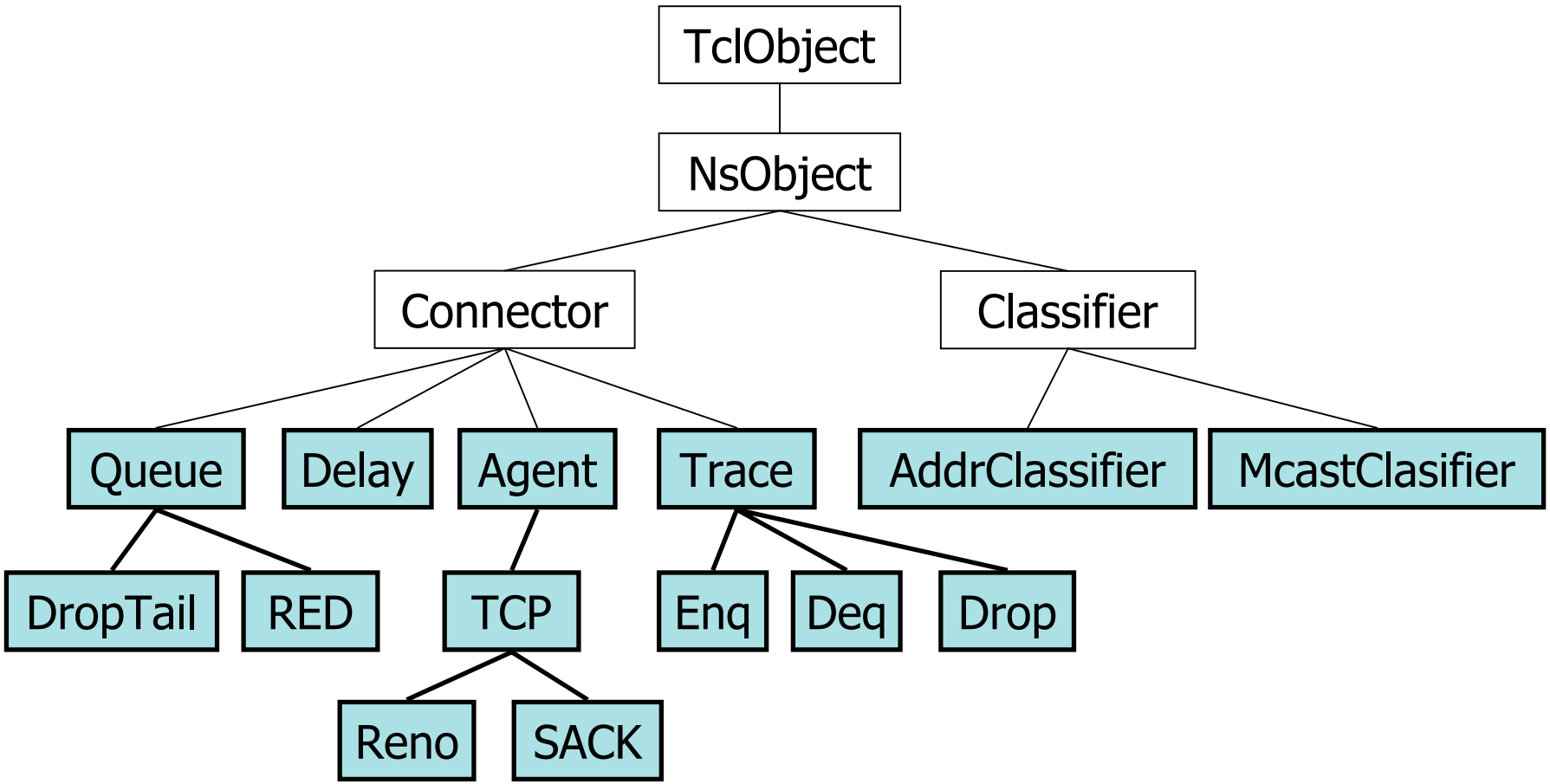
- One output data path
  - Receive incoming packets, and transmit them to their ‘target_’
  - Queue, Agent..
  - Defined in $ns/common/connector.{h,cc}
- Variables
  - NsObject* target_;
- Functions
  - send(Packet* p, Handler* h) { target_->recv(p,h); }
  - recv(Packet* p, Handler* h) { send(p,h); }
  - drop(Packet* p) { Packet::free(p); }

Classifier

- Multiple output data path
  - Used for routing
  - Defined in $ns/classifier/classifier.{h,cc}
- Variables
  - NsObject** slot_;
  - NsObject* default_target_;
- Functions
  - install(int slot, NsObject* p) { slot_[slot] = p; }
  - find(Packet* p) { return slot_[classify(p)]; }
  - recv(Packet* p, Handler* h) {
  - NsObject* node = find(p);
  - node->recv(p,h);
  - }

NS component

- Network Component



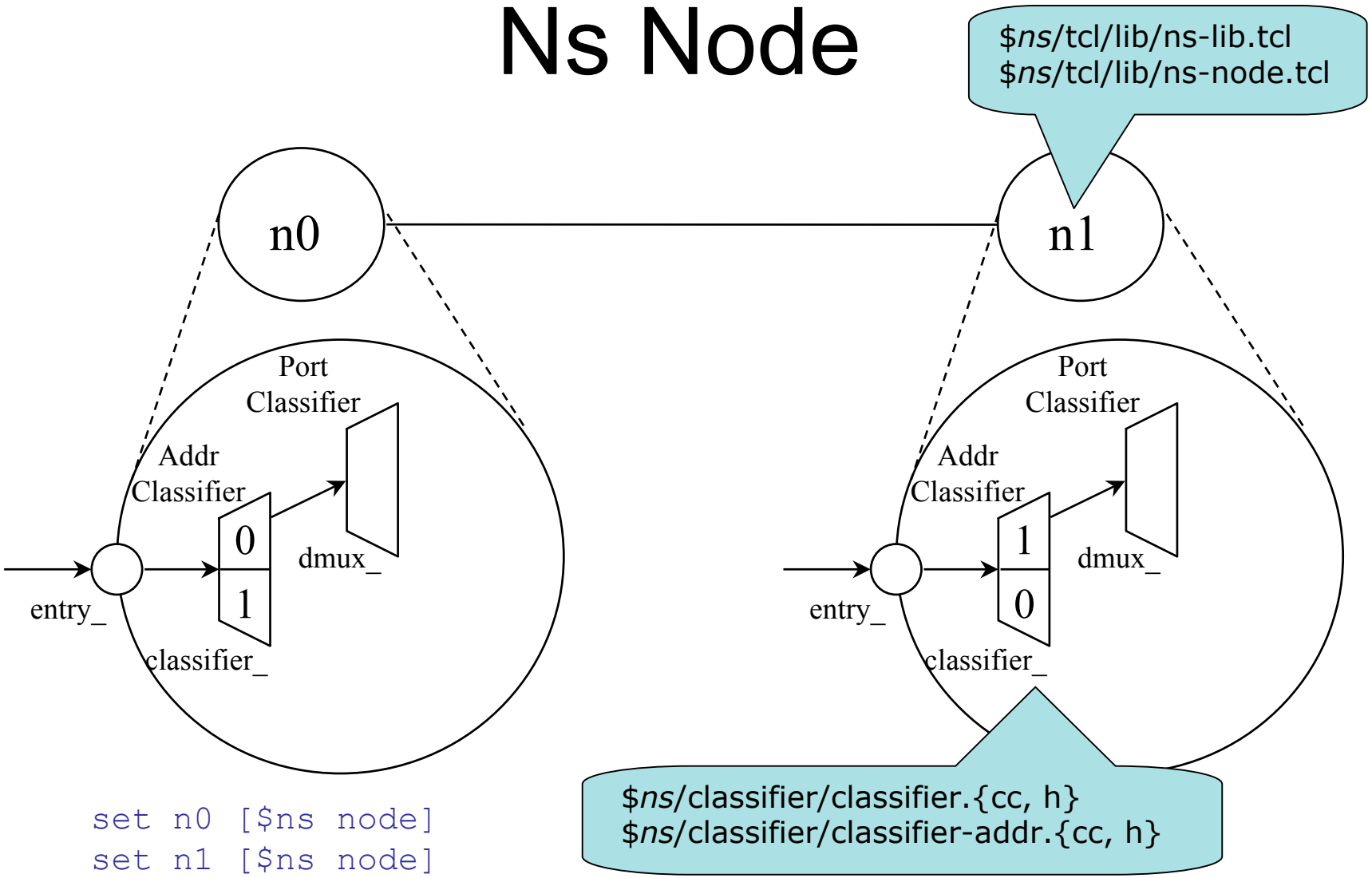
Four major types

- Node
  - Addressable entity
    - Classifier + RtModule
- Link
  - Set of queues
    - Queue + LinkDelay
- Agent
  - Packet generator/consumer
- Application
  - Communication instigator

Node

- Class Node
  - Defined in *$ns/common/node.{h,cc}, $ns/tcl/lib/nsnode.tcl*
  - Standalone class in OTCL
  - Most components are TclObjects

Ns Node



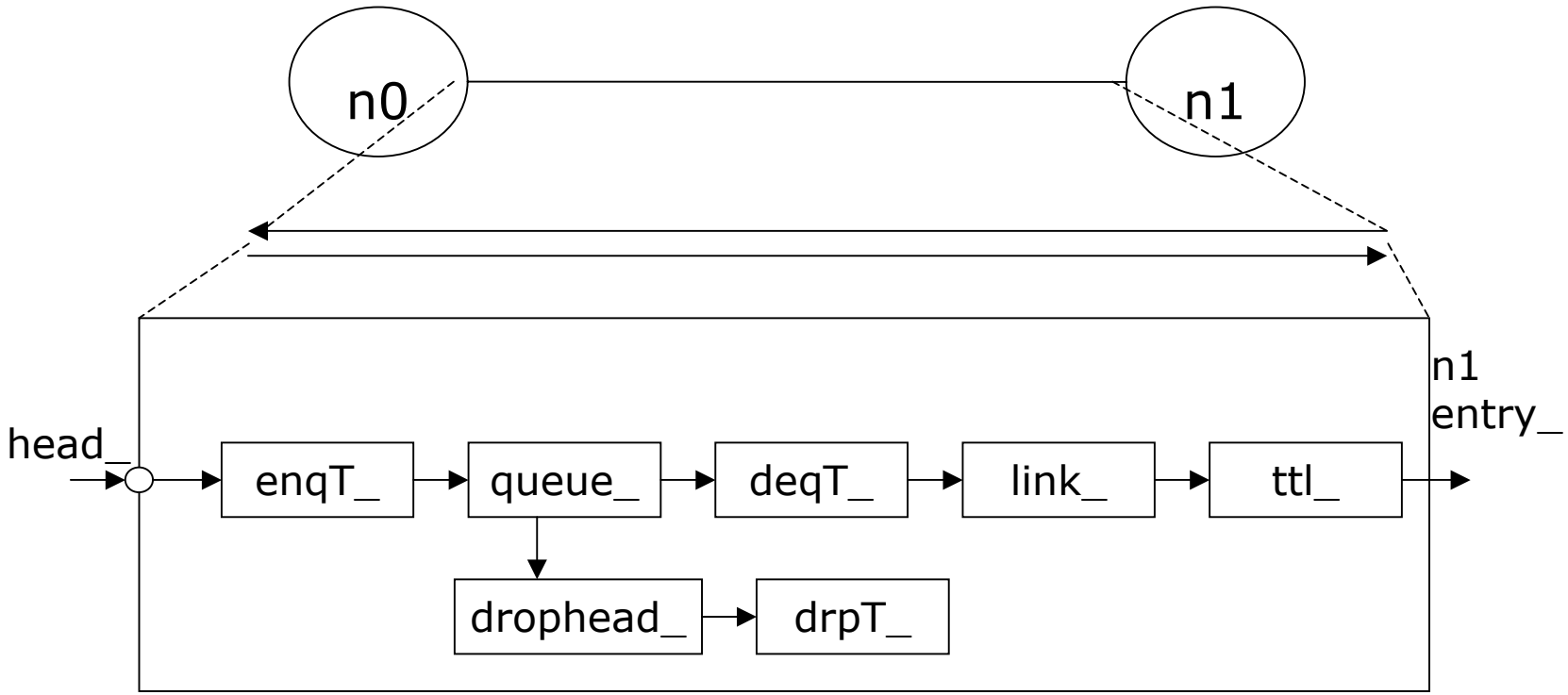

```
set n0 [$ns node]
set n1 [$ns node]
```

Link

- Class link
  - *$ns/tcl/lib/ns-link.tcl*
  - Standalone class in OTCL
  - Most components are TclObjects

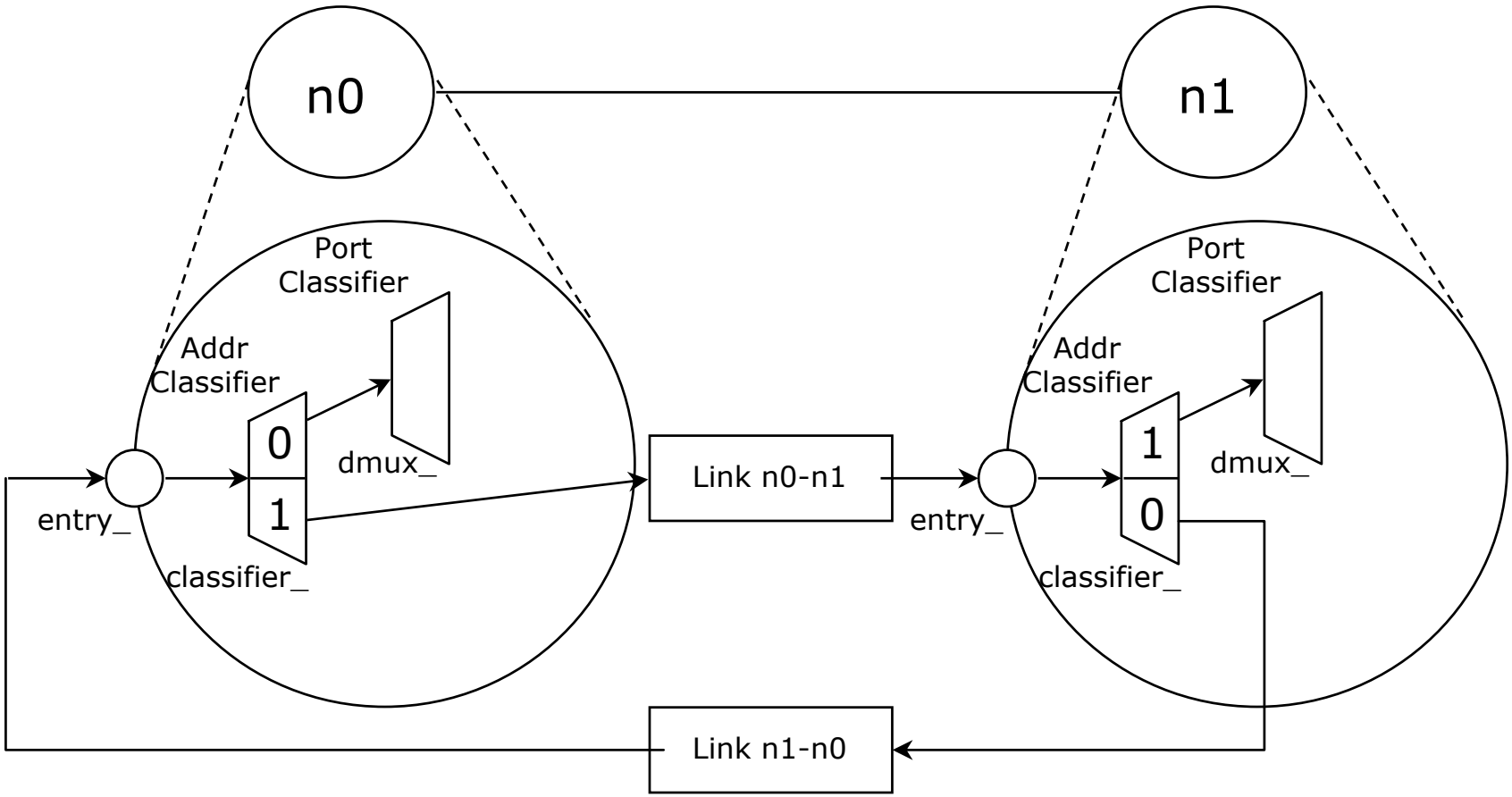
Network Topology - Link

```
SimpleLink instproc init { src dst bw delay q {lltype "DelayLink"} } {
        set drophead_ [new Connector]
        set head_ [new Connector]
        set queue_ $q
        set link_ [new $lltype]
        set ttl_ [new TTLChecker]
```



$ns duplex-link $n0 $n1 1Mb 10ms DropTail

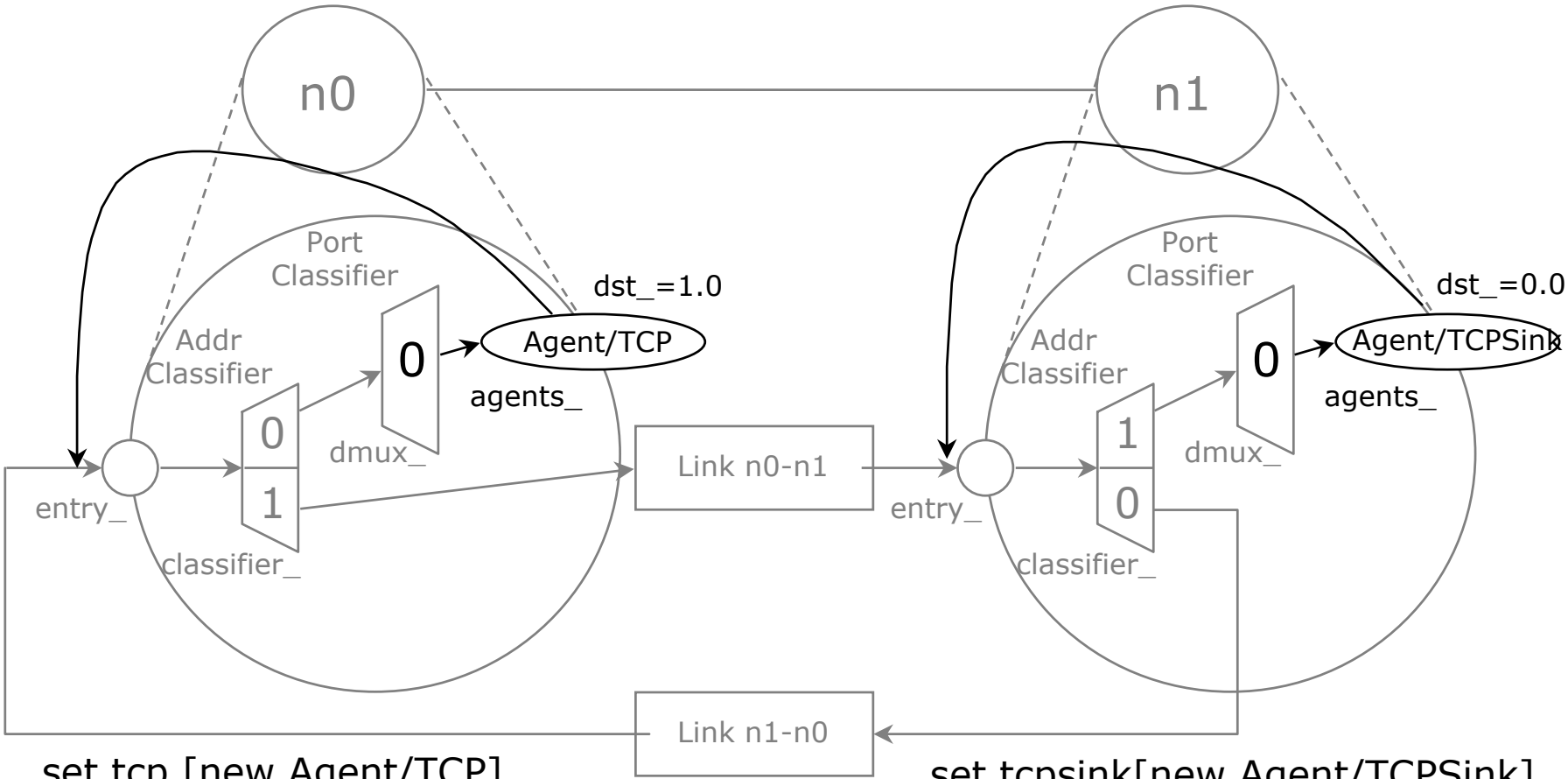
Network Topology - Link



Queue

- Location where packets may be held or dropped
  - Packet scheduling –CBQ, FQ, SFQ, DRR
  - Buffer Management –drop-tail(FIFO), RED
  - Defined in $ns/queue/queue.{h, cc}
- Inherit class Connector
- Virtual functions of enqueand deque
  - recvcalls enque
  - resumecalls dequeand target_->rec

Transport



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

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

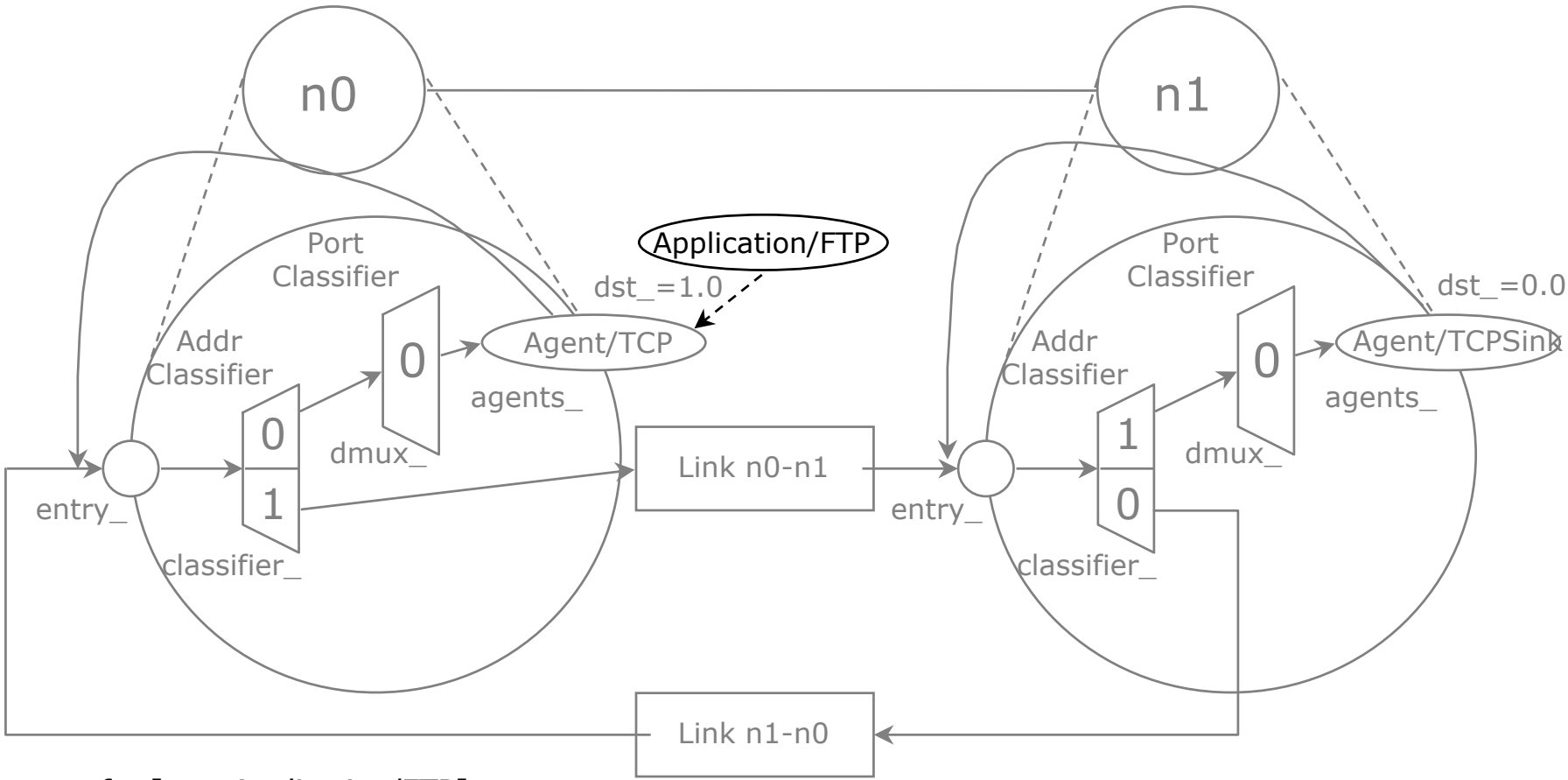
Agent

- Defined in
  - $ns/common/agent.{h,cc} & $ns/tcl/lib/ns-agent.tcl
- Internal states
  - Node address
  - Destination address
  - Fixed packet size
  - Type in packet header (protocol type)
  - IP flow ID, priority, Flags

Agent

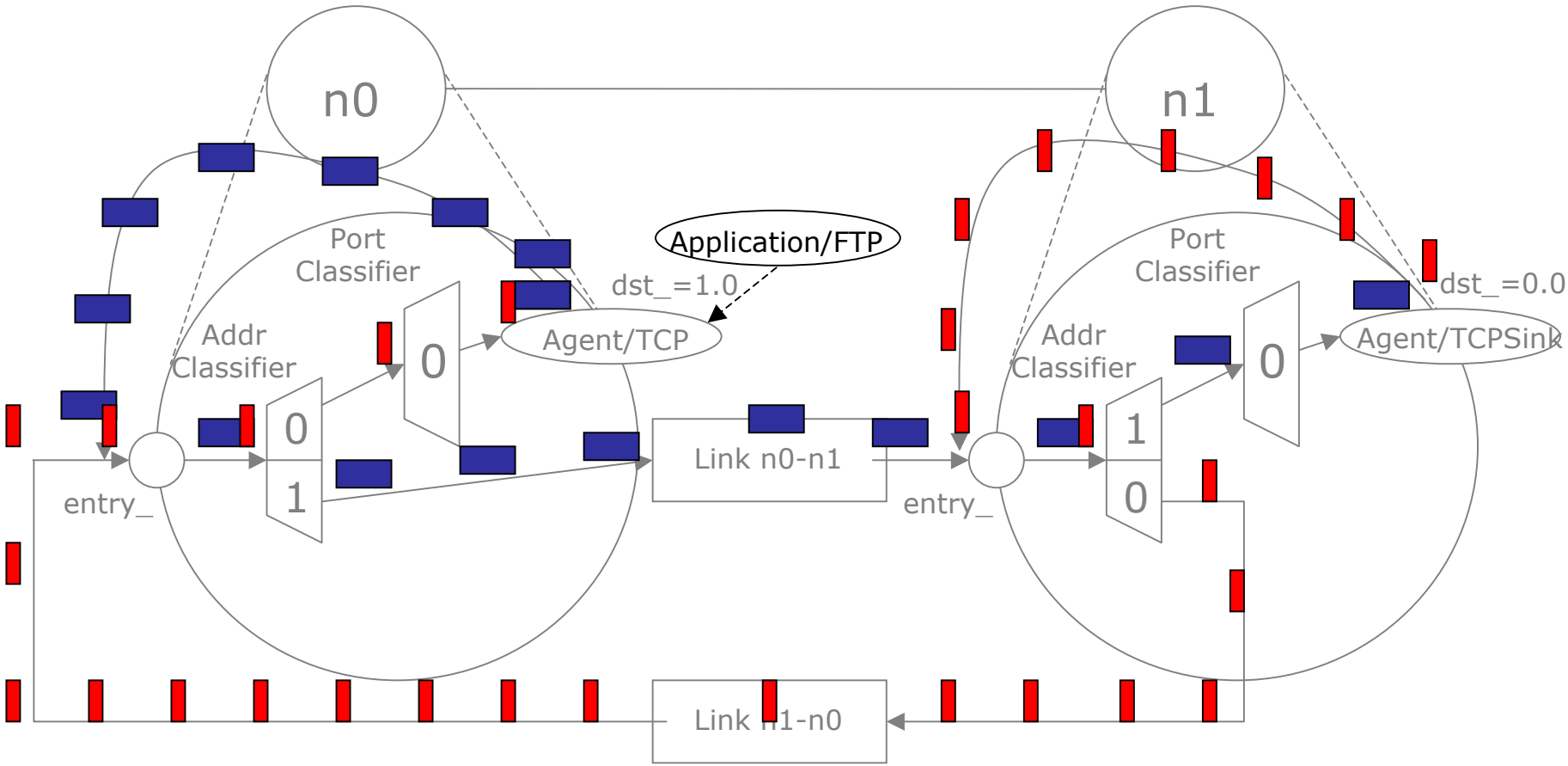
- Packet allocation
  - allocpkt()
- Main entry point
  - recv(Packet*, Handler*)
    - For incoming and/or outgoing packets
    - TcpAgentfor incoming ACK, PingAgentfor both
  - send(Packet* p, Handler* h) { target_->recv(p, h); }
    - Packet forwarding

Application : Traffic Generator




```
set ftp [new Application/FTP]
$tcp attach-agent $ftp
```

Packet Flow


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

Wireless Simulation

Ad hoc network

Wireless Simulation in NS

- Contributed from CMU's Monarch project (Wireless extension to ns-2)
- Various modules were added to ns-2 to simulate node mobility and wireless networking
  - Mobile Node
  - Ad-hoc Routing(DSR, DSDV, TORA, AODV)
  - MAC802.11
  - Radio Propagation Model
  - Channel

Mobile Node in Wireless

- Mobile Node inherits from Node object.
  - MobileNode = Node + ability to move +
    - Link layer, Interface queue, MAC, PHY
    - Radio propagation model
    - Type of ad-hoc routing
    - Type of antenna
- Receive/transmit signals from a wireless channel.
  - No link object is used

Ad Hoc Routing – An Example

- Scenario
  - 2 mobile nodes
  - moving within 500mX500m flat topology
  - One moving from left to right, the other vice versa with TCP traffic
  - using DSDV ad hoc routing protocol

- Examples:
  - ns-2/ns-tutorial/examples/simple-wireless.tcl
  - ns-2/tcl/ex/wireless-demo-csci694.tcl

Additional setup for wireless simulation

- node configuration
  - layer 3-2,layer 1
  - Tracing
  - energy
- node coordinates
- node movements

Create simulator

```
# create simulator
set ns_ [new Simulator]

# create a topology in a 500m x 500m area
set topo [new Topography]
$topo load_flatgrid 500 500
```

Set up trace

ns trace

```
set tracefd  [open demo.tr w]
$ns_ trace-all $tracefd
```

nam trace

```
set namtrace [open demo.nam w]
$ns_ namtrace-all-wireless $namtrace 500 500
```

God

Create God(General Operation Director)

set god [create-god 2]

- God: Stores all-pairs Dijkstra shortest path lengths
- Allows comparison of path length with optimal
- Automatically generated by scenario file

Mobile nodes configuration

```
# Define how a mobile node should be created
# L2, L3 configuration
$ns_ node-config                    \
        -adhocRouting DSDV    \   ;# L3
        -llType LL                  \   ;# L2
        -macType Mac/802_11 \   ;# L2
        -ifqLen 50                  \   ;# L2
        -ifqType Queue/DropTail/PriQueue \ ;# L2
```

Mobile nodes configuration

```
# Define how a mobile node should be created
# L1 configuration
$ns_ node-config                                \
   -antType Antenna/OmniAntenna                 \
   -propType Propagation/TwoRayGround           \
   -phyType Phy/WirelessPhy                     \
   -channelType Channel/WirelessChannel         \
   -topoInstance $topo                          \
   -agentTrace ON                               \
   -routerTrace OFF                             \
   -macTrace OFF
```

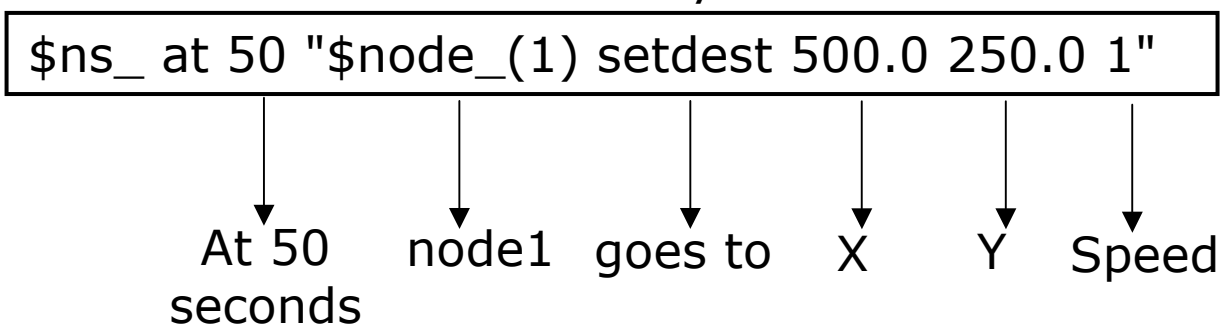
Set up nodes

```
# Set of Nodes
# Use "for" loop to create 2 or more nodes:
   for {set i < 0} {$i < 2} {incr i} {
       set node_($i) [$ns_ node]
            $node_($i) random-motion 0
}
```

Node movement

```
#start position
$node_(0) set X_ 5.0; $node_(0) set Y_ 2.0; $node_(0) set Z_ 0.0
$node_(1) set X_ 390.0;$node_(1) set Y_ 385.0;$node_(1) set Z_ 0.0
#node(1) move towards node(0)
$ns_ at 50.0 "$node_(1) setdest 25.0 20.0 15.0"
$ns_ at 10.0 "$node_(0) setdest 20.0 18.0 1.0“
#node(1) move away from node(0)
$ns_ at 100.0 "$node_(1) setdest 490.0 480.0 15.0"
```

Format of mobility commands



Node movement

- Disable random movement
  - $mnode random-motion 0
- Random movement
  - $ns at 1.0 “$mnode start”
- Specified movement
  - $ns at 1.0 “$mnode setdest <x> <y> <speed>”

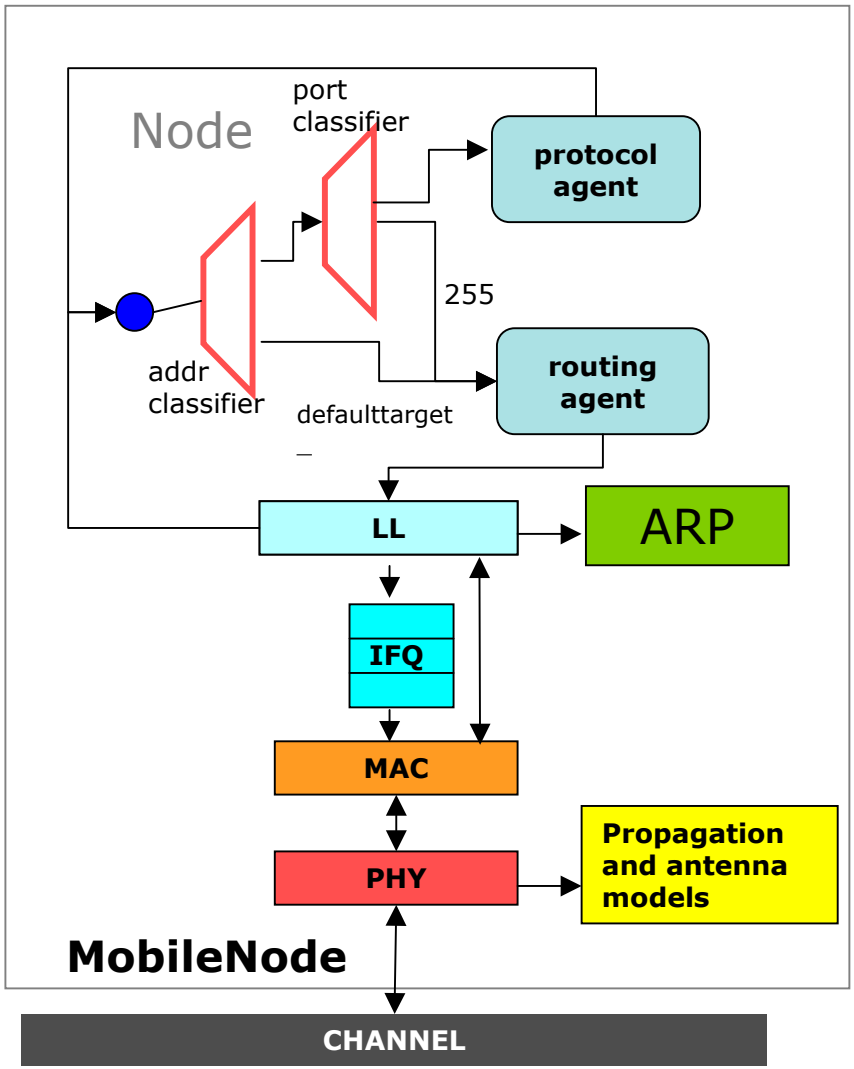
Traffic Scenario

- TCP connection from node 0 to node 1
  - set tcp [new Agent/TCP]
  - set sink [new Agent/TCPSink]
  - $ns_ attach-agent $node_(0) $tcp
  - $ns_ attach-agent $node_(1) $sink
  - $ns_ connect $tcp $sink
- Create data source
  - set ftp [new Application/FTP]
  - $ftp attach-agent $tcp
  - $ns_ at 10.0 "$ftp start"

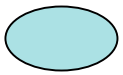
Running simulation

```
# Define node initial position in nam
for {set i 0} {$i < 2 } { incr i} {
    $ns_ initial_node_position $node($i) 20
}
set val_(stop)  50.0
# Tell ns/nam the simulation stop time
for { set i 0} { $i < 2} {incr I} {
  $ns_ at  $val(stop).0 “$node_($I) reset”;
}
$ns_ at $val_(stop).0001 “stop”
$ns_ at $val_(stop).0002 “$ns_ halt”

proc stop { } {
    global ns_ tracefd
    close $tracefd
}

# Start your simulation
$ns_ run
```

Mobile Node

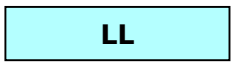


Classifier: Forwarding



Agent: Protocol Entity

Node Entry



LL: Link layer object

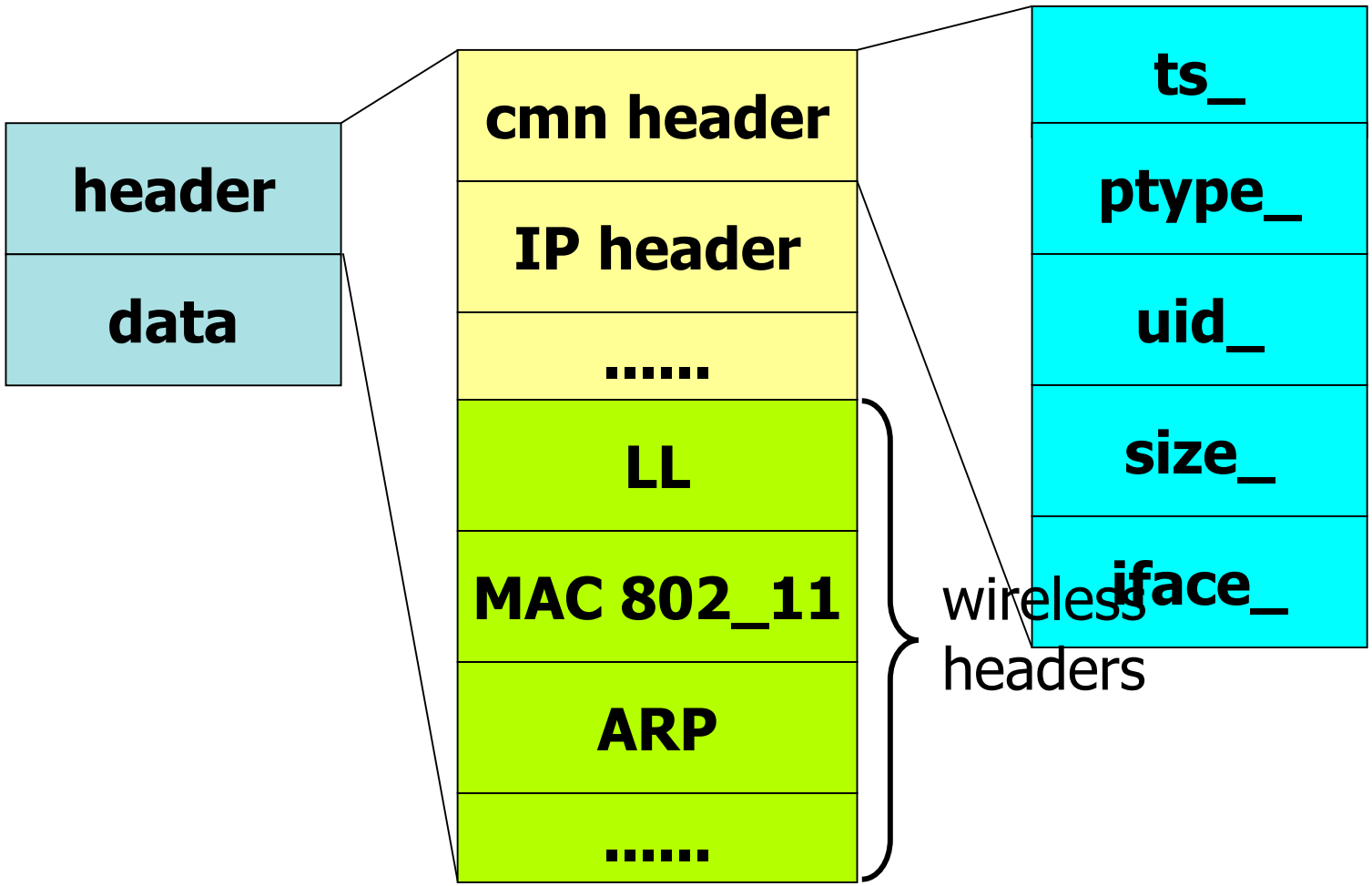
IFQ: Interface queue

MAC: Mac object

PHY: Net interface

Radio propagation/ antenna models

Wireless packet header

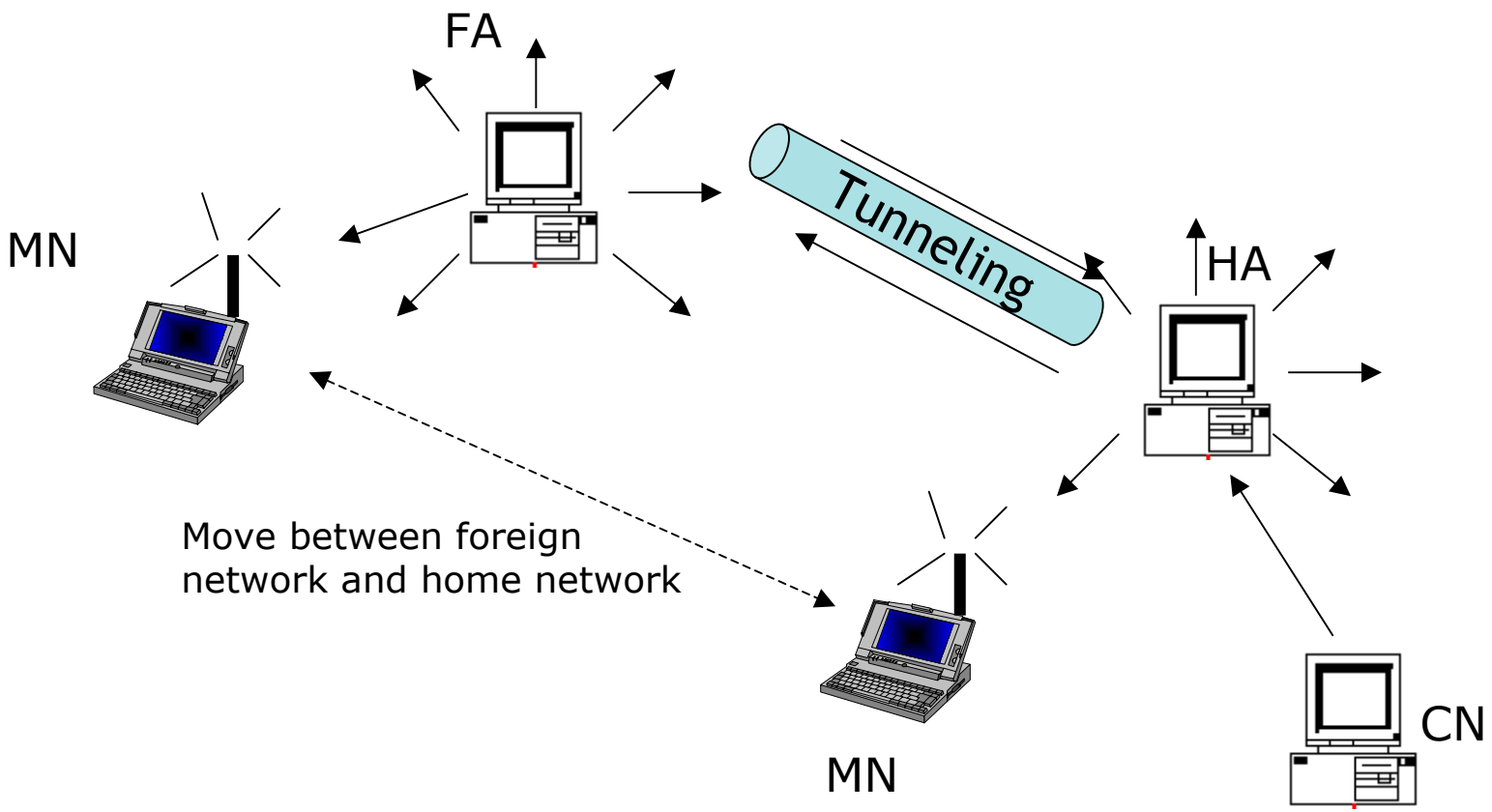


Mobile IP Simulation

Mobile IP Support

- Developed by Sun
  - Require a different Node structure than the MobileNode
  - Co-exists with wired world in ns
- Standard Mobile IP
  - Home Agent, Foreign Agent, MobileHosts…

Mobile IP – Demo Scenario



Reference

- Official ns Manual
- Padma Haldar's tutorial
- marc greis's Ns tutorial
- Ns by example
- Changhee joo's Ns tutorial

  – Examples , figures in this presentation are largely borrowed above materials

Q & A

Thanks for your attentions