An Introduction to NS-2[*]

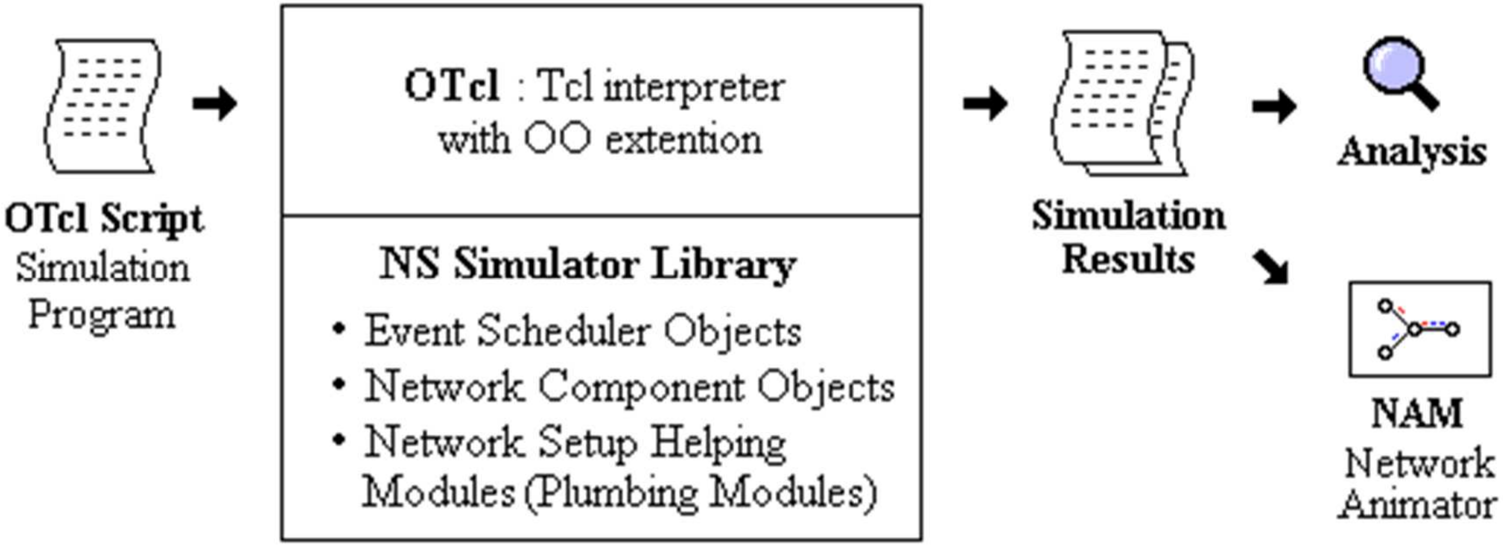
Roadmap For Today’s Lecture

1. ns Primer
2. Extending ns

Part I: ns Primer

What is ns?

- Object-oriented, discrete event-driven network simulator
- Written in C++ and OTcl
- By VINT: Virtual InterNet Testbed

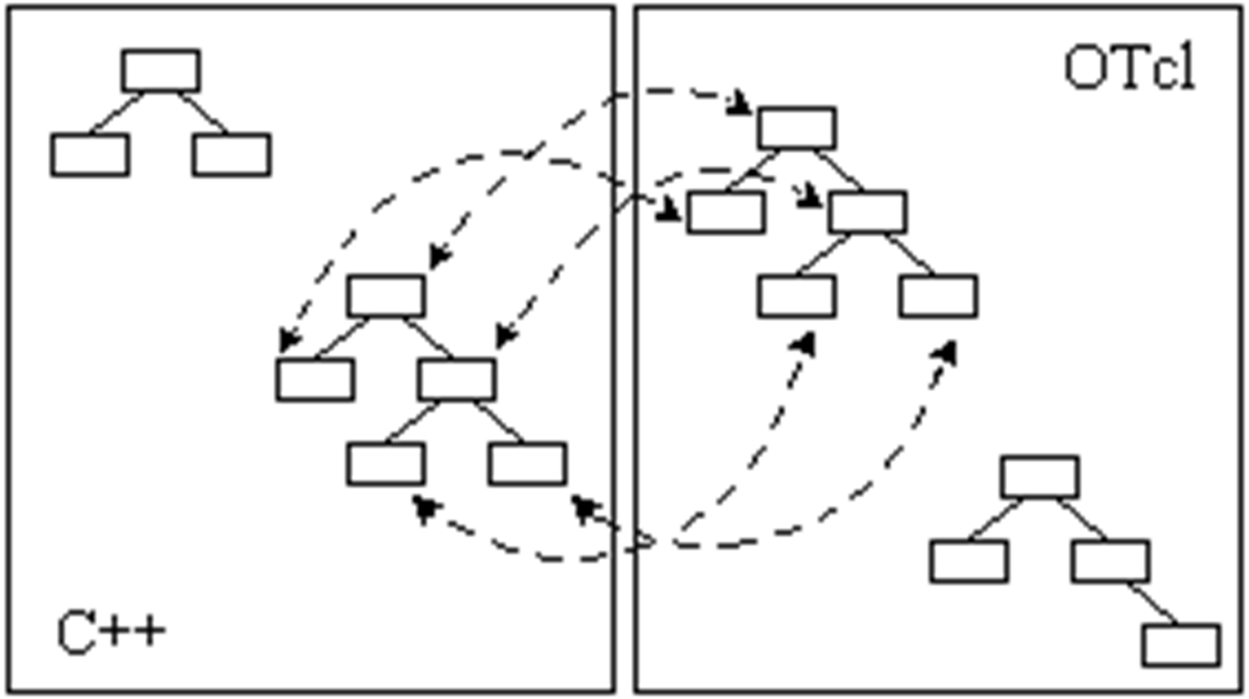

OTcl Script
Simulation Program
OTcl : Tcl interpreter with OO extention
NS Simulator Library
• Event Scheduler Objects
• Network Component Objects
• Network Setup Helping Modules (Plumbing Modules)
Simulation Results
Analysis
NAM
Network Animator

ns Architecture

- Separate data path and control path implementations.

ns Architecture

- Separate data path and control path implementations.



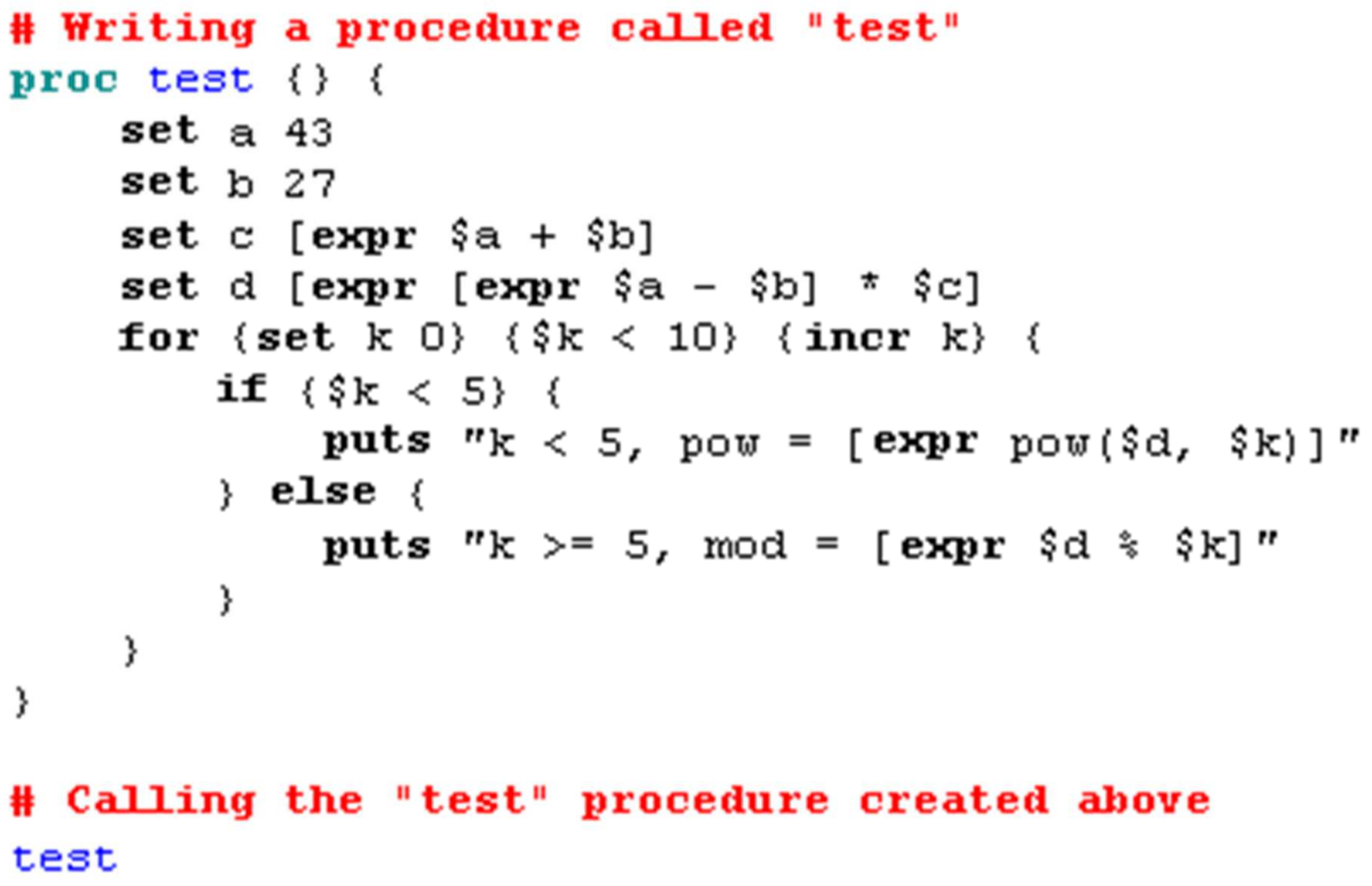
Hello World – Interactive mode

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

Hello World – Batch mode

```
simple.tcl
   set ns [new Simulator]
   $ns at 1 “puts \“Hello World!\””
   $ns at 1.5 “exit”
   $ns run
bash-shell$ ns simple.tcl
Hello World!
bash-shell$
```

Basic Tcl: ex-tcl.tcl



```
# Writing a procedure called "test"
proc test {} {
    set a 43
    set b 27
    set c [expr $a + $b]
    set d [expr [expr $a - $b] * $c]
    for {set k 0} {$k < 10} {incr k} {
        if {$k < 5} {
            puts "k < 5, pow = [expr pow($d, $k)]"
        } else {
            puts "k >= 5, mod = [expr $d % $k]"
        }
    }
}

# Calling the "test" procedure created above
test
```

NS-2 Generic Script Structure

1. Create Simulator object
2. [Turn on tracing]
3. Create network topology
4. [Setup packet loss, link dynamics]
5. Create routing agents
6. Create application and/or traffic sources
7. Post-processing procedures (i.e. nam)
8. Start simulation

Step1: Create Simulator Object

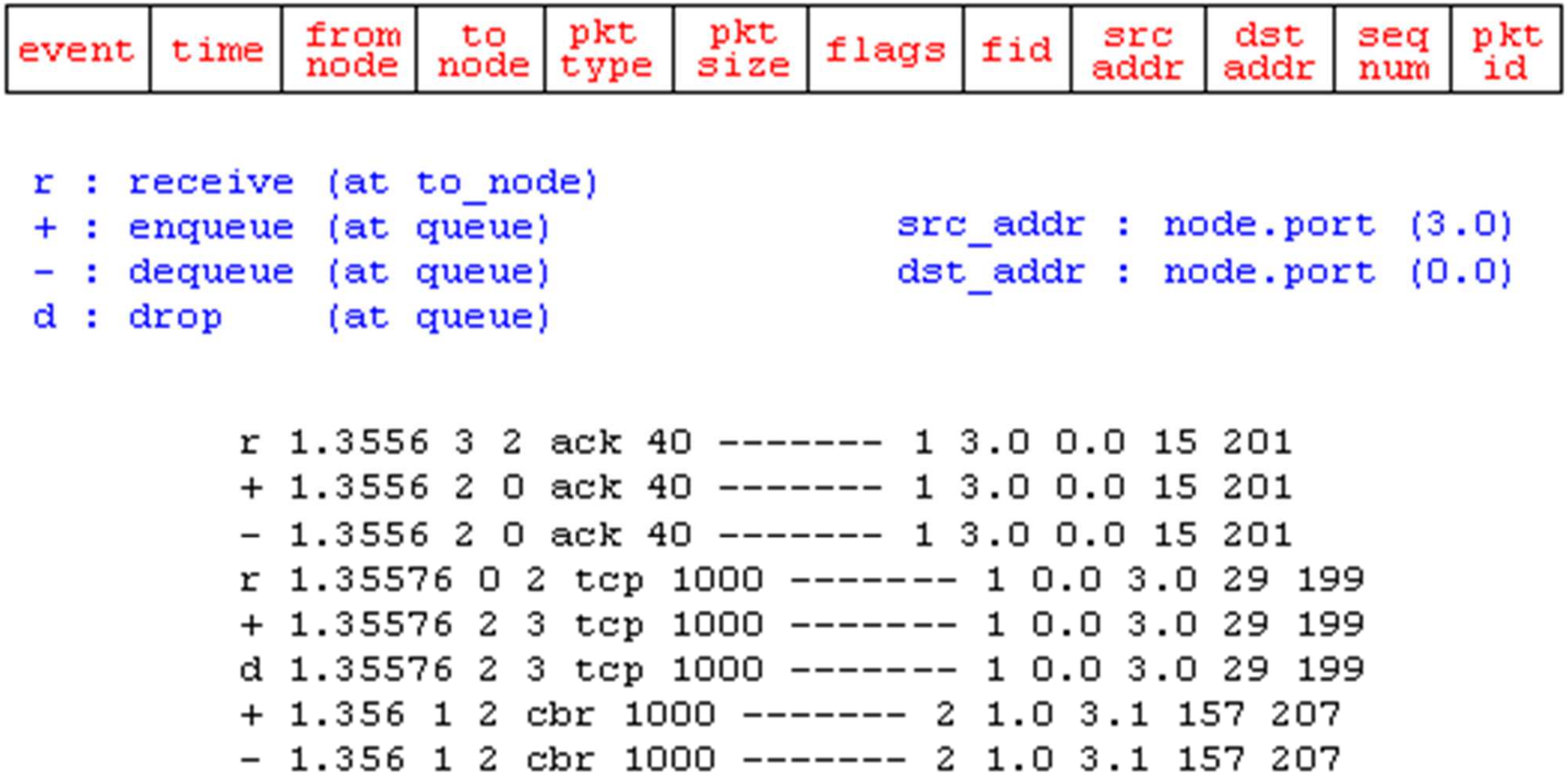
- Create event scheduler
  - `set ns [new Simulator]`

Step2: Tracing

- Insert immediately after scheduler!
- Trace packets on all links

```
set nf [open out.nam w]
$ns trace-all $nf

$ns namtrace-all $nf
```

Step2: Tracing



event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

```
r : receive (at to_node)
+ : enqueue (at queue)
- : dequeue (at queue)
d : drop    (at queue)
```

```
src_addr : node.port (3.0)
dst_addr : node.port (0.0)
```

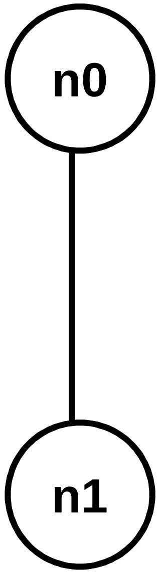
```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

NS-2 Generic Script Structure

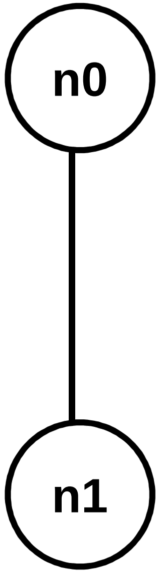
1. Create Simulator object
2. [Turn on tracing]
3. Create topology
4. [Setup packet loss, link dynamics]
5. Create routing agents
6. Create application and/or traffic sources
7. Post-processing procedures (i.e. nam)
8. Start simulation

Step 3: Create network

- Two nodes, One link



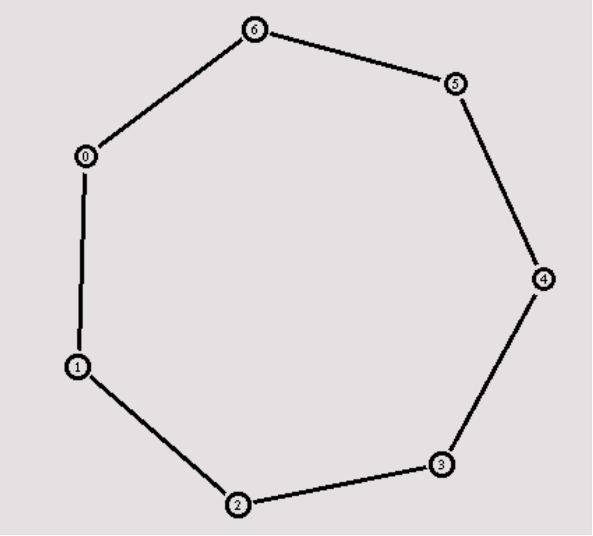
Step 3: Create Network



- Nodes
  - ❑ `set n0 [$ns node]`
  - ❑ `set n1 [$ns node]`
- Links and queuing
  - ❑ `$ns duplex-link $n0 $n1 1Mb 10ms RED`
  - ❑ $ns duplex-link $n0 $n1 <capacity> <delay> <queue_type>
  - ❑ <queue_type>: DropTail, RED, SFQ, etc.

Creating a larger topology

```
for {set i 0} {$i < 7} {incr i} {
set n($i) [$ns node]
}
for {set i 0} {$i < 7} {incr i} {
$ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms RED
}
```



NS-2 Generic Script Structure

1. Create Simulator object
2. [Turn on tracing]
3. Create topology
4. [Setup packet loss, link dynamics]
5. Create routing agents
6. Create application and/or traffic sources
7. Post-processing procedures (i.e. nam)
8. Start simulation

Step 4: Network Dynamics

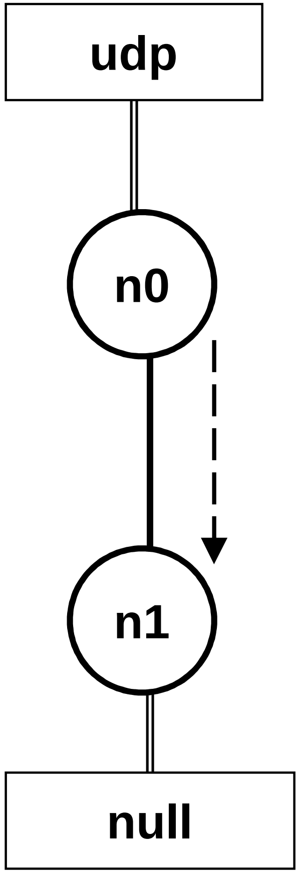
- Link failures
  - Hooks in routing module to reflect routing changes
- `$ns rtmodel-at <time> up|down $n0 $n1`

- For example:

```
$ns rtmodel-at 1.0 down $n0 $n1
$ns rtmodel-at 2.0 up $n0 $n1
```

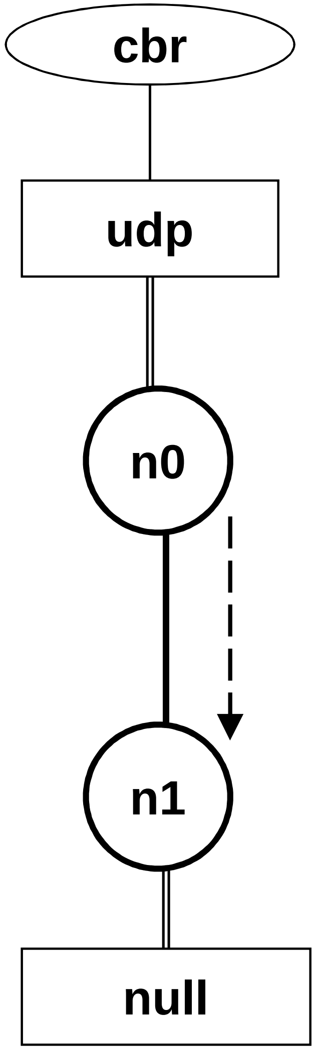
Step 5: Creating UDP connection

```
set udp [new Agent/UDP]
set null [new Agent/Null]

$ns attach-agent $n0 $udp
$ns attach-agent $n1 $null

$ns connect $udp $null
```



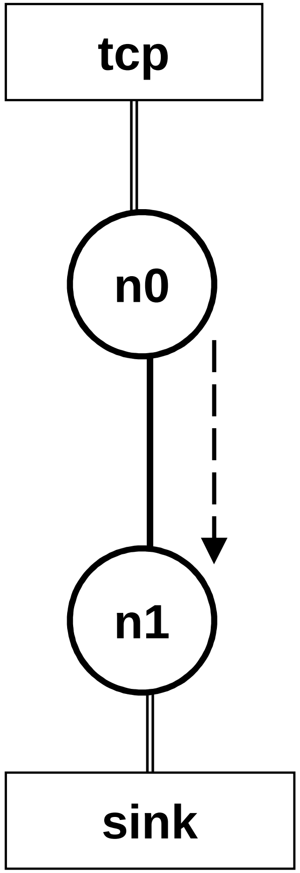
Step 6: Creating Traffic (On Top of UDP)

- CBR
  - ❑ `set cbr [new Application/Traffic/CBR]`
  - ❑ `$cbr set packetSize_ 500`
  - ❑ `$cbr set interval_ 0.005`
  - ❑ `$cbr attach-agent $udp`

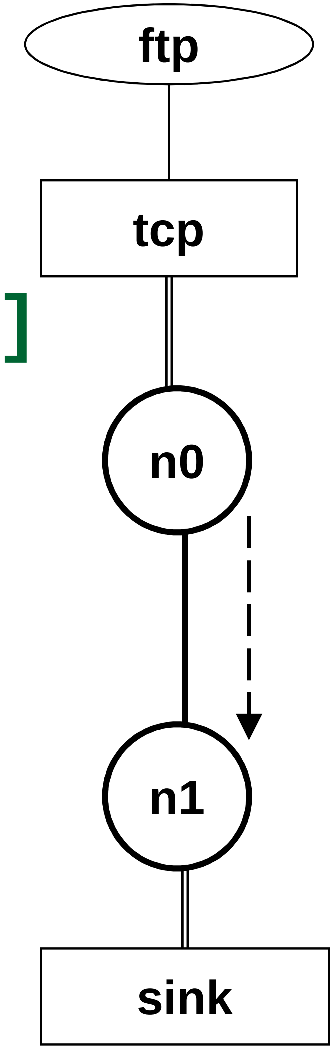


Creating TCP connection

```
set tcp [new Agent/TCP]
set tcpsink [new Agent/TCPSink]

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

$ns connect $tcp $tcpsink
```



Step 6: Creating Traffic (On Top of TCP)



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

Recall: Generic Script Structure

1. set ns [new Simulator]
2. [Turn on tracing]
3. Create topology
4. [Setup packet loss, link dynamics]
5. Create agents
6. Create application and/or traffic sources
7. Post-processing procedures (i.e. nam)
8. Start simulation

Examples

Post-Processing Procedures

- Add a 'finish' procedure that closes the trace file and starts nam.

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

Run Simulation

- Schedule Events

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

  - <event>: any legitimate ns/tcl command

```
$ns at 0.5 "$cbr start"
$ns at 4.5 "$cbr stop“
```

- Call ‘finish’

```
$ns at 5.0 "finish"
```

- Run the simulation

```
$ns run
```

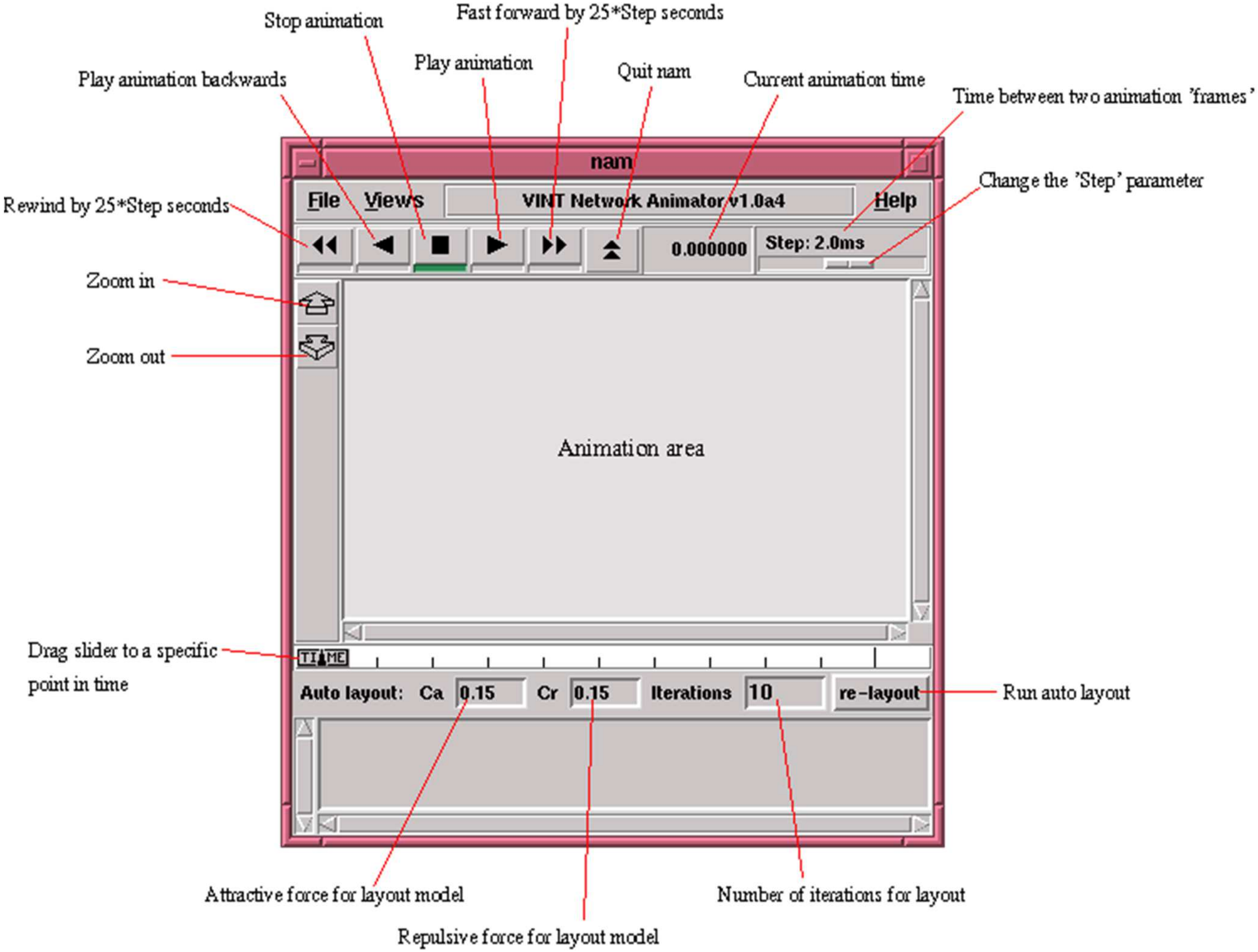
Recall: Generic Script Structure

1. set ns [new Simulator]
2. [Turn on tracing]
3. Create topology
4. [Setup packet loss, link dynamics]
5. Create routing agents
6. Create application and/or traffic sources
7. Post-processing procedures (i.e. nam)
8. Start simulation

Examples

Visualization Tools

- nam (Network AniMator)
  - Packet-level animation
  - Well supported by ns
- Xgraph (Matlab, Excel …)
  - Simulation results



0.000000

Step: 2.0ms

Auto layout: Ca 0.15 Cr 0.15 Iterations 10

Stop animation

Fast forward by 25*Step seconds

Play animation

Play animation backwards

Quit nam

Current animation time

Time between two animation 'frames'

Change the 'Step' parameter

Rewind by 25*Step seconds

Zoom in

Zoom out

Animation area

Drag slider to a specific point in time

Run auto layout

Attractive force for layout model

Repulsive force for layout model

Number of iterations for layout

nam Interface: Nodes

- Color

```
$node color red
```

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

```
$node shape box (circle, box, hexagon)
```

- 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
  “backbone"
```

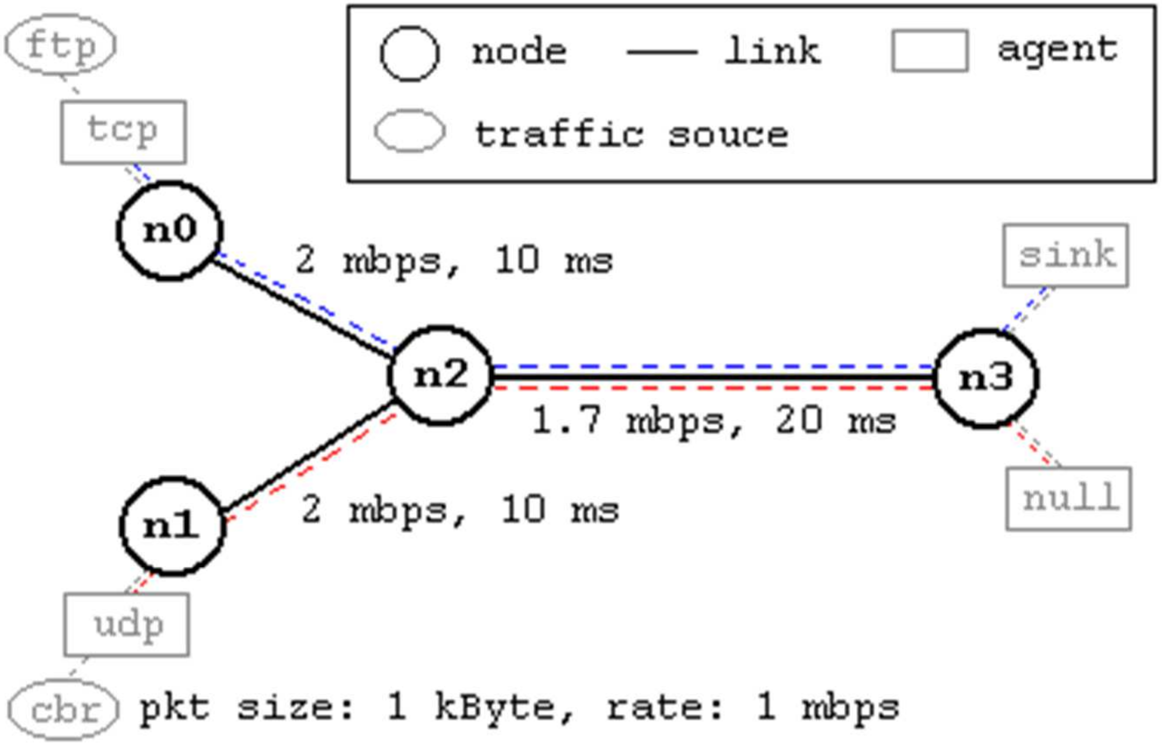
nam Interface: Topology Layout

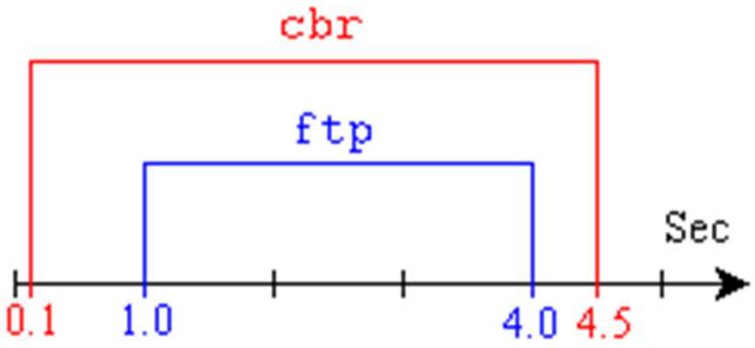
- “Manual” layout: specify everything

```
$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

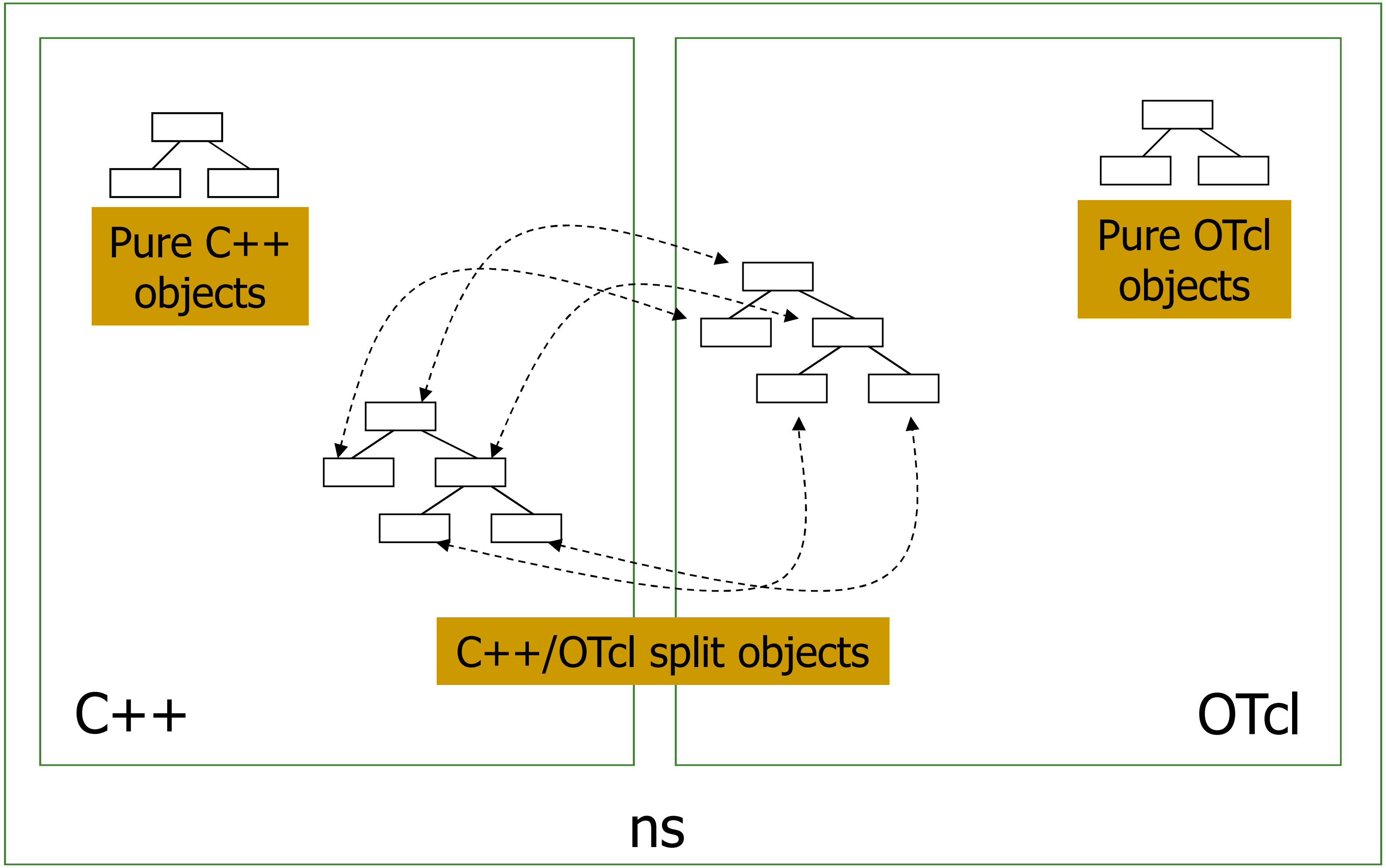
Simulation Example



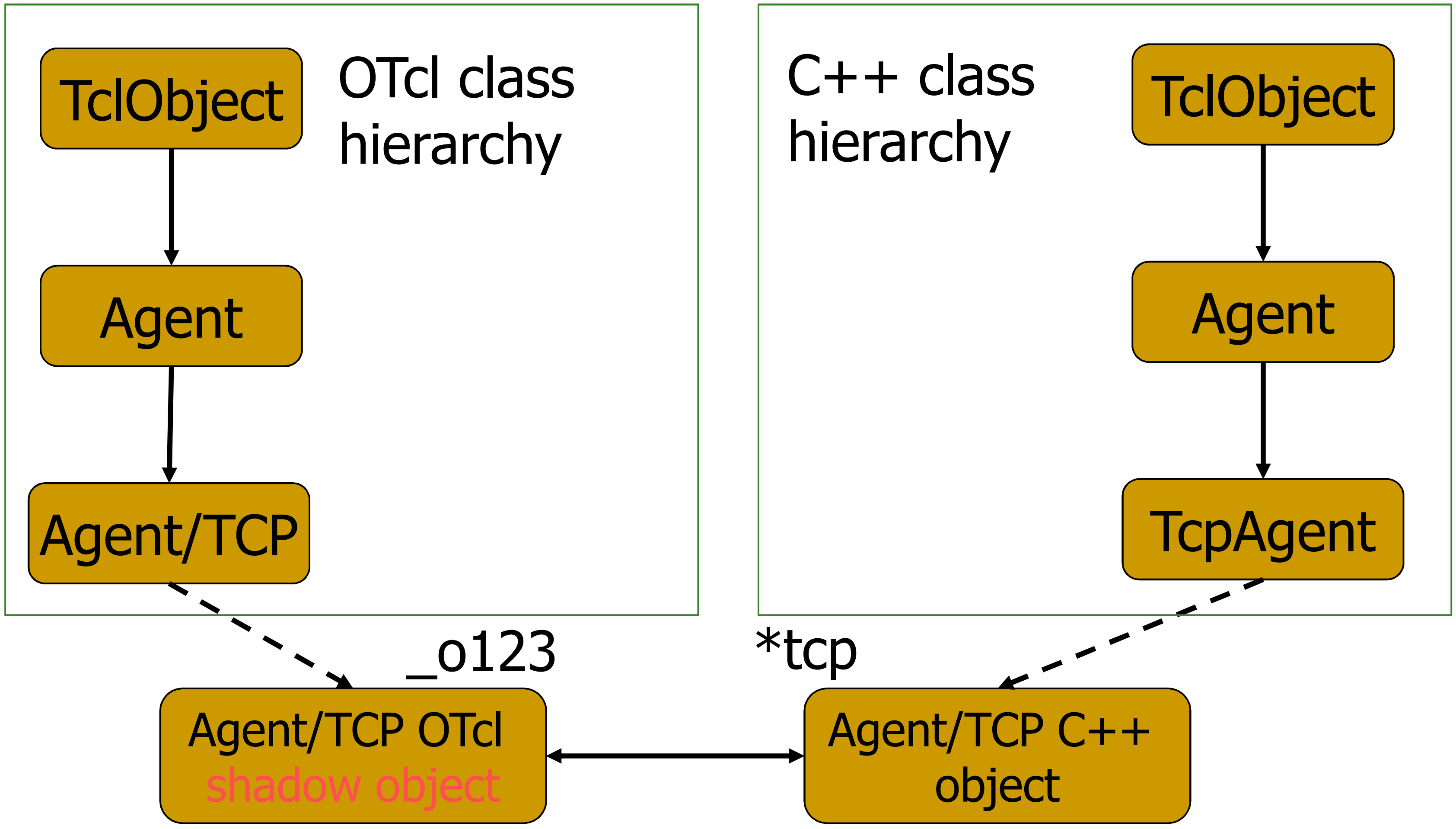


Part II: Extending ns

OTcl and C++: The Duality

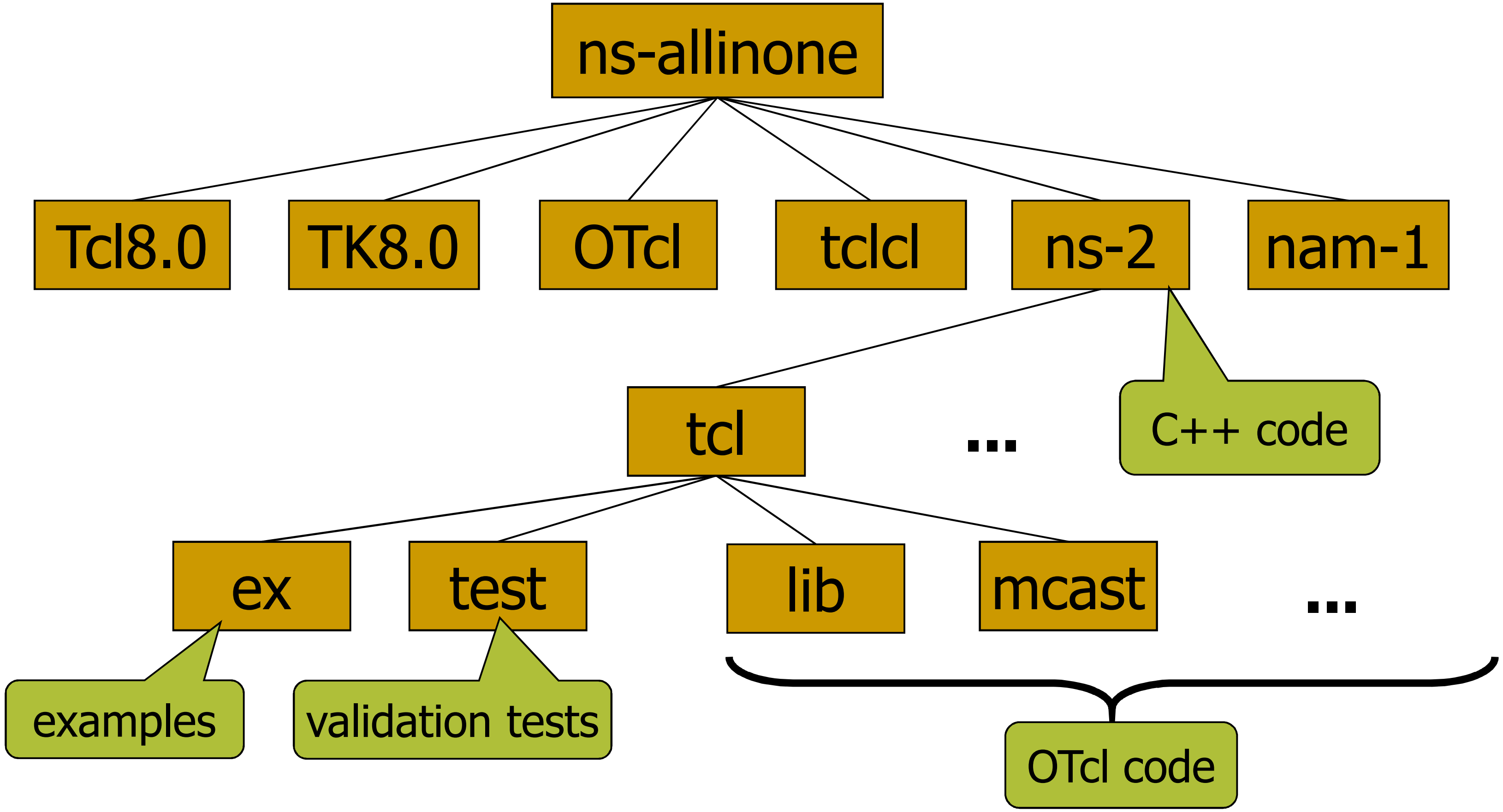


TclObject: Hierarchy and Shadowing



Extending ns

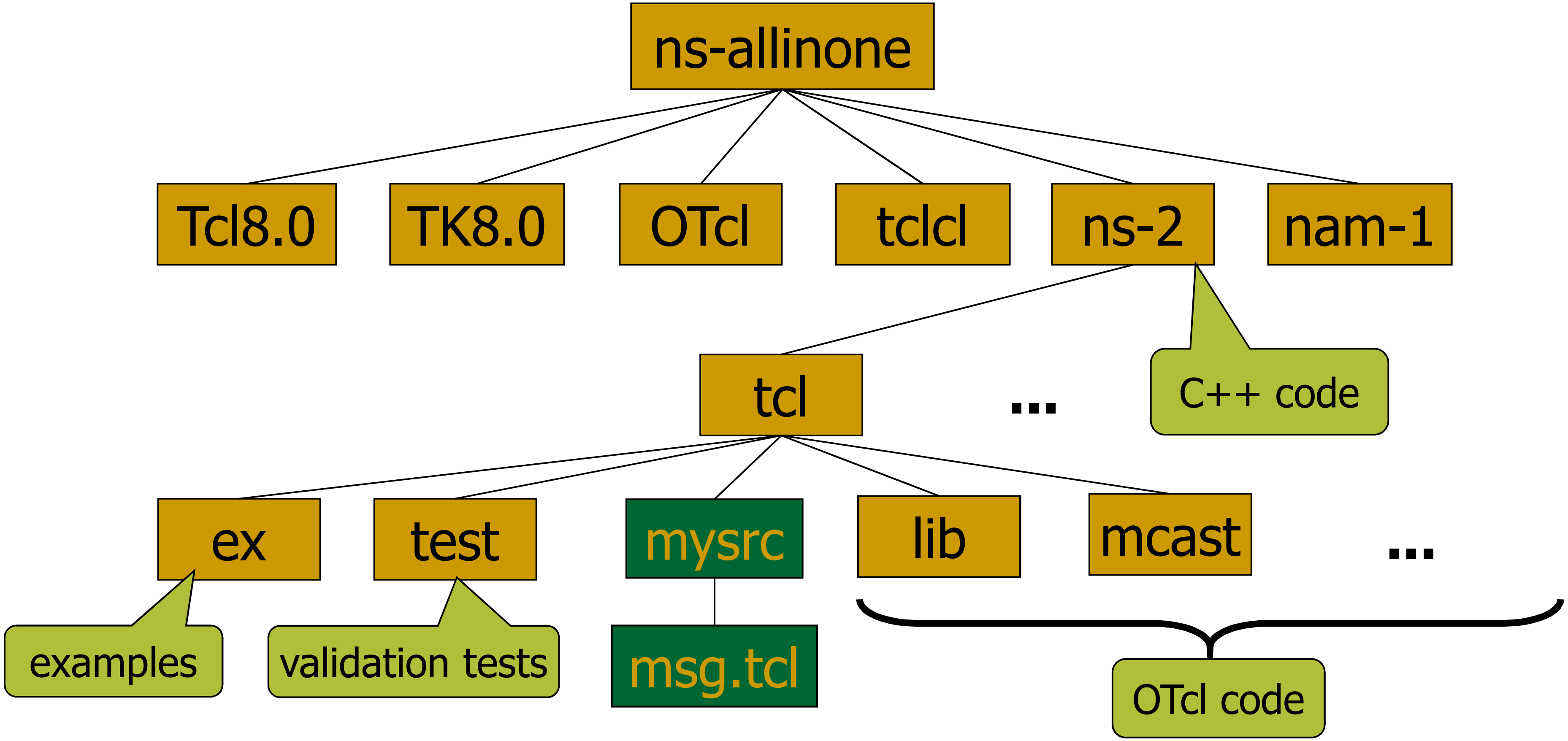
- In OTcl
- In C++



Extending ns in OTcl

- If you don’t want to compile
  - source your changes in your sim scripts
- Modifying exisiting code
  - Recompile
- Adding new files
  - Change Makefile (NS_TCL_LIB),
  - Update tcl/lib/ns-lib.tcl
  - Recompile

Add Your Changes into ns



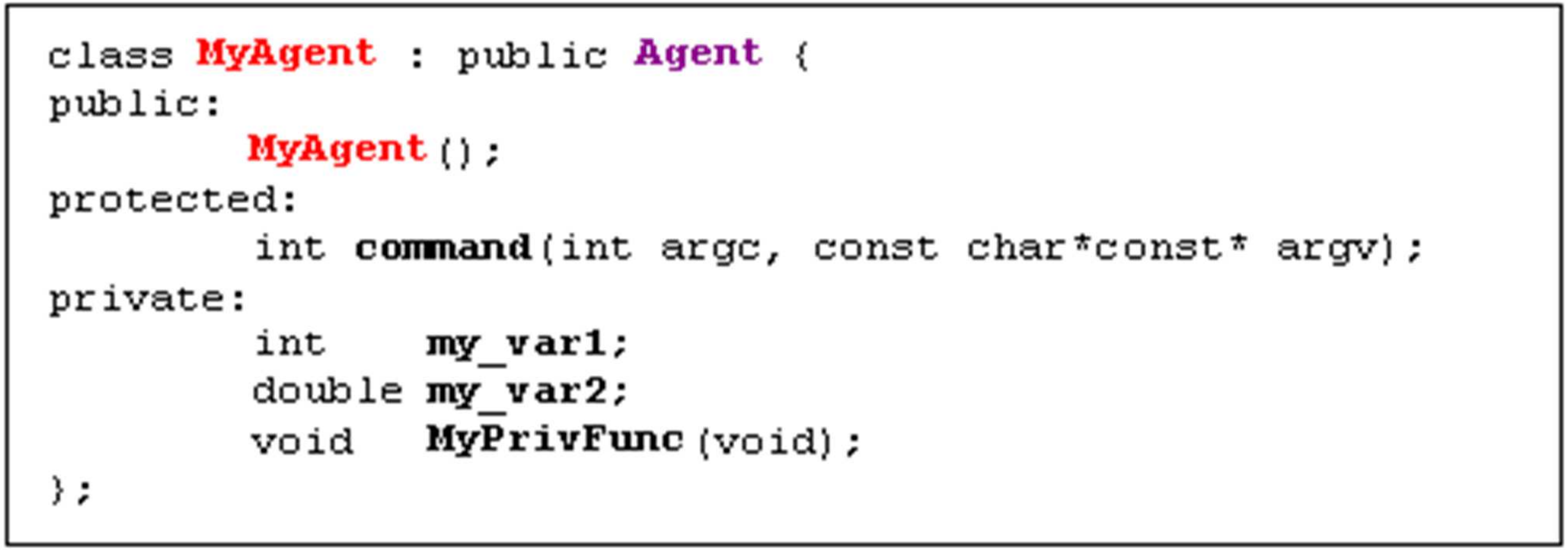
Extending ns in C++

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

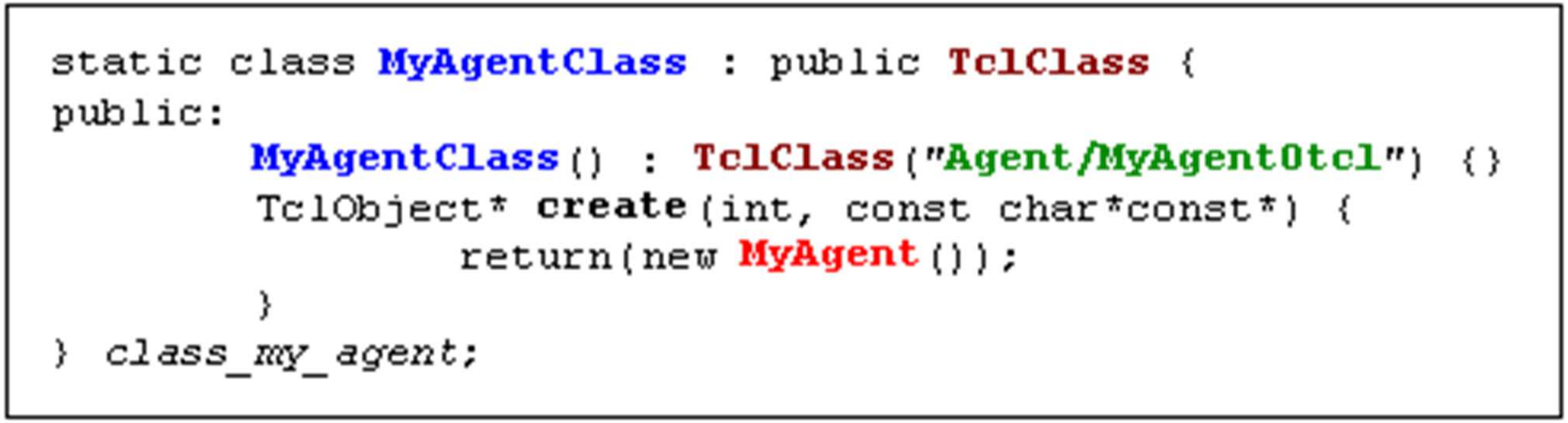
OTcl Linkage

- Lets create a new agent “MyAgent”
  - Dummy agent
  - Derived from the “Agent” class

Step 1: Export C++ class to OTcl

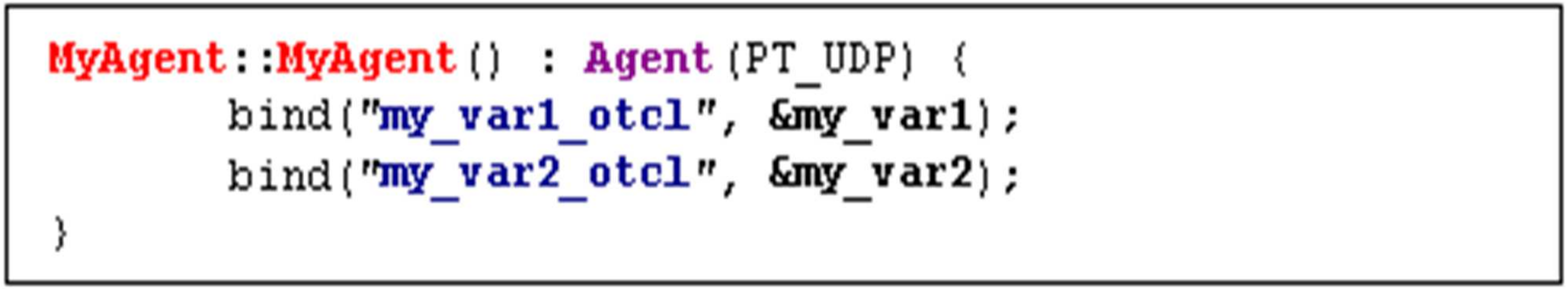


```
class MyAgent : public Agent {
public:
        MyAgent();
protected:
        int command(int argc, const char*const* argv);
private:
        int     my_var1;
        double my_var2;
        void   MyPrivFunc(void);
};
```



```
static class MyAgentClass : public TclClass {
public:
        MyAgentClass() : TclClass("Agent/MyAgentOtcl") {}
        TclObject* create(int, const char*const*) {
                return(new MyAgent());
        }
} class_my_agent;
```

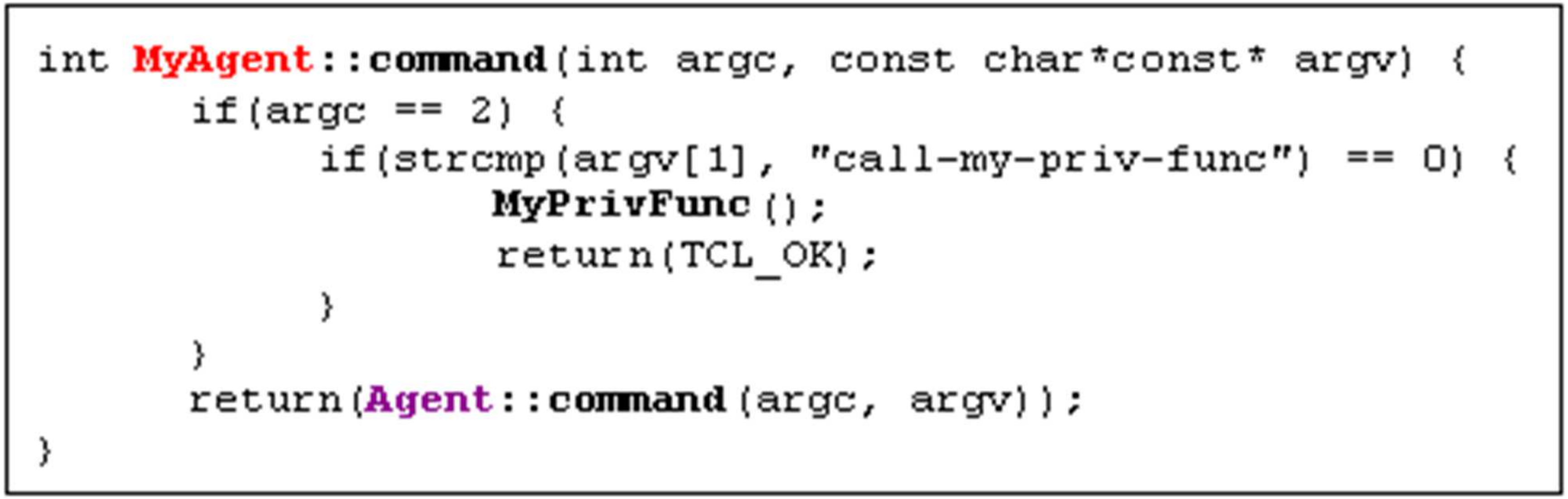
Step 2 : Export C++ class variables to OTcl



```
MyAgent::MyAgent() : Agent(PT_UDP) {
        bind("my_var1_otcl", &my_var1);
        bind("my_var2_otcl", &my_var2);
}
```

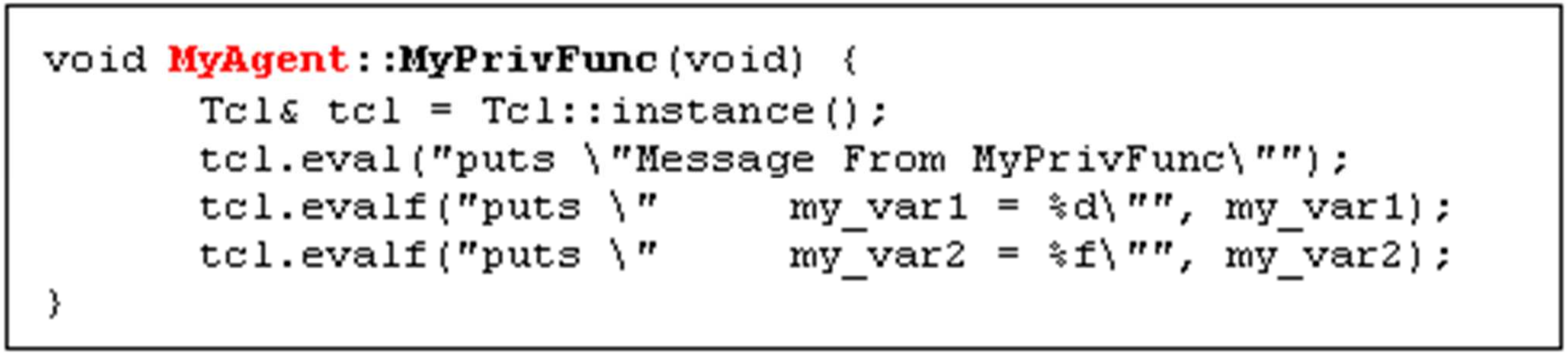
- set the default value for the variables in the "ns-2/tcl/lib/ns-lib.tcl" file

Step 3: Export C++ Object Control Commands to OTcl



```
int MyAgent::command(int argc, const char*const* argv) {
        if(argc == 2) {
                if(strcmp(argv[1], "call-my-priv-func") == 0) {
                        MyPrivFunc();
                        return(TCL_OK);
                }
        }
        return(Agent::command(argc, argv));
}
```

Step 4: Execute an OTcl command from C++.



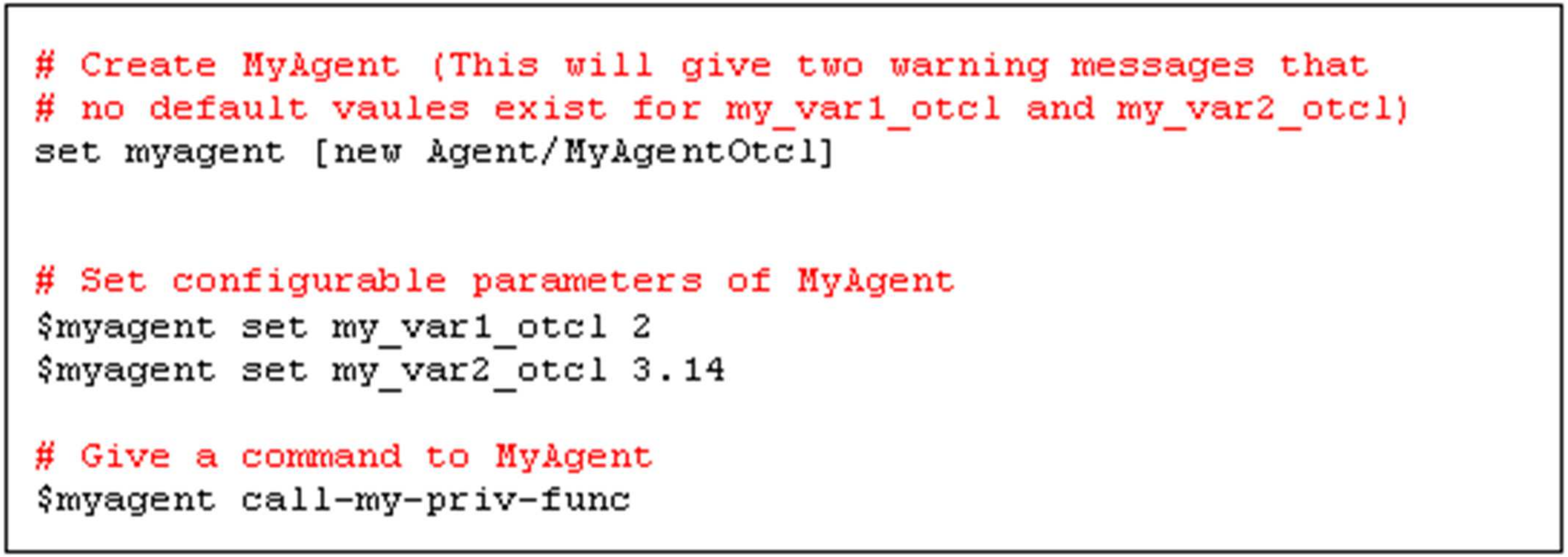
```
void MyAgent::MyPrivFunc(void) {
        Tcl& tcl = Tcl::instance();
        tcl.eval("puts \"Message From MyPrivFunc\"");
        tcl.evalf("puts \"     my_var1 = %d\"", my_var1);
        tcl.evalf("puts \"     my_var2 = %f\"", my_var2);
}
```

Step 5: Compile

- Save above code as “`ex-linkage.cc`”
- Open "`Makefile`", add "`ex-linkage.o`" at the end of object file list.
- Re-compile NS using the "`make`" command.

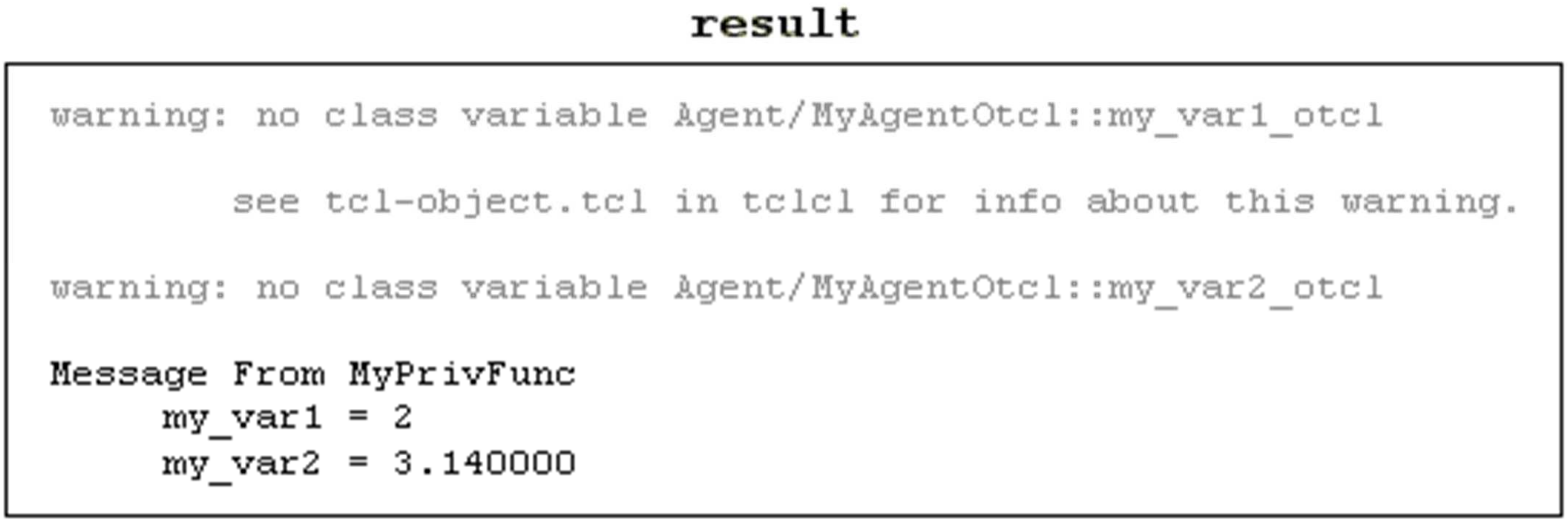
Step 5: Run and Test “MyAgent”

ex-linkage.tcl



```
# Create MyAgent (This will give two warning messages that
# no default vaules exist for my_var1_otcl and my_var2_otcl)
set myagent [new Agent/MyAgentOtcl]


# Set configurable parameters of MyAgent
$myagent set my_var1_otcl 2
$myagent set my_var2_otcl 3.14

# Give a command to MyAgent
$myagent call-my-priv-func
```

Step 5: Run and Test “MyAgent”



result

```
warning: no class variable Agent/MyAgentOtcl::my_var1_otcl

        see tcl-object.tcl in tclcl for info about this warning.

warning: no class variable Agent/MyAgentOtcl::my_var2_otcl

Message From MyPrivFunc
        my_var1 = 2
        my_var2 = 3.140000
```