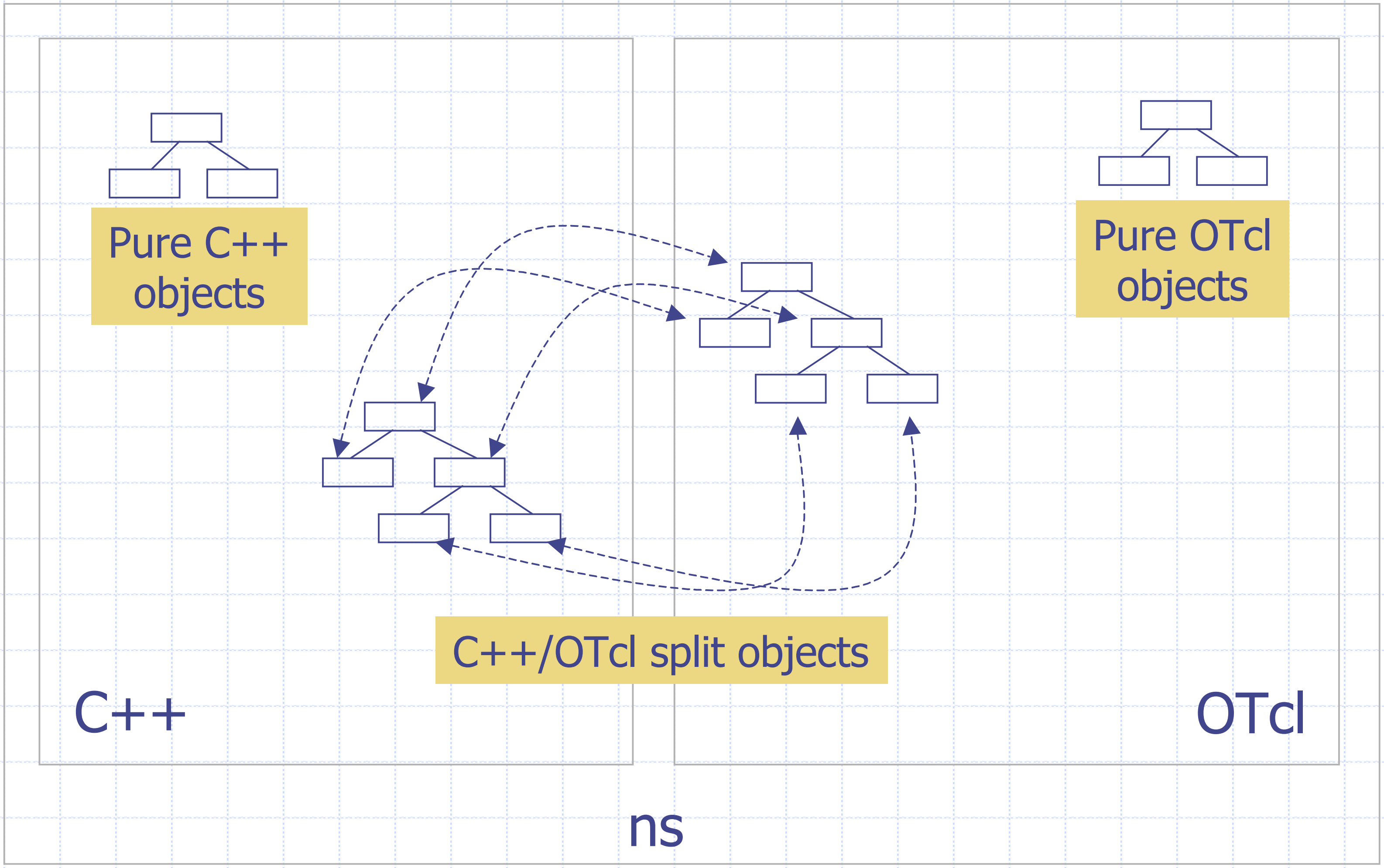
Part II: ns Internals

Outline

- Fundamental concept
  - Split object: C++/OTcl linkage
- Plumbing
  - Wired
  - Wireless
- Scaling

OTcl and C++: The Duality



C++/OTcl Linkage

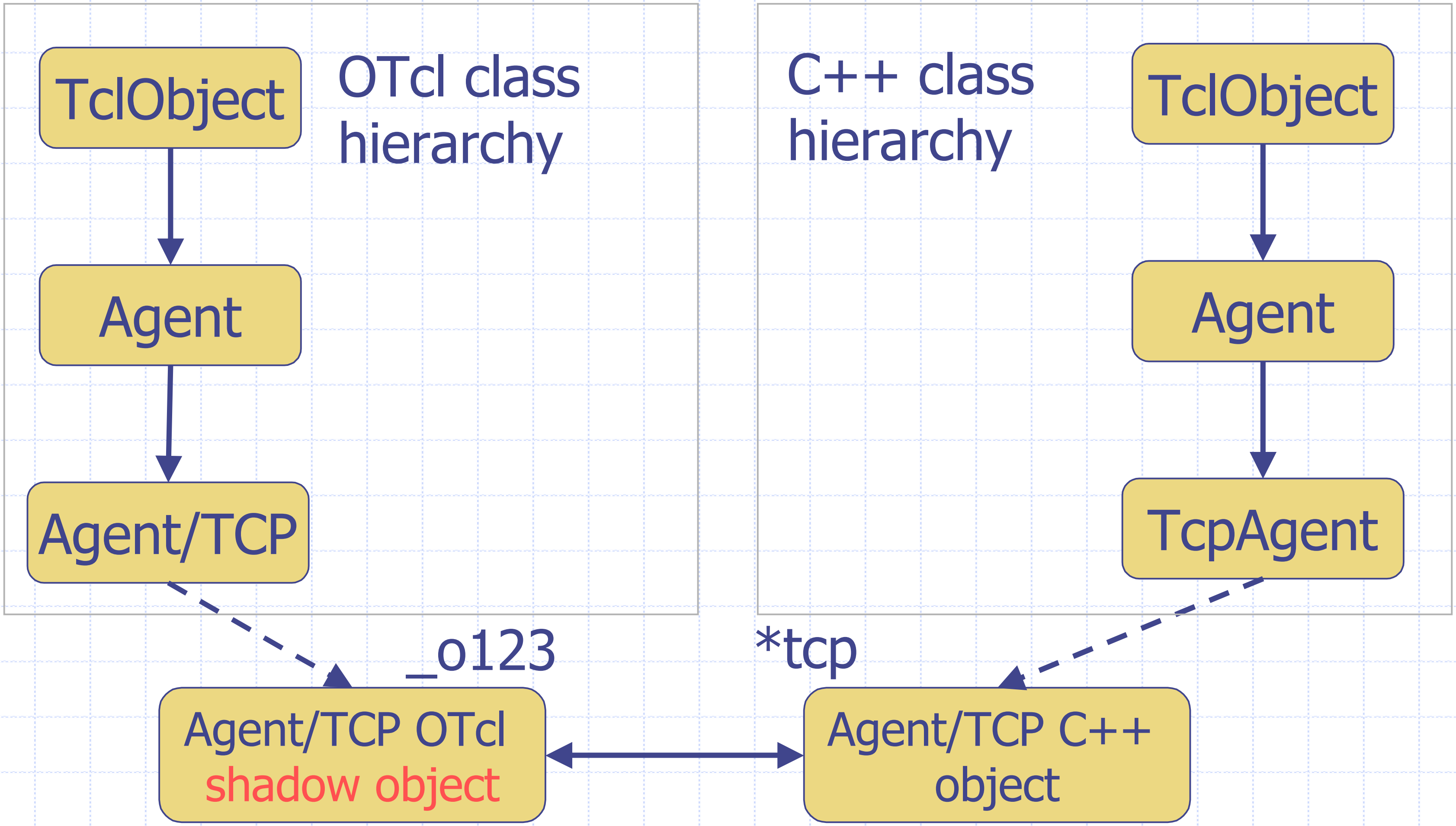
TclObject	Root of ns-2 object hierarchy
	bind(): link variable values between C++ and OTcl
	command(): link OTcl methods to C++ implementations
TclClass	Create and initialize TclObject’s
Tcl	C++ methods to access Tcl interpreter
TclCommand	Standalone global commands
EmbeddedTcl	ns script initialization

TclObject

- Basic hierarchy in ns for split objects
- Mirrored in both C++ and OTcl
- Example

```
set tcp [new Agent/TCP]
$tcp set packetSize_ 1024
$tcp advanceby 5000
```

TclObject: Hierarchy and Shadowing



TclObject::bind()

- Link C++ member variables to OTcl object variables
- C++

```
TcpAgent::TcpAgent() {
   bind(“window_”, &wnd_);
   … …
}
```

  - bind_time(), bind_bool(), bind_bw()
- OTcl

```
set tcp [new Agent/TCP]
$tcp set window_ 200
```

Initialization of Bound Variables

- Initialization through OTcl class variables

```
 Agent/TCP set window_ 50
```

- Do all initialization of bound variables in ~ns/lib/ns-default.tcl
  - Otherwise a warning will be issued when the shadow object is created

Implementation of Bound Variables

- Class InstVar
  - One object per bound variable – expensive!
  - InstVarInt, InstVarReal, …
- Created by TclObject::bind()
  - Create instance variable in OTcl stack
  - Enable trap read/write to OTcl variable using `Tcl_TraceVar()`
  - Connect to C++ variable in the trap

TclObject::command()

- Implement OTcl methods in C++
- Trap point: OTcl method cmd{}
- Send all arguments after cmd{} call to TclObject::command()

TclObject::command()

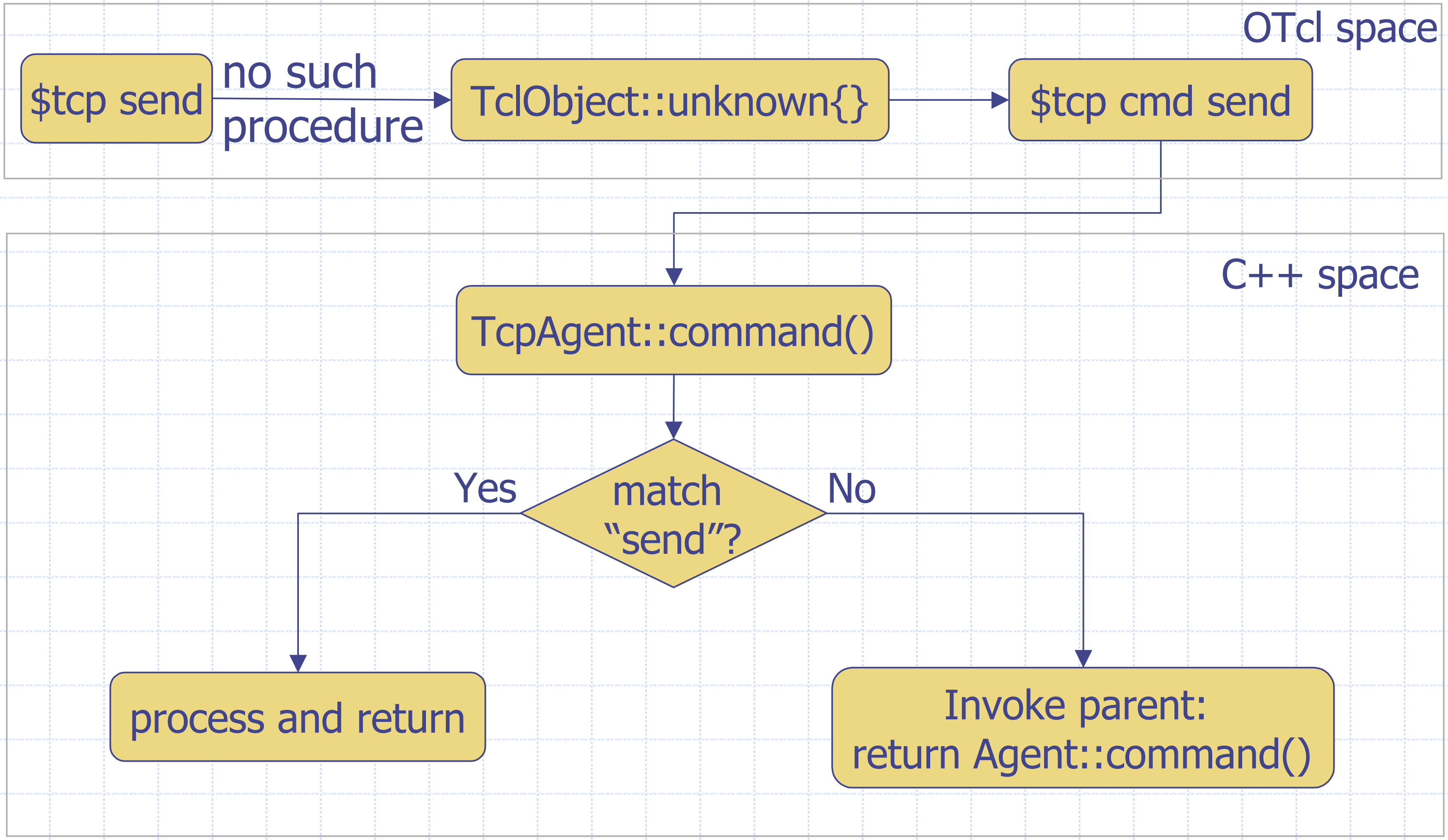
- OTcl

```
set tcp [new Agent/TCP]
$tcp advance 10
```

- C++

```
int TcpAgent::command(int argc,
                      const char*const* argv) {
    if (argc == 3) {
        if (strcmp(argv[1], “advance”) == 0) {
            int newseq = atoi(argv[2]);
            ……
            return(TCL_OK);
        }
    }
    return (Agent::command(argc, argv);
}
```

TclObject::command()

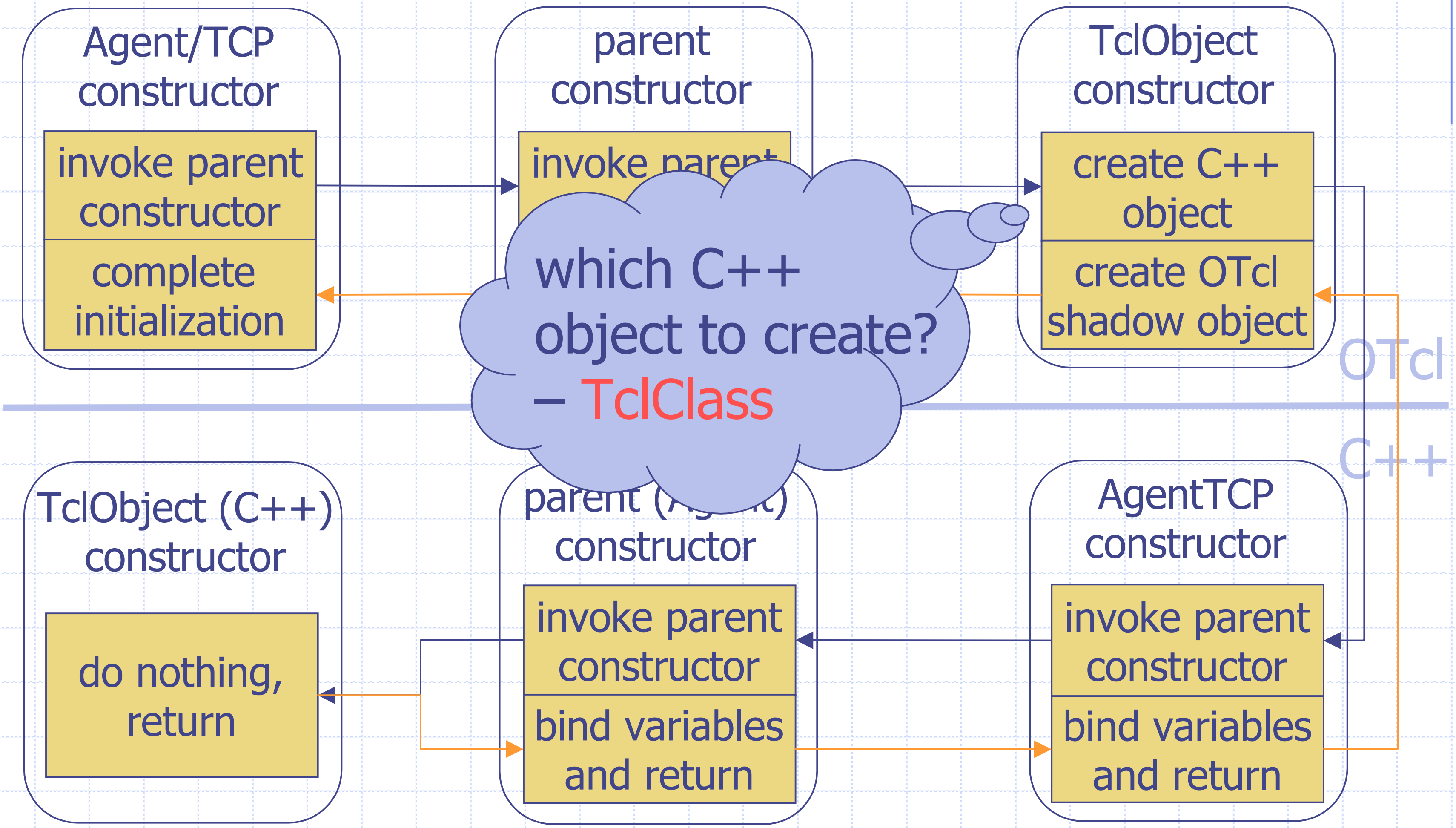


TclObject: Creation and Deletion

- Global procedures: new{}, delete{}
- Example

```
set tcp [new Agent/TCP]
…
delete $tcp
```

TclObject: Creation and Deletion



Agent/TCP constructor

invoke parent constructor

complete initialization

parent constructor

invoke parent

TclObject constructor

create C++ object

create OTcl shadow object

OTcl

C++

which C++ object to create? – TclClass

TclObject (C++) constructor

do nothing, return

parent (Agent) constructor

invoke parent constructor

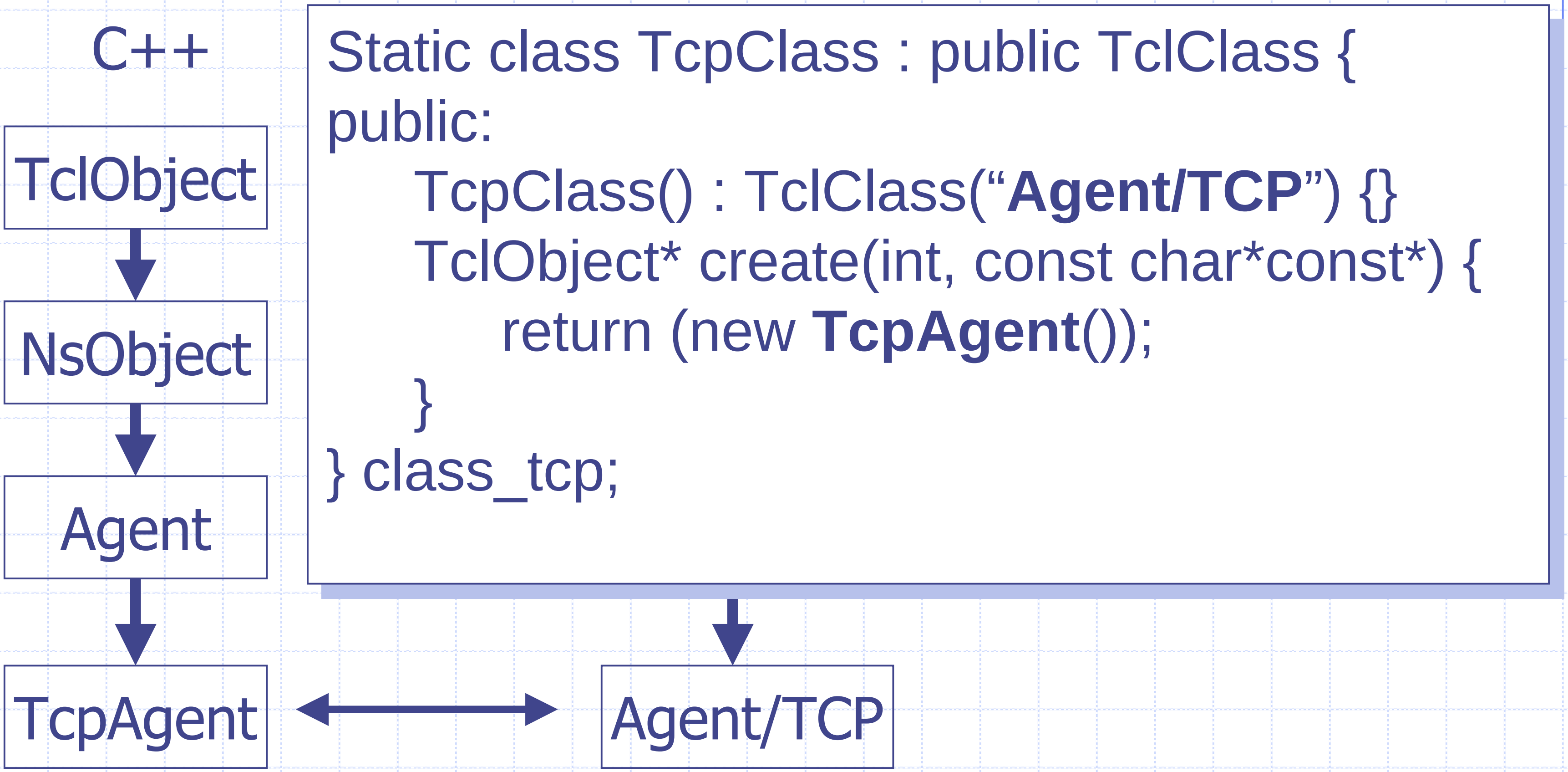
bind variables and return

AgentTCP constructor

invoke parent constructor

bind variables and return

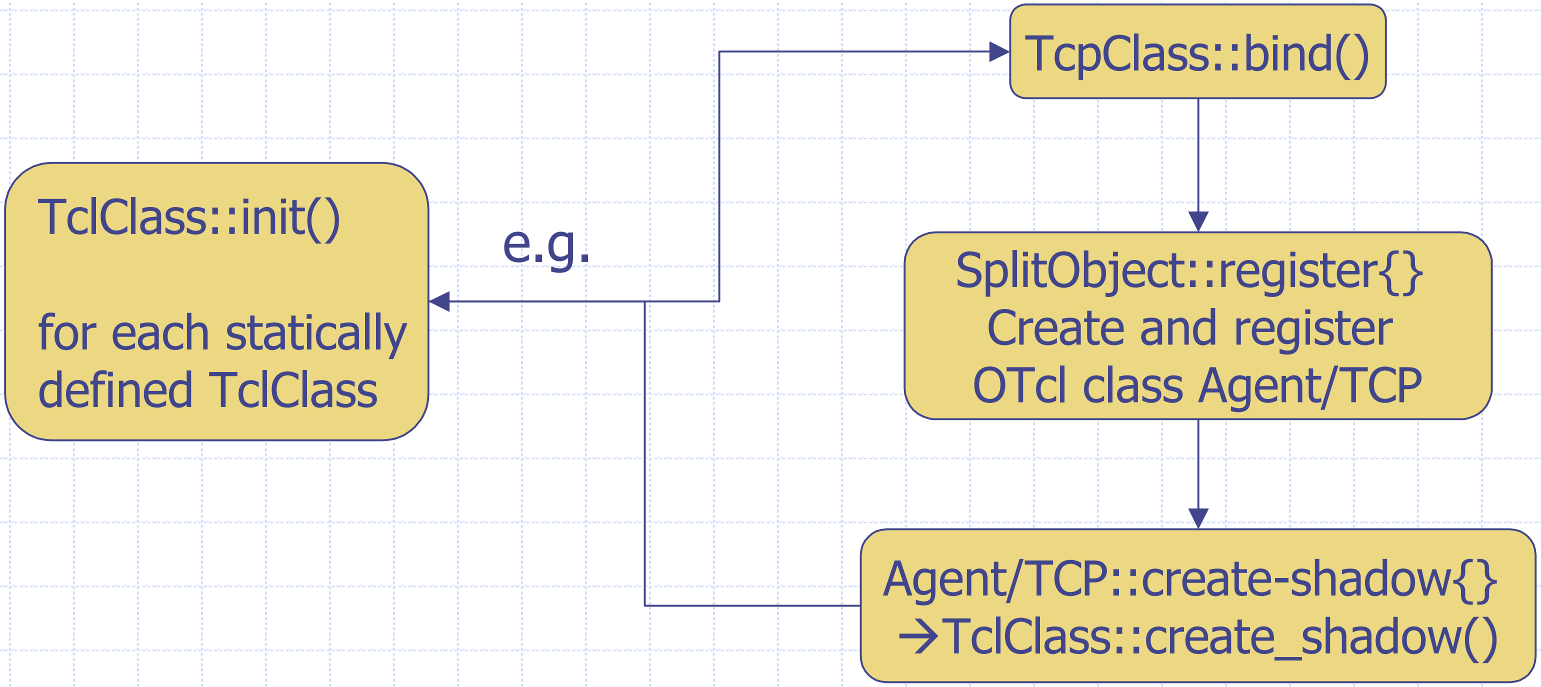
TclClass



```
Static class TcpClass : public TclClass {
public:
      TcpClass() : TclClass(“Agent/TCP”) {}
      TclObject* create(int, const char*const*) {
            return (new TcpAgent());
      }
} class_tcp;
```

TclClass: Mechanism

- Initialization at runtime startup



TclClass::init()

for each statically defined TclClass

e.g.

TcpClass::bind()

SplitObject::register{}
Create and register
OTcl class Agent/TCP

Agent/TCP::create-shadow{}
→TclClass::create_shadow()

Class Tcl

- Singleton class with a handle to Tcl interpreter
- Usage
  - Invoke OTcl procedure
  - Obtain OTcl evaluation results
  - Pass a result string to OTcl
  - Return success/failure code to OTcl

Class Tcl

```
Tcl& tcl = Tcl::instance();
if (argc == 2) {
    if (strcmp(argv[1], “now”) == 0) {
          tcl.resultf(“%g”, clock());
          return TCL_OK;
    }
    tcl.error(“command not found”);
    return TCL_ERROR;
 } else if (argc == 3) {
    tcl.eval(argv[2]);
    clock_ = atof(tcl.result());
    return TCL_OK;
}
```

Class TclCommand

- C++ implementation of global OTcl commands

```
class RandomCommand : public TclCommand {
public:
        RandomCommand() : TclCommand("ns-random") {}
        virtual int command(int argc, const char*const* argv);
};

int RandomCommand::command(int argc, const char*const* argv)
{
        Tcl& tcl = Tcl::instance();
        if (argc == 1) {
                sprintf(tcl.buffer(), "%u", Random::random());
                tcl.result(tcl.buffer());
        }
```

EmbeddedTcl

- Pre-load OTcl scripts at ns runtime startup
  - Recursively load `~ns/tcl/lib/ns-lib.tcl:`

    ```
    source ns-autoconf.tcl
    source ns-address.tcl
    source ns-node.tcl
    .......
    ```

  - Load everything into a single C++ string
  - Execute this string at runtime startup
- Tcl::init(): load `~tclcl/tcl-object.tcl`
- Tcl_AppInit(): load `~ns/tcl/lib/ns-lib.tcl`

EmbeddedTcl

- How it works
  - `tcl2c++`: provided by TclCL, converts tcl scripts into a C++ static character array
  - Makefile.in:

```
tclsh8.0 bin/tcl-expand.tcl tcl/lib/ns-lib.tcl | tcl2c++ et_ns_lib > gen/ns_tcl.cc
```

Summary

- TclObject
  - Unified interpreted (OTcl) and compiled (C++) class hierarchies
  - Seamless access (procedure call and variable access) between OTcl and C++
- TclClass
  - The mechanism that makes TclObject work
- Tcl: primitives to access Tcl interpreter

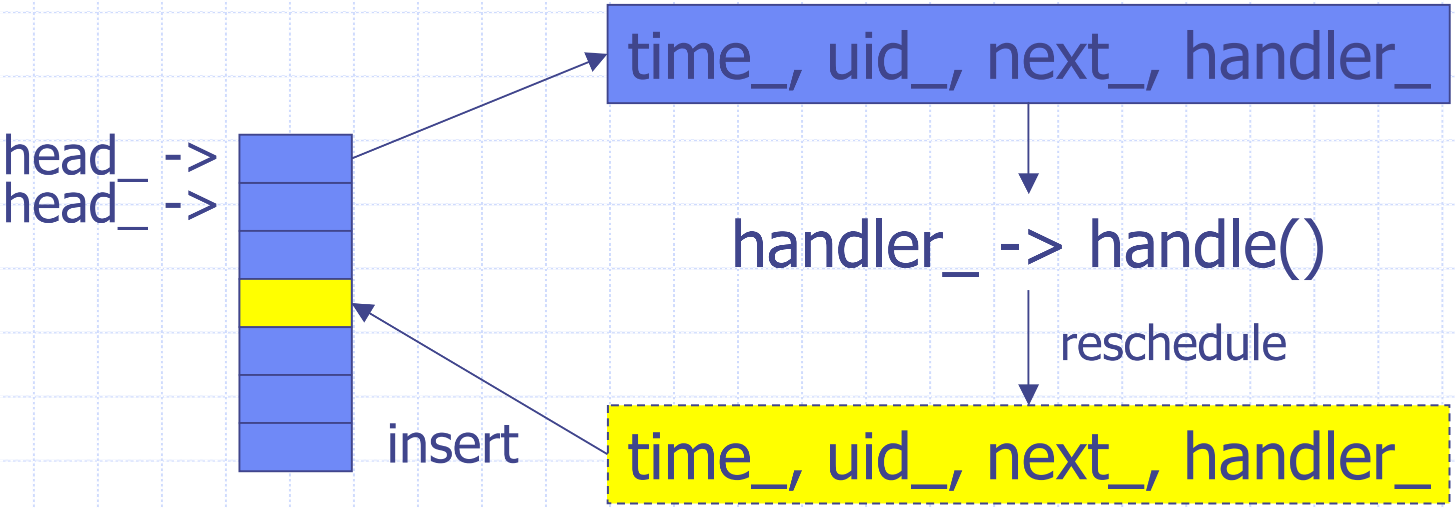
Outline

- Fundamental concept
  - Split object
- Plumbing
  - Wired world
  - Wireless world
- Scaling

ns Internals

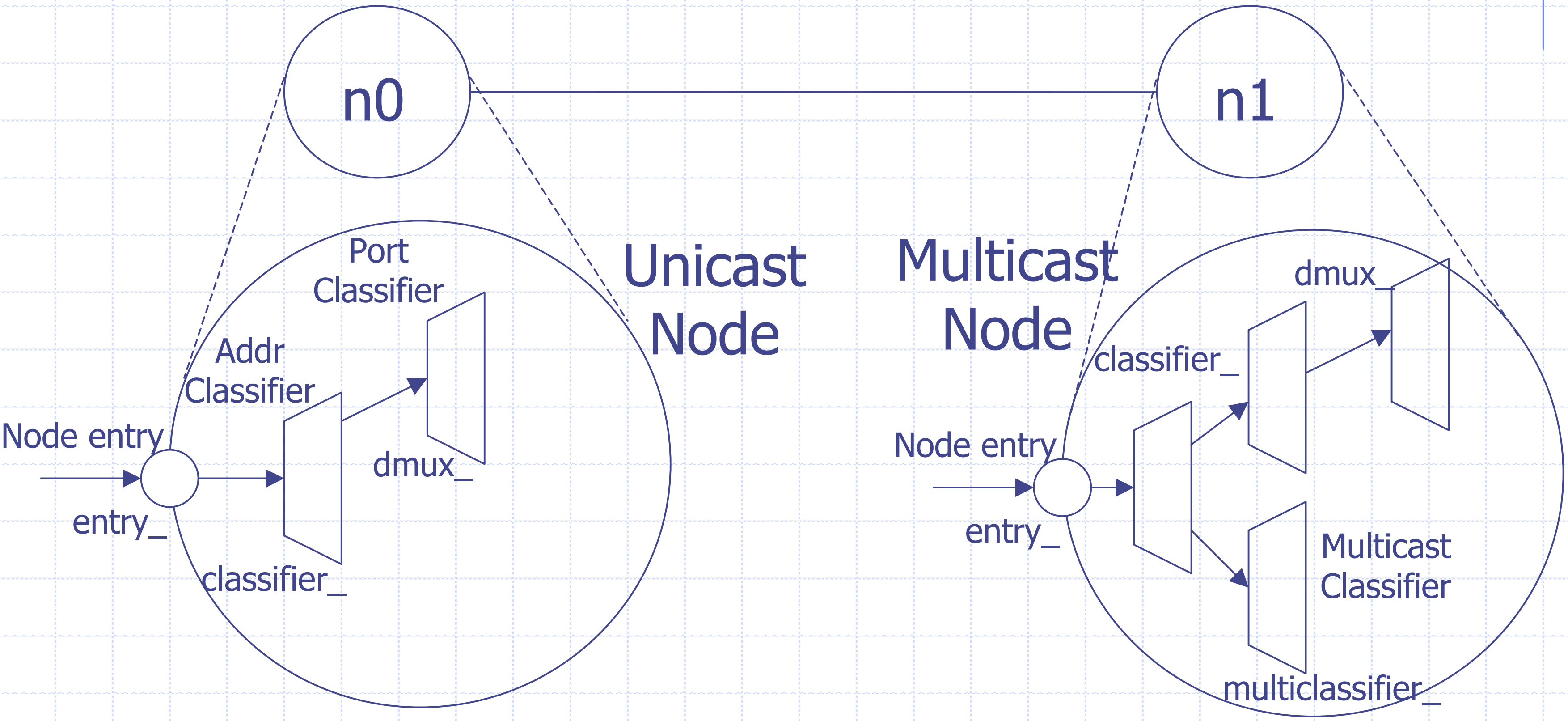
- Discrete event scheduler
- Network topology
- Routing
- Transport
- Packet flow
- Packet format
- Application

Discrete Event Scheduler

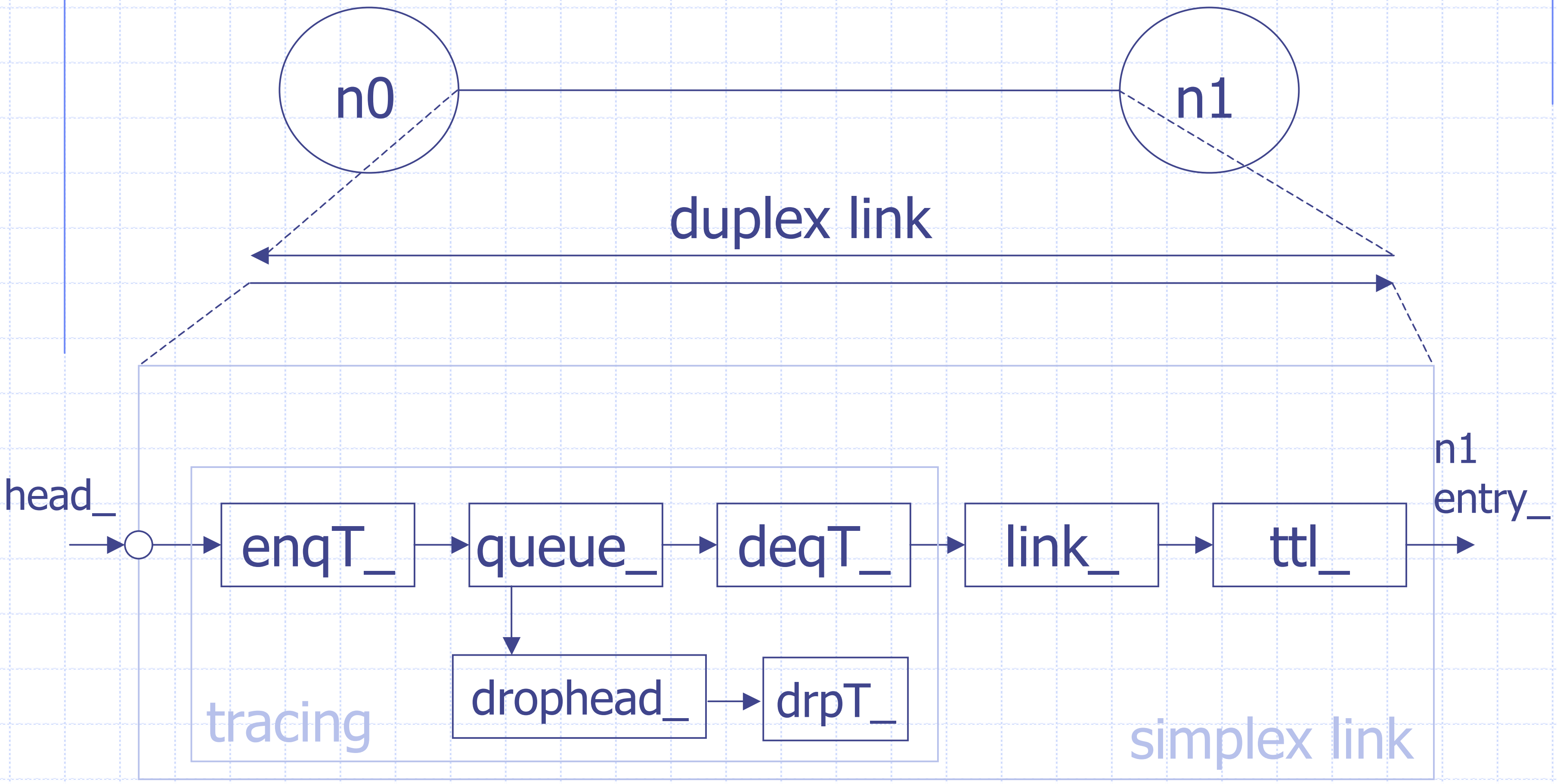


- Three types of schedulers
  - List: simple linked list, order-preserving, O(N)
  - Heap: O(logN)
  - Calendar: hash-based, fastest, O(1)

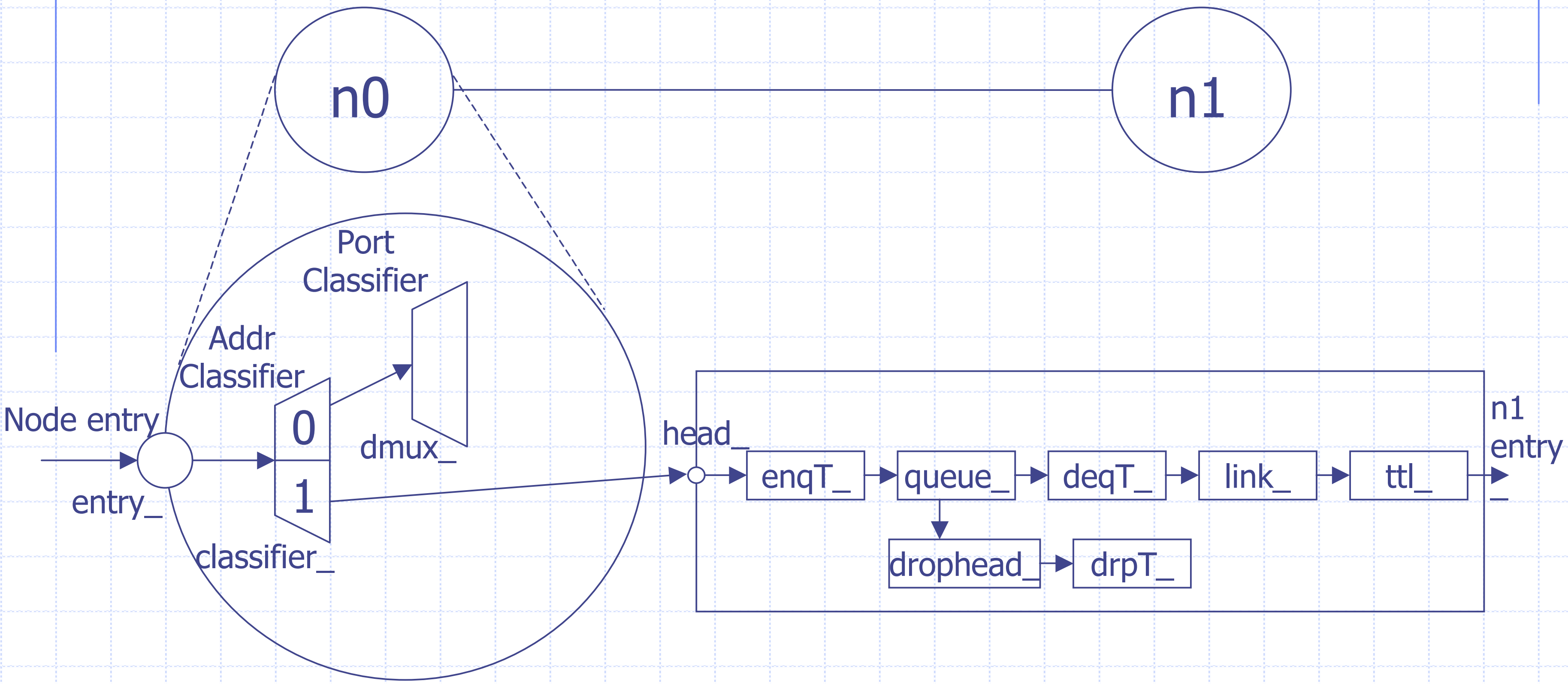
Network Topology: Node



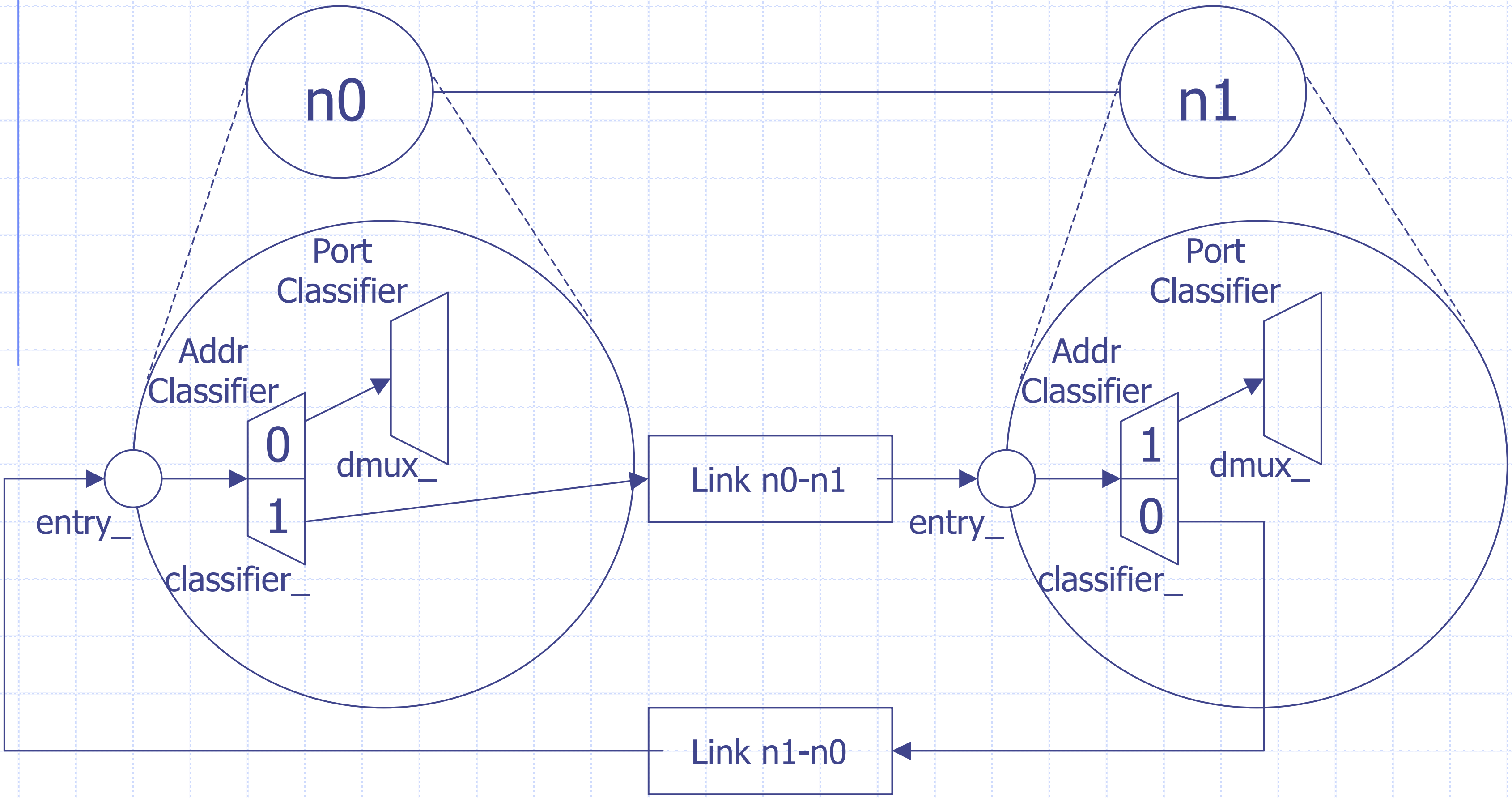
Network Topology: Link



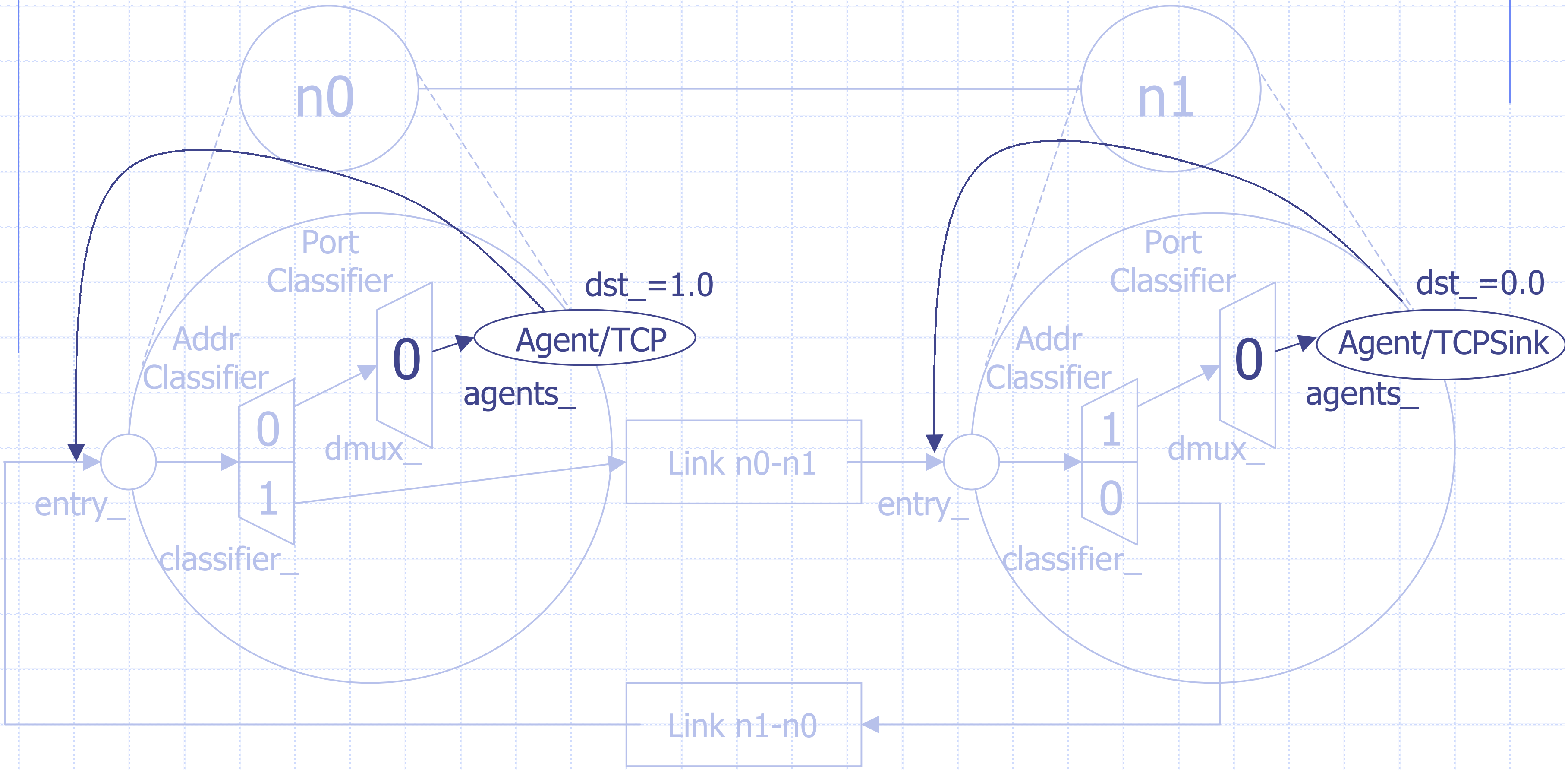
Routing


n0
n1
Port
Classifier
Addr
Classifier
Node entry
entry_
0
1
dmux_
classifier_
head_
enqT_
queue_
deQT_
link_
ttl_
drophead_
drpT_
n1
entry
_

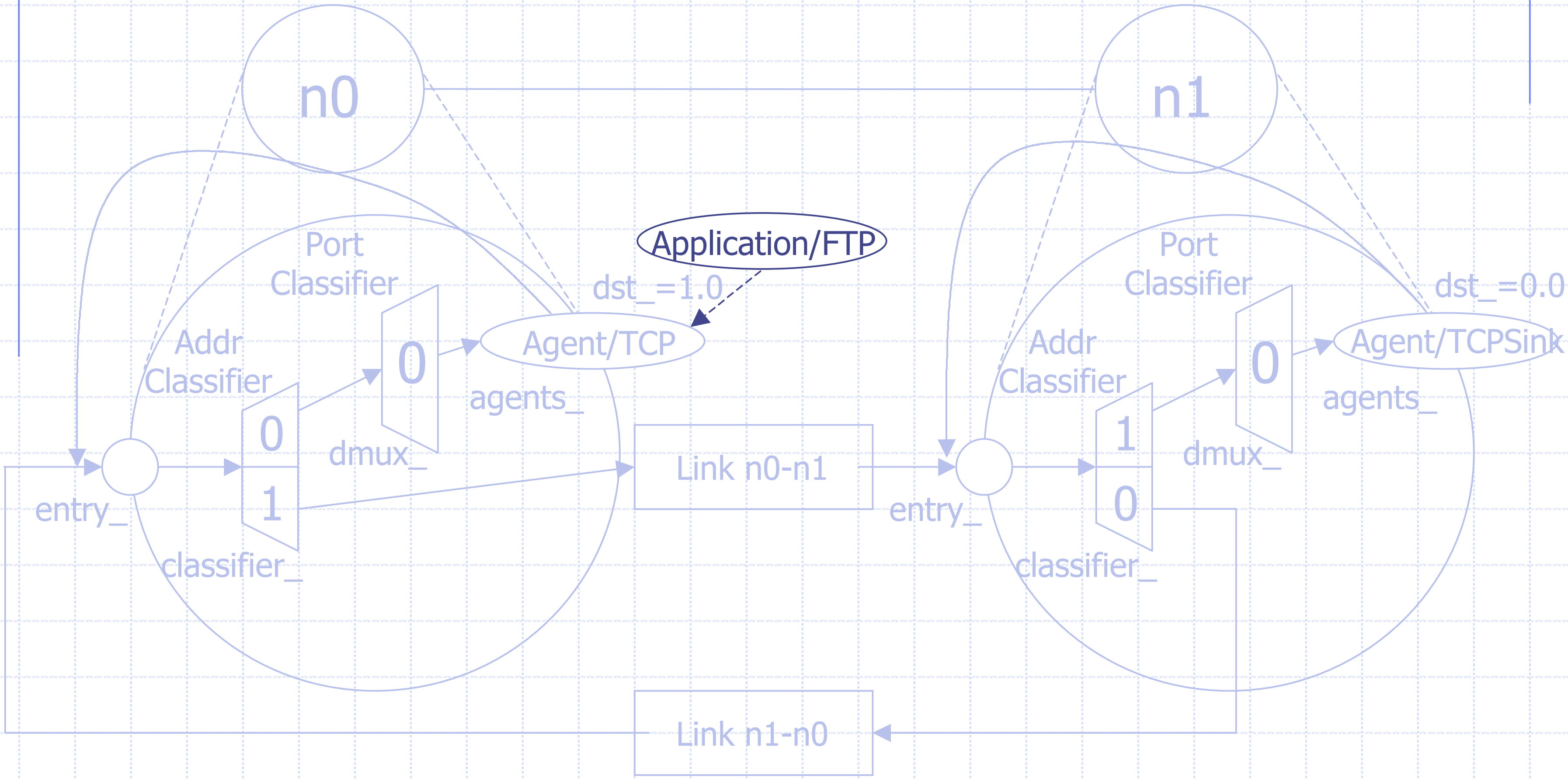
Routing (con't)



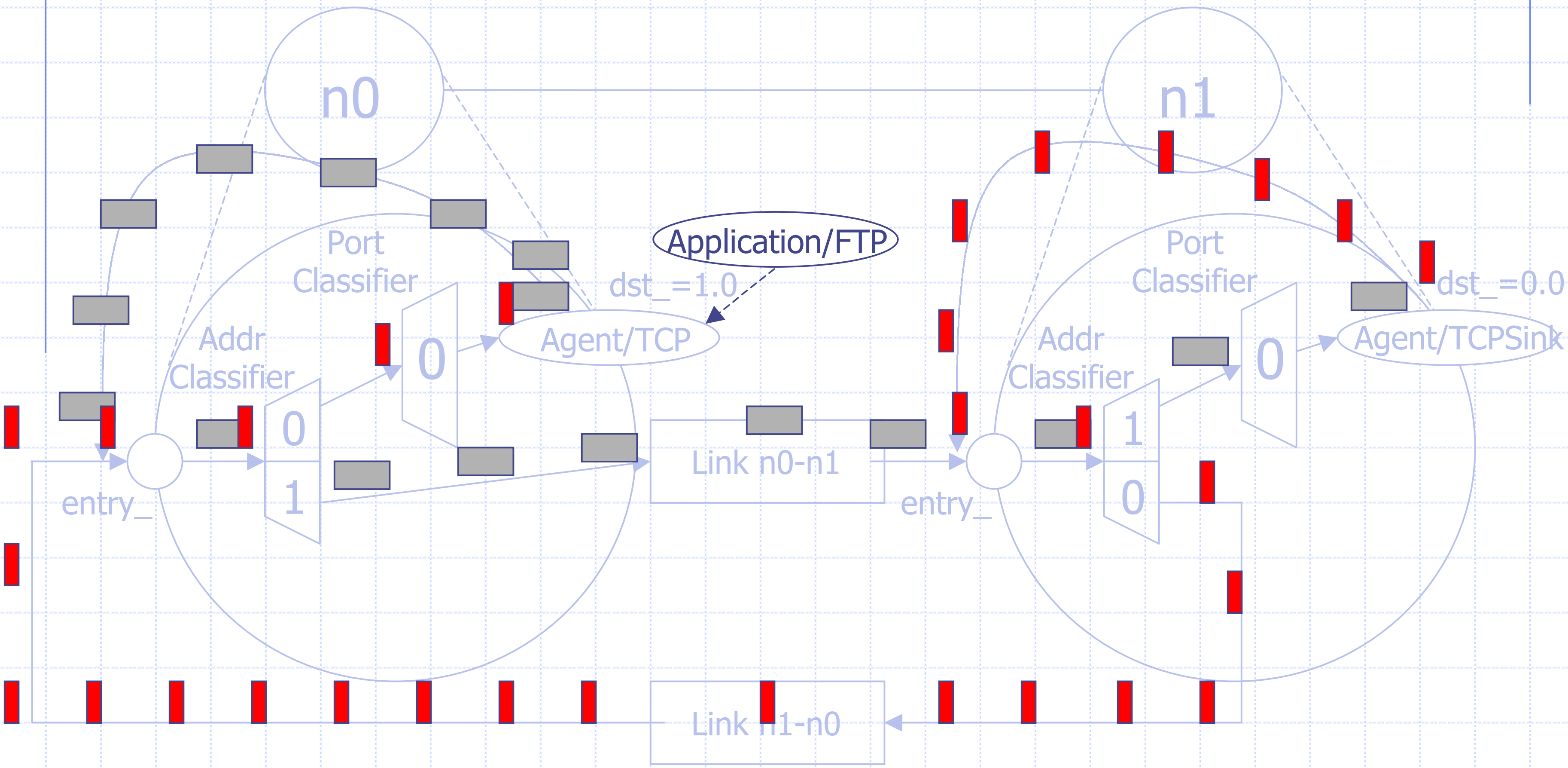
Transport


n0
n1
Port
Classifier
Addr
Classifier
dst_=1.0
Agent/TCP
0
agents_
0
1
dmux_
entry_
classifier_
Link n0-n1
Port
Classifier
Addr
Classifier
dst_=0.0
Agent/TCPSink
0
agents_
1
0
dmux_
entry_
classifier_
Link n1-n0

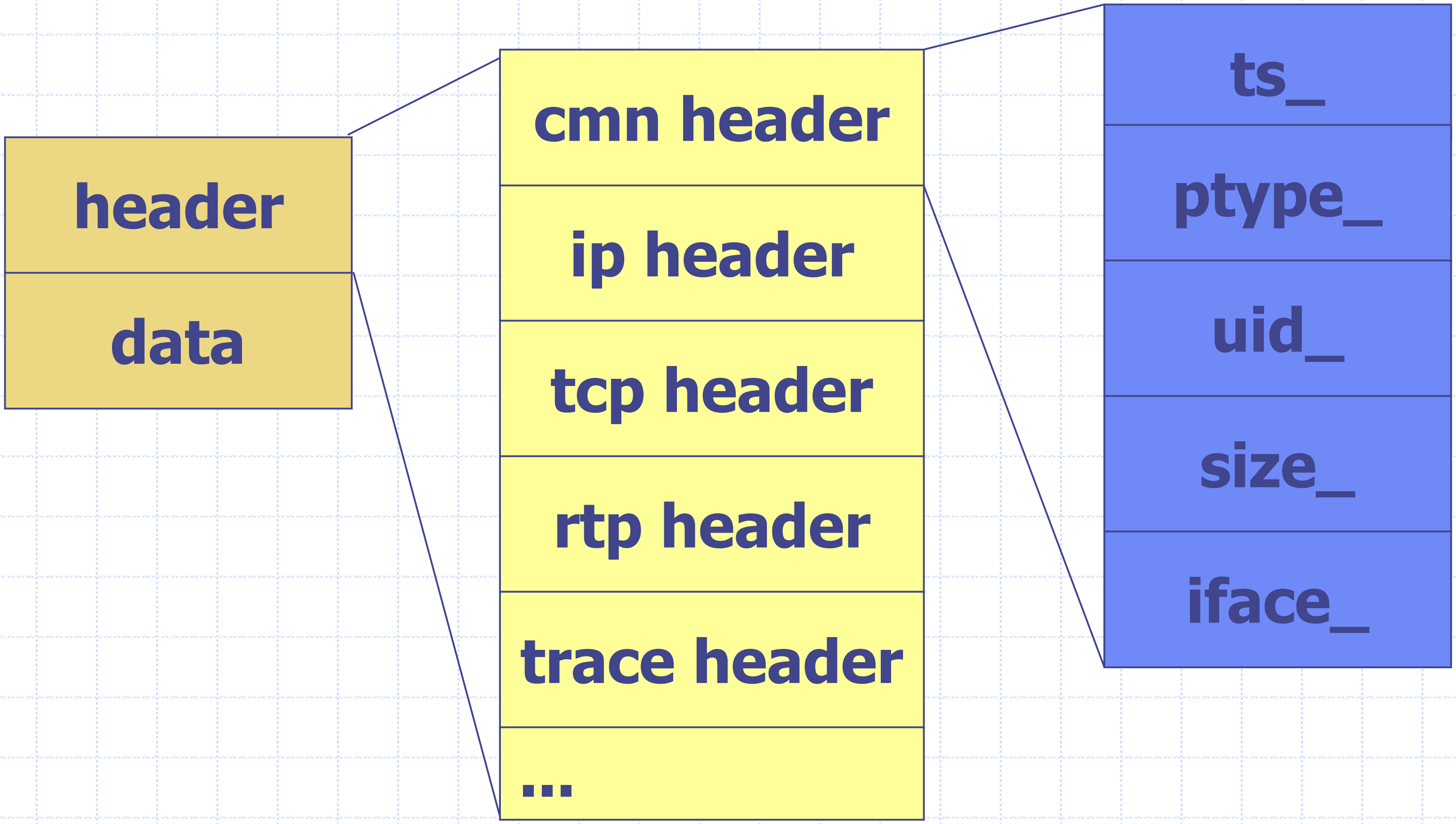
Application: Traffic Generator



Plumbing: Packet Flow



Packet Format



header

data

cmn header

ip header

tcp header

rtp header

trace header

...

ts_

ptype_

uid_

size_

iface_

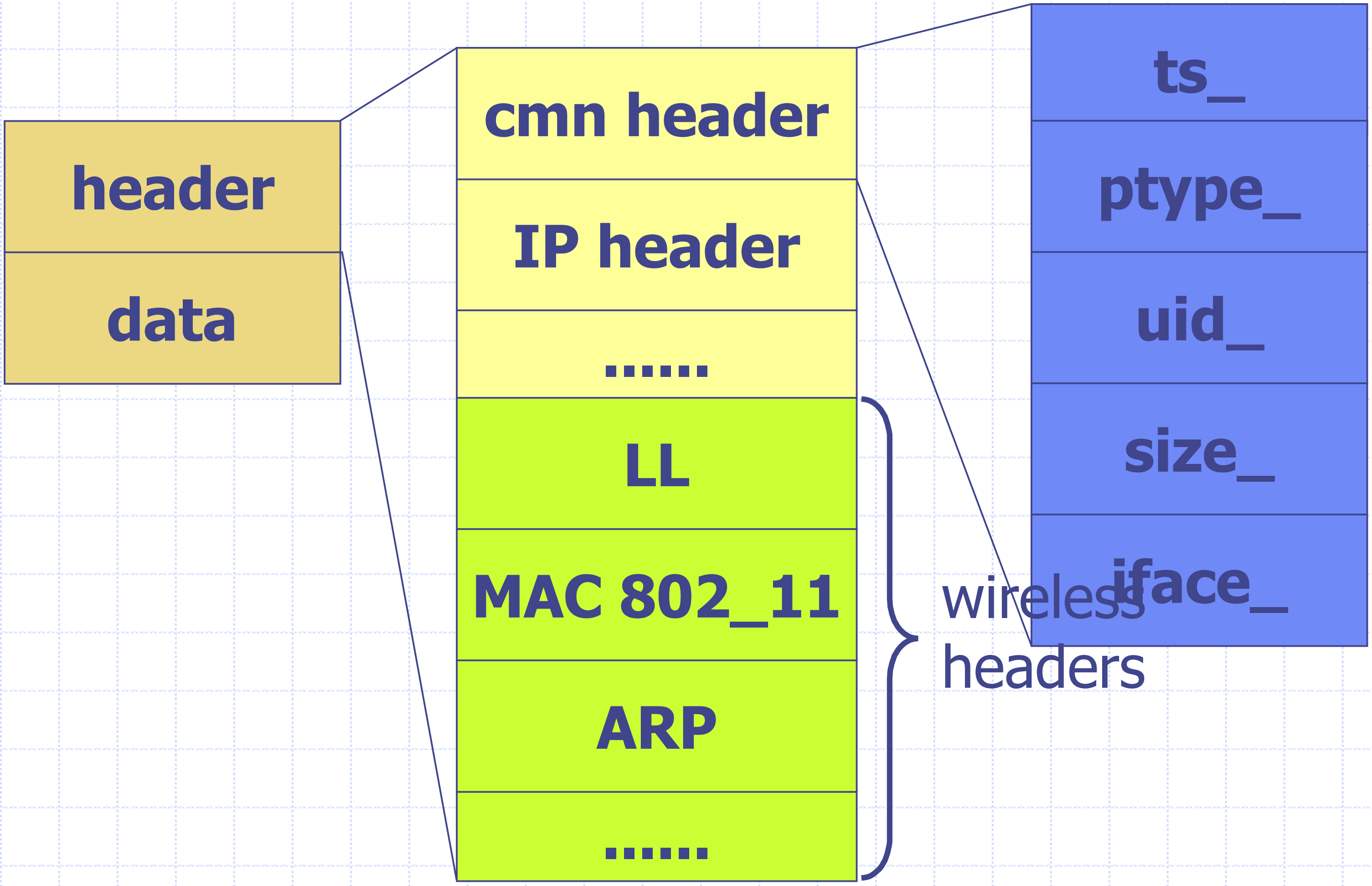
Outline

- Fundamental concept
  - Split object
- Plumbing
  - Wired world
  - Wireless world
- ns scaling

Abstract the Real World

- Packet headers
- Mobile node
- Wireless channel
- Forwarding and routing
- Visualization

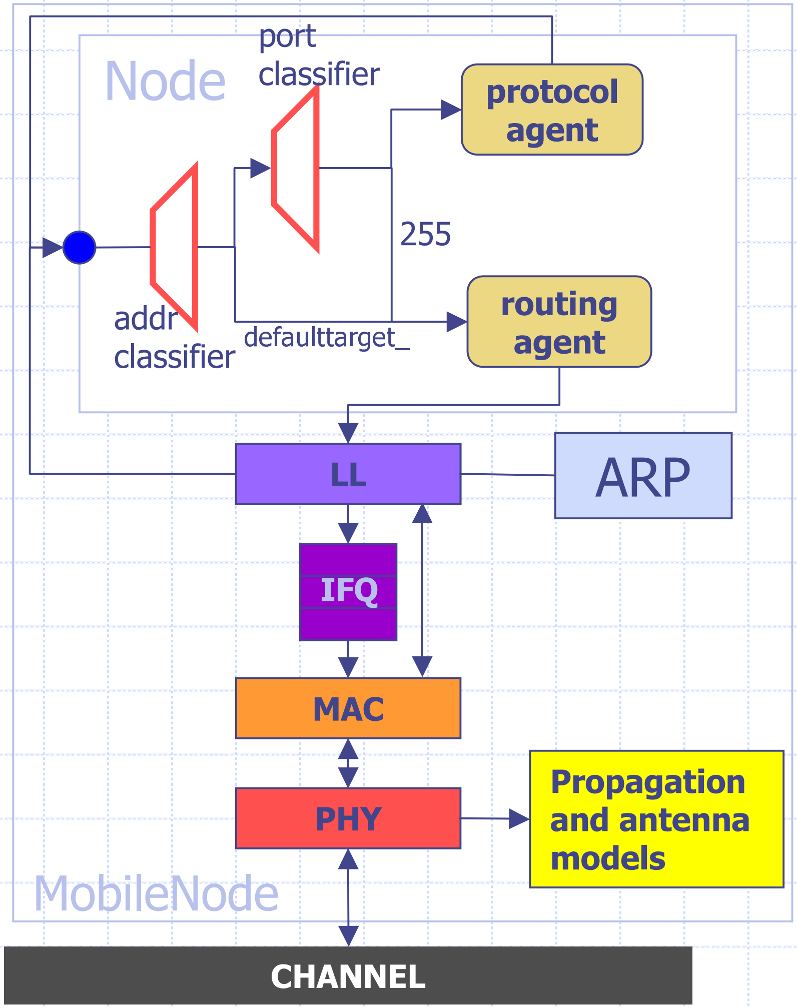
Wireless Packet Format



Mobile Node Abstraction

- Location
  - Coordinates (x,y,z)
- Movement
  - Speed, direction, starting/ending location, time ...

Portrait of A Mobile Node



- ⏢ **Classifier**: Forwarding
- ⬭ **Agent**: Protocol Entity
- ● **Node Entry**
- LL **LL**: Link layer object
- IFQ **IFQ**: Interface queue
- MAC **MAC**: Mac object
- PHY **PHY**: Net interface

Mobile Node: Components

- Link Layer
  - Same as LAN, but with a separate ARP module
- Interface queue
  - Give priority to routing protocol packets
- Mac Layer
  - IEEE 802.11
  - RTS/CTS/DATA/ACK for all unicast packets
  - DATA for all broadcast packets

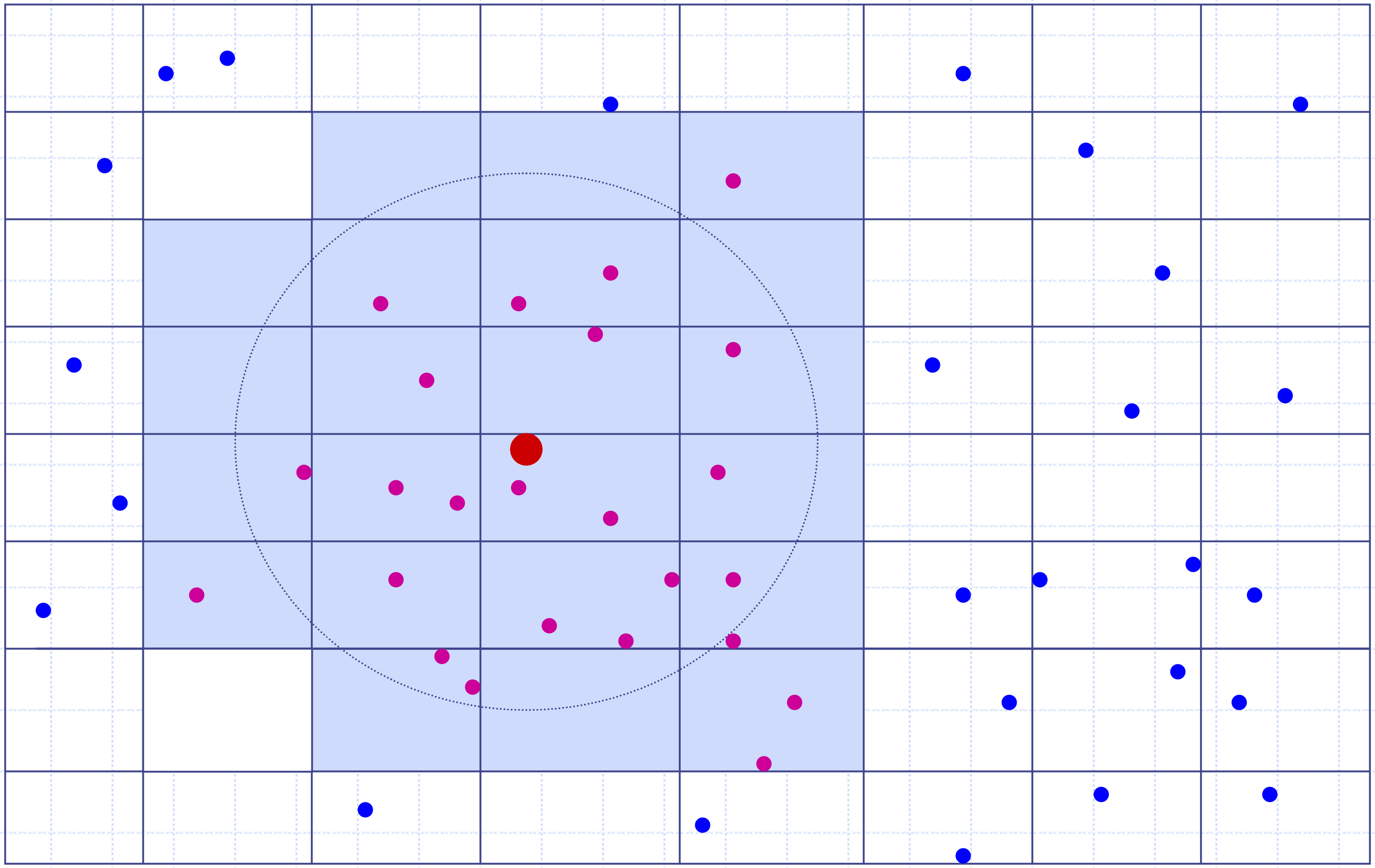
Mobile Node: Components

- Network interface (PHY)
  - Parameters based on Direct Sequence Spread Spectrum (WaveLan)
  - Interface with: antenna and propagation models
  - Update energy: transmission and reception
- Radio Propagation Model
  - Friss-space attenuation($1/r^2$) at near distance
  - Two-ray Ground ($1/r^4$) at far distance
- Antenna
  - Omni-directional, unity-gain

Wireless Channel

- Duplicate packets to all mobile nodes attached to the channel except the sender
- It is the receiver’s responsibility to decide if it will accept the packet
    - Collision is handled at individual receiver
    - O(N^2) messages → grid keeper

Grid-keeper: An Optimization



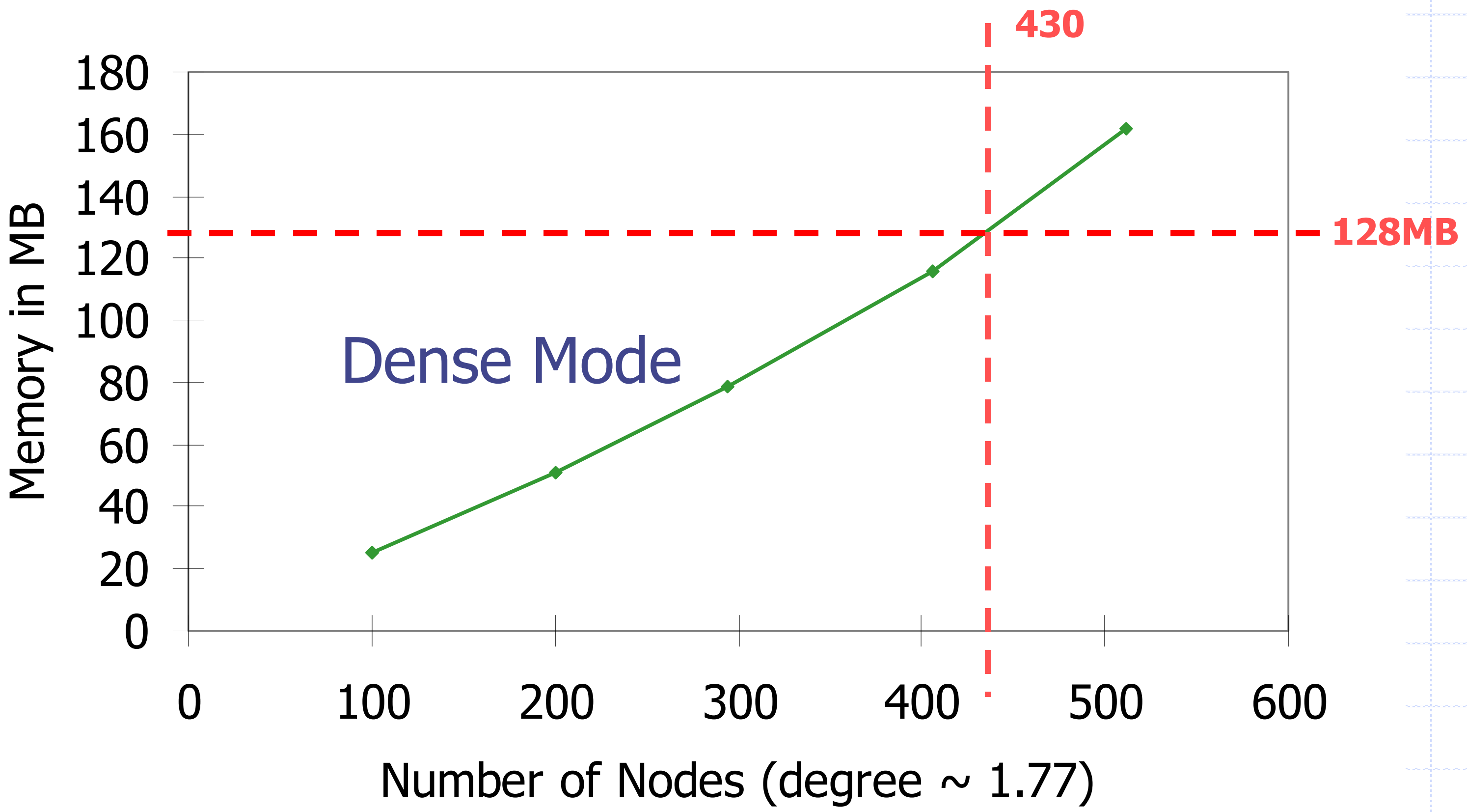
Outline

- Fundamental concept
  - Split object
- Plumbing
  - Wired world
  - Wireless world
- ns scaling

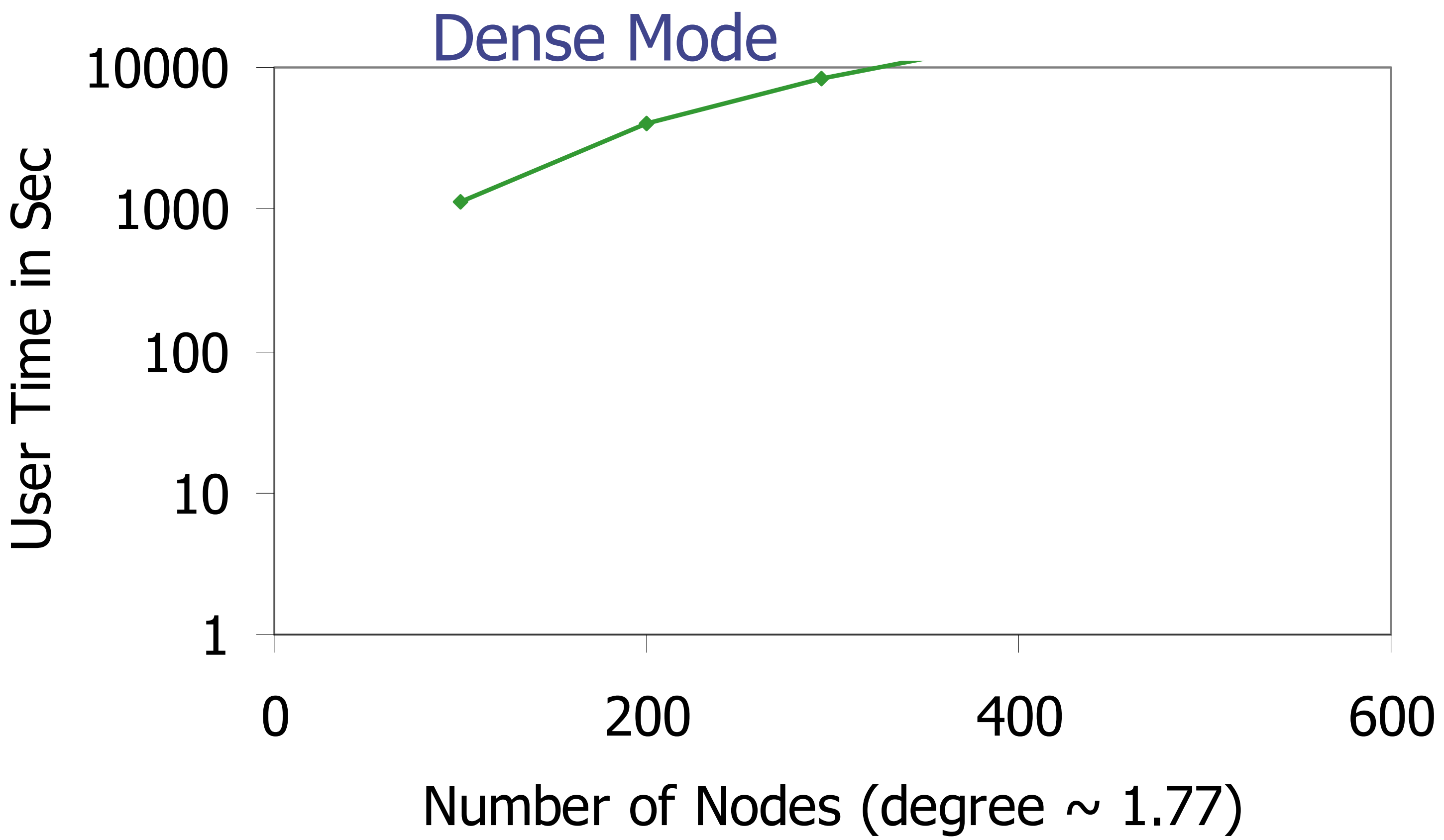
ns Scaling

- Limitations to simulation size
  - Memory
  - Run time
- Solutions
  - Abstraction
  - Fine-tuning (next session)

Memory Footprint of ns



Run Time of ns

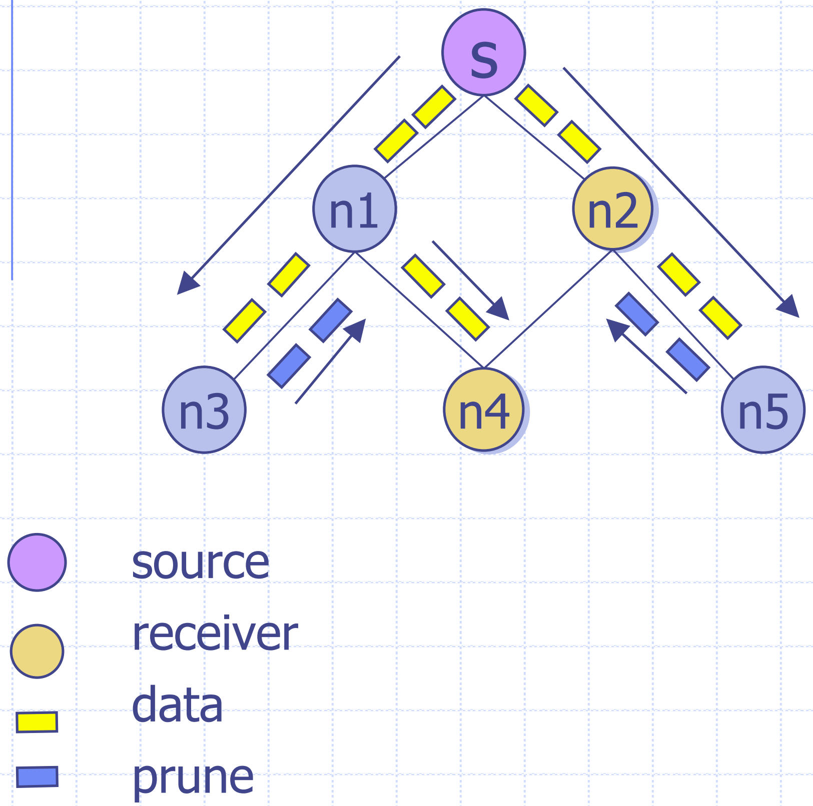


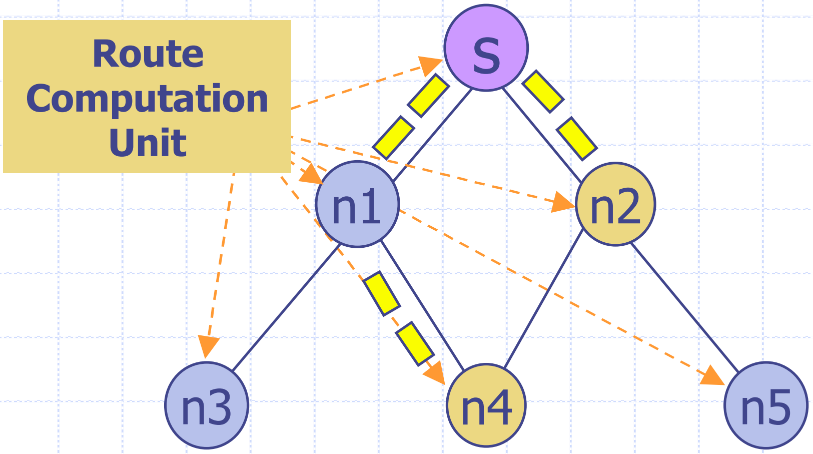
Culprit: Details, Details

- Dense mode tries to capture packet-level behavior
  - Prunes, Joins, …
- Let’s abstract it out
  - Centralized multicast
  - SessionSim

Centralized Multicast

Dense Mode Multicast





Centralized Multicast

Centralized Multicast

- Usage

```
$ns mrtproto CtrMcast
```

- Limitation
  - No exact dynamic behavior, e.g., routing convergence time
  - Does not mean to replace DM

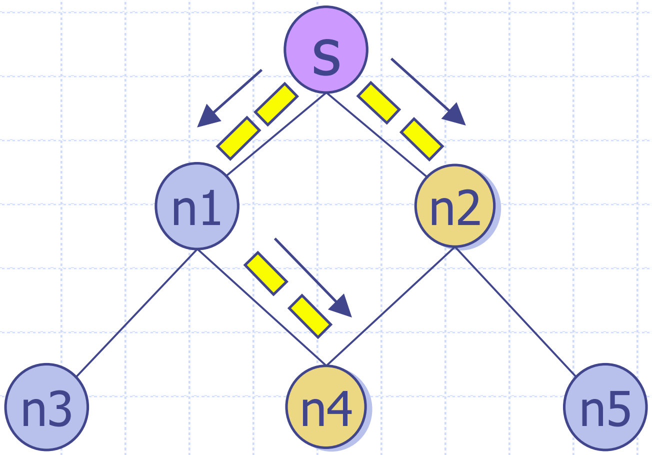
Further Abstraction

ns Object Sizes	
Multicast Node	6KB
Duplex link	14KB

- Remove all intermediate nodes and links
- Do not model:
  - Detailed queueing
  - Detailed packet delivery

SessionSim

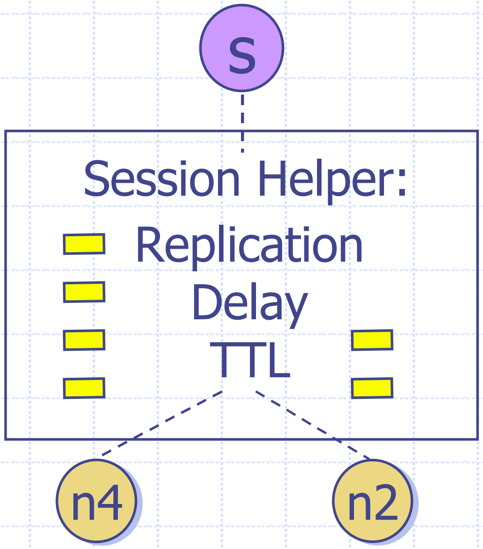
Detailed Packet Distribution



● source

● receiver

▭ data



Session Helper:

Replication

Delay

TTL

Session Multicast

SessionSim

- Usage

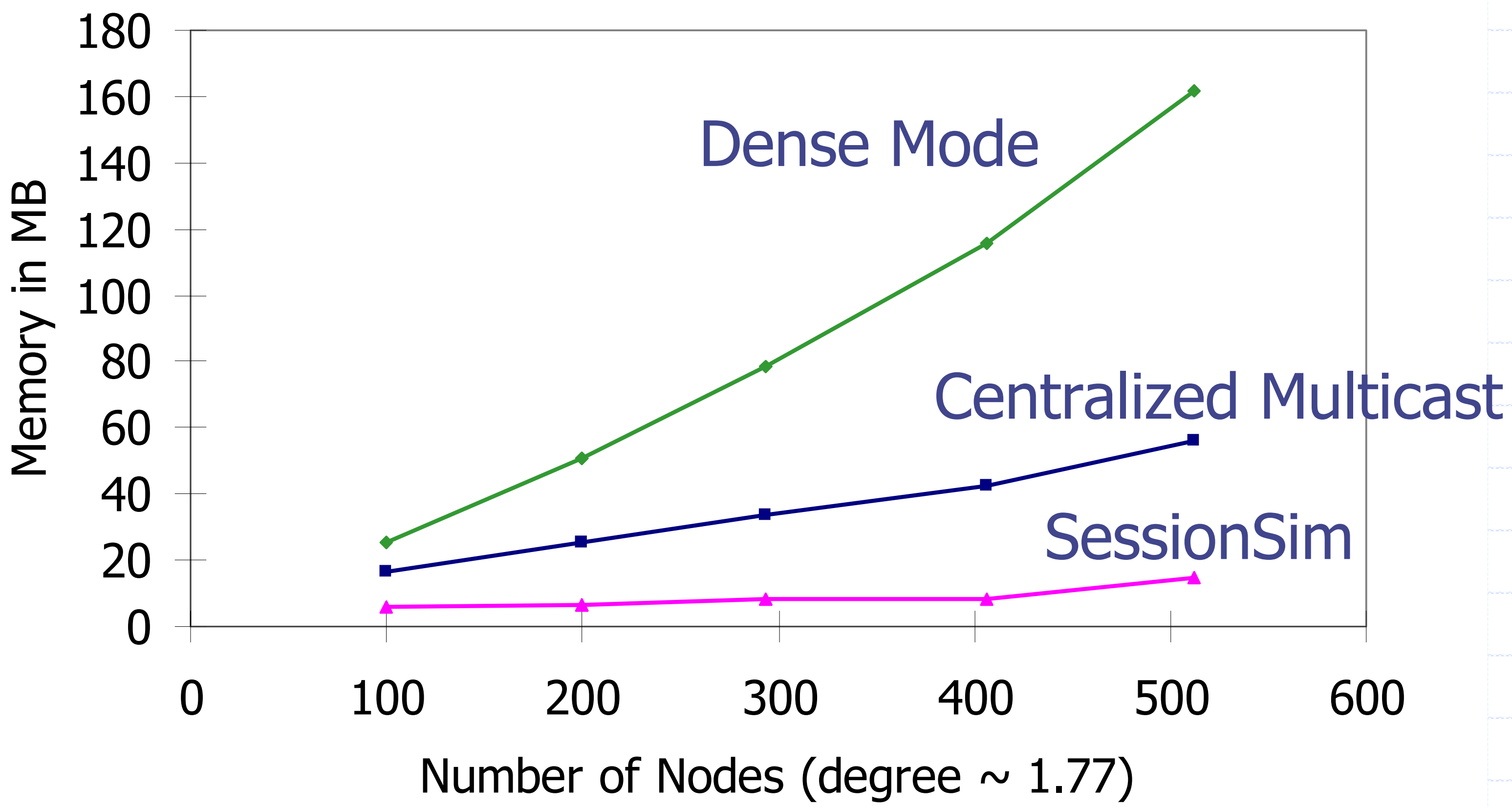
```
set ns [new SessionSim]
```

instead of

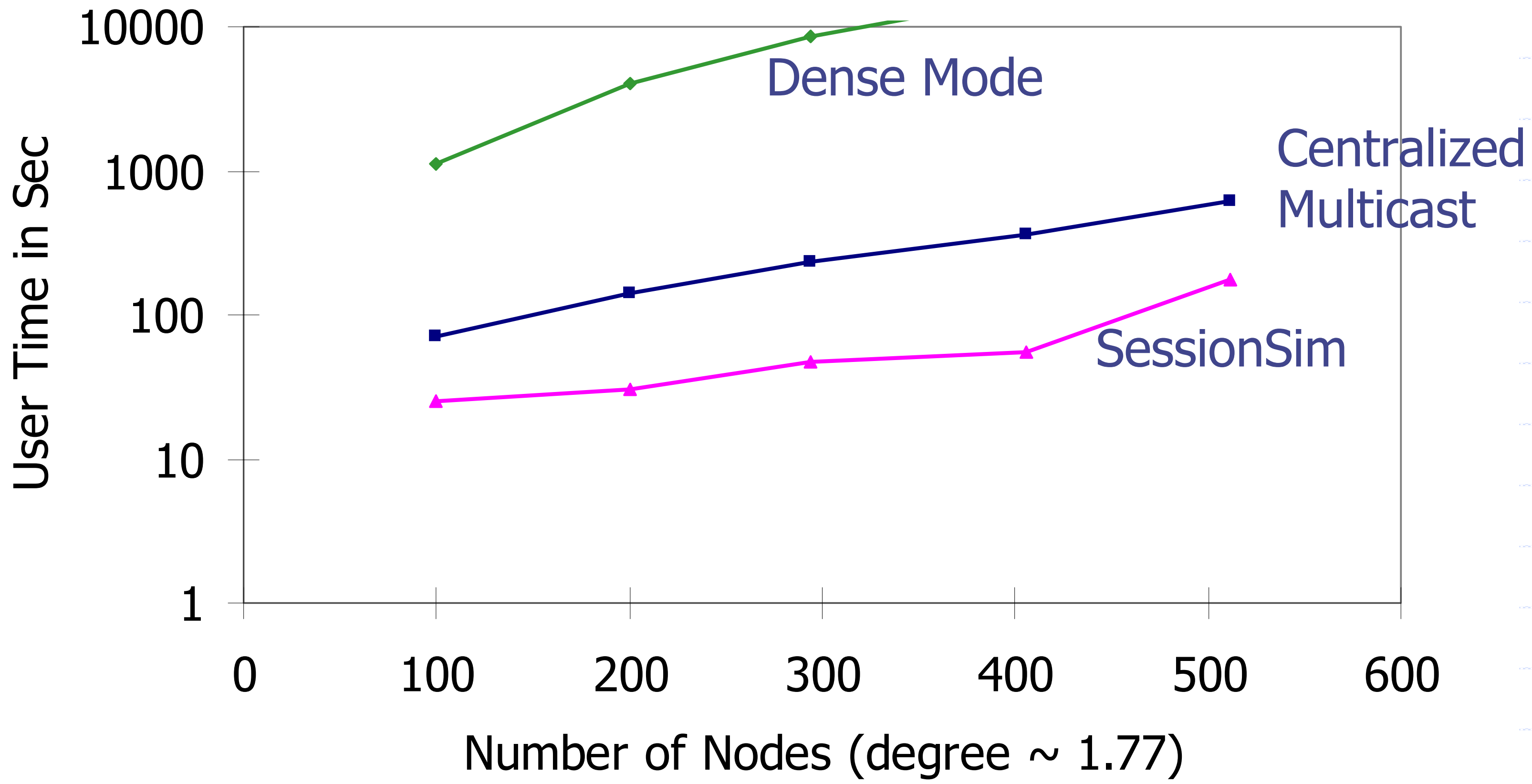
```
set ns [new Simulator]
```

- Limitation
  - Distorted end-to-end delay
  - Packet loss due to congestion

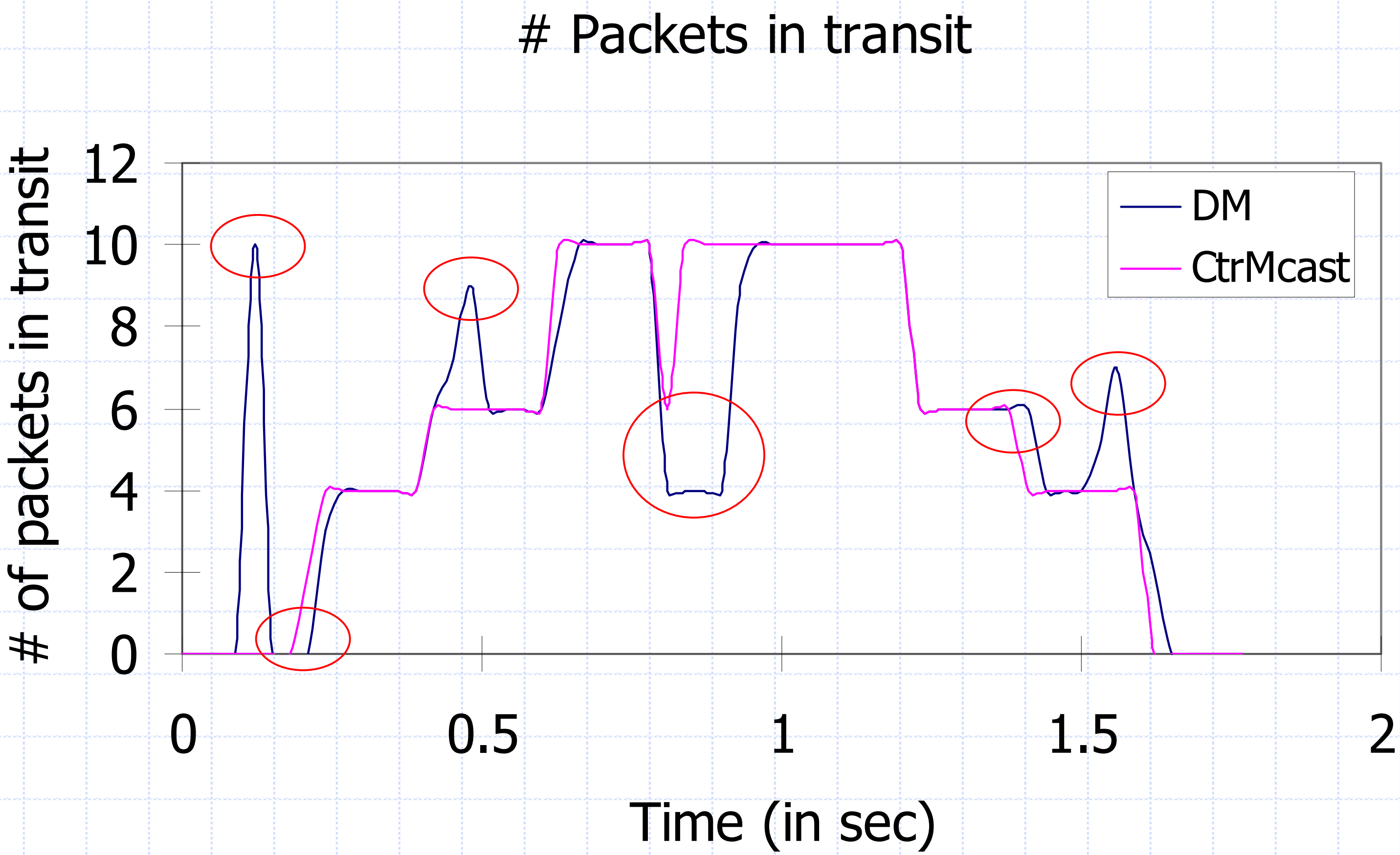
Memory Footprint of ns



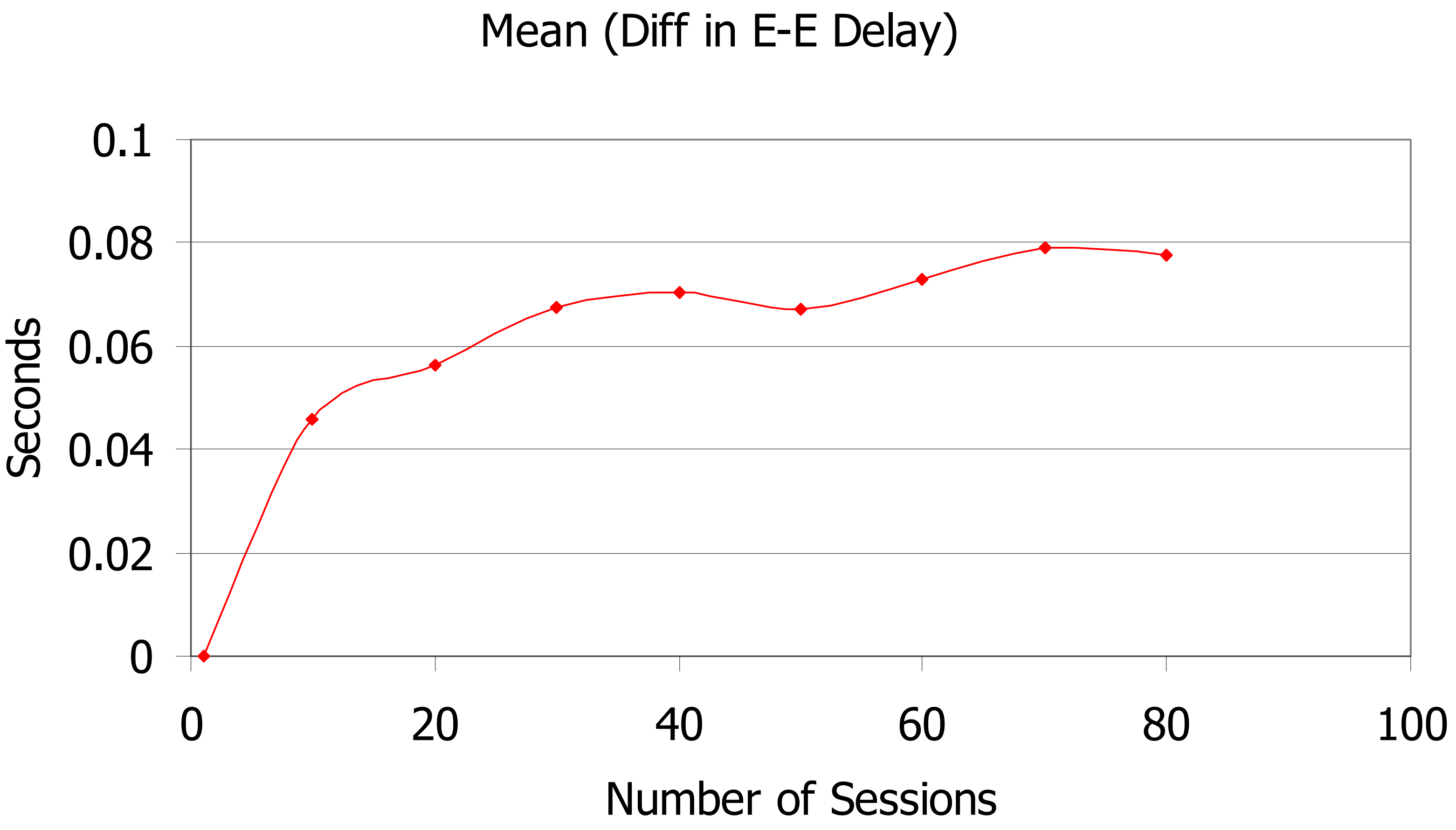
Run Time of ns



Distortion of Centralized Multicast



Distortion of SessionSim



Footnotes

- My sim still uses too much memory?

  Or

- I want large detailed simulation, e.g., Web traffic pattern?

- Fine-tune your simulator
  - We'll cover it this afternoon