Tierless Programming and Reasoning for Software-Defined Networks

Tim Nelson, Andrew D. Ferguson,
Michael J. G. Scheer, Shriram Krishnamurthi
(Brown University)

Control plane program
(Java, C++, Python, Haskell, ...)

Data plane (on switches)

"match-action" OpenFlow Rules

Controller Program State
(Onix, Cassandra, Zookeeper, …)

Control plane program
(Java, C++, Python, Haskell, ...)

Data plane (on switches)

"match-action" OpenFlow Rules

Controller Program State
(Onix, Cassandra, Zookeeper, …)

Datastore

Control plane program
(Java, C++, Python, Haskell, …)

Control logic

Data plane (on switches)

"match-action" OpenFlow Rules

Forwarding Rules

Database Tier

Datastore

Server / Business logic Tier

Control logic

Client / Presentation Tier

Forwarding Rules

Database Tier

Datastore

Server / Business logic Tier

Control logic

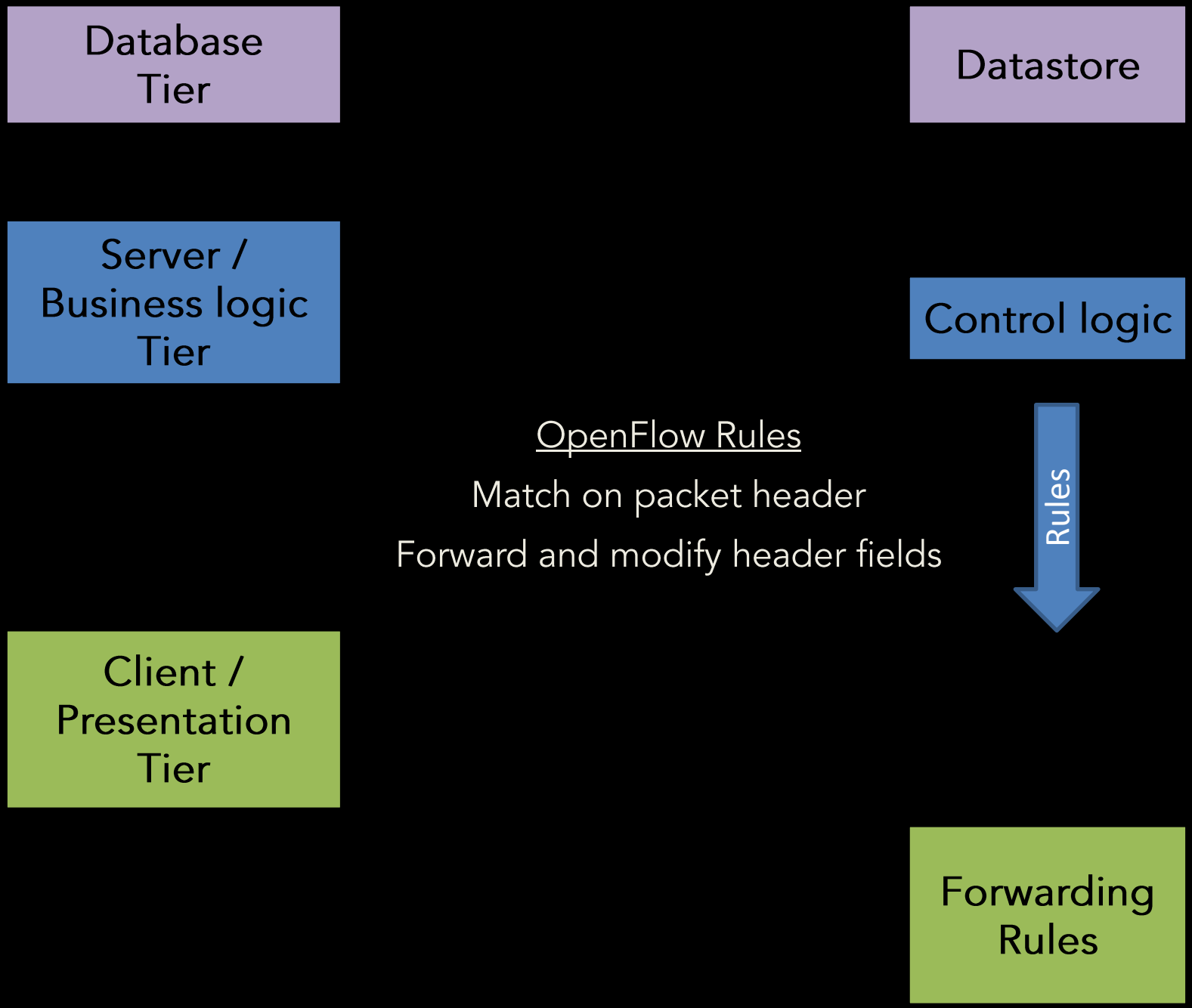
OpenFlow Rules

Match on packet header

Forward and modify header fields

Client / Presentation Tier

Forwarding Rules


Database Tier
Datastore
Server / Business logic Tier
Control logic
OpenFlow Rules
Match on packet header
Forward and modify header fields
Rules
Client / Presentation Tier
Forwarding Rules

NAT Library (POX) =

```
from pox.core import core
import pox
log = core.getLogger()

from pox.lib.packet.ipv4 import ipv4
from pox.lib.packet.arp import arp
import pox.lib.packet as pkt

from pox.lib.addresses import IPAddr
from pox.lib.addresses import EthAddr
from pox.lib.util import str_to_bool, dpid_to_str,
str_to_dpid
from pox.lib.revent import EventMixin, Event
from pox.lib.recoco import Timer
import pox.lib.recoco as recoco

import pox.openflow.libopenflow_01 as of
from pox.proto.dhcpd import DHCPD, SimpleAddressPool

import time
import random

FLOW_TIMEOUT = 60
FLOW_MEMORY_TIMEOUT = 60 * 10


class Record (object):
  def __init__ (self):
    self.touch()
    self.outgoing_match = None
    self.incoming_match = None
    self.real_srcport = None
    self.fake_srcport = None
    self.outgoing_fm = None
    self.incoming_fm = None

  @property
  def expired (self):
    return time.time() > self._expires_at

  def touch (self):
    self._expires_at = time.time() + FLOW_MEMORY_TIMEOUT

  def __str__ (self):
    s = "%s:%s" % (self.outgoing_match.nw_src,
self.real_srcport)
    if self.fake_srcport != self.real_srcport:
      s += "/%s" % (self.fake_srcport,)
    s += " -> %s:%s" % (self.outgoing_match.nw_dst,
self.outgoing_match.tp_dst)
    return s


class NAT (object):
  def __init__ (self, inside_ip, outside_ip, gateway_ip,
dns_ip, outside_port,
      dpid, subnet = None):

    self.inside_ip = inside_ip
    self.outside_ip = outside_ip
    self.gateway_ip = gateway_ip
    self.dns_ip = dns_ip # Or None
    self.outside_port = outside_port
    self.dpid = dpid
    self.subnet = subnet

    self._outside_portno = None
    self._gateway_eth = None
    self._connection = None

    # Which NAT ports have we used?
    # proto means TCP or UDP
    self._used_ports = set() # (proto,port)

    # Flow records indexed in both directions
    # match -> Record
    self._record_by_outgoing = {}
    self._record_by_incoming = {}

    core.listen_to_dependencies(self)

  def _all_dependencies_met (self):
    log.debug('Trying to start...')
    if self.dpid in core.openflow.connections:
      self._start(core.openflow.connections[self.dpid])
    else:
      core.openflow.addListenerByName('ConnectionUp',
          self.__handle_dpid_ConnectionUp)

    self.expire_timer = Timer(60, self._expire, recurring
= True)

  def _expire (self):
    dead = []
    for r in self._record_by_outgoing.itervalues():
      if r.expired:
        dead.append(r)

    for r in dead:
      del self._record_by_outgoing[r.outgoing_match]
      del self._record_by_incoming[r.incoming_match]

self._used_ports.remove((r.outgoing_match.nw_proto,r.fake
_srcport))

    if dead and not self._record_by_outgoing:
      log.debug("All flows expired")

  def _is_local (self, ip):
    if ip.is_multicast: return True
    if self.subnet is not None:
      if ip.in_network(self.subnet): return True
      return False
    if ip.in_network('192.168.0.0/16'): return True
    if ip.in_network('10.0.0.0/8'): return True
    if ip.in_network('172.16.0.0/12'): return True
    return False

  def _pick_port (self, flow):
    """
    Gets a possibly-remapped outside port

    flow is the match of the connection
    returns port (maybe from flow, maybe not)
    """

    port = flow.tp_src

    if port < 1024:
      # Never allow these
      port = random.randint(49152, 65534)

    # Pretty sloppy!

    cycle = 0
    while cycle < 2:
      if (flow.nw_proto,port) not in self._used_ports:
        self._used_ports.add((flow.nw_proto,port))
        return port
      port += 1
      if port >= 65534:
        port = 49152
        cycle += 1

    log.warn("No ports to give!")
    return None

  @property
  def _outside_eth (self):
    if self._connection is None: return None
    #return self._connection.eth_addr
    return
self._connection.ports[self._outside_portno].hw_addr

  def _handle_FlowRemoved (self, event):
    pass

  @staticmethod
  def strip_match (o):
    m = of.ofp_match()

    fields = 'dl_dst dl_src nw_dst nw_src tp_dst tp_src
dl_type nw_proto'

    for f in fields.split():
      setattr(m, f, getattr(o, f))

    return m

  @staticmethod
  def make_match (o):
    return NAT.strip_match(of.ofp_match.from_packet(o))

  def _handle_PacketIn (self, event):
    if self._outside_eth is None: return

    #print
    #print
"PACKET",event.connection.ports[event.port].name,event.po
rt,
    #print self.outside_port, self.make_match(event.ofp)

    incoming = event.port == self._outside_portno

    if self._gateway_eth is None:
      # Need to find gateway MAC -- send an ARP
      self._arp_for_gateway()
      return

    packet = event.parsed
    dns_hack = False

    # We only handle TCP and UDP
    tcpp = packet.find('tcp')
    if not tcpp:
      tcpp = packet.find('udp')
      if not tcpp: return
      if tcpp.dstport == 53 and tcpp.prev.dstip ==
self.inside_ip:
        if self.dns_ip and not incoming:
          # Special hack for DNS since we've lied and
claimed to be the server
          dns_hack = True
    ipp = tcpp.prev

    if not incoming:
      # Assume we only NAT public addresses
      if self._is_local(ipp.dstip) and not dns_hack:
return
    else:
      # Assume we only care about ourselves
      if ipp.dstip != self.outside_ip: return

    match = self.make_match(event.ofp)

    if incoming:
      match2 = match.clone()
      match2.dl_dst = None # See note below
      record = self._record_by_incoming.get(match2)
      if record is None:
        # Ignore for a while
        fm = of.ofp_flow_mod()
        fm.idle_timeout = 1
        fm.hard_timeout = 10
        fm.match = of.ofp_match.from_packet(event.ofp)
        event.connection.send(fm)
        return
      log.debug("%s reinstalled", record)
      record.incoming_fm.data = event.ofp # Hacky!
    else:
      record = self._record_by_outgoing.get(match)
      if record is None:
        record = Record()

        record.real_srcport = tcpp.srcport
        record.fake_srcport = self._pick_port(match)

        # Outside heading in
        fm = of.ofp_flow_mod()
        fm.flags |= of.OFPFF_SEND_FLOW_REM
        fm.hard_timeout = FLOW_TIMEOUT

        fm.match = match.flip()
        fm.match.in_port = self._outside_portno
        fm.match.nw_dst = self.outside_ip
        fm.match.tp_dst = record.fake_srcport
        fm.match.dl_src = self._gateway_eth

        # We should set dl_dst, but it can get in the
way.  Why?  Because
        # in some situations, the ARP may ARP for and get
the local host's
        # MAC, but in others it may not.
        #fm.match.dl_dst = self._outside_eth
        fm.match.dl_dst = None


fm.actions.append(of.ofp_action_dl_addr.set_src(packet.ds
t))

fm.actions.append(of.ofp_action_dl_addr.set_dst(packet.sr
c))

fm.actions.append(of.ofp_action_nw_addr.set_dst(ipp.srcip
))

        if dns_hack:
          fm.match.nw_src = self.dns_ip

fm.actions.append(of.ofp_action_nw_addr.set_src(self.insi
de_ip))
        if record.fake_srcport != record.real_srcport:

fm.actions.append(of.ofp_action_tp_port.set_dst(record.re
al_srcport))

        fm.actions.append(of.ofp_action_output(port =
event.port))

        record.incoming_match =
self.strip_match(fm.match)
        record.incoming_fm = fm

        # Inside heading out
        fm = of.ofp_flow_mod()
        fm.data = event.ofp
        fm.flags |= of.OFPFF_SEND_FLOW_REM
        fm.hard_timeout = FLOW_TIMEOUT
        fm.match = match.clone()
        fm.match.in_port = event.port

fm.actions.append(of.ofp_action_dl_addr.set_src(self._out
side_eth))

fm.actions.append(of.ofp_action_nw_addr.set_src(self.outs
ide_ip))
        if dns_hack:

fm.actions.append(of.ofp_action_nw_addr.set_dst(self.dns_
ip))
        if record.fake_srcport != record.real_srcport:

fm.actions.append(of.ofp_action_tp_port.set_src(record.fa
ke_srcport))

fm.actions.append(of.ofp_action_dl_addr.set_dst(self._gat
eway_eth))
        fm.actions.append(of.ofp_action_output(port =
self._outside_portno))

        record.outgoing_match =
self.strip_match(fm.match)
        record.outgoing_fm = fm

        self._record_by_incoming[record.incoming_match] =
record
        self._record_by_outgoing[record.outgoing_match] =
record

        log.debug("%s installed", record)
      else:
        log.debug("%s reinstalled", record)
        record.outgoing_fm.data = event.ofp # Hacky!

    record.touch()

    # Send/resend the flow mods

    if incoming:
      data = record.outgoing_fm.pack() +
record.incoming_fm.pack()
    else:
      data = record.incoming_fm.pack() +
record.outgoing_fm.pack()
    self._connection.send(data)

    # We may have set one of the data fields, but they
should be reset since
    # they won't be valid in the future.  Kind of hacky.
    record.outgoing_fm.data = None
    record.incoming_fm.data = None

  def __handle_dpid_ConnectionUp (self, event):
    if event.dpid != self.dpid:
      return
    self._start(event.connection)

  def _start (self, connection):
    self._connection = connection

    self._outside_portno =
connection.ports[self.outside_port].port_no

    fm = of.ofp_flow_mod()
    fm.match.in_port = self._outside_portno
    fm.priority = 1
    connection.send(fm)

    fm = of.ofp_flow_mod()
    fm.match.in_port = self._outside_portno
    fm.match.dl_type = 0x800 # IP
    fm.match.nw_dst = self.outside_ip

fm.actions.append(of.ofp_action_output(port=of.OFPP_CONTR
OLLER))
    fm.priority = 2
    connection.send(fm)
```

NAT Library (POX) =

Control logic

Notified about first packet in every flow…

```
from pox.core import core
import pox
log = core.getLogger()

from pox.lib.packet.ipv4 import ipv4
from pox.lib.packet.arp import arp
import p

from pox
from pox
from pox
str_to_d
from pox
from pox
import p

import p
from pox

import t
import r

FLOW_TIM
FLOW_MEM


class Re
  def __
    self
    self
    self.incoming_match = None
    self.real_srcport = None
    self.fake_srcport = None
    self.outgoing_fm = None
    self.incoming_fm = None

  @property
  def expired (self):
    return time.time() > self._expires_at

  def touch (self):
    self._expires_at = time.time() + FLOW_MEMORY_TIMEOUT

  def __str__ (self):
    s = "%s:%s" % (self.outgoing_match.nw_src,
self.real_srcport)
    if self.fake_srcport != self.real_srcport:
      s += "/%s" % (self.fake_srcport,)
    s += " -> %s:%s" % (self.outgoing_match.nw_dst,
self.outgoing_match.tp_dst)
    return s


class NAT (object):
  def __init__ (self, inside_ip, outside_ip, gateway_ip,
dns_ip, outside_port,
      dpid, subnet = None):

    self.inside_ip = inside_ip
    self.outside_ip = outside_ip
    self.gateway_ip = gateway_ip
    self.dns_ip = dns_ip # Or None
    self.outside_port = outside_port
    self.dpid = dpid
    self.subnet = subnet

    self._outside_portno = None
    self._gateway_eth = None
    self._connection = None

    # Which NAT ports have we used?
    # proto means TCP or UDP
    self._used_ports = set() # (proto,port)

    # Flow records indexed in both directions
    # match -> Record
    self._record_by_outgoing = {}
    self._record_by_incoming = {}

    core.listen_to_dependencies(self)

  def _all_dependencies_met (self):
    log.debug('Trying to start...')
    if self.dpid in core.openflow.connections:
      self._start(core.openflow.connections[self.dpid])
    else:
      core.openflow.addListenerByName('ConnectionUp',
          self.__handle_dpid_ConnectionUp)

    expire_timer = Timer(60, self._expire, recurring

    pire (self):
    = []
    in self._record_by_outgoing.itervalues():
    r.expired:

h]
h]

,r.fake


    return False
  if ip.in_network('192.168.0.0/16'): return True
  if ip.in_network('10.0.0.0/8'): return True
  if ip.in_network('172.16.0.0/12'): return True
  return False

def _pick_port (self, flow):
  """
  Gets a possibly-remapped outside port

  flow is the match of the connection
  returns port (maybe from flow, maybe not)
  """

  port = flow.tp_src

  if port < 1024:
    # Never allow these
    port = random.randint(49152, 65534)

  # Pretty sloppy!

  cycle = 0
  while cycle < 2:
    if (flow.nw_proto,port) not in self._used_ports:
      self._used_ports.add((flow.nw_proto,port))
      return port
    port += 1
    if port >= 65534:
      port = 49152
      cycle += 1

  log.warn("No ports to give!")
  return None

@property
def _outside_eth (self):
  if self._connection is None: return None
  #return self._connection.eth_addr
  return
self._connection.ports[self._outside_portno].hw_addr

  def _handle_FlowRemoved (self, event):
    pass

@staticmethod
def strip_match (o):
  m = of.ofp_match()

  fields = 'dl_dst dl_src nw_dst nw_src tp_dst tp_src
dl_type nw_proto'

  for f in fields.split():
    setattr(m, f, getattr(o, f))

  return m

@staticmethod
def make_match (o):
  return NAT.strip_match(of.ofp_match.from_packet(o))

def _handle_PacketIn (self, event):
  if self._outside_eth is None: return

  #print
  #print
"PACKET",event.connection.ports[event.port].name,event.po
rt,
    #print self.outside_port, self.make_match(event.ofp)

    incoming = event.port == self._outside_portno

    if self._gateway_eth is None:
      # Need to find gateway MAC -- send an ARP
      self._arp_for_gateway()
      return

    packet = event.parsed
    dns_hack = False

    # We only handle TCP and UDP
    tcpp = packet.find('tcp')
    if not tcpp:
      tcpp = packet.find('udp')
      if not tcpp: return
      if tcpp.dstport == 53 and tcpp.prev.dstip ==
self.inside_ip:
        if self.dns_ip and not incoming:
          # Special hack for DNS since we've lied and
claimed to be the server
          dns_hack = True
    ipp = tcpp.prev

    if not incoming:
      # Assume we only NAT public addresses
      if self._is_local(ipp.dstip) and not dns_hack:
return
      else:
        # Assume we only care about ourselves
        if ipp.dstip != self.outside_ip: return

    match = self.make_match(event.ofp)

    if incoming:
      match2 = match.clone()
      match2.dl_dst = None # See note below
      record = self._record_by_incoming.get(match2)
      if record is None:
        # Ignore for a while
        fm = of.ofp_flow_mod()
        fm.idle_timeout = 1
        fm.hard_timeout = 10
        fm.match = of.ofp_match.from_packet(event.ofp)
        event.connection.send(fm)
        return
      log.debug("%s reinstalled", record)
      record.incoming_fm.data = event.ofp # Hacky!
    else:
      record = self._record_by_outgoing.get(match)
      if record is None:
        record = Record()

        record.real_srcport = tcpp.srcport
        record.fake_srcport = self._pick_port(match)

        # Outside heading in
        fm = of.ofp_flow_mod()
        fm.flags |= of.OFPFF_SEND_FLOW_REM
        fm.hard_timeout = FLOW_TIMEOUT

        fm.match = match.flip()
        fm.match.in_port = self._outside_portno
        fm.match.nw_dst = self.outside_ip
        fm.match.tp_dst = record.fake_srcport
        fm.match.dl_src = self._gateway_eth

        # We should set dl_dst, but it can get in the
way.  Why?  Because
        # in some situations, the ARP may ARP for and get
the local host's
        # MAC, but in others it may not.
        #fm.match.dl_dst = self._outside_eth
        fm.match.dl_dst = None


fm.actions.append(of.ofp_action_dl_addr.set_src(packet.ds
t))

fm.actions.append(of.ofp_action_dl_addr.set_dst(packet.sr
c))

fm.actions.append(of.ofp_action_nw_addr.set_dst(ipp.srcip
))

        if dns_hack:
          fm.match.nw_src = self.dns_ip

fm.actions.append(of.ofp_action_nw_addr.set_src(self.insi
de_ip))
        if record.fake_srcport != record.real_srcport:

fm.actions.append(of.ofp_action_tp_port.set_dst(record.re
al_srcport))

        fm.actions.append(of.ofp_action_output(port =
event.port))

        record.incoming_match =
self.strip_match(fm.match)
        record.incoming_fm = fm

        # Inside heading out
        fm = of.ofp_flow_mod()
        fm.data = event.ofp
        fm.flags |= of.OFPFF_SEND_FLOW_REM
        fm.hard_timeout = FLOW_TIMEOUT
        fm.match = match.clone()
        fm.match.in_port = event.port

fm.actions.append(of.ofp_action_dl_addr.set_src(self._out
side_eth))

fm.actions.append(of.ofp_action_nw_addr.set_src(self.outs
ide_ip))
        if dns_hack:

fm.actions.append(of.ofp_action_nw_addr.set_dst(self.dns_
ip))
        if record.fake_srcport != record.real_srcport:

fm.actions.append(of.ofp_action_tp_port.set_src(record.fa
ke_srcport))

fm.actions.append(of.ofp_action_dl_addr.set_dst(self._gat
eway_eth))
        fm.actions.append(of.ofp_action_output(port =
self._outside_portno))

        record.outgoing_match =
self.strip_match(fm.match)
        record.outgoing_fm = fm

        self._record_by_incoming[record.incoming_match] =
record
        self._record_by_outgoing[record.outgoing_match] =
record

        log.debug("%s installed", record)
      else:
        log.debug("%s reinstalled", record)
        record.outgoing_fm.data = event.ofp # Hacky!

    record.touch()

    # Send/resend the flow mods

    if incoming:
      data = record.outgoing_fm.pack() +
record.incoming_fm.pack()
    else:
      data = record.incoming_fm.pack() +
record.outgoing_fm.pack()
    self._connection.send(data)

    # We may have set one of the data fields, but they
should be reset since
    # they won't be valid in the future.  Kind of hacky.
    record.outgoing_fm.data = None
    record.incoming_fm.data = None

  def __handle_dpid_ConnectionUp (self, event):
    if event.dpid != self.dpid:
      return
    self._start(event.connection)

  def _start (self, connection):
    self._connection = connection

    self._outside_portno =
connection.ports[self.outside_port].port_no

    fm = of.ofp_flow_mod()
    fm.match.in_port = self._outside_portno
    fm.priority = 1
    connection.send(fm)

    fm = of.ofp_flow_mod()
    fm.match.in_port = self._outside_portno
    fm.match.dl_type = 0x800 # IP
    fm.match.nw_dst = self.outside_ip

fm.actions.append(of.ofp_action_output(port=of.OFPP_CONTR
OLLER))
    fm.priority = 2
    connection.send(fm)
```

NAT Library (POX) =

Control logic

Notified about first packet in every flow…

Control logic

…reply with actions for first packet…

NAT Library (POX) =

Control logic

Notified about first packet in every flow…

…update NAT state…

Datastore

Control logic

…reply with actions for first packet…

NAT Library (POX) =

Control logic

Notified about first packet in every flow…

…update NAT state…

Datastore

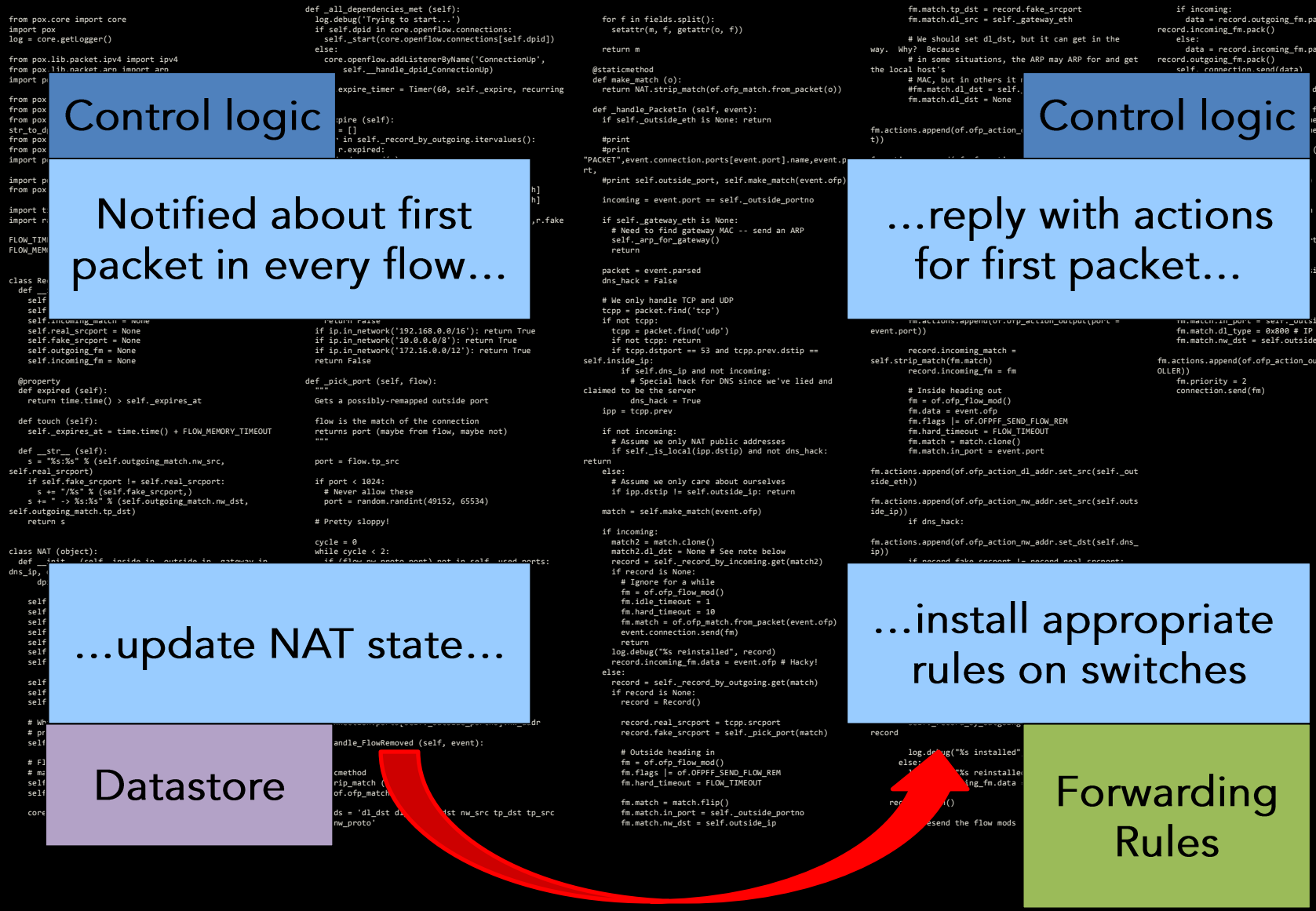
Control logic

…reply with actions for first packet…

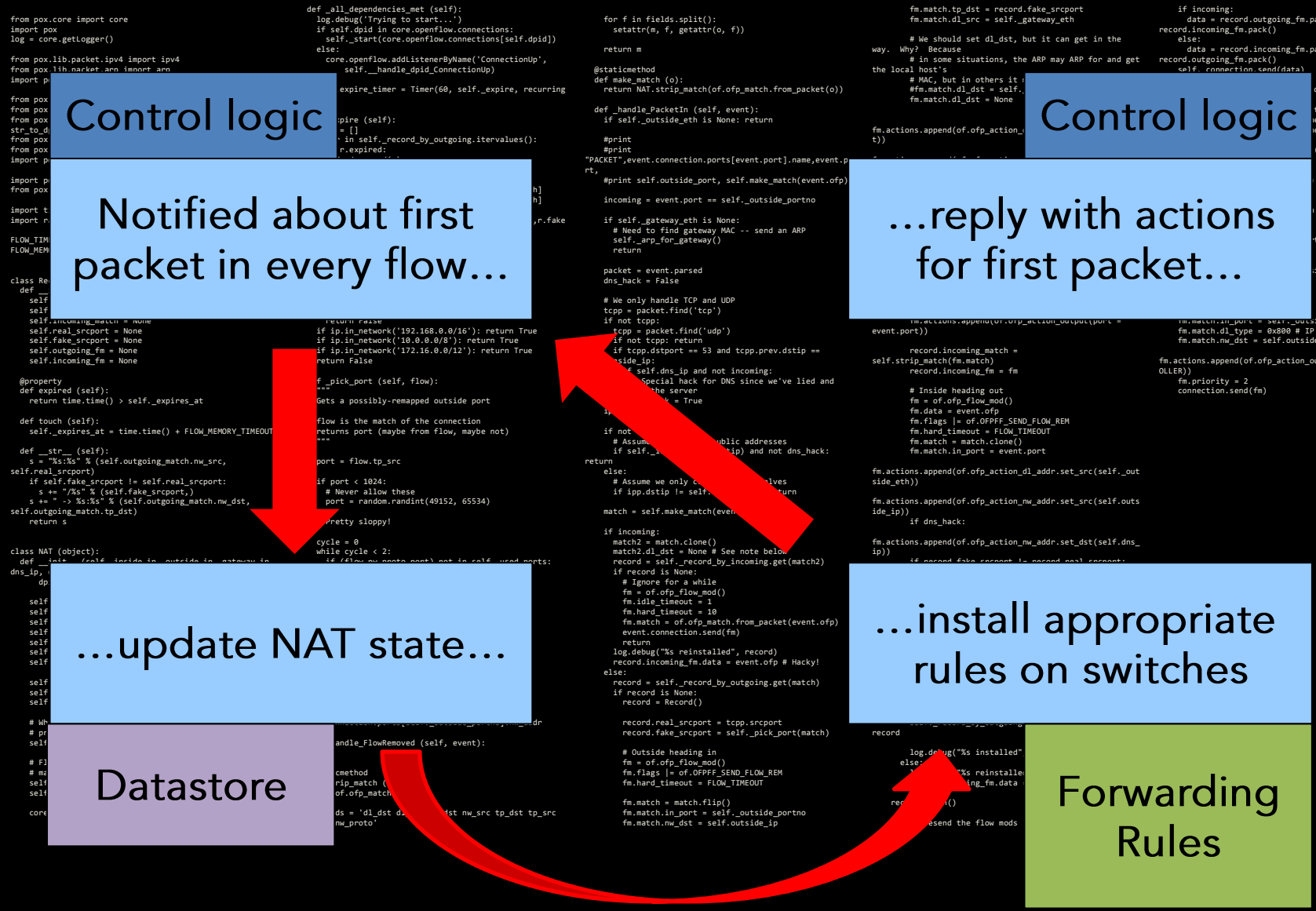
…install appropriate rules on switches

Forwarding Rules

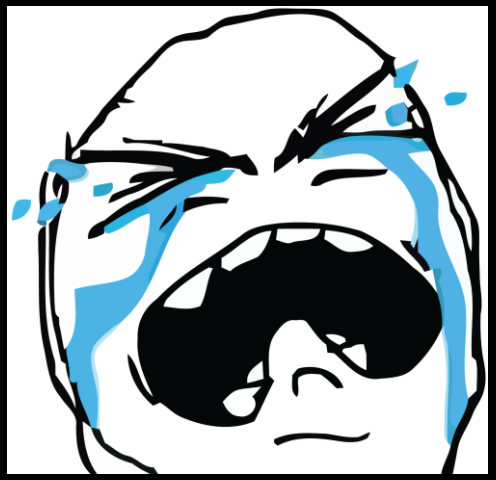
NAT Library (POX) =



NAT Library (POX) =



Bugs!

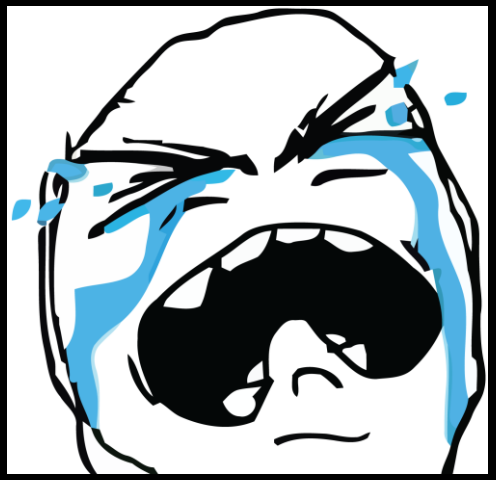


Datastore

Control logic

Forwarding Rules

Bugs!



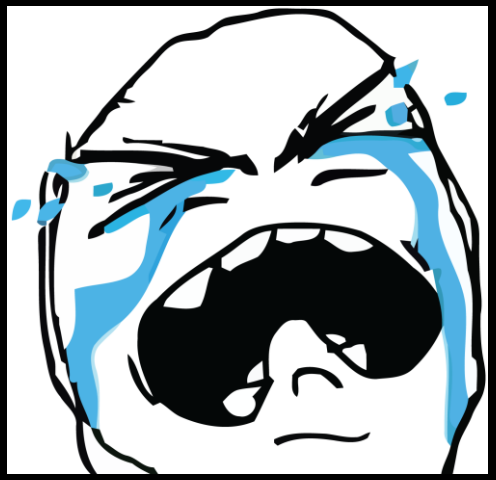
Datastore

Control logic

Inconsistent packet-handling between controller and switches

Forwarding Rules

Bugs!



Datastore

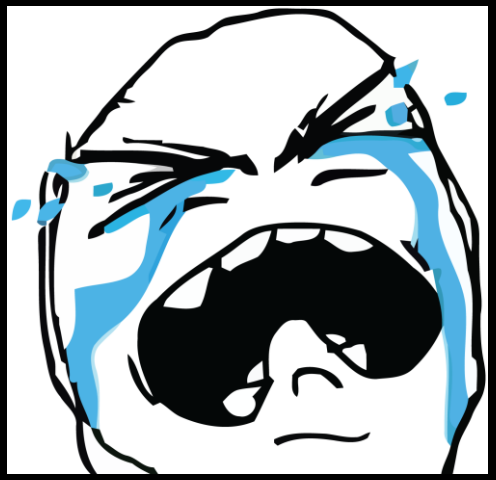
Control logic

Inconsistent packet-handling between controller and switches

Switch fails to notify controller when state update needed

Forwarding Rules

Bugs!



Datastore

Control logic

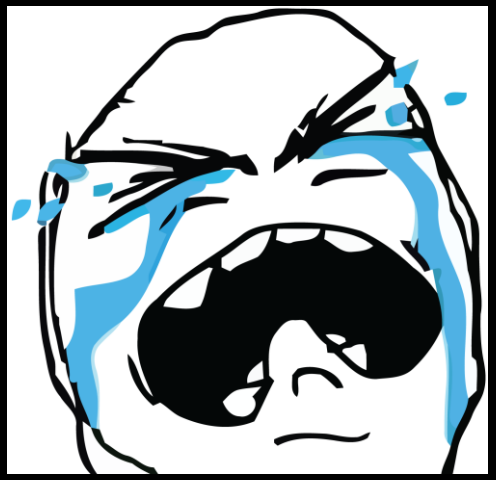
Inconsistent packet-handling between controller and switches

Switch fails to notify controller when state update needed

Switch notifies controller more than necessary

Forwarding Rules

Bugs!



Datastore

Control logic

Inconsistent packet-handling between controller and switches

Switch fails to notify controller when state update needed

Switch notifies controller more than necessary

Hard to verify Java, Python, …

Multiple tiers make it even harder.

Forwarding Rules

FLOWLOG

A Tierless SDN Programming Language

Datastore

Control logic

Forwarding Rules

Contributions

- Automatic compilation to flow tables (…including state and state updates)
- Built-in cross-tier verification support

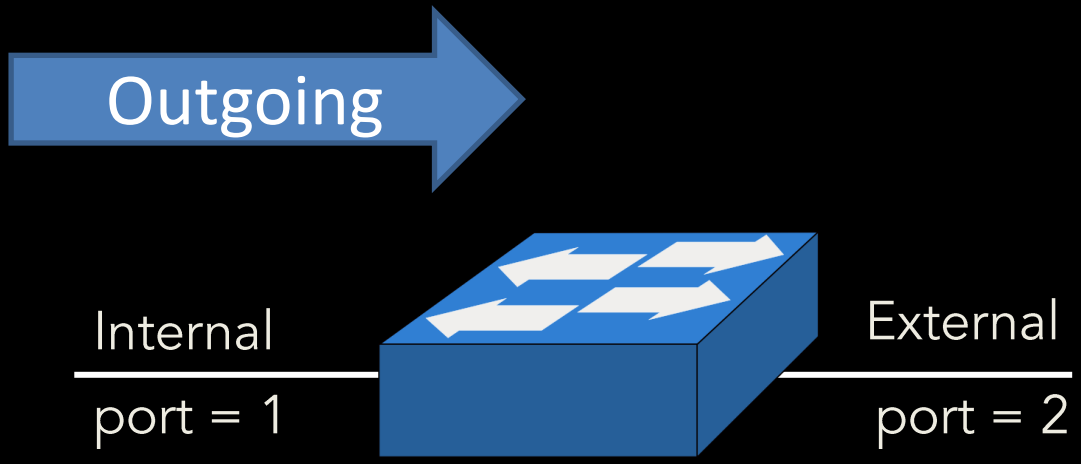
FLOWLOG

A Tierless SDN Programming Language

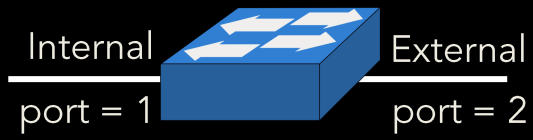
Contributions

- Automatic compilation to flow tables
  (…including state and state updates)
- Built-in cross-tier verification support

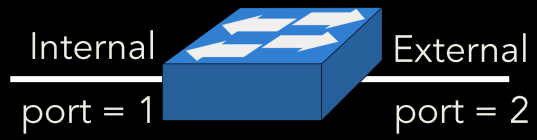
NAT IN FLOWLOG



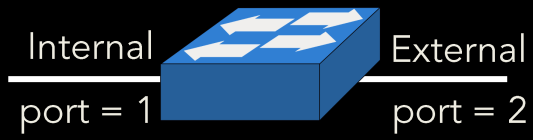
Public IP = `192.168.100.100`
Public MAC = `00:00:00:00:00:02`



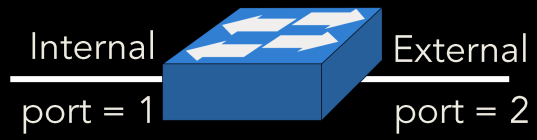

```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;

ON tcp_packet(p) WHERE p.locPt = 1 AND
            nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

ON tcp_packet(p) WHERE p.locPt = 1 AND
            NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```




```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;

ON tcp_packet(p) WHERE p.locPt = 1 AND
                nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

ON tcp_packet(p) WHERE p.locPt = 1 AND
                NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```




```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;

ON tcp_packet(p) WHERE p.locPt = 1 AND
                nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

ON tcp_packet(p) WHERE p.locPt = 1 AND
                NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```



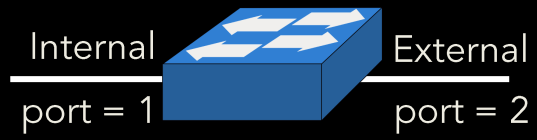

```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;
```

Existing Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;
```

New Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```



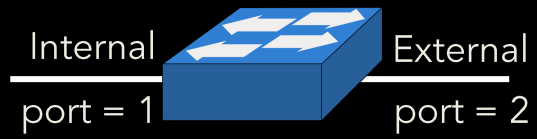

```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;
```

Existing Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;
```

New Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```



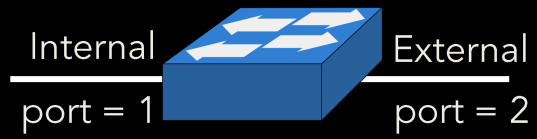

```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;
```

Existing Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;
```

New Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```



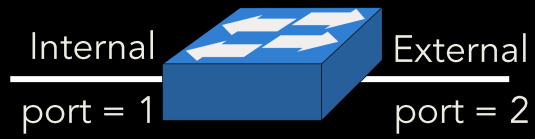

```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;
```

Existing Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;
```

New Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```



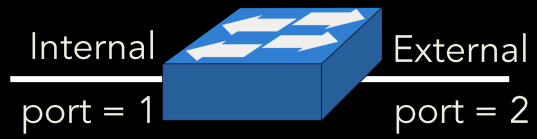

```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;
```

Existing Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;
```

New Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```



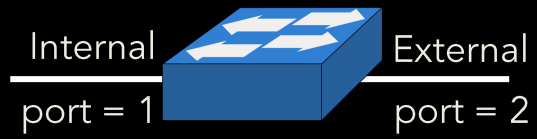

```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;
```

Existing Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;
```

New Flow:

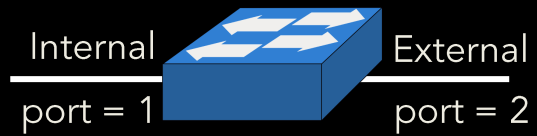
```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```




```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;

ON tcp_packet(p) WHERE p.locPt = 1 AND
                  nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

ON tcp_packet(p) WHERE p.locPt = 1 AND
                  NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```

Existing Flow:

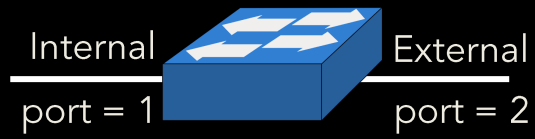
New Flow:




```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;

ON tcp_packet(p) WHERE p.locPt = 1 AND
                  nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

ON tcp_packet(p) WHERE p.locPt = 1 AND
                  NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```

Existing Flow:

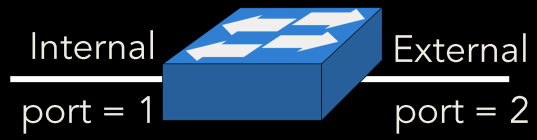
New Flow:




```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;

ON tcp_packet(p) WHERE p.locPt = 1 AND
                  nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

ON tcp_packet(p) WHERE p.locPt = 1 AND
                  NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```

Existing Flow:

New Flow:



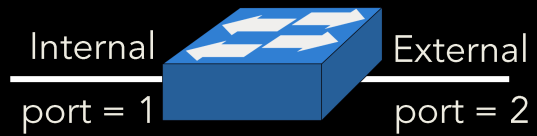

```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;
```

Existing Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;
```

New Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```



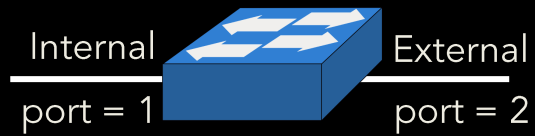

```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;
```

Existing Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                  nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;
```

New Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                  NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```




```
TABLE nat(macaddr, ipaddr, tpport, tpport);
VAR nextport: tpport = 10000;
```

Existing Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                nat(p.dlSrc, p.nwSrc, p.tpSrc, natport):
  DO forward(new) WHERE new.tpSrc = natport AND
       new.nwSrc = 192.168.100.100 AND
       new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;
```

New Flow:

```
ON tcp_packet(p) WHERE p.locPt = 1 AND
                NOT nat(p.dlSrc, p.nwSrc, p.tpSrc, ANY):
  DO forward(new) WHERE new.tpSrc = nextport AND
    new.nwSrc = 192.168.100.100 AND
    new.dlSrc = 00:00:00:00:00:02 AND new.locPt = 2;

  INSERT(p.dlSrc, p.nwSrc, p.tpSrc, nextport) INTO nat;

  INCREMENT nextport;
```

And that's it.

Another example

Campus Network Controller

Campus Network Controller

Campus Network Controller

Campus Network Controller

Campus Network Controller

Campus Network Controller

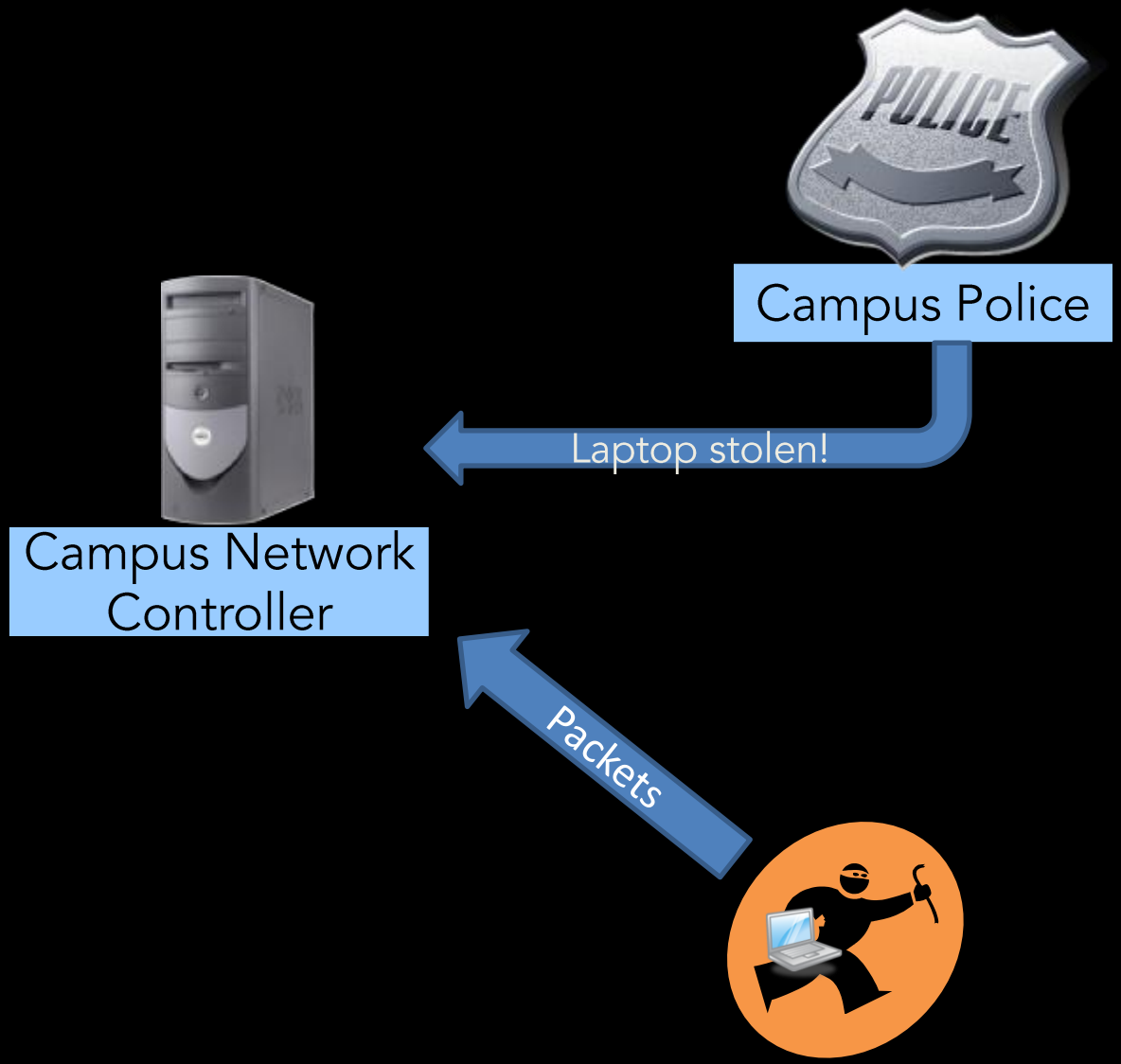
Campus Police

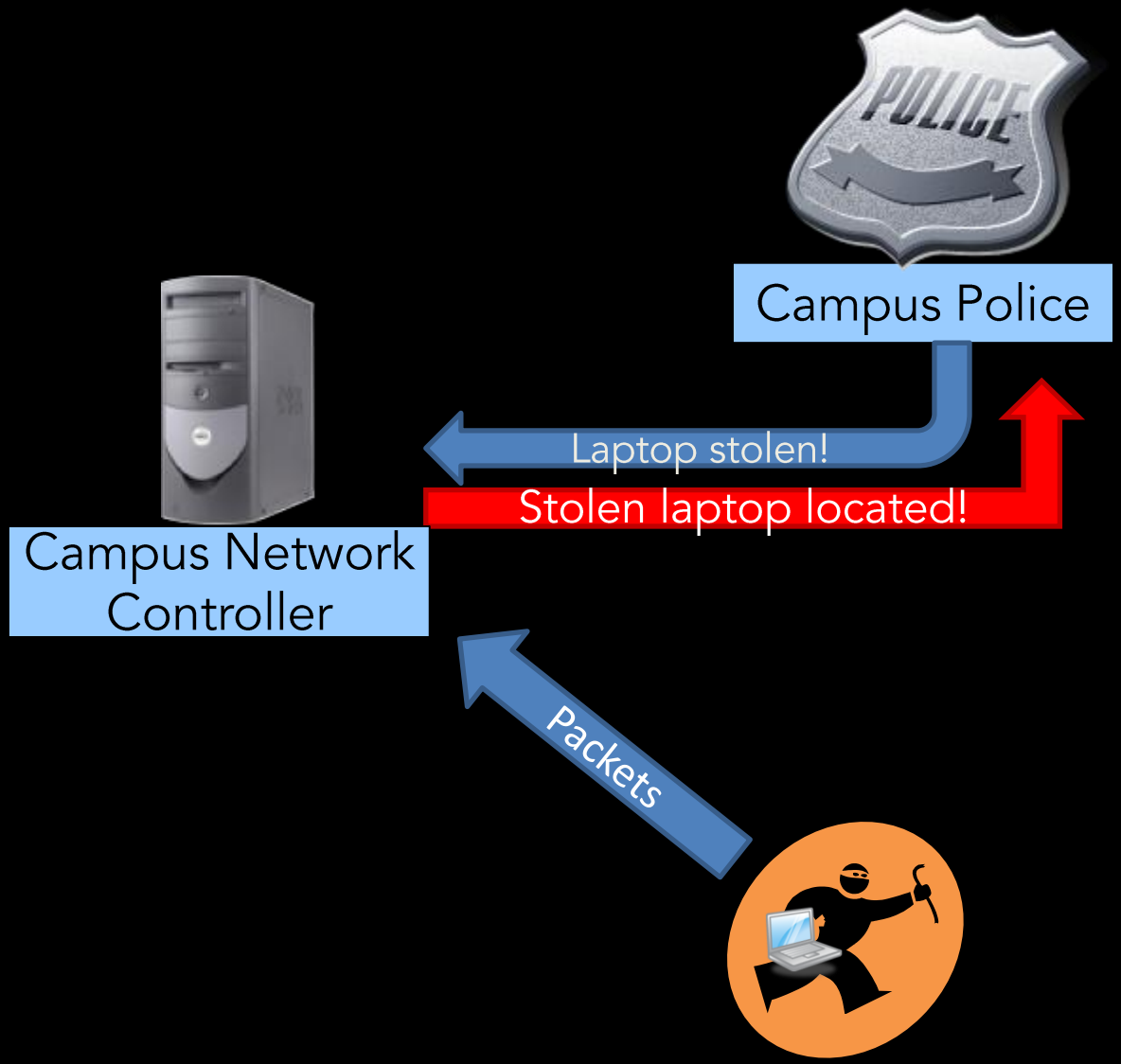
Campus Network Controller

Campus Police

Laptop stolen!

Campus Network Controller


POLICE
Campus Police
Laptop stolen!
Campus Network Controller
Packets


POLICE
Campus Police
Laptop stolen!
Stolen laptop located!
Campus Network Controller
Packets

Core Logic

Northbound API

Remote Tables

Campus Police

Laptop stolen!

Stolen laptop located!

Campus Network Controller

Packets

Core Logic

Northbound API

Remote Tables

pus Police

cated!

```
// Event API
EVENT stolen_laptop_report
  {laptop: macaddr};
EVENT stolen_laptop_cancel
  {laptop: macaddr};
EVENT stolen_laptop_found
  {laptop: macaddr,
   swid: switchid,
   time: int};

// Named pipe for event output
OUTGOING notify_police(stolen_laptop_found)
  THEN SEND TO 127.0.0.1:5050;
```

Core Logic

Northbound API

Remote Tables

Thrift

```
// Event API
EVENT stolen_laptop_report
  {laptop: macaddr};
EVENT stolen_laptop_cancel
  {laptop: macaddr};
EVENT stolen_laptop_found
  {laptop: macaddr,
   swid: switchid,
   time: int};

// Named pipe for event output
OUTGOING notify_police(stolen_laptop_found)
  THEN SEND TO 127.0.0.1:5050;
```

Core Logic

Northbound API

Remote Tables

pus Police

cated!

Thrift

```
// Event API
EVENT stolen_laptop_report
  {laptop: macaddr};
EVENT stolen_laptop_cancel
  {laptop: macaddr};
EVENT stolen_laptop_found
  {laptop: macaddr,
   swid: switchid,
   time: int};

// Named pipe for event output
OUTGOING notify_police(stolen_laptop_found)
  THEN SEND TO 127.0.0.1:5050;
```

Core Logic

Northbound API

Remote Tables

Thrift

```
// Event API
EVENT stolen_laptop_report
  {laptop: macaddr};
EVENT stolen_laptop_cancel
  {laptop: macaddr};
EVENT stolen_laptop_found
  {laptop: macaddr,
   swid: switchid,
   time: int};

// Named pipe for event output
OUTGOING notify_police(stolen_laptop_found)
  THEN SEND TO 127.0.0.1:5050;
```

Core Logic

Northbound API

Remote Tables

pus Police

cated!

Thrift

```
// Event API
EVENT stolen_laptop_report
  {laptop: macaddr};
EVENT stolen_laptop_cancel
  {laptop: macaddr};
EVENT stolen_laptop_found
  {laptop: macaddr,
   swid: switchid,
   time: int};

// Named pipe for event output
OUTGOING notify_police(stolen_laptop_found)
  THEN SEND TO 127.0.0.1:5050;
```

Core Logic

Northbound API

Remote Tables

Thrift

```
// Event API
EVENT stolen_laptop_report
  {laptop: macaddr};
EVENT stolen_laptop_cancel
  {laptop: macaddr};
EVENT stolen_laptop_found
  {laptop: macaddr,
   swid: switchid,
   time: int};

// Named pipe for event output
OUTGOING notify_police(stolen_laptop_found)
  THEN SEND TO 127.0.0.1:5050;
```

Core Logic

Northbound API

Remote Tables

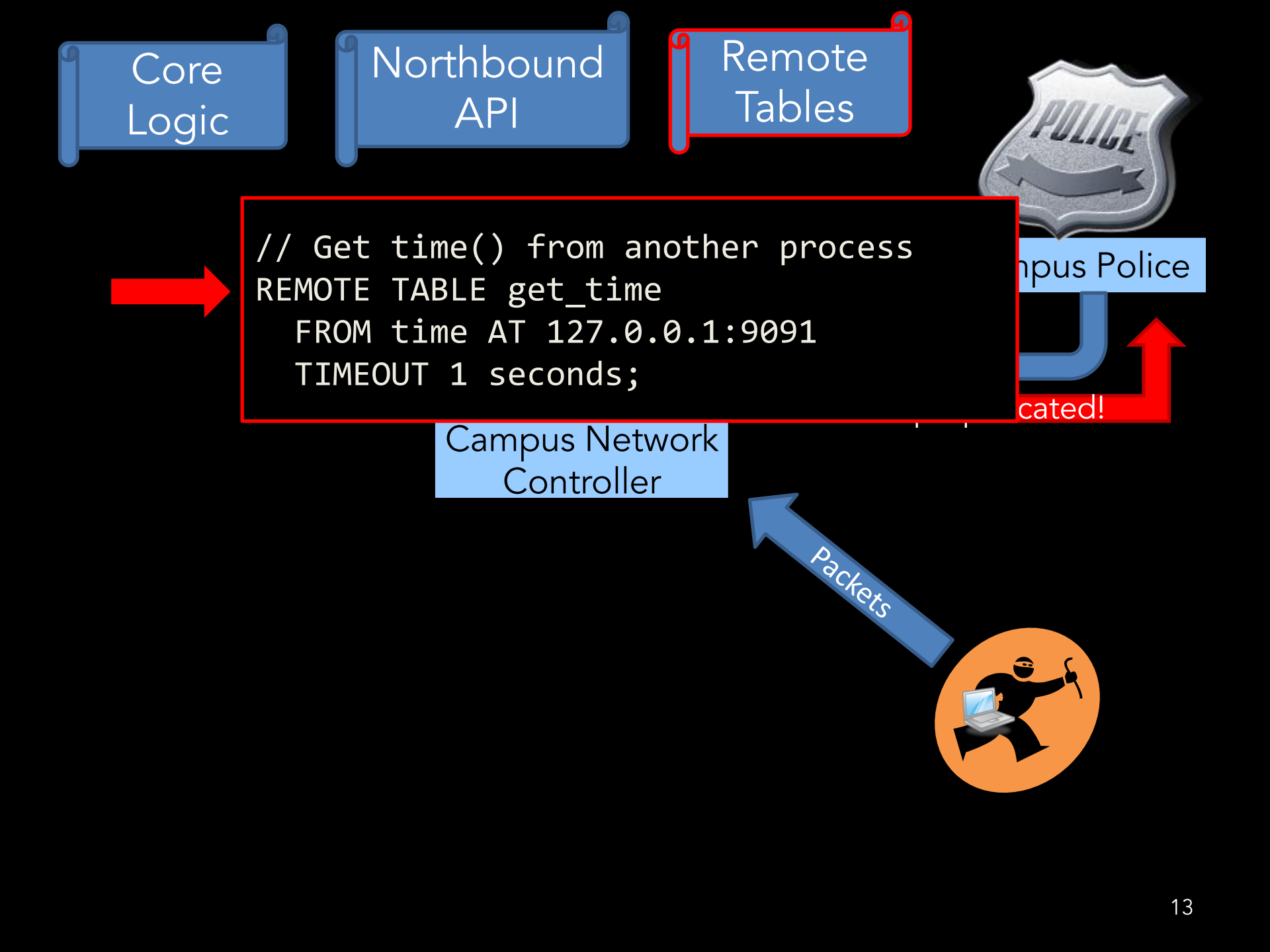
Campus Police

Laptop stolen!

Stolen laptop located!

Campus Network Controller

Packets


Core Logic
Northbound API
Remote Tables
// Get time() from another process
REMOTE TABLE get_time
FROM time AT 127.0.0.1:9091
TIMEOUT 1 seconds;
POLICE
npus Police
cated!
Campus Network Controller
Packets

Core Logic

Northbound API

Remote Tables

Campus Police

Laptop stolen!

Stolen laptop located!

Campus Network Controller

Packets

Core Logic

Northbound API

Remote Tables

Police

```
// store MAC of stolen devices
TABLE stolen(macaddr);

// react to events from police
ON stolen_laptop_cancel(ev):
  DELETE (ev.mac) FROM stolen;
ON stolen_laptop_report(ev):
  INSERT (ev.mac) INTO stolen;

// notify police if stolen laptop appears
ON packet(pkt) WHERE
    stolen(pkt.dlSrc) AND
    edgePort(pkt.locSw, pkt.locPt):
  DO notify_police(sto) WHERE
    sto.mac = pkt.dlSrc AND
    sto.swid = pkt.locSw AND
    get_time(sto.time);
```

Core Logic

Northbound API

Remote Tables

Police

```
// store MAC of stolen devices
TABLE stolen(macaddr);

// react to events from police
ON stolen_laptop_cancel(ev):
  DELETE (ev.mac) FROM stolen;
ON stolen_laptop_report(ev):
  INSERT (ev.mac) INTO stolen;

// notify police if stolen laptop appears
ON packet(pkt) WHERE
    stolen(pkt.dlSrc) AND
    edgePort(pkt.locSw, pkt.locPt):
  DO notify_police(sto) WHERE
    sto.mac = pkt.dlSrc AND
    sto.swid = pkt.locSw AND
    get_time(sto.time);
```

Core Logic

Northbound API

Remote Tables

Police

```
// store MAC of stolen devices
TABLE stolen(macaddr);

// react to events from police
ON stolen_laptop_cancel(ev):
  DELETE (ev.mac) FROM stolen;
ON stolen_laptop_report(ev):
  INSERT (ev.mac) INTO stolen;

// notify police if stolen laptop appears
ON packet(pkt) WHERE
    stolen(pkt.dlSrc) AND
    edgePort(pkt.locSw, pkt.locPt):
  DO notify_police(sto) WHERE
    sto.mac = pkt.dlSrc AND
    sto.swid = pkt.locSw AND
    get_time(sto.time);
```

Core Logic

Northbound API

Remote Tables

Police

```
// store MAC of stolen devices
TABLE stolen(macaddr);

// react to events from police
ON stolen_laptop_cancel(ev):
  DELETE (ev.mac) FROM stolen;
ON stolen_laptop_report(ev):
  INSERT (ev.mac) INTO stolen;

// notify police if stolen laptop appears
ON packet(pkt) WHERE
    stolen(pkt.dlSrc) AND
    edgePort(pkt.locSw, pkt.locPt):
  DO notify_police(sto) WHERE
    sto.mac = pkt.dlSrc AND
    sto.swid = pkt.locSw AND
    get_time(sto.time);
```

Core Logic

Northbound API

Remote Tables

Police

```
// store MAC of stolen devices
TABLE stolen(macaddr);

// react to events from police
ON stolen_laptop_cancel(ev):
  DELETE (ev.mac) FROM stolen;
ON stolen_laptop_report(ev):
  INSERT (ev.mac) INTO stolen;

// notify police if stolen laptop appears
ON packet(pkt) WHERE
    stolen(pkt.dlSrc) AND
    edgePort(pkt.locSw, pkt.locPt):
  DO notify_police(sto) WHERE
    sto.mac = pkt.dlSrc AND
    sto.swid = pkt.locSw AND
    get_time(sto.time);
```

Core Logic

Northbound API

Remote Tables

Police

```
// store MAC of stolen devices
TABLE stolen(macaddr);

// react to events from police
ON stolen_laptop_cancel(ev):
  DELETE (ev.mac) FROM stolen;
ON stolen_laptop_report(ev):
  INSERT (ev.mac) INTO stolen;

// notify police if stolen laptop appears
ON packet(pkt) WHERE
    stolen(pkt.dlSrc) AND
    edgePort(pkt.locSw, pkt.locPt):
  DO notify_police(sto) WHERE
    sto.mac = pkt.dlSrc AND
    sto.swid = pkt.locSw AND
    get_time(sto.time);
```

Core Logic

Northbound API

Remote Tables

Police

```
// store MAC of stolen devices
TABLE stolen(macaddr);

// react to events from police
ON stolen_laptop_cancel(ev):
  DELETE (ev.mac) FROM stolen;
ON stolen_laptop_report(ev):
  INSERT (ev.mac) INTO stolen;

// notify police if stolen laptop appears
ON packet(pkt) WHERE
    stolen(pkt.dlSrc) AND
    edgePort(pkt.locSw, pkt.locPt):
  DO notify_police(sto) WHERE
    sto.mac = pkt.dlSrc AND
    sto.swid = pkt.locSw AND
    get_time(sto.time);
```

Core Logic

Northbound API

Remote Tables

Police

```
// store MAC of stolen devices
TABLE stolen(macaddr);

// react to events from police
ON stolen_laptop_cancel(ev):
  DELETE (ev.mac) FROM stolen;
ON stolen_laptop_report(ev):
  INSERT (ev.mac) INTO stolen;

// notify police if stolen laptop appears
ON packet(pkt) WHERE
    stolen(pkt.dlSrc) AND
    edgePort(pkt.locSw, pkt.locPt):
  DO notify_police(sto) WHERE
    sto.mac = pkt.dlSrc AND
    sto.swid = pkt.locSw AND
    get_time(sto.time);
```

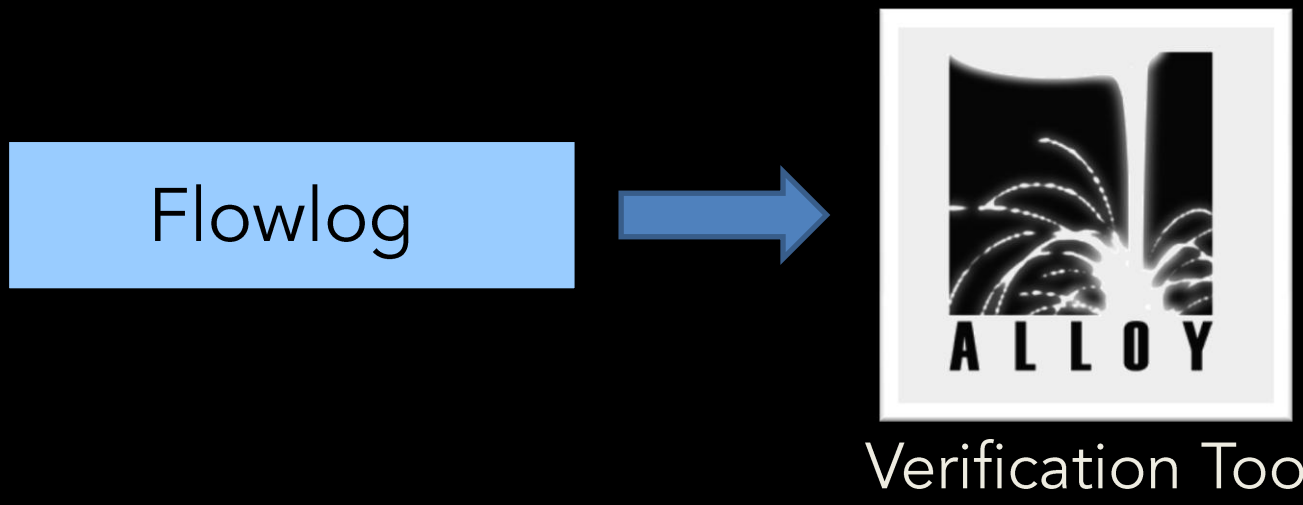
And that's it.

PROPERTY VERIFICATION

Example

"Only a stolen_laptop_cancel event can remove addresses from the stolen table."

Flowlog programs are equivalent to first-order logic formulas.



“Only a stolen_laptop_cancel event can remove addresses from the stolen table.”

```
assert onlyPoliceRemoveStolen {
   all st1, st2: State, ev: Event |
      transition[st1, ev, st2] and
        some (st1.stolen – st2.stolen)
         implies
         ev in EVstolen_laptop_cancel
}
```

“Only a stolen_laptop_cancel event can remove addresses from the stolen table.”

```
assert onlyPoliceRemoveStolen {
    all st1, st2: State, ev: Event |
        transition[st1, ev, st2] and
           some (st1.stolen – st2.stolen)
            implies
            ev in EVstolen_laptop_cancel
}
```

973 vars. 130 primary vars. 1781 clauses. 97ms.
No counterexample found. Assertion may be valid. 1ms.

“Only a stolen_laptop_**report** event can remove addresses from the stolen table.”

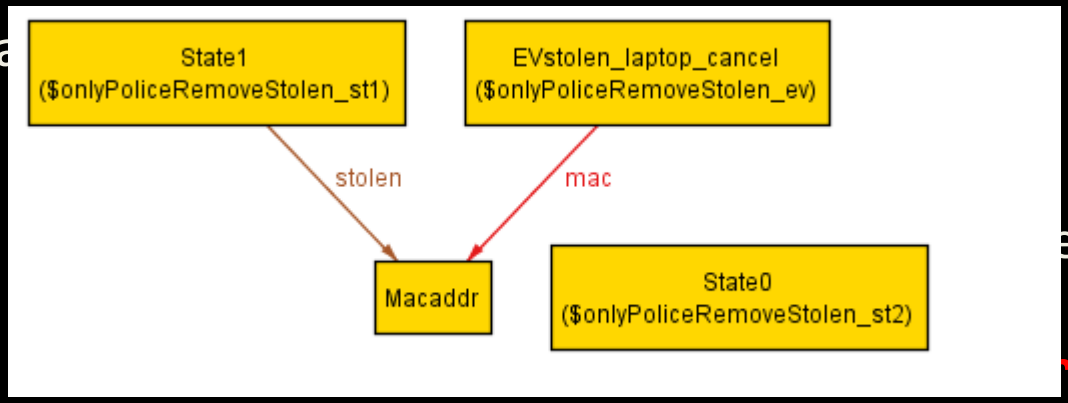
```
assert onlyPoliceRemoveStolen {
    all st1, st2: State, ev: Event |
        transition[st1, ev, st2] and
           some (st1.stolen – st2.stolen)
            implies
            ev in EVstolen_laptop_report
}
```

“Only a stolen_laptop_**report** event can remove addresses from the stolen table.”

```
assert onlyPoliceRemoveStolen {
    all st1, st2: State, ev: Event |
        transition[st1, ev, st2] and
           some (st1.stolen – st2.stolen)
            implies
            ev in EVstolen_laptop_report
}
```

973 vars. 130 primary vars. 1781 clauses. 34ms.
Counterexample found. Assertion is invalid. 23ms.

"Only a stolen_laptop_**report** event can remove addresses from the stolen table."



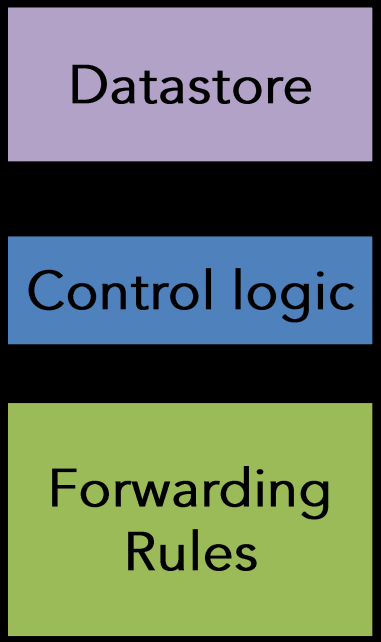

```
}
```

973 vars. 130 primary vars. 1781 clauses. 34ms.
Counterexample found. Assertion is invalid. 23ms.

"Only a stolen_laptop_cancel event can remove addresses from the stolen table."

"Only a stolen_laptop_cancel event can remove addresses from the *stolen table*."

"Only a stolen_laptop_cancel event can remove addresses from the *stolen table*."



"Whenever a packet is sent out of an external-facing port, the *controller has stored the packet's source address*."

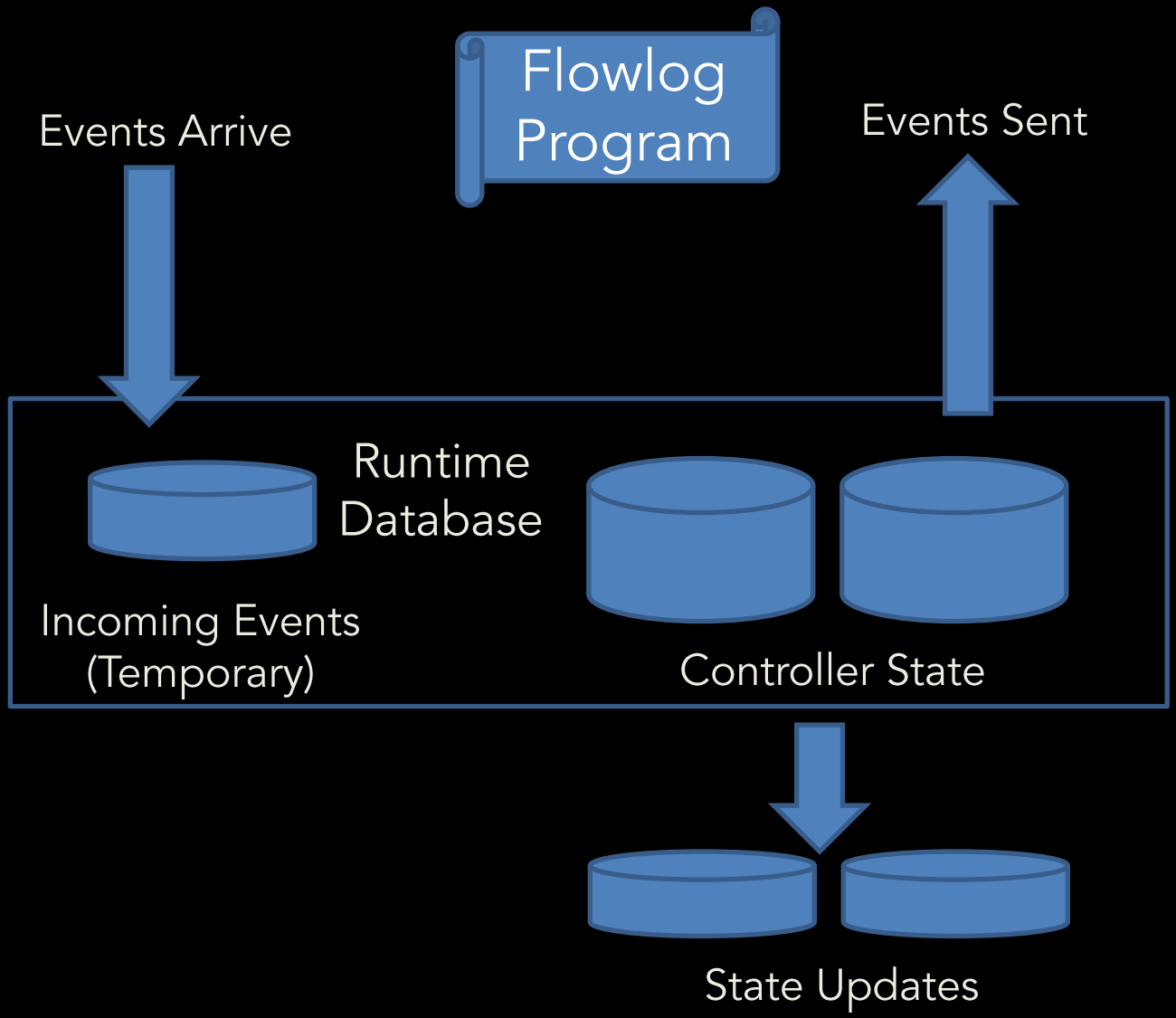
"Packets arriving on port 1 will always be forwarded with a new source port, and that port is always stored *in the NAT table*."

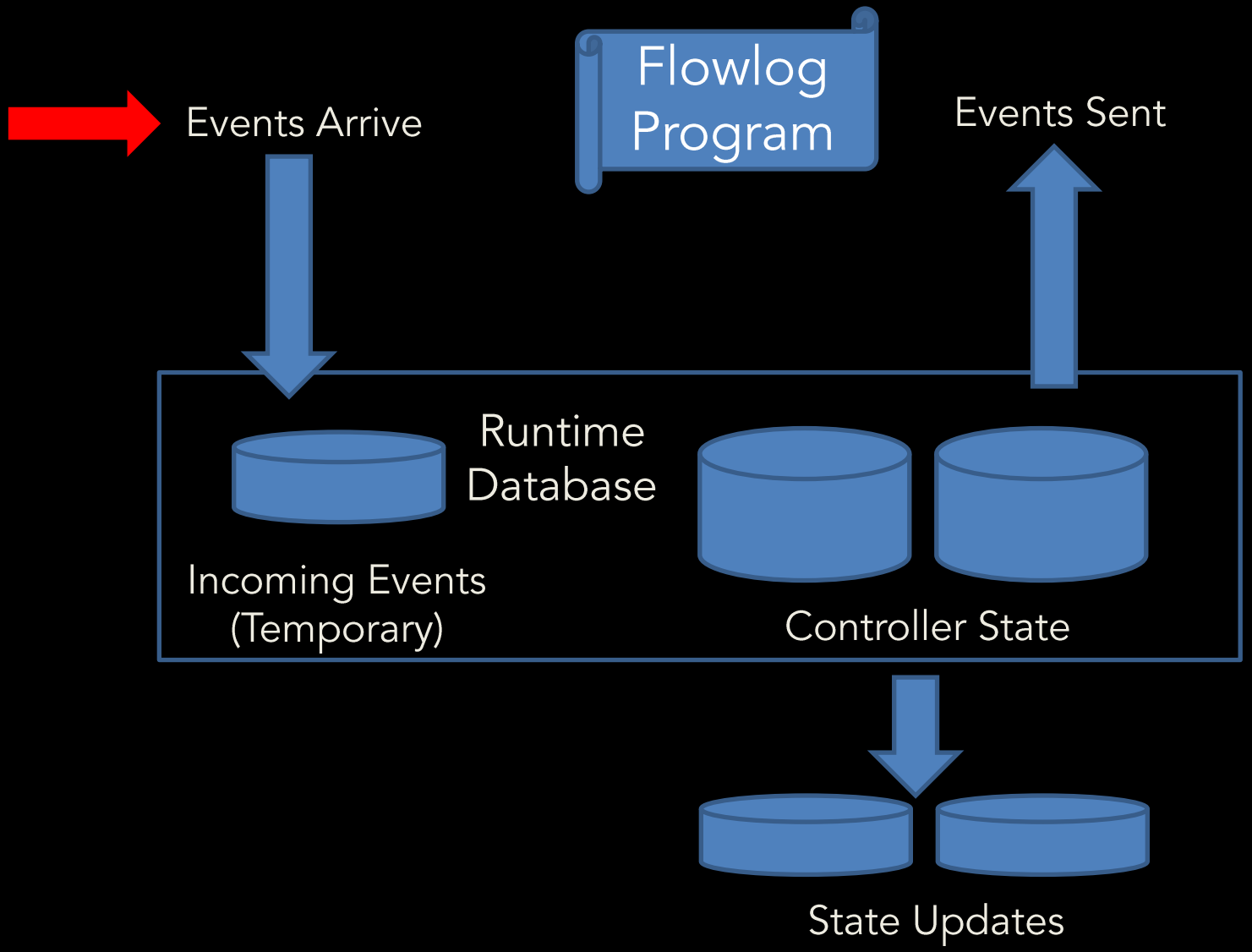
"Packets arriving on port 2 will be dropped unless their destination port *is used in the NAT table*."

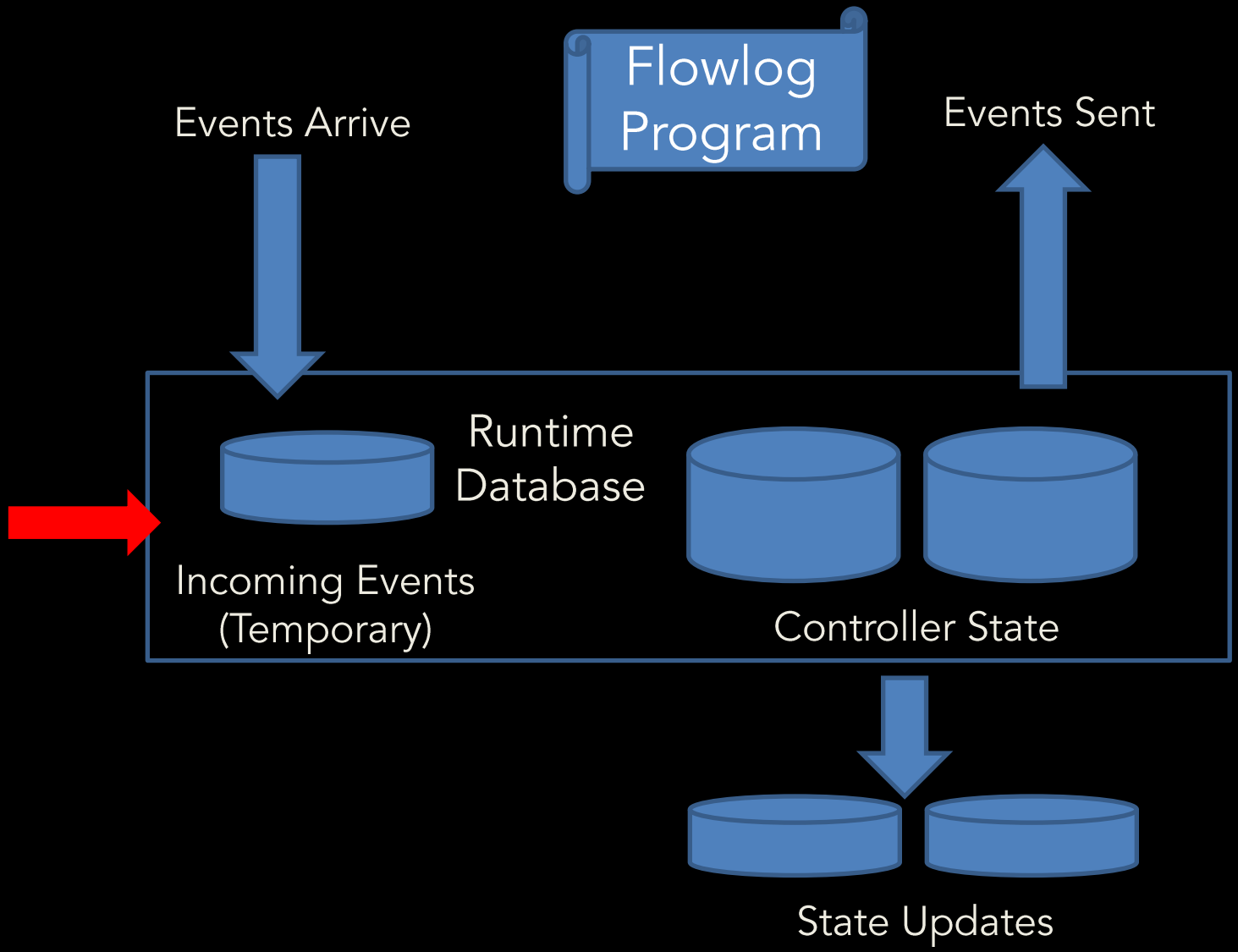
"If packets *from a stolen laptop* appear on an edge switch, the police will be notified."

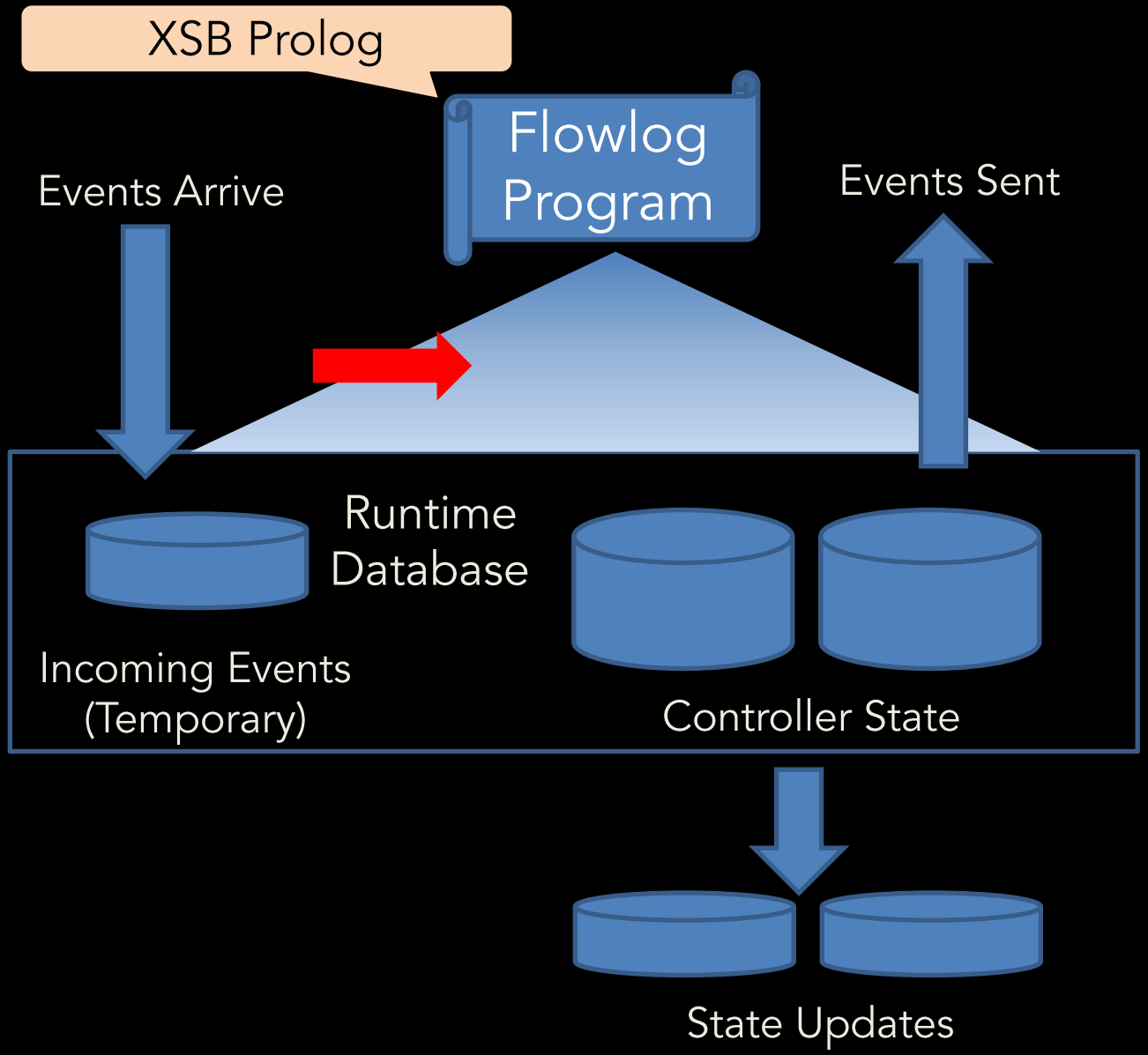
Flowlog as a system

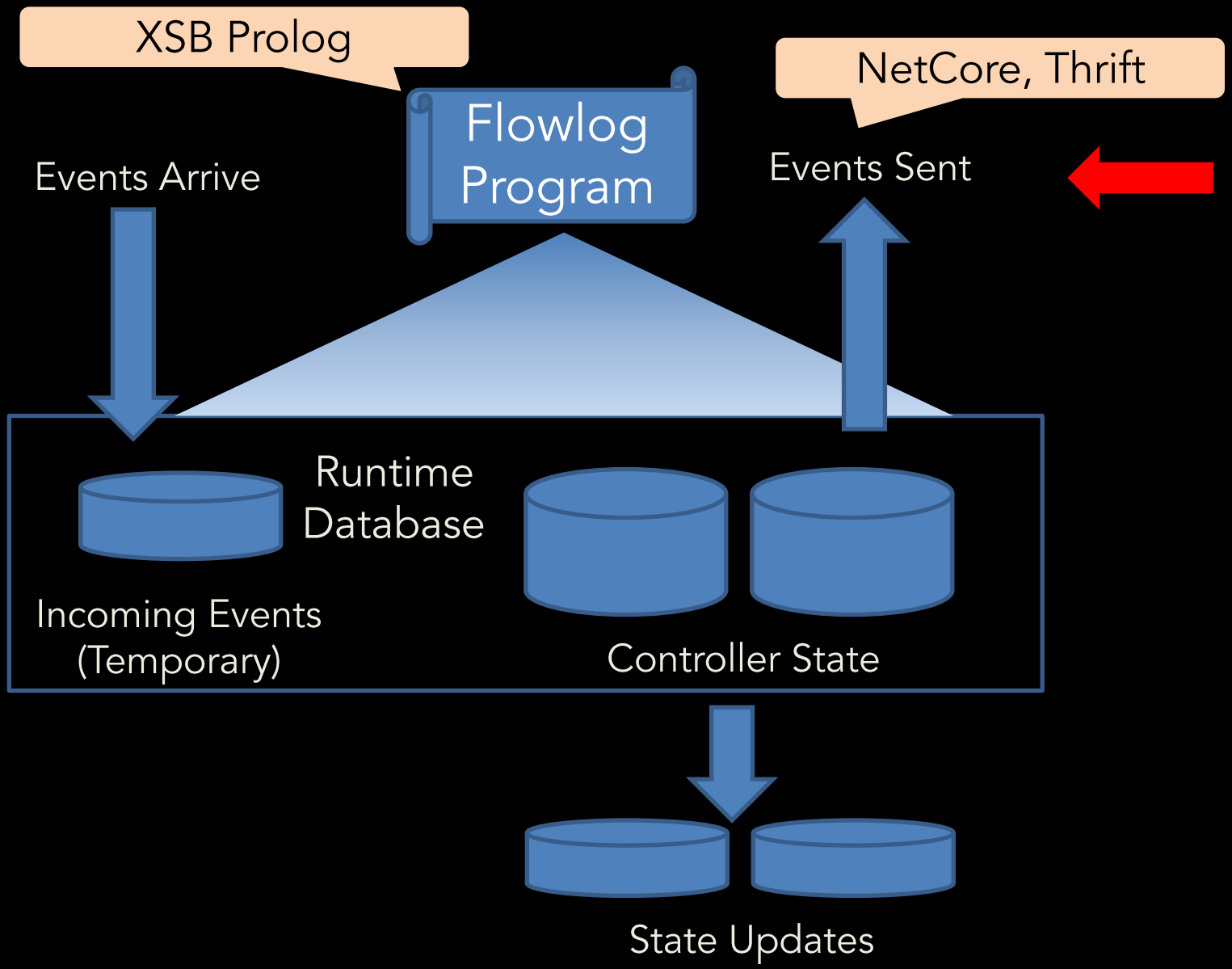
Flowlog Program

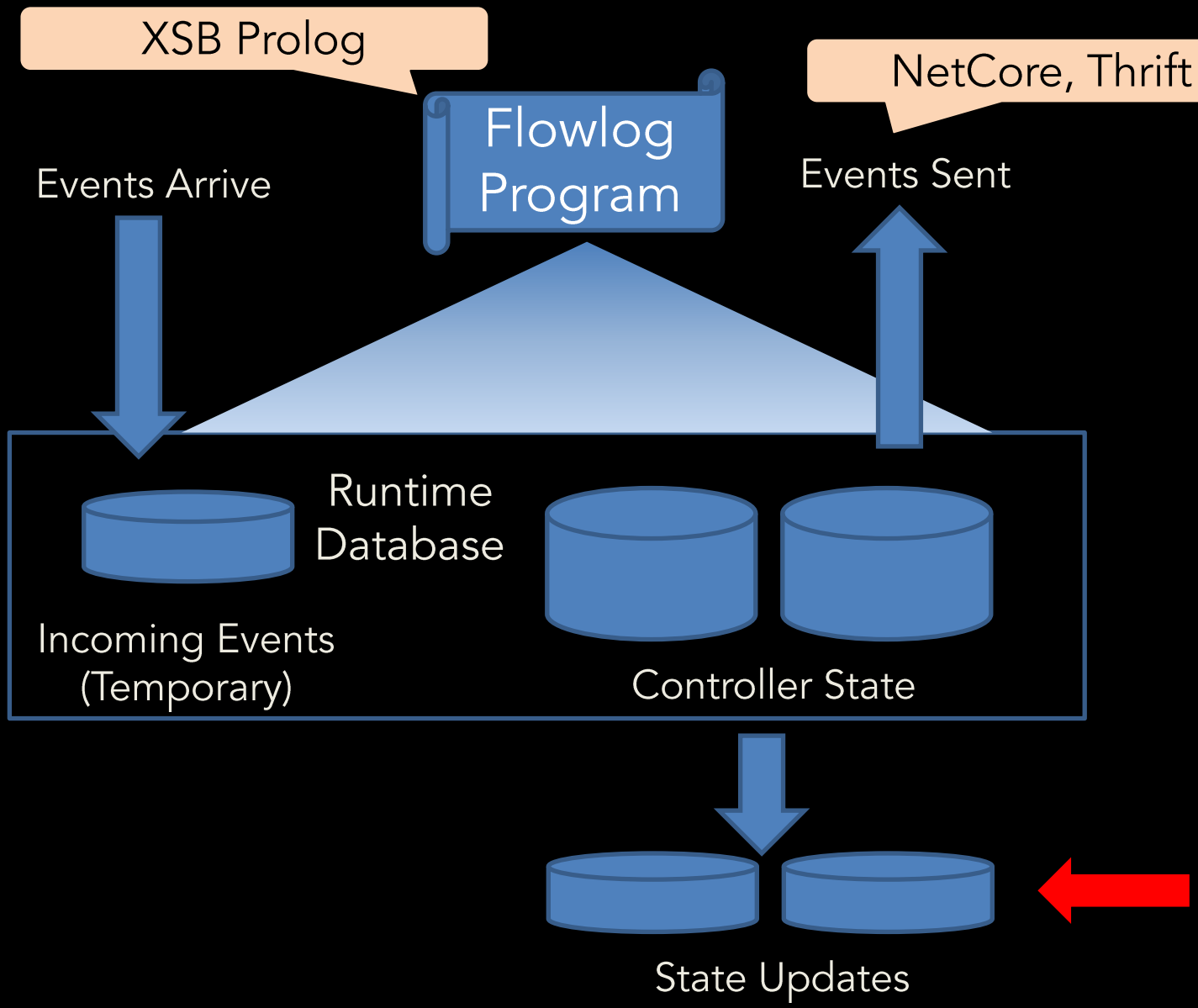

Flowlog Program
Events Arrive
Events Sent
Runtime Database
Incoming Events (Temporary)
Controller State
State Updates

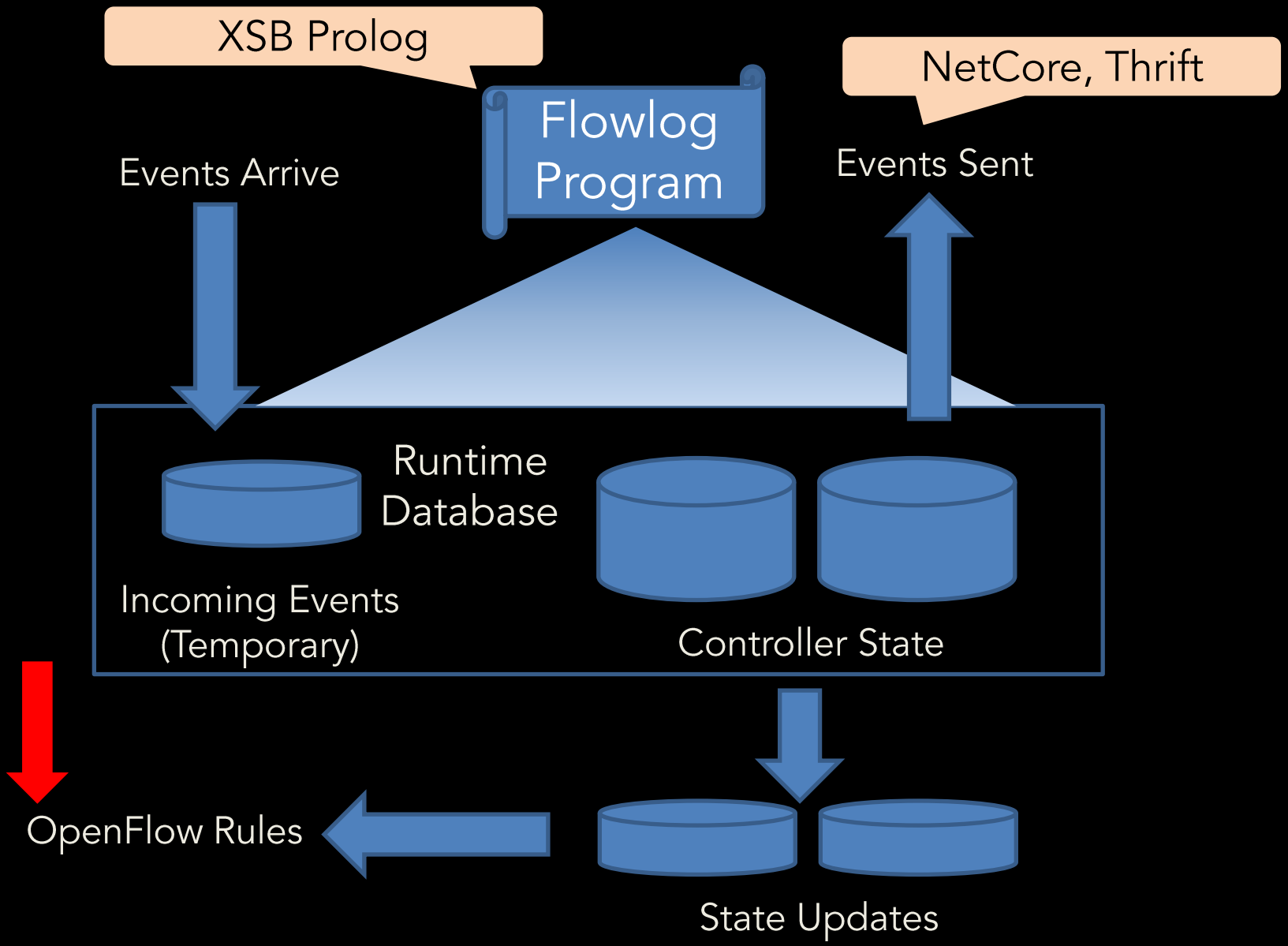

Flowlog Program
Events Arrive
Events Sent
Runtime Database
Incoming Events (Temporary)
Controller State
State Updates


Flowlog Program
Events Arrive
Events Sent
Runtime Database
Incoming Events (Temporary)
Controller State
State Updates


XSB Prolog
Flowlog Program
Events Arrive
Events Sent
Runtime Database
Incoming Events (Temporary)
Controller State
State Updates


XSB Prolog
NetCore, Thrift
Flowlog Program
Events Arrive
Events Sent
Runtime Database
Incoming Events (Temporary)
Controller State
State Updates

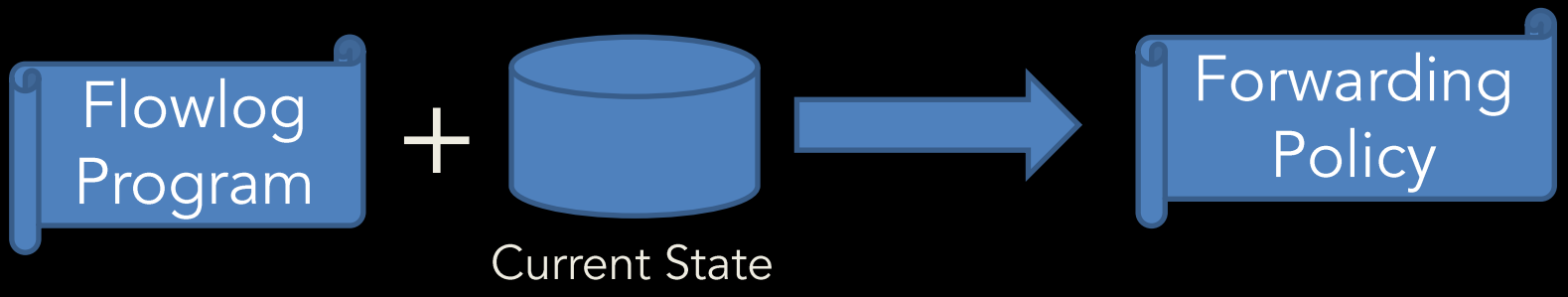

XSB Prolog
Flowlog Program
NetCore, Thrift
Events Arrive
Events Sent
Runtime Database
Incoming Events (Temporary)
Controller State
State Updates


XSB Prolog
NetCore, Thrift
Flowlog Program
Events Arrive
Events Sent
Runtime Database
Incoming Events (Temporary)
Controller State
OpenFlow Rules
State Updates

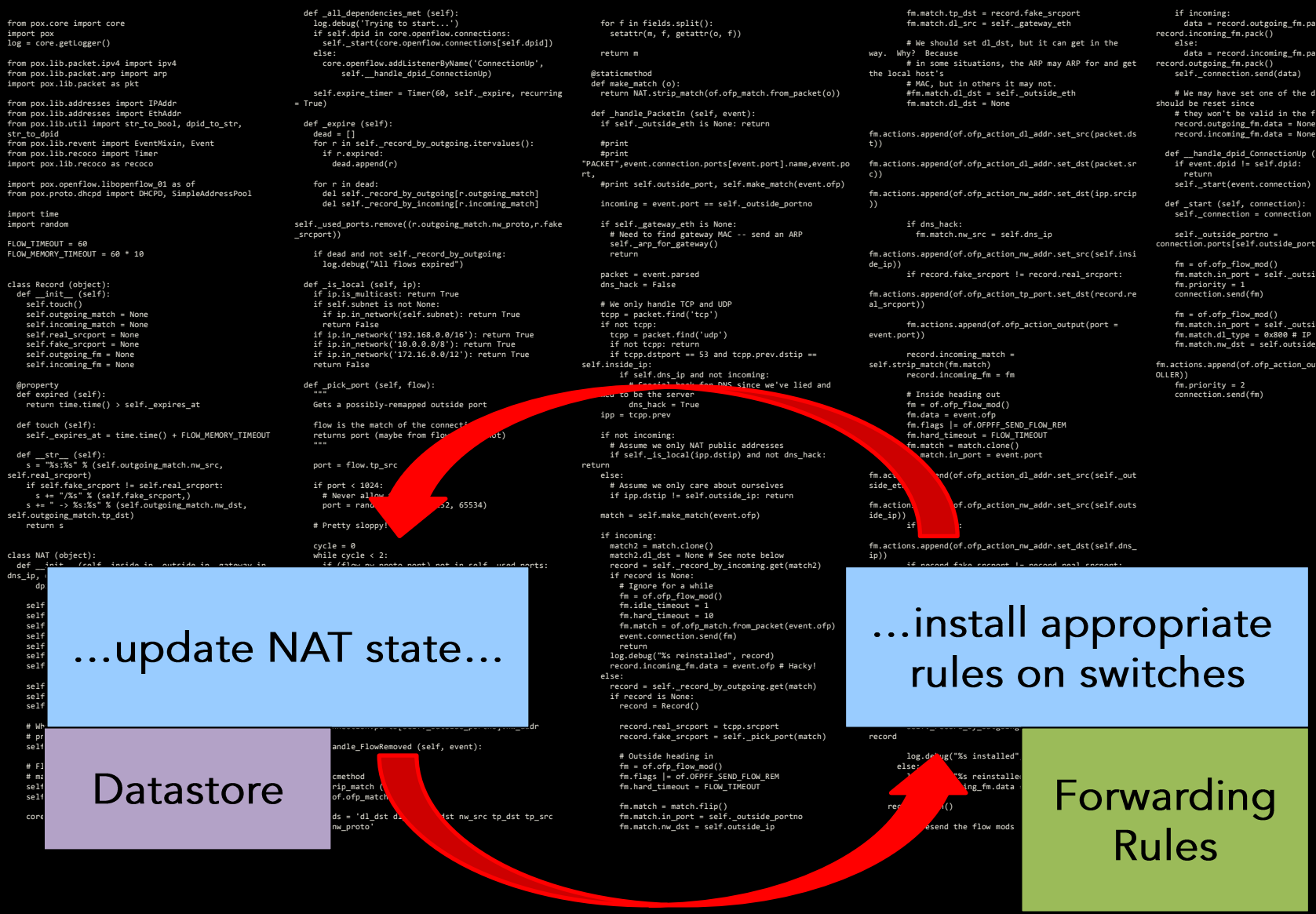
Proactive Compilation (Monsanto, et al. POPL 2012)

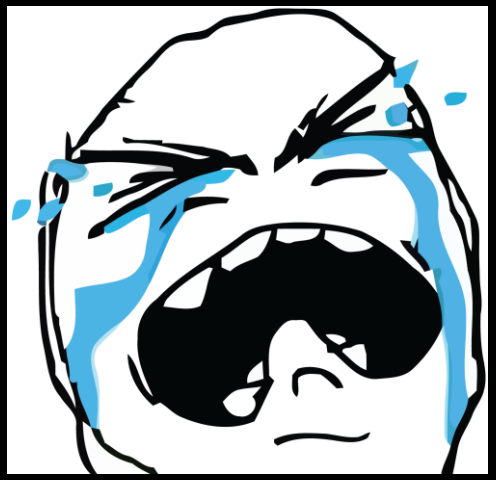
Proactive Compilation (Monsanto, et al. POPL 2012)

Flowlog's Stateful Proactive Compilation



Where we started:

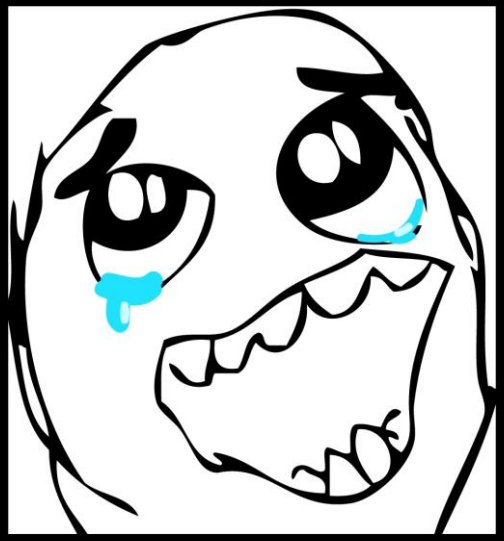




Inconsistent packet-handling between controller and switches

Switch fails to notify controller when state update needed

Switch notifies controller more than necessary



~~Inconsistent packet-handling between controller and switches~~

~~Switch fails to notify controller when state update needed~~

~~Switch notifies controller more than necessary~~

SDN Language Comparison

SDN Language Comparison

Language	Type	State	Rec?	Neg?	Compilation	Reasoning?	Callouts
Flog [11]	Rule-Based	✓	✓	✗	Reactive	✗	✗
FML [9]	Rule-Based	✓	✗	✓	Reactive	✗	✗
Frenetic [5]	FRP	$✓_{pol}$	✗	✓	Reactive	✗	✗
Frenetic OCaml Environment	Functional	$✓_{PL}$	$✓_{PL}$	✓	via NetCore	$✗_{PL}$	$✓_{PL}$
NetCore [21]	DSL	✗	✗	✓	Proactive	✓	✗
Nlog [14]	Rule-Based	✓	✗	✗	Proactive	✗	✓
NOX [6]	Imperative	$✓_{PL}$	$✓_{PL}$	✓	Manual	$✗_{PL}$	$✓_{PL}$
Procera [31]	FRP	✓	✗	✓	Unclear	✗	✗
Pyretic [22]	Imperative	$✓_{pol}$	$✓_{PL}$	✓	Proactive	✗	$✓_{PL}$
Flowlog	Rule-Based	✓	✗	✓	Proactive	✓	✓

Table 4

(See paper for legend and details)

Flowlog

Tierless SDN programming

- Proactive compilation (+ state)
- Built-in cross-tier verification support

Contact: tn@cs.brown.edu

Download the repo:
http://www.tinyurl.com/flowlog