Maple: Simplifying SDN Programming Using Algorithmic Policies

Andreas Voellmy (andreas.voellmy@yale.edu)

Junchang Wang, Y. Richard Yang, Bryan Ford, Paul Hudak
Yale University
Department of Computer Science

Maple: Simplifying SDN Programming Using Algorithmic Policies

Andreas Voellmy (andreas.voellmy@yale.edu)

Junchang Wang, Y. Richard Yang, Bryan Ford, Paul Hudak
Yale University
Department of Computer Science

A Key Source of Complexity in Openflow Controllers

```
onPacketIn(p):
```

A Key Source of Complexity in Openflow Controllers

```
onPacketIn(p):
```

Step 1 `examine p and decide what to do with p.`

A Key Source of Complexity in Openflow Controllers

```
onPacketIn(p):
```

Step 1 `examine p and decide what to do with p.`

Step 2 `construct and install OF rules so that similar packets are processed at switches with same action.`

Simple, generic solution using exact matches

```
onPacketIn(p):
```

Step 1 `examine p and decide what to do with p.`

Step 2 `insert rule with` **“exact match”** `for p, i.e. match on ALL attributes, with action determined above.`

Simple, generic solution using exact matches

```
onPacketIn(p):
```

Step 1 `examine p and decide what to do with p.`

Step 2 `insert rule with` **“exact match”** `for p, i.e. match on ALL attributes, with action determined above.`

Flow table correctness: packet matching in flow table is indistinguishable from the packet that led to the rule being installed.

Simple, generic solution using exact matches

```
onPacketIn(p):
```

Every flow incurs flow setup delay.

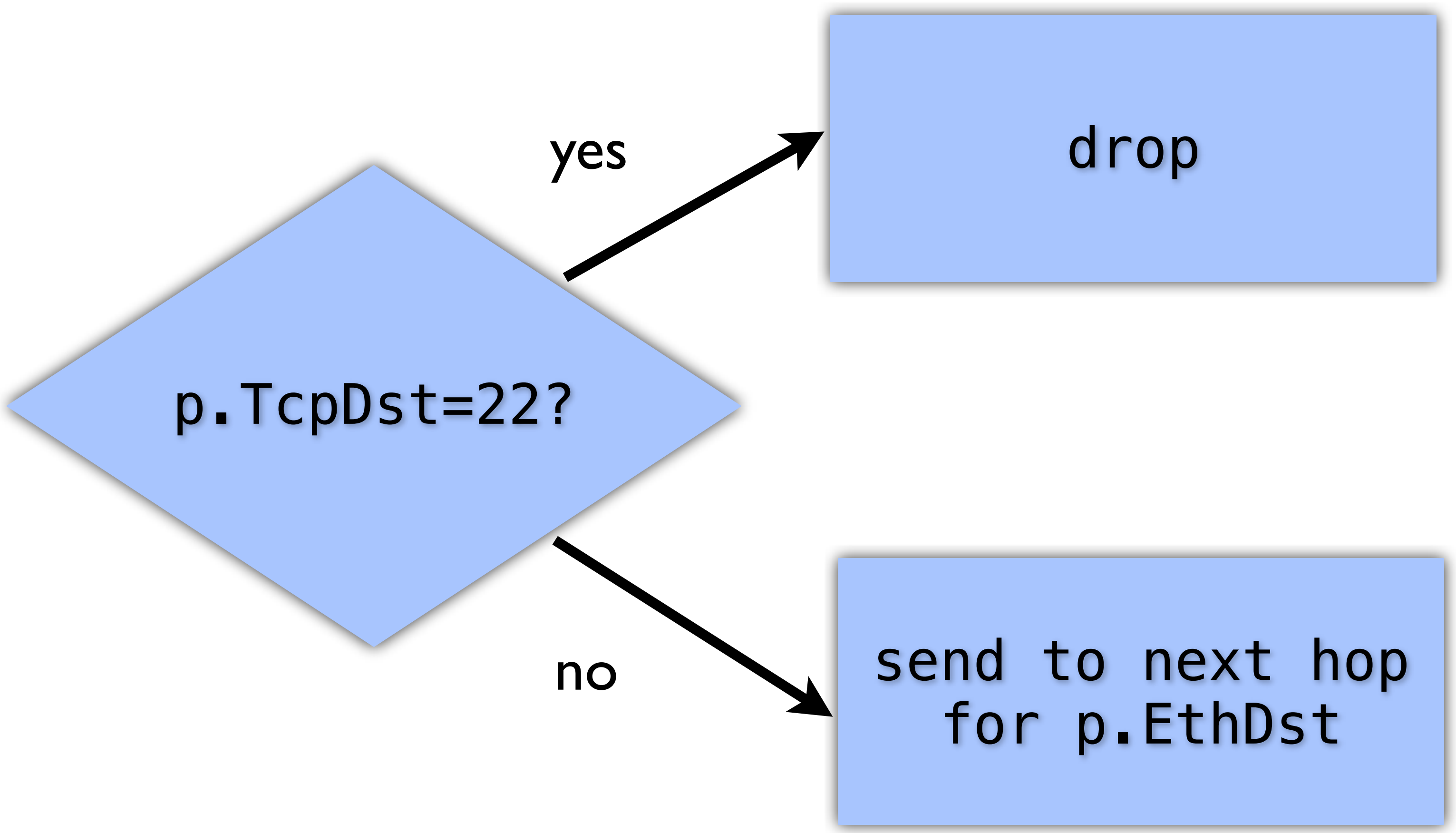
Step 1 `examine p and decide what to do with p.`

Step 2 `insert rule with` **“exact match”** `for p, i.e. match on ALL attributes, with action determined above.`

Flow table correctness: packet matching in flow table is indistinguishable from the packet that led to the rule being installed.

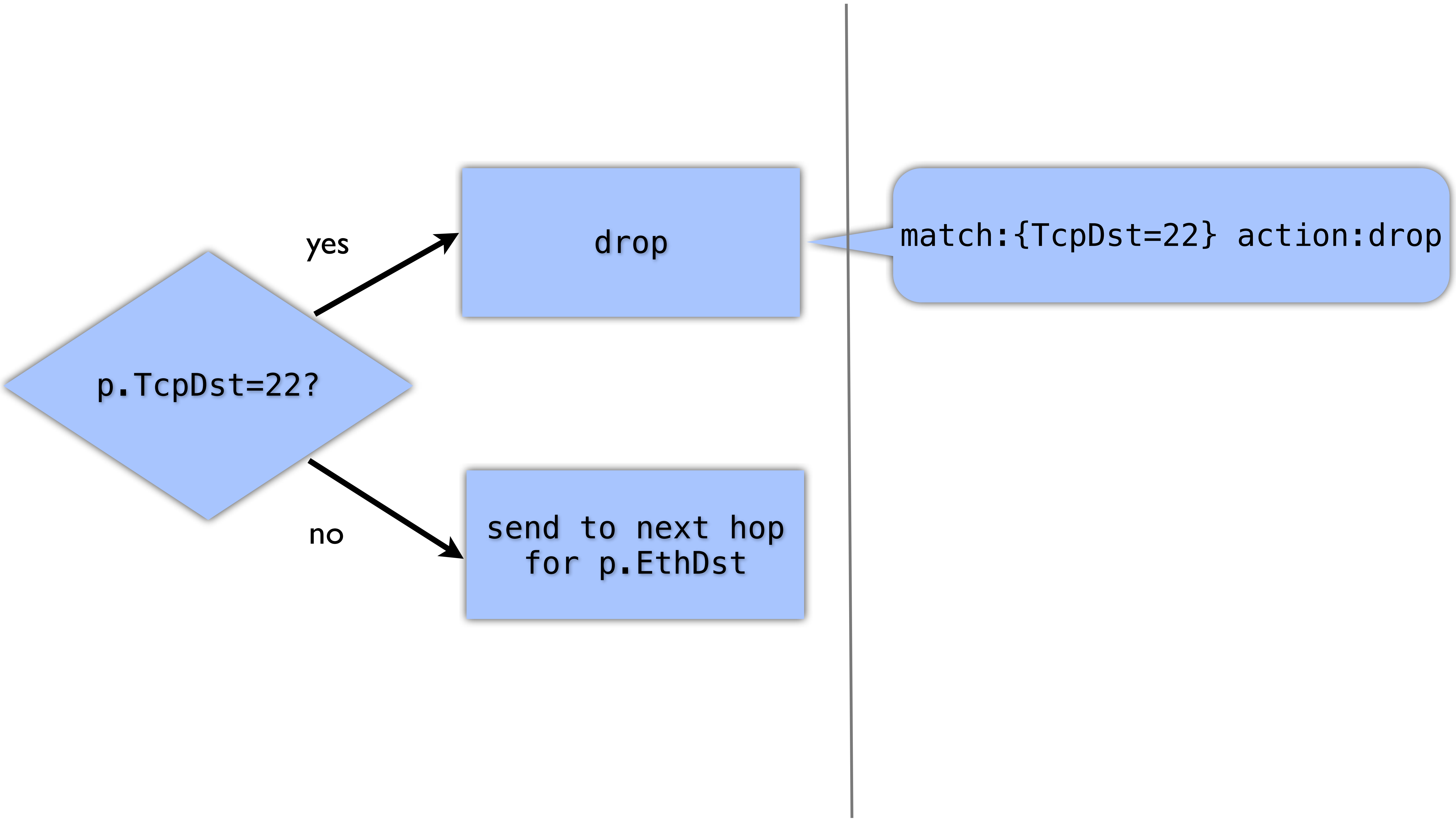
Step 1

Step 2



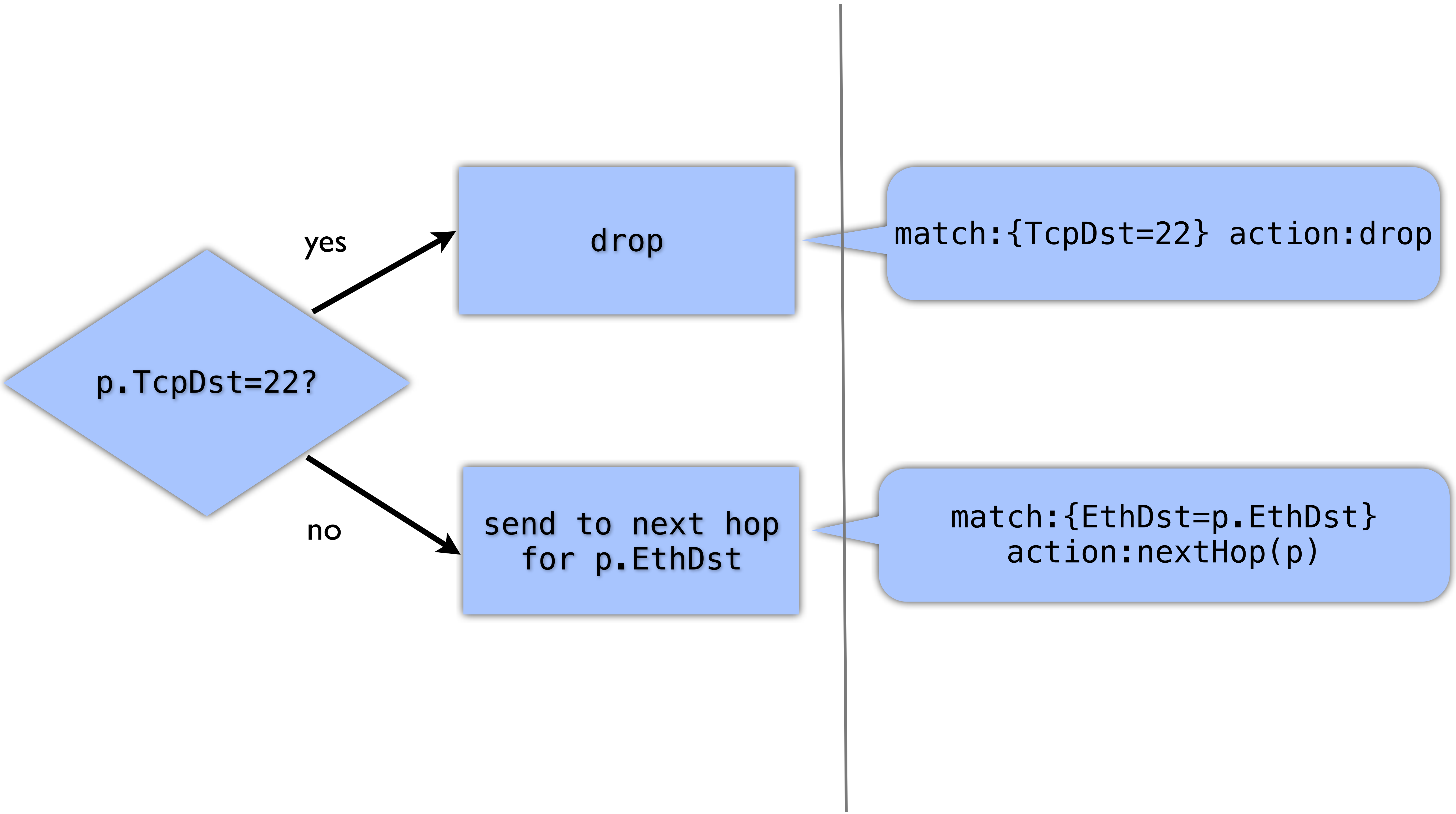
Step 1

Step 2


yes
drop
p.TcpDst=22?
no
send to next hop
for p.EthDst
match:{TcpDst=22} action:drop

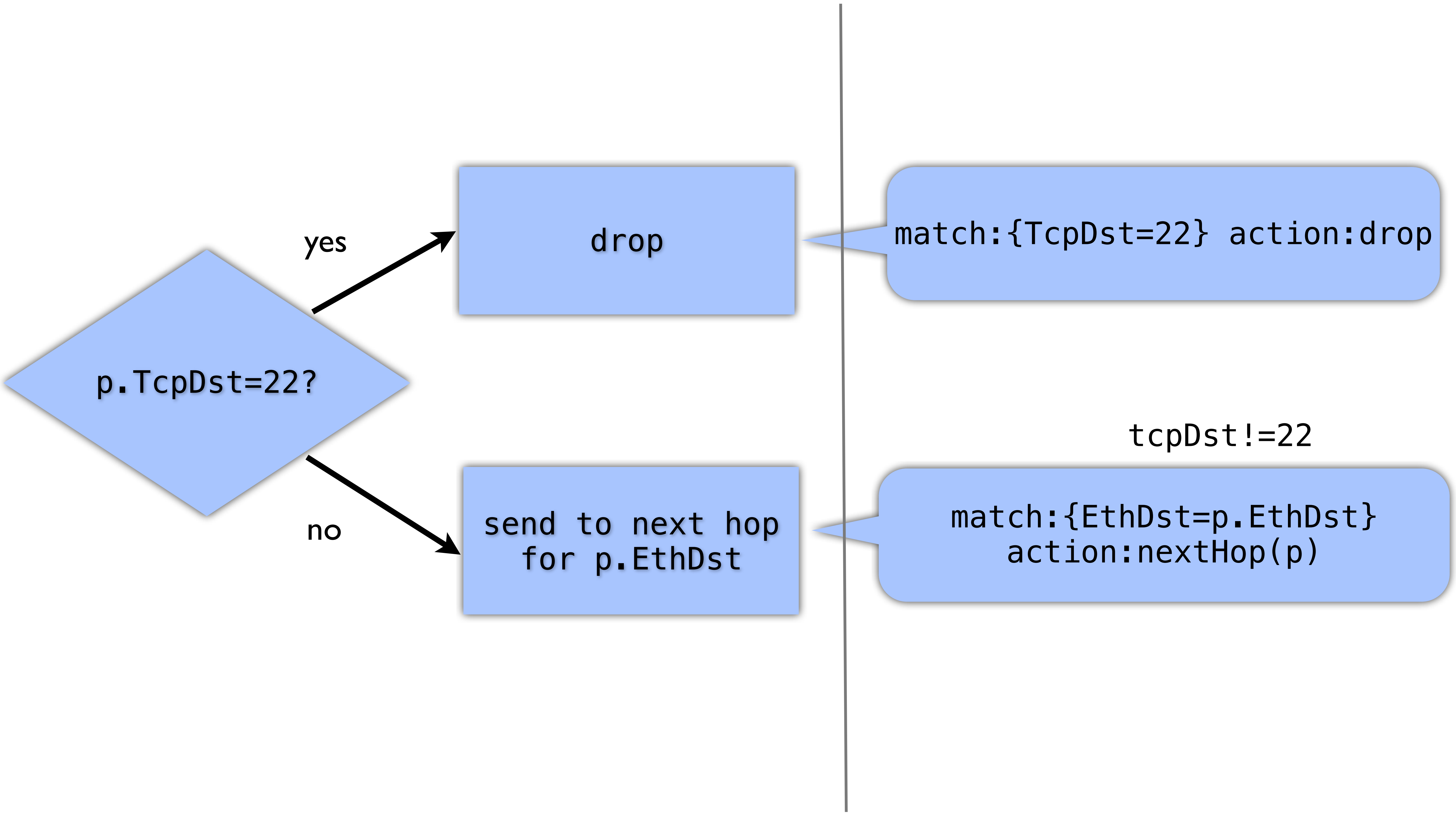
Step 1

Step 2


yes
drop
match:{TcpDst=22} action:drop
p.TcpDst=22?
no
send to next hop
for p.EthDst
match:{EthDst=p.EthDst}
action:nextHop(p)

Step 1

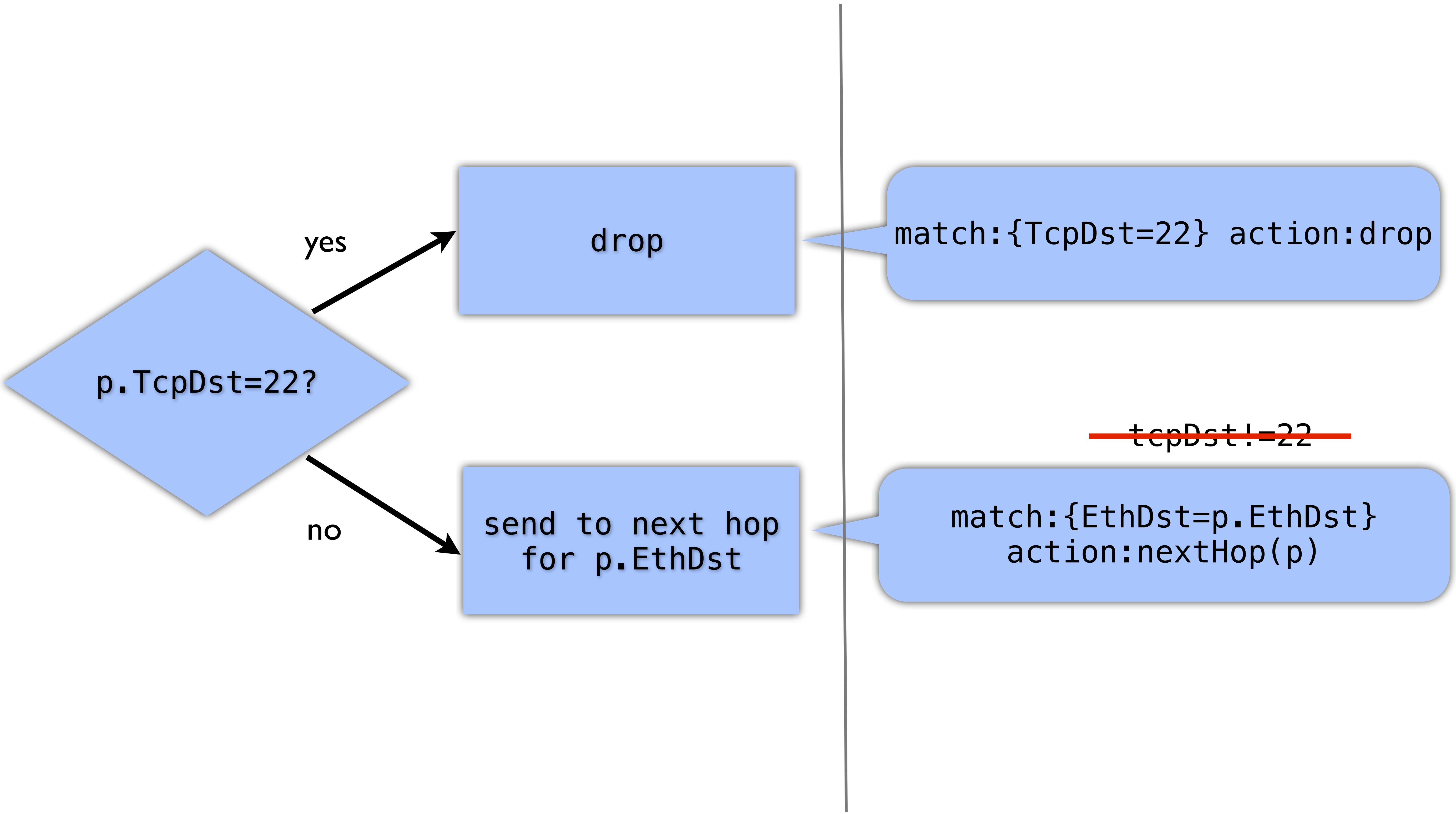
Step 2



p.TcpDst=22?
yes
no
drop
send to next hop for p.EthDst
match:{TcpDst=22} action:drop
tcpDst!=22
match:{EthDst=p.EthDst} action:nextHop(p)

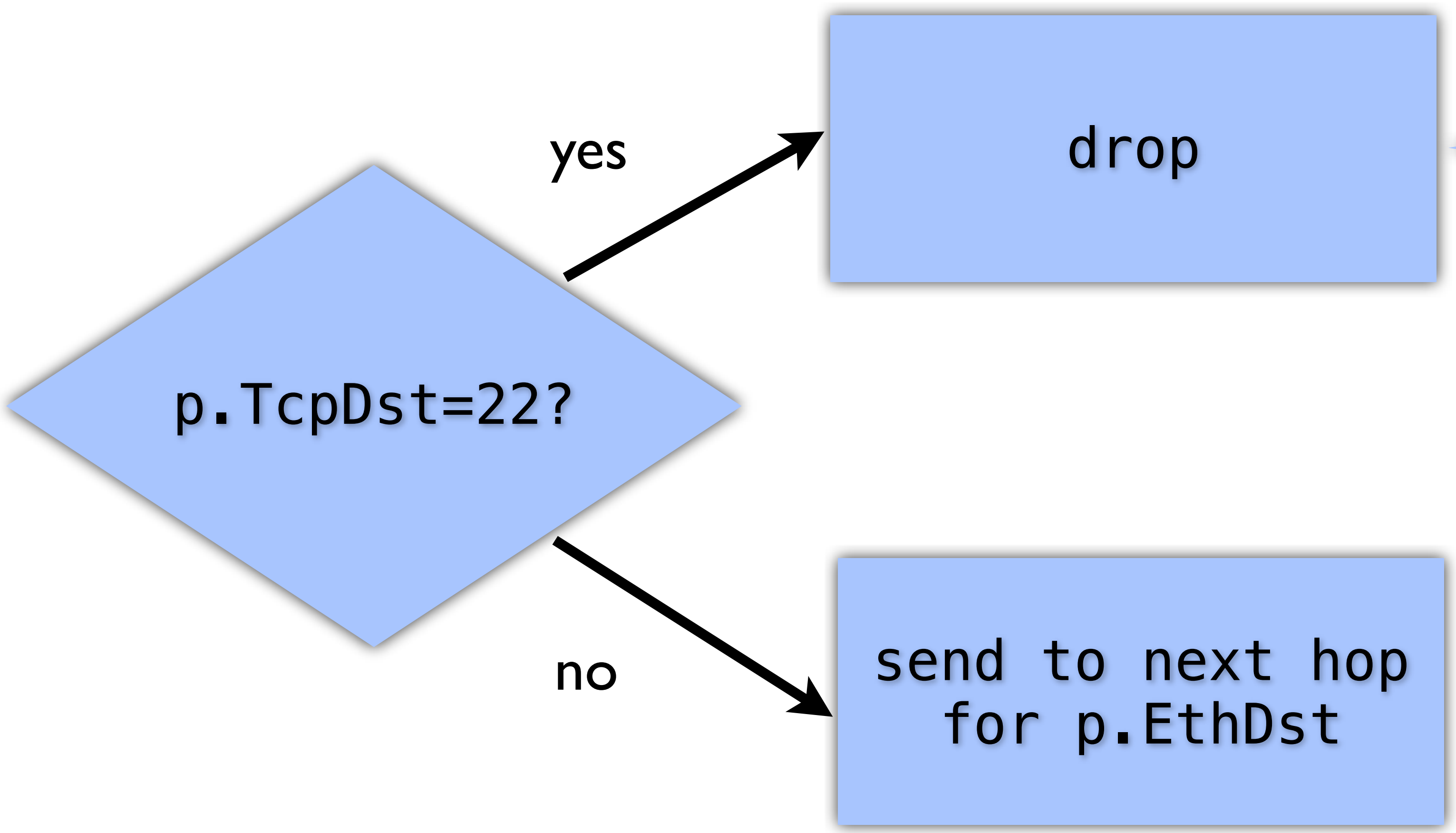
Step 1

Step 2

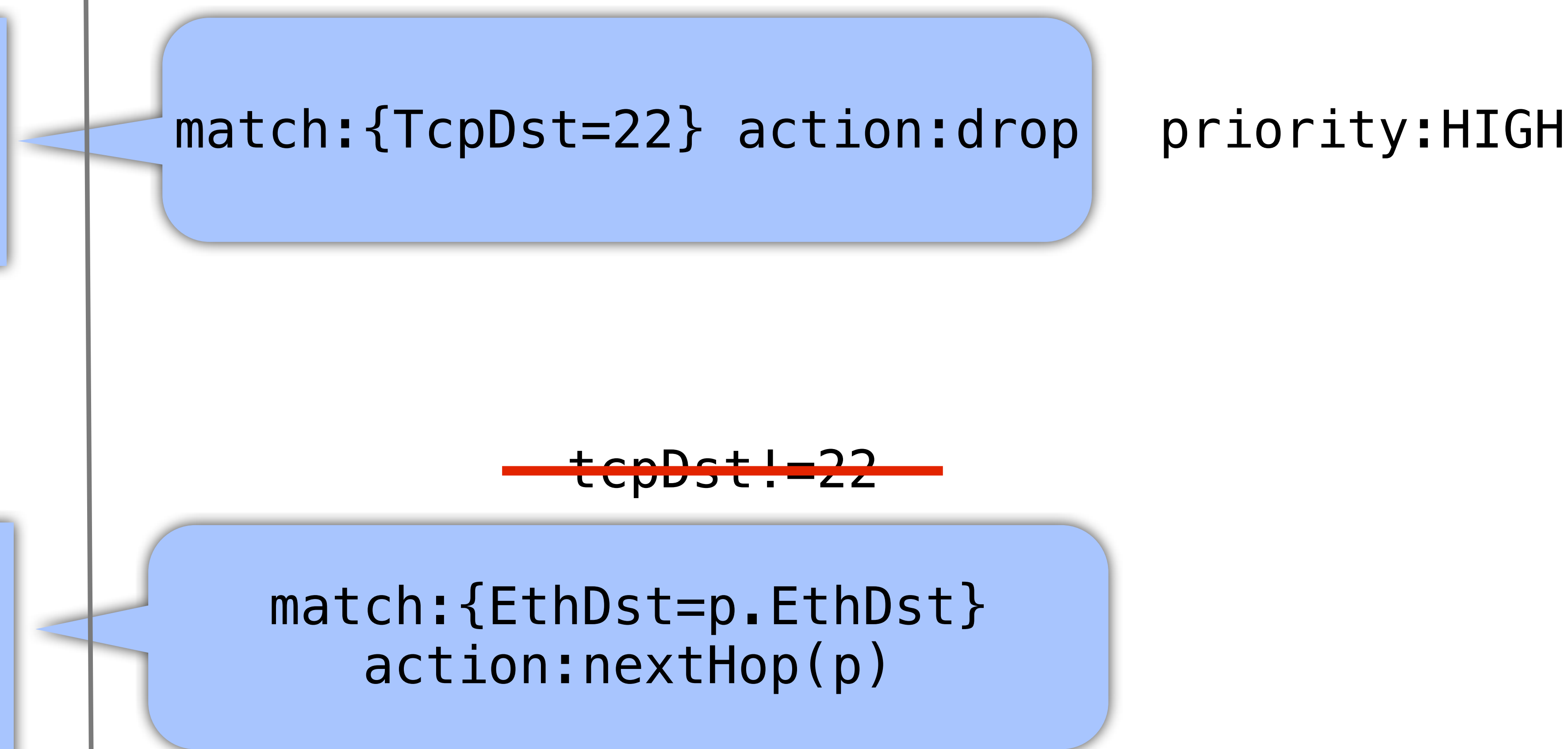


yes
no
p.TcpDst=22?
drop
send to next hop
for p.EthDst
match:{TcpDst=22} action:drop
~~tcpDst!=22~~
match:{EthDst=p.EthDst}
action:nextHop(p)

Step 1

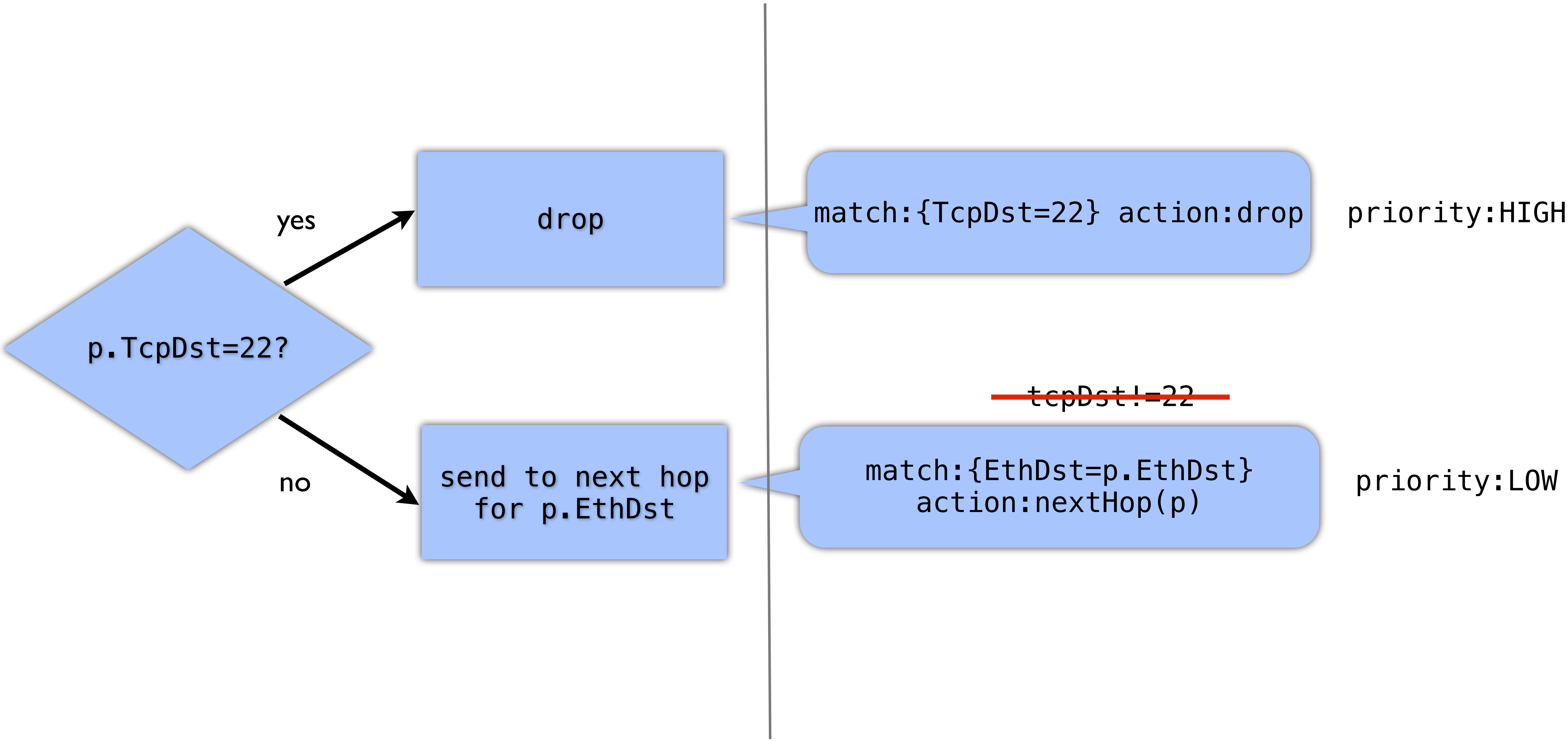

p.TcpDst=22?
yes
drop
no
send to next hop
for p.EthDst

Step 2


match:{TcpDst=22} action:drop
priority:HIGH
tcpDst!=22
match:{EthDst=p.EthDst}
action:nextHop(p)

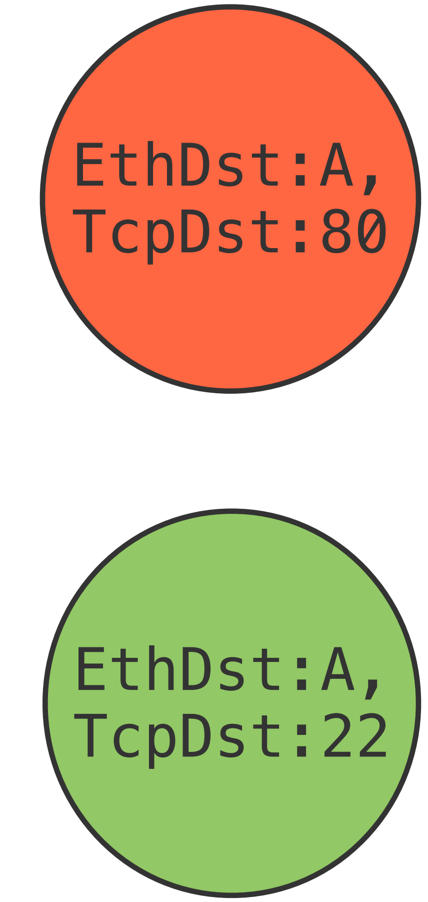
Step 1

Step 2



p.TcpDst=22?
yes
no
drop
send to next hop
for p.EthDst
match:{TcpDst=22} action:drop
priority:HIGH
tcpDst!=22
match:{EthDst=p.EthDst}
action:nextHop(p)
priority:LOW

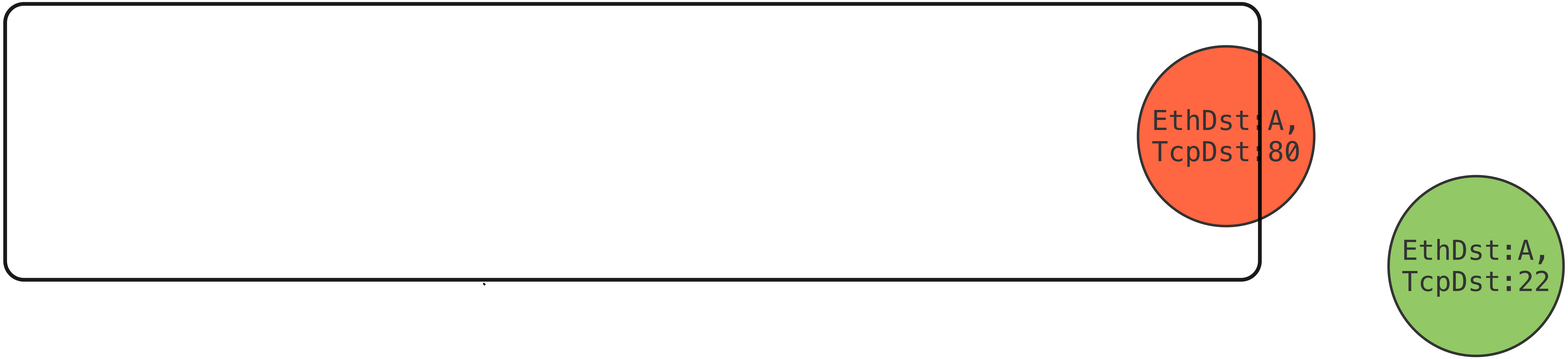
Switch



Controller

```
If  p.TcpDst=22:
  insert rule
   {prio:HIGH, match:{TcpDst=22}, action:drop }
Else:
  insert rule
   {prio:LOW, match:{EthDst=p.EthDst},action:nextHop(p.EthDst)
```

Switch



Controller

```
If  p.TcpDst=22:
  insert rule
   {prio:HIGH, match:{TcpDst=22}, action:drop }
Else:
  insert rule
   {prio:LOW, match:{EthDst=p.EthDst},action:nextHop(p.EthDst)
```

Switch

EthDst:A,
TcpDst:22

Controller

```
If  p.TcpDst=22:
  insert rule
   {prio:HIGH, match:{TcpDst=22}, action:drop }
Else:
  insert rule
   {prio:LOW, match:{EthDst=p.EthDst},action:nextHop(p.EthDst)}
```

EthDst:A,
TcpDst:80

Switch

Low	EthDst:A	Port 1

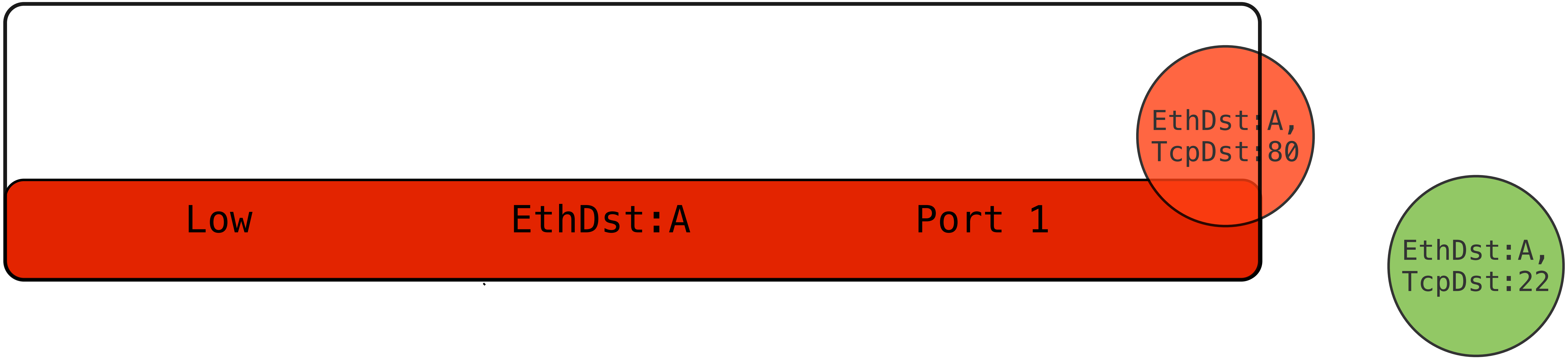
EthDst:A,
TcpDst:22

Controller

```
If  p.TcpDst=22:
  insert rule
   {prio:HIGH, match:{TcpDst=22}, action:drop }
Else:
  insert rule
   {prio:LOW, match:{EthDst=p.EthDst},action:nextHop(p.EthDst)
```

EthDst:A,
TcpDst:80

Switch



Controller

```
If  p.TcpDst=22:
  insert rule
   {prio:HIGH, match:{TcpDst=22}, action:drop }
Else:
  insert rule
   {prio:LOW, match:{EthDst=p.EthDst},action:nextHop(p.EthDst)
```

Switch

TcpDst:80

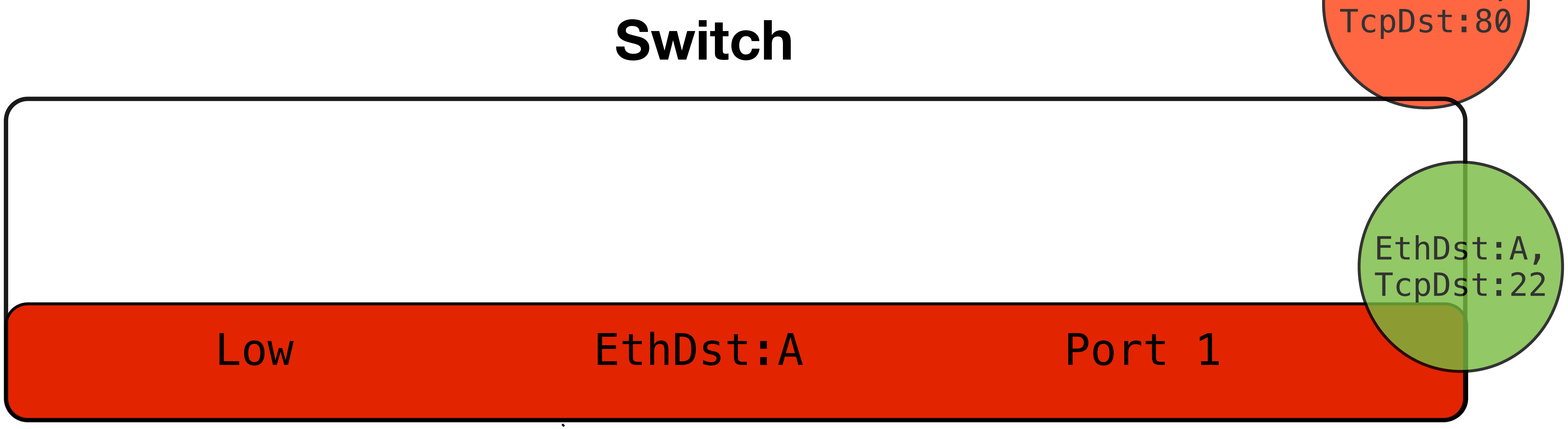
Low	EthDst:A	Port 1

EthDst:A,
TcpDst:22

Controller

```
If  p.TcpDst=22:
  insert rule
   {prio:HIGH, match:{TcpDst=22}, action:drop }
Else:
  insert rule
   {prio:LOW, match:{EthDst=p.EthDst},action:nextHop(p.EthDst)
```

Switch



Controller

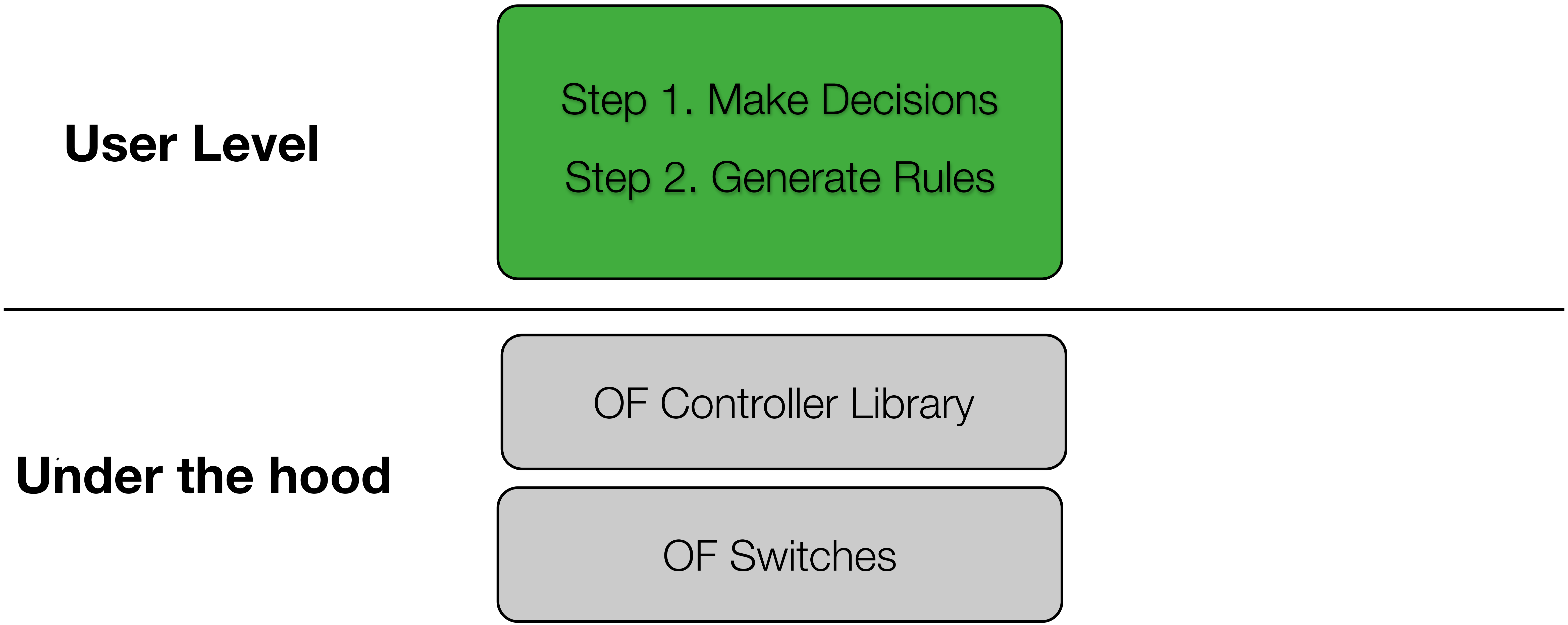
```
If  p.TcpDst=22:
  insert rule
   {prio:HIGH, match:{TcpDst=22}, action:drop }
Else:
  insert rule
   {prio:LOW, match:{EthDst=p.EthDst},action:nextHop(p.EthDst)
```

Switch

Low	EthDst:A	Port 1

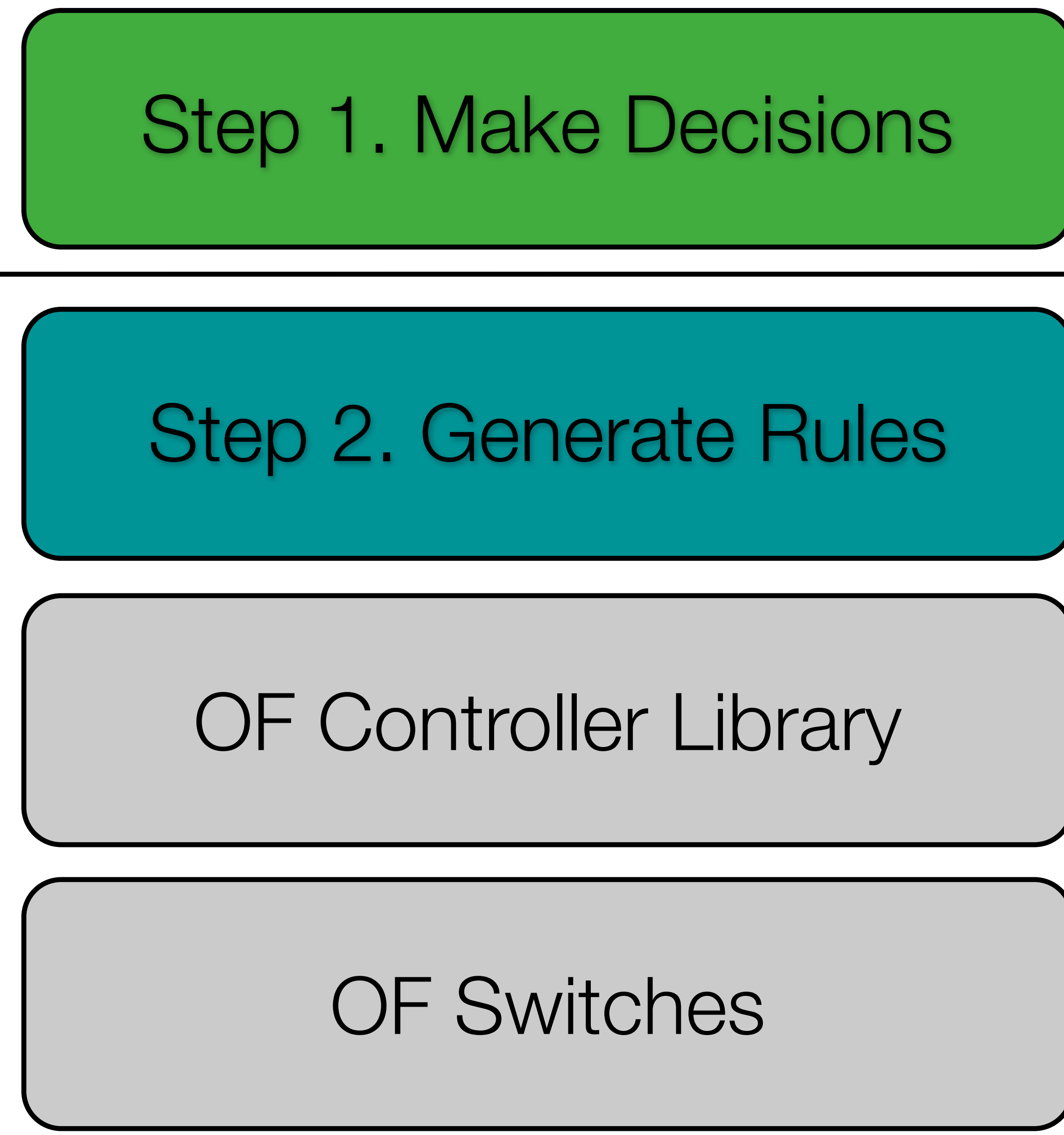
Controller

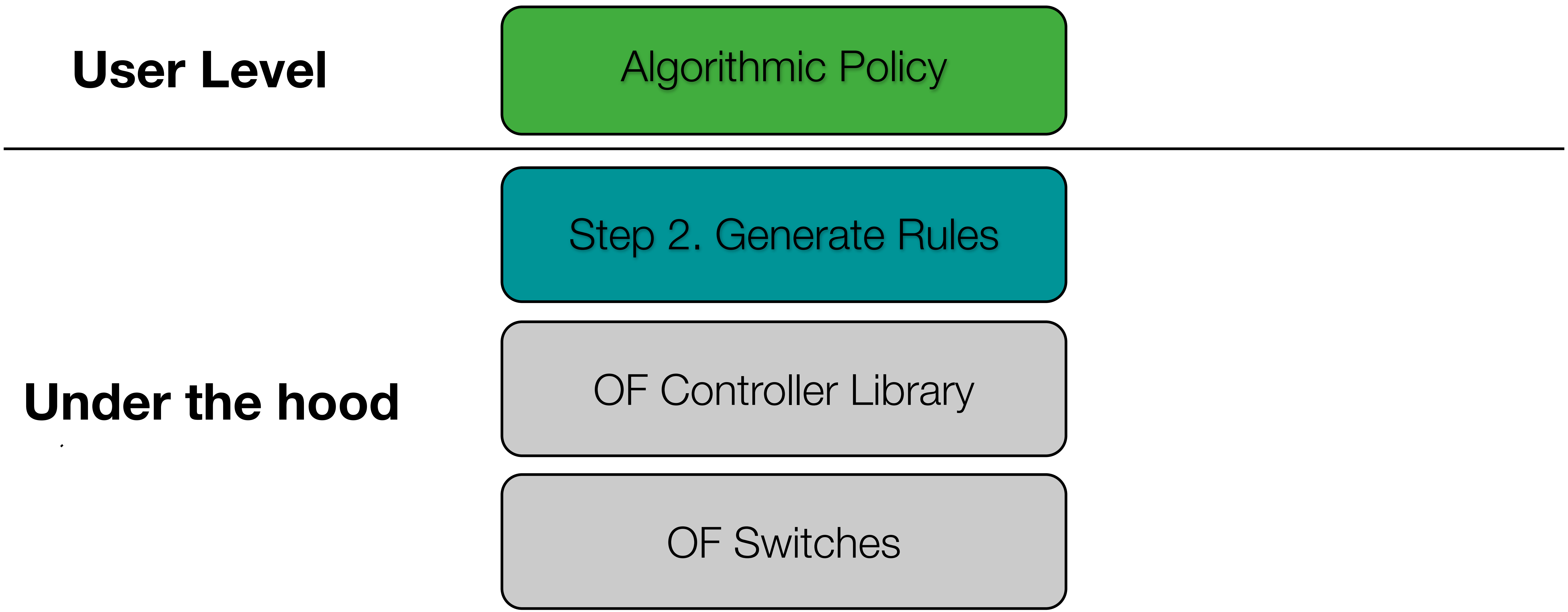
```
If  p.TcpDst=22:
  insert rule
   {prio:HIGH, match:{TcpDst=22}, action:drop }
Else:
  insert rule
   {prio:LOW, match:{EthDst=p.EthDst},action:nextHop(p.EthDst)
```


User Level
Step 1. Make Decisions
Step 2. Generate Rules
Under the hood
OF Controller Library
OF Switches

User Level

Under the hood




User Level
Algorithmic Policy
Step 2. Generate Rules
Under the hood
OF Controller Library
OF Switches

Algorithmic Policies

Algorithmic Policies

- Function in a **general purpose language** that describes how a packet should be routed, **not** how flow tables are configured.

Algorithmic Policies

- Function in a **general purpose language** that describes how a packet should be routed, **not** how flow tables are configured.

- **Conceptually invoked on every packet entering the network**; may also access network environment state; hence it has the form:

Algorithmic Policies

- Function in a **general purpose language** that describes how a packet should be routed, **not** how flow tables are configured.

- **Conceptually invoked on every packet entering the network**; may also access network environment state; hence it has the form:

$$f : (packet \times env) \rightarrow route$$

Algorithmic Policies

- Function in a **general purpose language** that describes how a packet should be routed, **not** how flow tables are configured.

- **Conceptually invoked on every packet entering the network**; may also access network environment state; hence it has the form:

$$f : (packet \times env) \rightarrow route$$

- Written in a familiar language such as Java, Python, or Haskell.

Example Algorithmic Policy in Java

Example Algorithmic Policy in Java

```
Route f(Packet p, Env e) {
  if (p.tcpDstIs(22))
    return null();
  else {
    Location sloc = e.location(p.ethSrc());
    Location dloc = e.location(p.ethDst());
    Path path = shortestPath(e.links(), sloc,dloc);
    return unicast(sloc,dloc,path);
  }
}
```

Example Algorithmic Policy in Java

```
Route f(Packet p, Env e) {
  if (p.tcpDstIs(22))
    return null();
  else {
    Location sloc = e.location(p.ethSrc());
    Location dloc = e.location(p.ethDst());
    Path path = shortestPath(e.links(), sloc,dloc);
    return unicast(sloc,dloc,path);
  }
}
```

Does not specify flow table configutation

How to implement algorithmic policies?

How to implement algorithmic policies?

- Naive solutions -- process every packet at controller or use only exact match rules -- perform poorly.

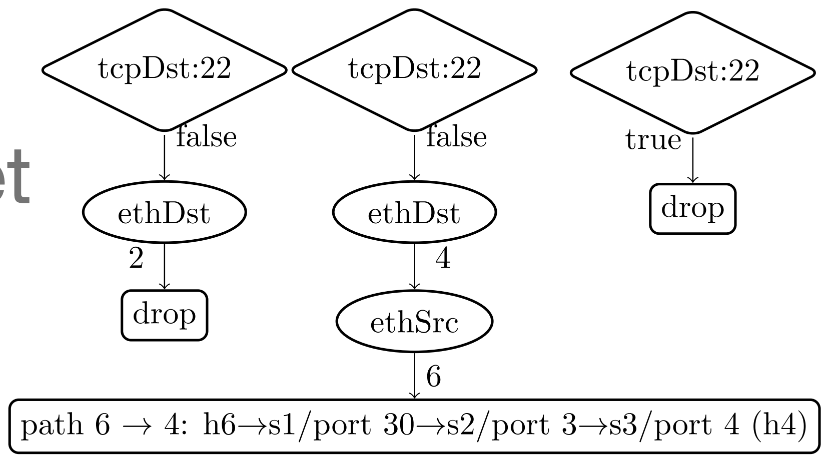
How to implement algorithmic policies?

- Naive solutions -- process every packet at controller or use only exact match rules -- perform poorly.
- Static analysis to determine layout of flow tables is possible, but has drawbacks:
  - Static analysis of program in general-purpose language is hard and is typically conservative.
  - System becomes source-language dependent.

Maple's approach: runtime tracing

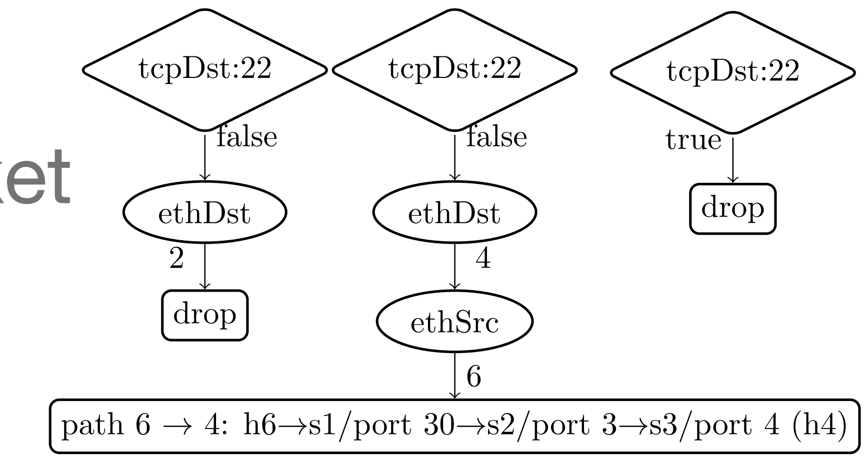
Maple's approach: runtime tracing

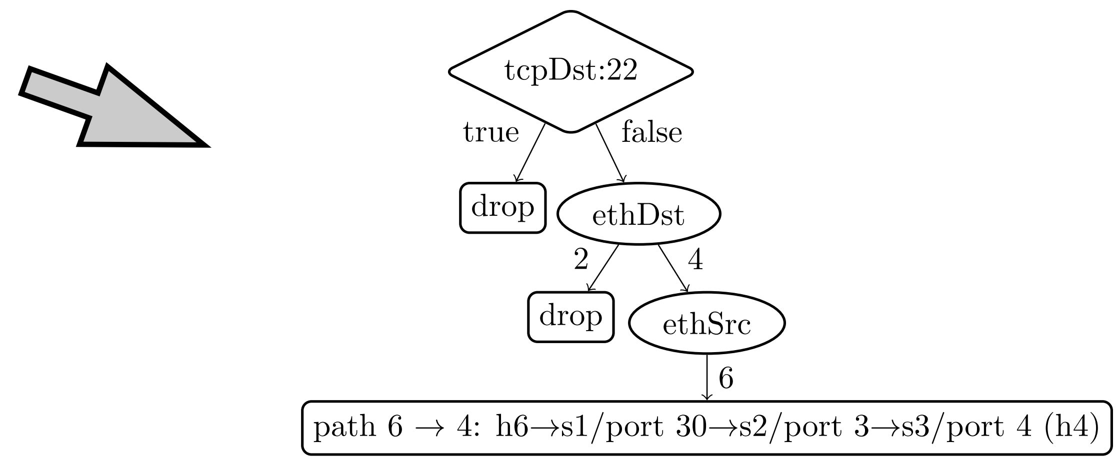
1. Maple **observes the dependency** of `f` on packet data.



Maple's approach: runtime tracing

1. Maple **observes the dependency** of `f` on packet data.

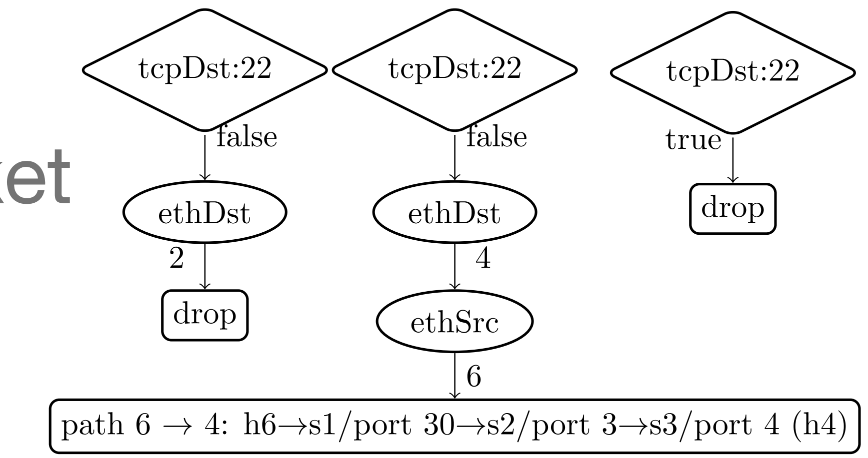


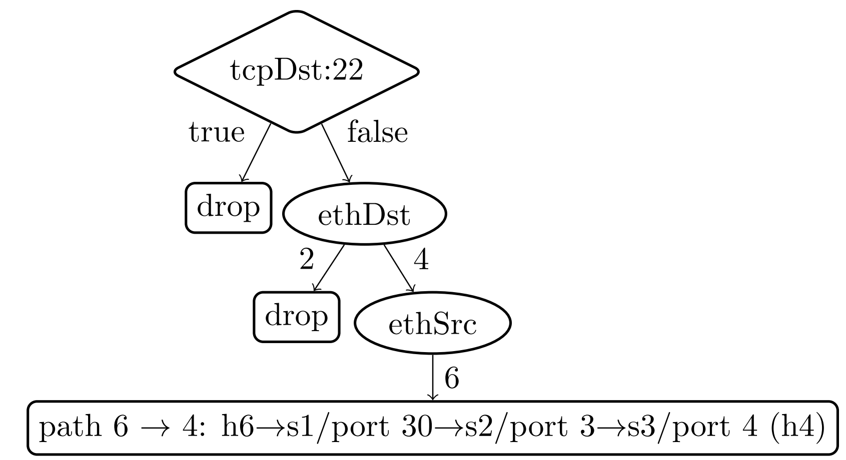


2. Build a **trace tree (TT)**, a partial decision tree for f.

Maple's approach: runtime tracing

1. Maple **observes the dependency** of `f` on packet data.





2. Build a **trace tree (TT)**, a partial decision tree for f.

3. Compile flow tables (FTs) from a trace tree.

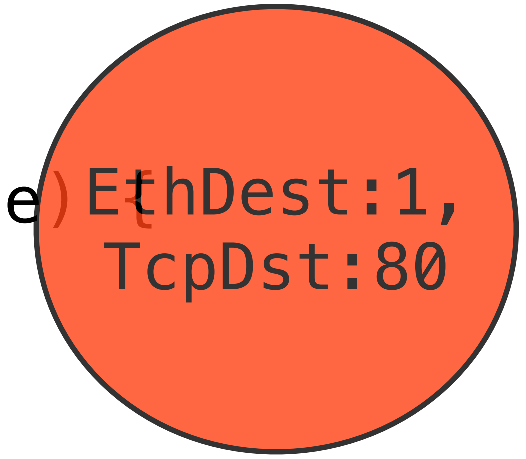
Prio	Match	Action
1	tcpDst:22	ToController
0	ethDst:2	discard
0	ethDst:4, ethSrc:6	port 30

Policy

EthDest:1,
TcpDst:80

```
Route f(Packet p, Env e) {

  if (p.tcpDstIs(22))


    return null();

  else {

    Location sloc =
      e.location(p.ethSrc());

    Location dloc =
      e.location(p.ethDst());

    Path path     =
      shortestPath(
        e.links(),sloc,dloc);

    return
      unicast(sloc,dloc,path);
  }
}
```

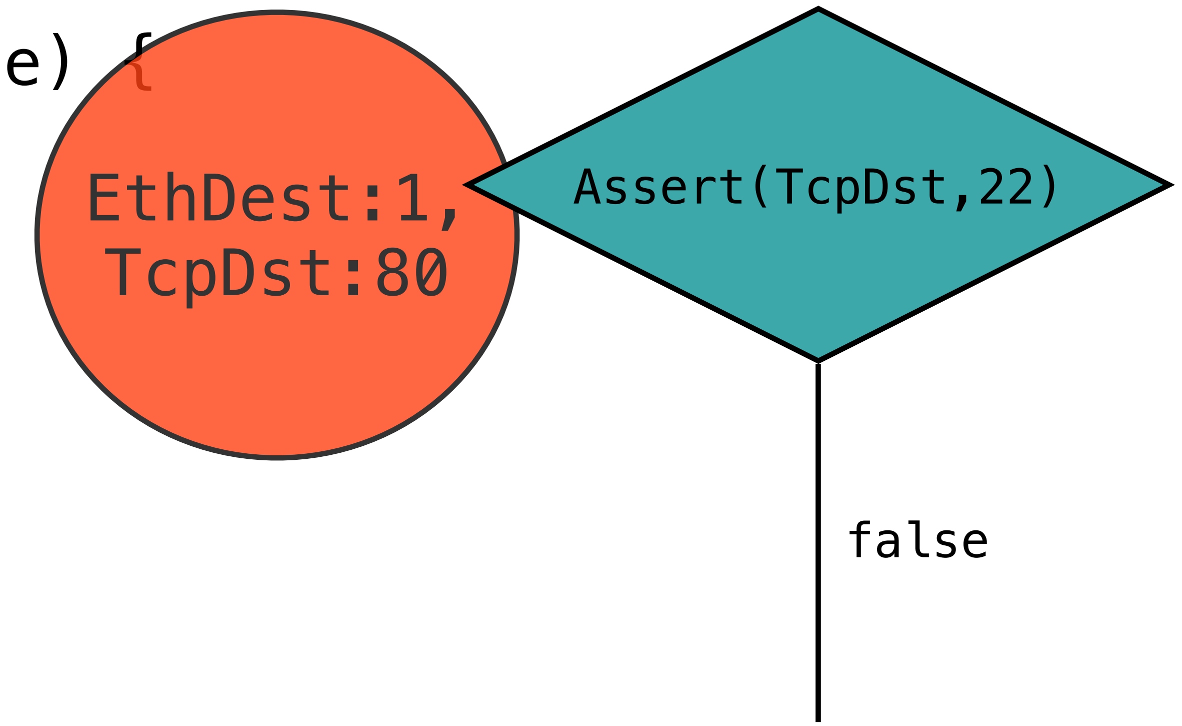
Policy




```
Route f(Packet p, Env e) {
  if (p.tcpDstIs(22))


    return null();

  else {

    Location sloc =
      e.location(p.ethSrc());

    Location dloc =
      e.location(p.ethDst());

    Path path     =
      shortestPath(
        e.links(),sloc,dloc);

    return
      unicast(sloc,dloc,path);
  }
}
```

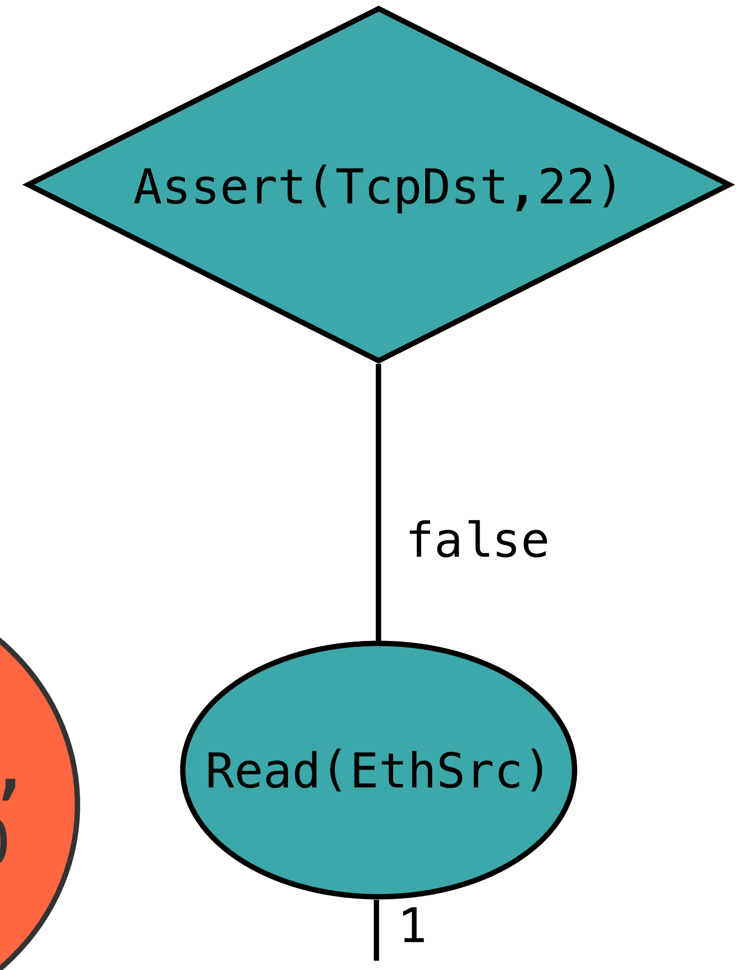
Policy

```
Route f(Packet p, Env e) {

  if (p.tcpDstIs(22))

    return null();

  else {

    Location sloc =
      e.location(p.ethSrc());

    Location dloc =
      e.location(p.ethDst());

    Path path    =
      shortestPath(
        e.links(),sloc,dloc);

    return
      unicast(sloc,dloc,path);
  }
}
```



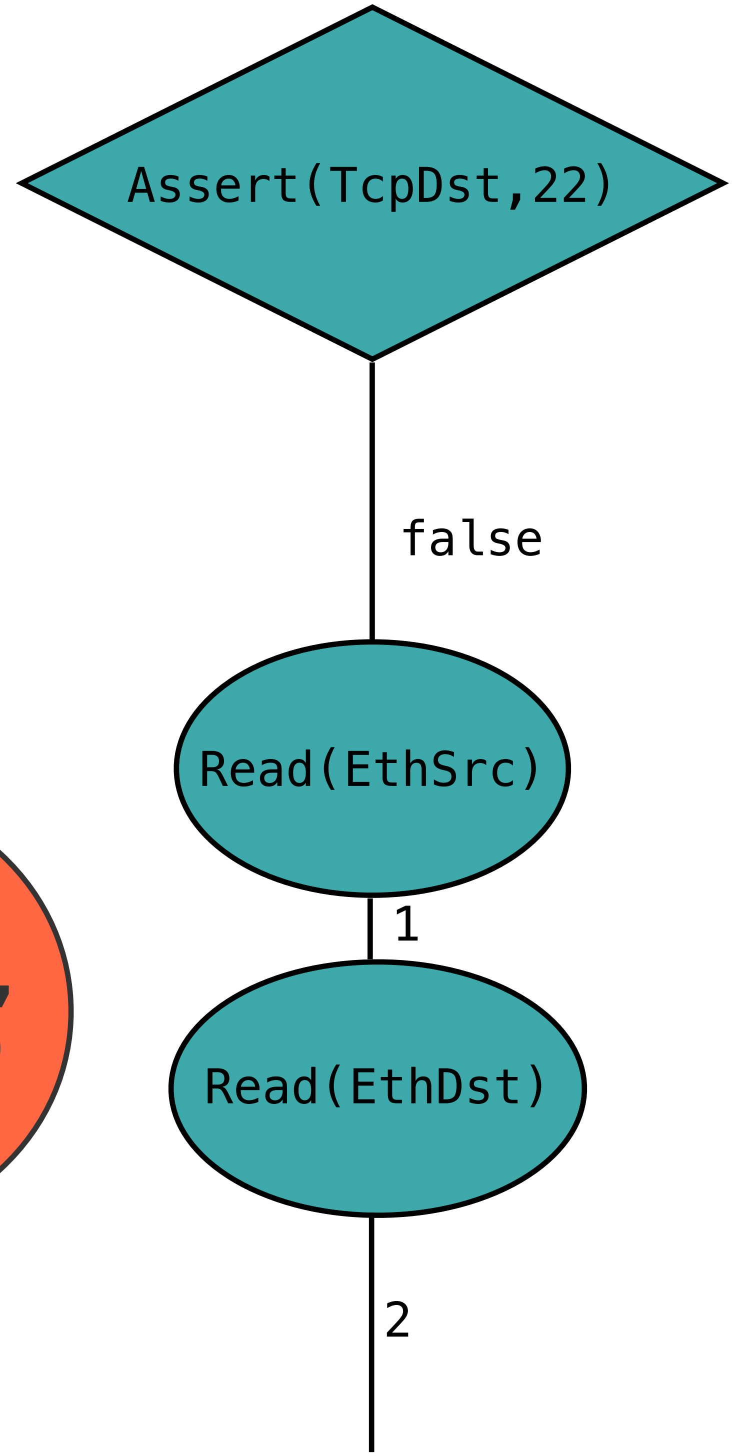
Policy

```
Route f(Packet p, Env e) {

  if (p.tcpDstIs(22))


    return null();

  else {

    Location sloc =
      e.location(p.ethSrc());

    Location dloc =
      e.location(p.ethDst());

    Path path     =
      shortestPath(
        e.links(),sloc,dloc);

    return
      unicast(sloc,dloc,path);
  }
}
```



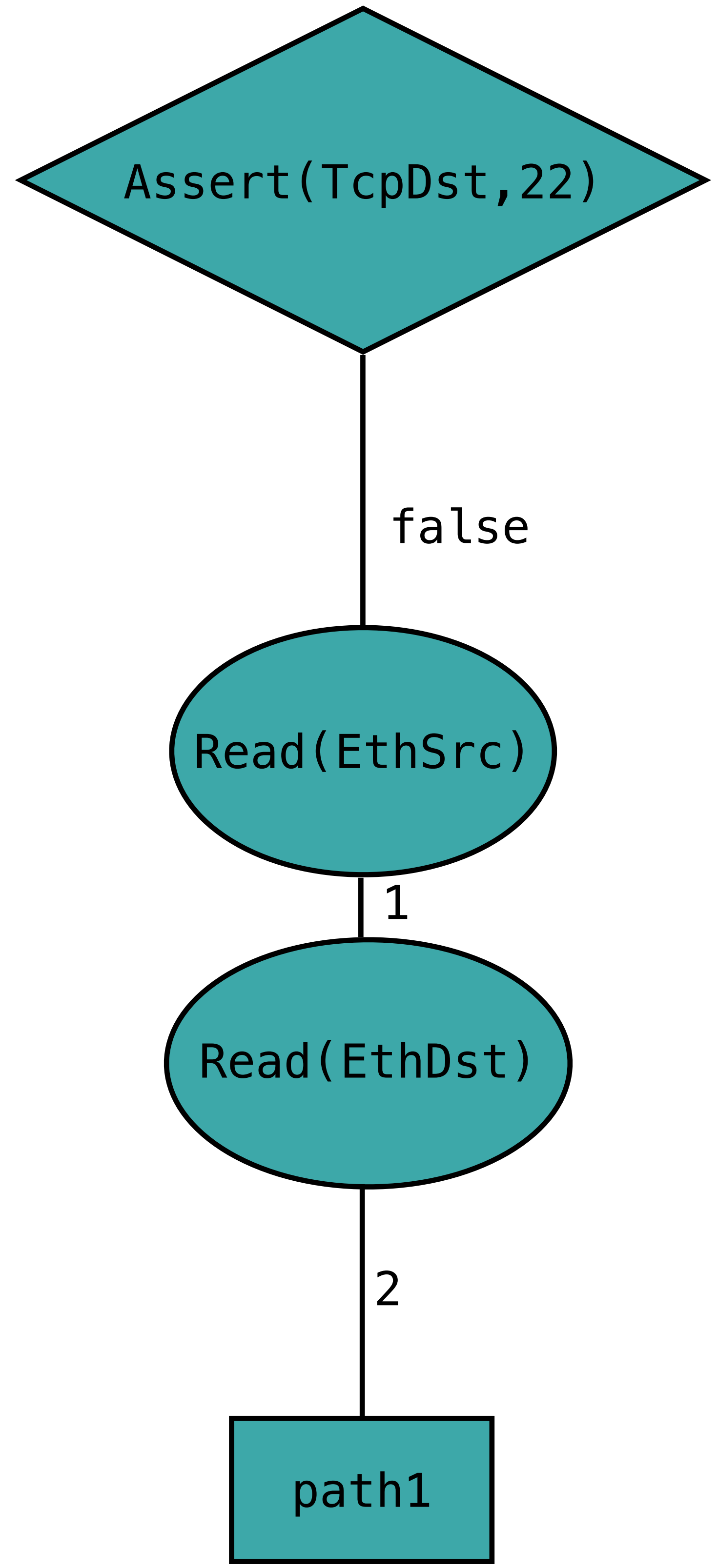
Policy

```
Route f(Packet p, Env e) {

  if (p.tcpDstIs(22))


    return null();

  else {

    Location sloc =
      e.location(p.ethSrc());

    Location dloc =
      e.location(p.ethDst());

    Path path     =
      shortestPath(
        e.links(),sloc,dloc);

    return
      unicast(sloc,dloc,path);
  }
}
```



Policy

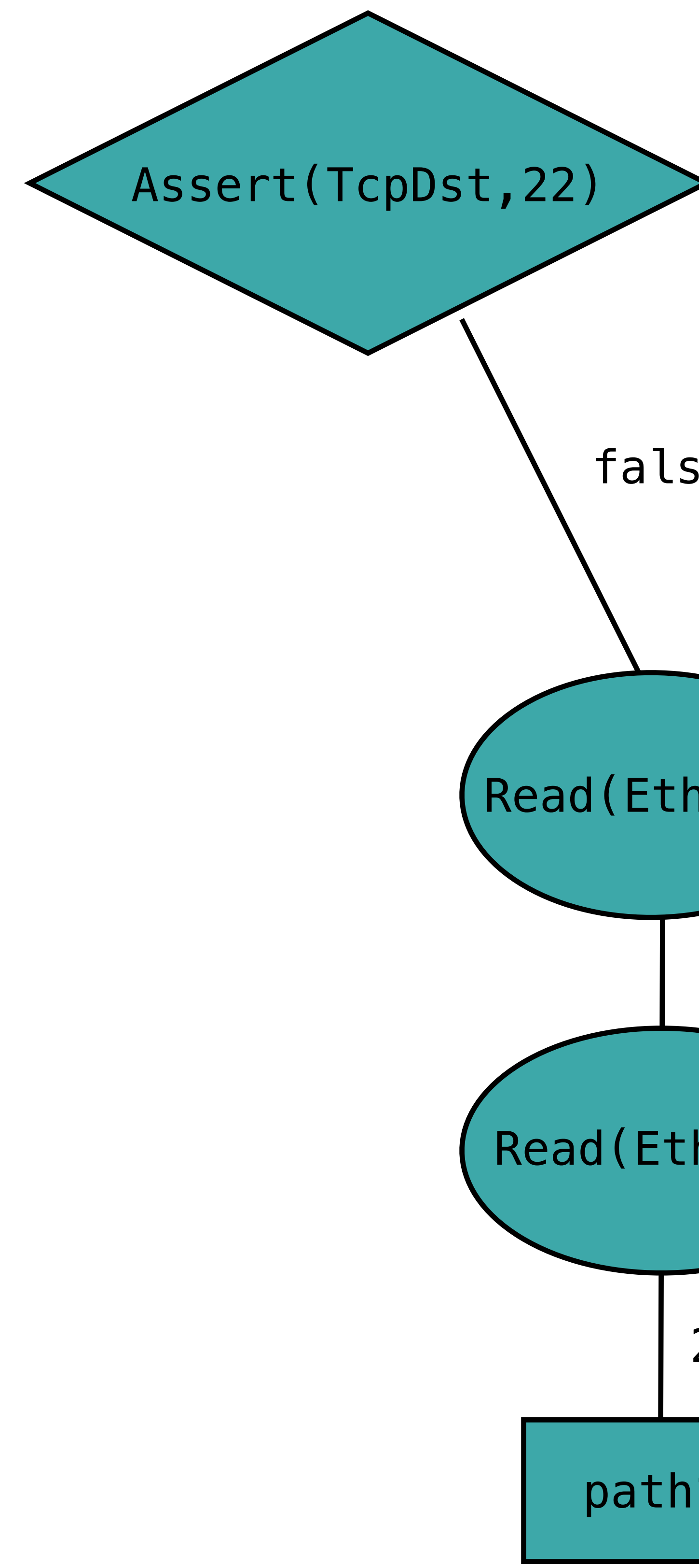
```
Route f(Packet p, Env e) {

  if (p.tcpDstIs(22))


    return null();

  else {

    Location sloc =
      e.location(p.ethSrc());

    Location dloc =
      e.location(p.ethDst());

    Path path     =
      shortestPath(
        e.links(),sloc,dloc);

    return
      unicast(sloc,dloc,path);
  }
}
```



Policy

```
Route f(Packet p, Env e) {
  if (p.tcpDstIs(22))

    return null();
  else {
    Location sloc =
      e.location(p.ethSrc());
    Location dloc =
      e.location(p.ethDst());
    Path path    =
      shortestPath(
        e.links(),sloc,dloc);
    return
      unicast(sloc,dloc,path);
  }
}
```

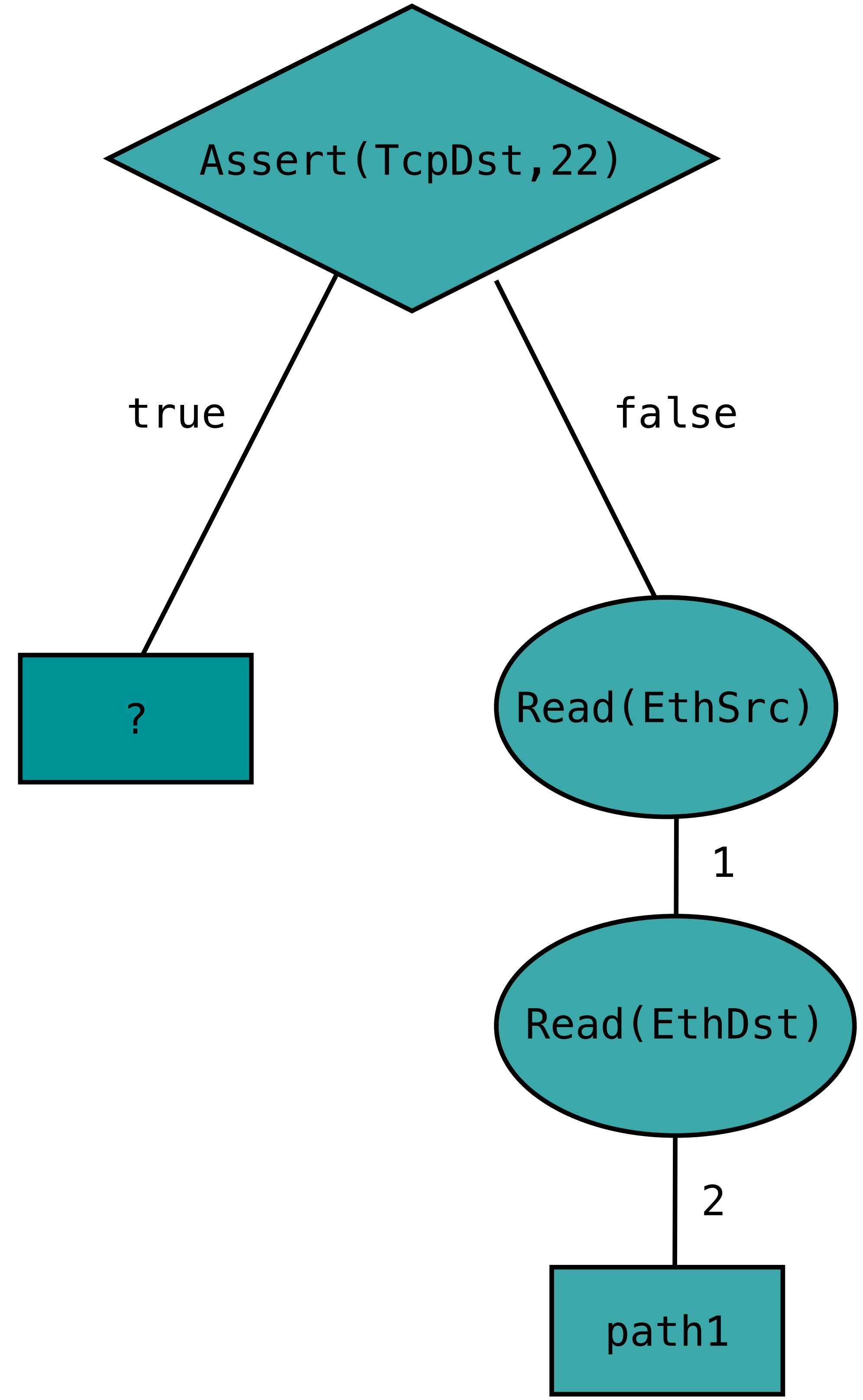
Trace Tree



Policy

```
Route f(Packet p, Env e) {
  if (p.tcpDstIs(22))


    return null();

  else {

    Location sloc =
      e.location(p.ethSrc());

    Location dloc =
      e.location(p.ethDst());

    Path path    =
      shortestPath(
        e.links(),sloc,dloc);

    return
      unicast(sloc,dloc,path);
  }
}
```

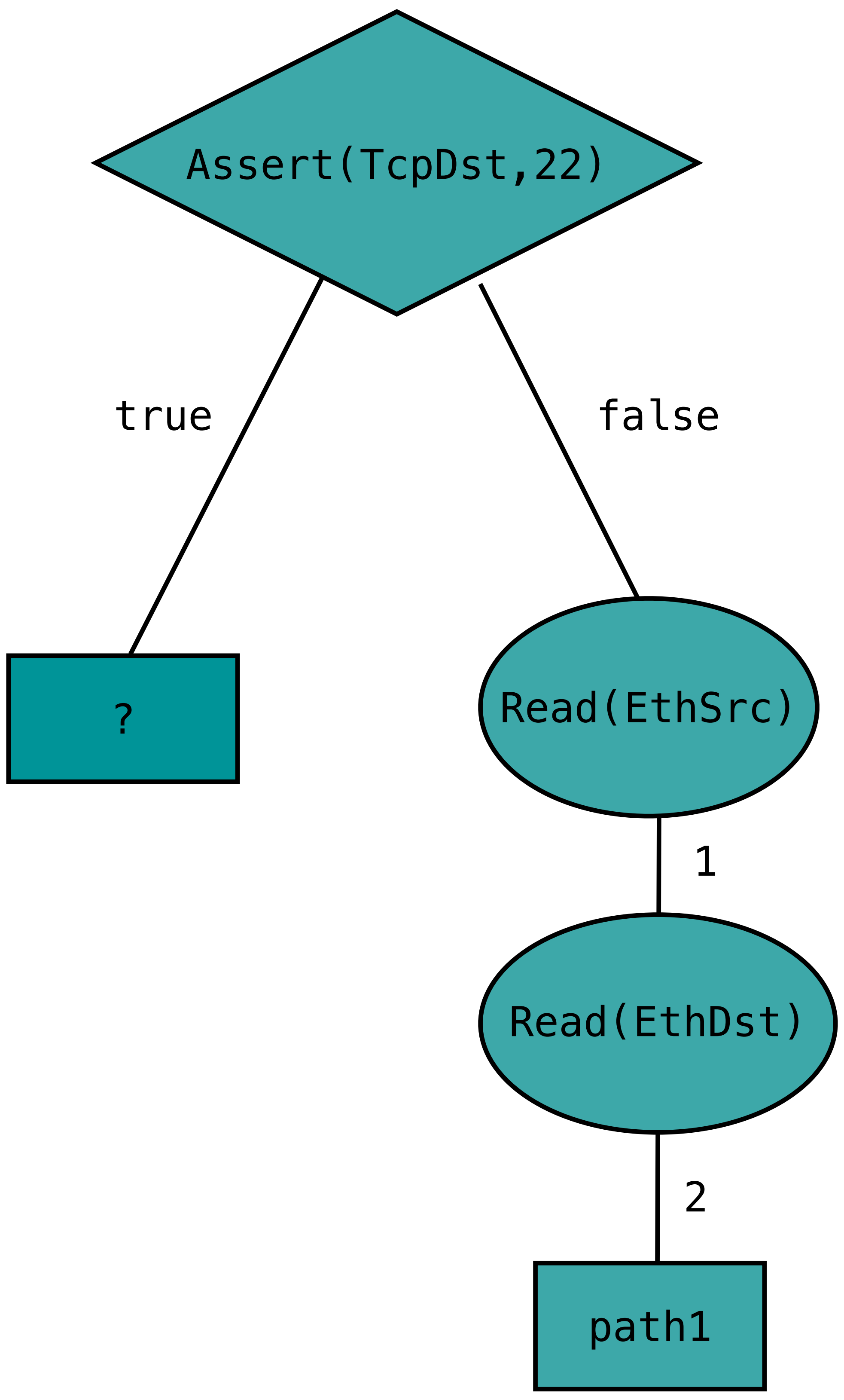
Trace Tree



Policy

```
Route f(Packet p, Env e) {
  if (p.tcpDstIs(22))

    return null();
  else {
    Location sloc =
      e.location(p.ethSrc());
    Location dloc =
      e.location(p.ethDst());
    Path path    =
      shortestPath(
        e.links(),sloc,dloc);
    return
      unicast(sloc,dloc,path);
  }
}
```

Trace Tree

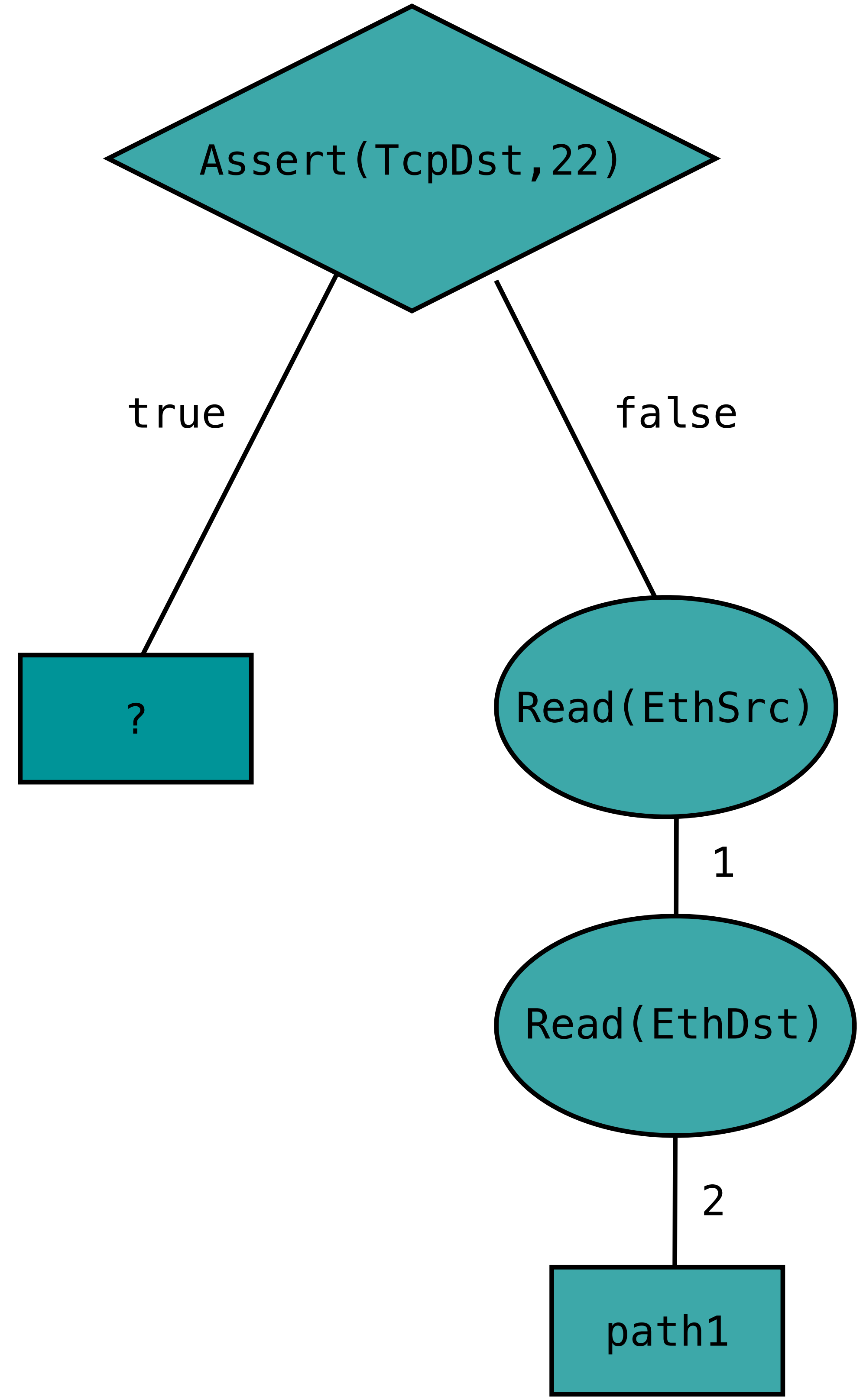


Policy

EthDst:1,
TcpDst:22

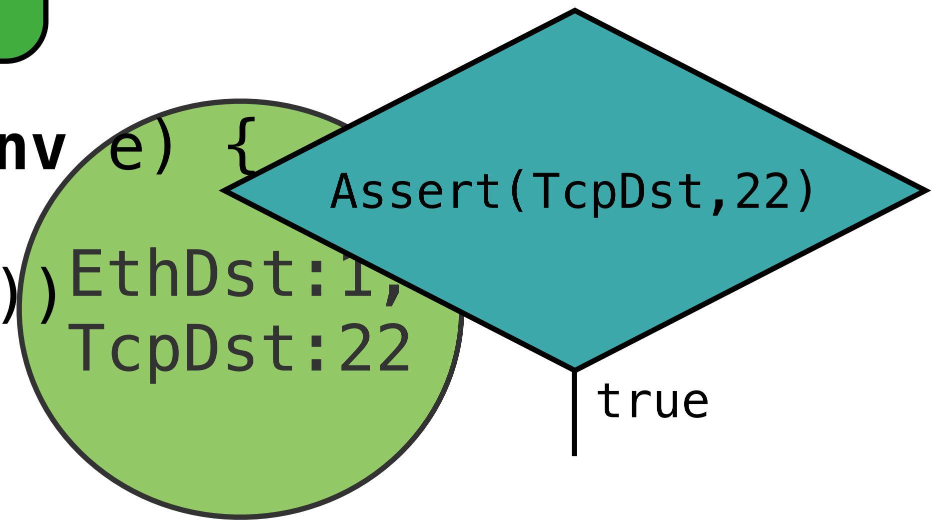
```
Route f(Packet p, Env e) {
  if (p.tcpDstIs(22))

    return null();
  else {
    Location sloc =
      e.location(p.ethSrc());
    Location dloc =
      e.location(p.ethDst());
    Path path    =
      shortestPath(
        e.links(),sloc,dloc);
    return
      unicast(sloc,dloc,path);
  }
}
```

Trace Tree

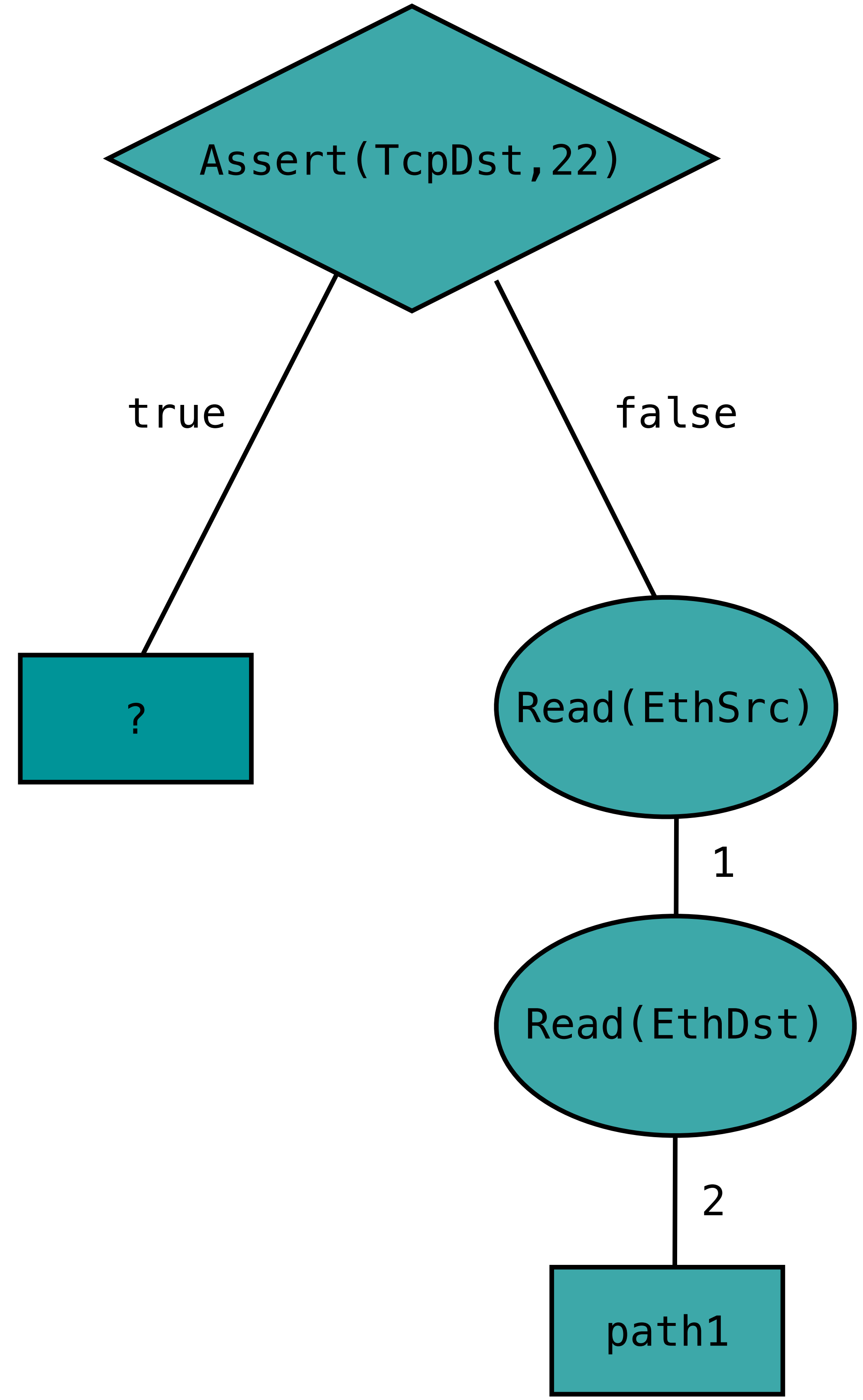


Policy

```
Route f(Packet p, Env e) {
  if (p.tcpDstIs(22))
    return null();
  else {
    Location sloc =
      e.location(p.ethSrc());
    Location dloc =
      e.location(p.ethDst());
    Path path    =
      shortestPath(
        e.links(),sloc,dloc);
    return
      unicast(sloc,dloc,path);
  }
}
```

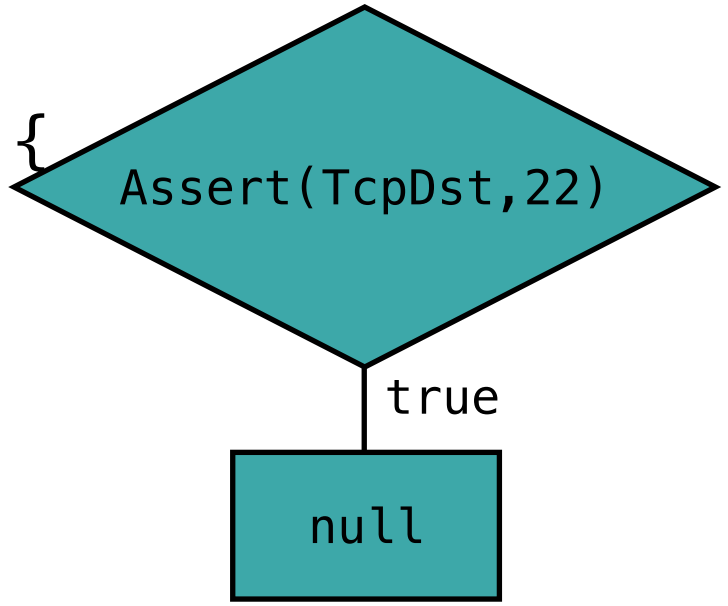


Trace Tree

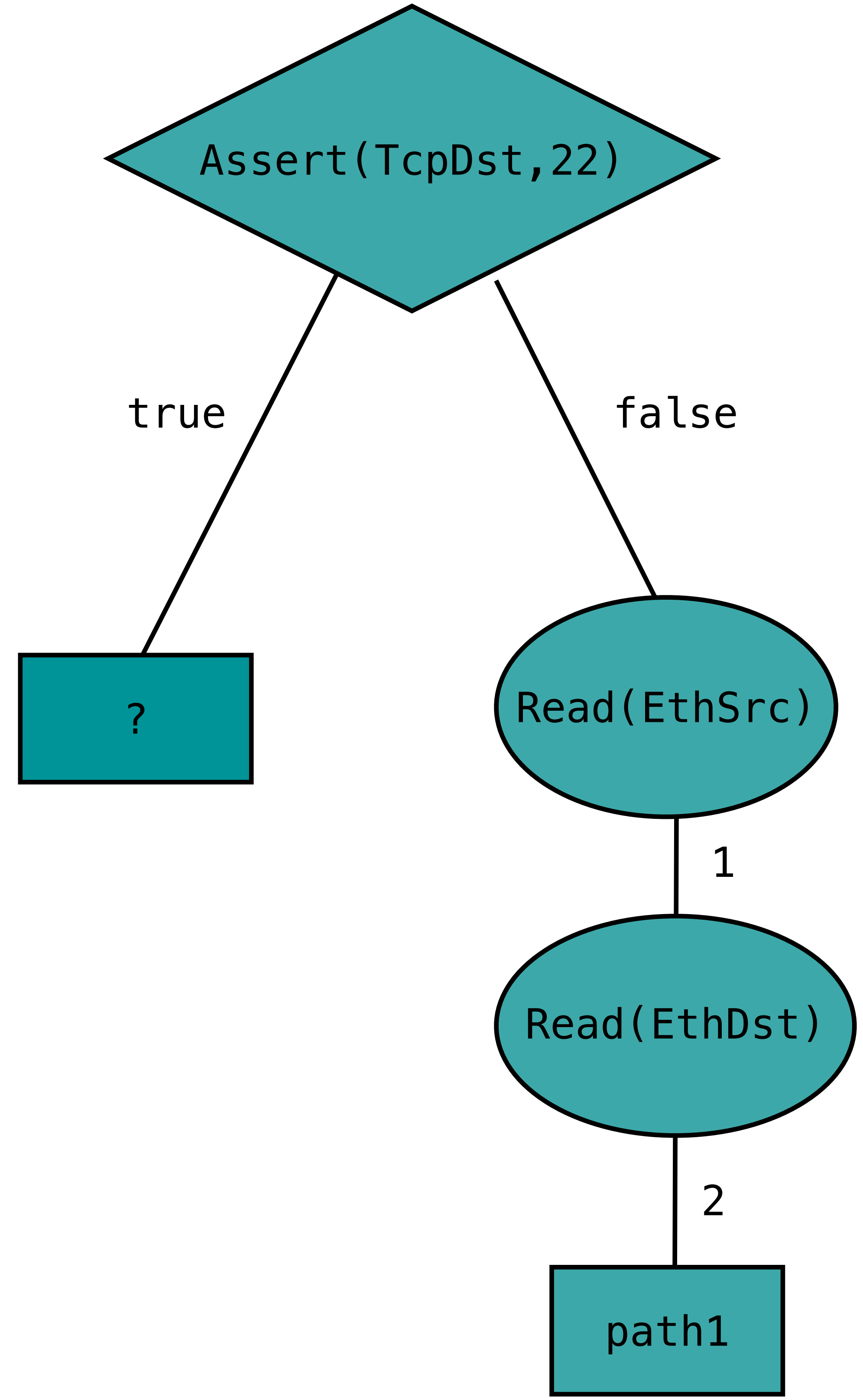


Policy

```
Route f(Packet p, Env e) {

  if (p.tcpDstIs(22))


    return null();

  else {

    Location sloc =
      e.location(p.ethSrc());

    Location dloc =
      e.location(p.ethDst());

    Path path    =
      shortestPath(
        e.links(),sloc,dloc);

    return
      unicast(sloc,dloc,path);
  }
}
```



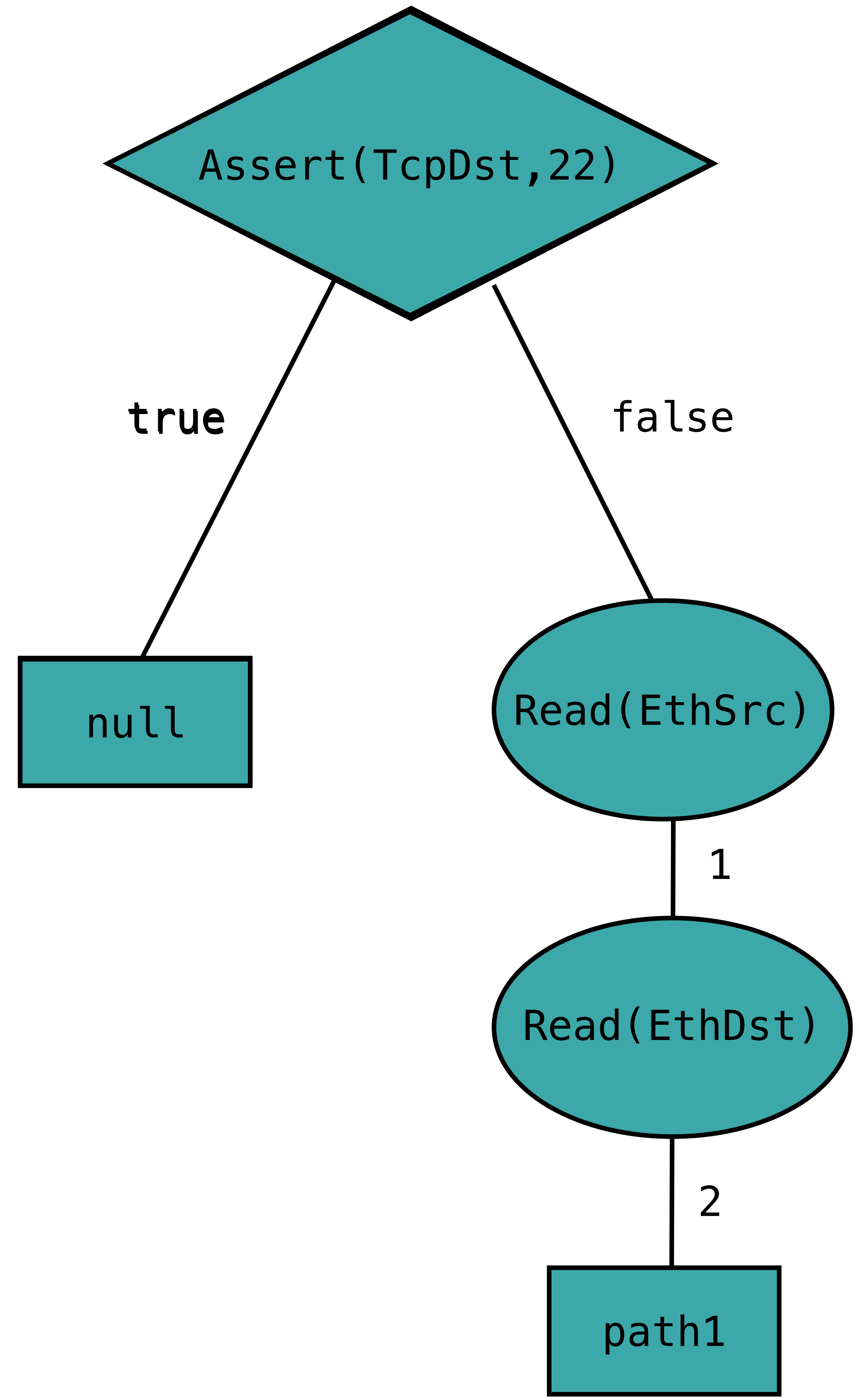
Trace Tree



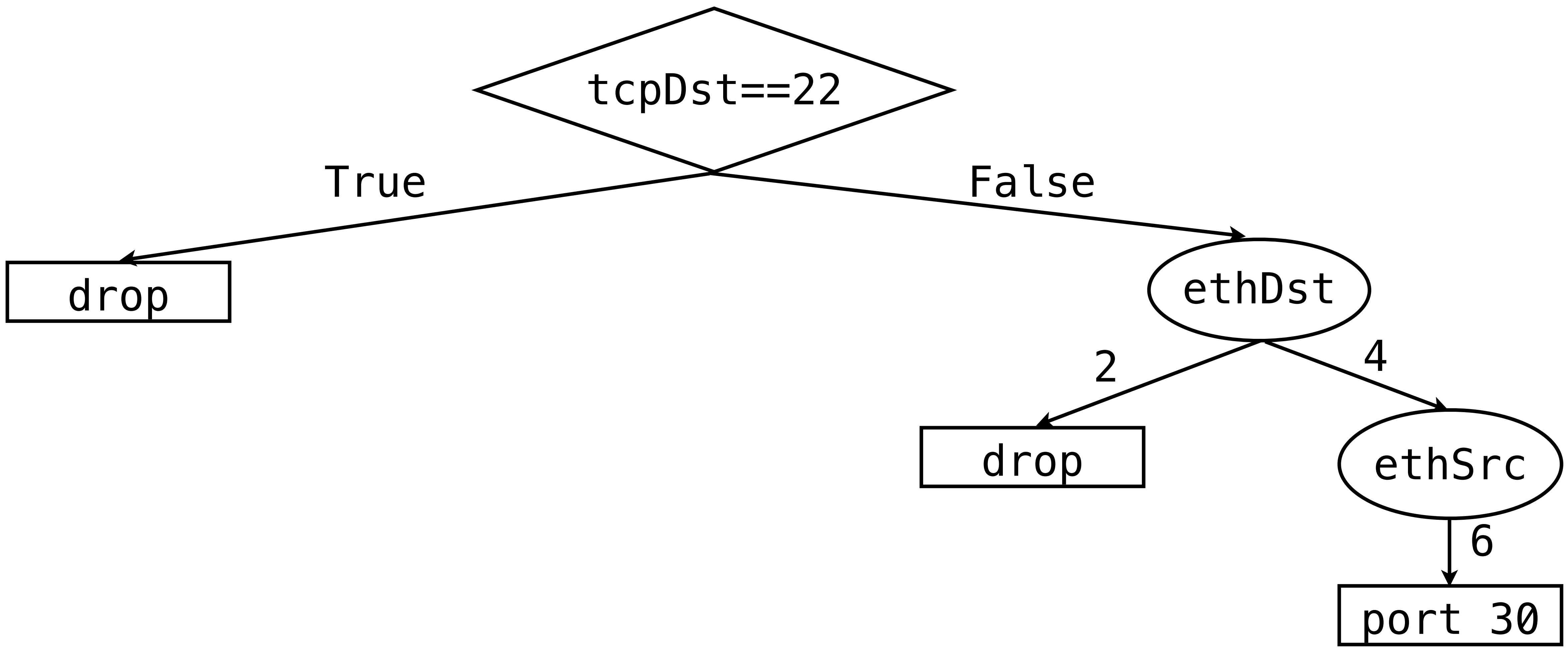
Policy

```
Route f(Packet p, Env e) {

  if (p.tcpDstIs(22))


    return null();

  else {

    Location sloc =
      e.location(p.ethSrc());

    Location dloc =
      e.location(p.ethDst());

    Path path    =
      shortestPath(
        e.links(),sloc,dloc);

    return
      unicast(sloc,dloc,path);
  }
}
```

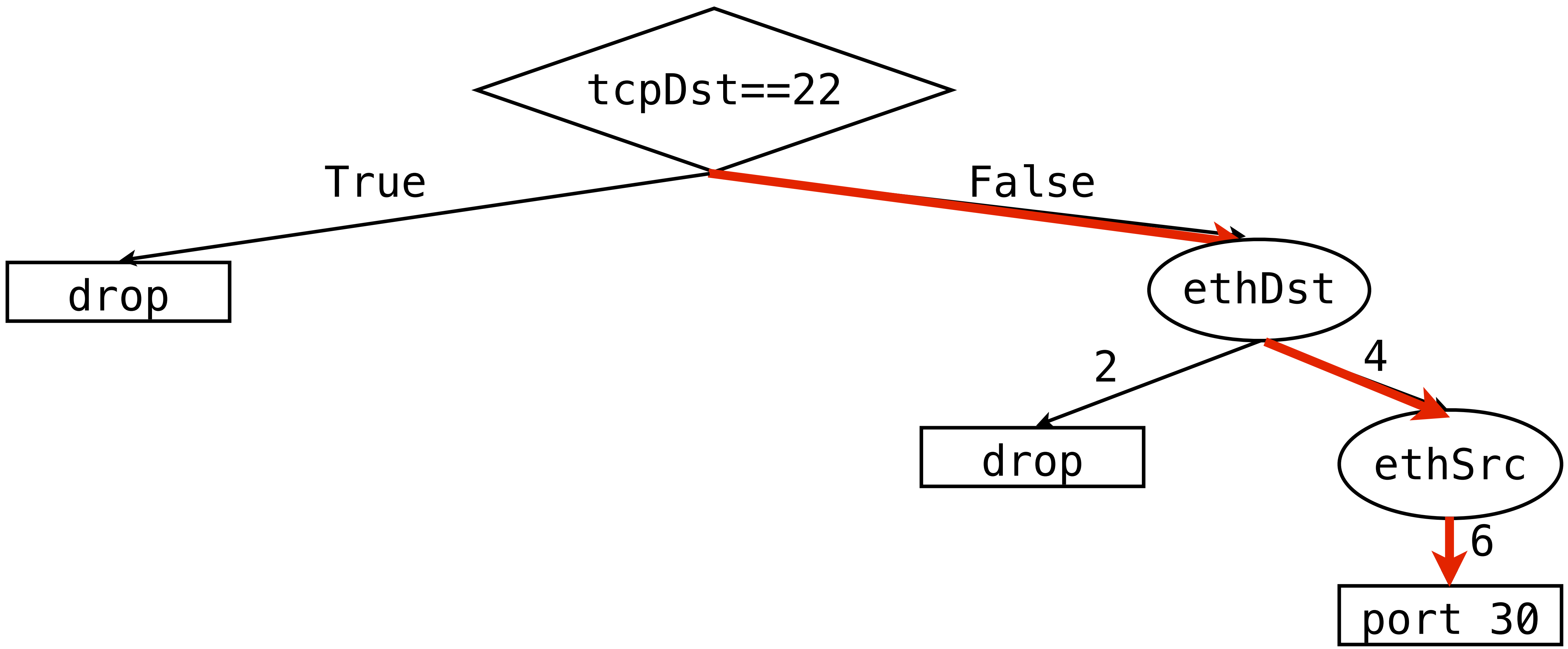
Trace Tree



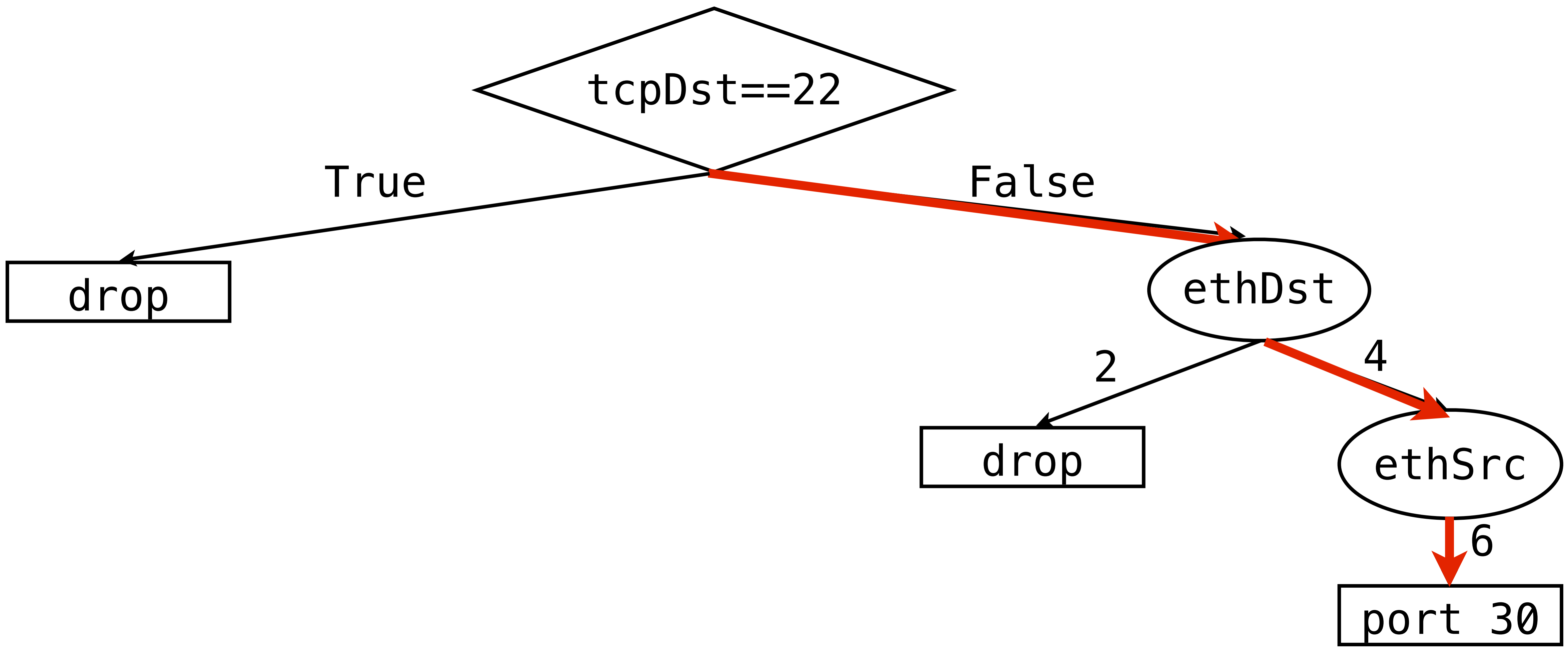
Compile recorded executions into flow table



Compile recorded executions into flow table

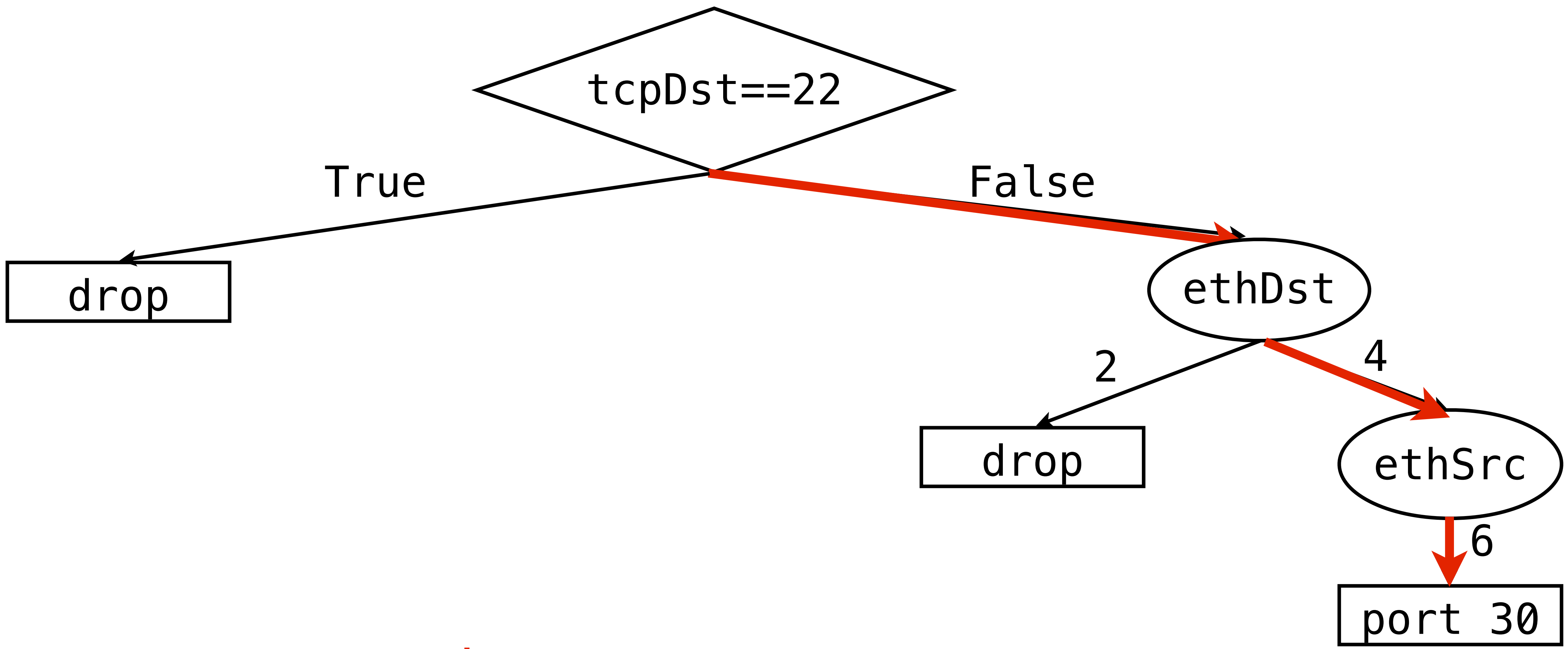


Compile recorded executions into flow table



match:{tcpDst!=22,ethDst:4,ethSrc:6}

Compile recorded executions into flow table

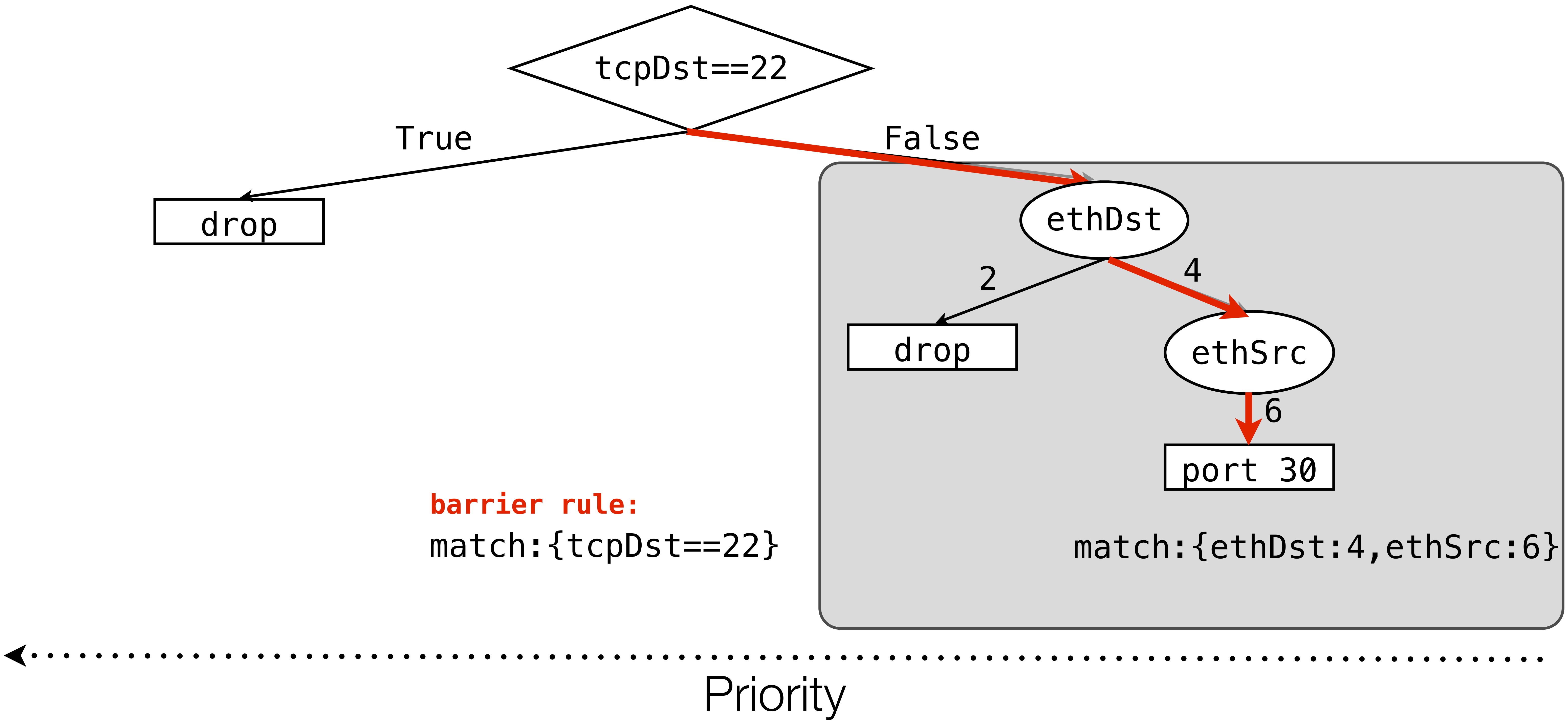


barrier rule:
match:{tcpDst==22}

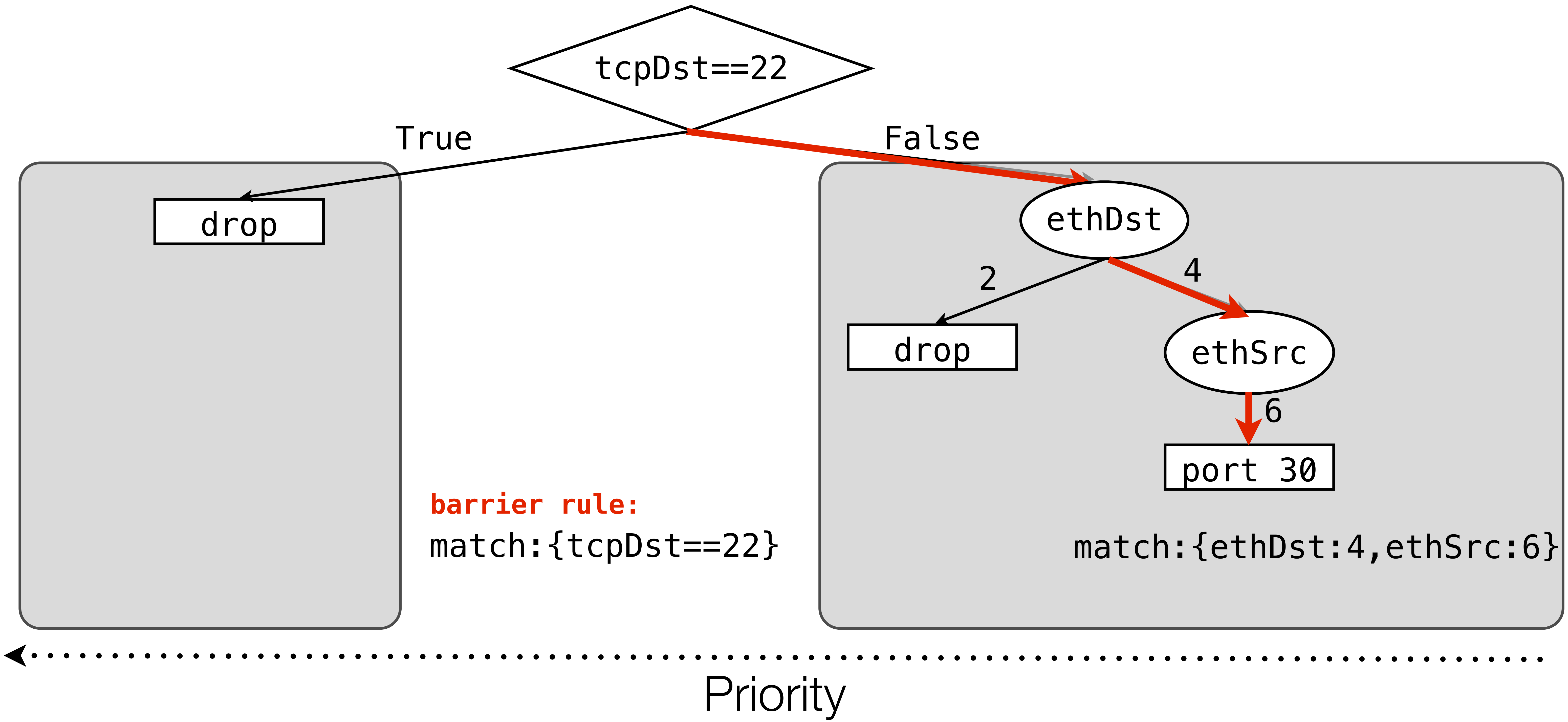
match:{ethDst:4,ethSrc:6}

Priority

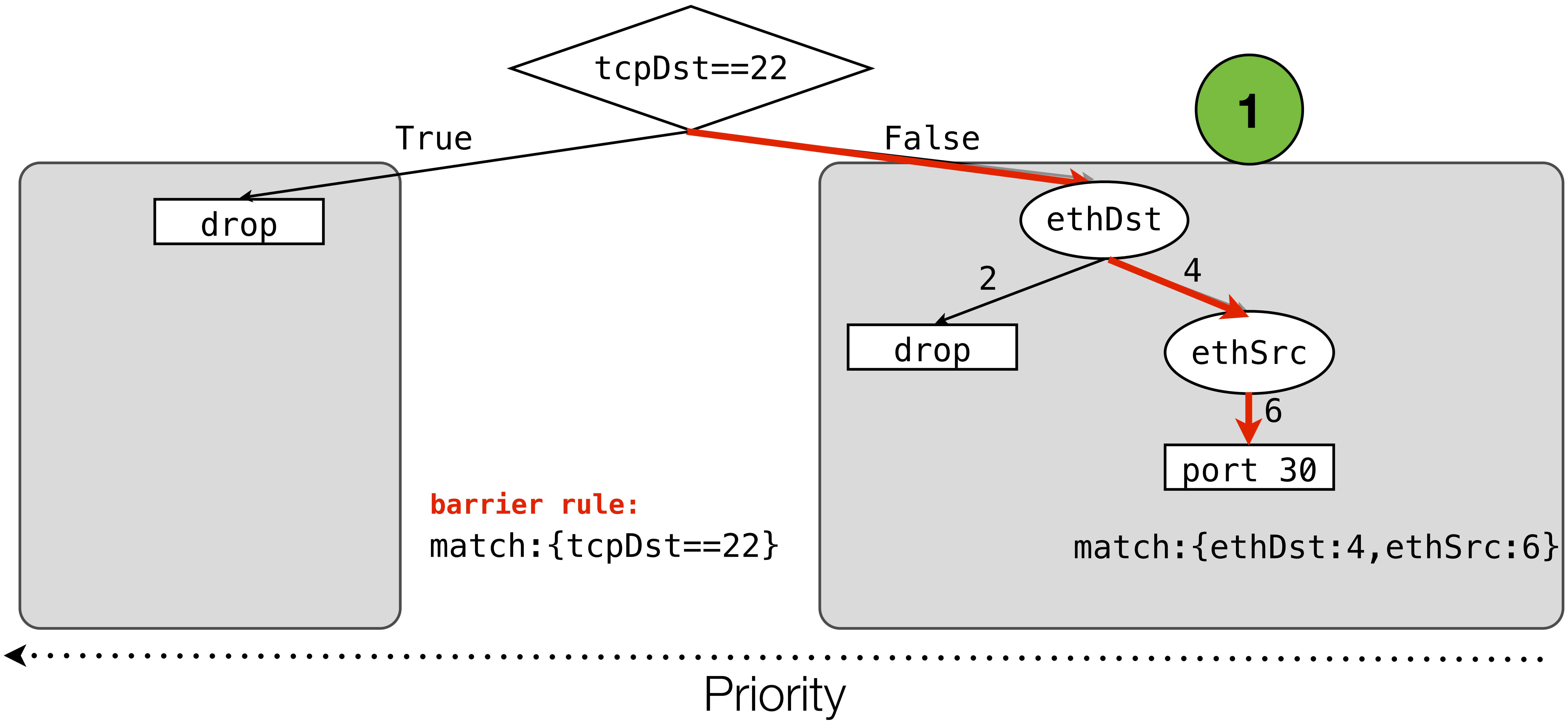
Compile recorded executions into flow table



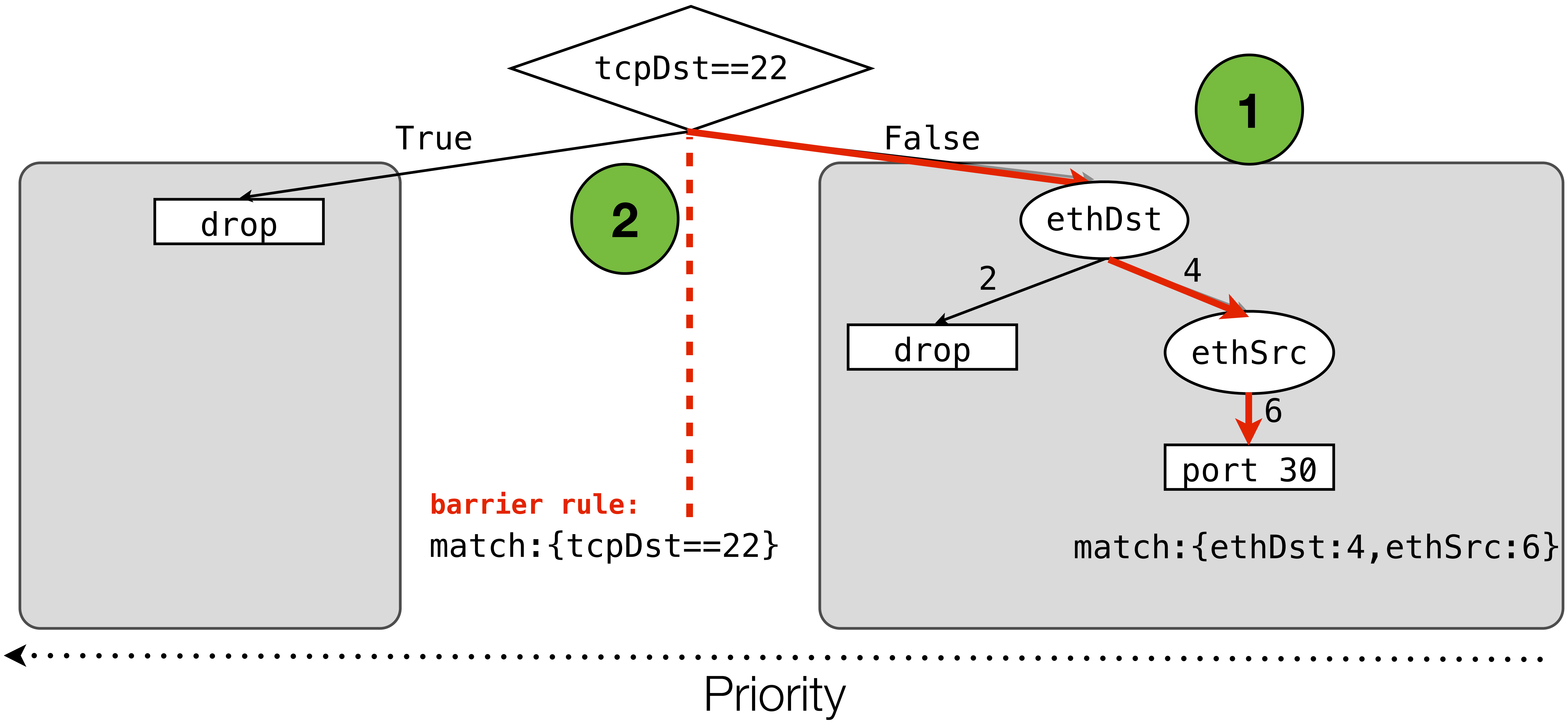
Compile recorded executions into flow table



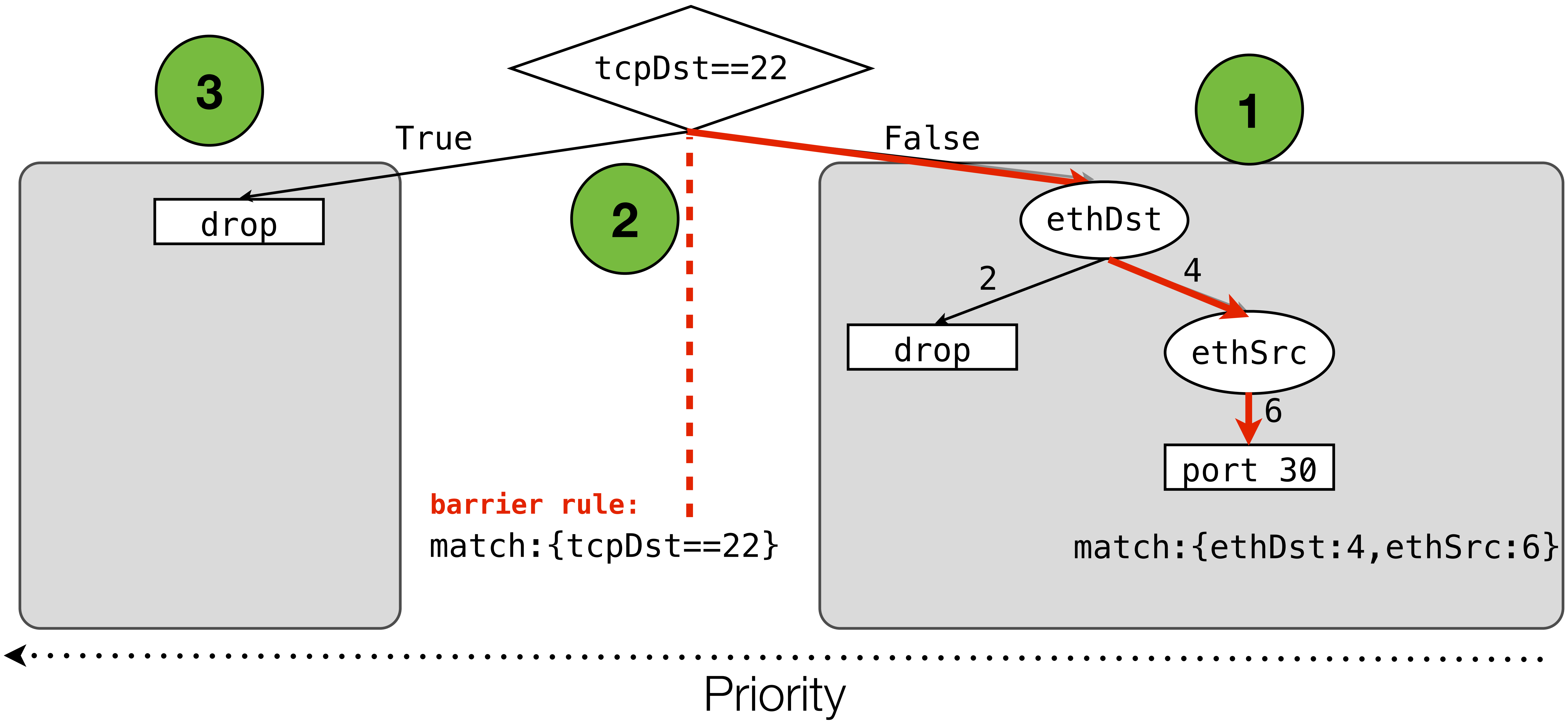
Compile recorded executions into flow table



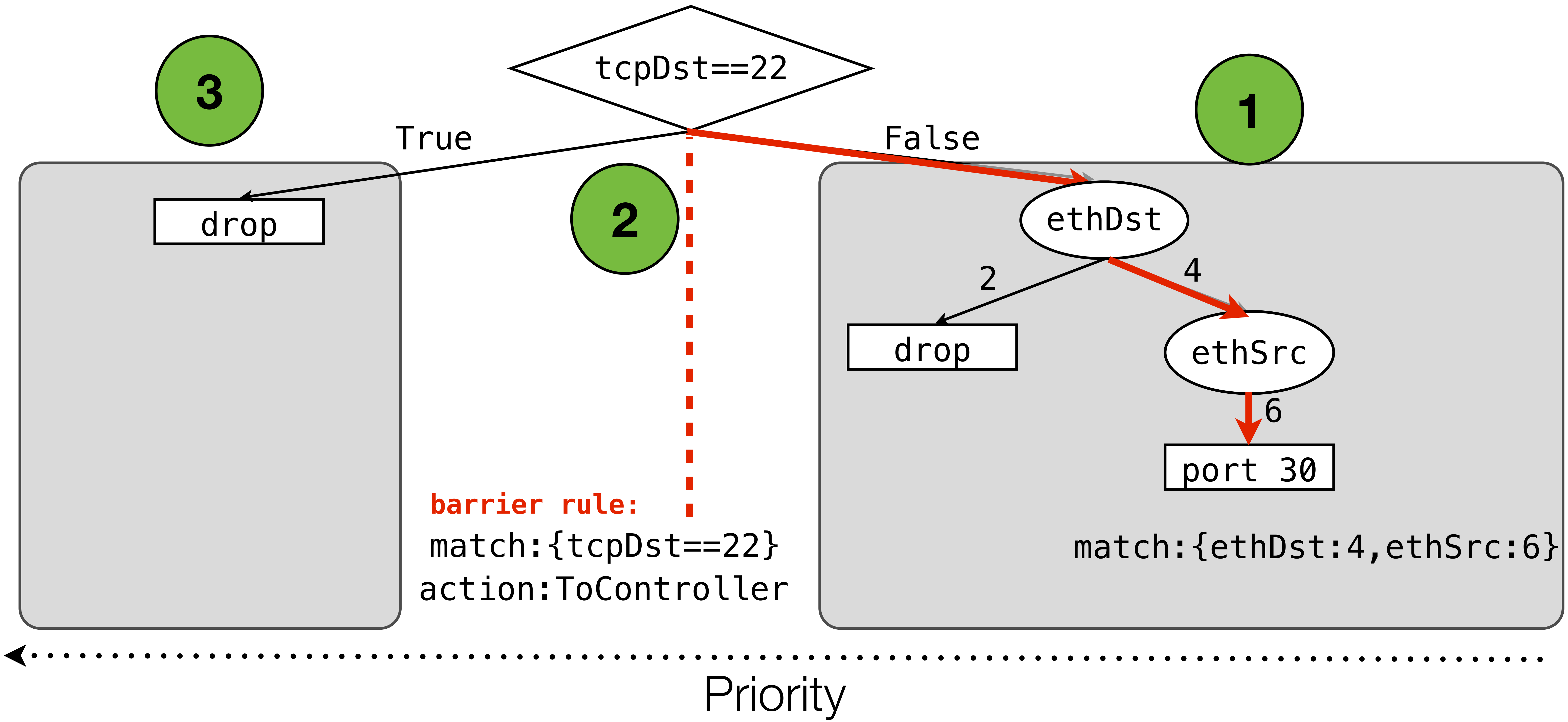
Compile recorded executions into flow table



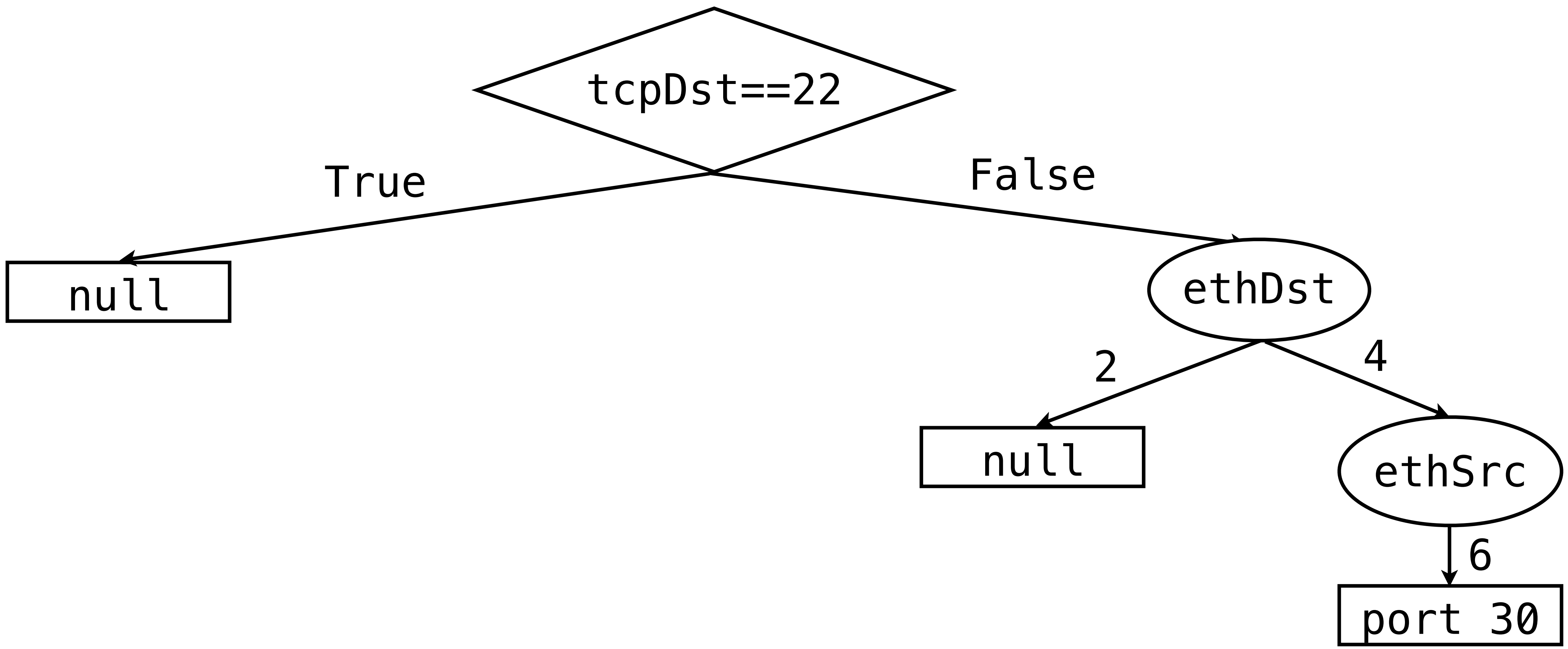
Compile recorded executions into flow table



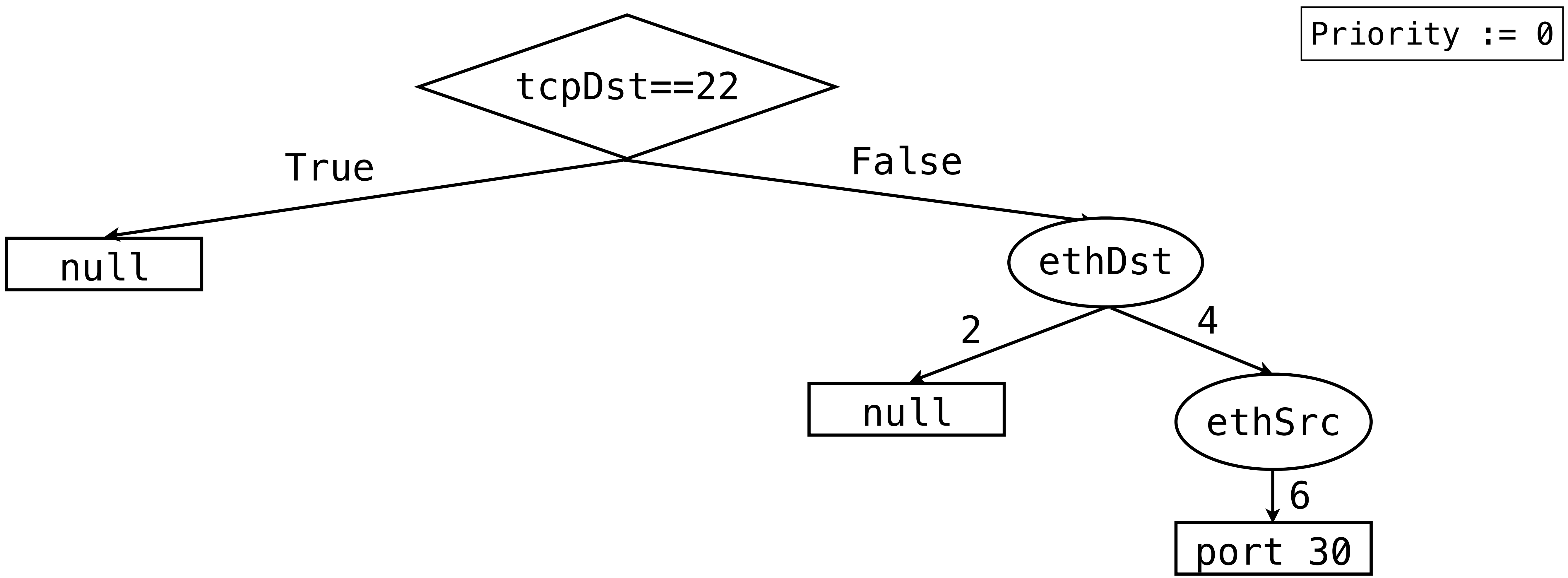
Compile recorded executions into flow table



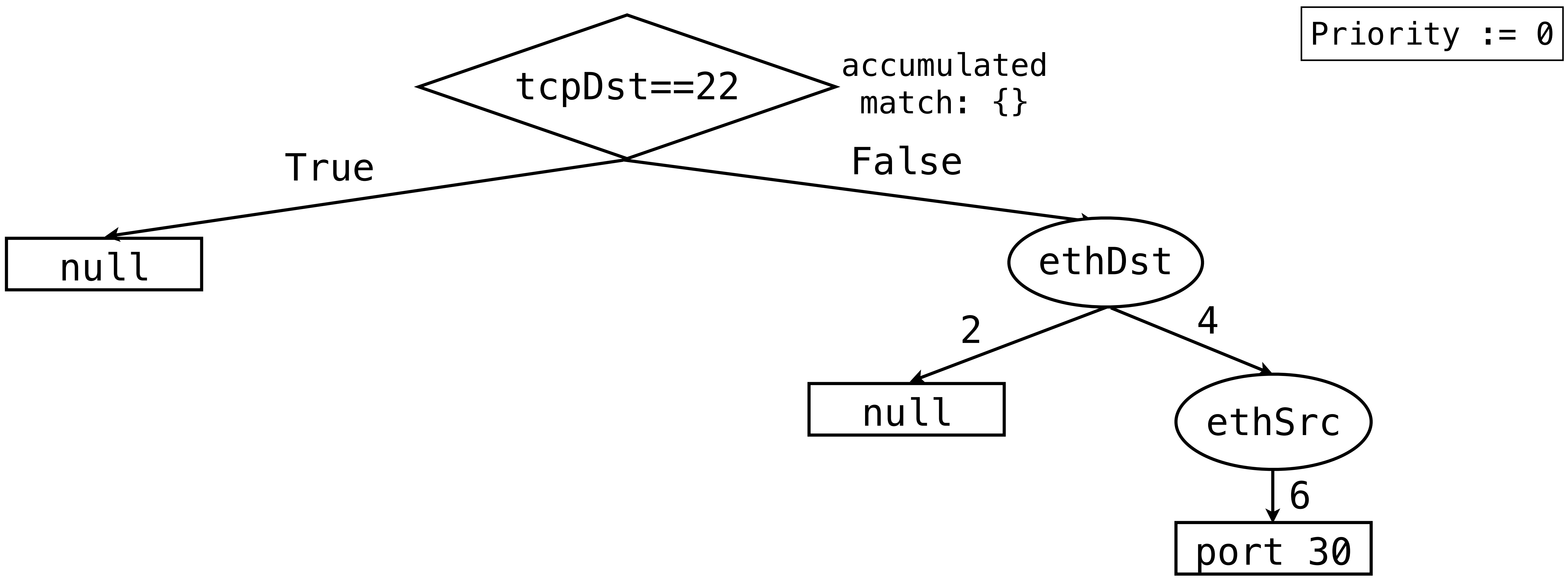
Basic compilation: in-order traversal & barrier rules



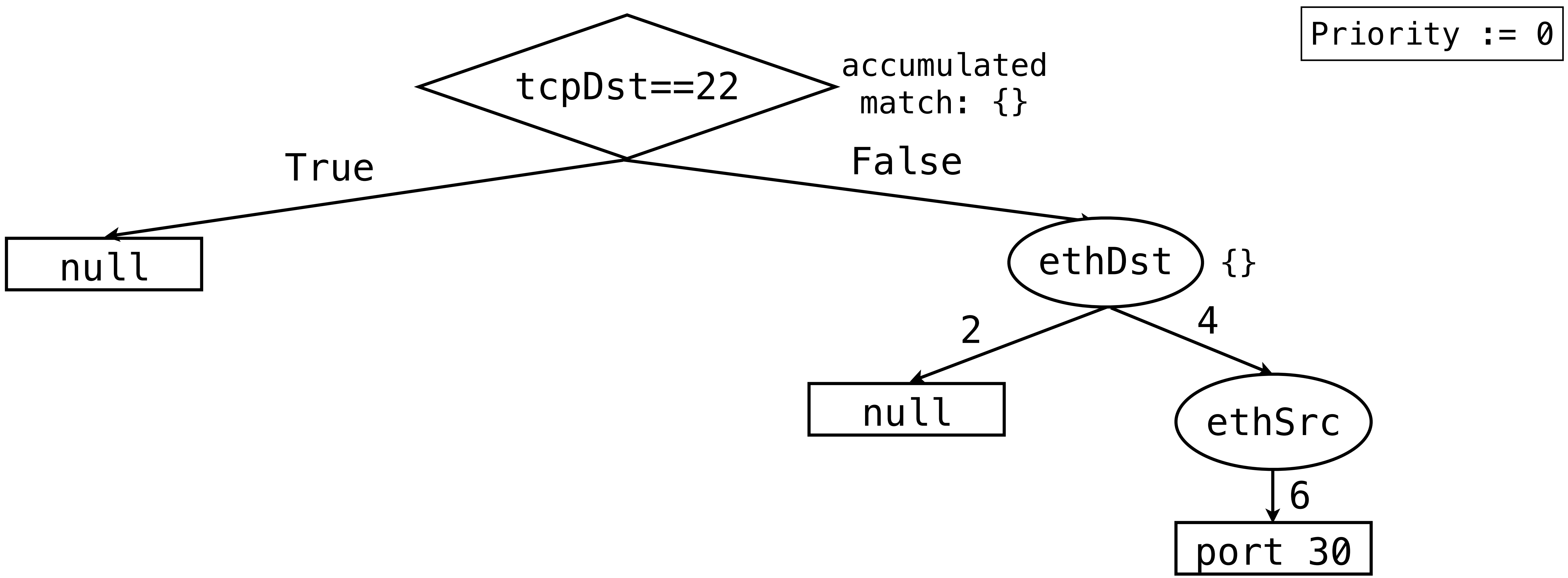
Basic compilation: in-order traversal & barrier rules



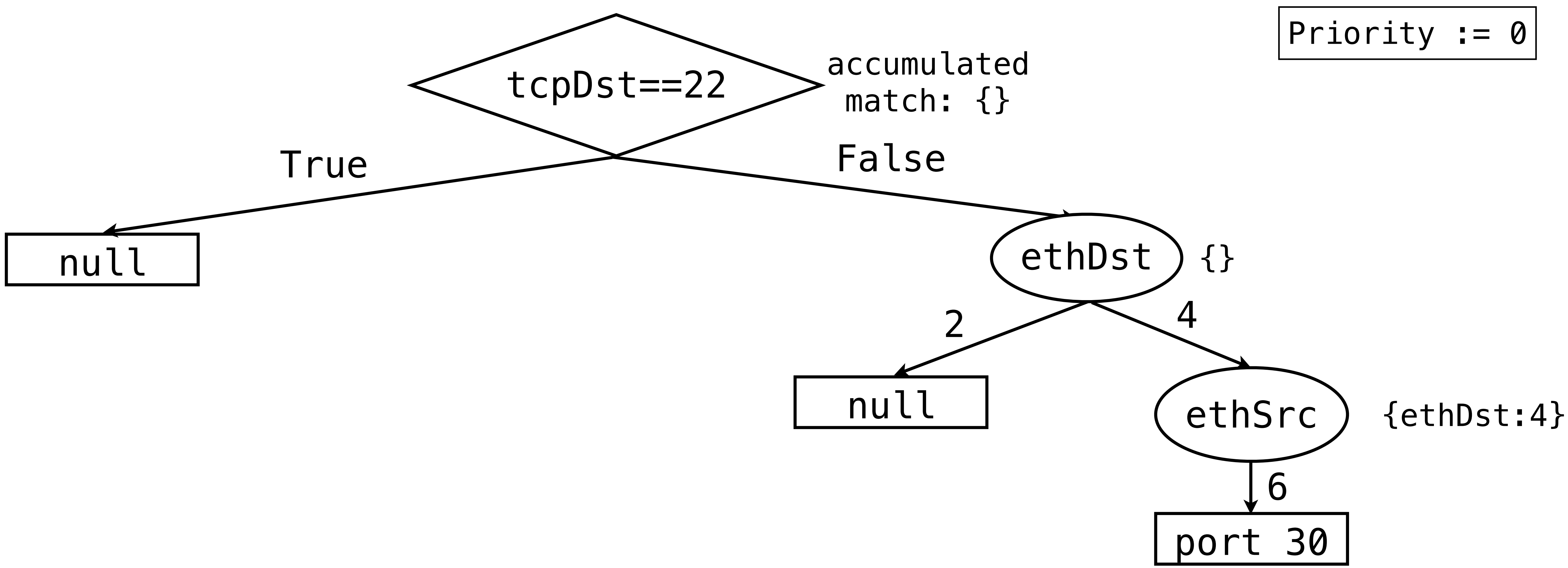
Basic compilation: in-order traversal & barrier rules



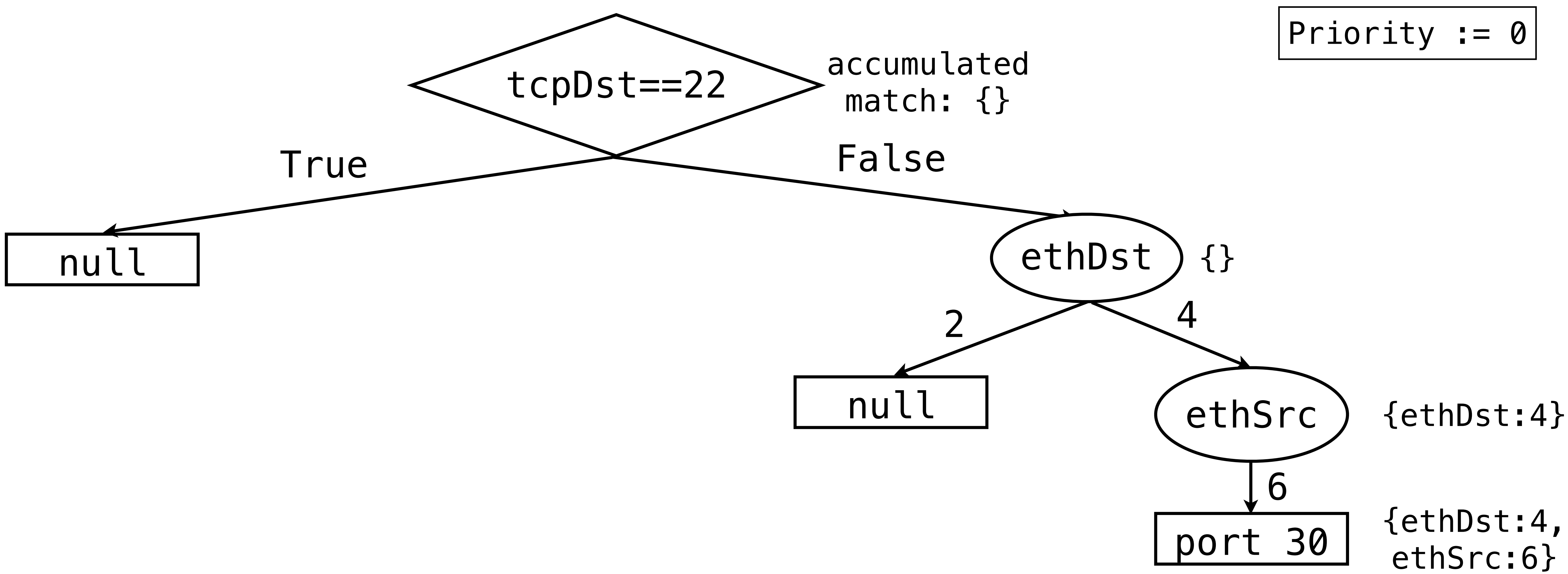
Basic compilation: in-order traversal & barrier rules



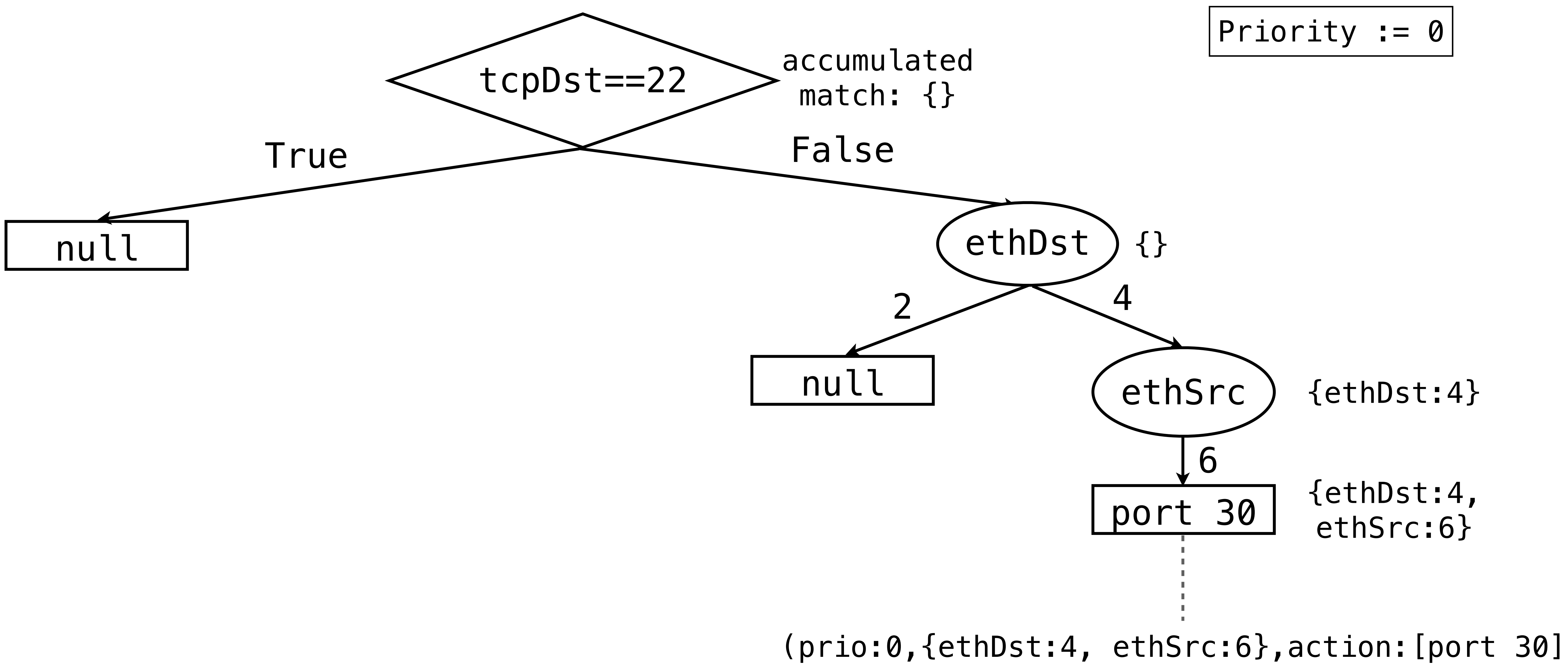
Basic compilation: in-order traversal & barrier rules



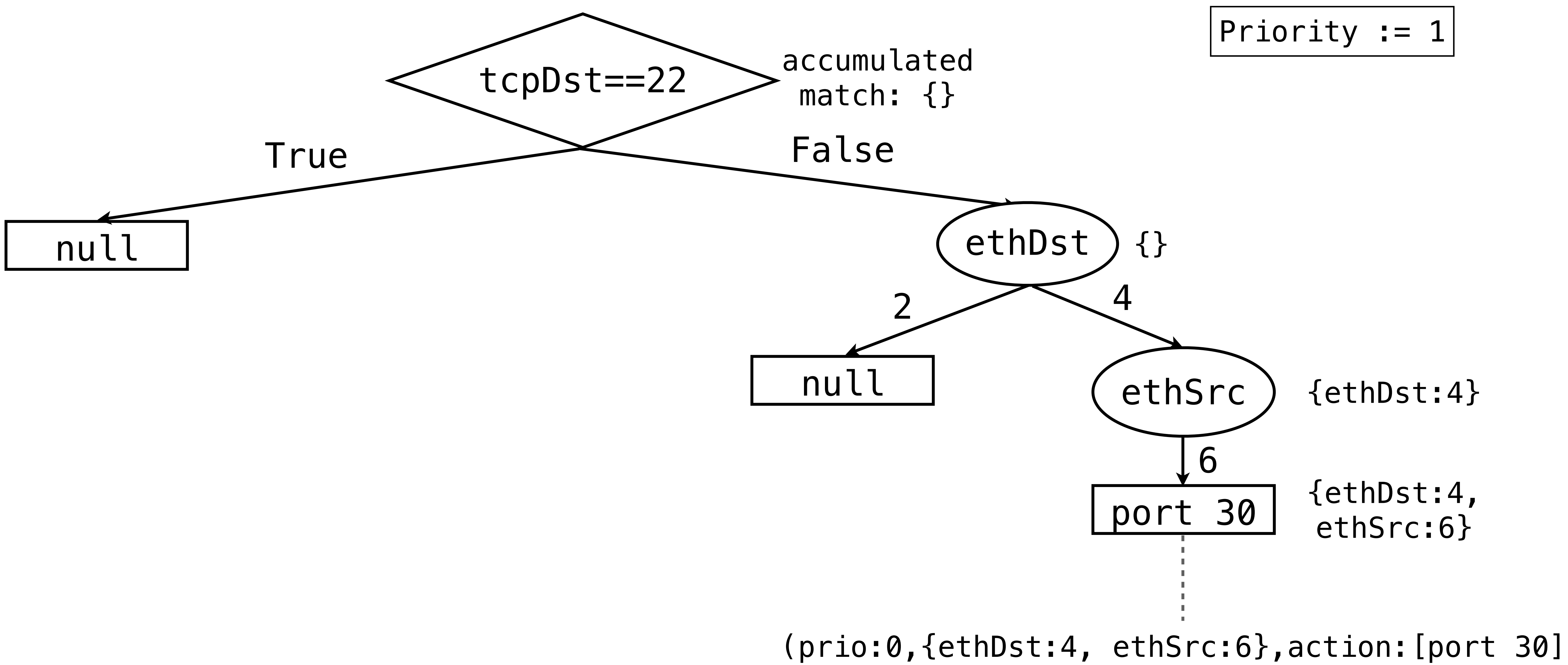
Basic compilation: in-order traversal & barrier rules



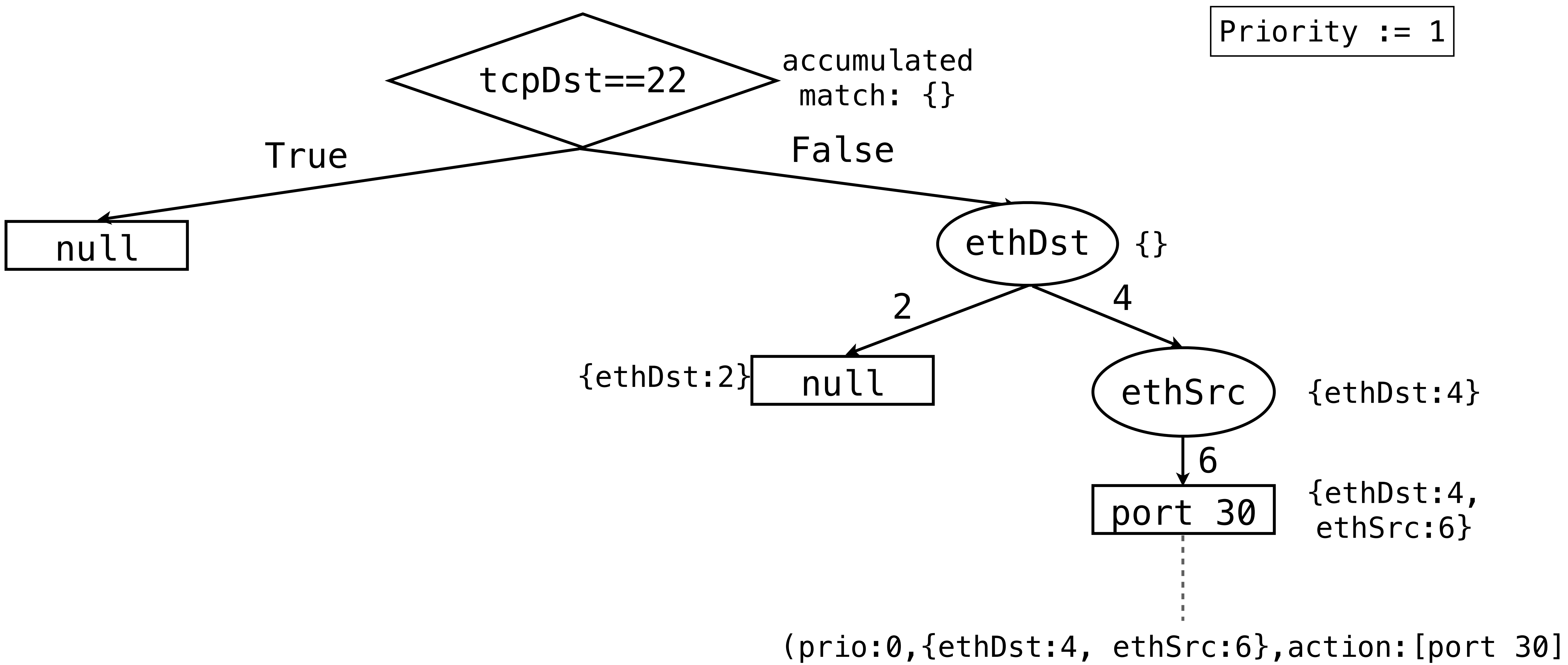
Basic compilation: in-order traversal & barrier rules



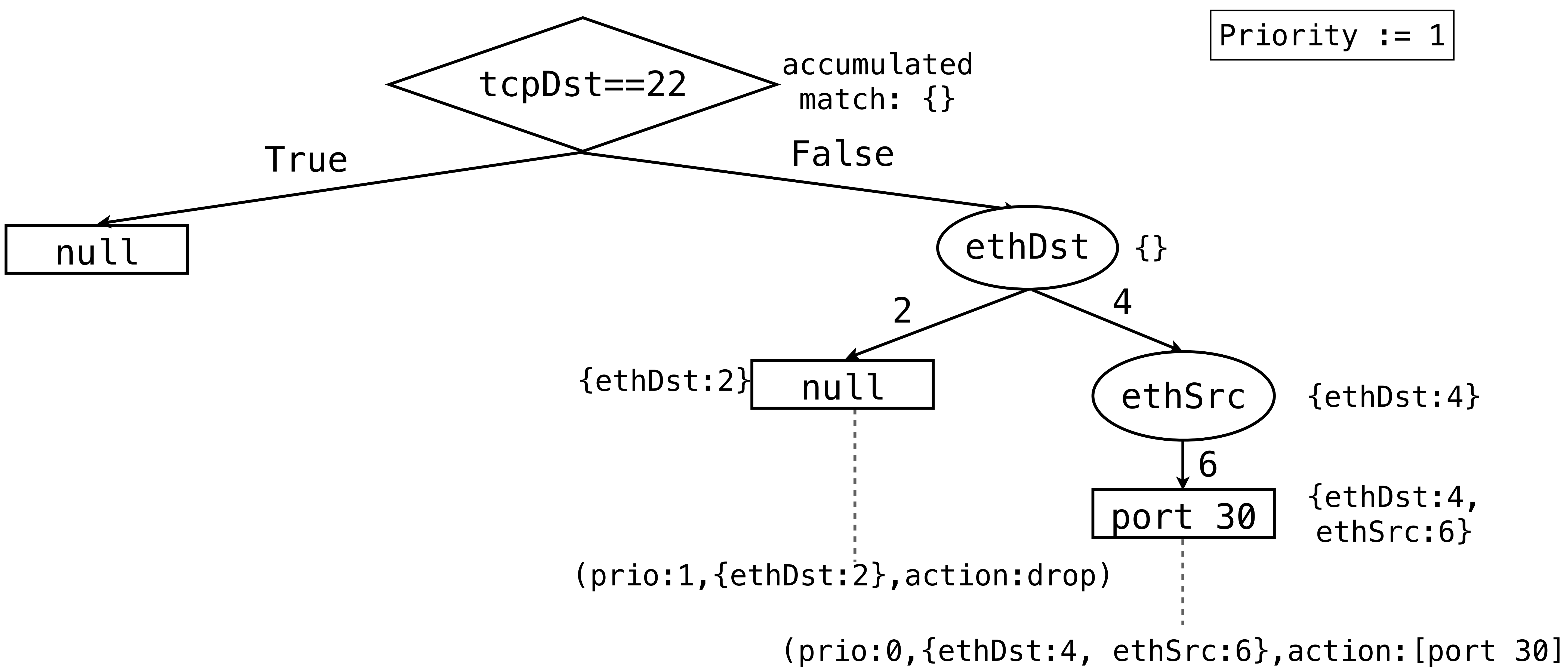
Basic compilation: in-order traversal & barrier rules



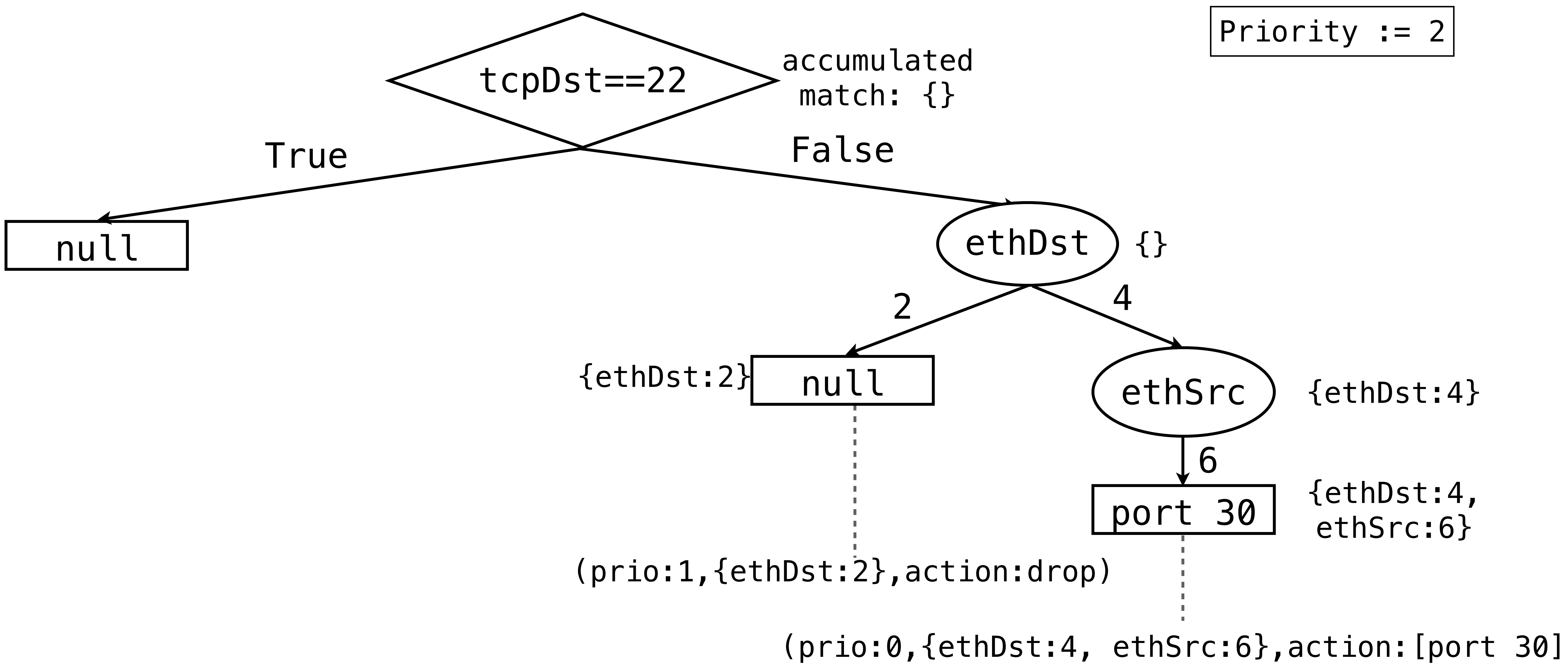
Basic compilation: in-order traversal & barrier rules



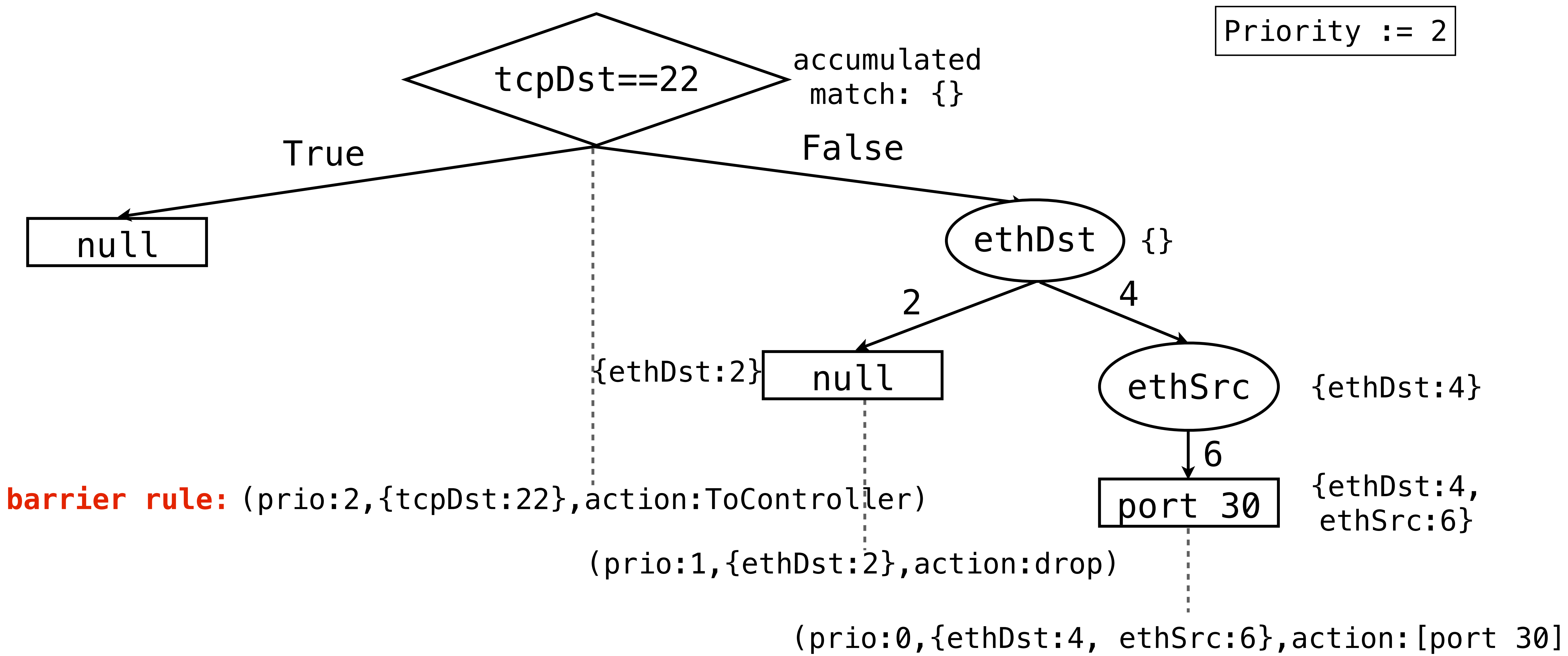
Basic compilation: in-order traversal & barrier rules



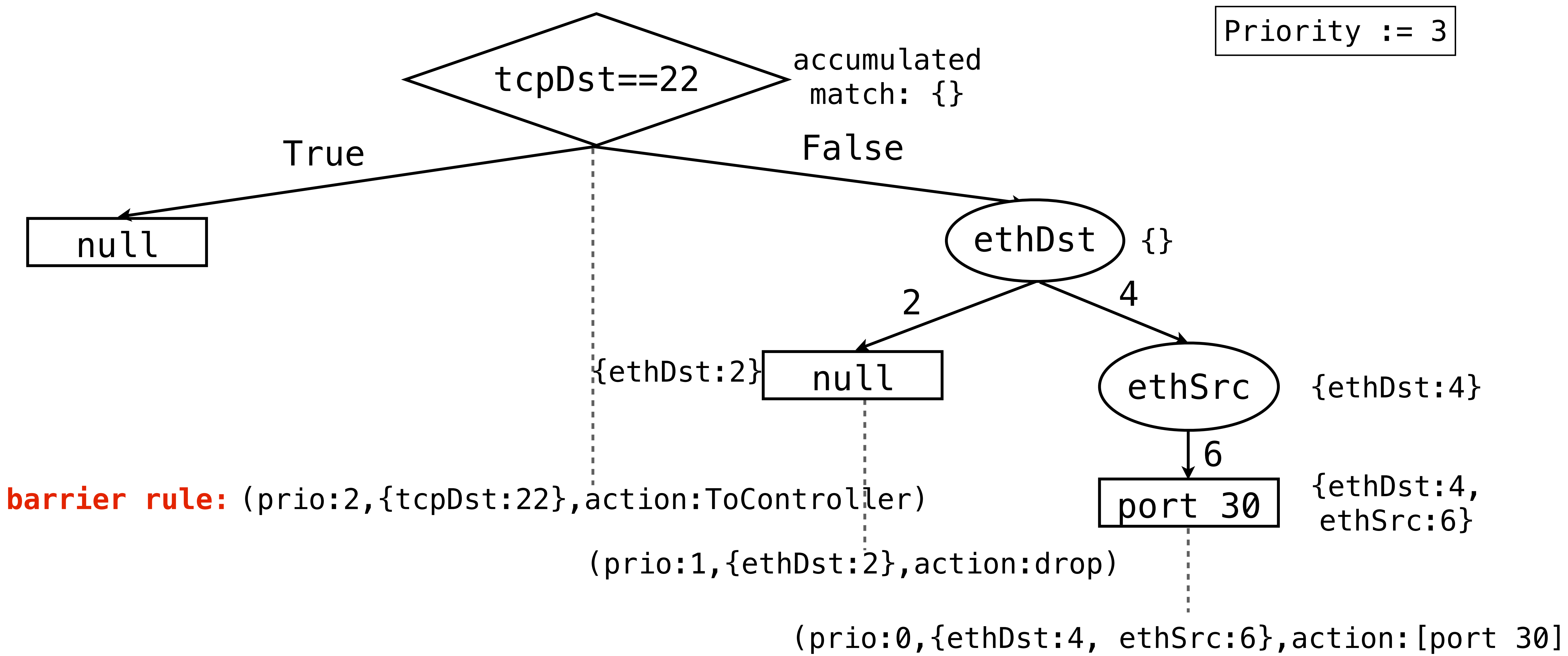
Basic compilation: in-order traversal & barrier rules



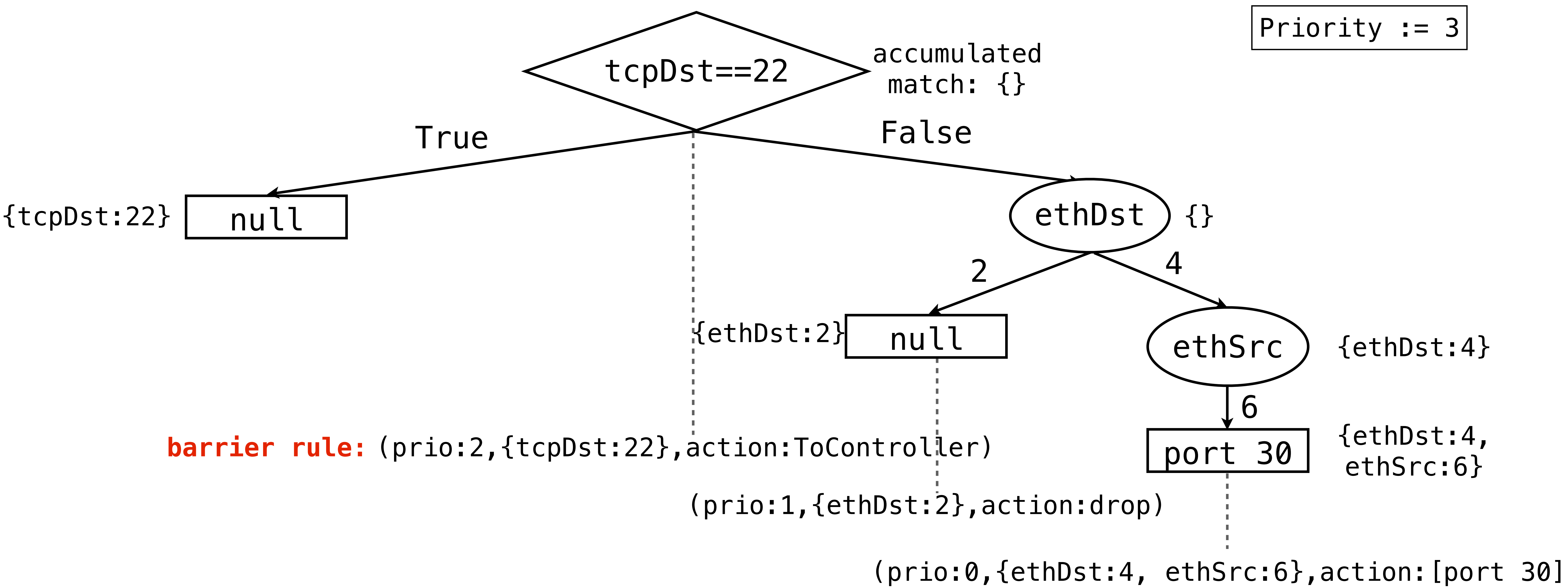
Basic compilation: in-order traversal & barrier rules



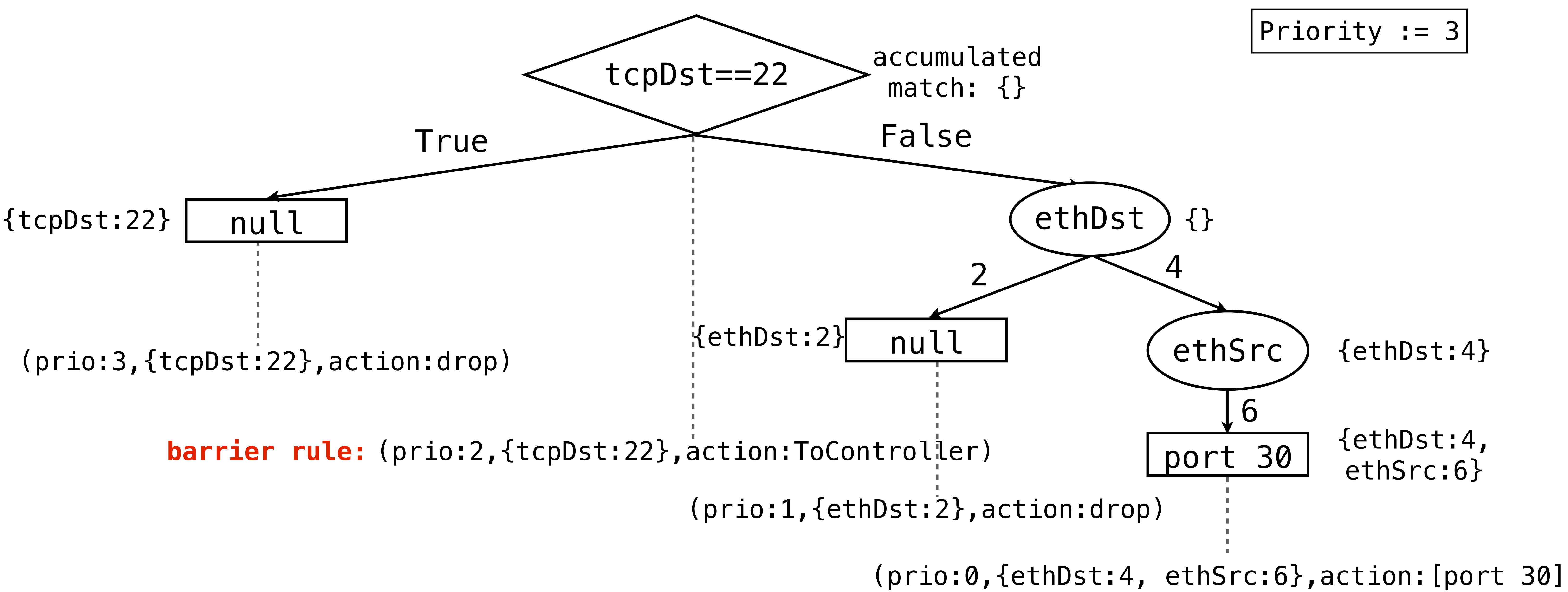
Basic compilation: in-order traversal & barrier rules



Basic compilation: in-order traversal & barrier rules



Basic compilation: in-order traversal & barrier rules



Basic compilation example result

```
(prio:3,{tcpDst:22},action:drop)
(prio:2,{tcpDst:22},action:ToController)
(prio:1,{ethDst:2},action:drop)
(prio:0,{ethDst:4, ethSrc:6},action:[port 30])
```

Basic compilation example result

```
(prio:3,{tcpDst:22},action:drop)
(prio:2,{tcpDst:22},action:ToController)
(prio:1,{ethDst:2},action:drop)
(prio:0,{ethDst:4, ethSrc:6},action:[port 30])
```

- Trace tree method converts arbitrary algorithmic policies into correct forwarding tables that effectively use wildcard rules.

Basic compilation example result

```
(prio:3,{tcpDst:22},action:drop)
(prio:2,{tcpDst:22},action:ToController)
(prio:1,{ethDst:2},action:drop)
(prio:0,{ethDst:4, ethSrc:6},action:[port 30])
```

- Trace tree method converts arbitrary algorithmic policies into correct forwarding tables that effectively use wildcard rules.
- Deficiencies:

Basic compilation example result

```
(prio:3,{tcpDst:22},action:drop)
(prio:2,{tcpDst:22},action:ToController)  <----  No effect
(prio:1,{ethDst:2},action:drop)
(prio:0,{ethDst:4, ethSrc:6},action:[port 30])
```

- Trace tree method converts arbitrary algorithmic policies into correct forwarding tables that effectively use wildcard rules.
- Deficiencies:
  - More rules than necessary.

Basic compilation example result

```
(prio:3,{tcpDst:22},action:drop)
(prio:2,{tcpDst:22},action:ToController)
(prio:1,{ethDst:2},action:drop)
(prio:0,{ethDst:4, ethSrc:6},action:[port 30])
```

No effect (→ prio:2 rule)

Can use priority 0 (→ prio:1 rule)

- Trace tree method converts arbitrary algorithmic policies into correct forwarding tables that effectively use wildcard rules.
- Deficiencies:
  - More rules than necessary.
  - More priorities levels than necessary.

Basic compilation example result

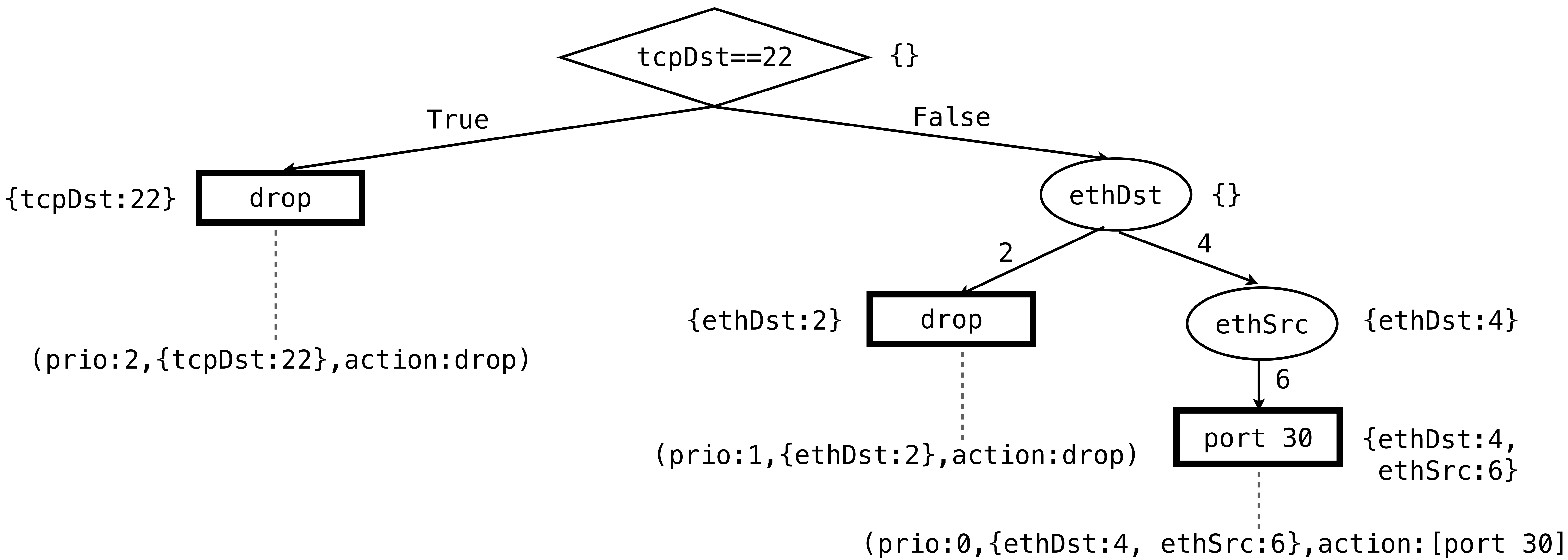
```
(prio:3,{tcpDst:22},action:drop)
(prio:2,{tcpDst:22},action:ToController)
(prio:1,{ethDst:2},action:drop)
(prio:0,{ethDst:4, ethSrc:6},action:[port 30])
```

No effect → (prio:2,{tcpDst:22},action:ToController)

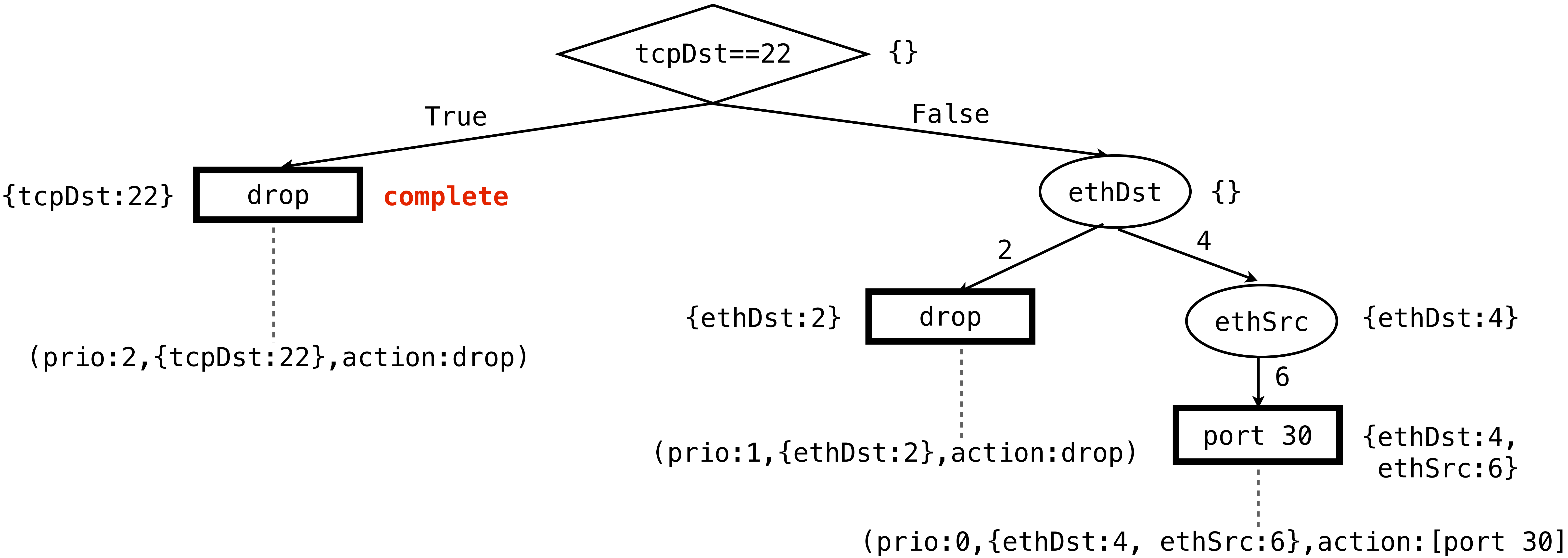
Can use priority 0 → (prio:1,{ethDst:2},action:drop)

- Trace tree method converts arbitrary algorithmic policies into correct forwarding tables that effectively use wildcard rules.
- Deficiencies:
  - More rules than necessary.
  - More priorities levels than necessary.
- We **annotate TT nodes** with extra information to improve compilation.

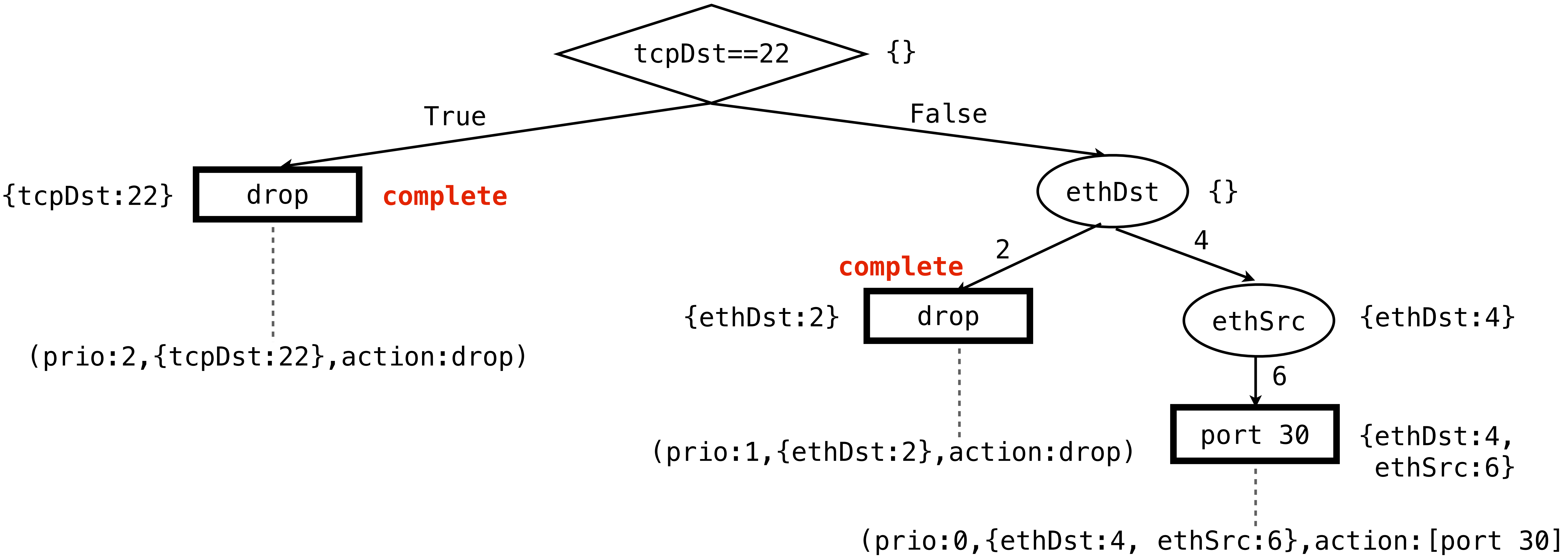
Optimization 1: Annotate TT nodes with completeness



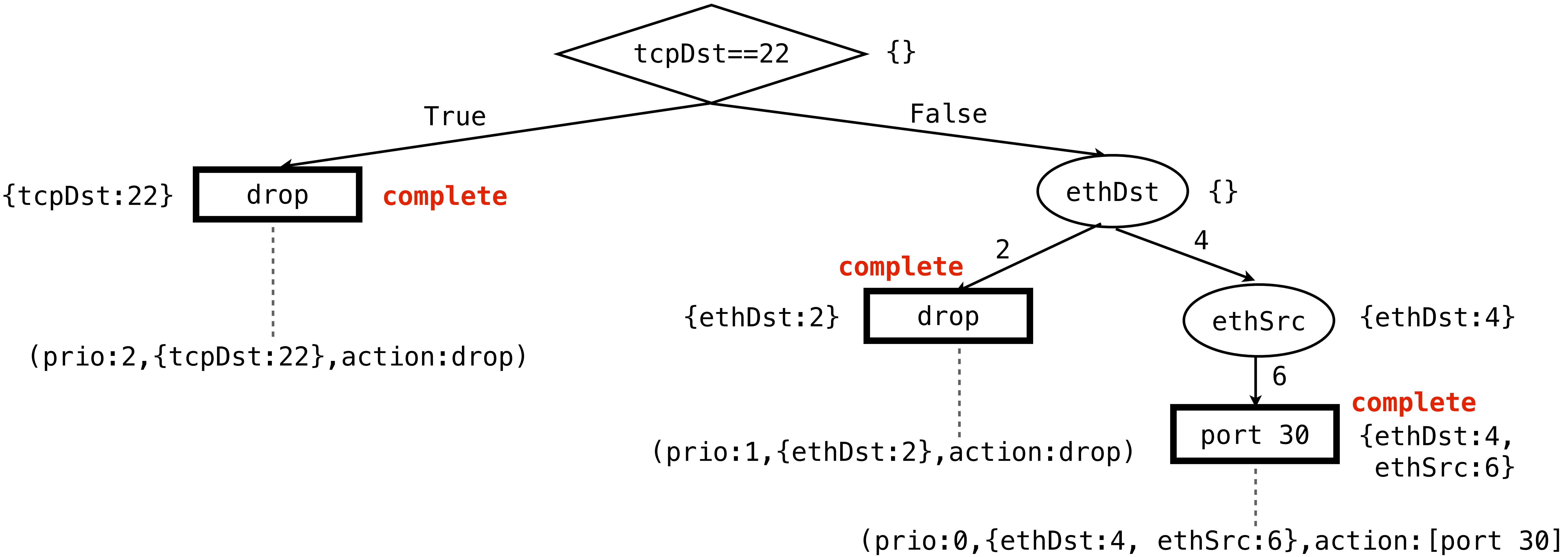
Optimization 1: Annotate TT nodes with completeness



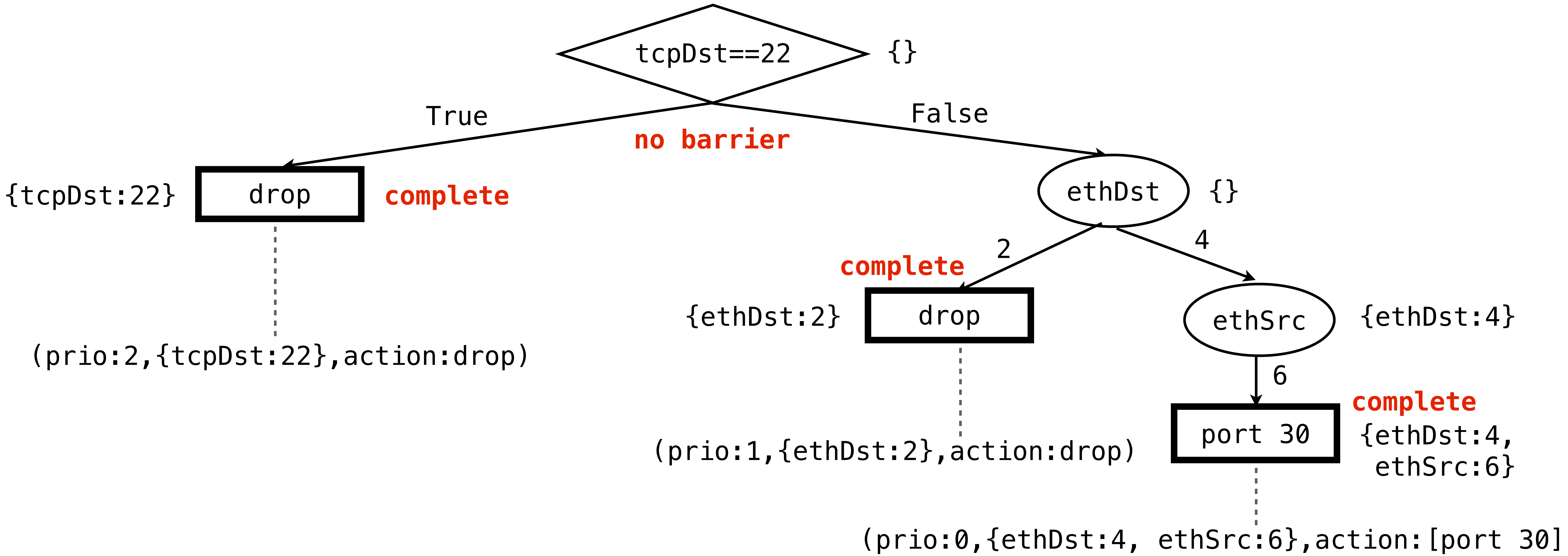
Optimization 1: Annotate TT nodes with completeness



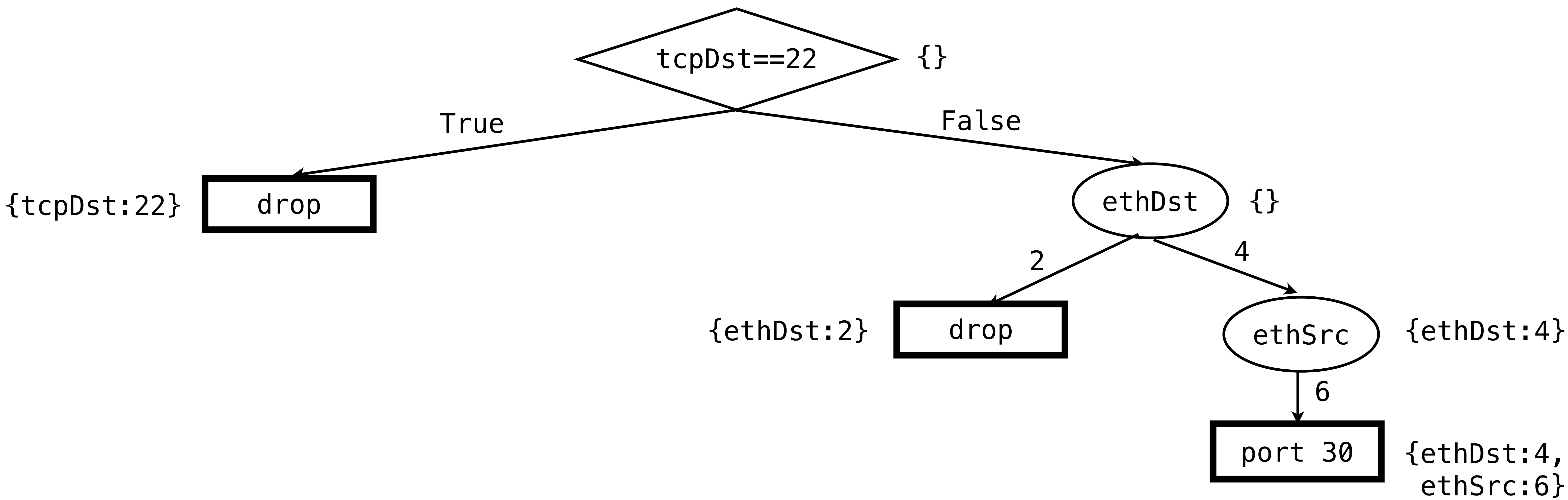
Optimization 1: Annotate TT nodes with completeness



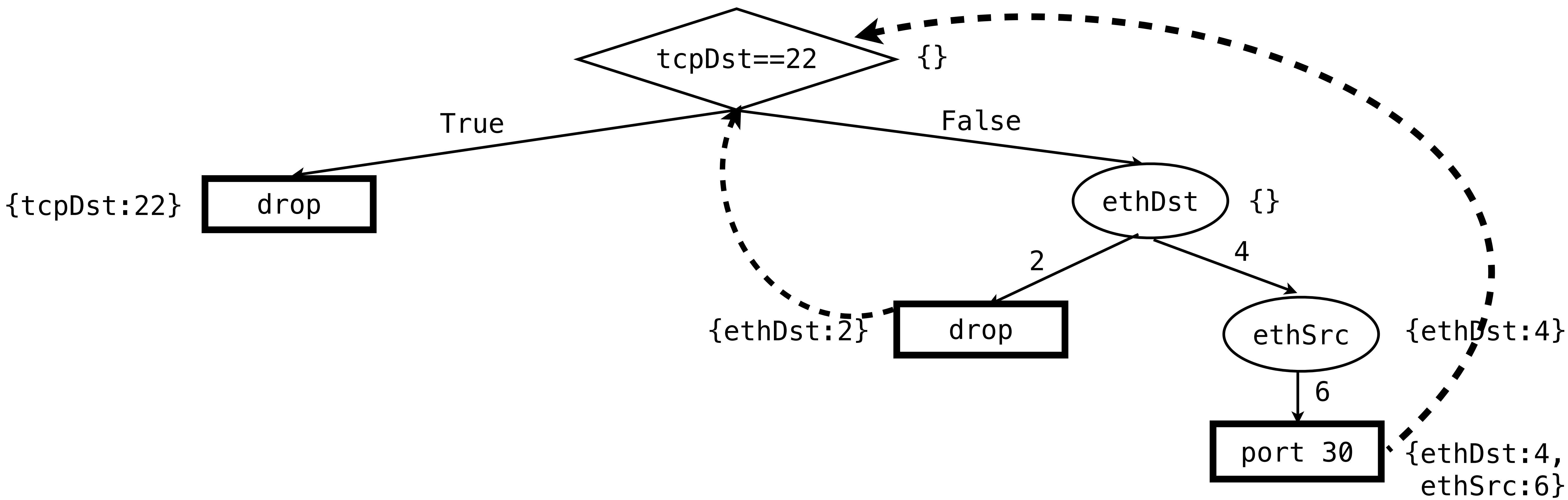
Optimization 1: Annotate TT nodes with completeness



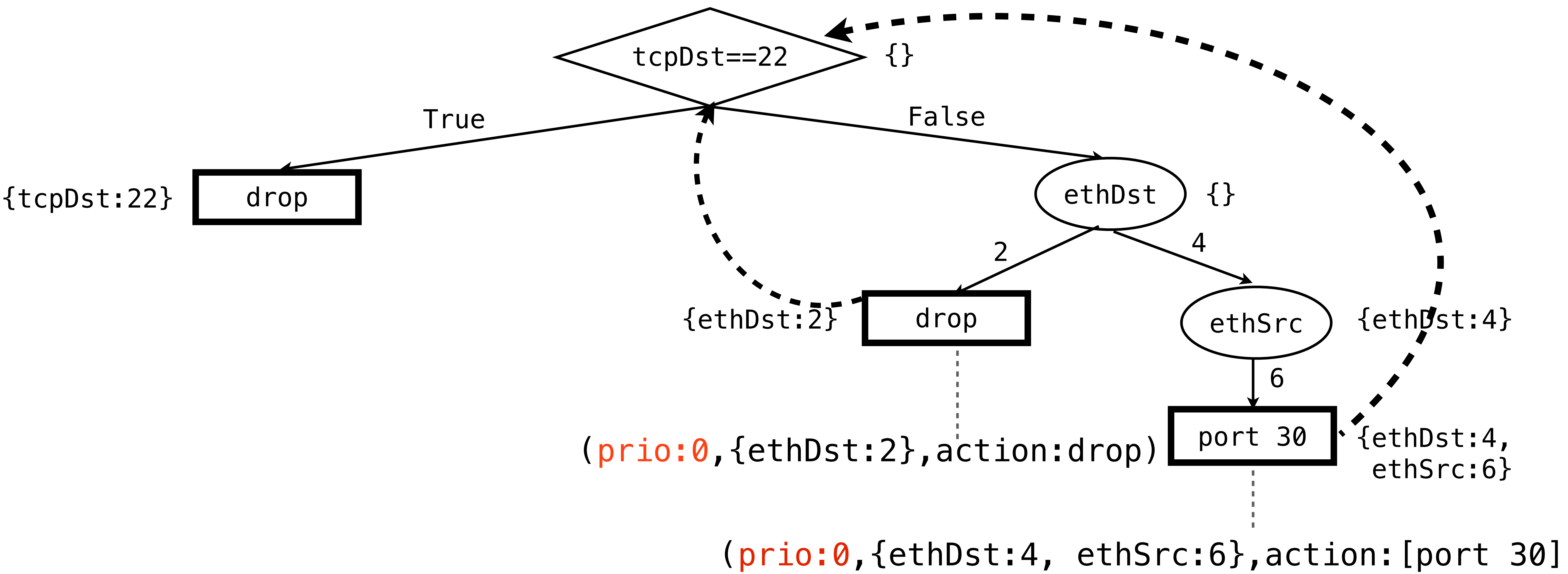
Optimization 2: Annotate nodes with priority dependencies



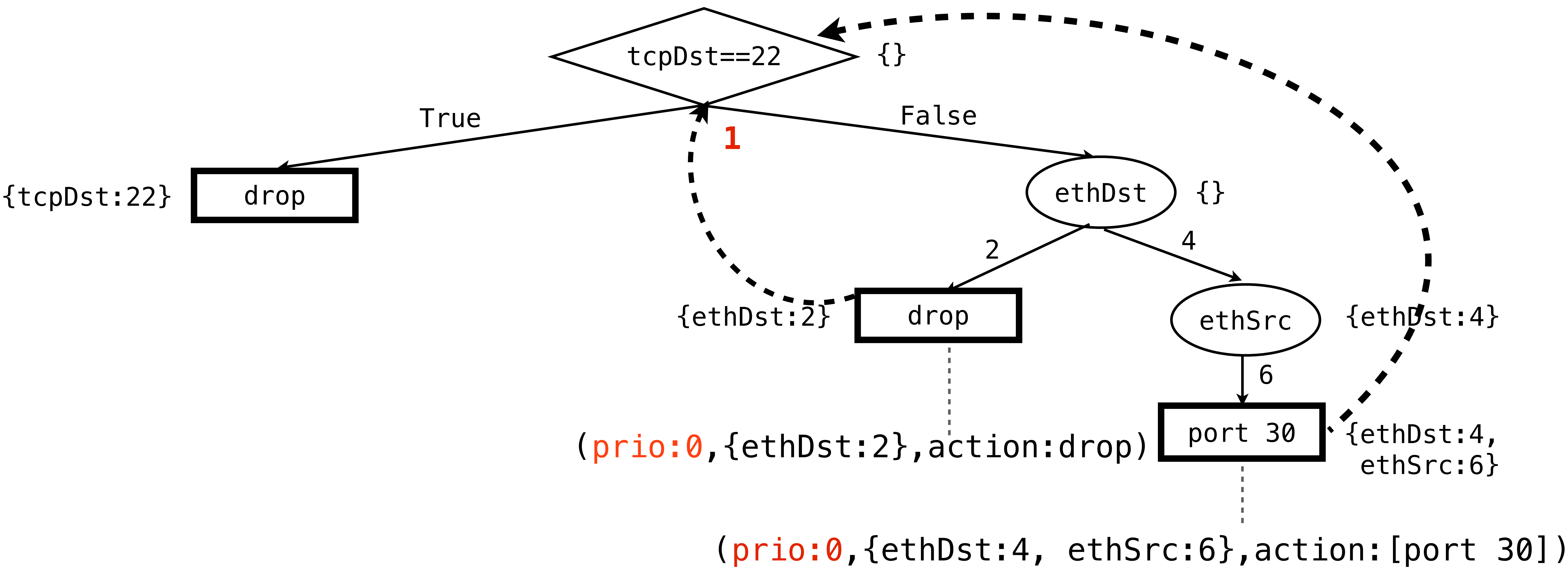
Optimization 2: Annotate nodes with priority dependencies



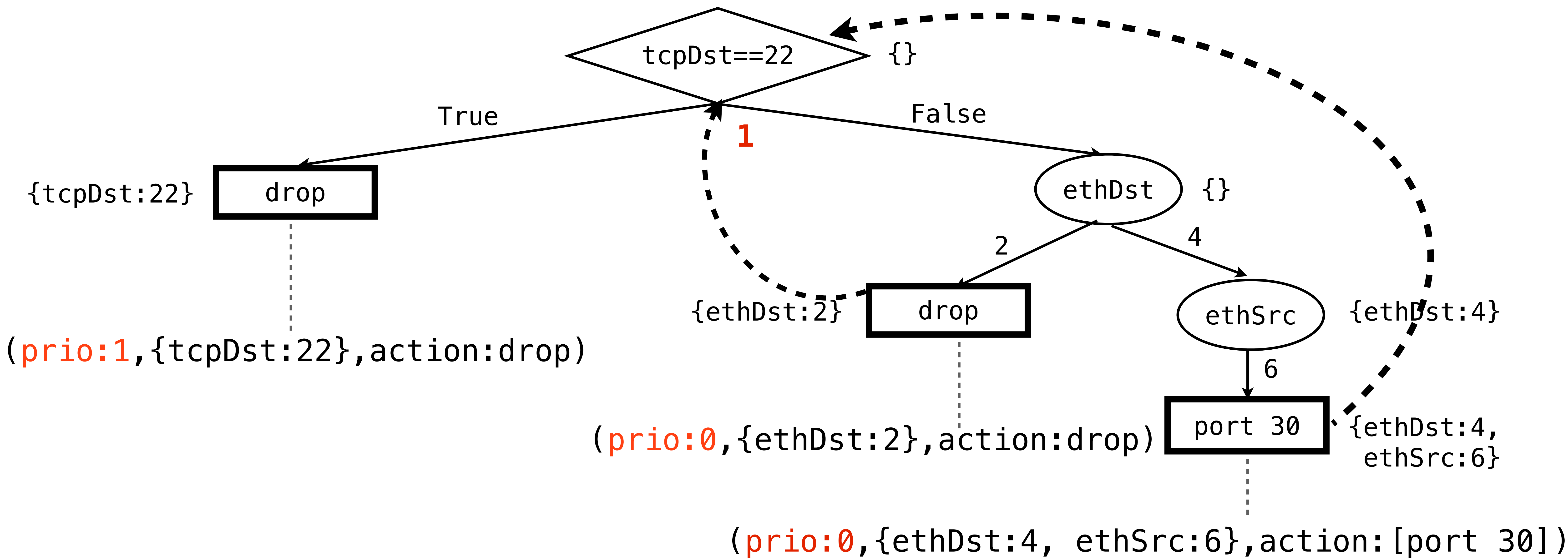
Optimization 2: Annotate nodes with priority dependencies



Optimization 2: Annotate nodes with priority dependencies



Optimization 2: Annotate nodes with priority dependencies



Improved compilation result

```
(prio:1,{tcpDst:22},action:drop)
(prio:0,{ethDst:2},action:drop)
(prio:0,{ethSrc:4, ethSrc:6},action:[port 30])
```

Maple Status

- Maple has been implemented in Haskell using the McNettle Openflow controller, which implements Openflow 1.0.

- The implementation includes several other features:

  - **Incremental TT compilation**, to avoid full recompilation on every update.

  - **Trace reduction**, to ensure traces and trace trees do not contain redundant nodes.

  - **Automatic and user-specified invalidation**, to support removing and updating TT and FT when network state changes.

Maple use case: ACL compiler

Maple use case: ACL compiler

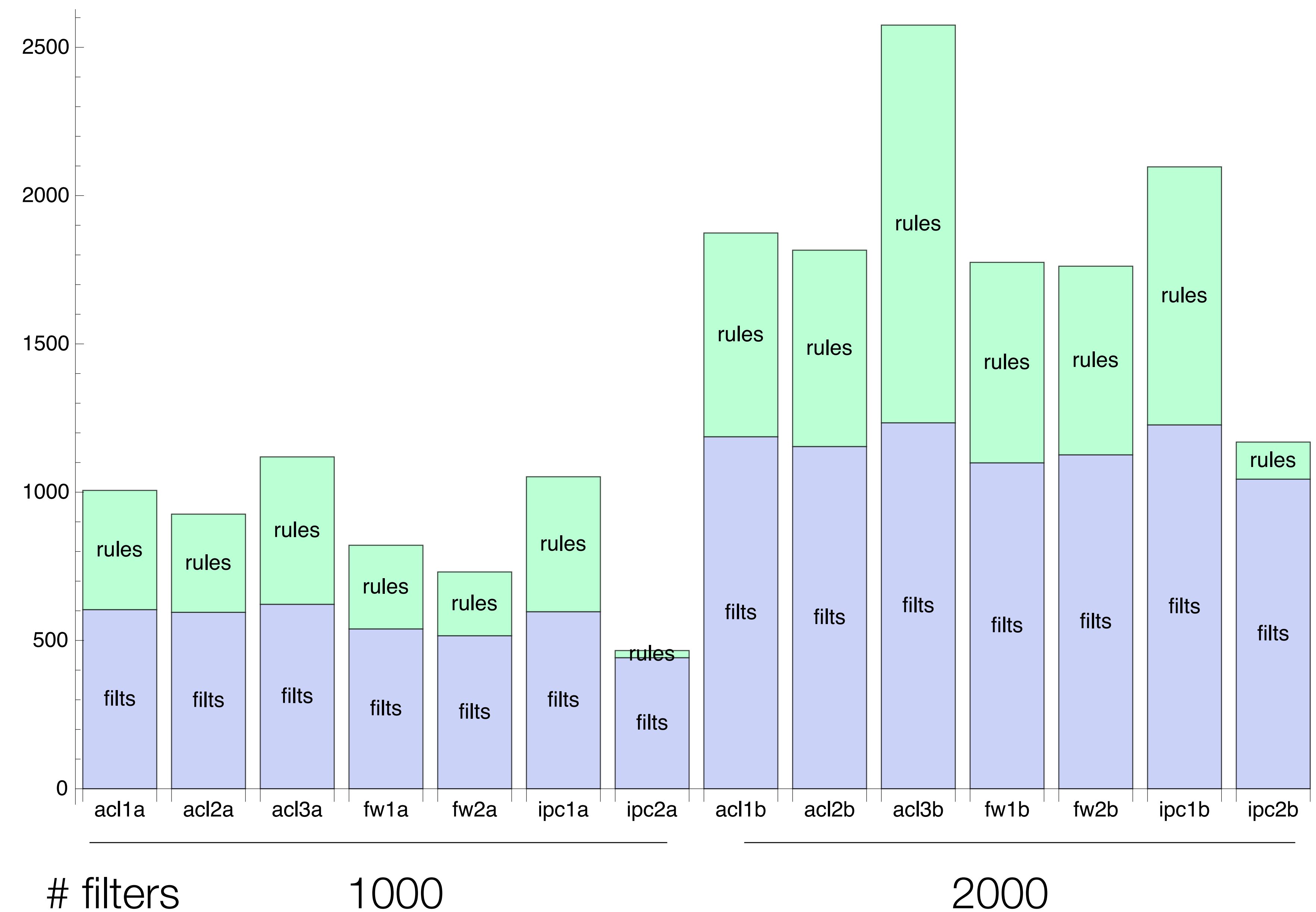
- Many networks use filter sets, e.g. for access control, to check IP addresses and port ranges.
  - How can we implement these using Openflow flow tables?
  - How can we combine filter sets with other policies, e.g. routing, load balancing?

Maple use case: ACL compiler

- Many networks use filter sets, e.g. for access control, to check IP addresses and port ranges.
  - How can we implement these using Openflow flow tables?
  - How can we combine filter sets with other policies, e.g. routing, load balancing?
- We implement a **filter set interpreter** as an algorithmic policy in Maple:
  - Loads a filter set from a file at startup.
  - When given a packet, `f` iterates over filters in filter set, testing if packet satisfies filter; return action based on first matching filter.

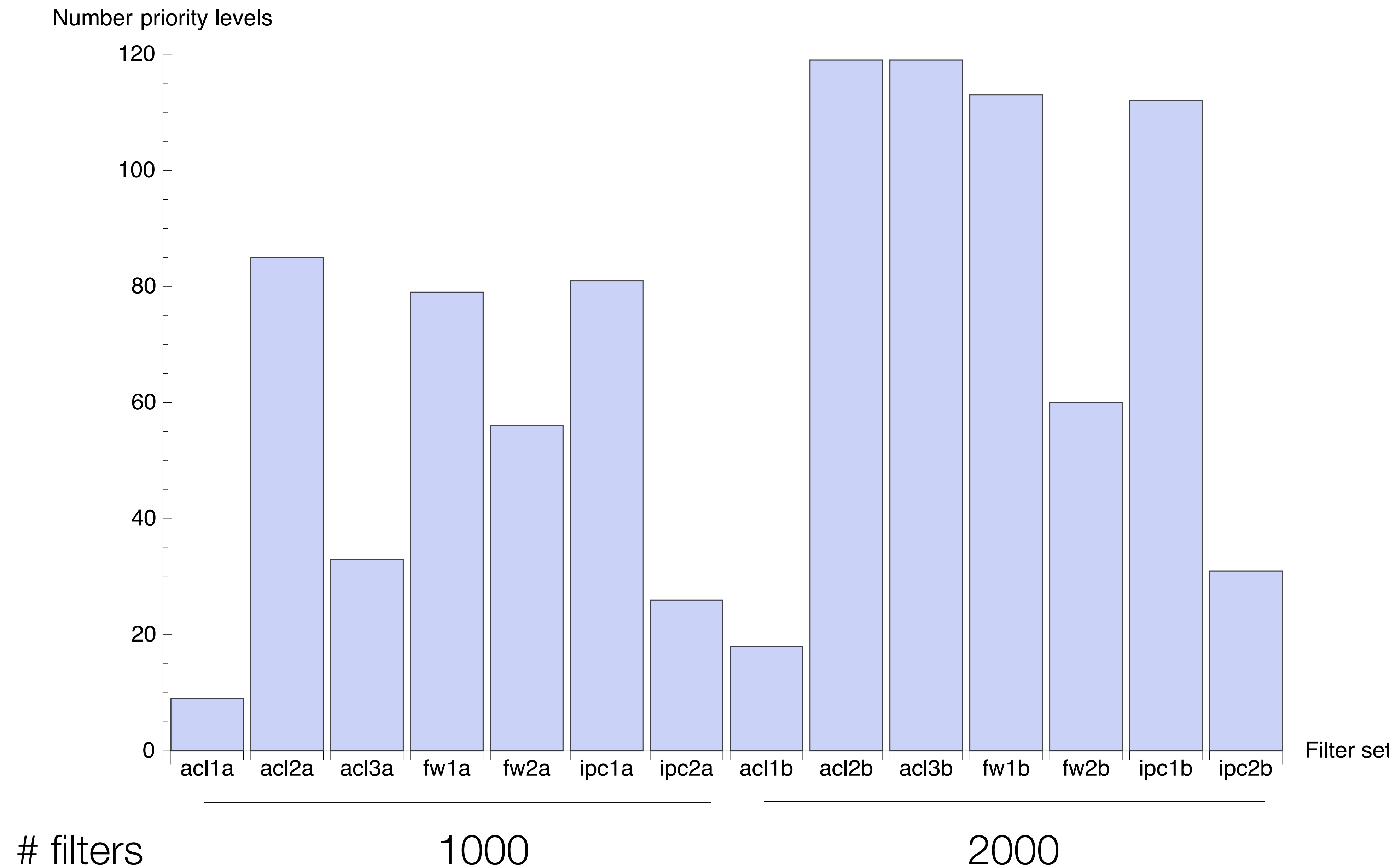
Maple generates compact flow tables

- Classbench filter sets and packet traces for 3 filter set types - ACL, FW, IPC - and two sizes - 1000 and 2000 filters per filter set.
- blue = # filters activated by packets in trace; green + blue = # rules generated by Maple for packet trace.
- More generated rules expected, since TCP port ranges expanded to exact matches.



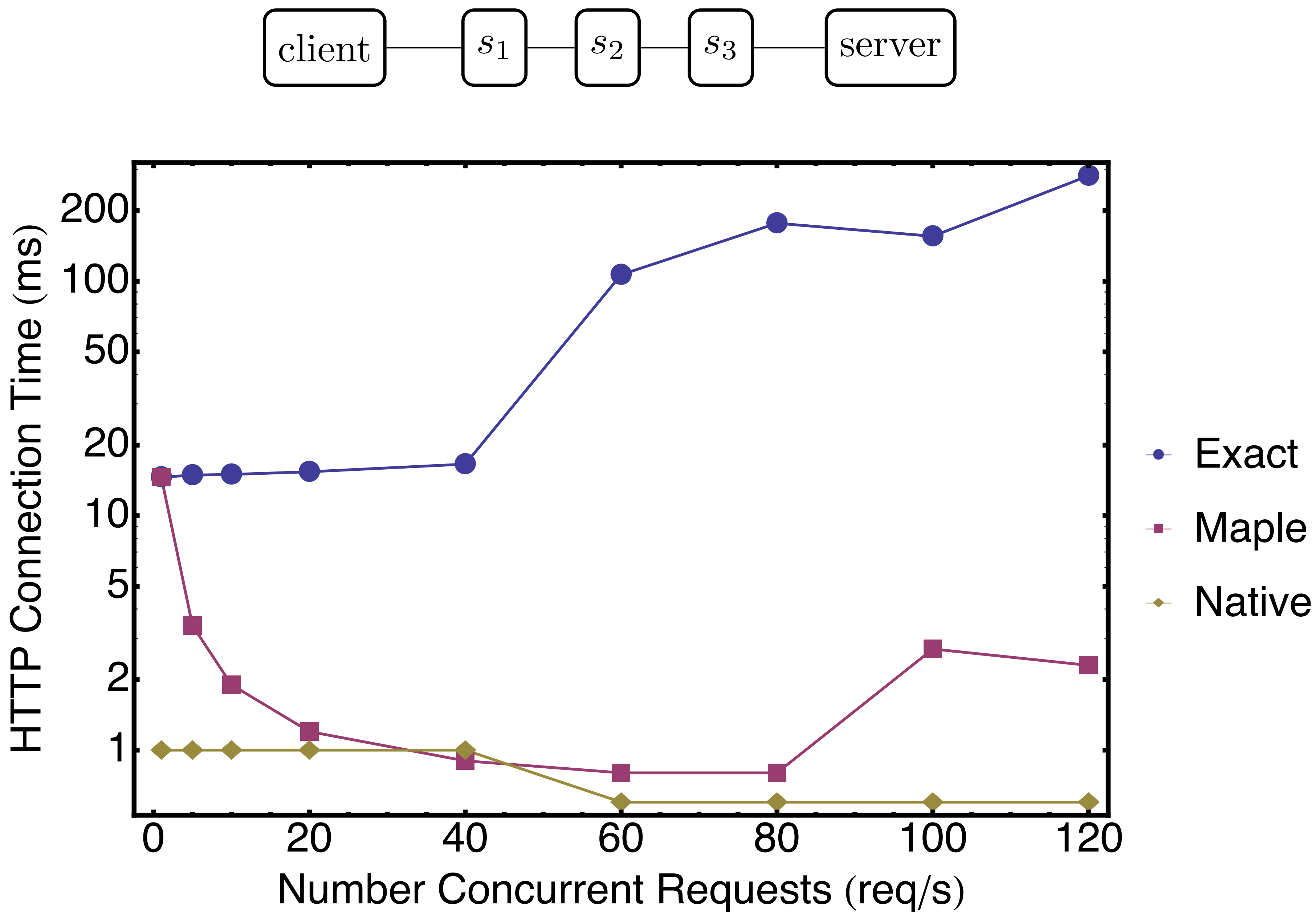
Maple uses few priority levels

- blue = # priority levels used by Maple-generated rules.
- **acl1a**: 9 priorities for nearly 1000 rules.
- **acl2b**: 120 priorities used for 2000 rules.



Maple reduces HTTP connection time

- 3 real HP 5406 switches, client on one side, server on the other.
- Client performs X HTTP reqs/sec. using httperf.
- Measure average HTTP connection time as X varies.
- Compare exact & maple
  - exact: up to 282 ms average
  - maple: 1-2 ms: 100x reduction.



Related Work

- FML, FSL: logic-based policy languages.
- Frenetic family of languages: Frenetic, NetCore, Pyretic offer more declarative approaches.
- Onix: introduces NIB abstraction to abstract distributed flow tables.
- Openflow controllers: Maestro, Beacon, NOX, NOX-MT, POX (and many more).

Summary: Contributions

- Algorithmic policies provide a simple, expressive programming model for SDN, eliminating a key source of errors and performance problems.
- Maple provides a scalable implementation of algorithmic policies through several novel techniques, including:
    - runtime tracing of algorithmic policies,
    - maintaining a trace tree and compiling TT to flow tables to distribute processing to switches;
    - using TT annotations to implement compiler optimizations such as rule and priority reductions.