libpcap: An Architecture and Optimization Methodology for Packet Capture

Sharkfest '11

Steve McCanne, CTO

Riverbed Technology

CS 164

- My story begins with a U.C. Berkeley course
  - Back in spring 1988, I took the the compilers course in computer science at U.C. Berkeley
  - Taught by a guest lecturer from LBL
    - Van Jacobson
  - Learned standard compiler topics
    - scanning, parsing, code generation, optimization
  - Took summer job in Van's group at end of term

LBL

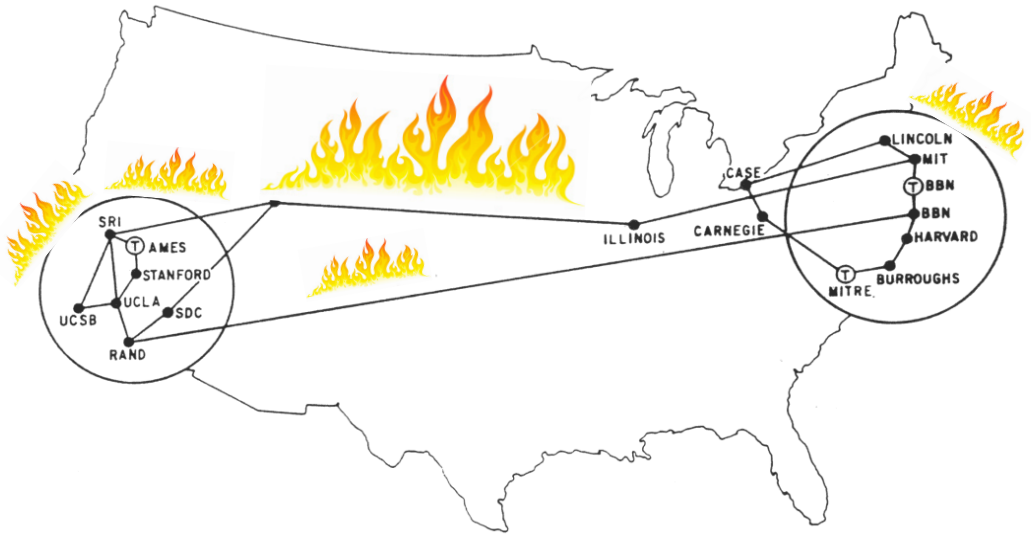
- It was a great time and place in Internet history
  - Summer job evolved into staff scientist position
- "Network Research Group"
  - Van Jacobson
  - Sally Floyd
  - Vern Paxson
  - Steve McCanne
- Lucky to be surrounded by such creative intellect

LBL Network Research Group

- flex
- TCP congestion control
- VJ header compression (CSLIP)
- BSD packet filter (BPF)
- tcpdump, pcap
- traceroute, pathchar
- BRO
- SDP/SIP
- VoIP (RTP)
- Mbone tools (vic, vat, wb)
- Scalable reliable multicast (SRM)
- ns - network simulator
- Class-based queuing (CBQ)
- Random early drop (RED)
- diffserv

Congestion Control

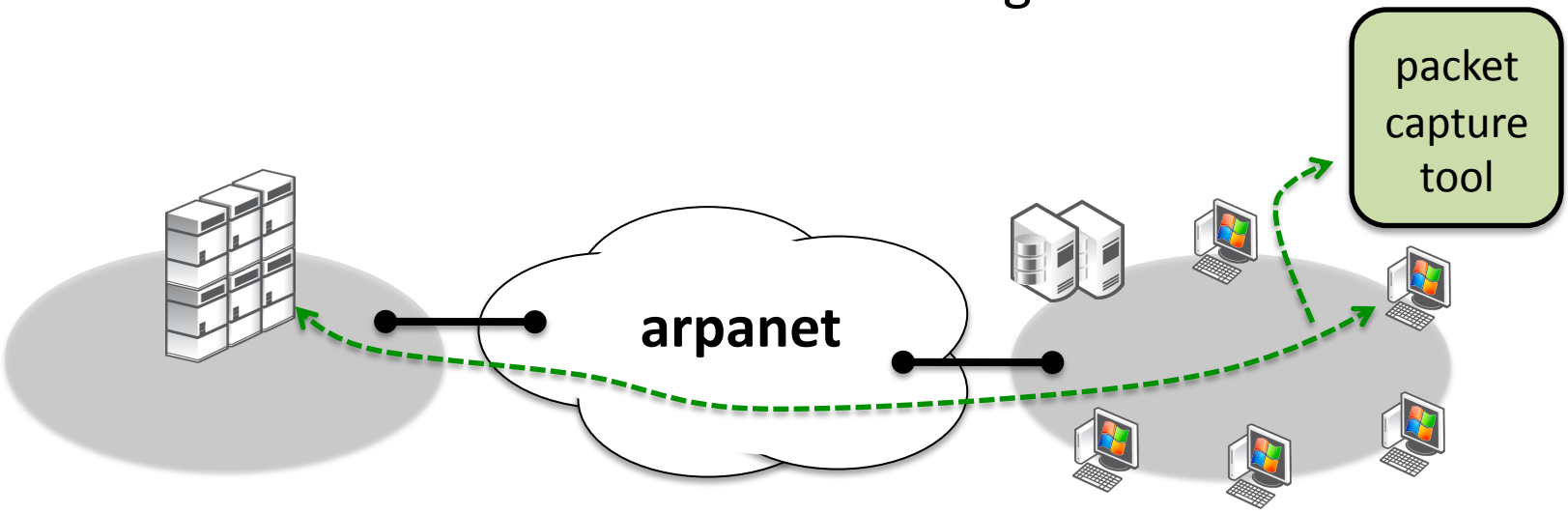
- When I first joined, Van was wrapping up his work on TCP congestion control
  - He had figured out why the Arpanet kept collapsing...



MAP 4 September 1971

Packet Capture

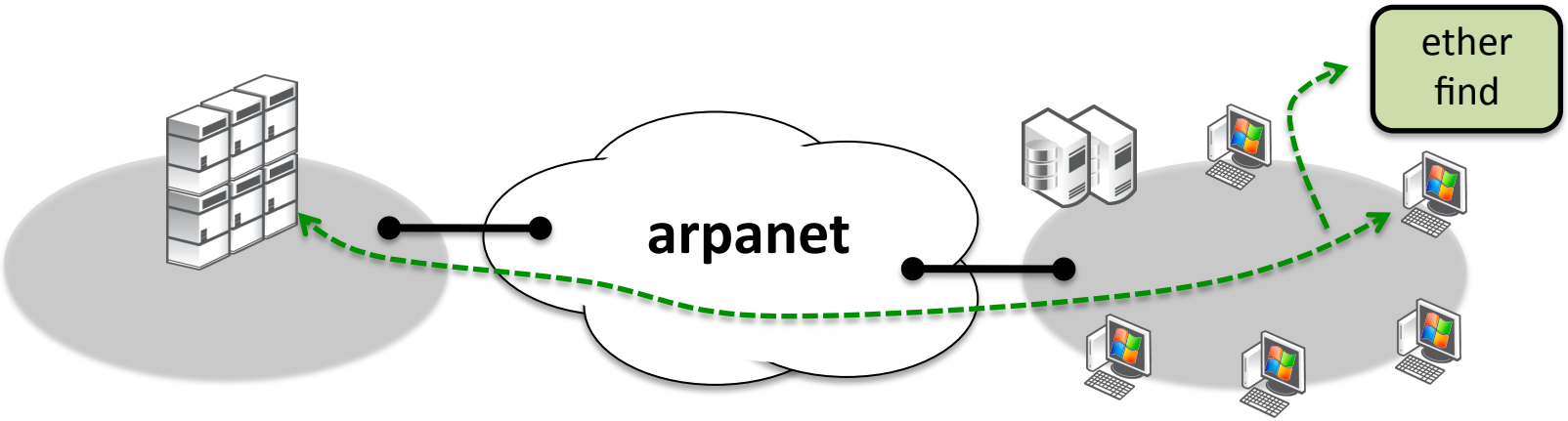
- Van needed to look at packet traces
  - to understand the problem
  - to experiment with fixes
  - to see that the solution was working



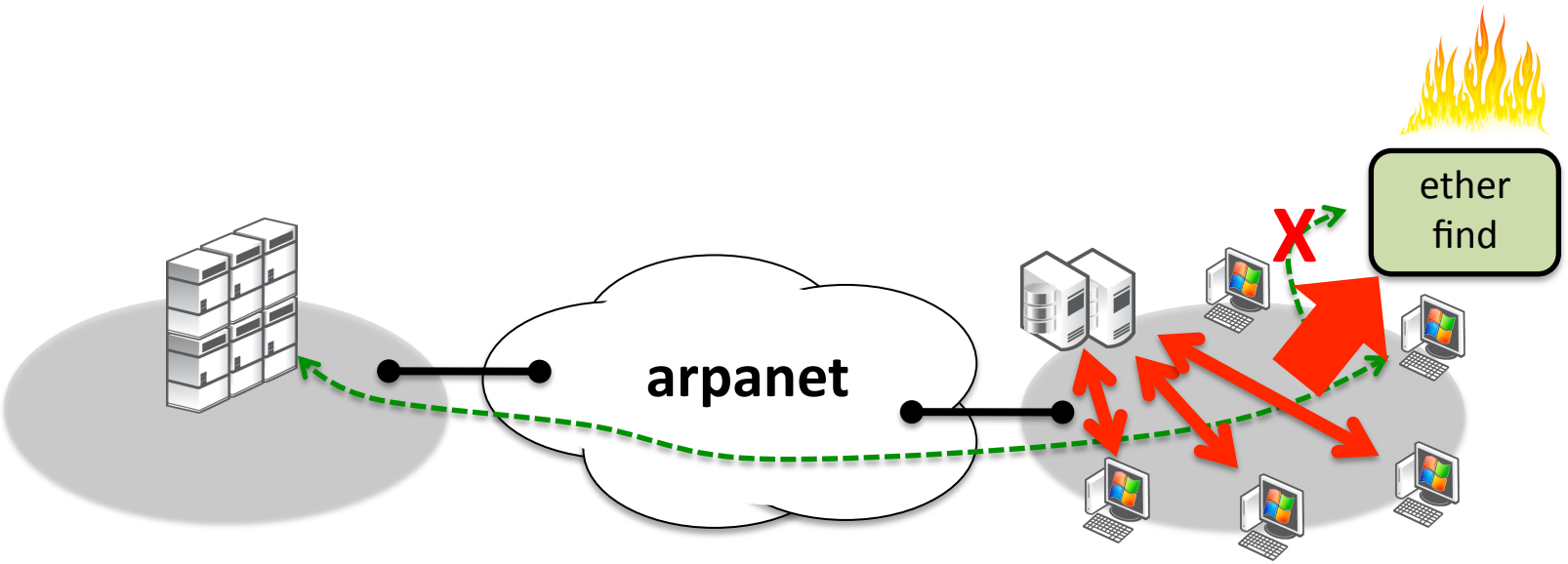
etherfind

- Frustrated with Sun's packet capture tool
  - "etherfind" based on Unix "find" command
- Several problems
  - Clumsy filtering syntax
    - How many of you do "find . | grep ..." instead?
  - Protocol decoding was weak and cryptic
  - Horrible performance

The LAN Bottleneck



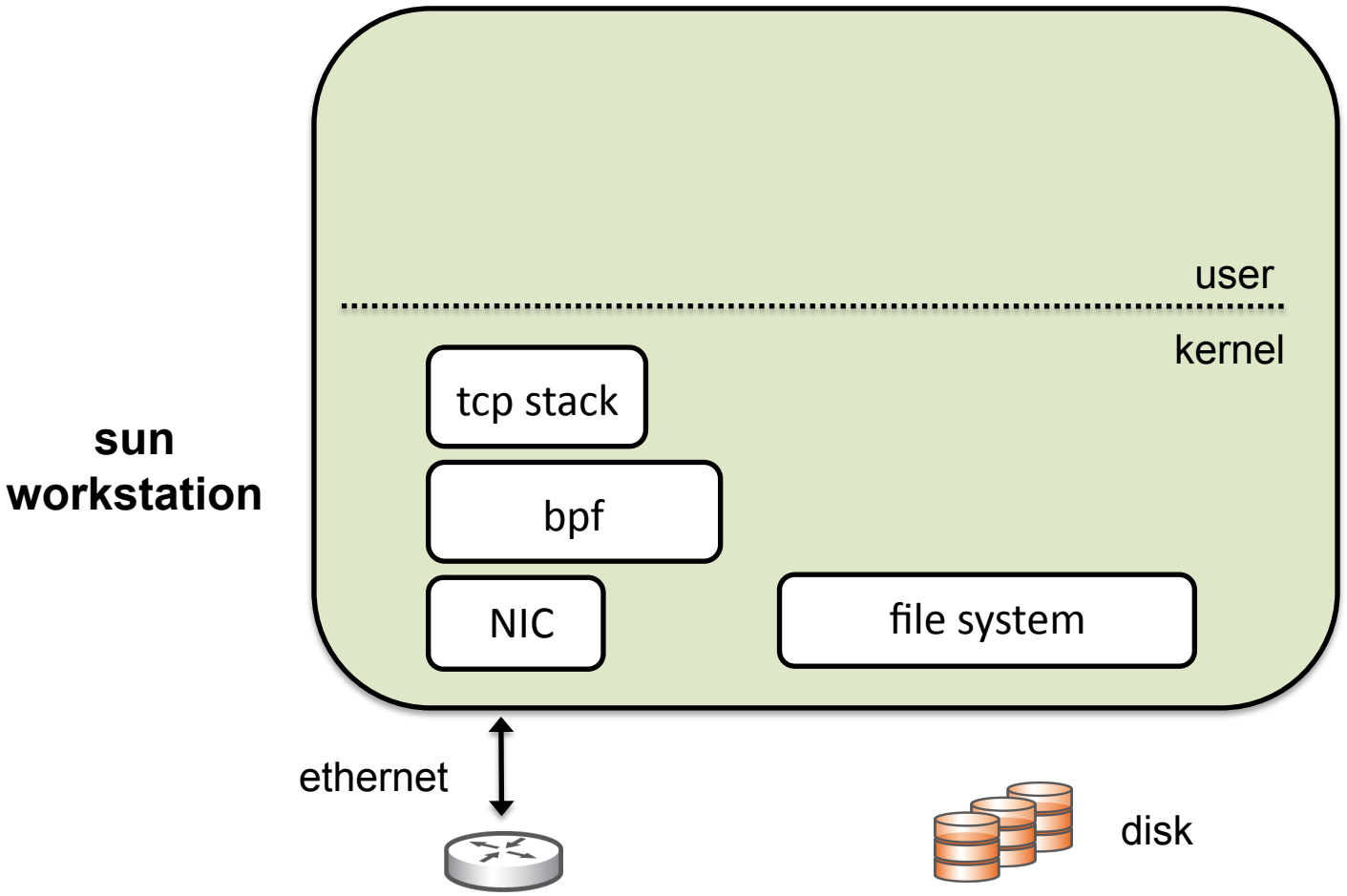
The LAN Bottleneck



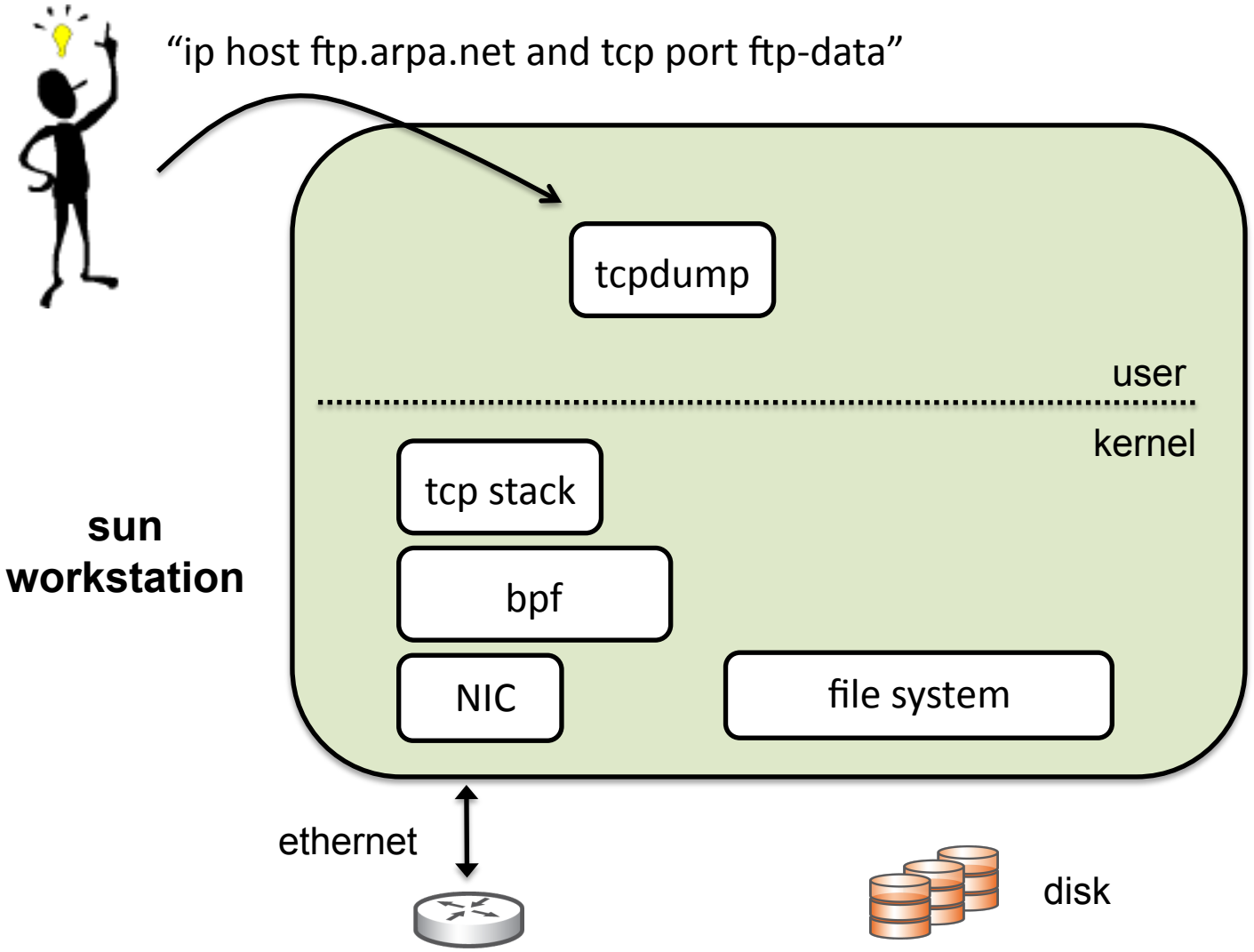
Enter tcpdump

- There must be a better way…
  - Set out to work on a new model in a tool called tcpdump
  - "Filter" packets before they come up the stack
    - Inspired by Jeff Mogul's prior work on "enet"
  - Compile high-level filter specification into low-level code that filters packets at driver level
  - Kernel module called Berkeley Packet Filter (BPF)

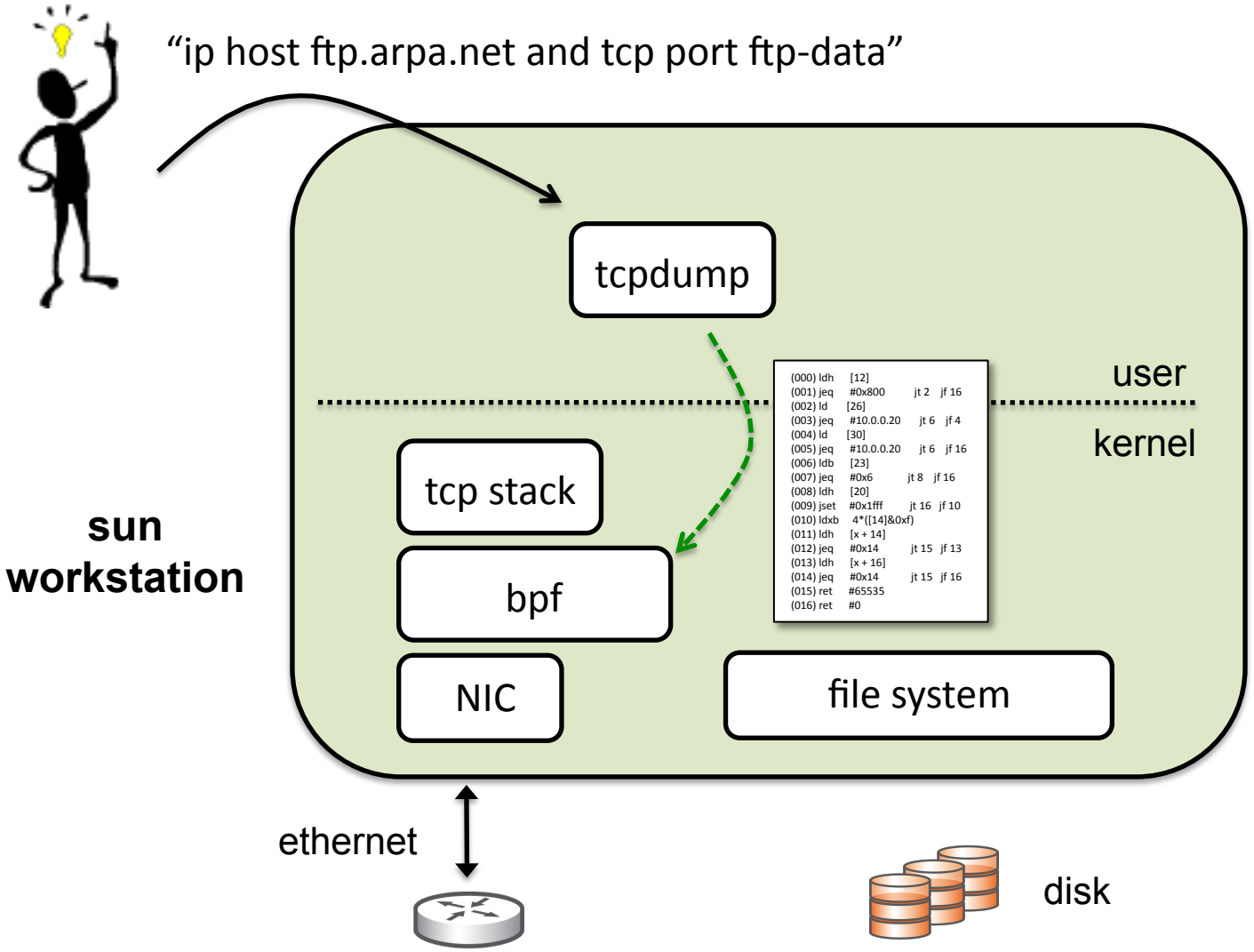
tcpdump


user
kernel
tcp stack
bpf
NIC
file system
sun
workstation
ethernet
disk

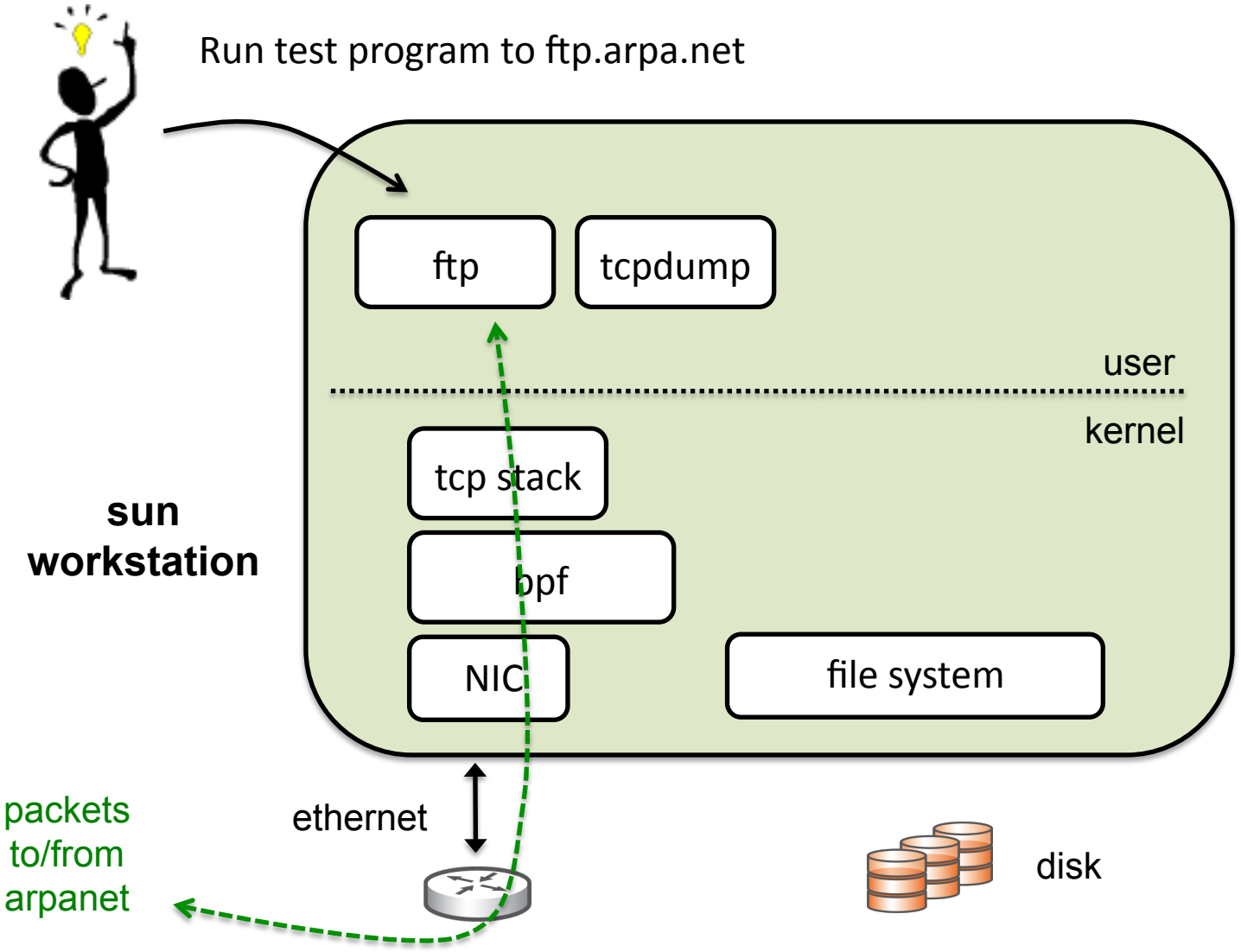
tcpdump


"ip host ftp.arpa.net and tcp port ftp-data"
tcpdump
user
kernel
tcp stack
bpf
NIC
file system
sun
workstation
ethernet
disk

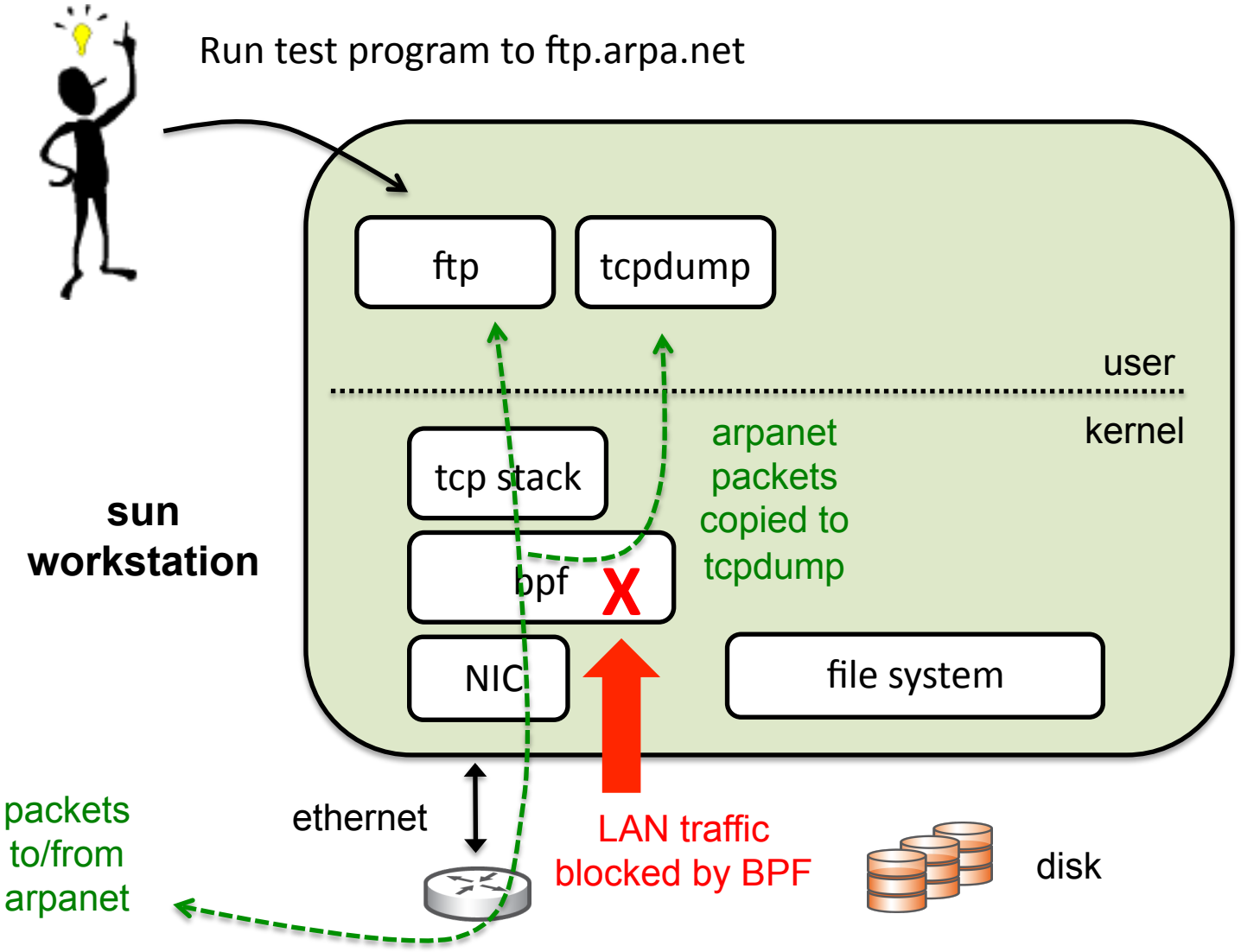
tcpdump


"ip host ftp.arpa.net and tcp port ftp-data"
tcpdump
user
kernel
(000) ldh [12]
(001) jeq #0x800 jt 2 jf 16
(002) ld [26]
(003) jeq #10.0.0.20 jt 6 jf 4
(004) ld [30]
(005) jeq #10.0.0.20 jt 6 jf 16
(006) ldb [23]
(007) jeq #0x6 jt 8 jf 16
(008) ldh [20]
(009) jset #0x1fff jt 16 jf 10
(010) ldxb 4*([14]&0xf)
(011) ldh [x + 14]
(012) jeq #0x14 jt 15 jf 13
(013) ldh [x + 16]
(014) jeq #0x14 jt 15 jf 16
(015) ret #65535
(016) ret #0
tcp stack
bpf
NIC
file system
sun workstation
ethernet
disk

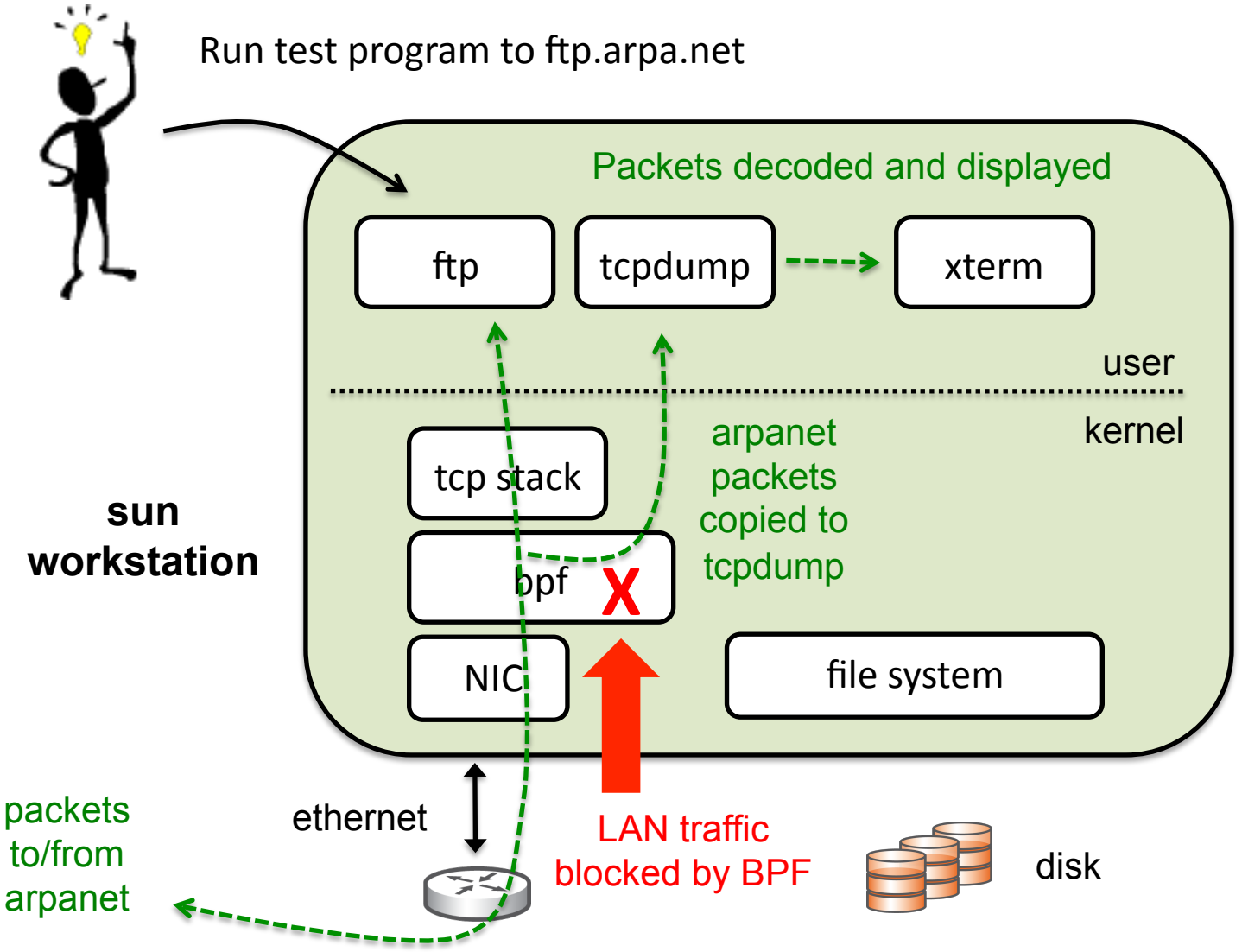
tcpdump


Run test program to ftp.arpa.net
ftp
tcpdump
user
kernel
tcp stack
bpf
NIC
file system
sun
workstation
packets
to/from
arpanet
ethernet
disk

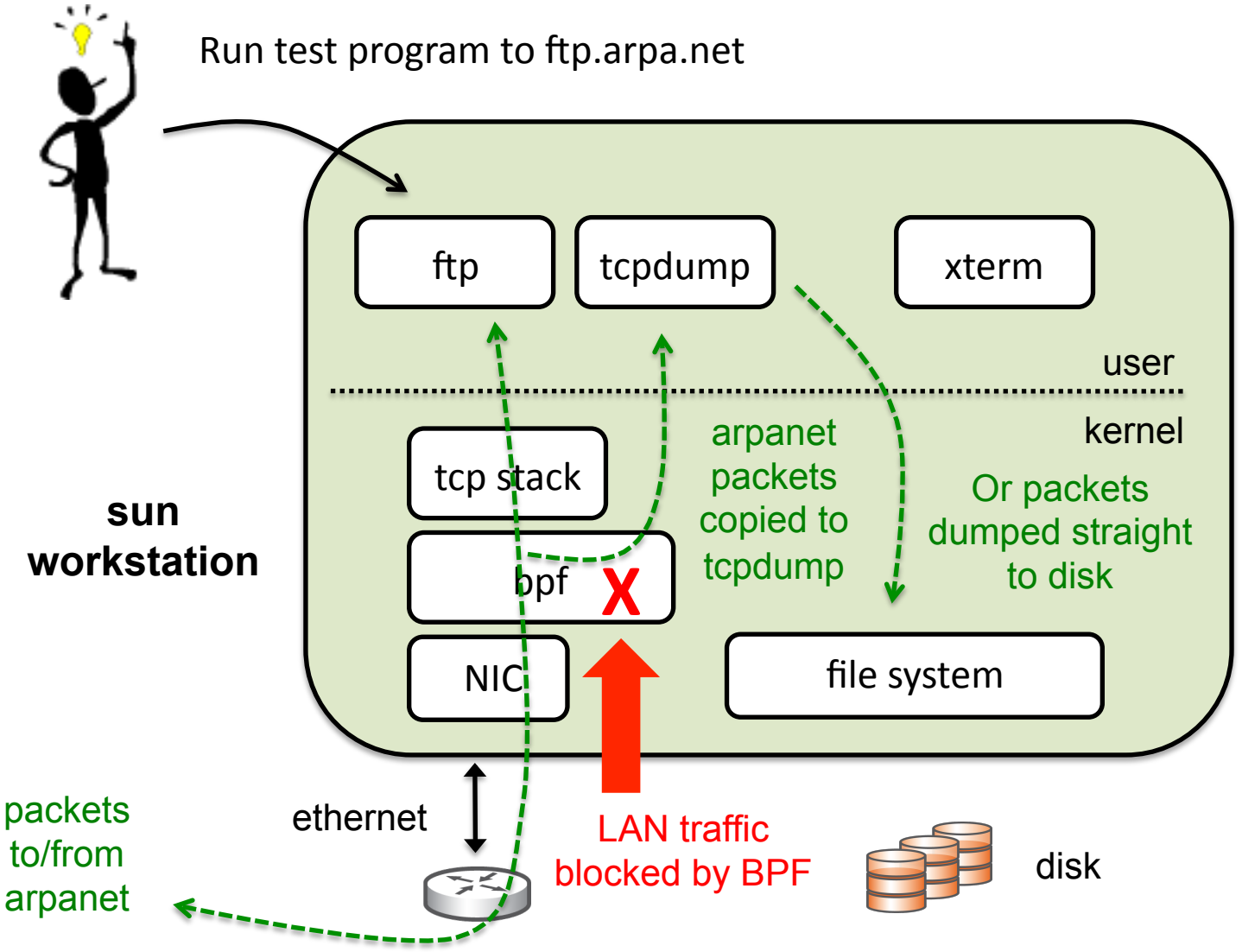
tcpdump


Run test program to ftp.arpa.net
ftp
tcpdump
user
kernel
tcp stack
bpf
X
NIC
file system
arpanet
packets
copied to
tcpdump
sun
workstation
ethernet
LAN traffic
blocked by BPF
disk
packets
to/from
arpanet

tcpdump


Run test program to ftp.arpa.net
Packets decoded and displayed
ftp
tcpdump
xterm
user
kernel
tcp stack
bpf
X
NIC
file system
arpanet
packets
copied to
tcpdump
sun
workstation
packets
to/from
arpanet
ethernet
LAN traffic
blocked by BPF
disk

tcpdump


Run test program to ftp.arpa.net
ftp
tcpdump
xterm
user
kernel
tcp stack
bpf
X
NIC
file system
sun
workstation
arpanet
packets
copied to
tcpdump
Or packets
dumped straight
to disk
packets
to/from
arpanet
ethernet
LAN traffic
blocked by BPF
disk

The BPF virtual machine

- First thing, I had to design a VM model that would run in the kernel
- Came up with a virtual machine architecture and set of machine instructions
  - Knew Apple II from my junior high days
  - Modeled after Motorola 6502
  - Accumulator (A), index register (X)
  - Packet-based memory model
  - Arithmetic and conditional logic

Example

FTP data packets
for host 10.0.0.20

```
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20       jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20       jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6             jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff          jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14            jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14            jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

Example

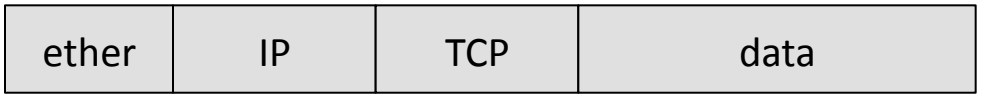
FTP data packets
for host 10.0.0.20

Is ethernet type IP?

```
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20       jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20       jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6             jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff          jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14            jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14            jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

A: X:

packet



Example

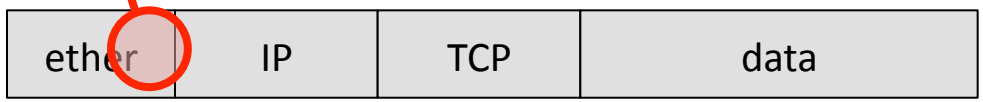
FTP data packets
for host 10.0.0.20

```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20      jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20      jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14           jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14           jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

Is ethernet type IP?

A: 0x800 X:

packet



Example

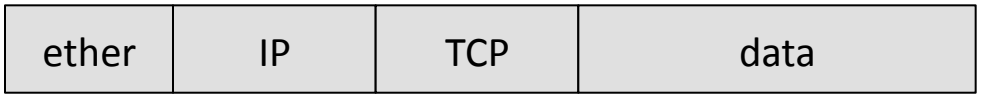
FTP data packets
for host 10.0.0.20

```
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20       jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20       jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6             jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff          jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14            jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14            jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

Is IP src address 10.0.0.20?

A: X:

packet



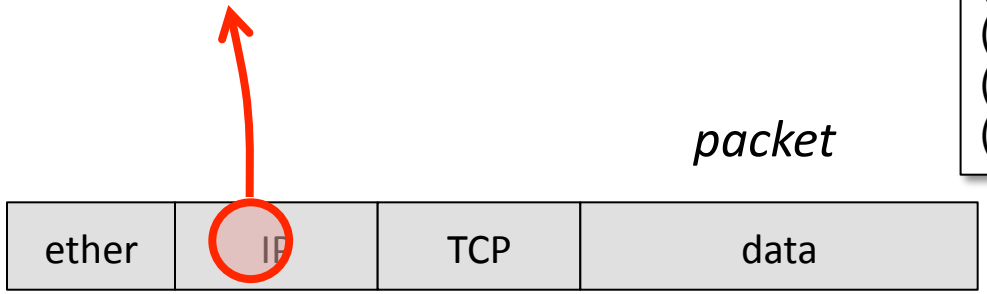
Example

FTP data packets
for host 10.0.0.20

```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20      jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20      jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14           jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14           jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

Is IP src address 10.0.0.20?

A: 10.0.1.11 X:



Example

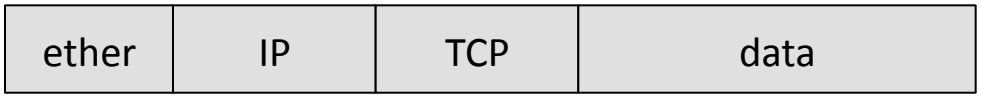
FTP data packets
for host 10.0.0.20

Is IP dst address 10.0.0.20?

```
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20       jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20       jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6             jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff          jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14            jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14            jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

A: X:

packet



Example

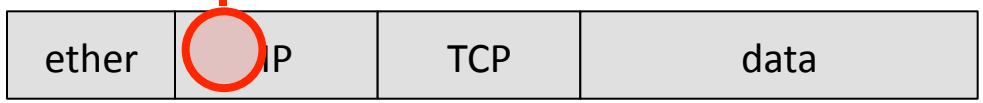
FTP data packets
for host 10.0.0.20

Is IP dst address 10.0.0.20?

```
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20       jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20       jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6             jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff          jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14            jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14            jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

A: 10.0.0.20 X:

packet



Example

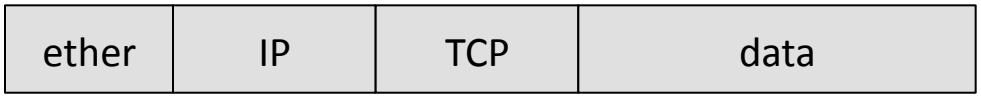
FTP data packets
for host 10.0.0.20

Is IP protocol TCP?

```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20      jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20      jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14           jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14           jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

A: X:

packet

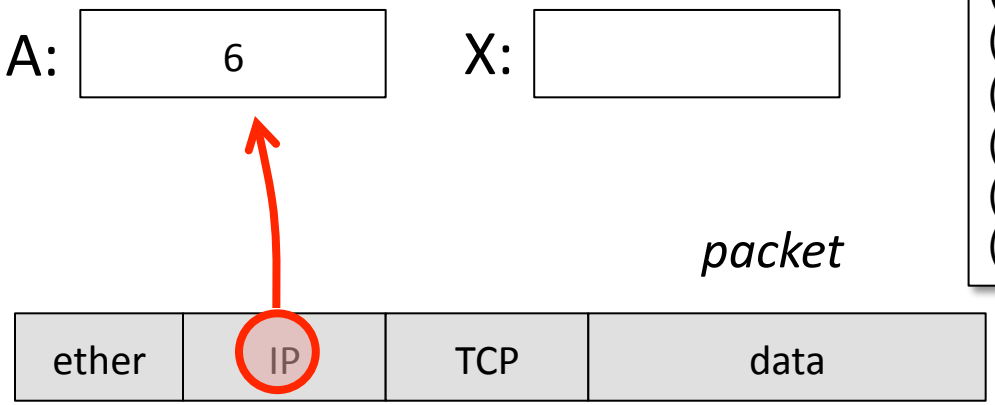


Example

FTP data packets
for host 10.0.0.20

Is IP protocol TCP?

```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20      jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20      jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14           jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14           jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```



Example

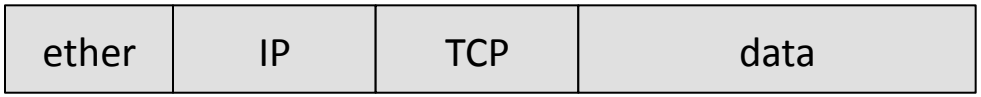
FTP data packets
for host 10.0.0.20

Is it first or only frag?

```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20      jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20      jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14           jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14           jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

A: X:

packet

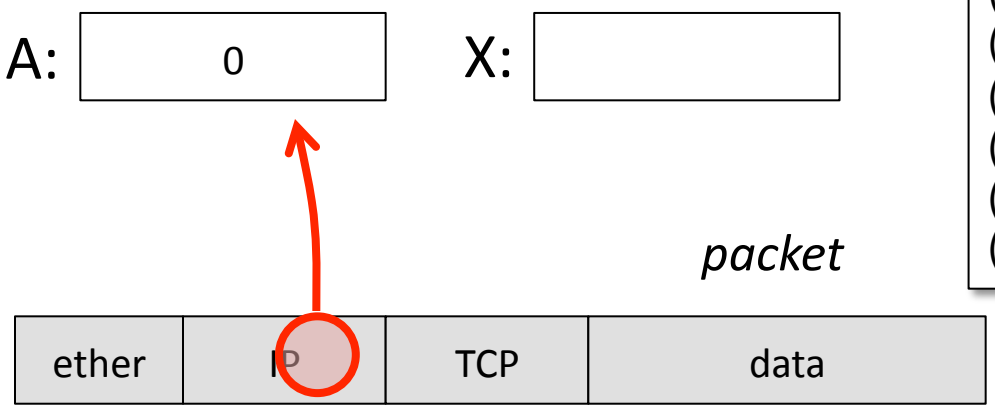


Example

FTP data packets
for host 10.0.0.20

Is it first or only frag?

```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20      jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20      jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14           jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14           jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```



Example

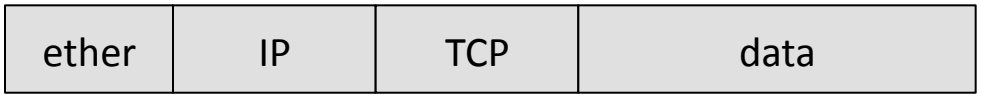
FTP data packets
for host 10.0.0.20

Is TCP src port FTP?

```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20      jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20      jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14           jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14           jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

A: X:

packet



Example

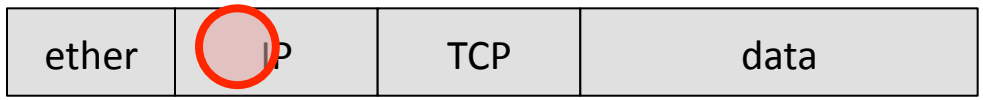
FTP data packets
for host 10.0.0.20

Is TCP src port FTP?

```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20      jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20      jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14           jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14           jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

A: [ ] X: [20]

packet

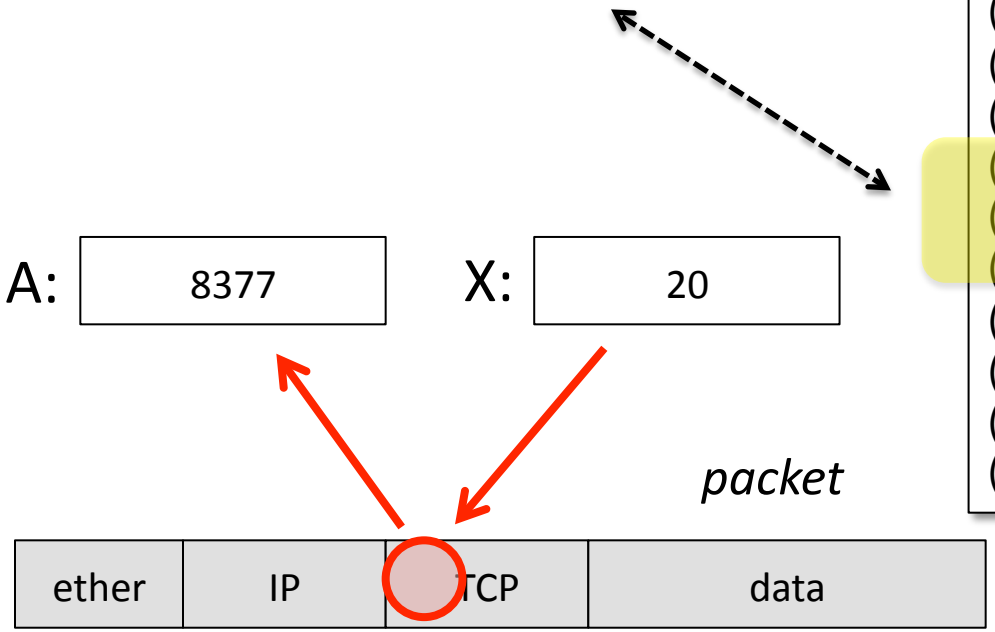


Example

FTP data packets
for host 10.0.0.20

Is TCP src port FTP?

```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20      jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20      jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14           jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14           jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```



Example

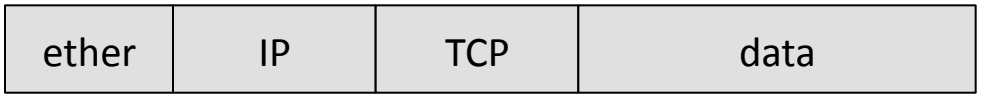
FTP data packets
for host 10.0.0.20

Is TCP dest port FTP?

A: 8377 X: 20

```
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20       jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20       jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6             jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff          jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14            jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14            jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

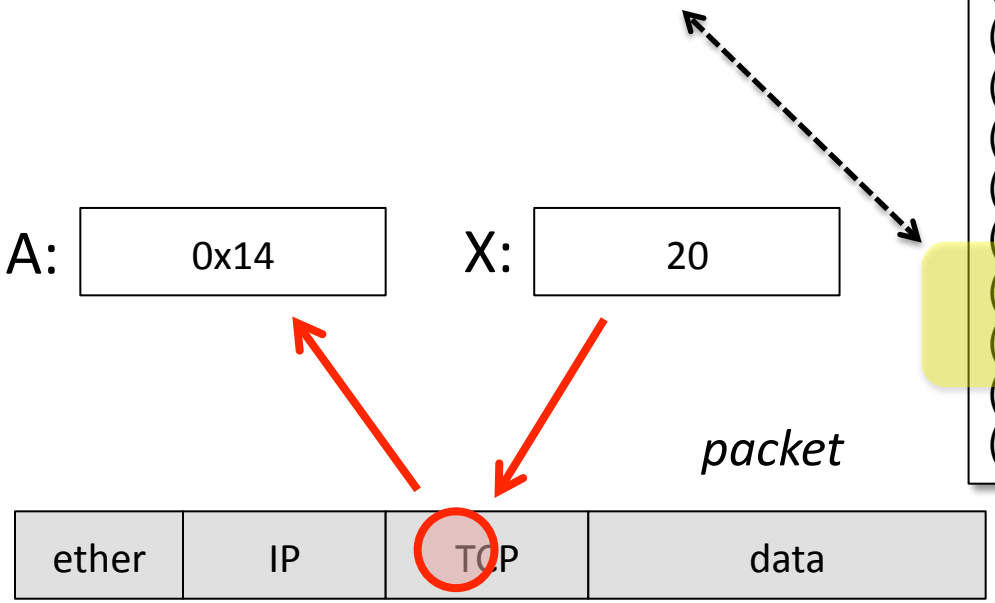
packet



Example

FTP data packets
for host 10.0.0.20

Is TCP dest port FTP?




```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20      jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20      jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14           jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14           jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

Example

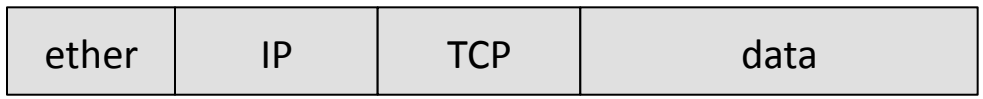
FTP data packets
for host 10.0.0.20

```
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20       jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20       jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6             jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff          jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14            jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14            jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

return TRUE

A: 0x14 X: 20

packet



The Challenge

- The BPF virtual machine model is a very flexible and efficient model for packet filtering
- But, you would never want to write low-level BPF programs every time you wanted to filter packets
- So, we needed a higher level model...

A Filter Language

Instead of writing this...

```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20      jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20      jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14           jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14           jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

A Filter Language

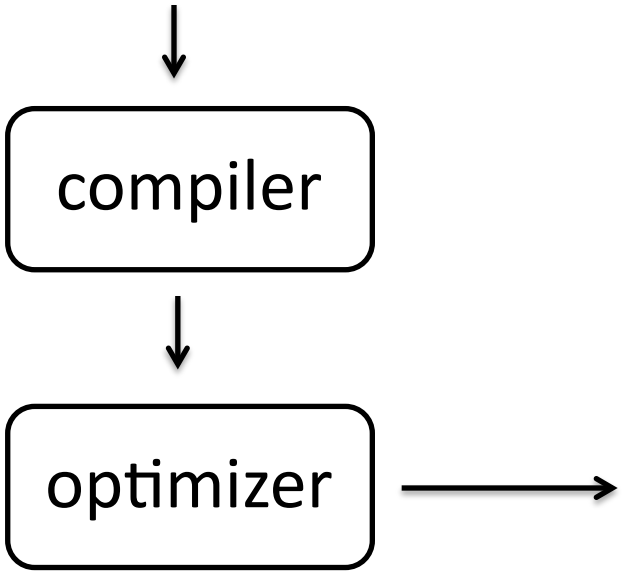
ip host ftp.arpa.net and tcp port ftp-data

Just say this...

```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20      jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20      jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14           jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14           jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

A Filter Language

ip host ftp.arpa.net and tcp port ftp-data




```
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 16
(002) ld       [26]
(003) jeq      #10.0.0.20      jt 6    jf 4
(004) ld       [30]
(005) jeq      #10.0.0.20      jt 6    jf 16
(006) ldb      [23]
(007) jeq      #0x6            jt 8    jf 16
(008) ldh      [20]
(009) jset     #0x1fff         jt 16   jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x14           jt 15   jf 13
(013) ldh      [x + 16]
(014) jeq      #0x14           jt 15   jf 16
(015) ret      #65535
(016) ret      #0
```

And let a compiler translate...

The Challenge

- This is where things got a bit tricky
- Designing the language and parser so it was easy on users turned out to be hard
- Learned an important life lesson from Van
  - It's easy to make things hard
  - It's hard to make things easy
  - It's usually better to do the latter

BPF Language

- The BPF filter language starts from a basic predicate, which is true iff the specified packet field equals the indicated value

  pred: field val

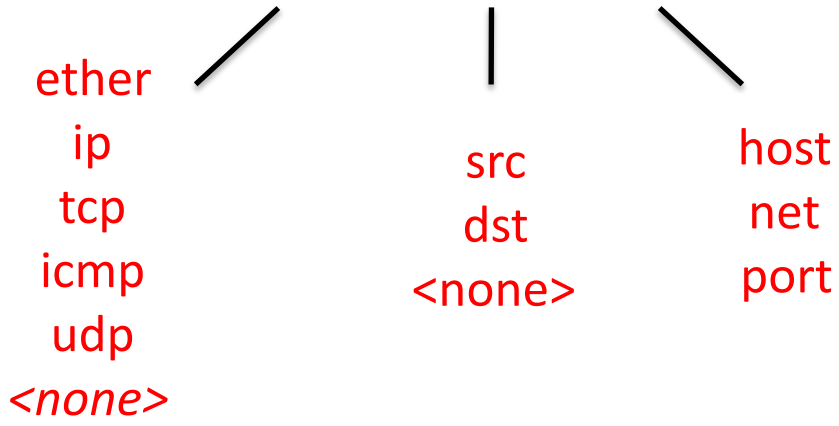
  field: protocol dir selector

BPF Language

- The BPF filter language starts from a basic predicate, which is true iff the specified packet field equals the indicated value

pred: field val

field: protocol dir selector

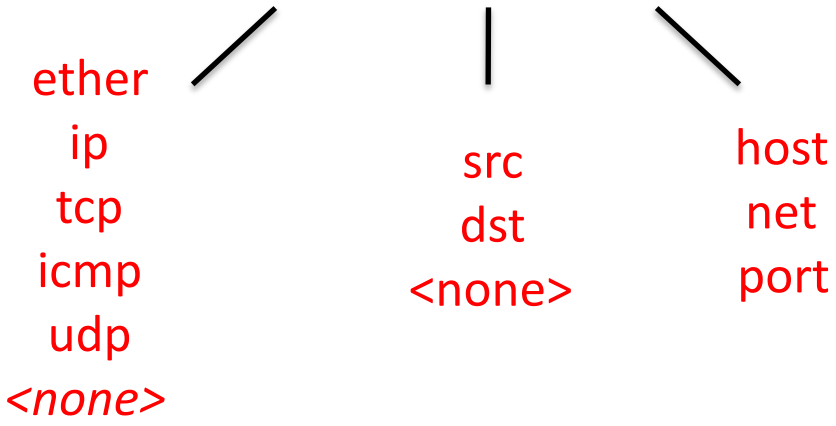


BPF Language

- The BPF filter language starts from a basic predicate, which is true iff the specified packet field equals the indicated value

pred: field val

field: protocol dir selector



examples

ip src host 10.0.0.1

tcp dst port 80

BPF Logic

- Language includes logic to stitch together predicates into complex logic operations
  - *pred* **or** *pred*
  - *pred* **and** *pred*
  - **not** *pred*
  - **‘(‘** *pred* **‘)’**

ip src host X and not port 80

My First Attempt

expr: *pred*

| *expr* AND *pred*

| *expr* OR *pred*

| NOT *expr*

| ‘(‘ *expr‘)’*

pred: *field val*

field: *protocol dir selector*

etc...

A Problem

- But Van didn’t like it... too clunky...

ip src host X or ip src host Y or ip src host Z

A Problem

- But Van didn’t like it... too clunky...

ip src host X or ip src host Y or ip src host Z

A Problem

- But Van didn't like it... too clunky...

  ip src host X or ip src host Y or ip src host Z

- Why not just say...

  ip src host X or Y or Z

- This should be easy enough to fix...

My Second Attempt

- Introduce two layers of logic
  - Lower layer would handle predicates with multiple values
    - ip host x or y
    - tcp port 80 or 1024
  - Upper layer would handle the combinations of the lower-layer expressions
    - (ip host x or y) and (tcp port 80 or 1024)

My Second Attempt

expr: *term*

| *expr* AND *term*

| *expr* OR *term*

| NOT *expr*

| ‘(‘ *expr*‘)’

term: *pred*

| *term* AND *val*

| *term* OR *val*

| NOT *term*

| ‘(‘ *term* ‘)’

pred: *field val*

field: *protocol dir selector*

etc...

The Second Problem

- But this didn't work at all
  - the parser needs to decide to parse as a *term*

    ip src host x or y and z
  - or parse input as an *expr*

    ip src host x or y and tcp port z
  - when the partial input didn't provide enough info

    ip src host x or y and [unknown]

    parsed input *look-ahead*

It’s easy to make things hard

- Some easy ways out...
  - require parens or another grouping symbol
    - ip host (x or y) and tcp port z
    - ip host { x or y } and tcp port z
  - have different families of logic symbols
    - e.g., “and”, “AND”, “or”, “OR”
    - ip host x or y AND tcp port z
  - introduce terminator symbol
    - ip host x or y . and tcp port z

It’s hard to make things easy

- But all those solutions made things harder on the user, even though they were easy outs
  - So, Van challenged me
  - “There must be a way. Figure it out.”
- I spent a week or two frustratingly thinking about it and finally the light bulb came on
  - Turns out this was a novel language construct

The Solution

- Have a single level of logic, not two
- Allow predicates *or* values to be tacked onto an expression
- i.e., an *expr* can be both
  - *expr* AND *pred*
  - *expr* AND *val*

Third Time's a Charm

expr: *pred*

| *expr* AND *pred*

| *expr* AND *val*

| *expr* OR *pred*

| *expr* OR *val*

| NOT *expr*

| '(' *expr* ')'

pred: *field val*

Not so fast...

- Ok, this grammar worked out fine, but now code generation became tricky
- Fortunately, this problem while tricky, had a solution...

My Third Attempt

expr: *pred*

| *expr* AND *pred*

| *expr* AND *val*

| *expr* OR *pred*

| *expr* OR *val*

| NOT *expr*

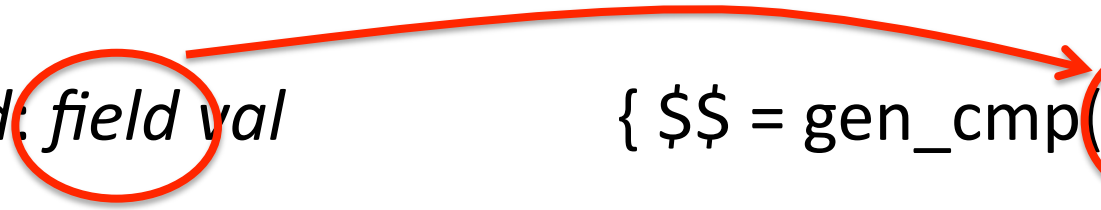
| ‘(‘ *expr*‘)’

pred: *field val* { $$ = gen_cmp($1, $2); }

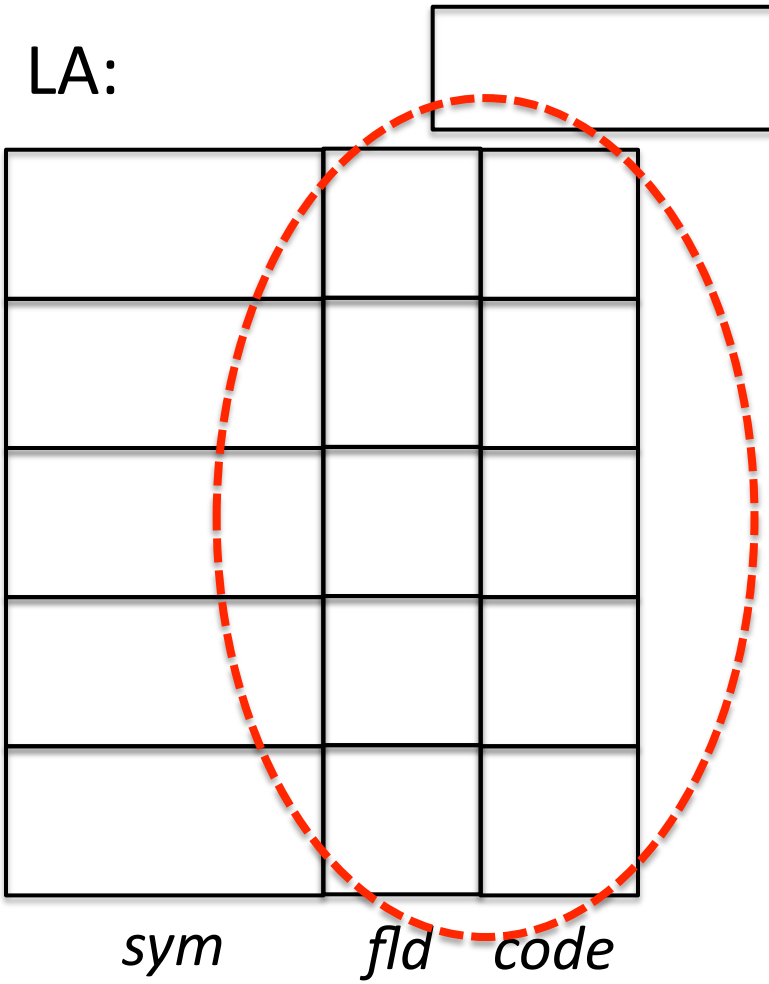
My Third Attempt

expr: *pred*

| *expr* AND *pred*	{ $$ = gen_and($1,$3); }
| *expr* AND *val*	**{ ??? }**
| *expr* OR *pred*	{ $$ = gen_or($1, $3); }
| *expr* OR *val*	**{ ??? }**
| NOT *expr*	{ $$ = gen_not($2); }
| '(' *expr* ')'	{ $$ = $2; }
pred: *field val*	{ $$ = gen_cmp($1, $2); }



The Solution



expr: *pred*

| *expr* AND *pred*

| *expr* AND *val*

| *expr* OR *pred*

| *expr* OR *val*

| NOT *expr*

| '(' *expr* ')'

pred: *field val*

My Third Attempt

expr: *pred*

| *expr* AND *pred*

| *expr* AND *val*

| *expr* OR *pred*

| *expr* OR *val*

| NOT *expr*

| '(' *expr* ')'

```
{ t = gen_cmp($1.fld, $3);
  $$.code = gen_and($1, t)
  $$.fld = $1.fld; }
```

pred: *field val*

```
{ $$.code = gen_cmp($1, $2);
  $$.fld = $1; }
```

My Third Attempt

expr: *pred*

| *expr* AND *pred*

| *expr* AND *val*

| *expr* OR *pred*

| *expr* OR *val*

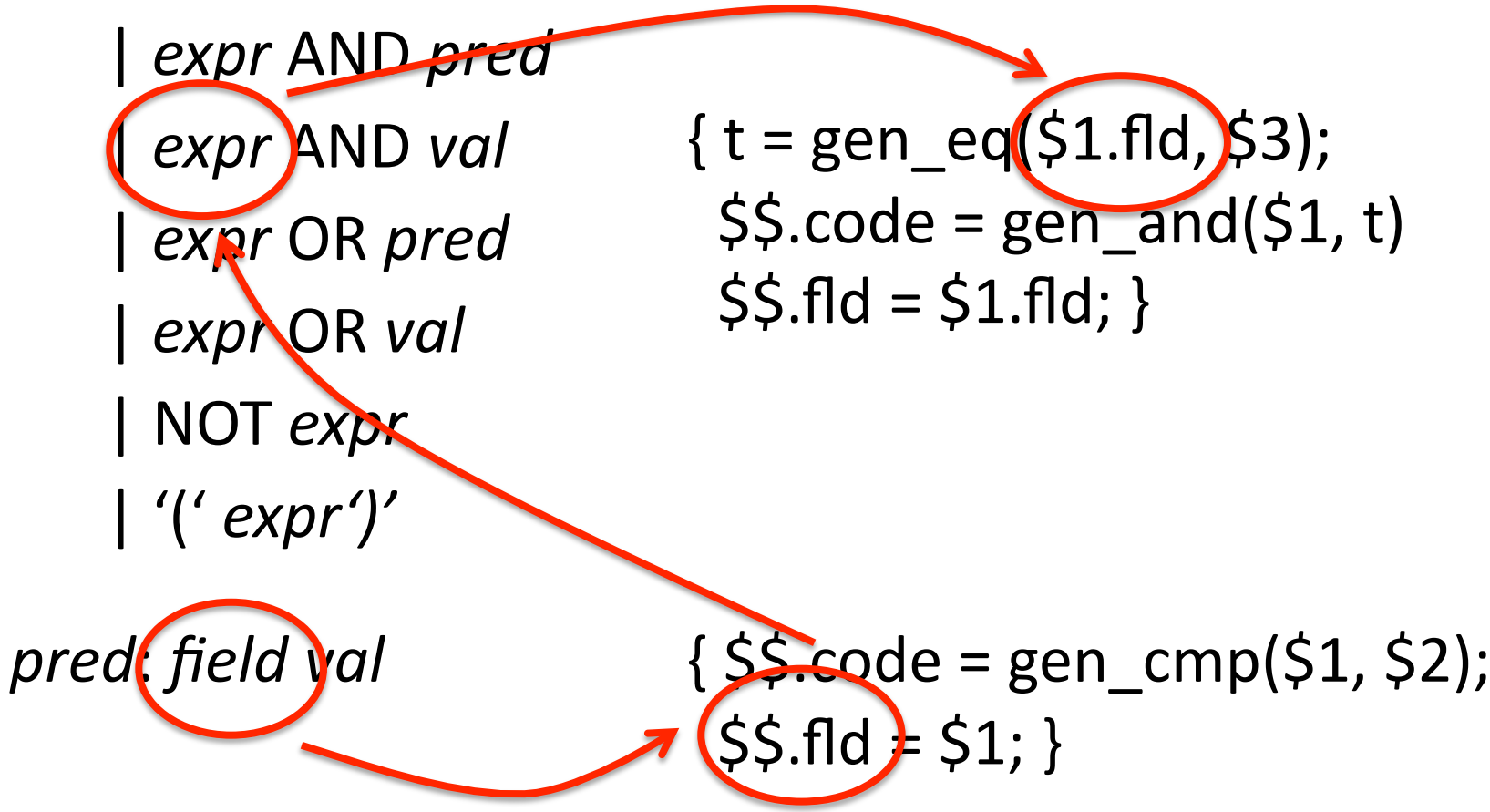
| NOT *expr*

| '(' *expr* ')'

pred: *field val*

```
{ t = gen_eq($1.fld, $3);
  $$.code = gen_and($1, t)
  $$.fld = $1.fld; }
```

```
{ $$.code = gen_cmp($1, $2);
  $$.fld = $1; }
```



My Third Attempt

expr: *pred*

| *expr* AND *pred*

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred*

| *expr* OR *val*

| NOT *expr*

| ‘(‘ *expr*‘)’

pred: *field val* { $$ = gen_cmp($1, $2); }

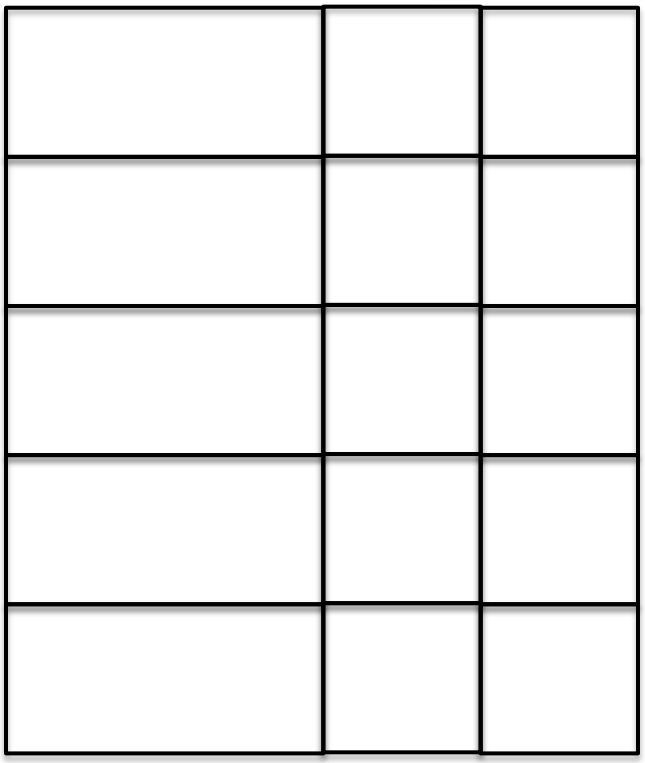
My Third Attempt

expr: *pred*	
| *expr* AND *pred*	{ $$ = gen_and($1,$3); }
| *expr* AND *val*	{ $$ = gen_vand($1, $3); }
| *expr* OR *pred*	{ $$ = gen_or($1, $3); }
| *expr* OR *val*	{ $$ = gen_vor($1, $3); }
| NOT *expr*	{ $$ = gen_not($2); }
| '(' *expr* ')'	{ $$ = $2; }
pred: *field val*	{ $$ = gen_cmp($1, $2); }

Example

LA: IP

ip src host x or y and tcp dst port z



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

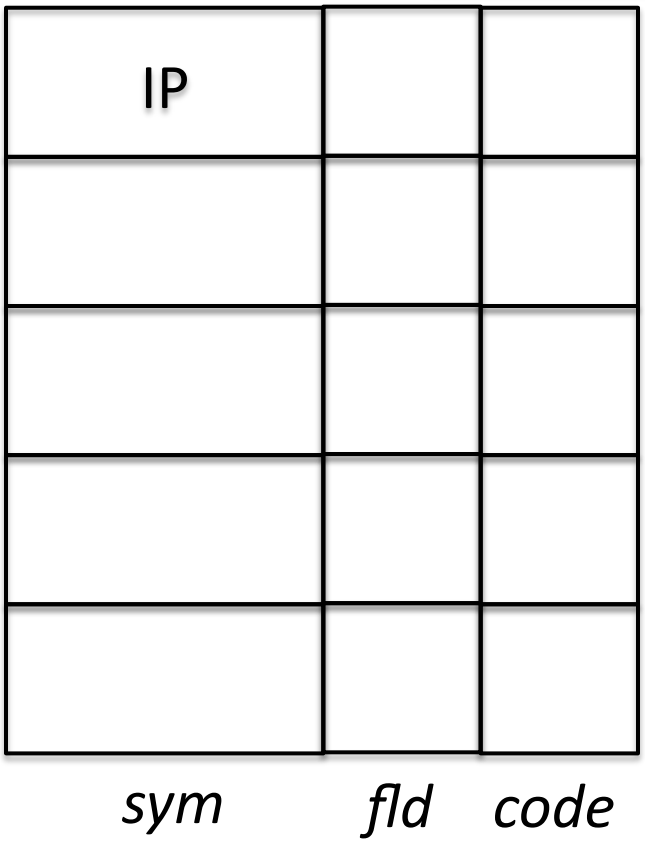
pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: SRC

src host x or y and tcp dst port z



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

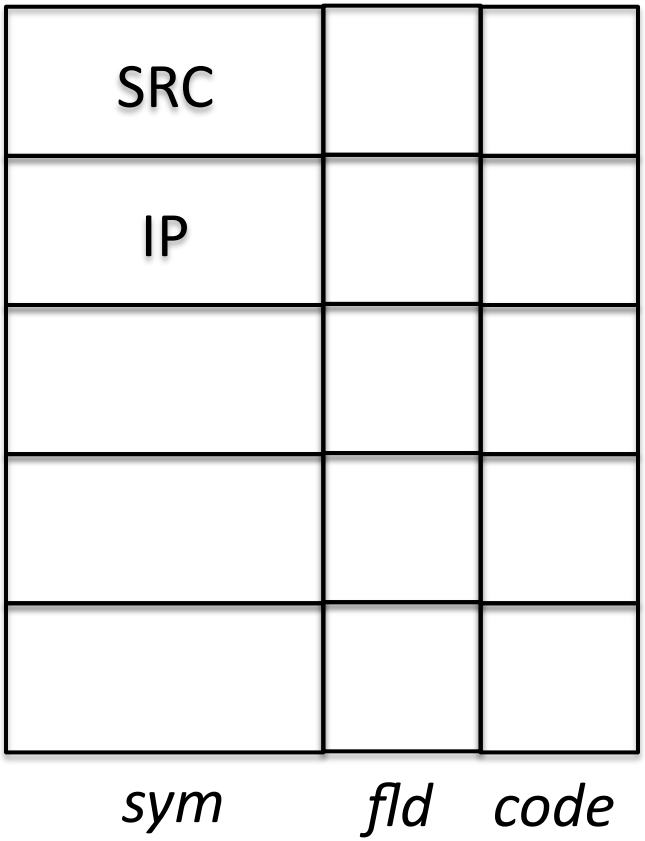
pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: HOST

host x or y and tcp dst port z



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

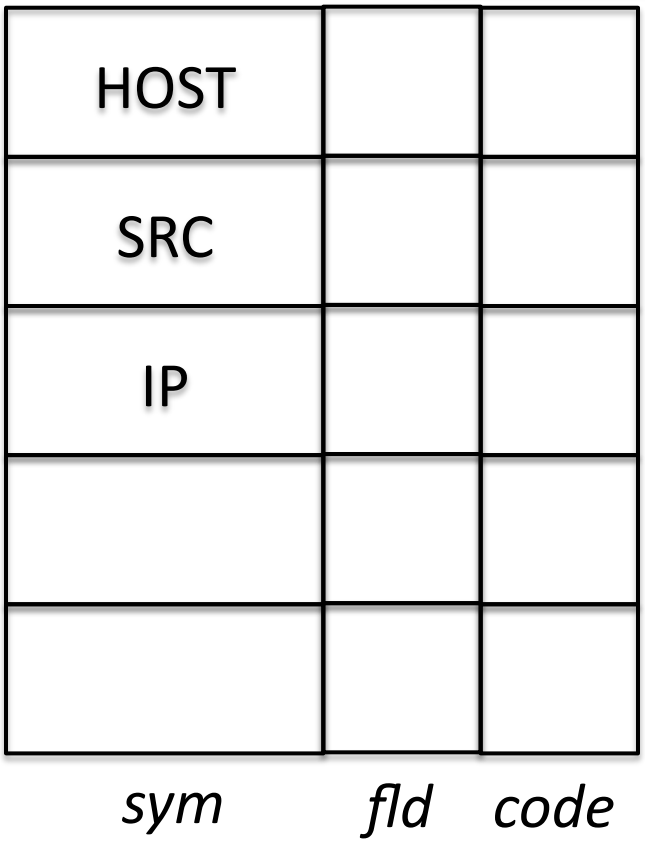
pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: X

x or y and tcp dst port z



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

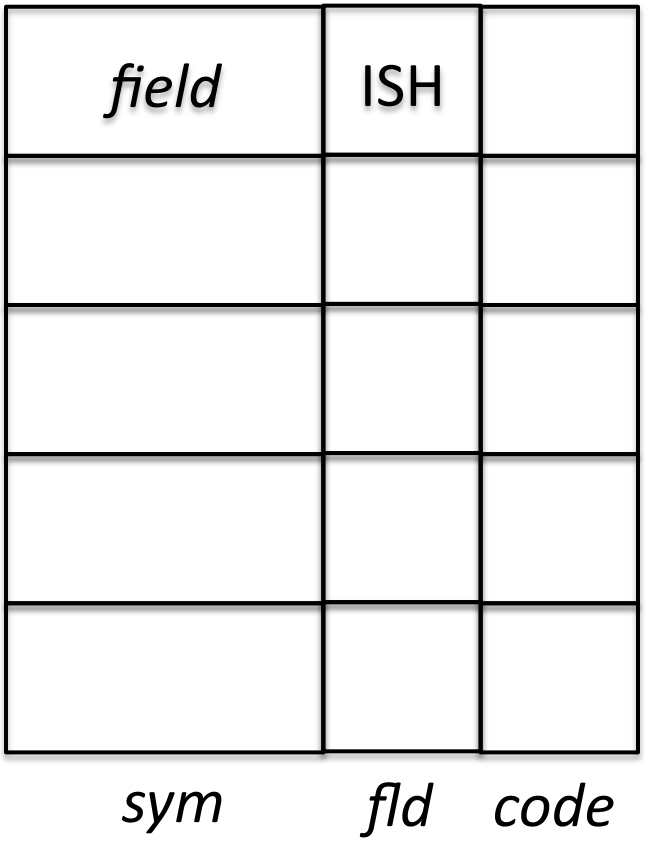
pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: X

x or y and tcp dst port z



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

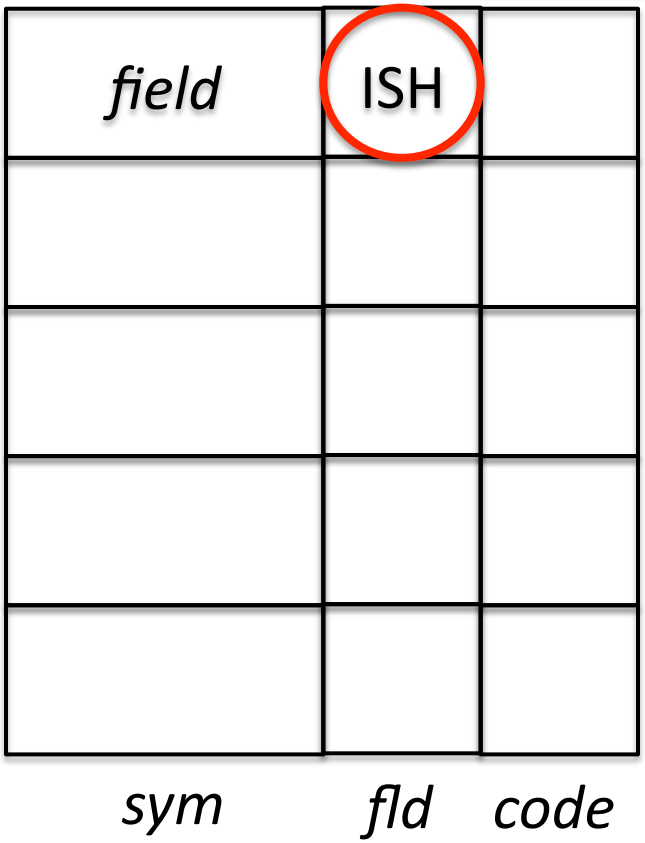
pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: X

x or y and tcp dst port z



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

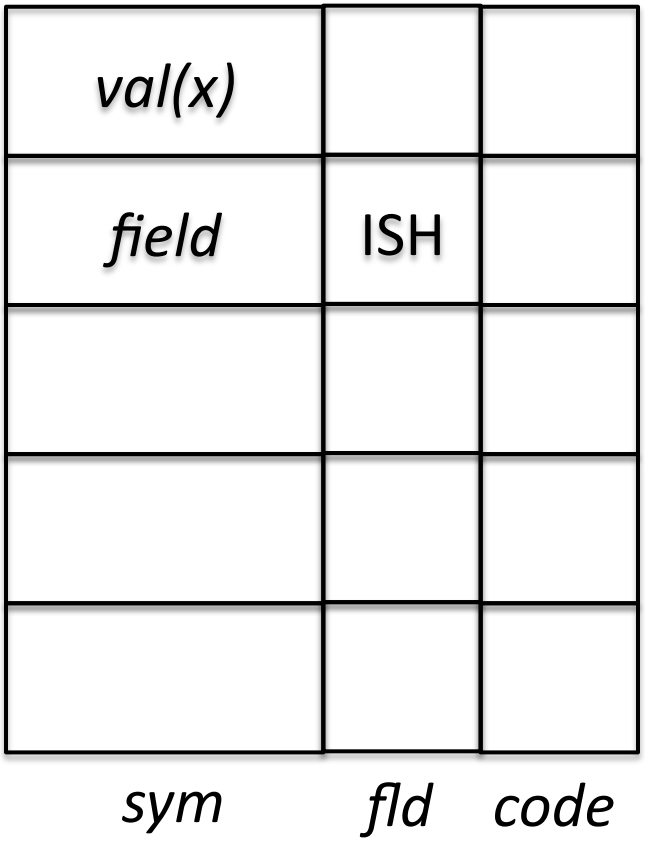
pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: OR

or y and tcp dst port z



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

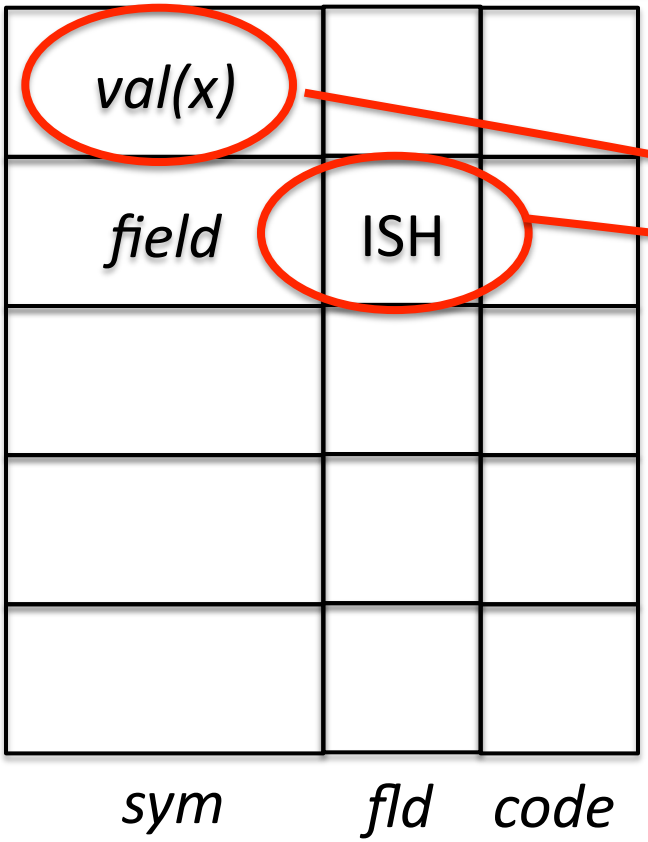
pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: OR

or y and tcp dst port z



expr : *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

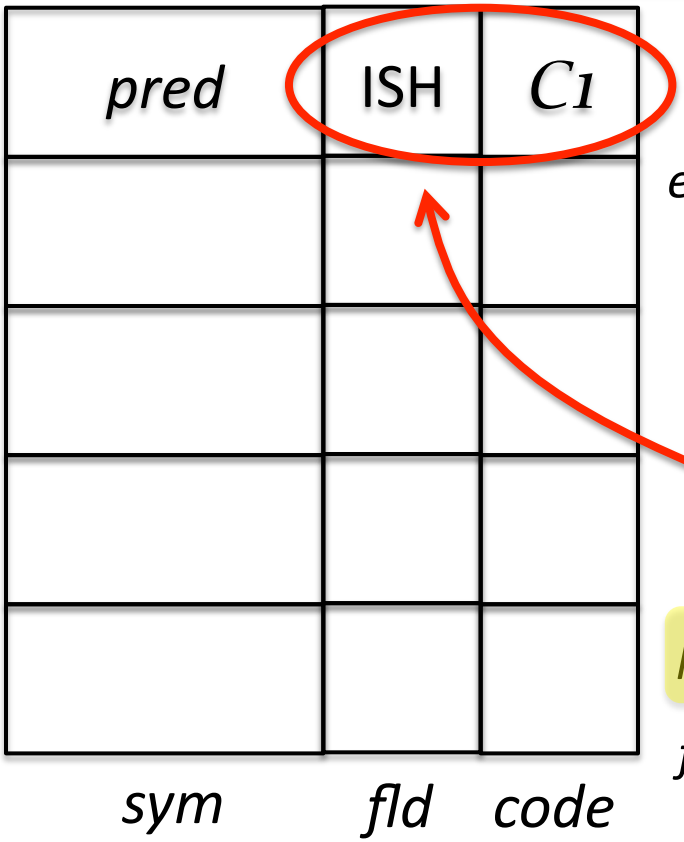
pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: OR

or y and tcp dst port z



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: OR

or y and tcp dst port z

pred	ISH	*C1*

sym *fld* *code*

expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: OR

or y and tcp dst port z

expr	ISH	*C1*

sym *fld* *code*

expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: Y

y and tcp dst port z

OR		
expr	ISH	*C1*

sym *fld* *code*

expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: AND

and tcp dst port z

val(y)		
OR		
expr	ISH	*C1*
sym	*fld*	*code*

expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: AND

and tcp dst port z

val(y)		
OR		
expr	ISH	*C1*
sym	*fld*	*code*

expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

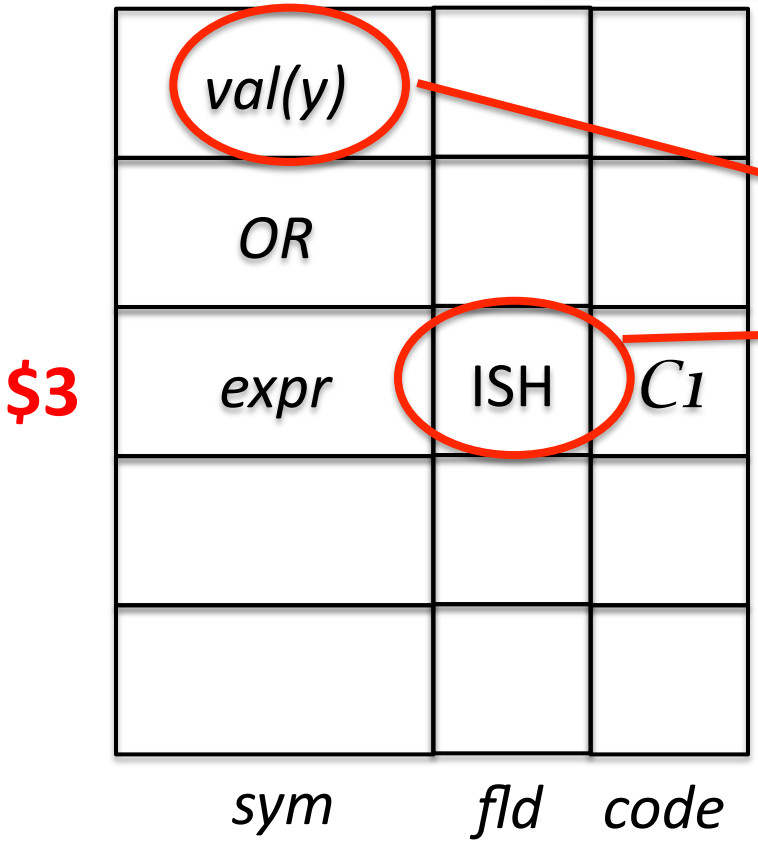
pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: AND

and tcp dst port z



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: AND

and tcp dst port z

expr	ISH	*C2*

sym *fld* *code*

expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

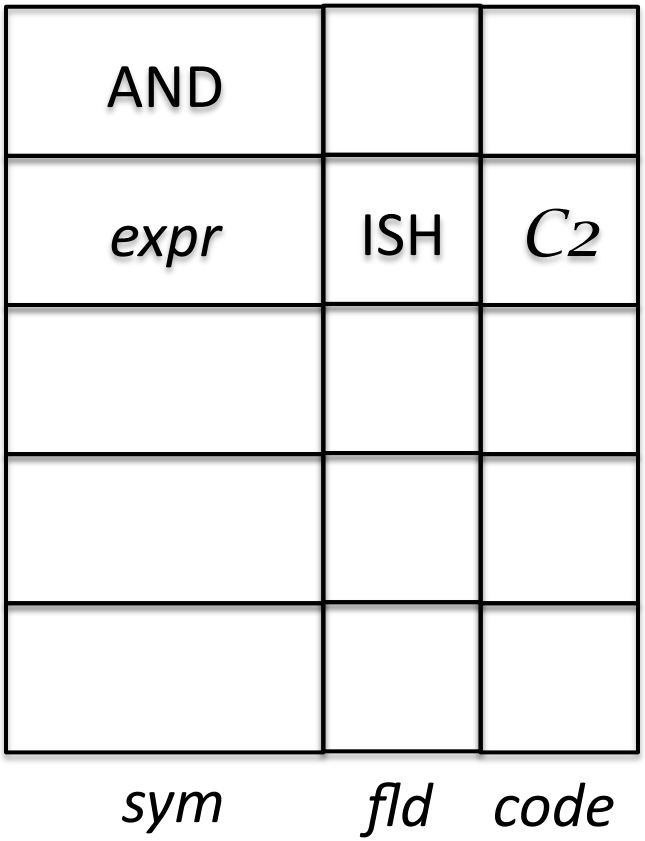
pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: TCP

tcp dst port z



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: DST

dst port z

sym	*fld*	*code*
TCP		
AND		
expr	ISH	C_2

expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: PORT

port z

DST		
TCP		
AND		
expr	ISH	*C2*
sym	*fld*	*code*

expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

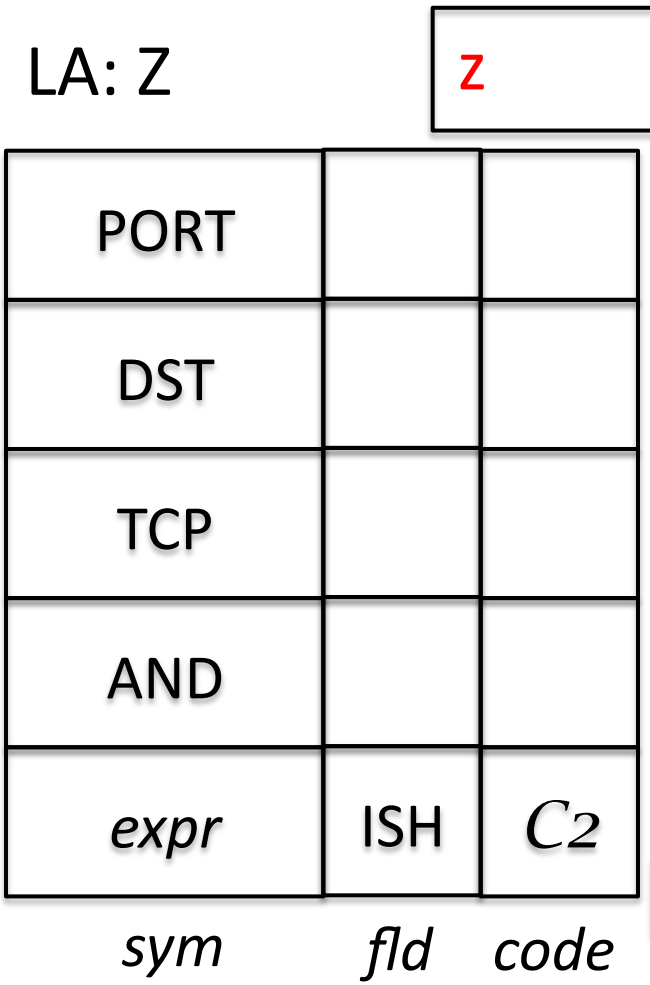
| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: Z

z

field	TDP	
AND		
expr	ISH	*C2*

sym *fld* *code*

expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

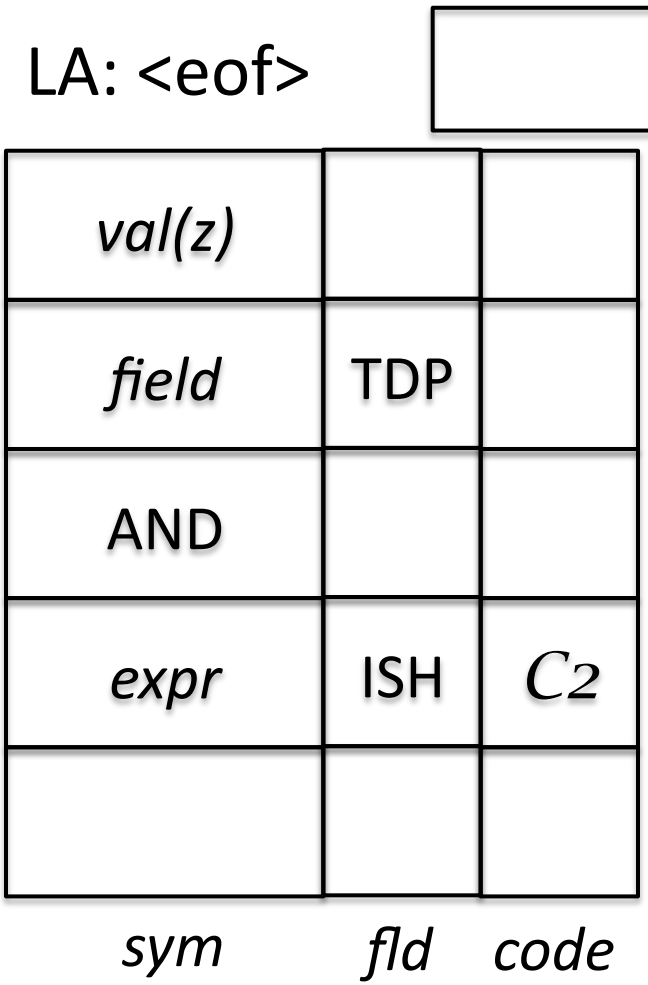
| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: <eof>



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

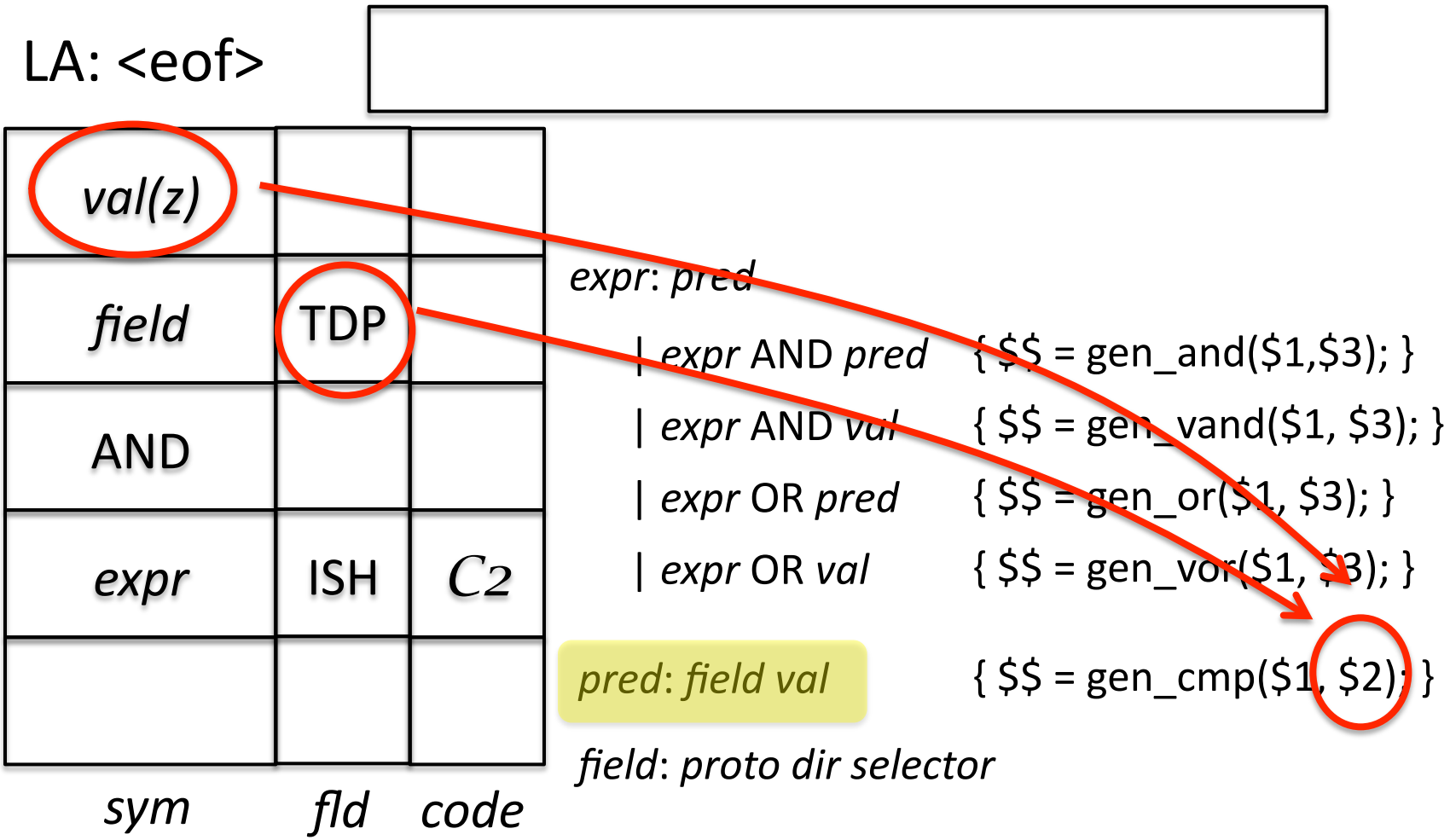
| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

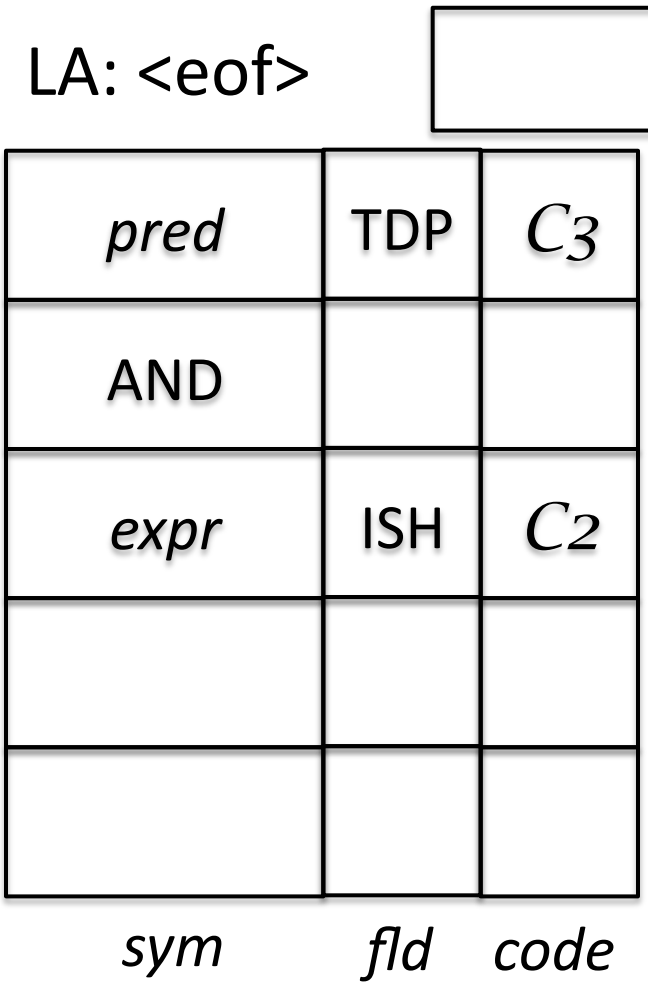
field: *proto dir selector*

Example


LA: <eof>
val(z)
field
TDP
AND
expr
ISH
C2
sym
fld
code
expr: pred
| expr AND pred { $$ = gen_and($1,$3); }
| expr AND val { $$ = gen_vand($1, $3); }
| expr OR pred { $$ = gen_or($1, $3); }
| expr OR val { $$ = gen_vor($1, $3); }
pred: field val { $$ = gen_cmp($1, $2); }
field: proto dir selector

Example

LA: <eof>



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

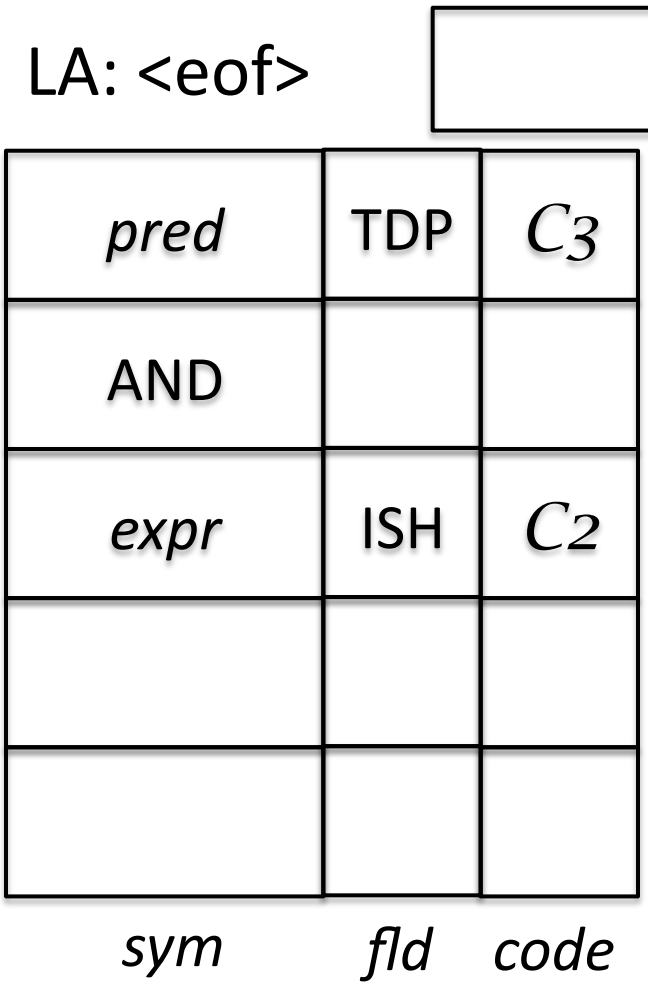
| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: <eof>



expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

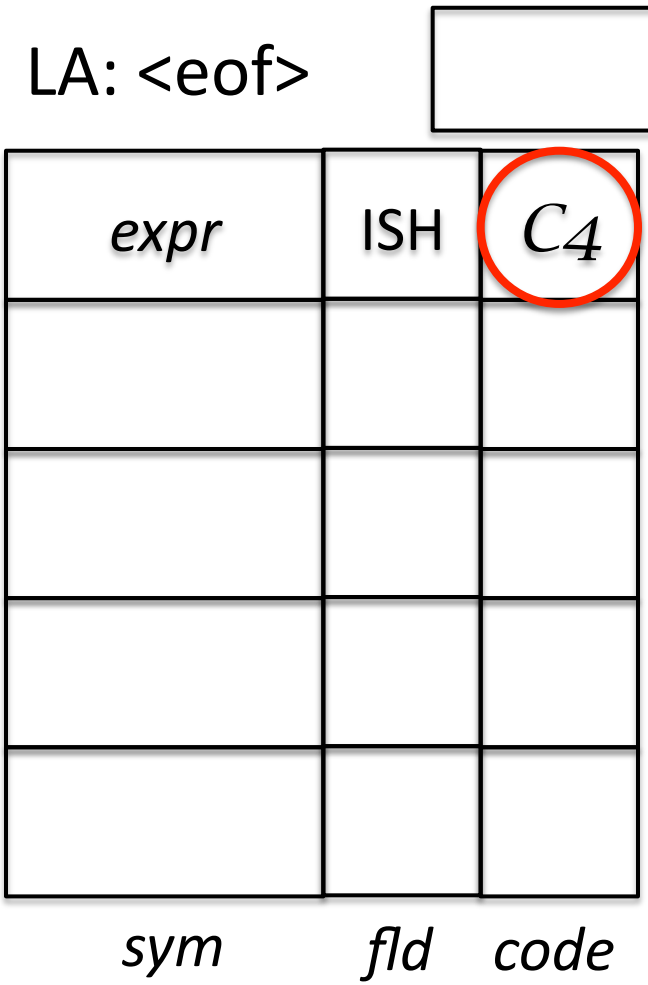
| *expr* OR *val* { $$ = gen_vor($1, $3); }

pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Example

LA: <eof>



output BPF code

expr: *pred*

| *expr* AND *pred* { $$ = gen_and($1,$3); }

| *expr* AND *val* { $$ = gen_vand($1, $3); }

| *expr* OR *pred* { $$ = gen_or($1, $3); }

| *expr* OR *val* { $$ = gen_vor($1, $3); }

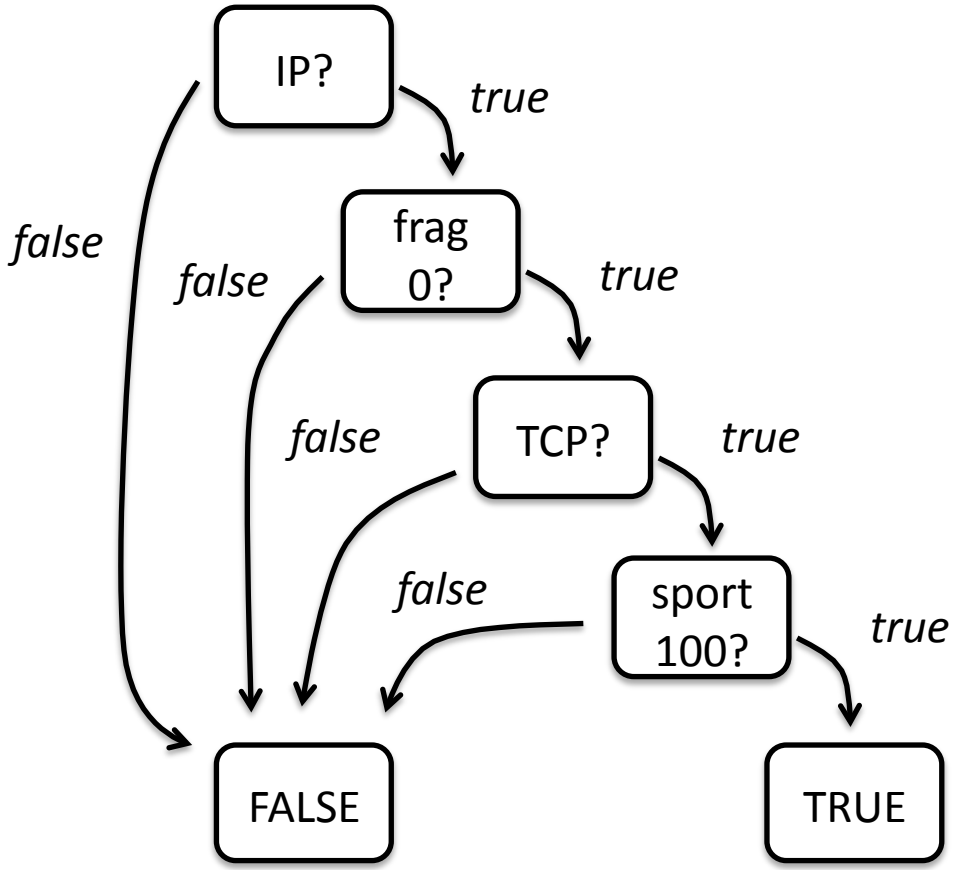
pred: *field val* { $$ = gen_cmp($1, $2); }

field: *proto dir selector*

Code Generation

- Now that we have a language and a parser to translate it, how do those gen() functions actually work?
  - gen_cmp() generates code to compare a packet field to a value
  - Ex: tcp src port 100
    - "tcp src port" is the field
    - 100 is the value

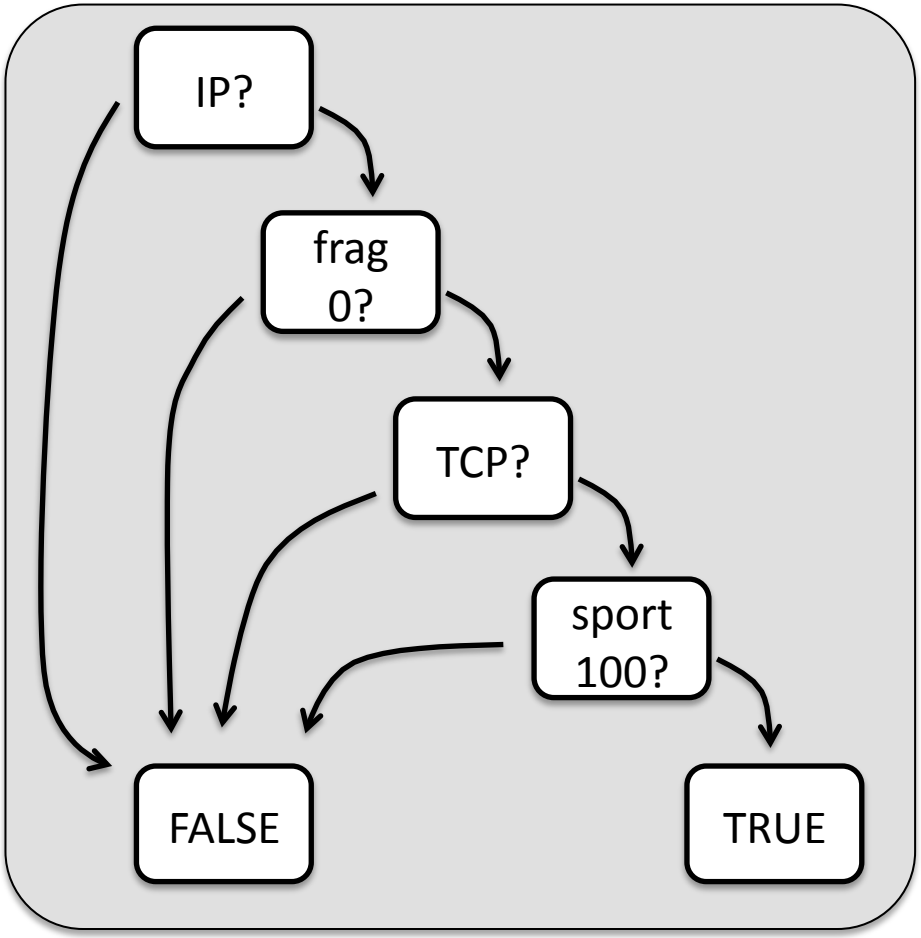
tcp src port 100



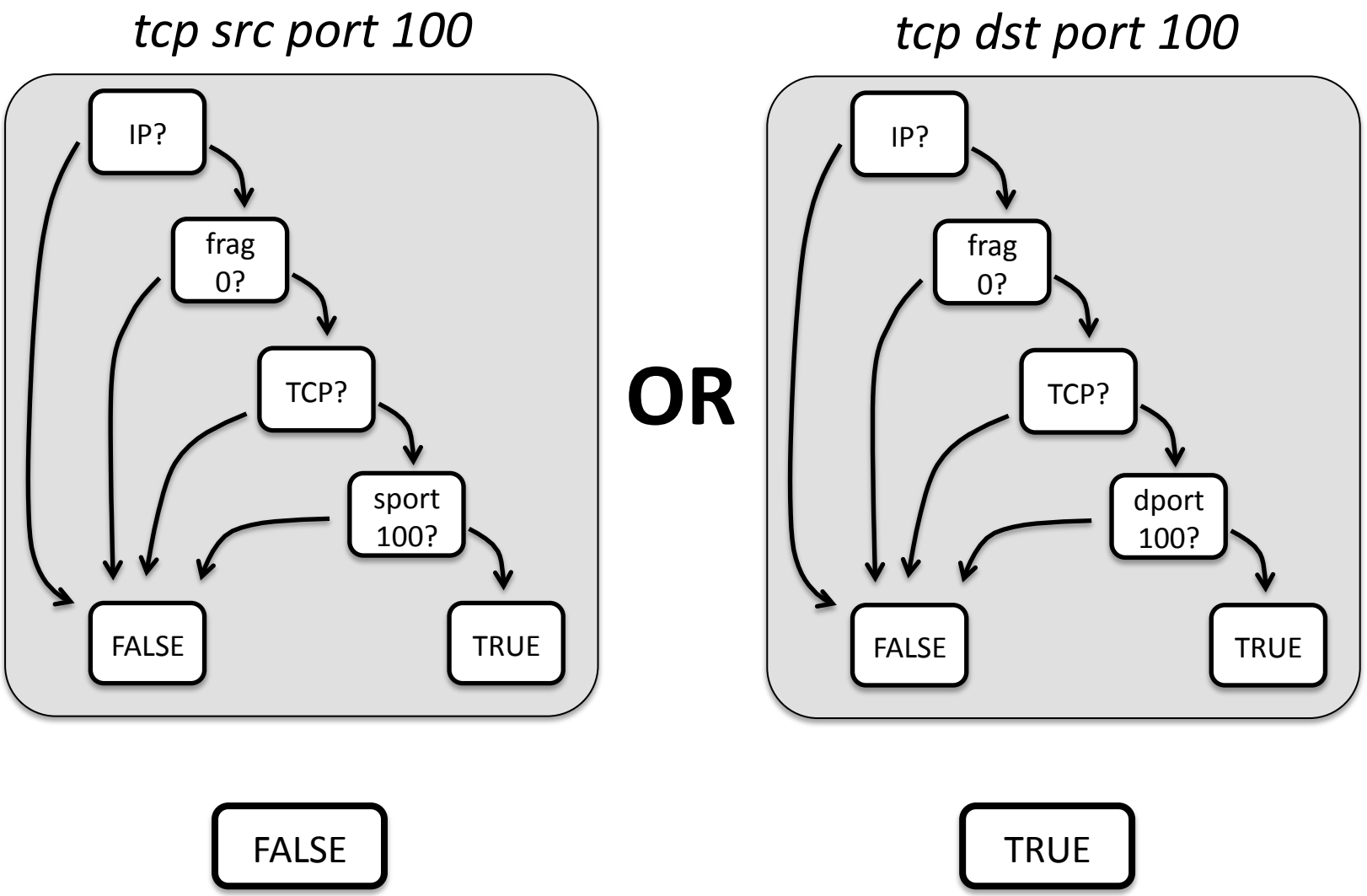
Compound Logic

- Now, what if want traffic in either direction for port 100?
  - tcp port 100
    - tcp src port 100
    - **OR**
    - tcp dst port 100

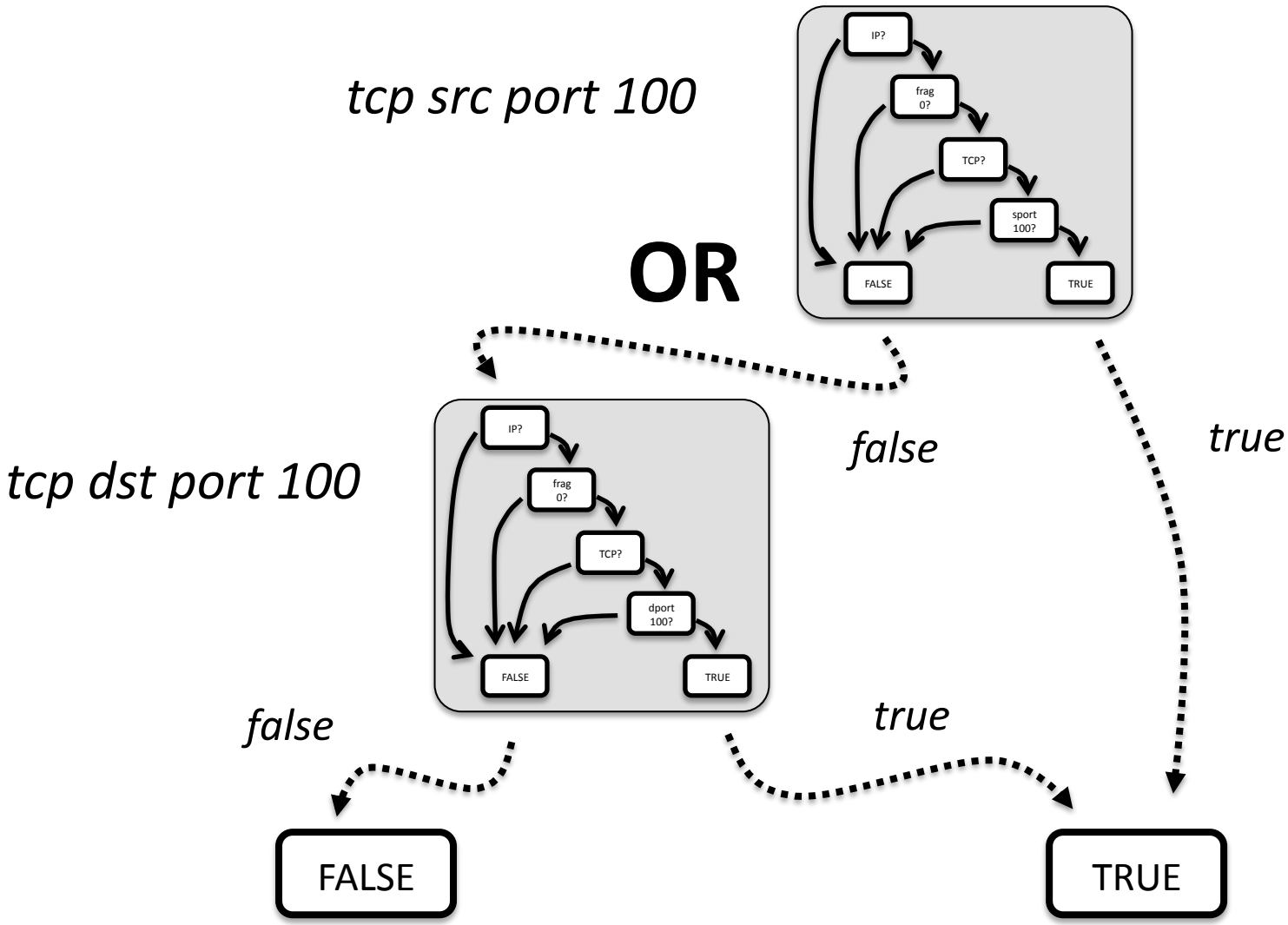
tcp src port 100



tcp port 100


tcp src port 100
IP?
frag 0?
TCP?
sport 100?
FALSE
TRUE
OR
tcp dst port 100
IP?
frag 0?
TCP?
dport 100?
FALSE
TRUE
FALSE
TRUE

tcp port 100

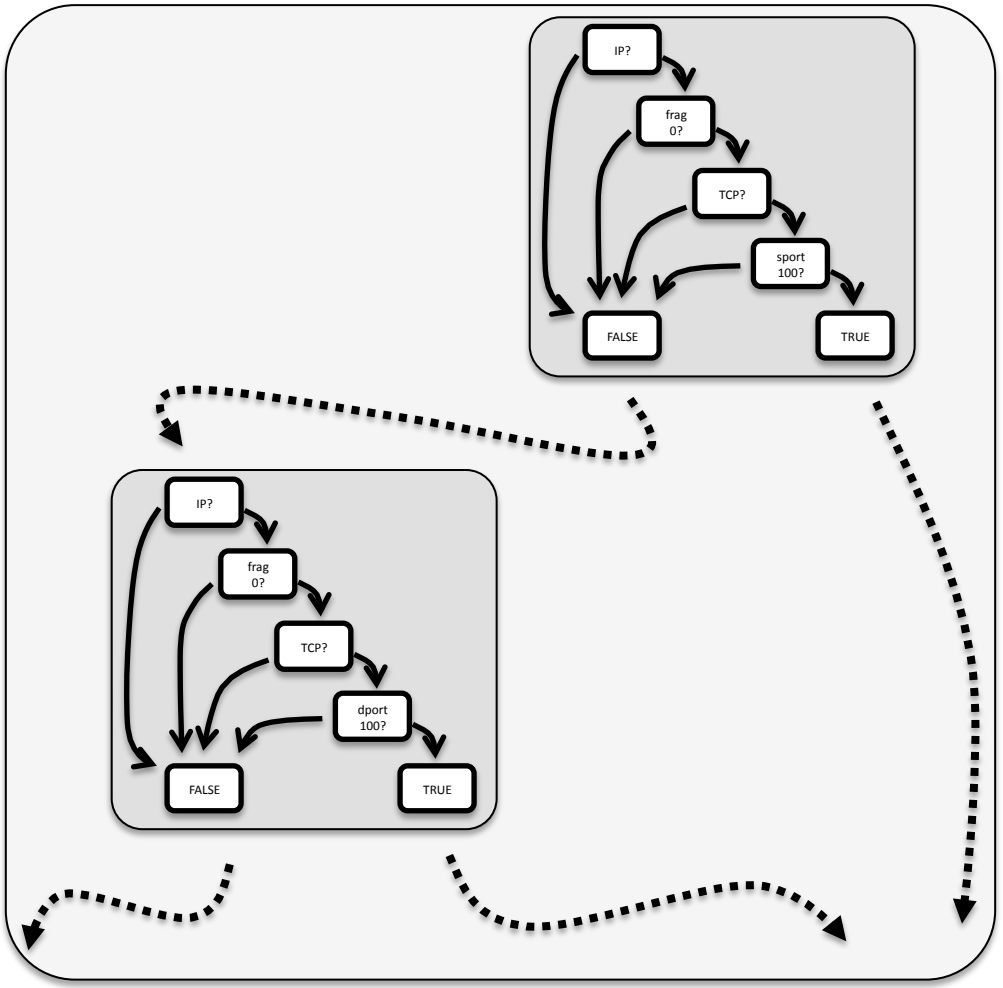

tcp src port 100
IP?
frag 0?
TCP?
sport 100?
FALSE
TRUE
OR
tcp dst port 100
IP?
frag 0?
TCP?
dport 100?
FALSE
TRUE
false
true
false
true
FALSE
TRUE

Compound Logic

- What if I want packets *between* port 100 and some other specific port 200?
  - tcp port 100
  - **AND**
  - tcp port 200
- The output gets ever more complex

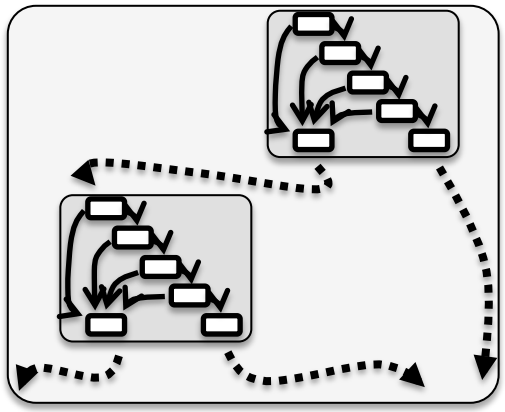
tcp port 100 and 200?

tcp port 100

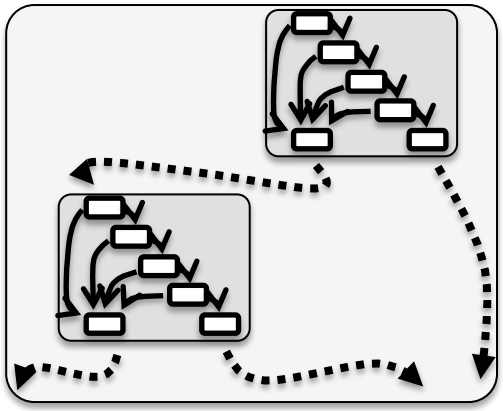


tcp port 100 and 200

tcp port 100



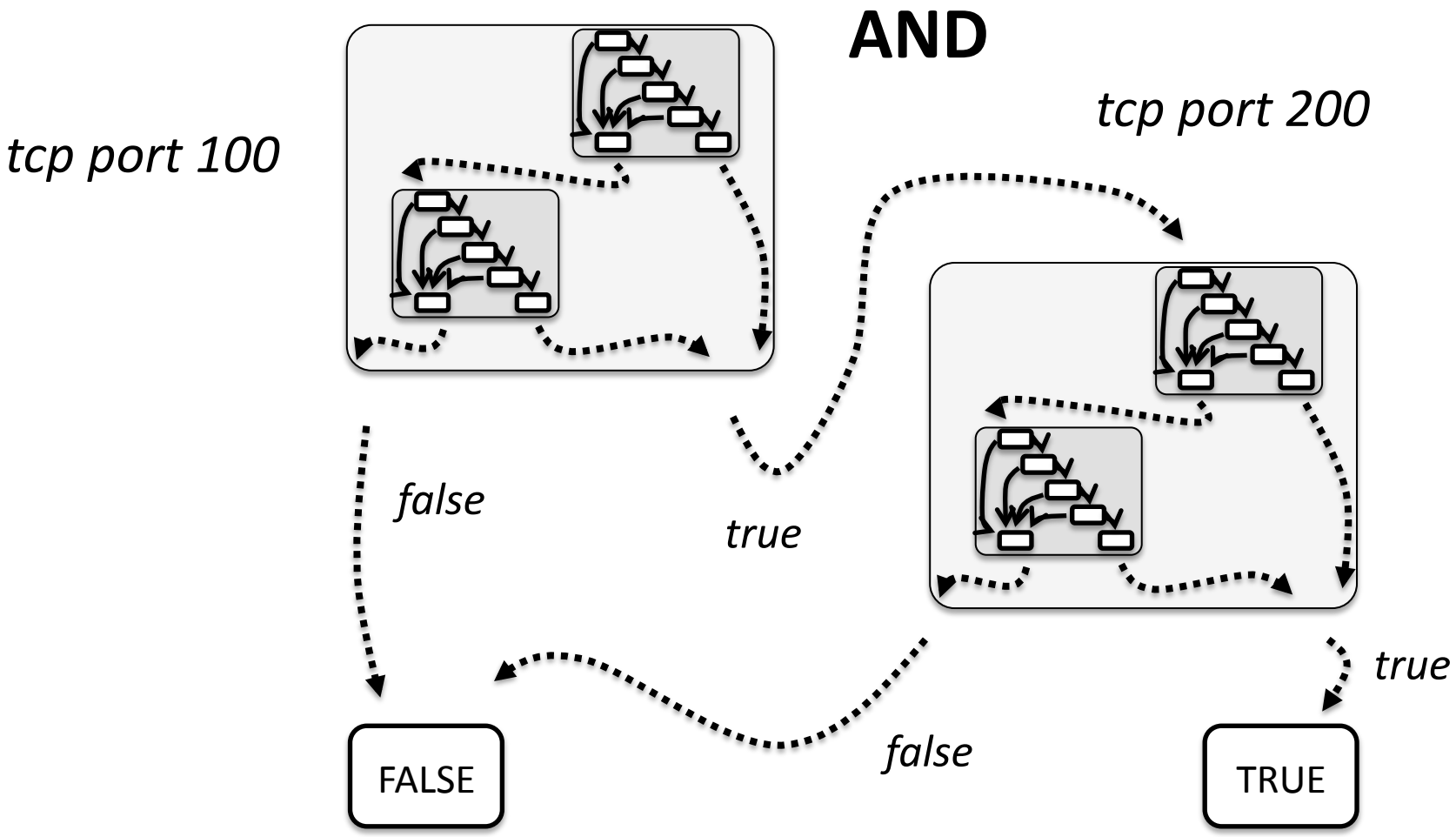
tcp port 200



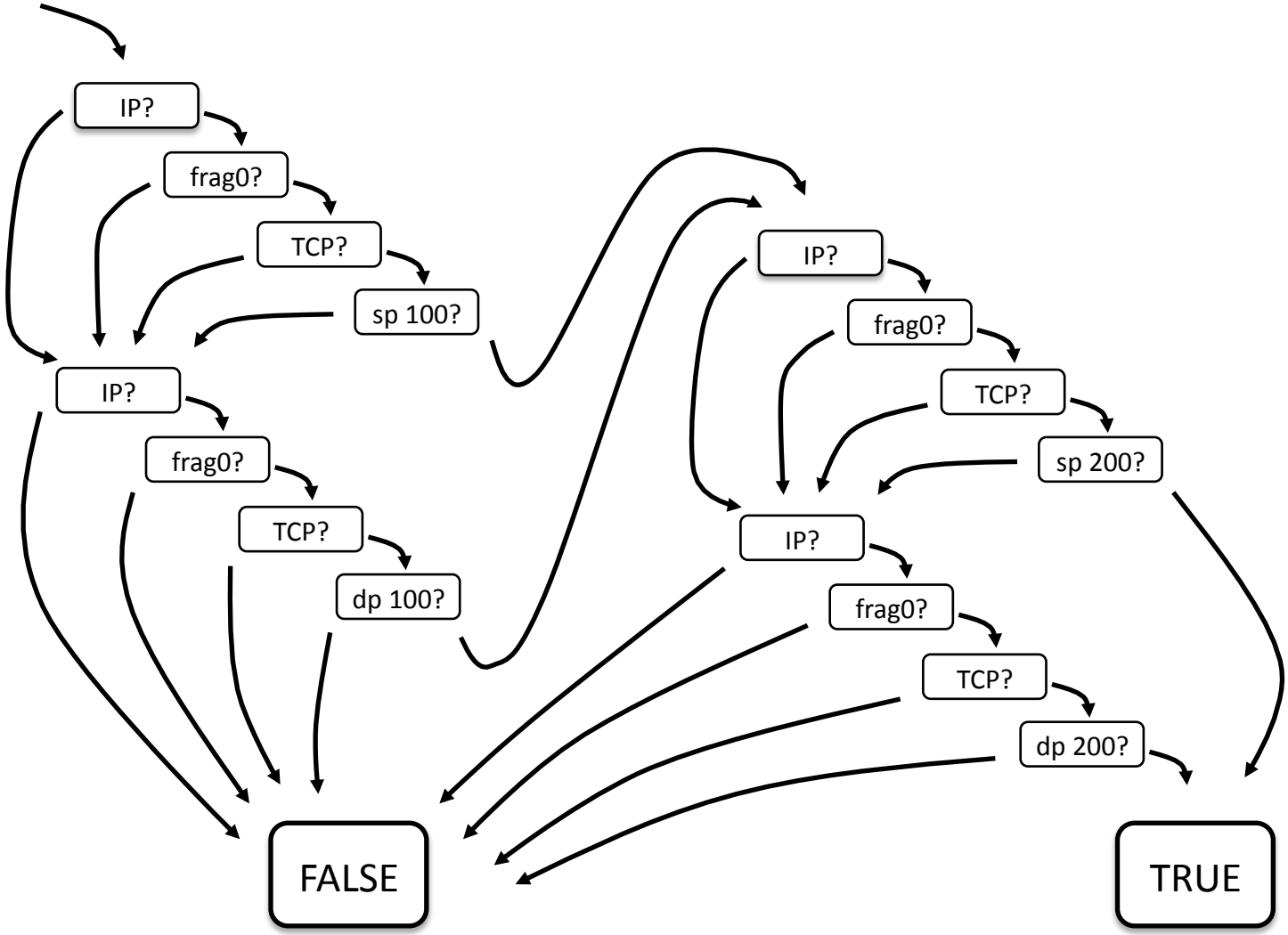
FALSE

TRUE

tcp port 100 and 200



tcp port 100 and 200

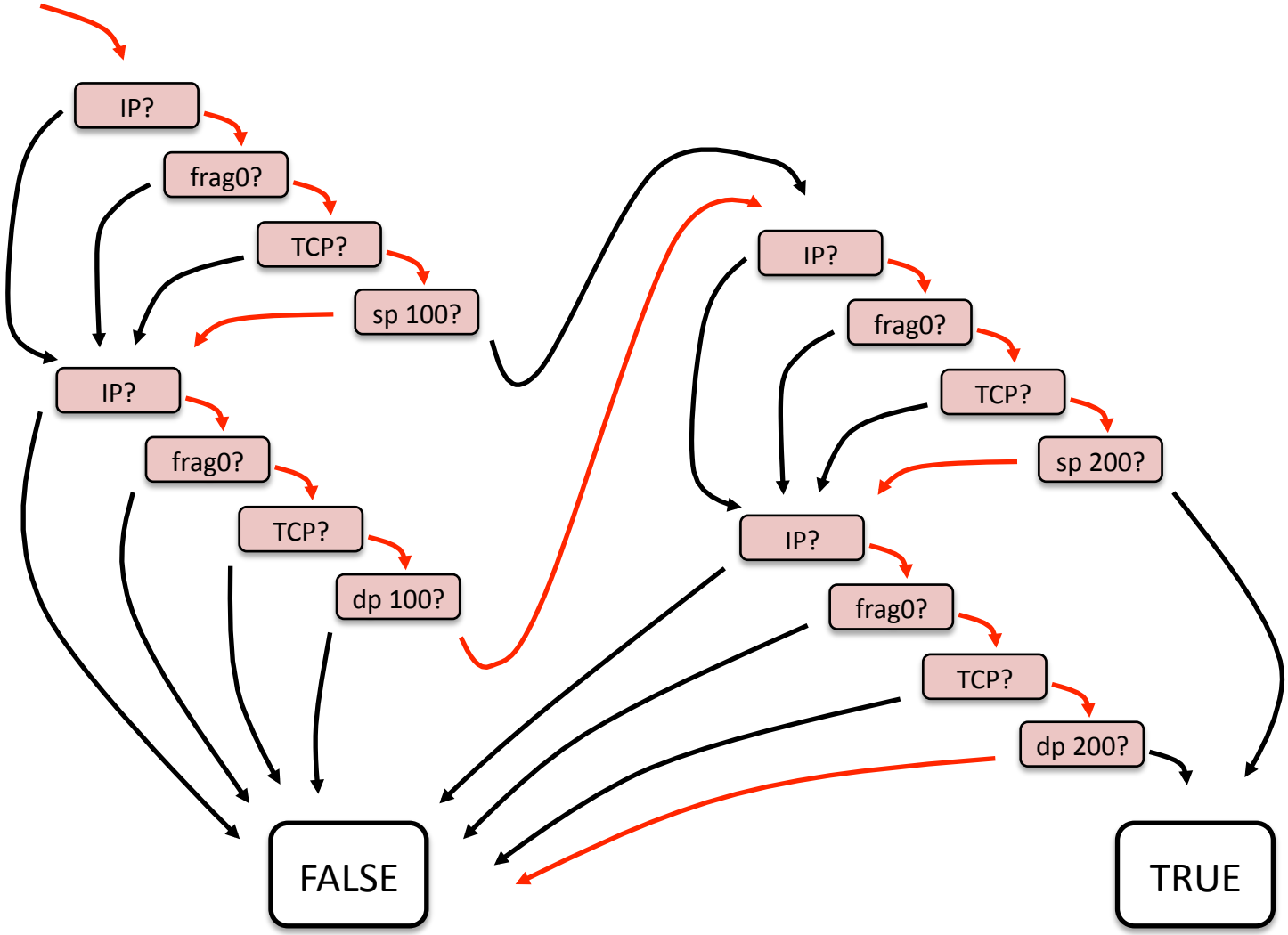


The Raw Code

```
(000) ldh      [16]
(001) jeq      #0x800          jt 2 jf 43
(002) ldh      [16]
(003) jeq      #0x86dd         jt 4 jf 10
(004) ldb      [24]
(005) jeq      #0x6            jt 6    jf 10
(006) ldh      [58]
(007) jeq      #0x64           jt 22jf 8
(008) ldh      [60]
(009) jeq      #0x64           jt 22jf 10
(010) ldh      [16]
(011) jeq      #0x800          jt 12 jf 43
(012) ldb      [27]
(013) jeq      #0x6            jt 14  jf 43
(014) ldh      [24]
(015) jset     #0x1fff         jt 43 jf 16
(016) ldxb     4*([18]&0xf)
(017) ldh      [x + 18]
(018) jeq      #0x64           jt 22jf 19
(019) ldxb     4*([18]&0xf)
(020) ldh      [x + 20]
(021) jeq      #0x64           jt 22jf 43
```

```
(022) ldh      [16]
(023) jeq      #0x86dd         jt 24  jf 30
(024) ldb      [24]
(025) jeq      #0x6            jt 26     jf 30
(026) ldh      [58]
(027) jeq      #0xc8           jt 42     jf 28
(028) ldh      [60]
(029) jeq      #0xc8           jt 42     jf 30
(030) ldh      [16]
(031) jeq      #0x800          jt 32    jf 43
(032) ldb      [27]
(033) jeq      #0x6            jt 34     jf 43
(034) ldh      [24]
(035) jset     #0x1fff         jt 43     jf 36
(036) ldxb     4*([18]&0xf)
(037) ldh      [x + 18]
(038) jeq      #0xc8           jt 42     jf 39
(039) ldxb     4*([18]&0xf)
(040) ldh      [x + 20]
(041) jeq      #0xc8           jt 42     jf 43
(042) ret      #65535
(043) ret      #0
```

Redundant and Inefficient

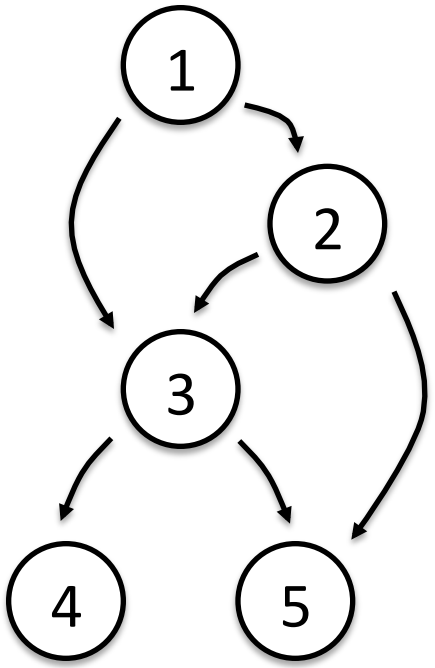


Enter Optimization

- Post-process generated code with optimization techniques
  - *libpcap/optimize.c*
- Leveraged a bunch of well known techniques from my compilers course

The Dominator Concept

- A well-known technique global data flow optimization at the time used dominators



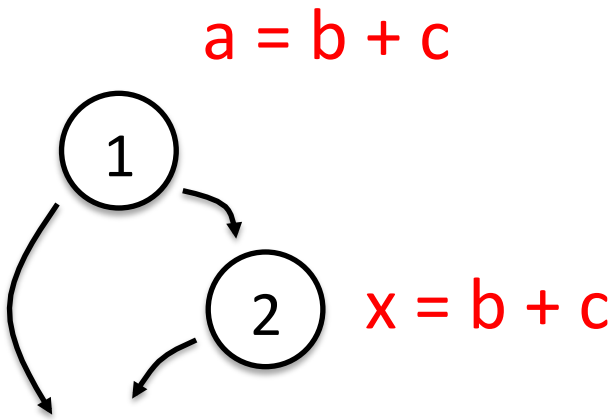
DOM(1) = { 2, 3, 4, 5 }

DOM(2) = { }

DOM(3) = { 4, 5}

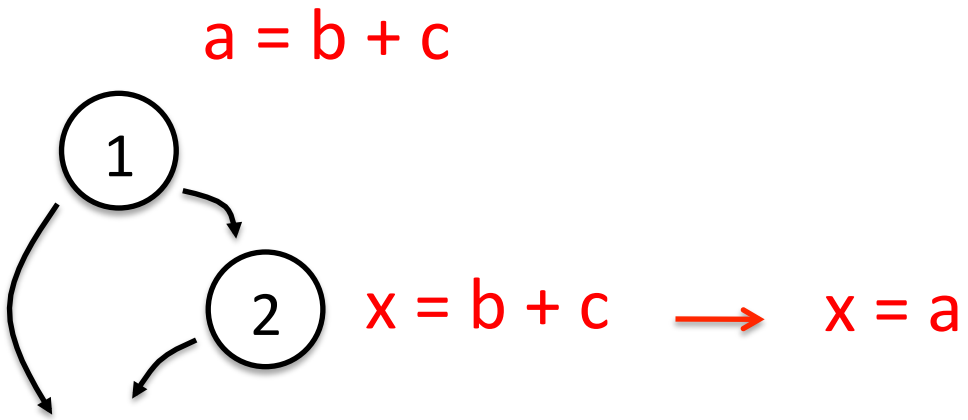
DOM Example

- Global common sub-expression elimination



DOM Example

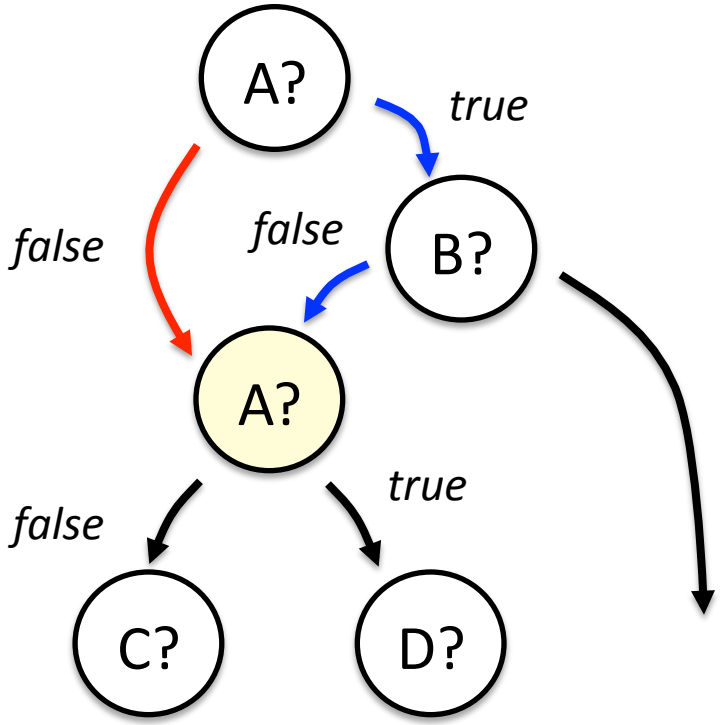
- Global common sub-expression elimination



- 2 in DOM(1) => variables on entry to 2 same as on exit to 1, so we can replace b + c with a

Not Quite Enough...

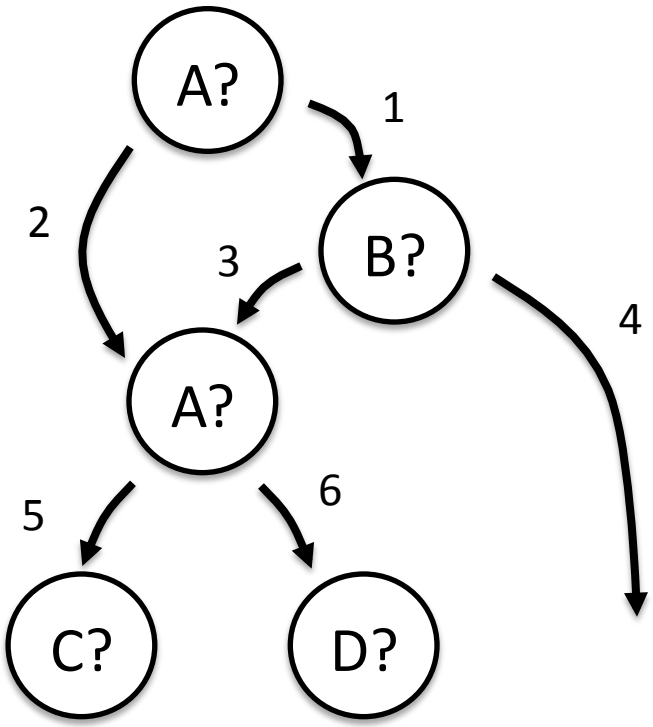
- While these traditional techniques are used by the BPF optimizer, they weren't enough...



Knowing the top node dominates yellow node doesn't let us eliminate the redundant test for A at the yellow node because either the red or blue path could happen

Edge Dominators

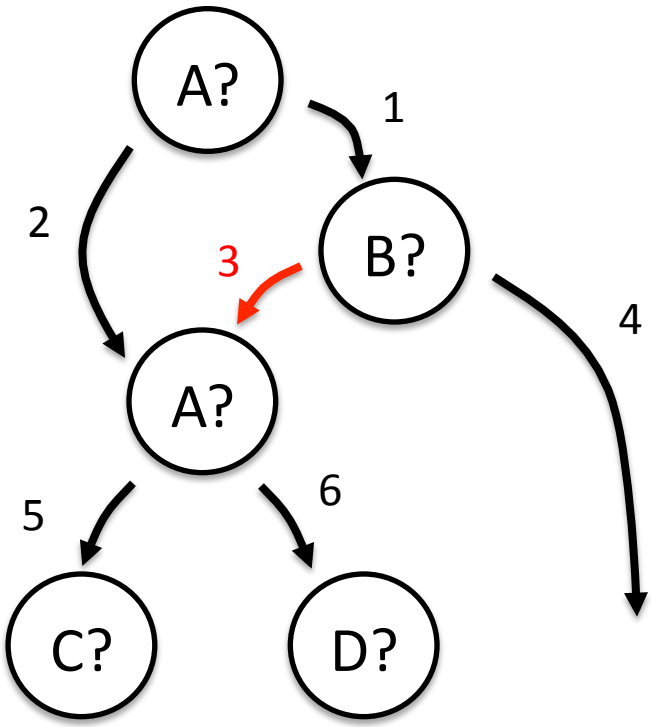
- But if we look at edge relationships instead of node relationships, we can solve the problem



EDOM(1) = { 3, 4 }
EDOM(2) = { }
EDOM(3) = { }

Edge Optimization

- With this knowledge, we can safely move edges to optimize the code...

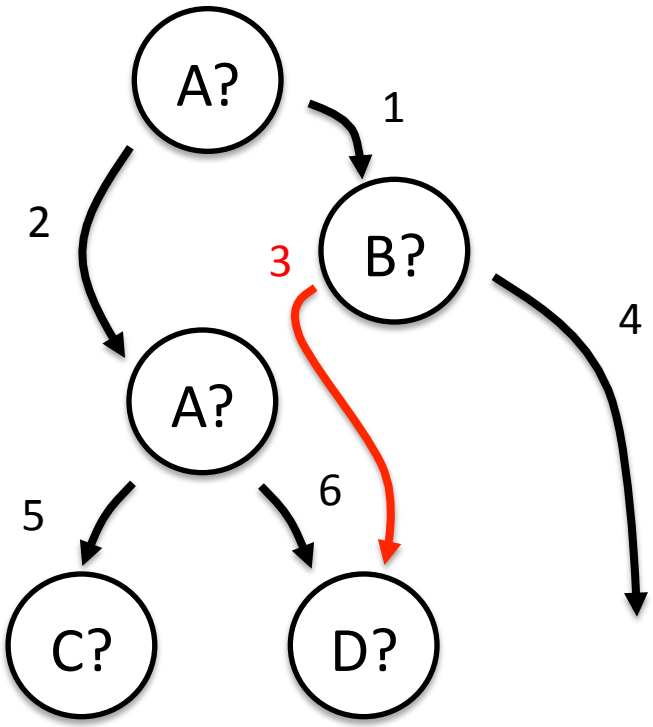


EDOM(1) = { 3, 4 }
EDOM(2) = { }
EDOM(3) = { }

3 in EDOM(1) =>
know A is true at 3 =>
we can move 3 past
second check

Edge Optimization

- With this knowledge, we can safely move edges to optimize the code...

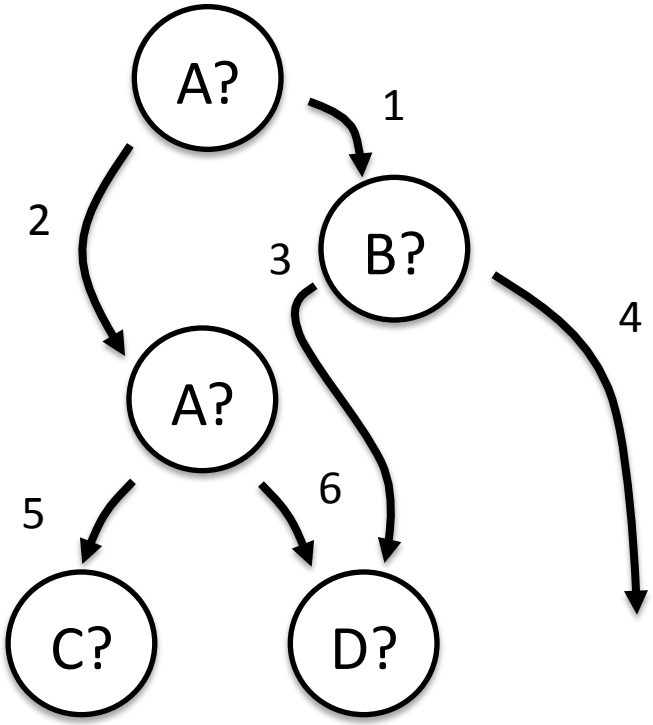


EDOM(1) = { 3, 4 }
EDOM(2) = { }
EDOM(3) = { }

3 in EDOM(1) =>
know A is true at 3 =>
we can move 3 past
second check

Edge Optimization

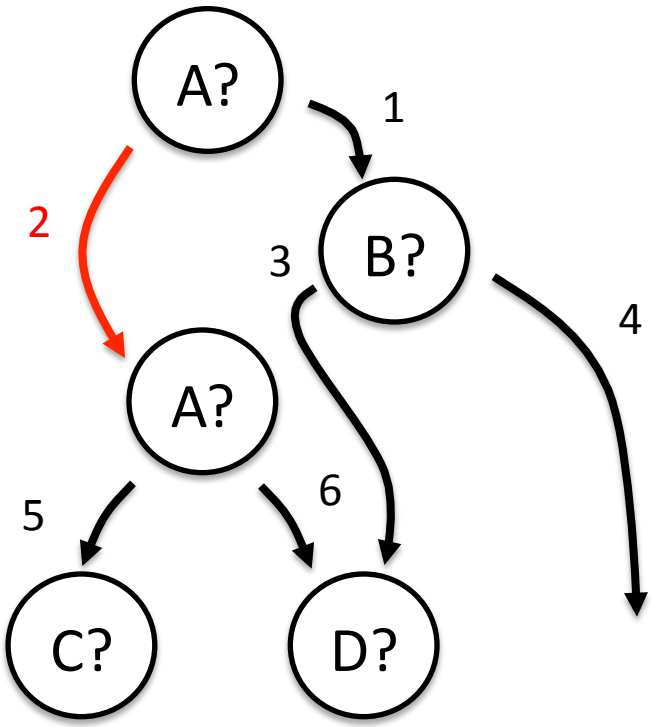
- Movements create new opportunities. Update EDOM and repeat...



EDOM(1) = { 3, 4 }
EDOM(2) = { }
EDOM(3) = { }

Edge Optimization

- Movements create new opportunities. Update EDOM and repeat...

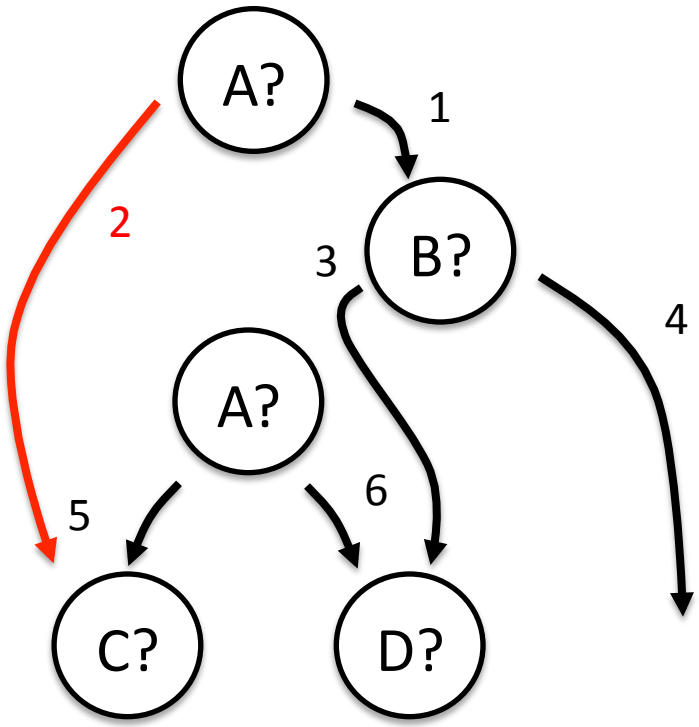


EDOM(1) = { 3, 4 }
EDOM(2) = { 5, 6}
EDOM(3) = { }

5 in EDOM(2) =>
know A is false at 5 =>
we can move 2

Edge Optimization

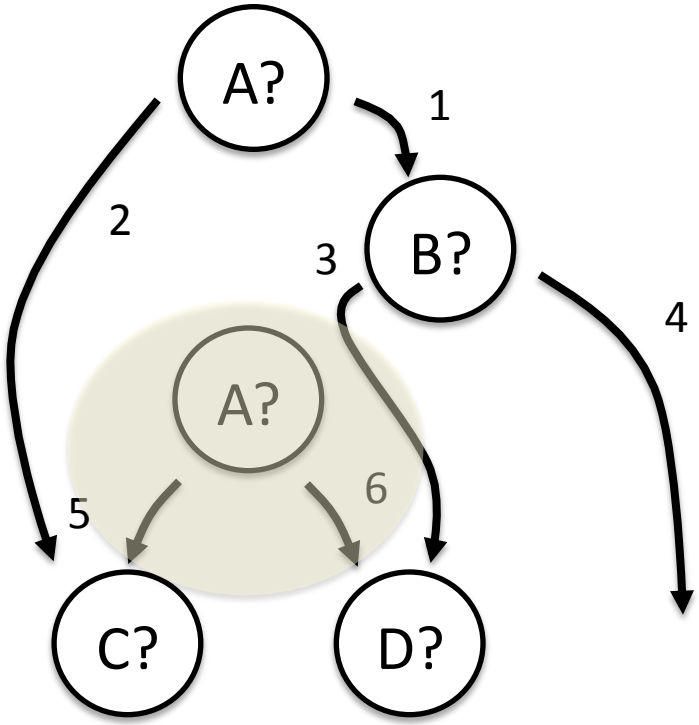
- Movements create new opportunities. Update EDOM and repeat...



EDOM(1) = { 3, 4 }
EDOM(2) = { 5, 6}
EDOM(3) = { }

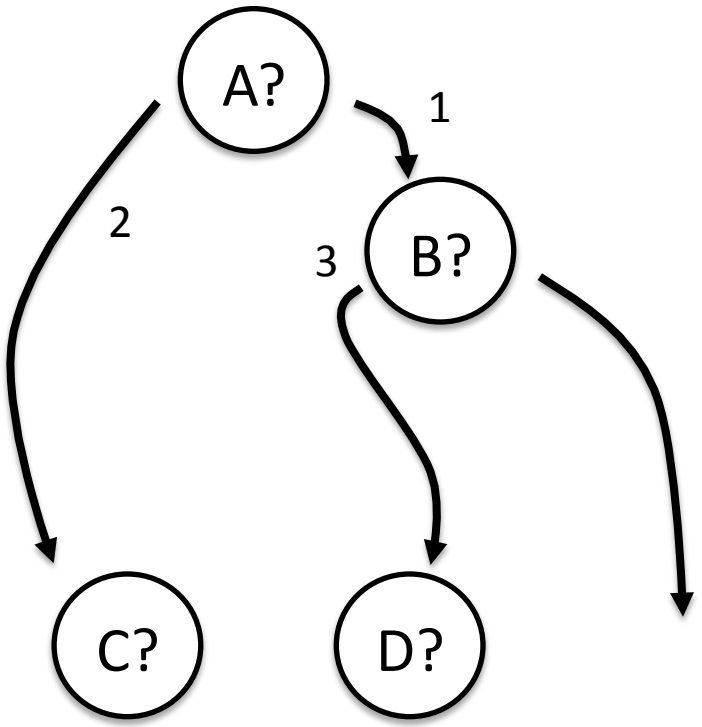
Edge Optimization

- Now we can delete unreachable code....



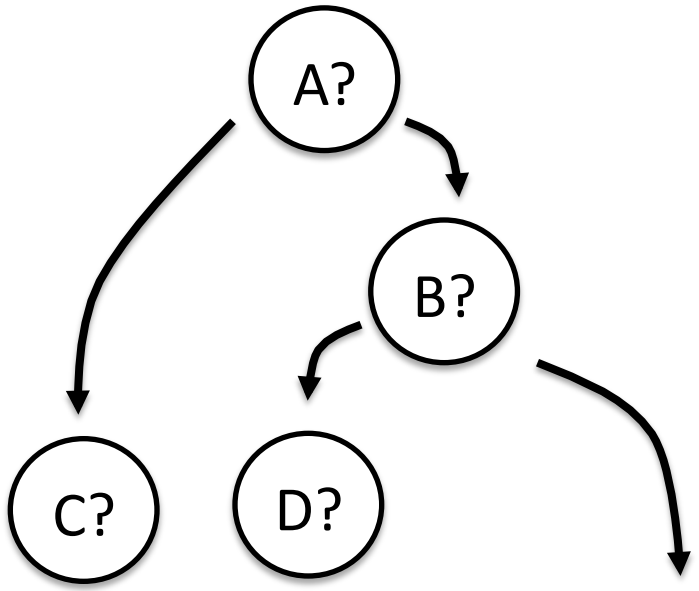
Edge Optimization

- Now we can delete unreachable code....

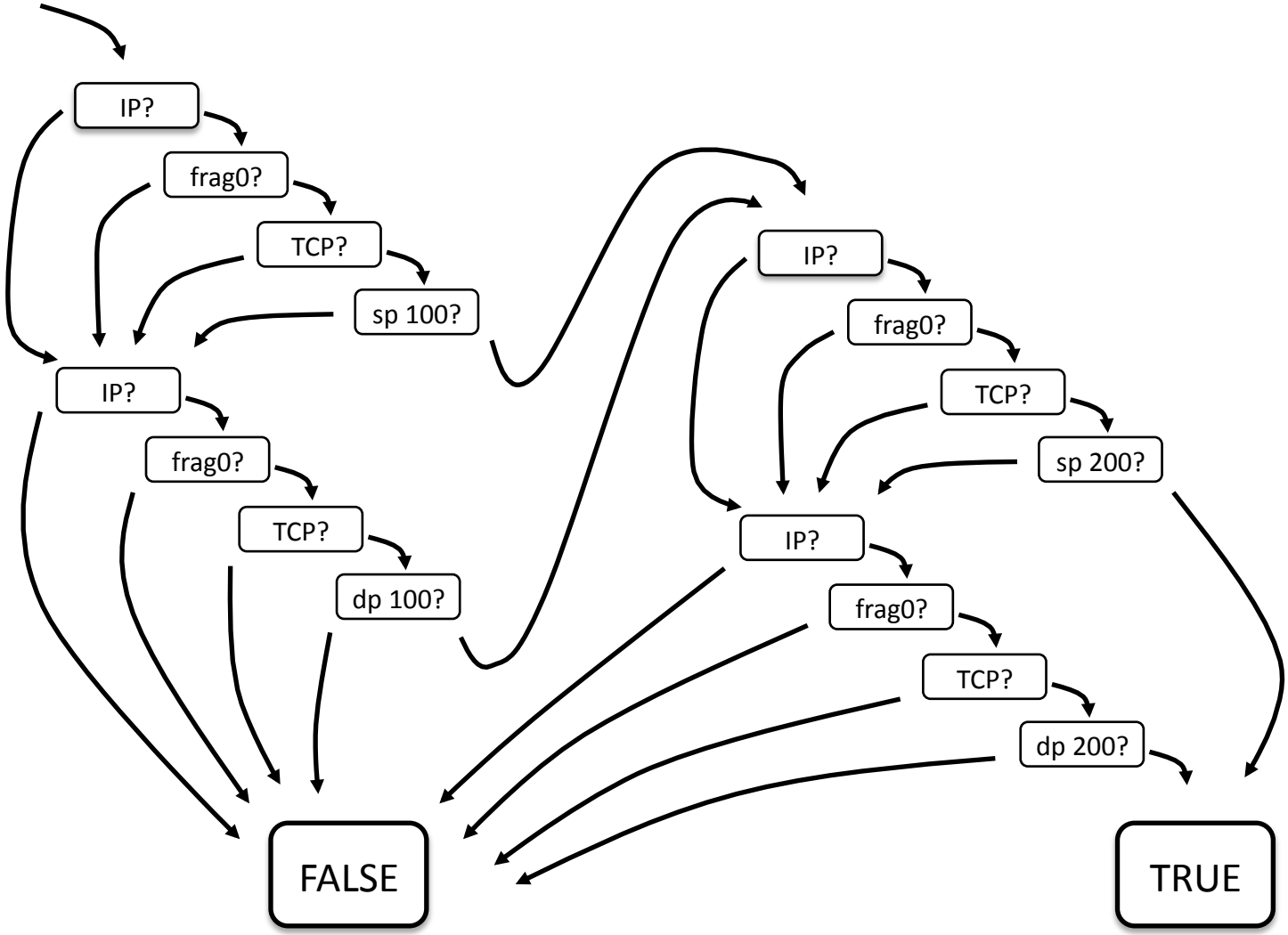


Edge Optimization

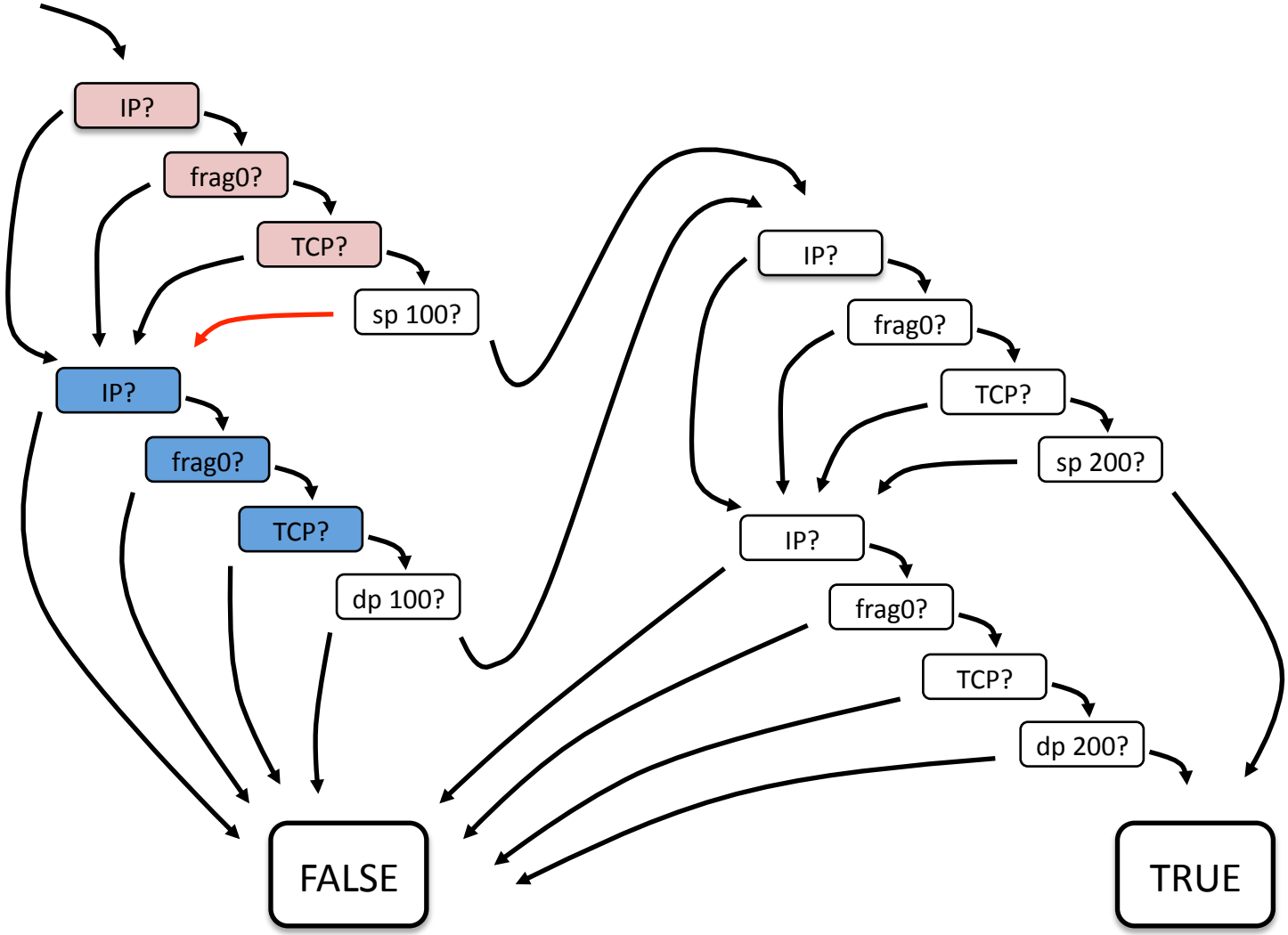
- and simplify the graph...



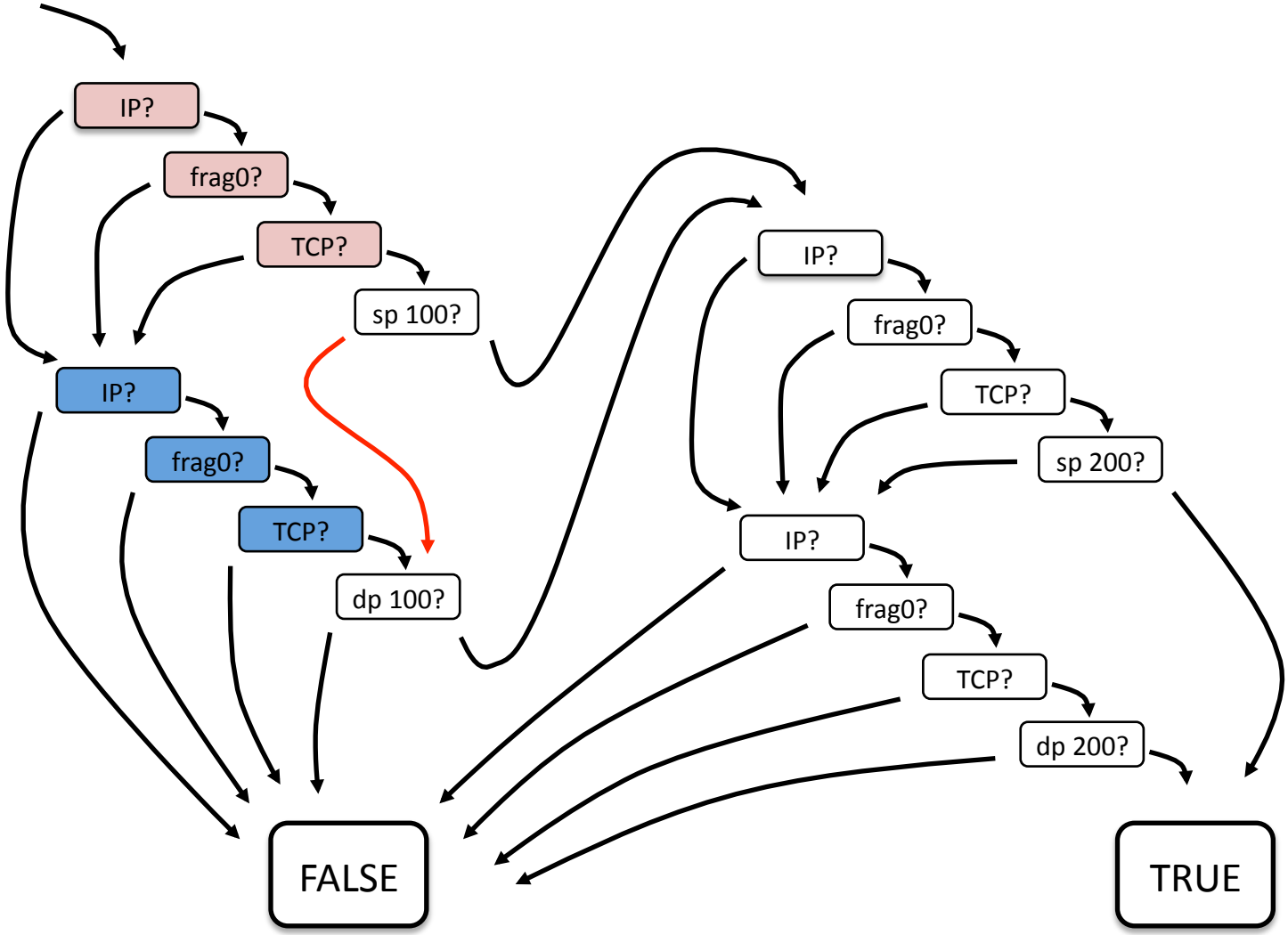
tcp port 100 and 200



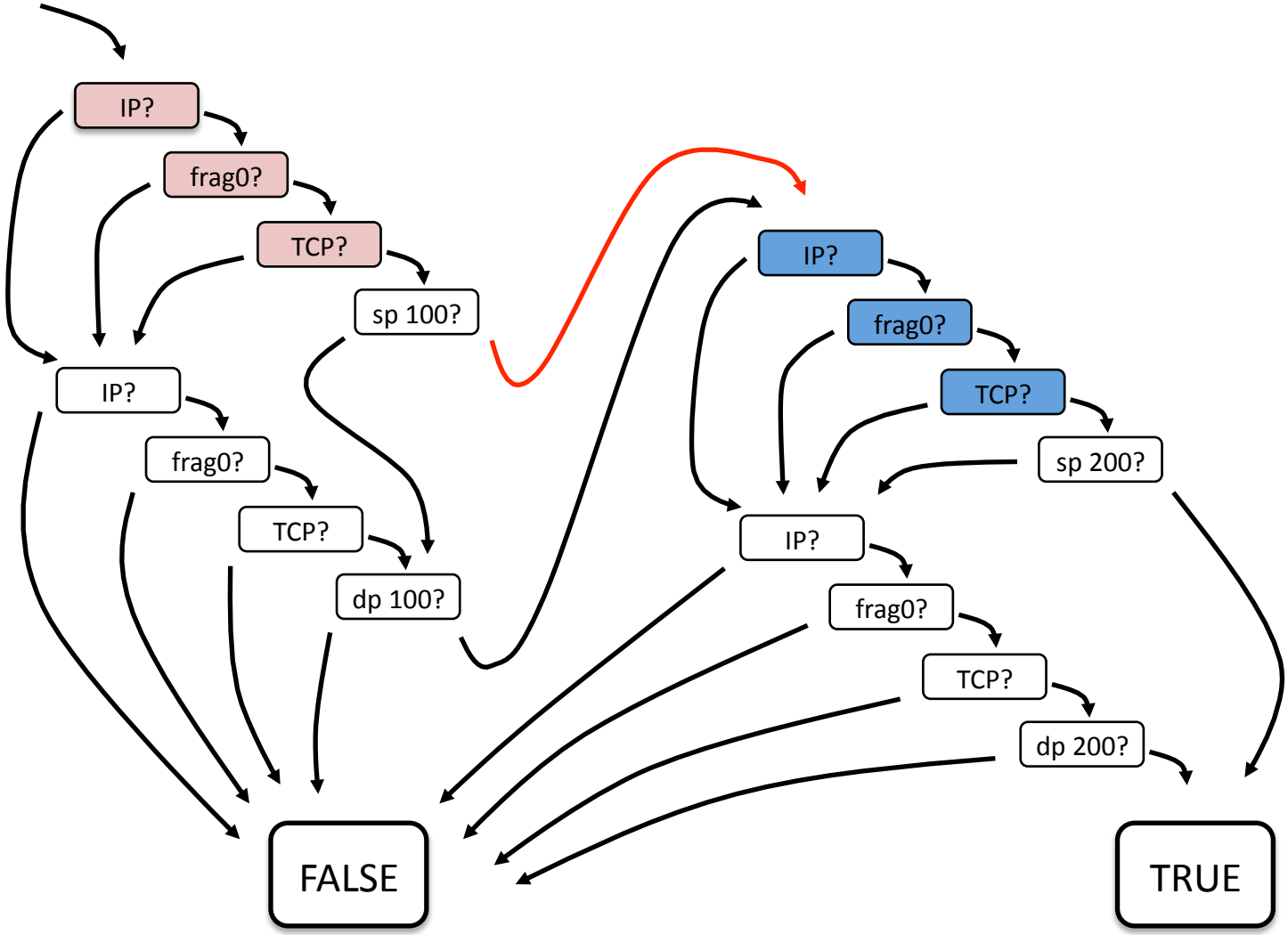
tcp port 100 and 200



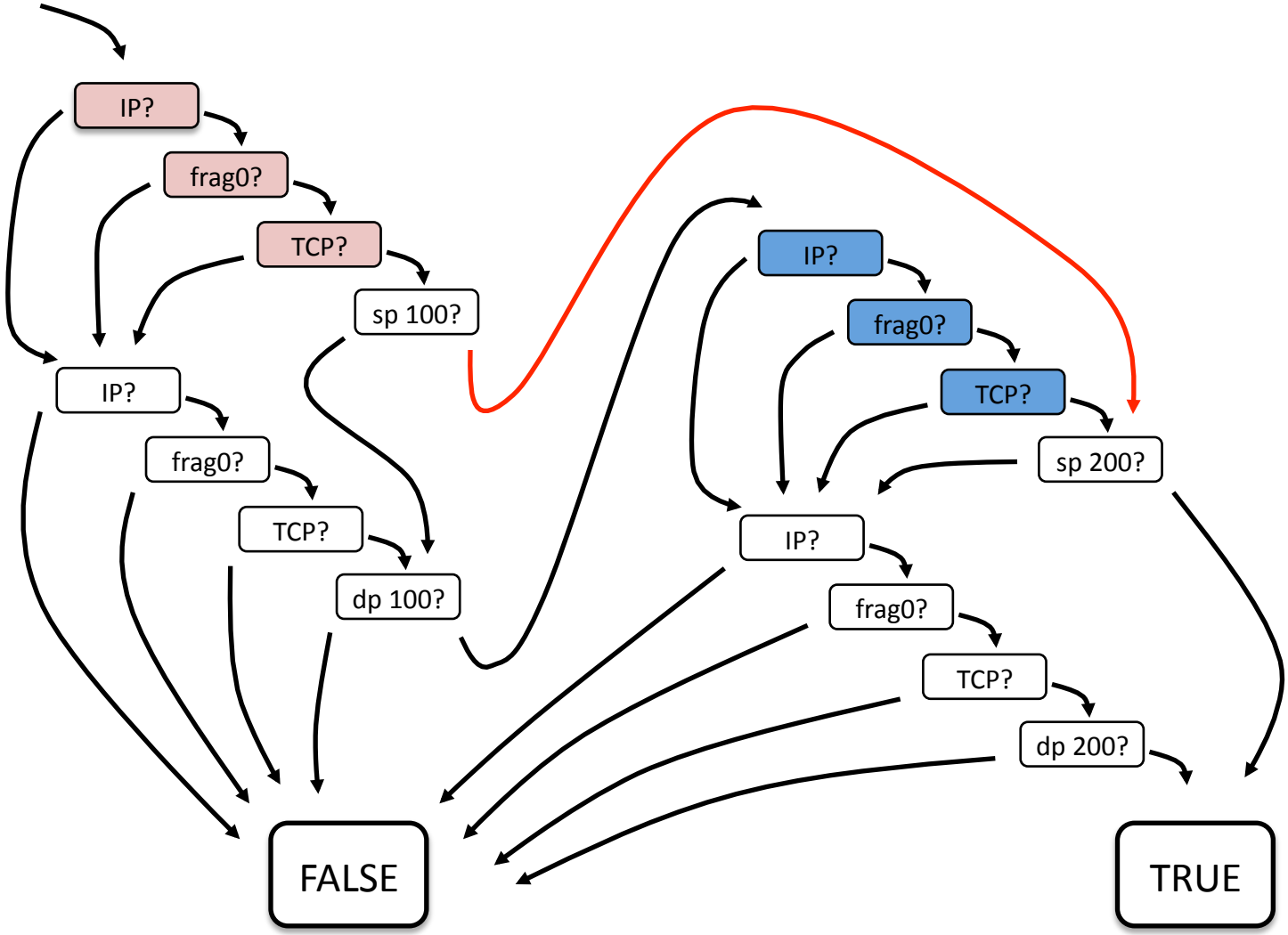
tcp port 100 and 200



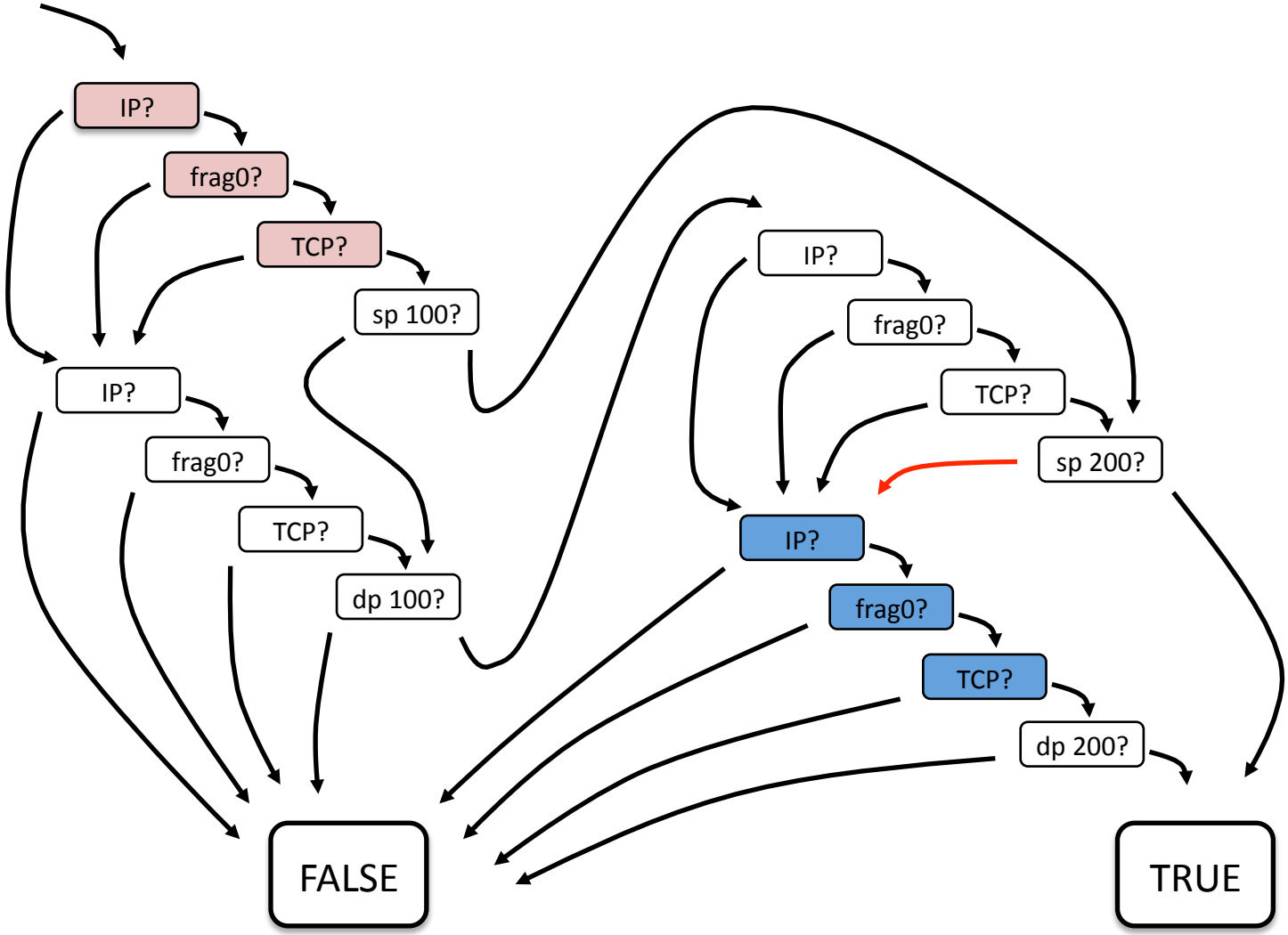
tcp port 100 and 200



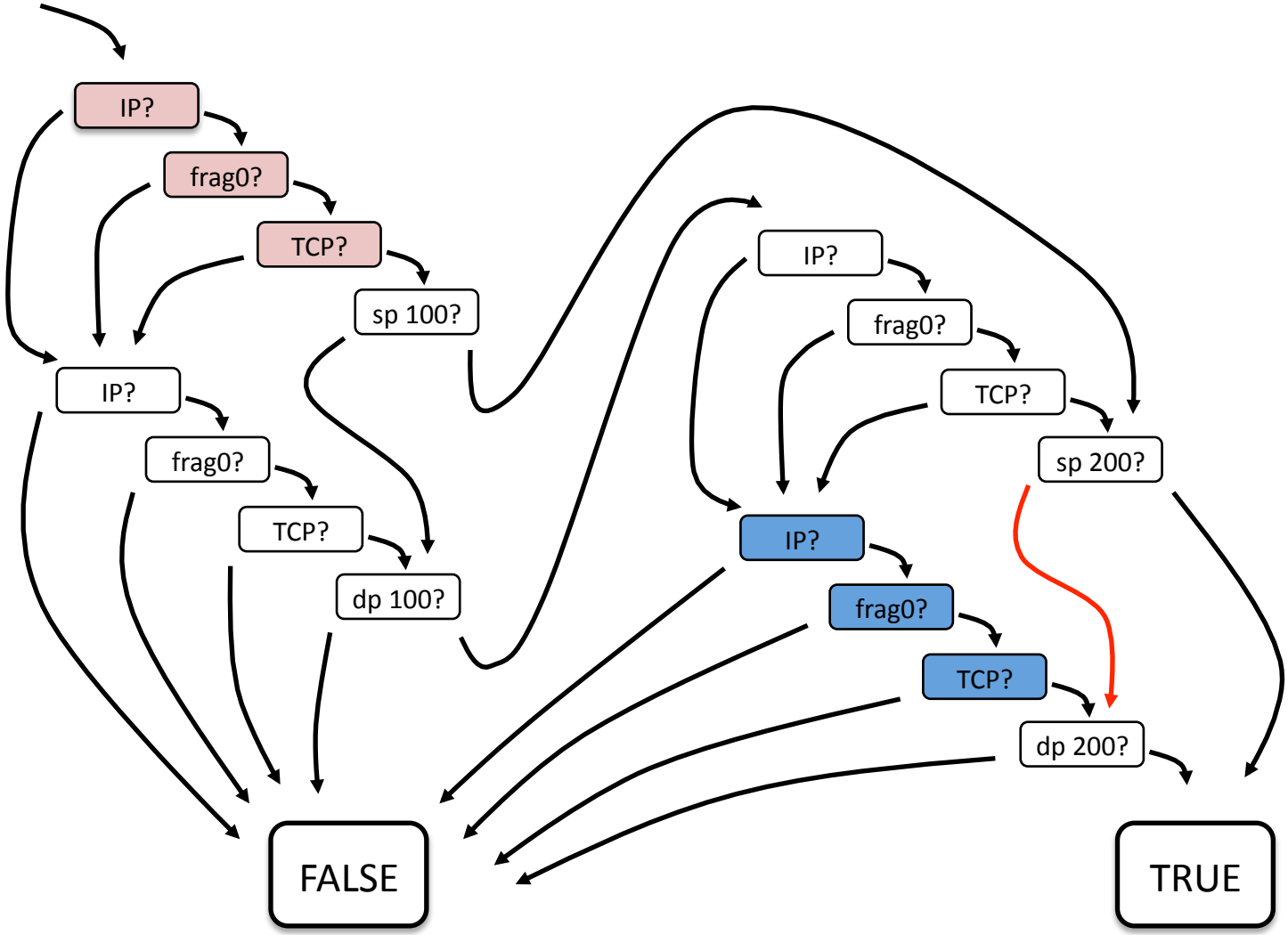
tcp port 100 and 200



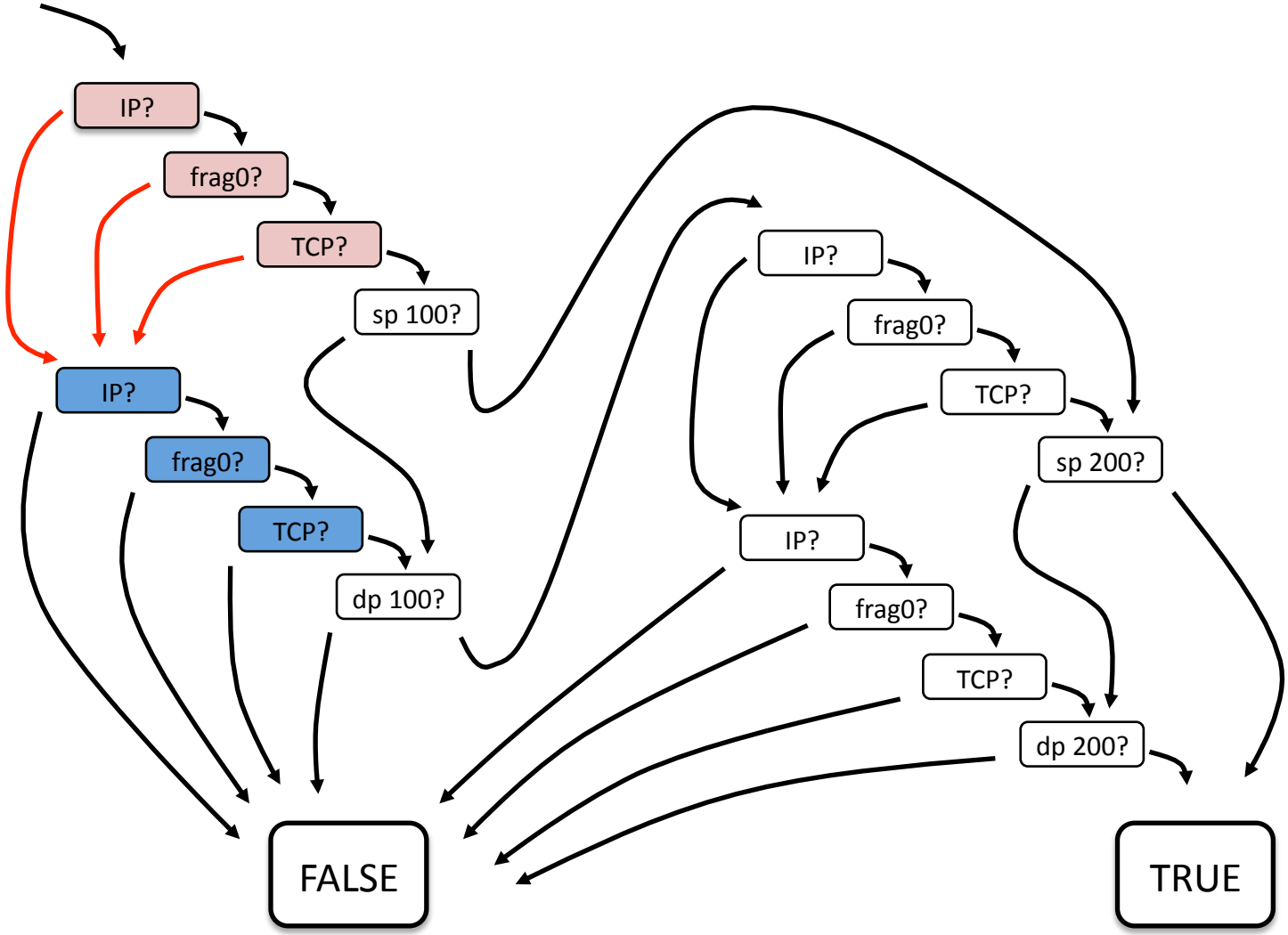
tcp port 100 and 200



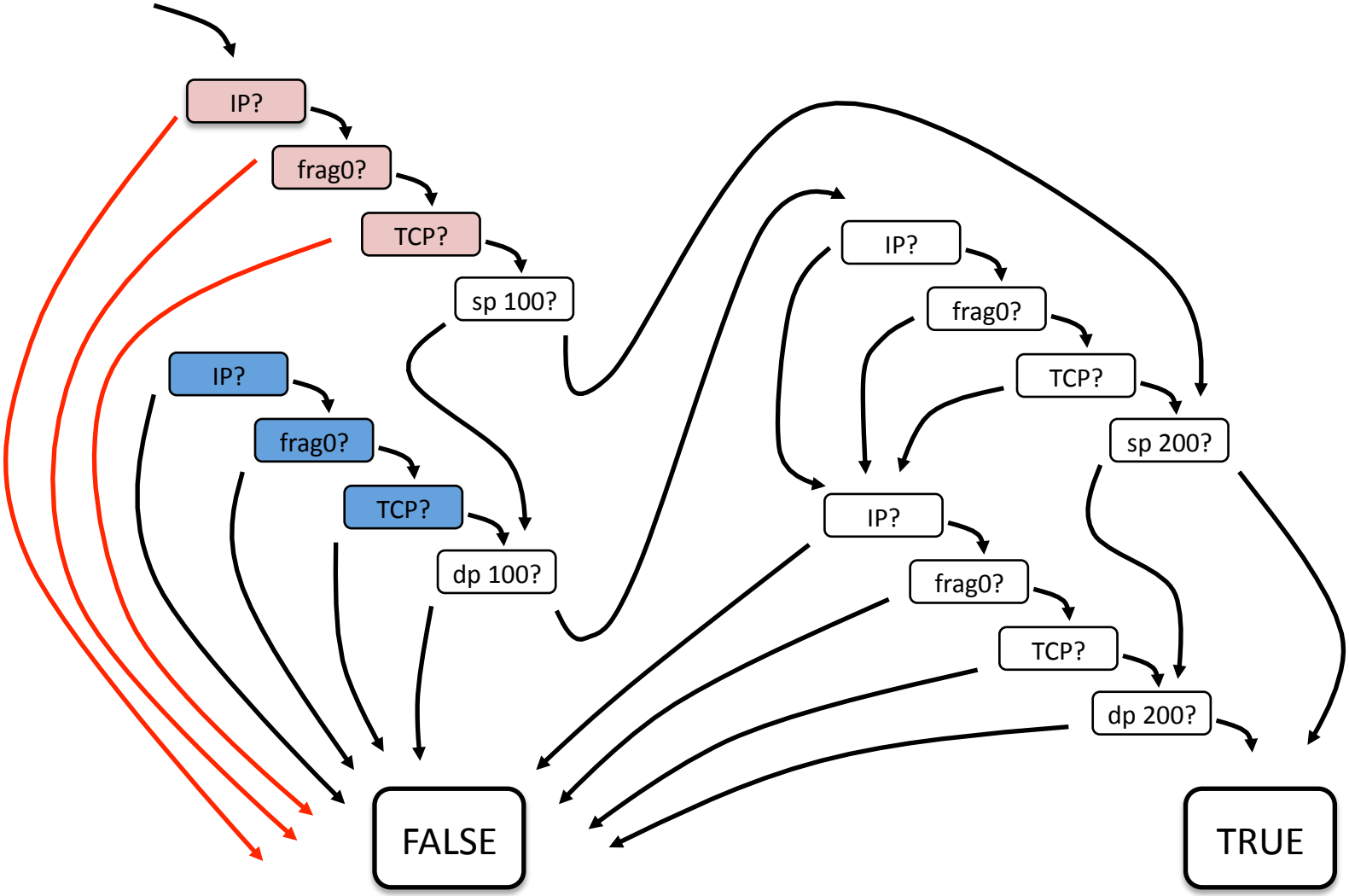
tcp port 100 and 200



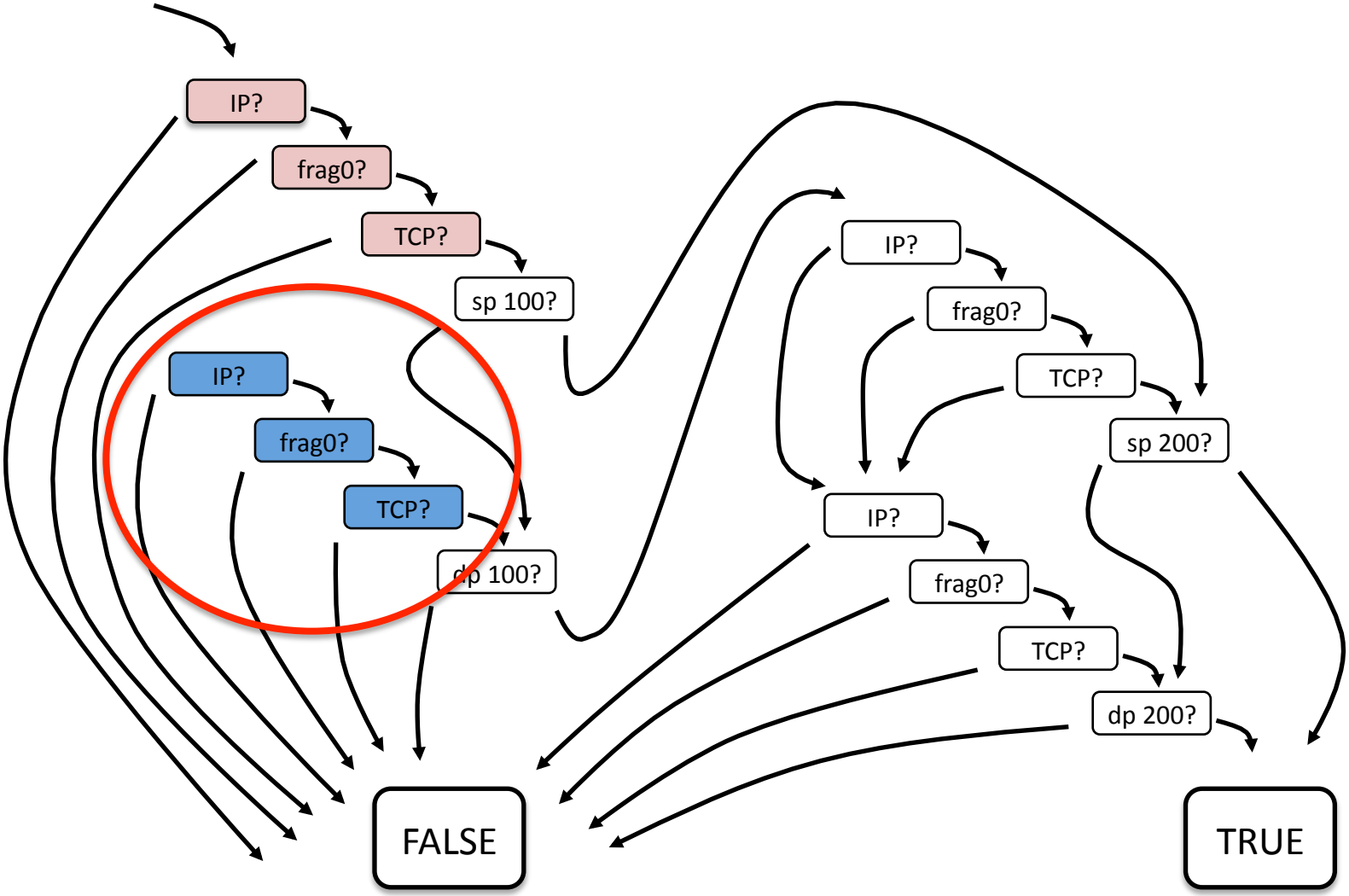
tcp port 100 and 200



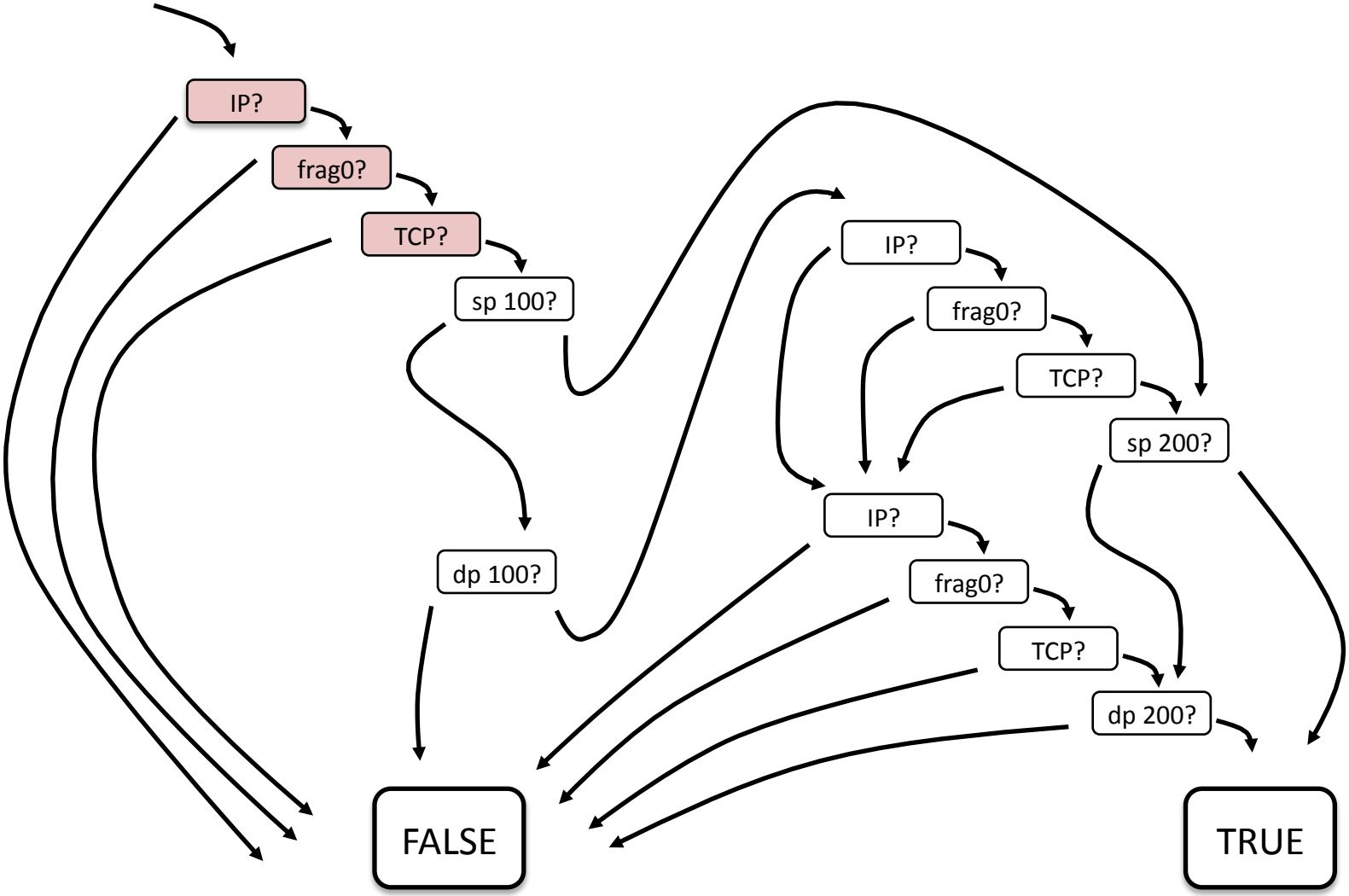
tcp port 100 and 200



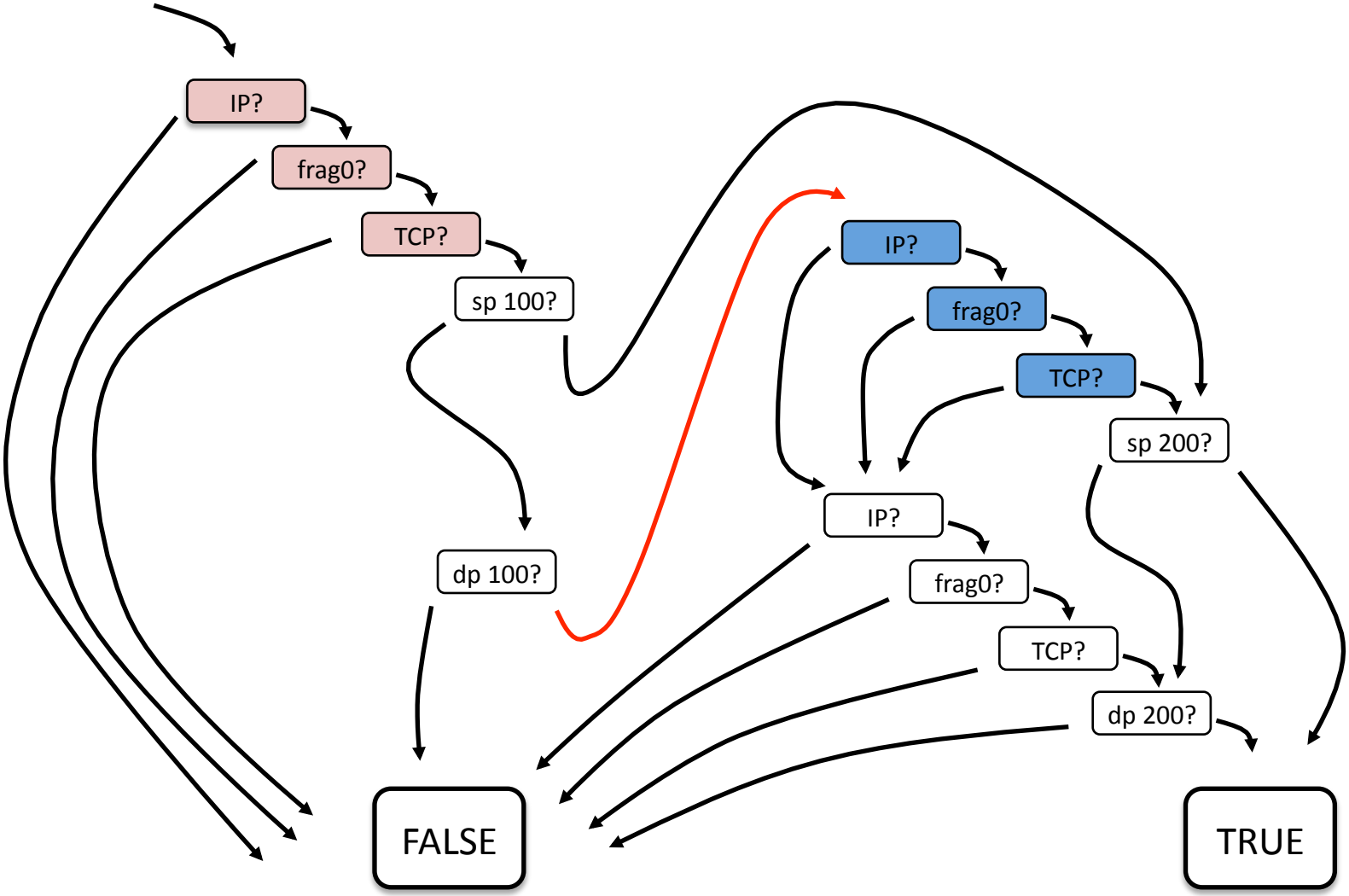
tcp port 100 and 200



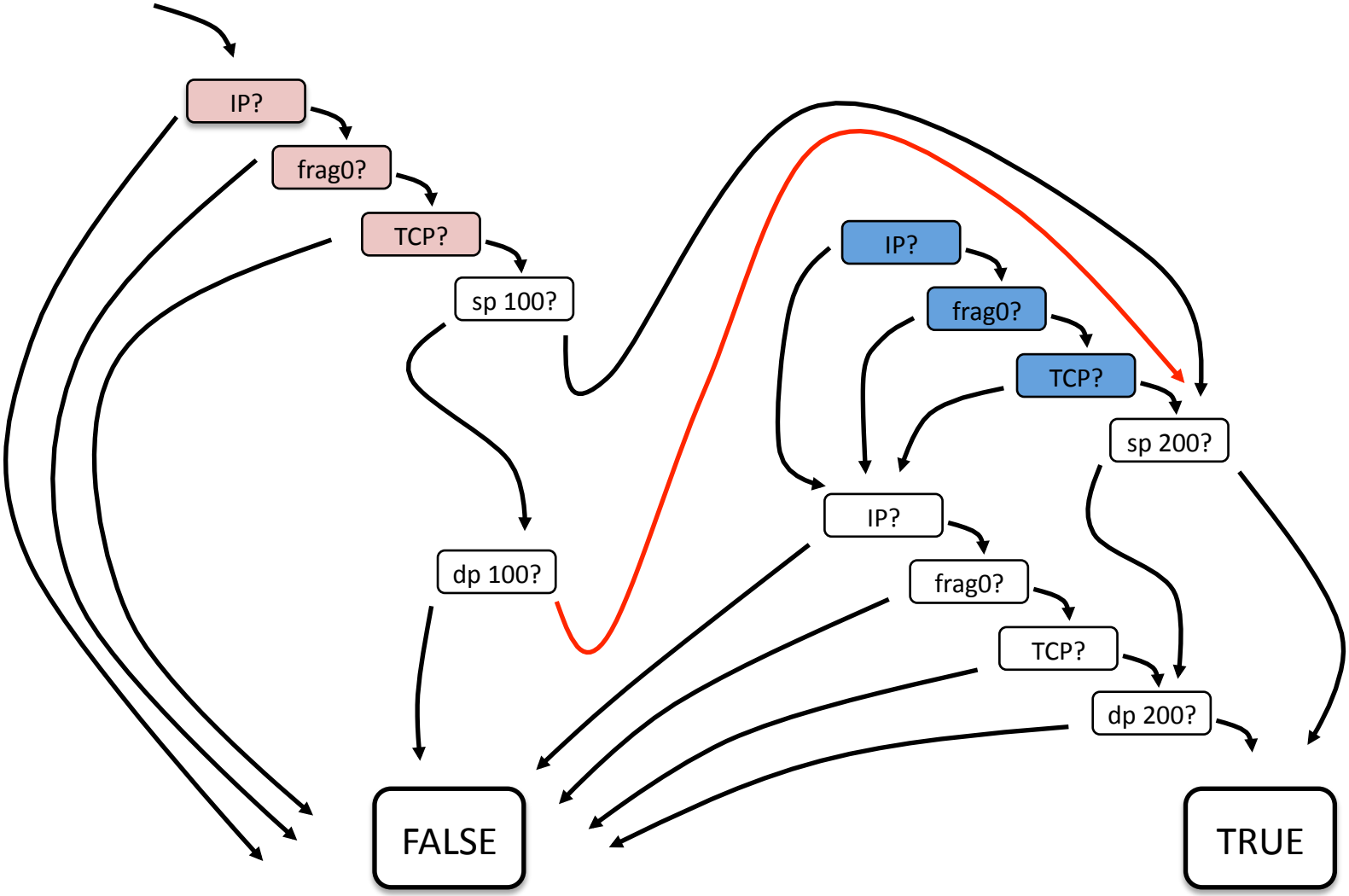
tcp port 100 and 200



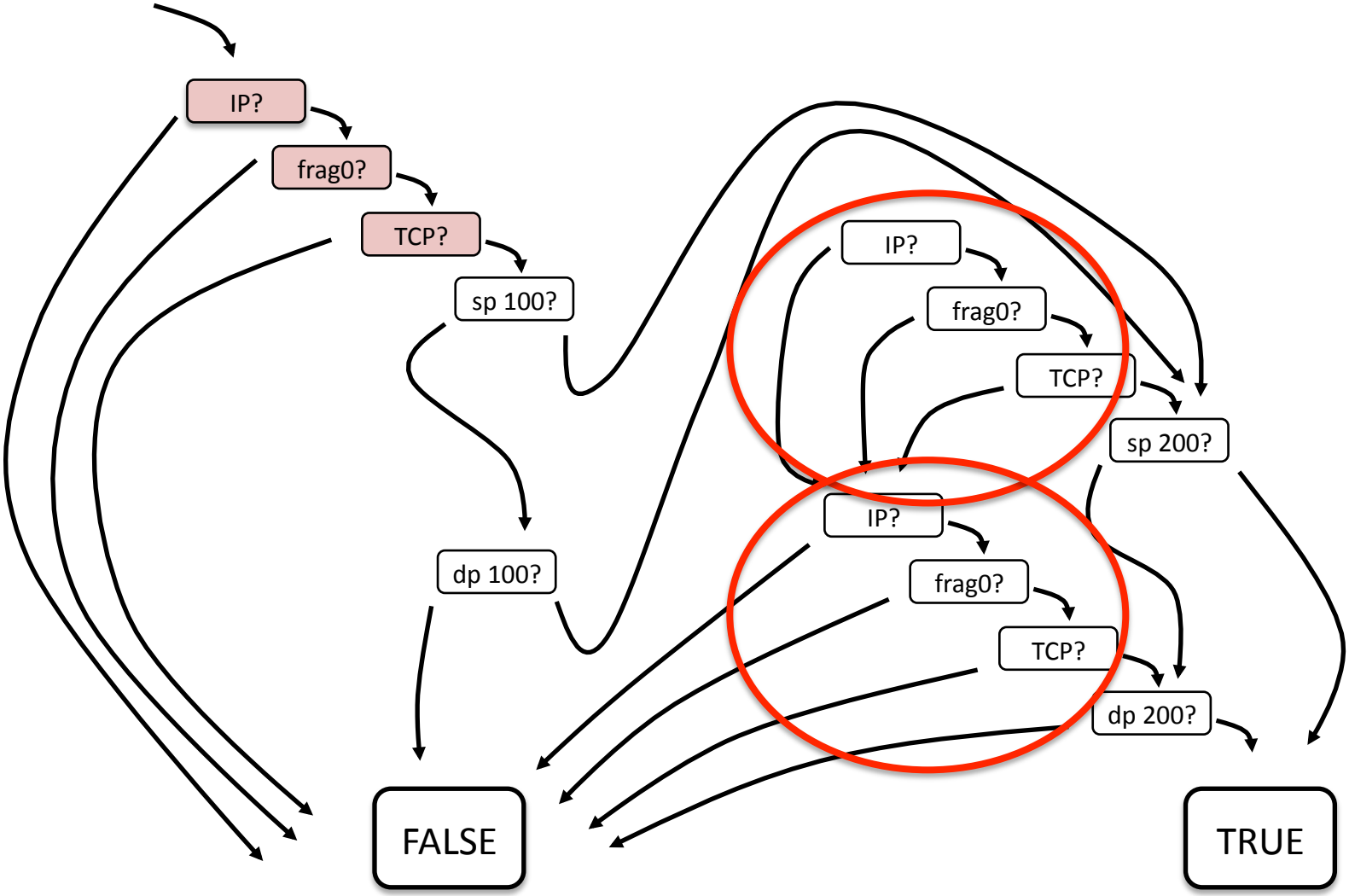
tcp port 100 and 200



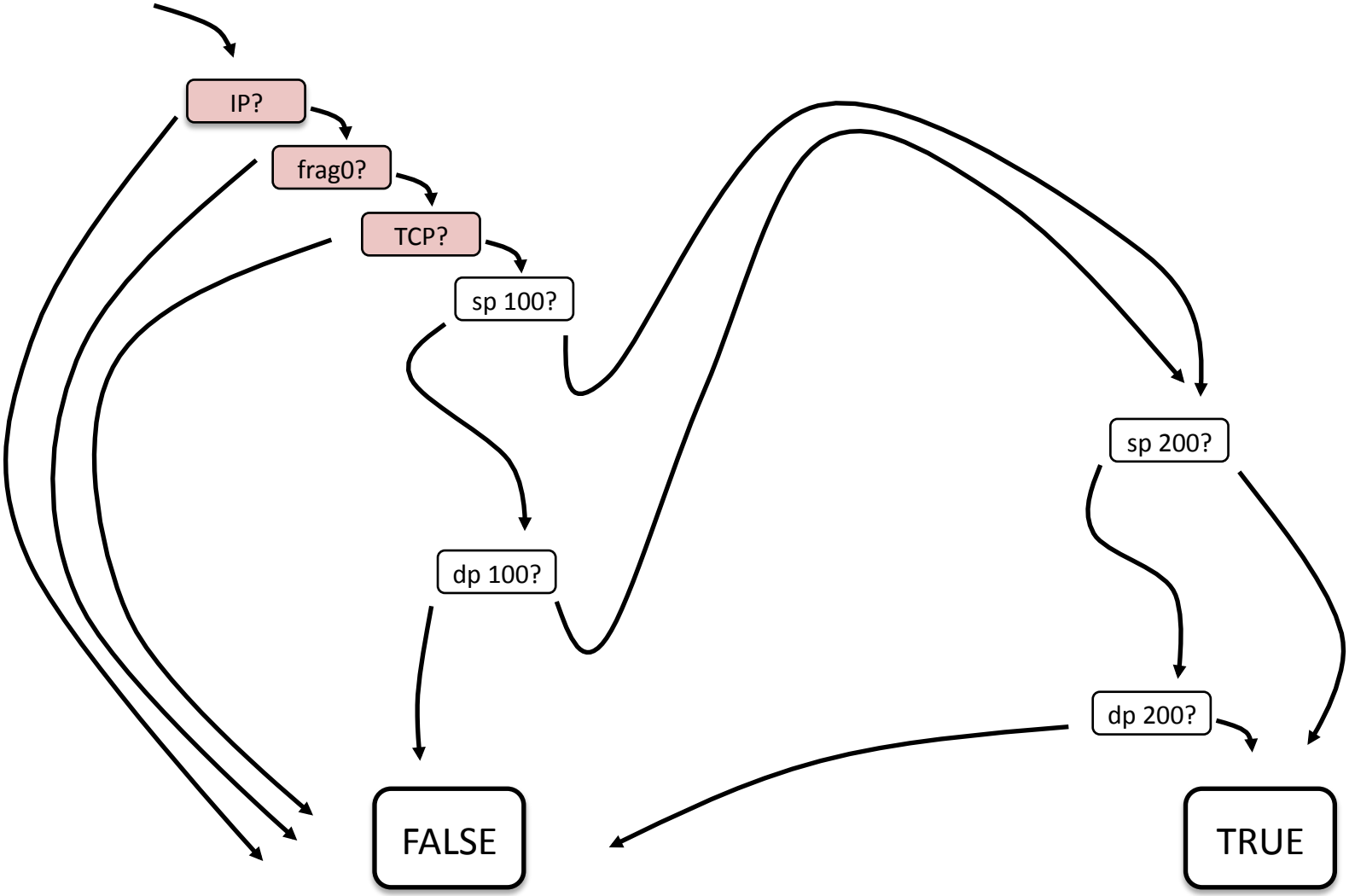
tcp port 100 and 200



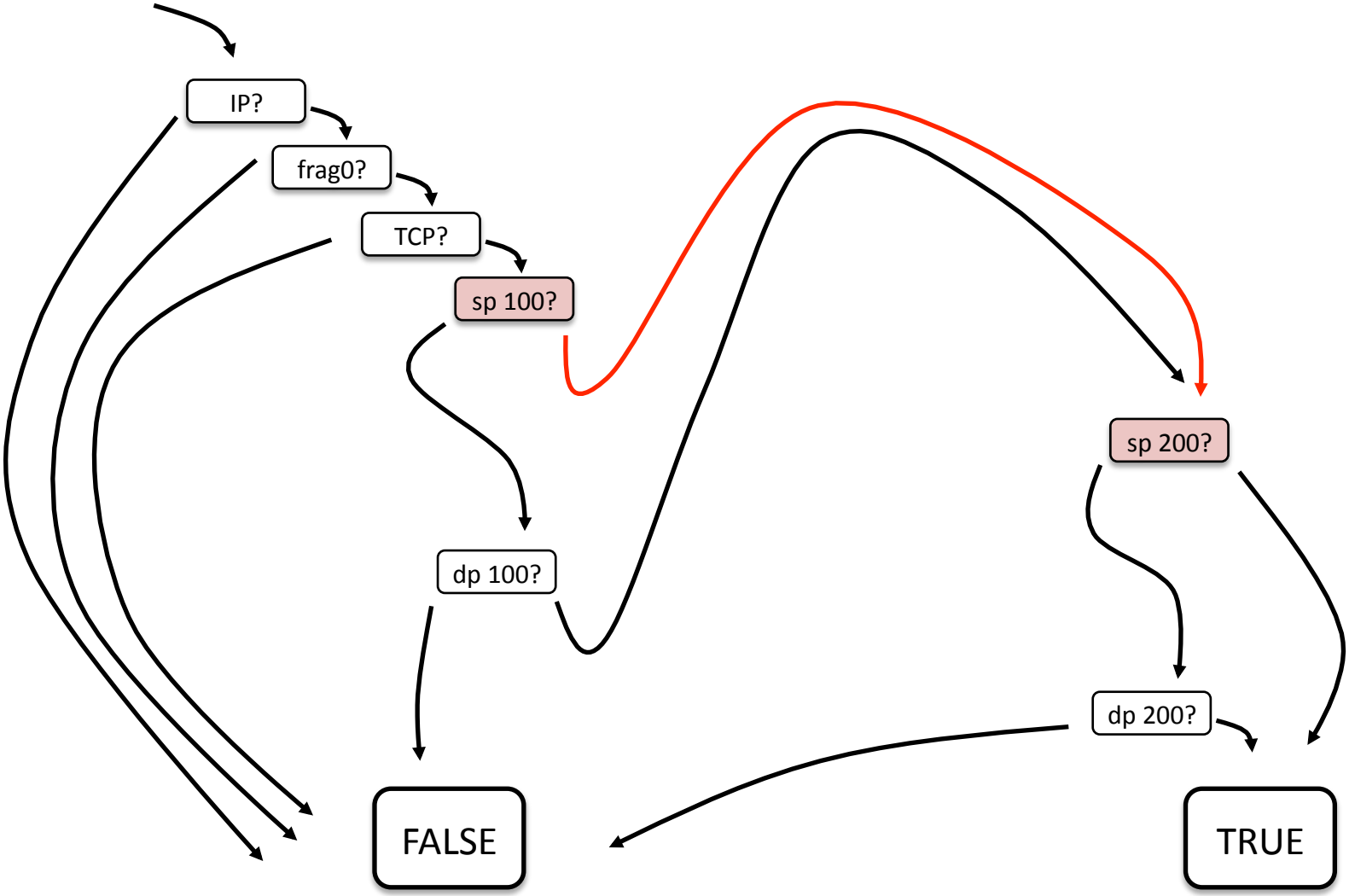
tcp port 100 and 200



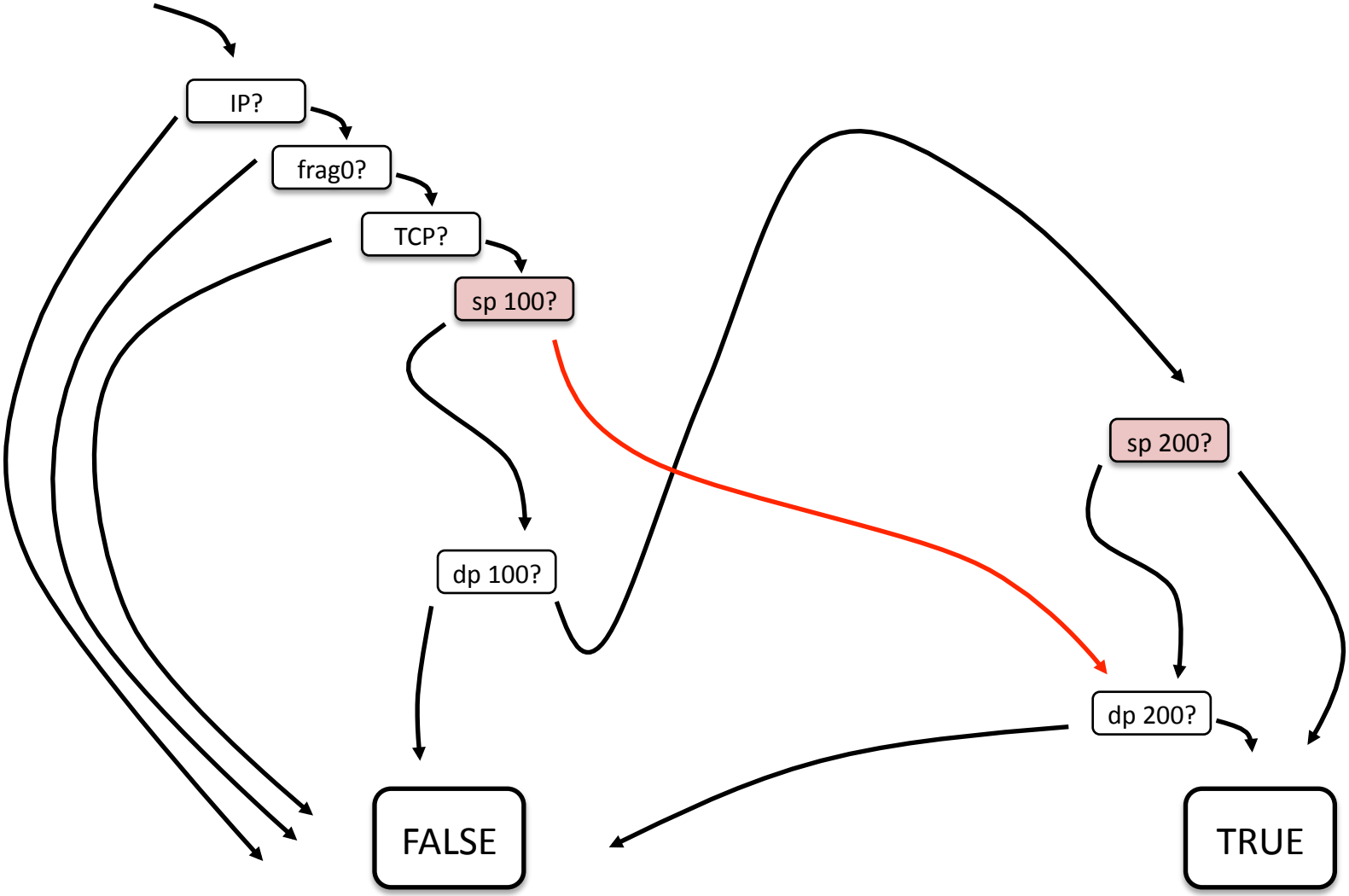
tcp port 100 and 200



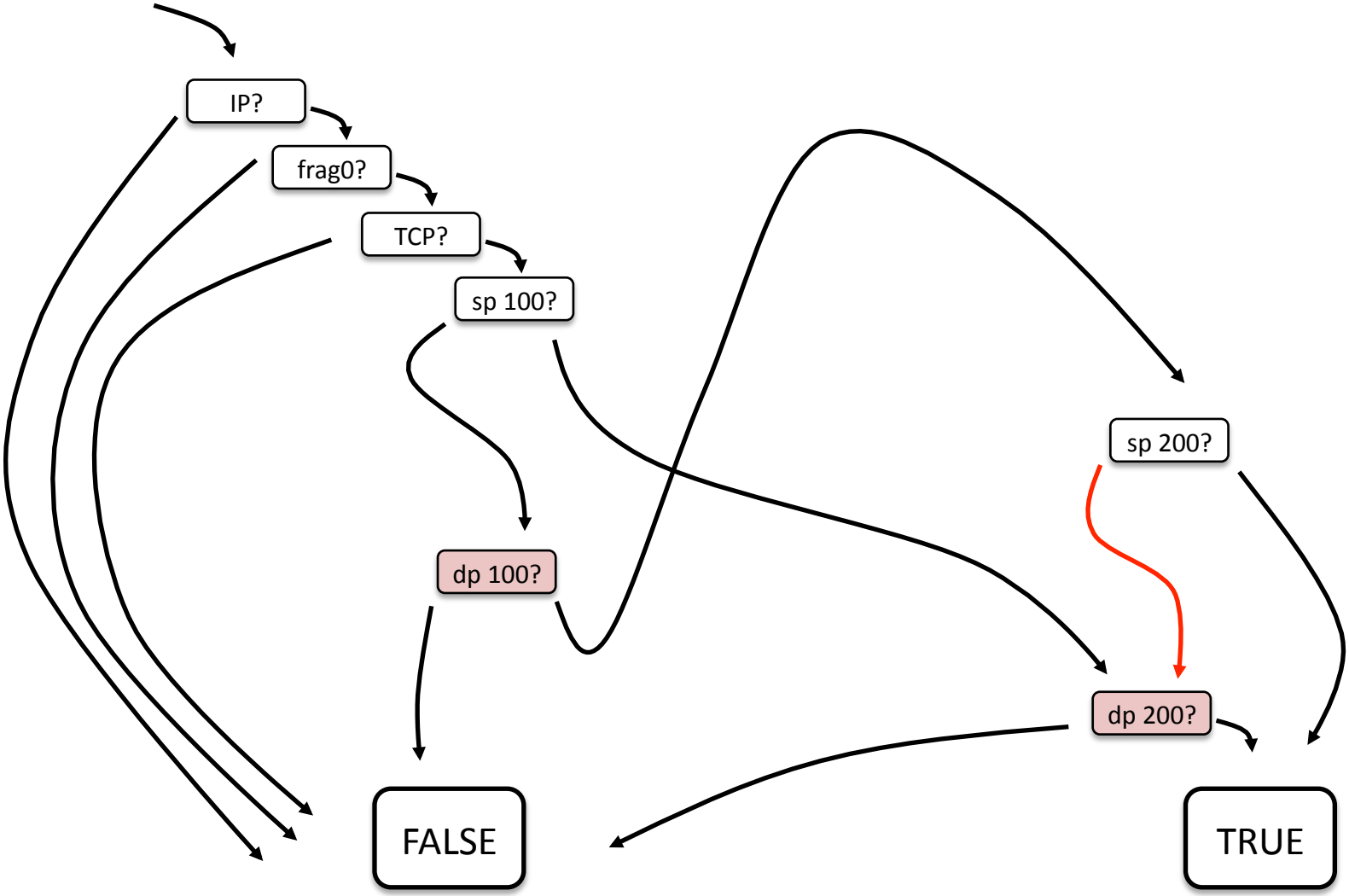
tcp port 100 and 200



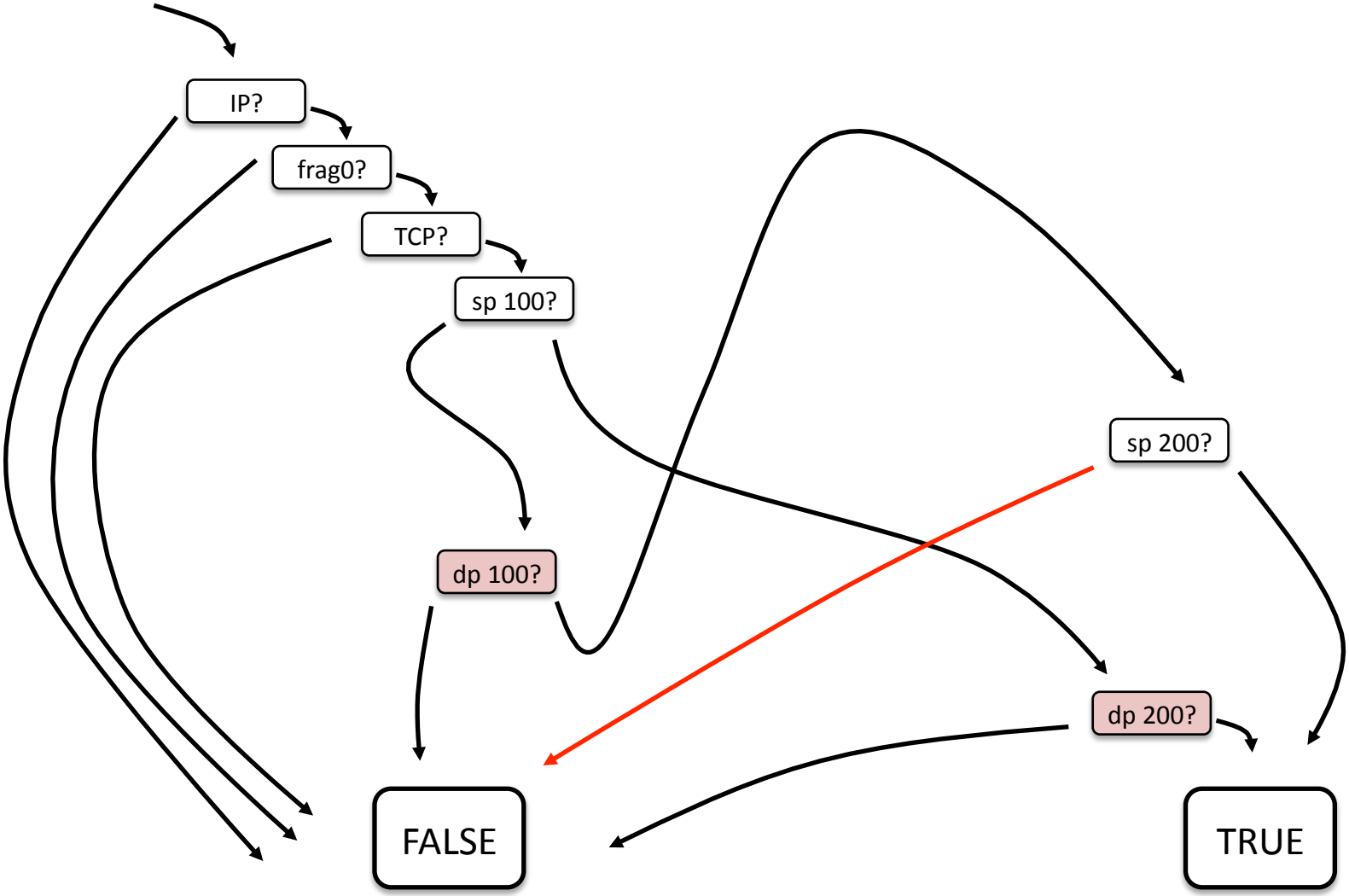
tcp port 100 and 200



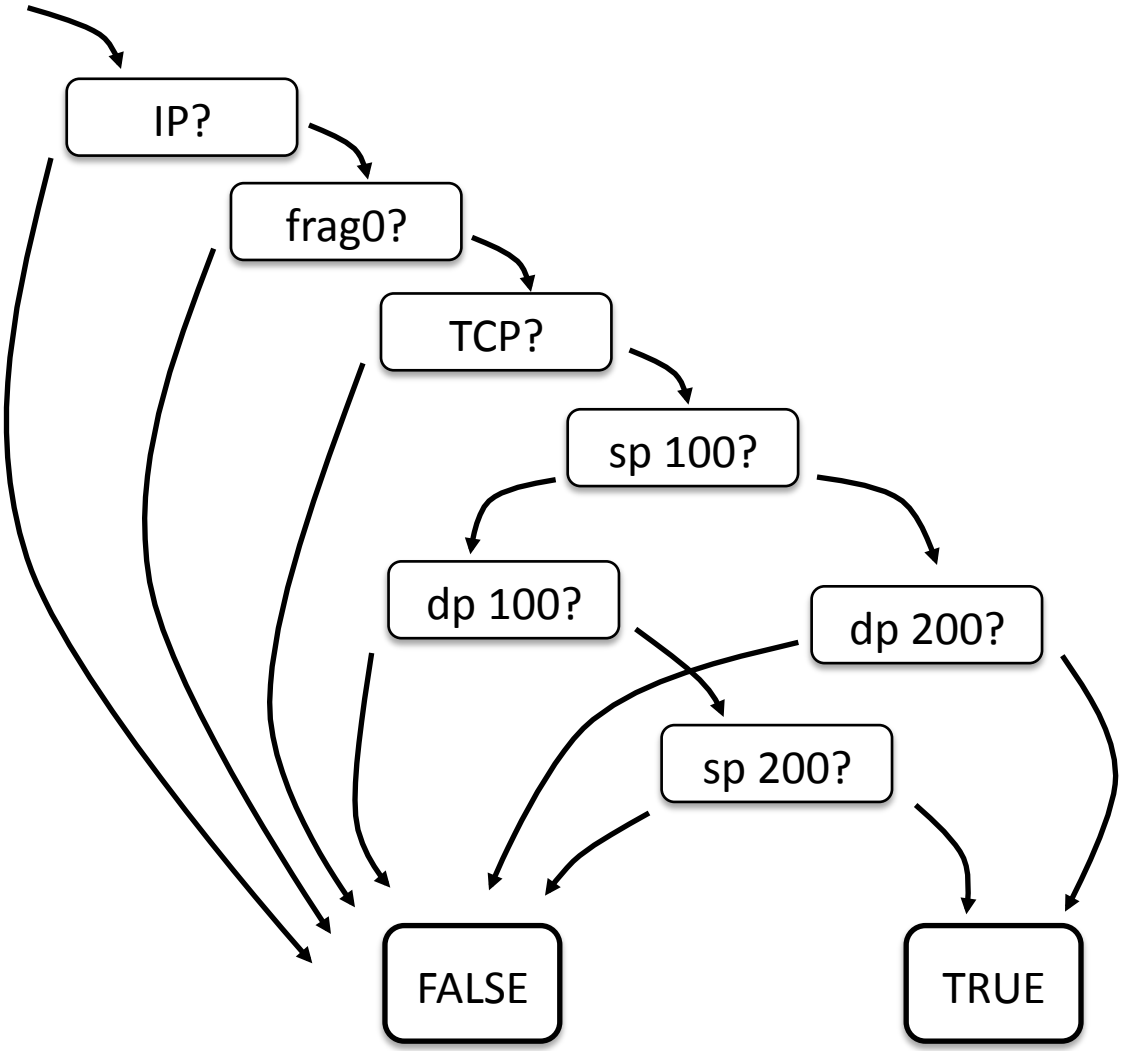
tcp port 100 and 200



tcp port 100 and 200



tcp port 100 and 200



Before

```
(000) ldh      [16]
(001) jeq      #0x800          jt 2 jf 43
(002) ldh      [16]
(003) jeq      #0x86dd         jt 4 jf 10
(004) ldb      [24]
(005) jeq      #0x6            jt 6    jf 10
(006) ldh      [58]
(007) jeq      #0x64           jt 22jf 8
(008) ldh      [60]
(009) jeq      #0x64           jt 22jf 10
(010) ldh      [16]
(011) jeq      #0x800          jt 12 jf 43
(012) ldb      [27]
(013) jeq      #0x6            jt 14  jf 43
(014) ldh      [24]
(015) jset     #0x1fff         jt 43 jf 16
(016) ldxb     4*([18]&0xf)
(017) ldh      [x + 18]
(018) jeq      #0x64           jt 22jf 19
(019) ldxb     4*([18]&0xf)
(020) ldh      [x + 20]
(021) jeq      #0x64           jt 22jf 43
```

```
(022) ldh      [16]
(023) jeq      #0x86dd         jt 24  jf 30
(024) ldb      [24]
(025) jeq      #0x6            jt 26     jf 30
(026) ldh      [58]
(027) jeq      #0xc8           jt 42     jf 28
(028) ldh      [60]
(029) jeq      #0xc8           jt 42     jf 30
(030) ldh      [16]
(031) jeq      #0x800          jt 32    jf 43
(032) ldb      [27]
(033) jeq      #0x6            jt 34     jf 43
(034) ldh      [24]
(035) jset     #0x1fff         jt 43     jf 36
(036) ldxb     4*([18]&0xf)
(037) ldh      [x + 18]
(038) jeq      #0xc8           jt 42     jf 39
(039) ldxb     4*([18]&0xf)
(040) ldh      [x + 20]
(041) jeq      #0xc8           jt 42     jf 43
(042) ret      #65535
(043) ret      #0
```

After

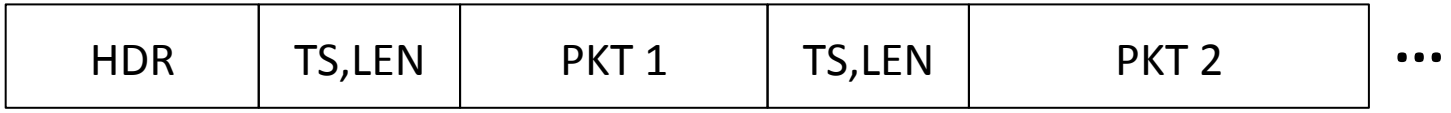
```
(000) ldh      [12]
(001) jeq      #0x800          jt 2  jf 15
(002) ldb      [23]
(003) jeq      #0x6            jt 4    jf 15
(004) ldh      [20]
(005) jset     #0x1fff         jt 15 jf 6
(006) ldxb     4*([14]&0xf)
(007) ldh      [x + 14]
(008) jeq      #0x64           jt 9   jf 11
(009) ldh      [x + 16]
(010) jeq      #0xc8           jt 14 jf 15
(011) jeq      #0xc8           jt 12 jf 15
(012) ldh      [x + 16]
(013) jeq      #0x64           jt 14jf 15
(014) ret      #65535
(015) ret      #0
```

libpcap

- We realized we wanted to build other packet capture applications beyond tcpdump
  - Pulled compiler system and filtering engine out of tcpdump
  - Created an “API” and reusable library
  - Released as “libpcap”
- If different apps were going to be built around this common library, we should have an interchangeable file format for packets traces

pcap File Format

- Elaboration of the "-w" flag to tcpdump
  - `tcpdump –w http.pcap port 80`
  - Bypass protocol decoding logic in tcpdump
  - Write packets straight to disk
  - Run as fast as possible to minimize drops



version#
timezone
snaplen
link type...

An Open Approach

- Released tcpdump, BPF, libpcap as open source
  - Ported to various operating systems
  - Berkeley Unix (BSD), SunOS, HP, SGI, DEC
- Eventually adopted in Linux and Mac OS X
- Published in USENIX 93, SIGCOMM 99
- My apologies... escaped before it was done
  - never quite finished lipcap API, then I read about it in Rich Stevens' TCP/IP Illustrated
  - Loris tells me I messed up the pcap file format ☹

Summary

- So, that's my story of libpcap
- I'm very honored and excited to return to the packet capture community after all these years...
  - I would have thought all the problems were solved but as we dig deeper every day, it's clear there is tons of opportunity for innovation...
  - I am looking forward to working with Loris, Gerald, and the community to continue to push the envelope
- It's hard to make things easy, but it's worth it in the end