BPF+

Exploiting Global Data-flow Optimization in a Packet Filter Architecture

Andrew Begel, Steven McCanne, Susan L. Graham

University of California, Berkeley

with acknowledgments to Van Jacobson

The Big Picture

With BPF+, we can express packet filters in a high-level language *and* compile them into efficient code.

The Packet Filter

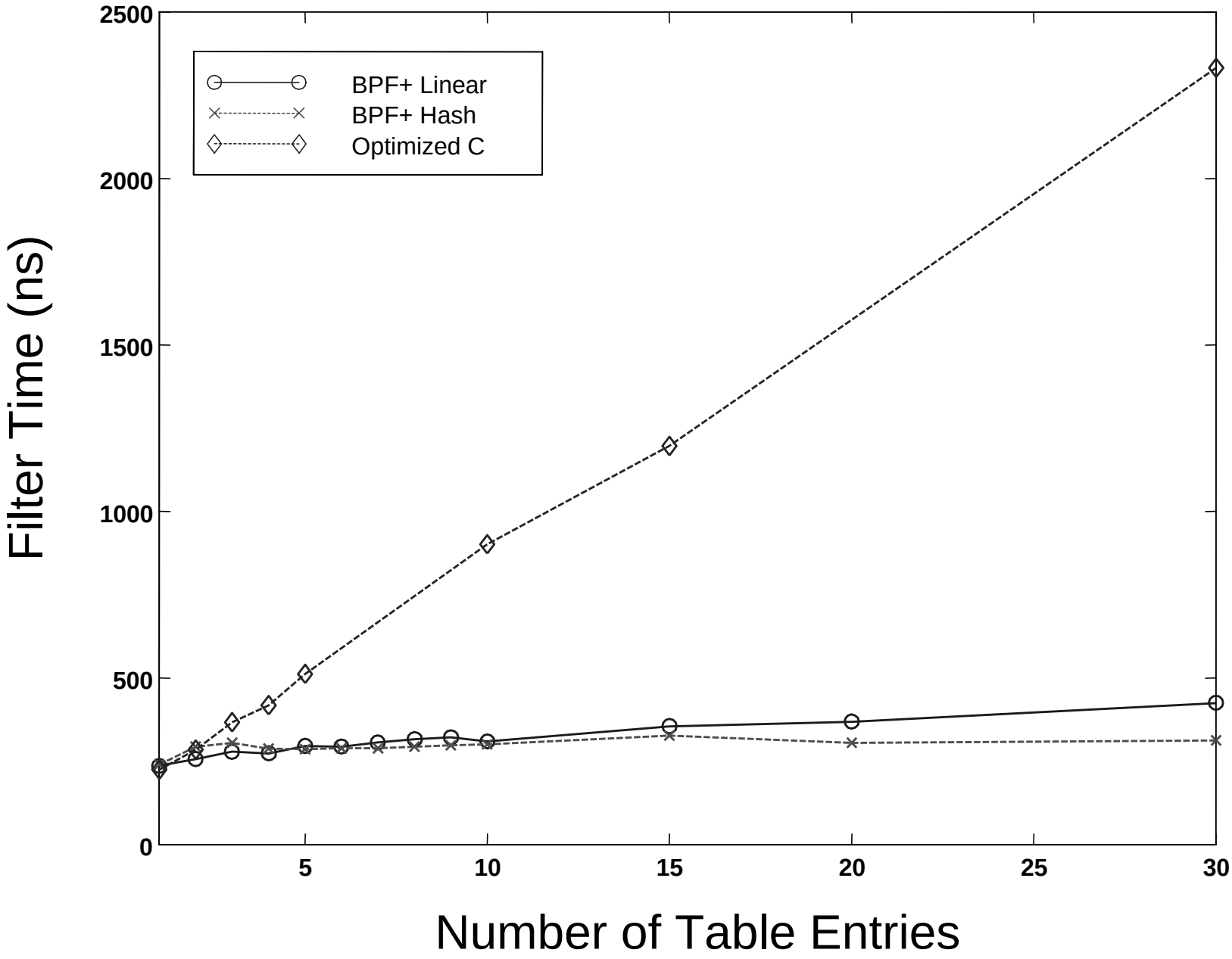
Definition: A tool for selecting packets from a packet stream using a programmable selection criterion.

Example: tcp and source net 128.32/16 and destination host web.mit.edu and port 80

Domain-Specific Optimization

- Tune traditional compiler optimizations for the application of packet filtering.
- Affords simpler analyses and *much* more effective optimization.
  - Packet filters are DAGs
  - Engenders linear-time algorithms
- Combine with Just-In-Time (JIT) assembly.

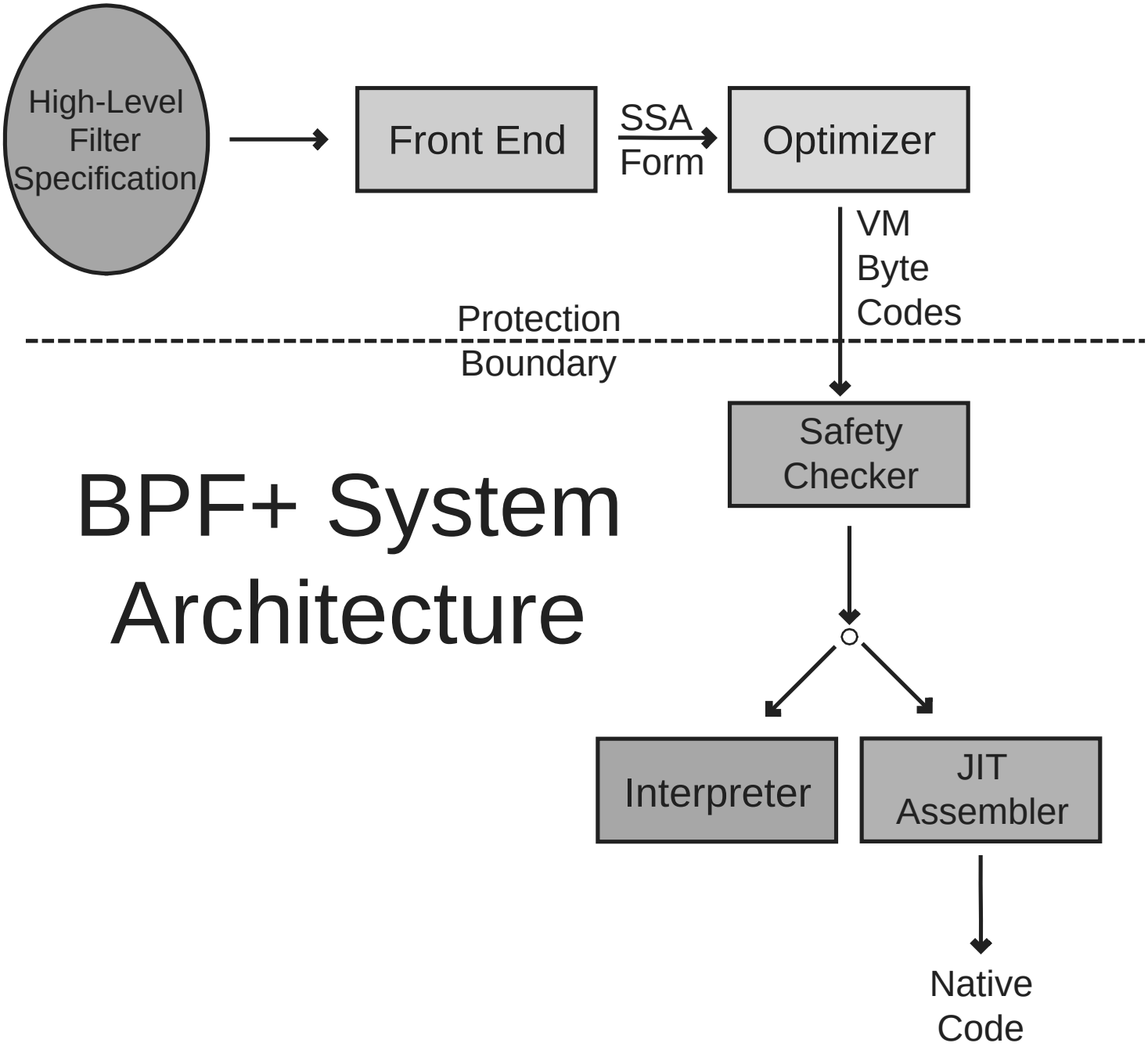
Comparison of C Optimizer to BPF+ Optimizer



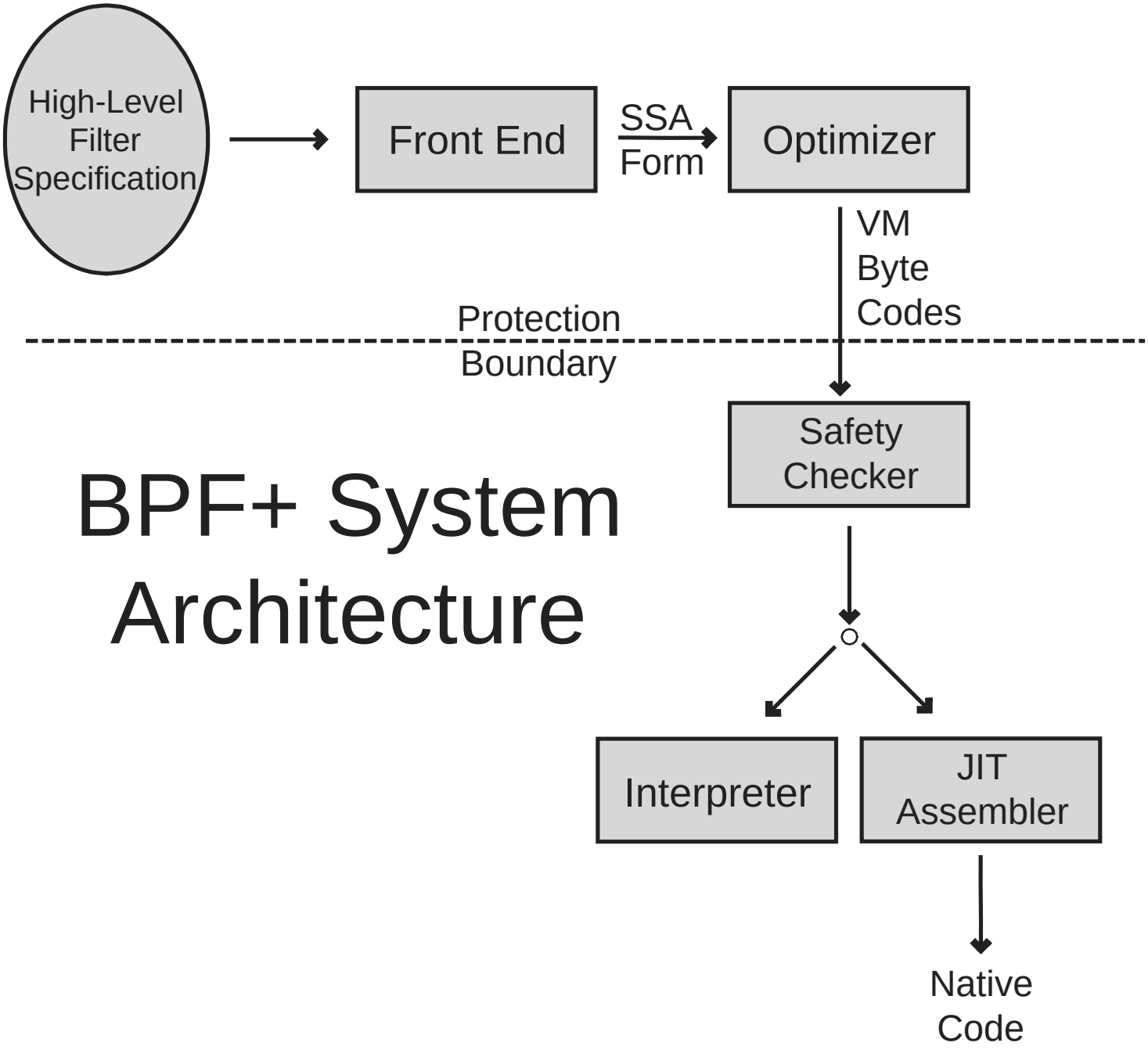
Related Work

- Virtual Machine Models:
  - CMU/Stanford Packet Filter: *MRA87*
  - BPF: Berkeley Packet Filter: *MJ93*
    - tcpdump and libpcap
  - MPF: *YBMEM94*
- Exploit Filter Structure:
  - PathFinder: *BGPP94*
  - DPF: *EK96*
  - Multi-dimensional Range Matching: *LS98*
  - Grid of Tries: *SVSW98*
- High-Level Approach:
  - LR Parsing: *JC96*

BPF+ System Architecture



BPF+ System Architecture



Compiler Optimizations

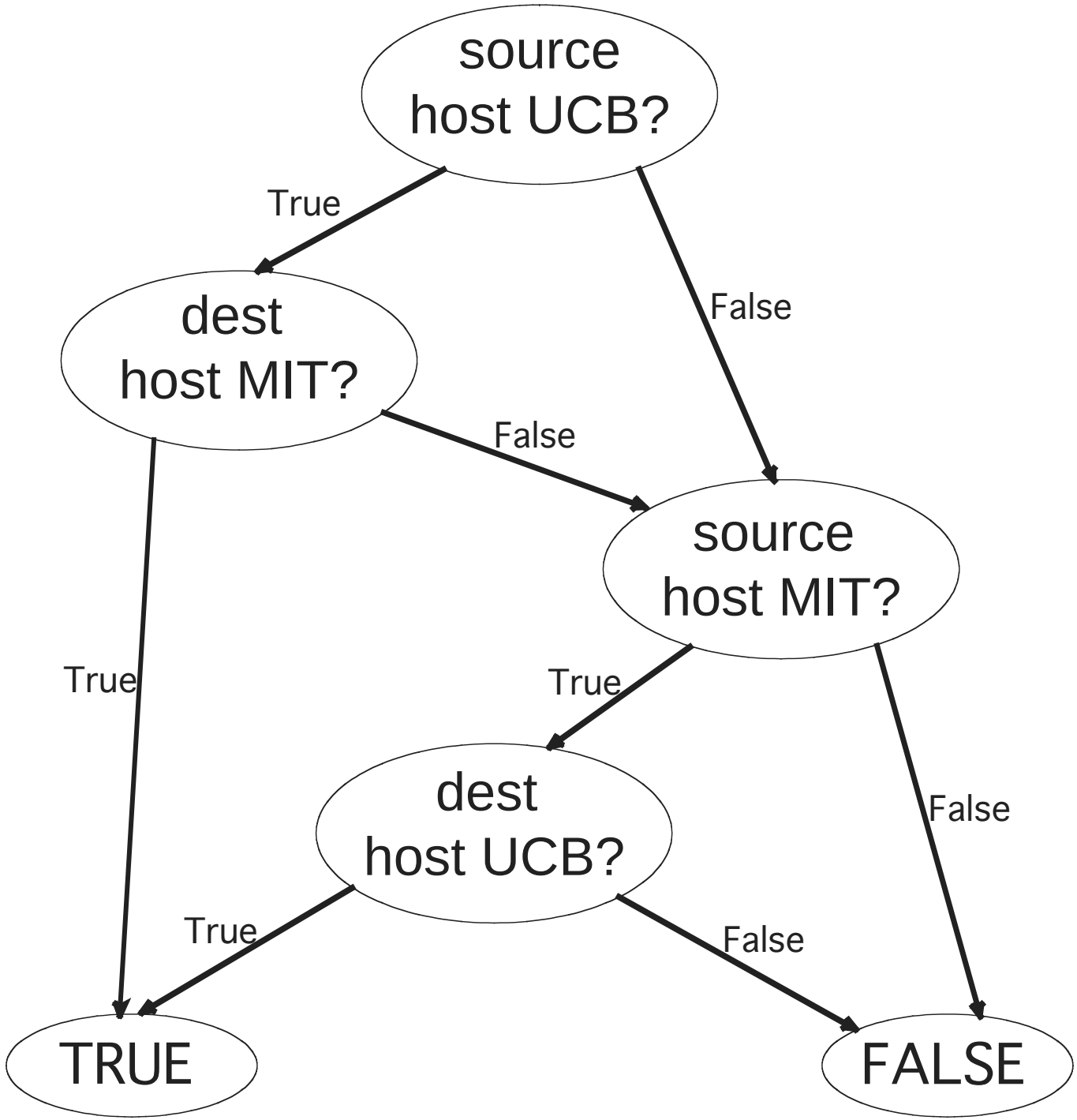
- Leverage modern compiler technology
- Powerful intermediate form
  - Static Single Assignment (SSA)
- Three key optimizations
  1. Redundant Predicate Elimination
  2. Partial Redundancy Elimination
  3. Lookup Table Encapsulation

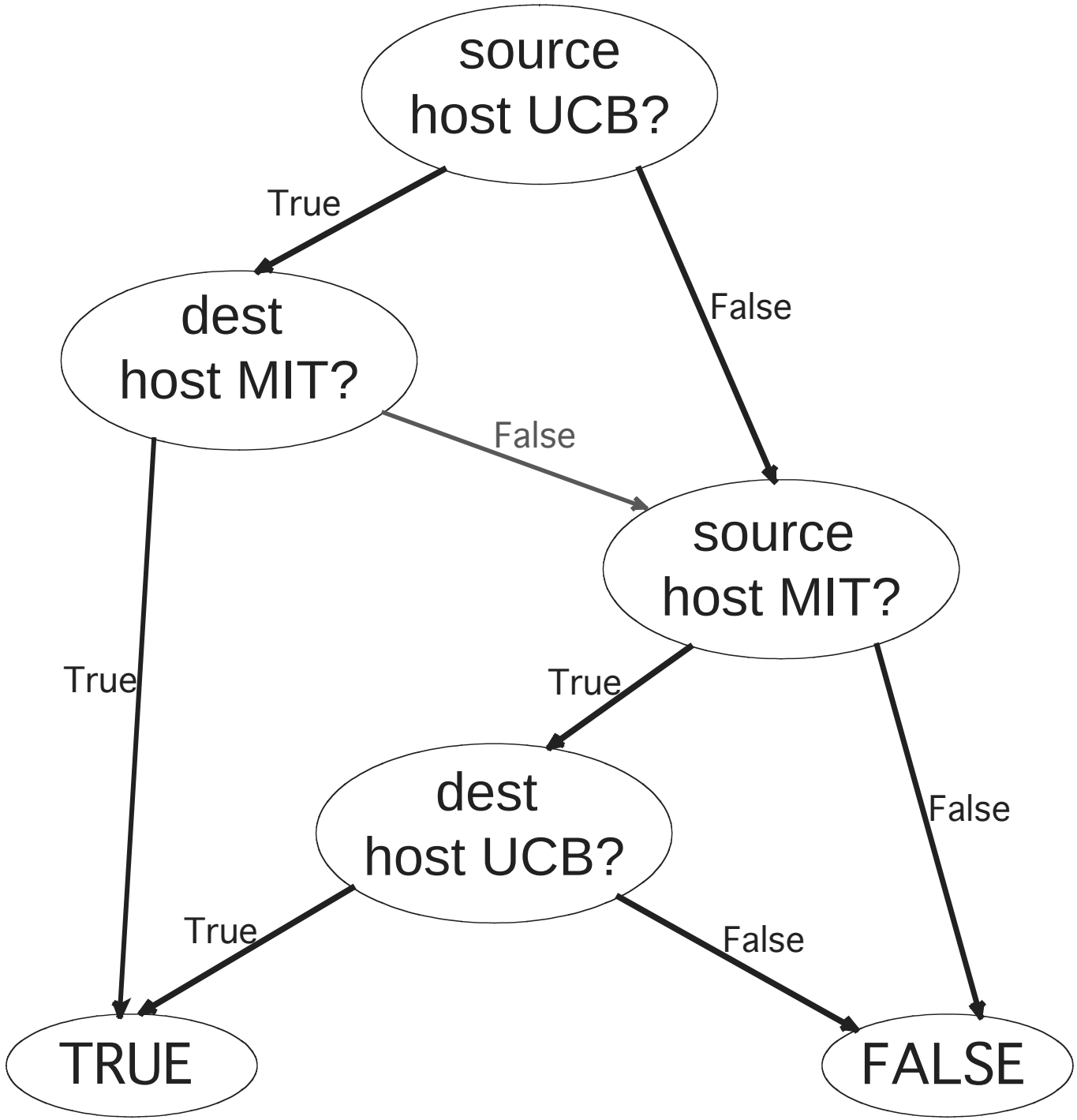
Redundant Predicate Elimination

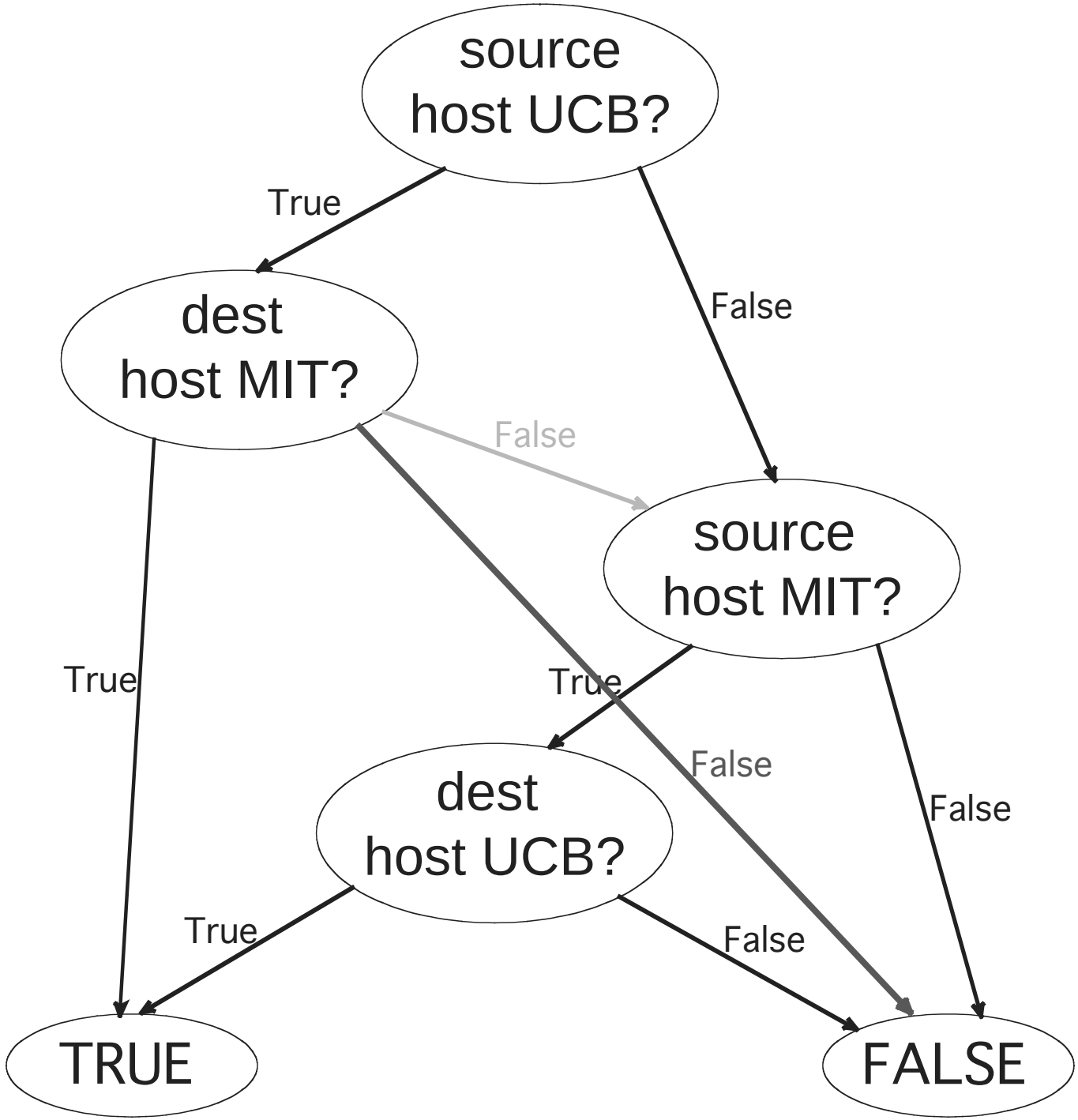
- Packet header parsing leads to many redundant predicates.
- Unlike traditional optimizations, detect redundant *edges* rather than redundant *computation.*
- Packet filter control flow graphs are edge-rich.

All packets between UCB and MIT

(src host UCB and dest host MIT)
or
(src host MIT and dest host UCB)


source
host UCB?
True
False
dest
host MIT?
False
source
host MIT?
True
True
dest
host UCB?
False
True
False
TRUE
FALSE


source
host UCB?
True
False
dest
host MIT?
False
source
host MIT?
True
True
dest
host UCB?
False
True
False
TRUE
FALSE


source
host UCB?
True
False
dest
host MIT?
False
source
host MIT?
True
True
False
False
dest
host UCB?
True
False
TRUE
FALSE

Partial Redundancy Elimination

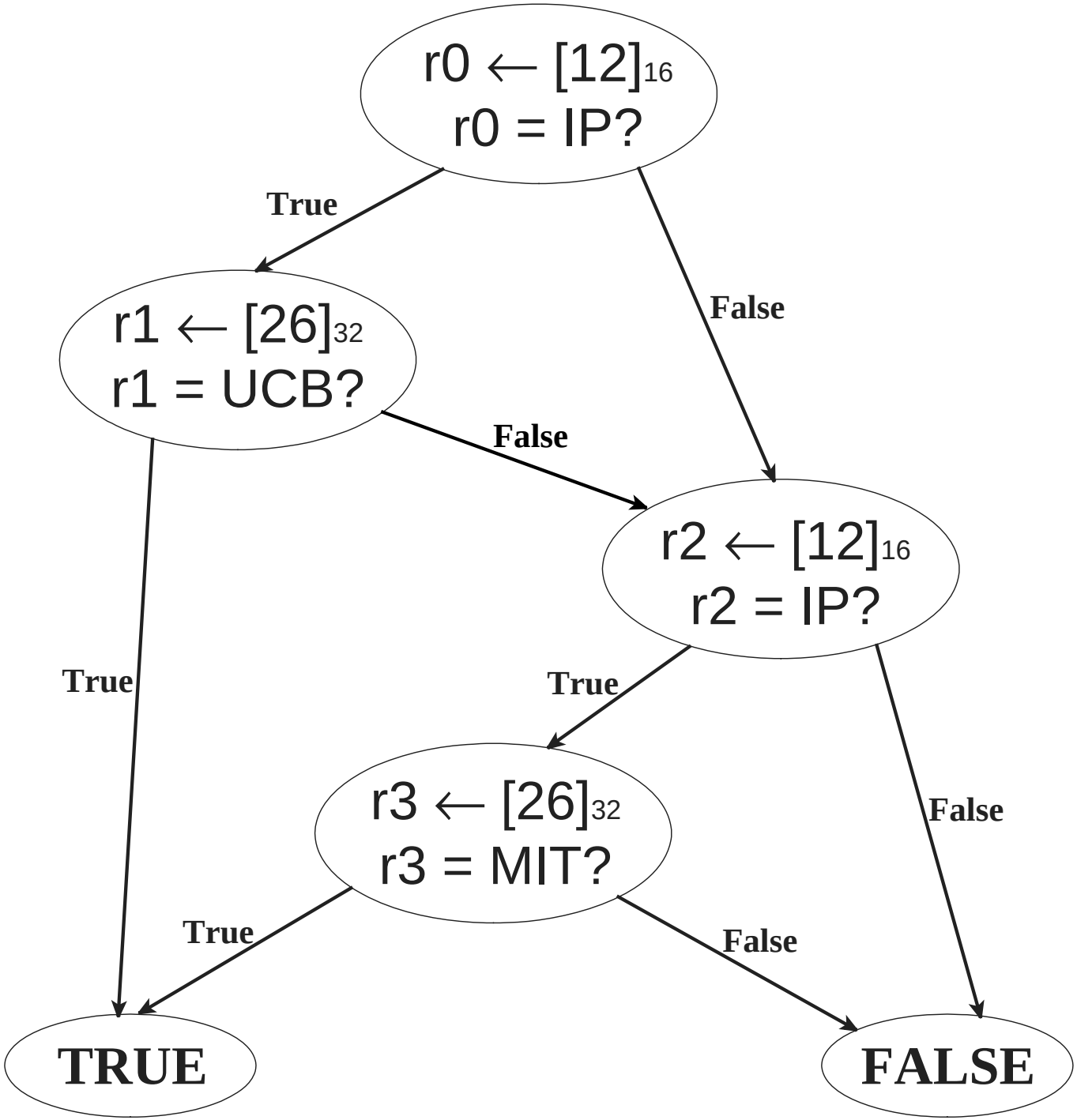
- Packet header parsing also leads to much redundant computation.

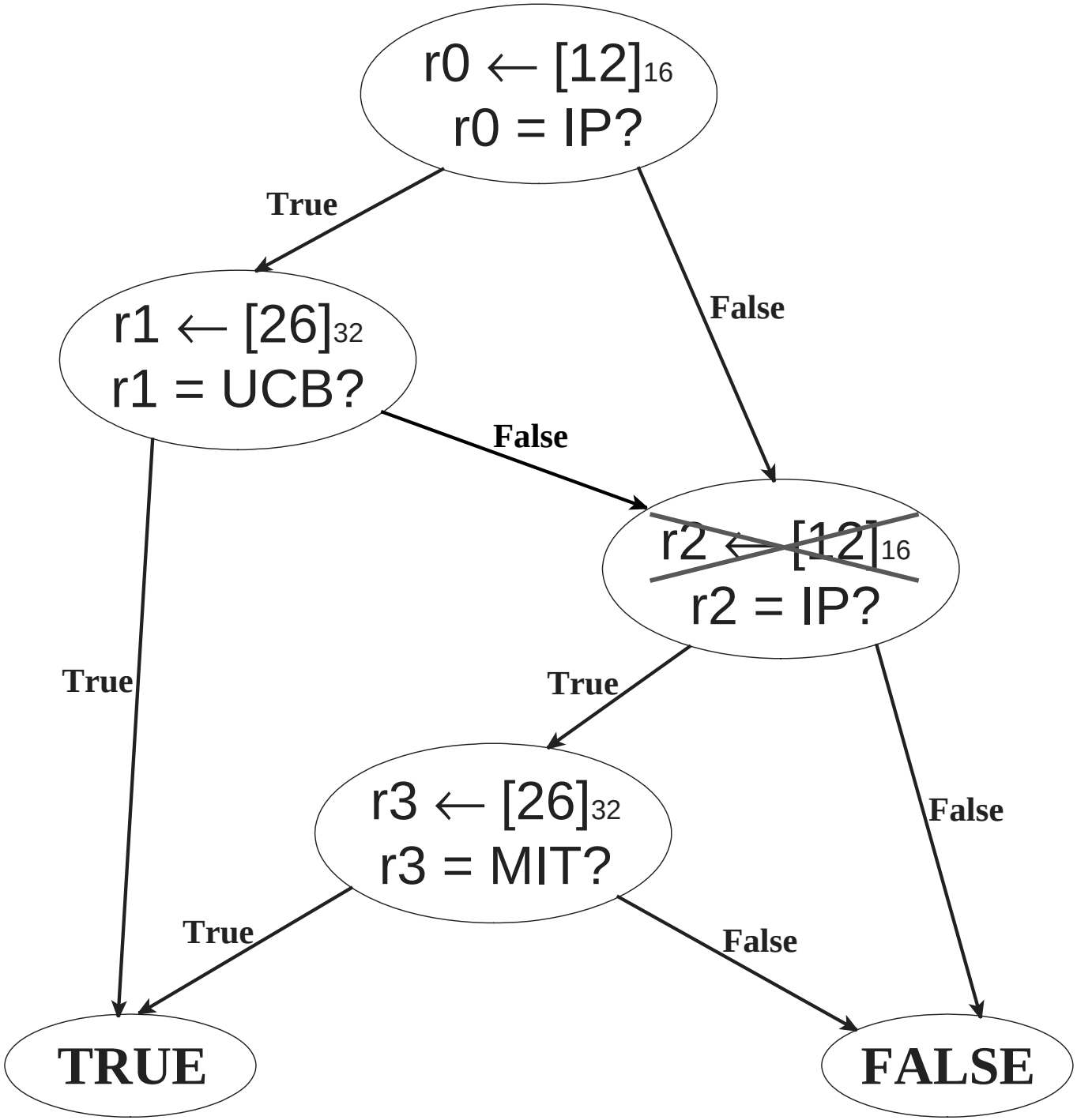
All packets that come from either UCB or MIT

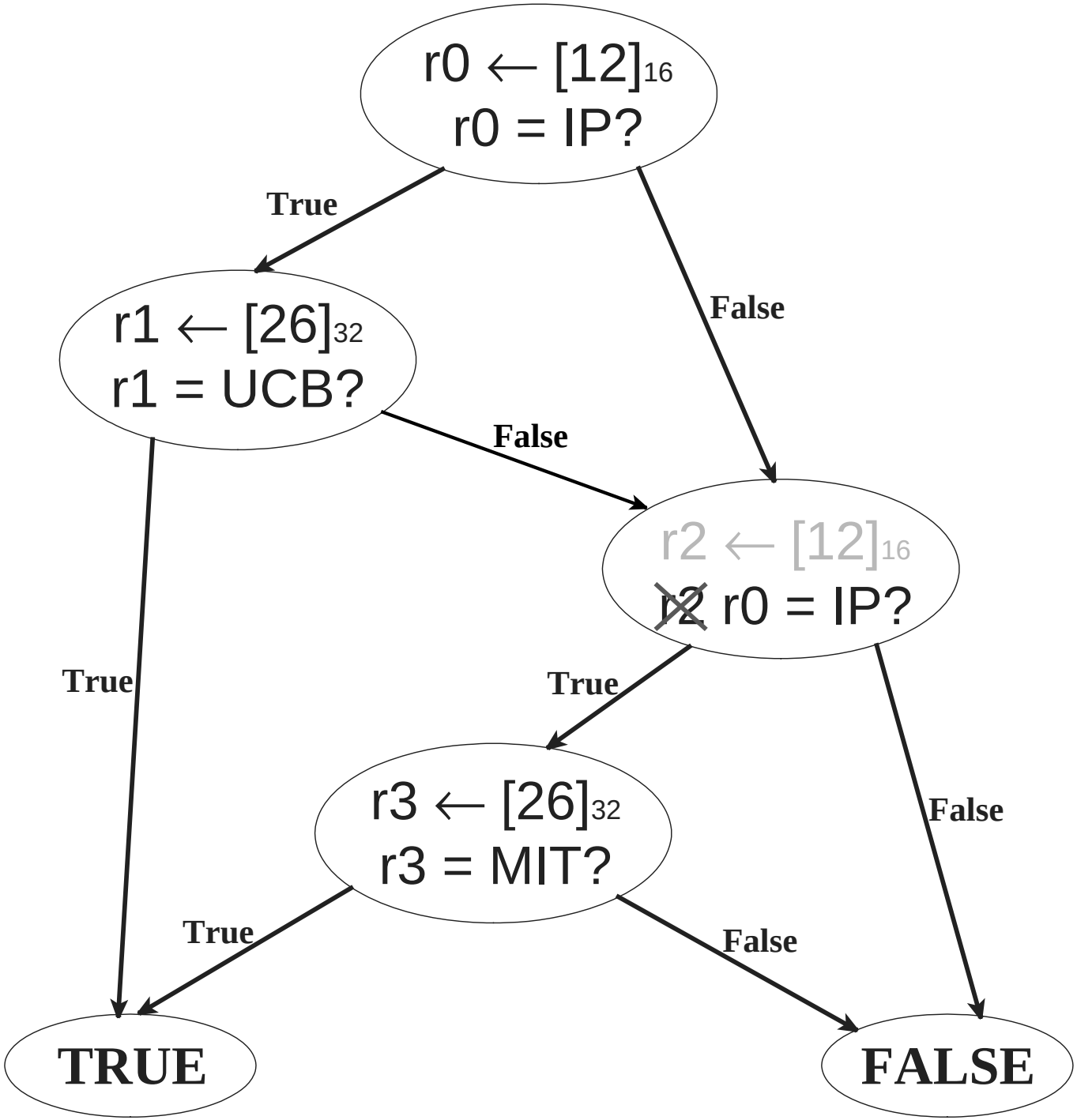
src host UCB

or

src host MIT


r0 ← [12]16
r0 = IP?
True
False
r1 ← [26]32
r1 = UCB?
False
True
r2 ← [12]16
r2 = IP?
True
True
False
r3 ← [26]32
r3 = MIT?
True
False
TRUE
FALSE

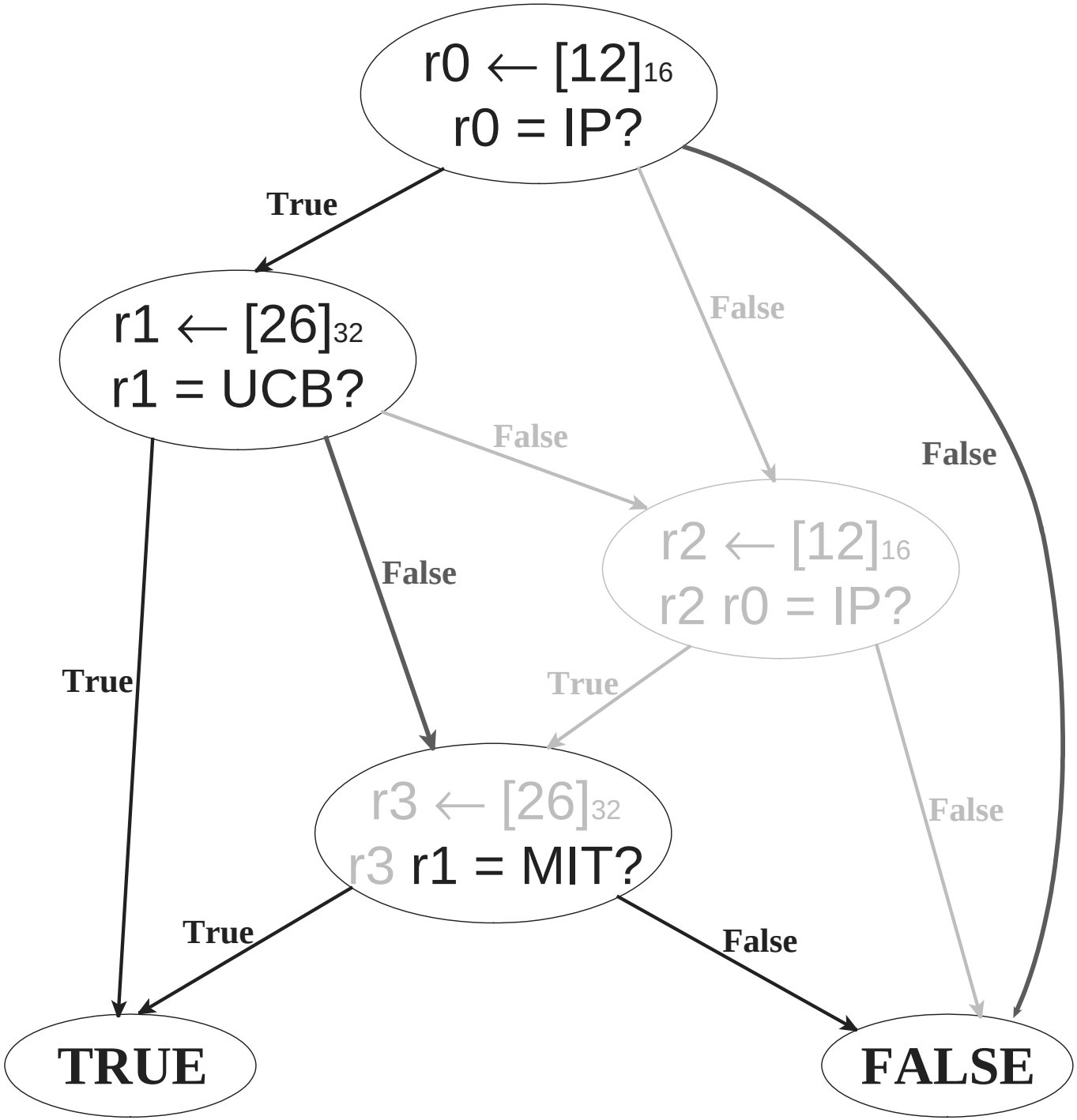

r0 ← [12]16
r0 = IP?
True
False
r1 ← [26]32
r1 = UCB?
False
r2 ← [12]16
r2 = IP?
True
True
False
r3 ← [26]32
r3 = MIT?
True
False
TRUE
FALSE


r0 ← [12]16
r0 = IP?
True
False
r1 ← [26]32
r1 = UCB?
False
r2 ← [12]16
r2 r0 = IP?
True
True
r3 ← [26]32
r3 = MIT?
False
True
False
TRUE
FALSE

Putting Them Both Together

- *Combining* partial redundancy elimination with redundant predicate elimination is much more effective than using either alone.

(6 steps later...)


r0 ← [12]16
r0 = IP?
True
False
False
r1 ← [26]32
r1 = UCB?
False
False
True
r2 ← [12]16
r2 r0 = IP?
True
True
r3 ← [26]32
r3 r1 = MIT?
False
True
False
TRUE
FALSE

Lookup Table Encapsulation

- After early optimizations, many predicates reduced to simple field comparisons.
- Use analysis to discover opportunities for moving into lookup tables.
- Implementation may use linear search, binary search, hash lookup or any combination of the three.

All packets that come from UCB, MIT or CMU

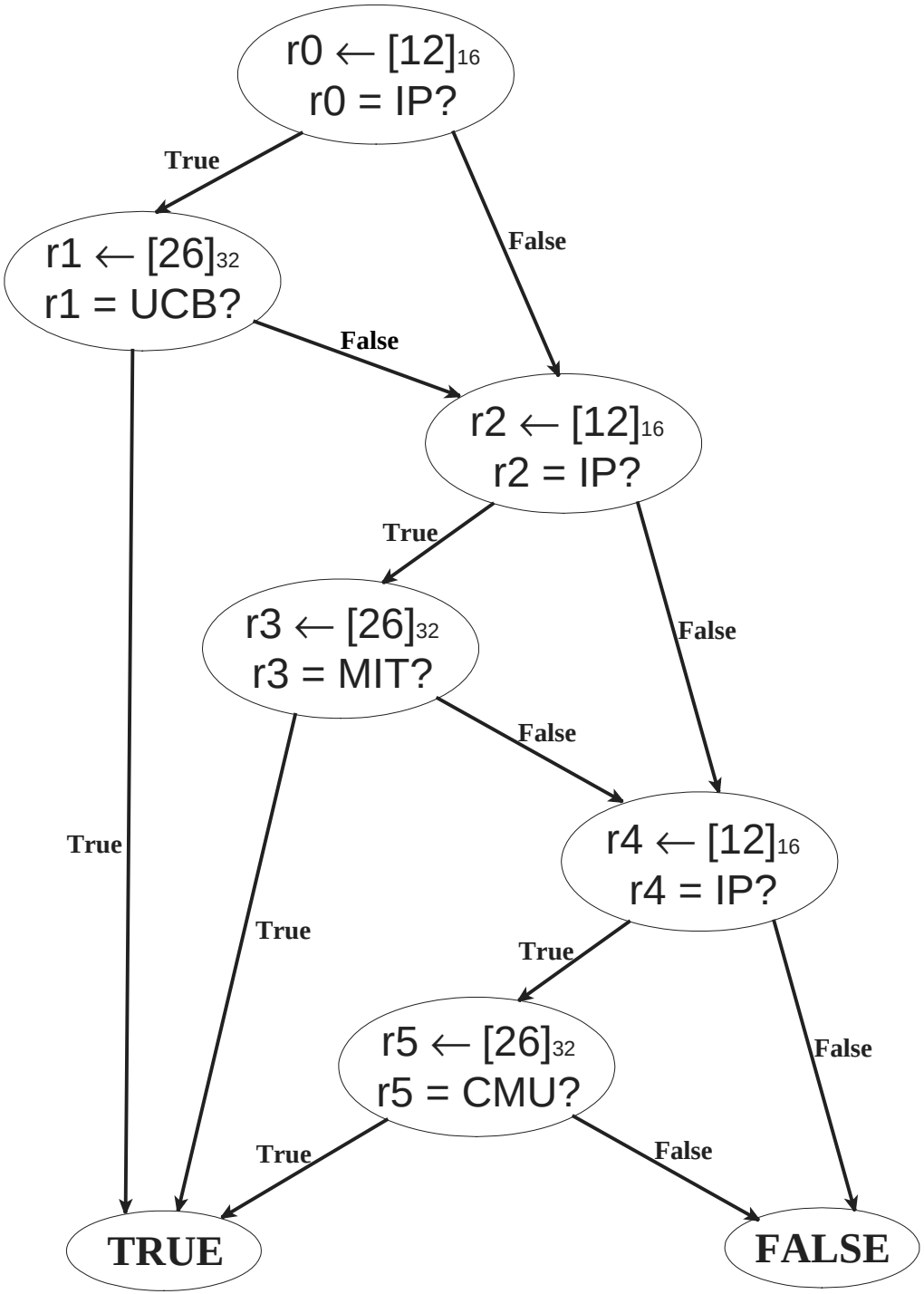
src host UCB

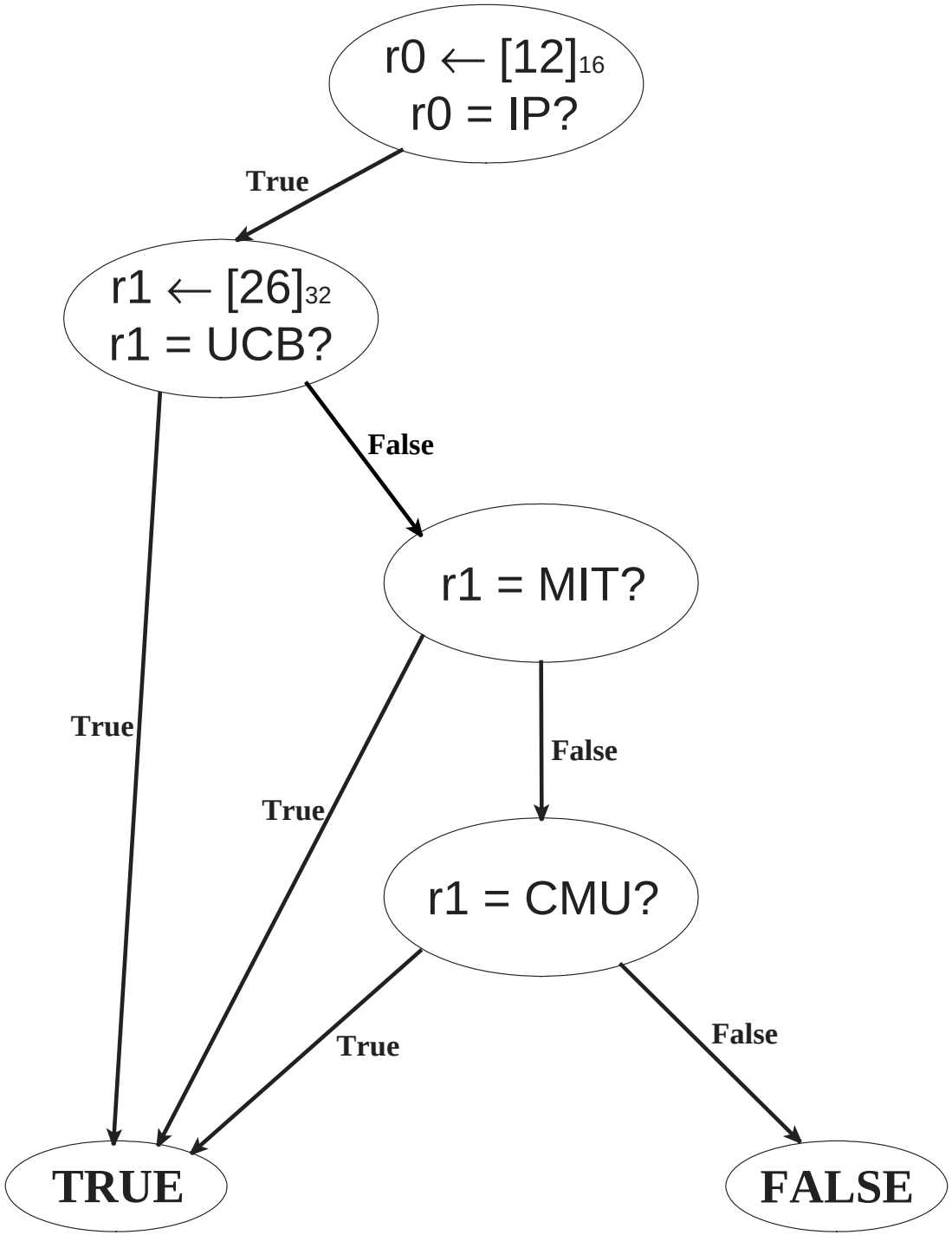
or

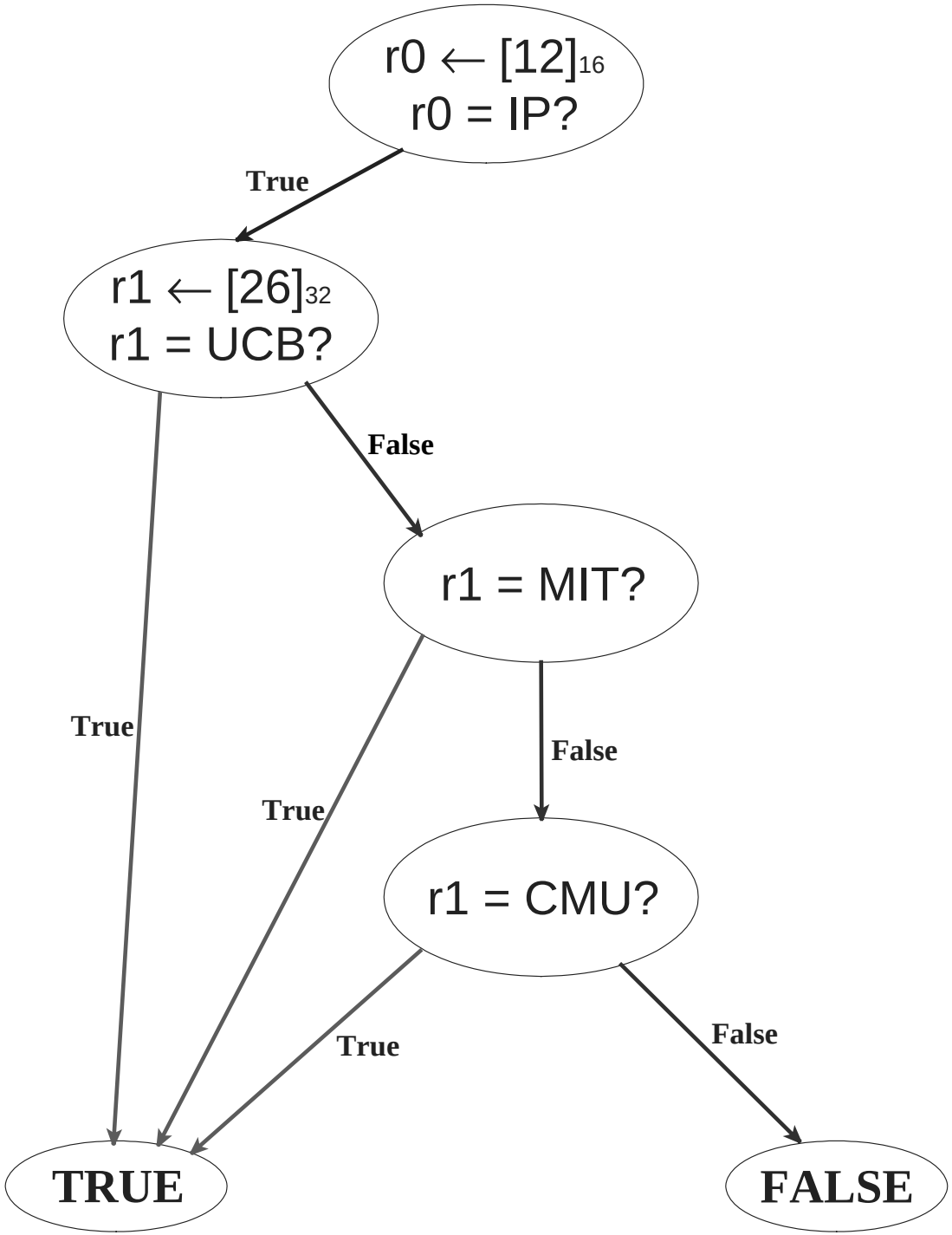
src host MIT

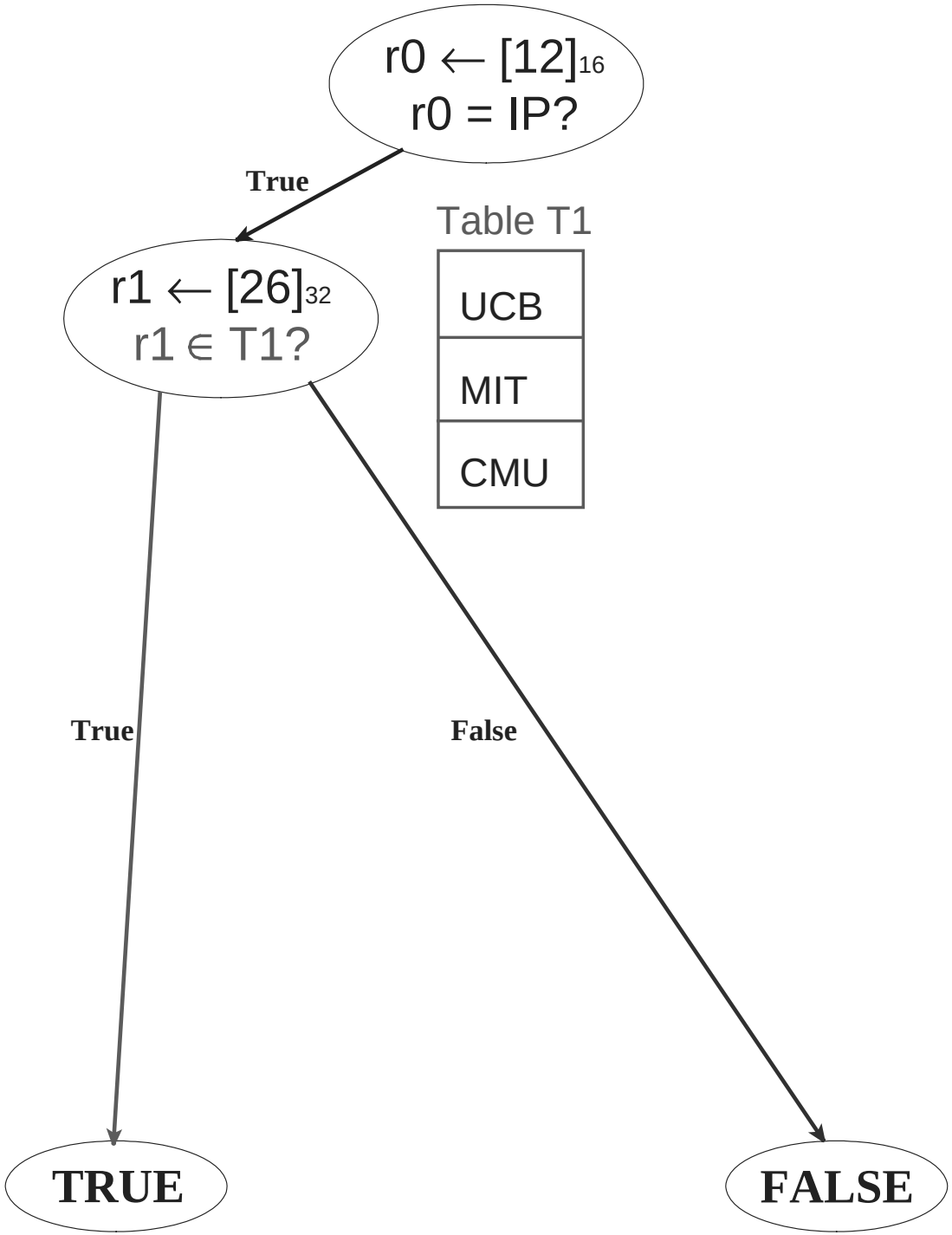
or

src host CMU

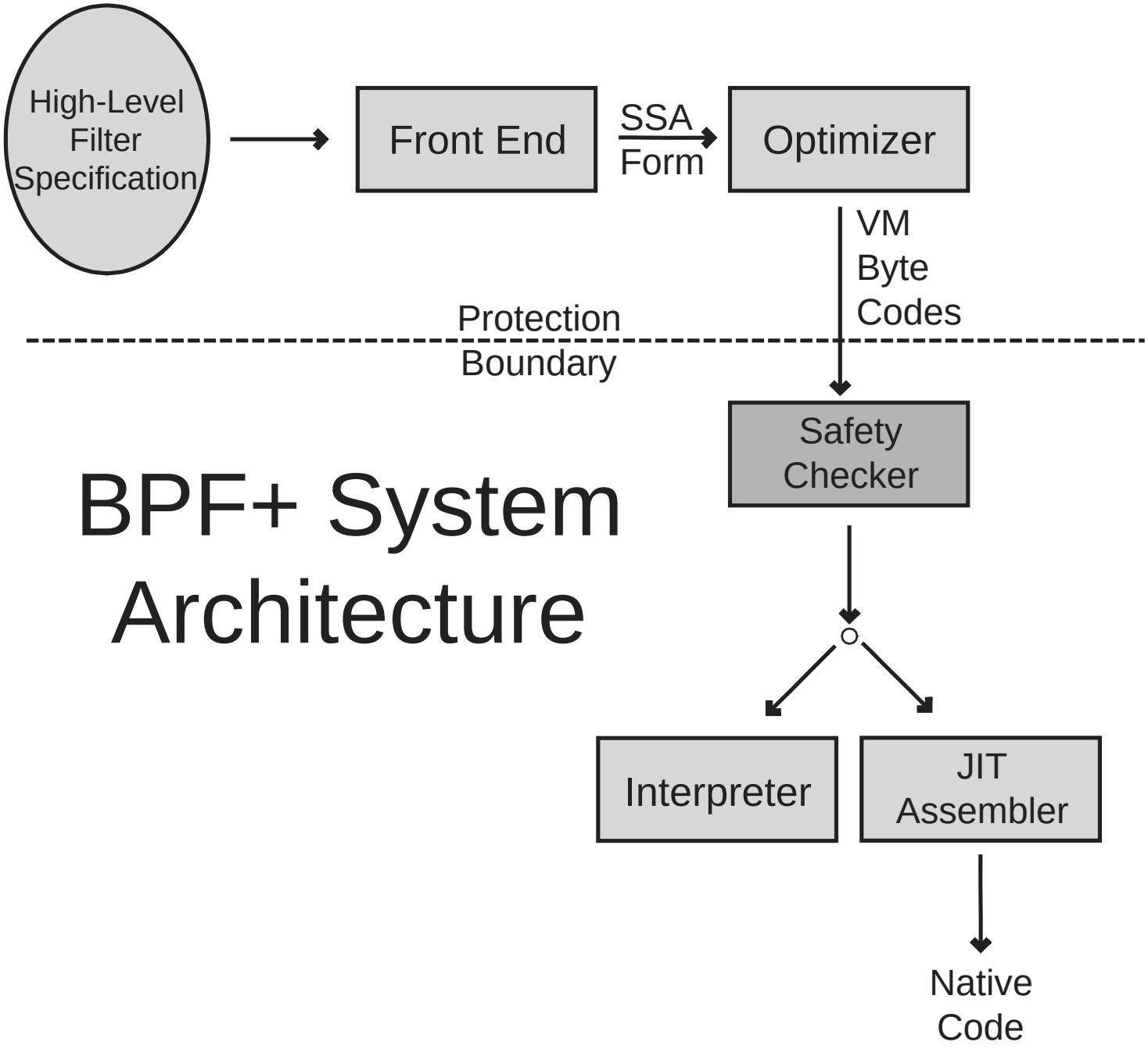

r0 ← [12]16
r0 = IP?
True
False
r1 ← [26]32
r1 = UCB?
False
r2 ← [12]16
r2 = IP?
True
False
r3 ← [26]32
r3 = MIT?
False
True
True
r4 ← [12]16
r4 = IP?
True
False
r5 ← [26]32
r5 = CMU?
True
False
TRUE
FALSE


r0 ← [12]16
r0 = IP?
True
r1 ← [26]32
r1 = UCB?
False
r1 = MIT?
True
False
True
r1 = CMU?
True
False
TRUE
FALSE


r0 ← [12]16
r0 = IP?
True
r1 ← [26]32
r1 = UCB?
False
r1 = MIT?
True
False
True
r1 = CMU?
True
False
TRUE
FALSE


r0 ← [12]16
r0 = IP?
True
Table T1
r1 ← [26]32
r1 ∈ T1?
UCB
MIT
CMU
True
False
TRUE
FALSE

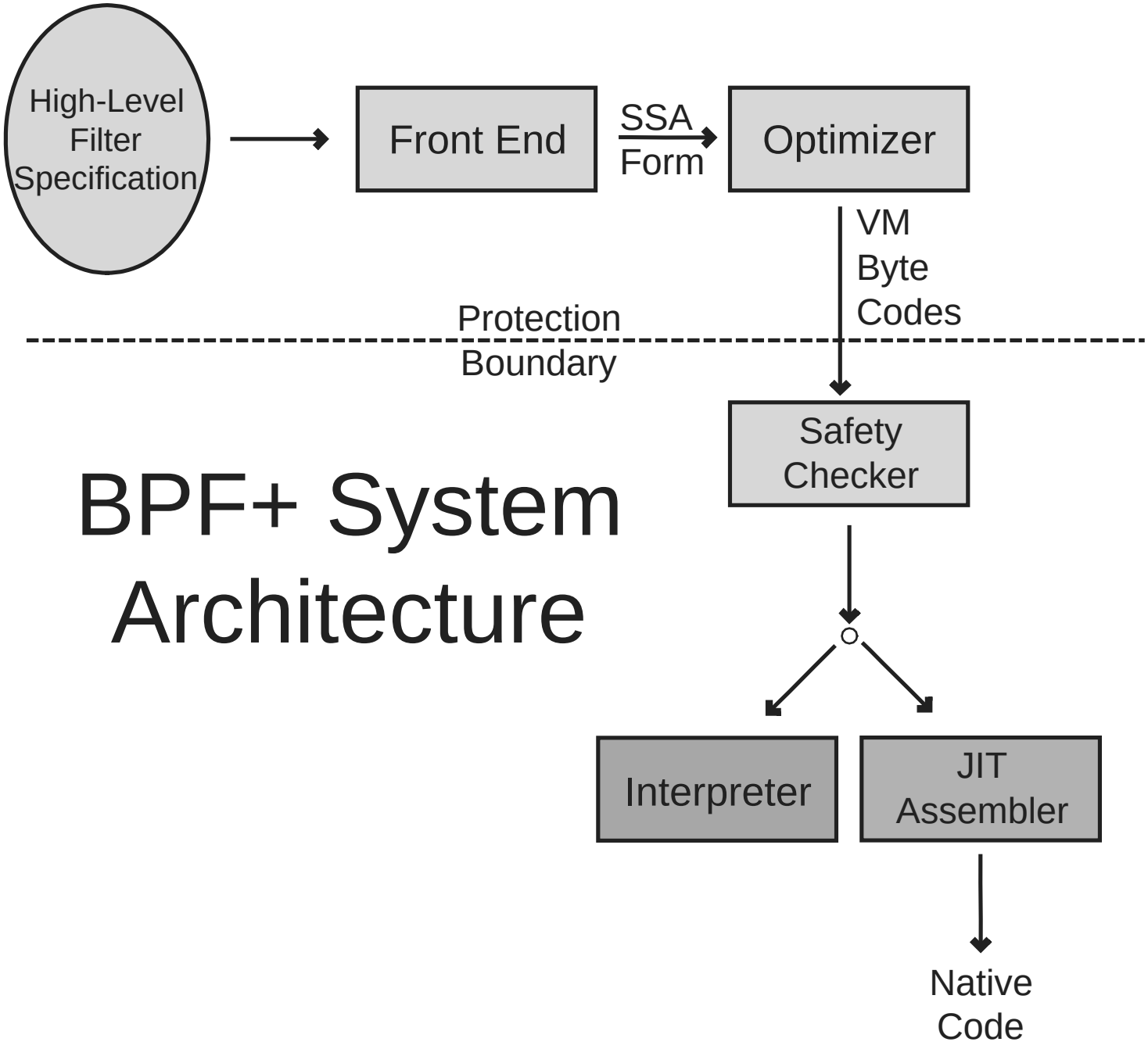
BPF+ System Architecture



Filter Safety Must Be Verified

- All bytecodes are valid.
- Jump targets are valid.
- No loops.
- All paths terminate with a **return** instruction.
- No out-of-bounds reads or writes.

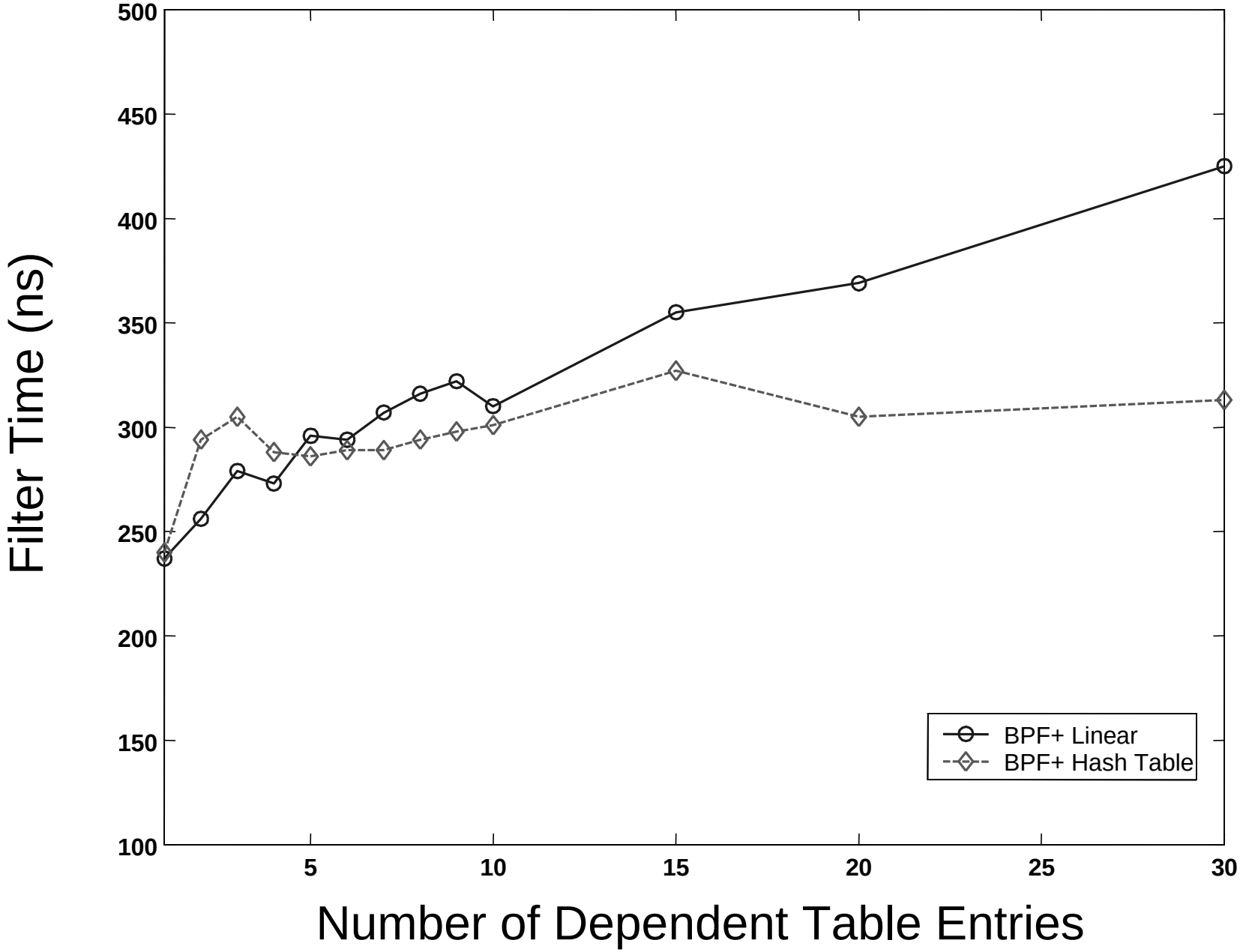
BPF+ System Architecture



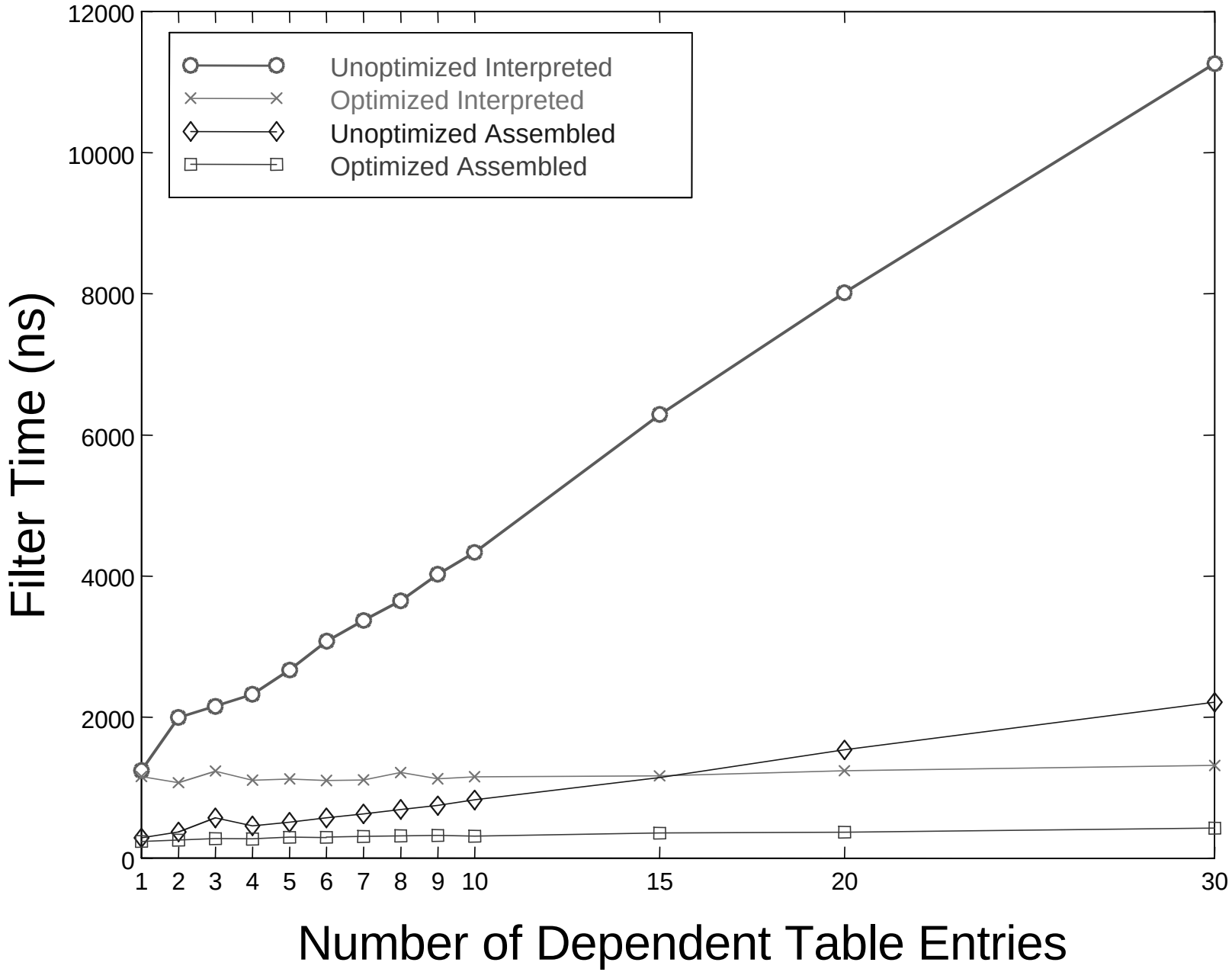
Performance Tests

- Two types of predicate expressions
  - Dependent (src host (UCB or MIT or CMU)
  - Independent (i.e. TCP, Port 80, dest host UCB)
- Run on Ultra 10 300 MHz UltraSPARC IIi
- Four ways to run a filter
  - Unoptimized, Interpreted
  - Optimized, Interpreted
  - Unoptimized, JIT Assembled
  - Optimized, JIT Assembled

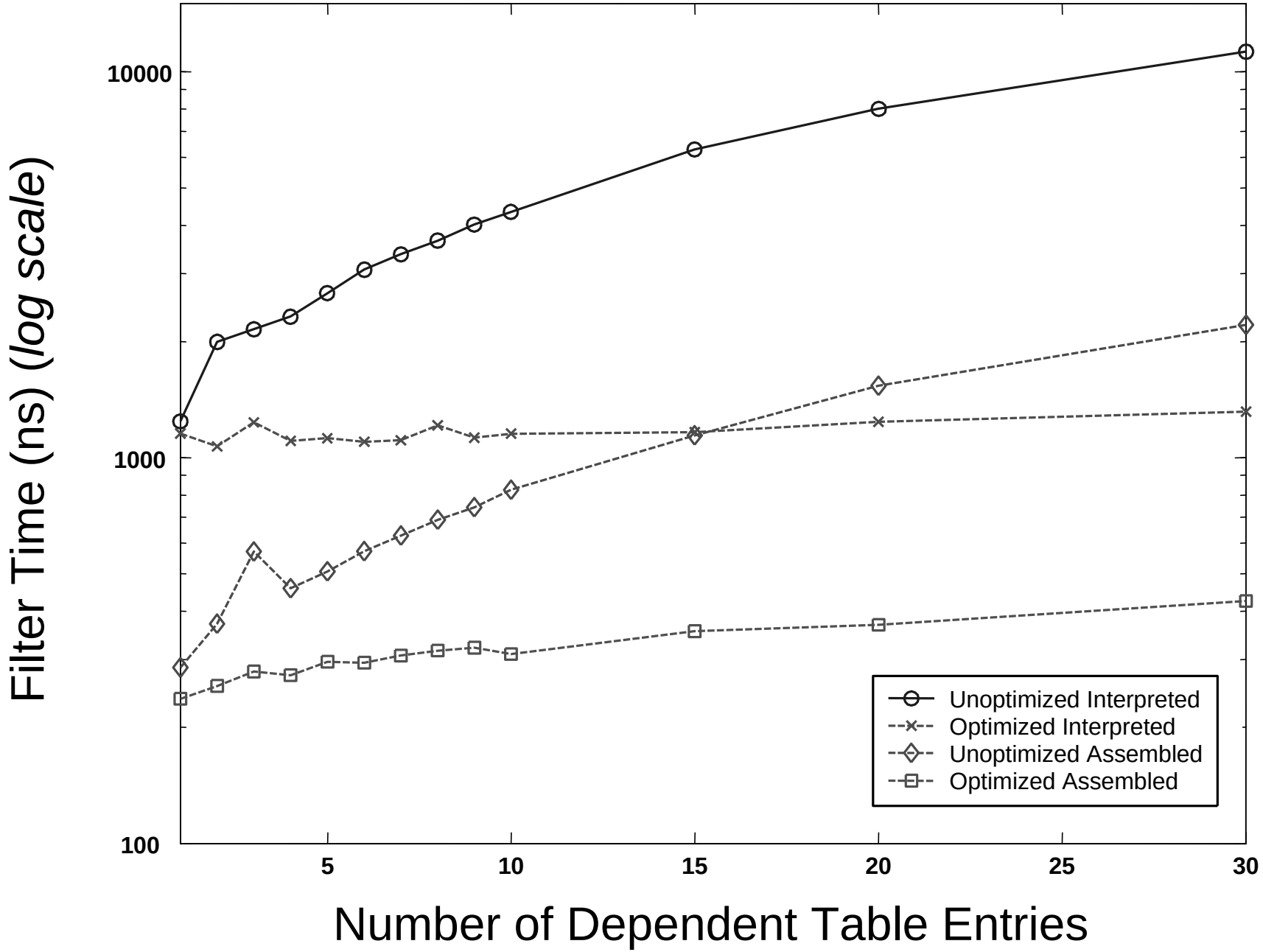
Comparison of Linear Search to Hash Lookup



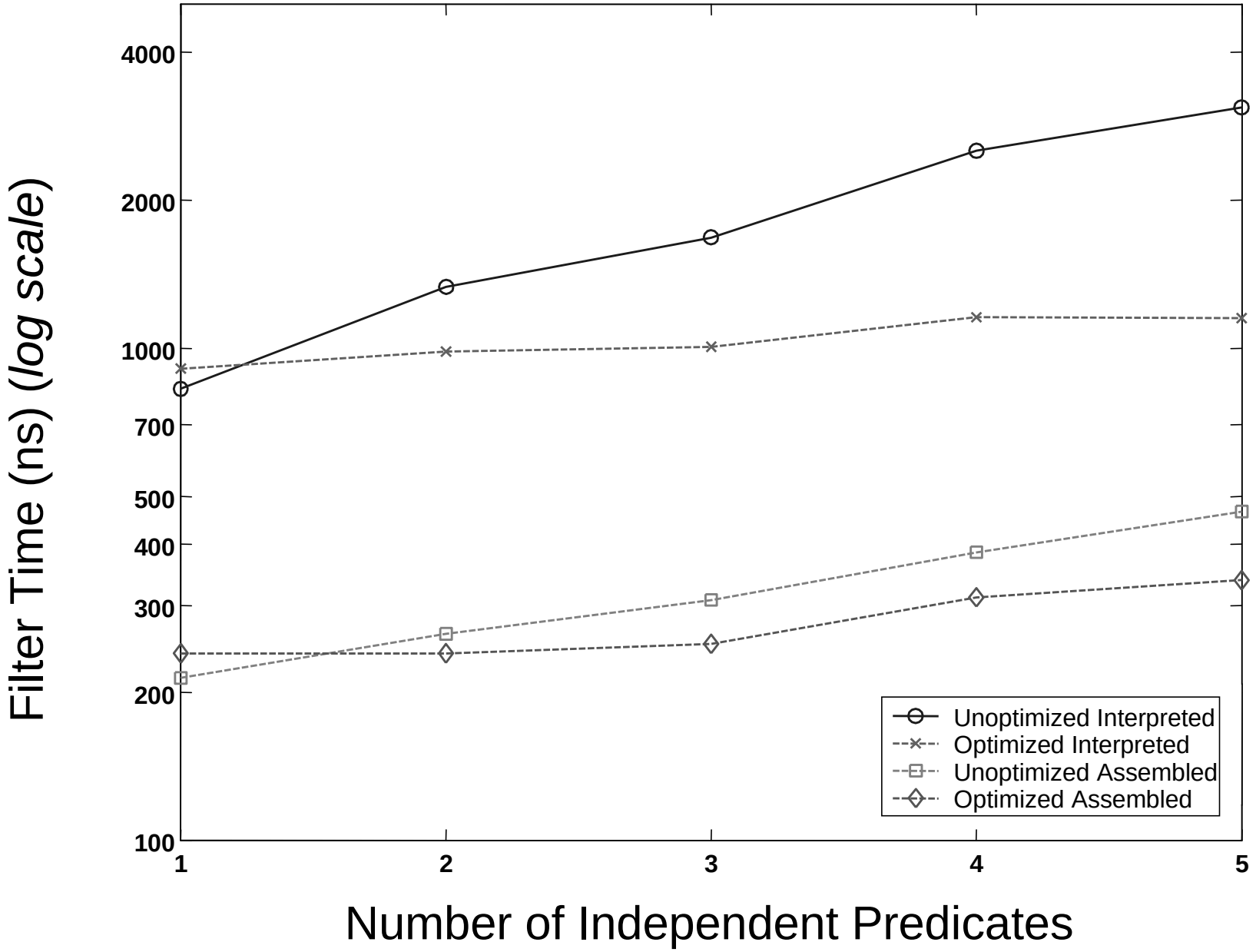
Effects of Optimization and JIT Assembly



Effects of Optimization and JIT Assembly (log scale)



Effects of Optimization and JIT Assembly on Independent Predicates (log scale)



Future Work

- More efficient table lookup representations (LS98, SVSW98)
- Better support for packet classification
- Loops
  - Proof-Carrying Code, Necula 96
- Intrusion detection
  - Online Updates

Conclusions

- Packet filters can be specified at a high-level *and* be efficiently executed.

- Key idea: Tune familiar global data-flow compiler analyses and optimizations for packet filtering.