Deflating the Big Bang: Fast and Scalable Deep Packet Inspection with Extended Finite Automata

THE UNIVERSITY of WISCONSIN MADISON

Randy Smith
Cristian Estan
Somesh Jha
Shijin Kong

Deep Packet Inspection

- Packet content increasingly used to classify net traffic
  - Used for intrusion detection, application identification, quality of service
  - Limited resources: classic time v. space tradeoff
- In this work
  - Techniques that reduce both memory and execution time by an order of magnitude or more

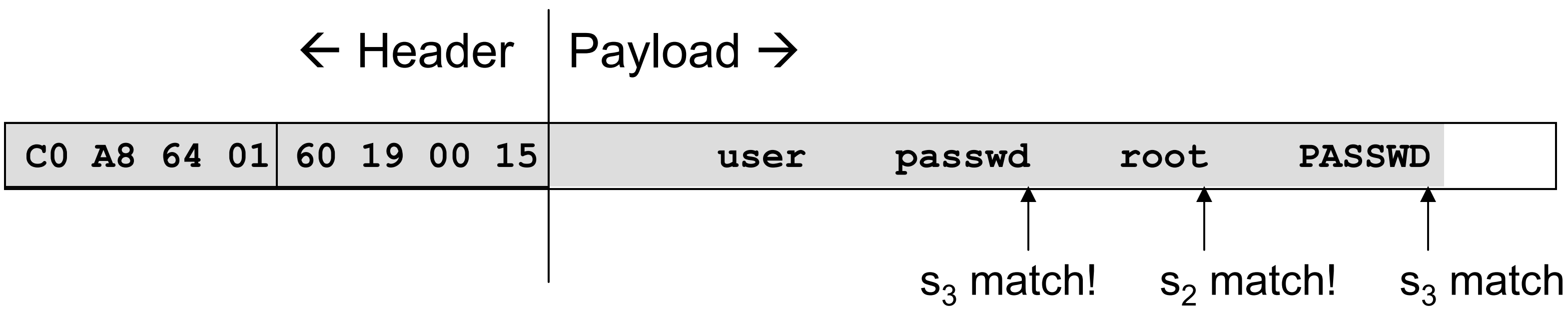
Signature Matching

- Problem: find all signature occurrences that match the payload up to the currently scanned byte
- Example:

Signatures = { *sig₁*: /(.*)shadow/ ,

sig$_2$: /(.*)user(.*)root/ ,

sig$_3$: /(.*)[Pp][Aa][Ss][Ss][Ww][Dd]/ }



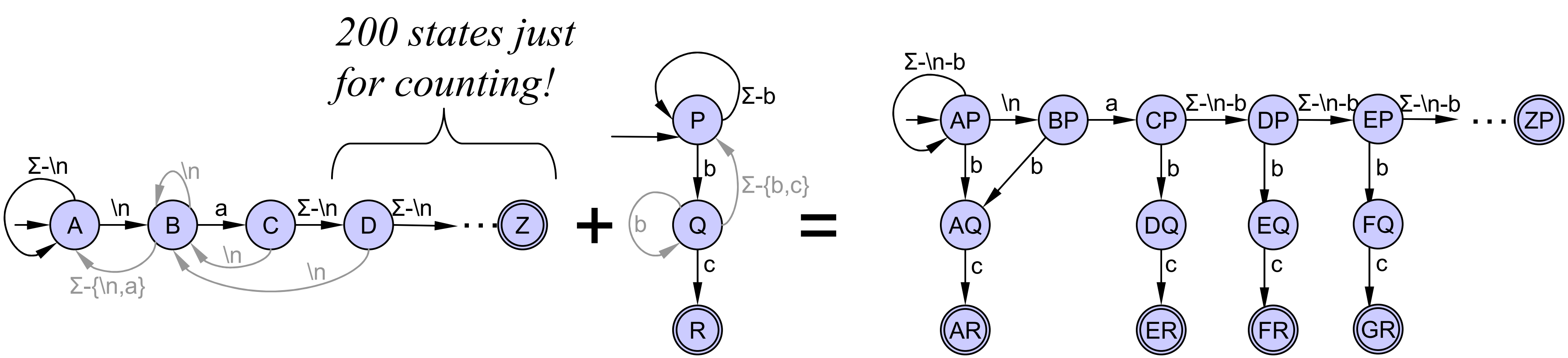
- Ideal: match signatures simultaneously in a single pass

Signatures

- Regular expressions used to *express* signatures
  - Capture vulnerabilities rather than specific exploits
    - Buffer overflow: `/^RETR\s[^\n]{100}/`
    - Format string: `/^SITE\s+EXEC\s[^\n]*%[^\n]*%/`
- Finite automata used to *match* signatures
  - Simple, well-understood model of computation
  - Combine using standard cross-product operation
- *But there's a problem…*

State-Space Explosion

- State-space "explodes" under combination
  - Less than 100 signatures requires more than 3 GB!
- Why? combined states = tuples of source states
  - Distinct state for each reachable combination



/.*\na[^\n]{200}/ /.*bc/

XFAs: Extended Finite Automata

- Introduced in *IEEE Symposium on Security and Privacy (Oakland) 2008*

- Problem with DFAs: No distinction between DFA state and computation state

- Idea: extend DFAs with variables that more efficiently track computation state
  - Variables reside in a small "scratch" memory
  - Small programs update variables during matching

Main Contributions

- Formal characterization of state-space explosion
- XFA model
  - XFA algorithms easily adapted from DFA algorithms
  - Framework for systematic optimization

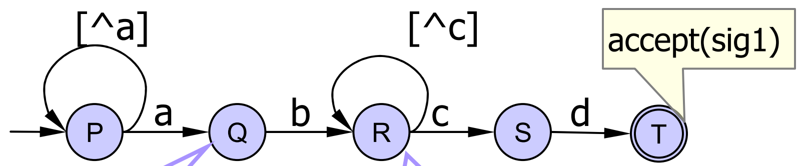
Outline

- State space explosion, formally
- Optimizations
- Experimental Evaluation

Ambiguity

- What are the input sequences that lead to a state?

/.*ab.*cd/



All input sequences leading to Q have the same suffix:

a, aha, aloha, hiya, aaa, aba,…

Input sequences leading to R have different suffixes:

ab, abe, abs, ab[^c]+,…

Unambiguous

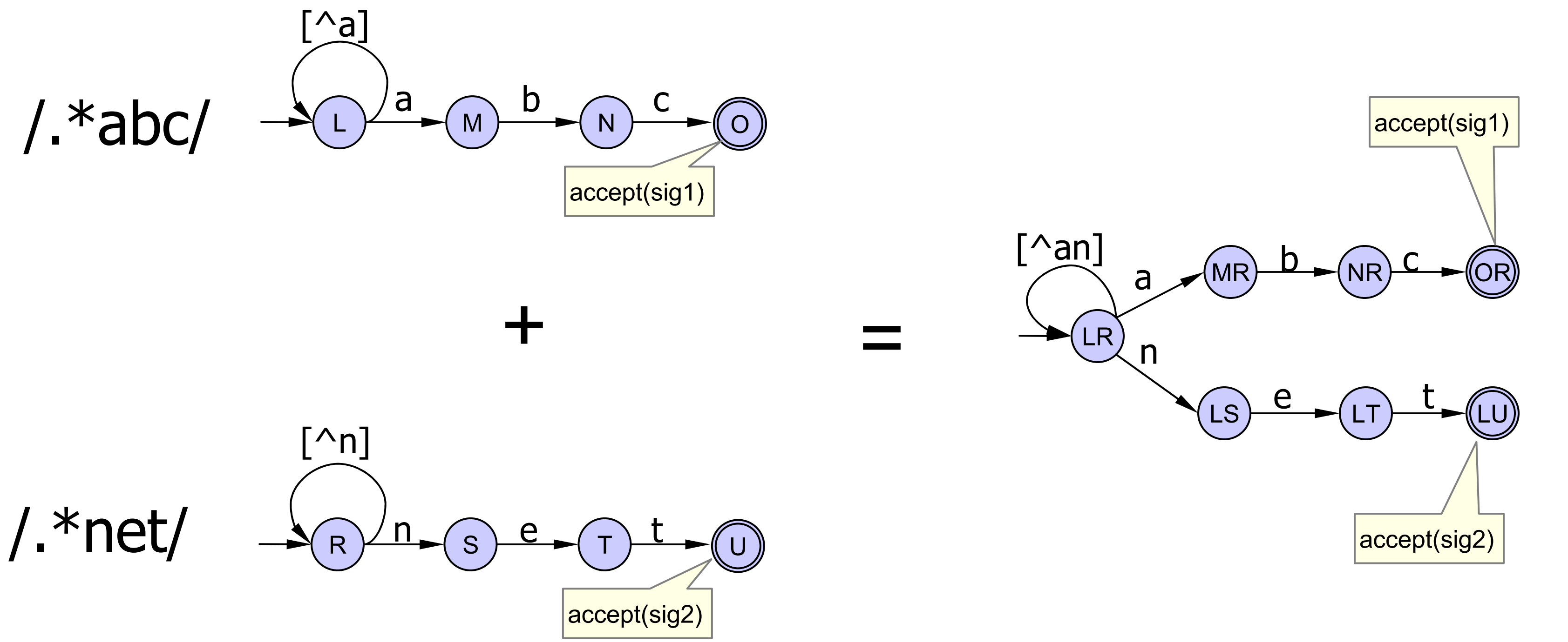
Ambiguous

- A DFA is *unambiguous* iff all its states are unambiguous

Unambiguous Automata

- Property 1:
  D_1 and D_2 unambiguous ➔ $D_1 + D_2$ unambiguous
- Property 2:
  D_1 and D_2 unambiguous ➔ $|D_1 + D_2| < |D_1| + |D_2|$
- Unambiguous automata may be freely combined with no state-space explosion

No State Explosion



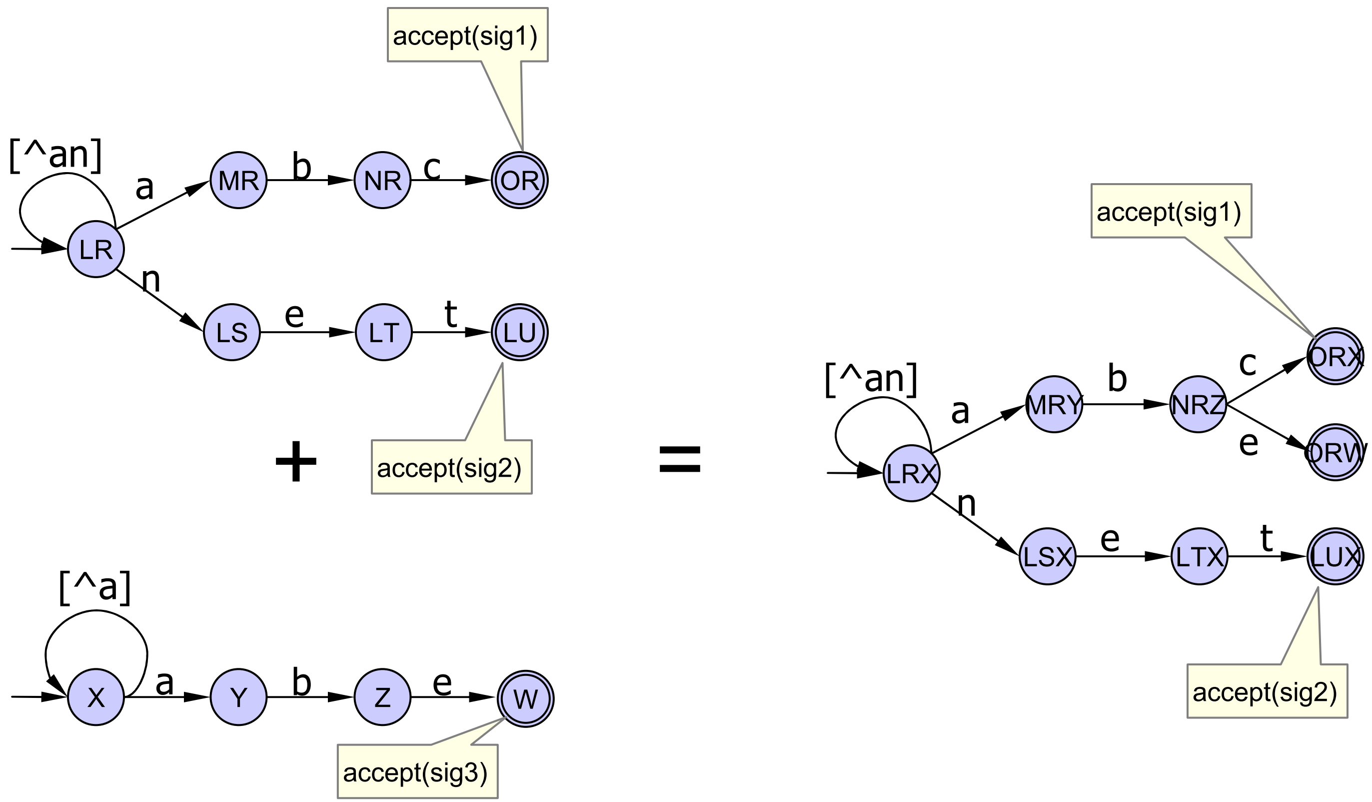
No State Explosion

/.*net/
/.*abc/

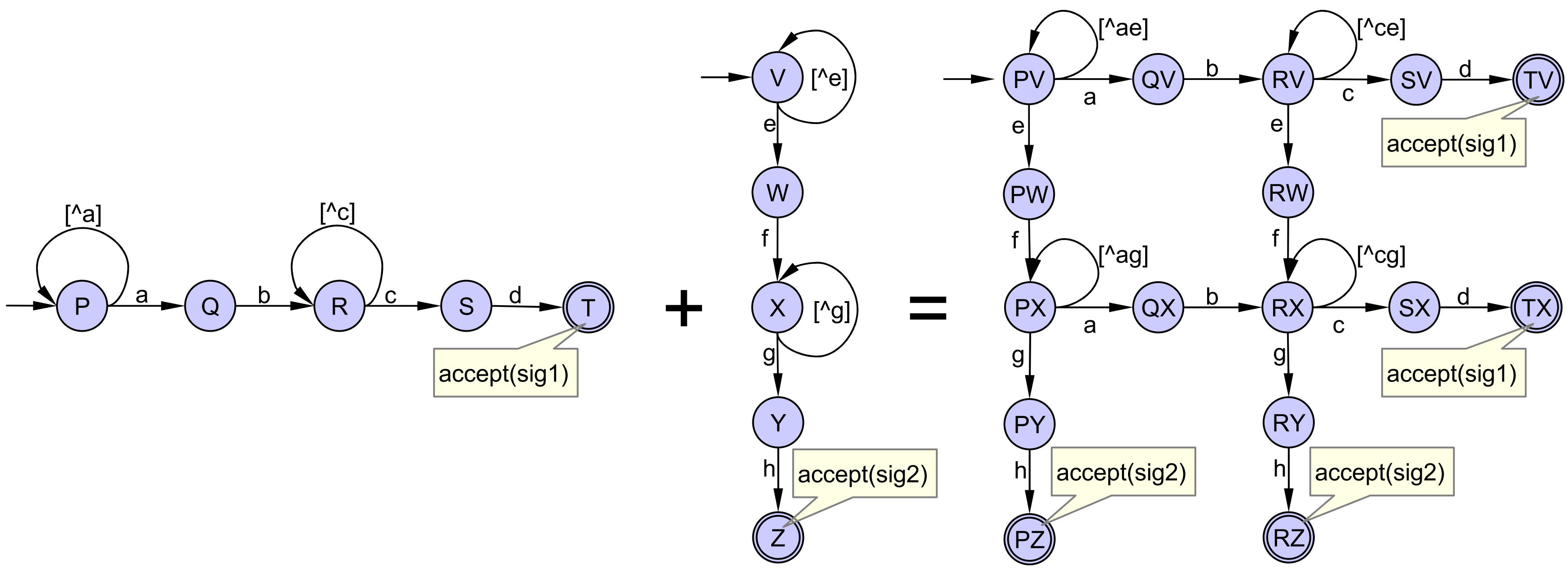
+

=

/.*abe/

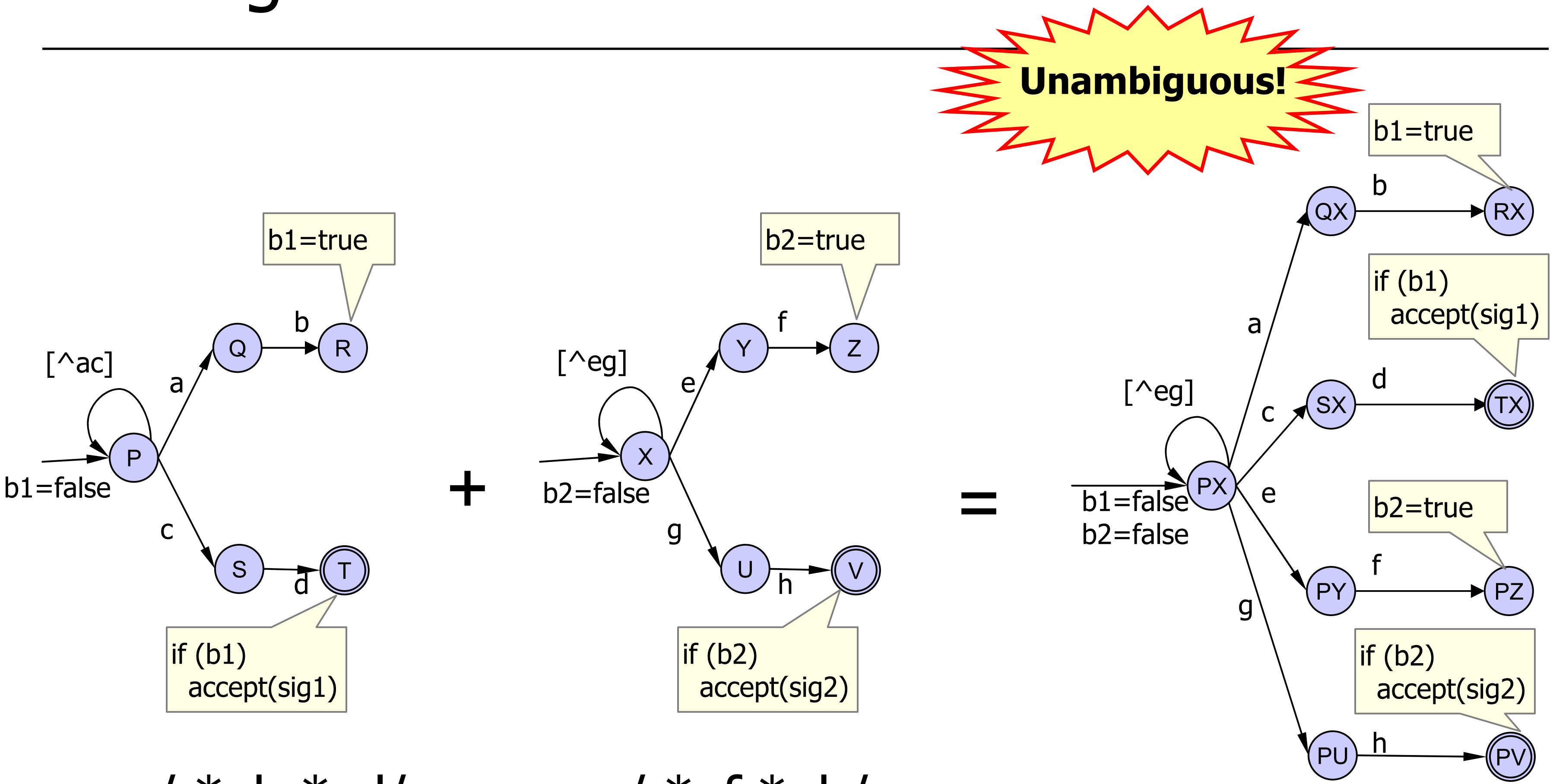


Exponential Explosion



/.*ab.*cd/ /.*ef.*gh/

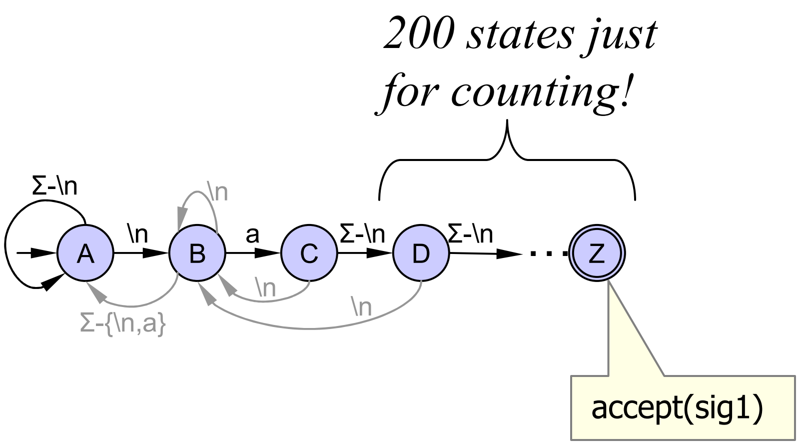
Adding a Bit

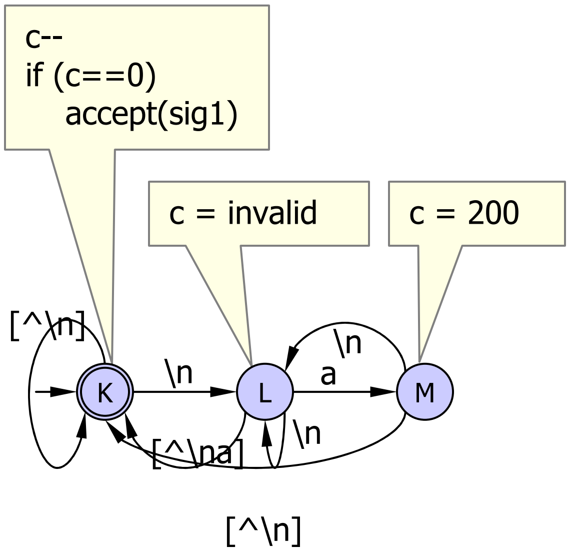


/.*ab.*cd/

/.*ef.*gh/

XFAs with Counters

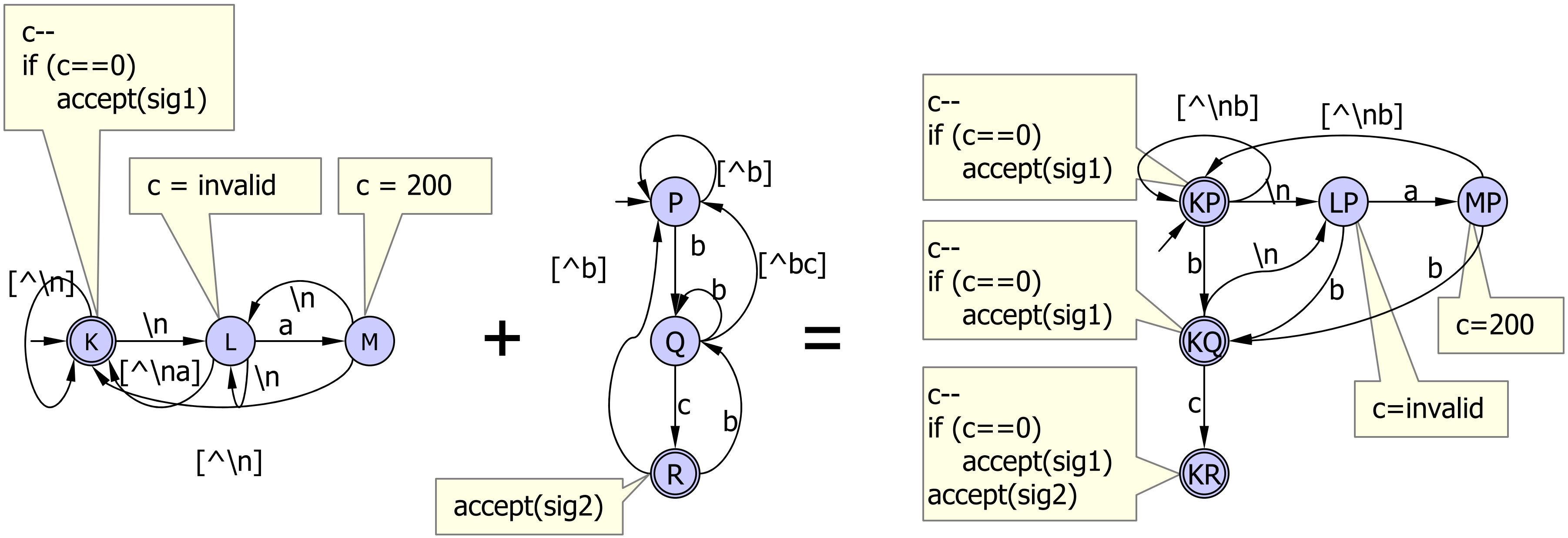




/.*\na[^\n]{200}/

/.*\na[^\n]{200}/

XFAs with Counters



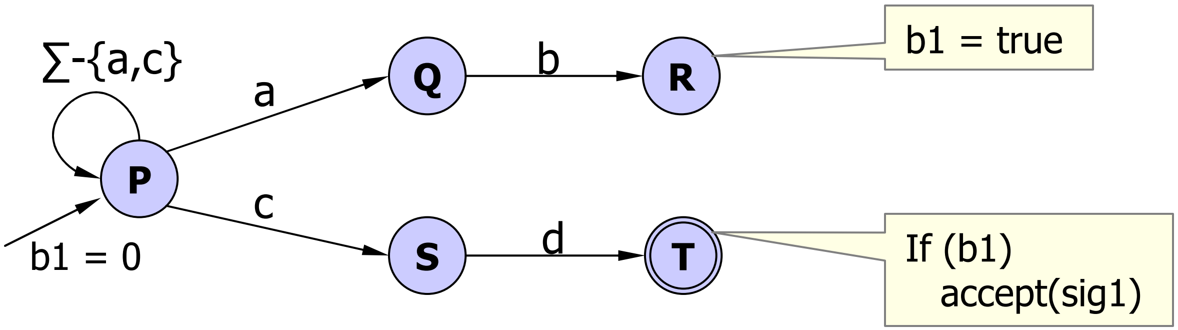
/.*\na[^\n]{200}/ /.*bc/

Key Contribution

- Ambiguity the culprit in state-space explosion
- XFAs provide a mechanism for controlling ambiguity

XFA Model

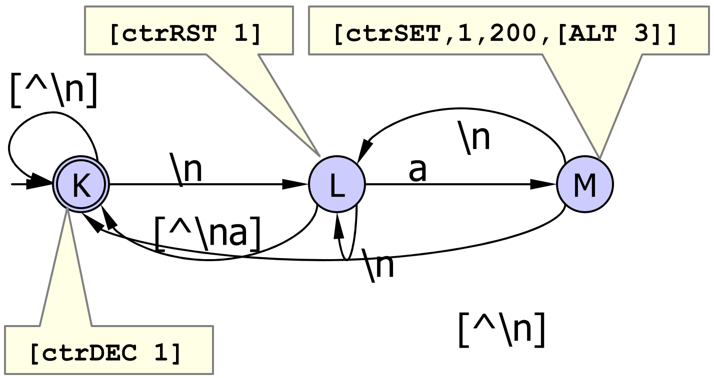
- Start with a DFA, add *update functions* to states



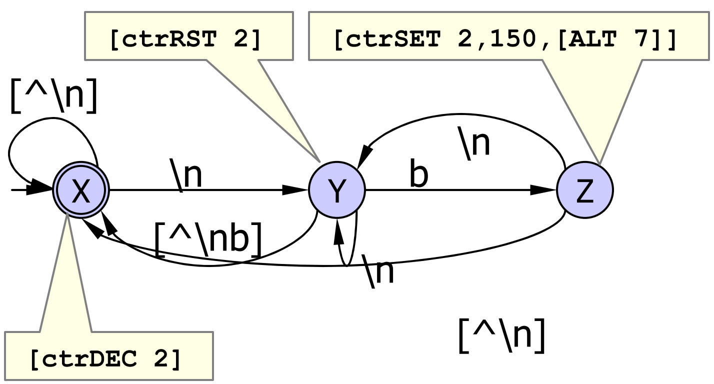
- States and transitions
  - Transitions a function of states and input
- Per-state update and test functions
  - Variable Update a function of states and variable values

Combining XFAs

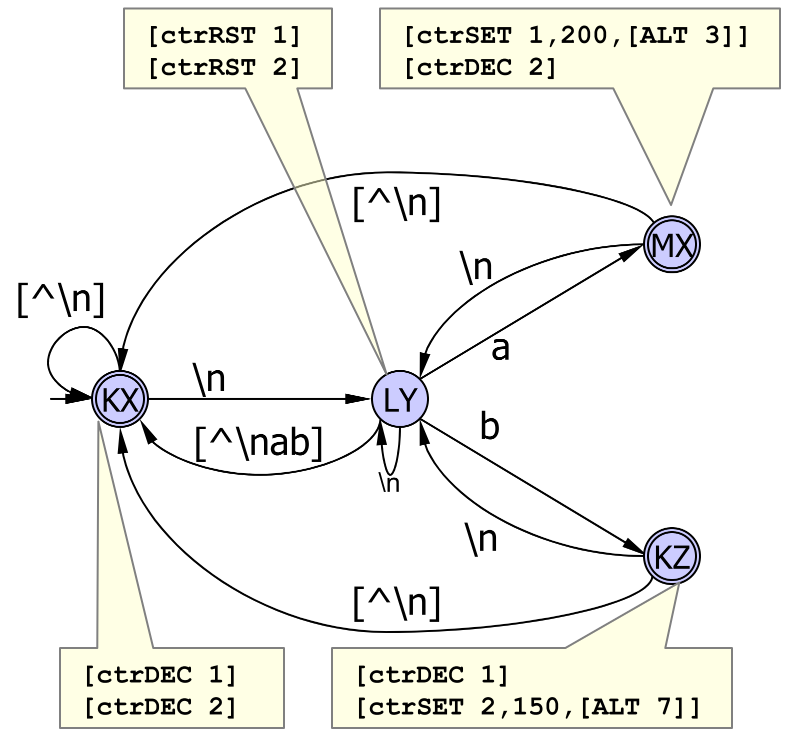
/.*\na[^\n]{200}/



+



=



- append instructions to corresponding states

/.*\nb[^\n]{150}/

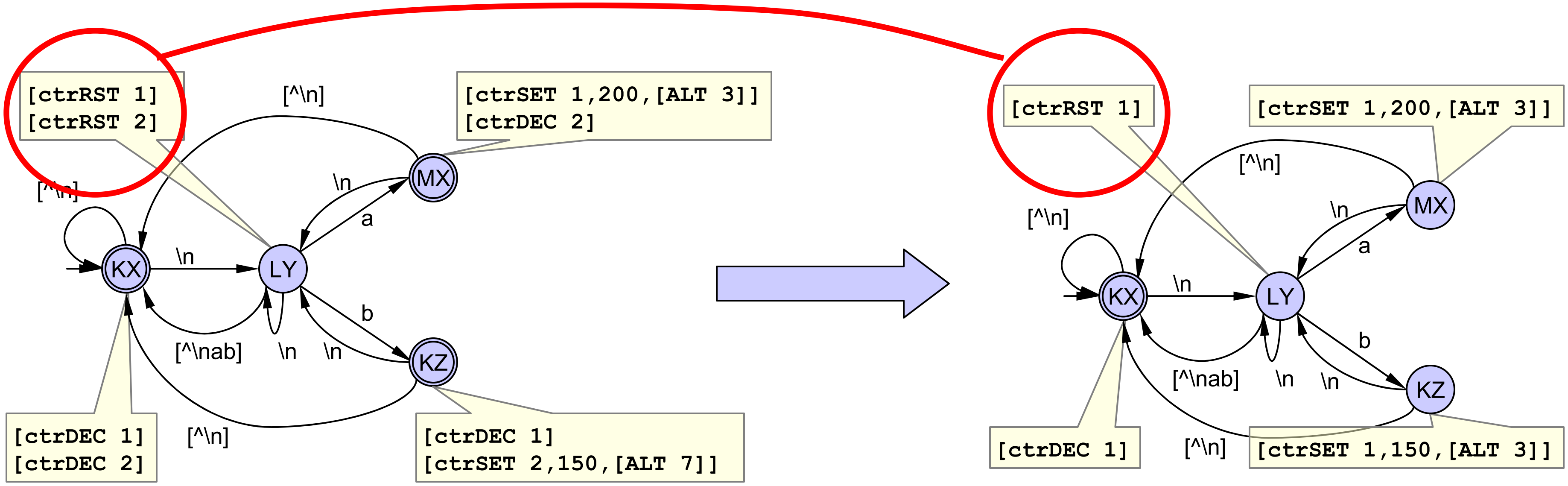
XFA Optimization

- XFA approach: construct individual XFAs, then combine
- Combination “collects” many variables and instructions
  - affects memory size, execution time, per-flow state
- Idea: borrow techniques used in compiler optimization

Optimization: Combine Independent Vars

- Analogous to register allocation used by compilers
  - Map many distinct *logical* vars to fewer *physical* vars

- Reduces instruction count (execution time) and number of variables (per-flow state)

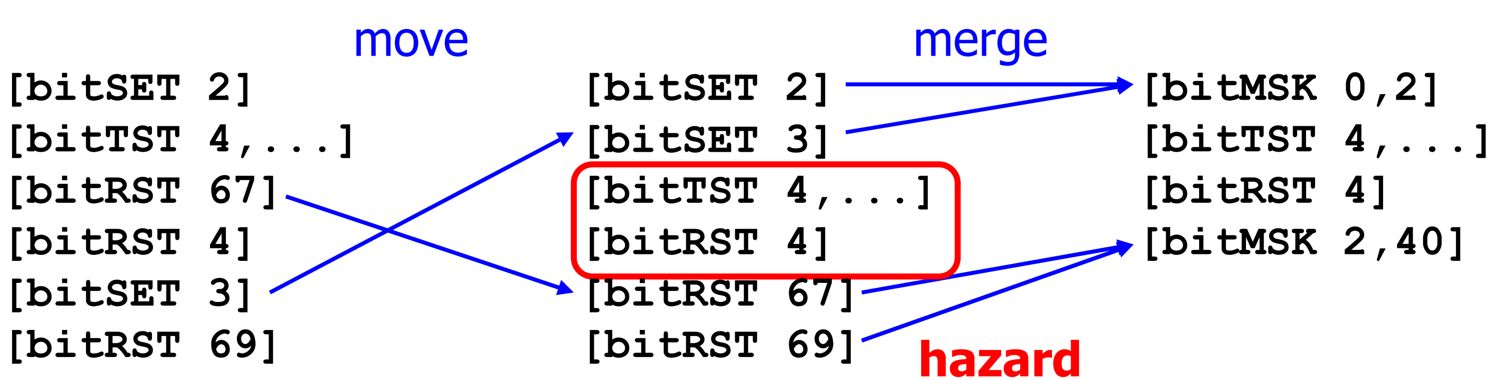


Optimization: Code Motion

- Analogous to code motion used by compilers

- *move* instructions to make bits adjacent,
  merge adjacent instructions to a single operation
  - Align on word boundaries, watch out for hazards

- Reduces instruction count (execution time)



Experiment Highlights

- Sizes of combined XFAs up to 10,000x smaller than combined DFAs
  - XFAs typically smaller *and* faster than other methods

- Optimization techniques significantly reduce instruction lengths and per-flow state requirements

Experiment Methodology

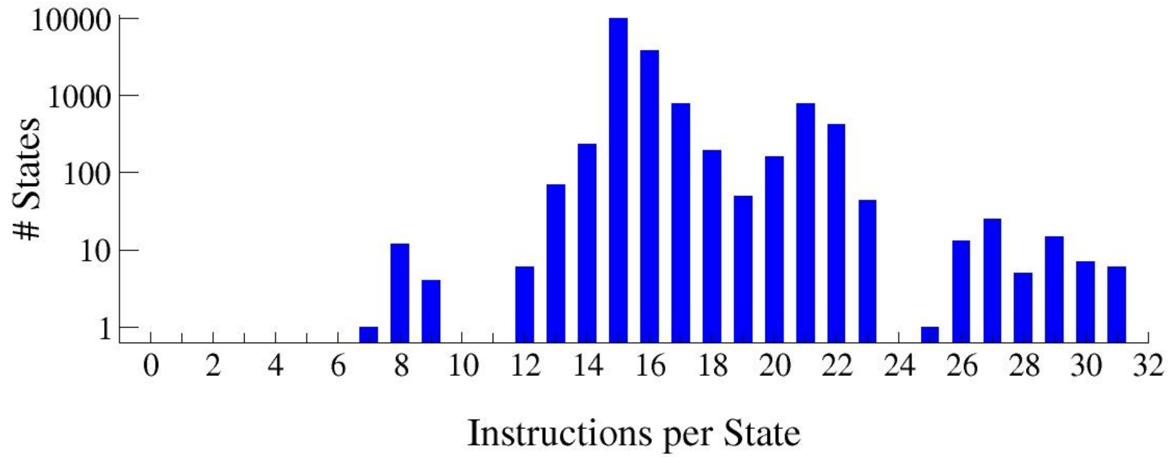
- Extracted FTP, SMTP, and HTTP regular expressions from Snort and Cisco rule sets
  - Constructed XFAs and DFAs for each signature
  - Separately combined XFAs, DFAs per-protocol

Signature Set	Num Sigs	# States (DFA)	# States (XFA)
Cisco FTP	38	> 3.1 M	527
Cisco SMTP	102	> 3.1 M	3,879
Cisco HTTP	551	> 3.1 M	11,982
Snort FTP	72	> 3.1 M	769
Snort SMTP	56	> 3.1 M	2,415
Snort HTTP	863	> 3.1 M	15,266

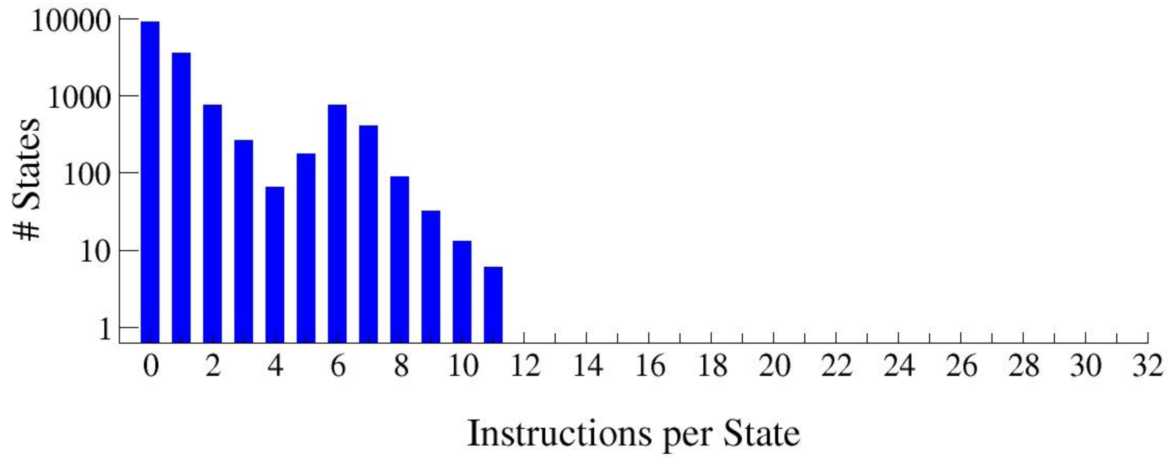
Optimization: Instruction Counts

Signature Set: Snort HTTP

Before Optimization



After Optimization



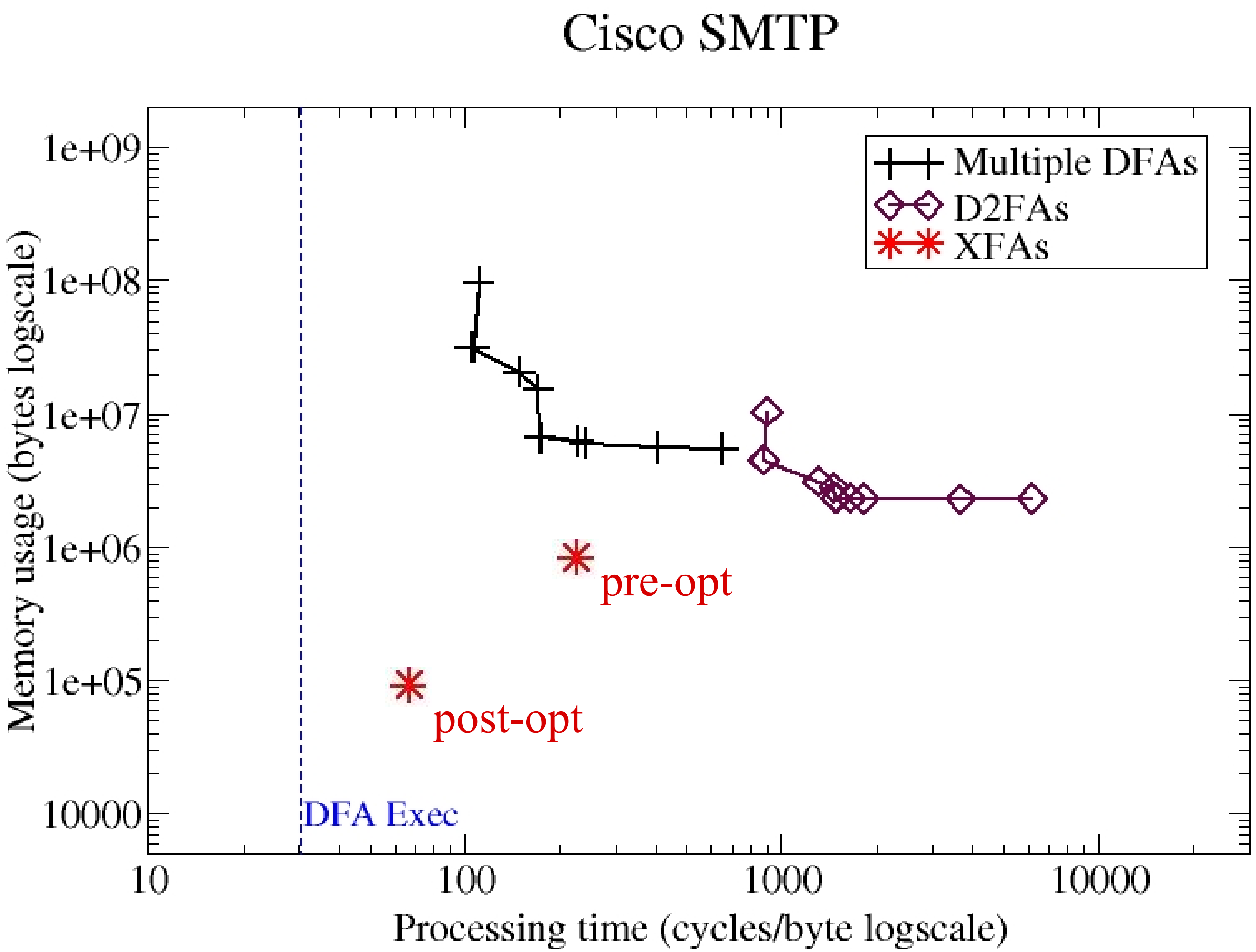
Optimization: Per-Flow State

Table: Per-flow state (in bytes) before and after optimization

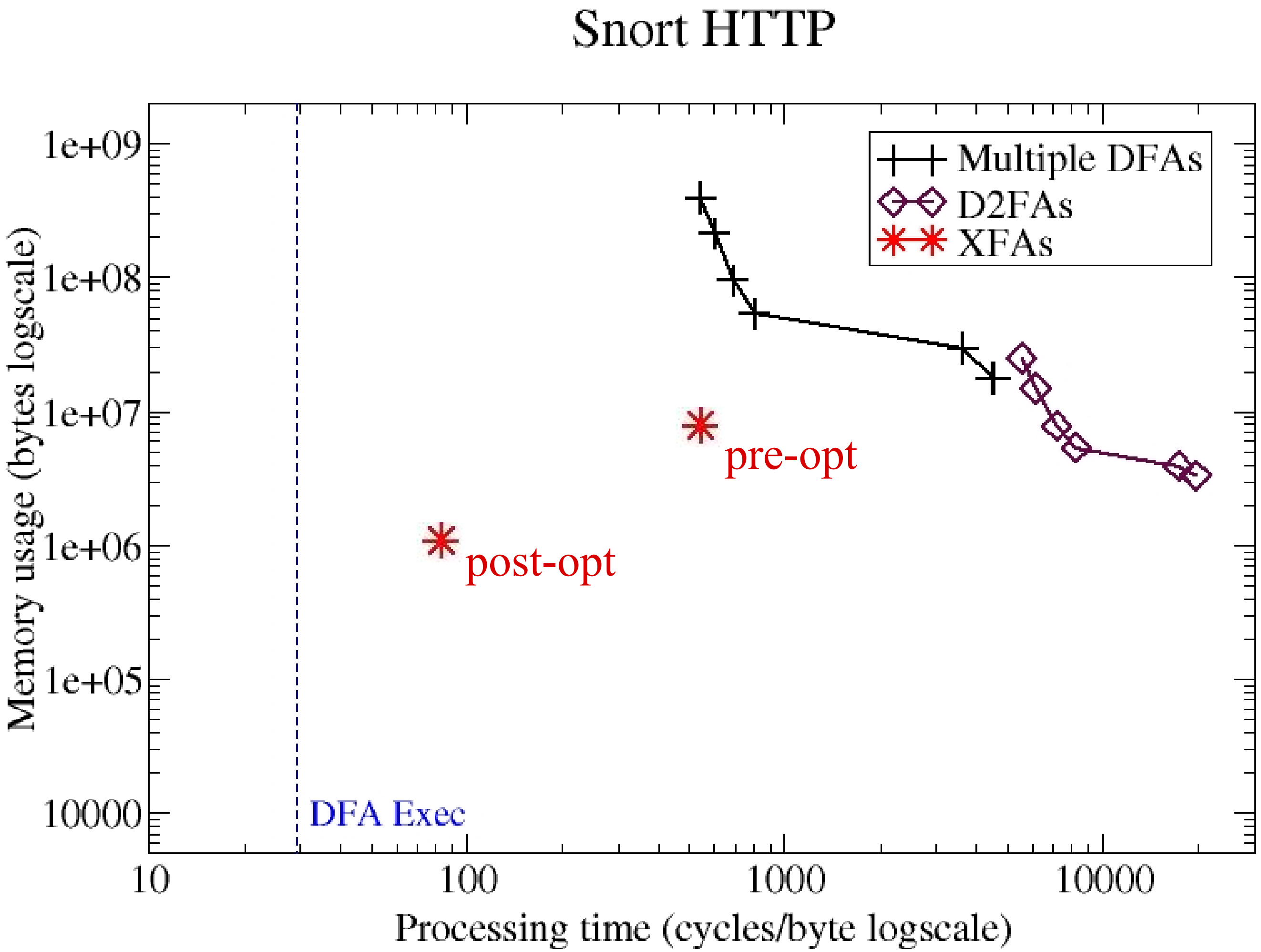
Signature Set	XFA Optimization	
	Before	After
Cisco FTP	28	10
Cisco SMTP	9	7
Cisco HTTP	24	7
Snort FTP	95	11
Snort SMTP	66	23
Snort HTTP	54	36

Summary: **~3x reduction**

Performance: Exec Time v. Memory



Performance: Exec Time v. Memory



Conclusion

- Deep Packet Inspection increasingly important
- Ambiguity is the culprit for state-space explosion
  - Control ambiguity with XFAs
- XFA Model provides framework for optimizations to be systematically applied
  - Optimization effects are significant
- XFAs smaller *and* faster

Deflating the Big Bang: Fast and Scalable Deep Packet Inspection with Extended Finite Automata

Thank you

intentionally blank