Another Brick off The Wall: Deconstructing Web Application Firewalls Using Automata Learning

George Argyros, Ioannis Stais

Joint Work with:
Suman Jana, Angelos D. Keromytis, Aggelos Kiayias

Overview

- A journey in the world of:
  - Code Injection attacks.
  - Web Application Firewalls.
  - Parsers.
  - Learning algorithms.
- And newly discovered vulnerabilities :)

Code Injection Attacks

- SQLi, XSS, XML, etc…
- Not going anywhere anytime soon.
- 14% increase in total web attacks in Q2 2016 [1]
- 150% - 200% increase in SQLi and XSS attacks in 2015 [2]

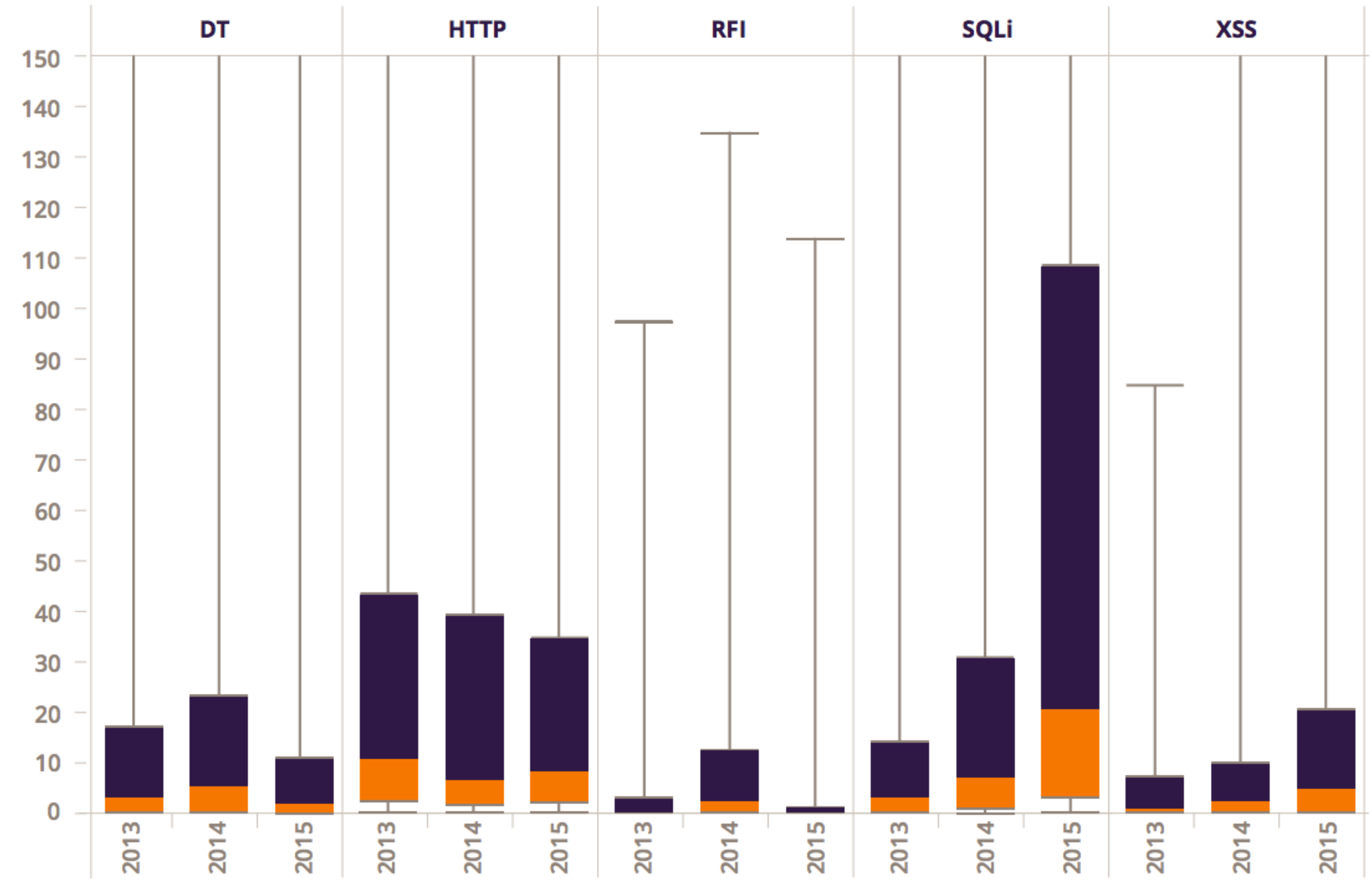
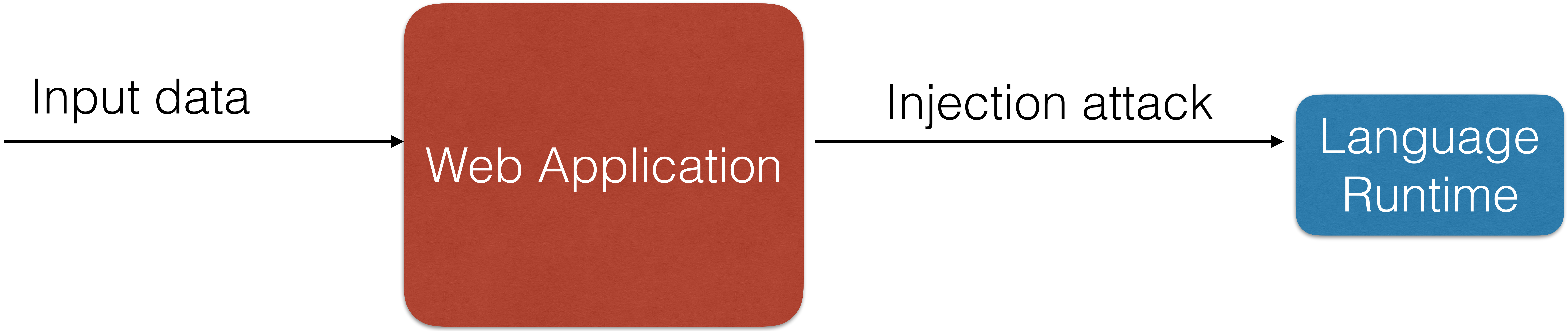


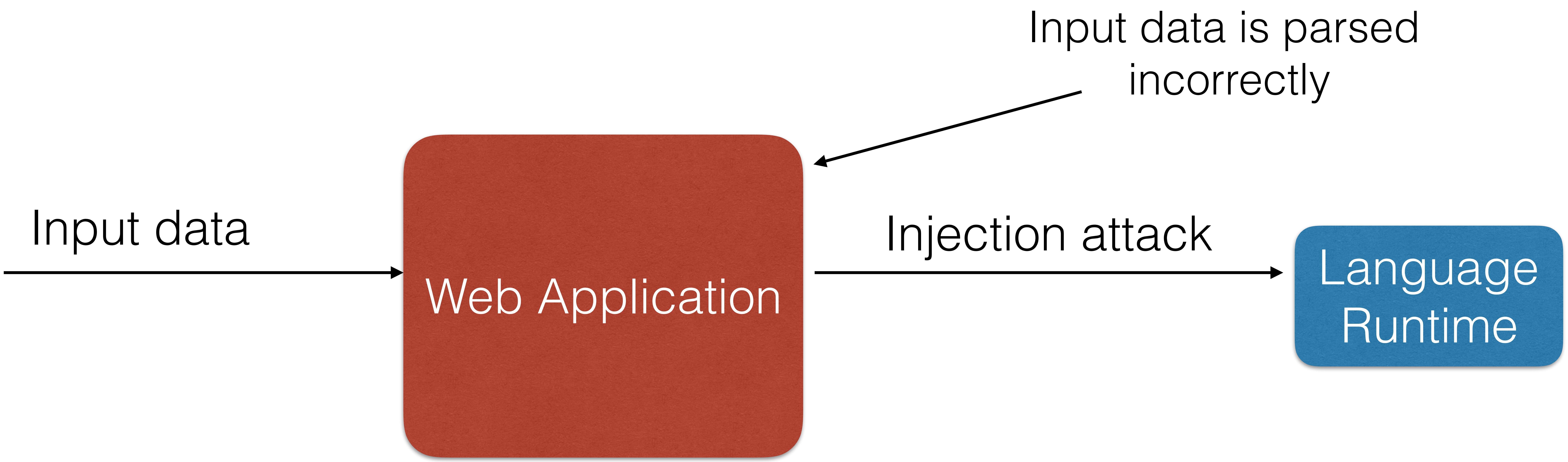
Figure 1: Comparison of Number of Incidents Between Years

[1] akamai's [state of the internet] / security Q2 2016 executive review
[2] Imperva: 2015 Web Application Attack Report (WAAR)

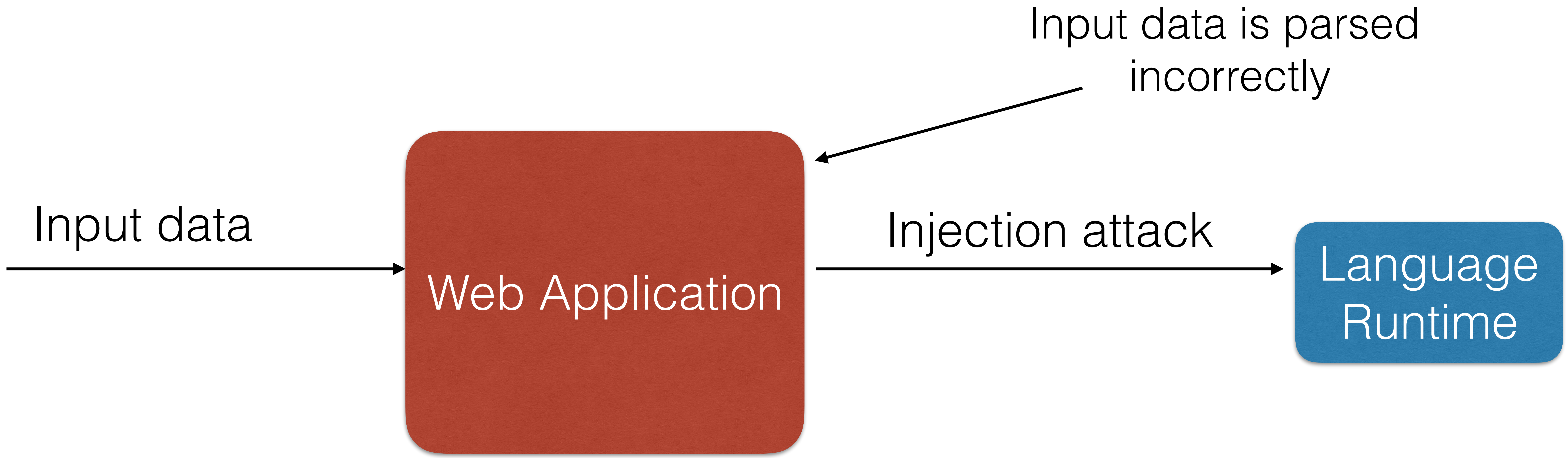
Code Injection is a Parsing Problem



Code Injection is a Parsing Problem



Code Injection is a Parsing Problem



Web application parsers are doing a really bad job in parsing user inputs.

Web Application Firewalls

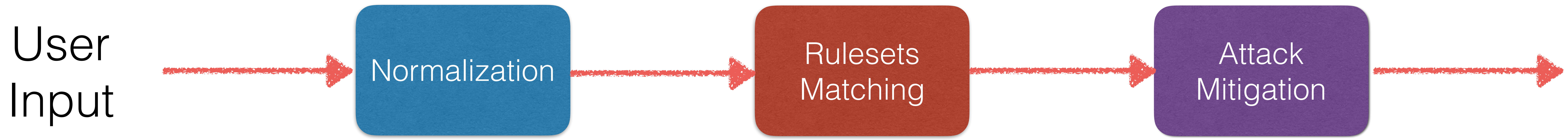
(or solving parsing problems with parsing)

Web Application Firewalls

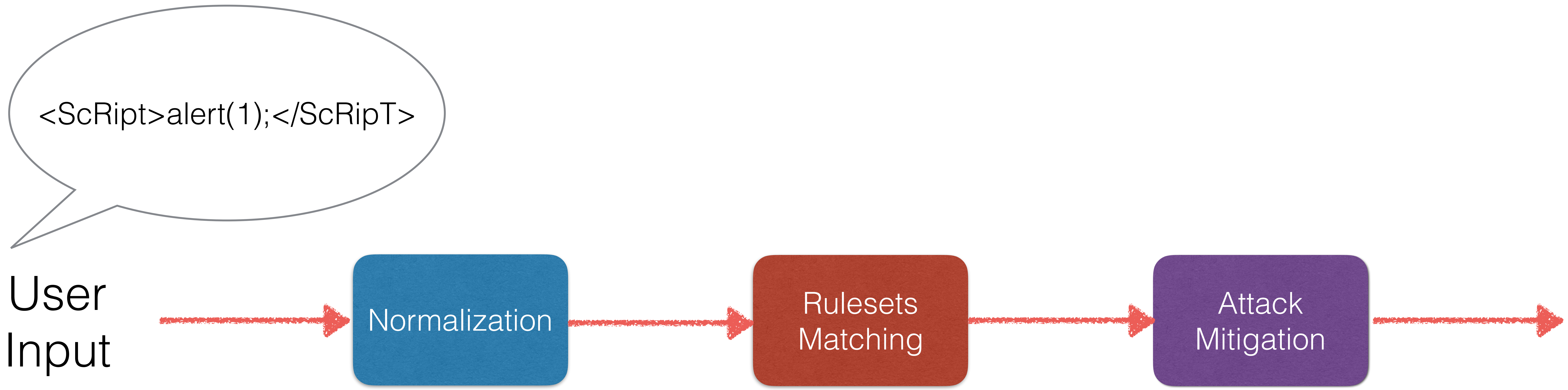
- Monitor traffic at the Application Layer: *Both HTTP Requests and Responses.*
- Detect and Prevent Attacks.
- Cost-effective compliance with PCI DSS requirement 6.6 [1]

[1] PCI DSS v3.2

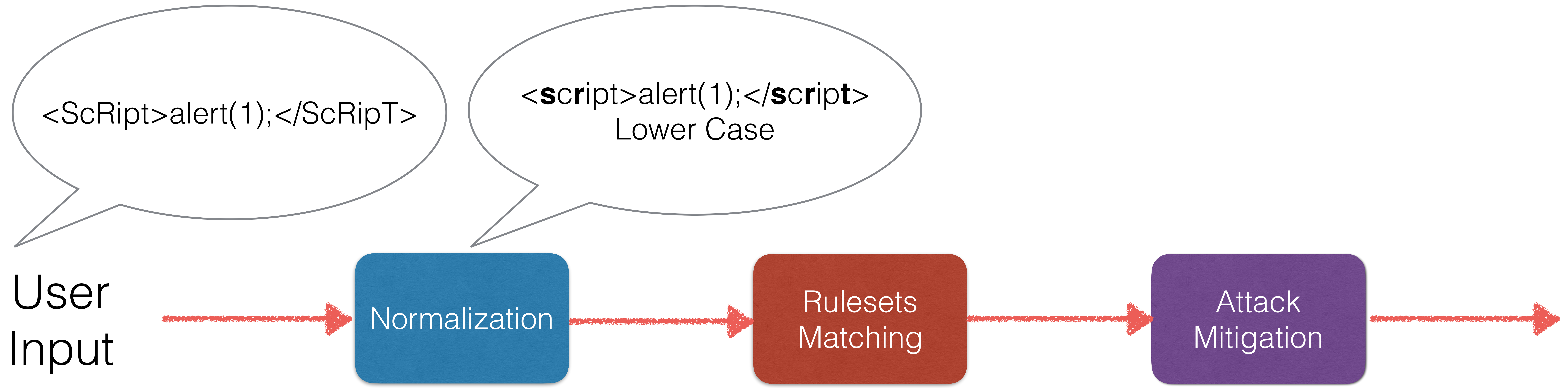
WAFs Internals



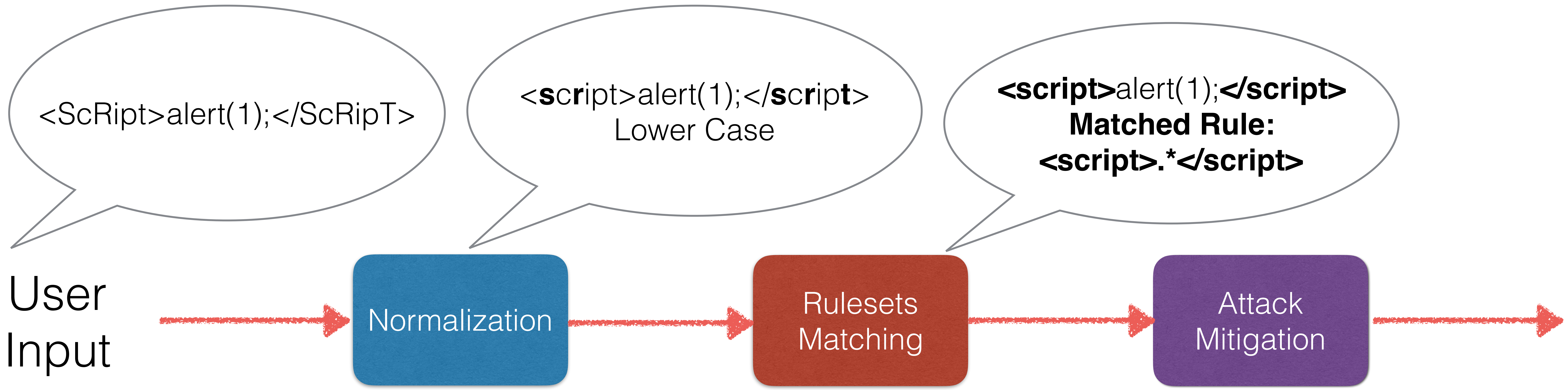
WAFs Internals



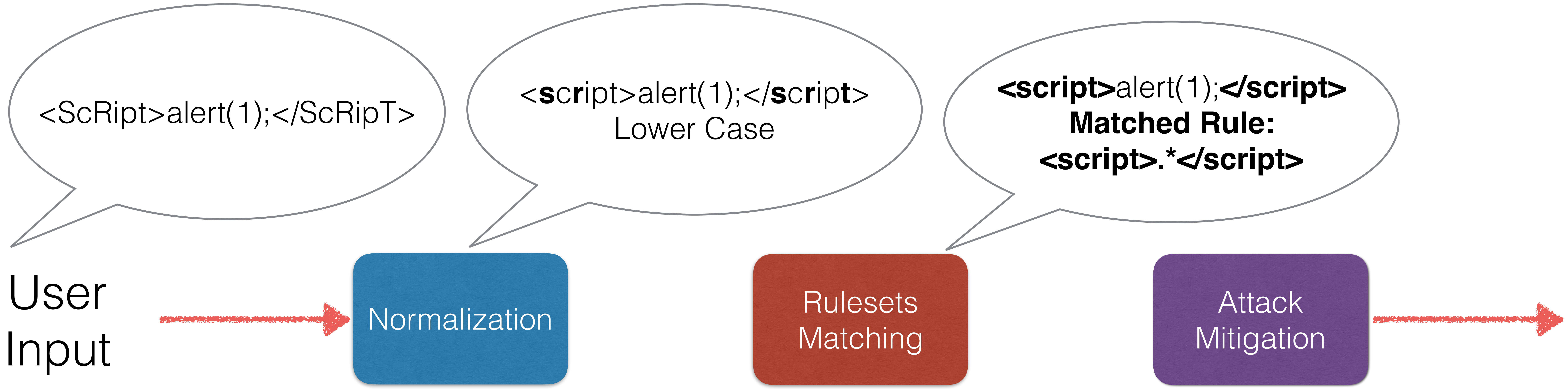
WAFs Internals



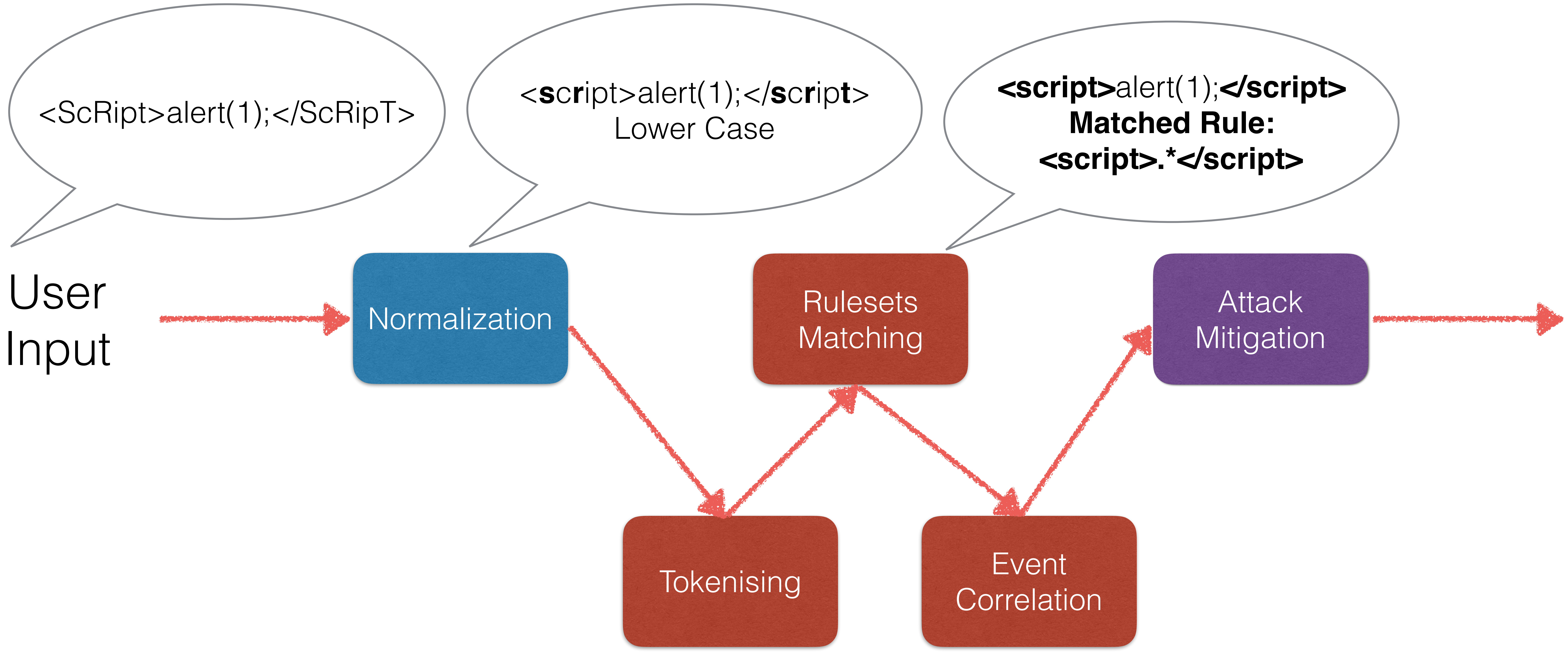
WAFs Internals



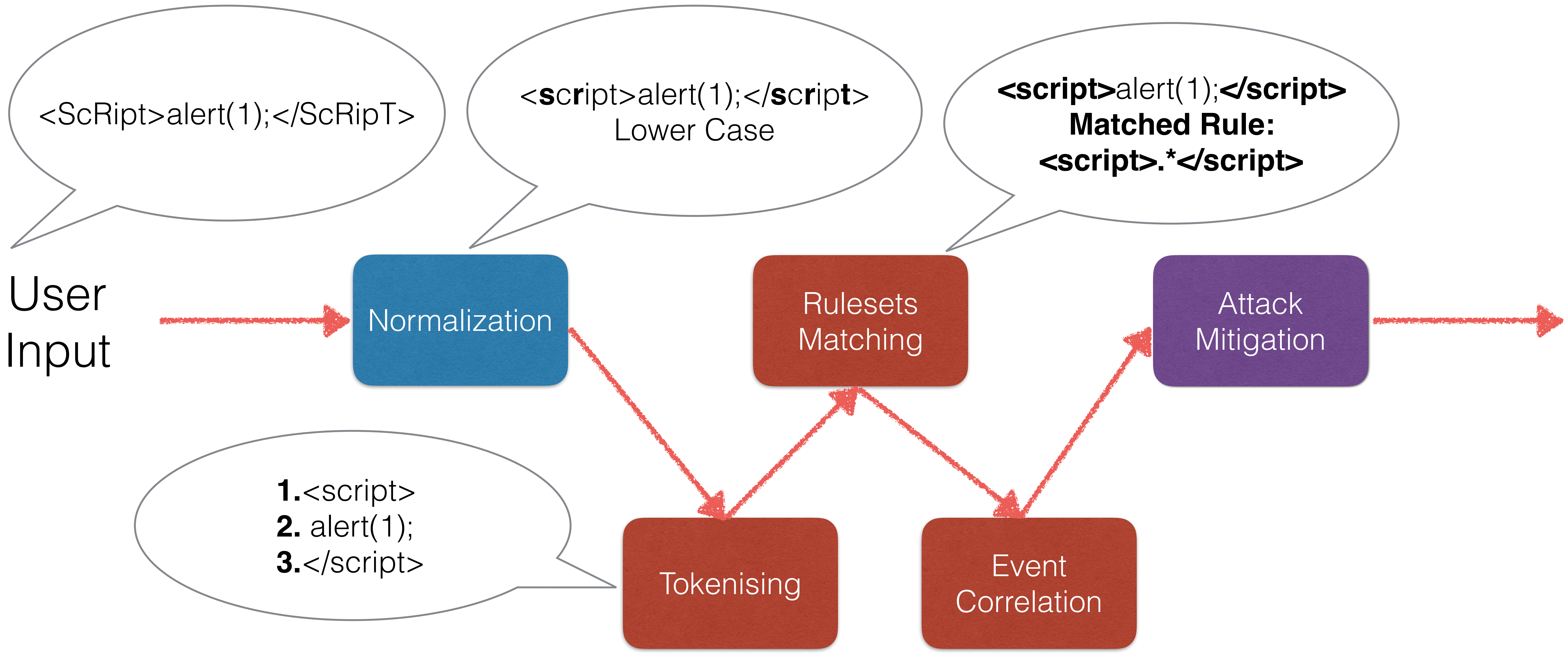
WAFs Internals



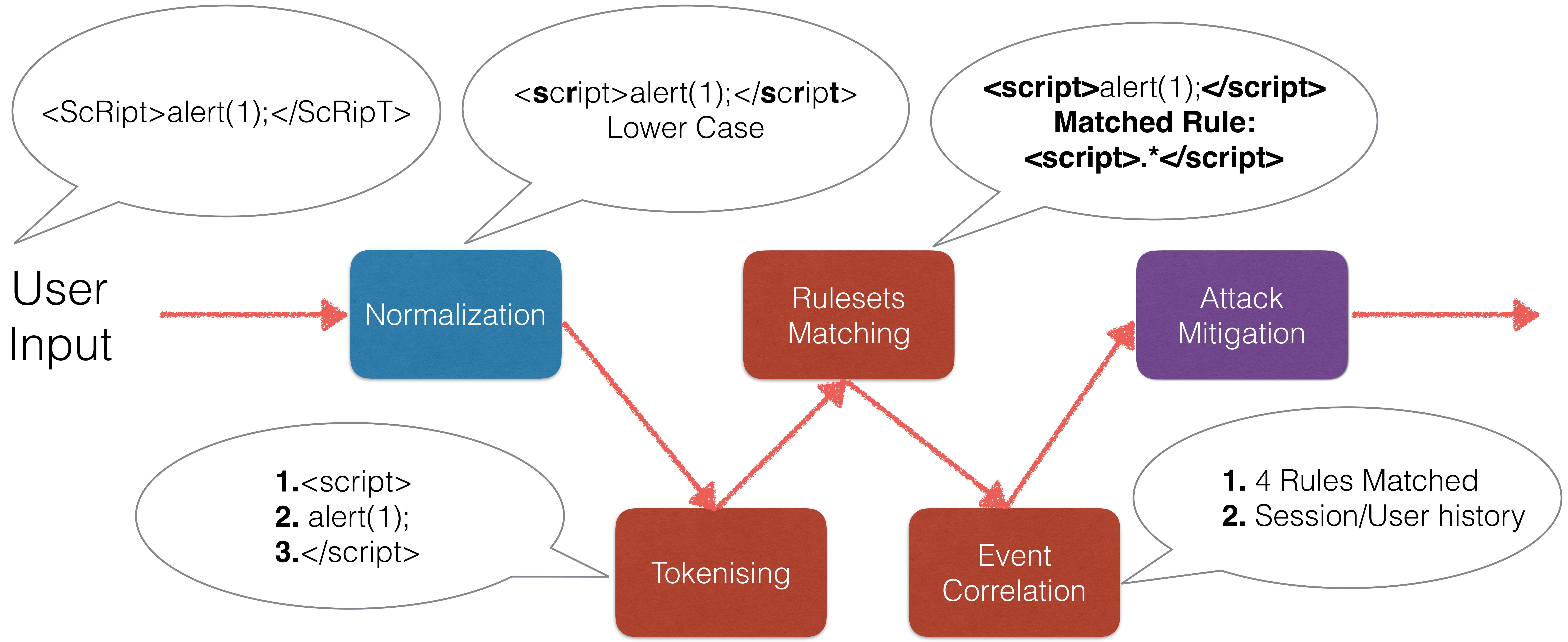
WAFs Internals



WAFs Internals



WAFs Internals



WAF Rulesets

- **Signatures:** *Strings or Regular Expressions*

E.g., [PHPIDS Rule 54] Detects Postgres pg_sleep injection, waitfor delay attacks and database shutdown attempts:

*(?:select\s*pg_sleep)|(?:waitfor\s*delay\s?"+\s?\d)|(?:;\s*shutdown\s*(?:;|--|#|\/*|{))*

WAF Rulesets

- **Signatures:** *Strings or Regular Expressions*
- **Rules:** Logical expressions and Condition/Control Variables

E.g., ModSecurity CRS Rule 981254:

```
SecRule REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|!REQUEST_COOKIES:/
_pk_ref/|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "(?i:(?:select\s*?
pg_sleep)|(?:waitfor\s*?delay\s?[\"'`´’‘]+\s?\d)|(?:;\s*?shutdown\s*?(?:;|--|#|\/\*|{)))" “phase:
2,capture,t:none,t:urlDecodeUni,block, setvar:tx.sql_injection_score=
+1,setvar:tx.anomaly_score=+%{tx.critical_anomaly_score},setvar:'tx.%{tx.msg}-
OWASP_CRS/WEB_ATTACK/SQLI-%{matched_var_name}=%{tx.0}'"
```

WAF Rulesets

- **Signatures:** *Strings or Regular Expressions*
- **Rules:** Logical expressions and Condition/Control Variables
- **Virtual Patches:** Application Specific Patches

E.g., ModSecurity: Turns off autocomplete for the forms on login and signup pages

```
SecRule REQUEST_URI "^(\/login|\/signup)" "id:1000,phase:4,chain,nolog,pass"
SecRule REQUEST_METHOD "@streq GET" "chain"
SecRule STREAM_OUTPUT_BODY "@rsub s/<form /<form autocomplete=\"off\" /"
```

WAF Rulesets

- **Signatures:** *Strings or Regular Expressions*
- **Rules:** Logical expressions and Condition/Control Variables
- **Virtual Patches:** Application Specific Patches
- *PHPIDS has more than 420K states*
- Shared between different WAFs and Log Auditing Software: *PHPIDS, Expose, ModSecurity*

Why Bypasses Exist

Why Bypasses Exist

- Simple hacks:

- Lack of support for different protocols, encodings, contents, etc
- Restrictions on length, character sets, byte ranges, types of parameters, etc

Why Bypasses Exist

- Rulesets sharing mistakes:

- Normalisation and Rulesets Failure

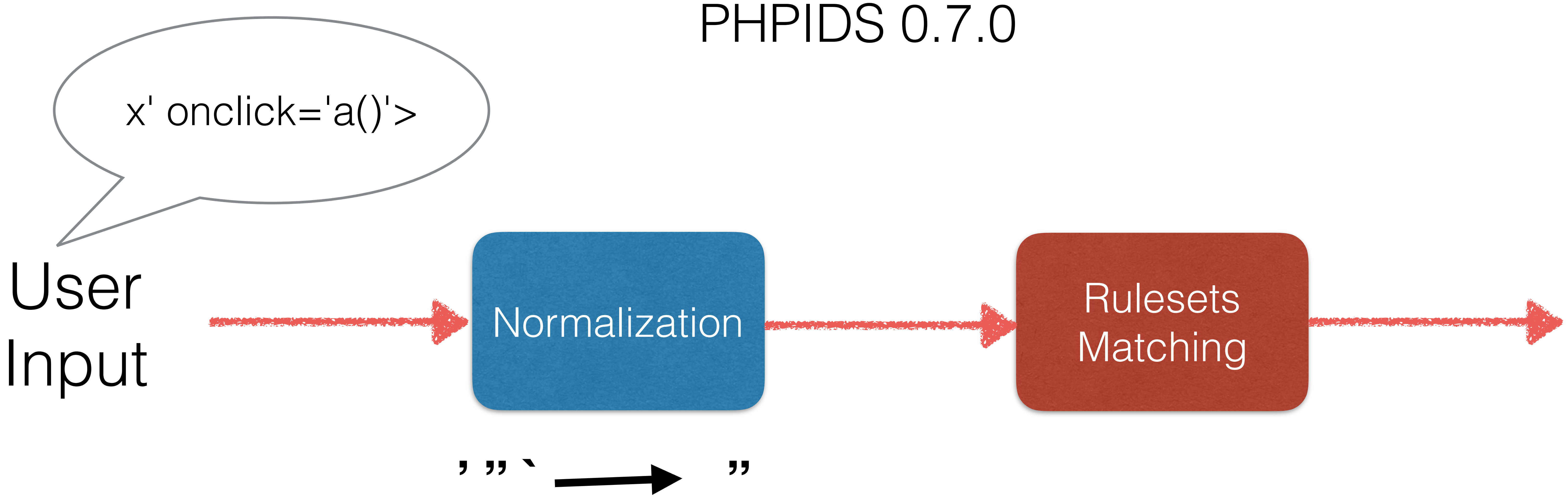
PHPIDS 0.7.0

Why Bypasses Exist

- Rulesets sharing mistakes:

- Normalisation and Rulesets Failure

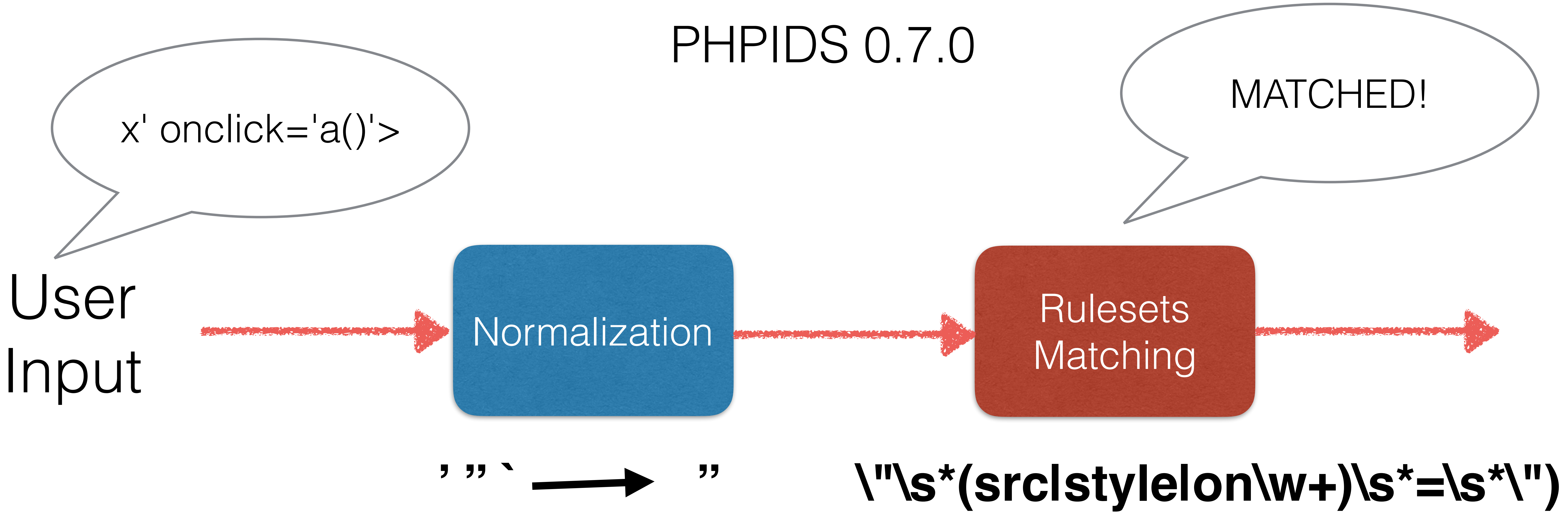
PHPIDS 0.7.0



Why Bypasses Exist

- Rulesets sharing mistakes:

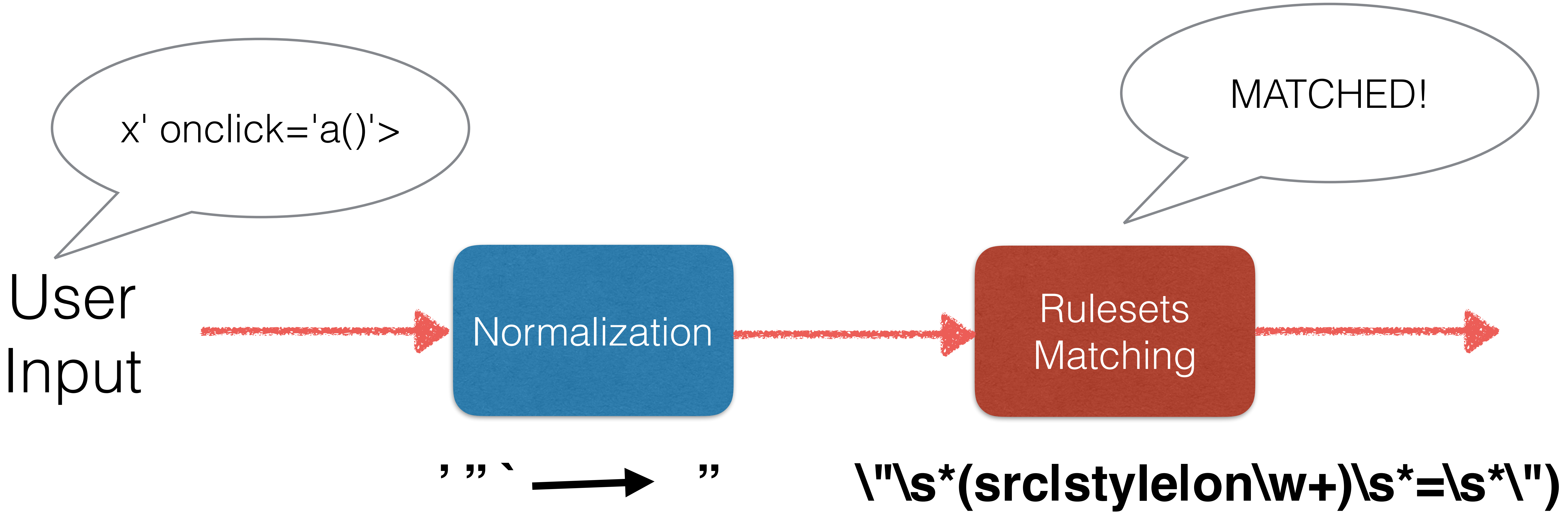
- Normalisation and Rulesets Failure



Why Bypasses Exist

- Rulesets sharing mistakes:

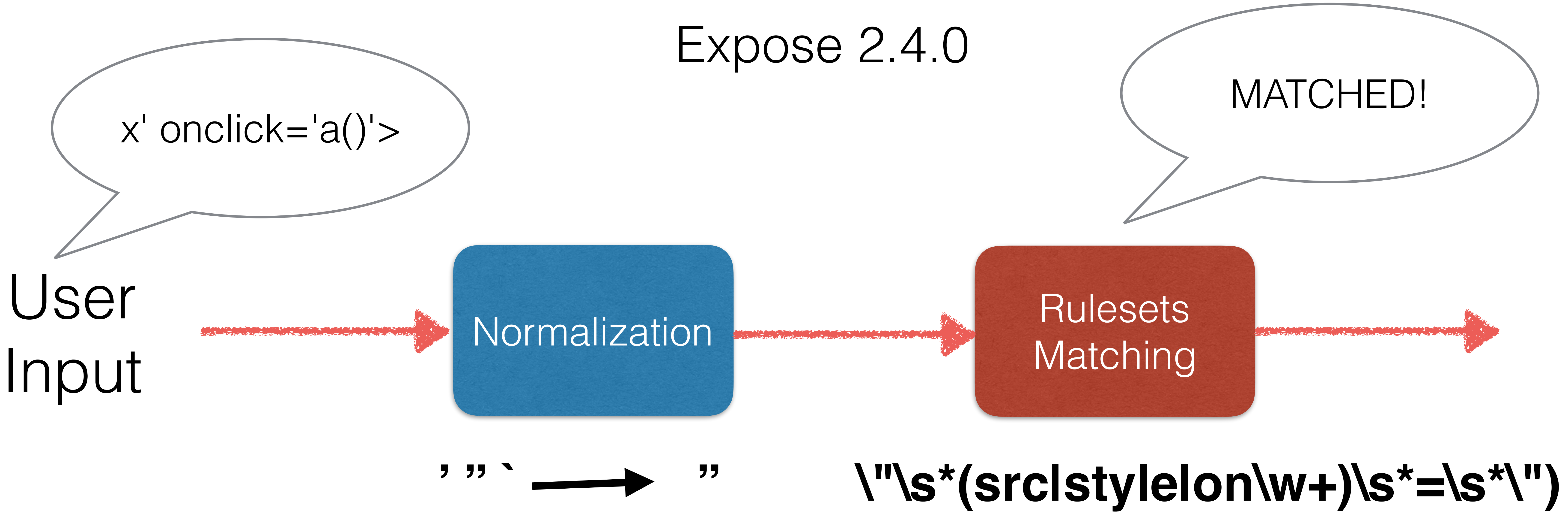
- Normalisation and Rulesets Failure



Why Bypasses Exist

- Rulesets sharing mistakes:

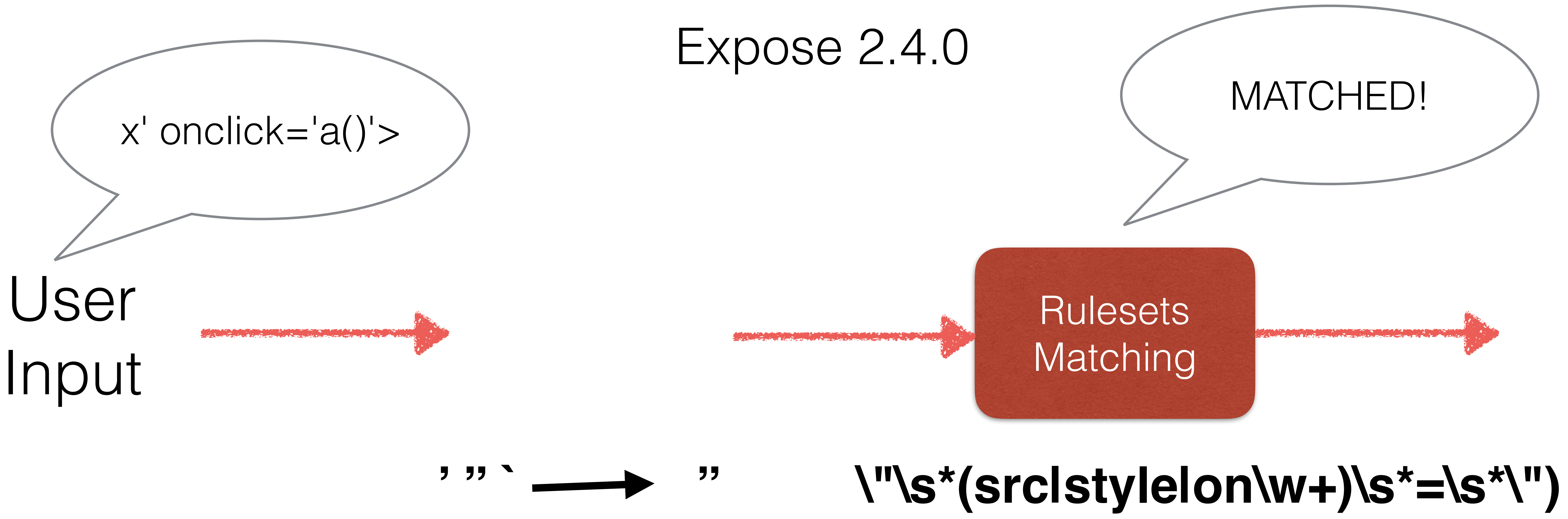
- Normalisation and Rulesets Failure



Why Bypasses Exist

- Rulesets sharing mistakes:

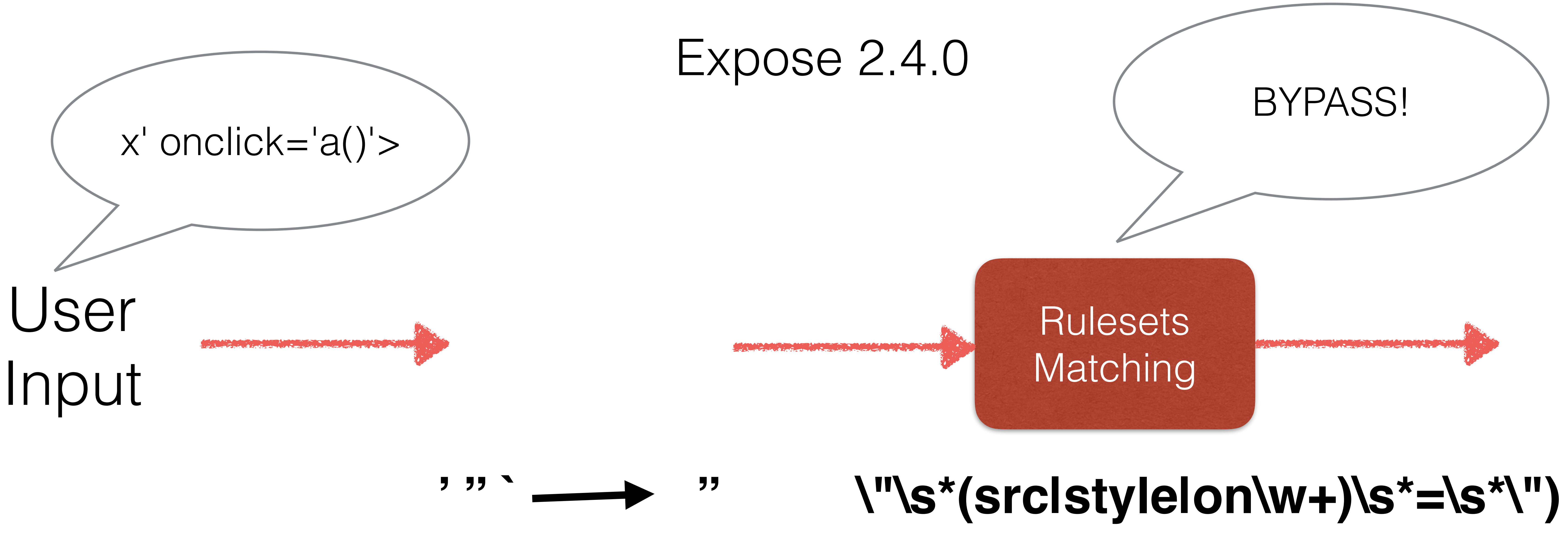
- Normalisation and Rulesets Failure



Why Bypasses Exist

- Rulesets sharing mistakes:

- Normalisation and Rulesets Failure



Why Bypasses Exist

- Critical WAF components are not being updated:

- E.g, ModSecurity *libinjection* library

REMOVED commented on Jun 29 owasp-modsecurity-crs contributor

I've been in touch with Nick recently. He was not able to reproduce the false negative I discovered in the reddit-XSS (see blogpost). The reason probably being, ModSec forked libinject instead of linking. So we are running on an outdated version of libinject.

Maybe we need a tag to collect all libinject false negatives and forward them upstream in batches.

REMOVED commented on Jun 29 owasp-modsecurity-crs contributor

REMOVED this was a serious concern with how it was used in v2 it being forked. In v3 it is required to be brought in as a submodule, so before you can compile it you must actually bring in an up to date copy from the repo. Has its negatives if an issue is introduced but also has its positives for situations

None yet

Labels

False Negative - Evasion

v3.1.0-rc1 Candidate Issue

Milestone

No milestone

Assignees

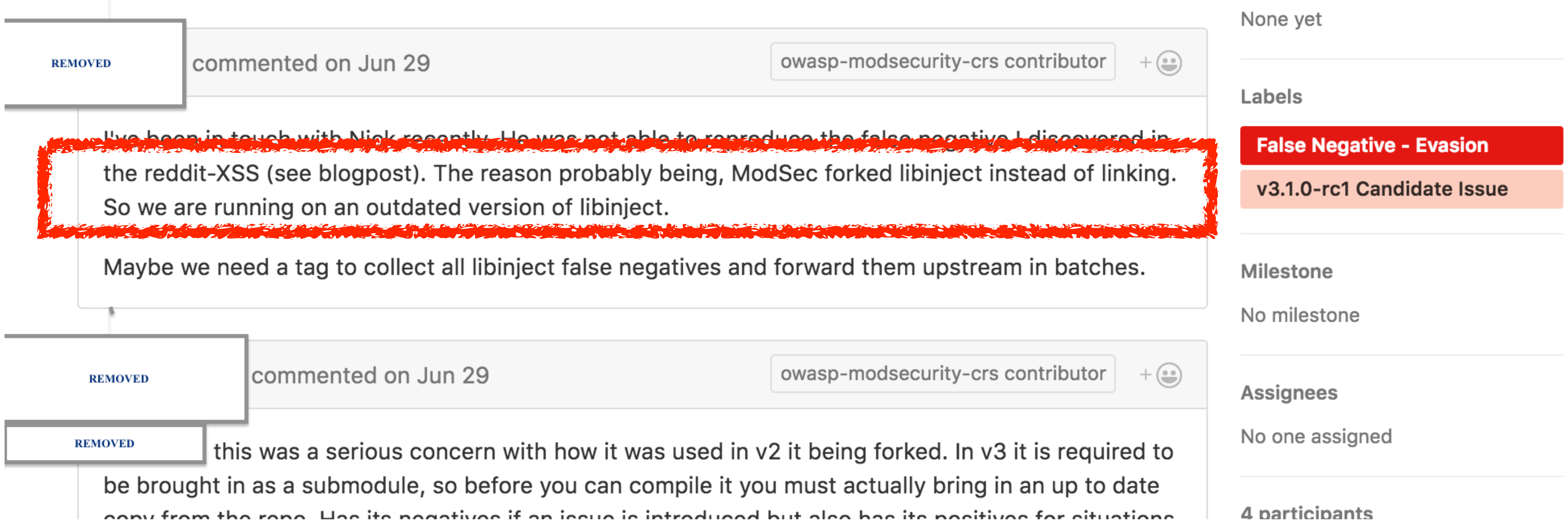
No one assigned

4 participants

Why Bypasses Exist

- Critical WAF components are not being updated:

- E.g, ModSecurity *libinjection* library



Why Bypasses Exist

- The Real Fundamental Reasons:

- Insufficient Signatures & Weak Rules
- Detecting vulnerabilities without context is HARD

Our Goal

1. Formalize knowledge in code injection attacks variations using context free grammars and automata.

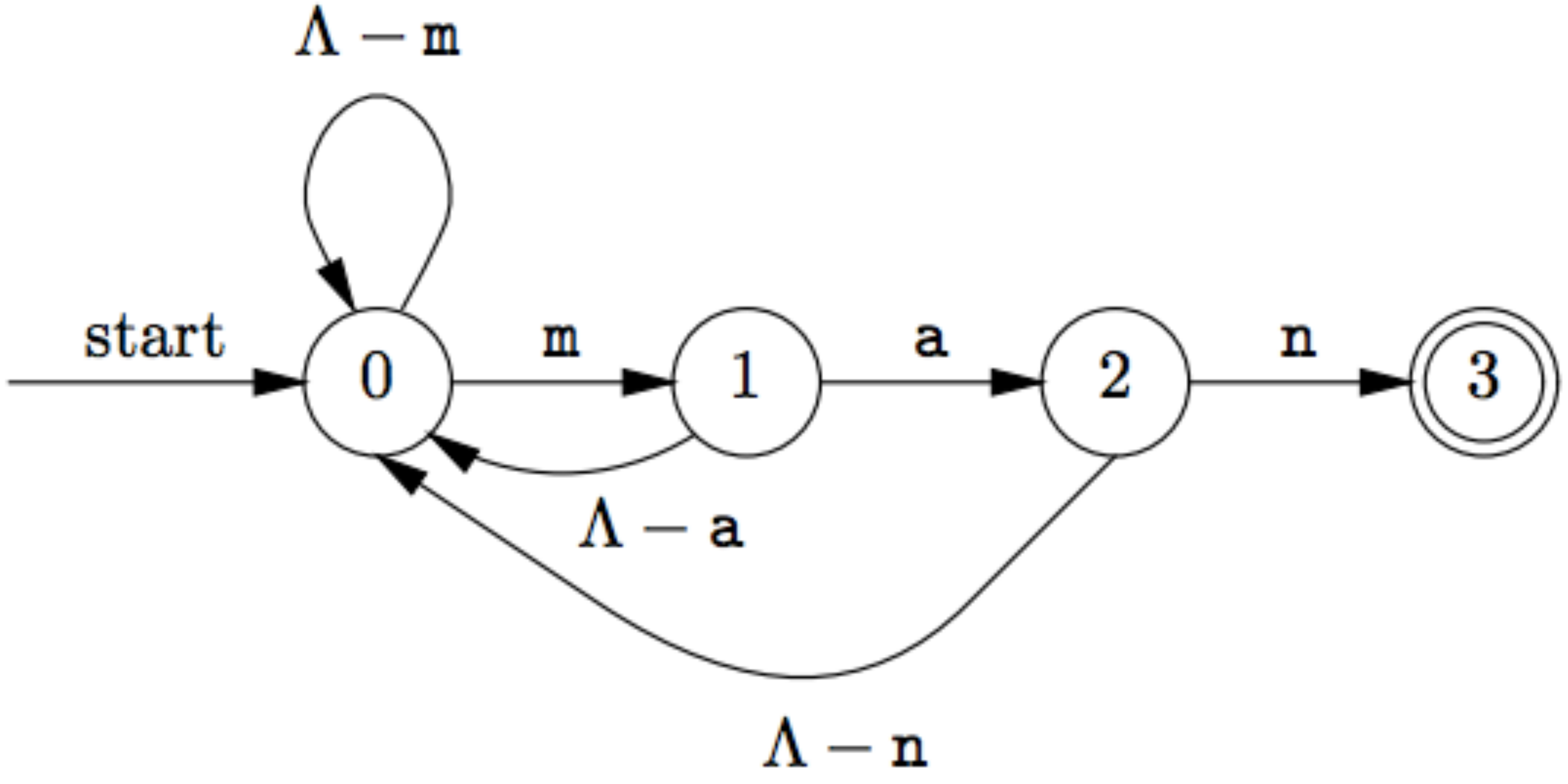
2. Use Learning algorithms to expand this knowledge by inferring system specifications.

Using parsers to break parsers

Regular Expressions and Finite Automata

Every regular expression can be converted to a Deterministic Finite Automaton.

(.*)man



Context Free Grammars

- Superset of Regular Expressions.
- Mostly used to write programming languages parsers.
- Equivalent to a DFA with a ***stack.***
- Can be used to count.
  - Example: matching parentheses.

$$\left.\begin{array}{l} E \rightarrow N \\ E \rightarrow E\ Op\ E \\ E \rightarrow (\ E\ ) \\ N \rightarrow N\ N \end{array}\right\}\ \text{Non Terminals}$$

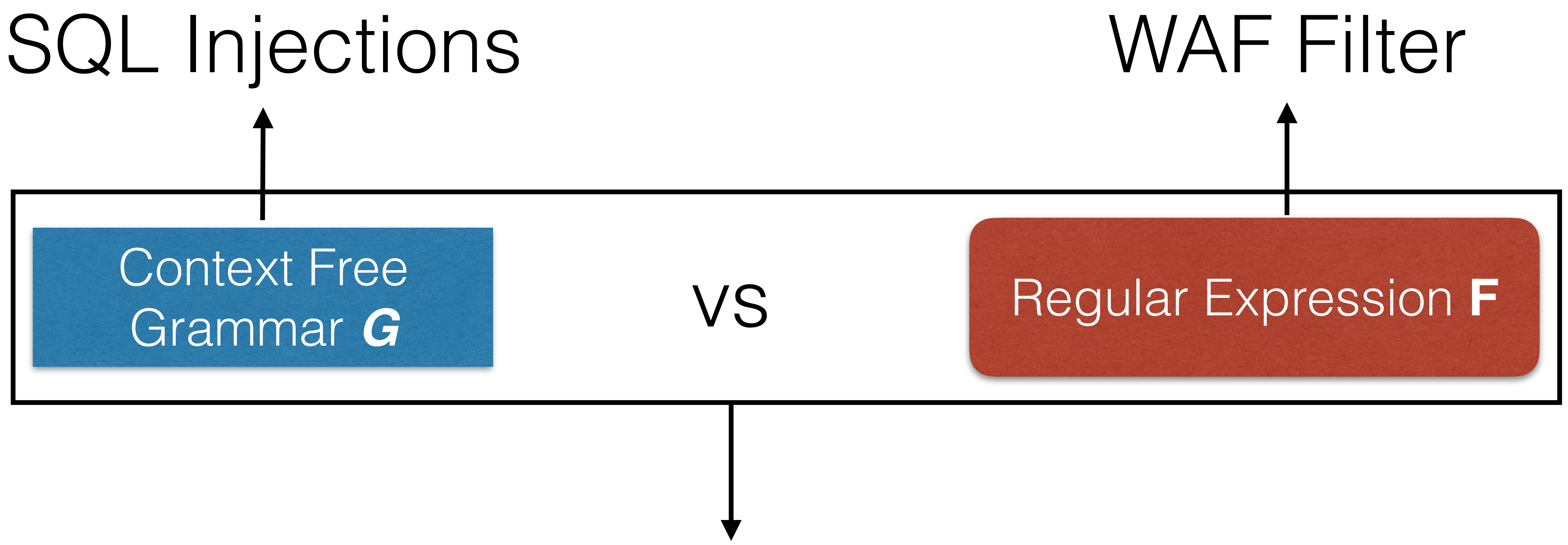
$$\left.\begin{array}{l} Op \rightarrow + \\ Op \rightarrow - \\ Op \rightarrow * \\ Op \rightarrow / \\ N \rightarrow [0\text{-}9] \end{array}\right\}\ \text{Terminals}$$

Attack of the Grammars

- Context Free Grammars can be used to encode attack vectors.
- Assume we would like to inject code into the query:
  - “SELECT * FROM users WHERE id=$id;”
- The valid suffixes (injections) for this query can be encoded as a CFG!

Why should I care?

Cross checking regular expressions with grammars is easy!



Find an SQL Injection attack in the Grammar **G**
which is not rejected by the filter **F**

However…

- In reality, we do not know the language parsed by most implementations.
  - MySQL is parsing a different SQL flavor than MS-SQL.
  - Browsers are **definitely not** parsing the HTML standard.
  - WAFs are doing much more than a simple RE matching.

Learning to Parse

- **Our Approach:** Use Learning algorithms in order to infer the specifications of parsers and WAFs.
  - Cross check the inferred models for vulnerabilities.
- By using learning we can actively figure out important details of the systems.

Learning Automata

Learning Automata

- **Active Learning** algorithm.
  - Instead of learning from corpus of data, query the program with input of his choice.
- Eventually a model is generated.
- Discovered inconsistencies of the model is used to refine it.

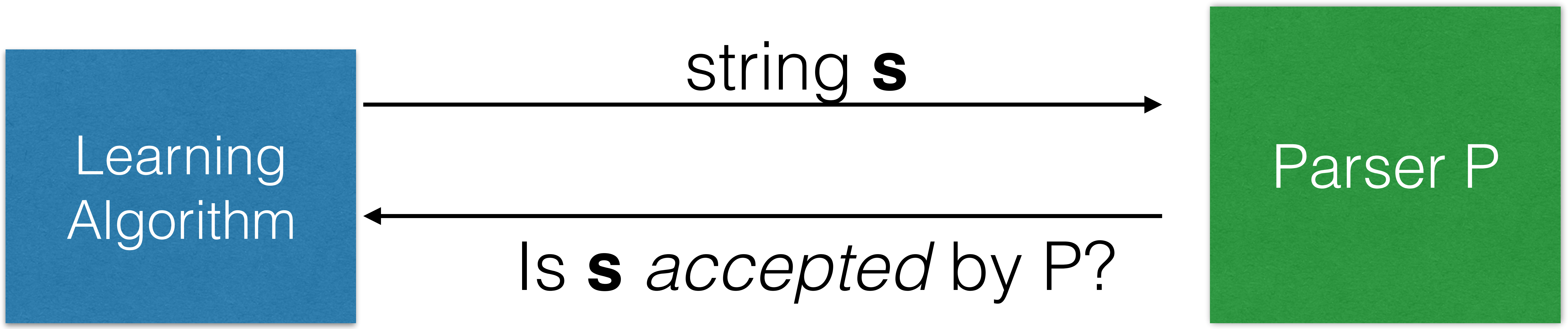
Learning Model

Learning Model

Membership Query

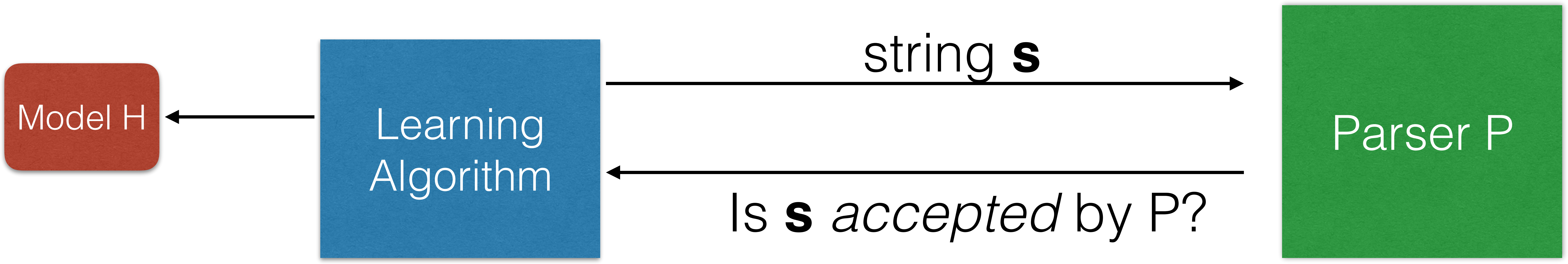
Learning Model

Membership Query

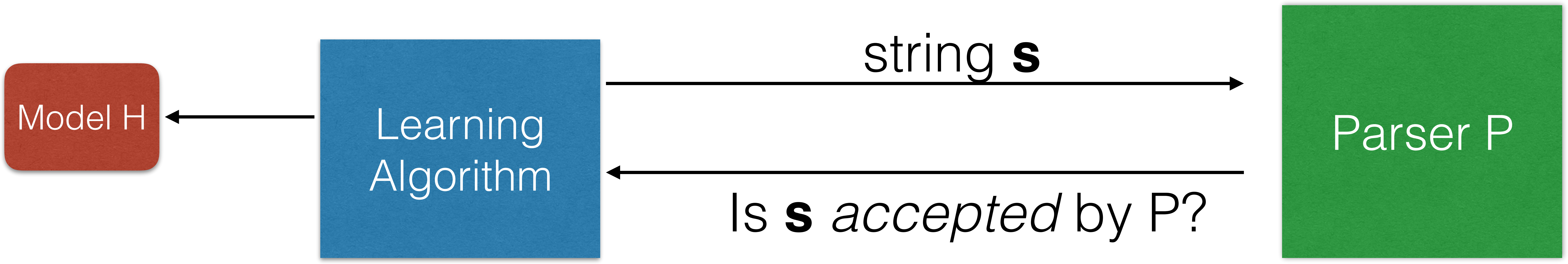


Learning Model

Membership Query

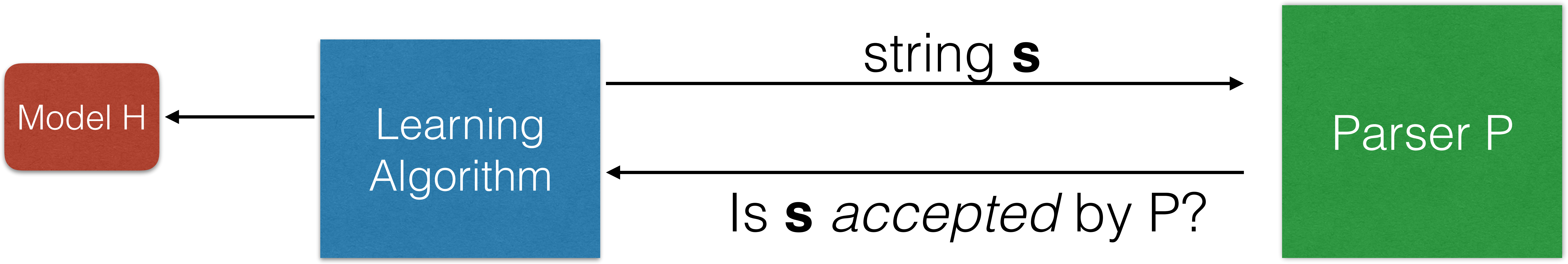


Learning Model



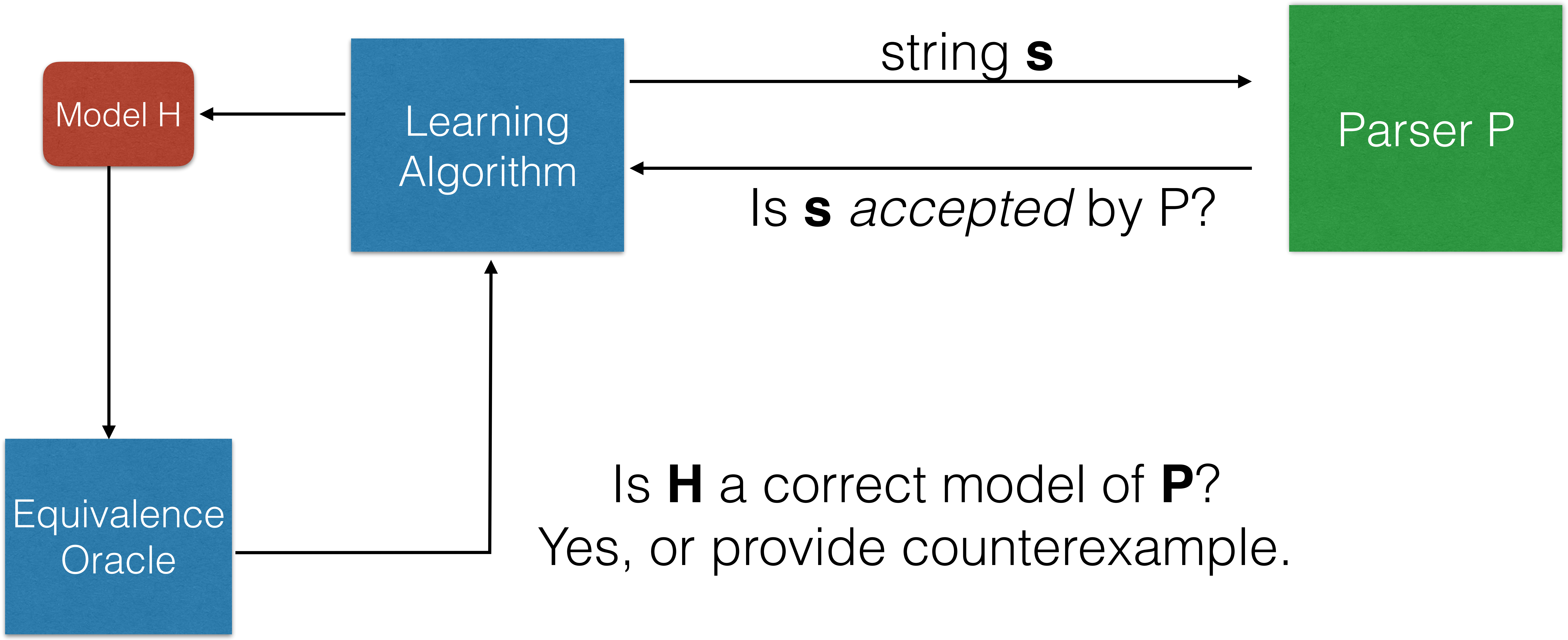
Learning Model

Equivalence Query

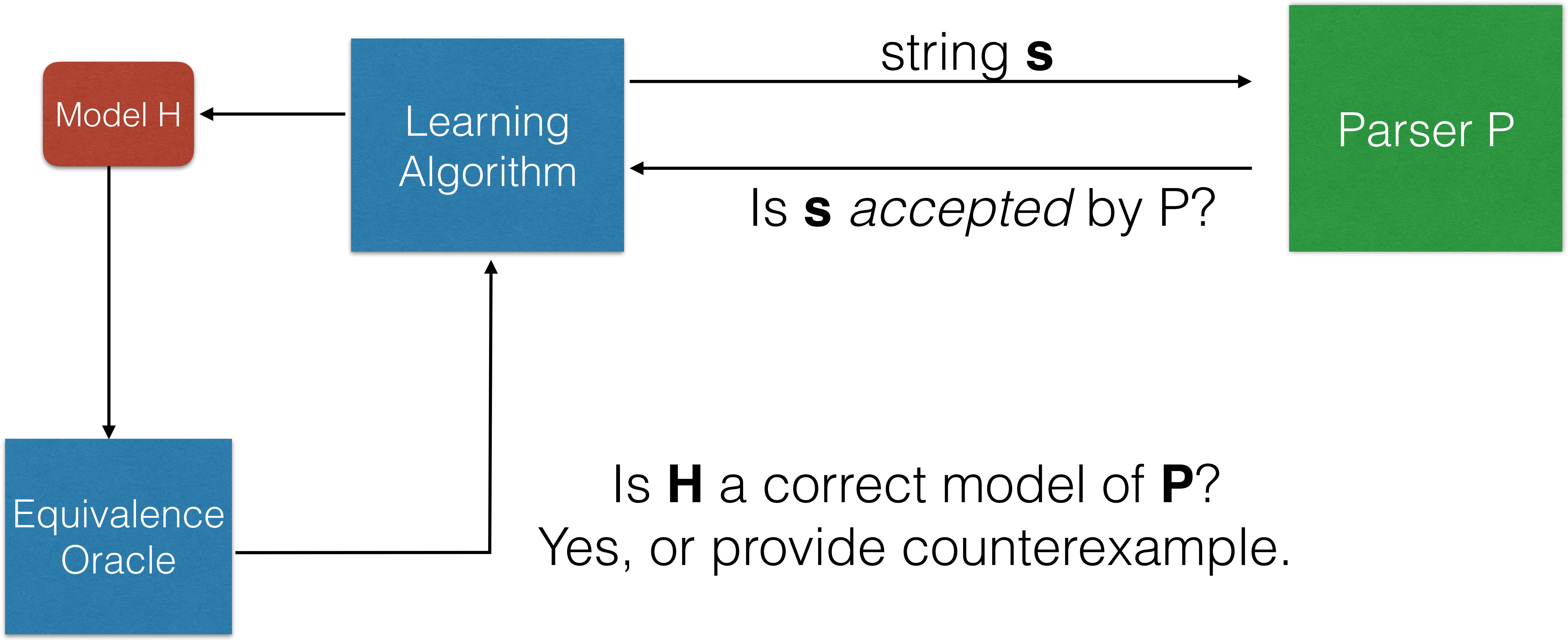


Learning Model

Equivalence Query



Learning Model



Learning DFAs

- Angluin’s algorithm is an active learning algorithm for learning DFAs.

- Learns the target DFA using a table data structure called the **observation table.**

- Let’s use it to learn the regular expression (.*)<a(.*)

  - Aggressive filtering of anchor tags.

Learning DFAs

OT	ε
ε	
ε<	
εa	

Learning DFAs

Empty string

OT	ε
ε	
ε<	
εa	

Learning DFAs

Empty string

OT	ε
ε	
ε<	
εa	

Strings for “testing” states.
(Distinguishing strings)

Learning DFAs

Empty string

Strings accessing different states in the target automaton. (Access Strings)

OT	ε
ε	
ε<	
εa	

Strings for “testing” states. (Distinguishing strings)

Learning DFAs

Empty string

Strings accessing different states in the target automaton. (Access Strings)

Strings for "testing" states. (Distinguishing strings)

OT	ε
ε	
ε<	
εa	

Strings which transition from the above states.

Learning DFAs

Empty string

Strings accessing different states in the target automaton. (Access Strings)

Strings for "testing" states. (Distinguishing strings)

OT	ε
ε	
ε<	
εa	

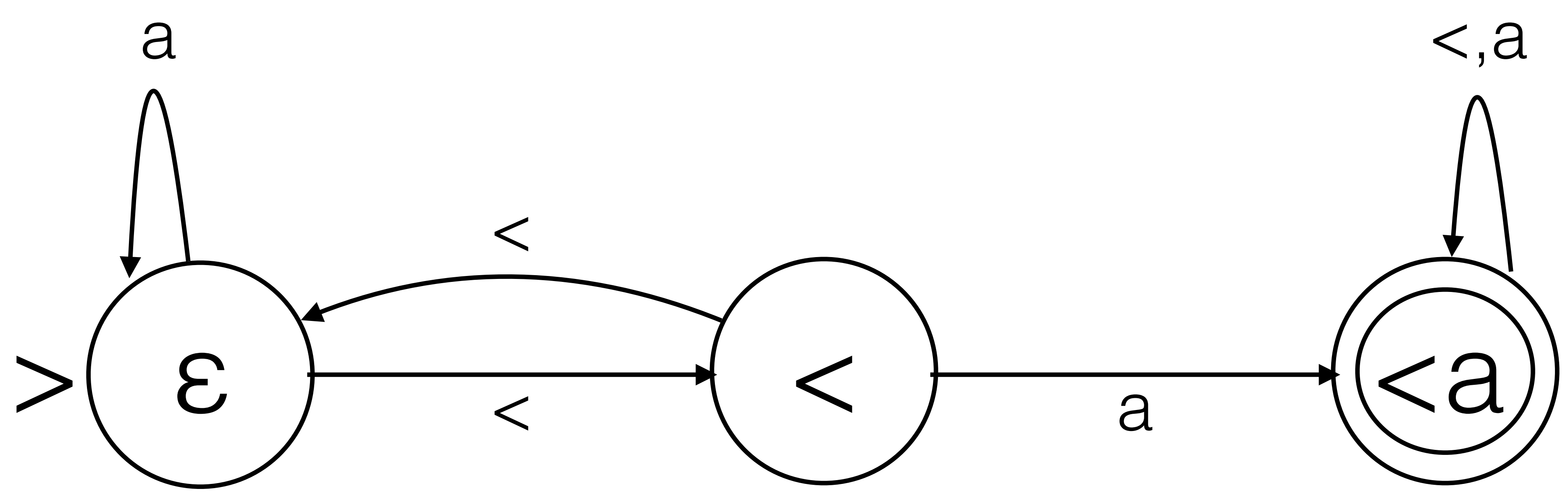
Strings which transition from the above states.

An entry is filled by concatenating the row and column string and filling with the output of the automaton.

Learning DFAs

Model:

Target:

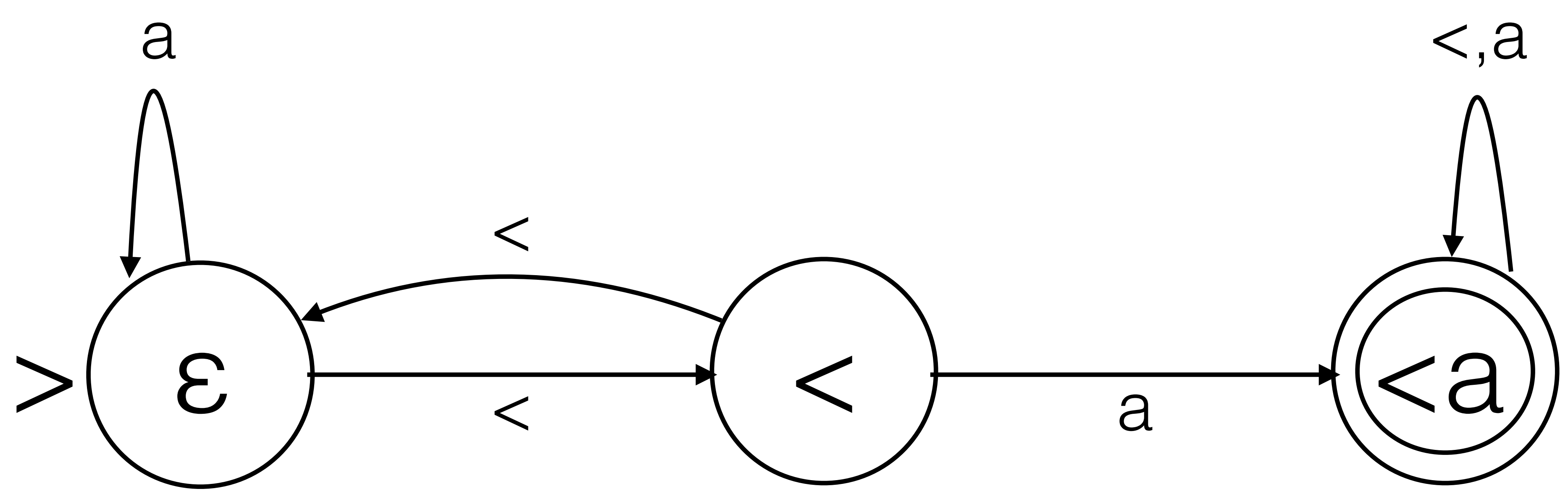


	OT	ε
q_0	ε	
q_0 trans.	ε<	
	εa	

Learning DFAs

Model:

Target:



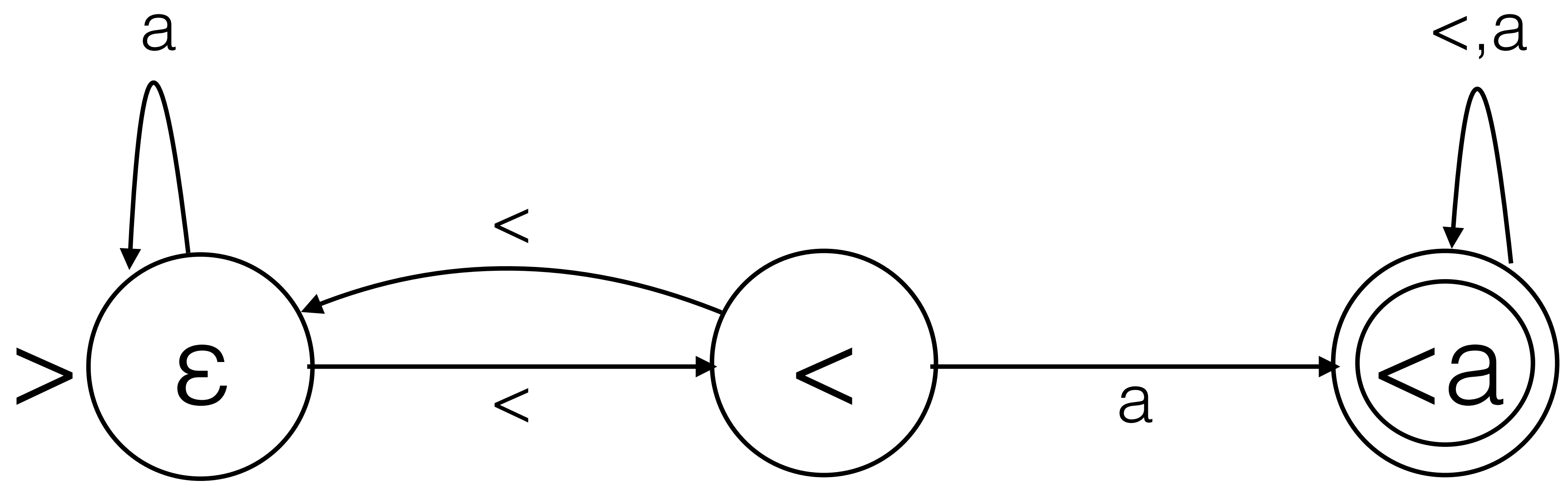
	OT	ε
q_0	ε	0
q_0 trans.	ε<	0
	εa	0

Learning DFAs

Model:

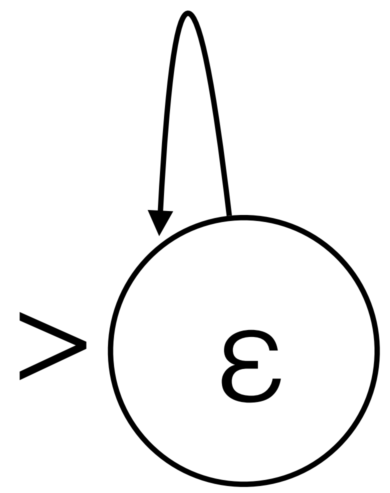
	OT	ε
q_0	ε	0
q_0 trans.	ε<	0
	εa	0

Target:

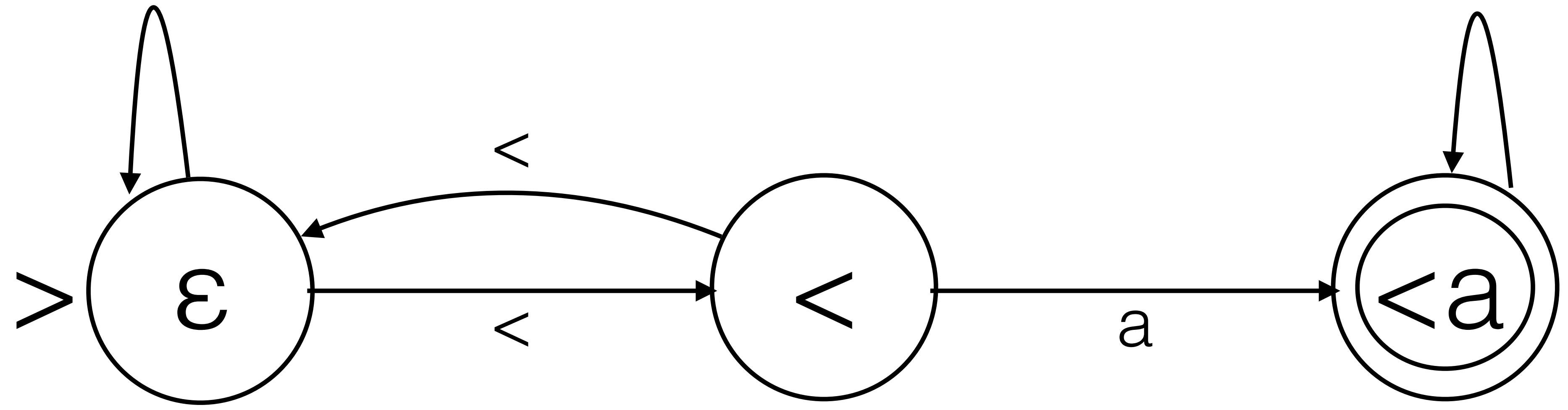


Learning DFAs

Model:

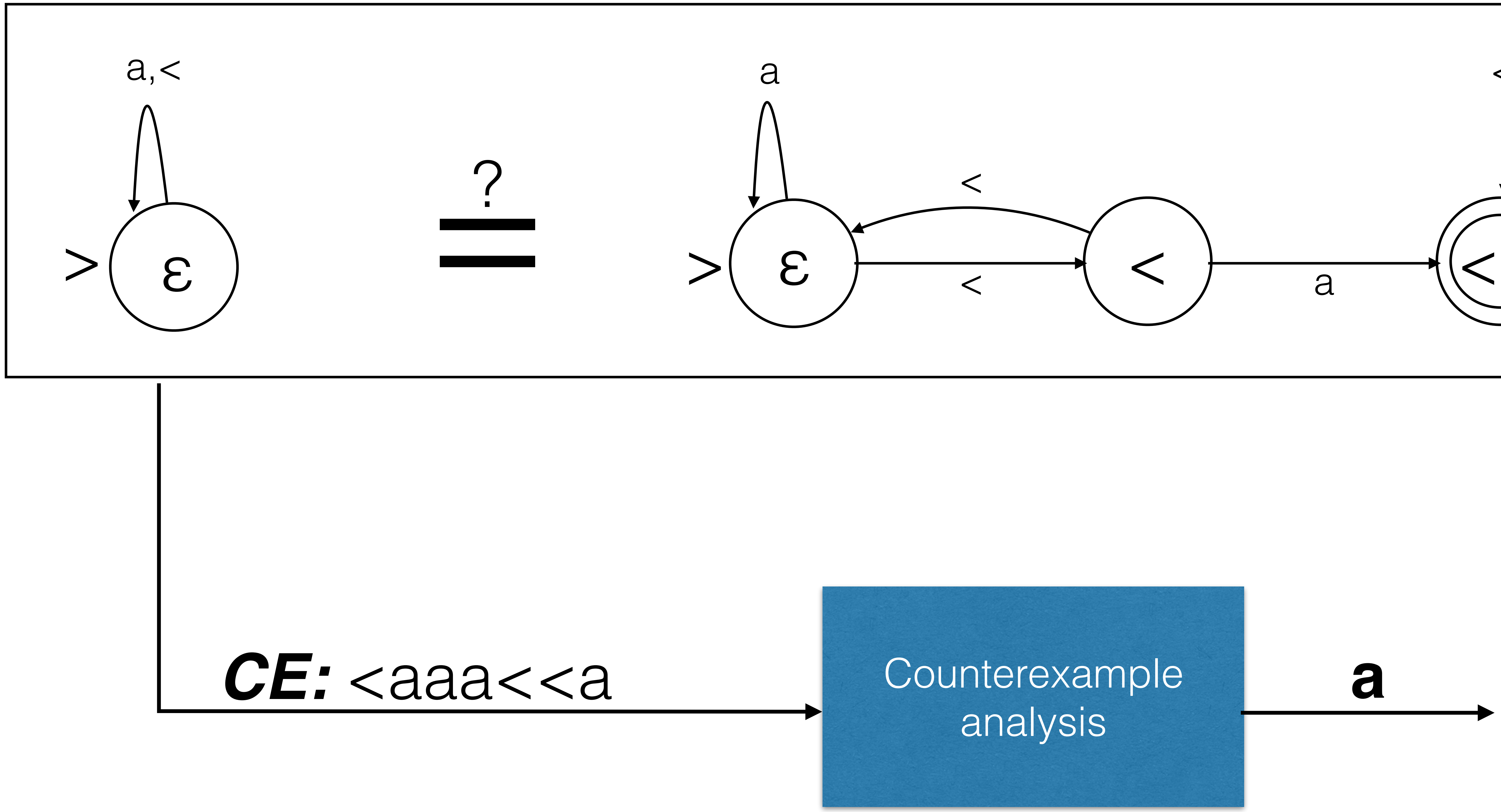


Target:



	OT	ε
q_0	ε	0
q_0 trans.	ε<	0
	εa	0

Equivalence Query

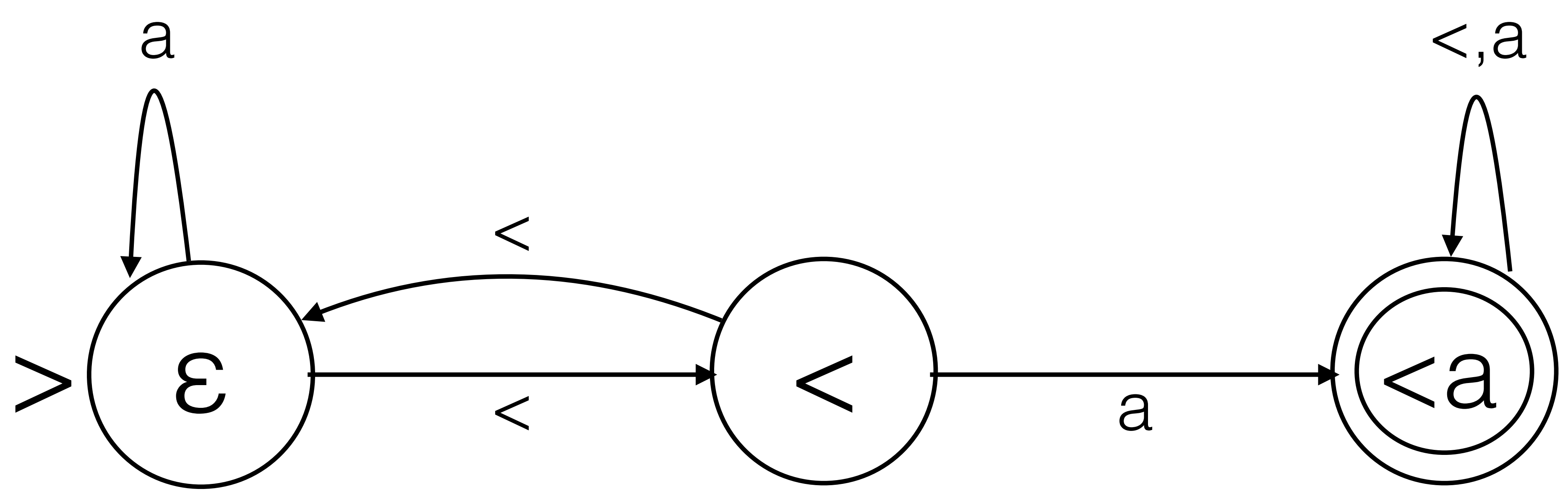


Add a new column with character “a” in the OT.

Learning DFAs

Model:

Target:

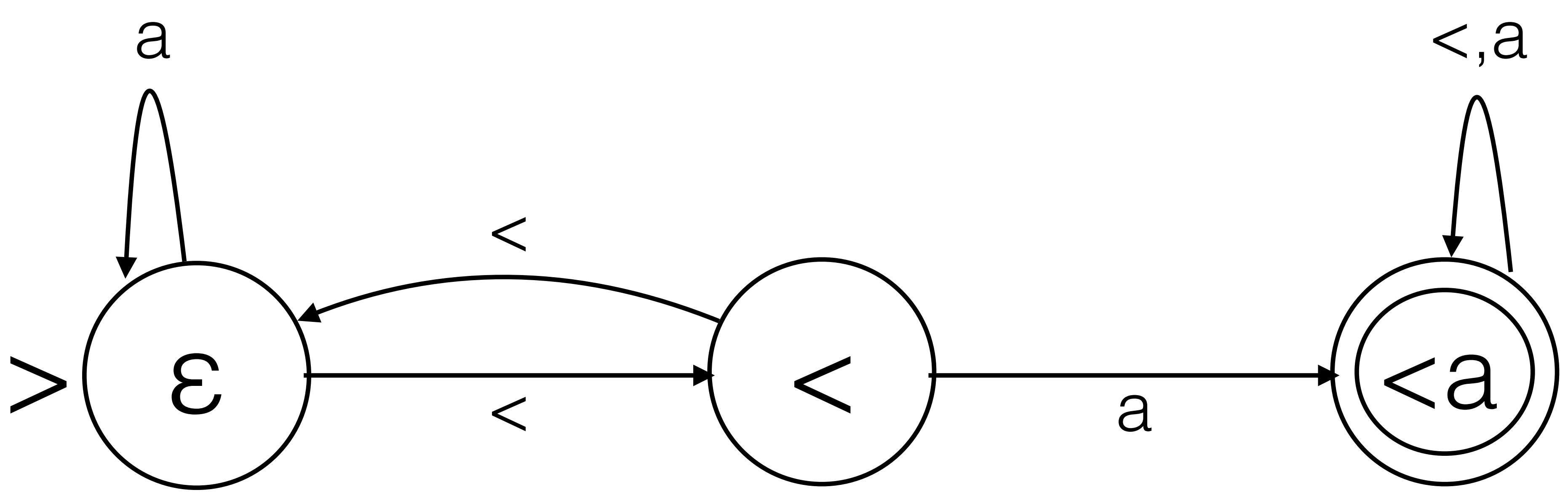


	OT	ε	a
q_0	ε	0	
q_0	ε<	0	
trans.	εa	0	

Learning DFAs

Model:

Target:

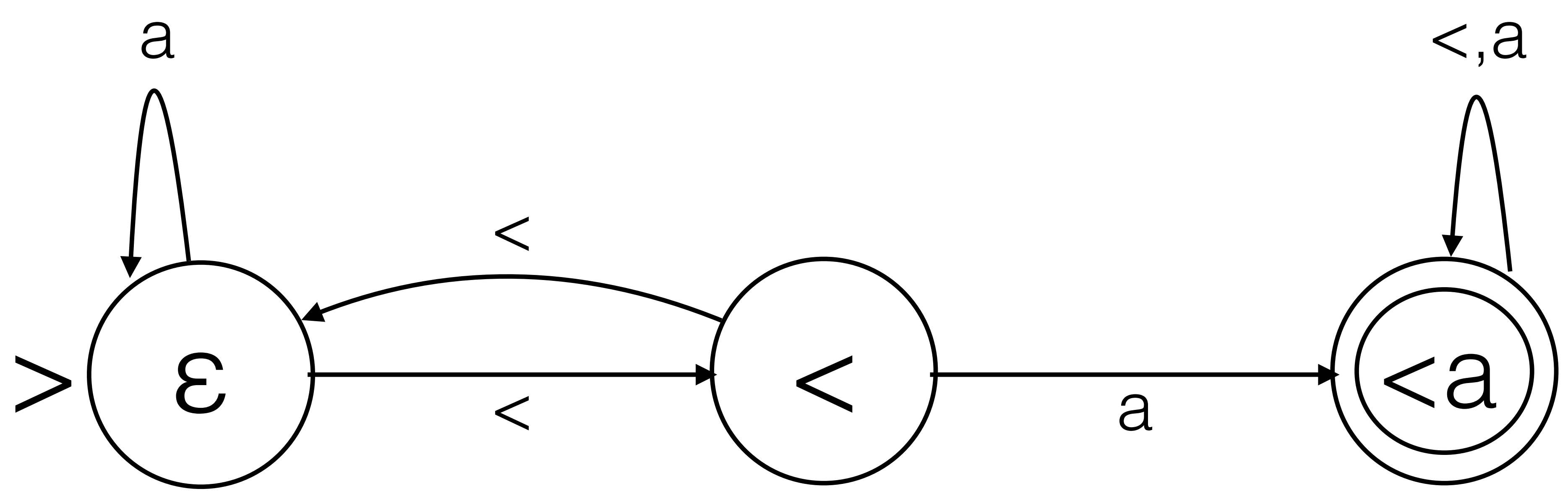


	OT	ε	a
q_0	ε	0	0
q_0 trans.	ε<	0	1
	εa	0	0

Learning DFAs

Model:

Target:



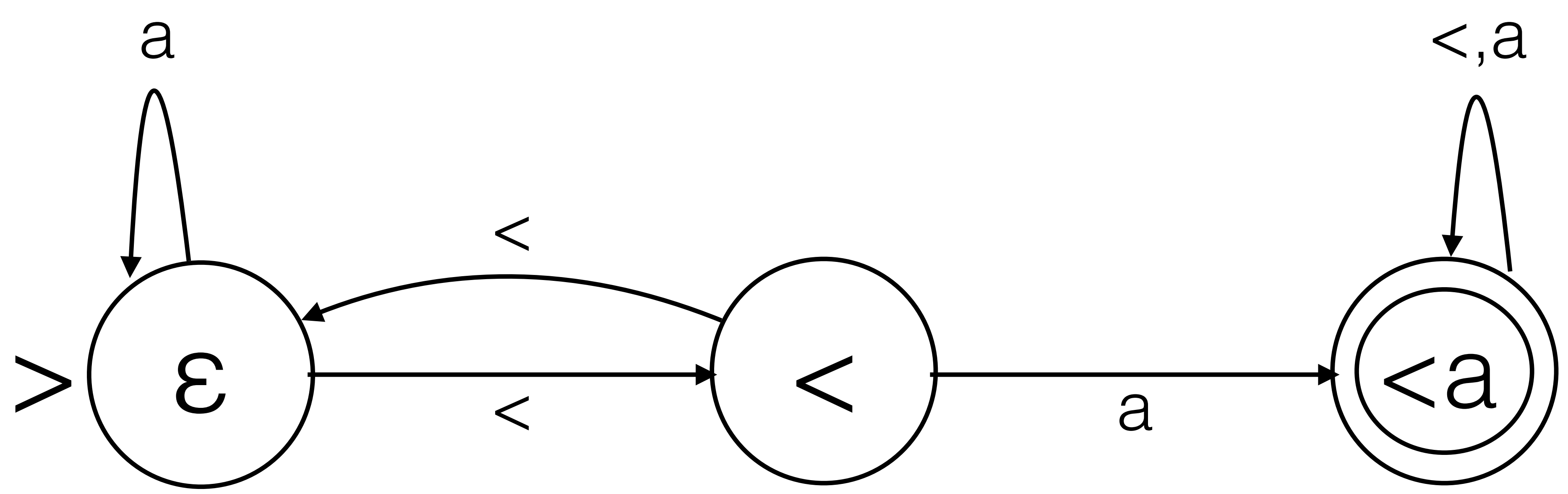
Must be a new state

	OT	ε	a
q_0	ε	0	0
q_0 trans.	ε<	0	1
	εa	0	0

Learning DFAs

Model:

Target:

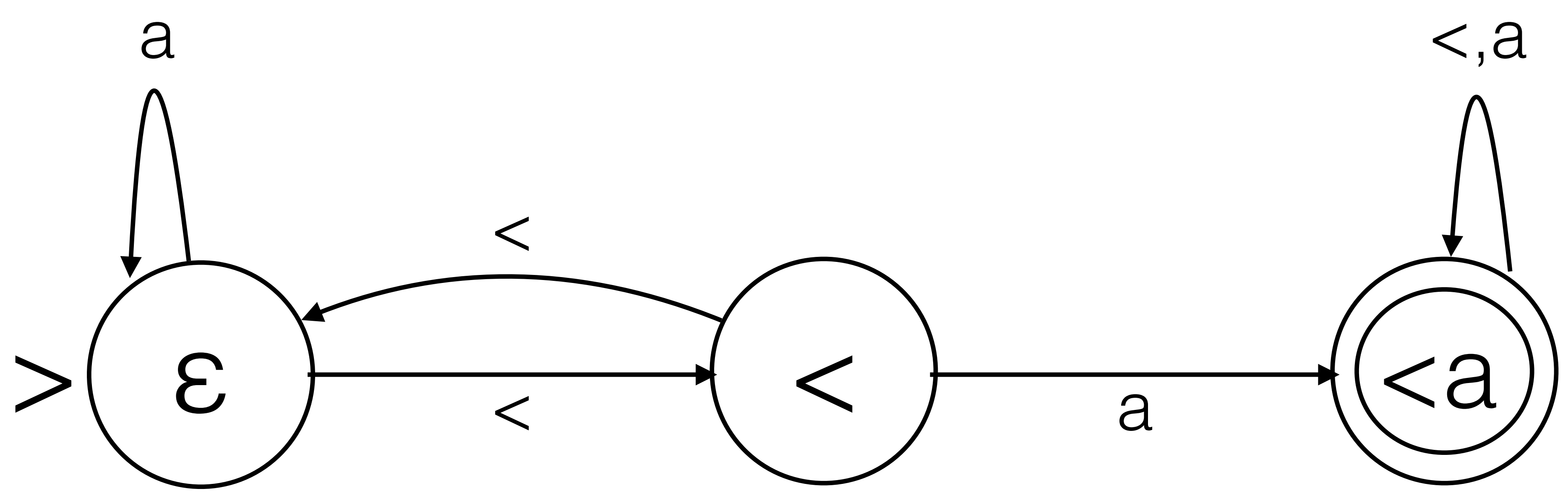


	OT	ε	a
q_0	ε	0	0
q_1	ε<	0	1
q_0 Trans.	εa	0	0
	ε<	0	1
q_1 Trans.	<a		
	<<		

Learning DFAs

Model:

Target:

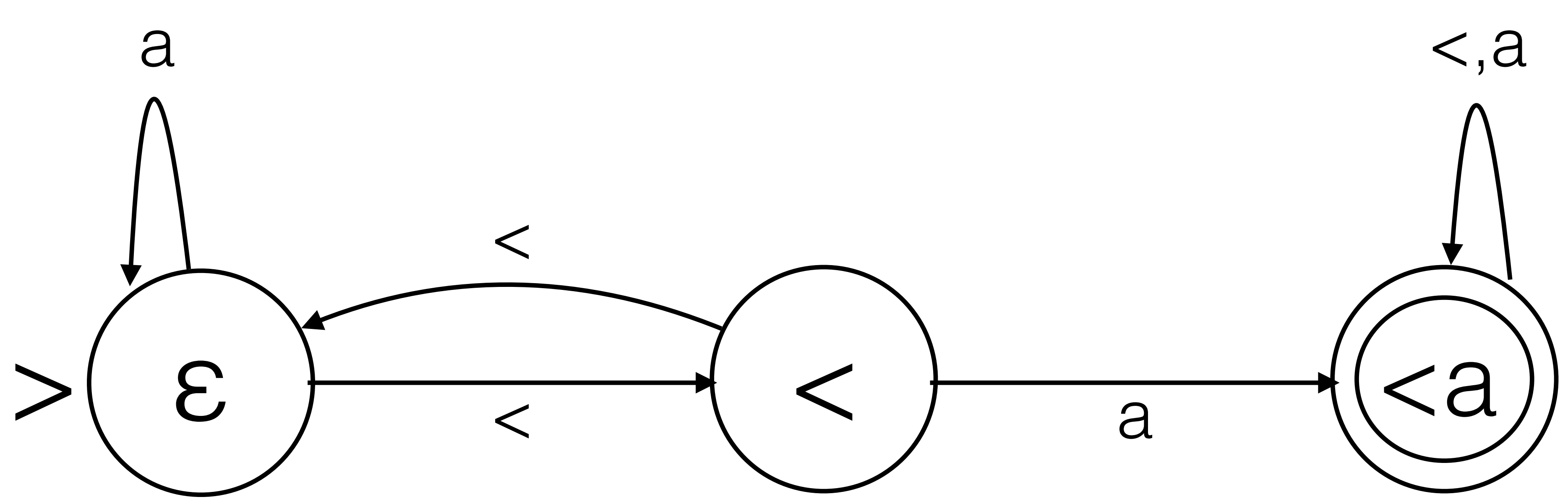


	OT	ε	a
q_0	ε	0	0
q_1	ε<	0	1
q_0 Trans.	εa	0	0
	ε<	0	1
q_1 Trans.	<a	1	1
	<<	0	1

Learning DFAs

Model:

Target:



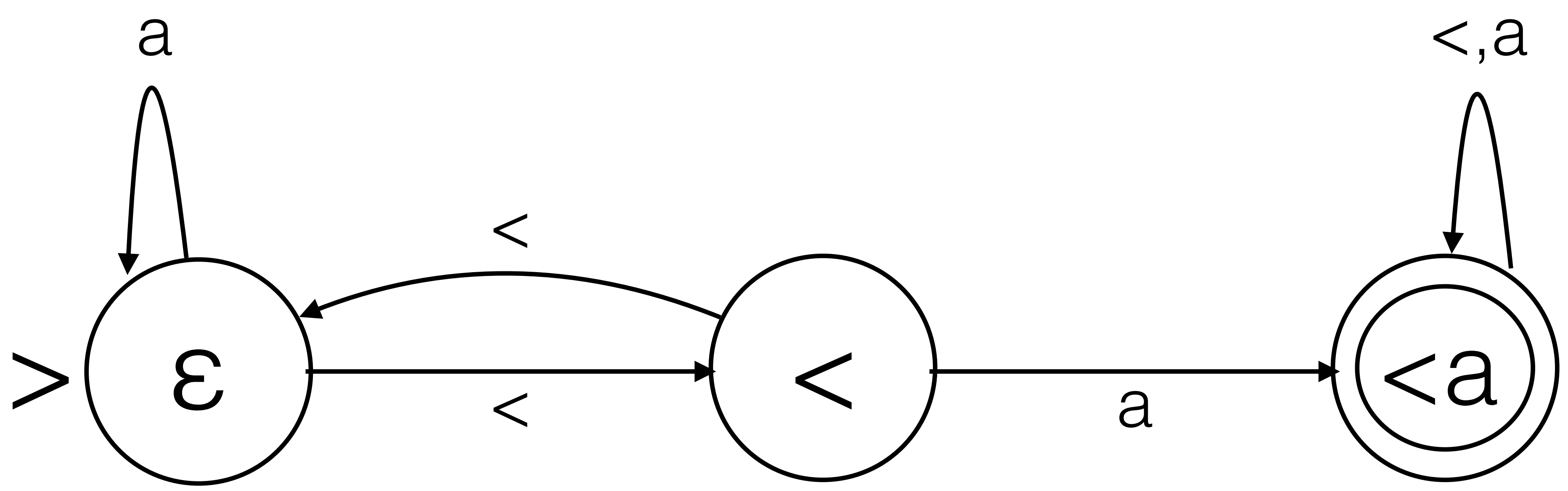
Must be a new state

	OT	ε	a
q_0	ε	0	0
q_1	ε<	0	1
q_0 Trans.	εa	0	0
	ε<	0	1
q_1 Trans.	<a	1	1
	<<	0	1

Learning DFAs

Model:

Target:



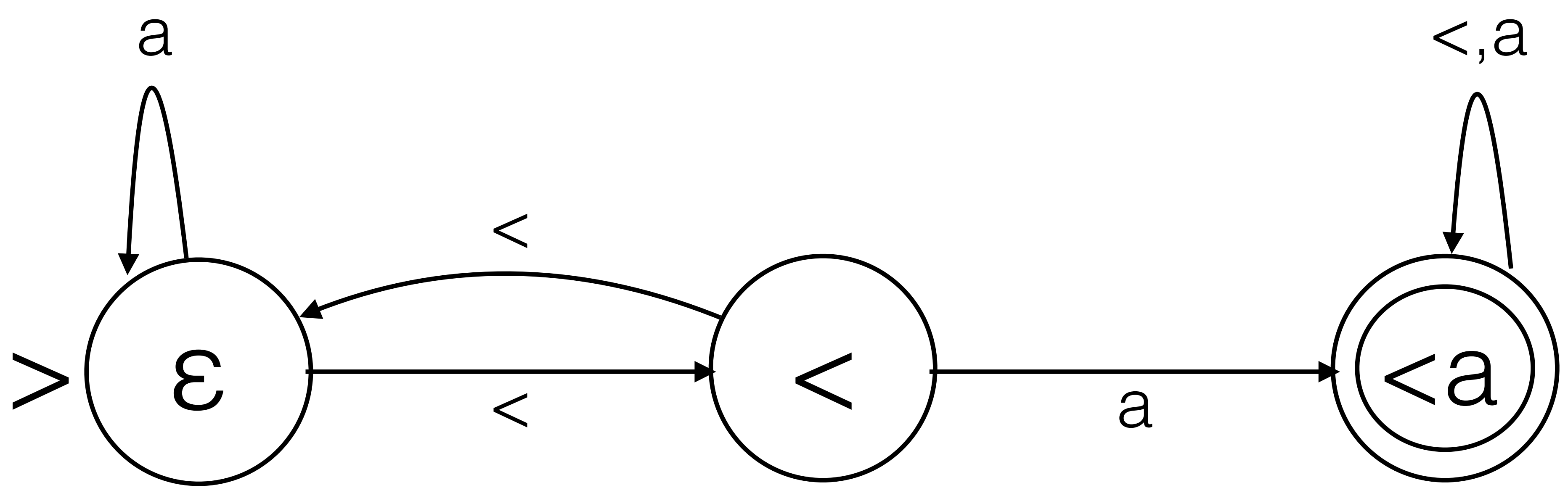
	OT	ε	a
q_0	ε	0	0
q_1	ε<	0	1
q_2	<a	1	1
q_0	εa	0	0
trans.	ε<	0	1
q_1	<a	1	1
trans.	<<	0	1
q_2	<aa		
trans.	<a<		

Learning DFAs

Model:

	OT	ε	a
q_0	ε	0	0
q_1	ε<	0	1
q_2	<a	1	1
q_0	εa	0	0
trans.	ε<	0	1
q_1	<a	1	1
trans.	<<	0	1
q_2	<aa	1	1
trans.	<a<	1	1

Target:



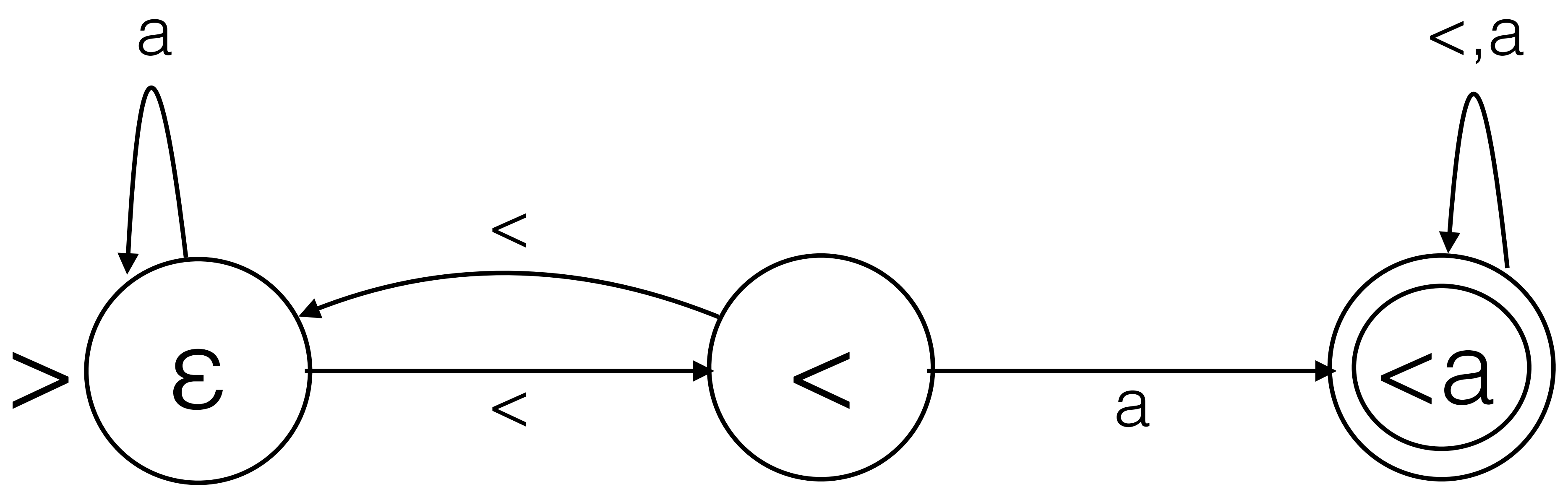
Learning DFAs

Model:

states

	OT	ε	a
q_0	ε	0	0
q_1	ε<	0	1
q_2	<a	1	1
q_0 trans.	εa	0	0
	ε<	0	1
q_1 trans.	<a	1	1
	<<	0	1
q_2 trans.	<aa	1	1
	<a<	1	1

Target:



Learning DFAs

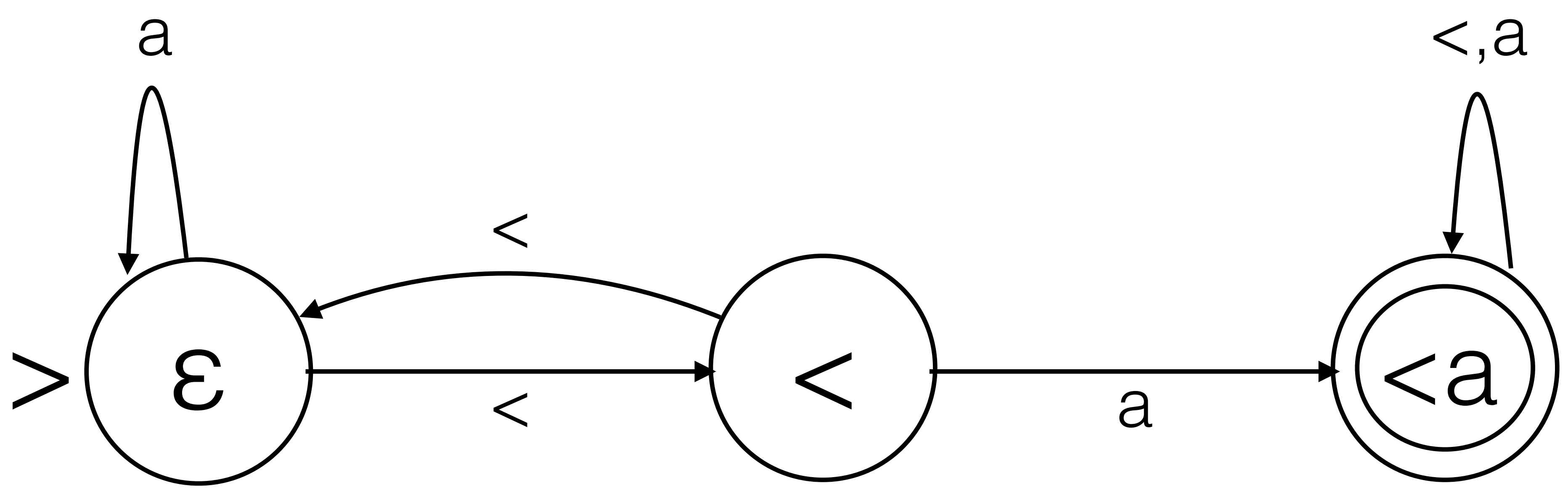
Model:

states

	OT	ε	a
q_0	ε	0	0
q_1	ε<	0	1
q_2	<a	1	1
q_0	εa	0	0
trans.	ε<	0	1
q_1	<a	1	1
trans.	<<	0	1
q_2	<aa	1	1
trans.	<a<	1	1

transitions

Target:



Learning DFAs

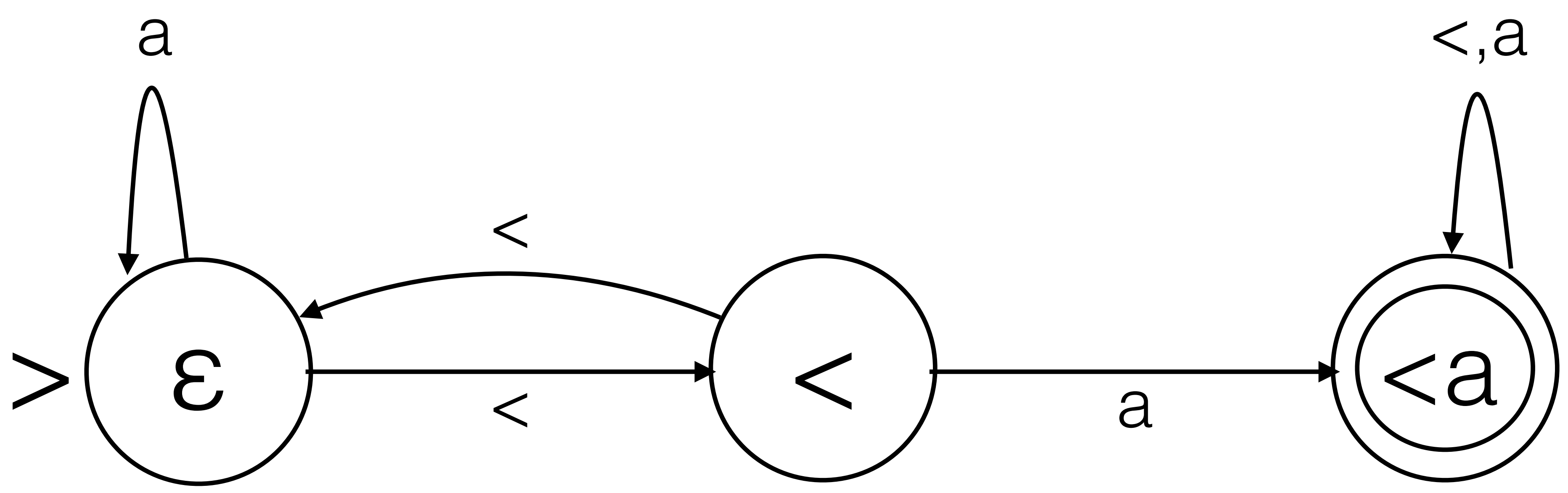
Model:

states

	OT	ε	a
q_0	ε	0	0
q_1	ε<	0	1
q_2	<a	1	1
q_0	εa	0	0
trans.	ε<	0	1
q_1	<a	1	1
trans.	<<	0	1
q_2	<aa	1	1
trans.	<a<	1	1

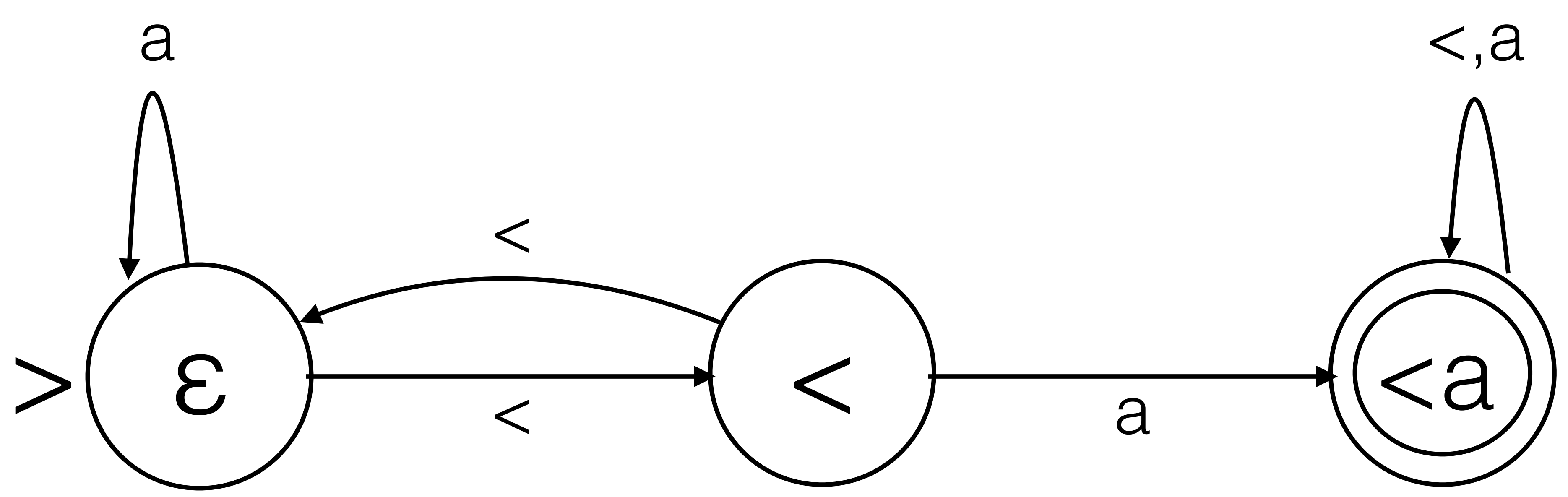
transitions

Target:

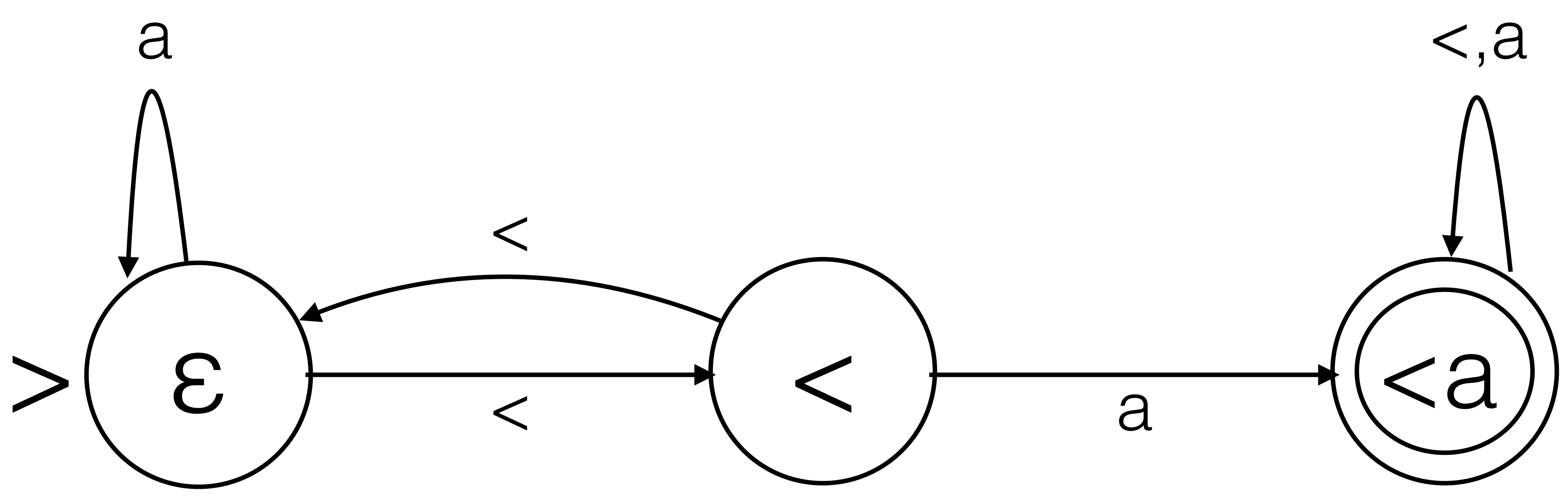


Learning DFAs

Model:



Target:



states

	OT	ε	a
q_0	ε	0	0
q_1	ε<	0	1
q_2	<a	1	1
q_0	εa	0	0
trans.	ε<	0	1
q_1	<a	1	1
trans.	<<	0	1
q_2	<aa	1	1
trans.	<a<	1	1

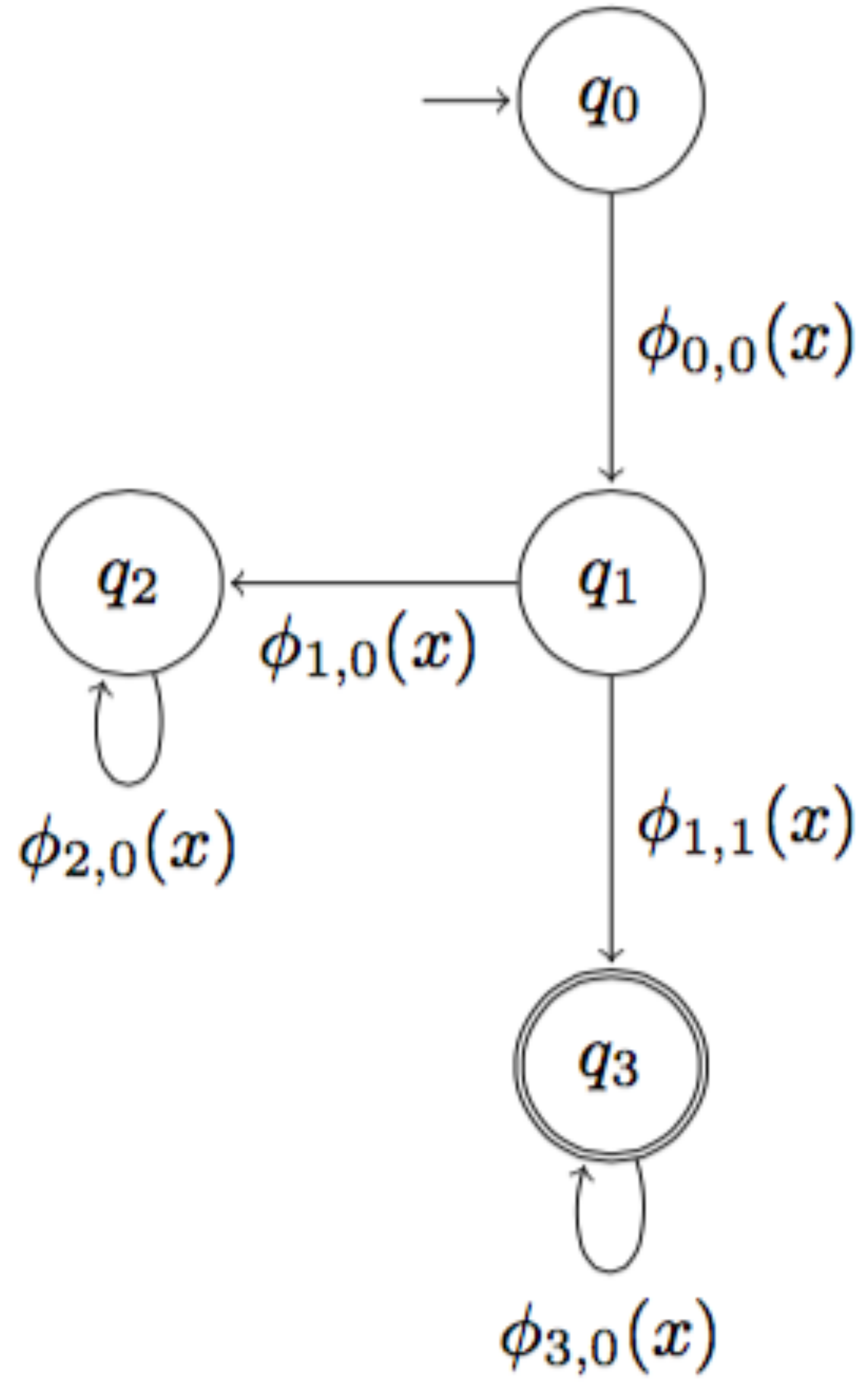
transitions

Learning DFAs

- This algorithm is inefficient for large alphabets/automata.
- For just one PHPIDS Rule (id. 72):
  - ((\={s}*(top|this|window|content|self|frames|_content))|(\/{s}*[gimx]*{s}*[\)\}])|([^]{s}*\={s}*script)|(\.{s}*constructor)|(default{s}+xml{s}+namespace{s}*\=)|(\/{s}*\+[^\+]+{s}*\+{s}*\/))
  - 72 states when represented as a DFA.
  - The OT will have ~650k entries.
- We need a faster algorithm in order to check real systems!

Symbolic Finite Automata

- ✓ Efficient modeling of large alphabets.
- ✓ We designed a novel, efficient learning algorithm.
- ✓ Details in the whitepaper!



Bootstrapping Automata Learning

- Similar concept with seed inputs in fuzzers.
    - Provide sample inputs and learning algorithm will discover additional states in the parser.
- Utilize previously inferred models, specifications, etc.
- Seed inputs are guiding the learning algorithm.
- Details in the white paper!

Grammar Oriented Filter Auditing (GOFA)

Grammar Oriented Filter Auditing

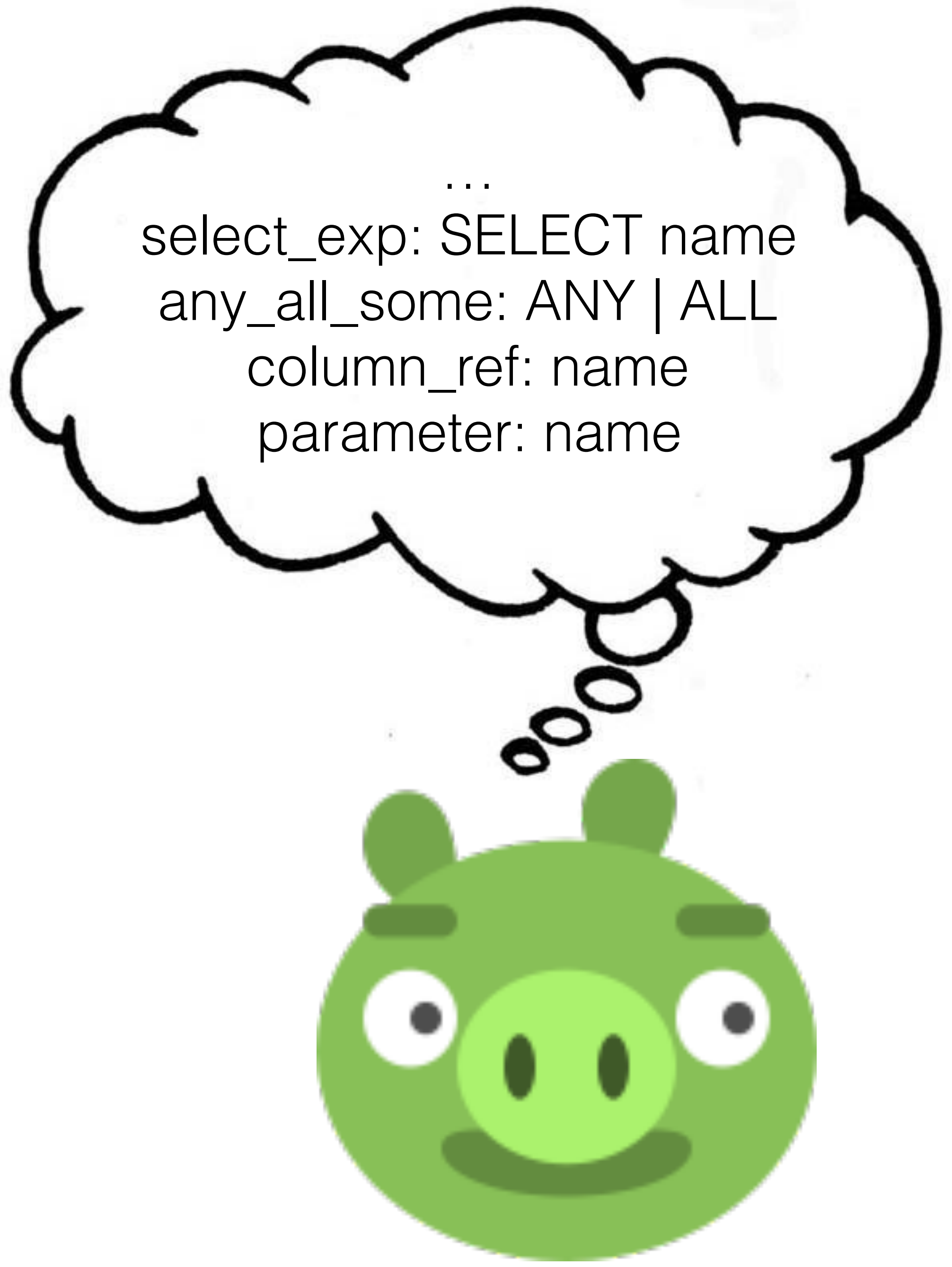
- Assume that we are given a grammar with attacks.
- How do we utilize it with the learning algorithm?

Main idea:

Use the grammar to drive the learning procedure.

Grammar Oriented Filter Auditing

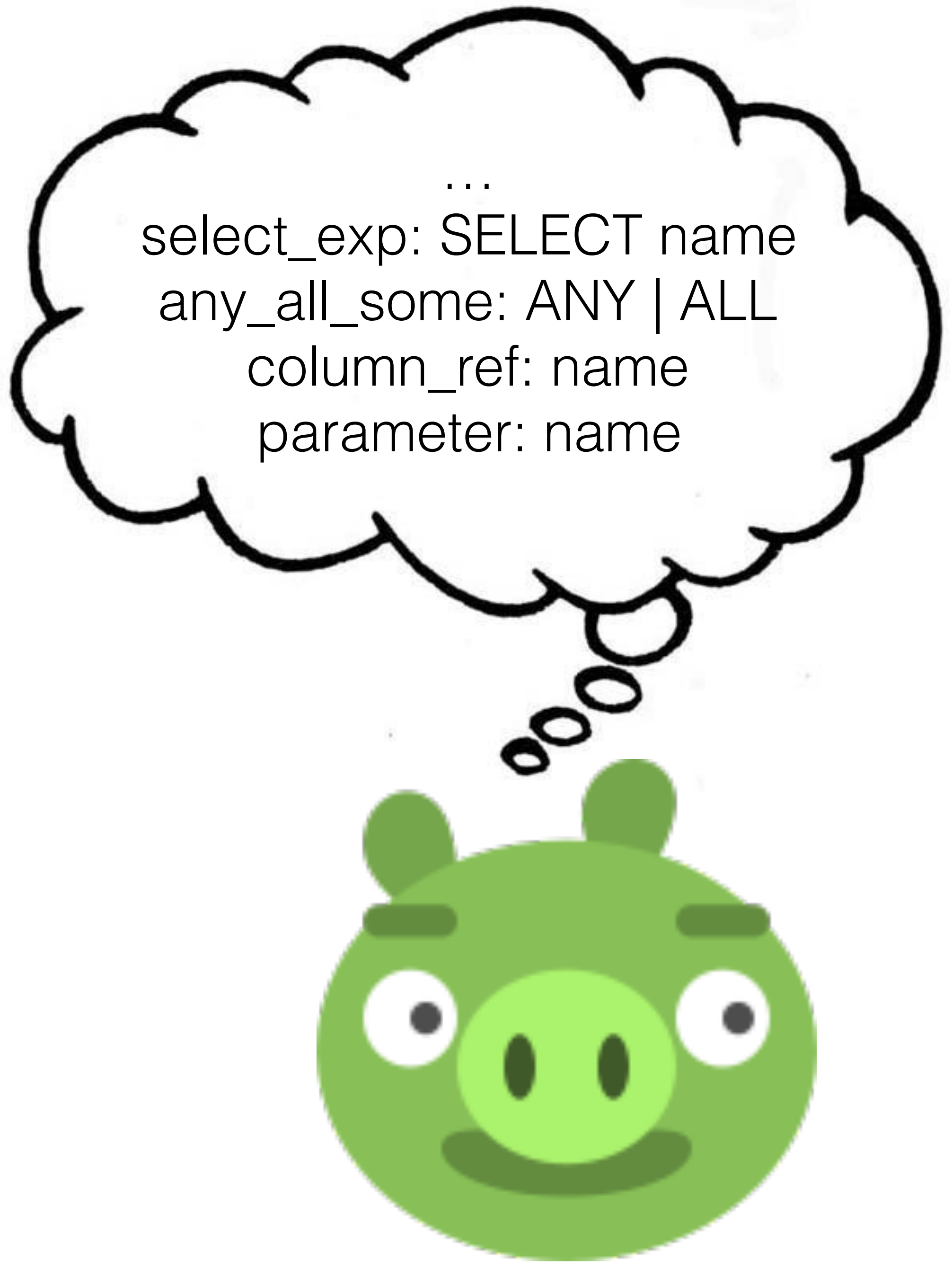
Context Free Grammar ***G***



Learning Algorithm

Grammar Oriented Filter Auditing

Context Free Grammar ***G***

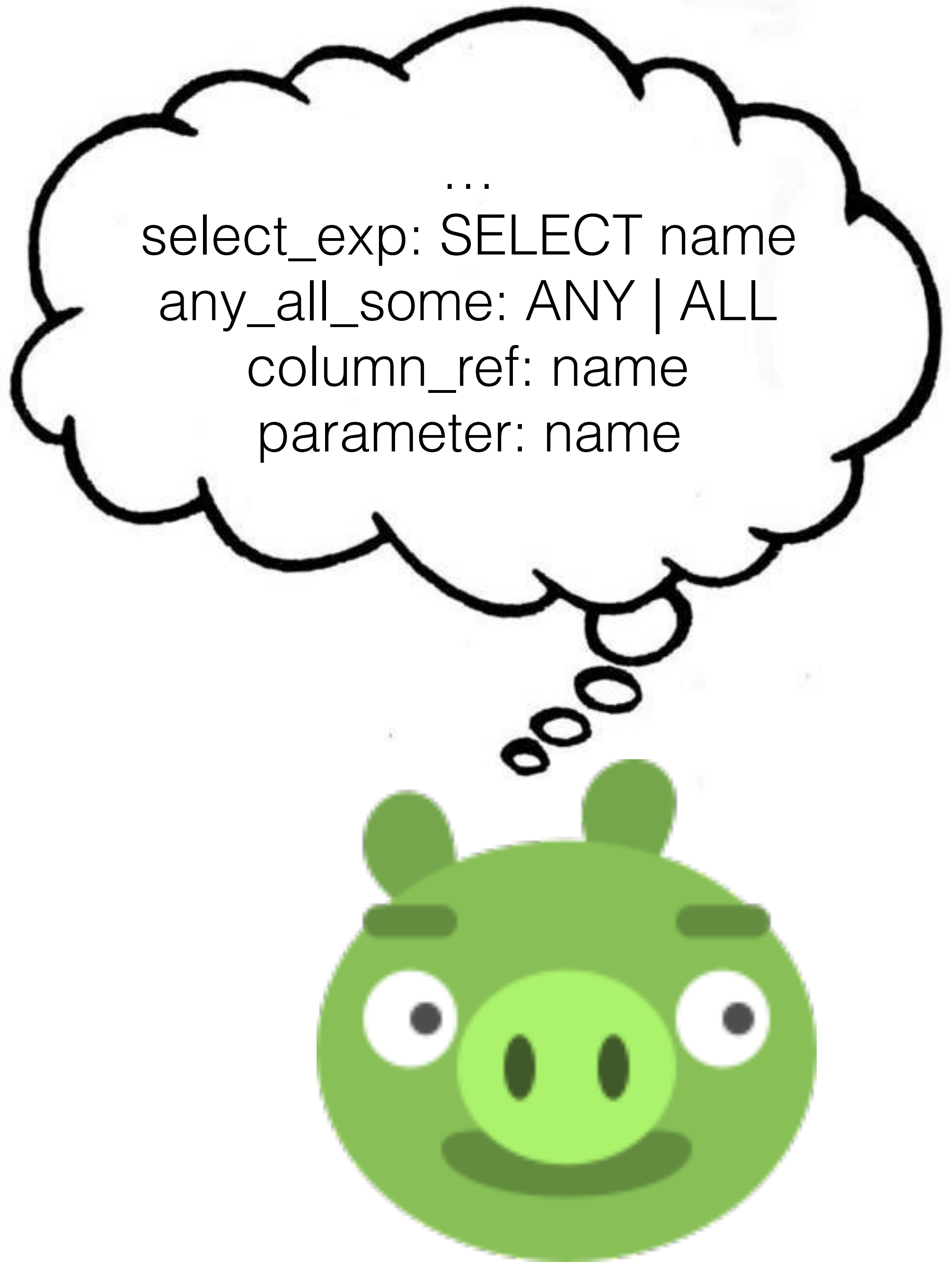


Step 1:
Learn a model of the WAF.

Learning Algorithm

Grammar Oriented Filter Auditing

Context Free Grammar ***G***

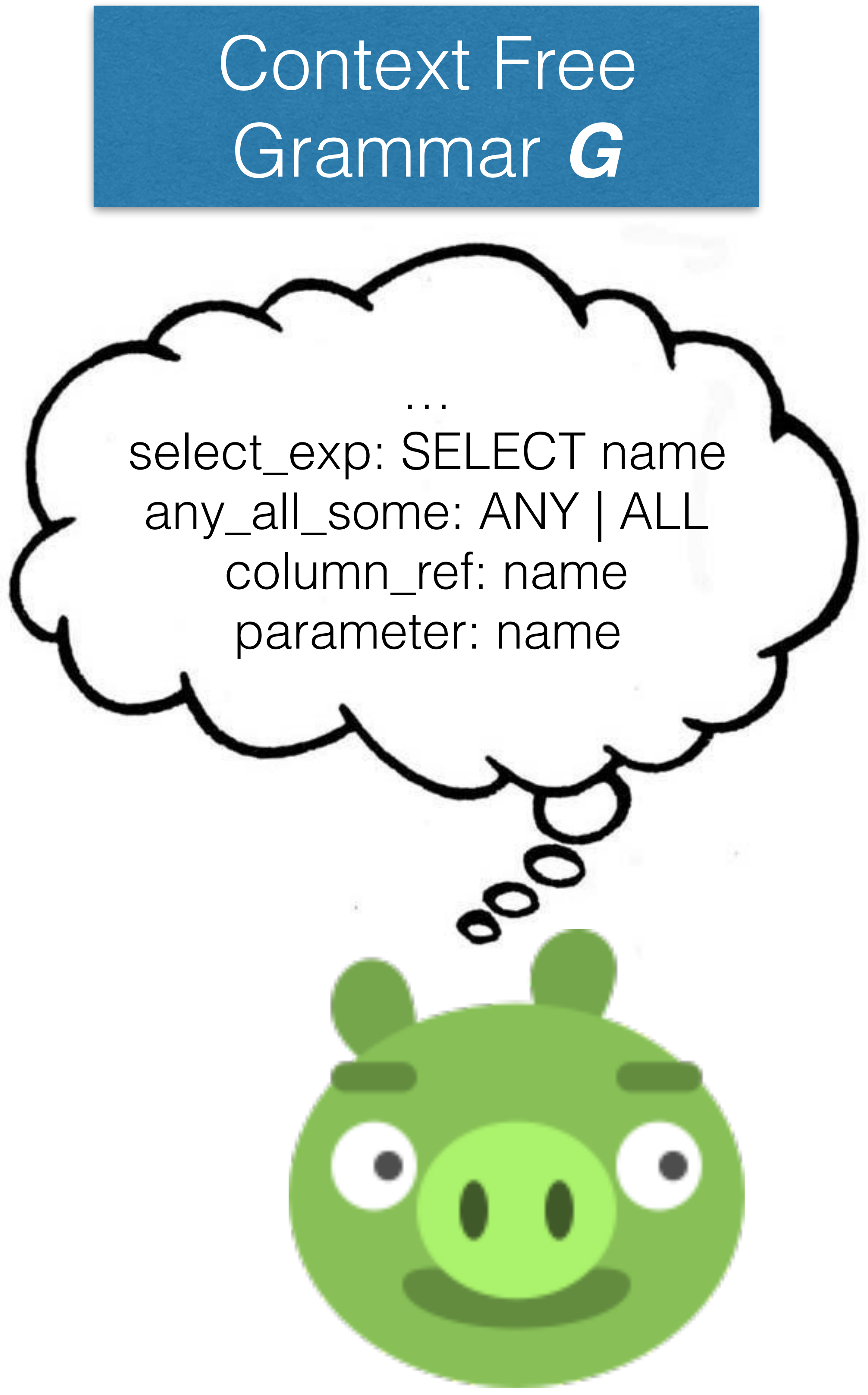


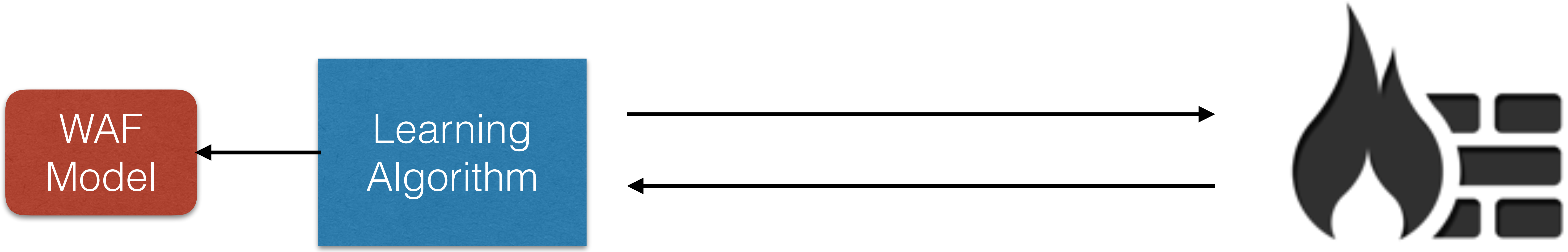
Step 1:
Learn a model of the WAF.

Learning Algorithm

Grammar Oriented Filter Auditing

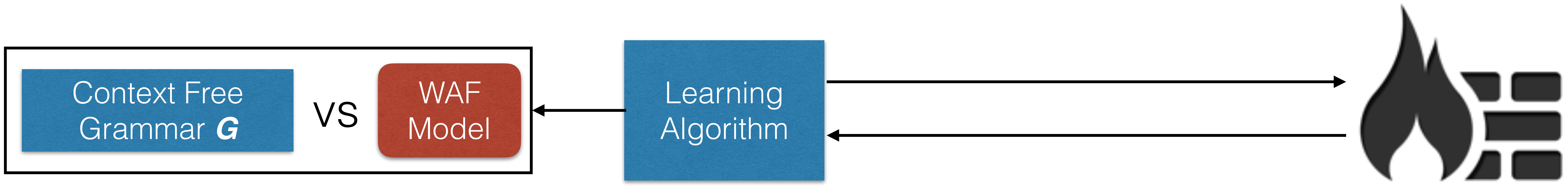
Step 1:
Learn a model of the WAF.



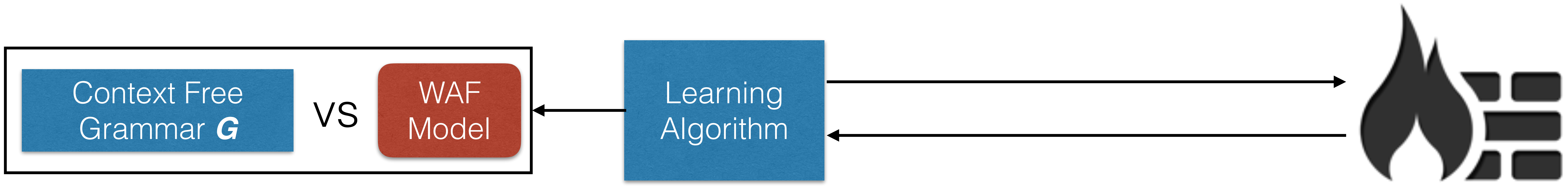


Grammar Oriented Filter Auditing

Step 2:
Find a vulnerability in the model using the grammar.

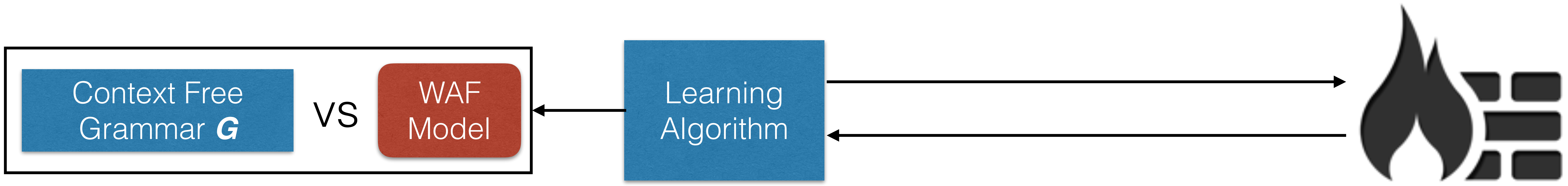


Grammar Oriented Filter Auditing



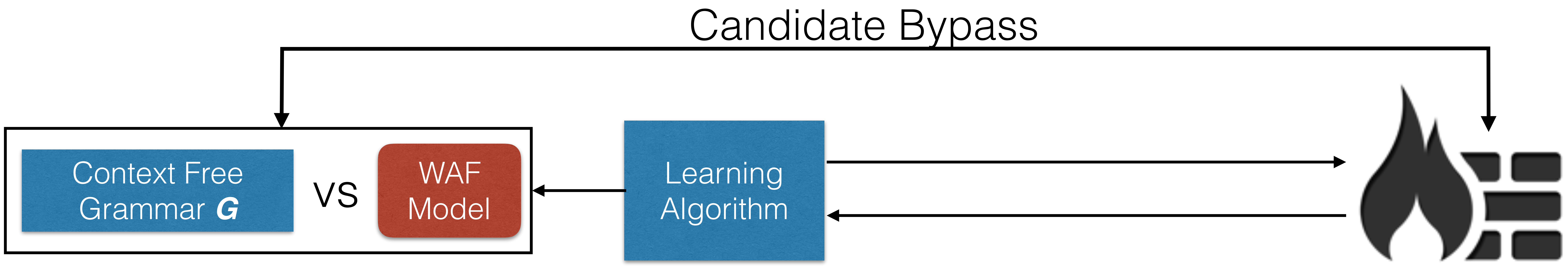
Grammar Oriented Filter Auditing

Step 3:
Verify WAF vulnerability.

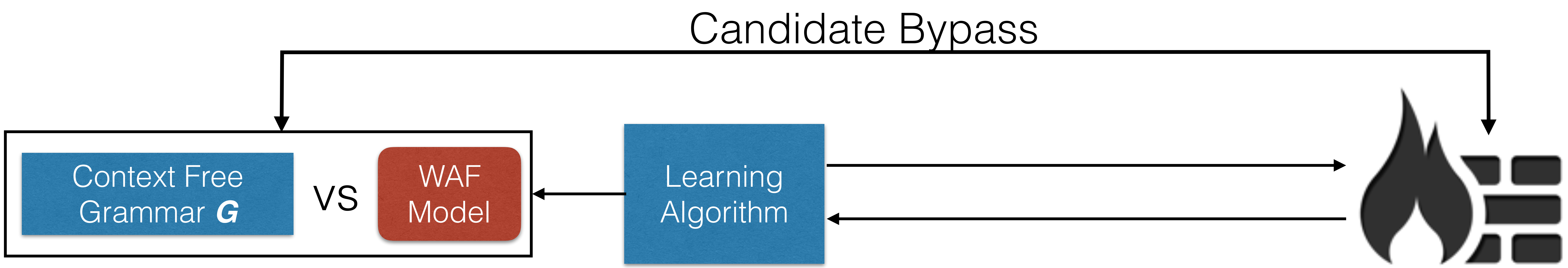


Grammar Oriented Filter Auditing

Step 3:
Verify WAF vulnerability.



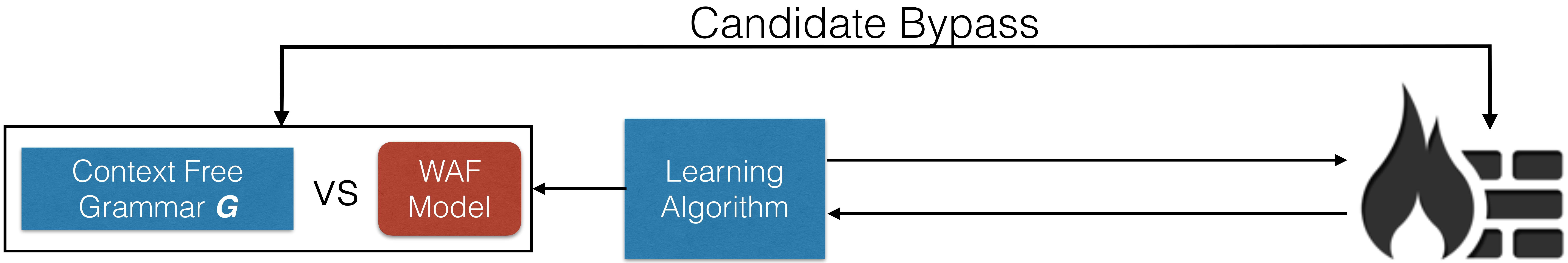
Grammar Oriented Filter Auditing



Grammar Oriented Filter Auditing

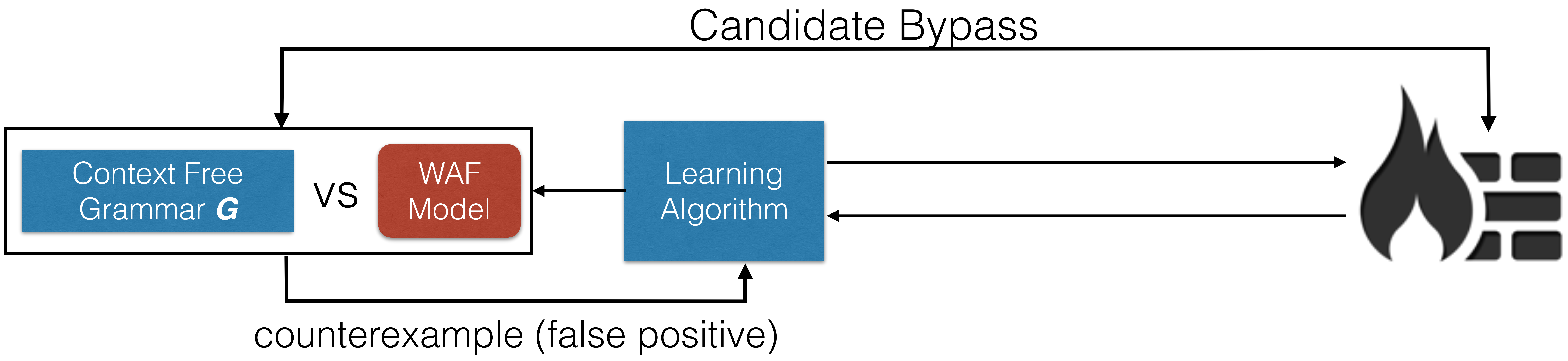
Step 4:

 or refine model and repeat.



Grammar Oriented Filter Auditing

Step 4:
or refine model and repeat.



Vulnerabilities

GOFA SQL Injections

- Grammar for extending search conditions:

  *select * from users where user = admin and email =* ***$_GET[c]***

GOFA SQL Injections

- Grammar for extending search conditions:

*select * from users where user = admin and email =* ***$_GET[c]***

```
S: A main
main: search_condition
search_condition: OR predicate | AND predicate
predicate: comparison_predicate | between_predicate | like_predicate | test_for_null | in_predicate
| all_or_any_predicate | existence_test
comparison_predicate: scalar_exp comparison scalar_exp | scalar_exp COMPARISON subquery
between_predicate: scalar_exp BETWEEN scalar_exp AND scalar_exp
like_predicate: scalar_exp LIKE atom
test_for_null: column_ref IS NULL
in_predicate: scalar_exp IN ( subquery ) | scalar_exp IN ( atom )
all_or_any_predicate: scalar_exp comparison any_all_some subquery
existence_test: EXISTS subquery
scalar_exp:  scalar_exp op scalar_exp | atom | column_ref  | ( scalar_exp )
atom: parameter | intnum
subquery: select_exp
select_exp: SELECT name
any_all_some: ANY | ALL | SOME
column_ref: name
parameter: name
intnum: 1
op: + | - | * | /
comparison: = | < | >
name: A
```

GOFA SQL Injections

- Authentication bypass using the vector: **or exists (select 1)**

Example:

select * from users where username = **$_GET['u']** and password = **$_GET['p']**;

select * from users where username = **admin** and password = **a or exists (select 1)**

Affected: ModSecurity Latest CRS, PHPIDS, WebCastellum, Expose

GOFA SQL Injections

- Authentication bypass using the vector: **1 or a = 1**

  1 or a like 1

Example:

select * from users where username = **$_GET['u']** and password = **$_GET['p']**;

select * from users where username = **admin** and password = **1 or isAdmin like 1**

Affected: ModSecurity Latest CRS, PHPIDS (only for statement with 'like'), WebCastellum, Expose

GOFA SQL Injections

- Columns/variables fingerprinting using the vectors: **and exists (select a)**

  a or a > any select a

Example:

select * from users where username = admin and id = **$_GET['u']**;

select * from users where username = admin and id = 1 **and exists (select email)**

Affected: ModSecurity Latest CRS, PHPIDS, WebCastellum, Expose

GOFA SQL Injections

- Grammar for extending select queries:

  *select * from users where user =* ***$_GET[c]***

GOFA SQL Injections

- Grammar for extending select queries:

*select * from users where user = **$_GET[c]***

```
S: A main
main: query_exp
query_exp: groupby_exp | order_exp | limit_exp | procedure_exp | into_exp | for_exp |
lock_exp | ; select_exp | union_exp | join_exp
groupby_exp: GROUP BY column_ref ascdesc_exp
order_exp: ORDER BY column_ref ascdesc_exp
limit_exp: LIMIT intnum
into_exp: INTO output_exp intnum
procedure_exp: PROCEDURE name ( literal )
literal: string | intnum
select_exp: SELECT name
union_exp: UNION select_exp
ascdesc_exp: ASC | DESC
column_ref: name
join_exp: JOIN name ON name
for_exp: FOR UPDATE
lock_exp: LOCK IN SHARE MODE
output_exp: OUTFILE | DUMPFILE
string: name
intnum: 1
name: A
```

GOFA SQL Injections

- Data retrieval bypass using the vector: **1 right join a on a = a**

Example:

select * from articles left join authors on author.id=**$_GET['id']**

select * from articles left join authors on author.id= **1 right join users on author.id = users.id**

Affected: ModSecurity Latest CRS, WebCastellum

GOFA SQL Injections

- Columns/variables fingerprinting using the vectors: **a group by a asc**

Example:

select * from users where username = **$_GET['u']**;

select * from users where username = **admin group by email asc**

Affected: ModSecurity Latest CRS, PHPIDS, WebCastellum, Expose

GOFA SQL Injections

- Columns/variables fingerprinting using the vectors: **procedure a (a)**

Example:

select * from users where username = **$_GET['u']**;

select * from users where username = **admin procedure analyze()**

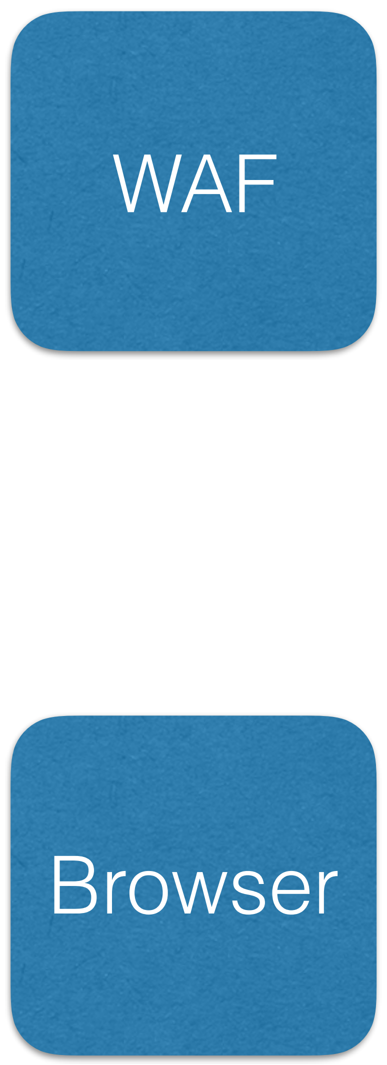
Affected: libInjection

SFADiff: Learning Attack Vectors

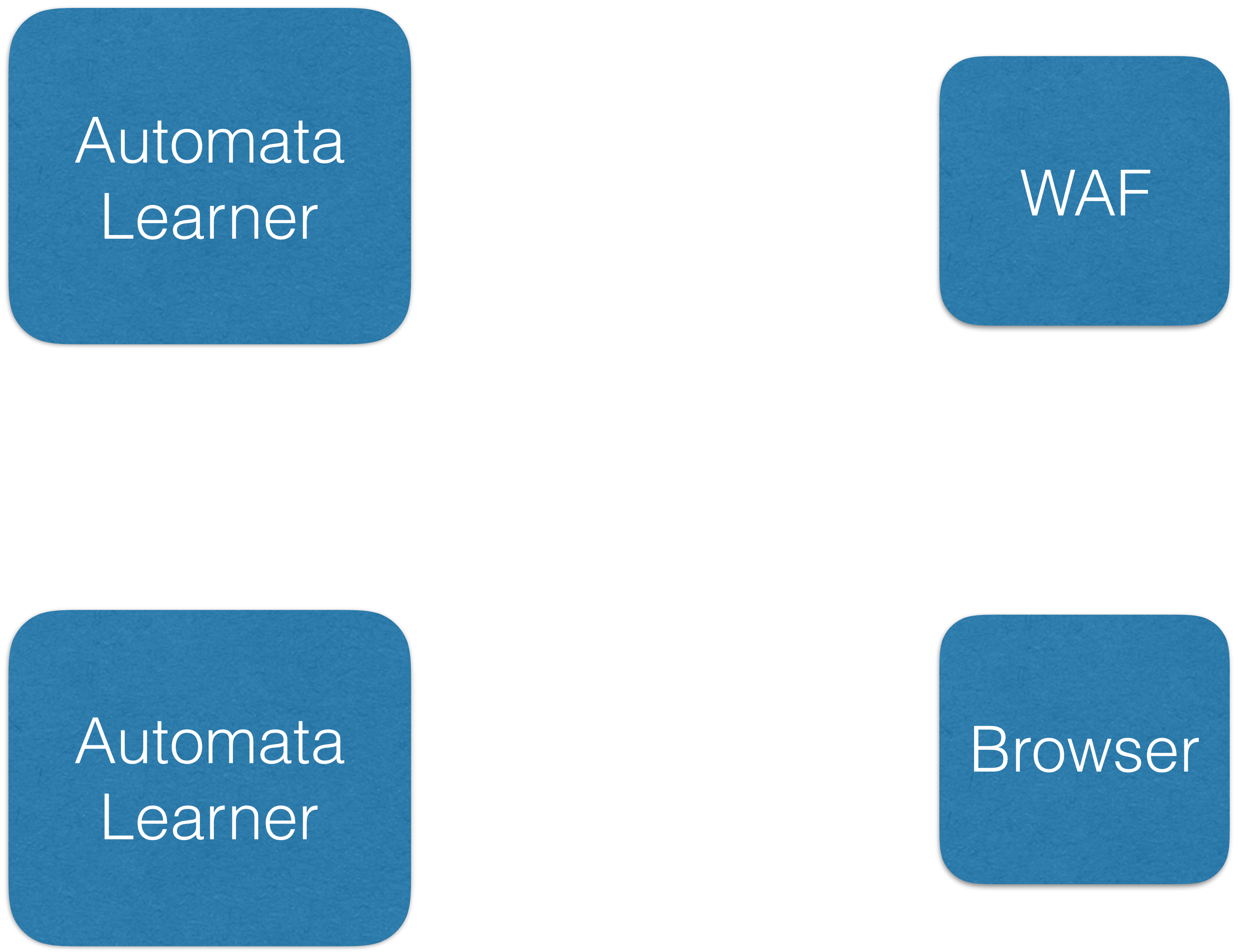
SFADiff

- Available grammars are not always good for finding vulnerabilities.
- Most XSS bypasses result from attack vectors deviating from the HTML standard.
  - <IMG SRC=“jav	ascript:alert(‘XSS');">
  - Tons of other examples.
- Use the same learning approach to infer the HTML parser specification!

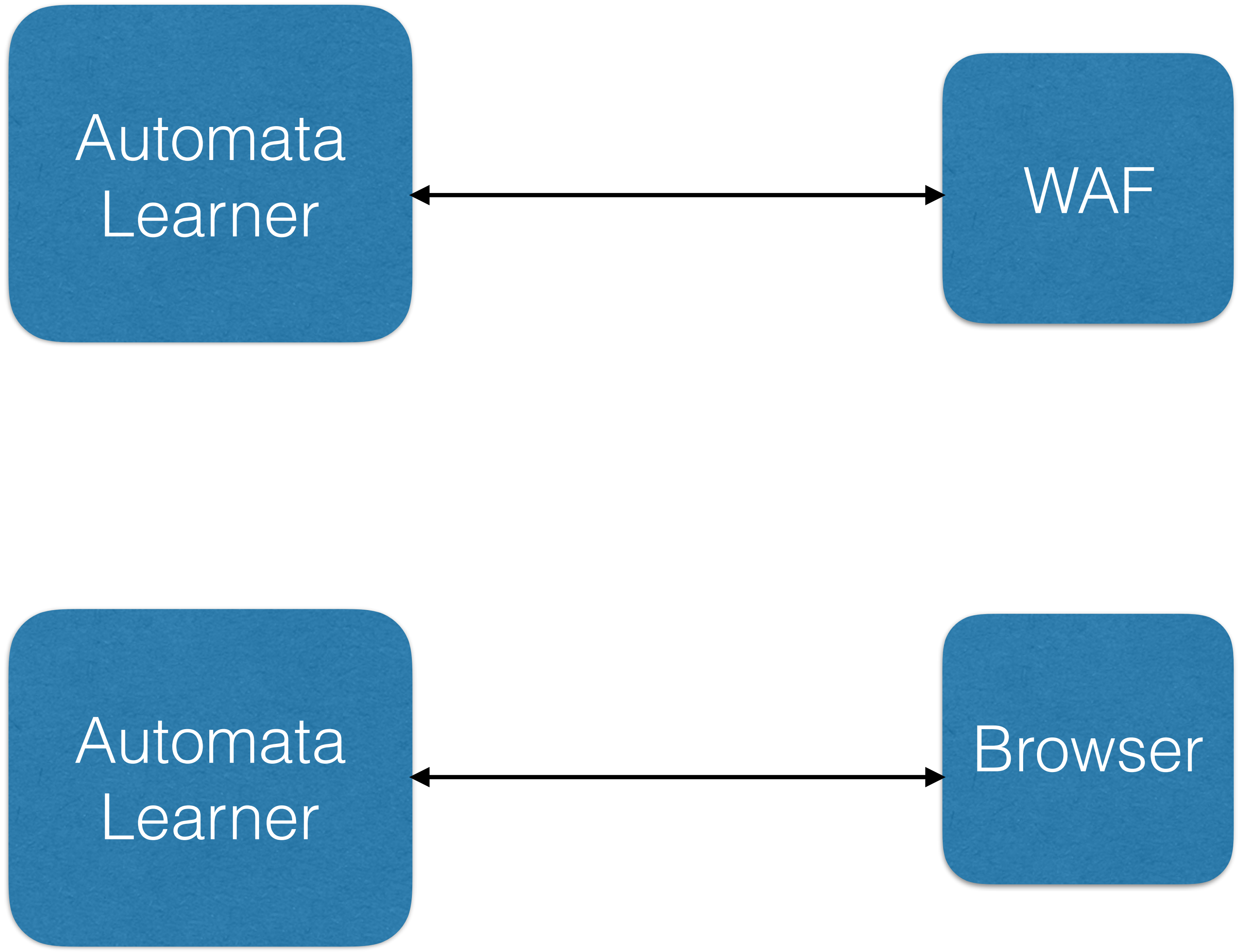
SFADiff



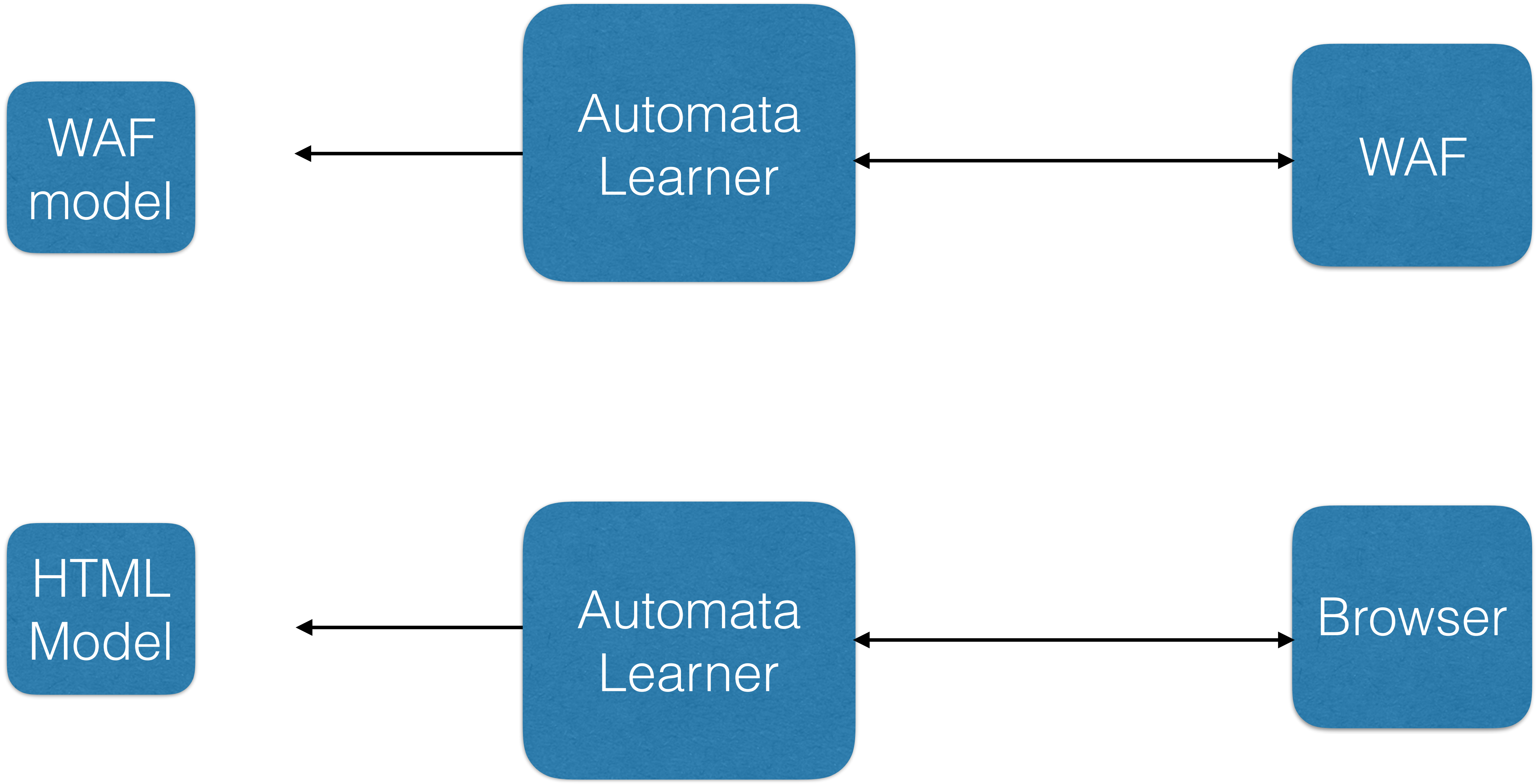
SFADiff



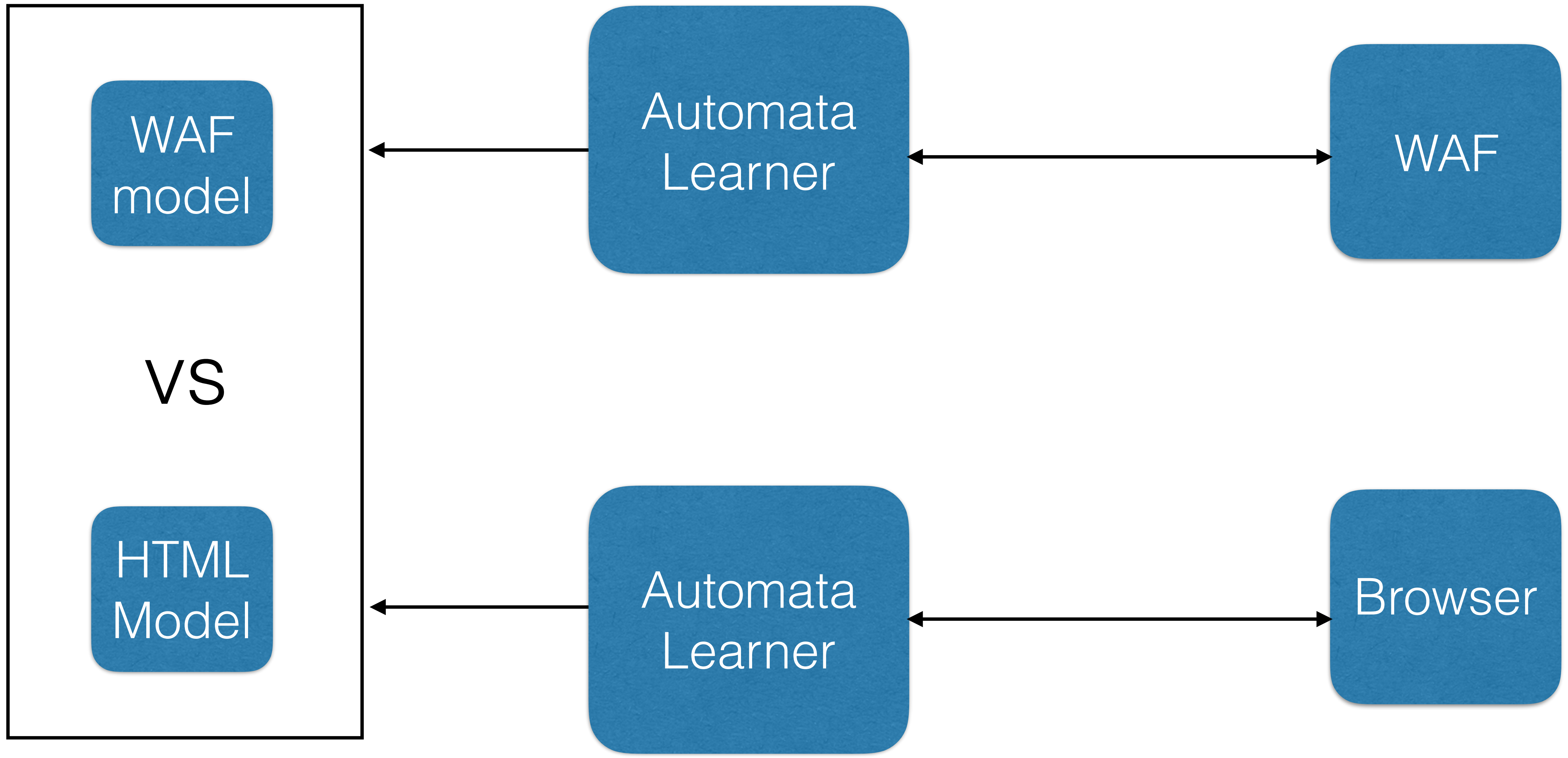
SFADiff



SFADiff

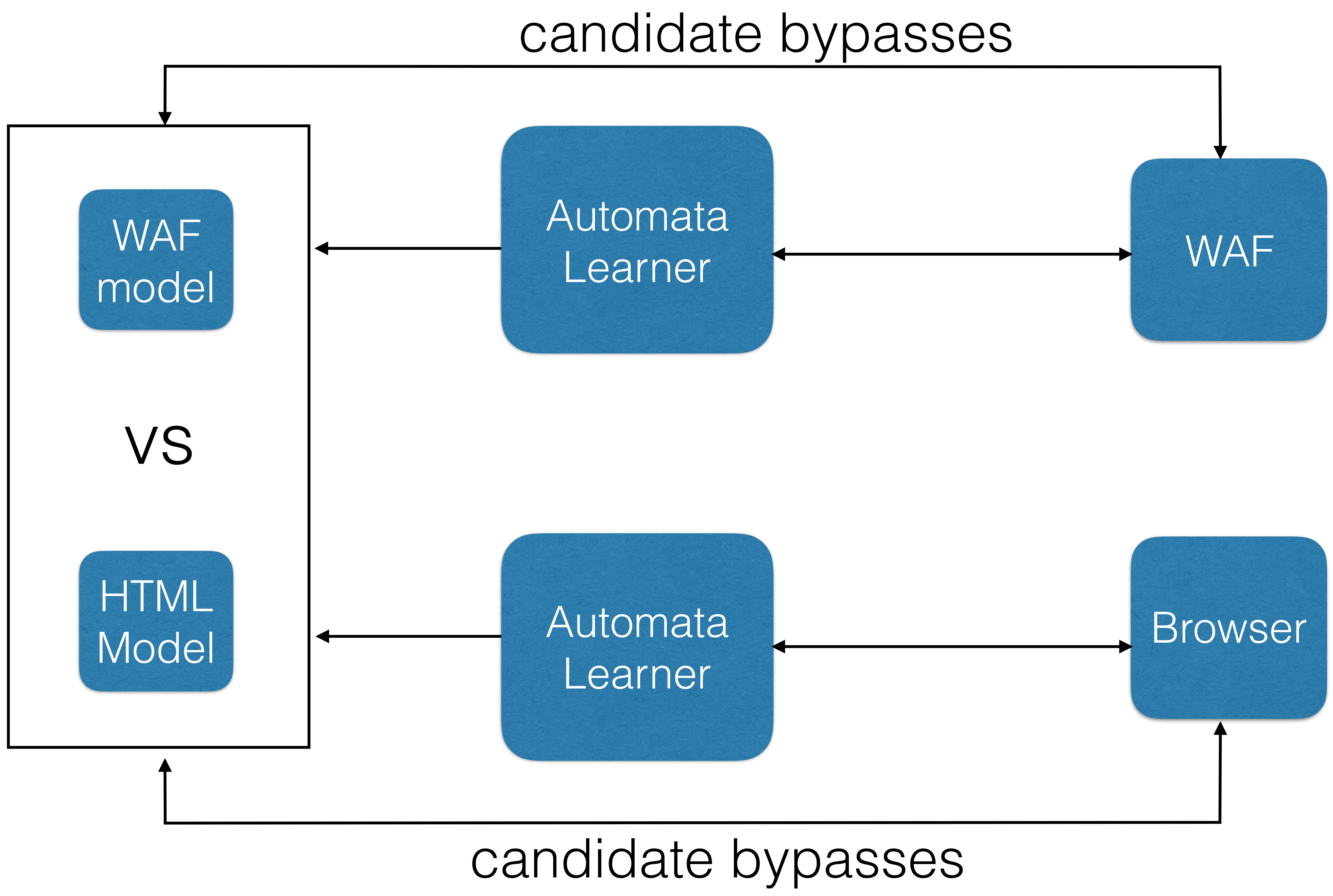

WAF
model
Automata
Learner
WAF
HTML
Model
Automata
Learner
Browser

SFADiff



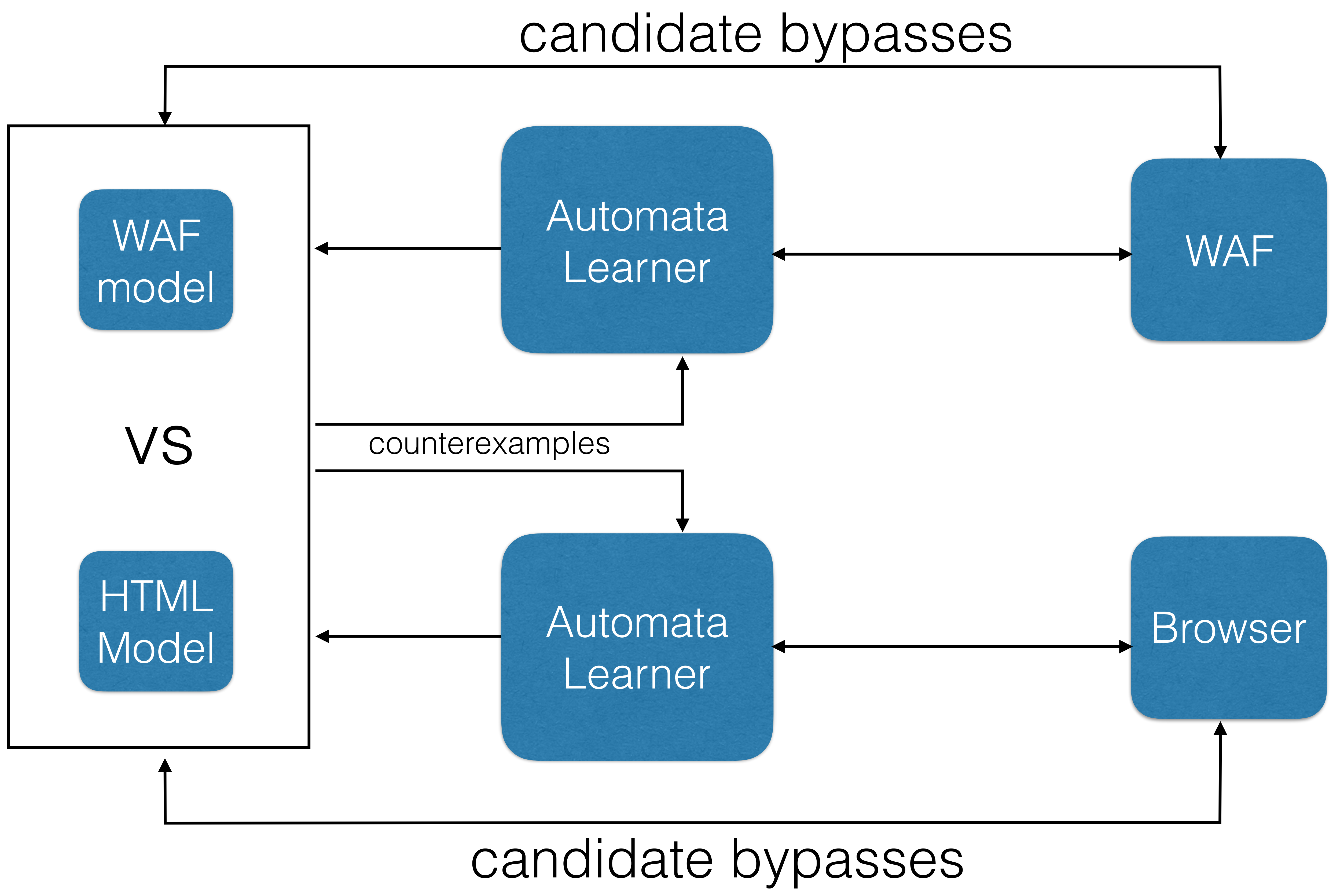
WAF
model
VS
HTML
Model
Automata
Learner
WAF
Automata
Learner
Browser

SFADiff

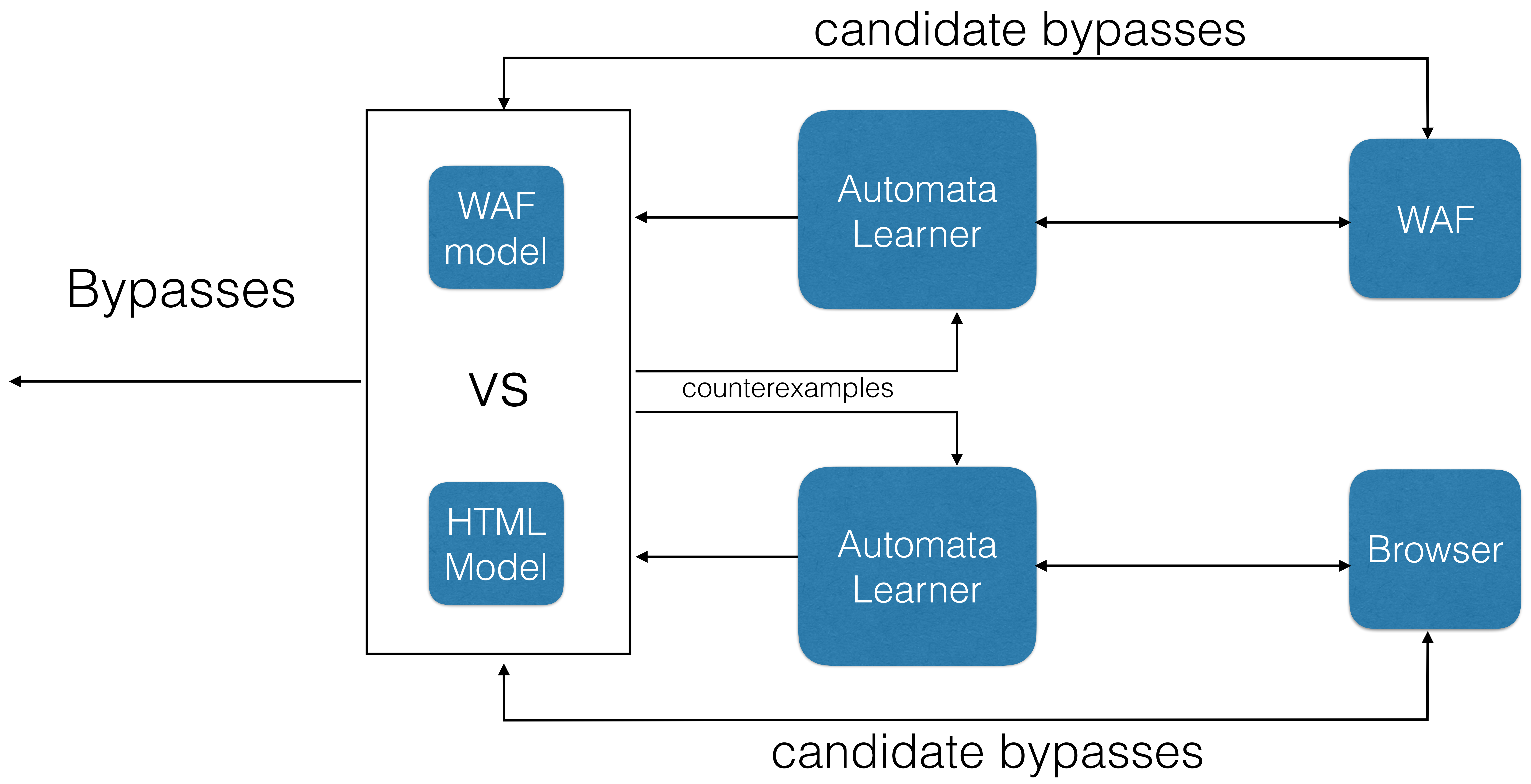


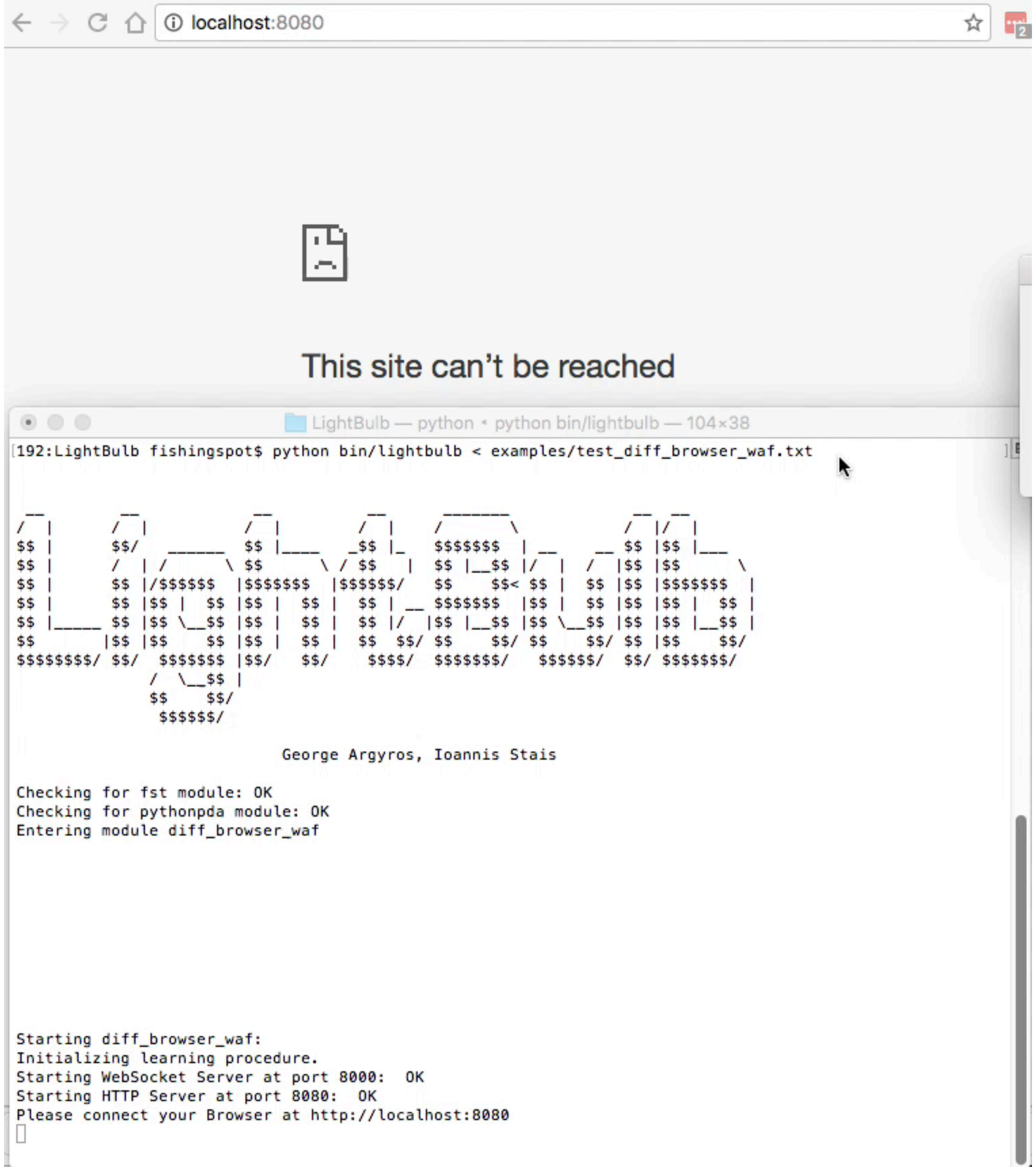
candidate bypasses
WAF model
VS
HTML Model
Automata Learner
WAF
Automata Learner
Browser
candidate bypasses

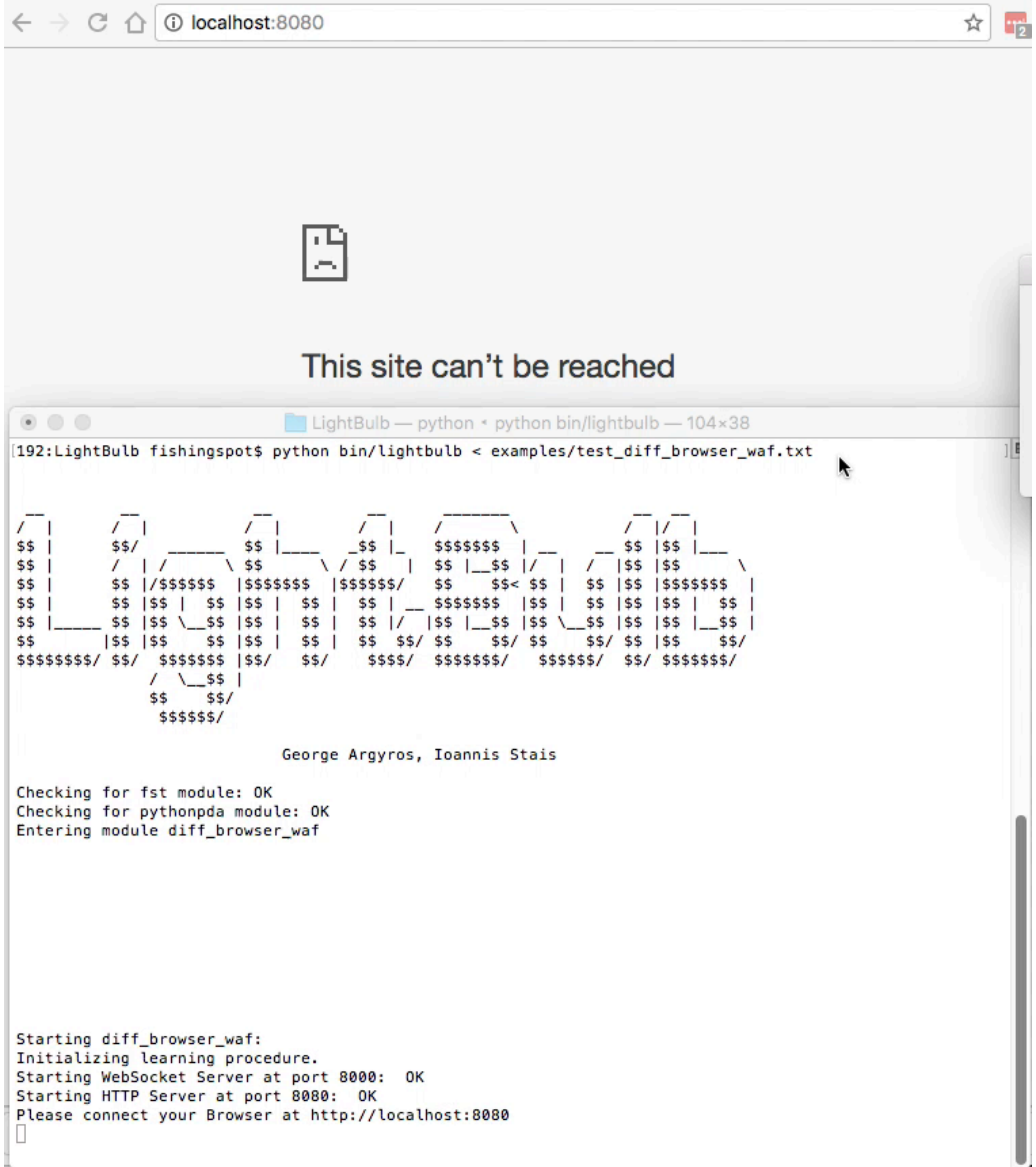
SFADiff


candidate bypasses
WAF model
VS
HTML Model
Automata Learner
WAF
counterexamples
Automata Learner
Browser
candidate bypasses

SFADiff


candidate bypasses
WAF model
VS
HTML Model
Bypasses
Automata Learner
WAF
counterexamples
Automata Learner
Browser
candidate bypasses


localhost:8080
This site can’t be reached
LightBulb — python ◂ python bin/lightbulb — 104×38
192:LightBulb fishingspot$ python bin/lightbulb < examples/test_diff_browser_waf.txt
George Argyros, Ioannis Stais
Checking for fst module: OK
Checking for pythonpda module: OK
Entering module diff_browser_waf
Starting diff_browser_waf:
Initializing learning procedure.
Starting WebSocket Server at port 8000: OK
Starting HTTP Server at port 8080: OK
Please connect your Browser at http://localhost:8080


localhost:8080
This site can’t be reached
LightBulb — python ◂ python bin/lightbulb — 104×38
192:LightBulb fishingspot$ python bin/lightbulb < examples/test_diff_browser_waf.txt
George Argyros, Ioannis Stais
Checking for fst module: OK
Checking for pythonpda module: OK
Entering module diff_browser_waf
Starting diff_browser_waf:
Initializing learning procedure.
Starting WebSocket Server at port 8000: OK
Starting HTTP Server at port 8080: OK
Please connect your Browser at http://localhost:8080

LightBulb — python ◂ python bin/lightbulb — 104×38

```
['', '>', 'p>', '/p>', '</p>', '></p>', ')></p>', '()></p>', 'a()></p>', '=a()></p>', 'k=a()></p>', 'ck=
a()></p>', 'ick=a()></p>', 'lick=a()></p>', 'click=a()></p>', 'nclick=a()></p>', 'onclick=a()></p>', ' o
nclick=a()></p>', 'p onclick=a()></p>', '<p onclick=a()></p>']
Verifying Web Socket connection: OK
Awaiting initialization command: OK
Initializing learning procedure.
Initialized from DFA em_vector table is the following:
['', '>', 'p>', '/p>', '</p>', '></p>', ')></p>', '()></p>', 'a()></p>', '=a()></p>', 'k=a()></p>', 'ck=
a()></p>', 'ick=a()></p>', 'lick=a()></p>', 'click=a()></p>', 'nclick=a()></p>', 'onclick=a()></p>', ' o
nclick=a()></p>', 'p onclick=a()></p>', '<p onclick=a()></p>']
Generating a closed and consistent observation table.
Generated conjecture machine with 26 states.
Generating a closed and consistent observation table.
Generated conjecture machine with 22 states.
Processing counterexample <p onclick=<()> with length 15.
Processing counterexample <p oncli; onclick=a()></p> with length 26.
Generated conjecture machine with 26 states.
Generated conjecture machine with 24 states.
Processing counterexample <p onclick=a()< with length 15.
Processing counterexample <p onclick=a()< with length 15.
Generated conjecture machine with 29 states.
Generated conjecture machine with 24 states.
Processing counterexample <p onclick=a();p0=a()></p> with length 26.
Processing counterexample <p onclick=:</p> with length 16.
Generated conjecture machine with 29 states.
Generated conjecture machine with 24 states.
Processing counterexample <p onclick=nclick=a()></p> with length 26.
Processing counterexample <p onclick=a;p> with length 15.
Generated conjecture machine with 29 states.
Generated conjecture machine with 24 states.
Processing counterexample <p onclick=nclick=a()></p> with length 26.
Processing counterexample <p onclick=a;p> with length 15.
Generated conjecture machine with 29 states.
Generated conjecture machine with 24 states.
Processing counterexample <p onclick=nclick=a()></p> with length 26.
Processing counterexample <p onclick=a;p> with length 15.
Generated conjecture machine with 29 states.
```

SFADiff XSS Bypass

- XSS Attack vectors in PHPIDS 0.7/ Expose 2.4.0

<p onmouseover=-a() ></p>

<p onmouseover=(a()) ></p>

<p onmouseover=;a() ></p>

<p onmouseover=!a() ></p>

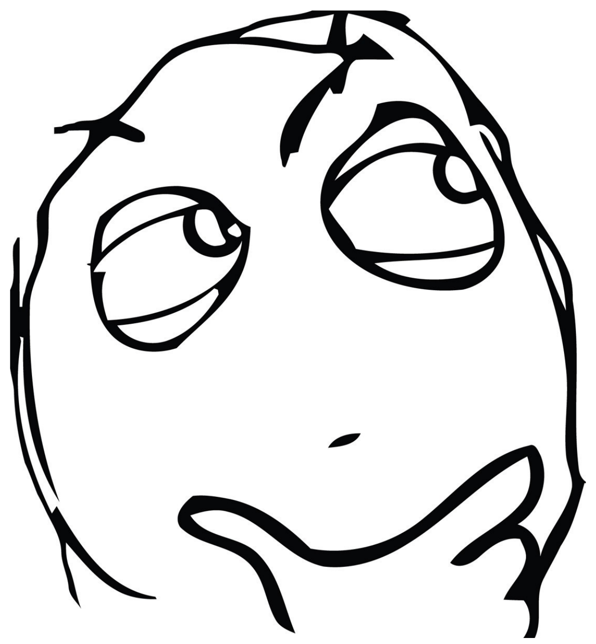
- Other types of events can also be use used for the attack (e.g. "onClick").
- Rules 71, 27, 2 and 65 are related to this insufficient pattern match.

Bonus:
Fingerprinting WAFs

Generating Program Fingerprints

P_1 P_2 … P_N

P_T

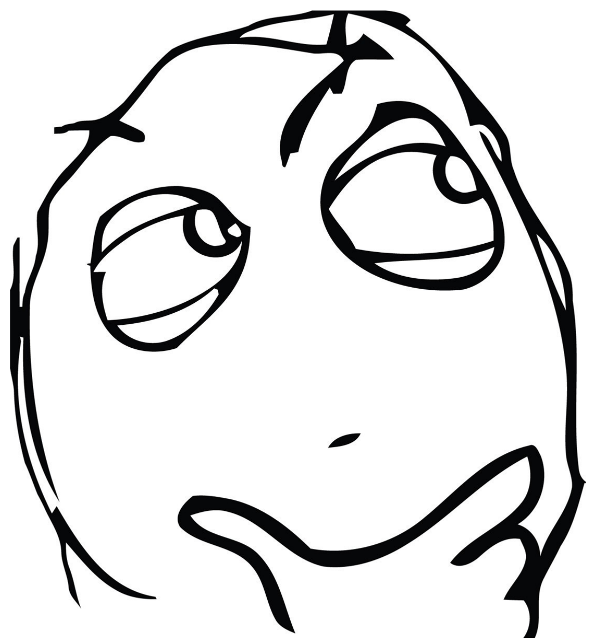


Generating Program Fingerprints

P_1 P_2 … P_N

Which program is running in the Black-box?

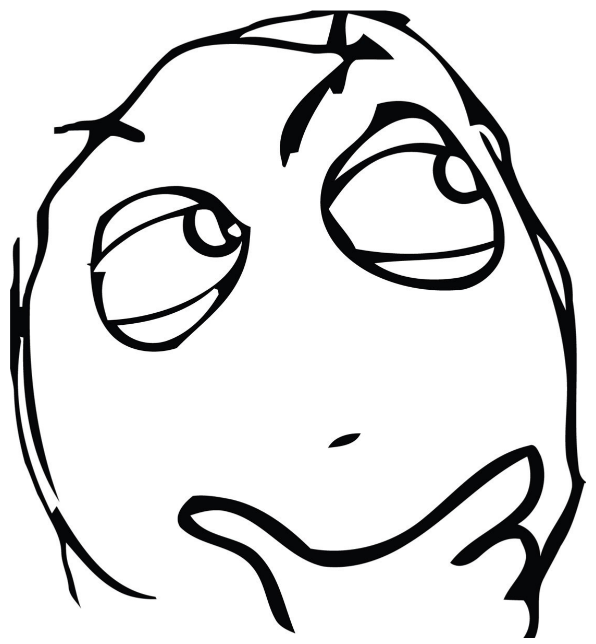
P_T



Generating Program Fingerprints

P_1 P_2 … P_N

P_T

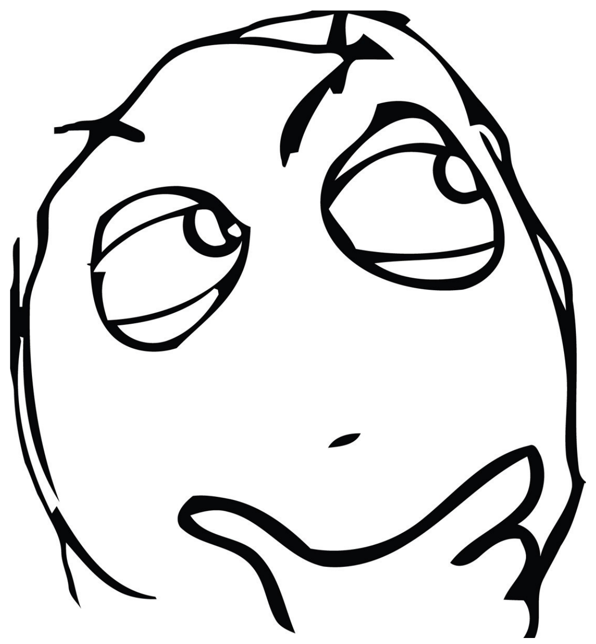


Generating Program Fingerprints

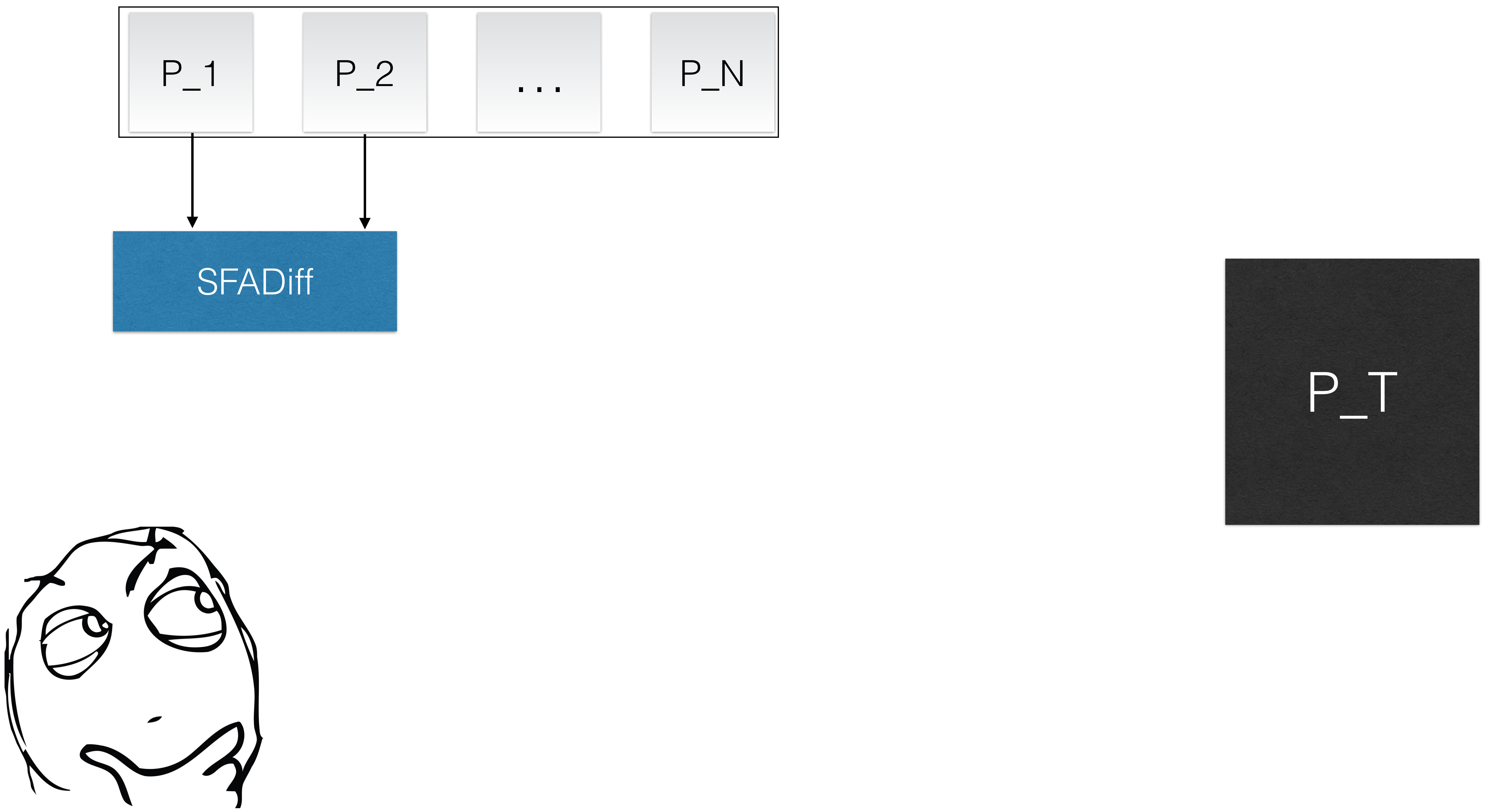
P_1 | P_2 | . . . | P_N

SFADiff

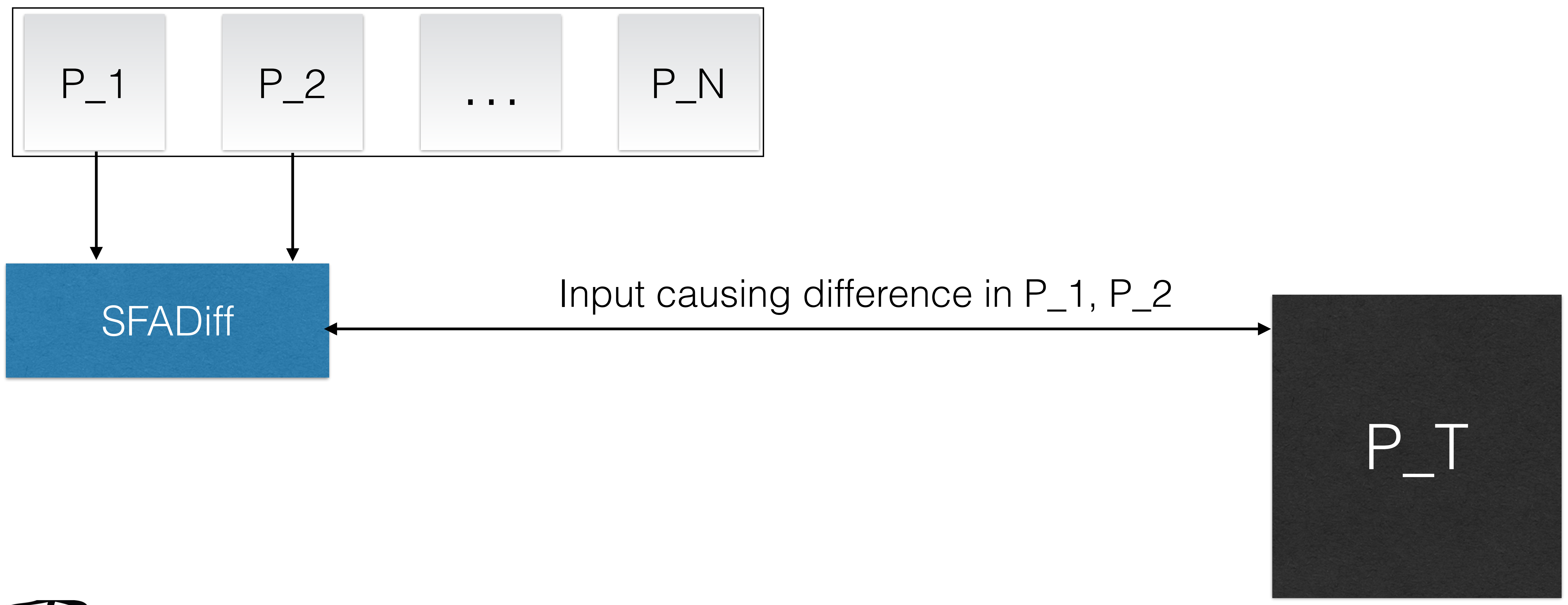
P_T

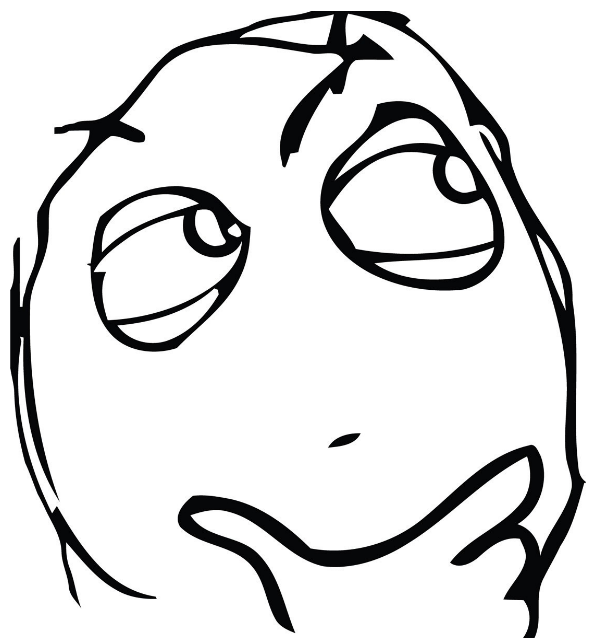


Generating Program Fingerprints

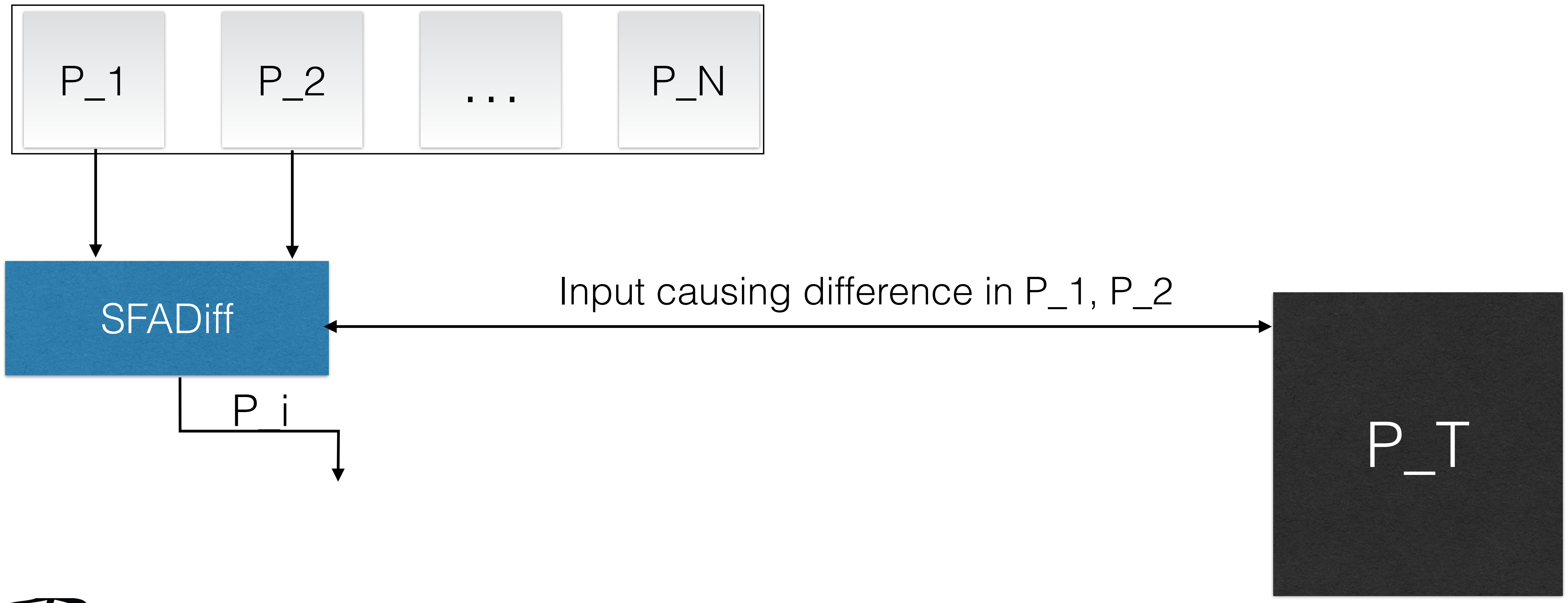


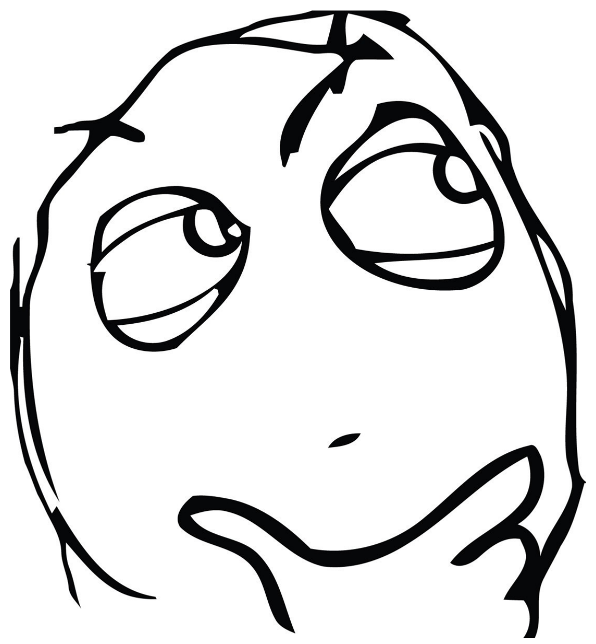
Generating Program Fingerprints



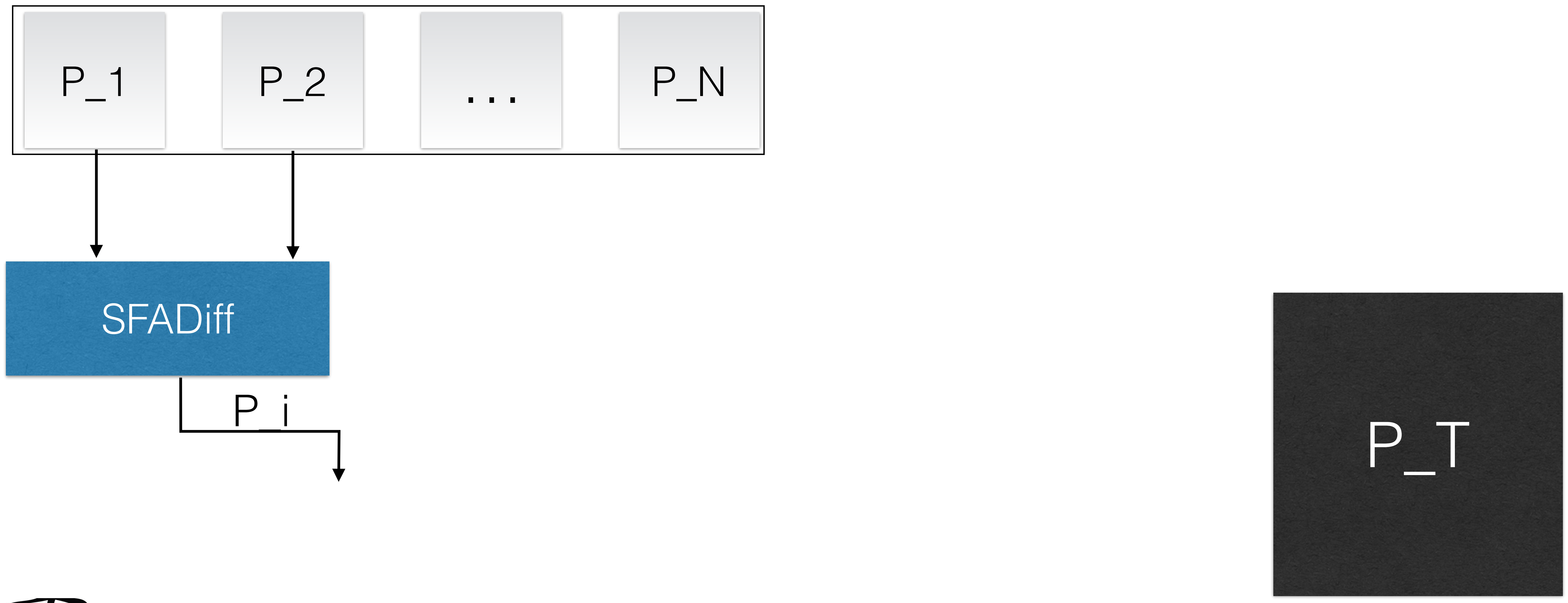


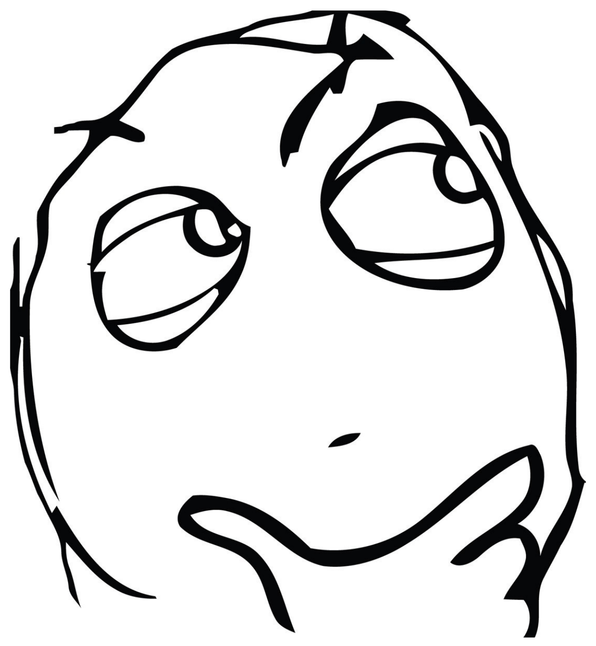
Generating Program Fingerprints



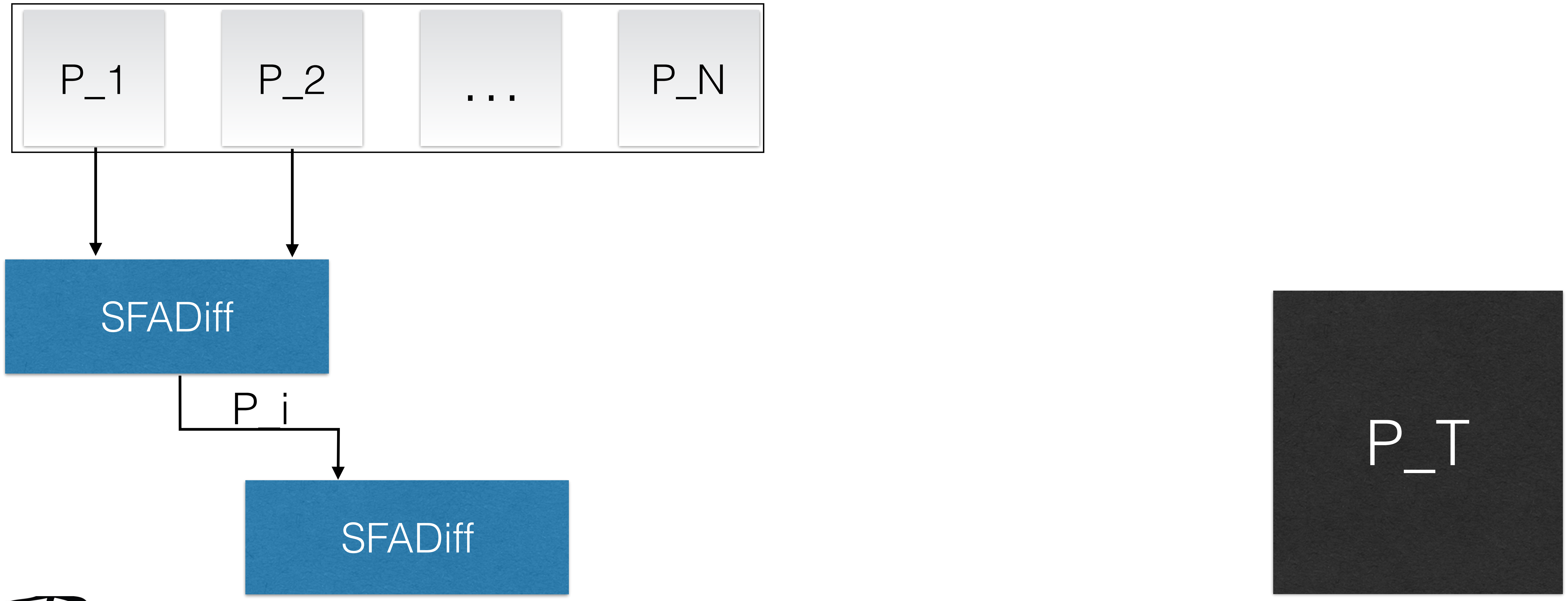


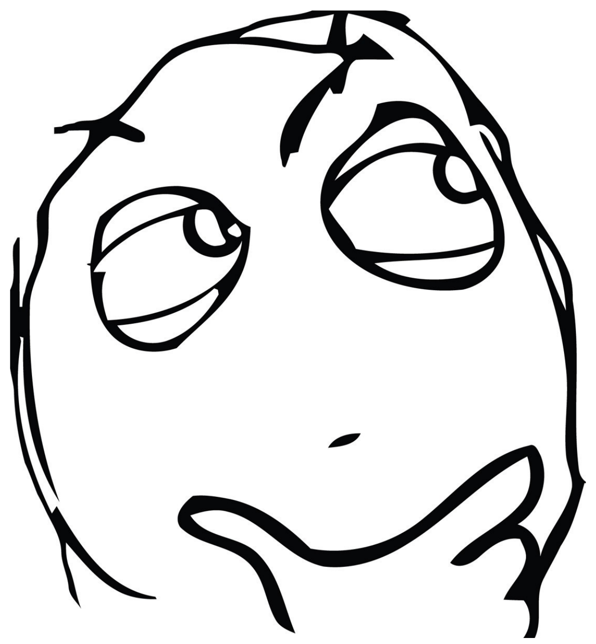
Generating Program Fingerprints



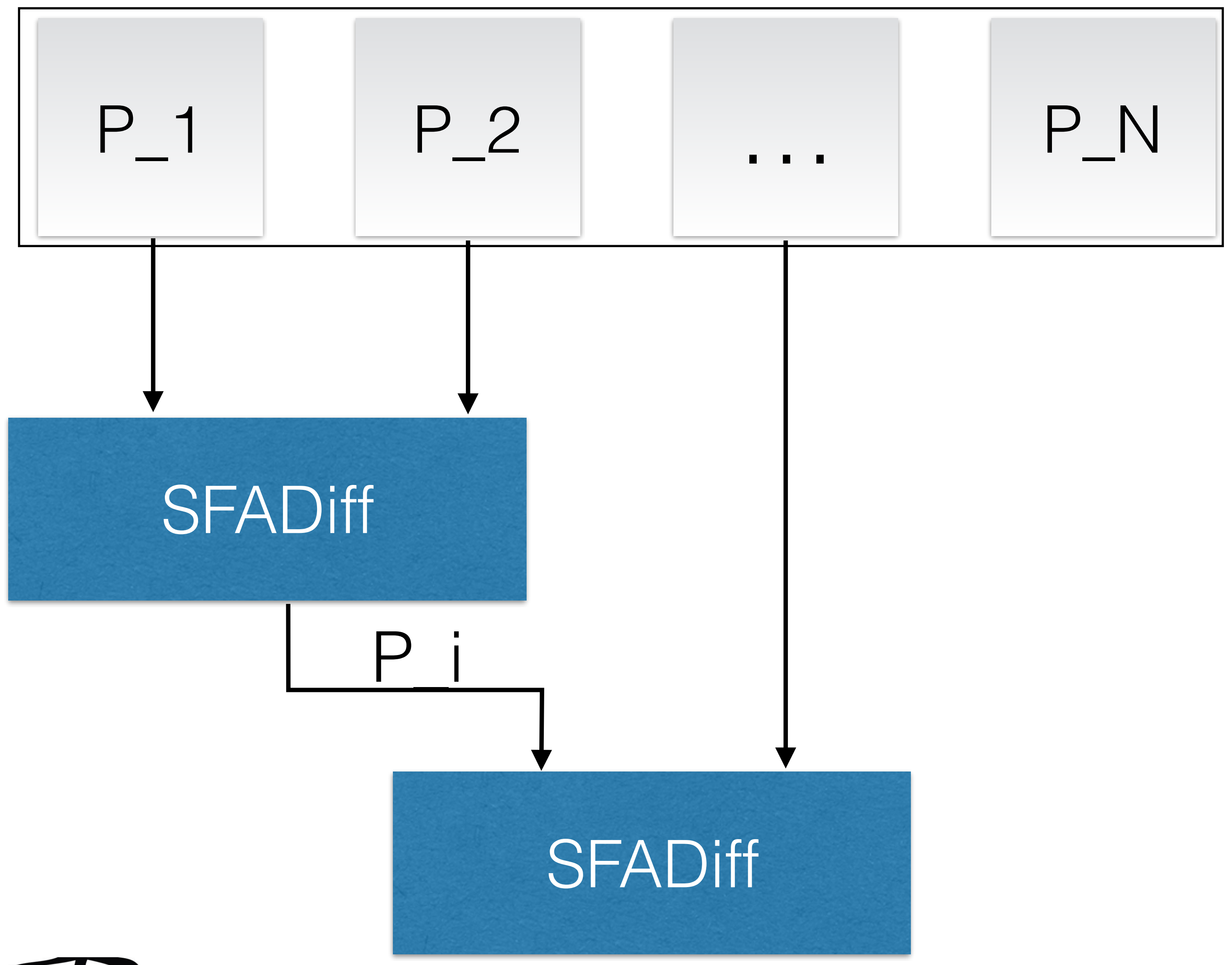


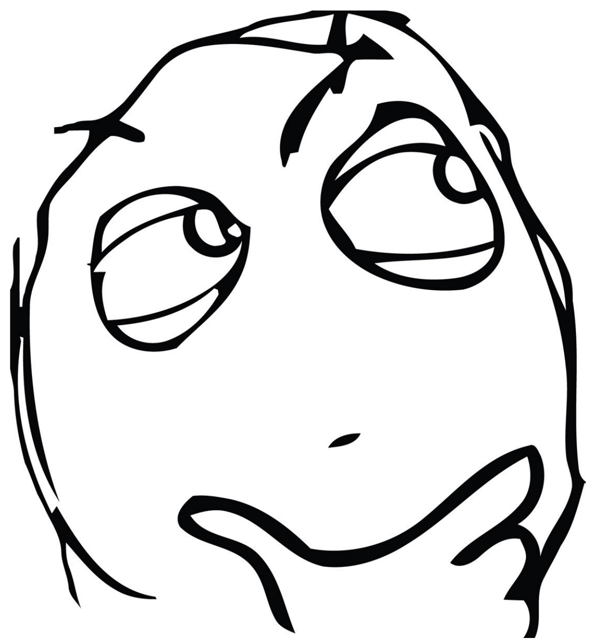
Generating Program Fingerprints

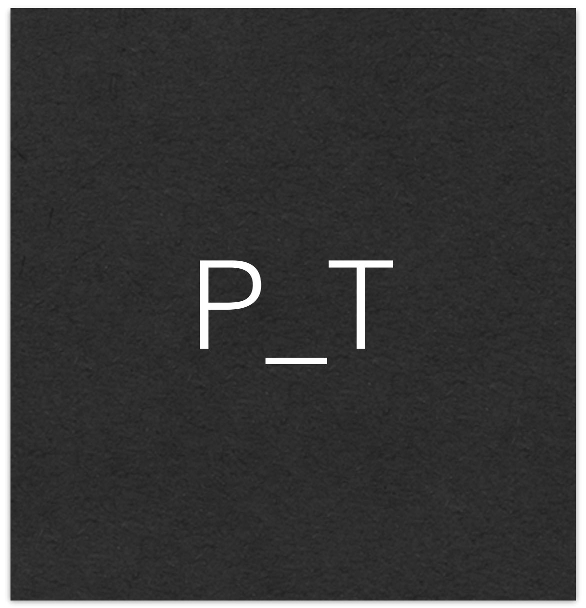




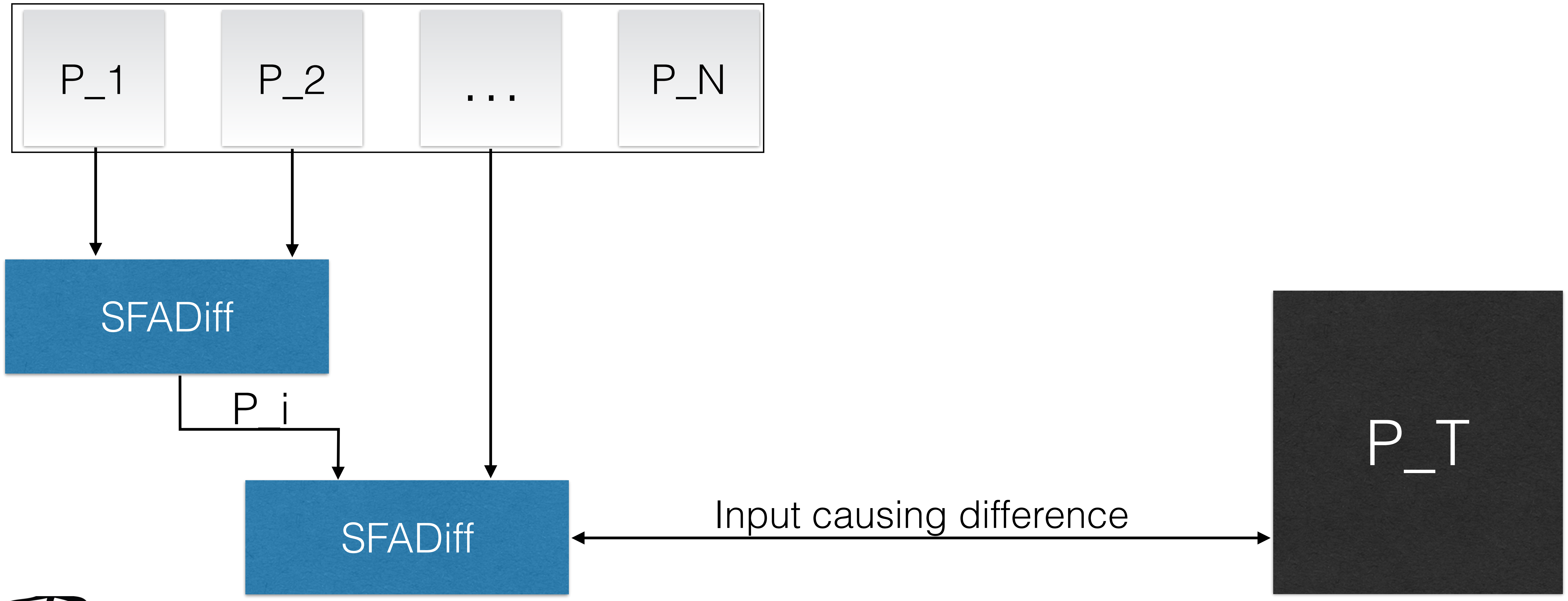
Generating Program Fingerprints

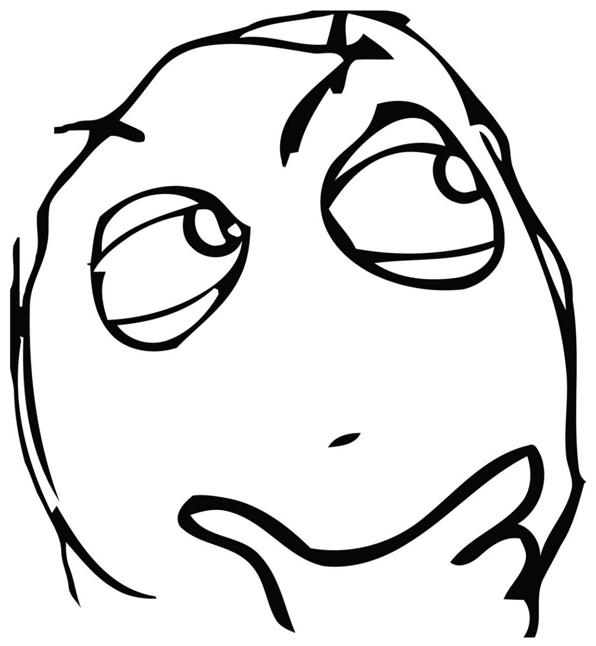




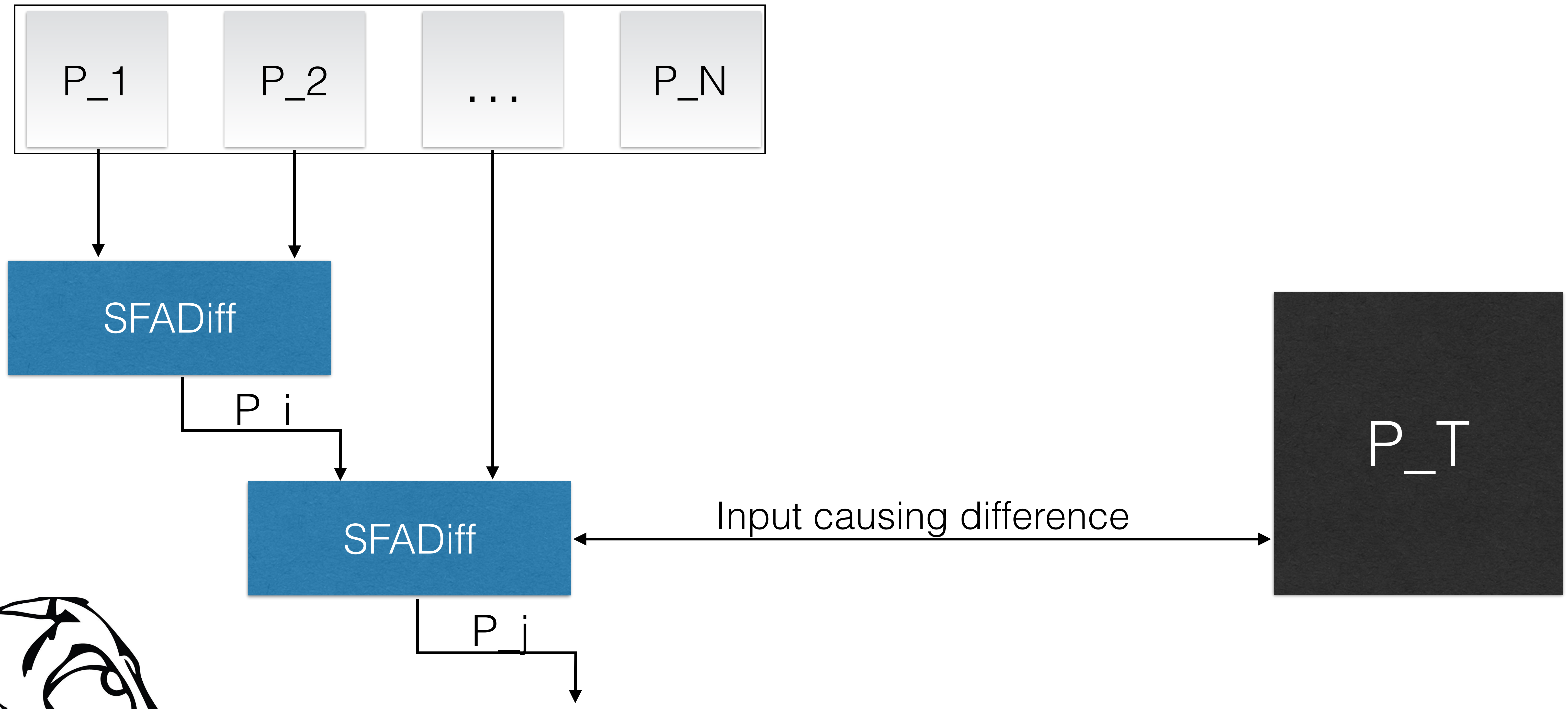


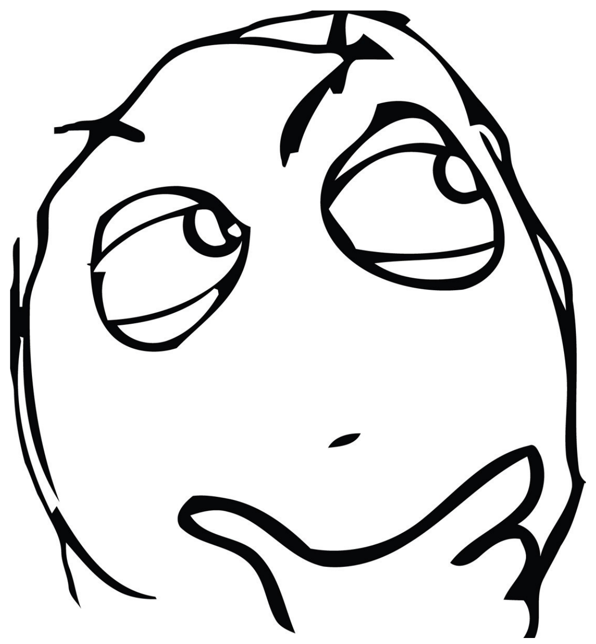
Generating Program Fingerprints



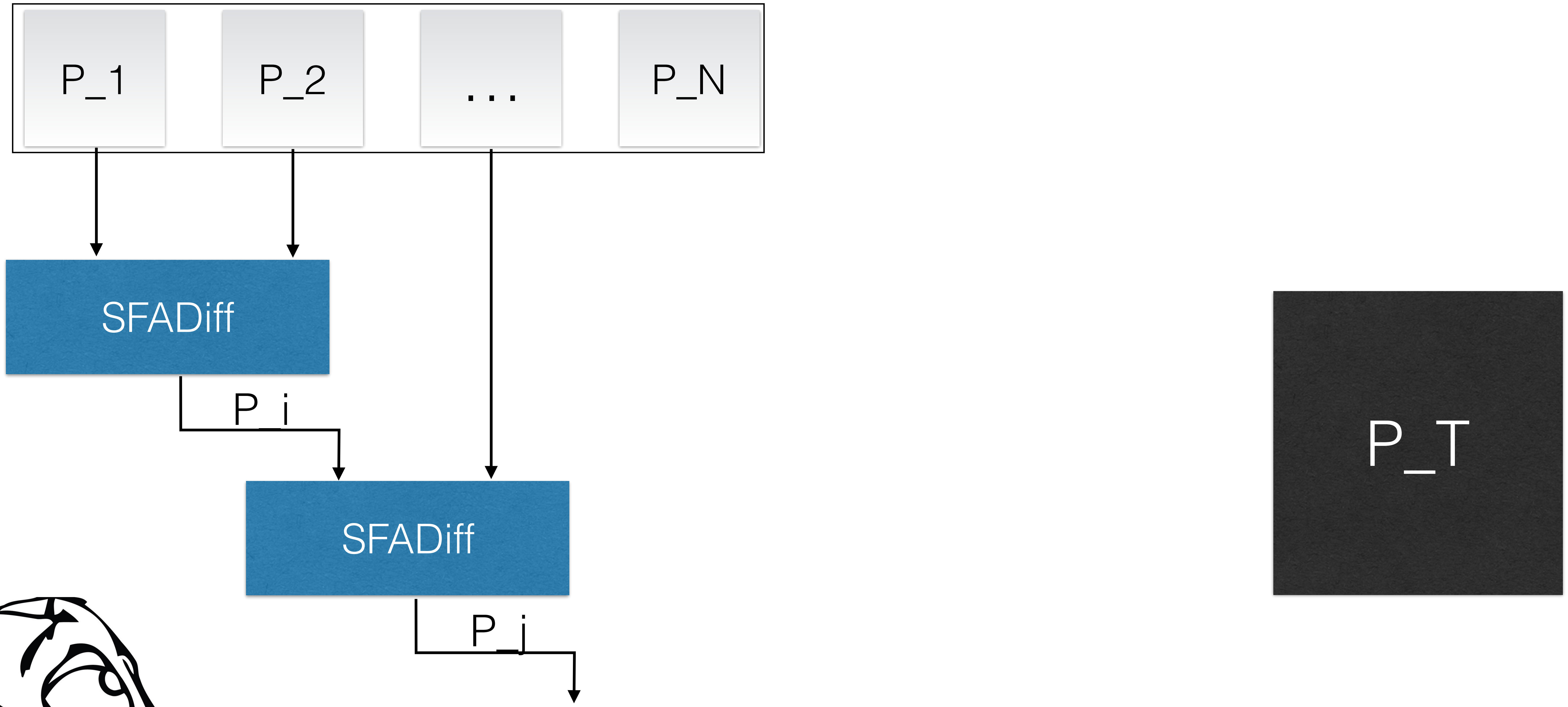


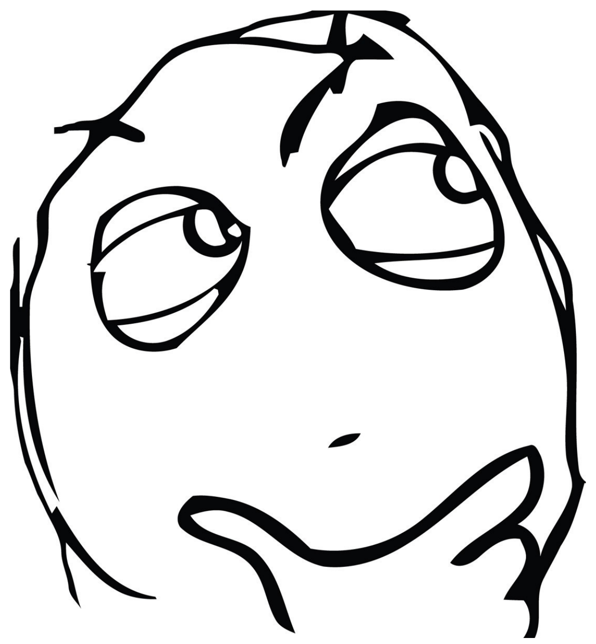
Generating Program Fingerprints



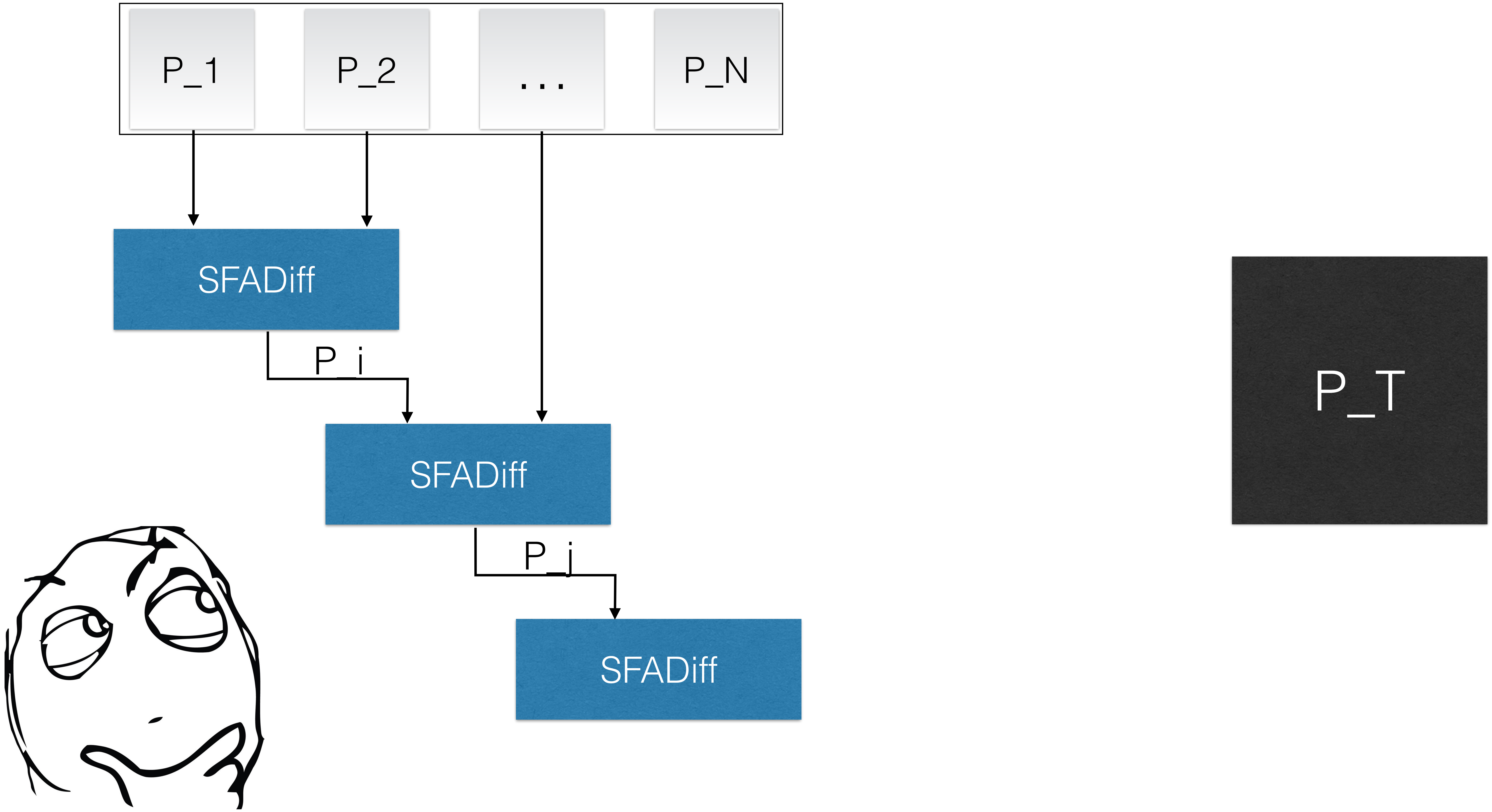


Generating Program Fingerprints

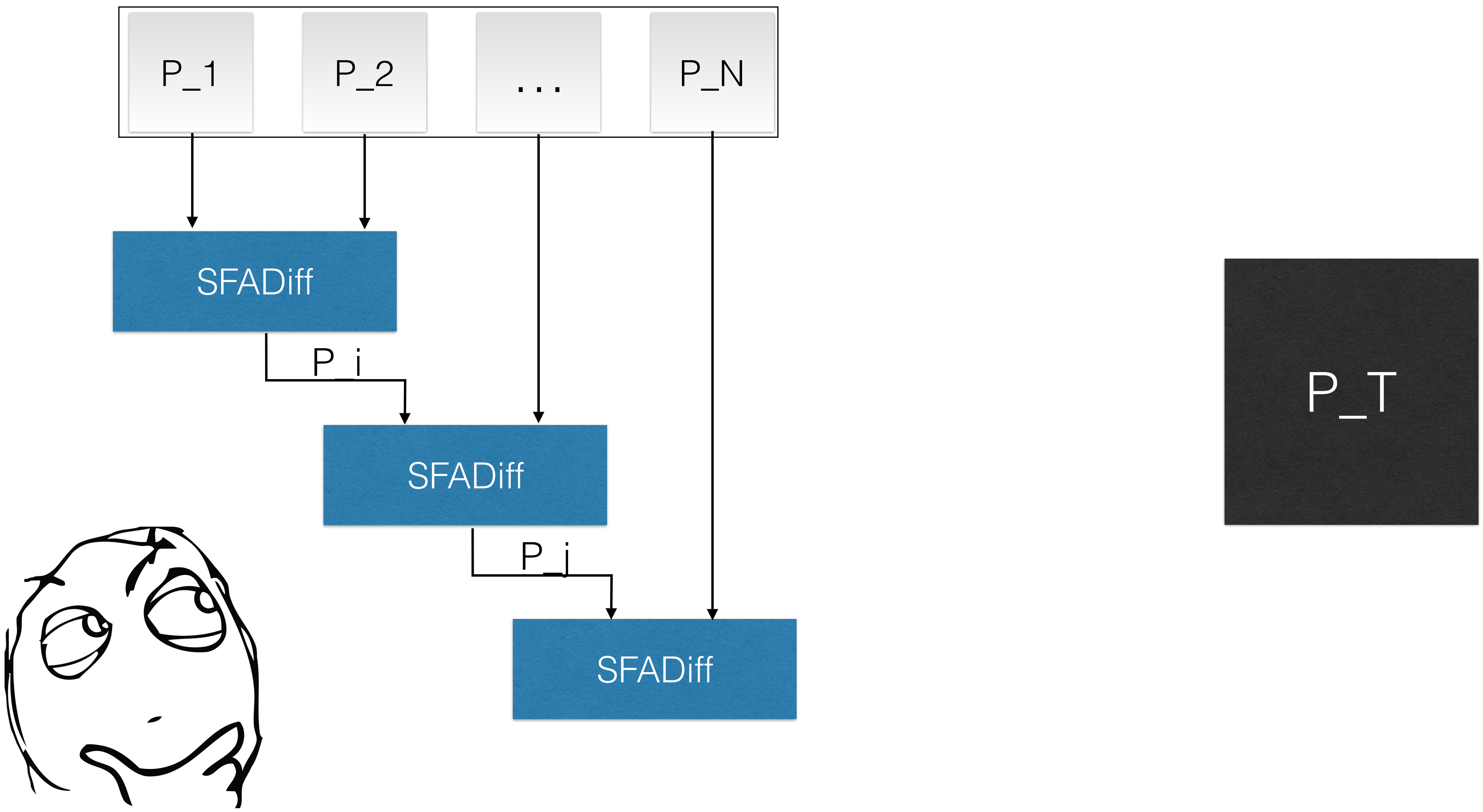




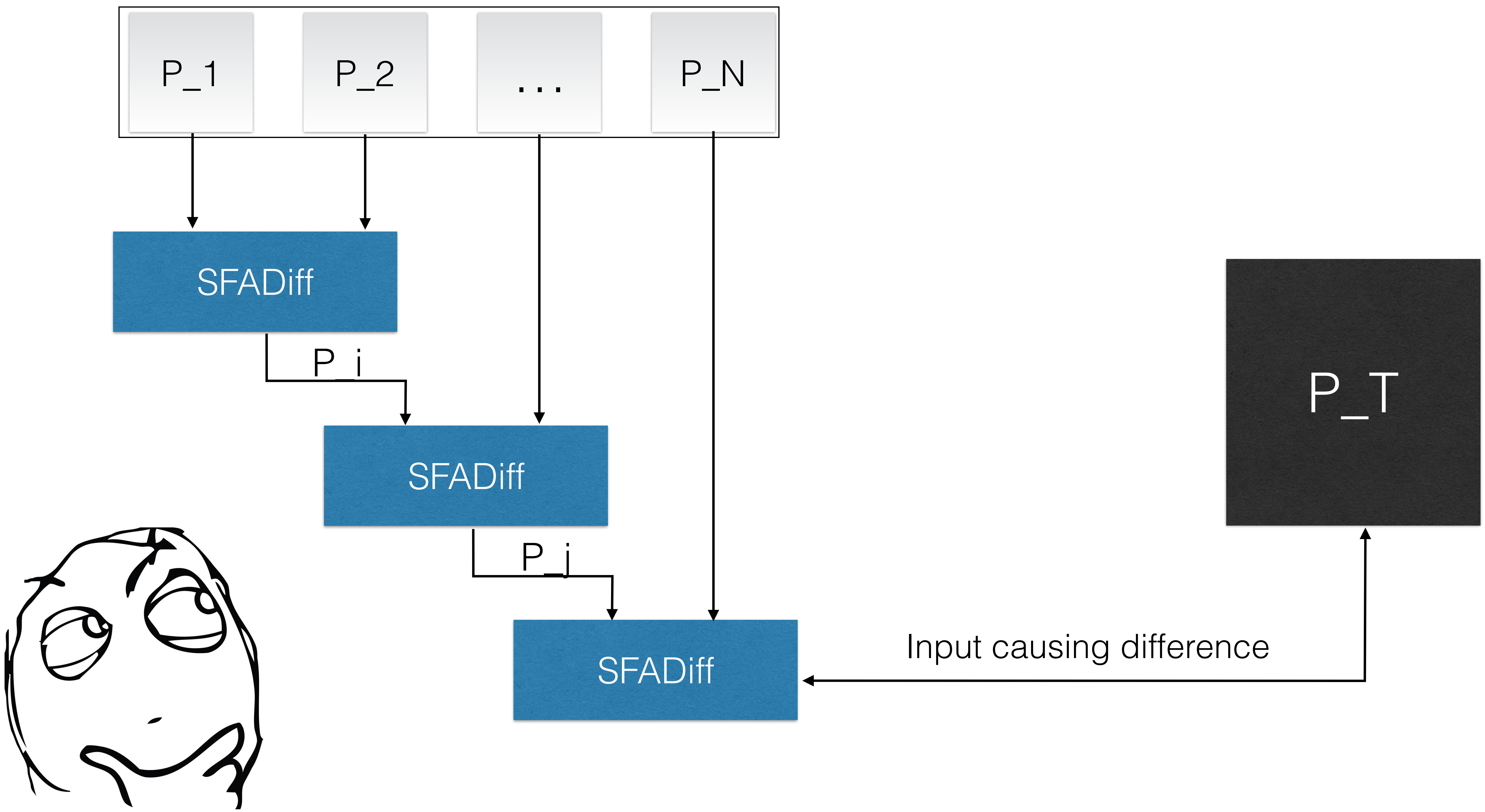
Generating Program Fingerprints



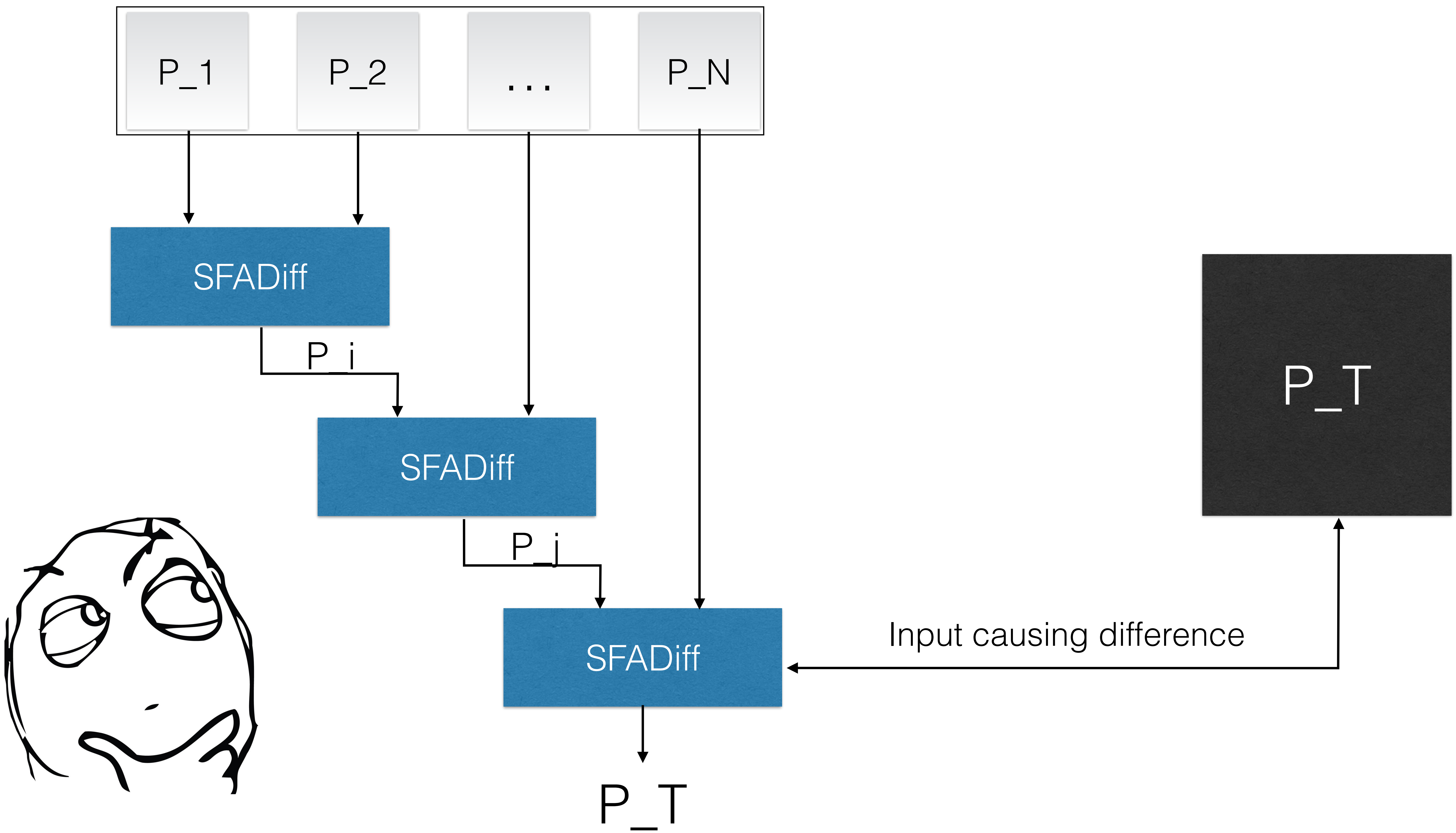
Generating Program Fingerprints



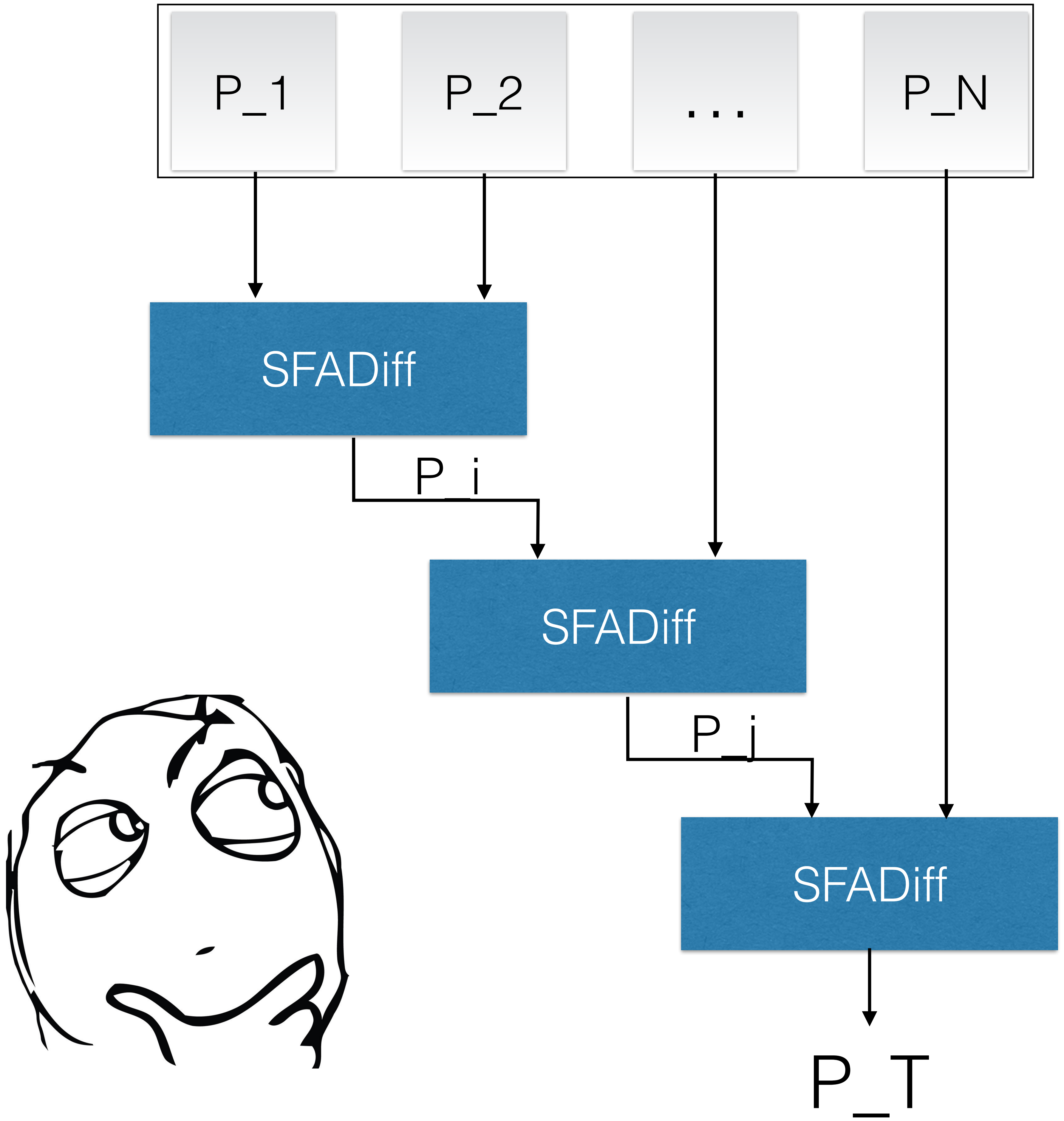
Generating Program Fingerprints



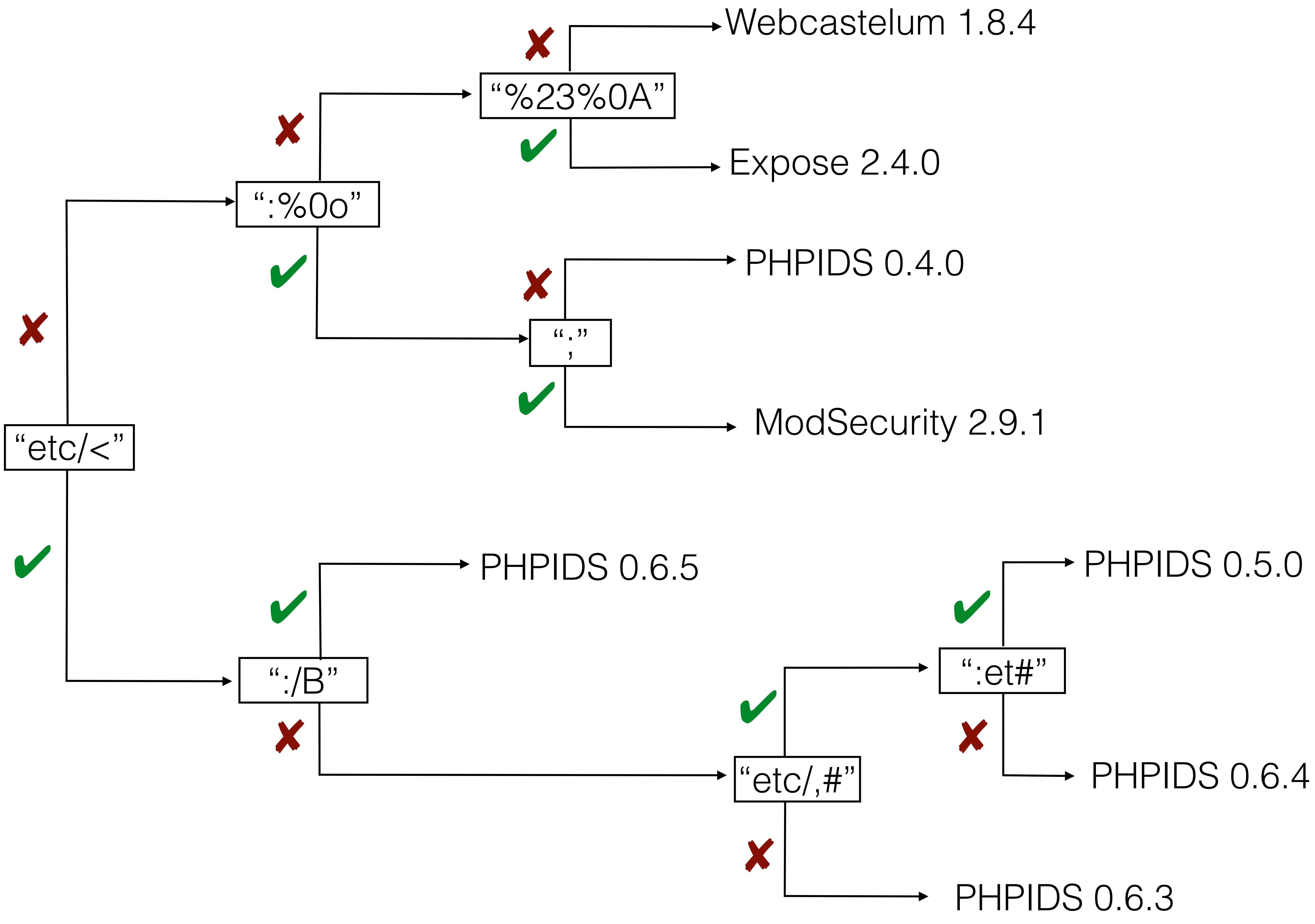
Generating Program Fingerprints



Generating Program Fingerprints



P_T



"etc/<"
":%0o"
"%23%0A"
Webcastelum 1.8.4
Expose 2.4.0
";"
PHPIDS 0.4.0
ModSecurity 2.9.1
":/B"
PHPIDS 0.6.5
"etc/,#"
":et#"
PHPIDS 0.5.0
PHPIDS 0.6.4
PHPIDS 0.6.3

LightBulb

Modular Design

- **Core Modules**:
    - Use automata models and operations
    - Extend the SFA learning algorithm

- **Built-in Query Handlers:**
    - Perform membership queries

- **Modules (and Built-in Modules):**
    - Use the Built-in Query Handlers
    - Extend the Core Modules: GOFA, SFADiff

- **Library:**
    - Set of grammars, filters, fingerprints trees and configurations

Core Modules

- Extend SFA Learning algorithm:
    - Accept the Alphabet**,** a Seed and/or a Tests file and a Query handler.
    - Initialise learning and manage results and models

- **The Alphabet:** Set of characters to be used

- **The Seed File:** Knowledge of what the examined inputs should look like

- **The Tests File:** Knowledge of specialised attacks

- **The Query Handler/Function:** Knowledge of how to perform queries for selected inputs

Core Modules

- **GOFA:**
  - Grammar Oriented Filter Auditing.

- **SFADiff:**
  - A black-box differential testing framework based on Symbolic Finite Automata (SFA) learning.

Simple Structure: *Class with five (5) basic functions:*
setup(), learn(), query(), getresults(), stats()

Built-in Query Handlers

- **HTTP Request Handler**:
  - Perform queries on WAF filters and Sanitizers

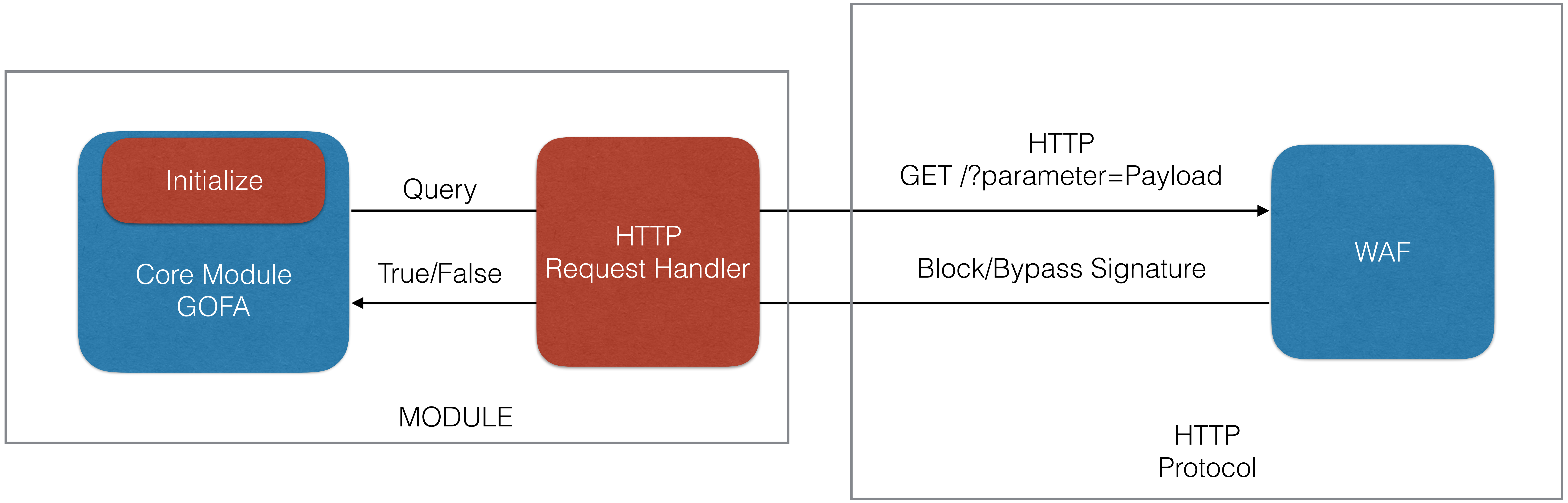
- **SQL Query Handler:**
  - Perform queries on MySQL Parser

- **Browser Parser Handler**:
  - Perform queries on Browser JavaScript Parsers

- **Browser Filter Handler**:
  - Perform queries on Browser Anti-XSS Filters

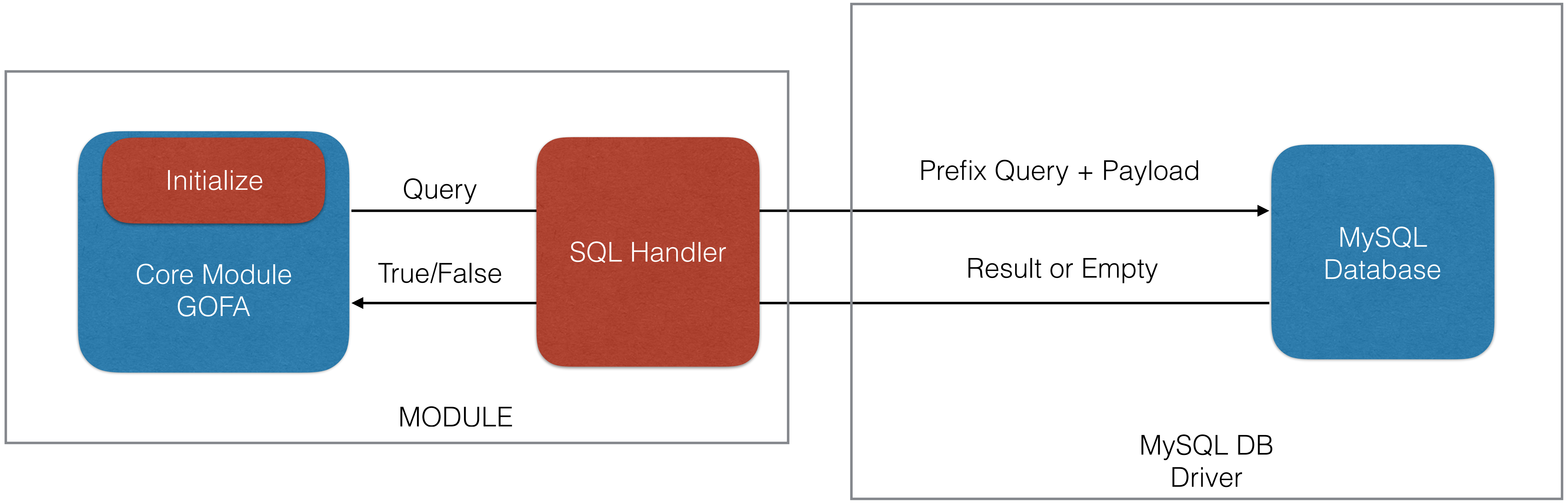
HTTP Request Handler

- Targets WAF Filter
- Requires URL, HTTP Request Type, Parameter and Block or Bypass Signature



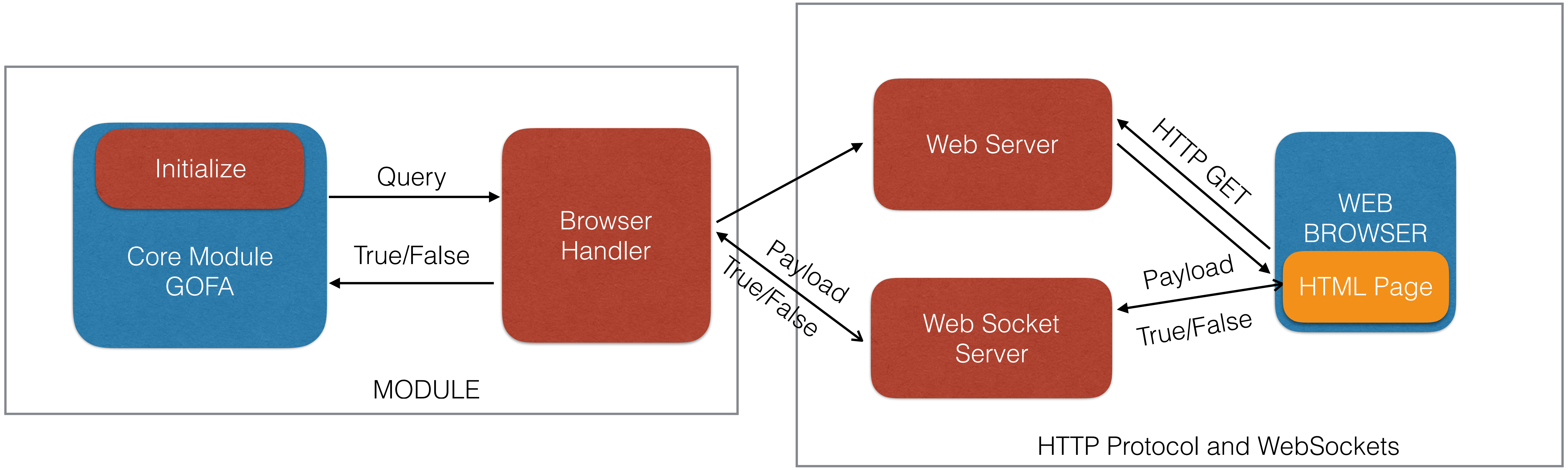
MySQL Query Handler

- Targets MySQL Database Parser
- Requires Database Credentials
- Requires Prefix Query: e.g, *“SELECT a FROM a WHERE a=**”*



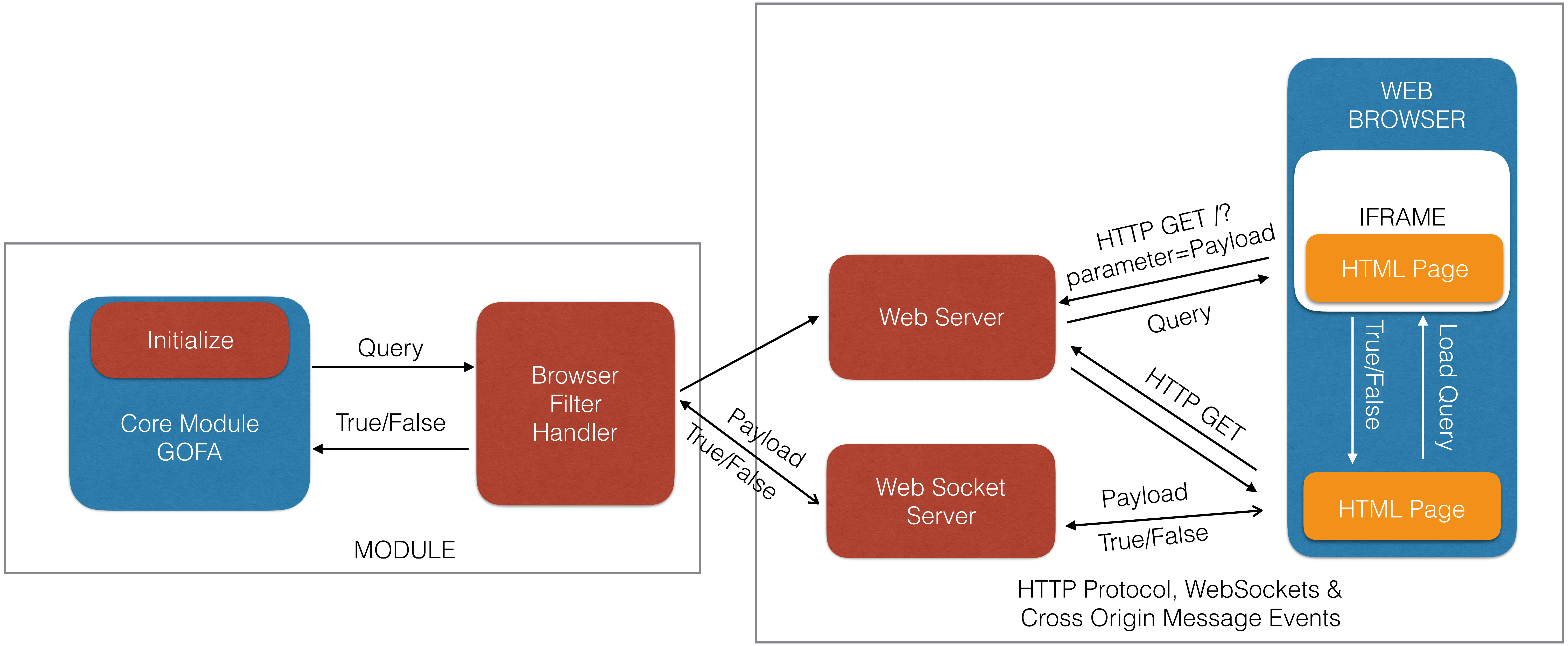
Browser Parser Handler

- Targets the Browser HTML and JavaScript Parsing Engine
- Requires web sockets port, web browser port, host and trigger delay
- Inputs must trigger function a() (e.g., <script>a();</script>)



Browser Filter Handler

- Targets the Browser Anti-XSS Filter, HTML and JavaScript Parsing Engine



Using GOFA module and HTTP Handler

Using GOFA module and HTTP Handler

```
use HTTPHandler as my_query_handler
define URL http://83.212.105.5/PHPIDS07/
define BLOCK impact
back
```

Using GOFA module and HTTP Handler

```
use HTTPHandler as my_query_handler
define URL http://83.212.105.5/PHPIDS07/
define BLOCK impact
back
```

Query Handler was created.
We now can perform
membership requests.

Using GOFA module and HTTP Handler

use HTTPHandler as my_query_handler
define URL http://83.212.105.5/PHPIDS07/
define BLOCK impact
back

use GOFA as my_gofa
define TESTS_FILE {library}/regex/PHPIDS070/12.y
define HANDLER my_query_handler
back

Query Handler was created.
We now can perform
membership requests.

Using GOFA module and HTTP Handler

```
use HTTPHandler as my_query_handler
define URL http://83.212.105.5/PHPIDS07/
define BLOCK impact
back
```

Query Handler was created.
We now can perform membership requests.

```
use GOFA as my_gofa
define TESTS_FILE {library}/regex/PHPIDS070/12.y
define HANDLER my_query_handler
back
```

Algorithm was selected and populated.
Know we can learn application states.

Using GOFA module and HTTP Handler

use HTTPHandler as my_query_handler
define URL http://83.212.105.5/PHPIDS07/
define BLOCK impact
back

use GOFA as my_gofa
define TESTS_FILE {library}/regex/PHPIDS070/12.y
define HANDLER my_query_handler
back

start my_gofa

Query Handler was created.
We now can perform
membership requests.

Algorithm was selected and populated.
Know we can learn
application states.

Built-in Modules

- **WAF Fingerprints Tree Generator**:
    - Automatically generates a fingerprints tree for a set of WAFs
- **WAF Distinguisher:**
    - Identifies a WAF using a set of fingerprints trees
- **Model Operations**:
    - Perform automata operations on stored models, input filters and grammars
- **Browser and WAF Differential Testing**:
    - Queries both Browser and WAF using a predefined set of strings

Built-in Rulesets Library

- **Regular Expressions**
    - Set of WAF filters, and attack models in the form of regular expressions
- **Grammars:**
    - Set of grammars that can be used for GOFA algorithm.
- **Fingerprints Trees**:
    - Set of fingerprints trees for a predefined number of WAFs.
- **Configurations**:
    - Sample configurations for WAF distinguish tree generation

Grub LightBulb:

https://github.com/lightbulb-framework/

Future Work

- Currently building many optimizations.
  - Learning will be much faster in the next months.
  - Cross checking models is also getting better.
- Incorporate fuzzers to improve models.
- New ideas?

Conclusions

- Current state of WAFs is still (very) ugly.
  - Many low hanging fruits.
- Our vision is to enforce a standard for such products.
  - WAFs must effectively defend against inferred language specifications.
  - Learning can run continuously with the assistance of fuzzers.
- We have a similar line of work on sanitizers.

Another Brick off The Wall: Deconstructing Web Application Firewalls Using Automata Learning

George Argyros, Ioannis Stais

Joint Work with:
Suman Jana, Angelos D. Keromytis, Aggelos Kiayias