SlowFuzz: Automated Domain-Independent Detection of Algorithmic Complexity Vulnerabilities

Theofilos Petsios, *Jason Zhao, Angelos D. Keromytis, and Suman Jana*

Columbia University

ACM Conference on Computer and Communications Security (CCS) 2017, Dallas, Texas

COMPLEXITY VULNERABILITIES

- *Difference between average and worst-case complexity*
  - *CPU, memory, space etc.*
  - *User-controlled*
  - *Exploitability & Denial of Service (DoS)*

- *Several instances seen in the wild*

COMPLEXITY VULNERABILITIES

- *Difference between average and worst-case complexity*
  - *CPU, memory, space etc.*
  - *User-controlled*
  - *Exploitability & Denial of Service (DoS)*

- *Several instances seen in the wild*

Stack Exchange Network Status

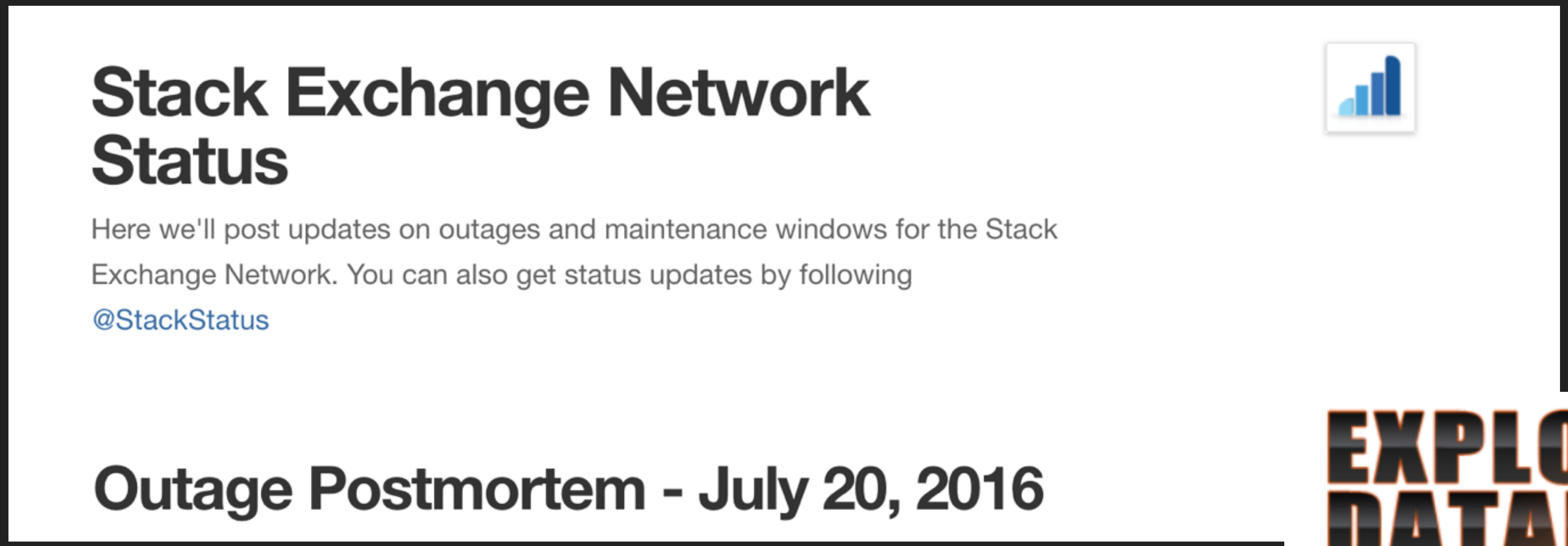
Here we'll post updates on outages and maintenance windows for the Stack Exchange Network. You can also get status updates by following @StackStatus

Outage Postmortem - July 20, 2016

COMPLEXITY VULNERABILITIES

- *Difference between average and worst-case complexity*
  - *CPU, memory, space etc.*
  - *User-controlled*
  - *Exploitability & Denial of Service (DoS)*

- *Several instances seen in the wild*



COMPLEXITY VULNERABILITIES

- *Difference between average and worst-case complexity*
    - *CPU, memory, space etc.*
    - *User-controlled*
    - *Exploitability & Denial of Service (DoS)*

- *Several instances seen in the wild*

Stack Exchange Network Status

Here we'll post updates on outages and maintenance windows for the Stack Exchange Network. You can also get status updates by following @StackStatus

Outage Postmortem - July 20, 2016

2009 30th IEEE Symposium on Security and Privacy

Exploiting Unix File-System Races via Algorithmic Complexity Attacks

EXPLOIT DATABASE

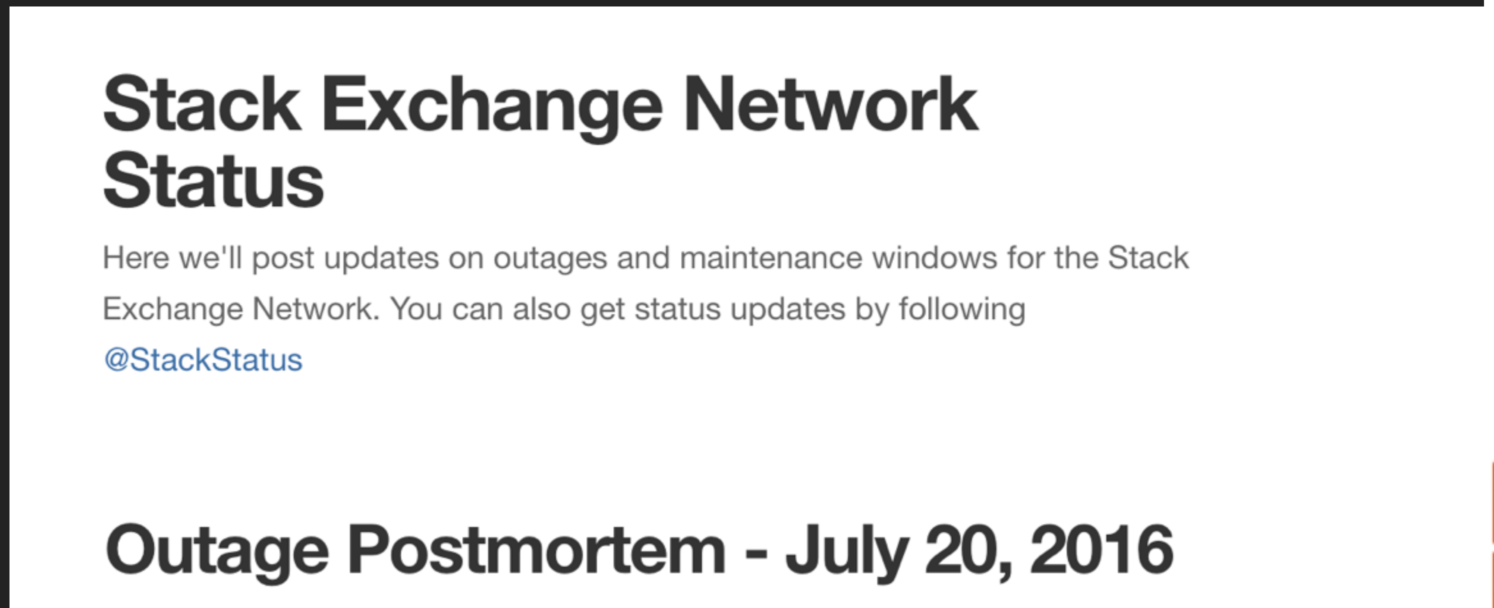
Home Exploits Shellcode Papers Google Hacking Database Subm

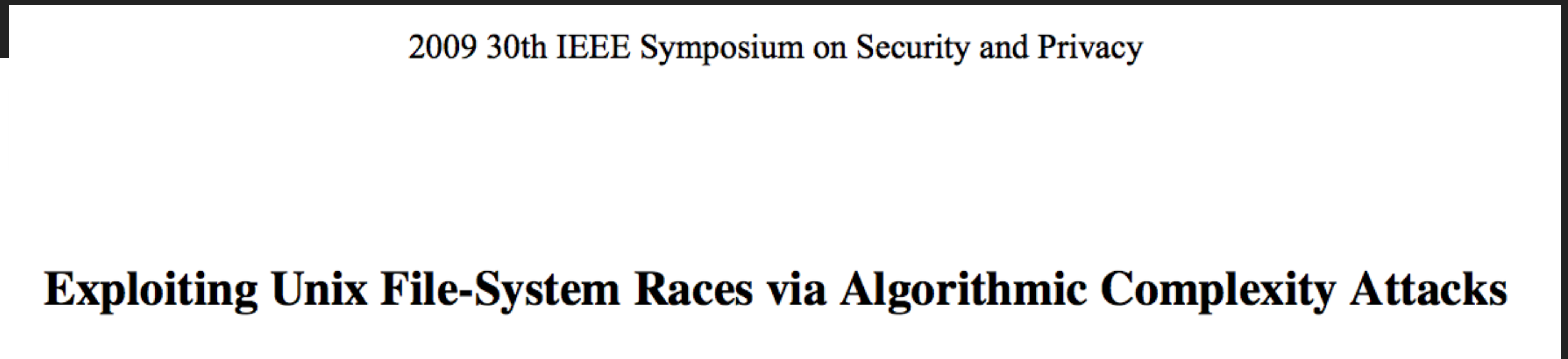
PHP Hash Table Collision - Denial of Service (PoC)

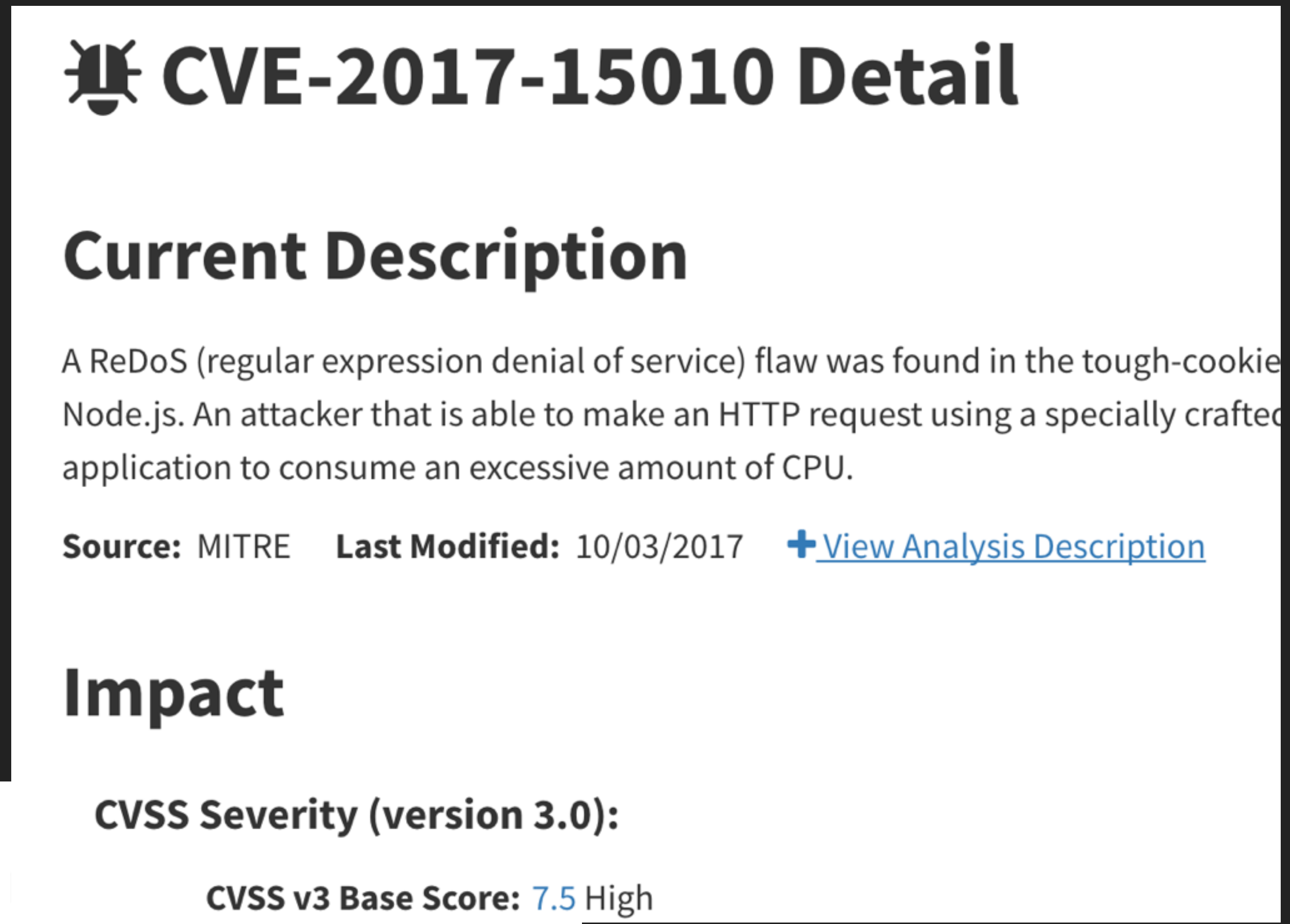
COMPLEXITY VULNERABILITIES

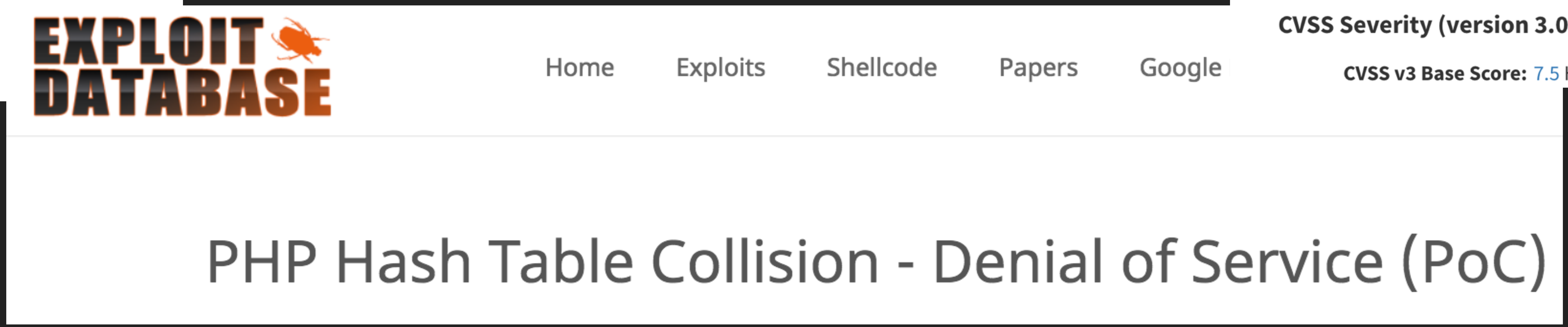
- *Difference between average and worst-case complexity*
  - *CPU, memory, space etc.*
  - *User-controlled*
  - *Exploitability & Denial of Service (DoS)*

- *Several instances seen in the wild*









DOMAIN INDEPENDENT DETECTION OF COMPLEXITY VULNERABILITIES

- *Heavily dependent on application logic*

- *Algorithmic worst-case vs implementation worst-case*
  - *Minor changes often drastically change complexity (e.g., pivot selection in quicksort)*

- *Reasoning about the problem in the generic case is hard:*
  - *Theoretical analysis is often non-trivial*
  - *Implementation varies*
  - *Domain-specific tools predominantly require expert knowledge*

EXAMPLE: QUICKSORT

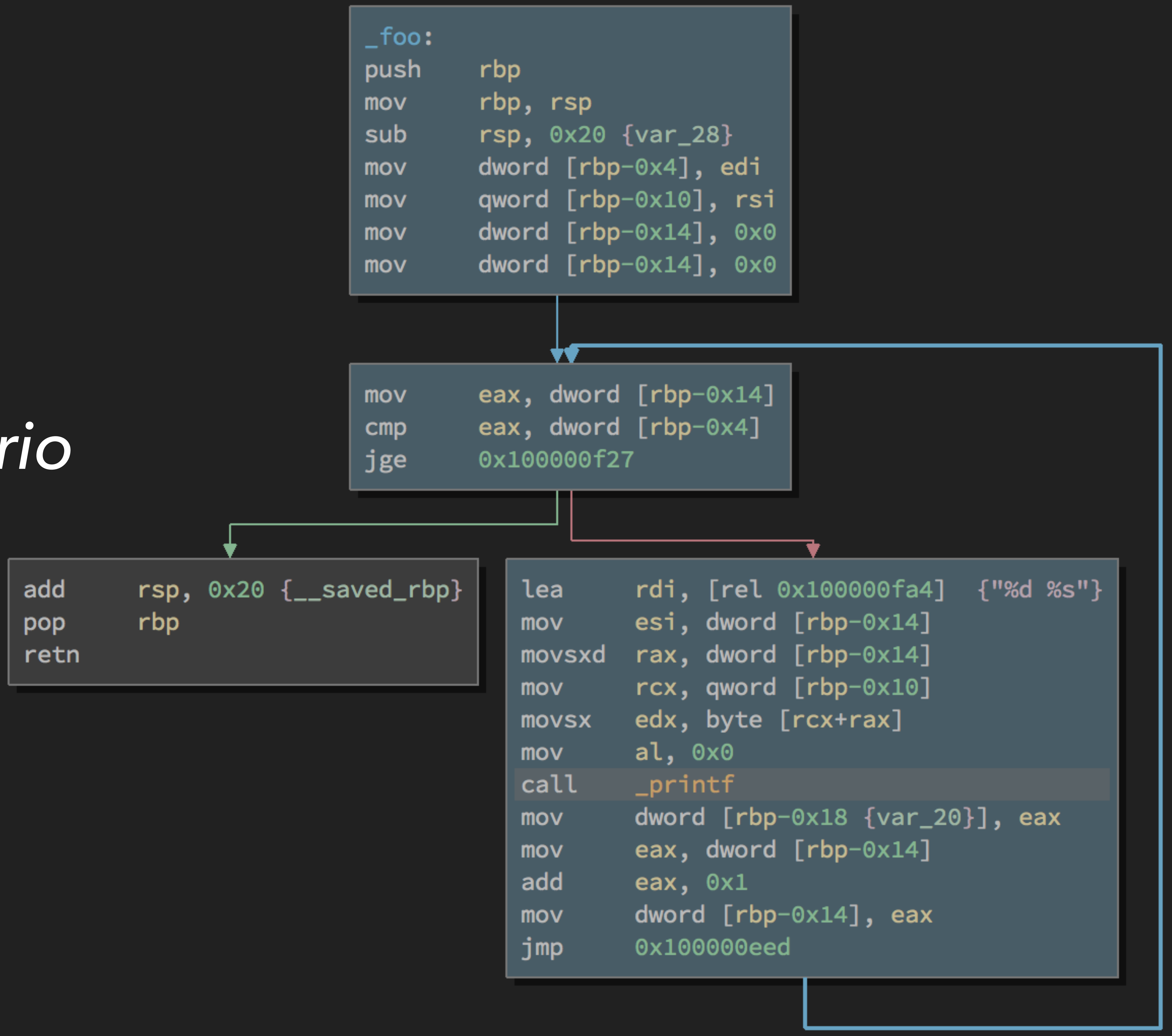
- *Average* $O(n\log n)$ *vs worst-case* $O(n^2)$ *complexity*
- *Implementation largely affects performance*
- *How do we reason on the effectiveness of a given implementation?*
- *How to test in a domain-agnostic manner?*

EVOLUTIONARY TESTING

- *Domain-independent test input generation*
- *Known to perform well in grey-box settings*
- *Very effective in modern fuzzers targeting crash/memory corruption bugs*
  - *No expert knowledge*
  - *Production tools compete with domain-specific engines*

EVOLUTIONARY TESTING

- *Can we steer evolutionary testing towards complexity bugs?*
- *Coverage is* **irrelevant** *in this scenario*
- *Re-use fuzzing infrastructure*



SLOWFUZZ PROTOTYPE

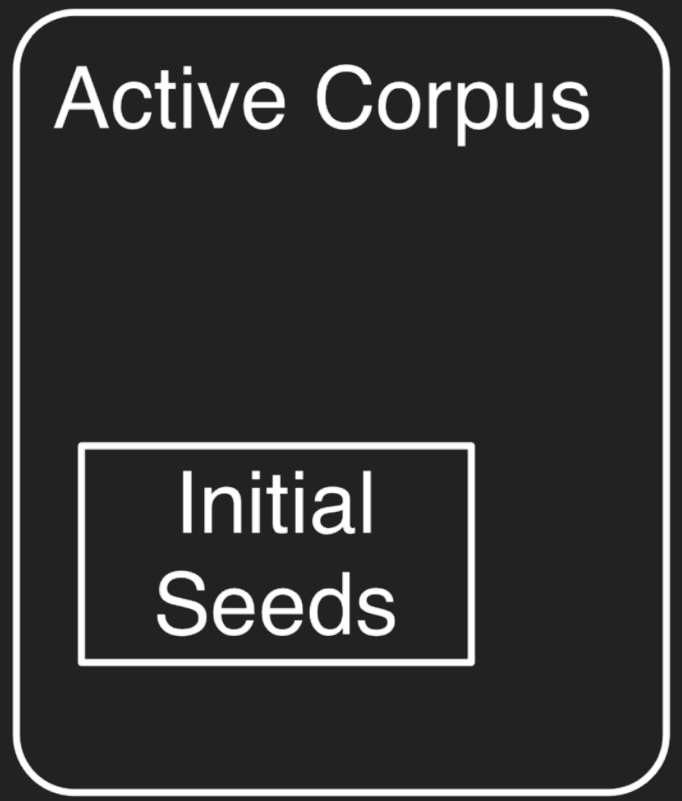
- *SlowFuzz prototype*
- *Maintain and evolve an input corpus towards slower executions*

SLOWFUZZ PROTOTYPE

- *Maintain and evolve an input corpus towards slower executions*

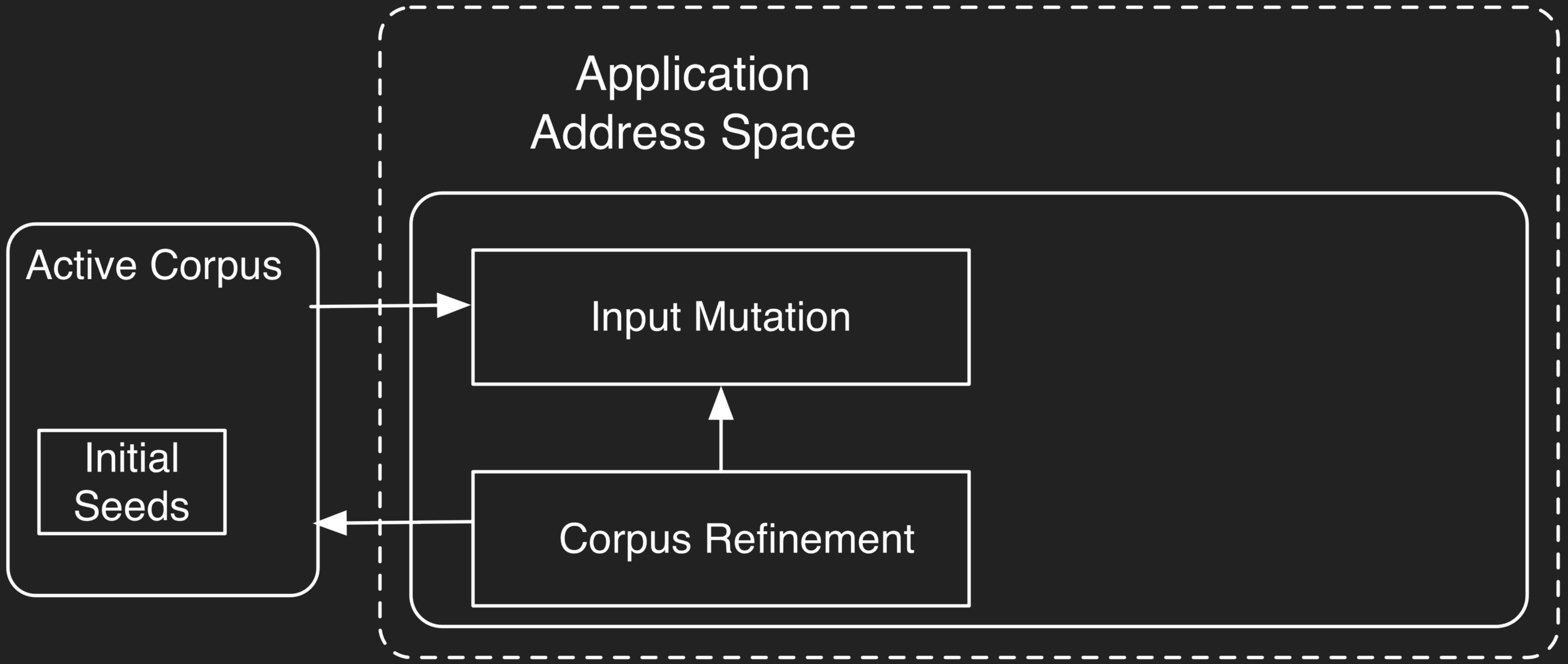
SLOWFUZZ PROTOTYPE

- *Maintain and evolve an input corpus towards slower executions*



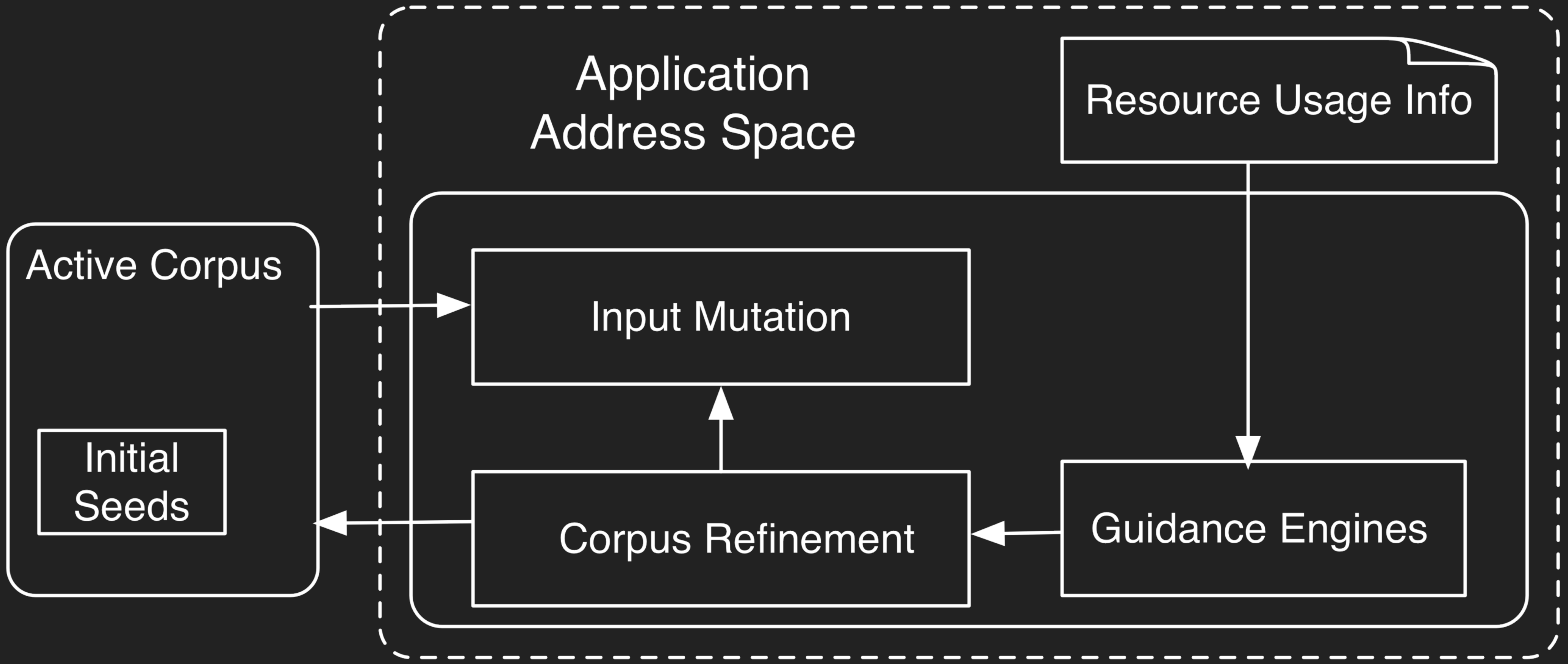
SLOWFUZZ PROTOTYPE

- *Maintain and evolve an input corpus towards slower executions*



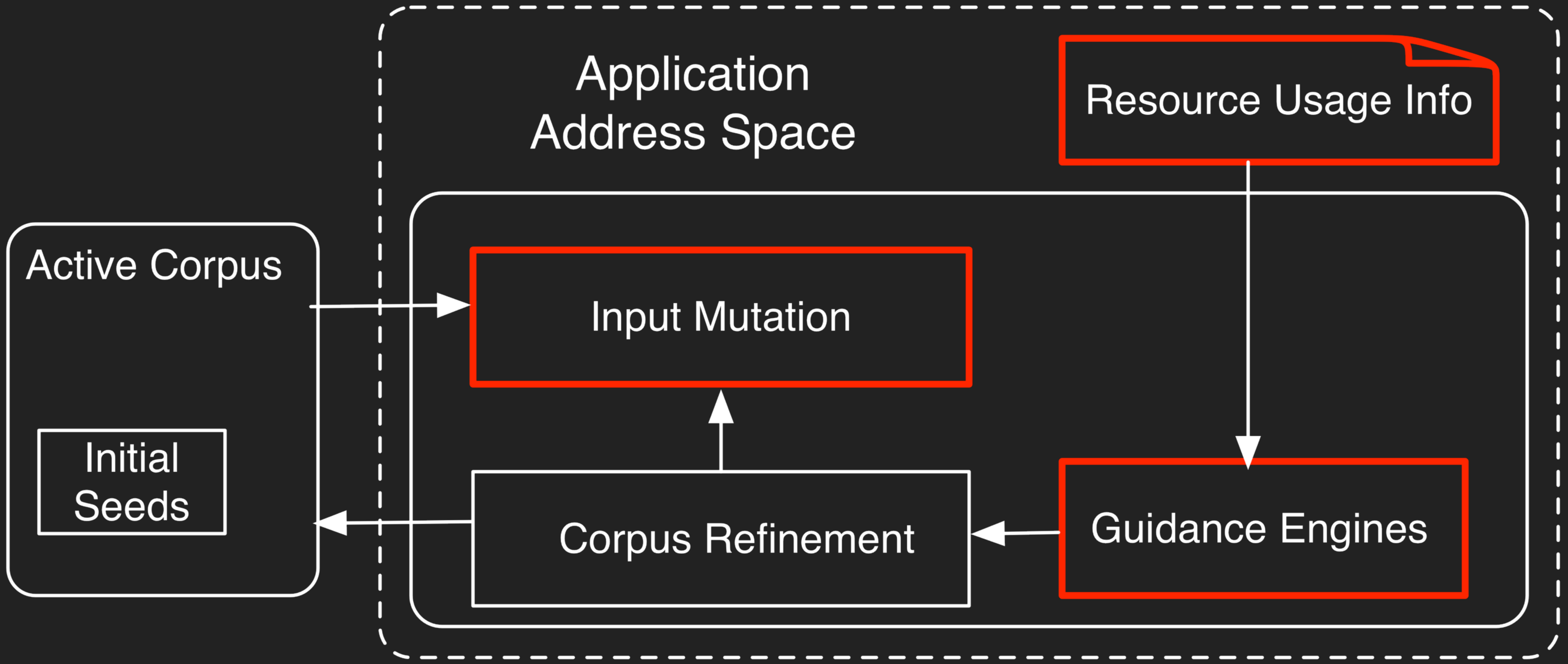
SLOWFUZZ PROTOTYPE

- *Maintain and evolve an input corpus towards slower executions*



SLOWFUZZ PROTOTYPE

- *Maintain and evolve an input corpus towards slower executions*



SLOWFUZZ KEY IDEAS

- *Three key controls:*
  - *Instrumentation, Fitness Function, Mutations*
- *Fitness Function should favor inputs that introduce slowdowns*
- *Mutation operations with locality in mind*
- *Avoid getting stuck!*

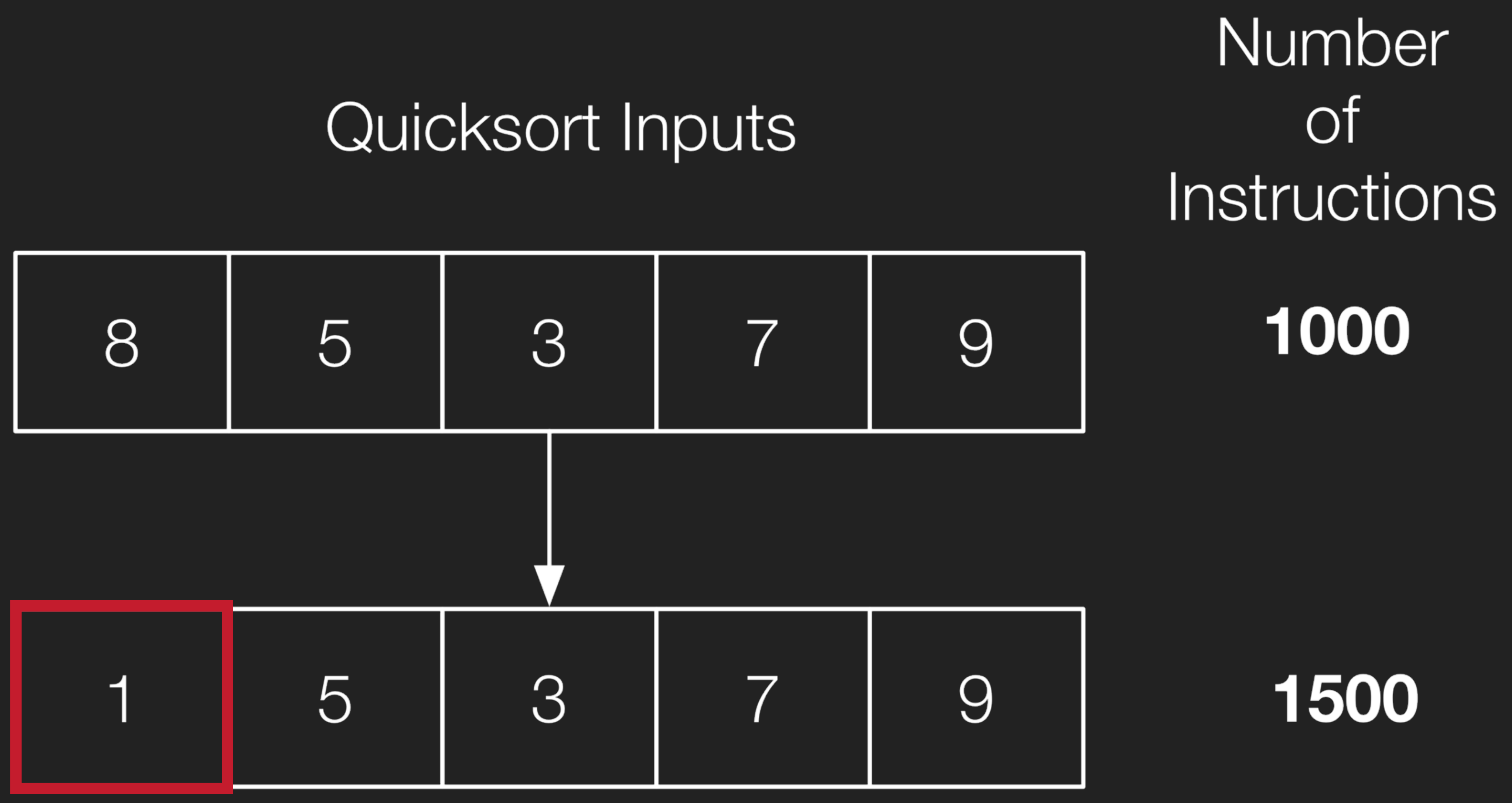
SLOWFUZZ KEY IDEAS

- *Three key controls:*
  - *Instrumentation, Fitness Function, Mutations*
- *Fitness Function should favor inputs that introduce slowdowns*
- *Mutation operations with locality in mind*
- *Avoid getting stuck!*

Quicksort Inputs					Number of Instructions
8	5	3	7	9	**1000**

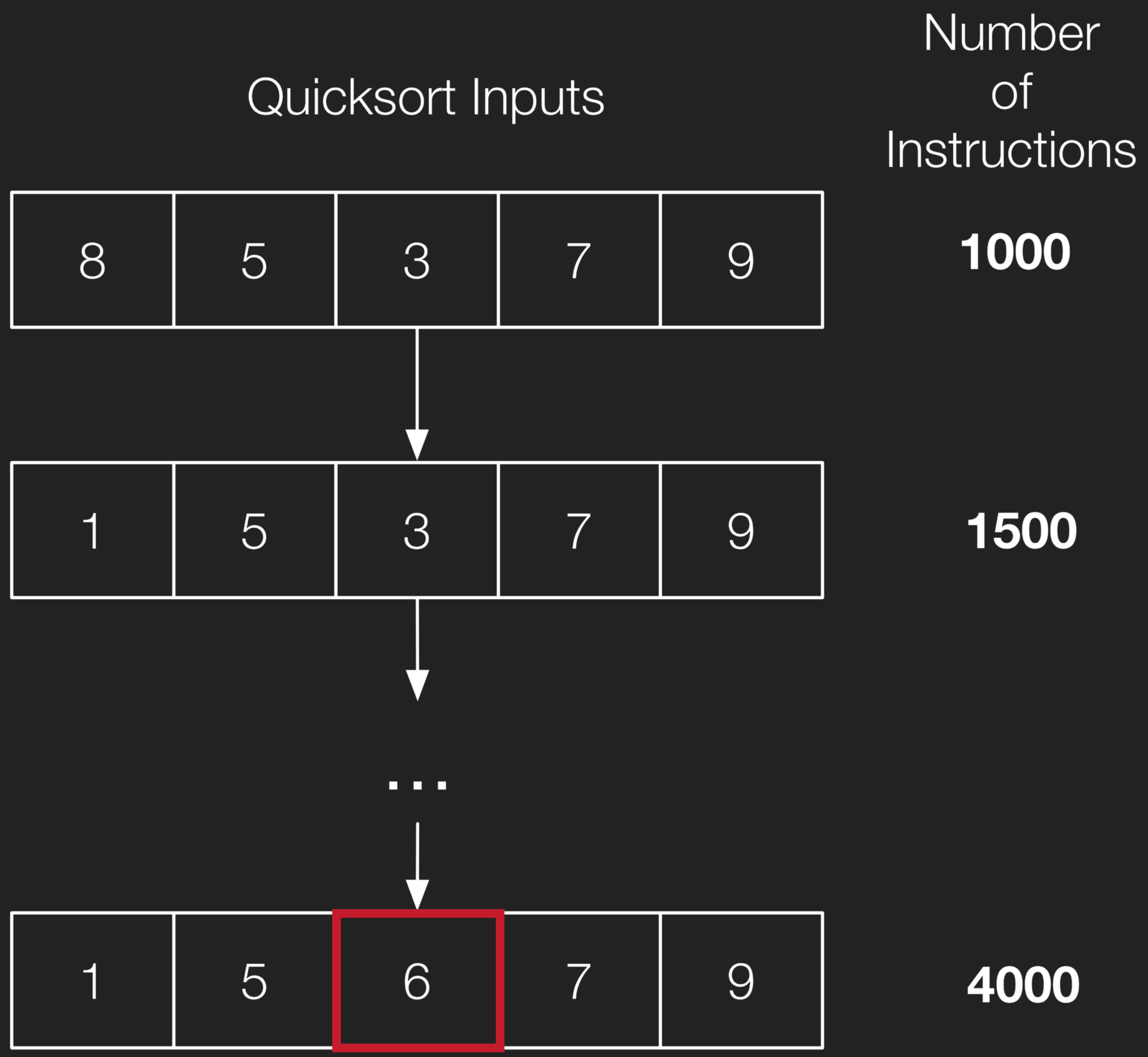
SLOWFUZZ KEY IDEAS

- *Three key controls:*
  - *Instrumentation, Fitness Function, Mutations*
- *Fitness Function should favor inputs that introduce slowdowns*
- *Mutation operations with locality in mind*
- *Avoid getting stuck!*



SLOWFUZZ KEY IDEAS

- *Three key controls:*
  - *Instrumentation, Fitness Function, Mutations*
- *Fitness Function should favor inputs that introduce slowdowns*
- *Mutation operations with locality in mind*
- *Avoid getting stuck!*

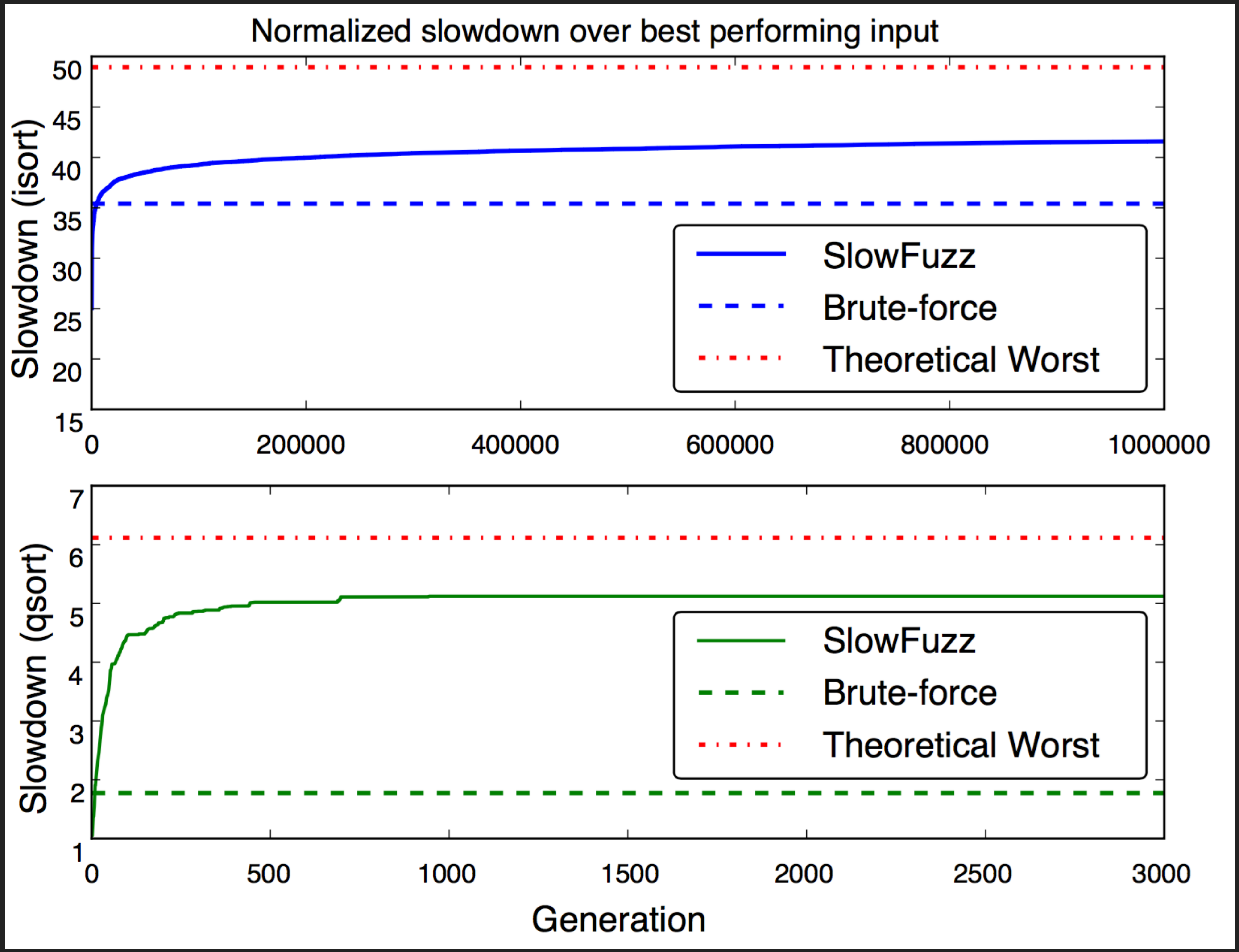


SLOWFUZZ KEY IDEAS

- *Fitness function maximizes CPU instructions*
- *Mutation Strategies:*
    - *Random*
    - *Offset Priority*
    - *Mutation Priority*
    - *Hybrid*

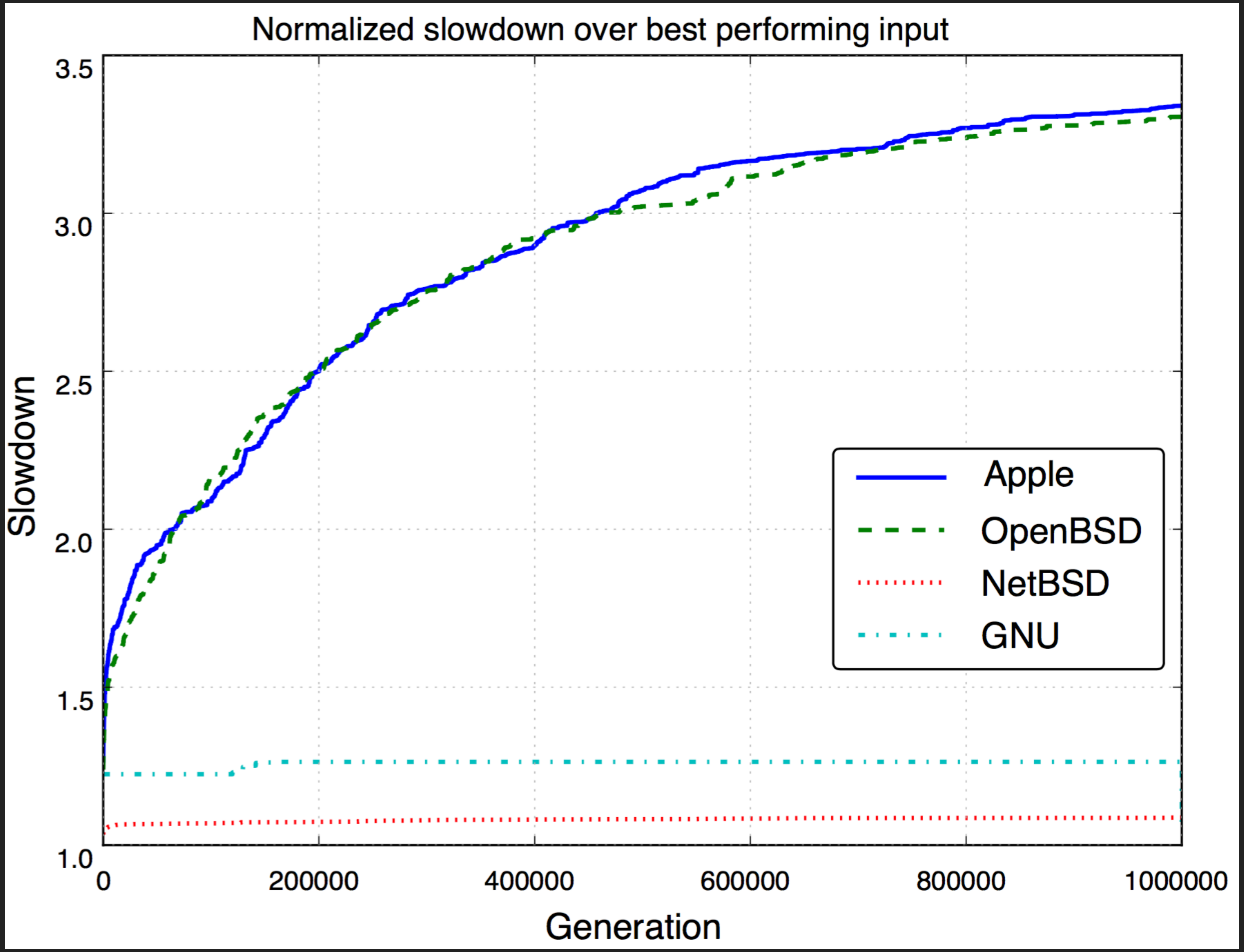
USECASE: SORTING

- *Insertion sort & quicksort implementations*
- *Quadratic worst-case performance*
- *How close do we get to the theoretical worst slowdown?*
- *Slowdowns of 84.97% and 83.74% of theoretical worst-case*



USECASE: SORTING / REAL WORLD EXAMPLES

- *Apple:3.34x*
- *OpenBSD: 3.3x*
- *GNU: 26.36%*
- *NetBSD: 8.7%*

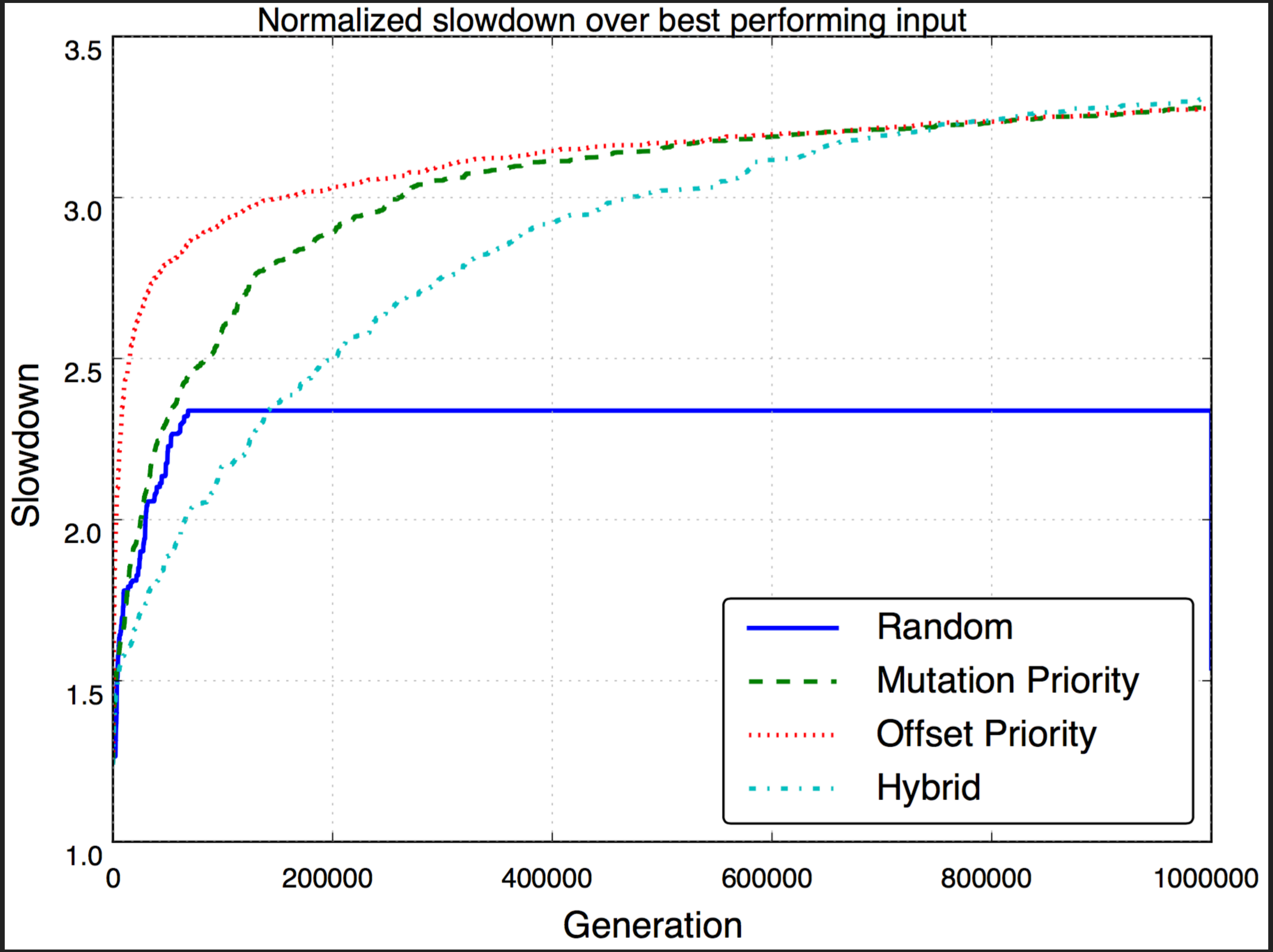


ENGINE PROPERTIES

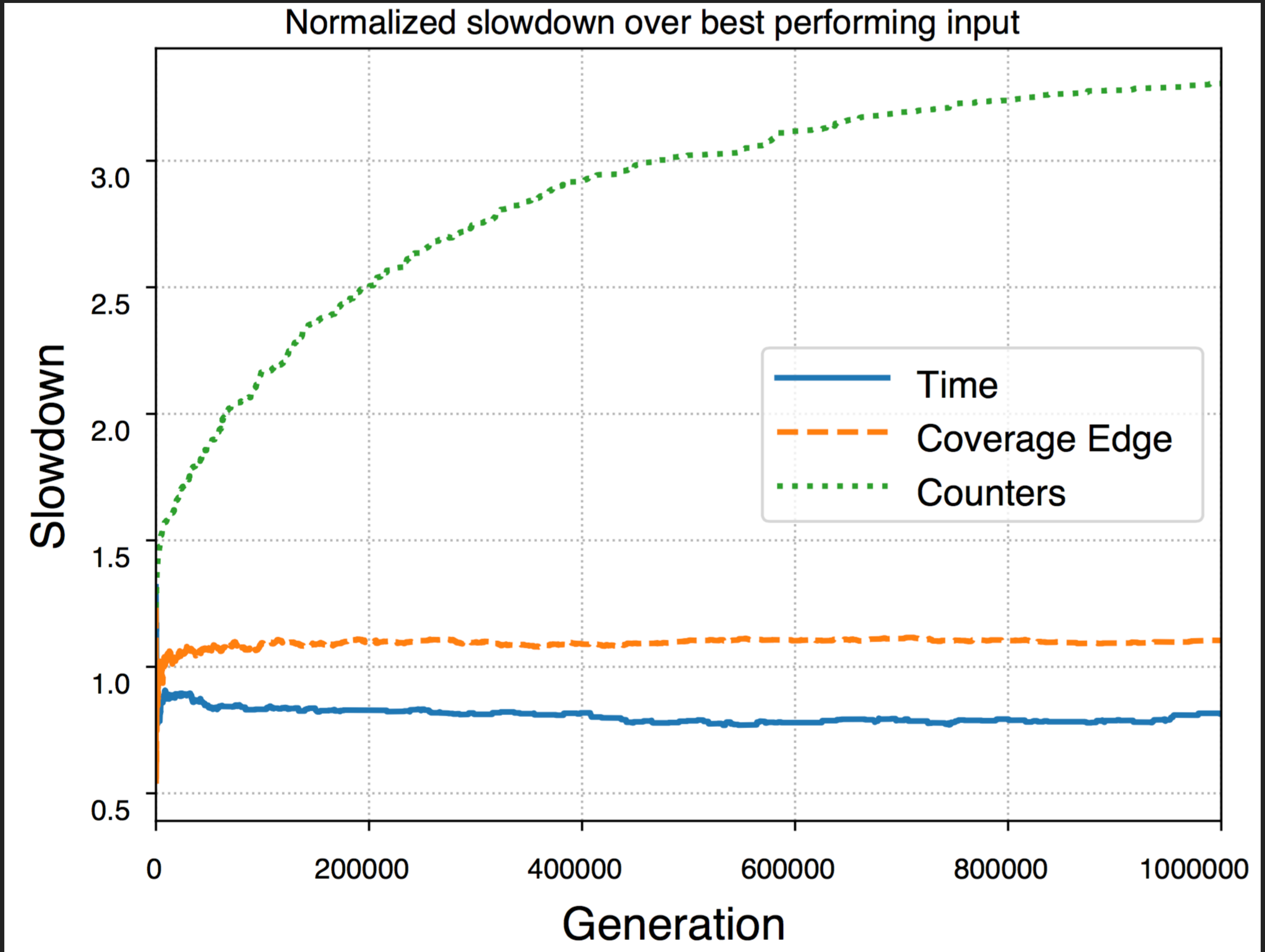
- *Fitness function:*
  - *CPU instructions vs Code Coverage vs Time-based tracing*

- *Mutation Strategies:*
  - *Random*
  - *Offset Priority*
  - *Mutation Priority*
  - *Hybrid*

ENGINE EVALUATION / MUTATION STRATEGIES – OPENBSD QUICKSORT



ENGINE EVALUATION / FITNESS FUNCTIONS - OPENBSD QUICKSORT



EVALUATION

- *Evolutionary testing for complexity bugs is promising*
- *Testcases: common instances of complexity vulnerabilities*
  - *Hashtables*
  - *Regular Expression Parsers*
  - *Compression/decompression routines*

USECASE: PHP'S DJBX33A HASH

- *Hash used for string keys in PHP*
- *Known worst-case performance*
- *Has been exploited in the wild*
- *For 'ab', 'cd' to collide it must hold*

  $c = a + n \wedge d = b - 33 * n, n \in Z$

- *If if two equal-length strings A and B collide, then strings xAy, xBy also collide*

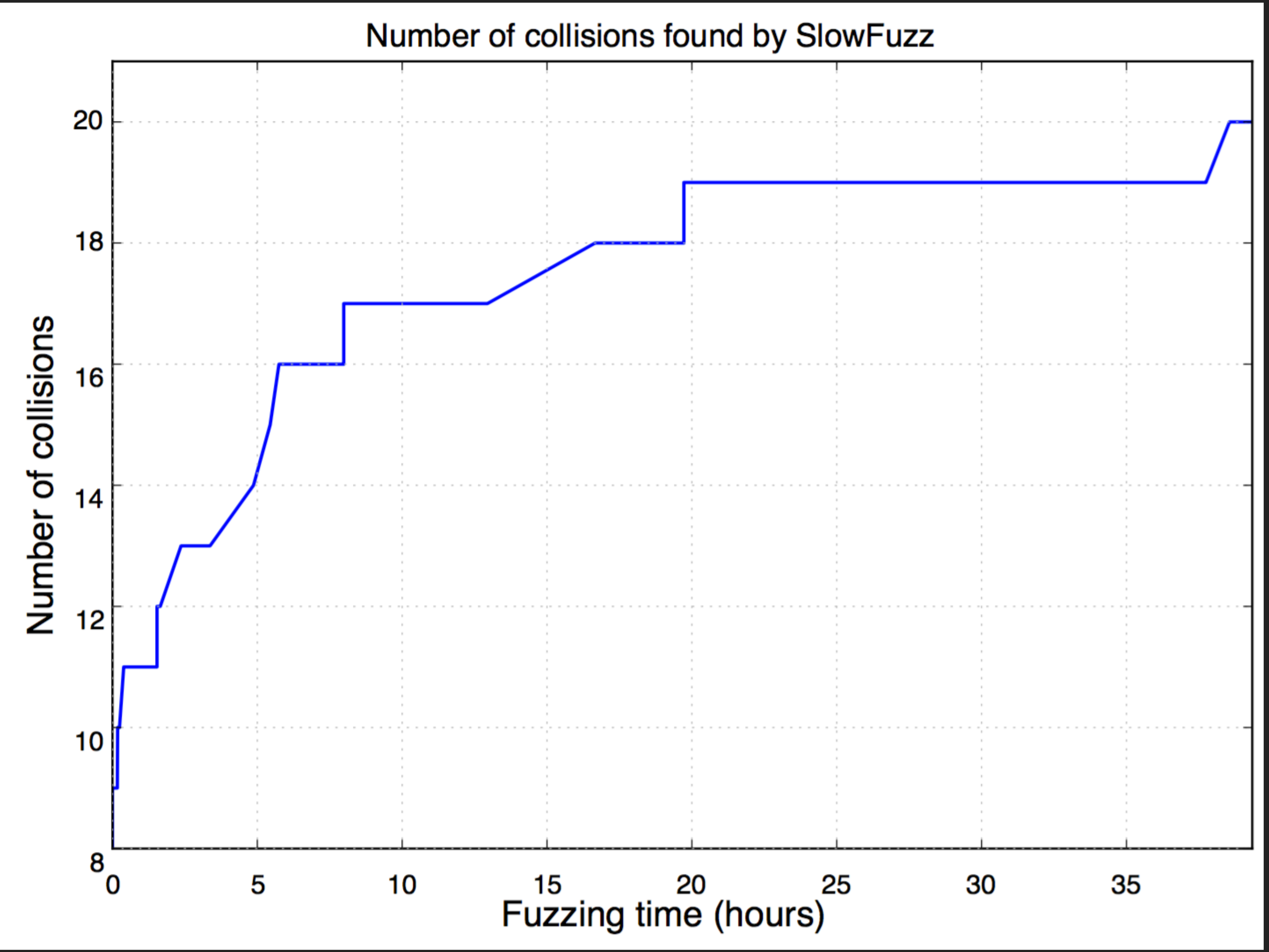
```c
/*
 * @arKey is the array key to be hashed
 * @nKeyLenth is the length of arKey
 */
static inline ulong
zend_inline_hash_func(const char *arKey, uint
      nKeyLength)
{
     register ulong hash = 5381;

     for (uint i = 0; i < nKeyLength; ++i) {
          hash = ((hash << 5) + hash) + arKey[i];
     }

     return hash;
}
```

USECASE: PHP'S DJBX33A HASH

- *64 hashtable entries & 64 insertions*
- *Slowfuzz generated inputs causing monotonically increasing collisions*
- *No knowledge of the internals of the hash function*



USECASE: REGEX PARSERS

- *Multiple instances of ReDoS in the wild*
- *Backtracking can be catastrophic*
- *Handling of both regexes and inputs*
    - *Evil Regexes*
    - *Slowdowns on given inputs*
- *Identifying evil regexes is a hard problem*
    - *Widely varying complexity: linear to exponential*
    - *Focus on super-linear & exponential matching*

```
regex_match(regex, string)
```

USECASE: REGEX PARSERS / PCRE

- *Can SlowFuzz find evil regexes given a fixed input?*

USECASE: REGEX PARSERS / PCRE

- *Can SlowFuzz find evil regexes given a fixed input?*
    - *Yes! Without any knowledge of the regex logic*

USECASE: REGEX PARSERS / PCRE

- *Can SlowFuzz find evil regexes given a fixed input?*
  - *Yes! Without any knowledge of the regex logic*

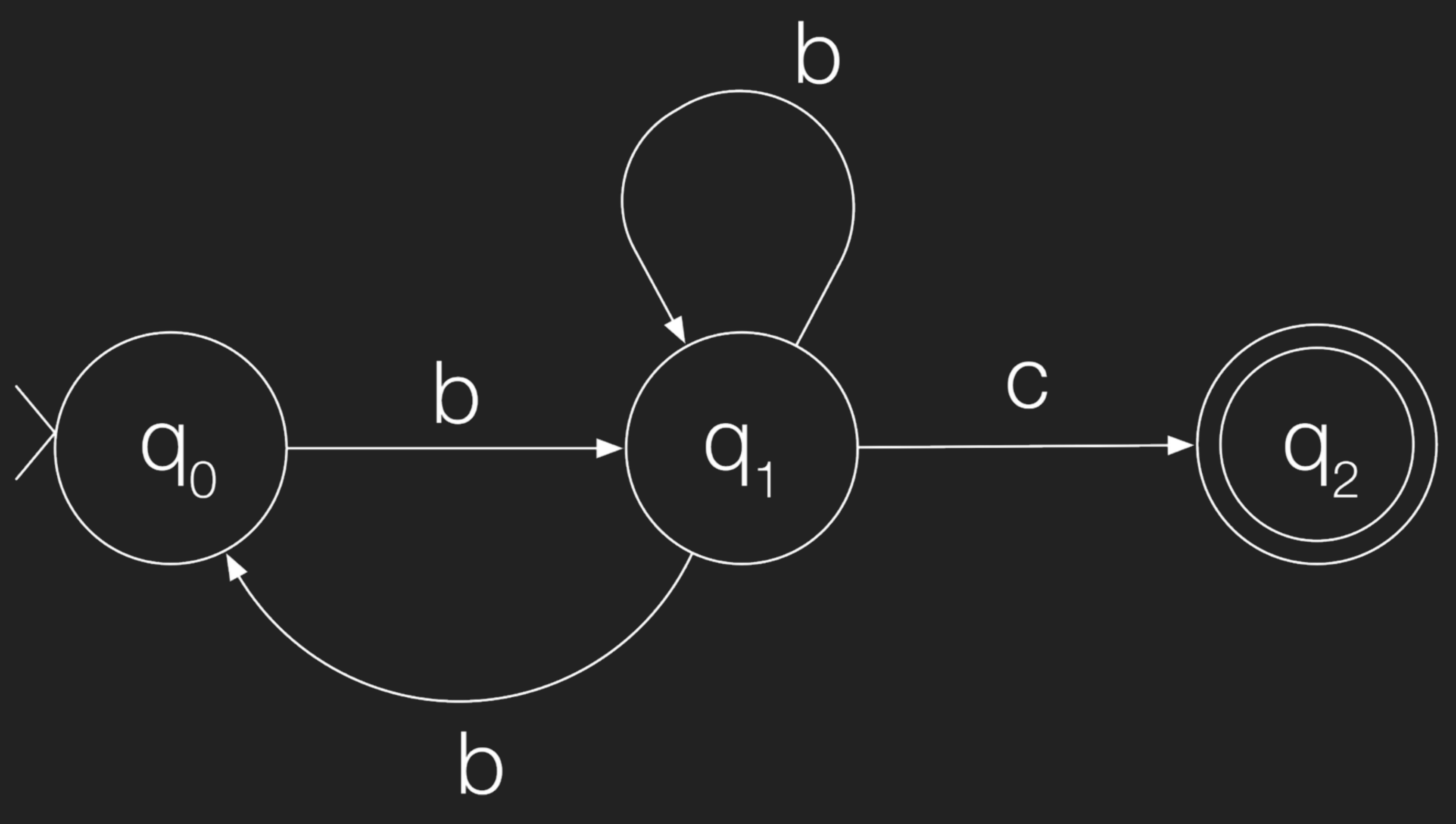
Super-linear	Exponential
c*ca*b*a*b	(b+)+c
a+b+b+b+a+	c*(b+b)+c
c*c+ccbc+	a(ala*)+a

USECASE: REGEX PARSERS / PCRE

- *Can SlowFuzz find evil regexes given a fixed input?*
  - *Yes! Without any knowledge of the regex logic*

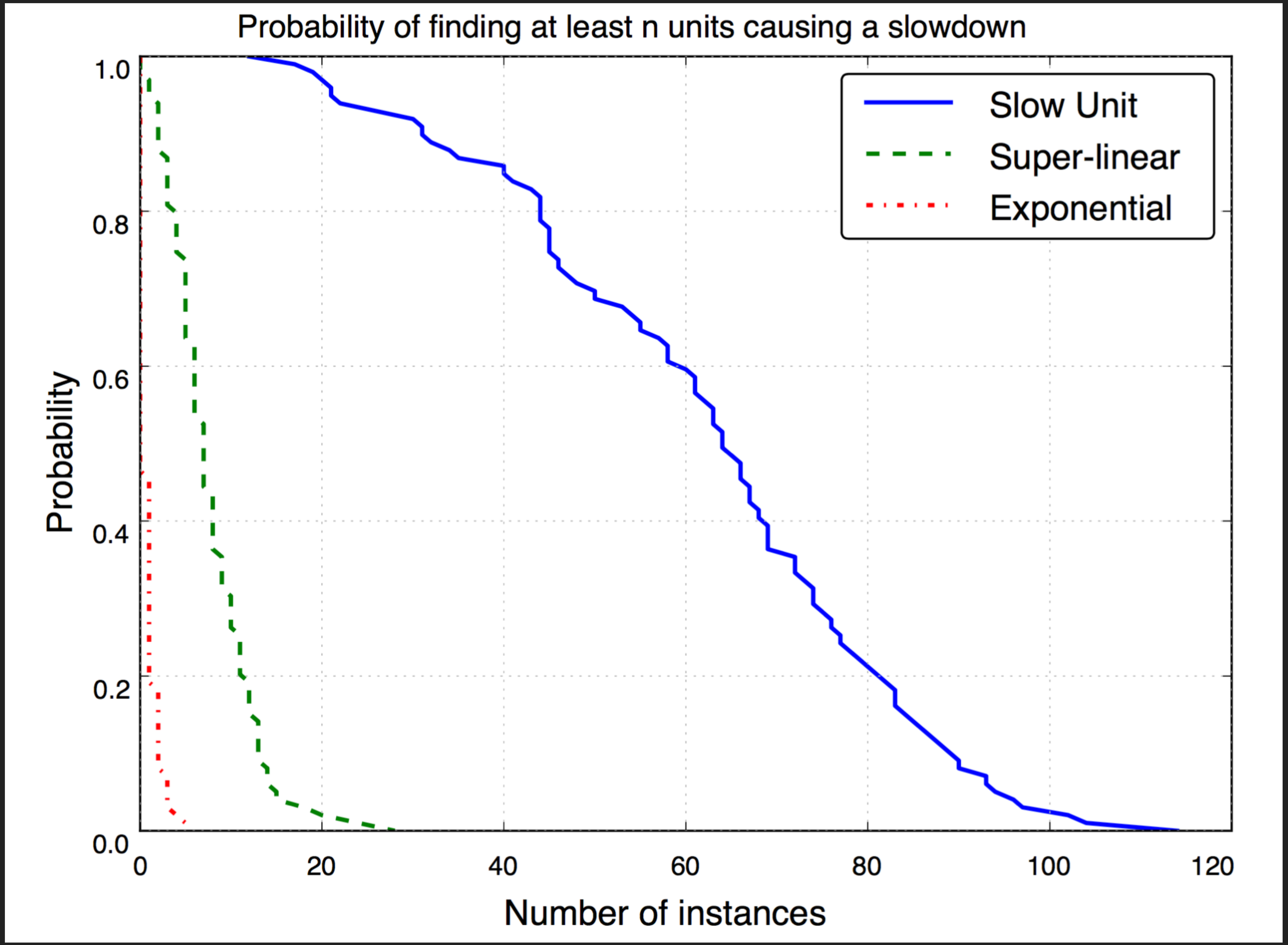
Super-linear	Exponential
c*ca*b*a*b	(b+)+c
a+b+b+b+a+	c*(b+b)+c
c*c+ccbc+	a(ala*)+a

- *Example:* (b+)+c



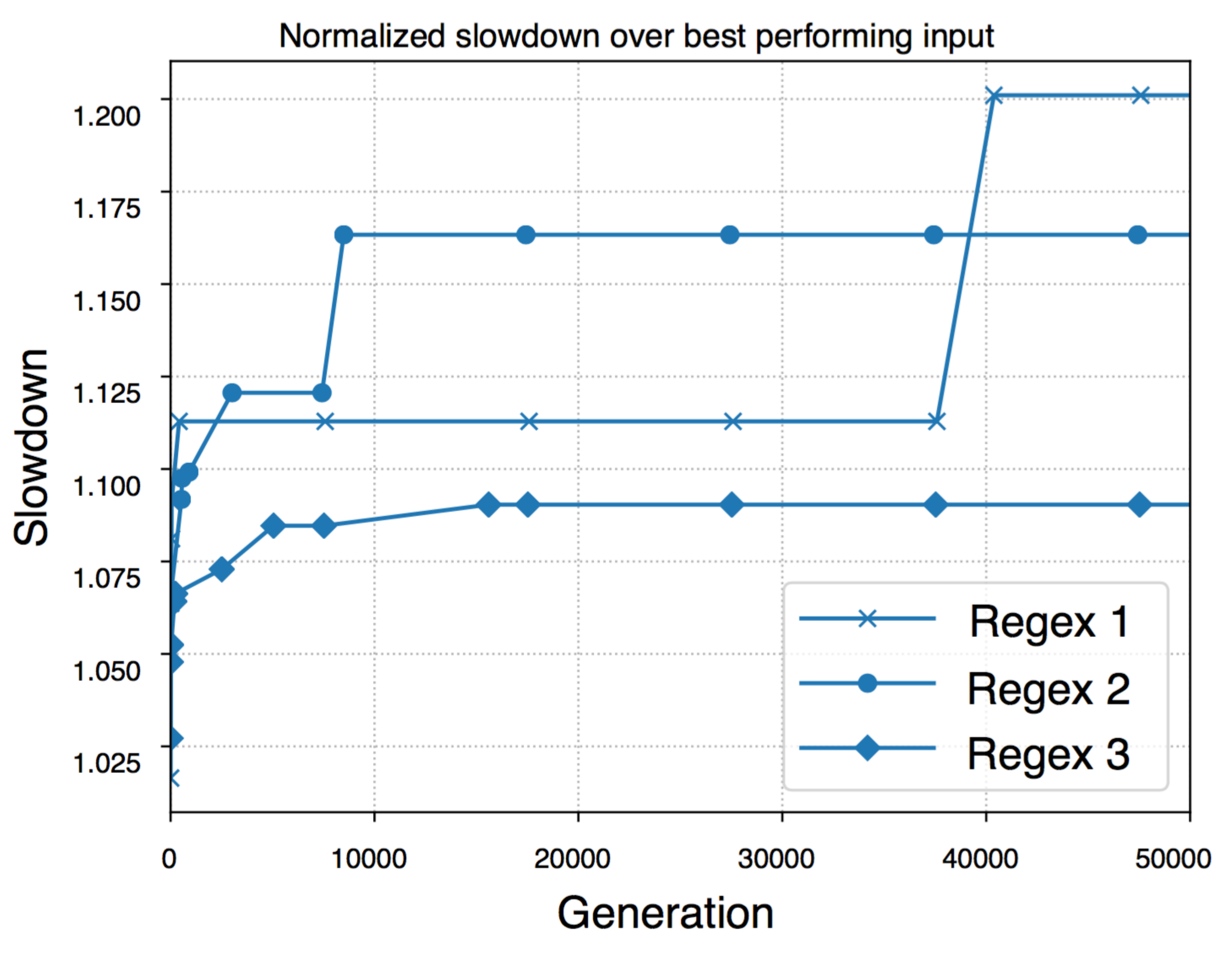
USECASE: REGEX PARSERS / PCRE

- *100 runs / 1 million generation each*
- *Regexes of 10 characters or less*
- *At least 31 regexes causing a slowdown with 90% probability*
- *At least 2 regexes with super-linear matching with 90% probability*
- *At least 1 regex with exponential matching with 45.45% probability*



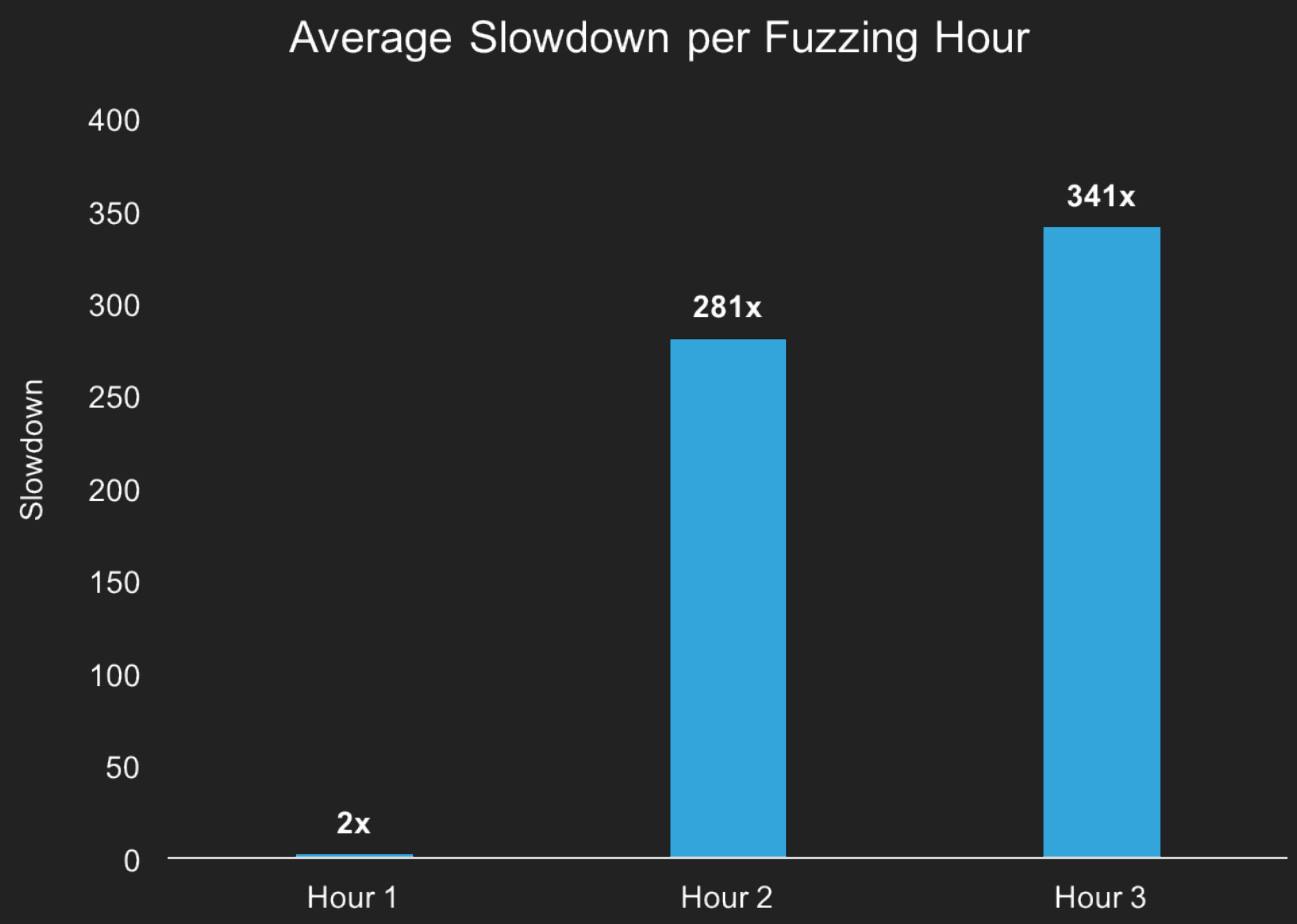
USECASE: REGEX PARSERS / PCRE

- *Can SlowFuzz find inputs causing a slowdown on a fixed regex?*
    - *Regexes from production WAFs*
    - *8 - 25% slowdowns*



USECASE: DECOMPRESSION / BZIP

- *bzip2*
- *250-byte inputs*
- *300x slowdown on fixed input size*



CONCLUSION

- *SlowFuzz: automated detection of complexity bugs through fuzzing*

- *Found non-trivial issues involving high performant code*
  - *PHP's hashtable implementation*
  - *PCRE regular expression library*
  - *bzip2*

- *Evolutionary fuzzing as a generic means of code exploration*
  - *Different objectives for different bug types*
  - *Beyond code coverage maximization*
  - *Objective vs Controls: Instrumentation, Fitness Functions, Mutations*