NEZHA: Efficient Domain-Independent Differential Testing

Theofilos Petsios*, Adrian Tang*,
Salvatore Stolfo, Angelos D. Keromytis, and Suman Jana

IEEE Security & Privacy (Oakland) 2017
**Joint primary authors*

Columbia University

Differential Testing

Differential Testing

- Fuzzing: memory corruption bugs
- Differential testing: logic bugs

Differential Testing

- Multiple apps of the same functionality
- Applications usually follow some specification/standard

Differential Testing

- Multiple apps of the same functionality
- All usually to follow some specification/standard
- Deviations from the specifications/standards likely to be bugs

Differential Testing

- Multiple apps of the same functionality
- All usually to follow some specification/standard
- Deviations from the specifications/standards likely to be bugs

Differential Testing

- Multiple apps of the same functionality
- All usually to follow some specification/standard
- Deviations from the specifications/standards likely to be bugs
- Applicable in different domains (e.g., compiler testing)

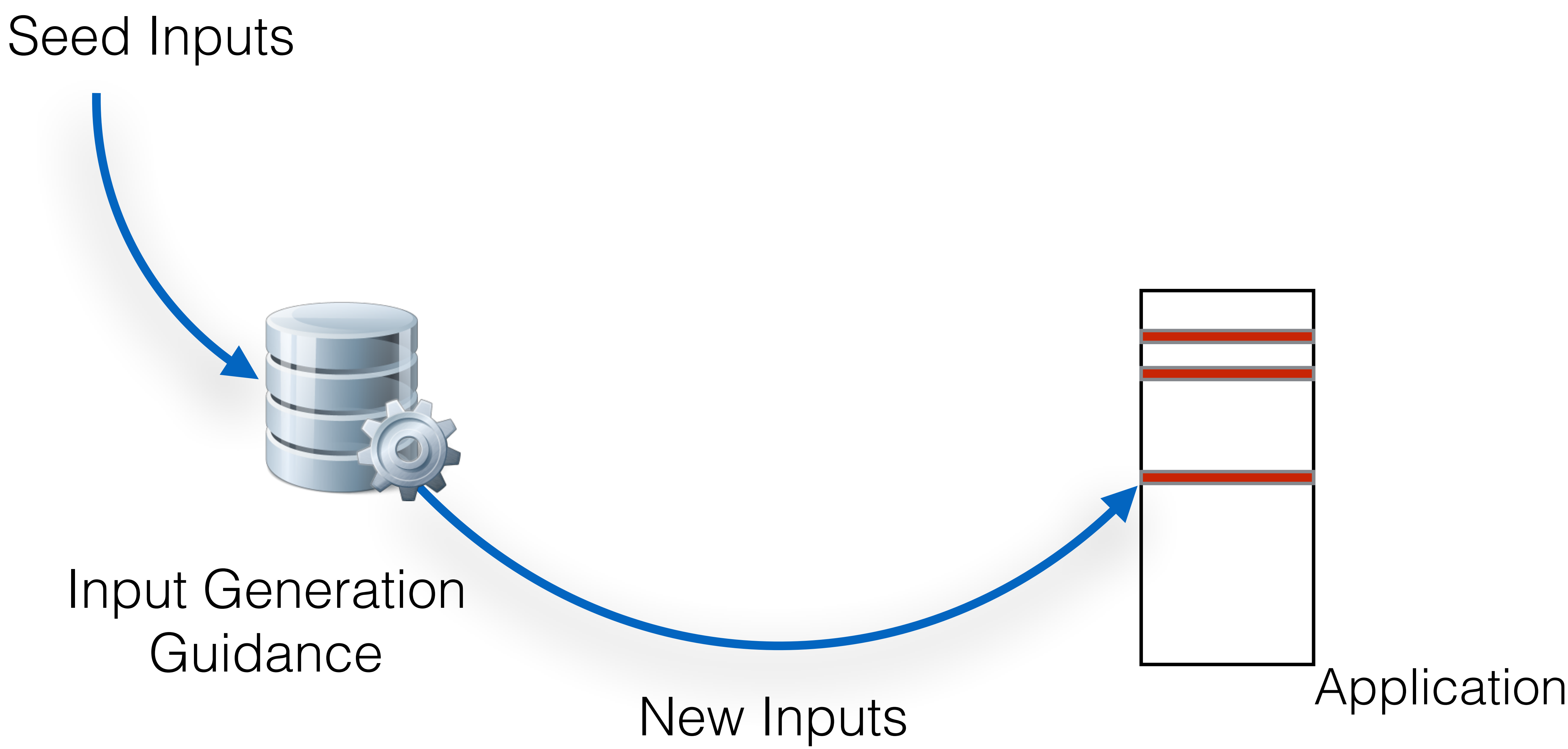
Key challenges

- Existing tools are domain-specific
- Inefficient input generation

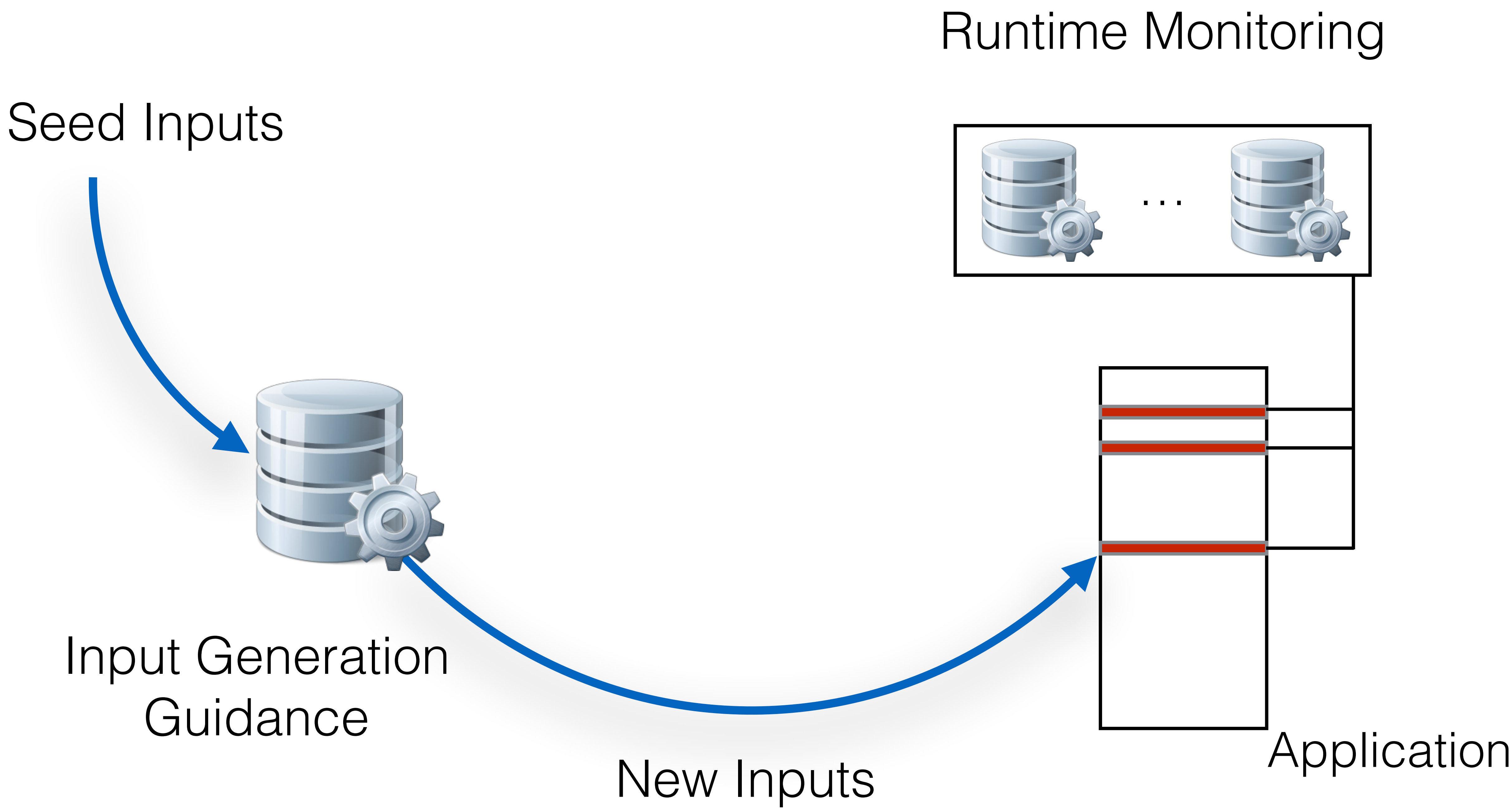
Goal of NEZHA

Efficient domain-independent differential testing

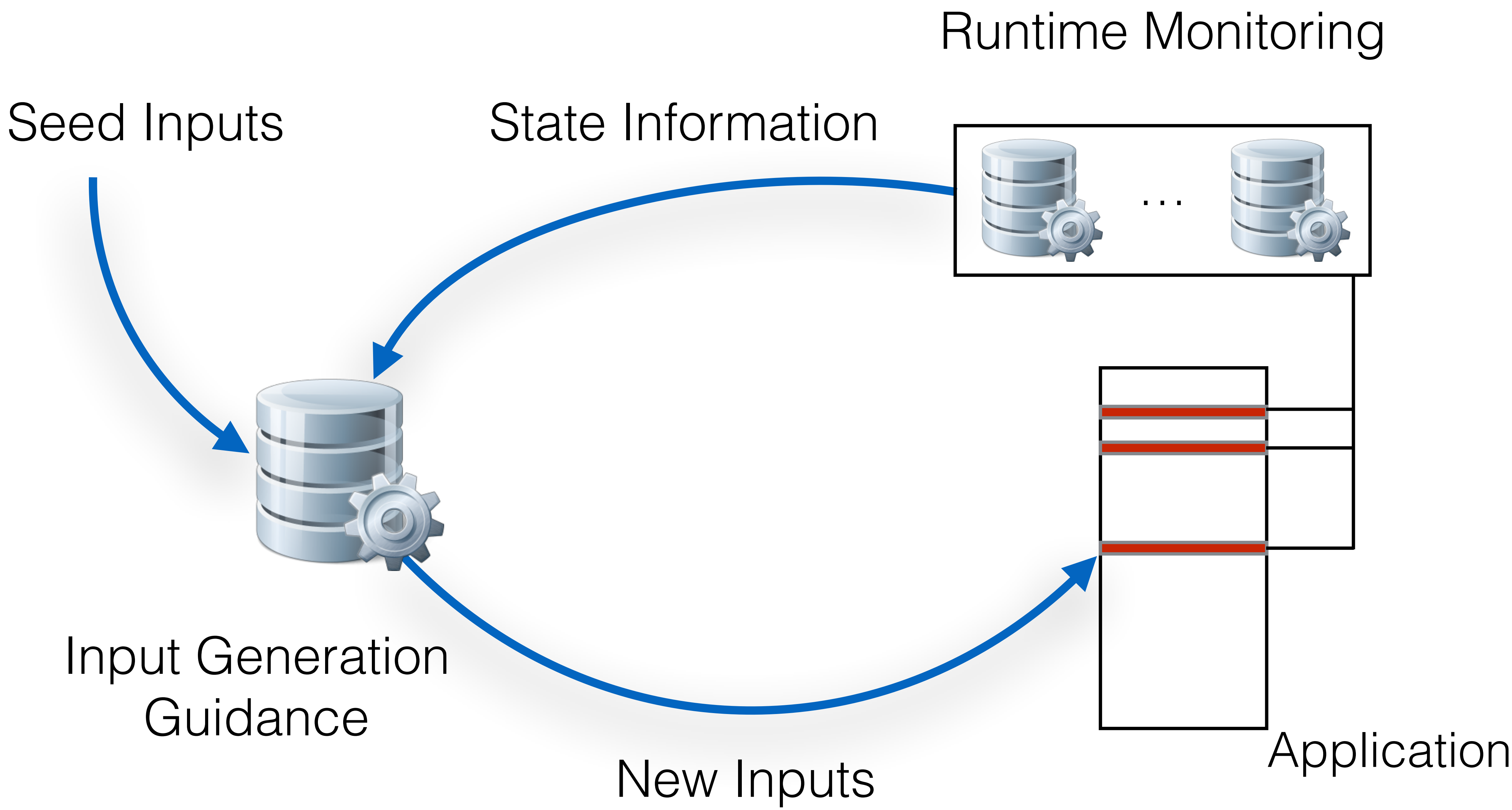
Domain-Independent Evolutionary Testing



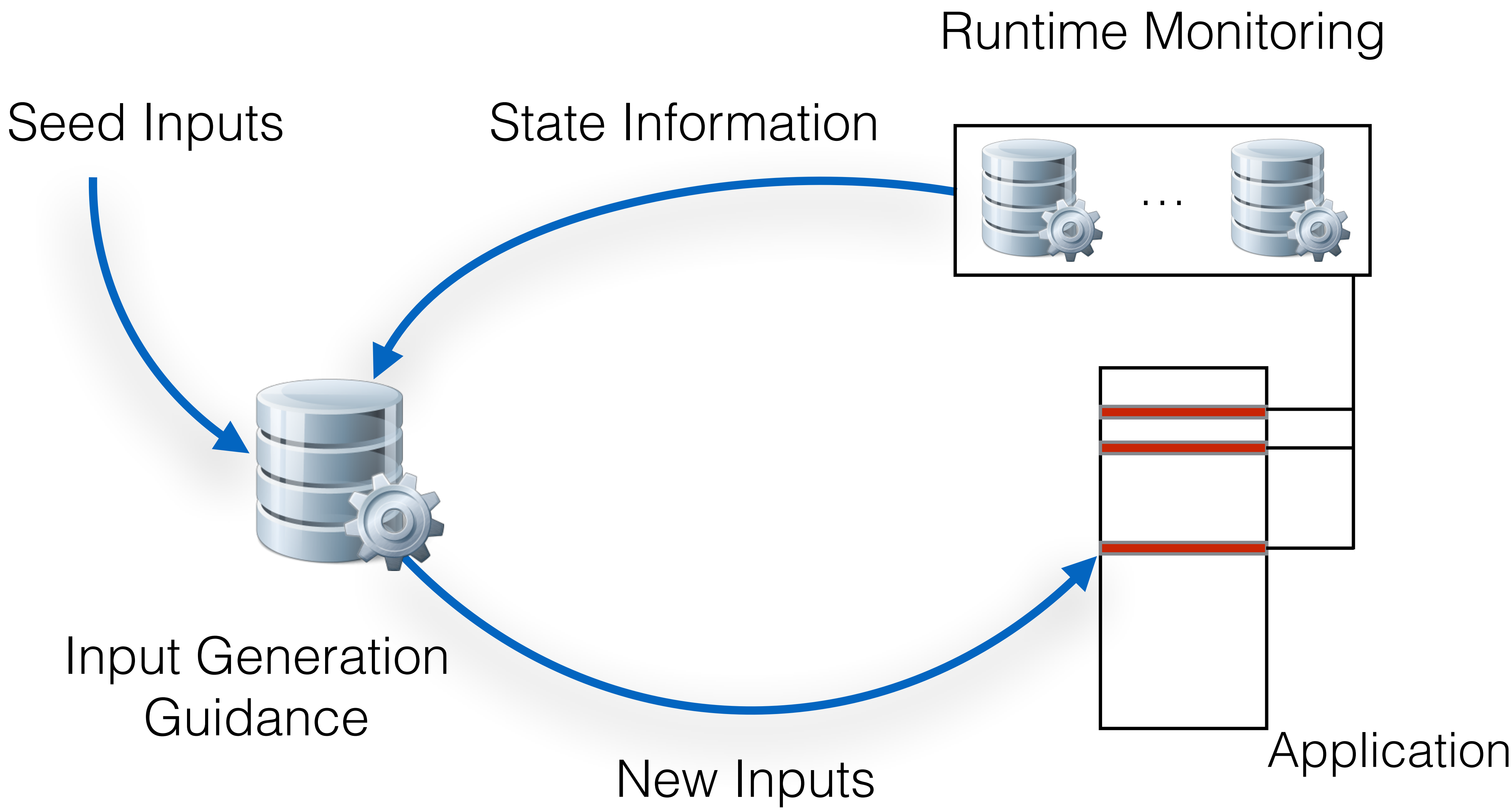
Domain-Independent Evolutionary Testing



Domain-Independent Evolutionary Testing

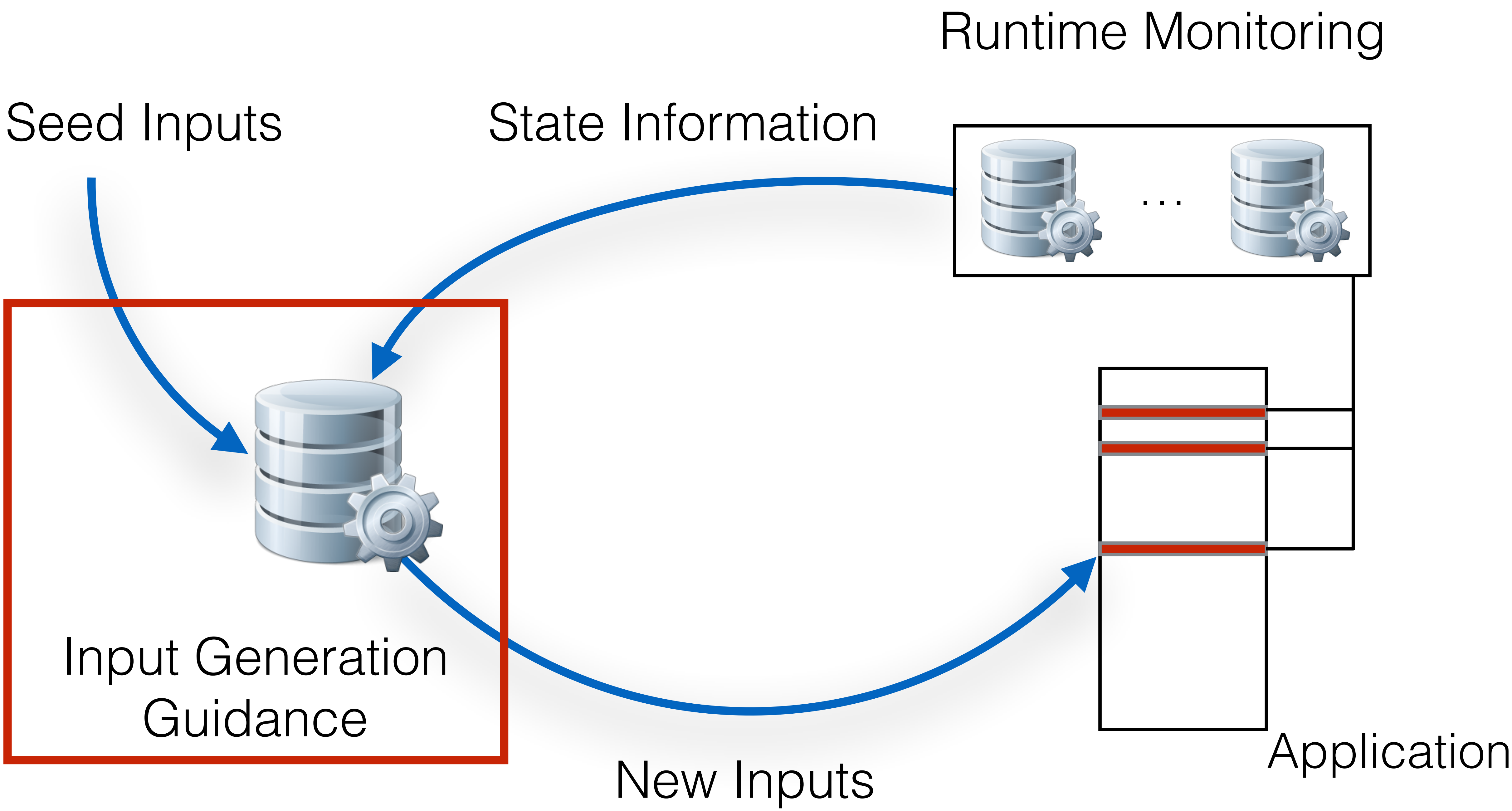


Domain-Independent Evolutionary Testing



Evolve an input corpus that is guided based on an analysis engine

Domain-Independent Evolutionary Testing

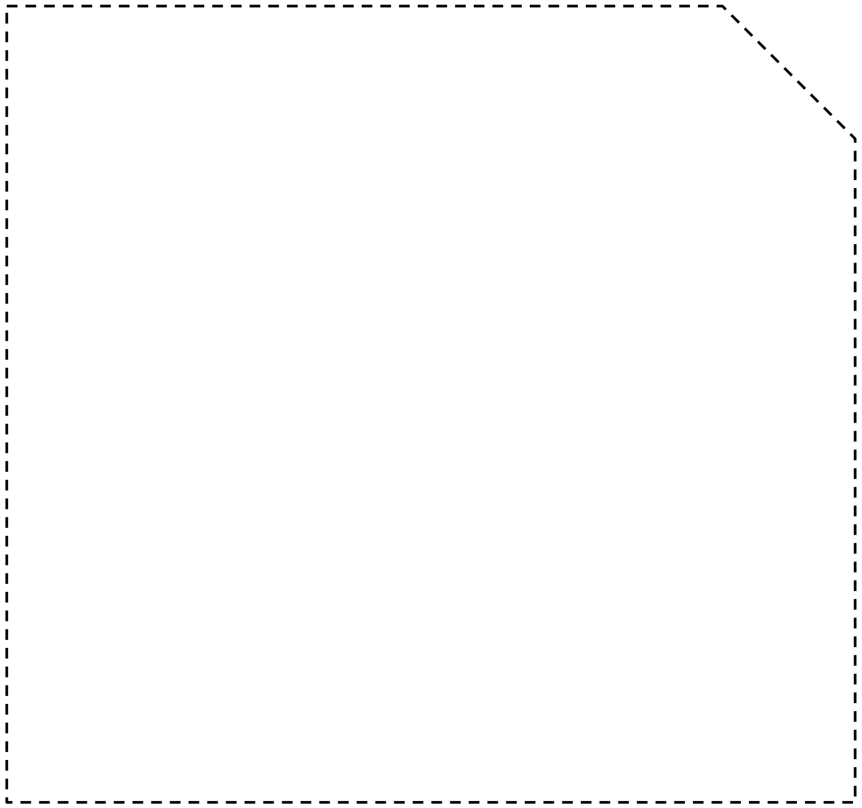


Evolve an input corpus that is guided based on an analysis engine

Code Coverage - Single-App

All possible code paths

Code Coverage - Single-App



Input Corpus

Per-Input
Coverage

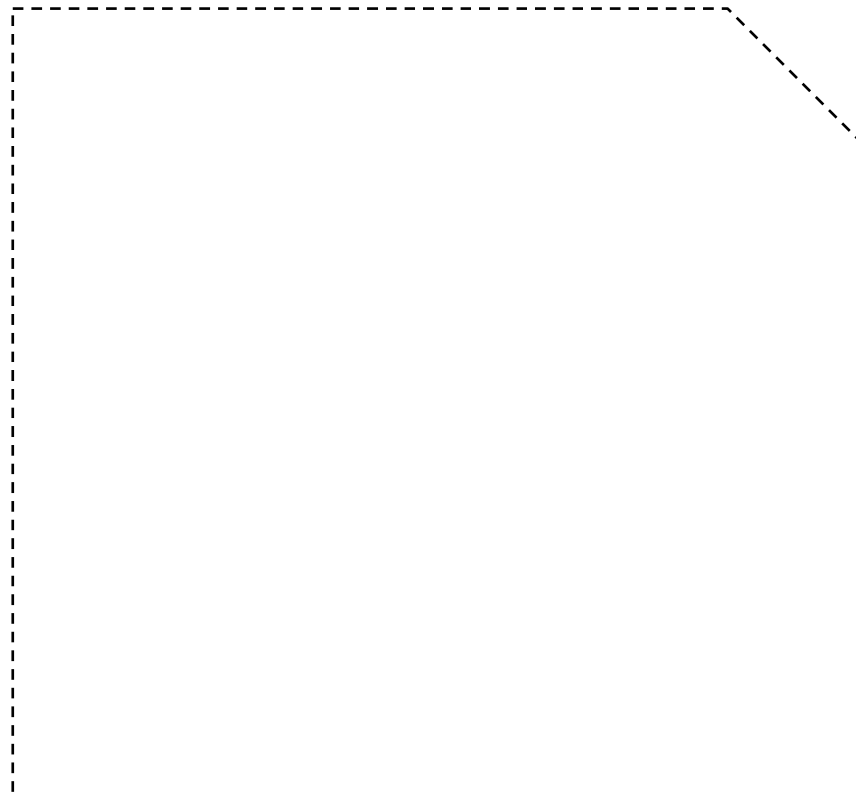
All possible code paths

Code coverage - Global

Code coverage - Input

Code Coverage - Single-App

Input 1



Input Corpus

Per-Input
Coverage

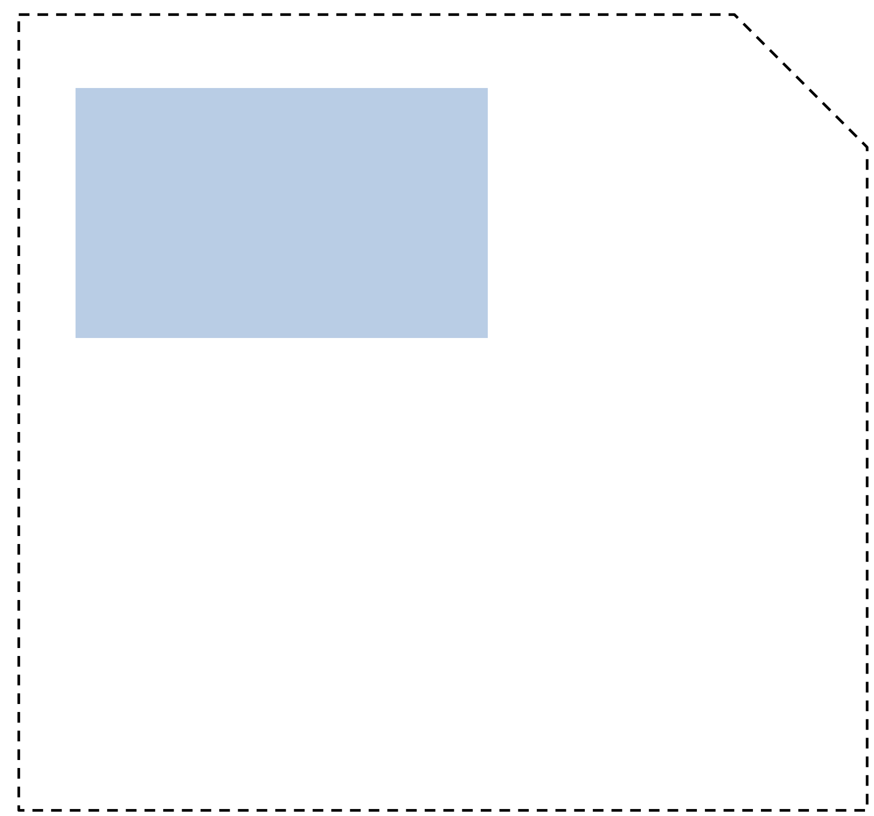
All possible code paths

Code coverage - Global

Code coverage - Input

Code Coverage - Single-App

Input 1



Input Corpus

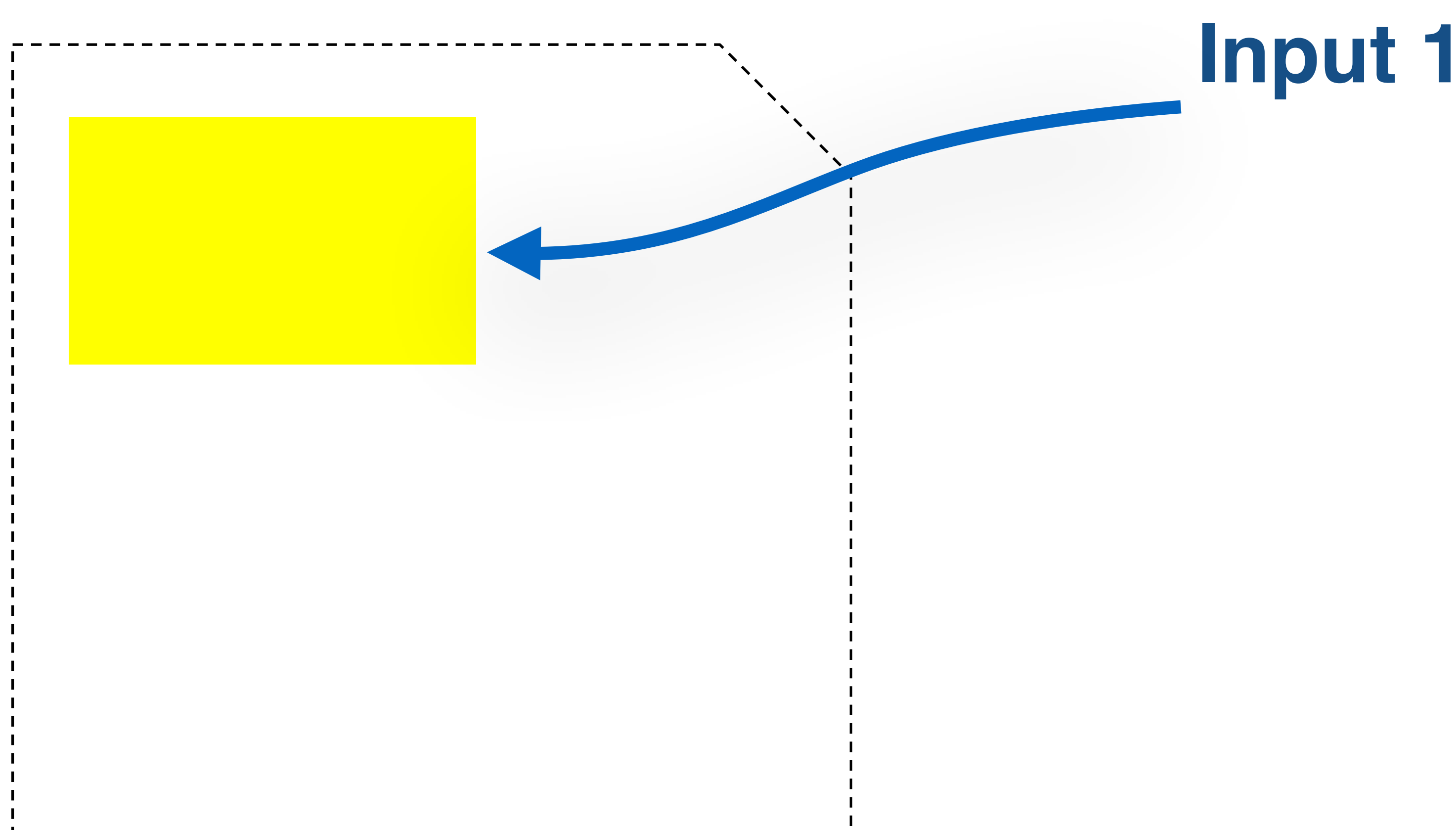
Per-Input
Coverage

All possible code paths

Code coverage - Global

Code coverage - Input

Code Coverage - Single-App



Input Corpus

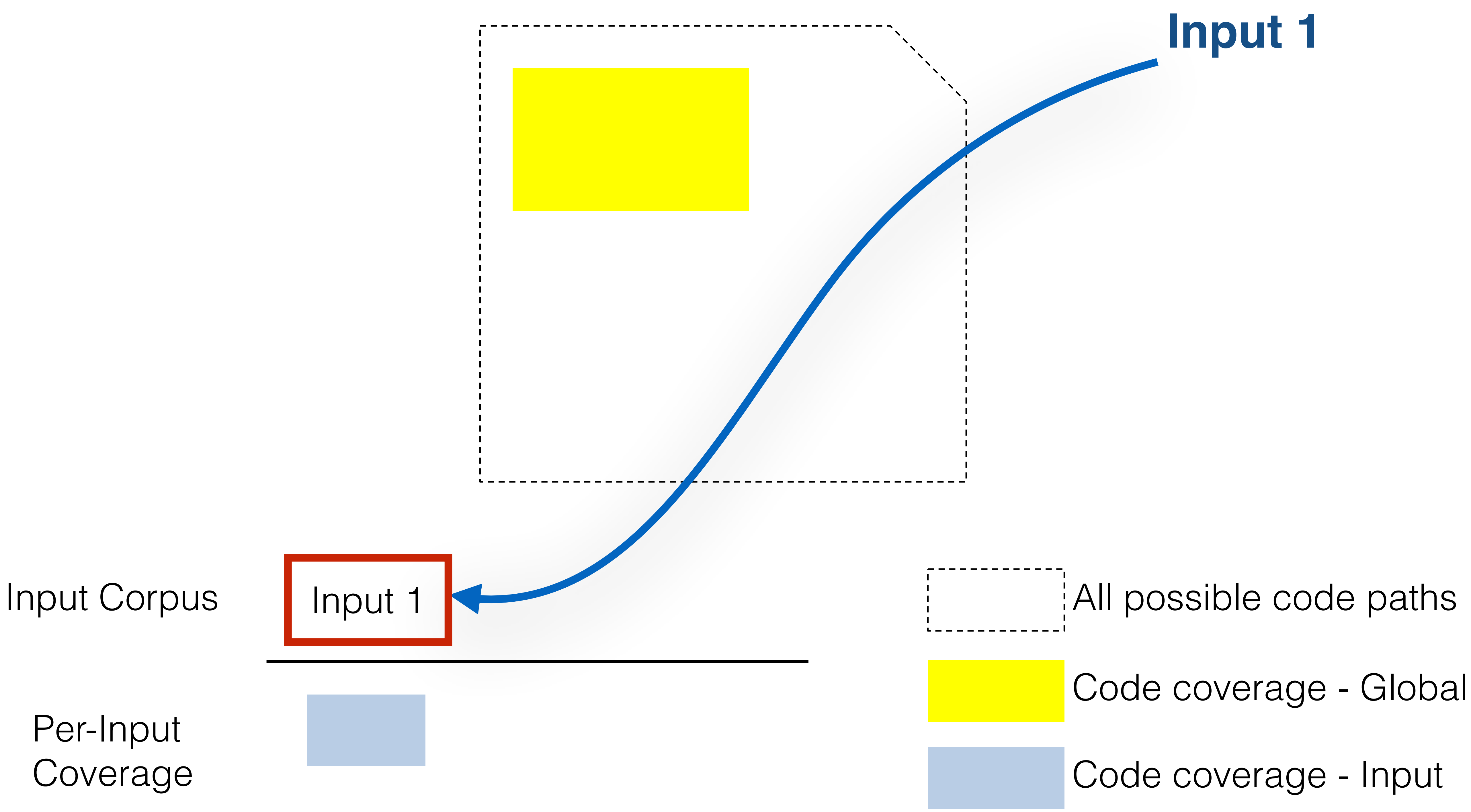
Per-Input
Coverage

All possible code paths

Code coverage - Global

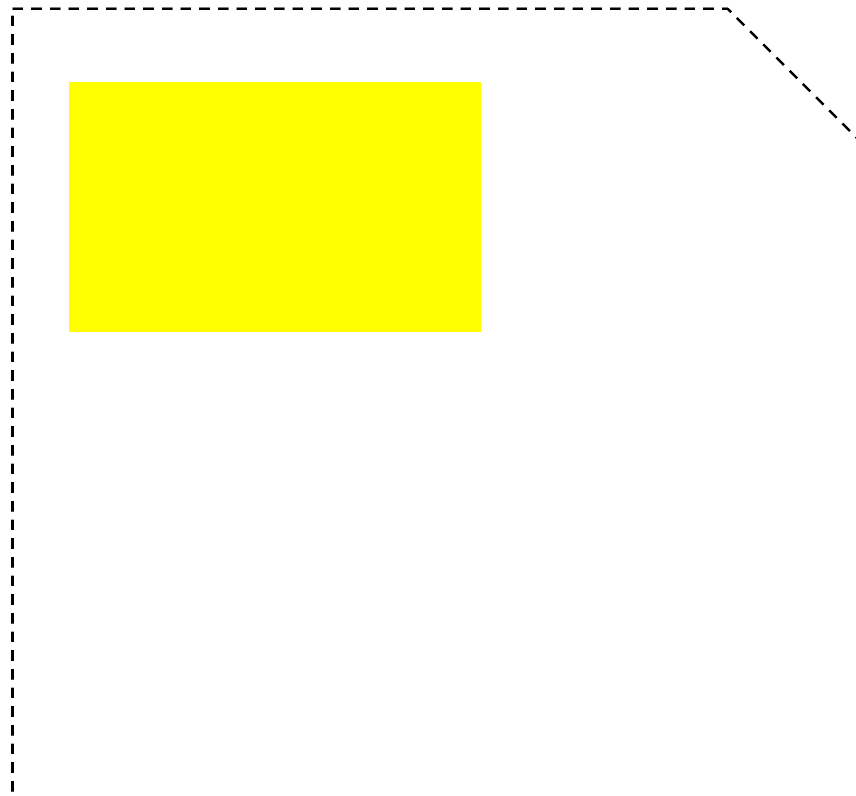
Code coverage - Input

Code Coverage - Single-App



Code Coverage - Single-App

Input 2



Input Corpus Input 1

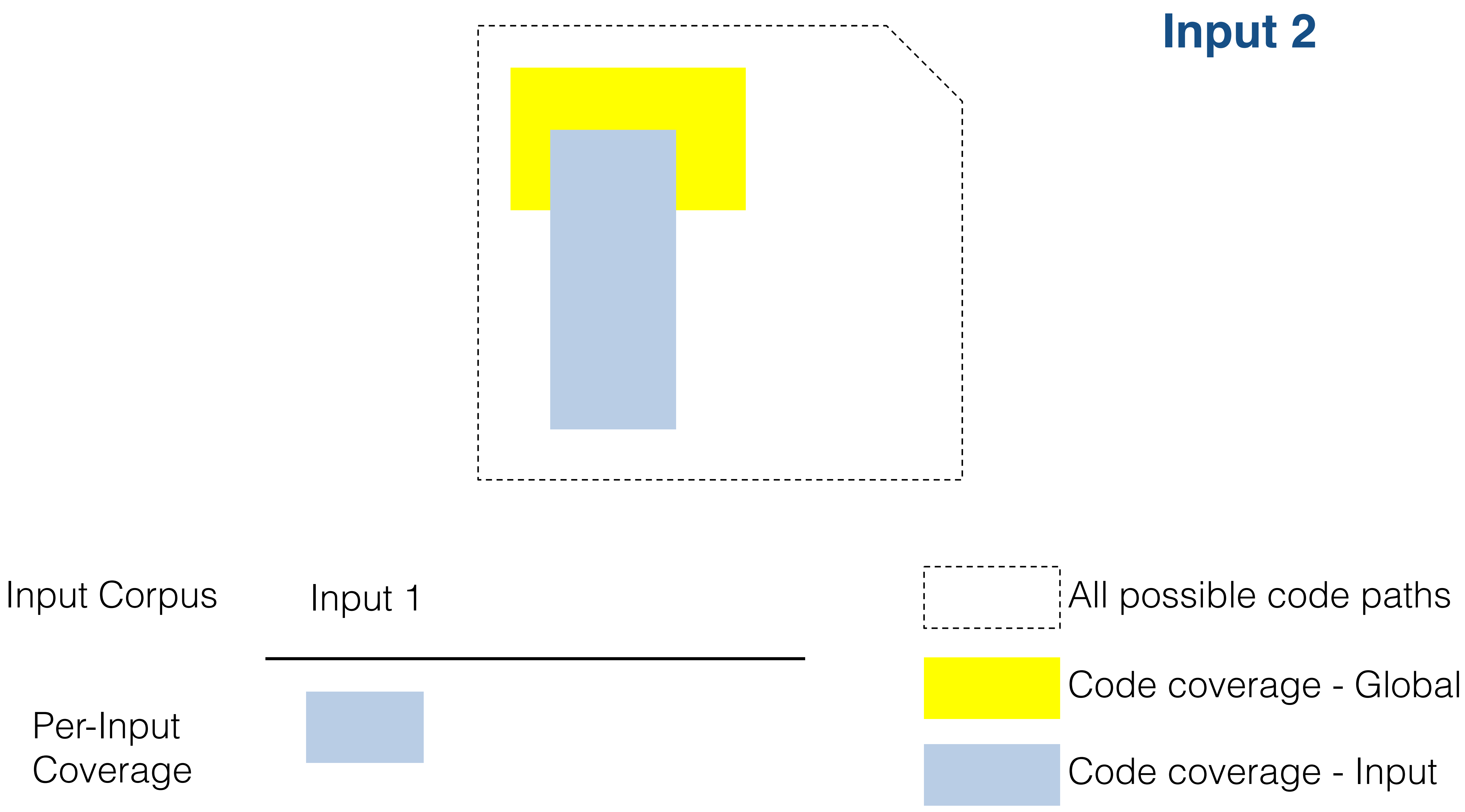
Per-Input Coverage

All possible code paths

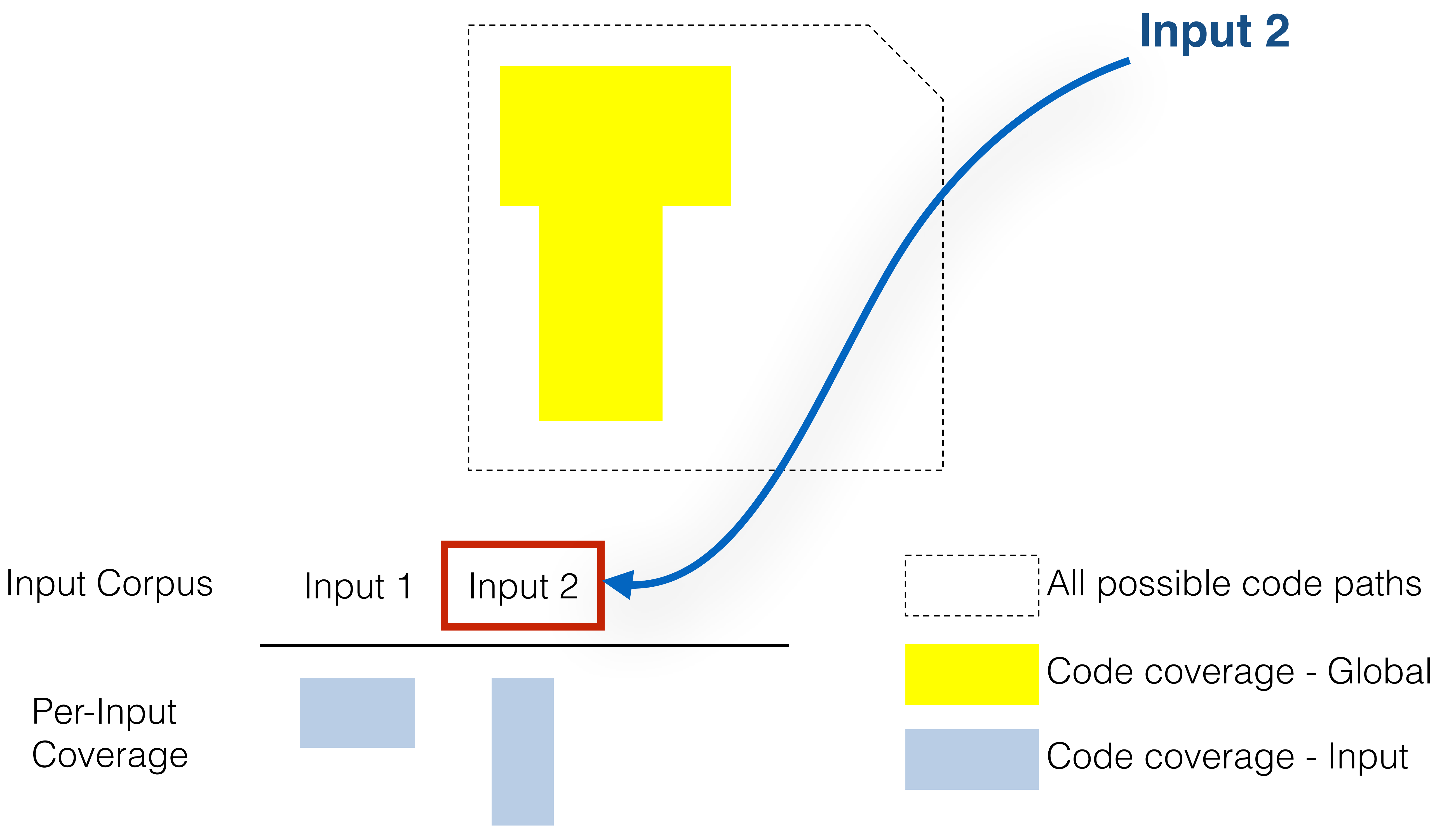
Code coverage - Global

Code coverage - Input

Code Coverage - Single-App

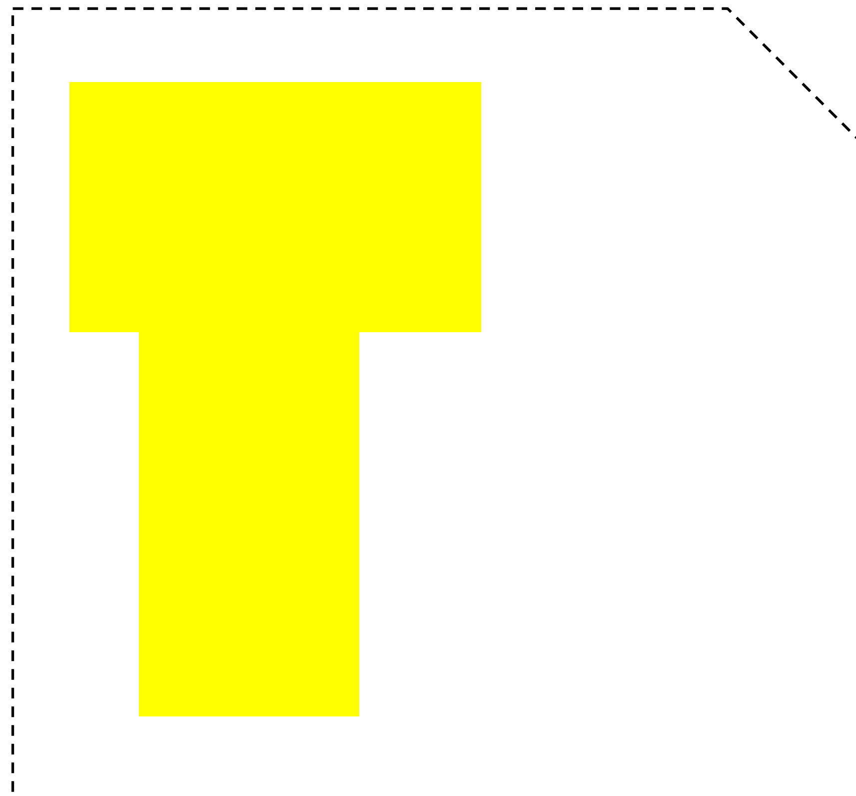


Code Coverage - Single-App



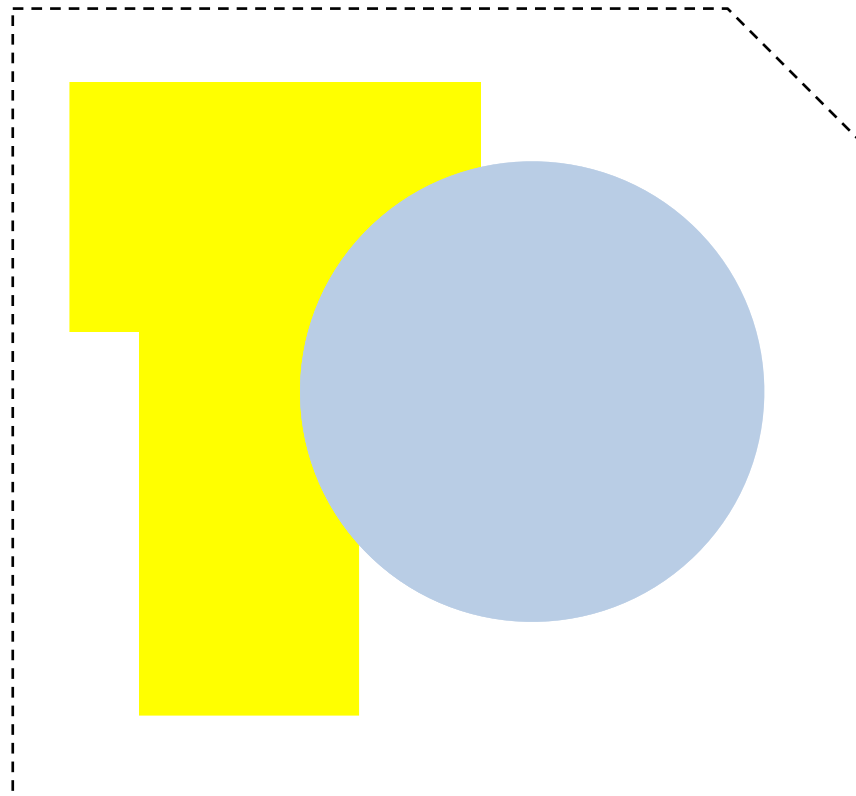
Code Coverage - Single-App

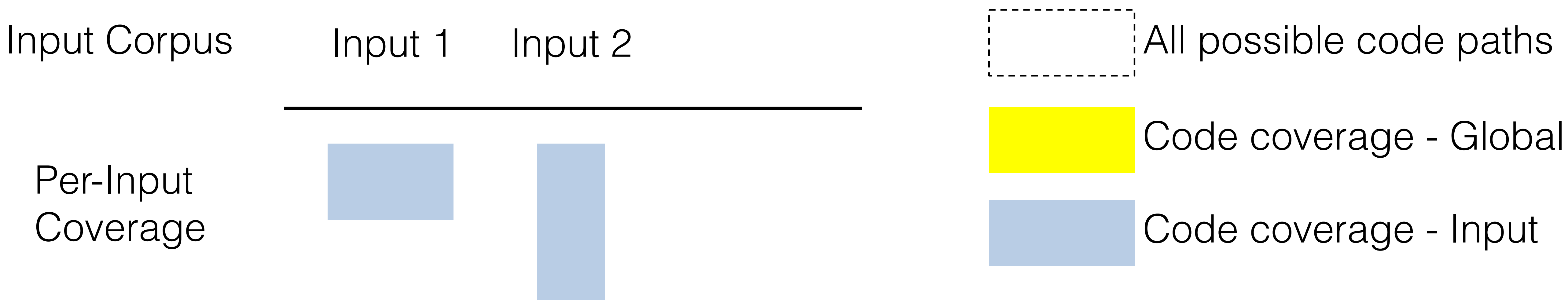
Input 3



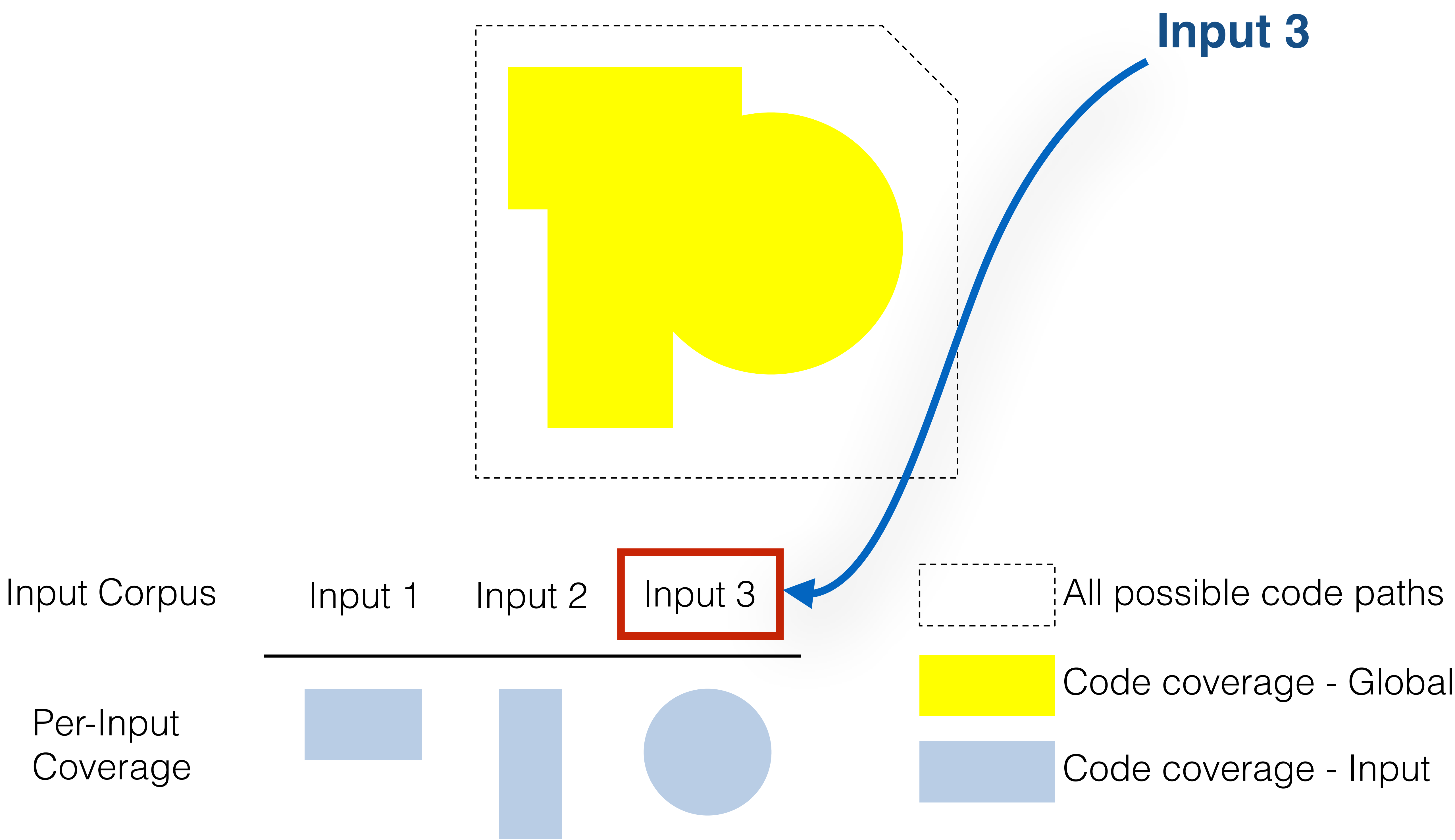
Code Coverage - Single-App

Input 3

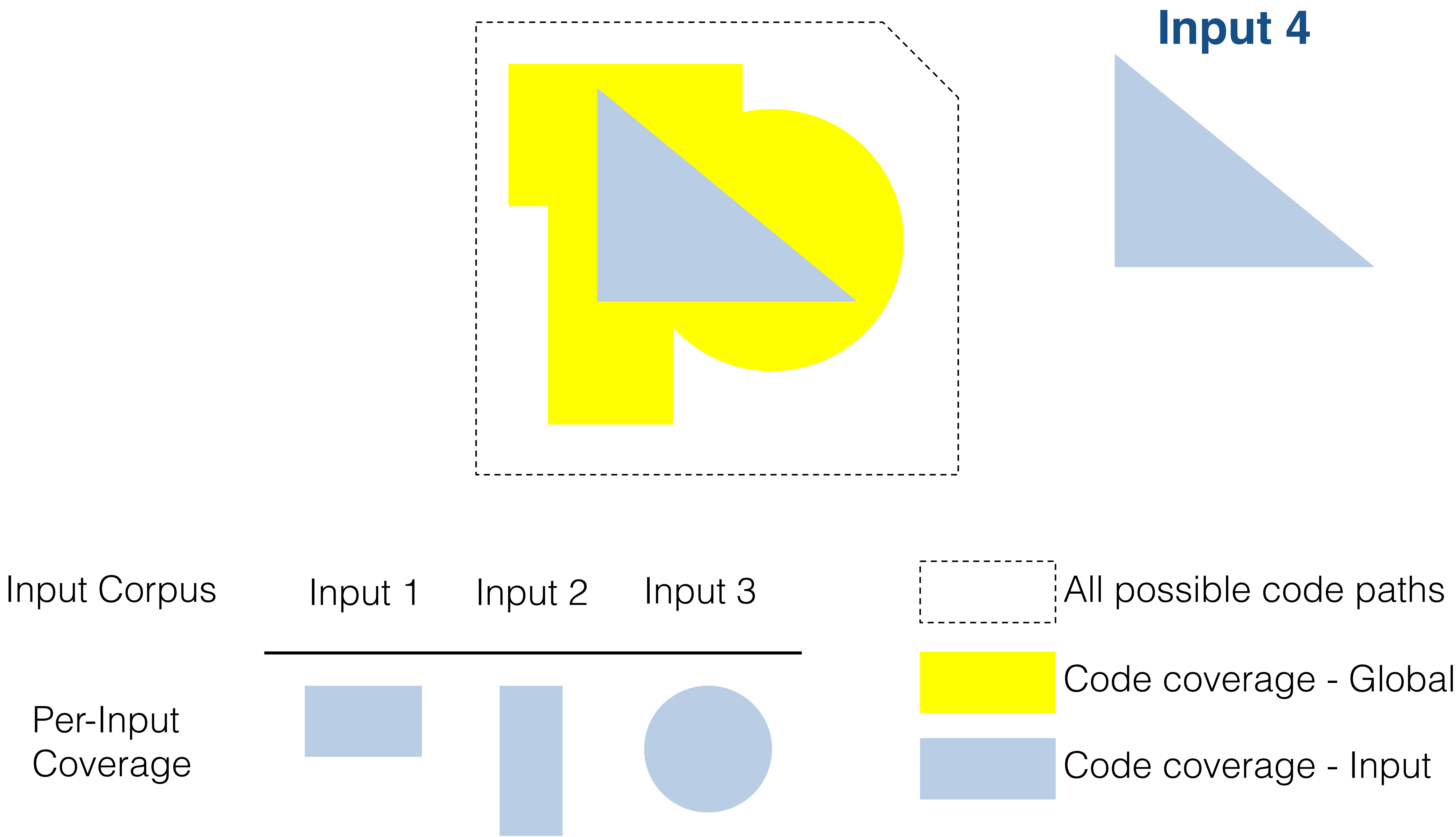




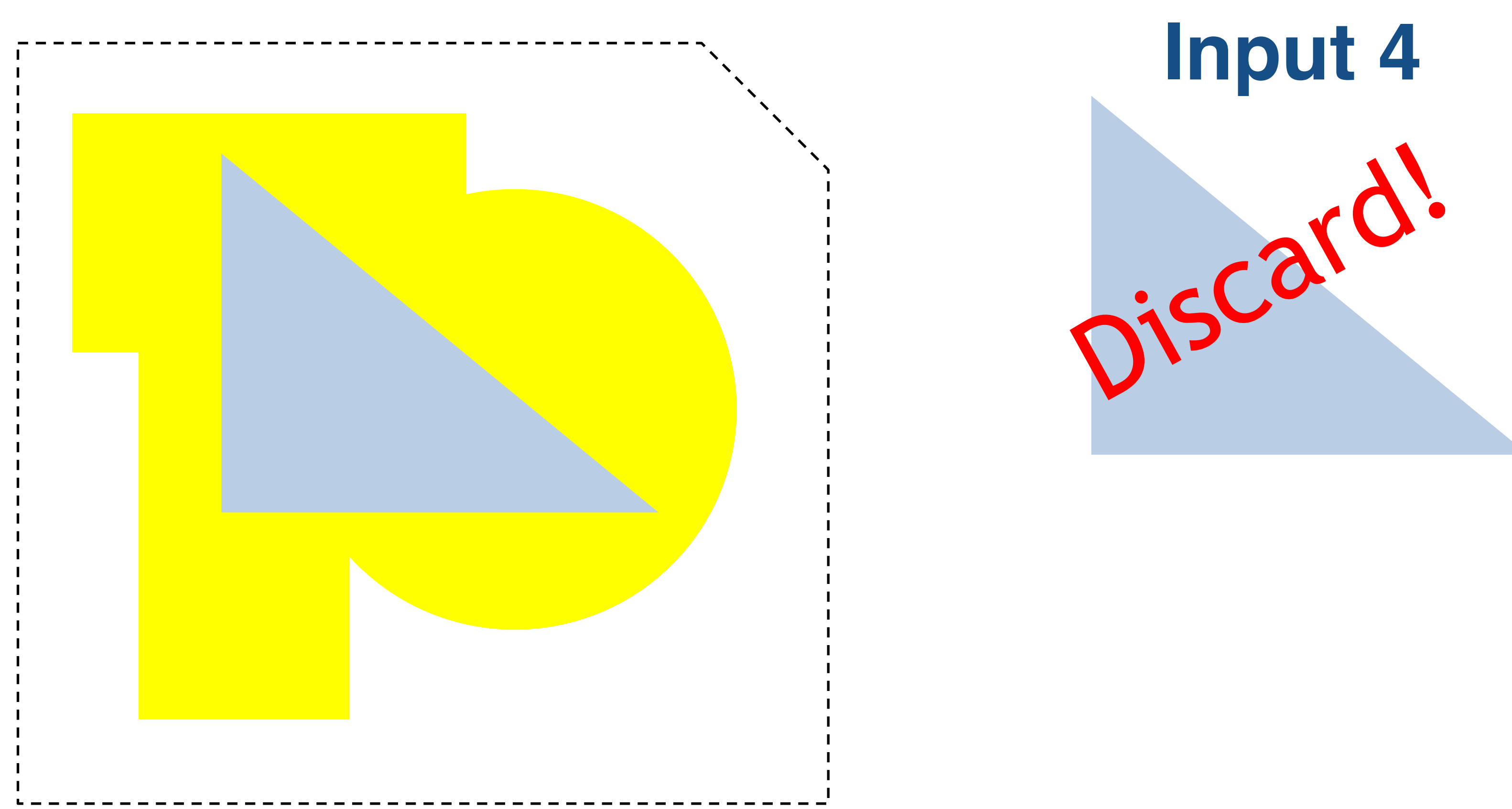
Code Coverage - Single-App



Code Coverage - Single-App

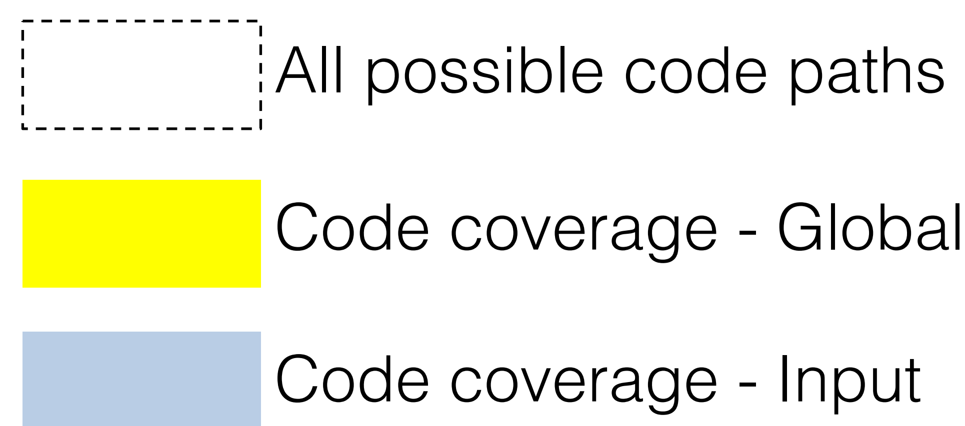


Code Coverage - Single-App

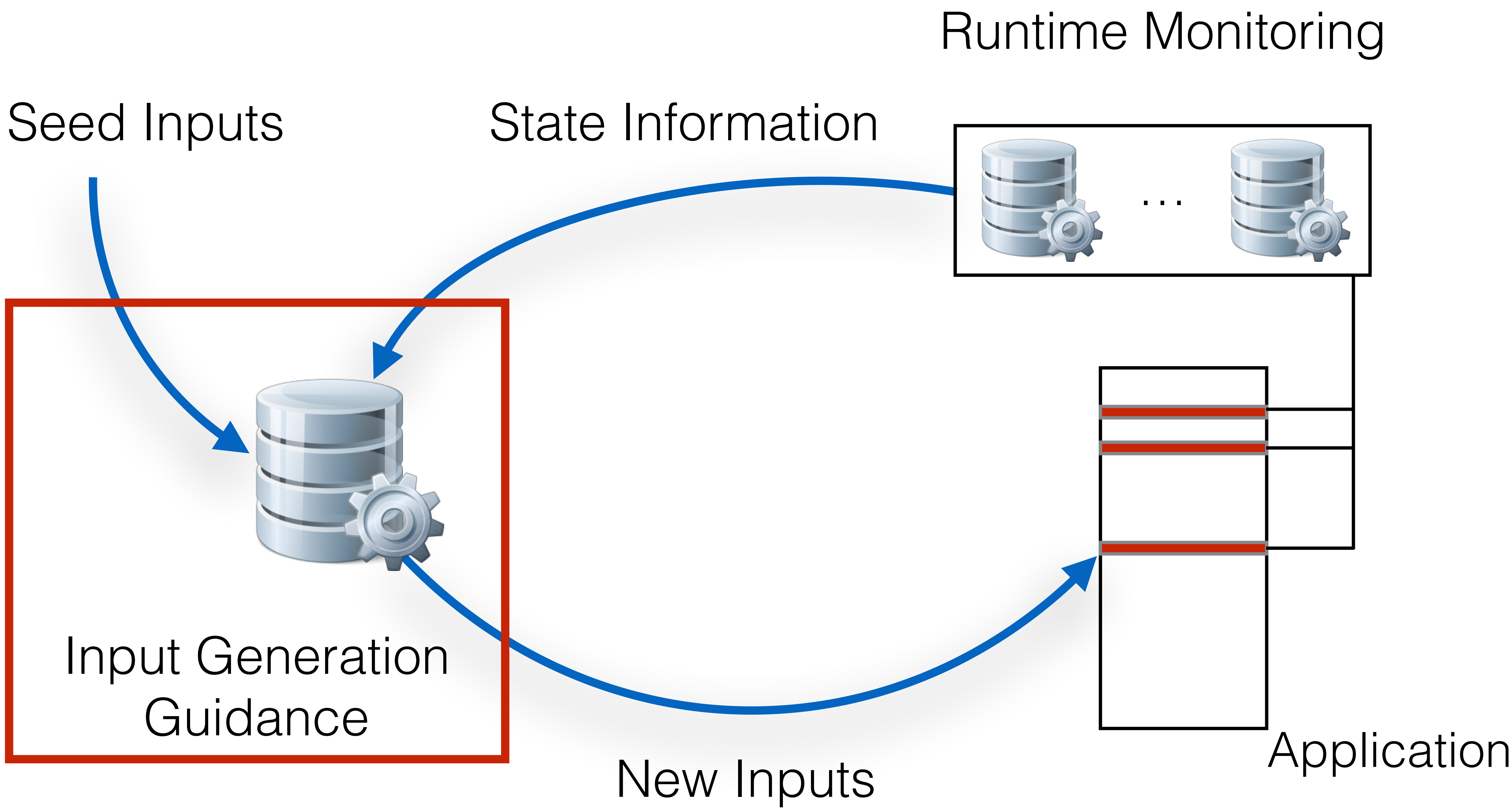


Input Corpus	Input 1	Input 2	Input 3

Per-Input Coverage



Domain-Independent Evolutionary Testing



Evolutionary Differential Testing - Multiple-Apps

What are the options to driving input generation?

1. Use program states solely from single application, like most modern fuzzers

2. Use global program states combined across all applications

3. Re-design guidance engine geared towards differential testing

Evolutionary Differential Testing - Multiple-Apps

What are the options for driving input generation?

1. Use program states solely from single application, like most modern fuzzers

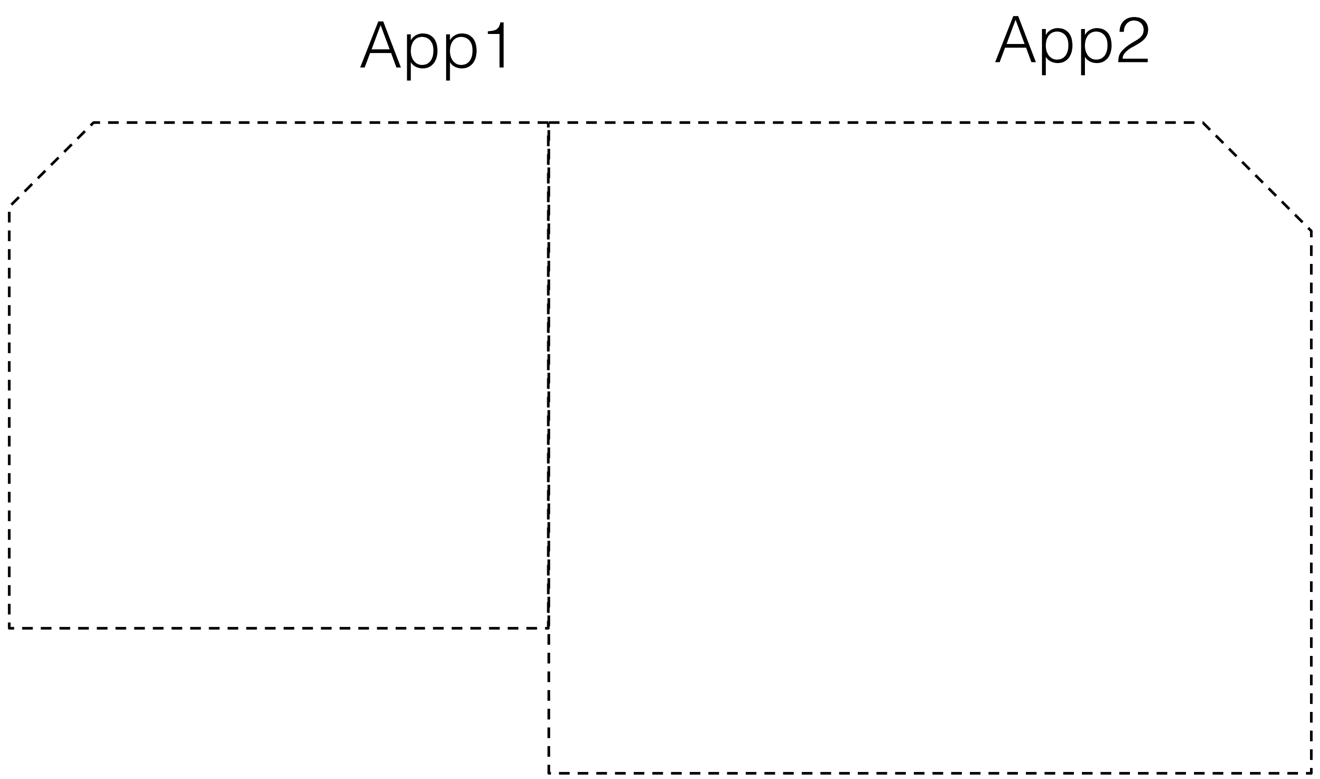
2. Use global program states combined across all applications

3. Re-design guidance engine geared towards differential testing

Key Insight

Techniques that work well in the context of single application testing may not be optimal for differential testing!

Multi-App Code Coverage



Input Corpus

Per-Input
Coverage

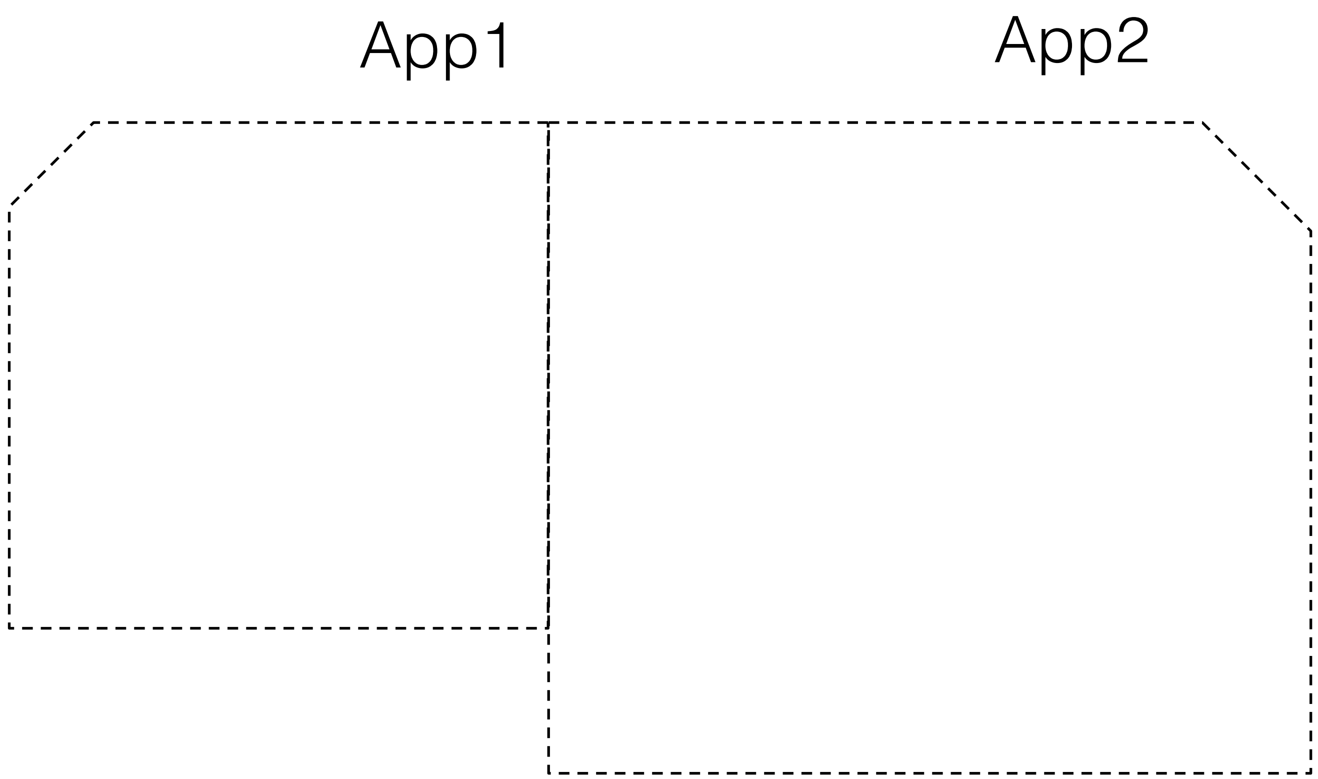
All possible code paths

Code coverage - Global

Code coverage - Input

Multi-App Code Coverage

Input 1



Input Corpus

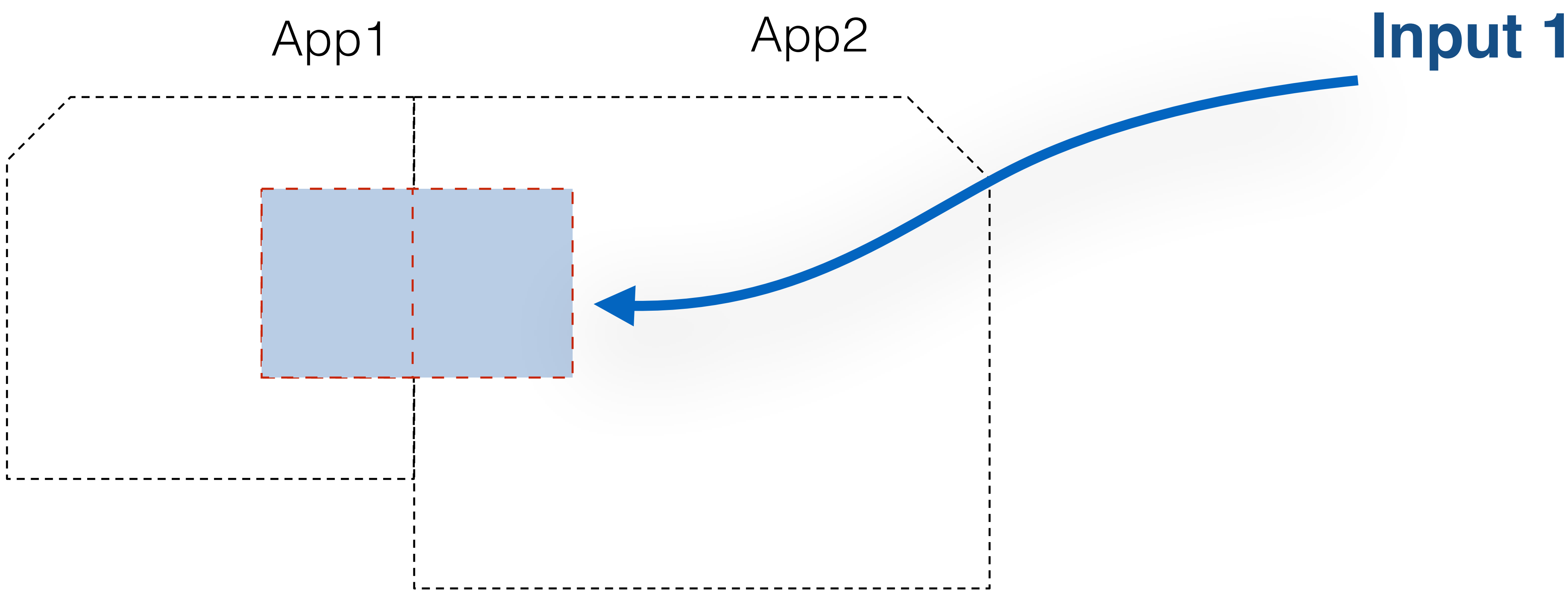
Per-Input
Coverage

All possible code paths

Code coverage - Global

Code coverage - Input

Multi-App Code Coverage



Input Corpus

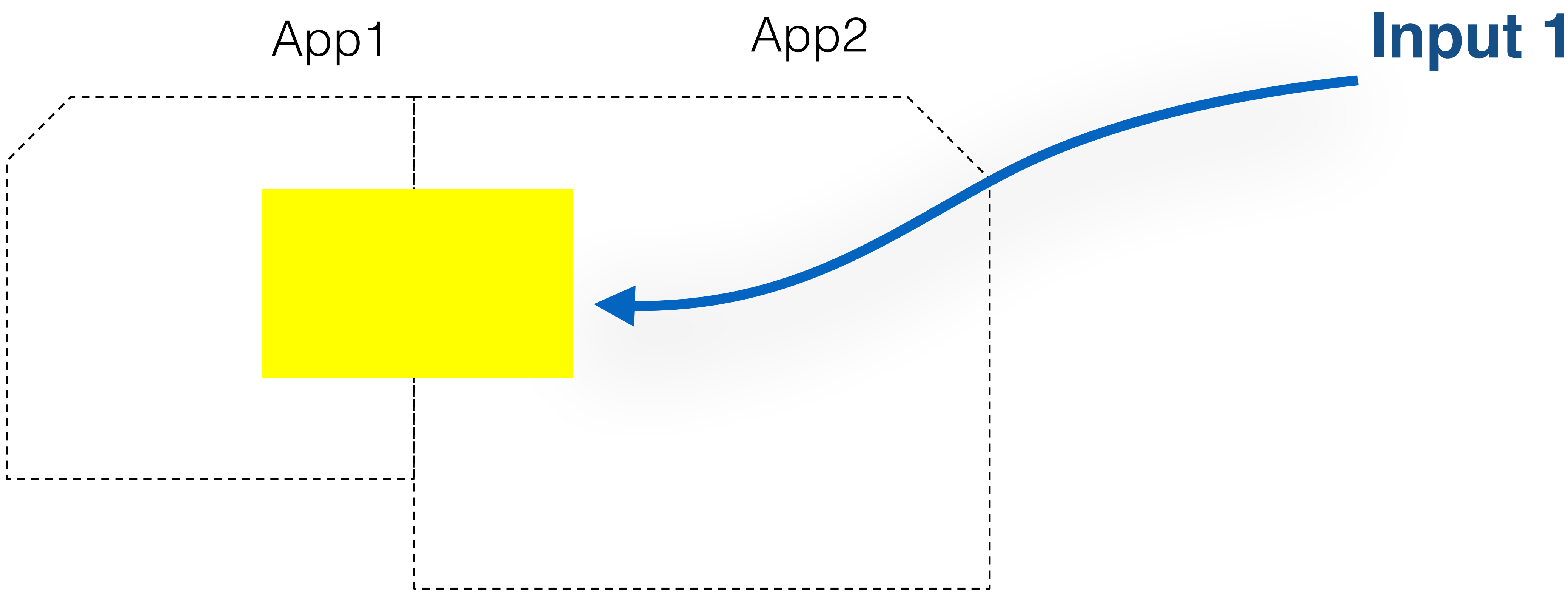
Per-Input
Coverage

All possible code paths

Code coverage - Global

Code coverage - Input

Multi-App Code Coverage



Input Corpus

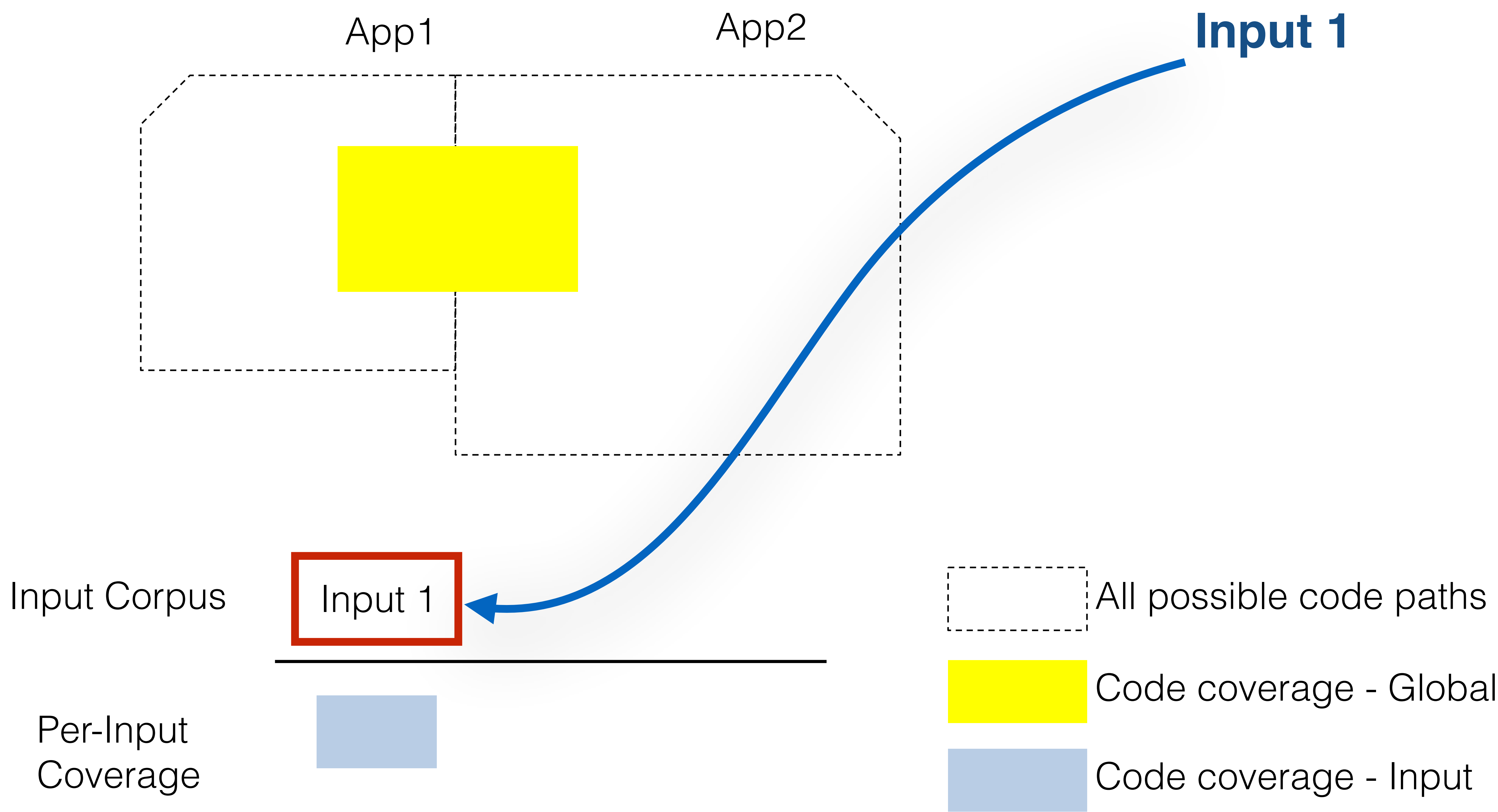
Per-Input Coverage

All possible code paths

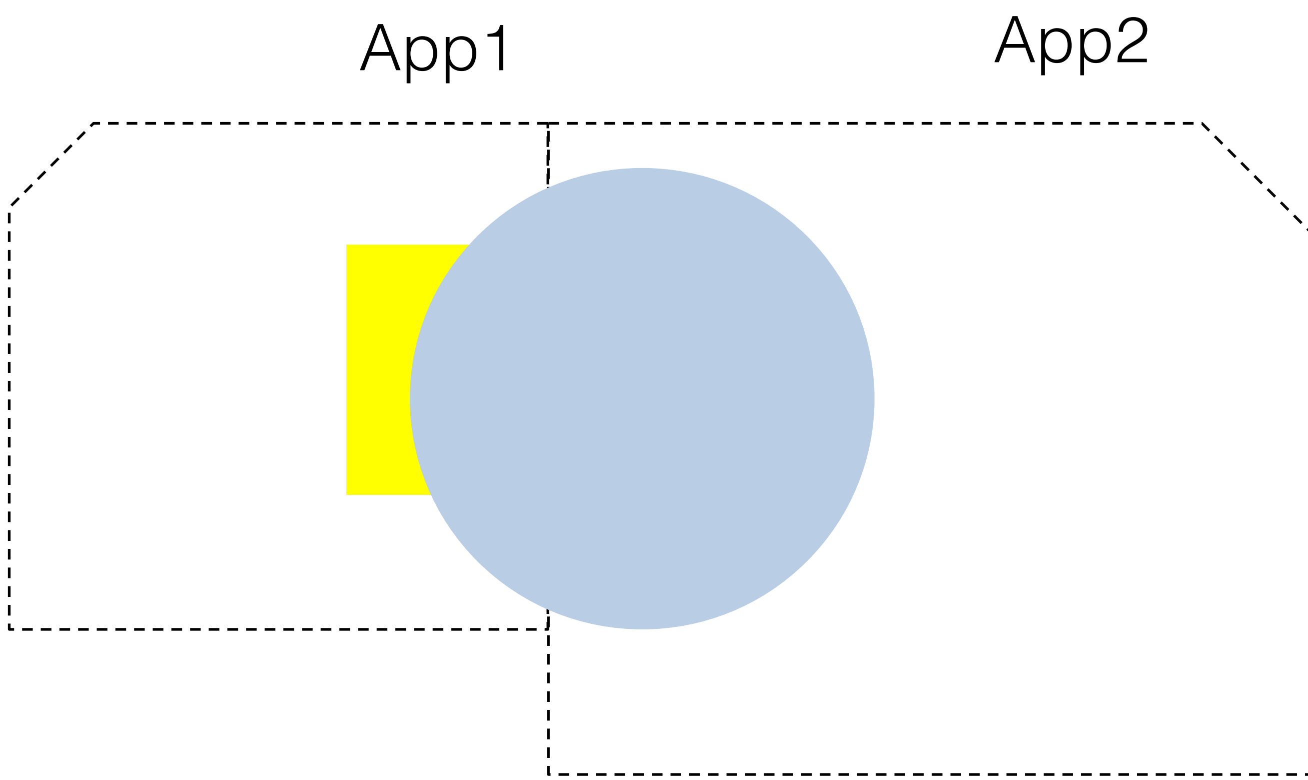
Code coverage - Global

Code coverage - Input

Multi-App Code Coverage



Multi-App Code Coverage



Input 2

Input Corpus — Input 1

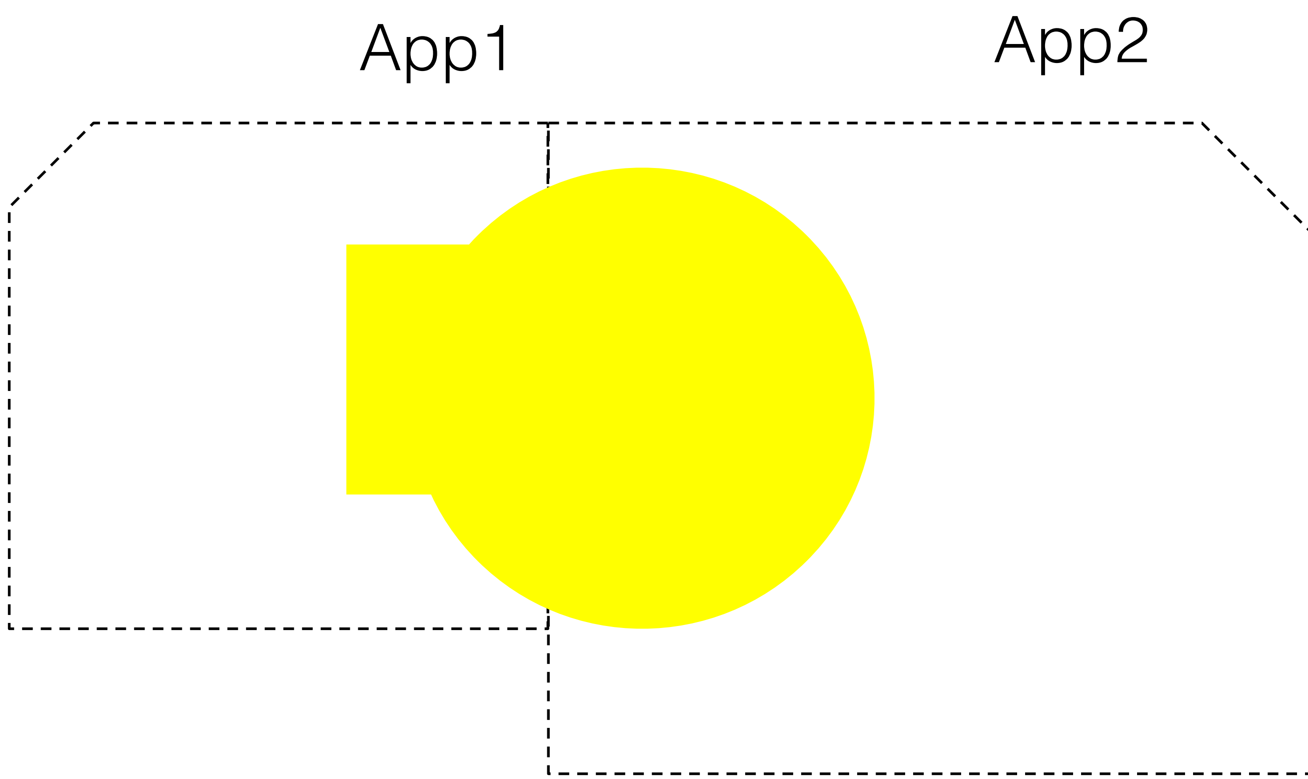
Per-Input Coverage

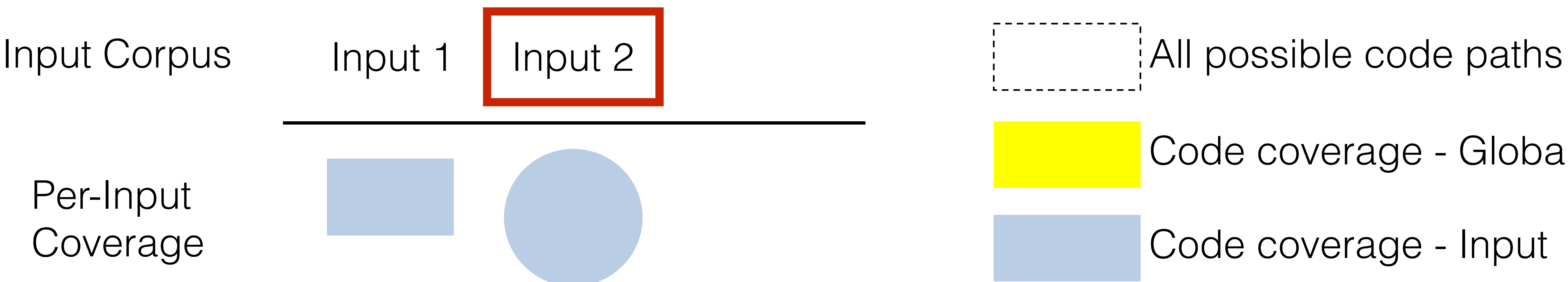
All possible code paths

Code coverage - Global

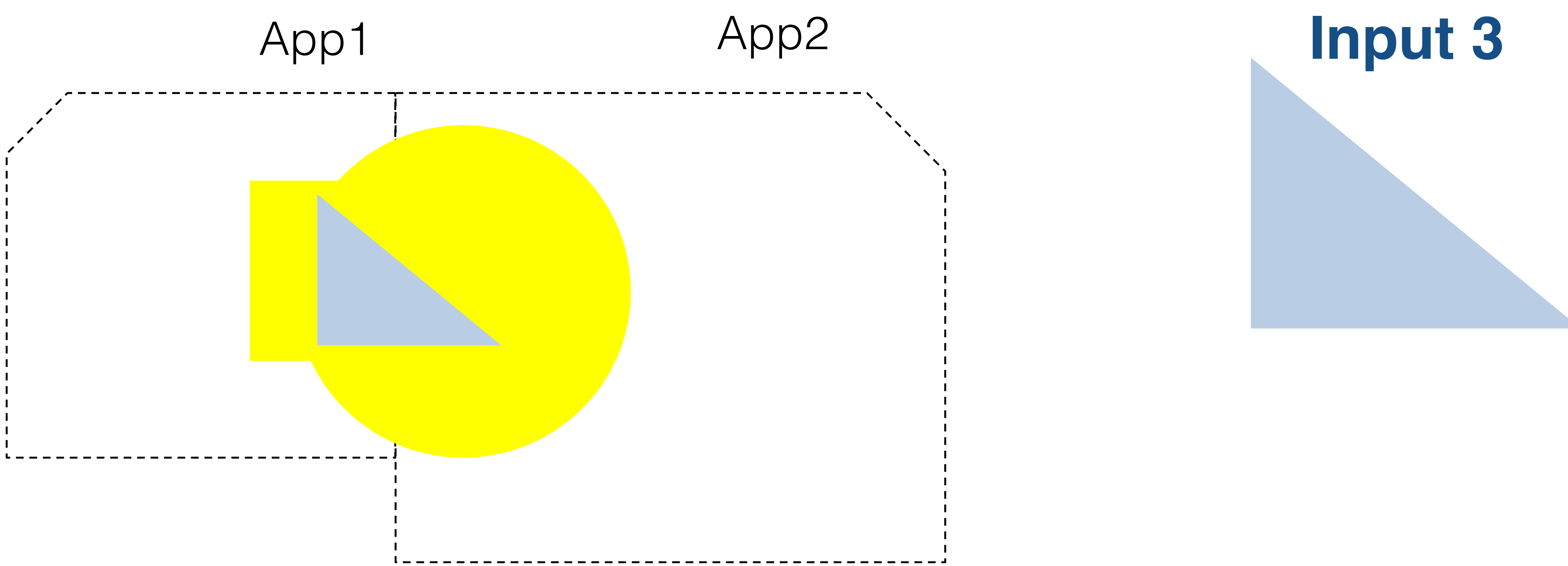
Code coverage - Input

Multi-App Code Coverage





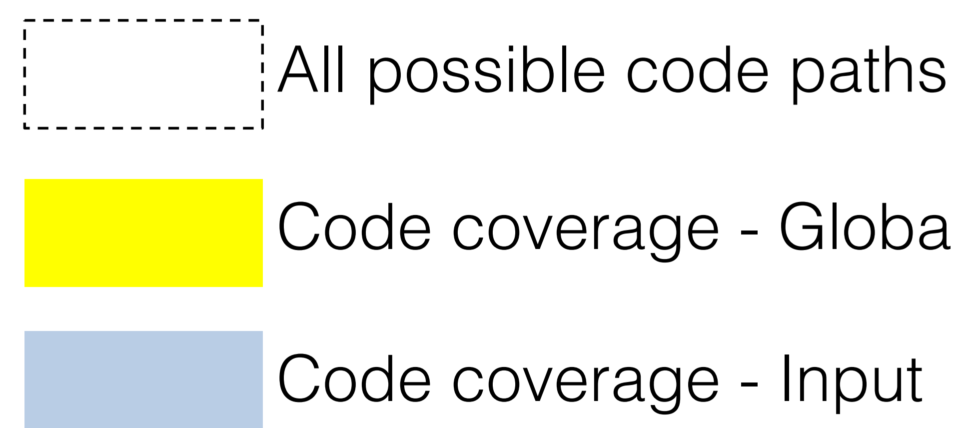
Multi-App Code Coverage



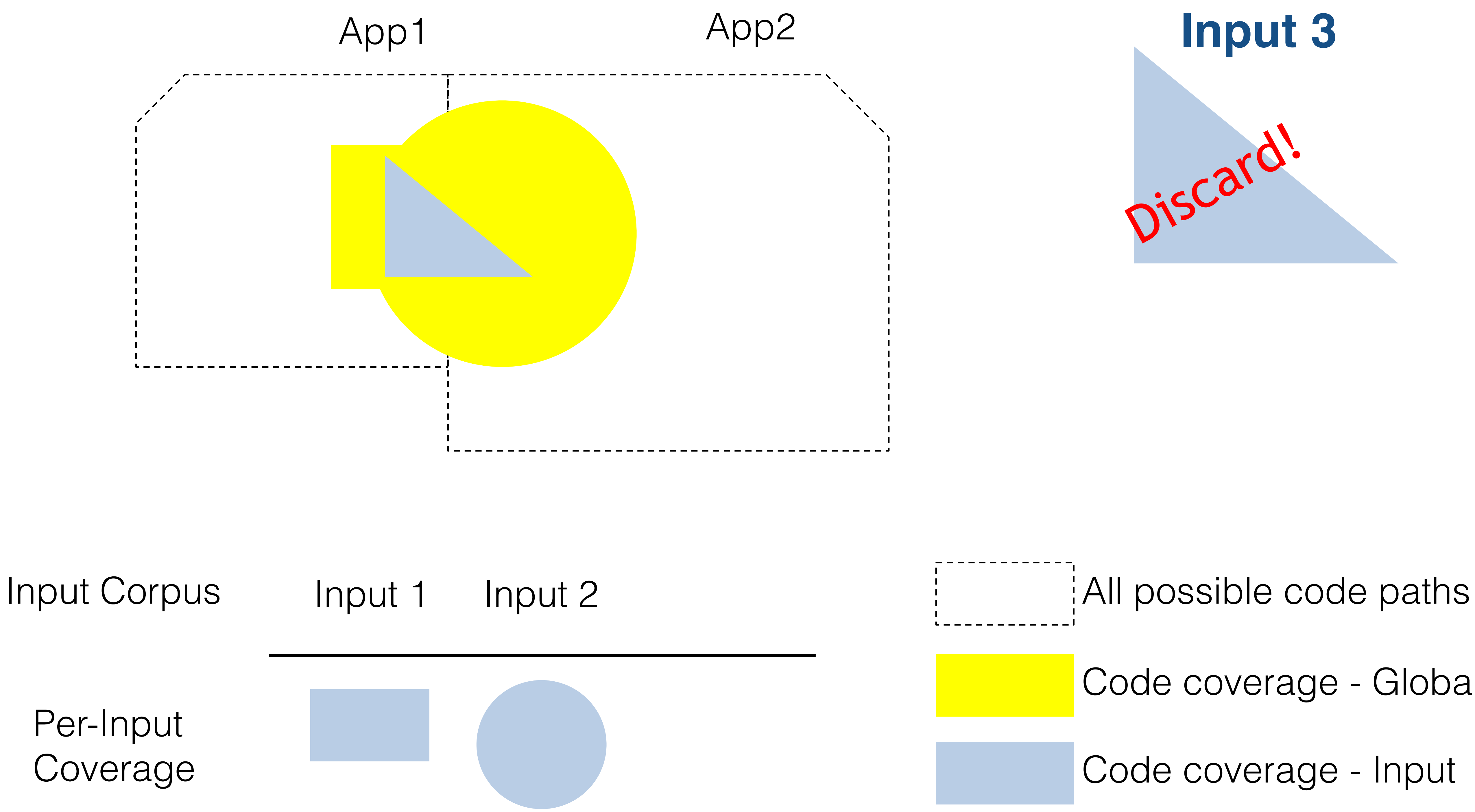
Input Corpus

Input 1 Input 2

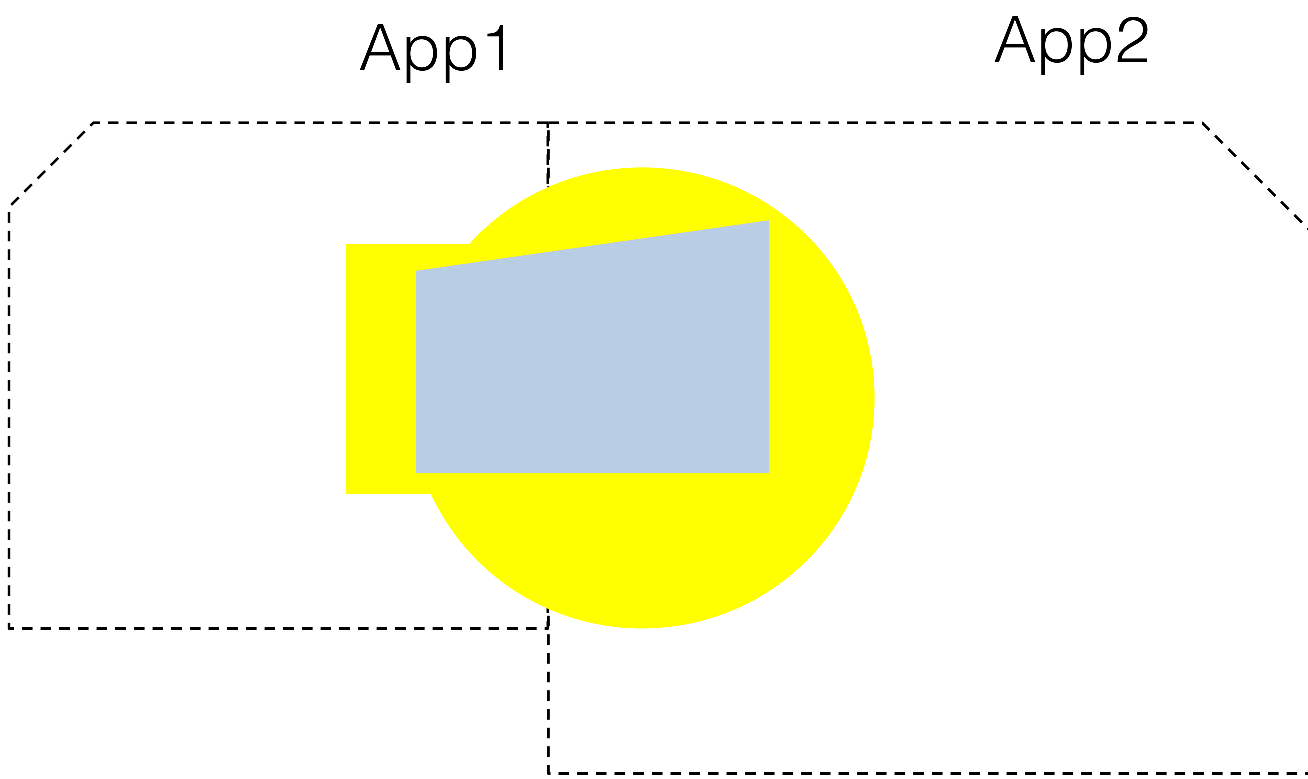
Per-Input Coverage



Multi-App Code Coverage



Multi-App Code Coverage



Input 4

Input Corpus — Input 1, Input 2

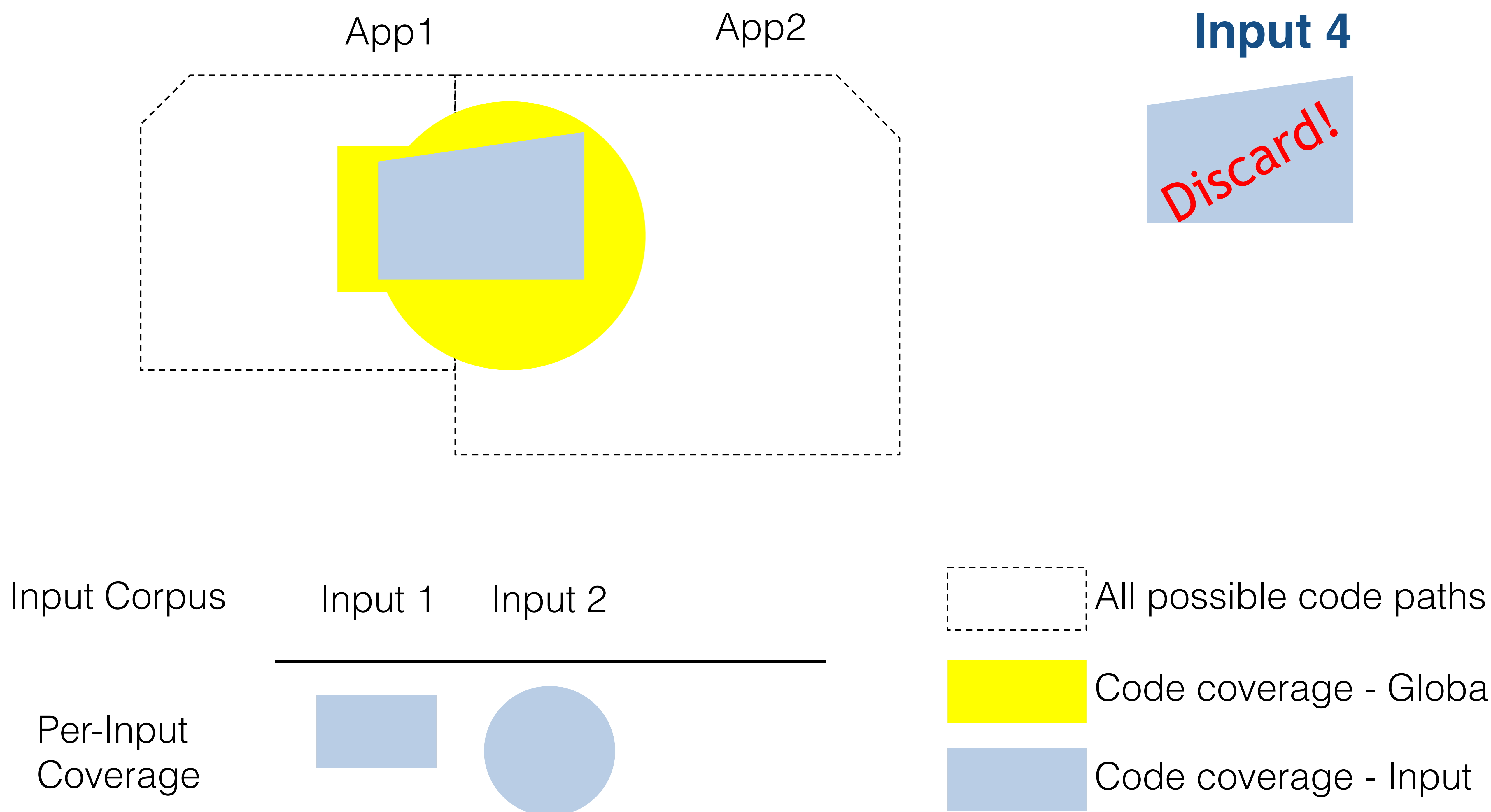
Per-Input Coverage

All possible code paths

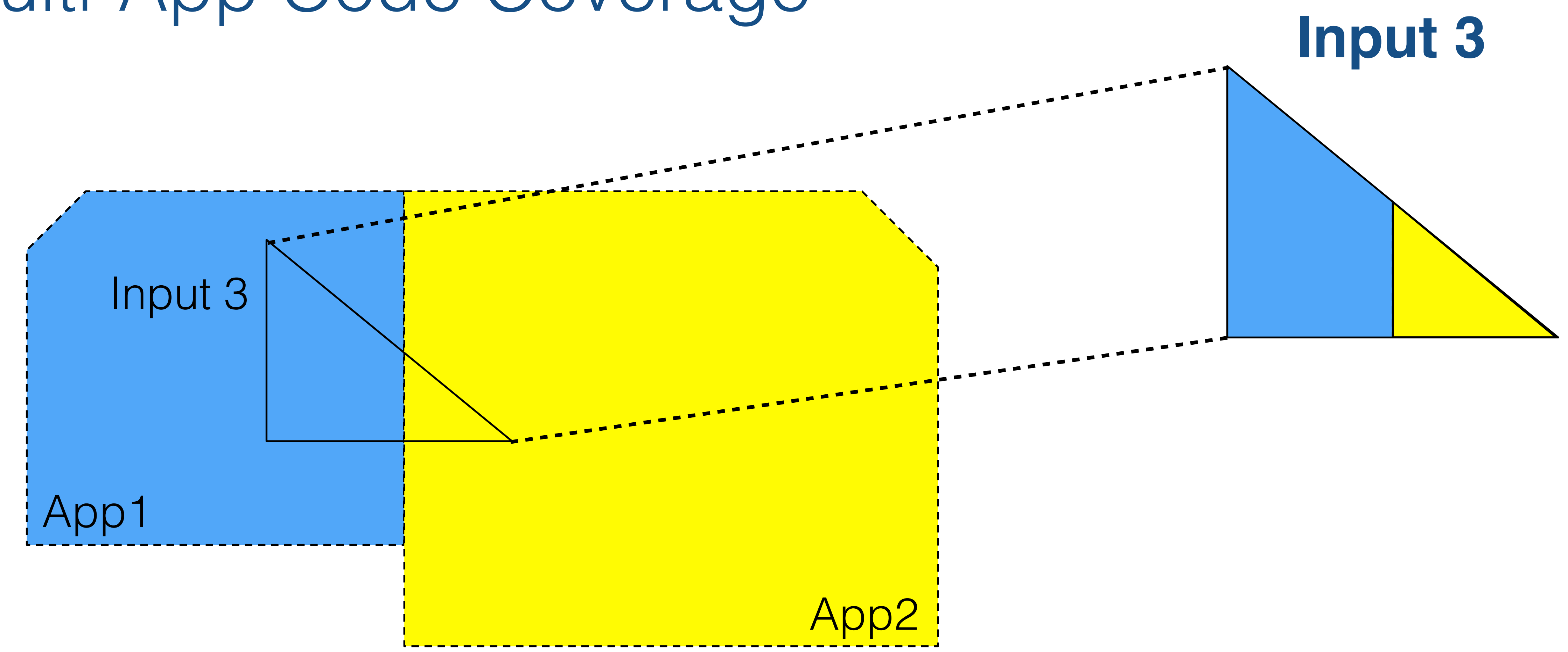
Code coverage - Global

Code coverage - Input

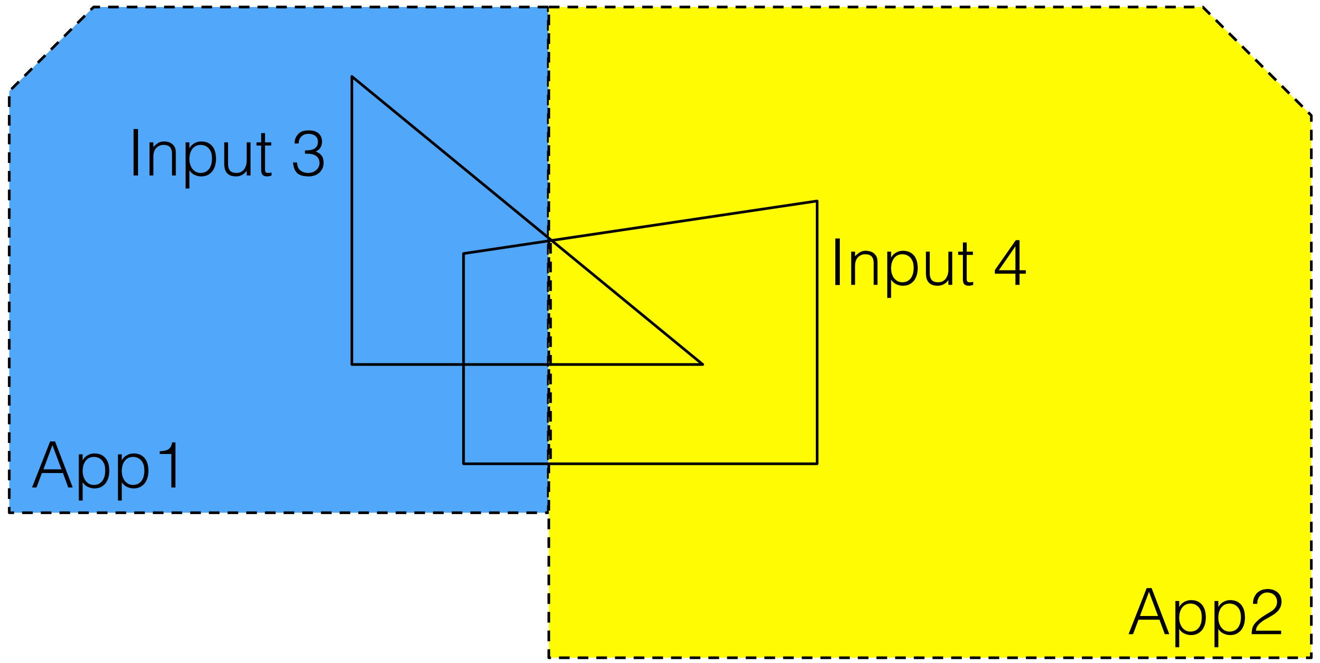
Multi-App Code Coverage

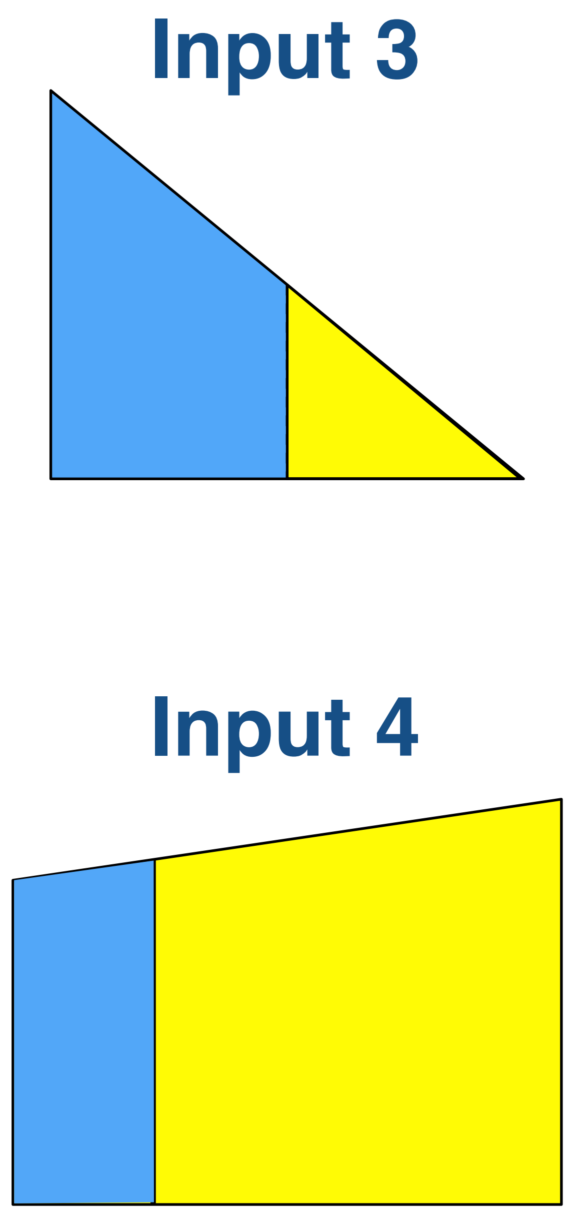


Multi-App Code Coverage

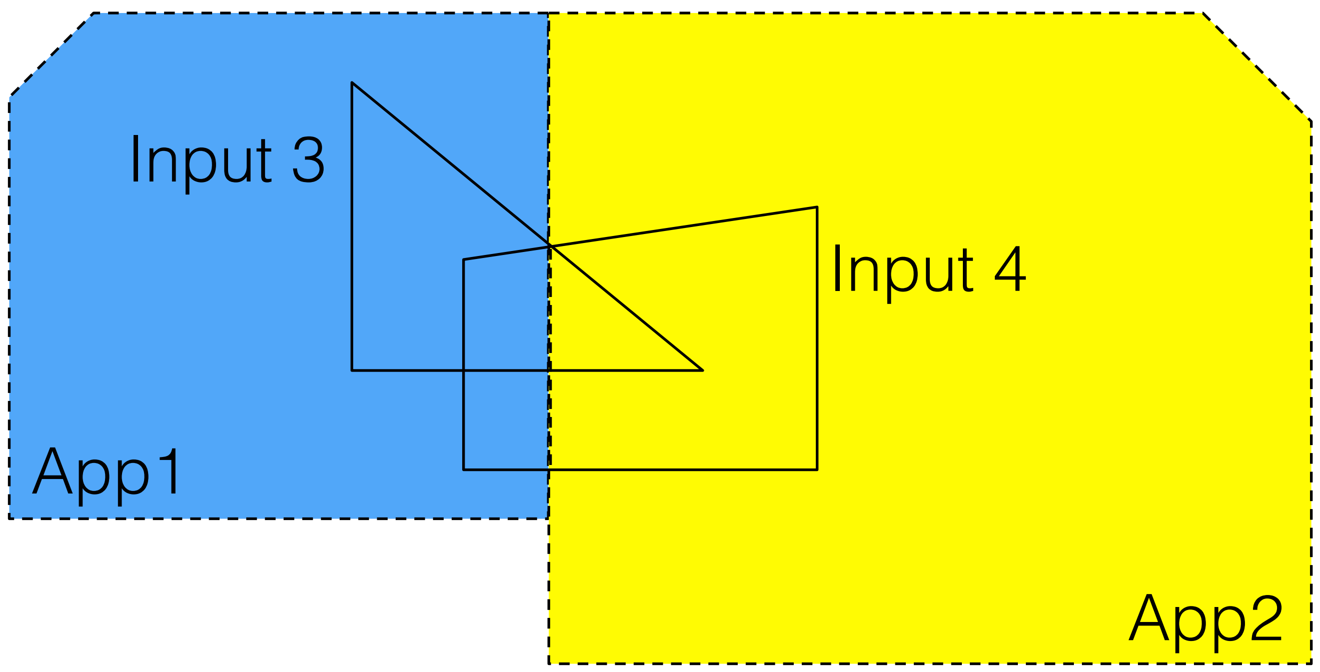


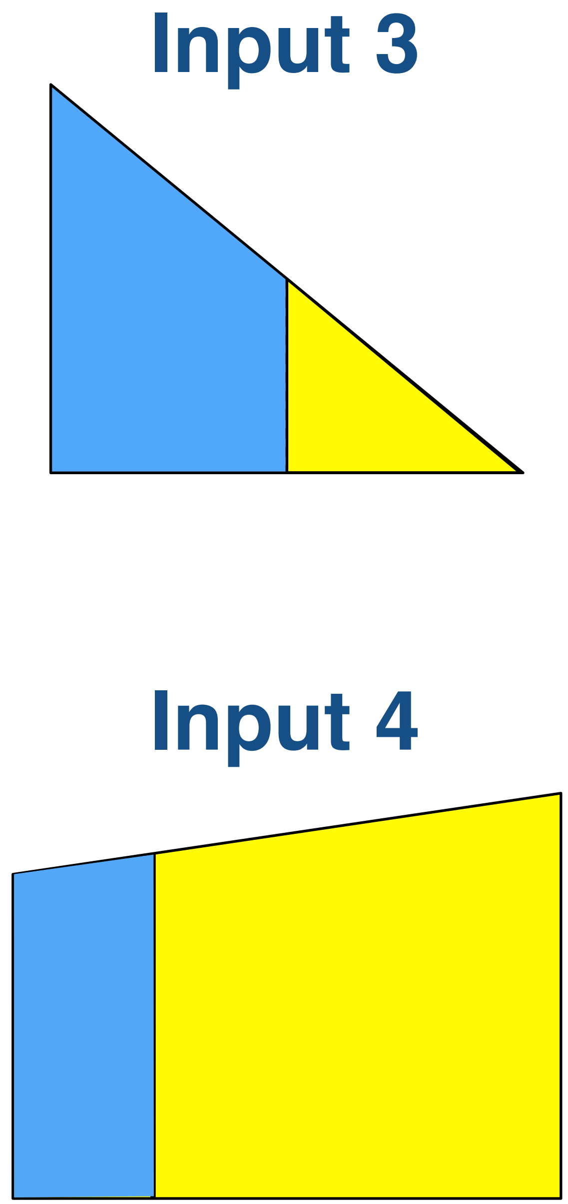
Multi-App Code Coverage





Multi-App Code Coverage





- These inputs exercise disproportionate code regions in the two apps
- This disproportion might imply differences in handling logic
- Retaining them in corpus speed up process of finding discrepancies

Relative program behavior is important in this context!

δ-diversity: a new approach to guided differential testing

Differential Testing under δ-diversity

- Obtain State Information
  - White-box (e.g., at compile time)
  - Gray-box (e.g., using Dynamic Binary Instrumentation)
  - Black-box (e.g., only examining system response to inputs)

- Behavioral Diversity

Differential Testing under δ-diversity

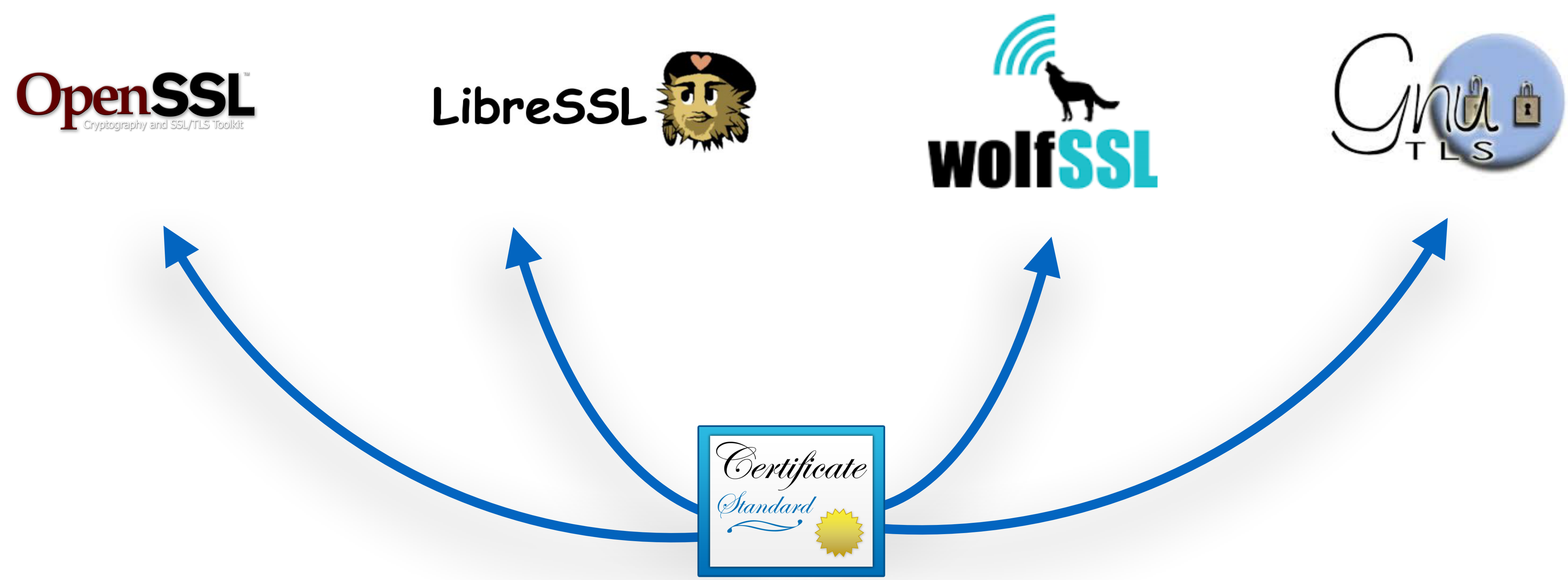
OpenSSL Cryptography and SSL/TLS Toolkit

LibreSSL

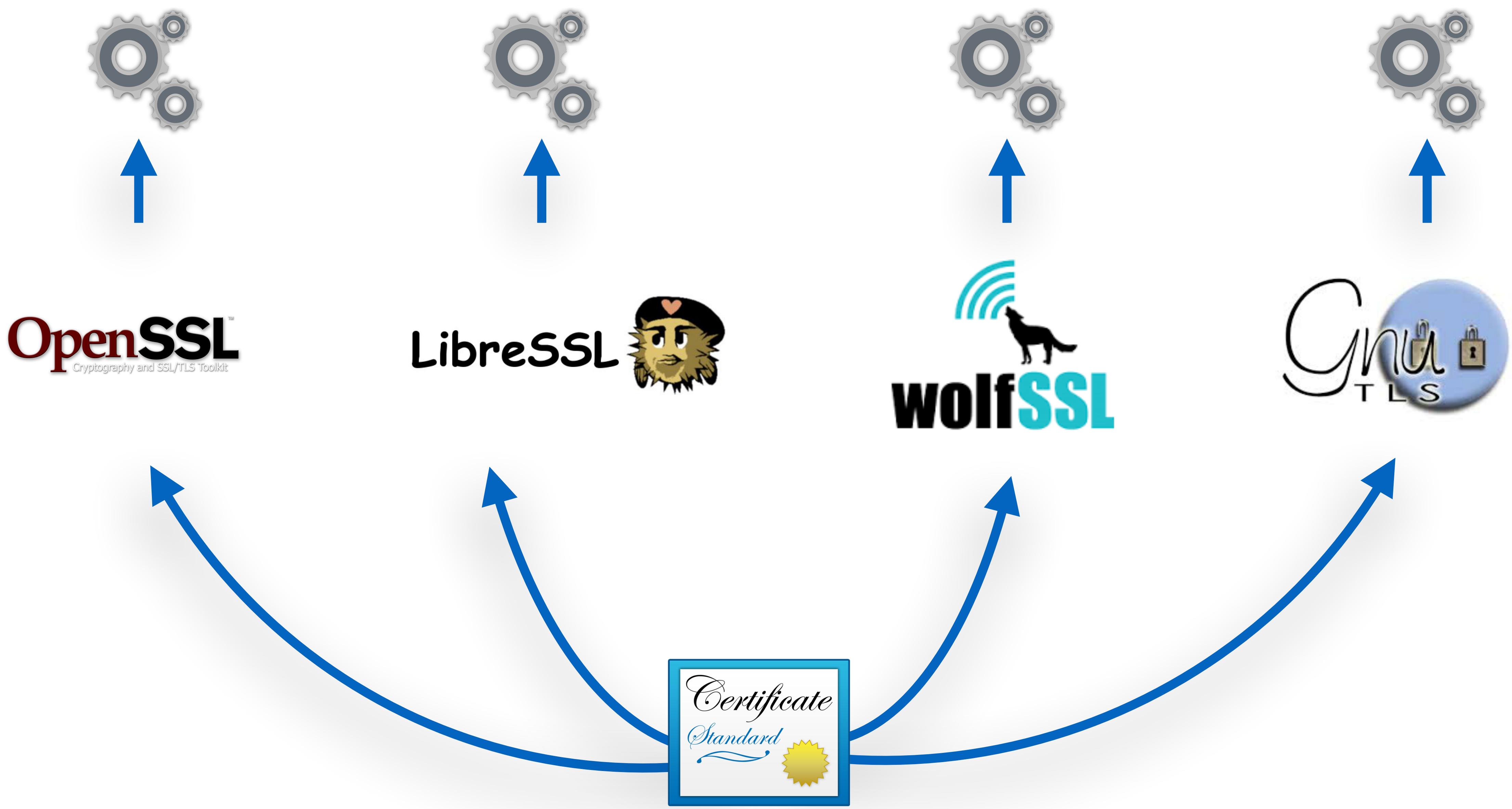
wolfSSL

Gnu TLS

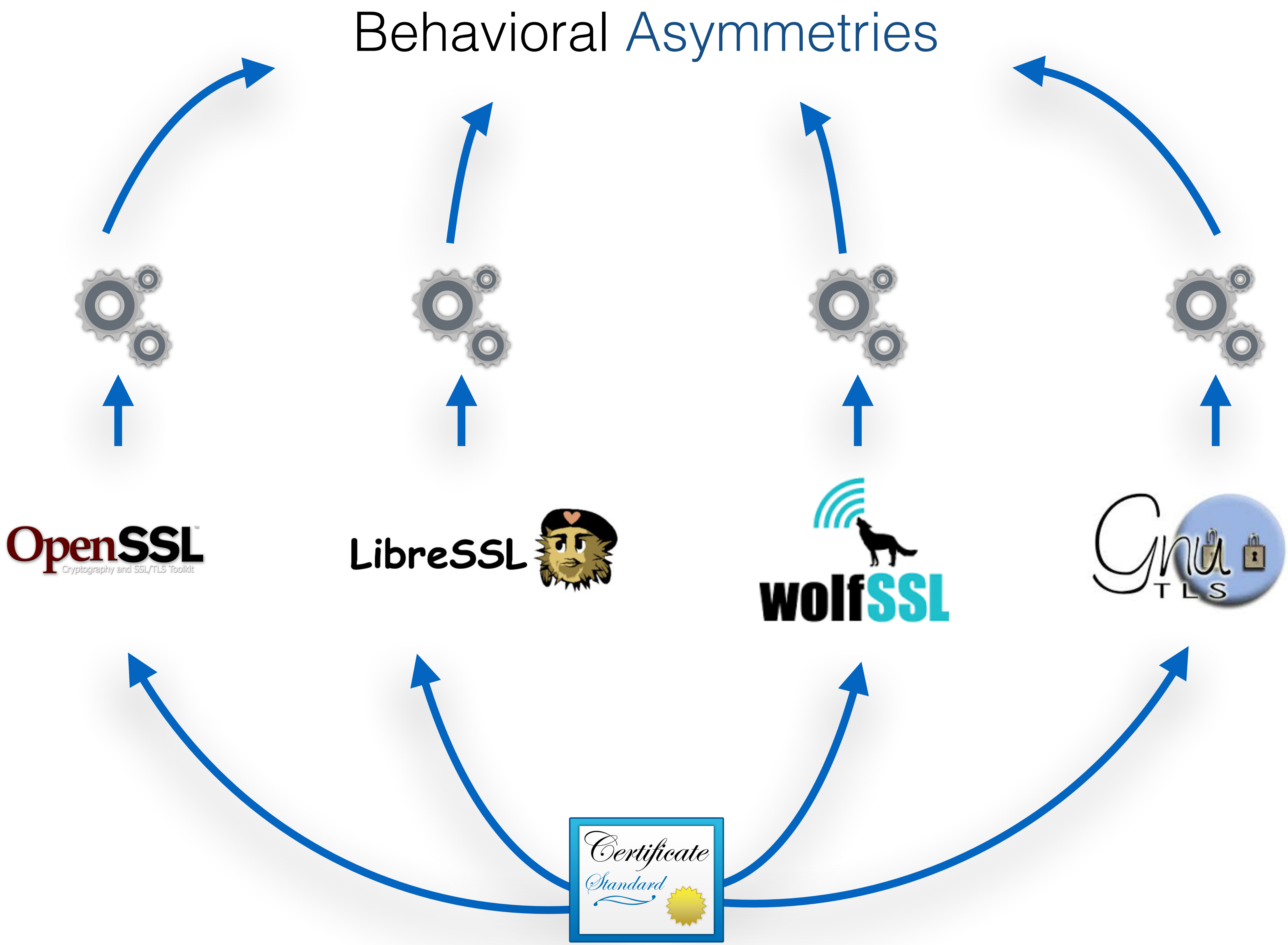
Differential Testing under δ-diversity



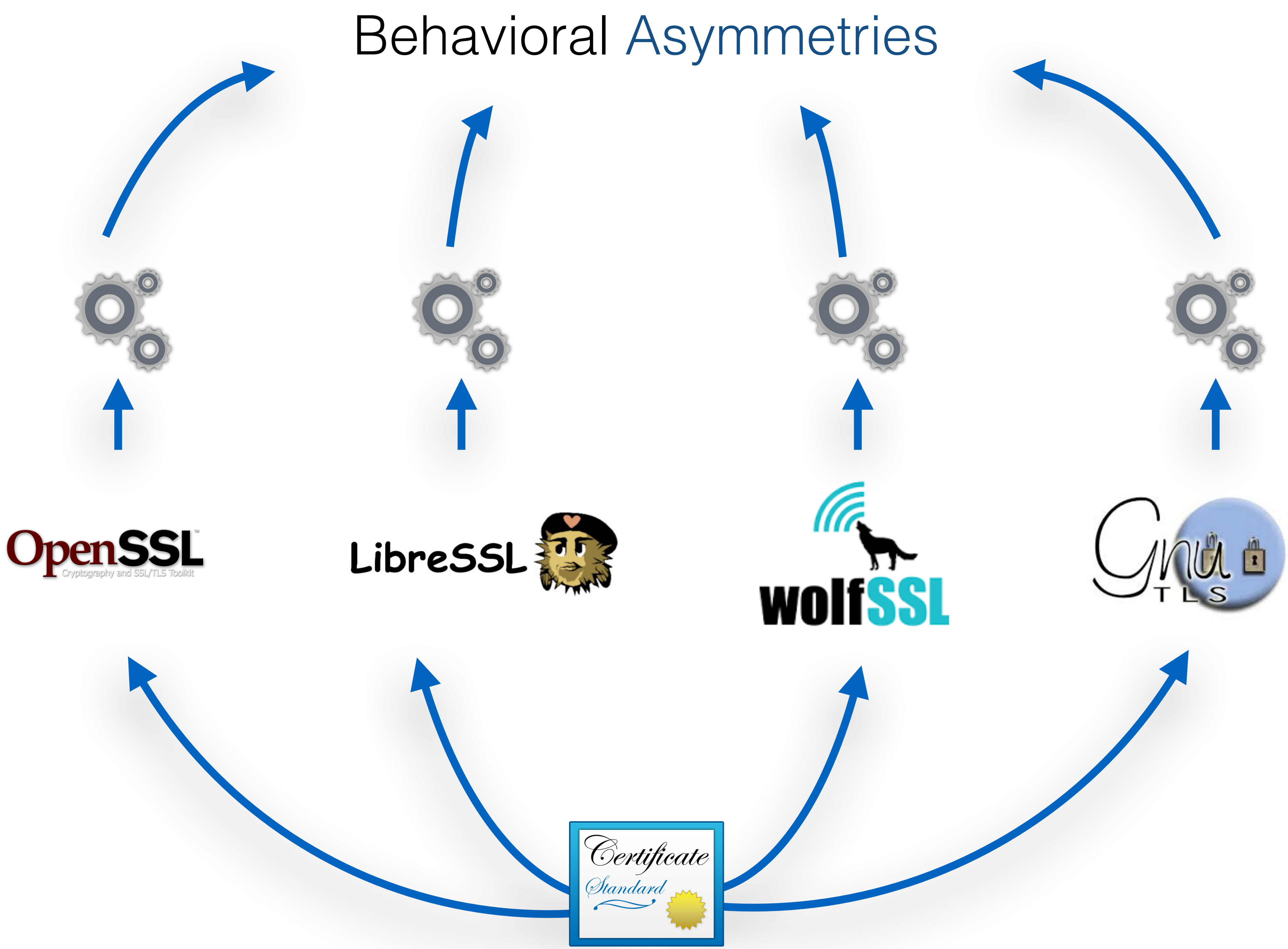
Differential Testing under δ-diversity



Differential Testing under δ-diversity



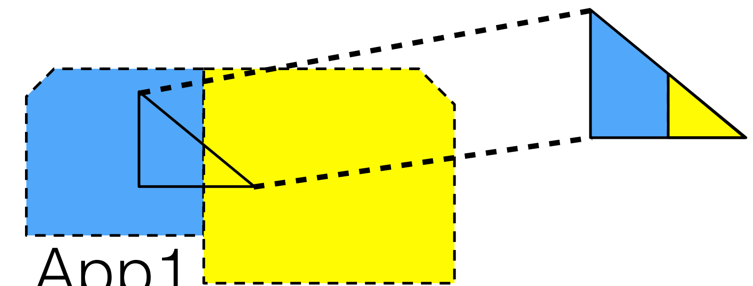
Differential Testing under δ-diversity



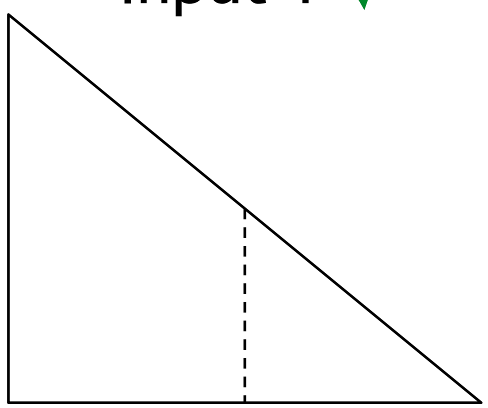
Differential Testing under δ-diversity

- Two examples:
  - Gray-box
  - Black-box
- Both outperform code coverage

Path δ-diversity: gray-box



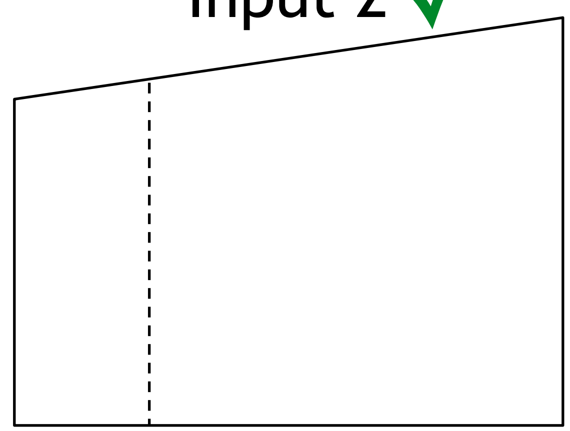
Input 1 ✓



A_1 A_2 A_3 | B_1

(3,1)

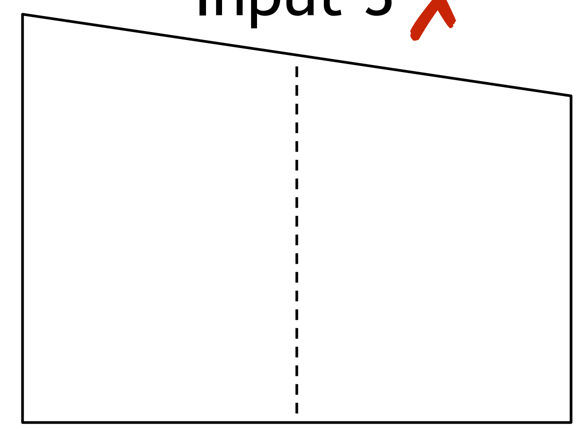
Input 2 ✓



A_1 A_2 | B_1 B_2 B_3

(2,3)

Input 3 ✗

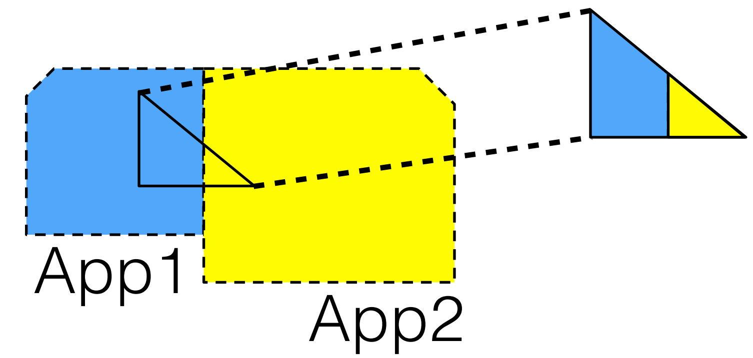


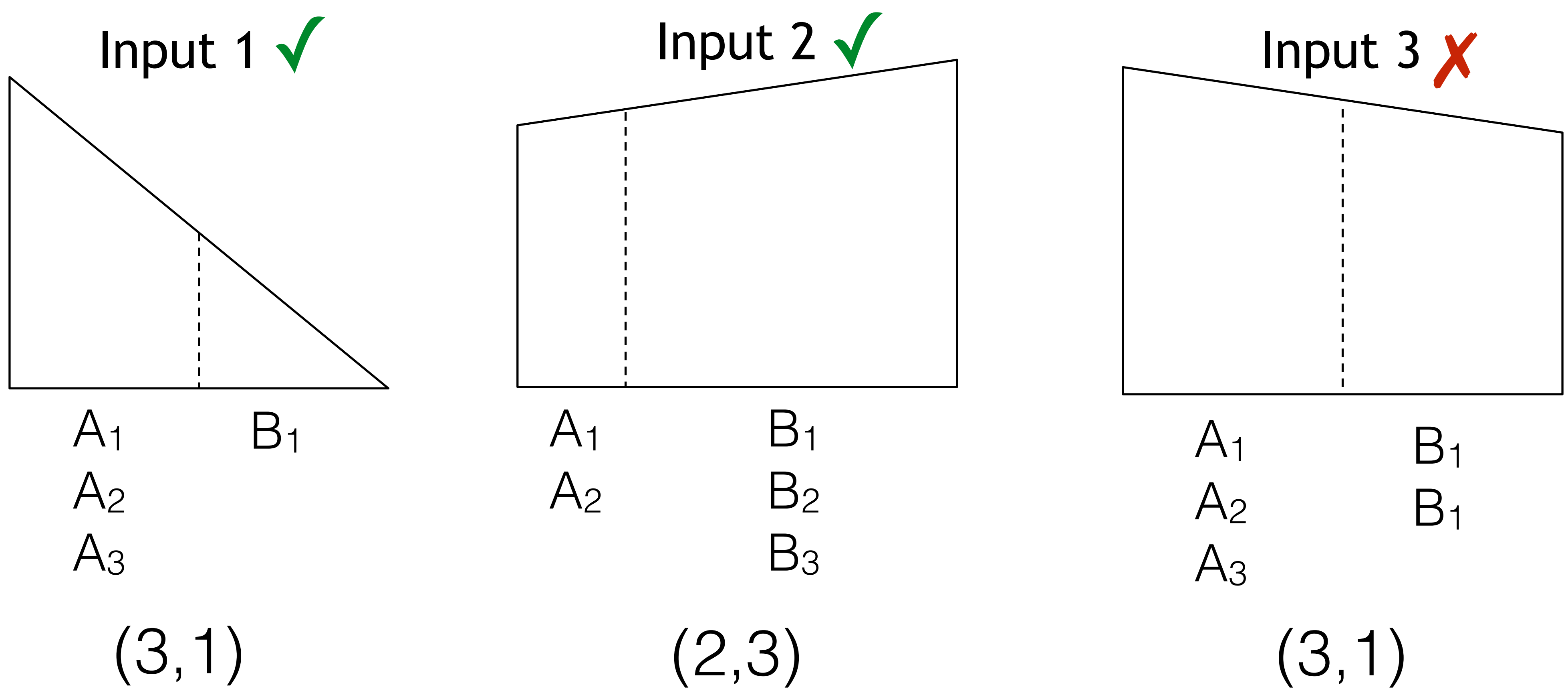
A_1 A_2 A_3 | B_1 B_1

(3,1)

Keep track of unique edges

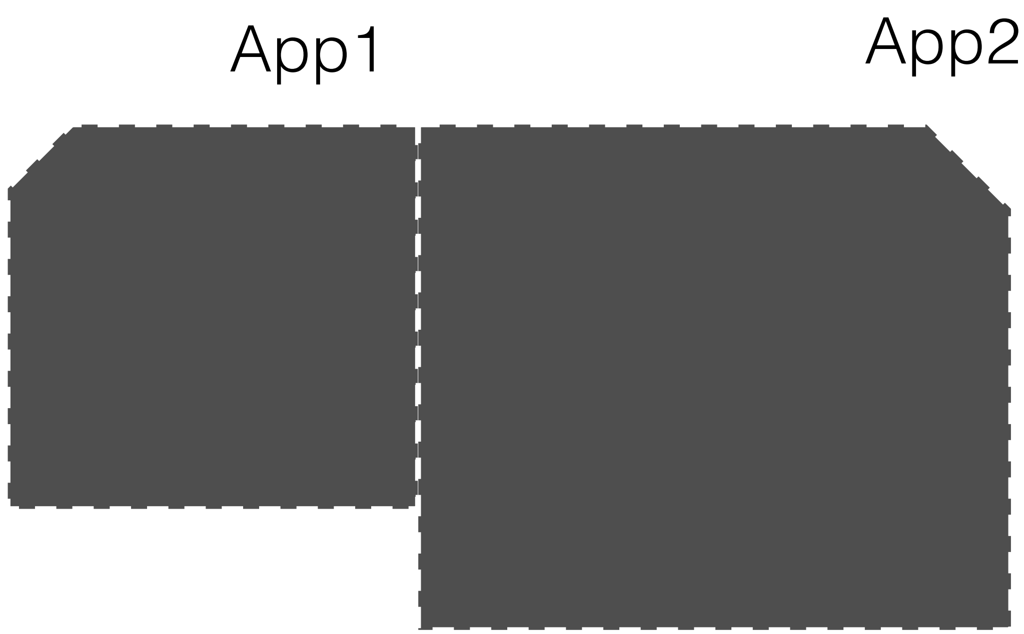
Path δ-diversity: gray-box





Keep track of unique edges

Output δ-diversity: black-box



All possible ~~code paths~~

Return values
Error codes
Exception messages

	Input	App1	App2
✓		0x0	0x0
✓		0x1	0xdead
✗		0x0	0x0
✓		0x0	0xbeef
✓		0xbabe	0x0
✗		0x0	0xbeef

δ-diversity

- Domain Independence
- Efficient differential guidance

Implementation

- NEZHA prototype
- Gray-box and black-box δ-diversity metrics
    - Path δ-diversity (fine & coarse)
    - Output δ-diversity
- Domain-independent input generation
    - Evolutionary, feedback-guided
- Built upon libFuzzer with NEZHA-specific hooks
- 1545 lines of C++

Use cases

- SSL libraries
- PDF readers
- ClamAV & XZ Parsers

Use cases

- SSL libraries
- PDF readers
- ClamAV & XZ Parsers

Certificate Verification Discrepancies

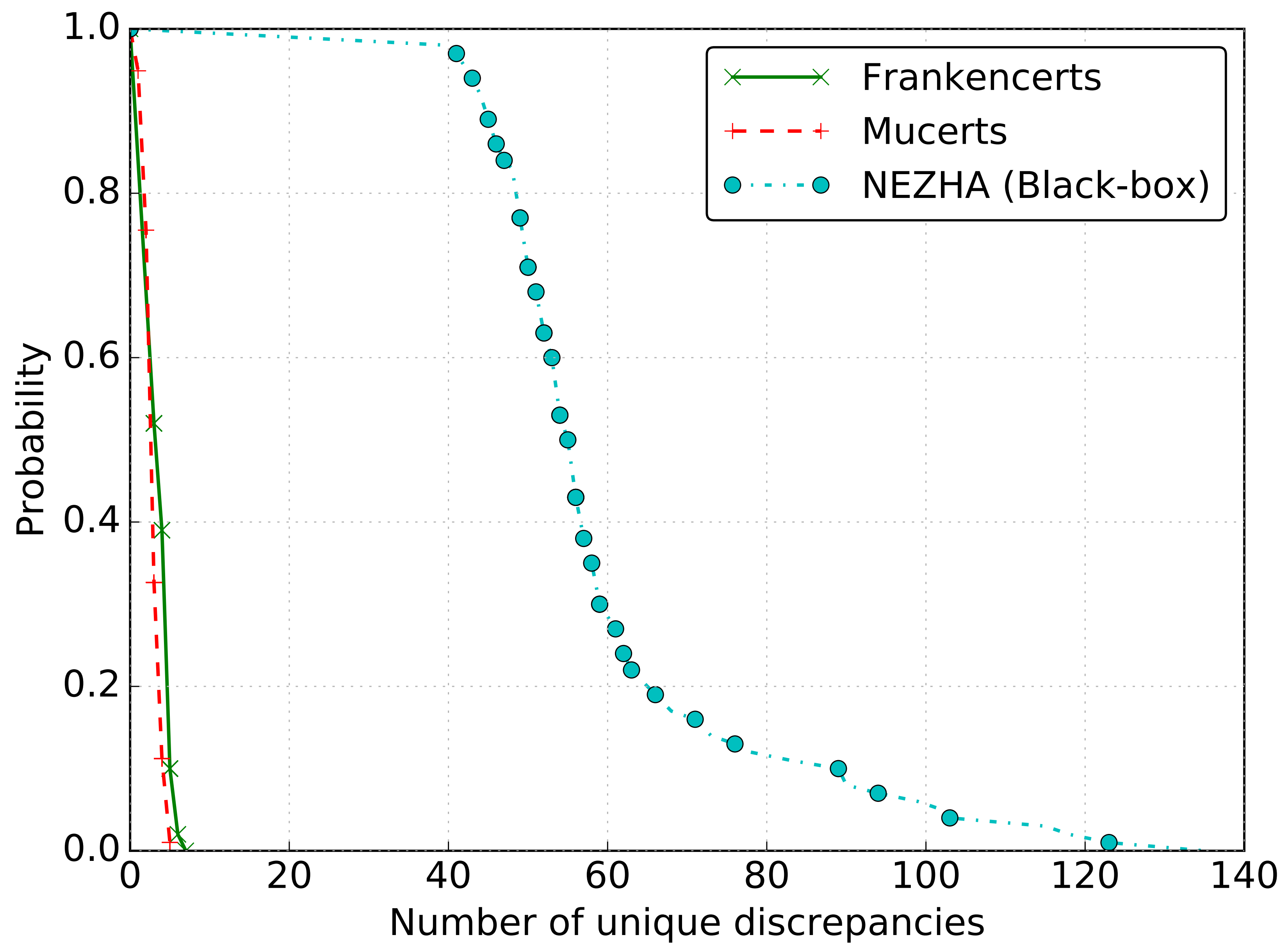
One library accepts one certificate, while another rejects it with an error code.

	LibreSSL	BoringSSL	wolfSSL	mbedTLS	GnuTLS
OpenSSL	10	1	8	33	25
LibreSSL	-	11	8	19	19
BoringSSL	-	-	8	33	25
wolfSSL	-	-	-	6	8
mbedTLS	-	-	-	-	31

Unique pair-wise discrepancies (based on error code tuples)

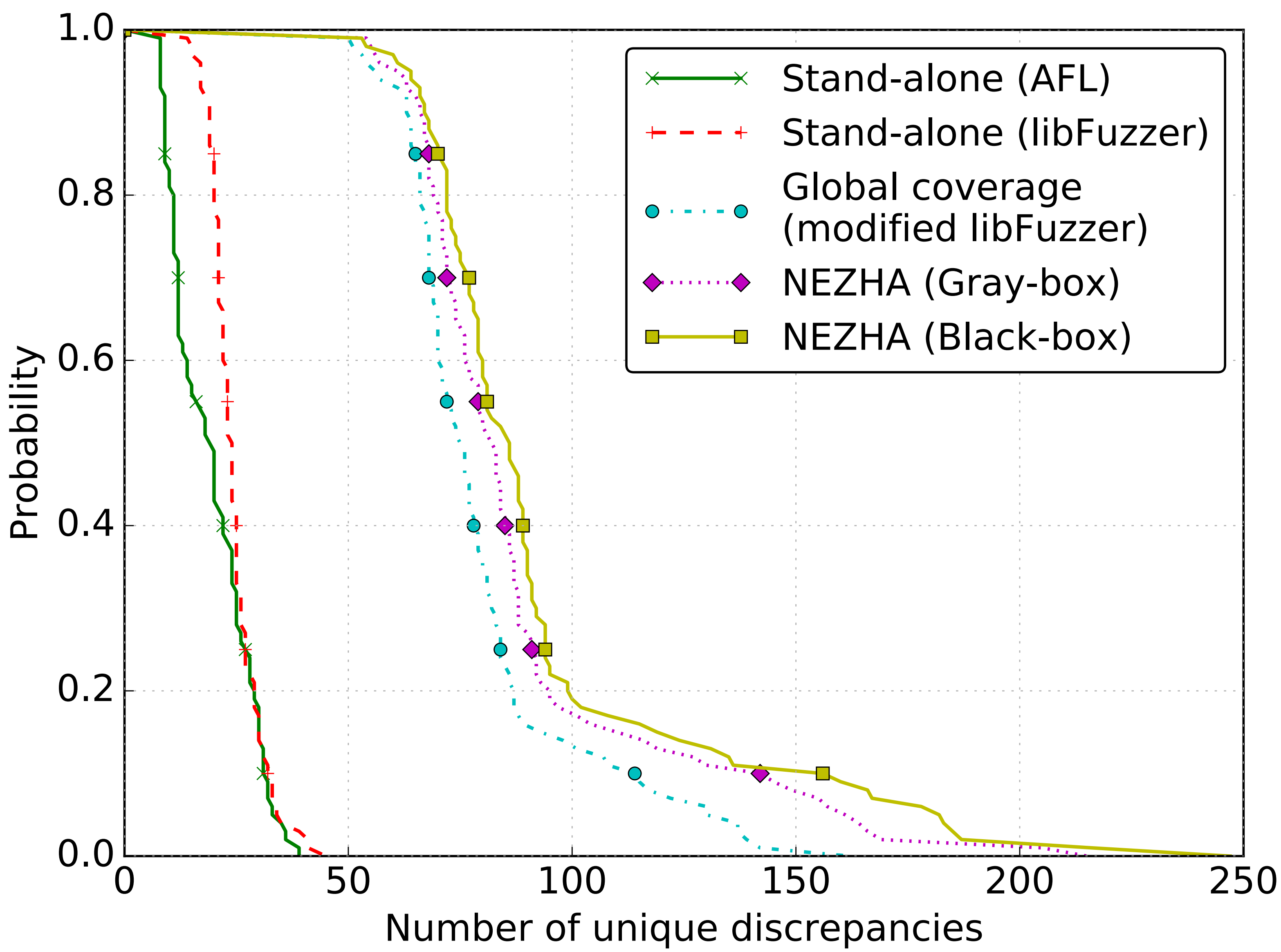
NEZHA vs domain-specific frameworks

- 52x more discrepancies than *Frankencerts*
- 27x more discrepancies than *Mucerts*



NEZHA vs popular evolutionary fuzzers

- Adapted popular evolutionary fuzzers for differential testing
    - Code coverage in single application
    - Global code coverage
- 6x more discrepancies than testing on a single application
- 30% more discrepancies than modified libFuzzer



Sample Bugs uncovered by NEZHA

(disclosed and patched)

Experimental Setting

Application Category	Tests
SSL Libraries	OpenSSL, LibreSSL, BoringSSL, GnuTLS, wolfSSL, mbedTLS
PDF Readers	Evince PDF, MuPDF, Xpdf
Parsers	ClamAV vs binutils ClamAV vs xz

BUG 1: Malicious ELF can evade ClamAV detection

CLAMAV (ELF parsing engine)

```
static int cli_elf_fileheader(...) {
  ...
  switch(file_hdr->hdr64.e_ident[4]) {
    case 1:
      ...
    case 2:
      ...
    default:
      ...
      return CL_EFORMAT;
  ...
```

BUG 1: Malicious ELF can evade ClamAV detection

CLAMAV (ELF parsing engine)

```
static int cli_elf_fileheader(...) {
  ...
  switch(file_hdr->hdr64.e_ident[4]) {
    case 1:
      ...
    case 2:
      ...
    default:
      ...
      return CL_EFORMAT;
  ...
```

Bug 1: Malicious ELF can evade ClamAV detection

ClamAV (ELF parsing engine)

```
static int cli_elf_fileheader(...) {
  ...
  switch(file_hdr->hdr64.e_ident[4]) {
    case 1:
      ...
    case 2:
      ...
    default:
      ...
      return CL_EFORMAT;
  ...
```

Linux ELF loader

```
static int load_elf_binary(struct linux_binprm *bprm) {
  ...
  retval = -ENOEXEC;
  if (memcmp(loc->elf_ex.e_ident, ELFMAG, SELFMAG) != 0)
    goto out;
  if (loc->elf_ex.e_type != ET_EXEC &&
        loc->elf_ex.e_type != ET_DYN)
    goto out;
  if (!elf_check_arch(&loc->elf_ex))
    goto out;
  ...
```

BUG 1: Malicious ELF can evade ClamAV detection

CLAMAV (ELF parsing engine)

```
static int cli_elf_fileheader(...) {
  ...
  switch(file_hdr->hdr64.e_ident[4]) {
    case 1:
      ...
    case 2:
      ...
    default:
      ...
      return CL_EFORMAT;
  ...
```

LINUX ELF loader

```
static int load_elf_binary(struct linux_binprm *bprm) {
  ...
  retval = -ENOEXEC;
  if (memcmp(loc->elf_ex.e_ident, ELFMAG, SELFMAG) != 0)
    goto out;
  if (loc->elf_ex.e_type != ET_EXEC &&
       loc->elf_ex.e_type != ET_DYN)
    goto out;
  if (!elf_check_arch(&loc->elf_ex))
    goto out;
  ...
```

BUG 2: LibreSSL misinterprets time in ASN.1 format

Time fields can be formatted in 2 ways:

UTC: YYMMDDHHMMSSZ (13 char long)

GMT: YYYYMMDDHHMMSSZ (15 char long)

BUG 2: LibreSSL misinterprets time in ASN.1 format

Time fields can be formatted in 2 ways:

UTC: YYMMDDHHMMSSZ (13 char long)

GMT: YYYYMMDDHHMMSSZ (15 char long)

LibreSSL ignores the ASN.1 time format tag, and determines format based on length of field

Bug 2: LibreSSL misinterprets time in ASN.1 format

```
int asn1_time_parse(..., size_t len, ..., int mode) {
  ...
  int type = 0;
  ...
  /* Constrain to valid lengths. */
  if (len != UTCTIME_LENGTH && len != GENTIME_LENGTH)
    return (-1);
  ...
  switch (len) {
  case GENTIME_LENGTH:
    // mode is "ignored" -- configured to 0 here
    if (mode == V_ASN1_UTCTIME)
      return (-1);
    ...
    type = V_ASN1_GENERALIZEDTIME;
  case UTCTIME_LENGTH:
    if (type == 0) {
      if (mode == V_ASN1_GENERALIZEDTIME)
        return (-1);
      type = V_ASN1_UTCTIME;
    }
    // parse time as UTCTIME
```

LibreSSL ignores the ASN.1 time format tag, and determines format based on length of field

BUG 2: LibreSSL misinterprets time in ASN.1 format

```
int asn1_time_parse(..., size_t len, ..., int mode) {
  ...
  int type = 0;
  ...
  /* Constrain to valid lengths. */
  if (len != UTCTIME_LENGTH && len != GENTIME_LENGTH)
    return (-1);
  ...
  switch (len) {
  case GENTIME_LENGTH:
    // mode is "ignored" -- configured to 0 here
    if (mode == V_ASN1_UTCTIME)
      return (-1);
    ...
    type = V_ASN1_GENERALIZEDTIME;
  case UTCTIME_LENGTH:
    if (type == 0) {
      if (mode == V_ASN1_GENERALIZEDTIME)
        return (-1);
      type = V_ASN1_UTCTIME;
    }
    // parse time as UTCTIME
```

LibreSSL ignores the ASN.1 time format tag, and determines format based on length of field

Bug 2: LibreSSL misinterprets time in ASN.1 format

```c
int asn1_time_parse(..., size_t len, ..., int mode) {
  ...
  int type = 0;
  ...
  /* Constrain to valid lengths. */
  if (len != UTCTIME_LENGTH && len != GENTIME_LENGTH)
    return (-1);
  ...
  switch (len) {
  case GENTIME_LENGTH:
    // mode is "ignored" -- configured to 0 here
    if (mode == V_ASN1_UTCTIME)
      return (-1);
    ...
    type = V_ASN1_GENERALIZEDTIME;
  case UTCTIME_LENGTH:
    if (type == 0) {
      if (mode == V_ASN1_GENERALIZEDTIME)
        return (-1);
      type = V_ASN1_UTCTIME;
    }
    // parse time as UTCTIME
```

LibreSSL ignores the ASN.1 time format tag, and determines format based on length of field

Jan 1 01:01:00 2012 GMT can interpreted as Dec 1 01:01:01 2020 GMT

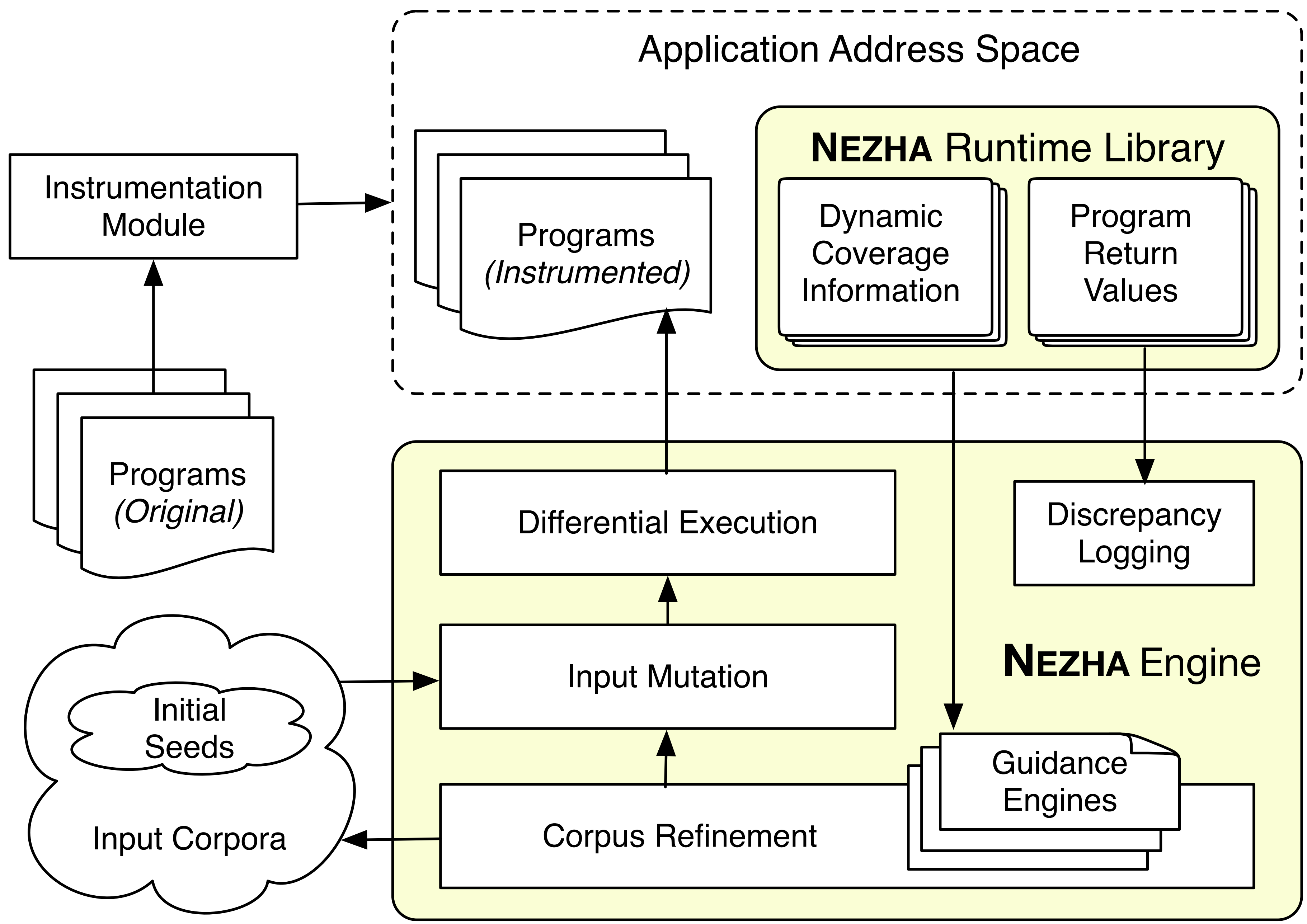
Conclusions

- δ-diversity outperforms code coverage for differential testing
- NEZHA: Domain independent, efficient differential testing
- Differential testing should be integrated, when possible, into the testing cycle

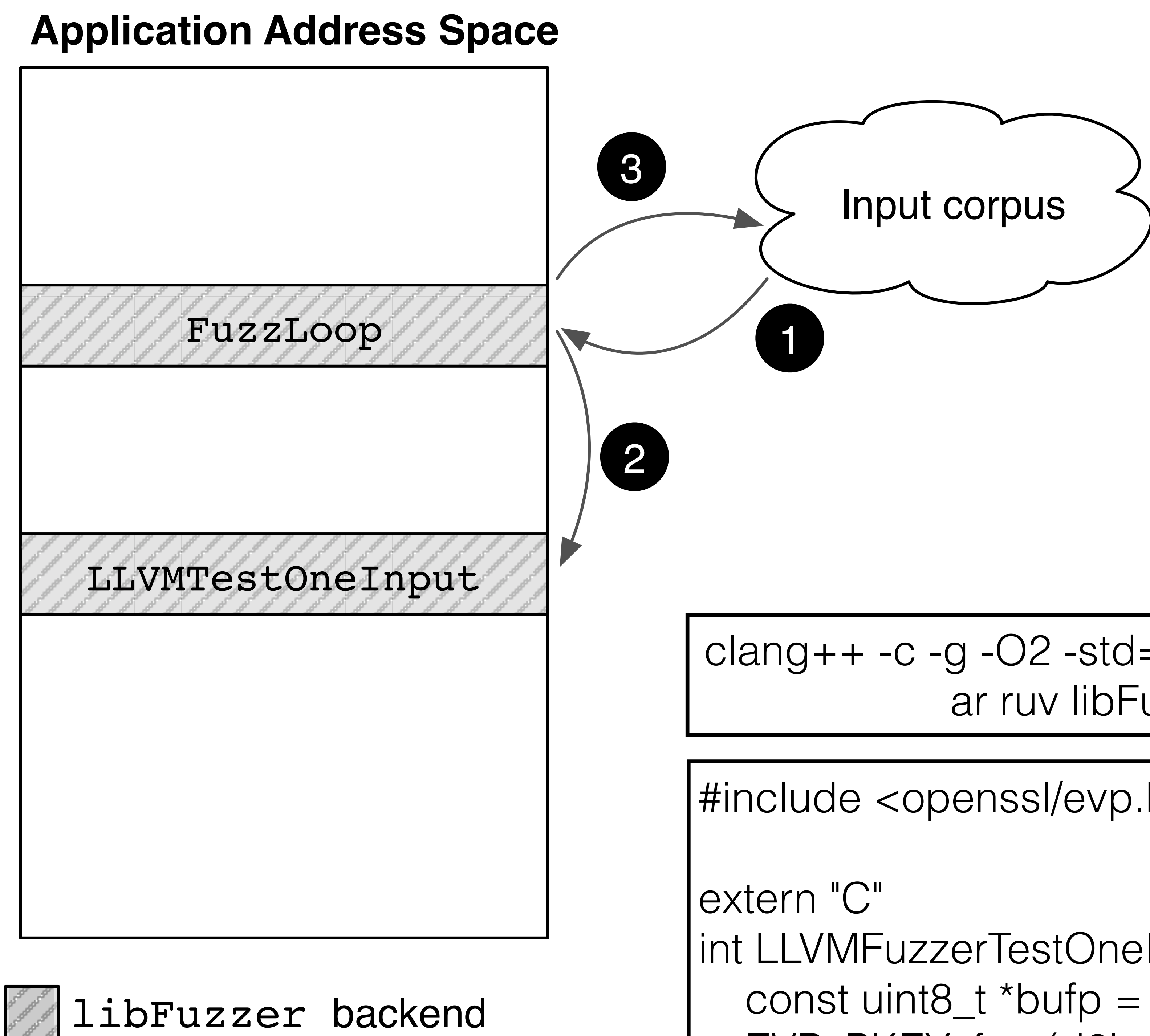
https://github.com/nezha-dt

Backup Slides

NEZHA: Architecture



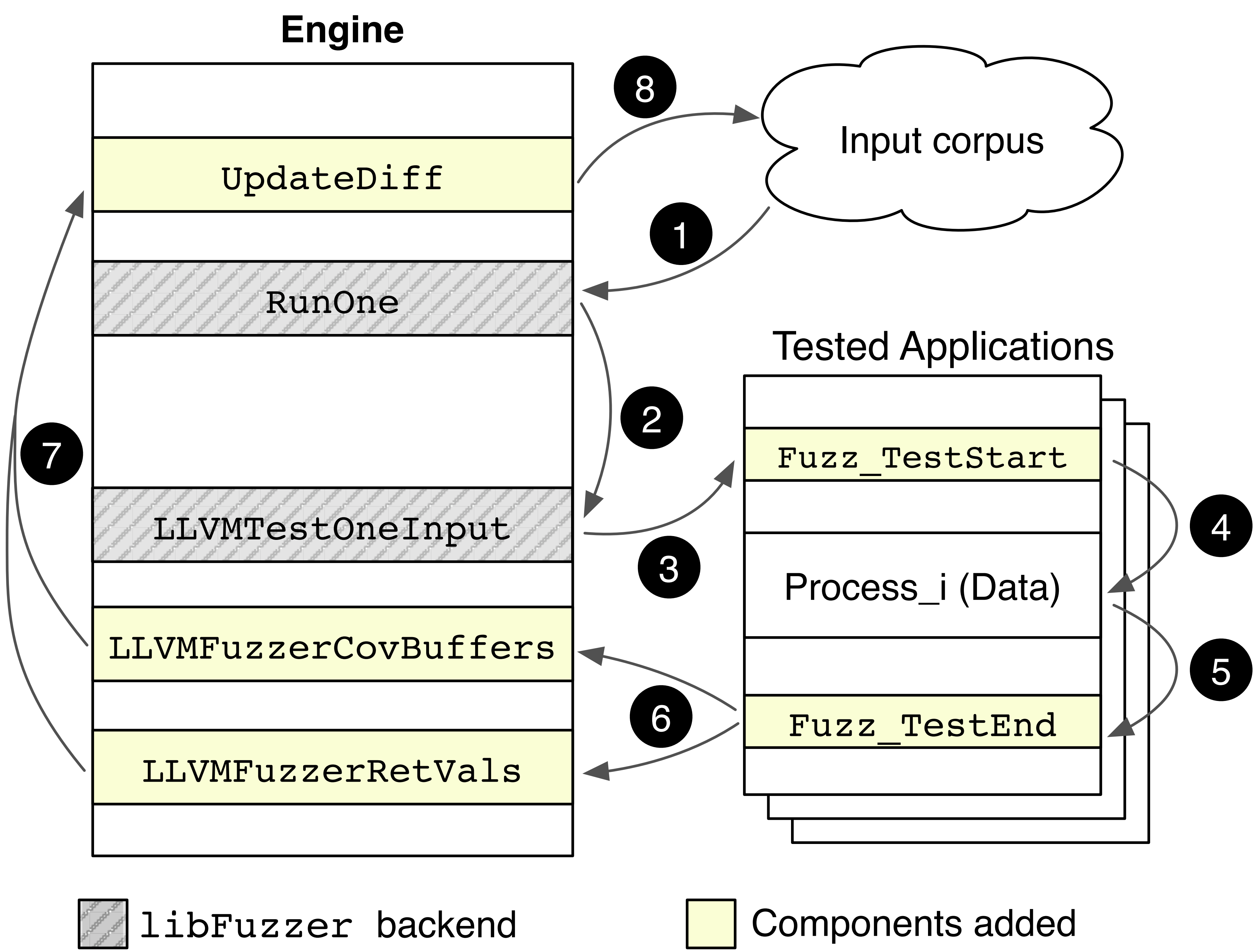
NEZHA: Architecture




```
clang++ -c -g -O2 -std=c++11 Fuzzer/*.cpp -IFuzzer
ar ruv libFuzzer.a Fuzzer*.o
```

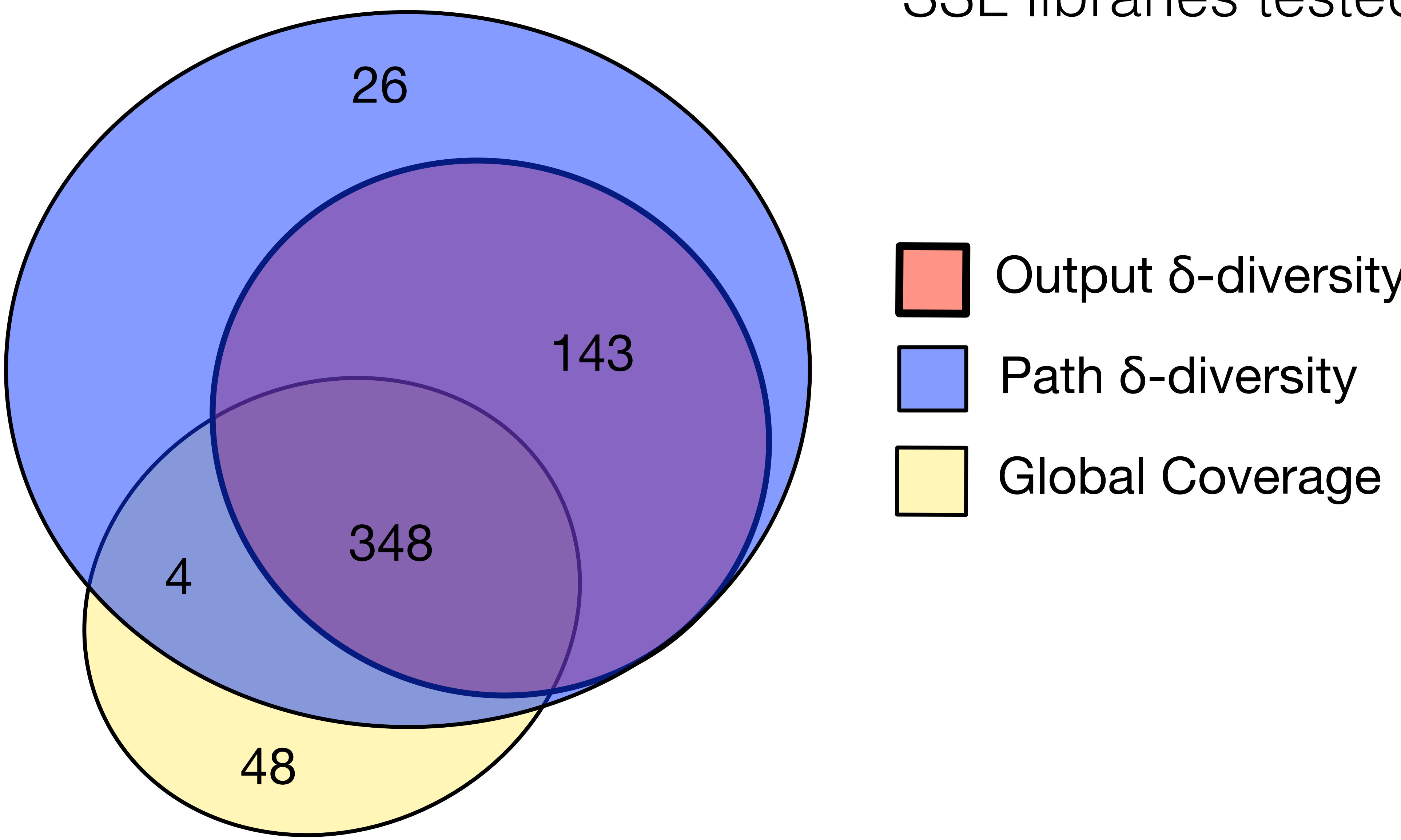
```
#include <openssl/evp.h>

extern "C"
int LLVMFuzzerTestOneInput(const uint8_t *buf, size_t len) {
    const uint8_t *bufp = buf;
    EVP_PKEY_free(d2i_AutoPrivateKey(NULL, &bufp, len));
    return 0;
}
```

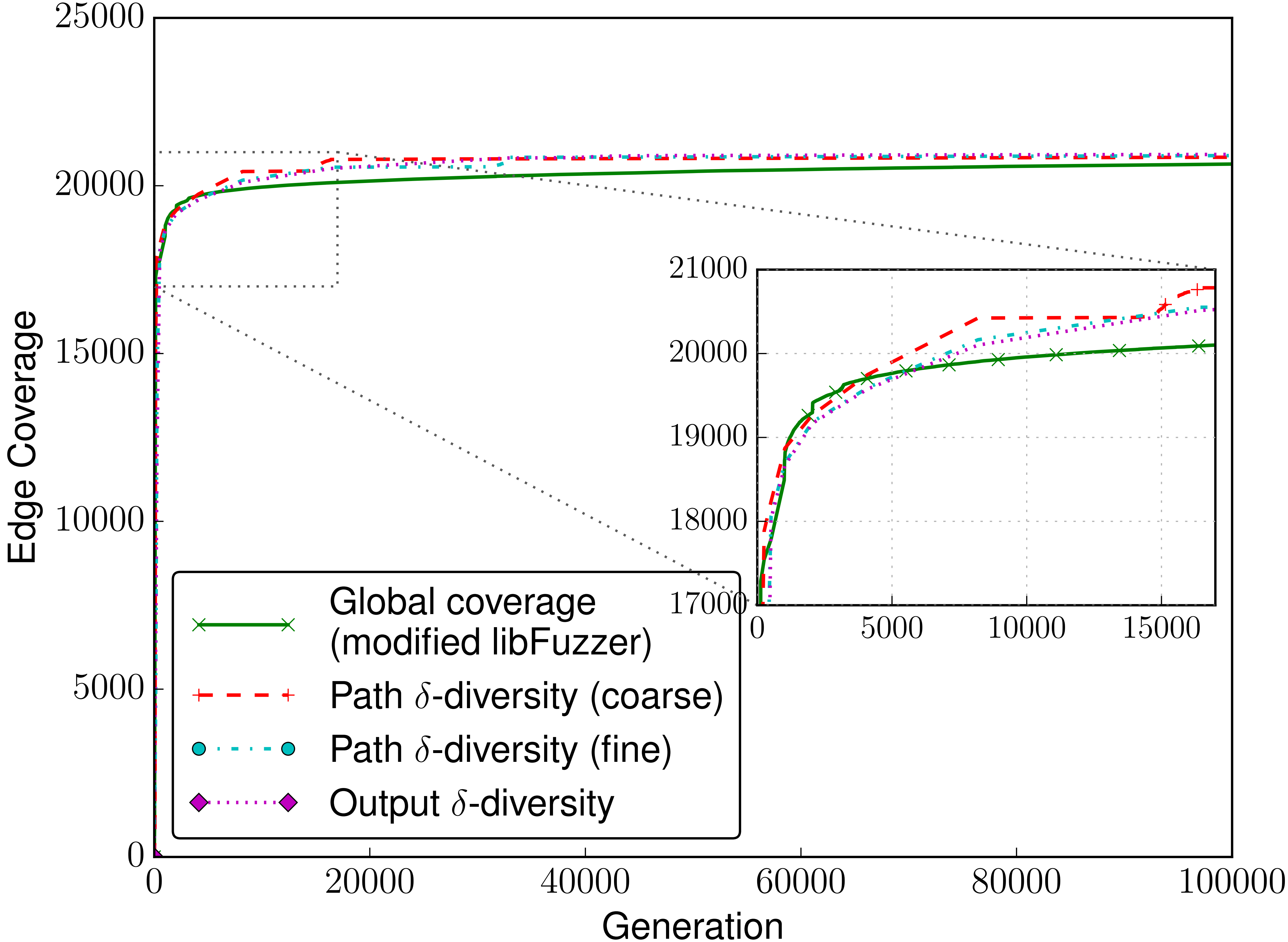
NEZHA: Architecture



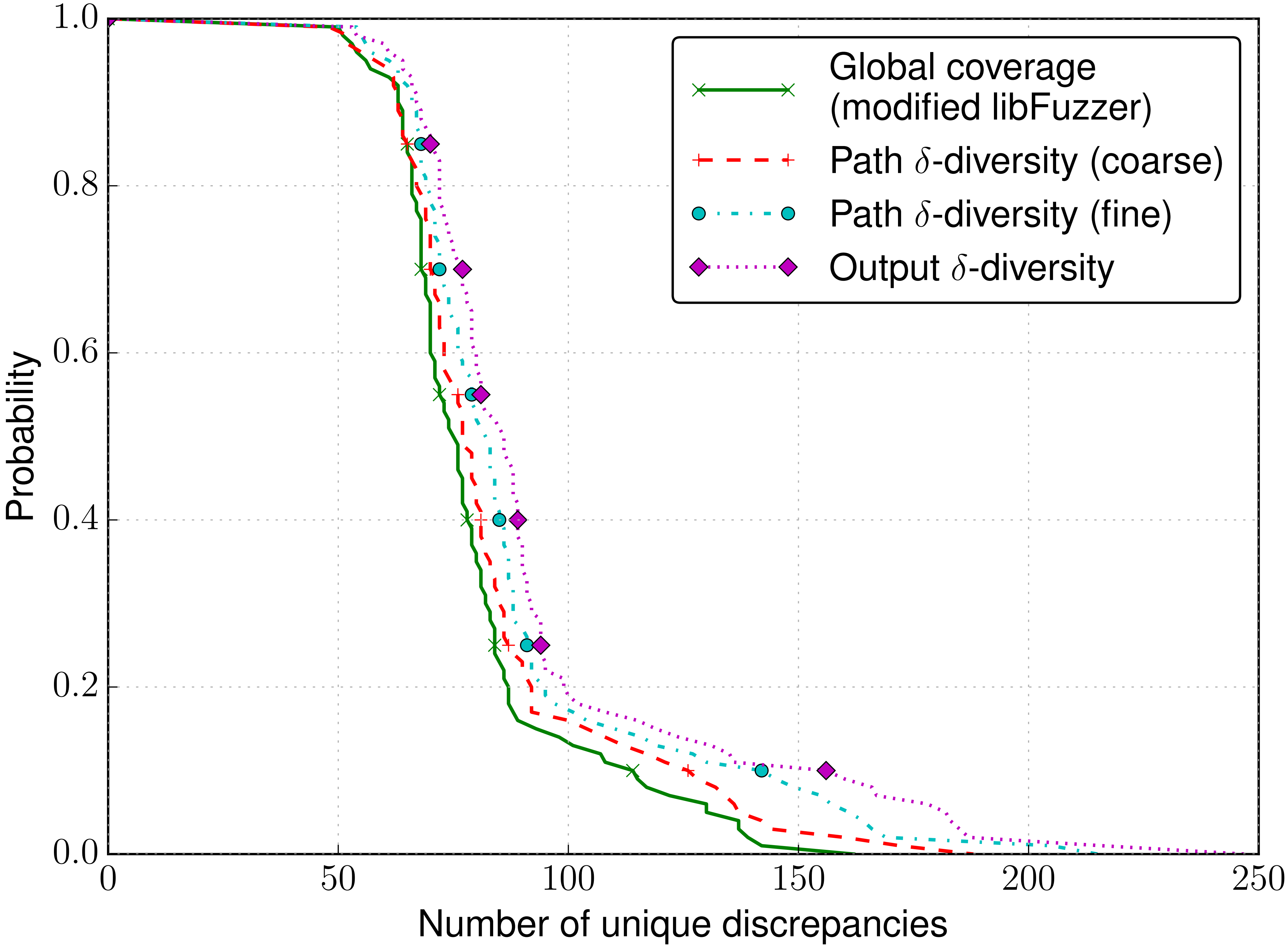
Discrepancy Distribution for SSL/TLS Libs

Same Inputs / Different mode
SSL libraries tested

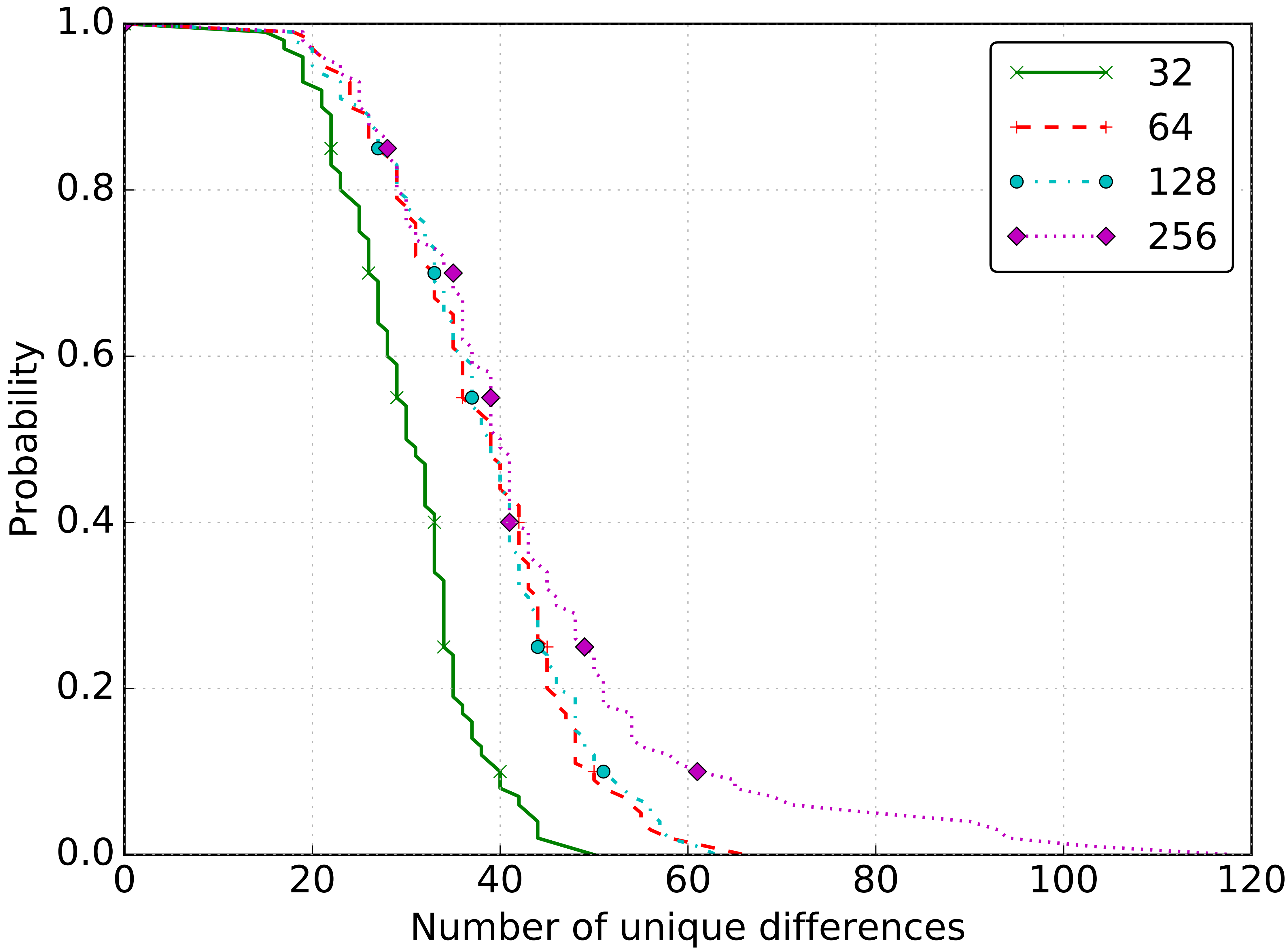




Edge Coverage
Generation
Global coverage
(modified libFuzzer)
Path δ-diversity (coarse)
Path δ-diversity (fine)
Output δ-diversity



Global coverage (modified libFuzzer)
Path δ-diversity (coarse)
Path δ-diversity (fine)
Output δ-diversity
Probability
Number of unique discrepancies
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250


32
64
128
256
Probability
Number of unique differences
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
120