FairFuzz: A Targeted Mutation Strategy for Increasing Greybox Fuzz Testing Coverage

Caroline Lemieux, Koushik Sen

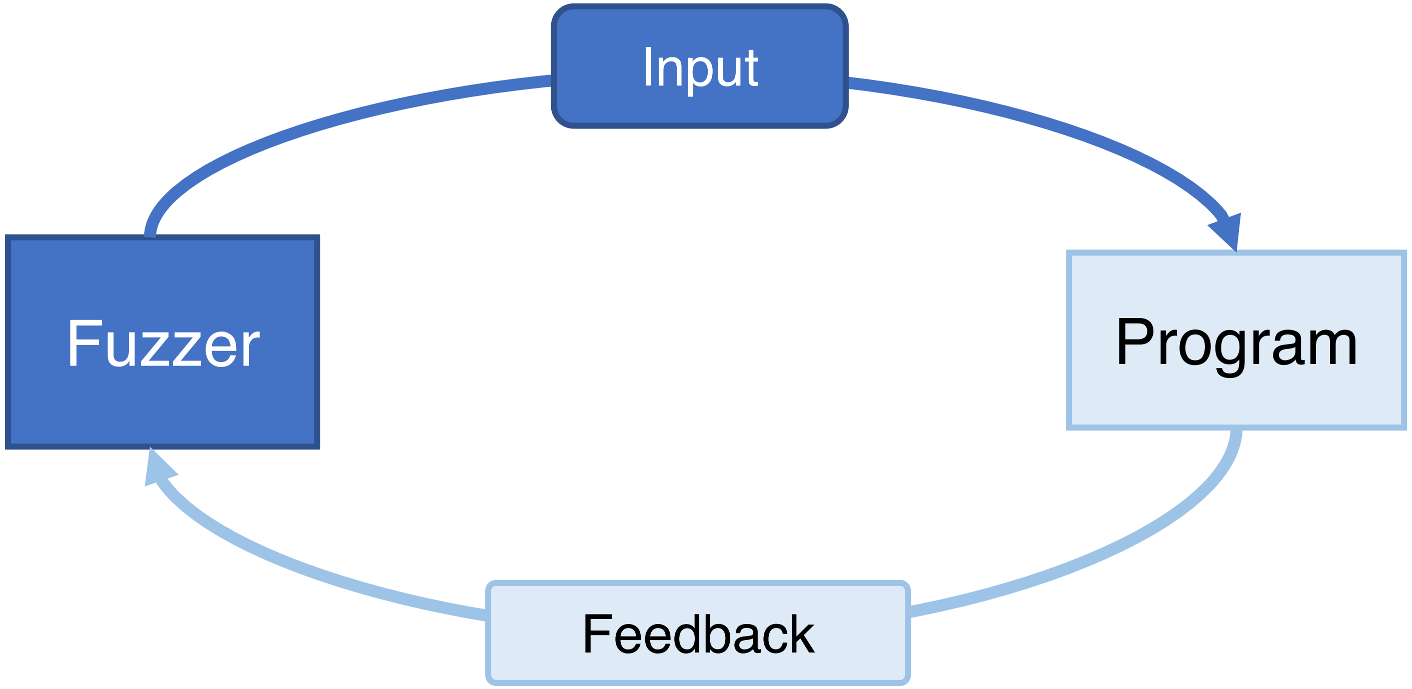
University of California, Berkeley

source: https://github.com/carolemieux/afl-rb

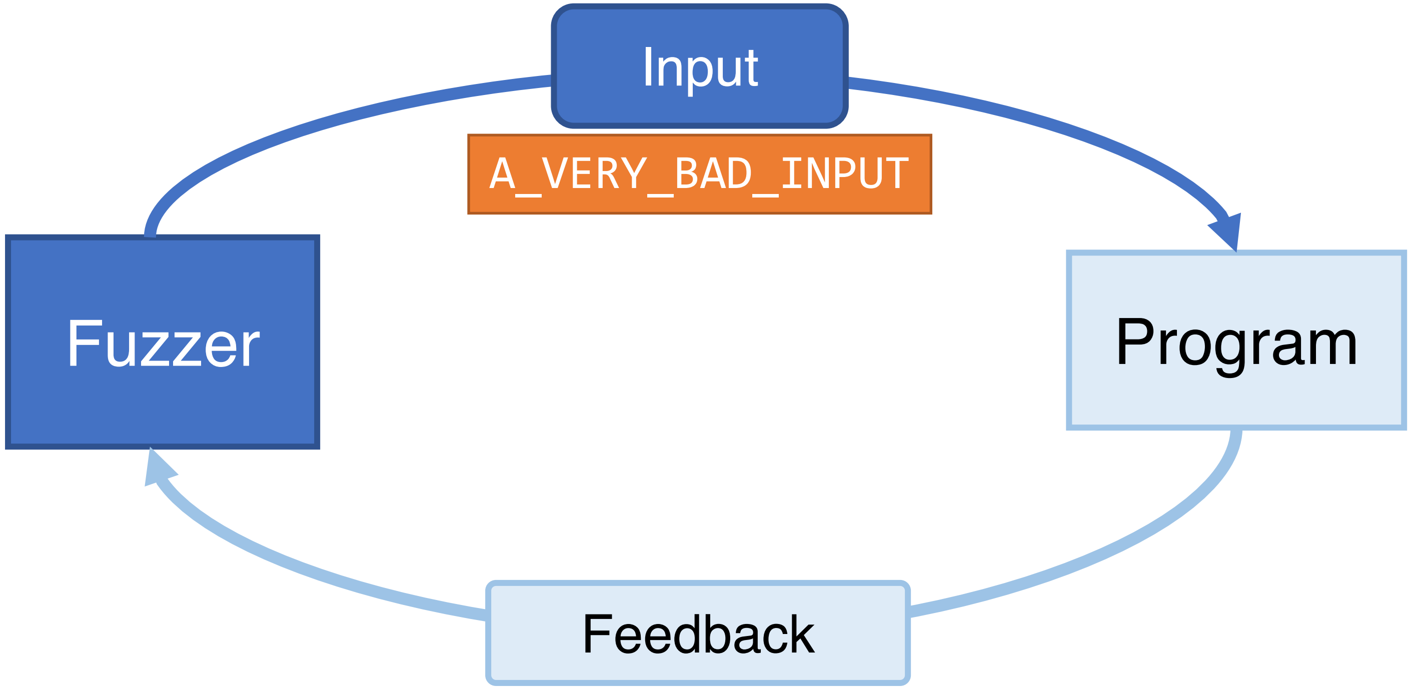
The Rise of Fuzz Testing

- Programs still have bugs.
- *Fuzz testing* has become very popular in practice and theory

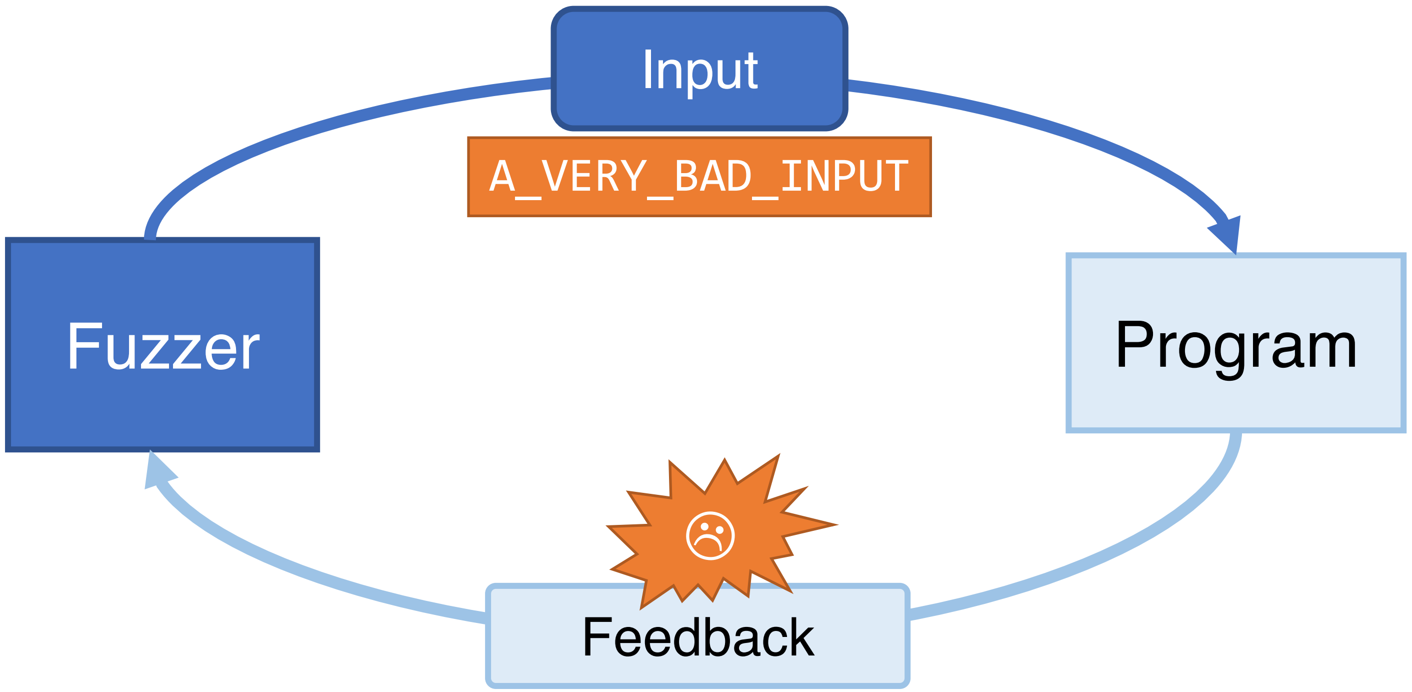
Fuzzing in One Slide



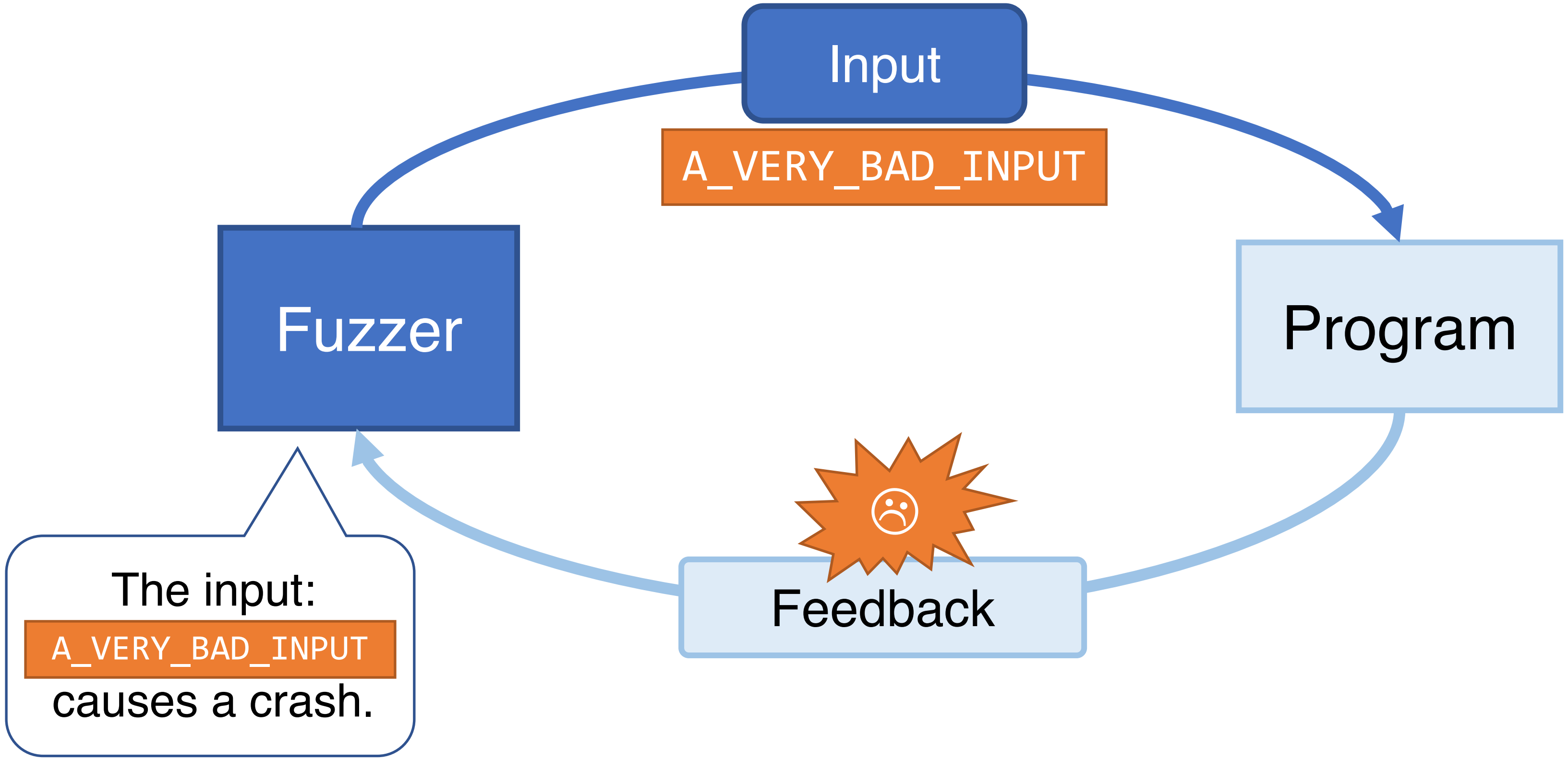
Fuzzing in One Slide



Fuzzing in One Slide



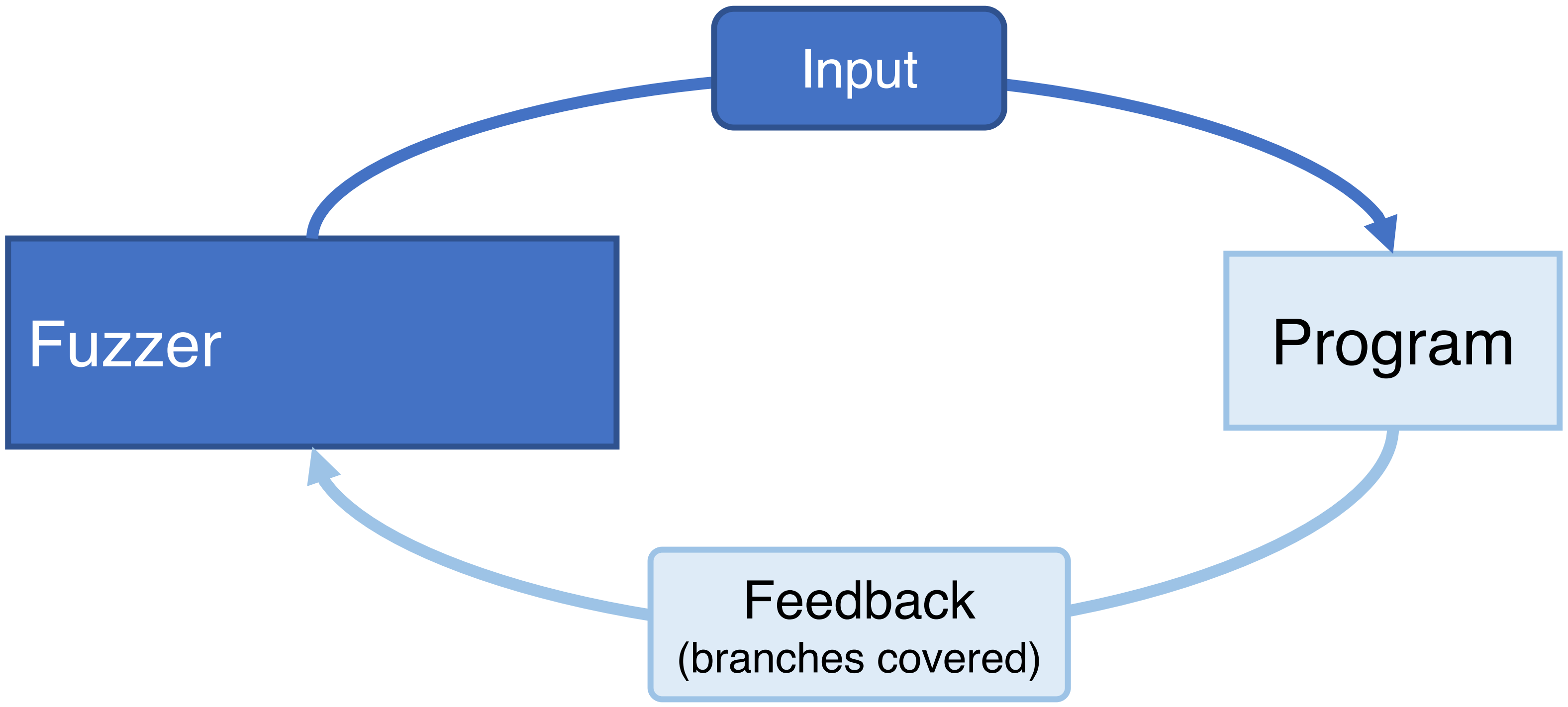
Fuzzing in One Slide



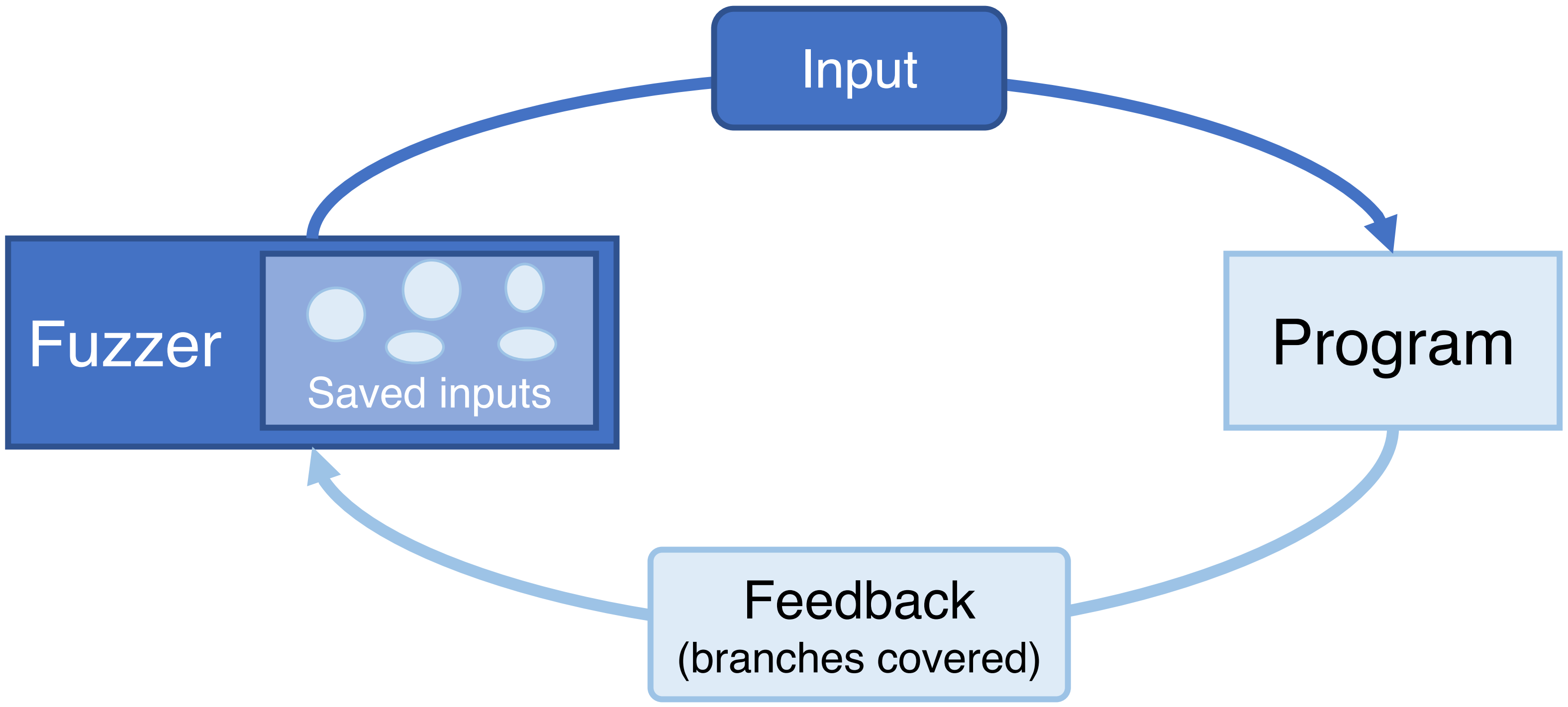
What Bugs Can Fuzzing Find?

- Most popular: basic correctness assertions (C/C++)
  - Segmentation faults
  - Anything address sanitizer can catch:
    - Buffer overflows
    - Use-after-frees
    - Etc…

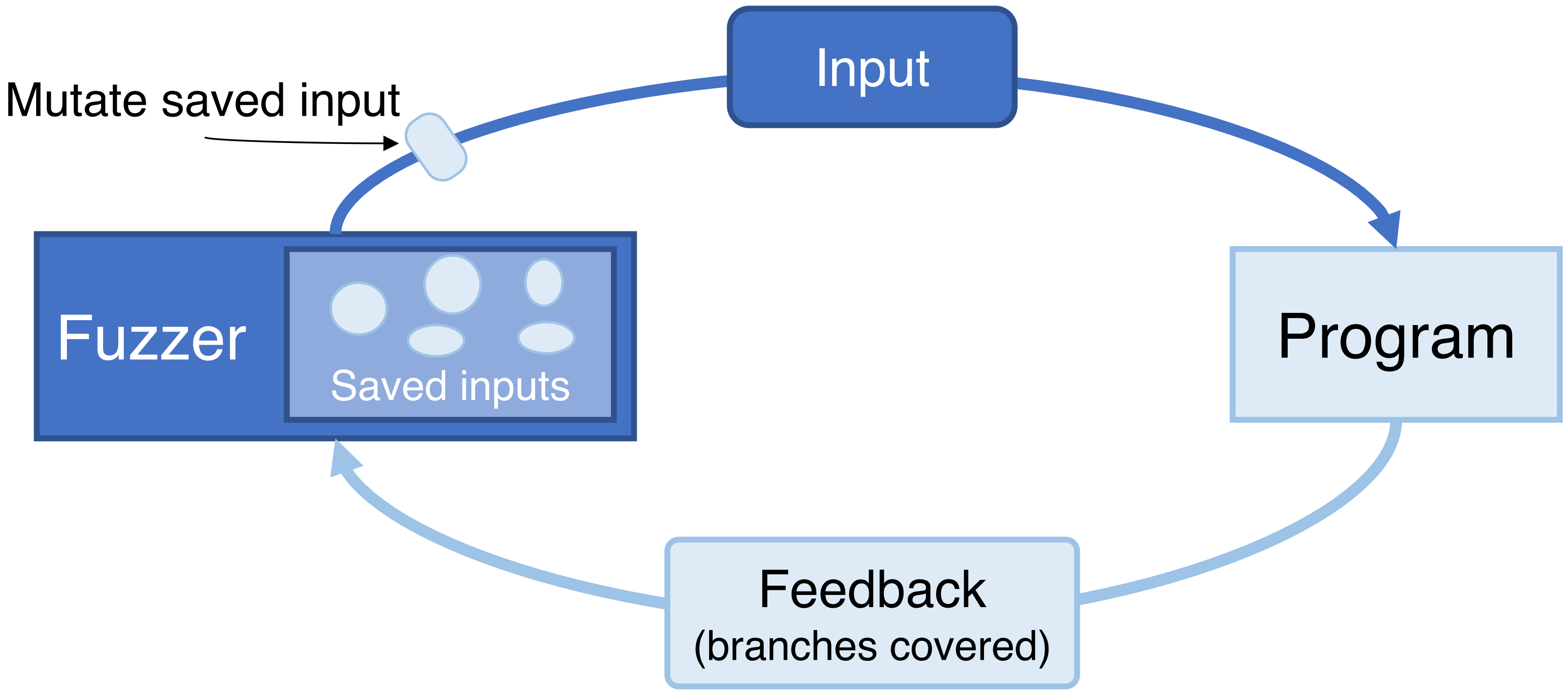
Coverage-Guided (Greybox) Mutational Fuzzing



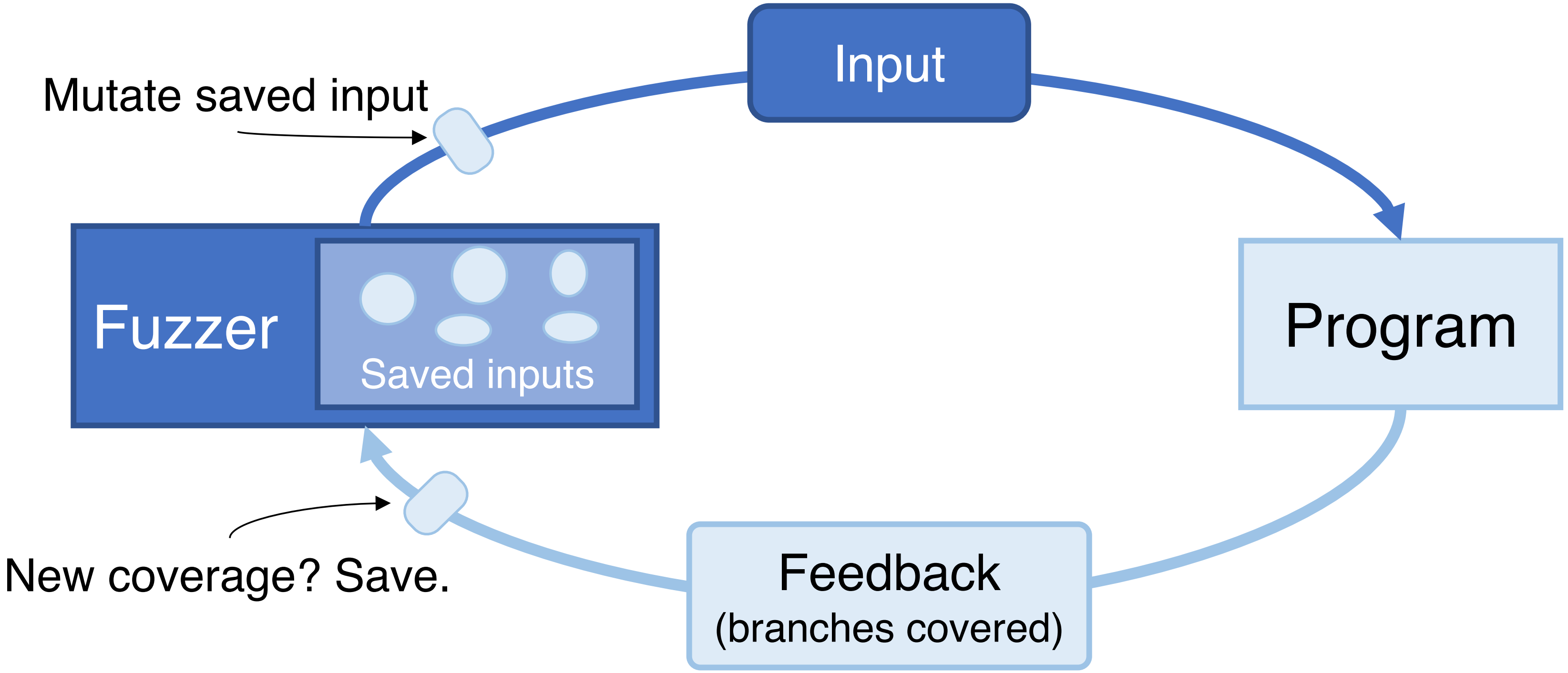
Coverage-Guided (Greybox) Mutational Fuzzing



Coverage-Guided (Greybox) Mutational Fuzzing



Coverage-Guided (Greybox) Mutational Fuzzing



What's Missing? Uneven Fuzzing Coverage

Observation: some parts of the program easier to cover

```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, “<!ATTLIST”)){
    // ...


  }
  // ...
  return process_result;
}
```

What's Missing? Uneven Fuzzing Coverage

Observation: some parts of the program easier to cover

Hit by 100k+ inputs

```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, “<!ATTLIST”)){
    // ...


  }
  // ...
  return process_result;
}
```

What's Missing? Uneven Fuzzing Coverage

Observation: some parts of the program easier to cover

Hit by 100k+ inputs

→ Code under if well-covered

```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, "<!ATTLIST")){
    // ...


  }
  // ...
  return process_result;
}
```

What’s Missing? Uneven Fuzzing Coverage

Observation: some parts of the program easier to cover

Hit by 100k+ inputs

→ Code under if well-covered

Hit by 1 input

```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, “<!ATTLIST”)){
    // ...


  }
  // ...
  return process_result;
}
```

What's Missing? Uneven Fuzzing Coverage

Observation: some parts of the program easier to cover

Hit by 100k+ inputs

→ Code under if well-covered

Hit by 1 input

→ Code under if *barely* covered

```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, “<!ATTLIST”)){
    // ...


  }
  // ...
  return process_result;
}
```

Uneven Fuzzing Coverage → Uncovered Code

Observation: some parts of the program easier to cover

Hit by 100k+ inputs
→ Code under if well-covered

Hit by 1 input
→ Code under if *barely* covered

Result: some functionality wholly uncovered by fuzzing

```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, “<!ATTLIST”)){
      if (starts_with(&fuzzed_data[10], “ID”)) {
        // lots more processing code
      }
  }
  // ...
  return process_result;
}
```

Why So Uneven?

Some branches hard to hit by naively mutated inputs

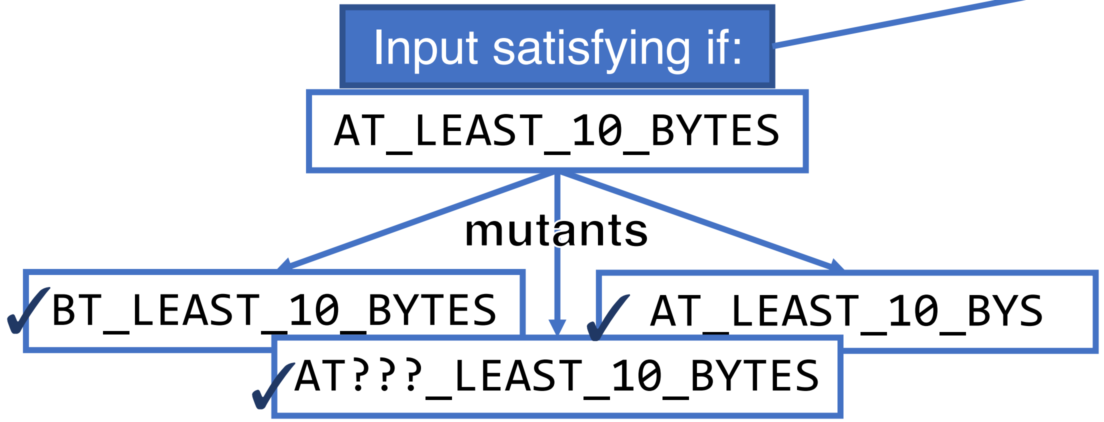
Input satisfying if:

AT_LEAST_10_BYTES

```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
       // more code
  }
  // ...
  if (starts_with(fuzzed_data, “<!ATTLIST”)){
     if (starts_with(&fuzzed_data[10], “ID”)) {
       // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

Why So Uneven?

Some branches hard to hit by naively mutated inputs




```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, “<!ATTLIST”)){
     if (starts_with(&fuzzed_data[10], “ID”)) {
       // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

Why So Uneven?

Some branches hard to hit by naively mutated inputs

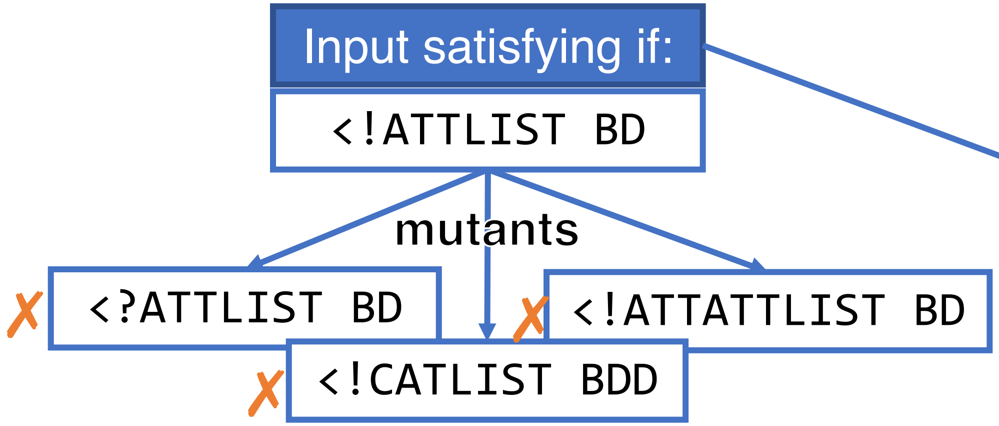
Input satisfying if:

`<!ATTLIST BD`

```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, "<!ATTLIST")){
     if (starts_with(&fuzzed_data[10], "ID")) {
       // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

Why So Uneven?

Some branches hard to hit
by naively mutated inputs

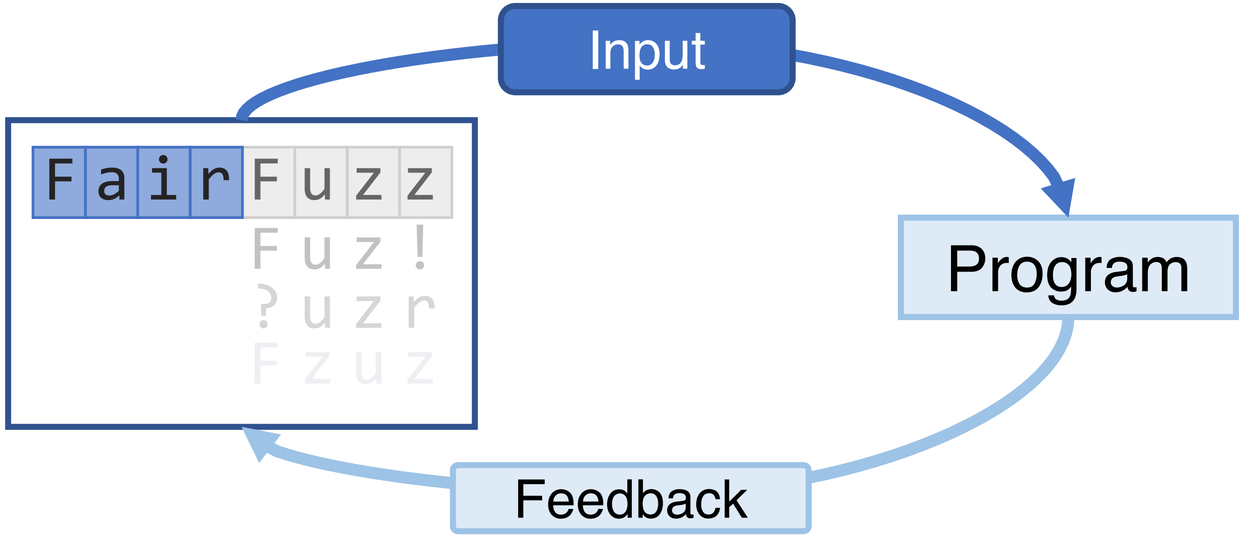



```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, “<!ATTLIST”)){
     if (starts_with(&fuzzed_data[10], “ID”)) {
       // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

Our Method: FairFuzz

Utilize existing greybox info

To target rarely-exercised code → increase coverage

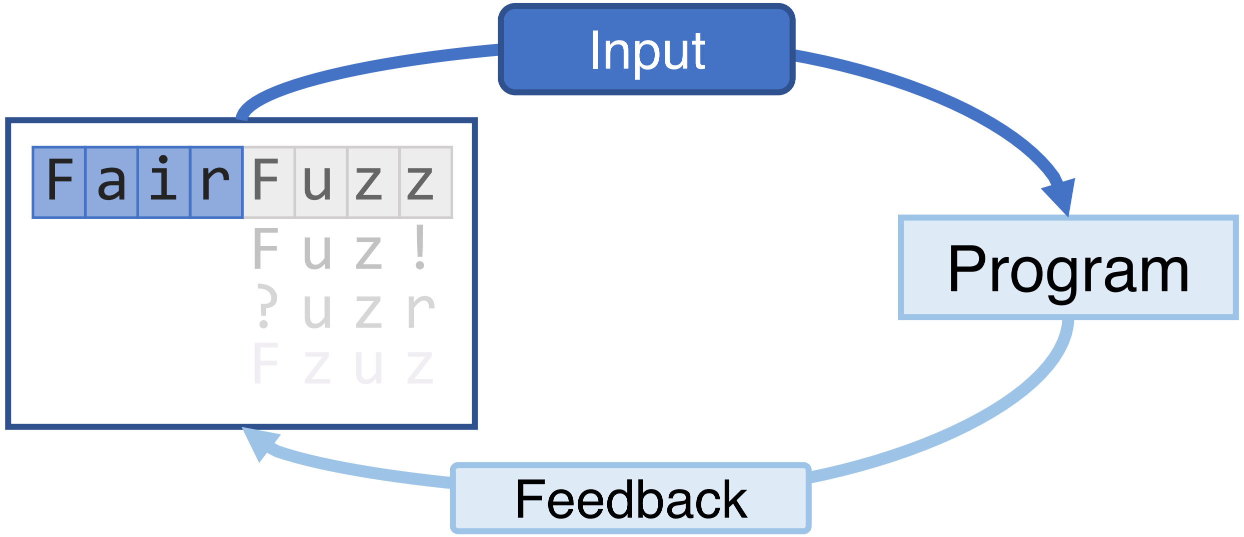


Our Method: FairFuzz

Utilize existing greybox info

To target rarely-exercised code → increase coverage

Identify: branches hit by few inputs (rare branches)

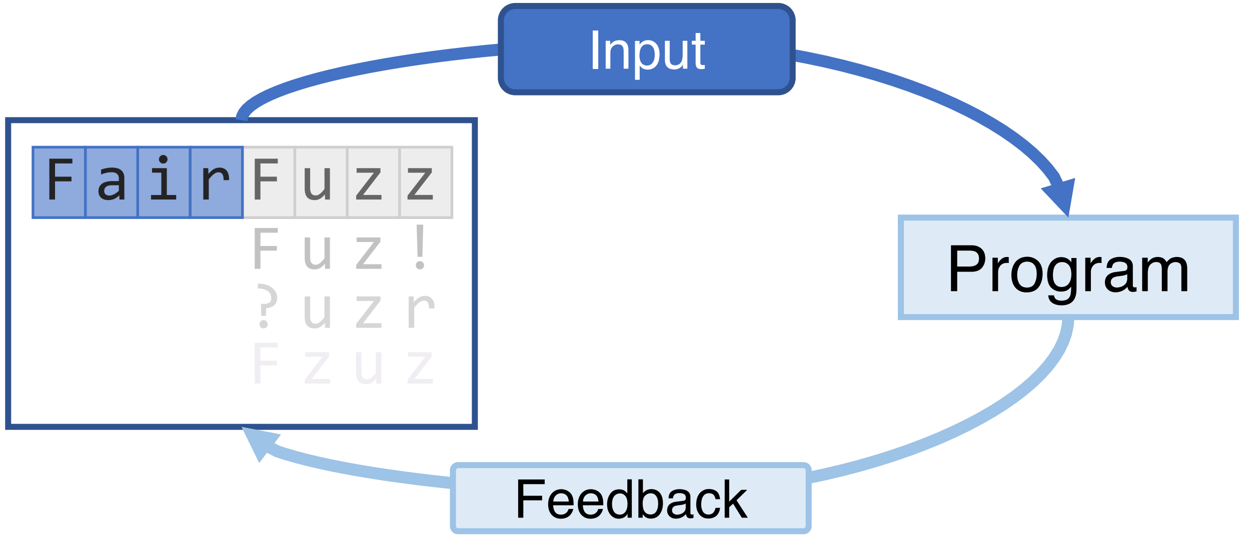


Our Method: FairFuzz

Utilize existing greybox info

To target rarely-exercised code → increase coverage

- **Identify**: branches hit by few inputs (rare branches)
- **Identify**: where input can be mutated and hit branch



Method

Recap: AFL

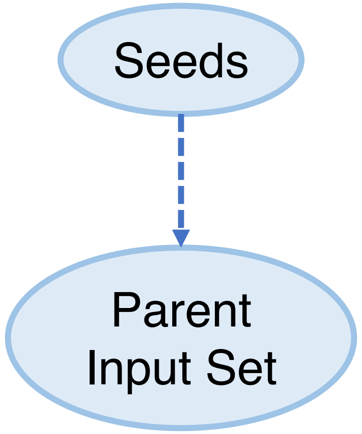
- AFL [1]: Popular coverage-guided greybox fuzzer
- Fuzzes programs taking in file or stdin
- Easy to use (just compile program with afl-gcc or afl-clang)
- Has found many bugs in practice

[1] http://lcamtuf.coredump.cx/afl/

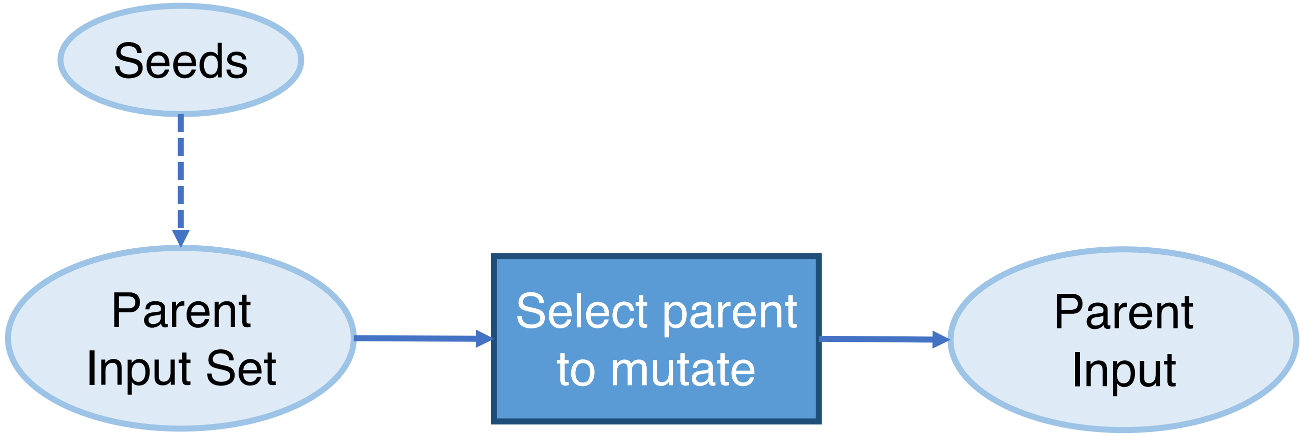
AFL Method

Seeds

AFL Method

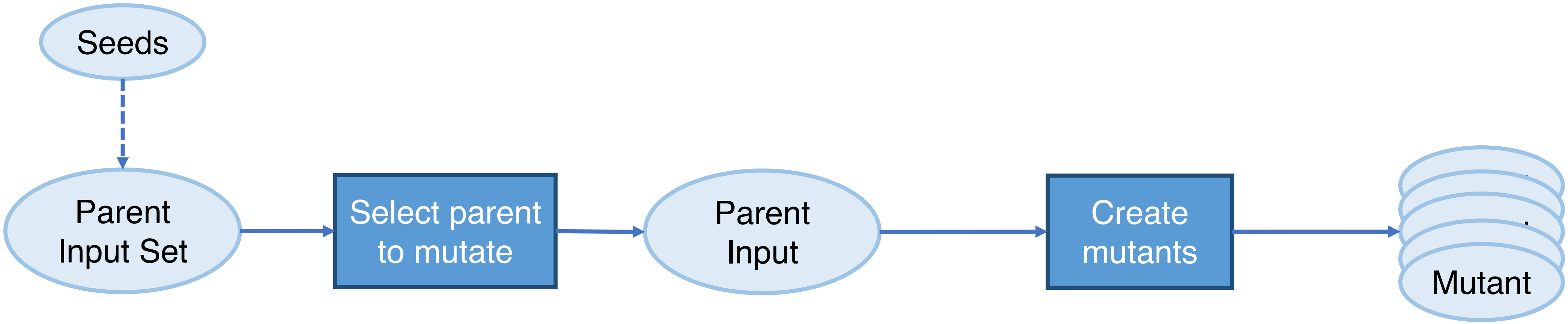


AFL Method



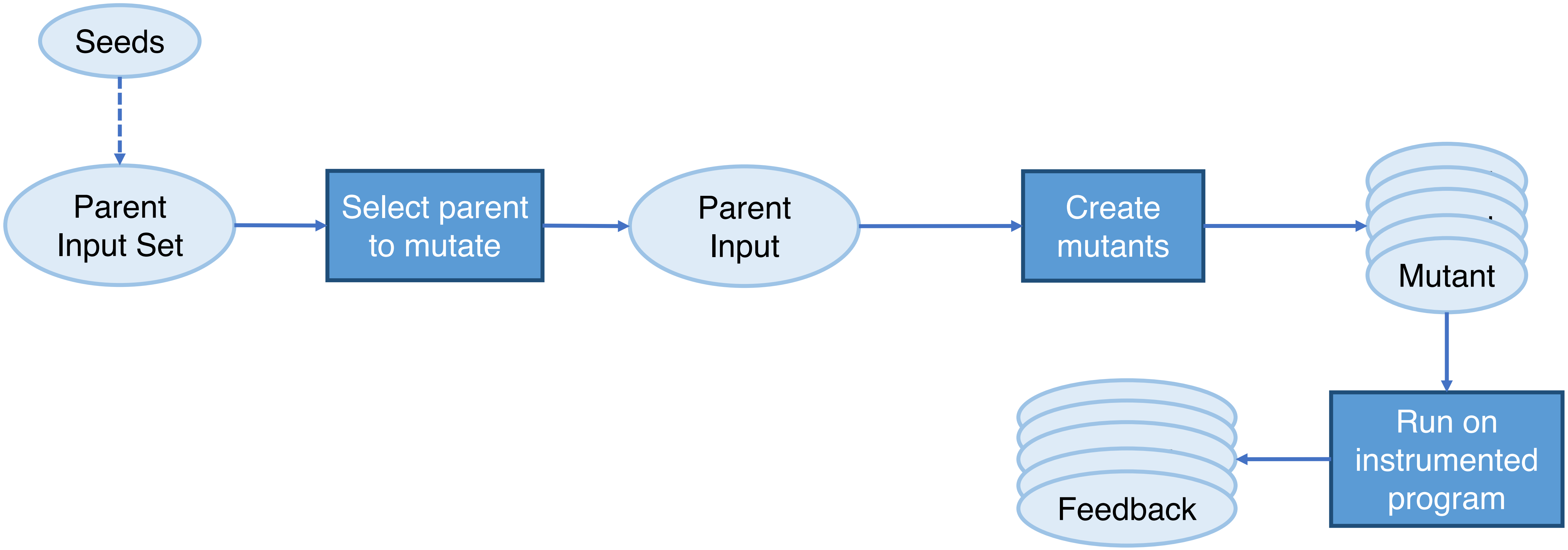
Seeds
Parent Input Set
Select parent to mutate
Parent Input

AFL Method



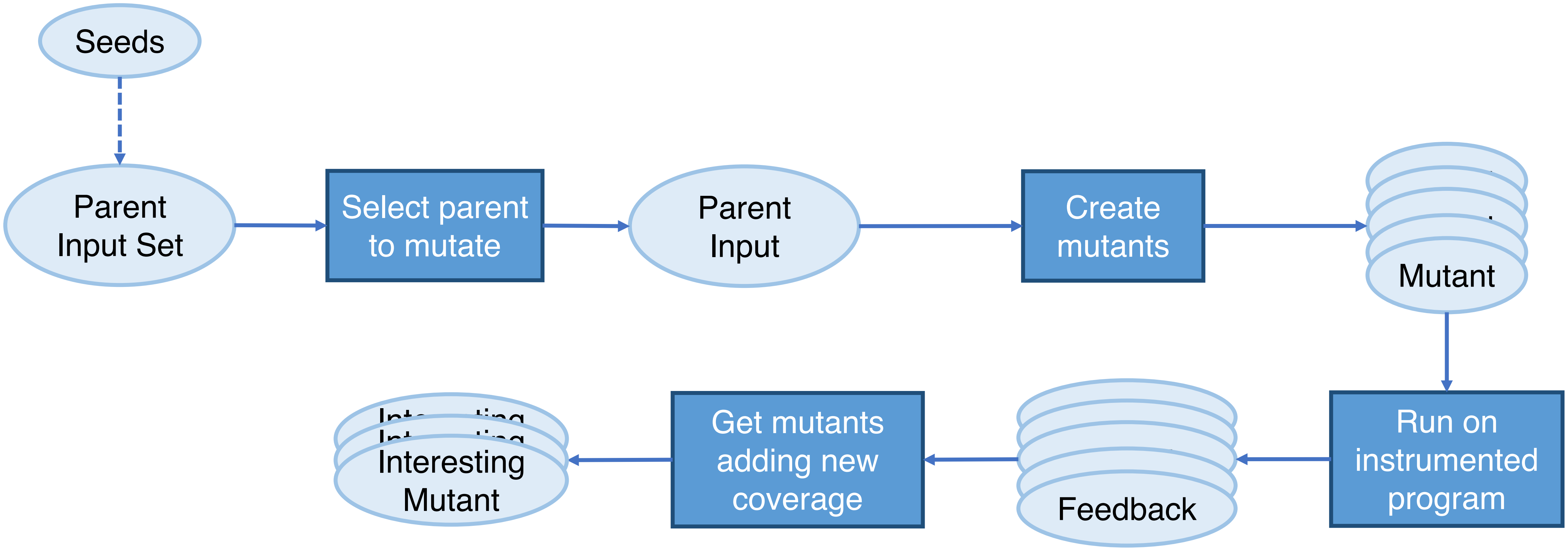
Seeds
Parent Input Set
Select parent to mutate
Parent Input
Create mutants
Mutant

AFL Method



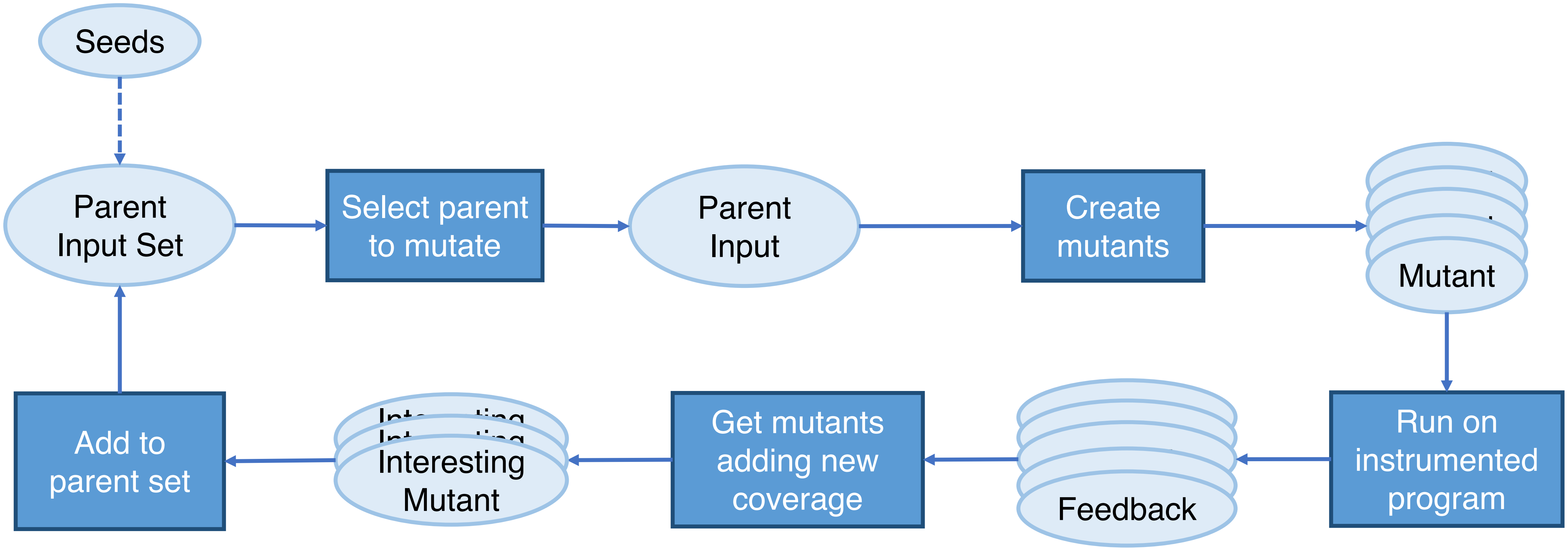
Seeds
Parent Input Set
Select parent to mutate
Parent Input
Create mutants
Mutant
Run on instrumented program
Feedback

AFL Method



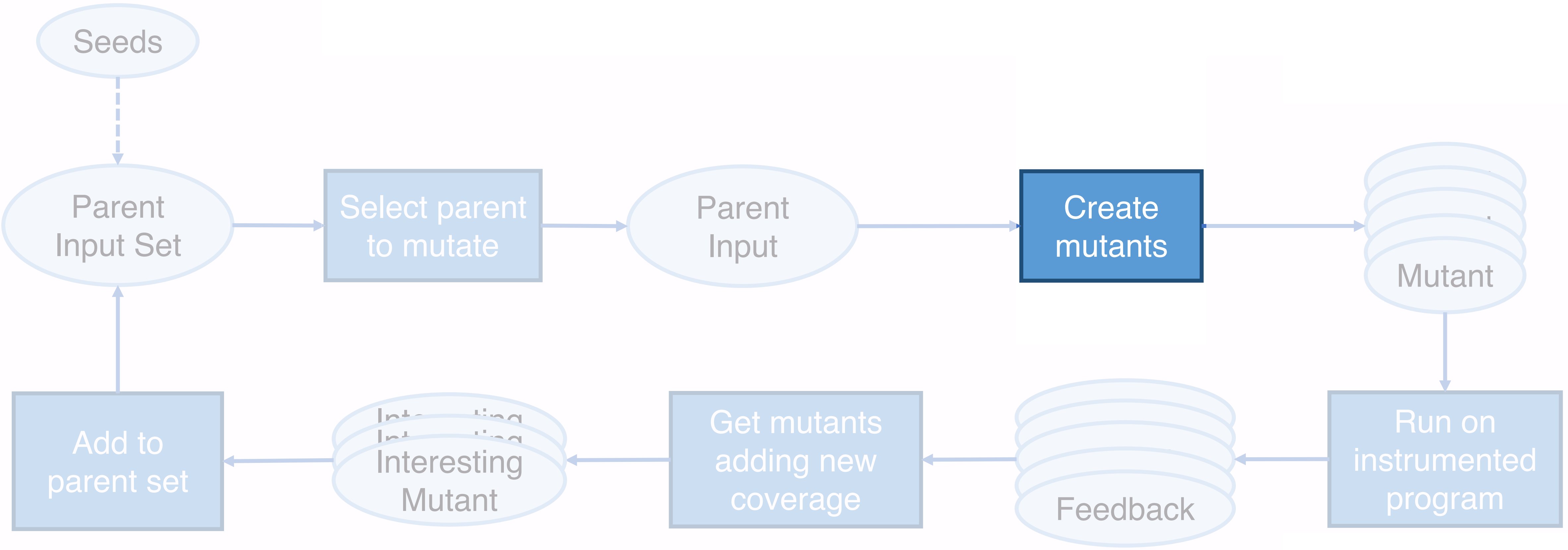
Seeds
Parent Input Set
Select parent to mutate
Parent Input
Create mutants
Mutant
Run on instrumented program
Feedback
Get mutants adding new coverage
Interesting Mutant

AFL Method



Seeds
Parent Input Set
Select parent to mutate
Parent Input
Create mutants
Mutant
Run on instrumented program
Feedback
Get mutants adding new coverage
Interesting Mutant
Add to parent set

AFL Method



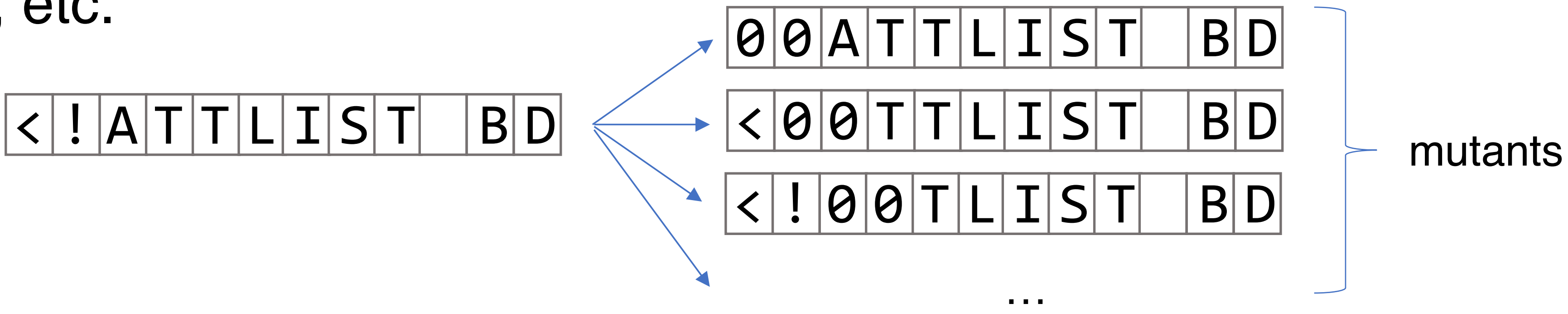
Seeds
Parent Input Set
Select parent to mutate
Parent Input
Create mutants
Mutant
Run on instrumented program
Feedback
Get mutants adding new coverage
Interesting Mutant
Add to parent set

AFL Mutation Types

- Fixed-location mutations
  - Choose mutation type, apply at all locations in input
  - Mutation types: byte flips, arithmetic inc/dec, replacing with “interesting” values, etc.

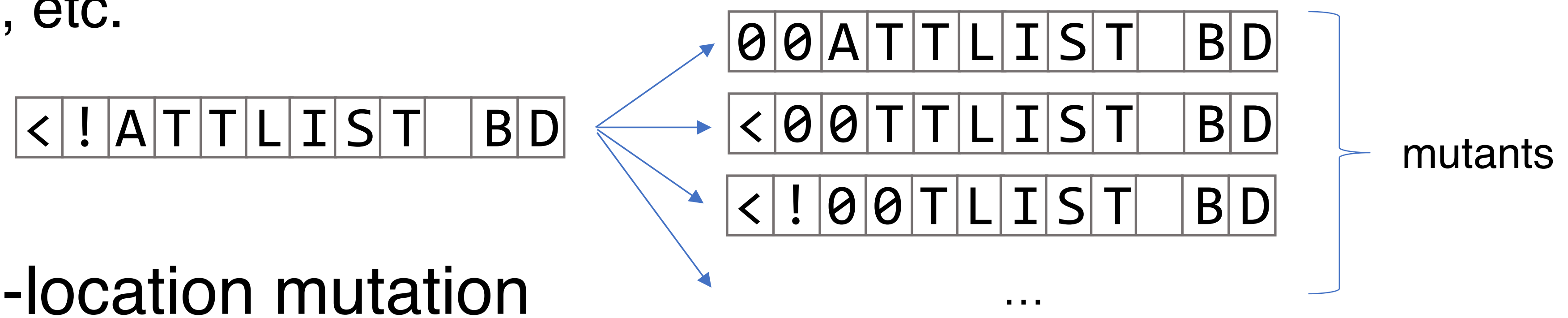
AFL Mutation Types

- Fixed-location mutations
  - Choose mutation type, apply at all locations in input
  - Mutation types: byte flips, arithmetic inc/dec, replacing with “interesting” values, etc.



AFL Mutation Types

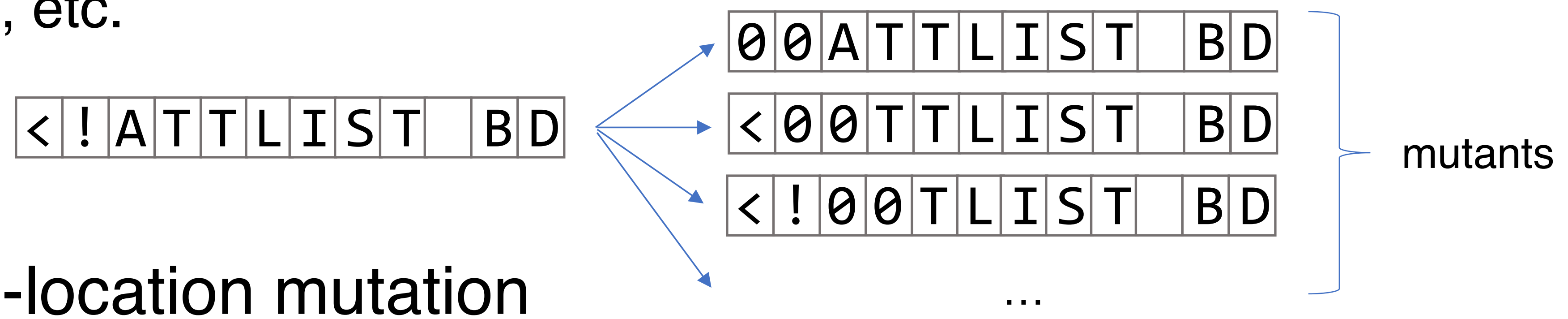
- Fixed-location mutations
  - Choose mutation type, apply at all locations in input
  - Mutation types: byte flips, arithmetic inc/dec, replacing with “interesting” values, etc.



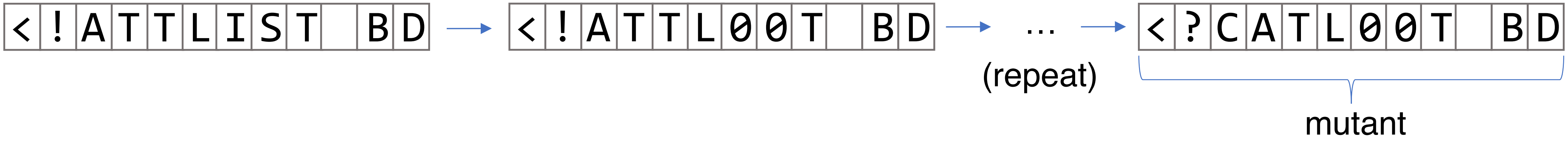
- Random-location mutation
  - Repeat: choose random mutation, apply at random location

AFL Mutation Types

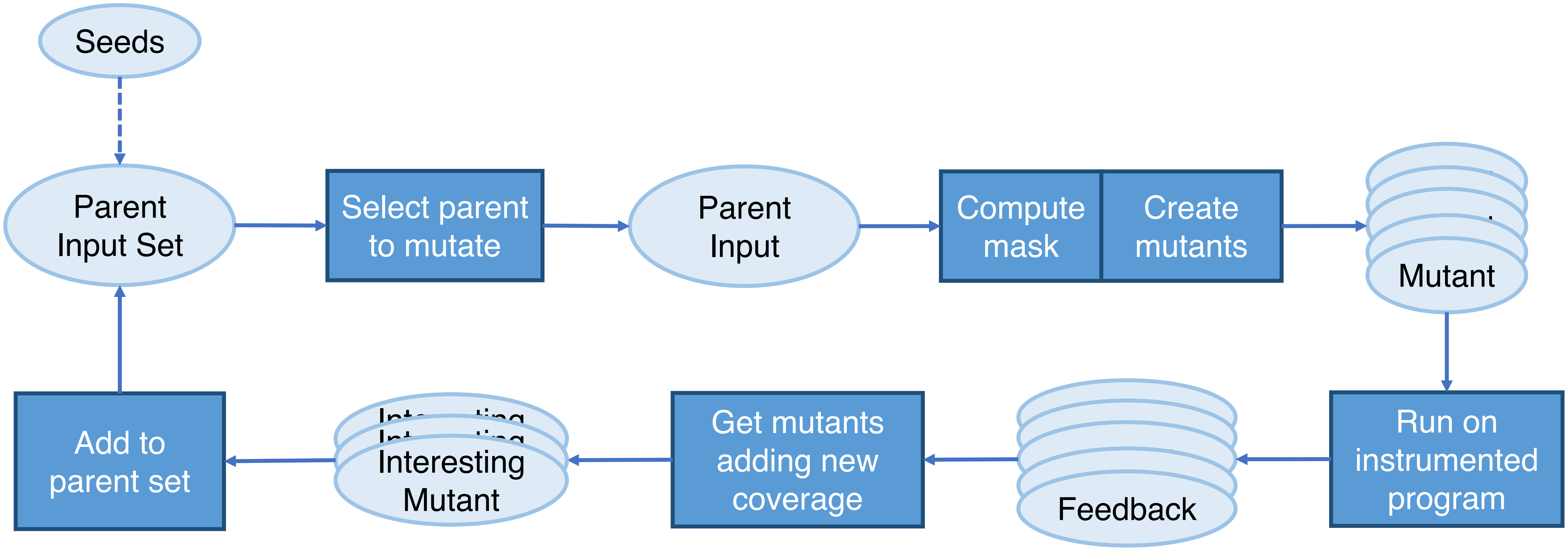
- Fixed-location mutations
  - Choose mutation type, apply at all locations in input
  - Mutation types: byte flips, arithmetic inc/dec, replacing with “interesting” values, etc.



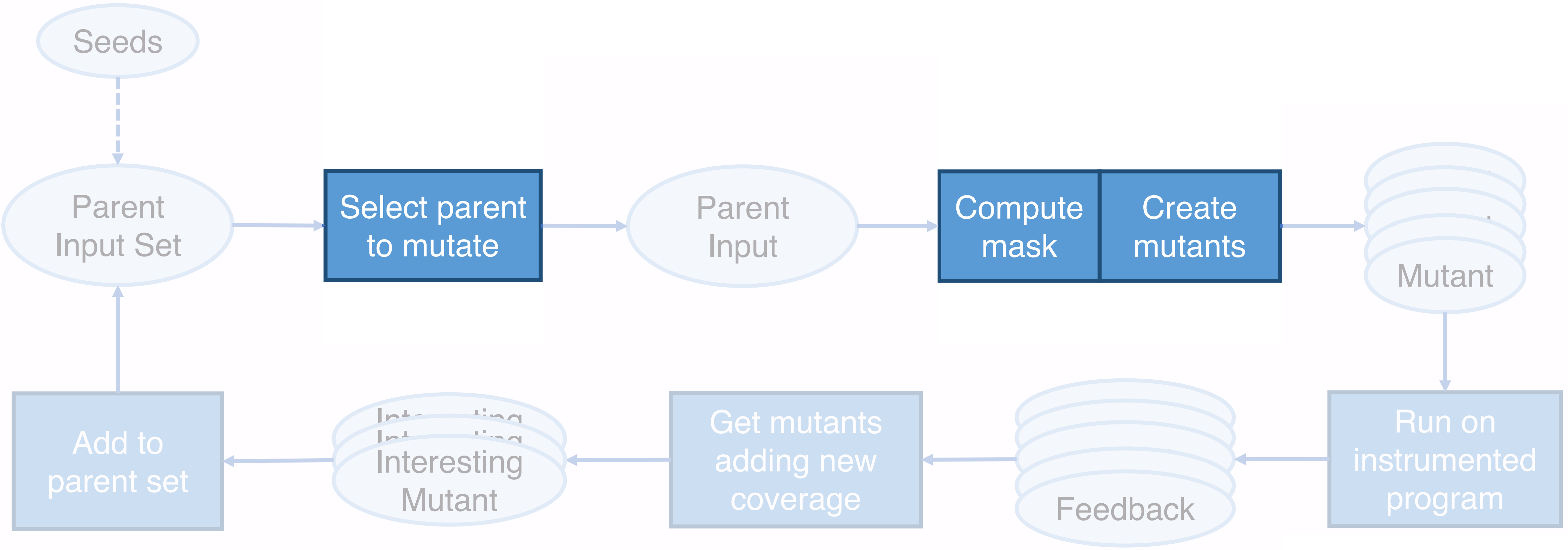
- Random-location mutation
  - Repeat: choose random mutation, apply at random location



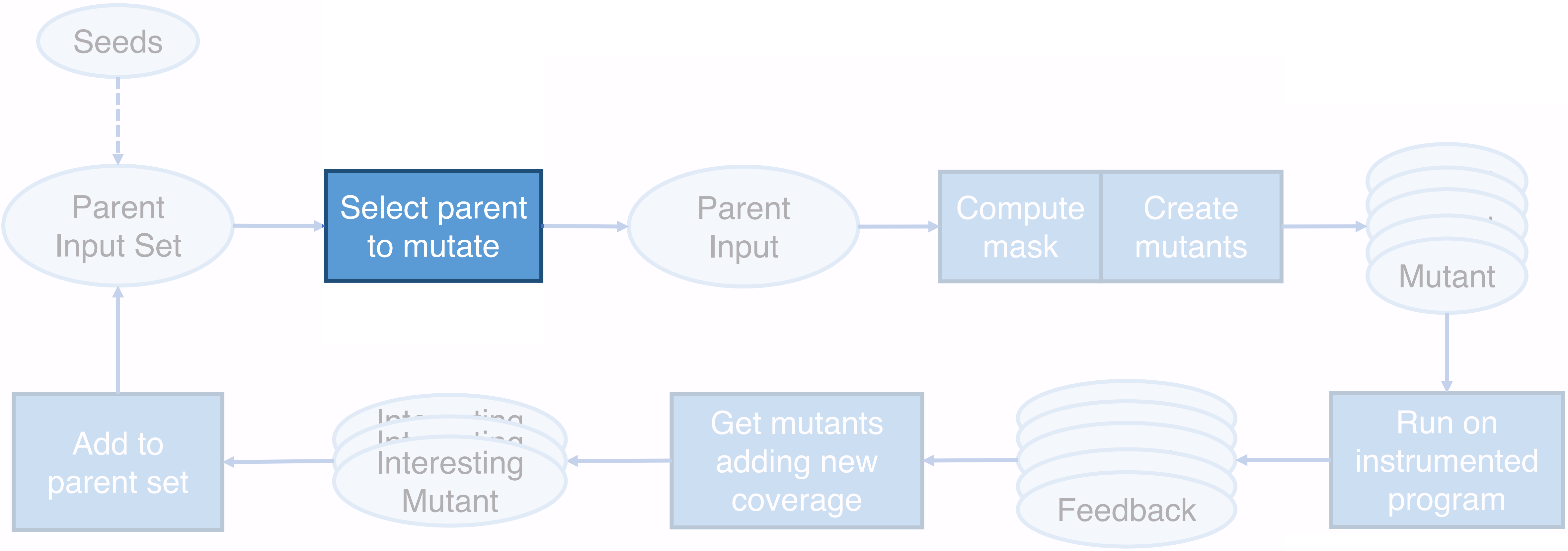
FairFuzz Method



FairFuzz Method – Key Differences



FairFuzz Method – Selecting Parent Inputs



FairFuzz Method – Selecting Parent Inputs

- Keep track of # of inputs produced exercising each branch
- Pick inputs that exercise a branch hit by relatively few inputs
- Rarest branch hit: target branch

FairFuzz Method – Selecting Parent Inputs

- Keep track of # of inputs produced exercising each branch
- Pick inputs that exercise a branch hit by relatively few inputs
- Rarest branch hit: target branch

Hit by 100k+ inputs

`AT_LEAST_10_BYTES`

Hit by 1 input

`<!ATTLIST BD`

```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, "<!ATTLIST")){
     if (starts_with(&fuzzed_data[10], "ID")) {
       // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

FairFuzz Method – Selecting Parent Inputs

- Keep track of # of inputs produced exercising each branch
- Pick inputs that exercise a branch hit by relatively few inputs
- Rarest branch hit: target branch

Hit by 100k+ inputs

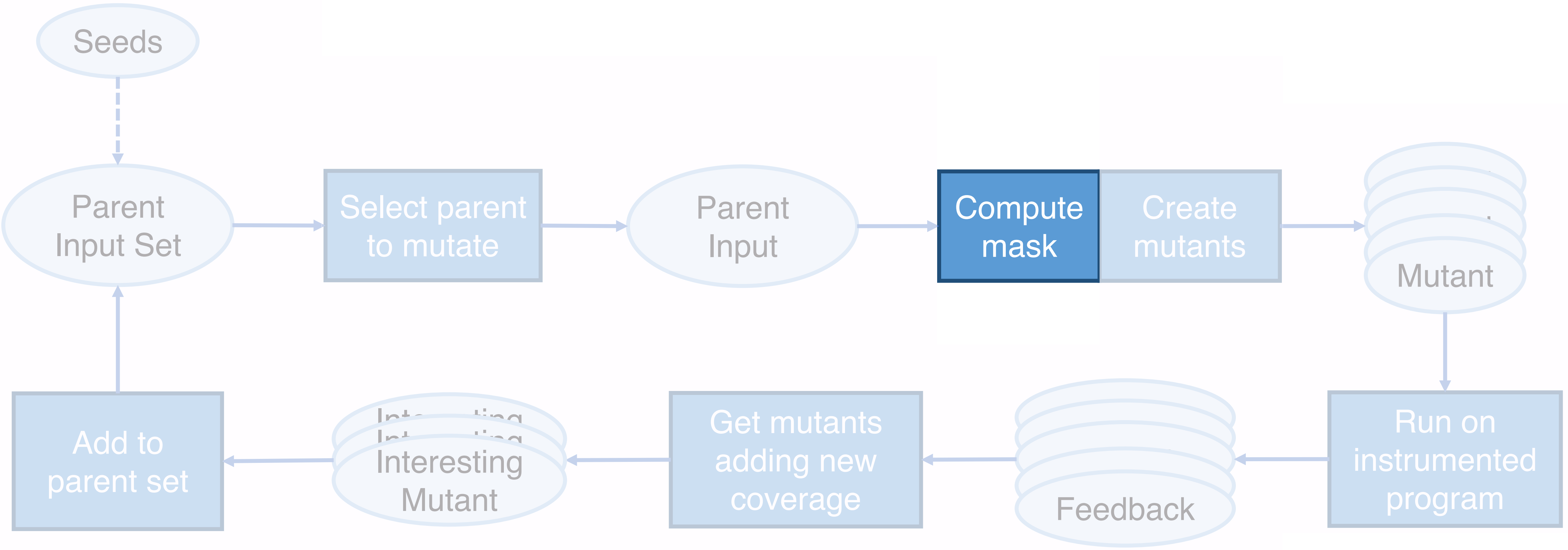
AT_LEAST_10_BYTES

Hit by 1 input

<!ATTLIST BD

```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, "<!ATTLIST")){
     if (starts_with(&fuzzed_data[10], "ID")) {
       // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

FairFuzz Method – Computing Branch Mask



FairFuzz Method – Computing Branch Mask

- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch

FairFuzz Method – Computing Branch Mask

- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch

```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, “<!ATTLIST”)){
     if (starts_with(&fuzzed_data[10], “ID”)) {
       // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

FairFuzz Method – Computing Branch Mask

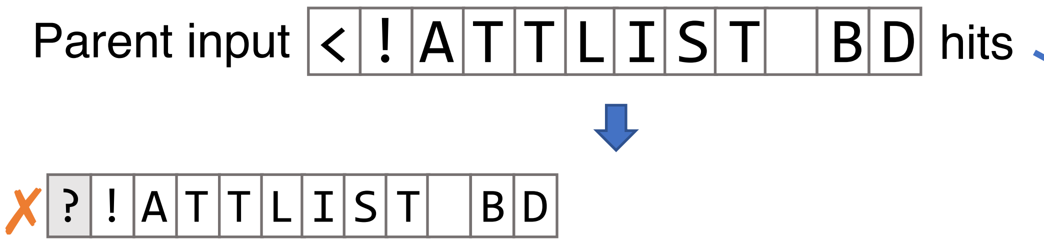
- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch

Parent input	<	!	A	T	T	L	I	S	T		B	D	hits

```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, “<!ATTLIST”)){
     if (starts_with(&fuzzed_data[10], “ID”)) {
       // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

FairFuzz Method – Computing Branch Mask

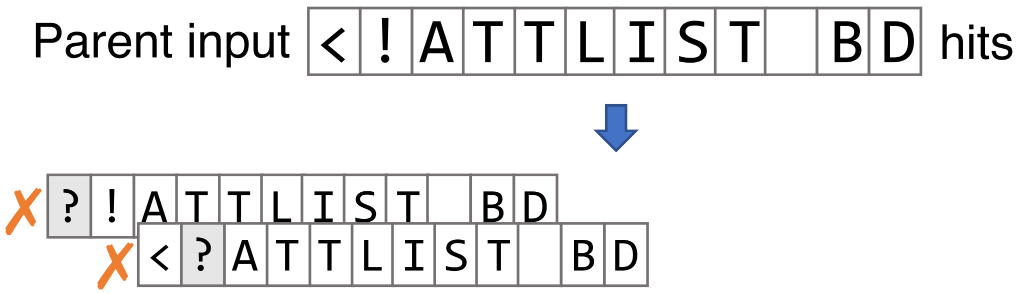
- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch




```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
     // more code
  }
  // ...
  if (starts_with(fuzzed_data, "<!ATTLIST")){
     if (starts_with(&fuzzed_data[10], "ID")) {
       // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

FairFuzz Method – Computing Branch Mask

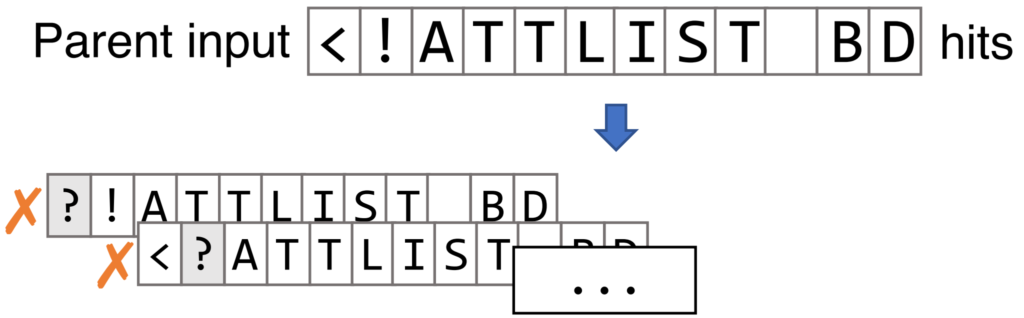
- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch




```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, "<!ATTLIST")){
     if (starts_with(&fuzzed_data[10], "ID")) {
       // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

FairFuzz Method – Computing Branch Mask

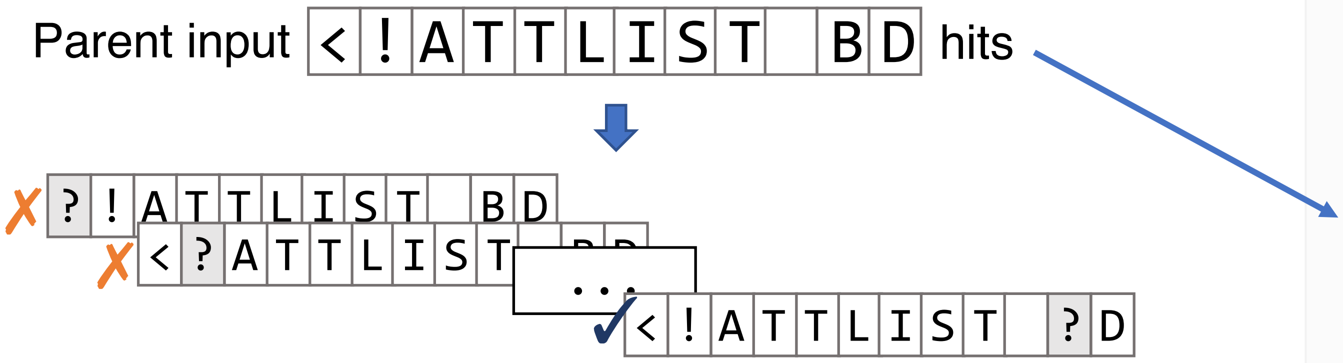
- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch




```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, “<!ATTLIST”)){
     if (starts_with(&fuzzed_data[10], “ID”)) {
       // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

FairFuzz Method – Computing Branch Mask

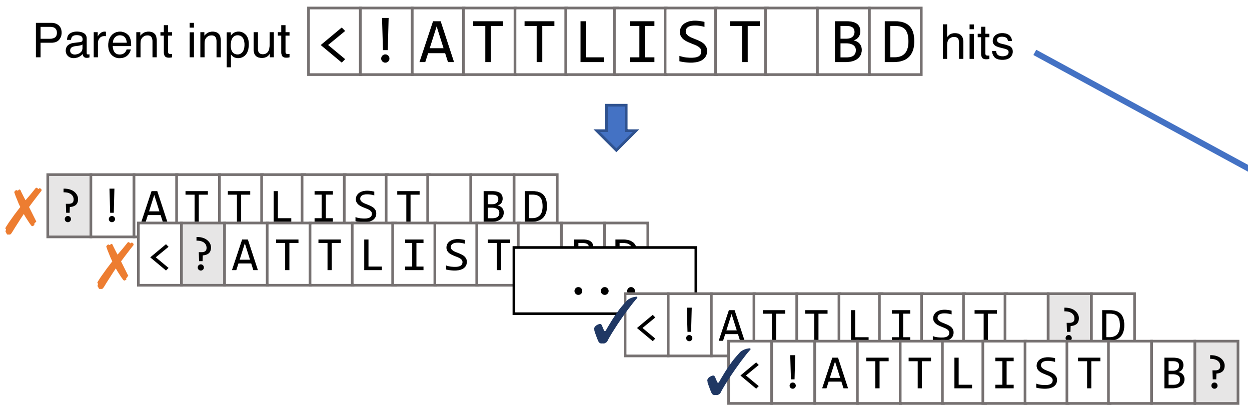
- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch




```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, "<!ATTLIST")){
     if (starts_with(&fuzzed_data[10], "ID")) {
       // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

FairFuzz Method – Computing Branch Mask

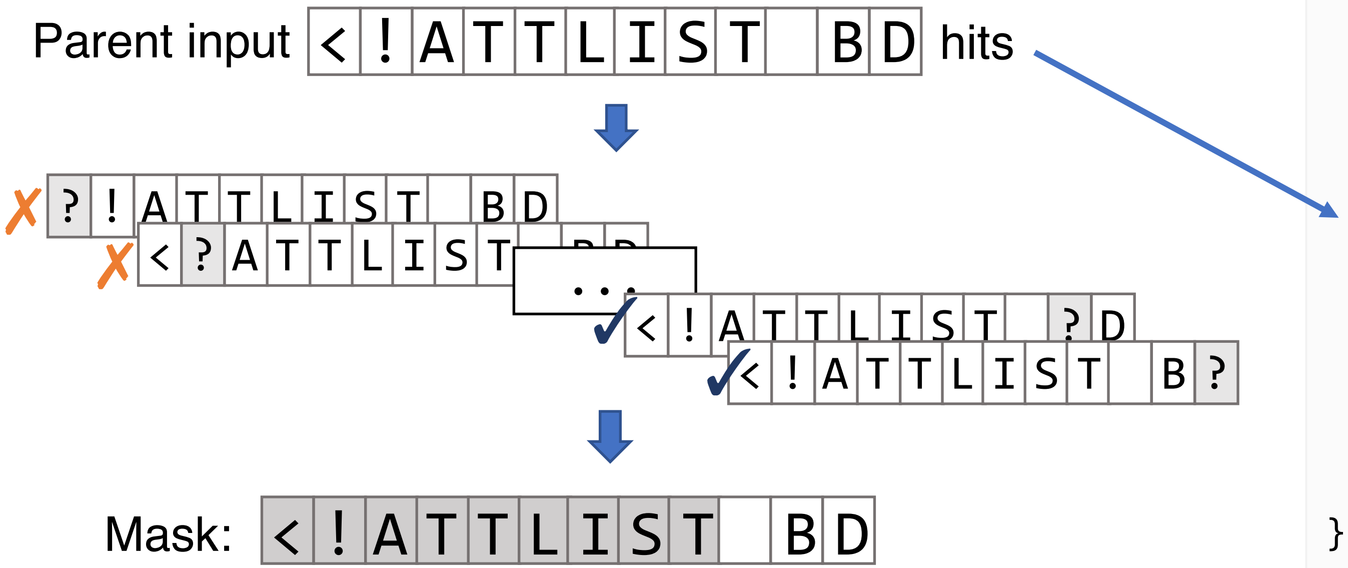
- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch




```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
     // more code
  }
  // ...
  if (starts_with(fuzzed_data, “<!ATTLIST”)){
     if (starts_with(&fuzzed_data[10], “ID”)) {
      // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

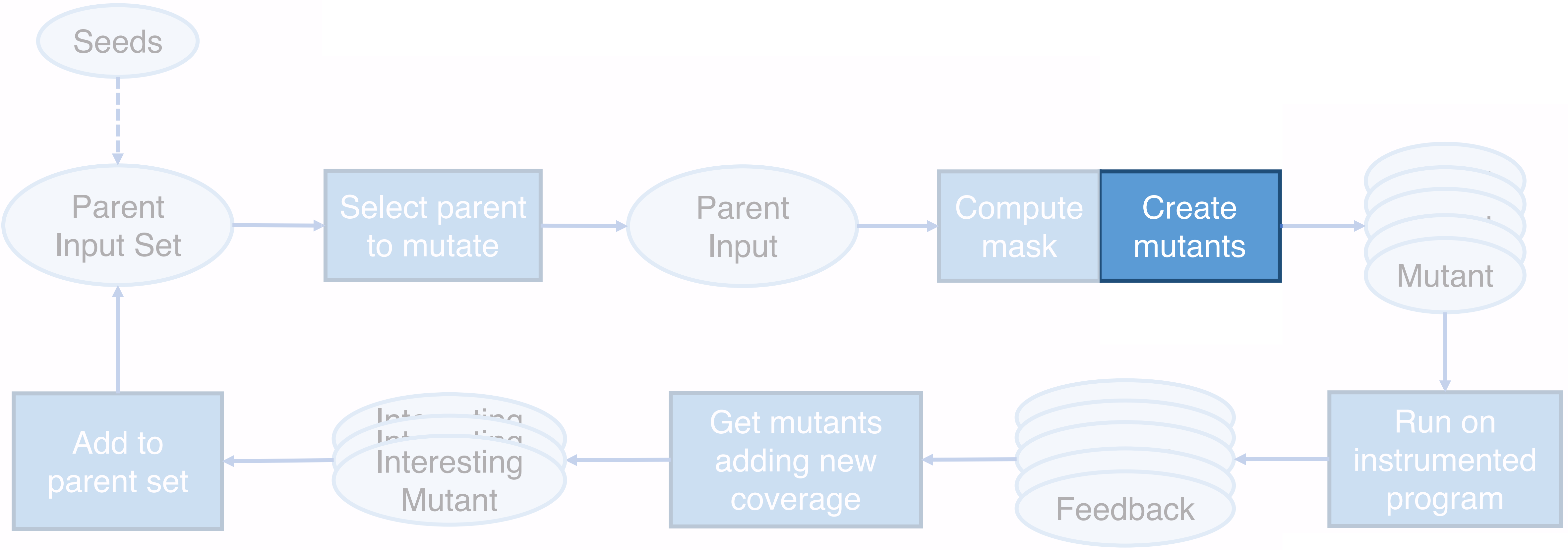
FairFuzz Method – Computing Branch Mask

- Easily integrated with fixed-location mutation phases of fuzzers
- Flip each byte, check if mutated input still hits target branch




```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
      // more code
  }
  // ...
  if (starts_with(fuzzed_data, "<!ATTLIST")){
     if (starts_with(&fuzzed_data[10], "ID")) {
       // lots more processing code
     }
  }
  // ...
  return process_result;
}
```

FairFuzz Method – Targeting Mutations



FairFuzz Method – Targeting Mutations

- Fixed-location mutation
  - Don’t produce mutants at locations in mask

<	!	A	T	T	L	I	S	T		B	D

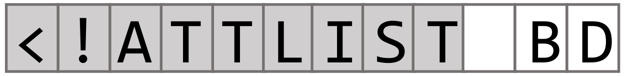
- Random-location mutation
  - Choose random locations outside mask

<	!	A	T	T	L	I	S	T		B	D

FairFuzz Method – Targeting Mutations

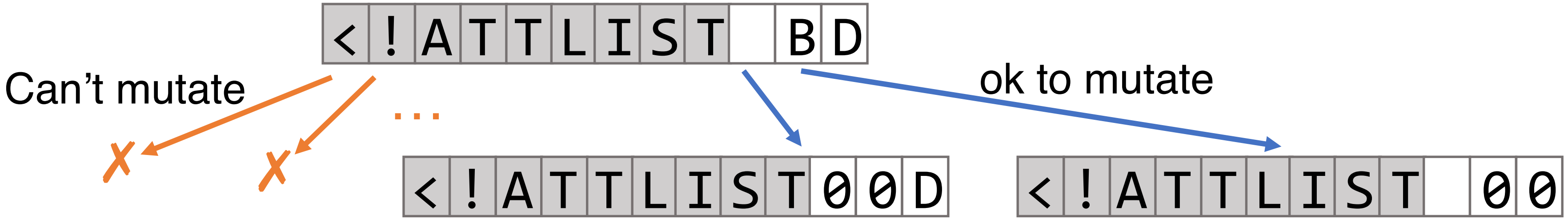
- Fixed-location mutation
  - Don’t produce mutants at locations in mask

- Random-location mutation
  - Choose random locations outside mask

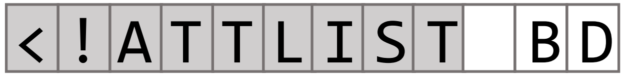


FairFuzz Method – Targeting Mutations

- Fixed-location mutation
  - Don't produce mutants at locations in mask

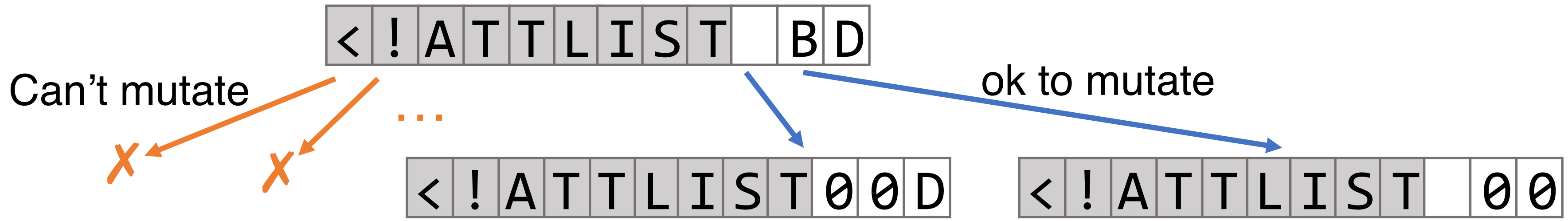


- Random-location mutation
  - Choose random locations outside mask

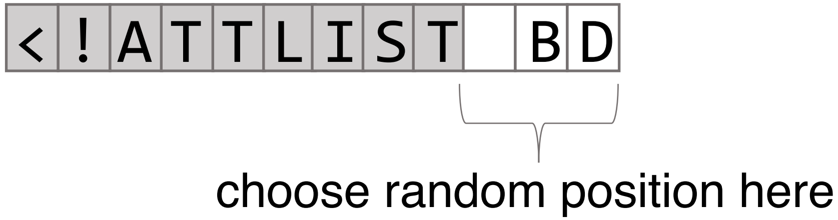


FairFuzz Method – Targeting Mutations

- Fixed-location mutation
  - Don’t produce mutants at locations in mask



- Random-location mutation
  - Choose random locations outside mask



Evaluation

Evaluation – Tools Compared

- **FairFuzz**: our tool, with highest-performing settings
- AFL: vanilla AFL, default settings
- FidgetyAFL: AFL with highest-performing settings
- AFLFast.new: AFLFast with highest-performing settings

[1] Zalewski, Michał. http://lcamtuf.coredump.cx/afl/

[2] Böhme et al. Coverage-based Greybox Fuzzing as Markov Chain. CCS'16.

Evaluation - Benchmarks

`djpeg`
`readpng`

FidgetyAFL benchmarks

`mutool draw`
`xmllint`

More complex input structures

`tcpdump`
`c++filt`
`objdump`
`readelf`
`nm`

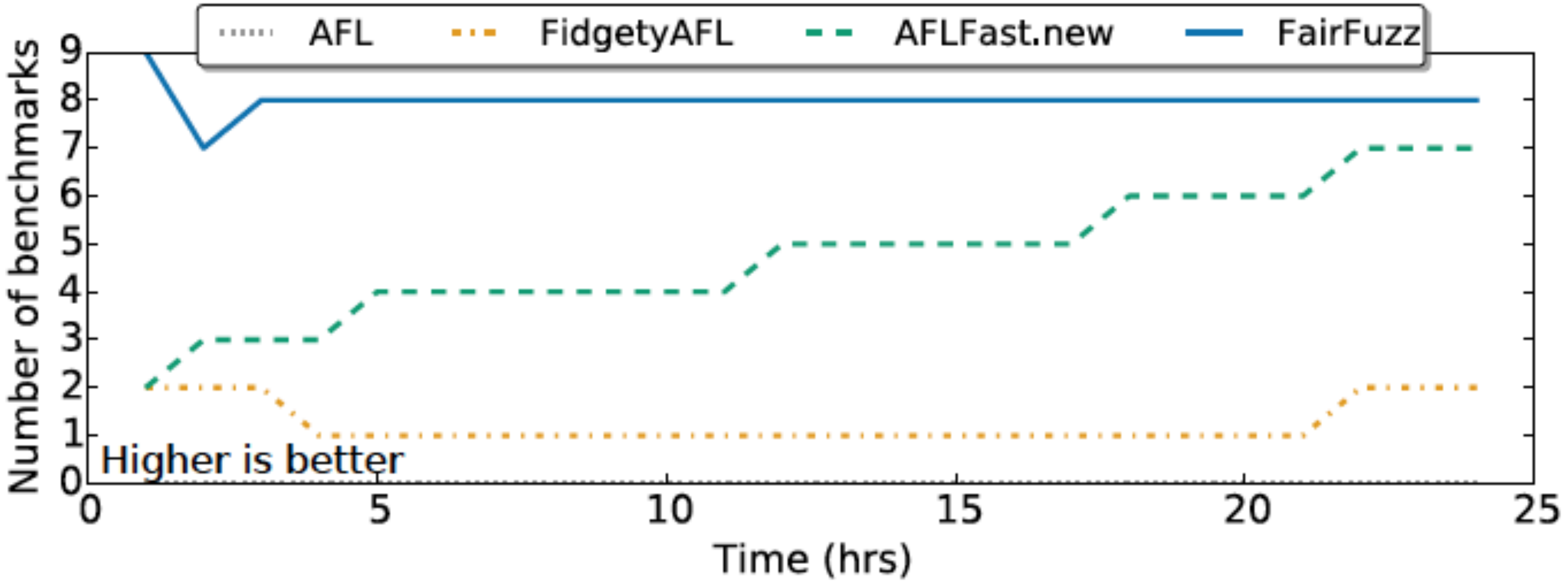
AFLFast benchmarks

Evaluation Setup

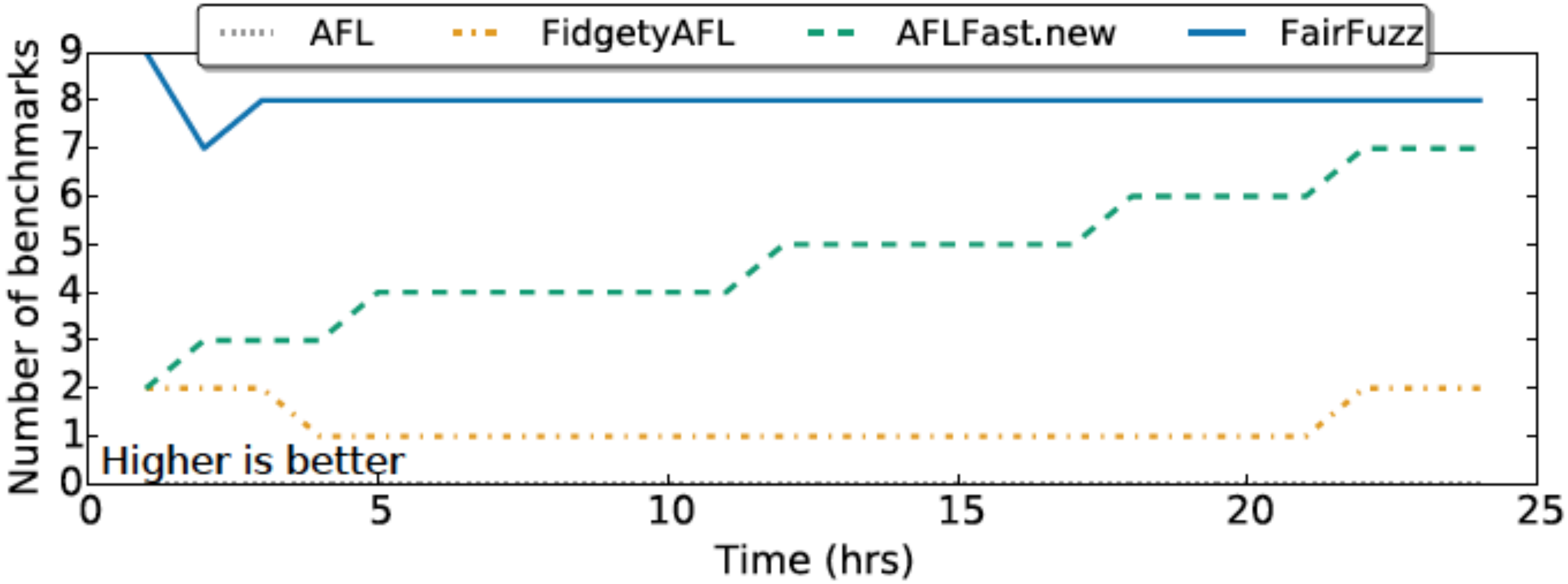
For each benchmark:

- Run each technique 24hrs
- Start with 1 valid seed file
- No dictionaries
- Repeat runs 20x
    - Calculated *confidence intervals*

Summary Results – Coverage Leaders

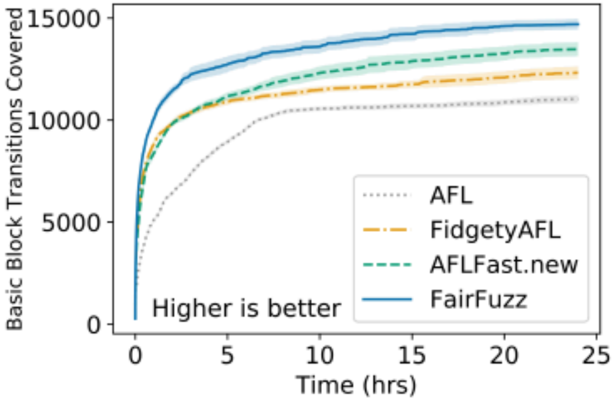


Summary Results – Coverage Leaders

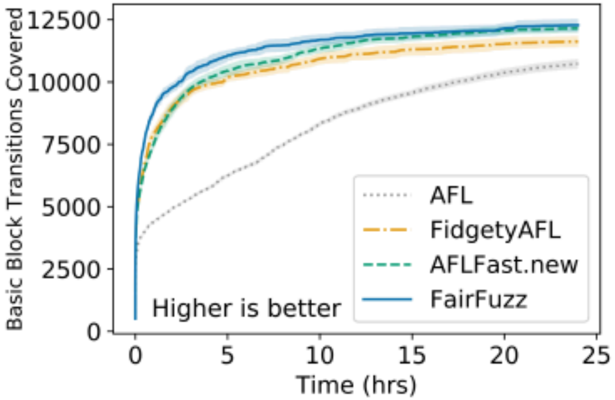


→ FairFuzz achieves the highest coverage fast, for nearly all benchmarks

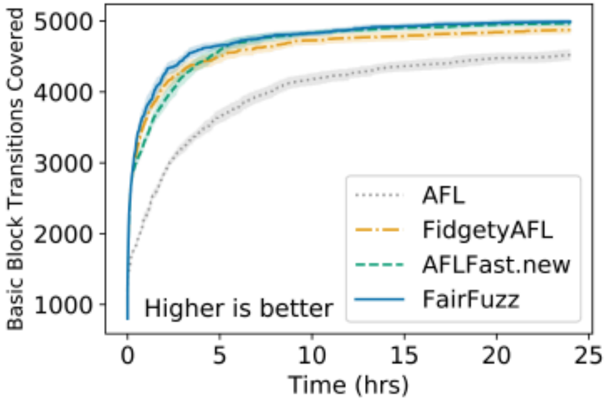
Branch Coverage Over Time



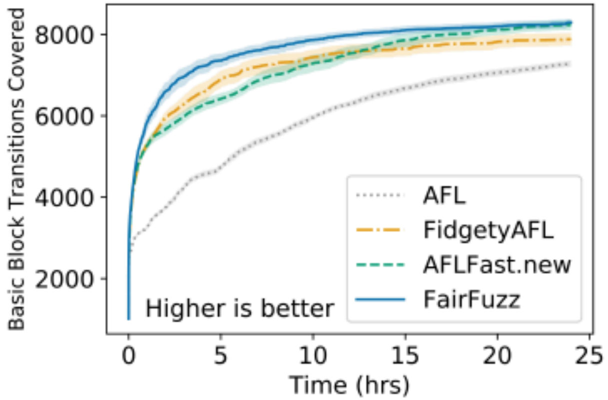
(a) tcpdump



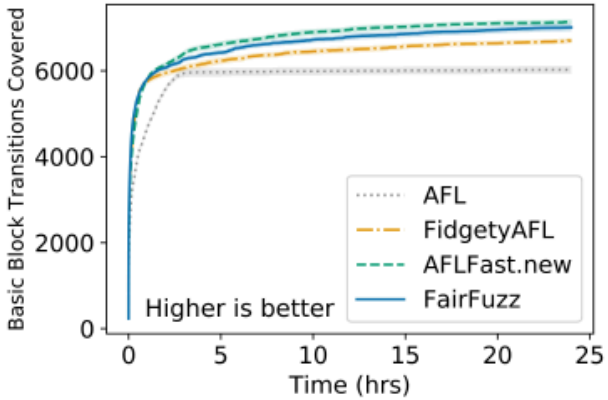
(b) readelf



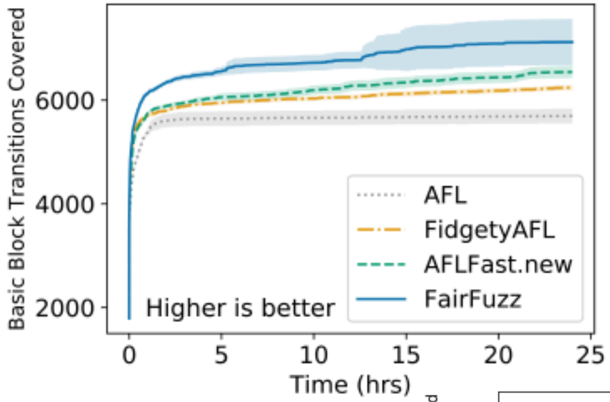
(c) nm



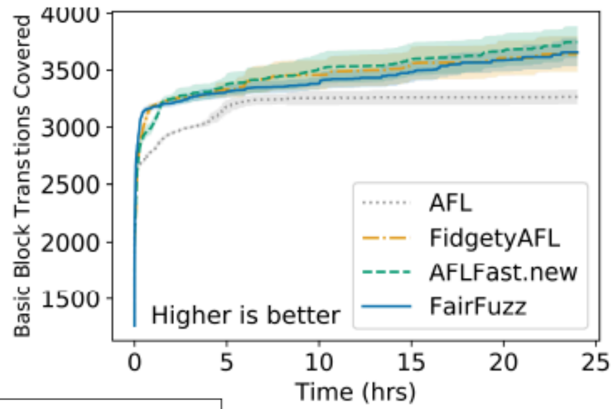
(d) objdump



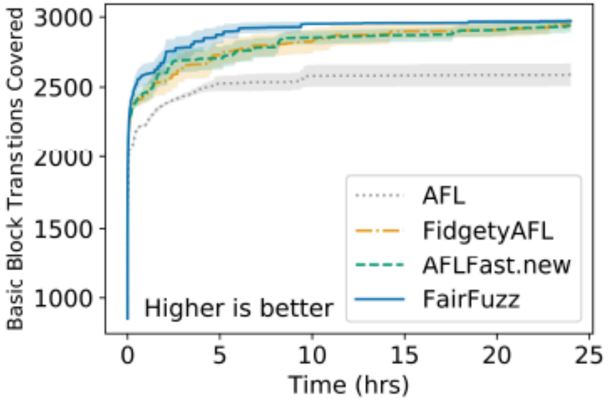
(e) c++filt



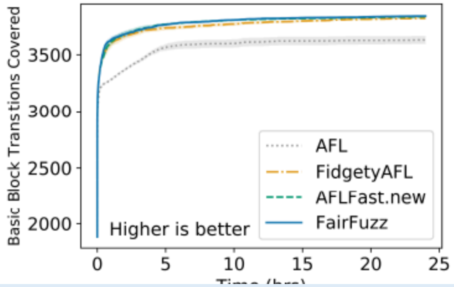
(f) xmllint



(h) djpeg

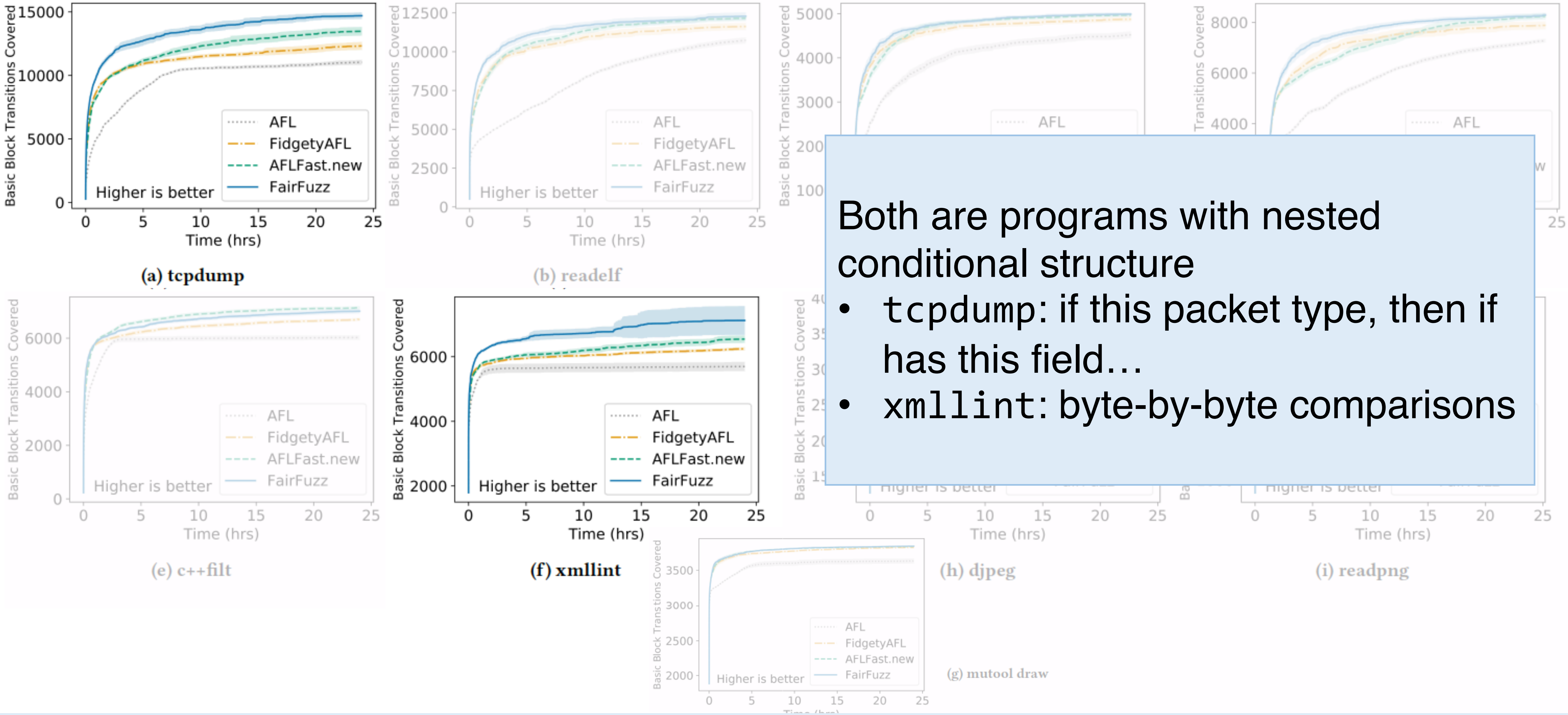


(i) readpng



(g) mutool draw

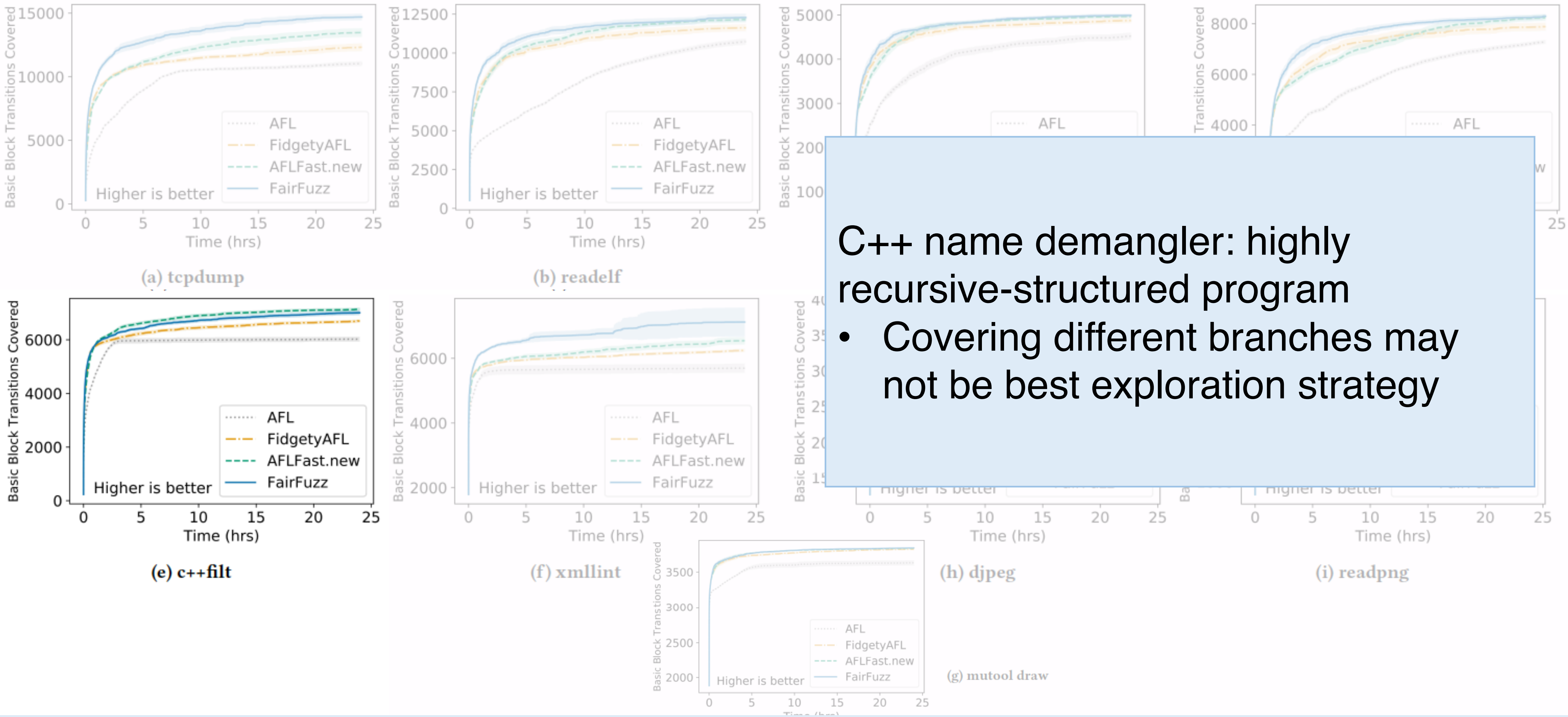
Where Does FairFuzz Perform Much Better?



Both are programs with nested conditional structure

- `tcpdump`: if this packet type, then if has this field…
- `xmllint`: byte-by-byte comparisons

Where Doesn’t FairFuzz Perform As Well?



C++ name demangler: highly recursive-structured program

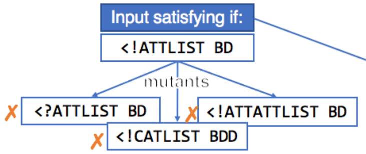
- Covering different branches may not be best exploration strategy

Conclusion

code: github.com/carolemieux/afl-rb slides: carolemieux.com/fairfuzz_ase18_slides.pdf

Why So Uneven?

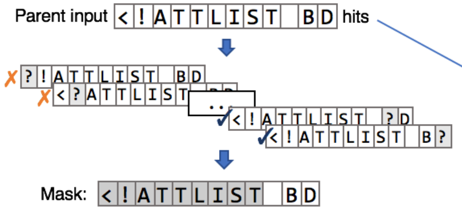
Some branches hard to hit
by naively mutated inputs




```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
    // more code
  }
  // ...
  if (starts_with(fuzzed_data, "<!ATTLIST")){
    if (starts_with(&fuzzed_data[10], "ID")) {
      // lots more processing code
    }
  }
  // ...
  return process_result;
}
```

FairFuzz Method – Computing Branch Mask

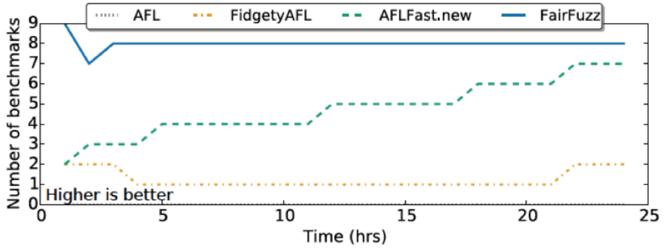
- Flip each byte, check if mutated input still hits target branch




```
int process_xml(char * fuzzed_data,
                int fuzzed_data_len) {
  if (fuzzed_data_len >= 10) {
    // more code
  }
  // ...
  if (starts_with(fuzzed_data, "<!ATTLIST")){
    if (starts_with(&fuzzed_data[10], "ID")) {
      // lots more processing code
    }
  }
  // ...
  return process_result;
}
```

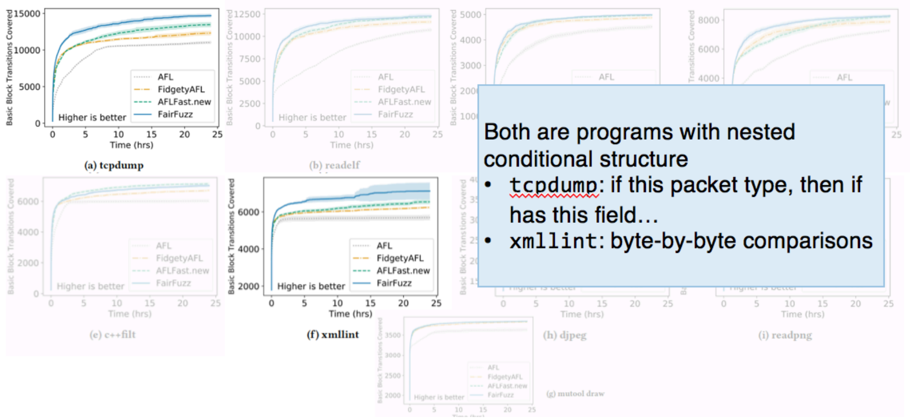
- Easily integrated with fixed-location mutation phases of fuzzers

Summary Results – Coverage Leaders



→ FairFuzz achieves the highest coverage fast, for nearly all benchmarks

Where Does FairFuzz Perform Much Better?



Both are programs with nested conditional structure

- `tcpdump`: if this packet type, then if has this field…
- `xmllint`: byte-by-byte comparisons

Branch Mask Performance

For a subset of benchmarks, run a cycle with “shadow run”:

- For each selected input, create mutants
  - (1) without branch mask
  - (2) without branch mask
- Compare % of inputs hitting target branch:
  - Average over all inputs selected for mutation in cycle

Branch Mask Performance

- Mask substantially increases % of inputs hitting target branch

	Fixed-Location Mutants		Random-Location Mutants	
	With Mask	Without Mask	With Mask	Without Mask
`xmllint`	90.3%	22.9%	32.8%	2.9%
`tcpdump`	98.7%	72.8%	36.1%	9.0%
`c++filt`	96.6%	14.8%	34.4%	1.1%
`readelf`	99.7%	78.2%	55.5%	11.4%
`readpng`	97.8%	39.0%	24.0%	2.4%
`objdump`	99.2%	66.7%	46.2%	7.6%