Snappy: Efficient Fuzzing with Adaptive and Mutable Snapshots

Elia Geretto Cristiano Giuffrida
Herbert Bos Erik van der Kouwe

Vrije Universiteit Amsterdam

Annual Computer Security Applications Conference, 2022

VU VRIJE UNIVERSITEIT AMSTERDAM

Teaser

- Snappy is a speed optimization for fuzzers
- Key insight: it skips the part of the program executions that is always the same (redundant)
- It achieves:
  - up to 1.76× speed increase in FuzzBench, with no significant regressions
  - up to 31% coverage increase on real-world programs (24 hours benchmarks)

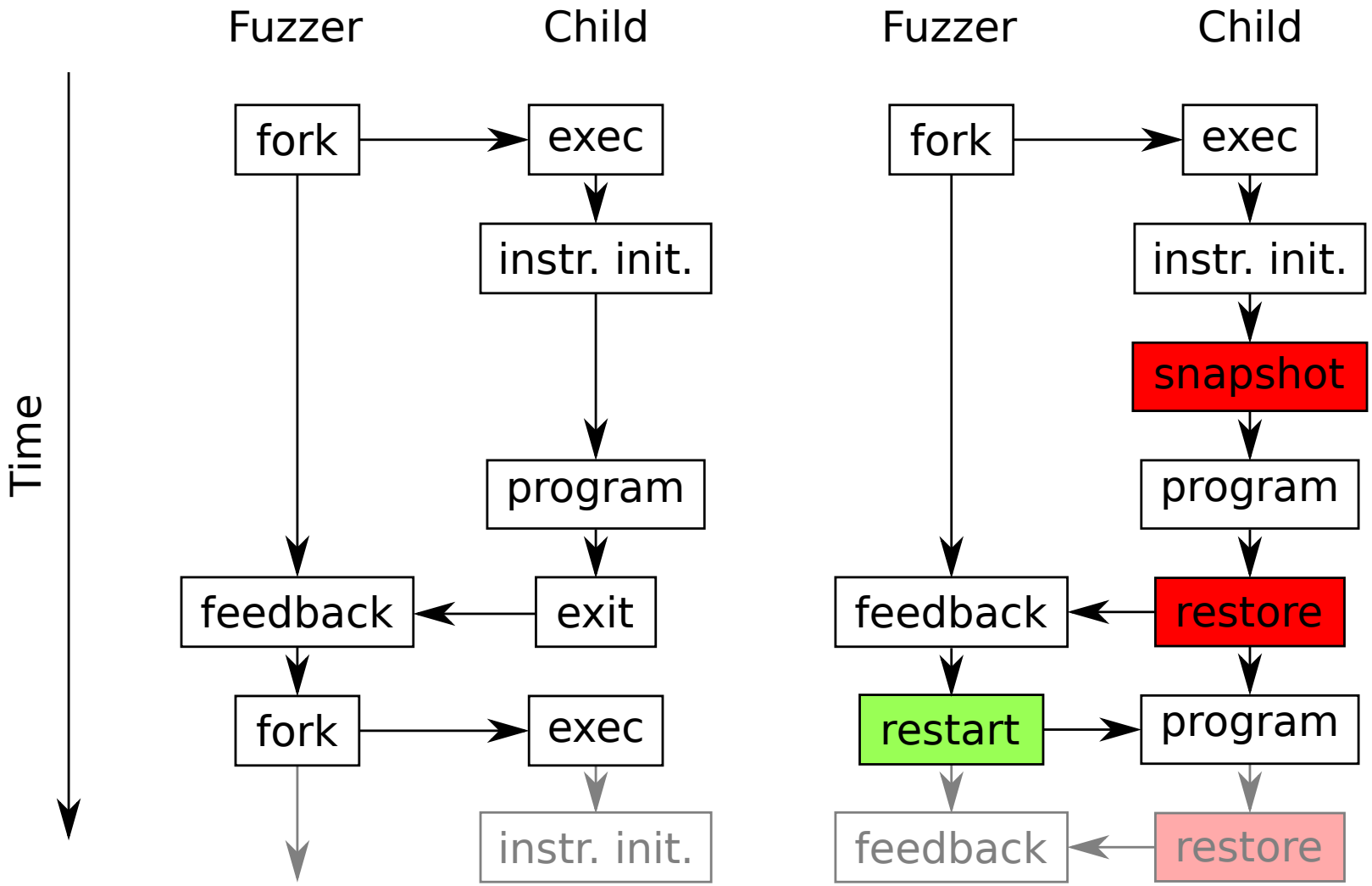
VU VRIJE UNIVERSITEIT AMSTERDAM

Background - User-space application fuzzing

- Fuzzing is a trial and error process
- More attempts make success (crashes) more likely
- Speed (exec/sec) is extremely important
- Operations that do not depend on the *mutated* input are *redundant*

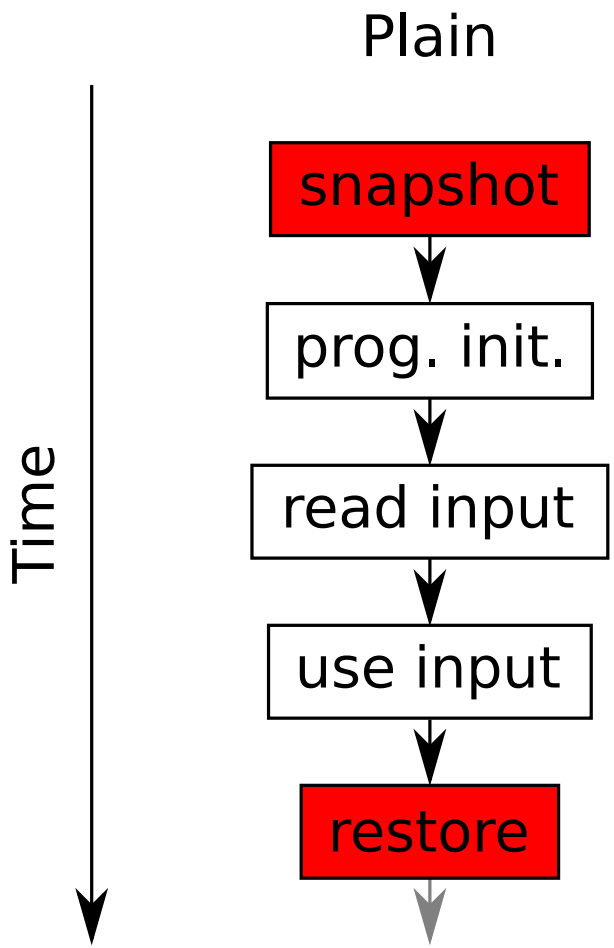
VU VRIJE UNIVERSITEIT AMSTERDAM

Background - Process snapshotting



VU VRIJE UNIVERSITEIT AMSTERDAM

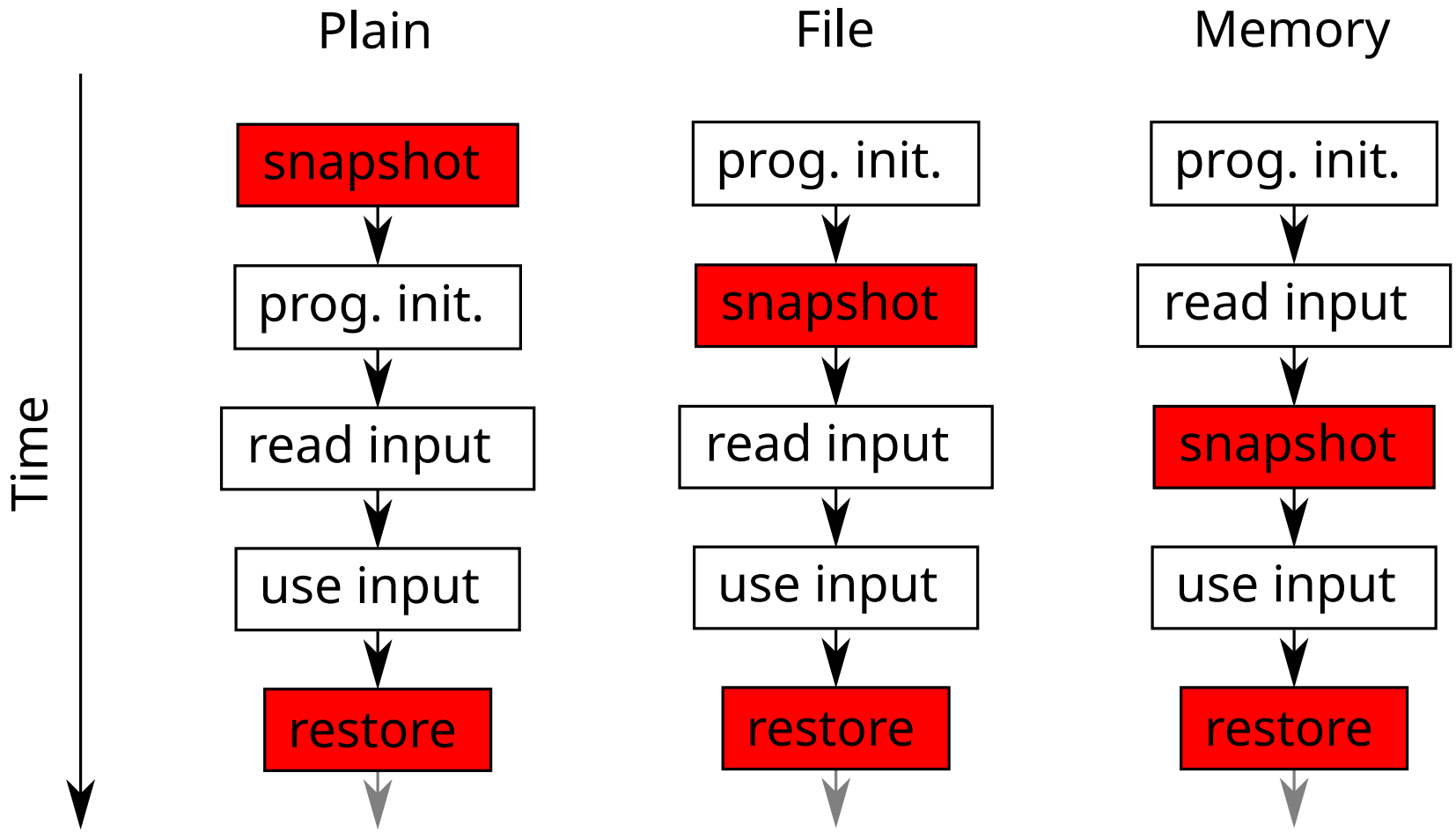
Optimization opportunities



- Program initialization is redundant
- Input data is copied before use, but does not influence the execution
- Several mutation operators leave most of the input unchanged
- Pushing the snapshot into the execution will remove redundant operations

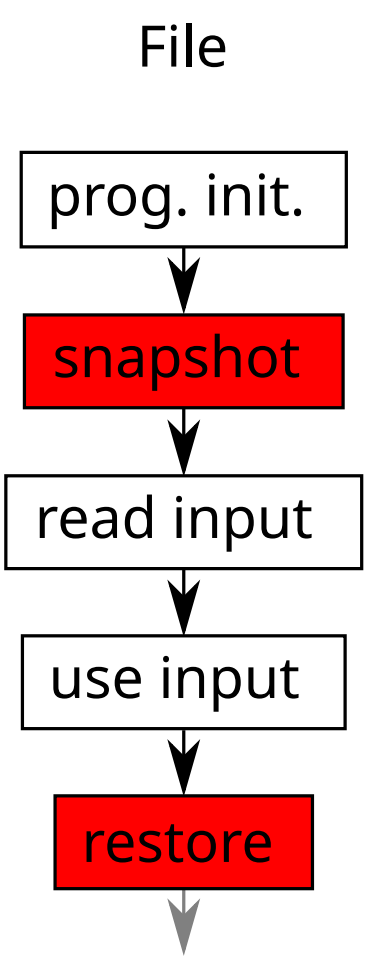
VU VRIJE UNIVERSITEIT AMSTERDAM

Snapshot positioning policies



VU VRIJE UNIVERSITEIT AMSTERDAM

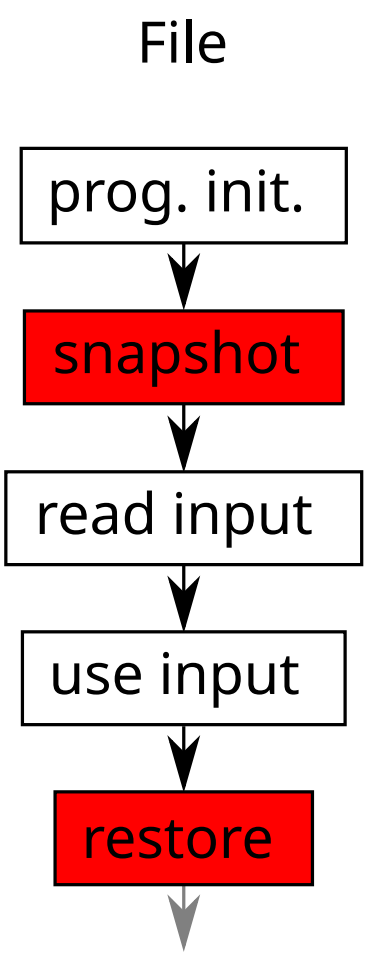
Example - File policy




```
1  int main(int argc, char* argv[argc + 1]) {
2    initialization();
3    FILE* file = fopen(argv[1], "rb");
4    ...
5    uint8_t buffer[10];
6    fread(buffer, 1, sizeof(buffer), file);
7    if (buffer[0] != 42) { exit(1); }
8    ...
9    uint8_t buffer2[sizeof(buffer)];
10   memcpy(buffer2, buffer, sizeof(buffer));
11   other_operation();
12   if (buffer2[1] == 10) { abort(); }
13   ...
14 }
```

VU VRIJE UNIVERSITEIT AMSTERDAM

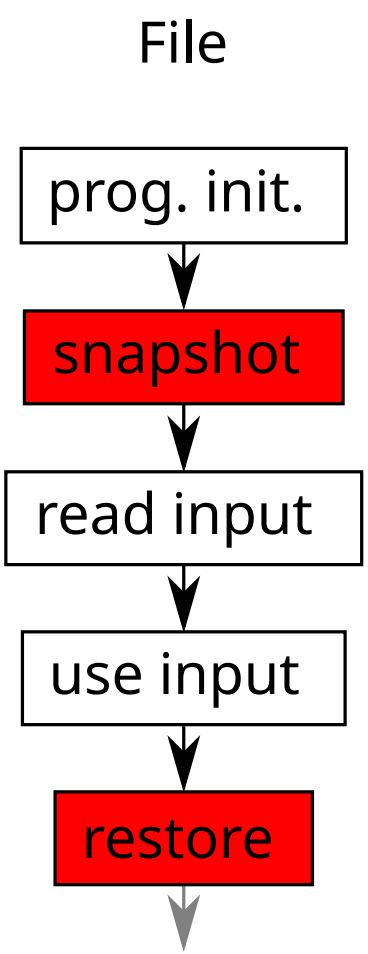
Example - File policy




```
1  int main(int argc, char* argv[argc + 1]) {
2    initialization();
3    FILE* file = fopen(argv[1], "rb");
4    ...
5    uint8_t buffer[10];
6    fread(buffer, 1, sizeof(buffer), file);
7    if (buffer[0] != 42) { exit(1); }
8    ...
9    uint8_t buffer2[sizeof(buffer)];
10   memcpy(buffer2, buffer, sizeof(buffer));
11   other_operation();
12   if (buffer2[1] == 10) { abort(); }
13   ...
14 }
```

VU VRIJE UNIVERSITEIT AMSTERDAM

Example - File policy




```
1  int main(int argc, char* argv[argc + 1]) {
2    initialization();
3    FILE* file = fopen(argv[1], "rb");
4    ...
5    uint8_t buffer[10];
6    fread(buffer, 1, sizeof(buffer), file);
7    if (buffer[0] != 42) { exit(1); }
8    ...
9    uint8_t buffer2[sizeof(buffer)];
10   memcpy(buffer2, buffer, sizeof(buffer));
11   other_operation();
12   if (buffer2[1] == 10) { abort(); }
13   ...
14 }
```

VU VRIJE UNIVERSITEIT AMSTERDAM

Memory policy

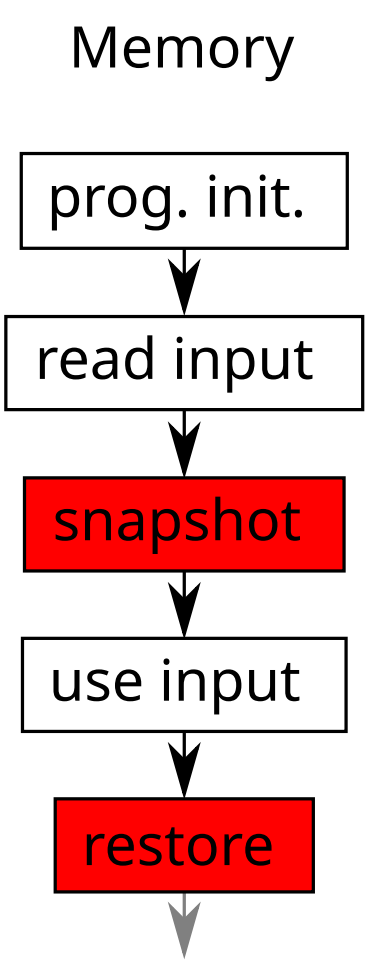
- Input bytes that will be modified are tainted
- Find first tainted load from memory through Dynamic Taint Analysis *(adaptive snapshots)*
- Record a snapshot keeping track of all tainted bytes in memory with DTA
- On restore, replace all tainted bytes with mutated ones *(mutable snapshots)*

Each new parent test case requires a new snapshot

VU VRIJE UNIVERSITEIT AMSTERDAM

Example - Memory policy

- Test cases of size 12 which differ only by their 2nd byte

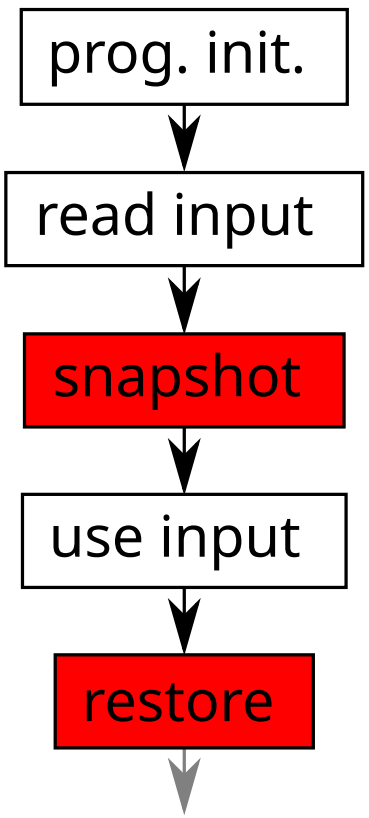



```
1  int main(int argc, char* argv[argc + 1]) {
2    initialization();
3    FILE* file = fopen(argv[1], "rb");
4    ...
5    uint8_t buffer[10];
6    fread(buffer, 1, sizeof(buffer), file);
7    if (buffer[0] != 42) { exit(1); }
8    ...
9    uint8_t buffer2[sizeof(buffer)];
10   memcpy(buffer2, buffer, sizeof(buffer));
11   other_operation();
12   if (buffer2[1] == 10) { abort(); }
13   ...
14 }
```

VU VRIJE UNIVERSITEIT AMSTERDAM

Example - Memory policy

- Test cases of size 12 which differ only by their 2nd byte

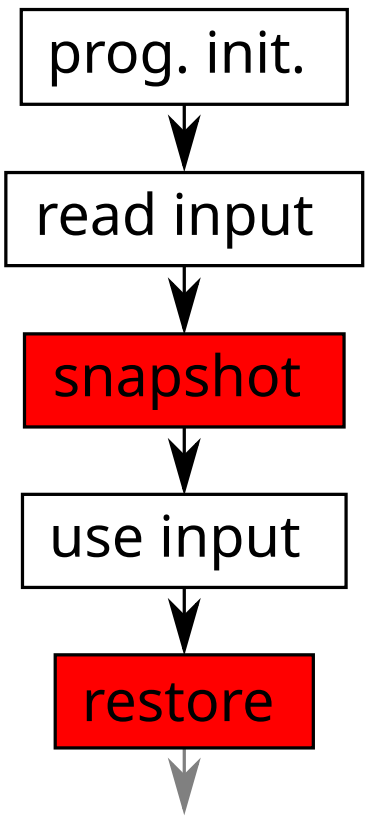



```
1  int main(int argc, char* argv[argc + 1]) {
2    initialization();
3    FILE* file = fopen(argv[1], "rb");
4    ...
5    uint8_t buffer[10];
6    fread(buffer, 1, sizeof(buffer), file);
7    if (buffer[0] != 42) { exit(1); }
8    ...
9    uint8_t buffer2[sizeof(buffer)];
10   memcpy(buffer2, buffer, sizeof(buffer));
11   other_operation();
12   if (buffer2[1] == 10) { abort(); }
13   ...
14 }
```

VU VRIJE UNIVERSITEIT AMSTERDAM

Example - Memory policy

- Test cases of size 12 which differ only by their 2nd byte

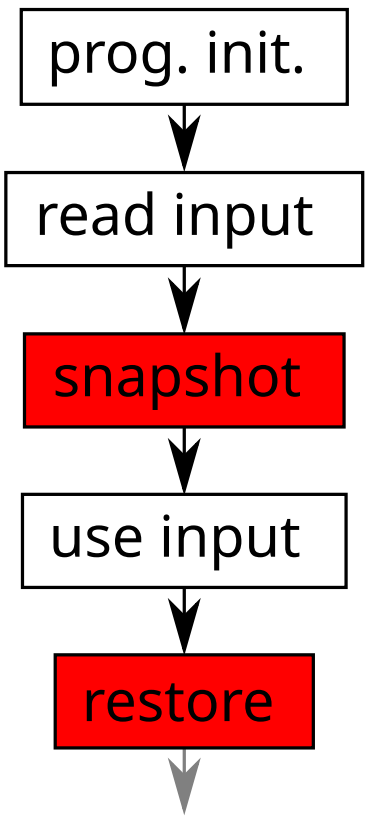



```
1  int main(int argc, char* argv[argc + 1]) {
2    initialization();
3    FILE* file = fopen(argv[1], "rb");
4    ...
5    uint8_t buffer[10];
6    fread(buffer, 1, sizeof(buffer), file);
7    if (buffer[0] != 42) { exit(1); }
8    ...
9    uint8_t buffer2[sizeof(buffer)];
10   memcpy(buffer2, buffer, sizeof(buffer));
11   other_operation();
12   if (buffer2[1] == 10) { abort(); }
13   ...
14 }
```

VU VRIJE UNIVERSITEIT AMSTERDAM

Example - Memory policy

- Test cases of size 12 which differ only by their 2nd byte

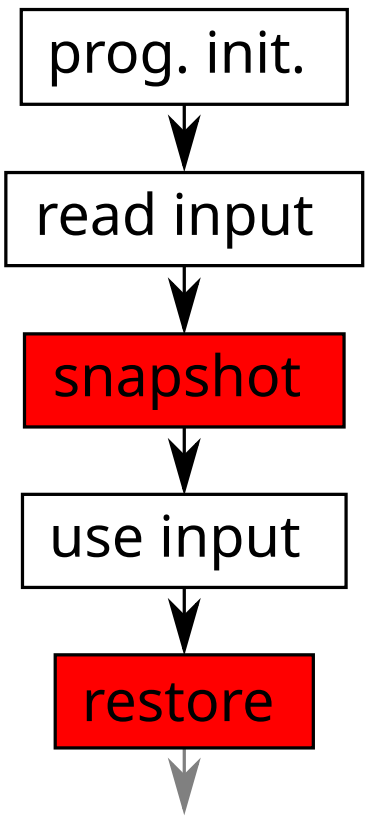



```
1  int main(int argc, char* argv[argc + 1]) {
2    initialization();
3    FILE* file = fopen(argv[1], "rb");
4    ...
5    uint8_t buffer[10];
6    fread(buffer, 1, sizeof(buffer), file);
7    if (buffer[0] != 42) { exit(1); }
8    ...
9    uint8_t buffer2[sizeof(buffer)];
10   memcpy(buffer2, buffer, sizeof(buffer));
11   other_operation();
12   if (buffer2[1] == 10) { abort(); }
13   ...
14 }
```

VU VRIJE UNIVERSITEIT AMSTERDAM

Example - Memory policy

- Test cases of size 12 which differ only by their 2nd byte

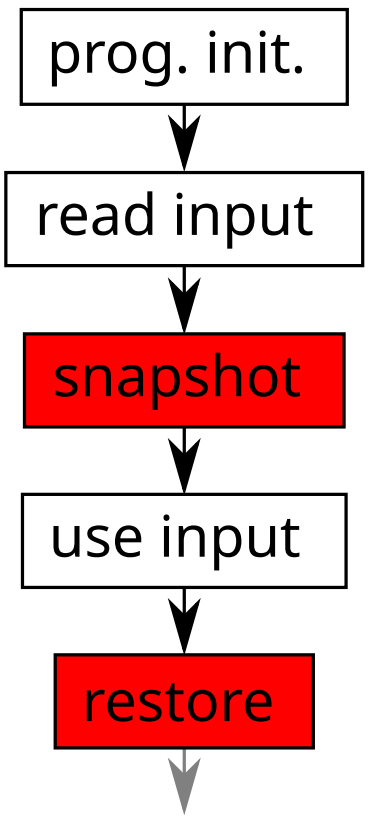



```
1  int main(int argc, char* argv[argc + 1]) {
2    initialization();
3    FILE* file = fopen(argv[1], "rb");
4    ...
5    uint8_t buffer[10];
6    fread(buffer, 1, sizeof(buffer), file);
7    if (buffer[0] != 42) { exit(1); }
8    ...
9    uint8_t buffer2[sizeof(buffer)];
10   memcpy(buffer2, buffer, sizeof(buffer));
11   other_operation();
12   if (buffer2[1] == 10) { abort(); }
13   ...
14 }
```

VU VRIJE UNIVERSITEIT AMSTERDAM

Example - Memory policy

- Test cases of size 12 which differ only by their 2nd byte

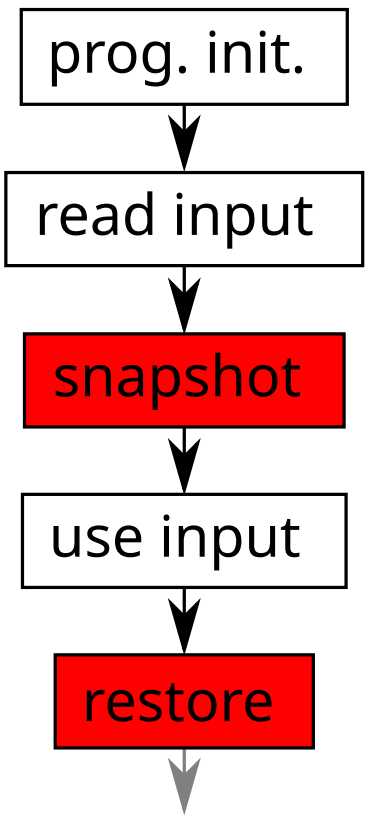



```
1  int main(int argc, char* argv[argc + 1]) {
2    initialization();
3    FILE* file = fopen(argv[1], "rb");
4    ...
5    uint8_t buffer[10];
6    fread(buffer, 1, sizeof(buffer), file);
7    if (buffer[0] != 42) { exit(1); }
8    ...
9    uint8_t buffer2[sizeof(buffer)];
10   memcpy(buffer2, buffer, sizeof(buffer));
11   other_operation();
12   if (buffer2[1] == 10) { abort(); }
13   ...
14 }
```

VU VRIJE UNIVERSITEIT AMSTERDAM

Example - Memory policy

- Test cases of size 12 which differ only by their 2nd byte

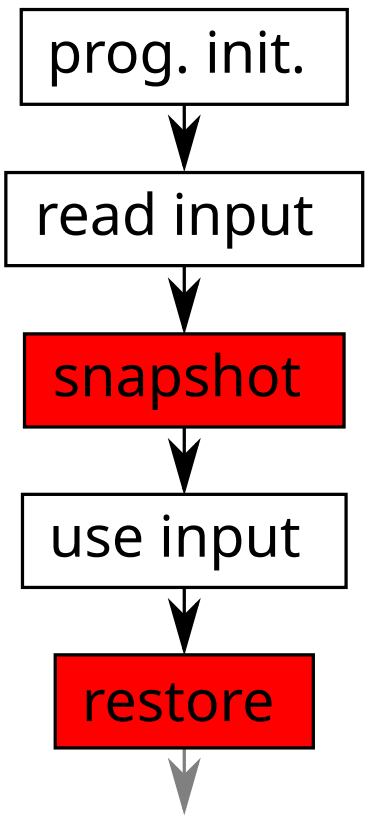



```
1  int main(int argc, char* argv[argc + 1]) {
2    initialization();
3    FILE* file = fopen(argv[1], "rb");
4    ...
5    uint8_t buffer[10];
6    fread(buffer, 1, sizeof(buffer), file);
7    if (buffer[0] != 42) { exit(1); }
8    ...
9    uint8_t buffer2[sizeof(buffer)];
10   memcpy(buffer2, buffer, sizeof(buffer));
11   other_operation();
12   if (buffer2[1] == 10) { abort(); }
13   ...
14 }
```

VU VRIJE UNIVERSITEIT AMSTERDAM

Example - Memory policy

- Test cases of size 12 which differ only by their 2nd byte




```
1  int main(int argc, char* argv[argc + 1]) {
2    initialization();
3    FILE* file = fopen(argv[1], "rb");
4    ...
5    uint8_t buffer[10];
6    fread(buffer, 1, sizeof(buffer), file);
7    if (buffer[0] != 42) { exit(1); }
8    ...
9    uint8_t buffer2[sizeof(buffer)];
10   memcpy(buffer2, buffer, sizeof(buffer));
11   other_operation();
12   if (buffer2[1] == 10) { abort(); }
13   ...
14 }
```

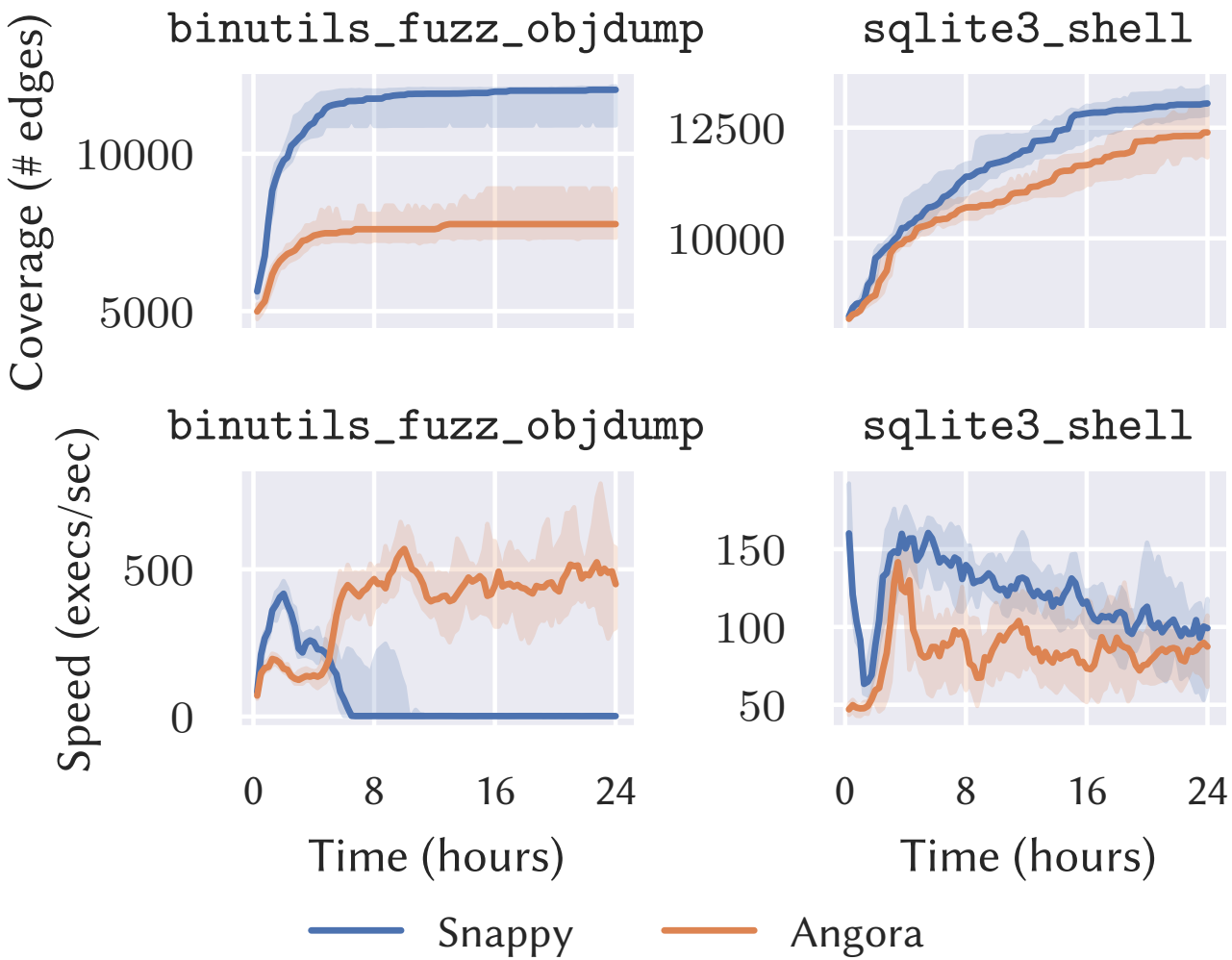
VU VRIJE UNIVERSITEIT AMSTERDAM

Other optimizations

- Dynamic switching between File and Memory policies
- Exit optimization for error handling code
- Other low level optimizations

VU VRIJE UNIVERSITEIT AMSTERDAM

Evaluation



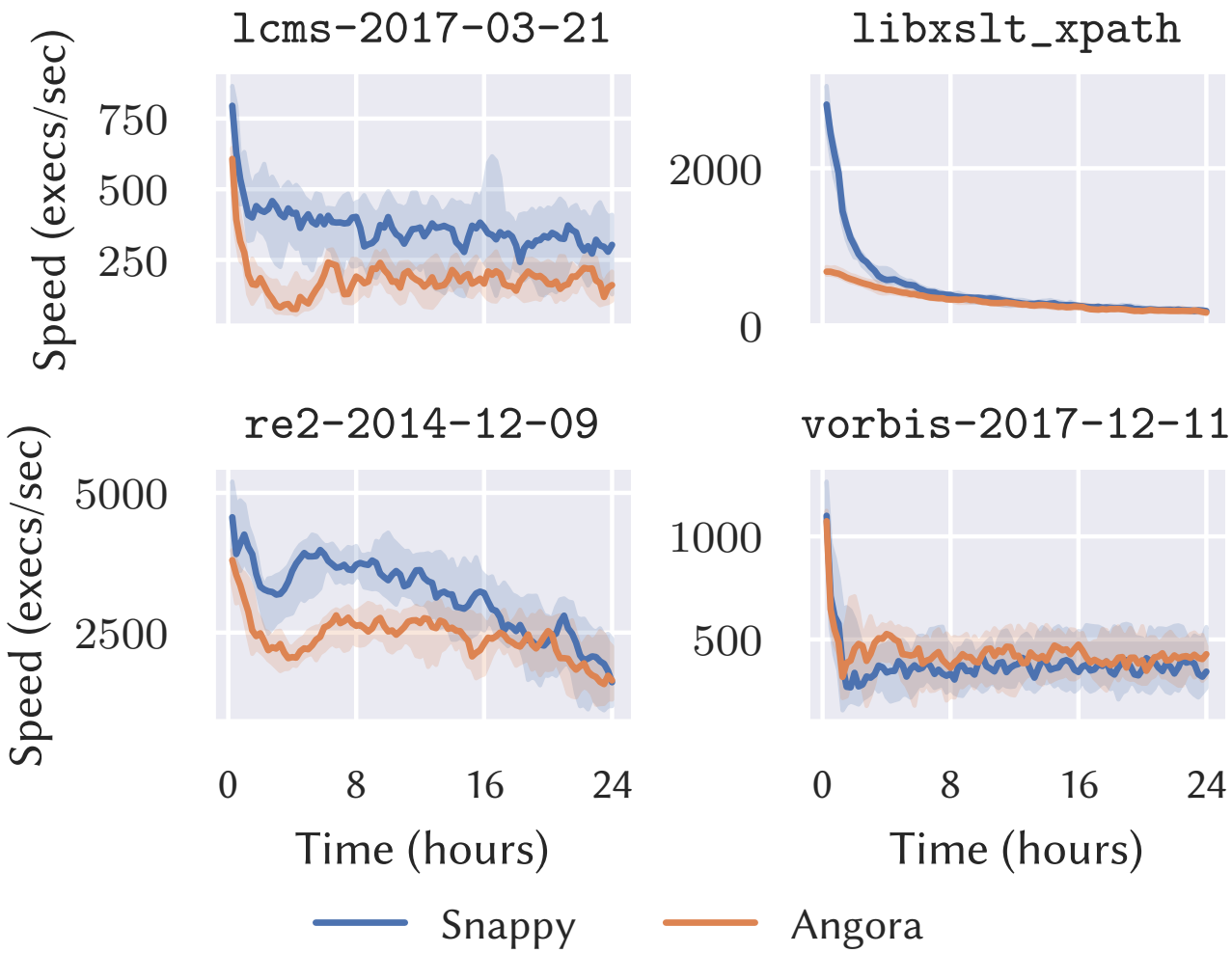
VU VRIJE UNIVERSITEIT AMSTERDAM

Conclusion

- Snappy increases fuzzing speed by moving snapshots further into the program trace
- It does so with two policies: File (simpler) and Memory (more complex)
- It produces:
  - up to 1.76× speed increase in FuzzBench, with no significant regressions
  - up to 31% coverage increase on real-world programs (24 hours benchmarks)

VU VRIJE UNIVERSITEIT AMSTERDAM

FuzzBench



VU VRIJE UNIVERSITEIT AMSTERDAM