Fuzz testing the BSD kernel

Using competitive analysis to increase the effectiveness of operating system fuzz testing

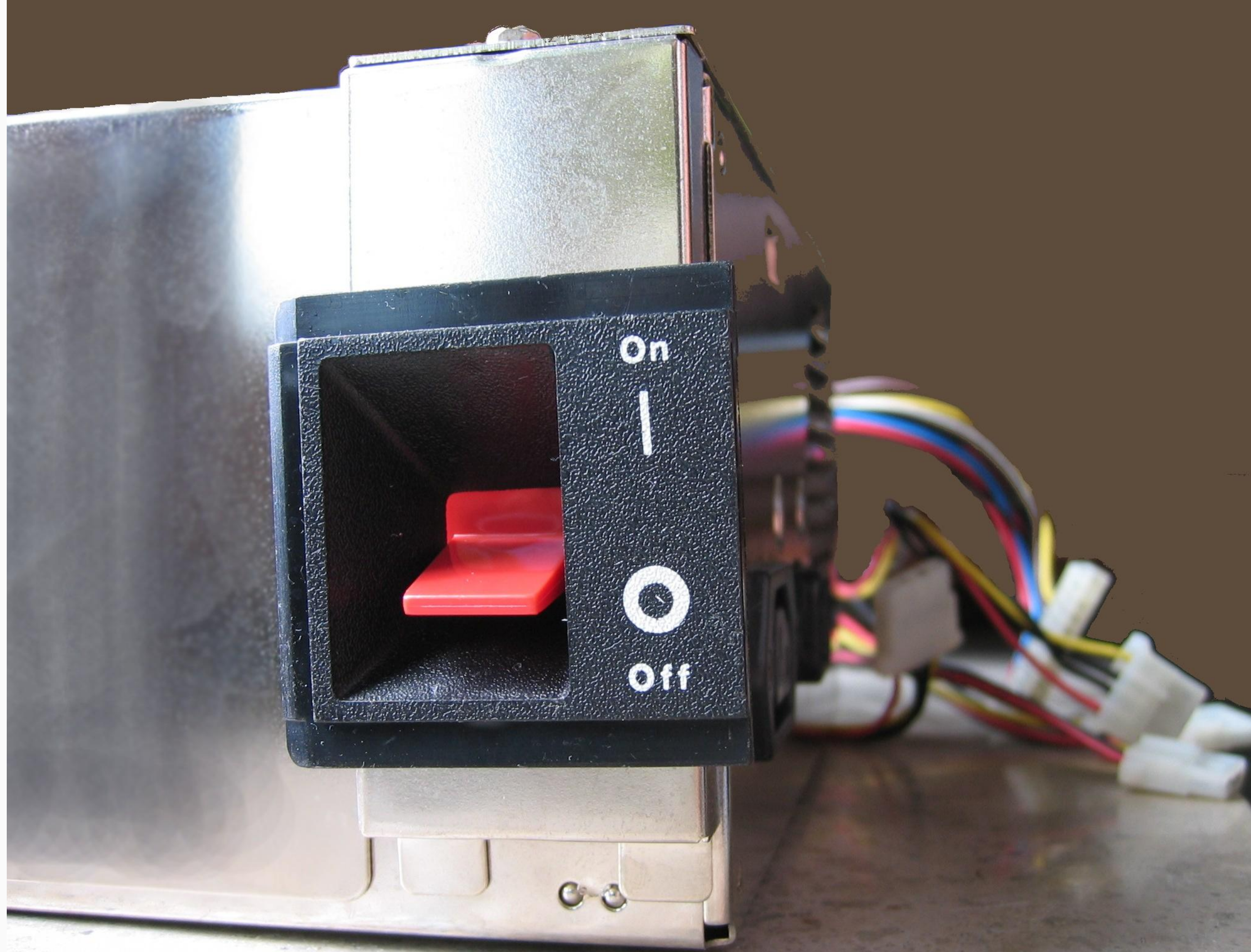
kirk.russell@ba23.org

Who is this guy?

- I was an operating system tester at QNX software systems
- I'm a former Google Storage System/Bigtable Site Reliability Engineer

Thanks to Google for allowing me to open source my file system testing framework.

This is an open source project. The views expressed on these pages are mine alone and not those of my current or past employers.


On
|
O
Off

What are you talking about?

- Terminology -- define fuzz testing

What are you talking about?

- Terminology -- define fuzz testing
- Why is fuzz testing difficult?

What are you talking about?

- Terminology -- define fuzz testing
- Why is fuzz testing difficult?
- Ideas to optimize fuzz testing.

What are you talking about?

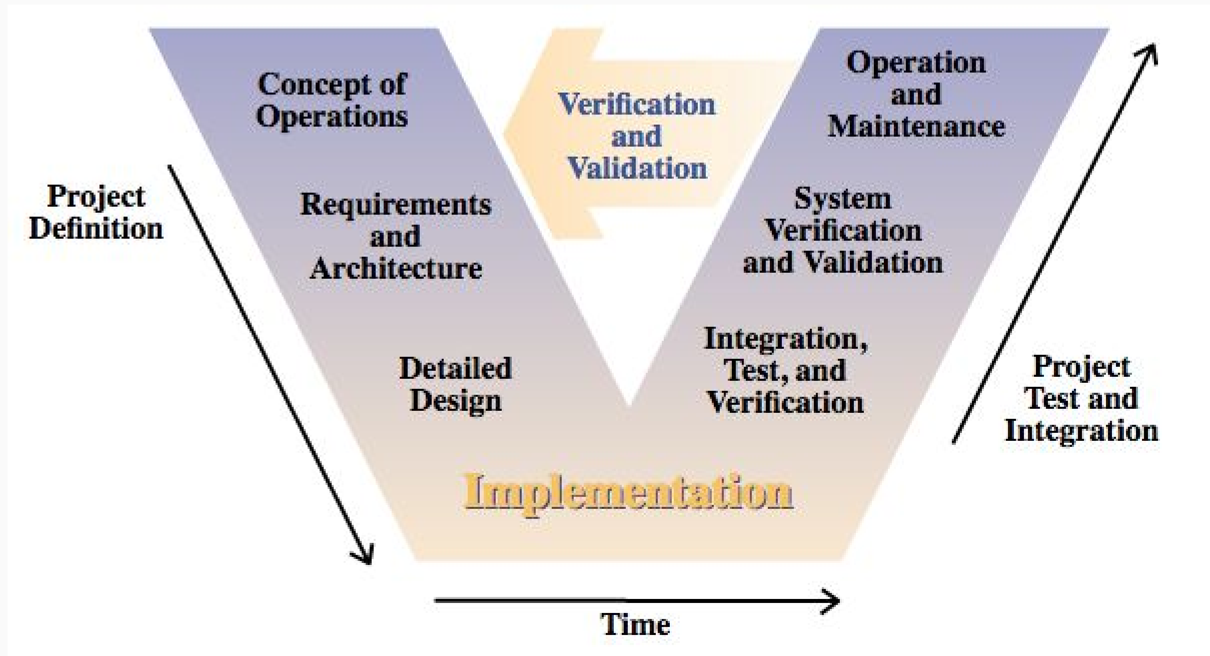
- Terminology -- define fuzz testing
- Why is fuzz testing difficult?
- Ideas to optimize fuzz testing.
- An example using FreeBSD

What are you talking about?

- Terminology -- define fuzz testing
- Why is fuzz testing difficult?
- Ideas to optimize fuzz testing.
- An example using FreeBSD
- Ideas for improvements

"Bad terminology is the enemy of good thinking" -- Warren Buffett

Testing terminology terms appear to have grown organically...



Terminology:

- **Non-functional**: tests do not relate to functionality

A successful test generates a core file.

Terminology:

- **Fuzz**: random data
- **Monkey**: random actions

A non-function test that deluges software-under-test with random stuff until software-under-test cores.

I will consider monkey to be a subset of Fuzz -- just a different interface.

Terminology:

- **Exploratory**: unscripted, iterative test design/implementation/execution

Testing frameworks should make is easy for experimentation.

Also make it easy for computers to iterative...

Fuzzing structured data

What kind of interfaces can you fuzz?
Protocols: structured interfaces

- Here is a black box approach

```
printf "int main() { return 0; }\n" > true.c
cc -o a.out true.c
mangle a.out "$( wc -c < ./a.out)"
ulimit -c 0
ulimit -t 1
./a.out
```

What kind of interfaces can you fuzz?
Protocols: structured interfaces

- White box approach:

- Mp3 music files can have metadata at the start and end of file
- ID3v1 tags are 128 bytes total and at end of mp3 file
- ID3v2 tags are variable sized and live at start of the file
- Fuzzing these tags and not the audio data part will find more bugs!

Fuzz testing of kernels is not new.

Classic fuzz testing...

- crashme.c -- execute random machine instructions

```
$ crashme +2000 666 100 1:00:00
Crashme: (c) Copyright 1990-1994 George J. Carrette
Version: 2.4 20-MAY-1994
crashme +2000 666 100 1:00:00
Subprocess run for 3600 seconds (0 01:00:00)
pid = 15628 0x3D0C (subprocess 1)
crashme: Bad address
pid 15628 0x3D0C exited with status 256
pid = 15629 0x3D0D (subprocess 2)
crashme: Bad address
```

Classic fuzz testing...

- fsx.c -- file system exerciser

```
$ fsx -d -P /tmp afile
truncating to largest ever: 0x13e76
1 trunc    from 0x0 to 0x13e76
2 write    0x17098 thru    0x26857    (0xf7c0 bytes)
3 read     0xc73e thru     0x1b801    (0xf0c4 bytes)
4 mapwrite 0x32e00 thru    0x331fc    (0x3fd bytes)
5 mapwrite 0x7ac1 thru     0x11029    (0x9569 bytes)
6 read     0x1f62e thru    0x2177f    (0x2152 bytes)
7 write    0x756 thru 0xede (0x789 bytes)
8 read     0x18f13 thru    0x27d18    (0xee06 bytes)
9 read     0xf369 thru     0x1b0af    (0xbd47 bytes)
10 mapwrite     0x17461 thru    0x19892    (0x2432 bytes)
```

Classic fuzz testing...

- Ballista OS Robustness Test Suite

```
Unistd.h function int dup    b_fd
Unistd.h function int execvp b_fname b_ptr_ptr_char
```

- Fuzz a single call
- Different combo of invalid and valid args

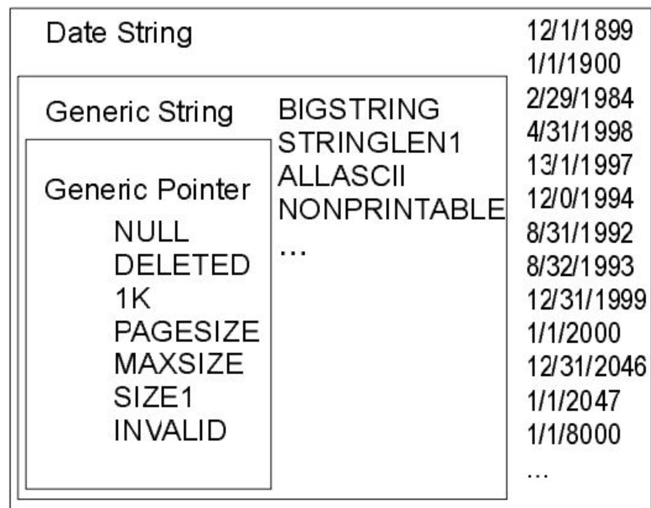


Figure 2. A date string data type inherits test cases from generic string and generic pointer.

What problem are you trying to solve?

Pesticide Paradox: fuzz tests are not immune

A test strategy becomes ineffective as the bugs get fixed.

"The phenomenon that the more you test software, the more immune it becomes to your tests - just as insects eventually build up resistance and the pesticide no longer works."   [Beizer]

The burden on the kernel developers is too high -- Long test, debug and fix cycles

""The SPARC Linux kernel is remarkably stable; David now requires that every kernel pass a "crashme" test for about 24 hours before releasing the source code for it." Linux Journal Issue #27/July 1996

What if the bug/corruption happens in hour #1 but the kernel doesn't panic until hour #22?

The burden on the kernel developers is too high -- complexity.

If we could somehow reduce the test program to two/three kernel calls and the execution time to 30 seconds, you will have a better chance of getting a kernel developer to fix this bug!

The burden on the kernel developers is too high -- regression tests.

How do you plan to turn this into a simple regression test?

What problems are you trying to solve?

We want a fuzz test framework that:

- Continues to find new bugs -- Pesticide Paradox resistant

What problem are you trying to solve?

We want a fuzz test framework that:

- Continues to find new bugs -- Pesticide Paradox resistant
- Reproduces bug with minimal time and minimal code

What problem are you trying to solve?

We want a fuzz test framework that:

- Continues to find new bugs -- Pesticide Paradox resistant
- Reproduces bug with minimal time and minimal code
- Test cases can be added to a regular regression test

"The true test of a good idea is if it lasts through the hangover." -- Robert Krten

How do you defend against the paradox? More complexity!

- I will ignore randomness, ordering and threading....
- Could crashme.c and fsx.c only be O(N)?

Would increasing the complexity of the fuzz strategy slow down the effects of the pesticide paradox?

How do you defend against the paradox? More complexity!

- Would Madlibs approach be considered $O(N^2)$?
  - create one list of objects -- files, fifos, directories, symlinks,...
  - create another list of operations -- open, readdir, truncate,...

Adding a new object or operation increases the surface area by N, not 1.

FreeBSD 6.1 -- No strategy for buffer at

```
    unlink("afifo");
    mkfifo("afifo", 0666);
    truncate("afifo", 16000);
```

FreeBSD 6.1 -- No strategy for buffer at

```
    unlink("afifo");
    mkfifo("afifo", 0666);
    truncate("afifo", 16000);
```

UNIX98 says "If the file is not a regular file or a shared memory object, the result is unspecified."

FreeBSD 6.1 -- No strategy for buffer at

Section 2 or function definitions	"unspecified" or "undefined"
SUSV2	659
SUSV3	1160
Freebsd 9.3	24

The Madlibs $O(N^2)$ approach can help increase test coverage that might be missed with a $O(N)$ approach.

What are you really fuzzing?

- Regular execution paths and not focused on exceptions
- Random execution of valid kernel call traces

- The ordering of kernel calls
- The objects used by the kernel calls
- The actual set of kernel calls in the competition

Example of operations:

- Every kernel call that I think can panic kernel
- LD_PRELOAD= random file under test
- Dynamically create #! scripts that reference files under test
- Rename files between directories
- lseek and writes
- gcore
- open() is all combinations of flags -- O_TRUNC on a directory
- mmap(): use cases from ar, cp, and file utilities
- Different filename lengths -- more about that later…..

How do you make work easier for kernel devs?

Especially after you increased complexity...

Frameworks and automation should make our lives easier.

If they don't, then you need a new model...

Frameworks -- First try -- a complete mess

Spaghetti Code

Test frameworks

Creating an API between the tests and execution is great idea

Frameworks -- Second try -- easier to write tests

Fuzz generation

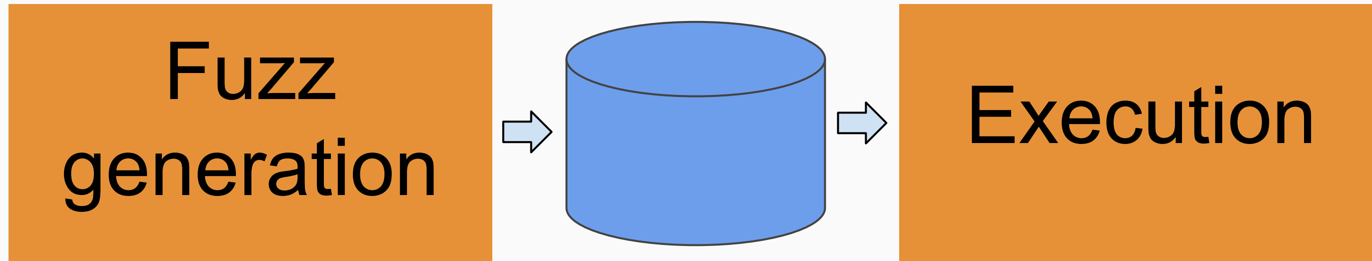
Execution

Split fuzz test frameworks in half -- use formated data

Needs to use a real data format not just a programming API:

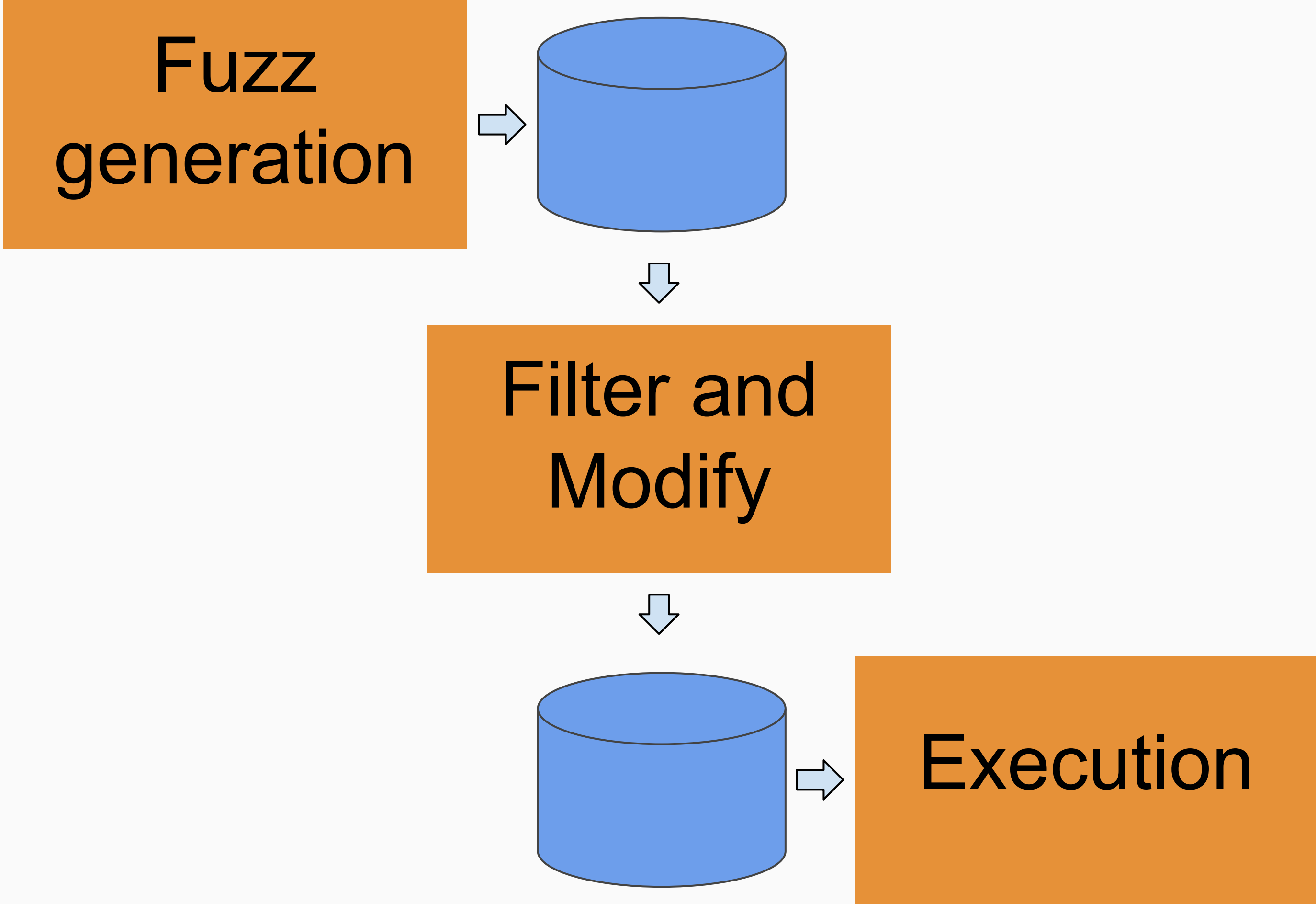
- execution engine takes operations/operands as input data
- operation/operand list is generated independently by another tool.

Frameworks -- Third try -- generation and execution are uncoupled

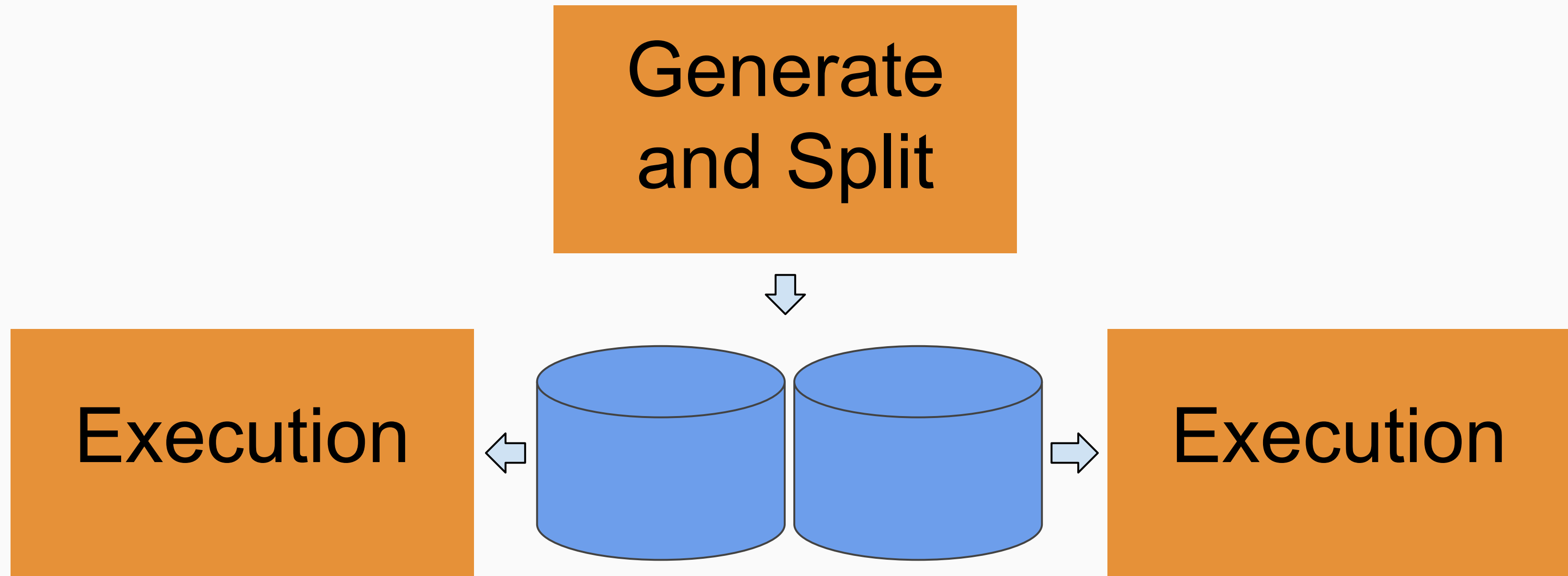


This seems like a good idea.....

- "Write programs that do one thing and do it well."
- "Write programs to work together."
- "Write programs to handle text streams, because that is a universal interface."


Fuzz generation
Filter and Modify
Execution

Competition: an example of a filter



Have a dataset competition

The operations/operands are in a data file now, so you can:

- Create two random sets with disjoint operations
- Have a competition -- treat finding a kernel panic as a game:
  - set that causes a panic first wins
  - If there is no winner, regenerate a new random set and start again
  - a winning file contains a collection of culprit operations/operands
  - Continue the competition with half the number of operations each time.

Why use the term competition and champion?

- You are trying to converge on bug at a time.
- This will likely be bug with the most aggressive behaviours
- When we eliminate operations, we will be also be removing other bugs

What does a good framework look like?

A fuzz testing framework that can:

- Be Immune to the pesticide paradox

A fuzz testing framework that can:

- Be Immune to the pesticide paradox
- Reproduce failure case in seconds, not hours/days

A fuzz testing framework that can:

- Be Immune to the pesticide paradox
- Reproduce failure case in seconds, not hours/days
- Reduce failure with a minimal test case

A fuzz testing framework that can:

- Be Immune to the pesticide paradox
- Reproduce failure case in seconds, not hours/days
- Reduce failure with a minimal test case
- Produce tests that can be incorporated into a simple regression test suite

A fuzz testing framework that can:

- Be Immune to the pesticide paradox
- Reproduce failure case in seconds, not hours/days
- Reduce failure with a minimal test case
- Produce tests that can be incorporated into a simple regression test suite
- Be Fully automated -- no manual steps required

A fuzz testing framework that can:

- Be Immune to the pesticide paradox
- Reproduce failure case in seconds, not hours/days
- Reduce failure with a minimal test case
- Produce tests that can be incorporated into a simple regression test suite
- Be Fully automated -- no manual steps required
- Take advantage of dozens of machines -- use scale to find more bugs

Strange stuff...

Problem: Beware of hidden objects

```
fd = open(tk[0], O_CREAT|O_RDWR, 0777);
truncate(fd, 20480);
close(fd);
execve(tk[0], tk, environ);
```

FreeBSD 6.1 -- panic: vnode_pager_getpages: unexpected missing page

Sparse files are a type of object to add to your list of objects.

What do we do with filenames?

How do we find the existing on disk objects with pathnames?

Just search the disk for the existing objects?

Do we expect the execution environment to just know all the paths?

Do we just generate simple filenames?

Problem: Beware of hidden objects

Filenames of different lengths are stored in different places?

Filename length	Storage on disk
1 to 16	Original metadata
17 to 48	Metadata extension
49 to 505	Special hidden file

Problem: Beware of hidden objects

FreeBSD fast filesystem

- Short symbolic links are stored in the inode (120 bytes)

4.4BSD?

"~~The internet?~~ That thing is still around?" - Homer Simpson

Enter Journaled Soft-Updates

Dr. McKusick gives 2010 BSDCan presentation:

> "Adding 'journaling lite" to soft updates and its incorporation into the FreeBSD fast file system"

There has to be a couple of latent bugs introduced. Can I find them?

Prototype: just try to produce a panic

Fuzz generation

Execution

- Provide general purpose test execution framework
- Two libraries: test operations vs operands/objects
- Easy to add new ideas to the libraries
- Stuff programming API -- not data format yet

Prototype: just try to produce a panic

After 6-8 hours of test execution, kernel panics but only when using Journaled Soft-Updates.

There is a latent bug.

Prototype: just try to produce a panic

After 6-8 hours of test execution, kernel panics but only when using Journaled Soft-Updates.

There is a latent bug.

Dijkstra was right!

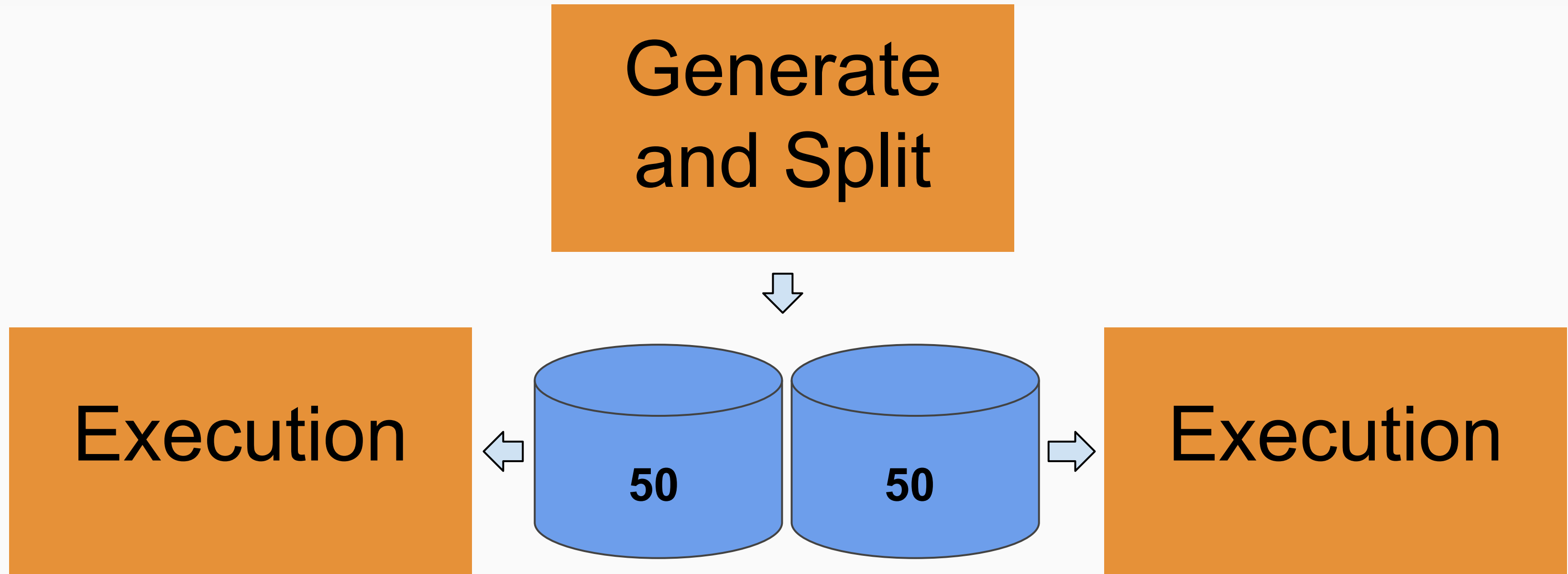
"If debugging is the process of removing bugs, then programming must be the process of putting them in." -- Dijkstra

Prototype: just try to produce a panic

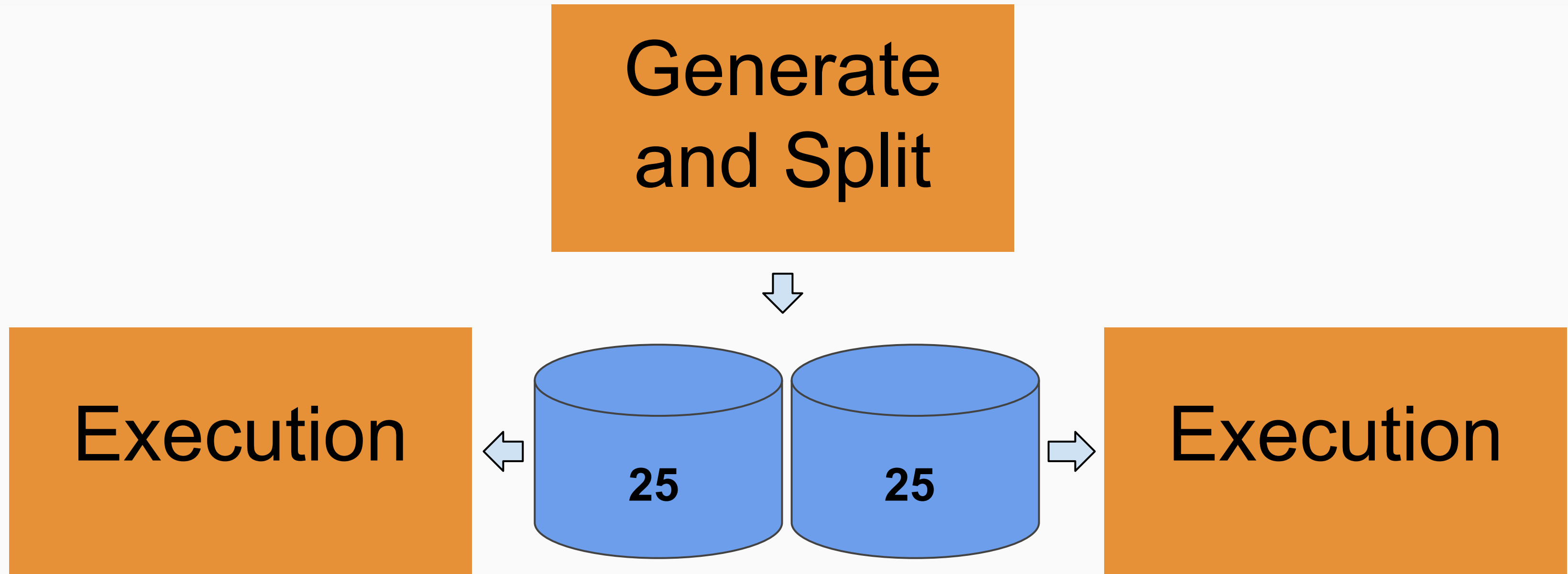
Now add data format support to this execution engine.

I can then experiment with mods/filters to reduce complexity.

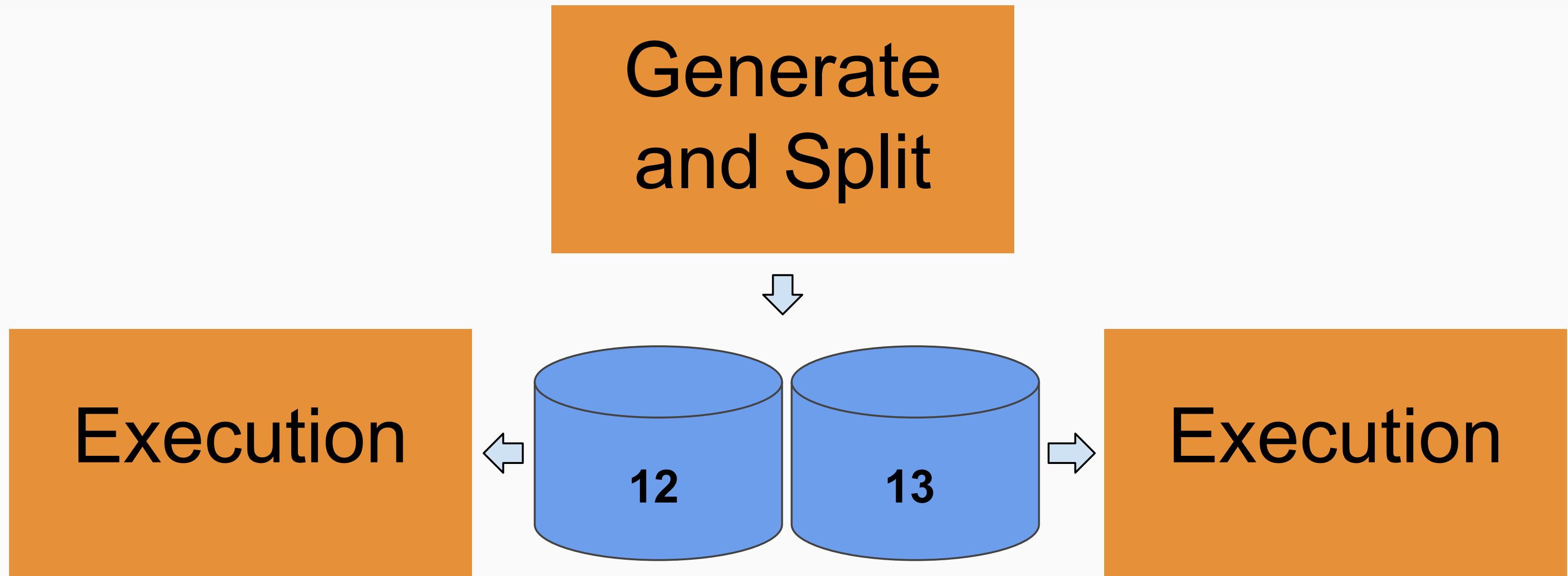
Competitions: Start with 100 operations



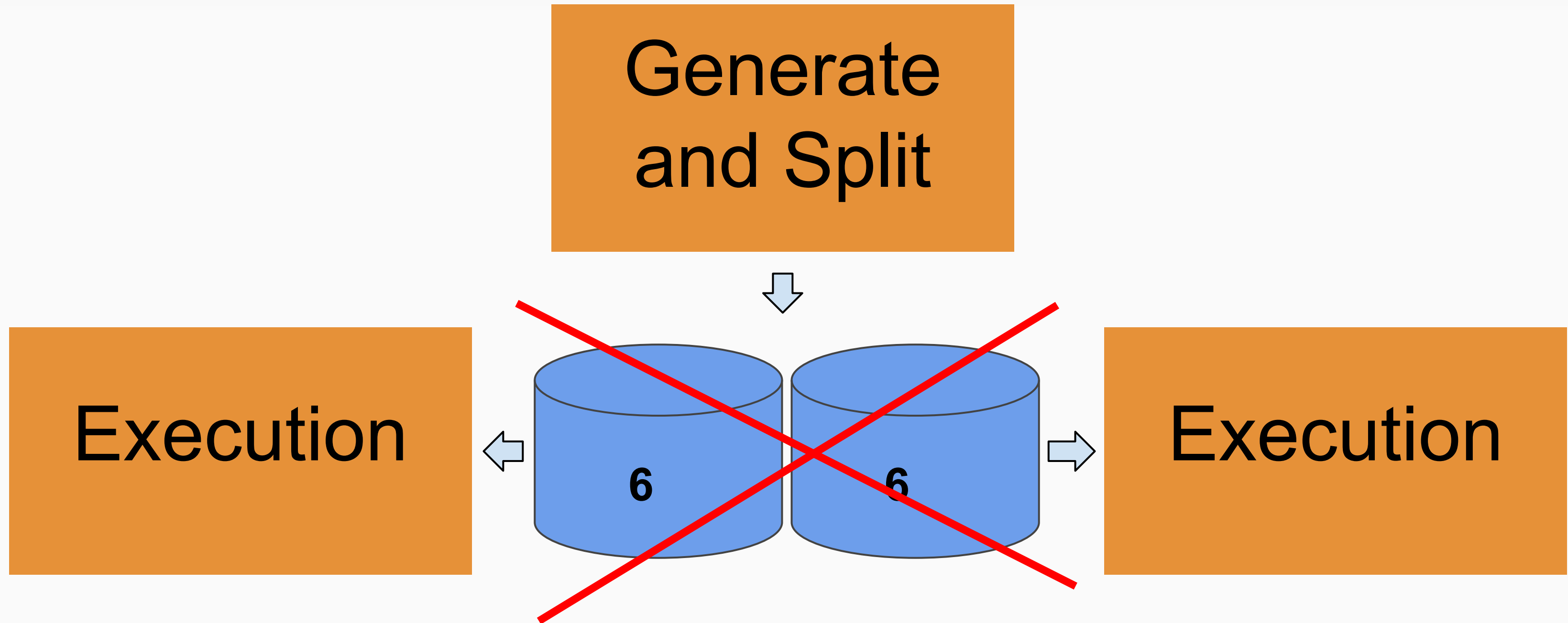
Competitions: Round Two



Competitions: Round Three



I was able to reduce from 100 operations to 12 operations



12 operations --
can we do better?

Culprit reduction -- operation competition

The operations/operands are in a data file now, so you can exclude one operation/operand to see if is the part of the culprit (or a NOP)

Culprit reduction -- 12 different datasets -- 11 operations each

	2	3	4	5	6	7	8	9	A	B	C

NOPE

Culprit reduction -- 12 different datasets -- 11 operations each

	2	3	4	5	6	7	8	9	A	B	C
1		3	4	5	6	7	8	9	A	B	C

NOPE
PANIC

Culprit reduction -- 12 different datasets -- 11 operations each

	2	3	4	5	6	7	8	9	A	B	C	NOPE
1		3	4	5	6	7	8	9	A	B	C	PANIC
1	2		4	5	6	7	8	9	A	B	C	NOPE
1	2	3		5	6	7	8	9	A	B	C	NOPE
1	2	3	4		6	7	8	9	A	B	C	PANIC
1	2	3	4	5		7	8	9	A	B	C	PANIC
1	2	3	4	5	6		8	9	A	B	C	PANIC
1	2	3	4	5	6	7		9	A	B	C	PANIC
1	2	3	4	5	6	7	8		A	B	C	NOPE
1	2	3	4	5	6	7	8	9		B	C	PANIC
1	2	3	4	5	6	7	8	9	A		C	PANIC
1	2	3	4	5	6	7	8	9	A	B		PANIC

Culprit reduction -- the champion archive: 4 operations/30 seconds

	2	3	4	5	6	7	8	9	A	B	C	NOPE
1	2		4	5	6	7	8	9	A	B	C	NOPE
1	2	3		5	6	7	8	9	A	B	C	NOPE
1	2	3	4	5	6	7	8		A	B	C	NOPE

1	3	4	9

Culprit reduction -- and the winners are...

- open()/write()
- /usr/bin/gcore -c
- link()
- unlink()

Notice that close() is missing. For more details....

https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=159971

Report card:

- Be Immune to the pesticide paradox
- Reproduce failure case in seconds, not hours/days
- Reduce failure with a minimal test case
- ~~Produce tests that can be incorporated into a simple regression test suite~~
- Can be fully automated
- Take advantage of dozens of machines -- use scale to find more bugs

Ideas for improvements: Use existing frameworks

- For bug reports, try to use an existing execution framework
- `pjdfstest` is already in the FreeBSD source tree, as an example
- Use `pjdfstest` as a test specification language
  - pjdfstest.c becomes execution engine
  - The shell scripts are the operation and operand lists
- Or translate dataset into `pjdfstest` scripts

Ideas for improvements: better data format

- Today, I just use a list of operations
- Maybe the ballista formal approach is better

Comments?