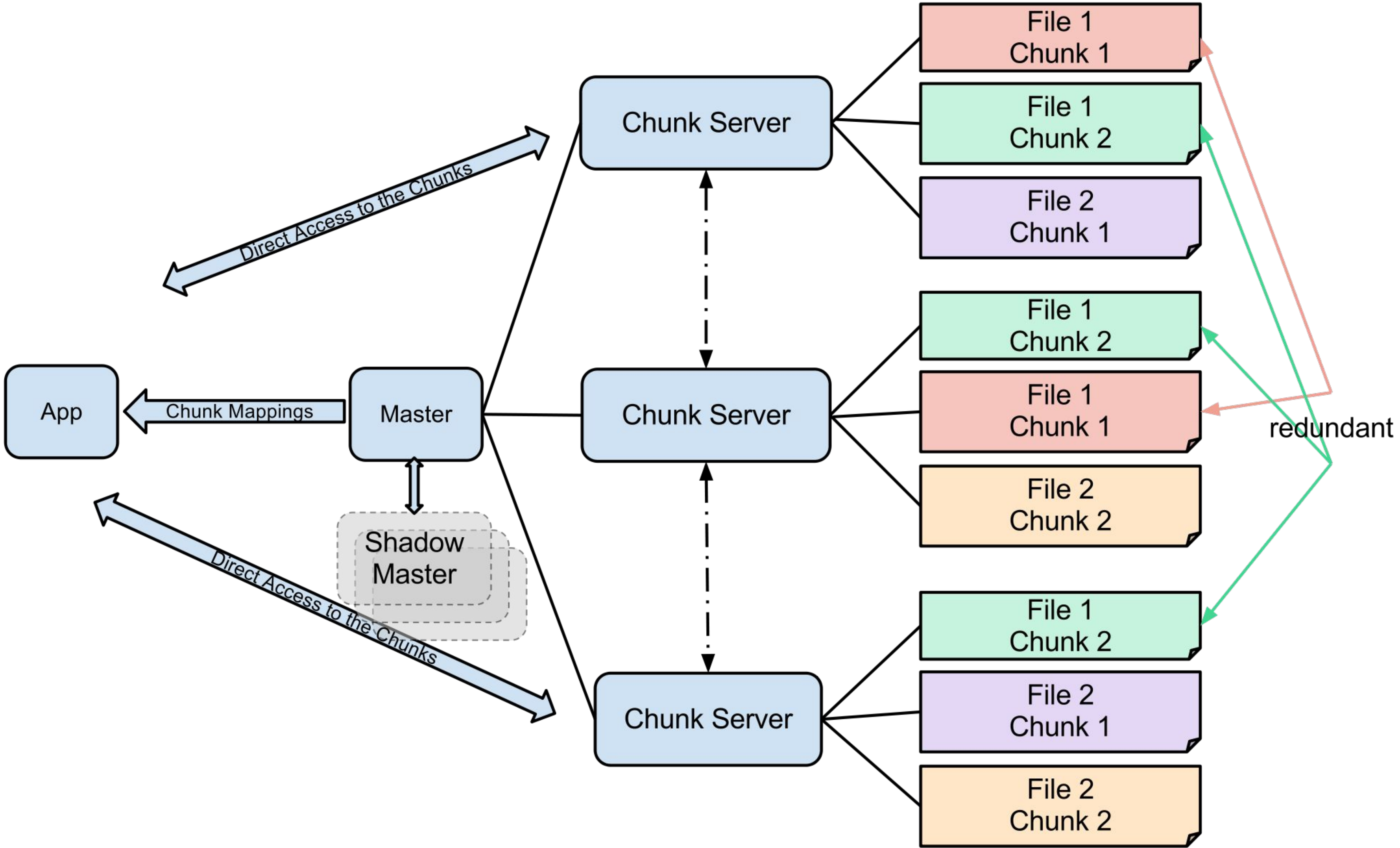
Extreme OS Kernel Testing

http://fuzz.ba23.org/

kirk.russell@shopify.com

Who is this guy?

I'm a former Google Storage Site Reliability Engineer.


Direct Access to the Chunks
App
Chunk Mappings
Master
Shadow Master
Direct Access to the Chunks
Chunk Server
Chunk Server
Chunk Server
File 1 Chunk 1
File 1 Chunk 2
File 2 Chunk 1
File 1 Chunk 2
File 1 Chunk 1
File 2 Chunk 2
File 1 Chunk 2
File 2 Chunk 1
File 2 Chunk 2
redundant

Who is this guy?

I'm a former Google Storage Site Reliability Engineer.

I was an operating system tester at QNX software systems.

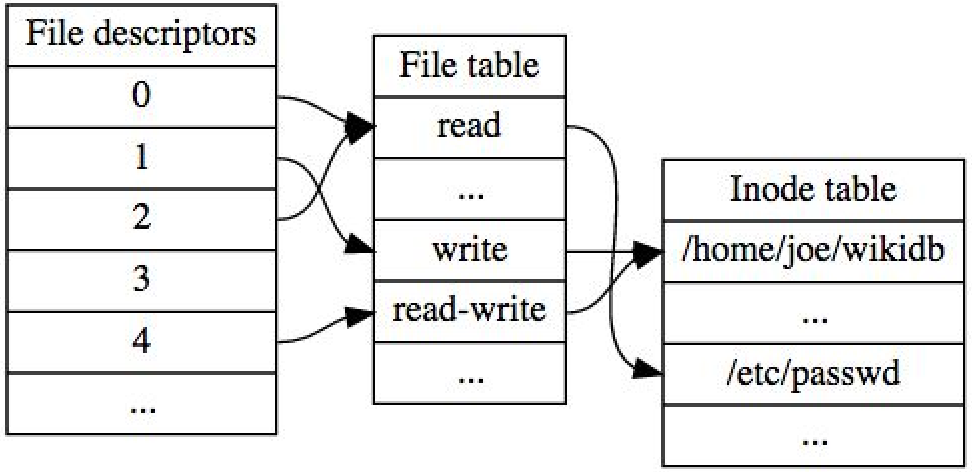
Who is this guy?

I'm a former Google Storage Site Reliability Engineer.

I was an operating system tester at QNX software systems.

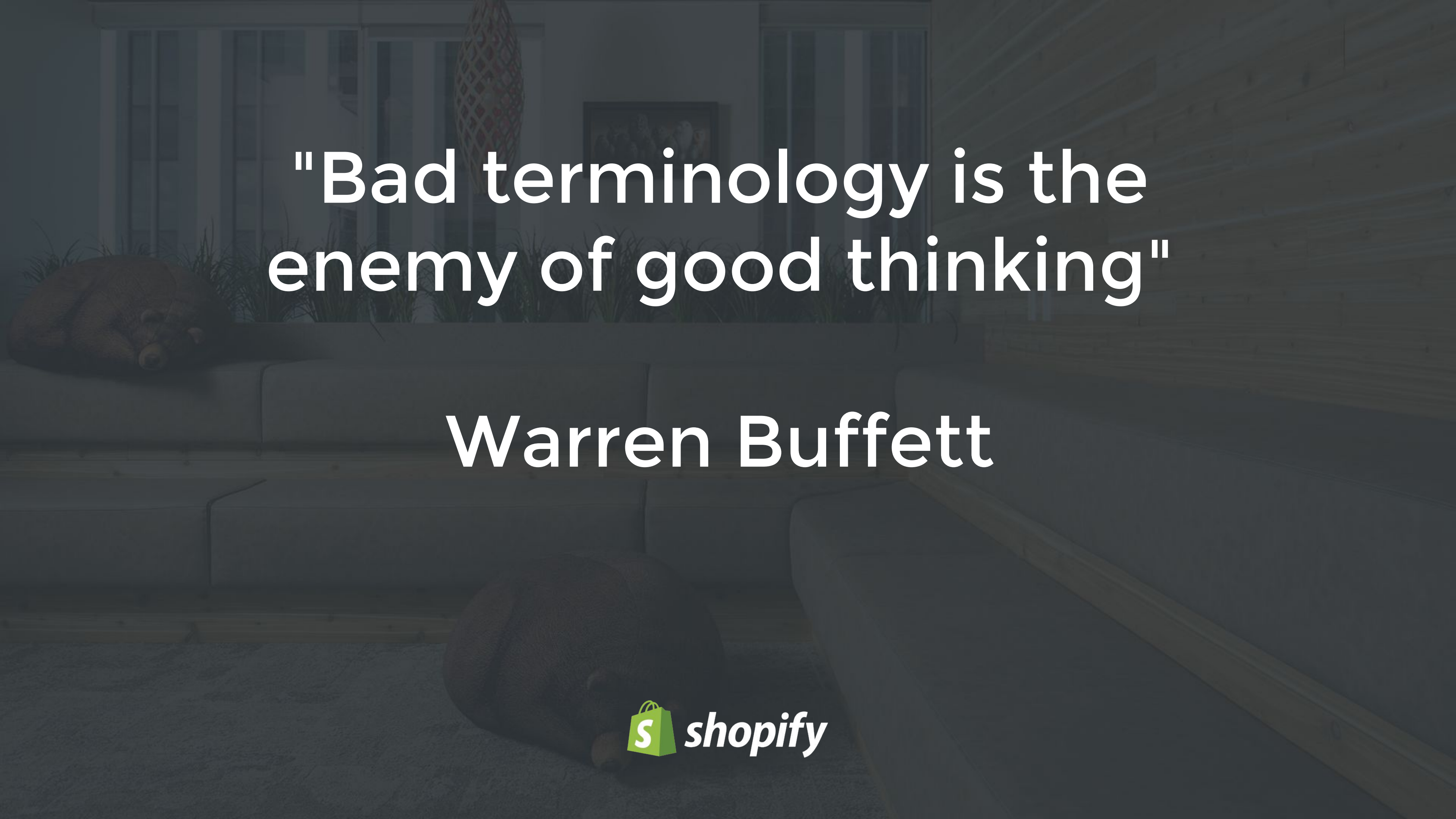
Currently, a Production Engineer at Shopify.

What tricks can an kernel tester learn from an SRE?



What are you talking about today?

- Define fuzz testing
- Give an example of kernel fuzz testing
- Problems with fuzz testing
- Ideas for a new framework
- Provide an example using FreeBSD.



"Bad terminology is the enemy of good thinking"

Warren Buffett

shopify

Terminology

- Non-functional: tests do not relate to functionality

Terminology

- Non-functional: tests do not relate to functionality
- Fuzz: non-function test that deluges software-under-test with random stuff

Terminology

- Non-functional: tests do not relate to functionality
- Fuzz: non-function test that deluges software-under-test with random stuff
- Exploratory: unscripted, iterative test design/implementation/execution

Classic Fuzz Testing

shopify

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
```

Problems with fuzz testing.

Pesticide Paradox

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]

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?

I want a new 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

How do you defend against the paradox?

More complexity!

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

Would increasing the complexity of the fuzz strategy slow down the effects of the pesticide 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."

What are you really fuzzing?

What are you really fuzzing?

Regular execution paths and not focused on exceptions.

Random execution of valid kernel call traces.

What are you really fuzzing?

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
- lseek and writes -- sparse files
- gcore -- appears to run code with extra assertions
- open() is all combinations of flags -- O_TRUNC on a directory
- mmap(): use cases from ar, cp, and file utilities

How do you make work easier for the kernel devs?

shopify

How do you make work easier for the kernel devs?

Especially after you increased complexity...

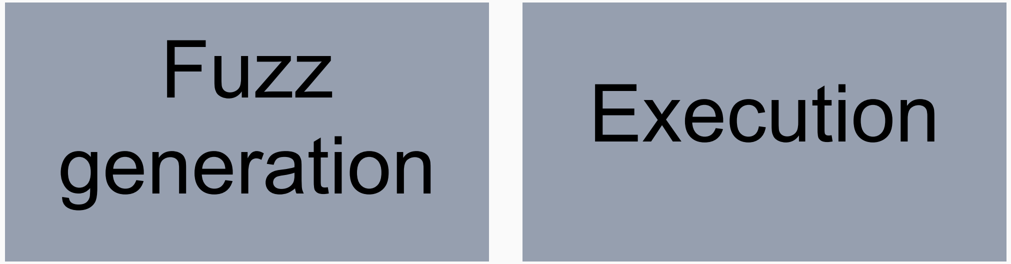
Frameworks and automation should make our lives easier.

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

Frameworks -- First try -- a complete mess:

Spaghetti Code

Create an API between the tests and execution:

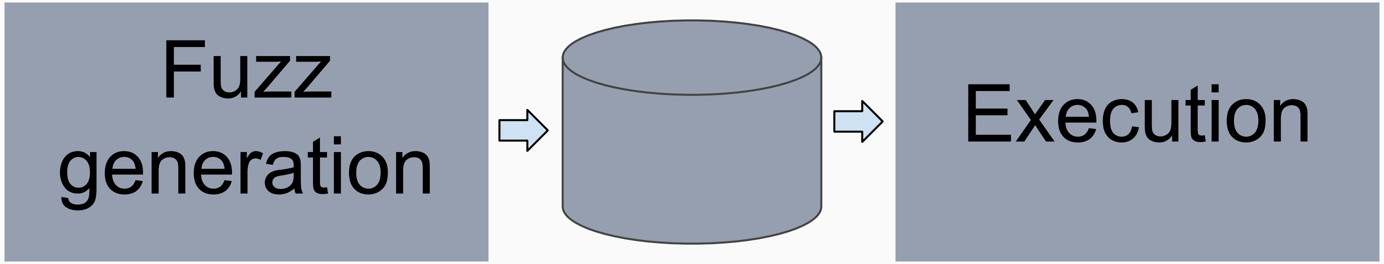


Split fuzz test frameworks in half -- use formated data

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

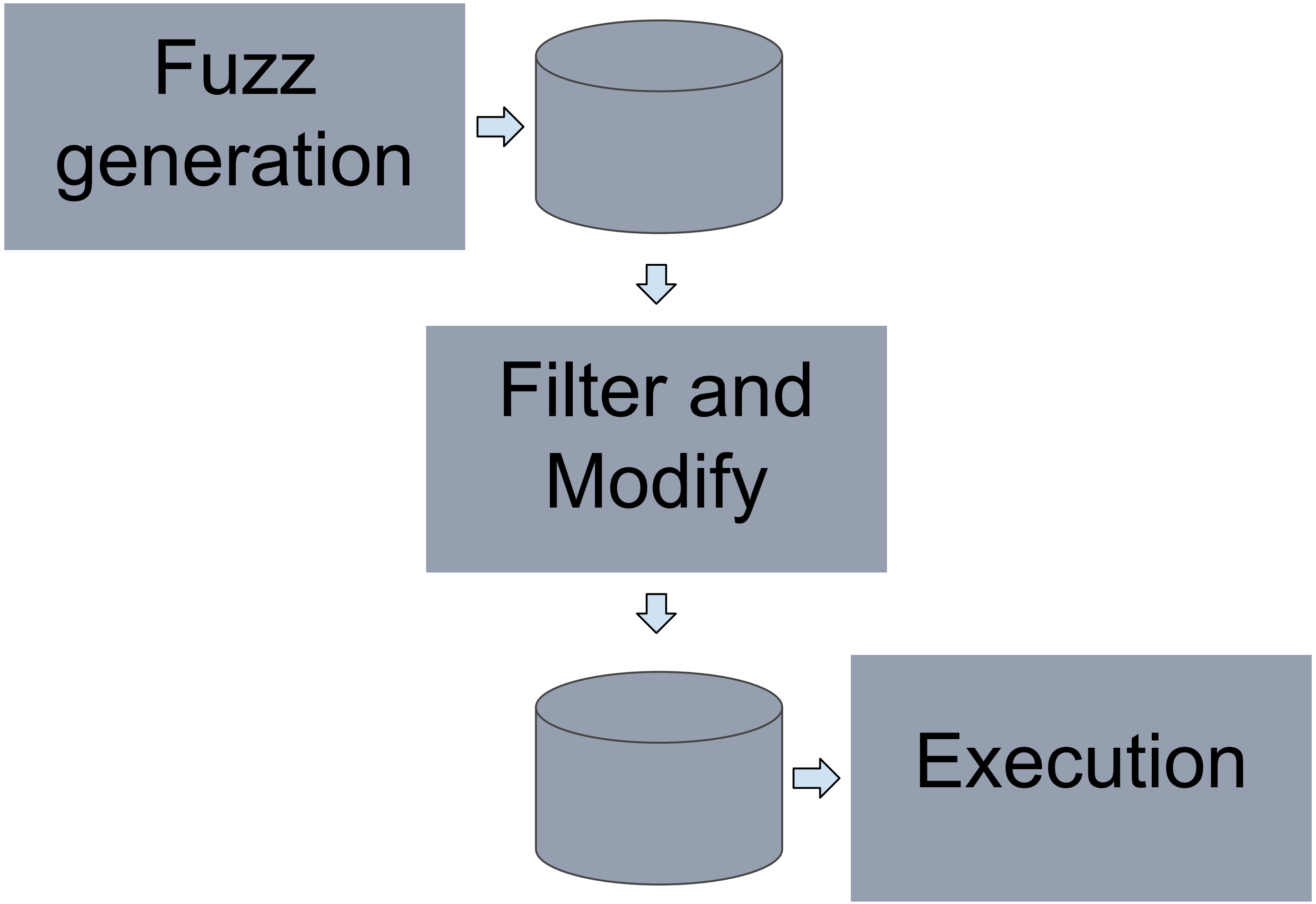
- execution engine takes operations/operands as input data
- operation/operand list is generated independently by another tool.

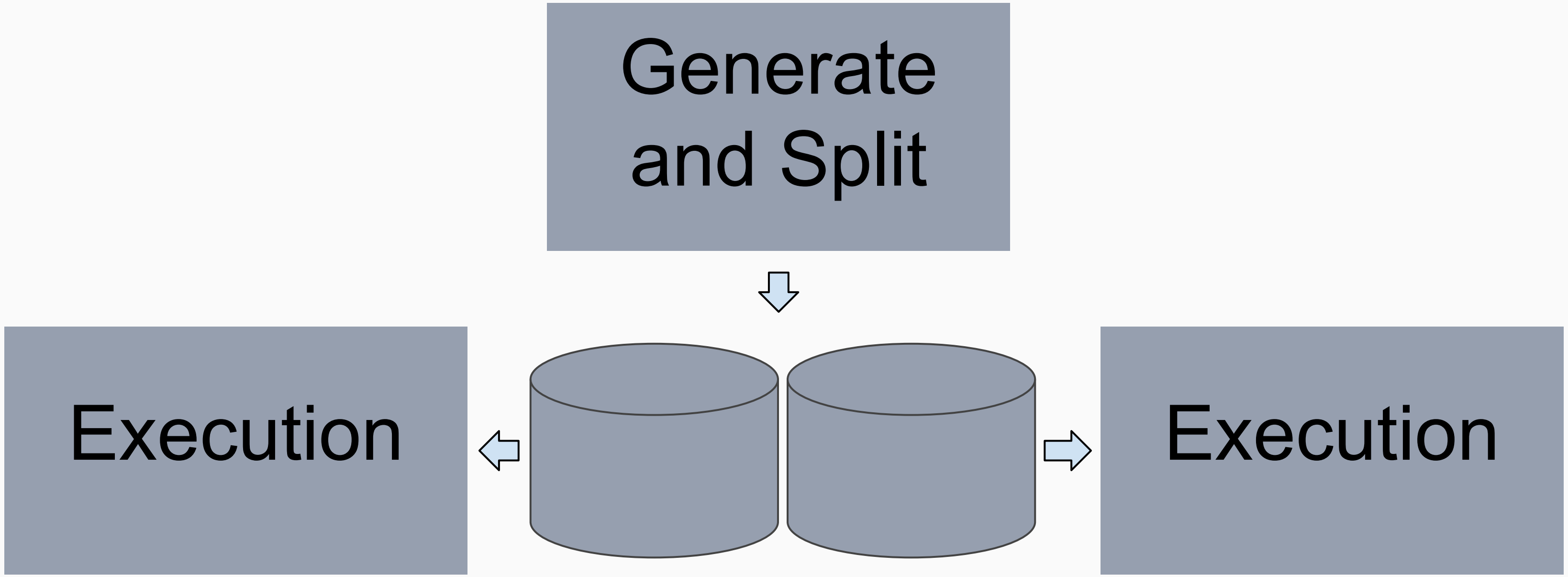
Frameworks -- Third try -- generation and execution are uncoupled



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


Generate and Split
Execution
Execution

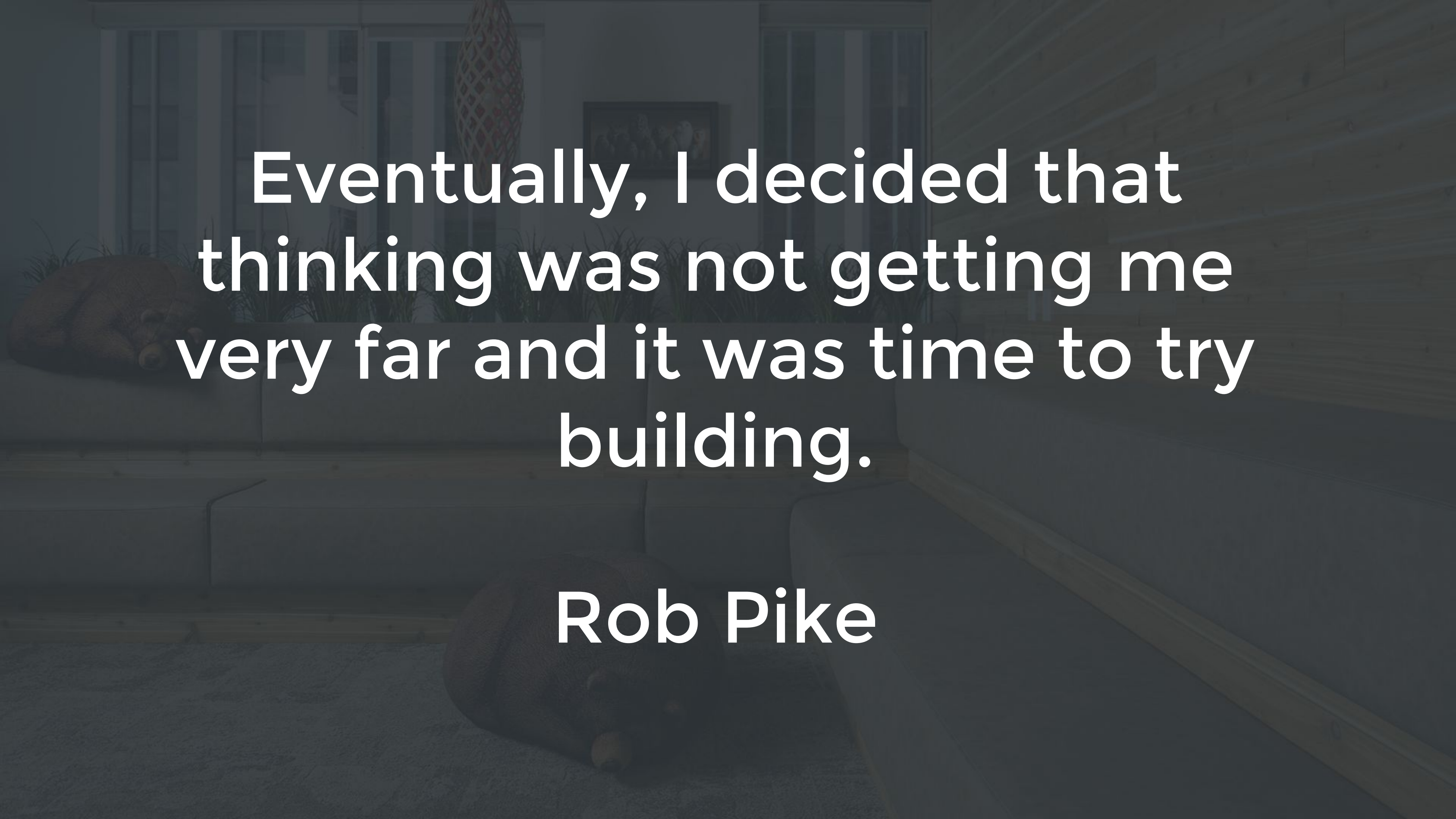
Have a dataset competition

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 one 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



Eventually, I decided that thinking was not getting me very far and it was time to try building.

Rob Pike

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.

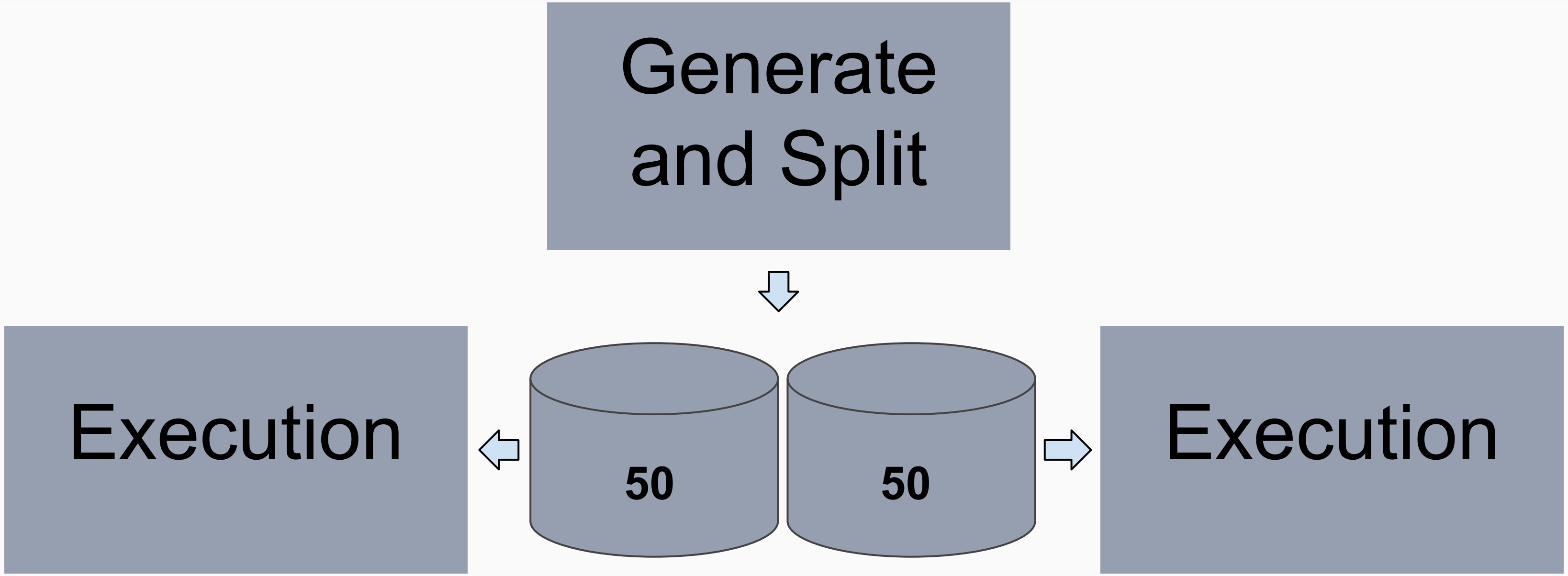
"If debugging is the process of removing bugs, then programming must be the process of putting them in." -- Dijkstra

Prototype: just try to produce a panic

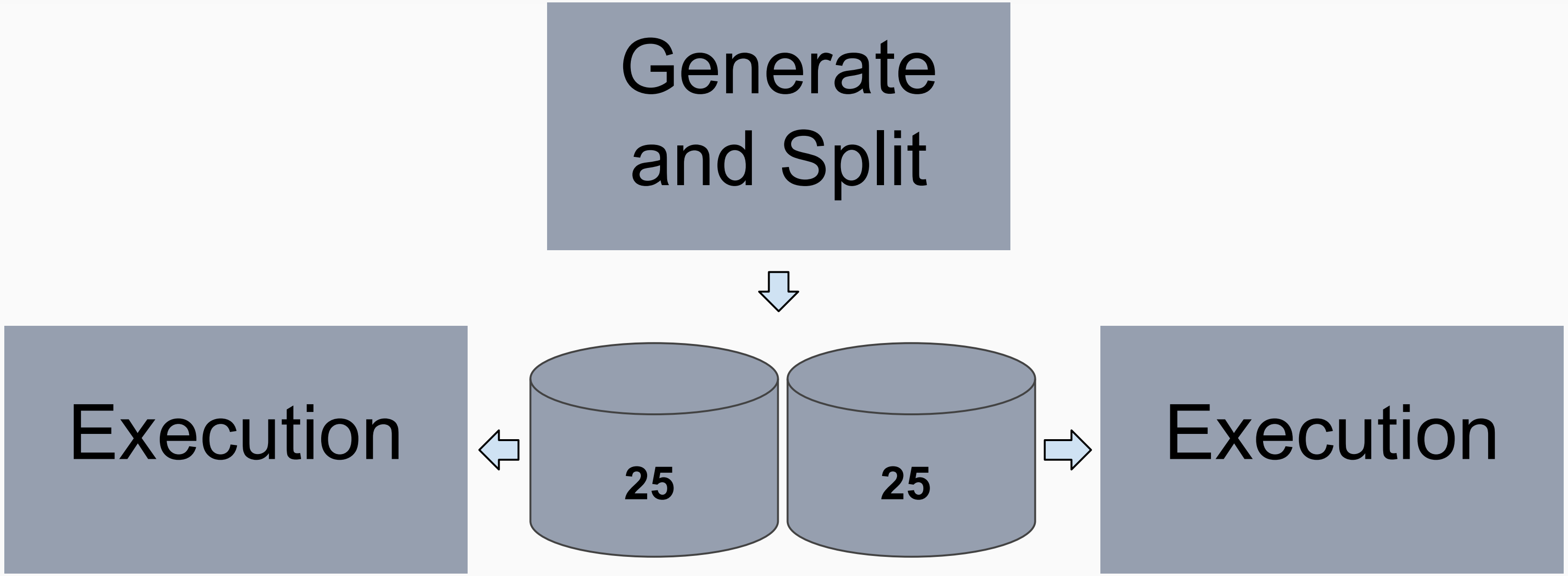
Now add data format support to this execution engine.

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

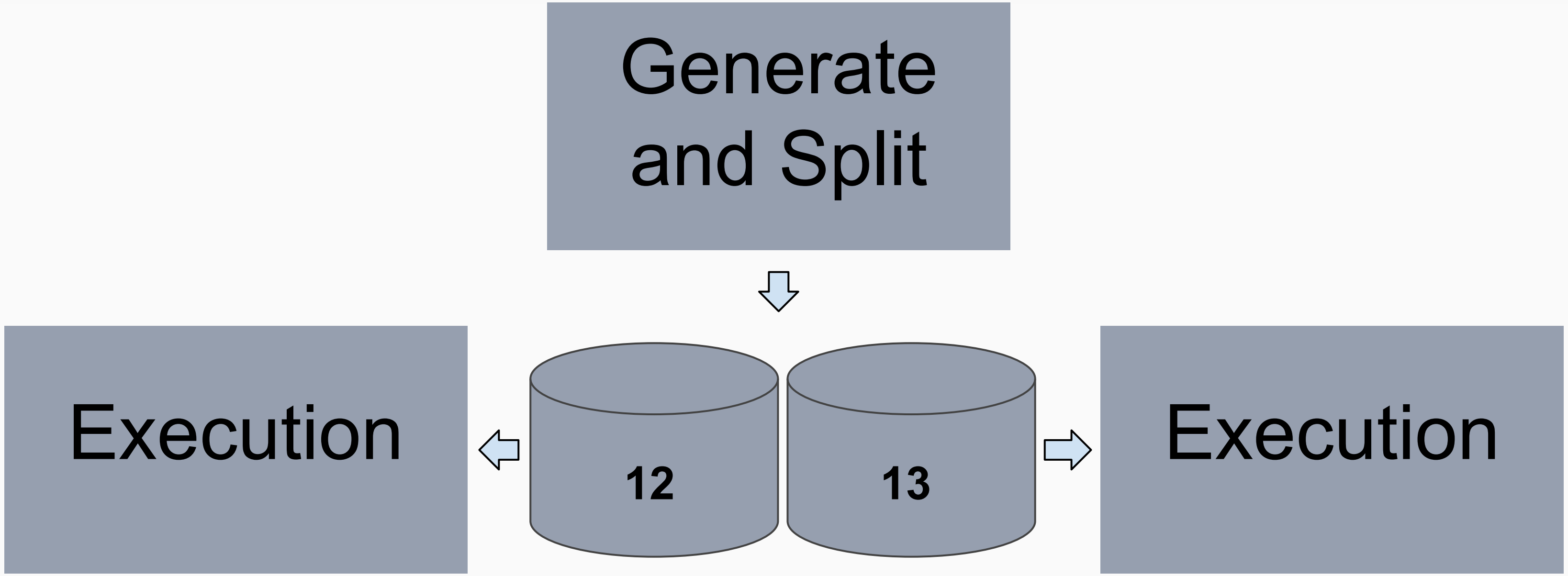
Competitions: Start with 100 operations



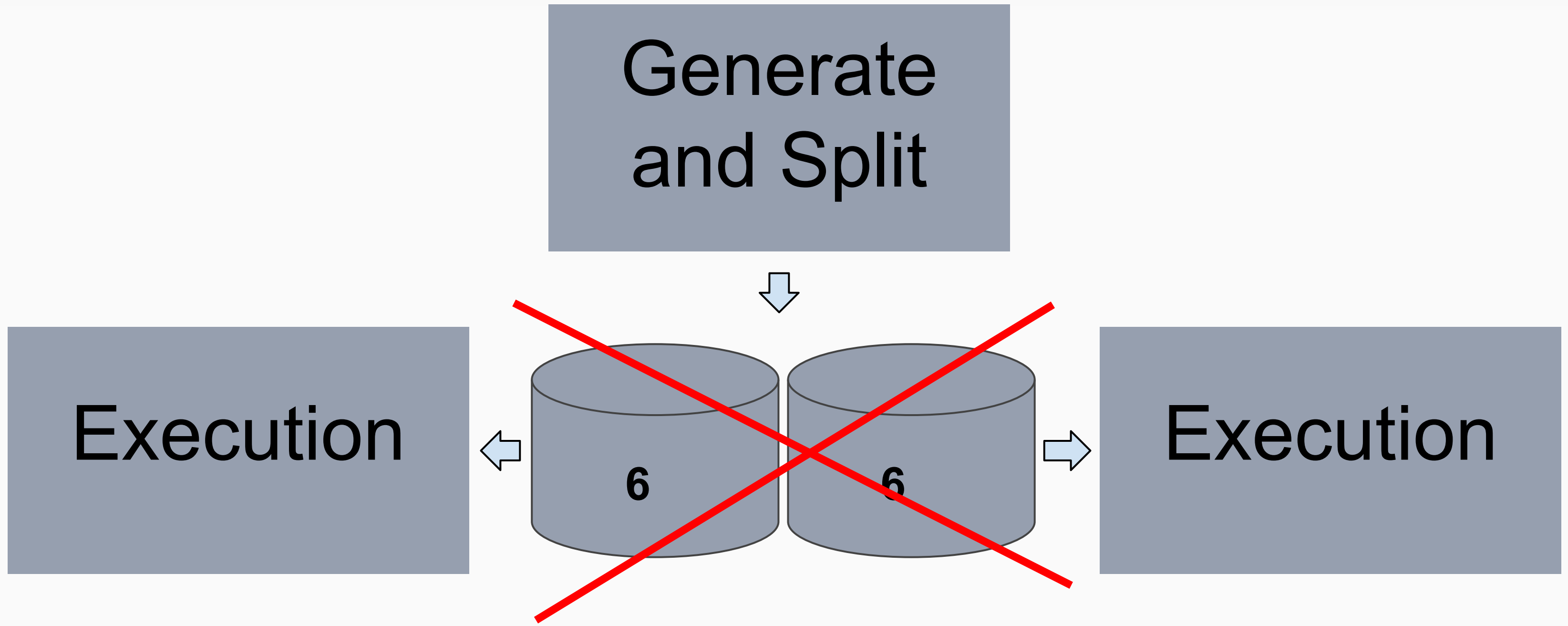
Competitions: Round Two



Competitions: Round Three



I was able to reduce from 100 operations to 12 operations

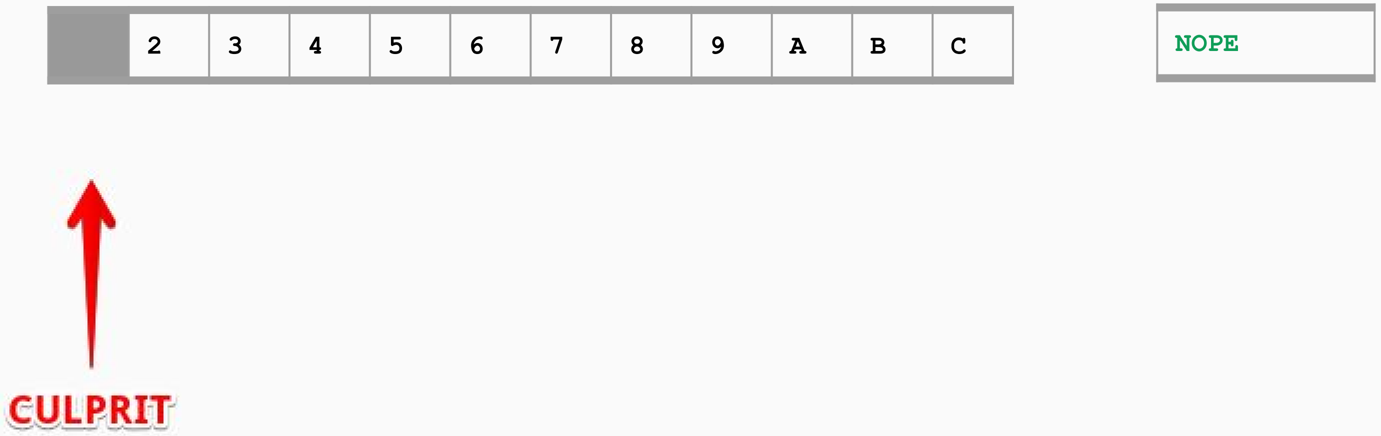


Culprit reduction - operation competition

There are 12 operations remaining -- can we still reduce the operations?

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



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

NOP

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 dataset: 4 operations/30 seconds

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

NOPE
NOPE
NOPE
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

Coda

shopify

Take away:

- Cross-training -- different job ladders can learn from each other

Take away:

- Cross-training -- different job ladders can learn from each other
- Treating testing as a software problem works too

Take away:

- Cross-training -- different job ladders can learn from each other
- Treating testing as a software problem works too
- Building better tools is difficult

Questions?

http://fuzz.ba23.org/

shopify