QCon London 2017

Property-based testing in practice

Alex Chan
alexwlchan.net/qcon17
8th March 2017

$ whoami

- Alex Chan (@alexwlchan)
- Software developer at the Wellcome Trust
- Python open-source developer:
  - python-hyper (HTTP/2)
  - PyOpenSSL
  - Hypothesis

You want to write correct software

You've never written correct software

NASA


NASA/Joel Kowsky

We have to make it cheaper to write correct software

What is property-based testing?

Anecdote-based testing

1) Write down some example inputs
2) Write down the expected outputs
3) Run the code – check they match

Property-based testing

1) Describe the input
2) Describe the *properties* of the output
3) Have the computer try lots of random examples – check they don't fail

Property-based testing

```
@given(lists(integers()))
def test_sorting_list_of_integers(xs):
    res = sorted(xs)
    assert isinstance(res, list)
    assert Counter(res) == Counter(xs)
    assert all(x <= y
               for x, y in zip(res, res[1:]))
```

Choosing a library

Python	Hypothesis
Haskell	QuickCheck
Scala	ScalaCheck
Java	JUnit-QuickCheck, QuickTheories
JavaScript	jsverify
PHP	Eris, PhpQuickCheck

http://hypothesis.works/articles/quickcheck-in-every-language

Choosing a library

C · C++ · C# · Chicken Scheme · Clojure
Common Lisp · D · Elm · Erlang · F# · Factor
Go · Io · Java · JavaScript · Julia · Logtalk
Lua · Node.js · Objective-C · OCaml · Perl
Prolog · PHP · Python · R · Racket · Ruby
Rust · Scala · Scheme · Smalltalk · Swift

https://en.wikipedia.org/wiki/QuickCheck

Testing patterns

Fuzzing, part 1

- You know what inputs your code expects – does it handle them correctly?
- Easy way to get started
- Good for:
  - Any non-trivial function

Fuzzing, part 1

```
try:
    my_function(*args, **kwargs)
except KnownException:
    pass
```

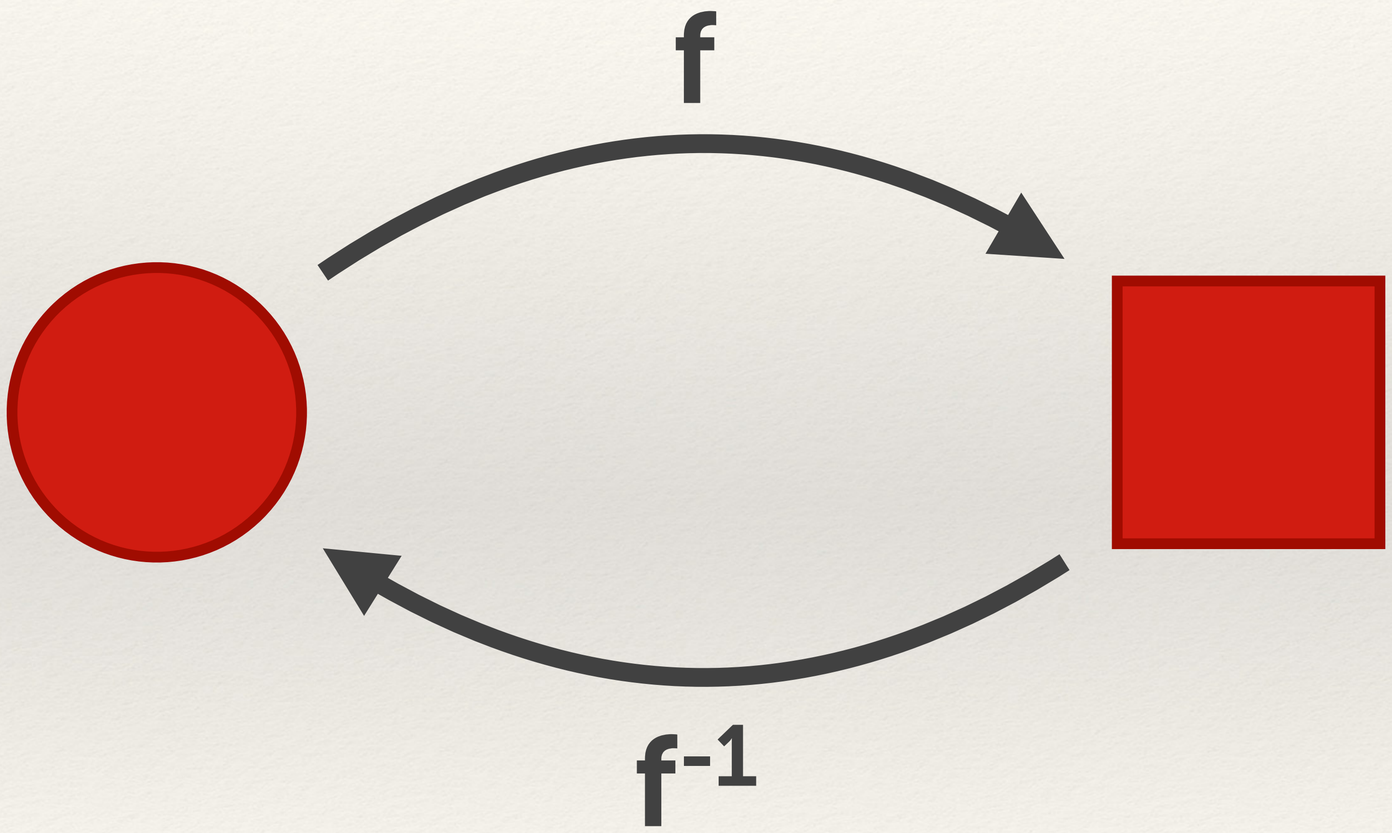
- Your function should never crash
- Your function should return the right type
- Your function should return a sensible value

Fuzzing, part 1

```
GET https://api.example.net/items?id={id}
```

- Expected HTTP return codes: 200, 4xx
- Response should be valid JSON
- Response should have the right schema

Round-trip/inverses



Round-trip/inverses

- Look for functions which are mutual inverses
- Applying both functions should be a no-op
- Good for:
  - Serialisation/deserialisation
  - Encryption/decryption
  - Read/write

Round-trip/inverses

```
from mercurial.encoding import *

@given(binary())
def test_decode_inverts_encode(s):
    assert fromutf8b(toutf8b(s)) == s
```

```
Falsifying example: s = '\xc2\xc2\x80'
```

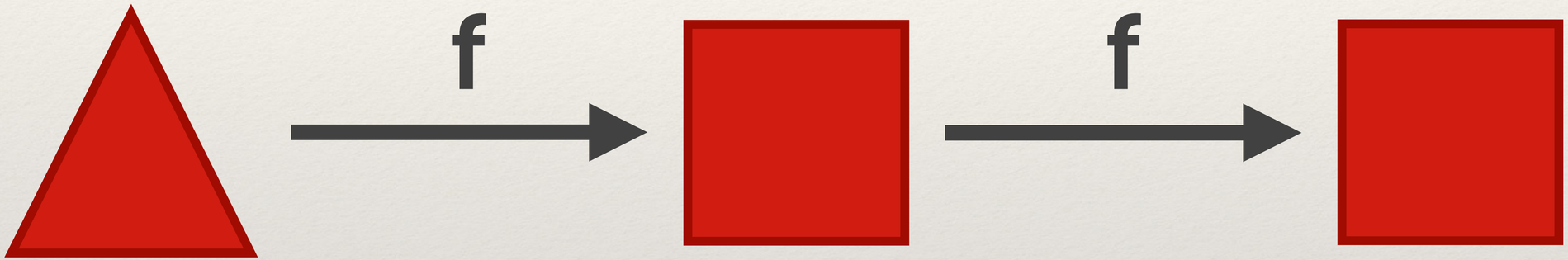
Round-trip/inverses

```
from dateutil.parser import parse

@given(datetimes())
def test_parsing_iso8601_dates(d):
    assert parse(str(d)) == d
```

```
Falsifying example:
d = datetime.datetime(4, 4, 1, 0, 0)
```

Idempotent functions



Idempotent functions

- Look for functions which are *idempotent*
- Applying the function to its output should be a no-op
- Good for:
  - Cleaning/fixing data
  - Normalisation
  - Escaping

Idempotent functions

```
from unicodedata import normalize


@given(text())
def test_normalizing_is_idempotent(string):
    result = normalize('NFC', string)
    assert result == normalize('NFC', result)
```

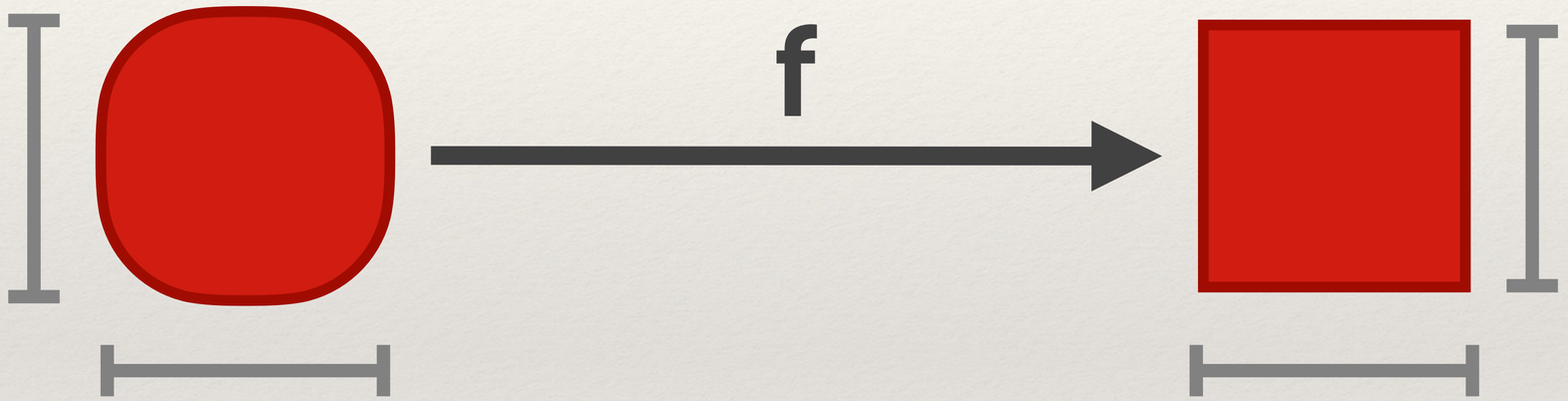
Idempotent functions

```
PUT https://api.example.net/items
{item_data}

GET https://api.example.net/items/count
```

- PUT'ing an item increments the item count
- PUT'ing the same item twice doesn't

Invariant properties



Invariant properties

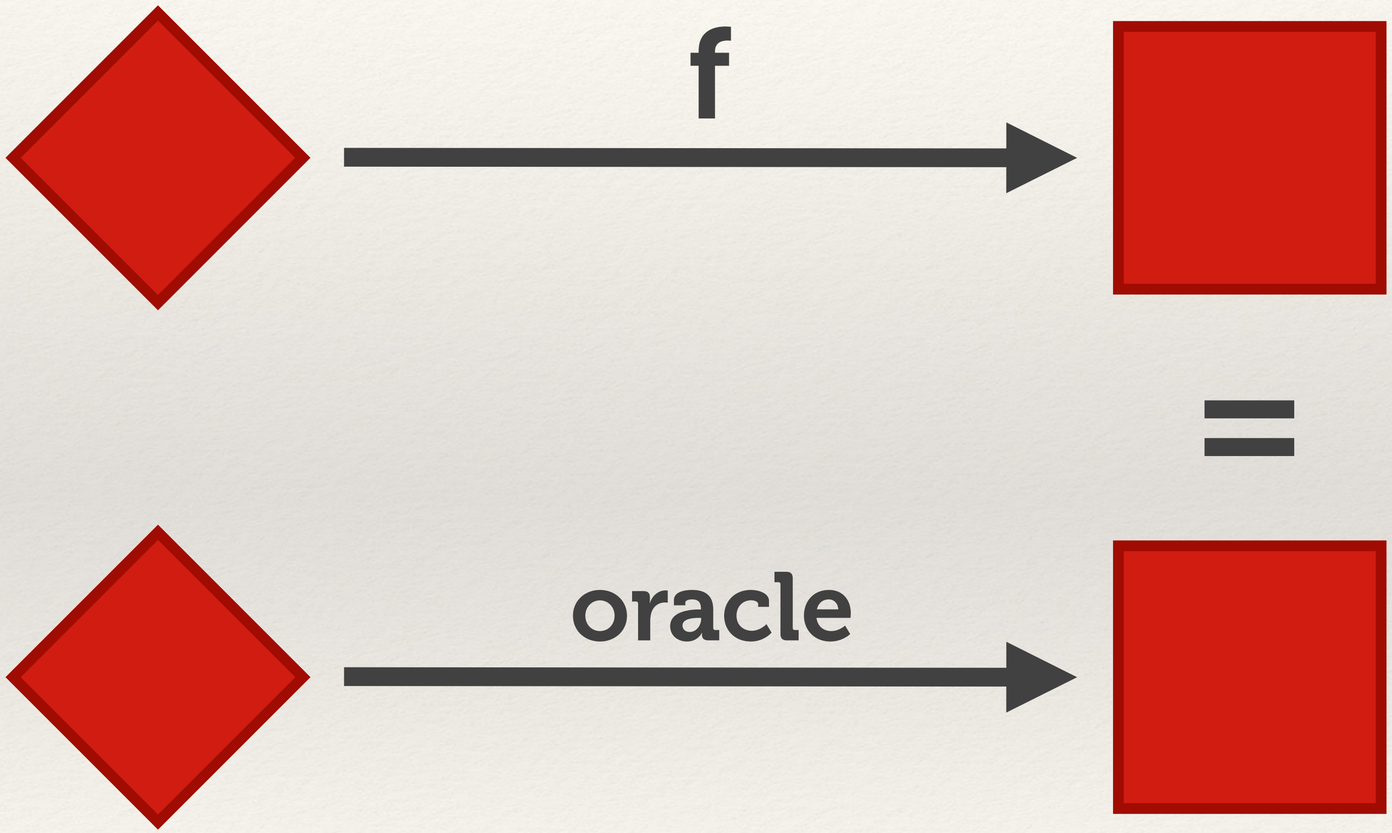
- Look for properties that don't change when you run your code
- Measuring them before and after should give the same result
- Good for:
  - Transformations
  - Anything where the result is reflected back

Invariant properties

```python
@given(text())
def test_lowercasing_preserves_cases(xs):
    assert len(xs.lower()) == len(xs)
```

```
Falsifying example:
xs = 'İ'
```

Test oracle



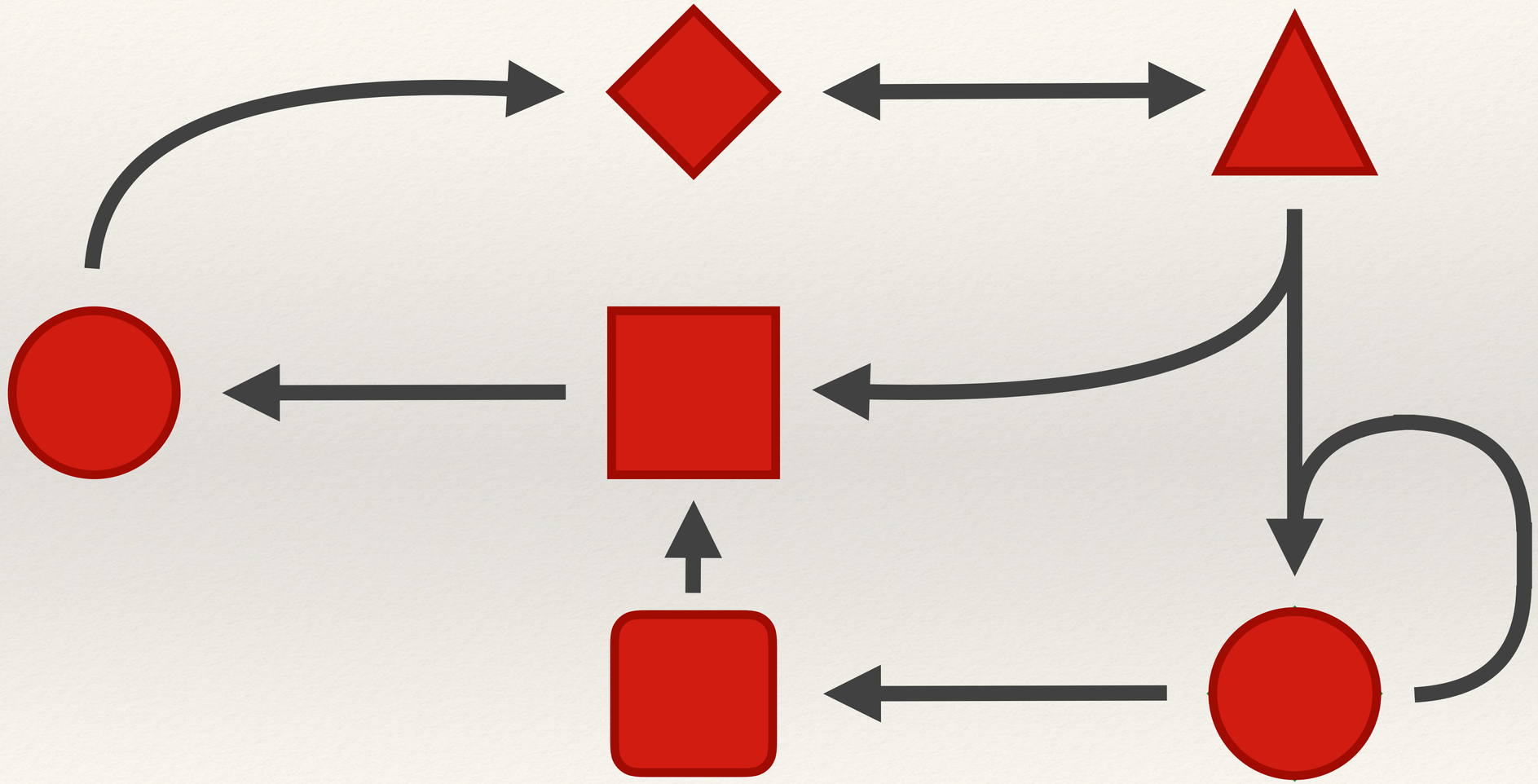
Test oracle

- Look for an alternative implementation (your oracle)
- Your code should always match the oracle
- Good for:
  - Refactoring legacy code
  - Mocking/emulating
  - Complicated code with a simple alternative

Testing patterns

Advanced techniques

Stateful testing



Stateful testing

1) Describe the possible states
2) Describe what actions can take place in each state
3) Describe how to tell if the state is correct
4) Have the computer try lots of random actions – look for a breaking combination

Stateful testing

- Testing a priority queue/binary heap
  - Create a new heap
  - Check if the heap is empty
  - Push a value/pop the first value
  - Merge two heaps together

```
def heap_new():
    return []

def is_heap_empty(heap):
    return not heap

def heap_push(heap, value):
    heap.append(value)
    idx = len(heap) - 1
    while idx > 0:
        parent = (idx - 1) // 2
        if heap[parent] > heap[idx]:
            heap[parent], heap[idx] = heap[idx], heap[parent]
            idx = parent
        else:
            break

def heap_pop(heap):
    return heap.pop(0)
```

```python
from hypothesis.stateful import *

class HeapMachine(RuleBasedStateMachine):

    def __init__(self):
        super(HeapMachine, self).__init__()
        self.heap = heap_new()

    @rule(value=integers())
    def push(self, value):
        heap_push(self.heap, value)

    @rule()
    @precondition(lambda self: self.heap)
    def pop(self):
        correct = min(self.heap)
        result = heap_pop(self.heap)
        assert correct == result
```

```
$ python -m unittest test_heap1.py
Step #1: push(value=0)
Step #2: push(value=1)
Step #3: push(value=0)
Step #4: pop()
Step #5: pop()
F
======================================================================
FAIL: runTest (hypothesis.stateful.HeapMachine.TestCase)
----------------------------------------------------------------------
```

```
def heap_merge(heap1, heap2):
    heap1, heap2 = sorted((heap1, heap2))
    return heap1 + heap2
```

```
class HeapMachine(RuleBasedStateMachine):
    Heaps = Bundle('heaps')

    @rule(target=Heaps)
    def new_heap(self):
        return heap_new()

    @rule(heap=Heaps, value=integers())
    def push(self, heap, value):
        heap_push(heap, value)

    @rule(heap=Heaps.filter(bool))
    def pop(self, heap):
        correct = min(heap)
        result = heap_pop(heap)
        assert correct == result

    @rule(target=Heaps, heap1=Heaps, heap2=Heaps)
    def merge(self, heap1, heap2):
        return heap_merge(heap1, heap2)
```

```
$ python -m unittest test_y.py
Step #1: v1 = newheap()
Step #2: push(heap=v1, value=0)
Step #3: push(heap=v1, value=1)
Step #4: push(heap=v1, value=1)
Step #5: v2 = merge(y=v1, heap1=v1)
Step #6: pop(heap=v2)
Step #7: pop(heap=v2)
F
======================================================================
FAIL: runTest (hypothesis.stateful.HeapMachine.TestCase)
----------------------------------------------------------------------
```

```
def heap_merge(heap1, heap2):
    result = []
    i = 0
    j = 0
    while i < len(heap1) and j < len(heap2):
        if heap1[i] <= heap2[j]:
            result.append(heap1[i])
            i += 1
        else:
            result.append(heap2[j])
            j += 1
    result.extend(heap1[i:])
    result.extend(heap2[j:])
    return result
```

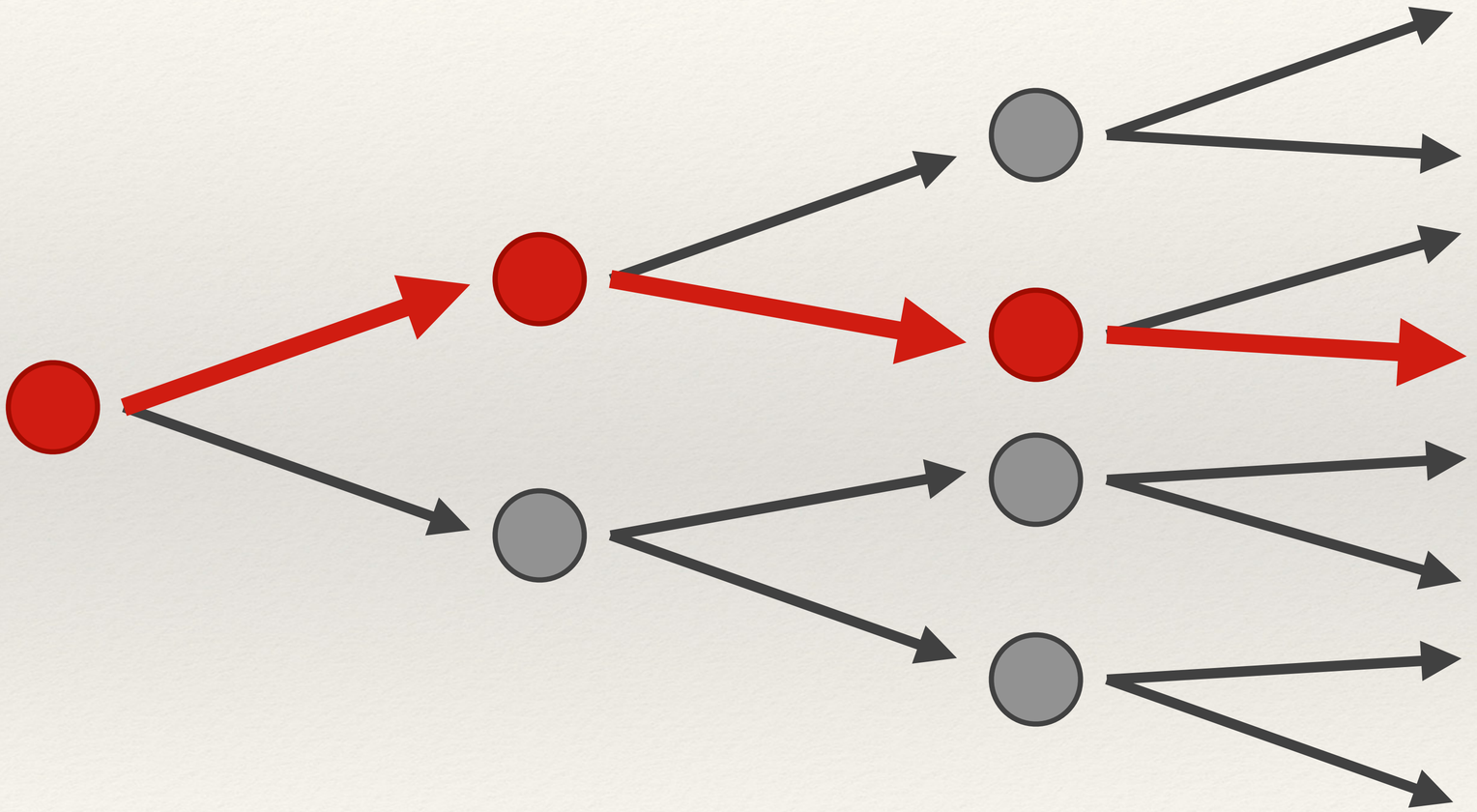
```
Step #1: v1 = newheap()
Step #2: push(heap=v1, value=0)
Step #3: v2 = merge(heap1=v1, heap2=v1)
Step #4: v3 = merge(heap1=v2, heap2=v2)
Step #5: push(heap=v3, value=-1)
Step #6: v4 = merge(heap1=v1, heap2=v2)
Step #7: pop(heap=v4)
Step #8: push(heap=v3, value=-1)
Step #9: v5 = merge(heap1=v1, heap2=v2)
Step #10: v6 = merge(heap1=v5, heap2=v4)
Step #11: v7 = merge(heap1=v6, heap2=v3)
Step #12: pop(heap=v7)
Step #13: pop(heap=v7)

>>> v7
[-1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0]
```

Stateful testing in practice

- Hypothesis itself – the examples database
- State of a Mercurial repo
- An HTTP/2 Priority tree
- Interacting HTTP/2 stacks

Fuzzing, part 2



Fuzzing, part 2

- Random fuzzing only scratches the surface – what if we want to go deeper?
- We need to get smarter!


Mia Munroe

Enter AFL

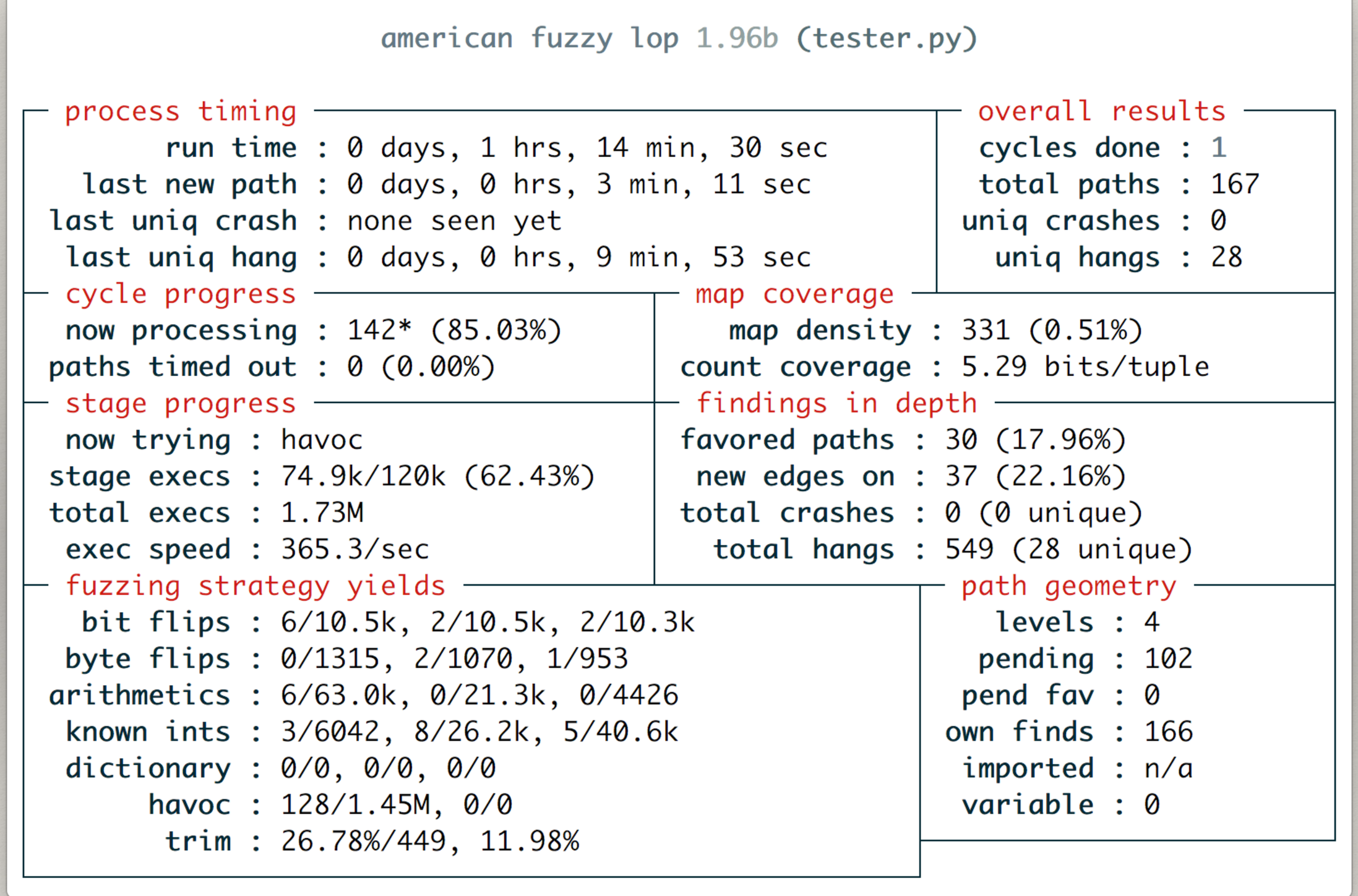
- AFL uses tracing to see different paths through our code. It can “learn” the data under test.
- Good for:
  - File formats
  - Parsers
  - Anywhere with untrusted input

Enter AFL

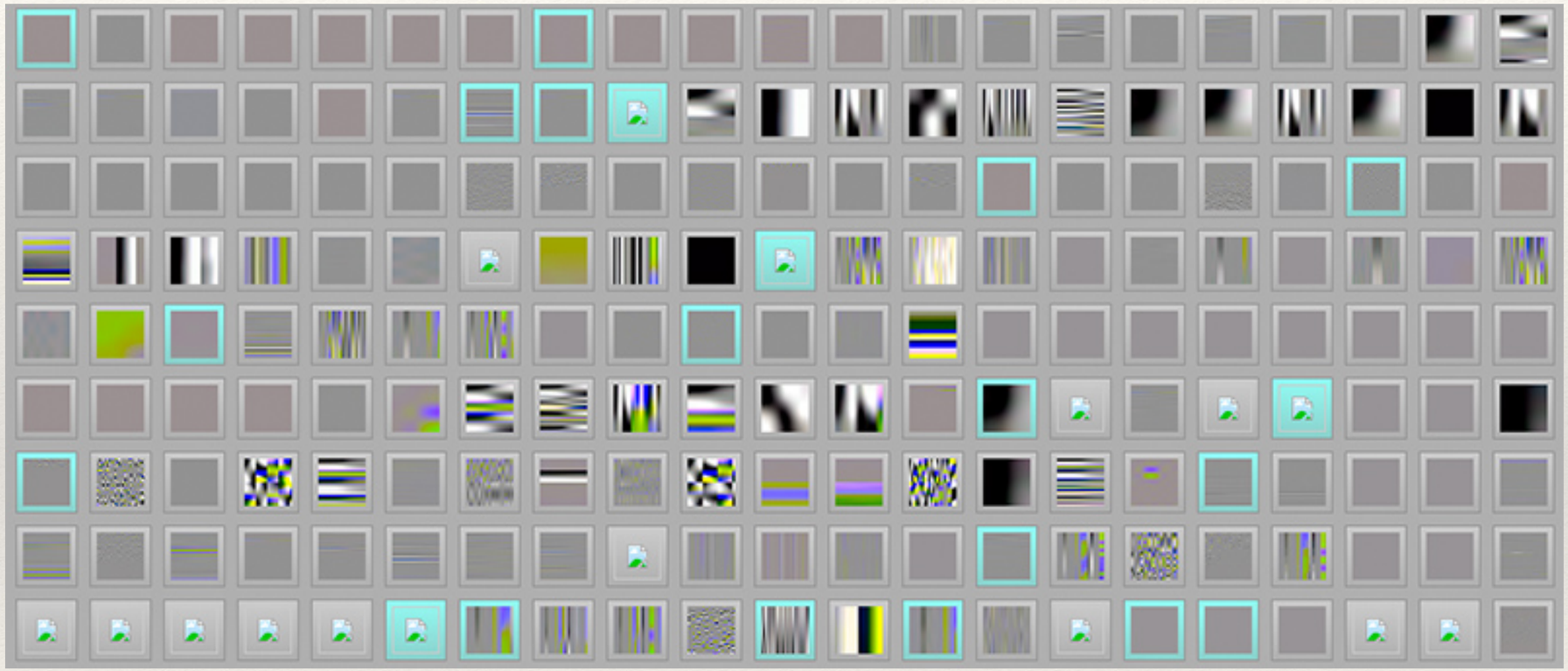
```
import afl, hpack, sys

afl.init()

d = hpack.Decoder()
try:
    d.decode(sys.stdin.buffer.read())
except hpack.HPACKError:
    pass
```


american fuzzy lop 1.96b (tester.py)
process timing
run time : 0 days, 1 hrs, 14 min, 30 sec
last new path : 0 days, 0 hrs, 3 min, 11 sec
last uniq crash : none seen yet
last uniq hang : 0 days, 0 hrs, 9 min, 53 sec
overall results
cycles done : 1
total paths : 167
uniq crashes : 0
uniq hangs : 28
cycle progress
now processing : 142* (85.03%)
paths timed out : 0 (0.00%)
map coverage
map density : 331 (0.51%)
count coverage : 5.29 bits/tuple
stage progress
now trying : havoc
stage execs : 74.9k/120k (62.43%)
total execs : 1.73M
exec speed : 365.3/sec
findings in depth
favored paths : 30 (17.96%)
new edges on : 37 (22.16%)
total crashes : 0 (0 unique)
total hangs : 549 (28 unique)
fuzzing strategy yields
bit flips : 6/10.5k, 2/10.5k, 2/10.3k
byte flips : 0/1315, 2/1070, 1/953
arithmetics : 6/63.0k, 0/21.3k, 0/4426
known ints : 3/6042, 8/26.2k, 5/40.6k
dictionary : 0/0, 0/0, 0/0
havoc : 128/1.45M, 0/0
trim : 26.78%/449, 11.98%
path geometry
levels : 4
pending : 102
pend fav : 0
own finds : 166
imported : n/a
variable : 0

Enter AFL



Pulling JPEGs out of thin air, Michael Zalweski

Advanced techniques

Advanced techniques

Wrap up

- Property-based testing is a very powerful way to test your code
- Ensure confidence, find more bugs!
- Stateful testing and AFL make it even more powerful

Property-based testing in practice

Slides and links

https://alexwlchan.net/qcon17/

Hypothesis

https://hypothesis.works/

AFL

http://lcamtuf.coredump.cx/afl/

QCon London 2017

Property-based testing in practice

Alex Chan
alexwlchan.net/qcon17
8th March 2017