What is random testing, and why do you need it?

Alperen Keles
Ph.D. Candidate at University of Maryland

Engineering

Software
“Engineering”

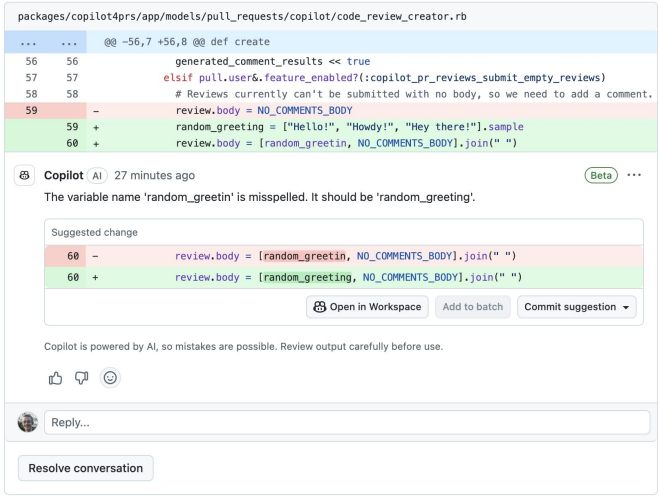
Structural Engineering

Software “Engineering”

Structural Engineering

Visual Inspection

Software “Engineering”



Code Review

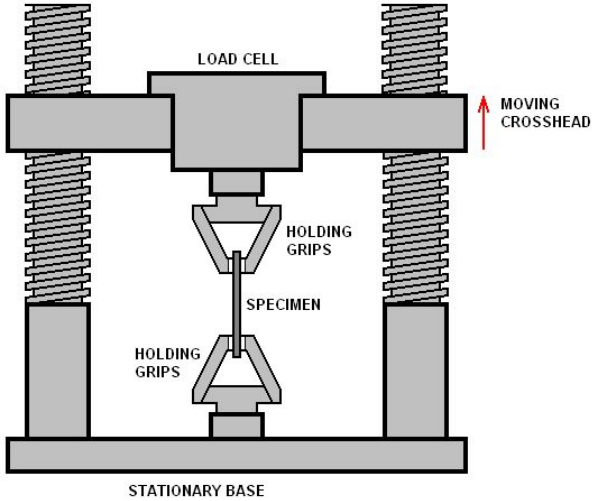
Structural Engineering

Visual Inspection

Software “Engineering”

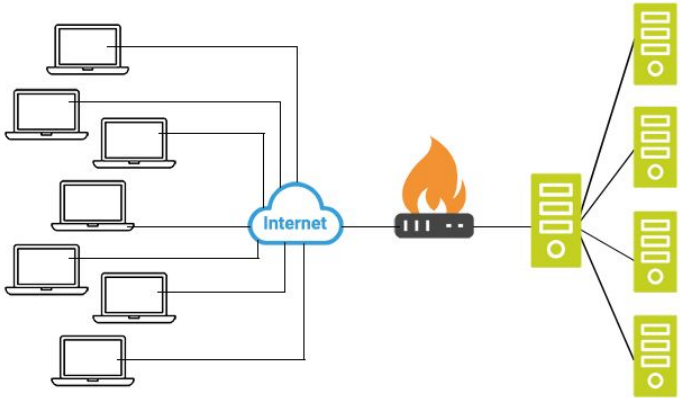
Quality Assurance

Materials Engineering



Tensile Testing

Software “Engineering”

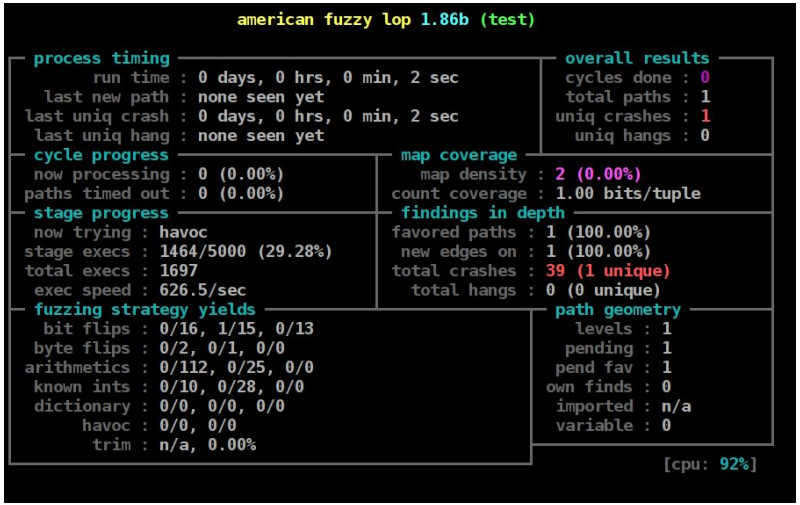


Stress Testing

Automotive Engineering

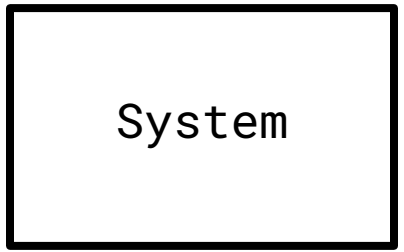
Crash Testing

Software “Engineering”

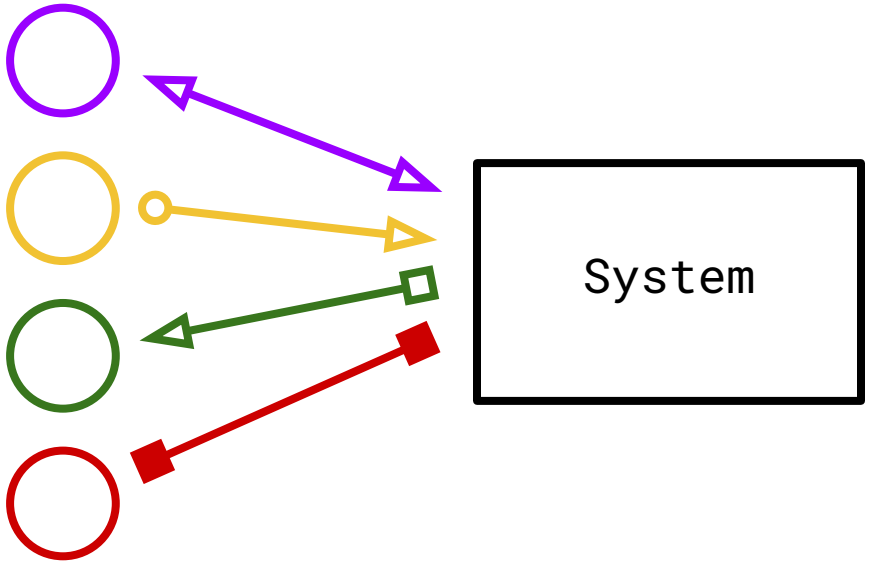


Fuzz Testing

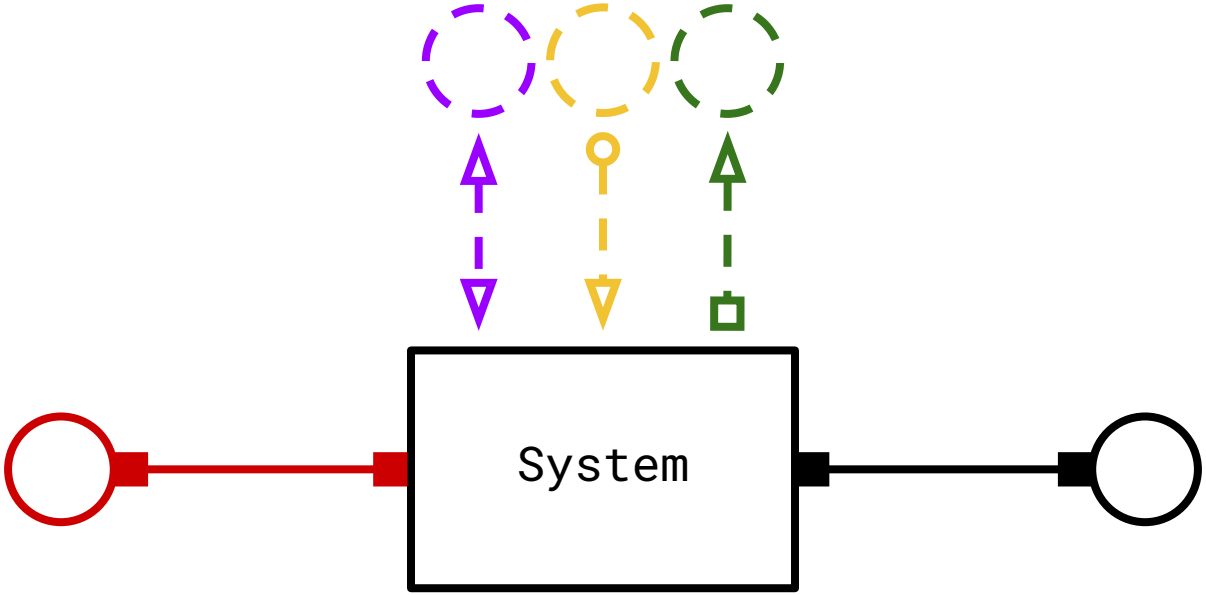
Engineering



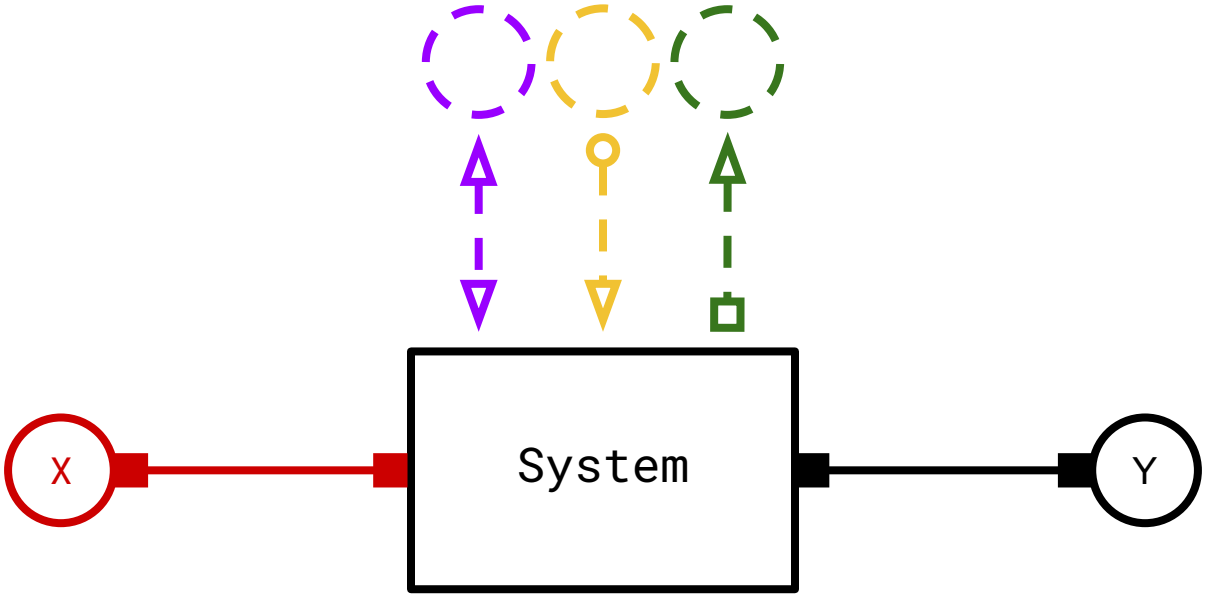
Engineering



Testing

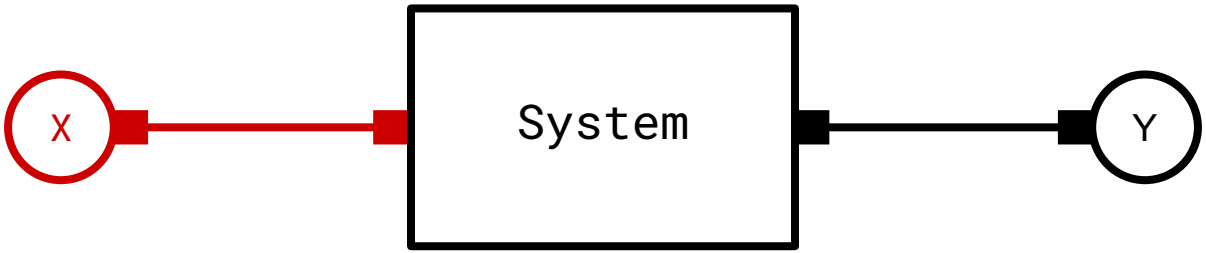


Example-Based Testing

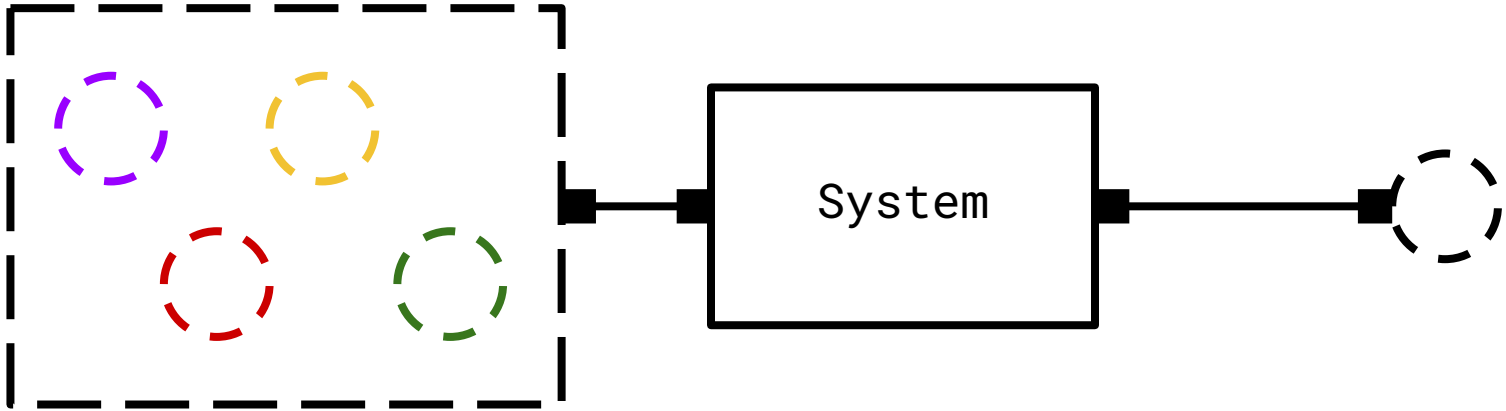


Example-Based Testing

The idea that you can use precise input-output pairs for testing large scale system behavior is unique to software engineering.



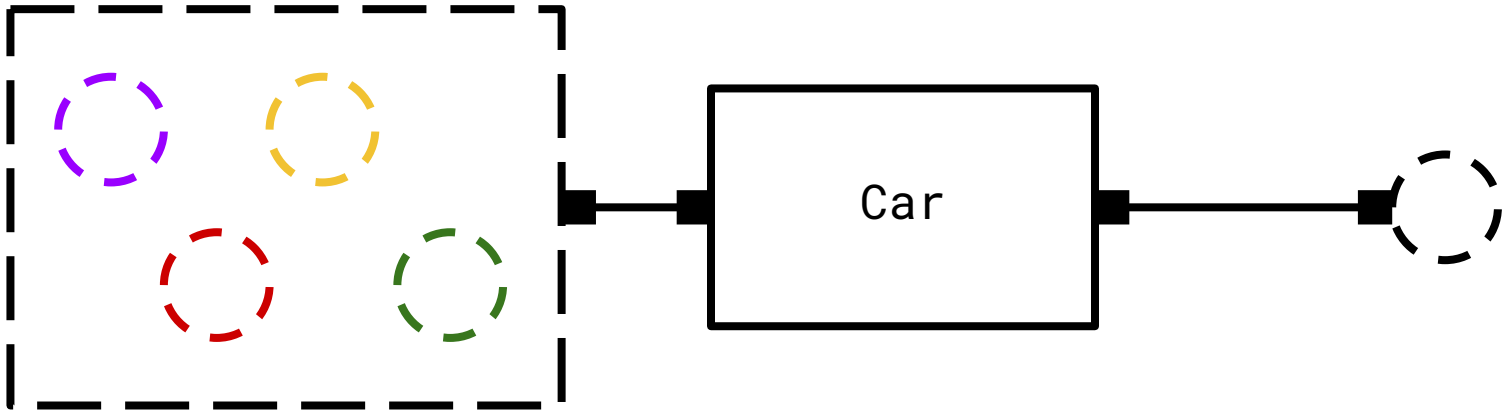
Testing



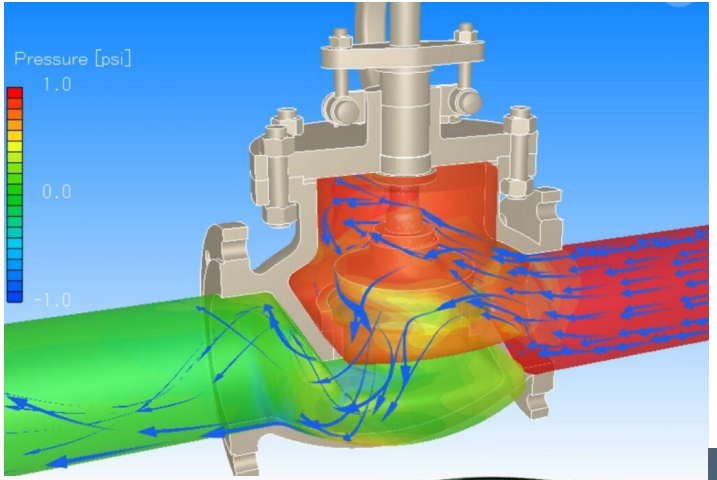
Crash Testing

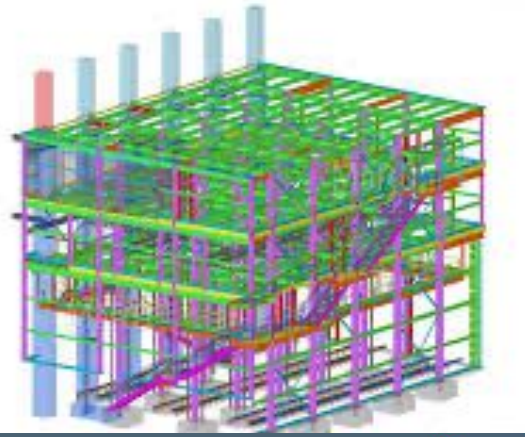
- Drive a car at some speed
- Place crash dummies in seats

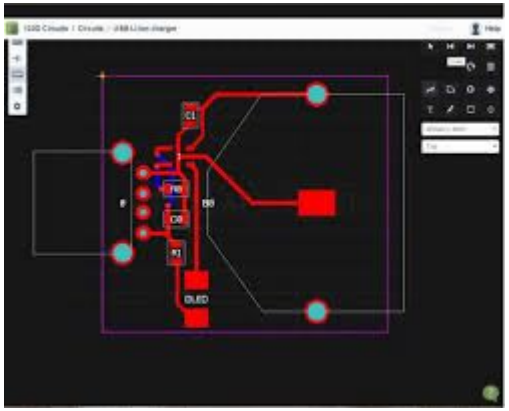
- Crash dummy is safe
- Car doesn’t explode

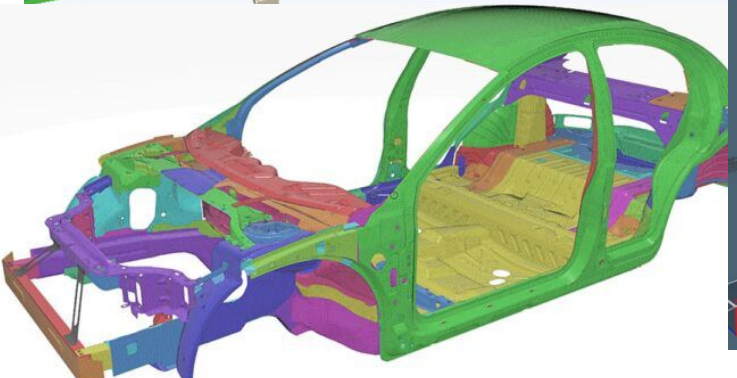


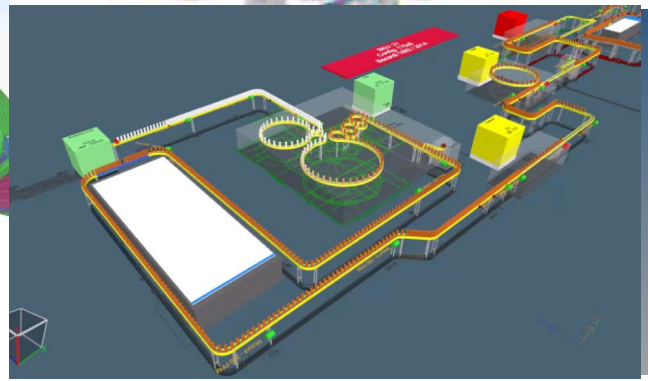
Simulation and Modelling

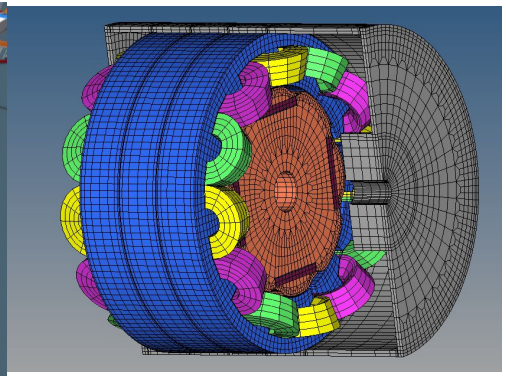












Economics of Software Systems

“Selling SaaS software is more similar to renting a car factory than selling the cars it produces.”

- Very high leverage of distribution
- Lower cost of faults
- Very exposed to the users

...

- **There are still costs to faults.**

Software Testing

Software Testing

Types of Testing

Unit Testing

Integration Tests

E2E Tests

Software Testing

Scopes Types **of Testing**

Unit Testing

Integration Tests

E2E Tests

Software Testing

Scopes Types of Testing

Unit Testing

Integration Tests

E2E Tests

Scopes of Testing

Function: f(x, y) = z

Module : M(A, B, C)

System : [S], SxA → S

Software Testing

Scopes of Testing

Function: f(x, y) = z

Module : M(A, B, C)

System : [S], SxA → S

Software Testing

Scopes of Testing

Function: f(x, y) = z

Module : M(A, B, C)

System : [S], SxA → S

Purposes of Testing

Functional Correctness

Performance

Security

Software Testing

Scopes of Testing

Function: f(x, y) = z

Module : M(A, B, C)

System : [S], SxA → S

Purposes of Testing

Functional Correctness

Performance

Security

Specifications of Testing

Software Testing

Scopes of Testing

Function: f(x, y) = z

Module : M(A, B, C)

System : [S], SxA → S

Purposes of Testing

Functional Correctness

Performance

Security

Specifications of Testing

Examples: f(x) = y, t(f(x)) < 5ms, sql(q(x)) = Error

Software Testing

Scopes of Testing

Function: f(x, y) = z

Module : M(A, B, C)

System : [S], SxA → S

Purposes of Testing

Functional Correctness

Performance

Security

Specifications of Testing

Examples: f(x) = y, t(f(x)) < 5ms, sql(q(x)) = Error

Tinkering: f(x) = y, f(x + 1) = f(x) + 1…

Software Testing

Scopes of Testing

Function: f(x, y) = z

Module : M(A, B, C)

System : [S], SxA → S

Purposes of Testing

Functional Correctness

Performance

Security

Specifications of Testing

Examples: f(x) = y, t(f(x)) < 5ms, sql(q(x)) = Error

Tinkering: f(x) = y, f(x + 1) = f(x) + 1…

Properties: ∀x. p(f(x)). Never crash. Never drop a message.

Testing Properties

Testing Properties

Examples

∀x. p(f(x))

```
assert(p(f(1))
assert(p(f(“a”))
assert(p(f({}))
assert(p(f(0))
assert(p(f(-42))
```

Testing Properties

Examples

```
∀x. p(f(x))

assert(p(f(1))
assert(p(f(“a”))
assert(p(f({}))
assert(p(f(0))
assert(p(f(-42))
```

Exhaustive

```
∀x: int. p(f(x))

assert(p(f(0))
assert(p(f(1))
assert(p(f(-1))
assert(p(f(2))
...
assert(p(f(INT_MAX))
assert(p(f(INT_MIN))
```

Testing Properties

Examples

```
∀x. p(f(x))

assert(p(f(1))
assert(p(f(“a”))
assert(p(f({}))
assert(p(f(0))
assert(p(f(-42))
```

Exhaustive

```
∀x: int. p(f(x))

assert(p(f(0))
assert(p(f(1))
assert(p(f(-1))
assert(p(f(2))
...
assert(p(f(INT_MAX))
assert(p(f(INT_MIN))
```

Random

```
∀x: int. p(f(x))

repeat N:
    r := random_int()
    assert(p(f(r))
```

Testing Properties*

Examples

```
∀x. p(f(x))

assert(p(f(1))
assert(p(f(“a”))
assert(p(f({}))
assert(p(f(0))
assert(p(f(-42))
```

Exhaustive

```
∀x: int. p(f(x))

assert(p(f(0))
assert(p(f(1))
assert(p(f(-1))
assert(p(f(2))
...
assert(p(f(INT_MAX))
assert(p(f(INT_MIN))
```

Random

```
∀x: int. p(f(x))

repeat N:
    r := random_int()
    assert(p(f(r))
```

* type systems,
  model checkers,
  static analysis

Property-Based Random Testing

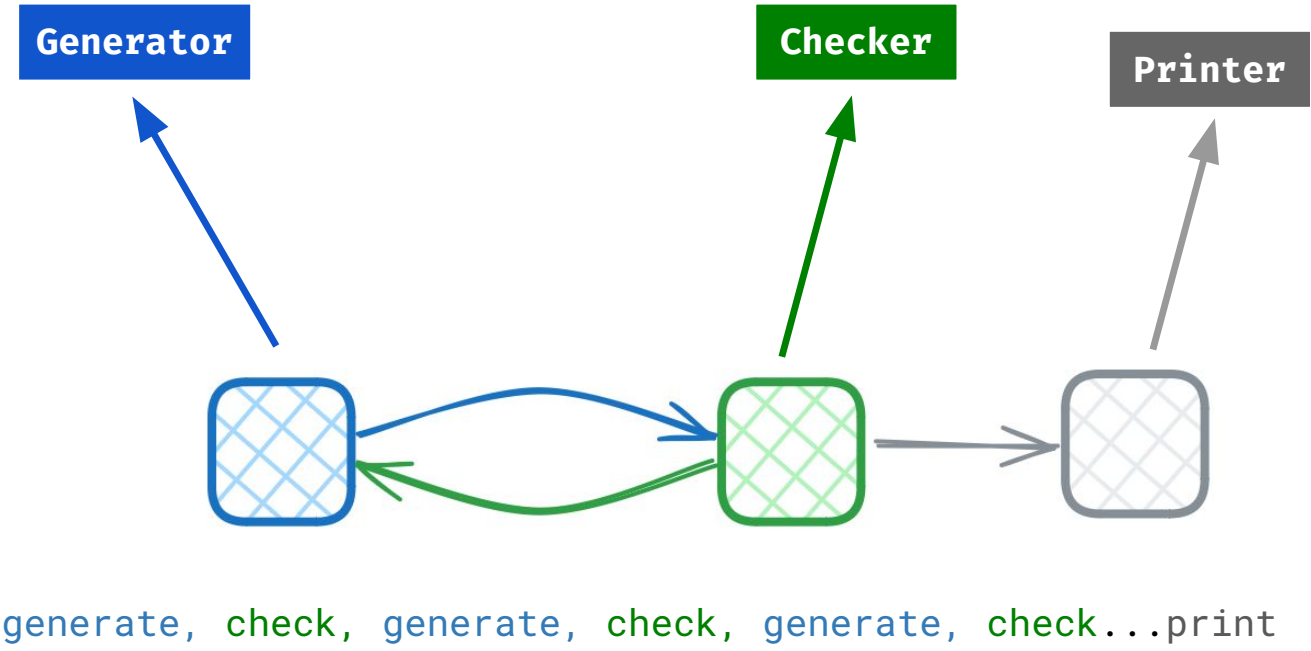
```
∀x: int. p(f(x))

repeat N:
    r := random_int()
    assert(p(f(r))
```

Property-Based Random Testing

$\forall x$: int. p(f(x))

```
repeat N:
    r := random_int()
    assert(p(f(r))
```



Property-Based Random Testing

Data Structures

Trees

CRDTs

Lock-Free

...

Property-Based Random Testing

Data Structures	**Functions**
Trees	JSON.parse(JSON.stringify(obj)) = obj
CRDTs	inline assertions (tiger style)
Lock-Free	never crash
...	metamorphic (sort(sort(l)) = (sort(l))

Property-Based Random Testing

Data Structures	**Functions**
Trees	JSON.parse(JSON.stringify(obj)) = obj
CRDTs	inline assertions (tiger style)
Lock-Free	never crash
...	metamorphic (sort(sort(l)) = (sort(l))

Systems

send M, **run** P, **ask** R, **take** A

put K V, R := **get** K, **assert** R = V

Some Practical Properties

Roundtrip Property

```
∀ j. JSON.parse(JSON.stringify(j)) == j
∀ k, v. get(put(k, v), k) == v
∀ s. decompress(compress(s)) == s
```

Idempotency

```
∀ x. f(x) == f(f(x)) == f(f(f(x))
```

Class Invariants

```
∀ s1, s2. is_utf8(s1) && is_utf8(s2) ==> is_utf8(s1 + s2)
∀ date. is_date(date) ==> is_date(date.next())
```

Fuzz Testing

Program does not crash.

Differential Testing

```
∀ x. f1(x) == f2(x)
```

Some Practical Properties

Frontend

- Page should never take more than 120ms to rerender.
- DOM Nodes should never intersect each other.
- No set of user interactions should ever lead to faults.
- Non-admin users should never acquire admin only information.
- Locale pages should not display English text.

Backend

- p95 should be less than 10ms.
- Request X should not affect resource Y.
- Caching should always speed up a repeated request.

Distributed System

- There should always be 1 leader.
- All messages should be idempotent.

Some Real World Examples

```
american fuzzy lop 0.47b (readpng)
```

- VLC
- Sqlite
- Vim
- Pure-FTPd
- Bftpd
- Tcpdump
- ProFTPd
- Gifsicle
- FFmpeg
- Glibc
- FreeRDP
- GNOME
- QEMU
- GNU coreutils
- PostgreSQL
- Node.js
- libjxl
- Perl
- zlog

- VLC
- Sqlite
- Vim
- Pure-FTPd
- Bftpd
- Tcpdump
- ProFTPd
- Gifsicle
- FFmpeg
- Glibc
- FreeRDP
- GNOME
- QEMU
- GNU coreutils
- PostgreSQL
- Node.js
- libjxl
- Perl
- zlog

- SQLite (179)
- MySQL (19)
- PostgreSQL (13)
- MariaDB (1)
- CockroachDB (56)
- TiDB (55)
- DuckDB (73)

- VLC
- Sqlite
- Vim
- Pure-FTPd
- Bftpd
- Tcpdump
- ProFTPd
- Gifsicle
- FFmpeg
- Glibc
- FreeRDP
- GNOME
- QEMU
- GNU coreutils
- PostgreSQL
- Node.js
- libjxl
- Perl
- zlog

SQLANCER

- SQLite (179)
- MySQL (19)
- PostgreSQL (13)
- MariaDB (1)
- CockroachDB (56)
- TiDB (55)
- DuckDB (73)

	GCC	LLVM
Front end	0	10
Middle end	49	75
Back end	17	74
Unclassified	13	43
Total	79	202

american fuzzy lop 0.47b (readpng)

- VLC
- Sqlite
- Vim
- Pure-FTPd
- Bftpd
- Tcpdump
- ProFTPd
- Gifsicle
- FFmpeg
- Glibc
- FreeRDP
- GNOME
- QEMU
- GNU coreutils
- PostgreSQL
- Node.js
- libjxl
- Perl
- zlog

SQLANCER

- SQLite (179)
- MySQL (19)
- PostgreSQL (13)
- MariaDB (1)
- CockroachDB (56)
- TiDB (55)
- DuckDB (73)

	GCC	LLVM
Front end	0	10
Middle end	49	75
Back end	17	74
Unclassified	13	43
Total	79	202

Geneva: **Evolving Censorship Evasion Strategies**

american fuzzy lop 0.47b (readpng)

- VLC
- Sqlite
- Vim
- Pure-FTPd
- Bftpd
- Tcpdump
- ProFTPd
- Gifsicle
- FFmpeg
- Glibc
- FreeRDP
- GNOME
- QEMU
- GNU coreutils
- PostgreSQL
- Node.js
- libjxl
- Perl
- zlog

SQLANCER

- SQLite (179)
- MySQL (19)
- PostgreSQL (13)
- MariaDB (1)
- CockroachDB (56)
- TiDB (55)
- DuckDB (73)

	GCC	LLVM
Front end	0	10
Middle end	49	75
Back end	17	74
Unclassified	13	43
Total	79	202

Geneva: **Evolving Censorship Evasion Strategies**

american fuzzy lop 0.47b (readpng)

- VLC
- Sqlite
- Vim
- Pure-FTPd
- Bftpd
- Tcpdump
- ProFTPd
- Gifsicle
- FFmpeg
- Glibc
- FreeRDP
- GNOME
- QEMU
- GNU coreutils
- PostgreSQL
- Node.js
- libjxl
- Perl
- zlog

SQLANCER

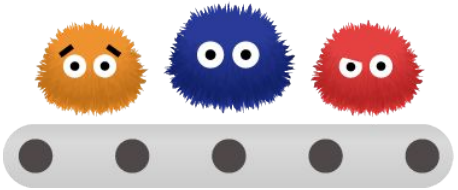
- SQLite (179)
- MySQL (19)
- PostgreSQL (13)
- MariaDB (1)
- CockroachDB (56)
- TiDB (55)
- DuckDB (73)

Csmith

	GCC	LLVM
Front end	0	10
Middle end	49	75
Back end	17	74
Unclassified	13	43
Total	79	202

Geneva: **Evolving Censorship Evasion Strategies**

Fray *test*



american fuzzy lop 0.47b (readpng)

- VLC
- Sqlite
- Vim
- Pure-FTPd
- Bftpd
- Tcpdump
- ProFTPd
- Gifsicle
- FFmpeg
- Glibc
- FreeRDP
- GNOME
- QEMU
- GNU coreutils
- PostgreSQL
- Node.js
- libjxl
- Perl
- zlog

SQLANCER

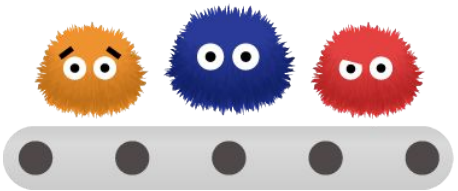
- SQLite (179)
- MySQL (19)
- PostgreSQL (13)
- MariaDB (1)
- CockroachDB (56)
- TiDB (55)
- DuckDB (73)

Csmith

	GCC	LLVM
Front end	0	10
Middle end	49	75
Back end	17	74
Unclassified	13	43
Total	79	202

Geneva: **Evolving Censorship Evasion Strategies**

Fray *test*



Random Testing is the best mechanism we know of for building resilient, high performance, safety-critical, and reliable systems.

That is why need it.

A bit of code.

Hypothesis

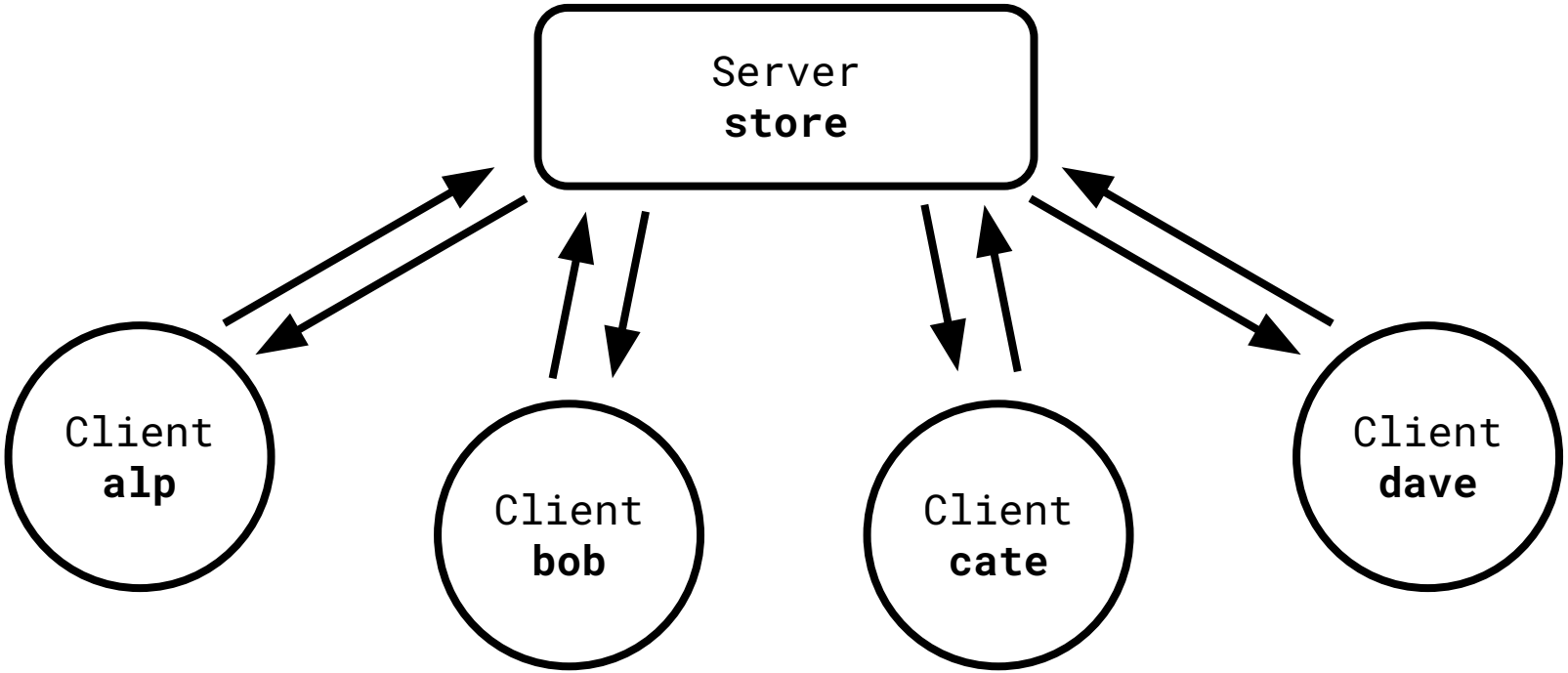
```
def test_append():
    l = [1, 2, 3]
    l.append(4)
    assert l == [1, 2, 3, 4]
```

```
def test_append2():
    l = [1, 2, 3]
    l.append(4)
    assert 4 in l
```

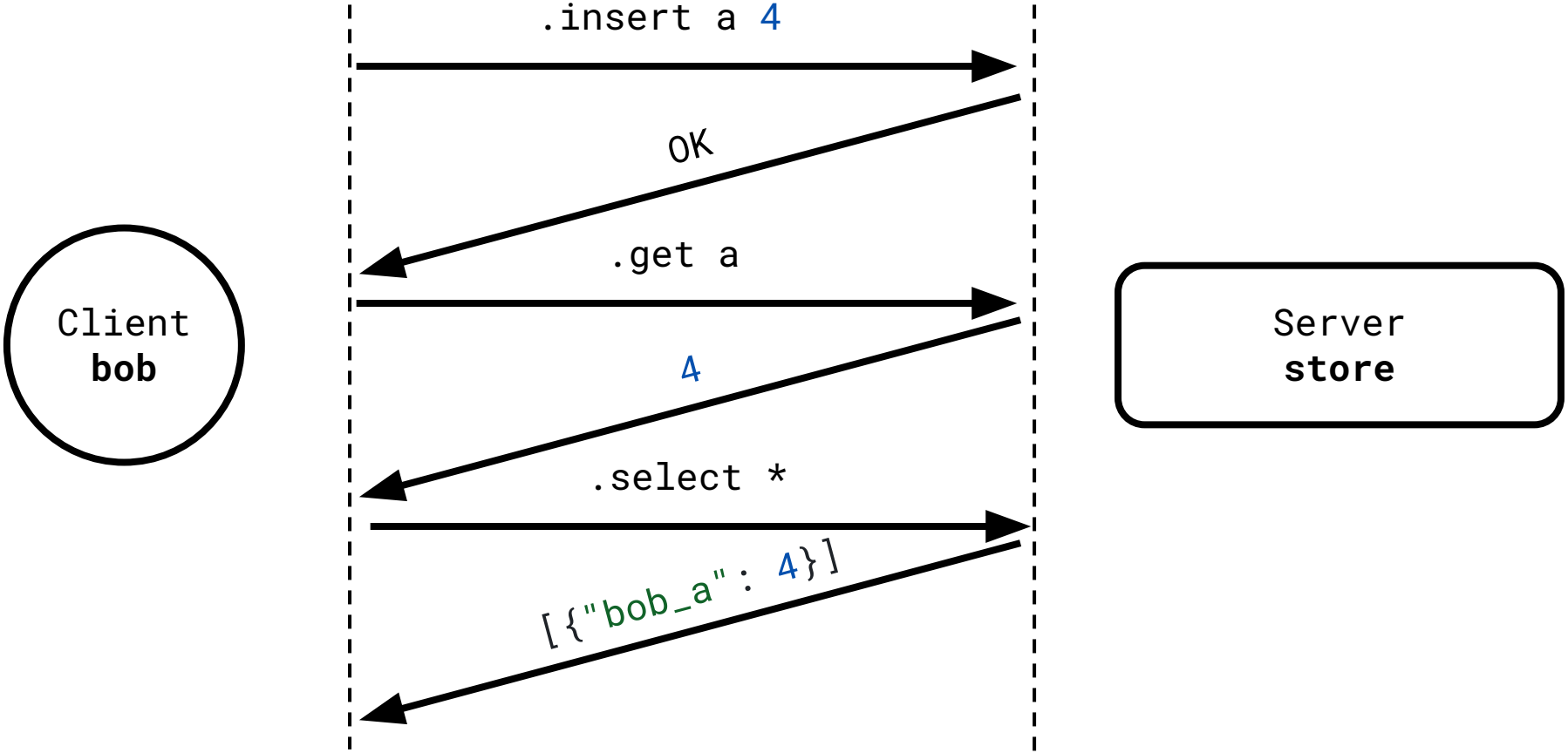
```
@given(lists(integers()))
def test_append3(l:
list[int]):
    l.append(4)
```

```
@given(lists(integers()),integers())
def test_append4(l: list[int], x):
    l.append(x)
    assert x in l
```

A Case Study: Key-Value Store



A Case Study: Key-Value Store



A Case Study: Key-Value Store

Roundtrip Properties

Saving the store and loading it restores its state.

$\forall$ store. store.save().load() == store

A message serialized from the client is correctly deserialized at the server, and vice versa.

```
@given(messages())
def test_serialize_deserialize(msg: Message) -> None:
    serialized = msg.serialize()
    deserialized = Message.deserialize(serialized)
    assert msg == deserialized, f"{msg} != {deserialized}"
```

A Case Study: Key-Value Store

Message Serialization/Deserialization

```
Insert(k="foo", v={"bar": 42})
```

```
string\r\ninsert\n\n
string\r\nfoo\n\n
object\r\n{'bar': 42}
```

type	data
"string"	"insert"
"string"	"foo"
"object"	{"bar": 42}

A Case Study: Key-Value Store

Delete(k='**\n**aN\t%\rR+!eg')
b'string\r\ndelete\n\nstring**\r\n\n**aN\t%\rR+!eg'

Insert(k='JnaOdp!I\n1[rb**\n**', v=None)
b'string\r\ninsert\n\nstring\r\nJnaOdp!I\n1[rb**\n\n\n**null\r\nNone'

Insert(k='', v='')
b'string\r\ninsert\n\nstring**\r\n\n\n**string**\r\n**'

A Case Study: Key-Value Store

```
@composite
def inserts(draw: DrawFn) -> Insert:
    k = draw(text(alphabet=string.printable, min_size=1))
    v = draw(json())
    return Insert(k=k, v=v)

@composite
def selects(draw: DrawFn) -> Select:
    k = draw(text(alphabet=string.printable, min_size=1))
    try:
        left = draw(integers(min_value=0, max_value=len(k) - 1))
        right = draw(integers(min_value=left, max_value=len(k)))
        k = k[:left] + "*" + k[right:]
    except Exception:
        pass
    return Select(k=k)
```

A Case Study: Key-Value Store

```
@composite
def inserts(draw: DrawFn) -> Insert:
   k = draw(text(alphabet=string.printable, min_size=0))
   v = draw(json())
   return Insert(k=k, v=v)


@given(messages())
def test_serialize_deserialize(msg: Message) -> None:
   assume len(msg.k) > 0
   serialized = msg.serialize()
   deserialized = Message.deserialize(serialized)
   assert msg == deserialized, f"{msg} != {deserialized}"
```

A Case Study: Key-Value Store

Message Serialization/Deserialization

```
Insert(k="foo", v={"bar": 42})
```

```
$6\r\ninsert\r\n
$3\r\nfoo\r\n
*1\r\n$3\r\nbar\r\n:42\r\n
```

type	data
"string"	"insert"
"string"	"foo"
"object"	{"bar": 42}

A Case Study: Key-Value Store

Postcondition Property

No client should read any other clients data.

```
def check_isolation(result: str, client: Client, message: Message):
   match message:
       case Select(k):
           result = json.loads(result)
           if isinstance(result, list):
               for obj in result:
                   key = next(iter(obj))
                   prefix = key[: key.find("_")]
                   assert prefix == client.prefix
       case _:
           pass
```

A Case Study: Key-Value Store

Model Property

State should conform to the model

```
def check_state_model(result: str, client: Client, message: Message, st: dict):
    match message:
        case Insert(k, v):
            st[k] = v
        case Delete(k):
            if k in st:
                del st[k]
        case Get(k):
            if k in st:
                assert st[k] == json.loads(result)
        case _:
            pass
```

A Case Study: Key-Value Store

```
@composite
def interactions(draw: DrawFn, clients: list[Client]) -> Interaction:
    choices = [
        (1, startups()),
        (1, stops()),
        (6, inserts()),
        (10, gets()),
        (4, deletes()),
        (10, selects()),
    ]
    choice = draw(weighted_choice(choices))

    # If the choice is a client interaction, choose a client
    if not isinstance(choice, tuple):
        client = draw(sampled_from(clients))
        choice = ("message", client, choice)

    return choice
```

Alperen Keles
akeles@umd.edu
alperenkeles.com

PLUM

Examples

- **List**
- **SortedList**
- **Key-Value Store**

https://github.com/alpaylan/testing-kvstore

Slides

alperenkeles.com/documents/topsort/slides.pdf