Property-Based Testing: The Past, The Present, and The Future

Alperen “Alp” Keles
Ph.D. Student at University of Maryland

Objectives

What is PBT?

How to apply PBT?

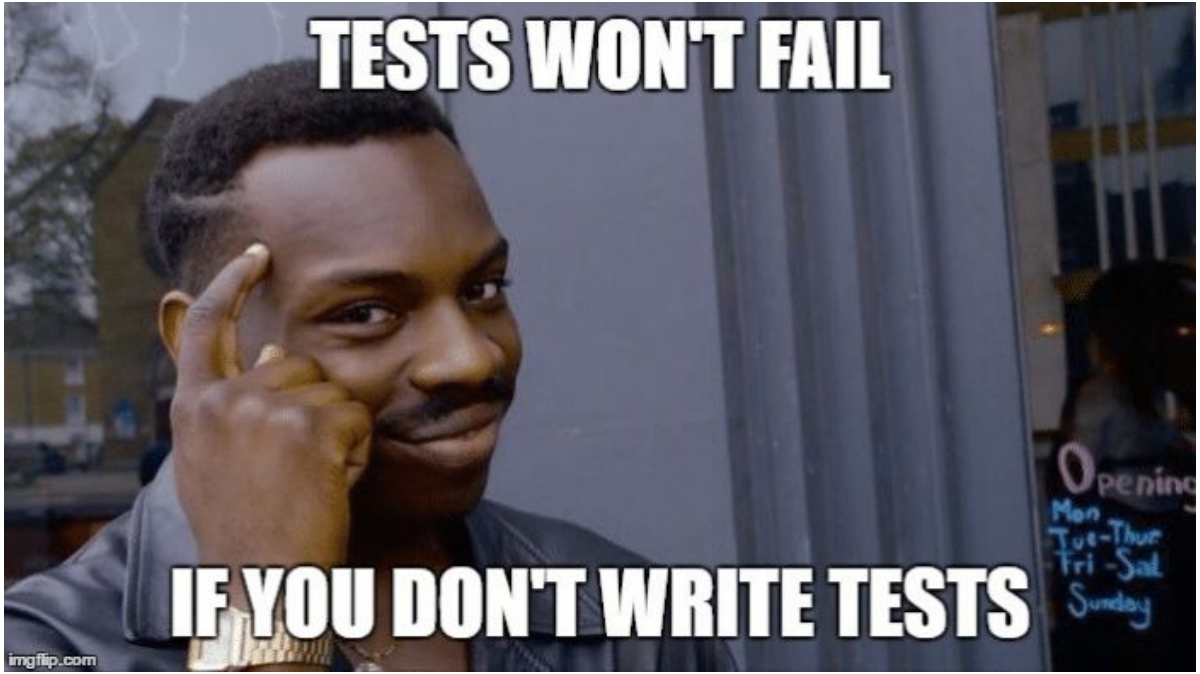
Why should one do PBT?

Outline

What PBT is.

Let’s test some things.

Things I don’t have the time to talk about.


TESTS WON'T FAIL
IF YOU DON'T WRITE TESTS
imgflip.com
Opening
Mon
Tue-Thur
Fri -Sat
Sunday

What is Property-Based Testing?

A lightweight tool for random testing of programs

PBT = Random Generation + Executable Specifications

What is Property-Based Testing?

A lightweight tool for random testing of programs

PBT = Random Generation + Properties

Example Based Test

Example based tests are executable specifications over specific inputs to programs.

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

*all examples will be using Python and Hypothesis

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

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

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

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

Example Based Test

```
append([1, 2, 3], 4) == [1, 2, 3, 4]
```

Property Based Tests

List contains the appended element.

```
∀ l, x. x in append(l, x)
```

Last element of the list is the appended element.

```
∀ l, x. append(l, x)[-1] == x
```

Prefix of the list does not change.

```
∀ l, x. append(l, x)[:-1] == l
```

List length increases by 1.

```
∀ l, x. len(append(l, x)) == len(l) + 1
```

```python
def append(l: list, x: any):
    l = l.copy()
    l.append(x)
    return l


@given(lists(), integers())
def test_contains(l: list, x):
   assert x in append(l, x)
```

Property

A property is an executable specification of a program over an abstract set of inputs.

Sorting is idempotent.

```
∀ l. sorted(sorted(l)) == sorted(l)
```

Reversing a list 2 times results in the original list.

```
∀ l. list(reversed(list(reversed(l)))) == l
```

```
class SortedList:
    def __init__(self, values=None):
        if values is None:
            values = []
        self.values = sorted(values)
```

```
class SortedList:
    def __init__(self, values=None):
        if values is None:
            values = []
        self.values = sorted(values)

    def insert(self, value):
        values = self.values.copy()
        i = 0
        while i < len(values) and values[i] < value:
            i += 1
        values.insert(i, value)
        return SortedList(values)
```

```
class SortedList:
    def __init__(self, values=None):
        if values is None:
            values = []
        self.values = sorted(values)

    def insert(self, value):
        values = self.values.copy()
        i = 0
        while i < len(values) and values[i] < value:
            i += 1
        values.insert(i, value)
        return SortedList(values)

    def is_sorted(self):
        for i in range(1, len(self.values)):
            if self.values[i] < self.values[i - 1]:
                return False
        return True
```

```
class SortedList:
    def __init__(self, values=None):
        if values is None:
            values = []
        self.values = sorted(values)

    def insert(self, value):
        values = self.values.copy()
        i = 0
        while i < len(values) and values[i] < value:
            i += 1
        values.insert(i, value)
        return SortedList(values)

    def is_sorted(self):
        for i in range(1, len(self.values)):
            if self.values[i] < self.values[i - 1]:
                return False
        return True
```

```
def delete(self, value):
    values = self.values.copy()
    v = self.find(value)
    if v ≠ -1:
        values.pop(v)
    return SortedList(values)
```

```
class SortedList:
    def __init__(self, values=None):
        if values is None:
            values = []
        self.values = sorted(values)

    def insert(self, value):
        values = self.values.copy()
        i = 0
        while i < len(values) and values[i] < value:
            i += 1
        values.insert(i, value)
        return SortedList(values)

    def is_sorted(self):
        for i in range(1, len(self.values)):
            if self.values[i] < self.values[i - 1]:
                return False
        return True
```

```
    def delete(self, value):
        values = self.values.copy()
        v = self.find(value)
        if v ≠ -1:
            values.pop(v)
        return SortedList(values)

    def find(self, value) → int:
        left = 0
        right = len(self.values) - 1
        while left ⩽ right:
            mid = (left + right) // 2
            if self.values[mid] == value:
                return mid
            elif self.values[mid] < value:
                left = mid + 1
            else:
                right = mid - 1
        return -1
```

```
@composite
def sorted_lists(draw: DrawFn) -> SortedList:
    pass
```

```
@composite
def sorted_lists(draw: DrawFn) -> SortedList:
    values = draw(lists(integers()))
    return SortedList(values)
```

```python
@composite
def sorted_lists(draw: DrawFn) -> SortedList:
    values = []
    lower_bound = draw(integers())

    length = draw(integers(min_value=0, max_value=100))
    for _ in range(length):
        value = draw(integers(min_value=lower_bound))
        lower_bound = value
        values.append(value)

    sl = SortedList()
    sl.values = values
    return sl
```

```python
@composite
def sorted_lists(draw: DrawFn) -> SortedList:
    values = []
    lower_bound = draw(integers())

    length = draw(integers(min_value=0, max_value=100))
    for _ in range(length):
        value = draw(integers(min_value=lower_bound))
        lower_bound = value
        values.append(value)

    sl = SortedList()
    sl.values = values
    return sl
```

```
@composite
def sorted_lists(draw: DrawFn) → SortedList:
    values = []
    lower_bound = draw(integers())

    length = draw(integers(min_value=0, max_value=100))
    for _ in range(length):
        value = draw(integers(min_value=lower_bound))
        lower_bound = value
        values.append(value)

    sl = SortedList()
    sl.values = values
    return sl
```

```
@composite
def sorted_lists(draw: DrawFn) → SortedList:
    values = []
    lower_bound = draw(integers())

    length = draw(integers(min_value=0, max_value=100))
    for _ in range(length):
        value = draw(integers(min_value=lower_bound))
        lower_bound = value
        values.append(value)

    sl = SortedList()
    sl.values = values
    return sl
```

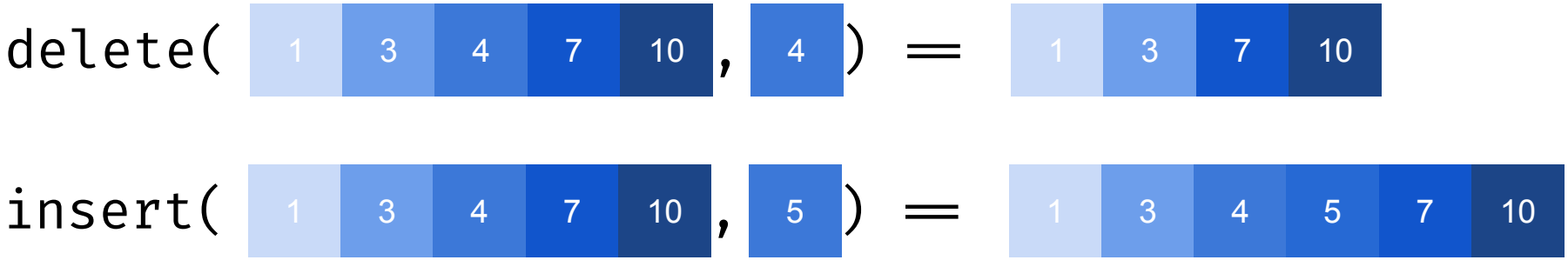
```
@composite
def sorted_lists(draw: DrawFn) → SortedList:
    values = []
    lower_bound = draw(integers())

    length = draw(integers(min_value=0, max_value=100))
    for _ in range(length):
        value = draw(integers(min_value=lower_bound))
        lower_bound = value
        values.append(value)

    sl = SortedList()
    sl.values = values
    return sl
```

Types of Properties

Validity Testing
Every operation should return valid results.



Every insert/delete on a sorted list should result in a sorted list.

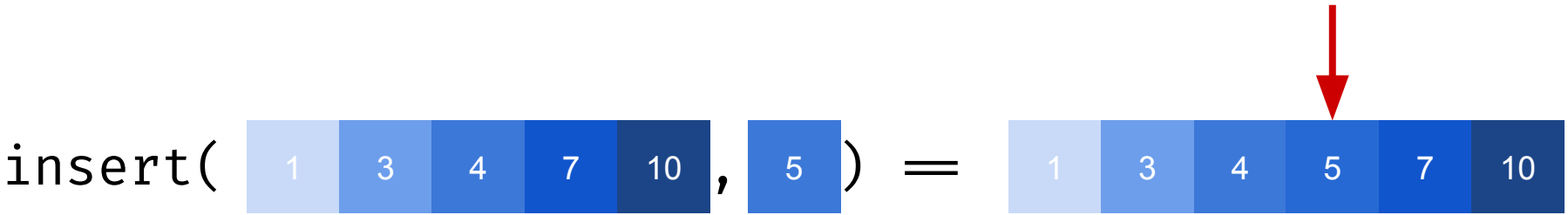
```
@given(sorted_lists(), integers())
def test_sorting_validity(sl, x):
    assert sl.insert(x).is_sorted()
    assert sl.delete(x).is_sorted()
```

* taken from How to Specify It!

Types of Properties

Validity Testing
Every operation should return valid results.

Postcondition Testing
The predicate should hold after any occurrence of the operation.



Inserted value should be in the list.

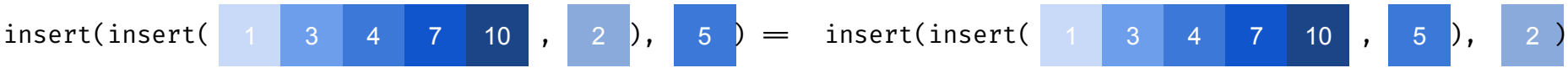
```
@given(sorted_lists(), integers())
def test_sorting_postcondition(sl: SortedList, x: int):
    assert x in sl.insert(x)
```

Types of Properties

Validity Testing
Every operation should return valid results.

Postcondition Testing
The predicate should hold after any occurrence of the operation.

Metamorphic Testing
Two operations should relate to each other.



Insertion order should be irrelevant.

```
@given(sorted_lists(), integers(), integers())
def test_sorting_metamorphic(sl: SortedList, x: int, y: int):
    assert sl.insert(x).insert(y) == sl.insert(y).insert(x)
```

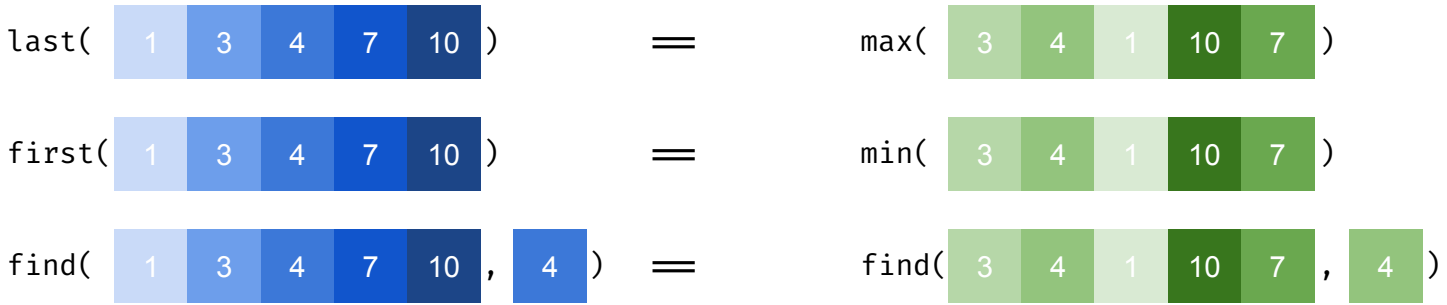
Types of Properties

Validity Testing
Every operation should return valid results.

Postcondition Testing
The predicate should hold after any occurrence of the operation.

Metamorphic Testing
Two operations should relate to each other.

Model Testing
The implementation should conform to the model.



```python
@given(lists(integers()), integers())
def test_sorting_model(l: list[int], x: int):
    sl = SortedList(l)
    if len(l) != 0:
        assert sl.last() == max(l)
        assert sl.first() == min(l)
        assert sl.find(x) == (x in l)
    else:
        assert len(sl) == 0
```

```python
@given( ... )
def test_sorting_model(l: list[int], x: int):
    sl = SortedList(l)
    if len(l) != 0:
        assert sl.last() == max(l)
        assert sl.first() == min(l)
        assert sl.find(x) == (x in l)
    else:
        assert len(sl) == 0
```

Some Practical Properties

Roundtrip Property

```
∀ j. JSON.parse(JSON.stringify(j)) == j
∀ j. JSON.stringify(JSON.parse(JSON.stringify(j))) ==
JSON.stringify(j)
∀ s. decompress(compress(s)) == s
```

Idempotency

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

Class Invariants

```
∀ t, x. is_bst(t) ⟹ is_bst(t.insert(x))
∀ date. is_date(date) ⟹ is_date(date.next())
```

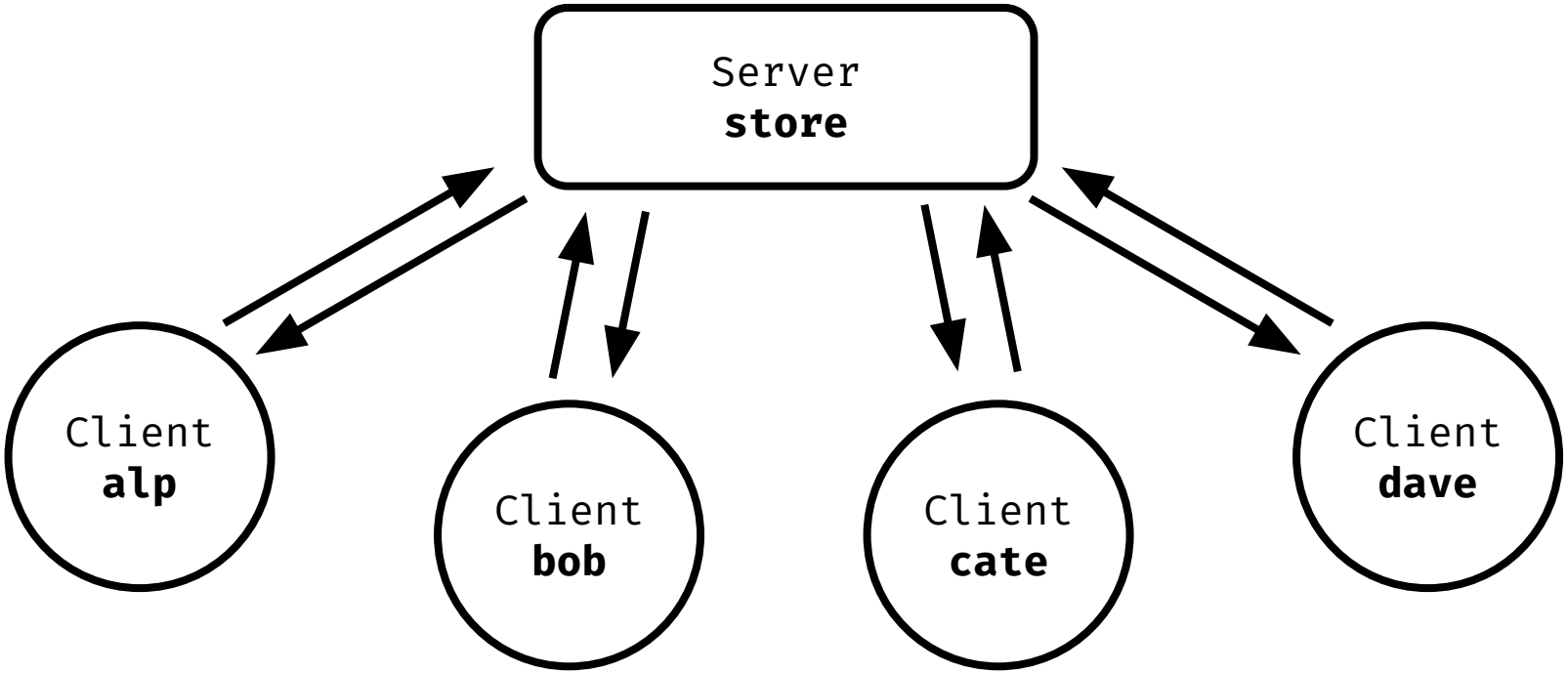
Fuzz Testing

```
Program does not crash.
```

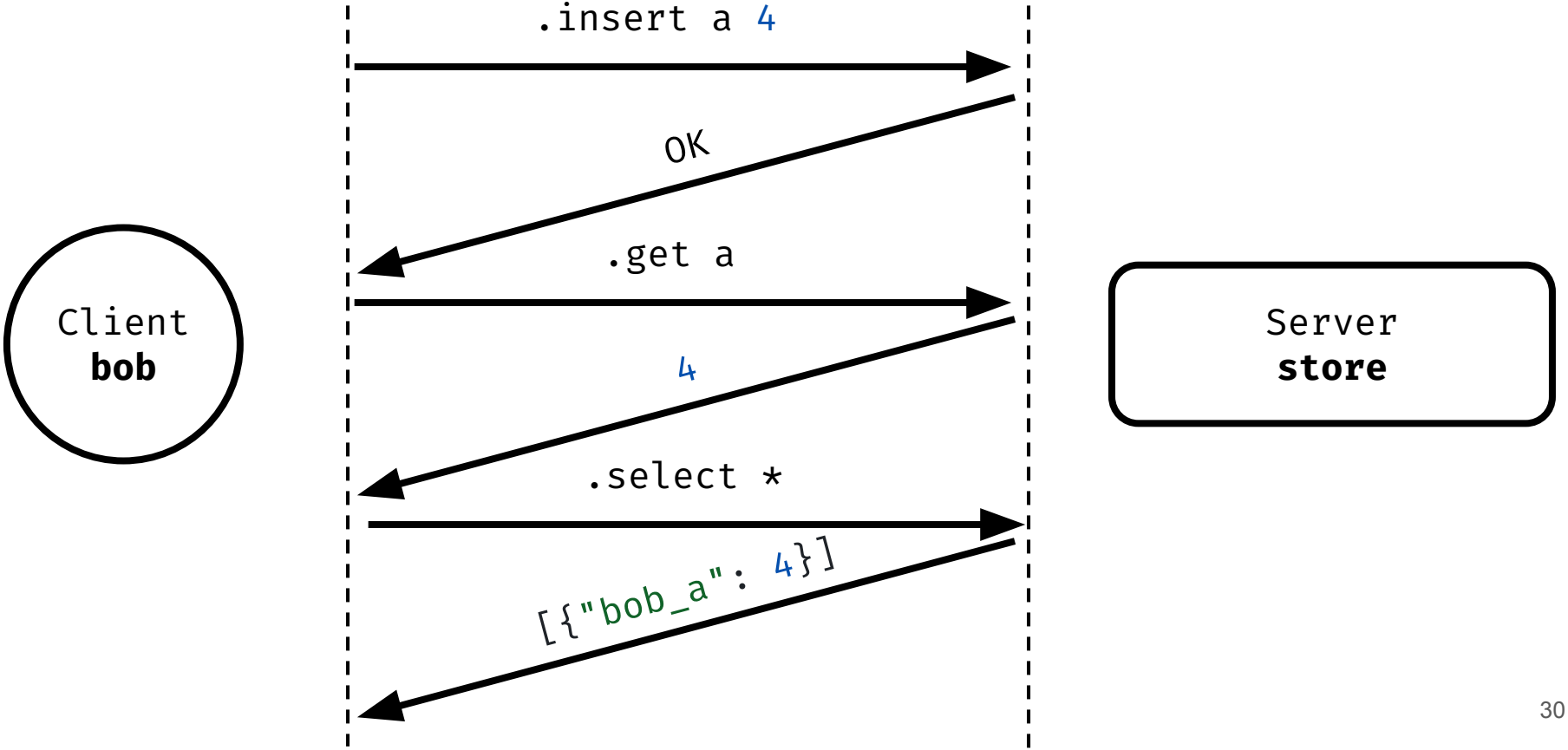
Differential Testing

```
∀ x. f1(x) == f2(x)
```

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.

```
∀ store. store.save().load() == store
```

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

```python
@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='\naN\t%\rR+!eg')
b'string\r\ndelete\n\nstring\r\n\naN\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\nnull\r\nNone'
```

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

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

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
```

```
[{"bob_a": 4}]
```

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
```

```
[
    {"bob_a": 4},
    {"bo_x": 2},
]
[bo] .select *
.select bo_*
```

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
```

```
[
    {"bob_a": 4},
    {"bo_x": 2},
]
[bo] .select *
.select bo_*
```

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

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

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

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
```

Control

Replaying Faulty Executions
Deterministic Tests

Integration

Integrated with testing and fuzzing frameworks

Automation

Deriving generators
Deriving shrinkers

Exploration

Systematic exploration of parameters

Inspection

Debugging
Statistics and Observability

Configurability

Picking your own parameters

Performance

Caching
Parallelization

Minimal Examples

Minimal Counterexamples via Shrinking

Guided Search

Targeted PBT
Fuzzing

Alperen Keles
akeles@umd.edu
alperenkeles.com

Examples

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

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

Slides

`alperenkeles.com/documents/bobkonf/slides.pdf`