Resilience Engineering

Identifying Reliability Dependencies And Common Mitigation Strategies

Nisan Haramati
@nisanharamati

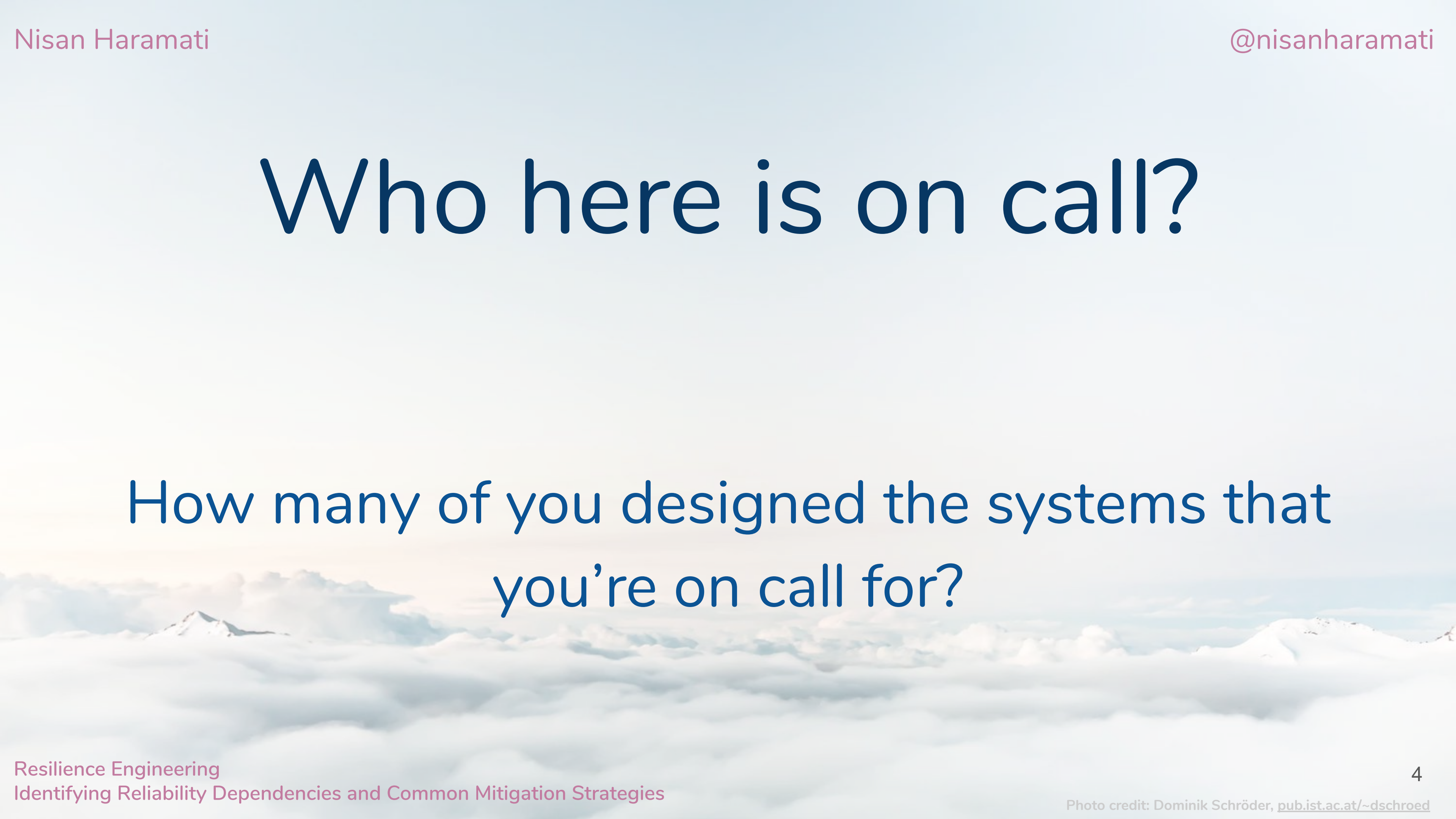
About

- Distributed data systems engineer at Wallaroo Labs
- Infrastructure and data engineering for about a decade
- **USB** principle for systems and tools
  - **U**se it, **S**cale it, **B**reak it

Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed

Who here is on call?

Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed



Who here is on call?

How many of you designed the systems that you’re on call for?

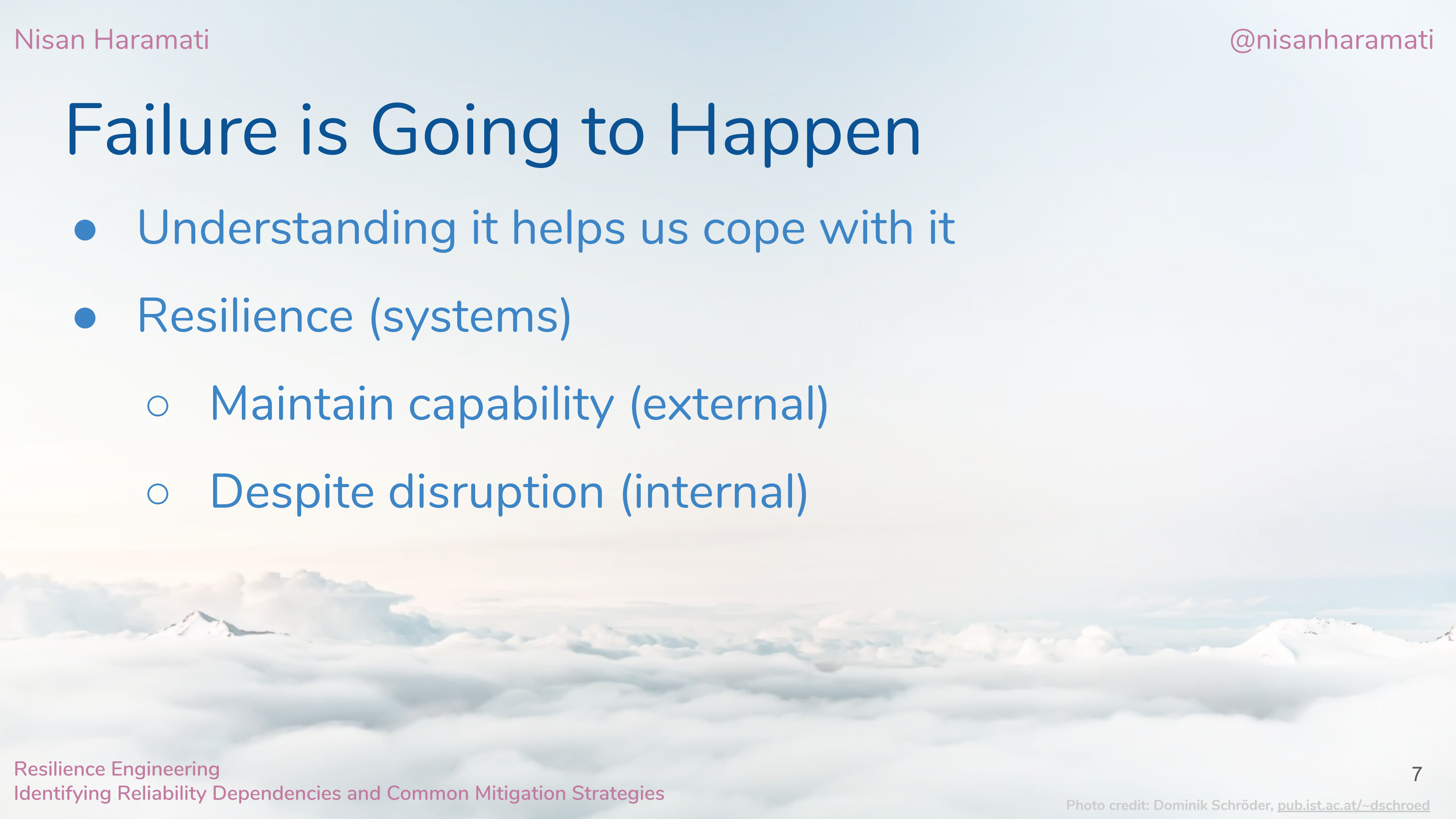
Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed

Not Your Fault

Failure is Inevitable

Failure is going to happen

Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed



Failure is Going to Happen

- Understanding it helps us cope with it
- Resilience (systems)
  - Maintain capability (external)
  - Despite disruption (internal)

Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed

Tools & Methods

- Mapping dependencies
- Failure isolation, hedging, graceful degradation
- Destructive testing
- End-to-end testing
- Fuzzing

Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed

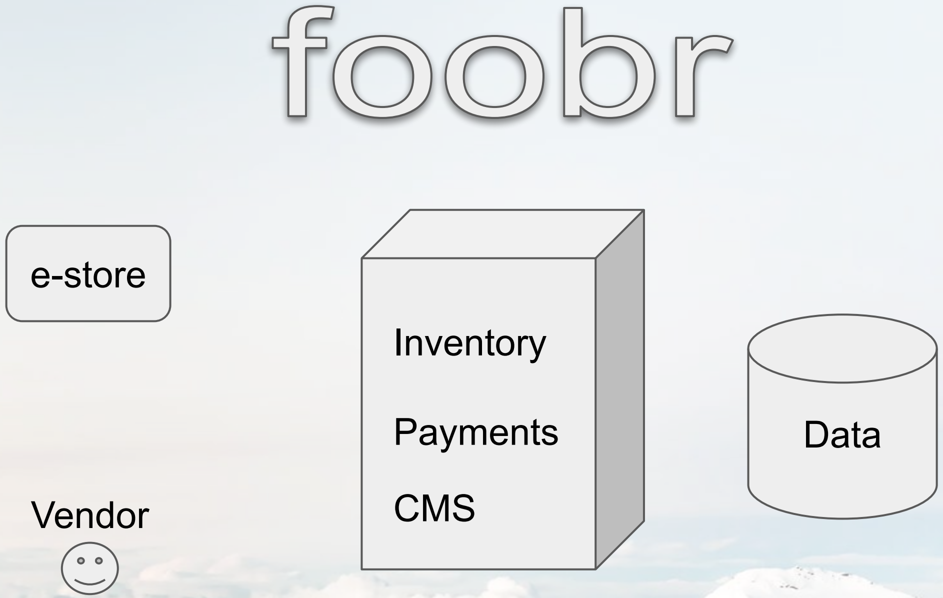
It’s about understanding…

- Failure
  - ➢ Components
  - ➢ Systems
  - ➢ … Us? 🤔

Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed

Mapping Dependencies

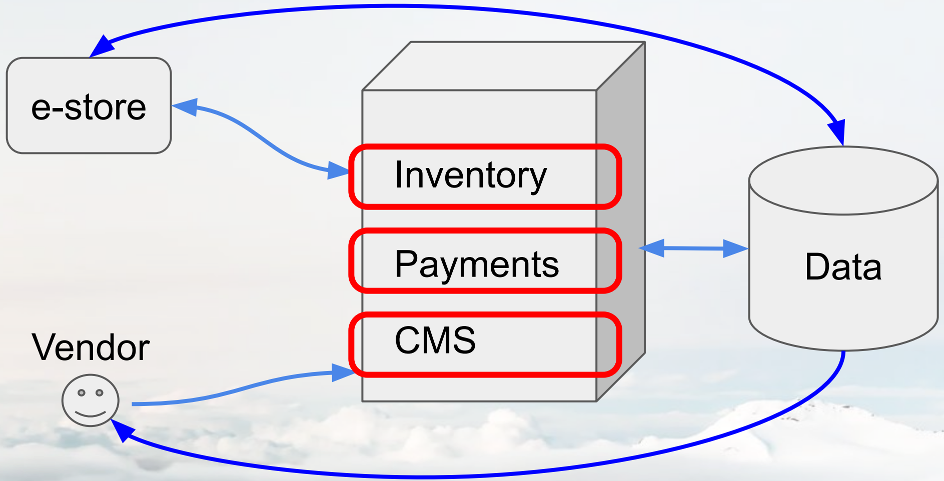
- Inventory
- CMS
- Payments



Mapping Dependencies

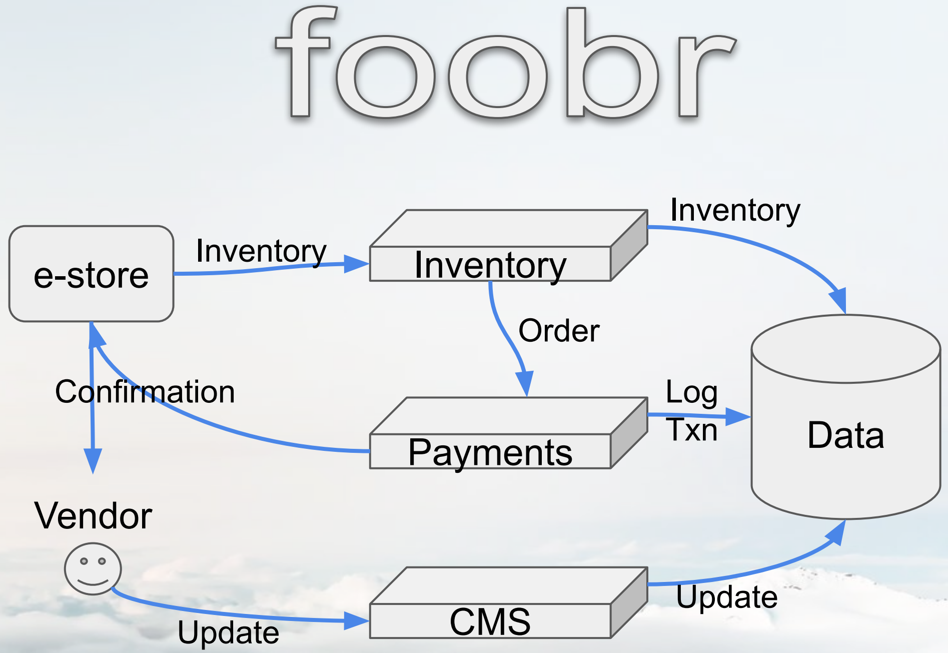
- Direct dependencies
- Indirect dependencies
- Failure isolation

foobr



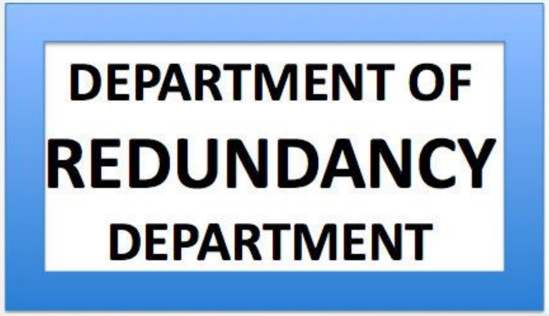
Remove Unnecessary Coupling

- Direct dependencies
- Indirect dependencies
- Failure isolation



Hedging

- Add redundancies

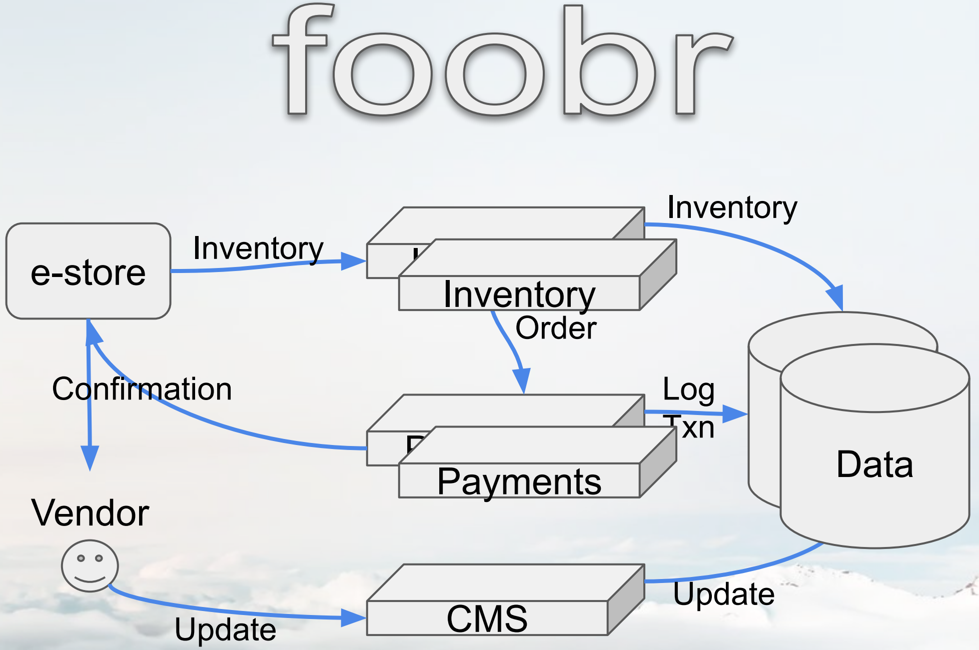


DEPARTMENT OF
REDUNDANCY
DEPARTMENT

Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed

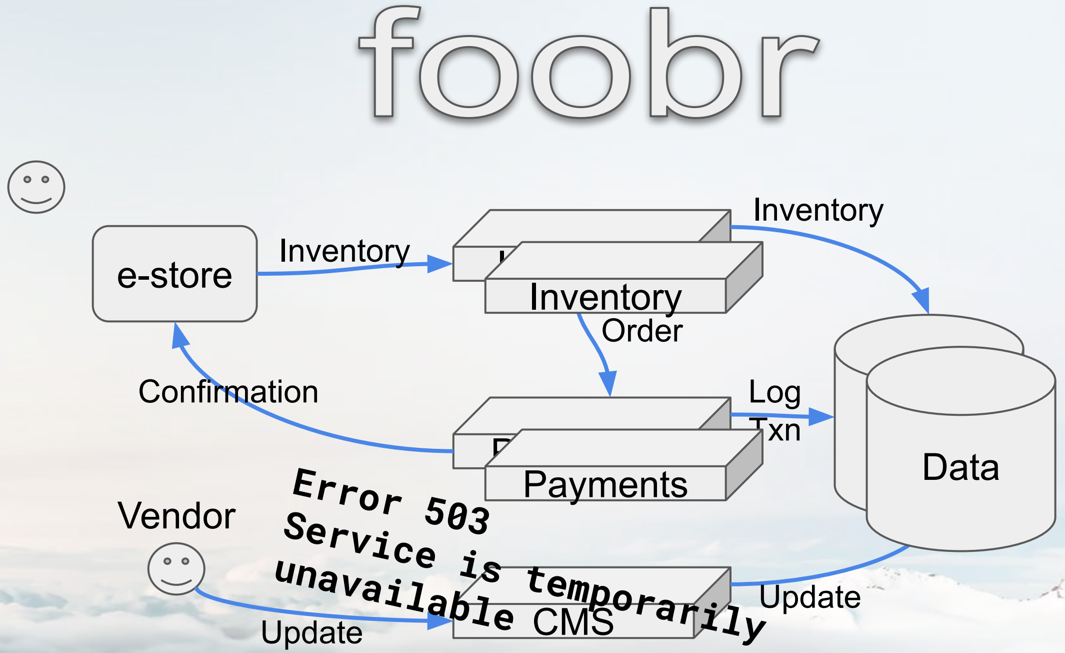
Hedging

- Add redundancies
- Plumbing
- Understanding



Graceful Degradation

- Partial availability
- Partial value

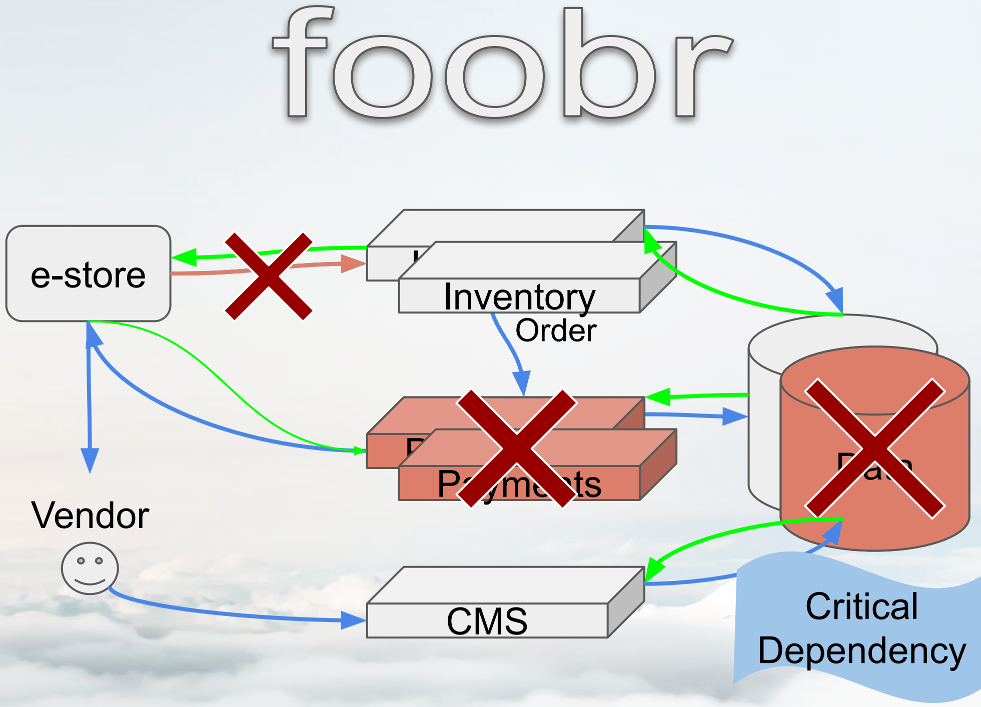


Failure is (still) going to happen

Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed

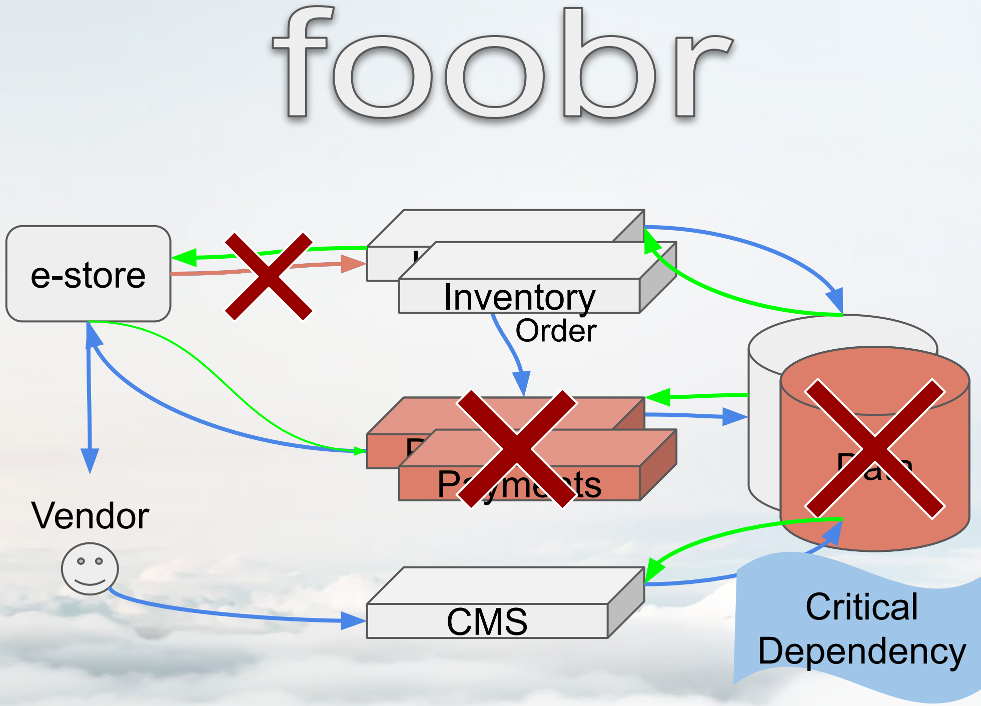
Destructive Testing

- Fault injection
- Interpolate dependencies
- Extrapolate properties



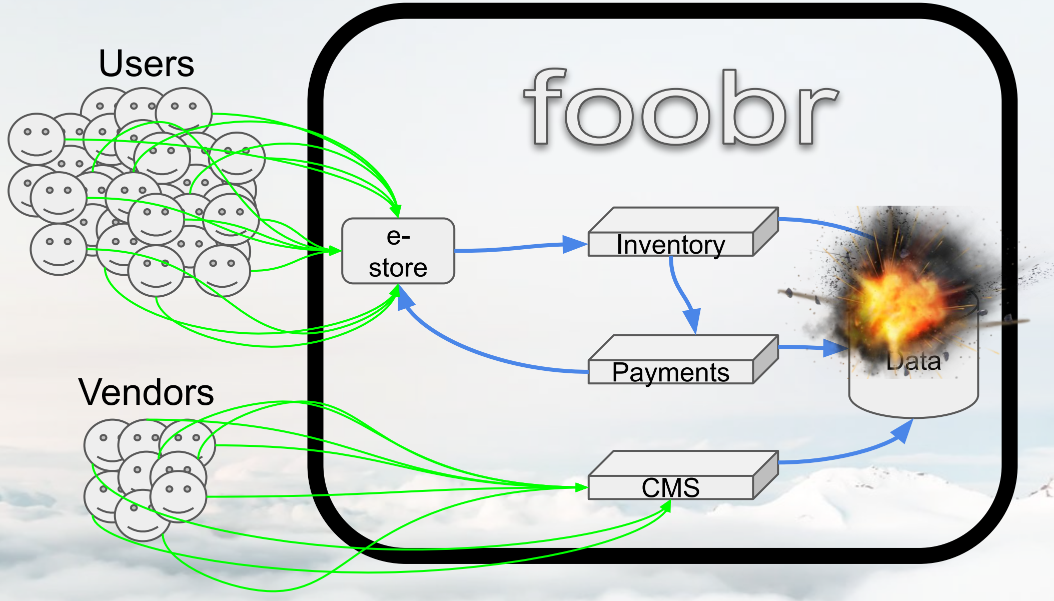
Destructive Testing

- An idea...
- Test in prod?



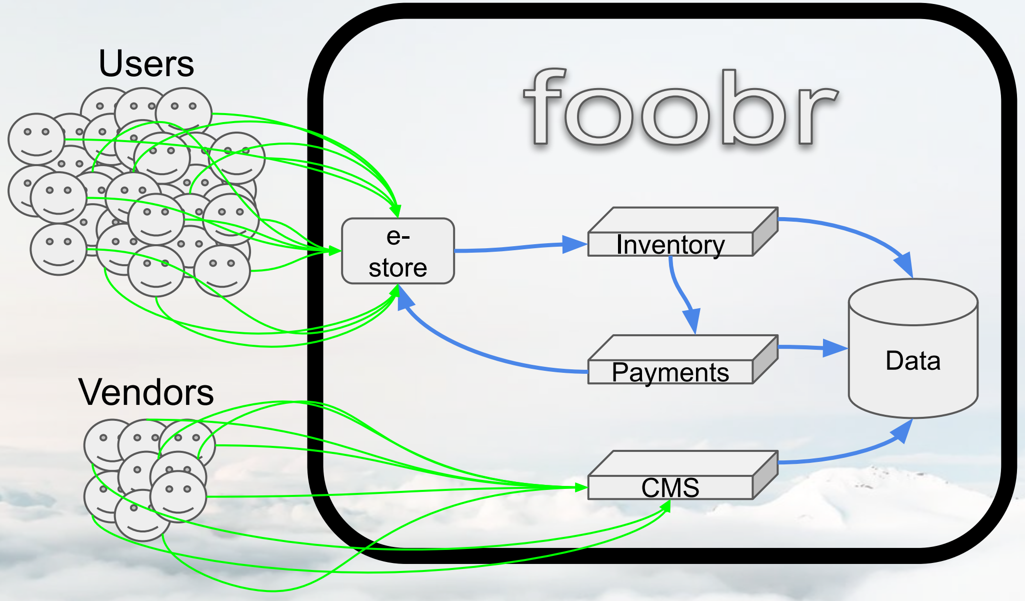
End-to-end Testing

- Entire system as a box
- Realistic conditions
    - Happy paths :)
    - Sad paths   :(



End-to-end Testing

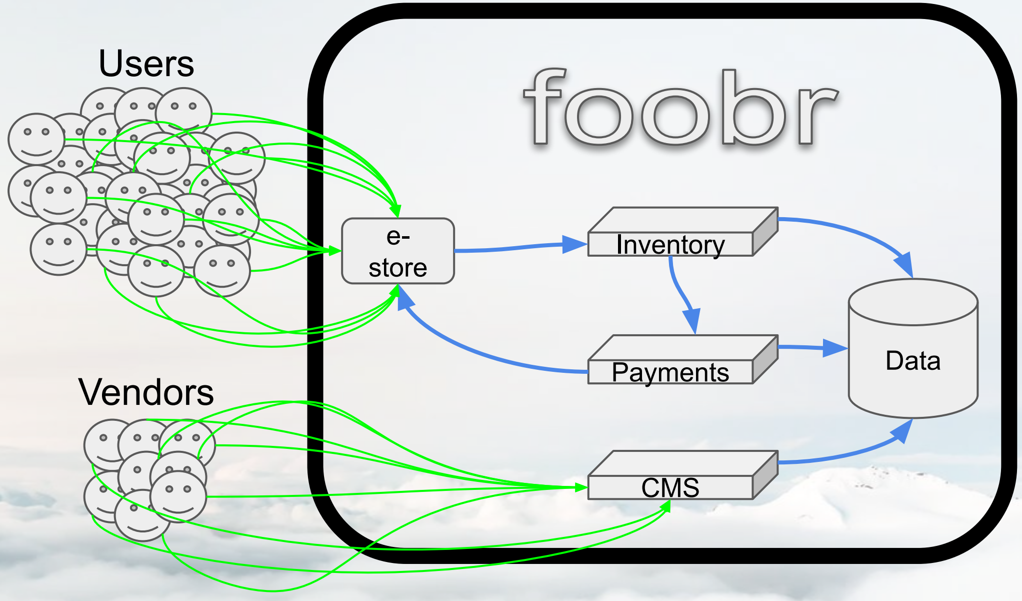
- Integration tests
  - But all together
  - The whole system
- Production-like conditions



End-to-end Testing

Requires

- Instrumentation
- Distribution
- Remote control?
- Measurement?



Failure is going to happen

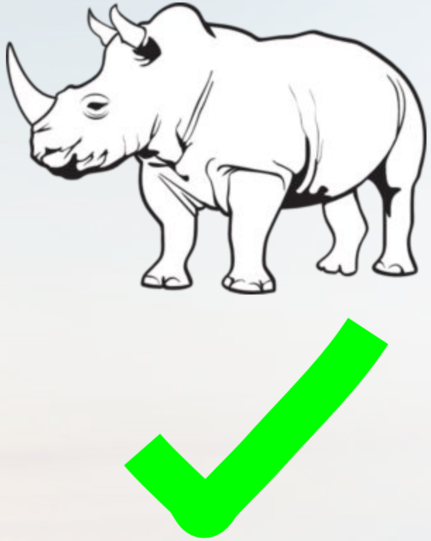
(just expect it at this point)

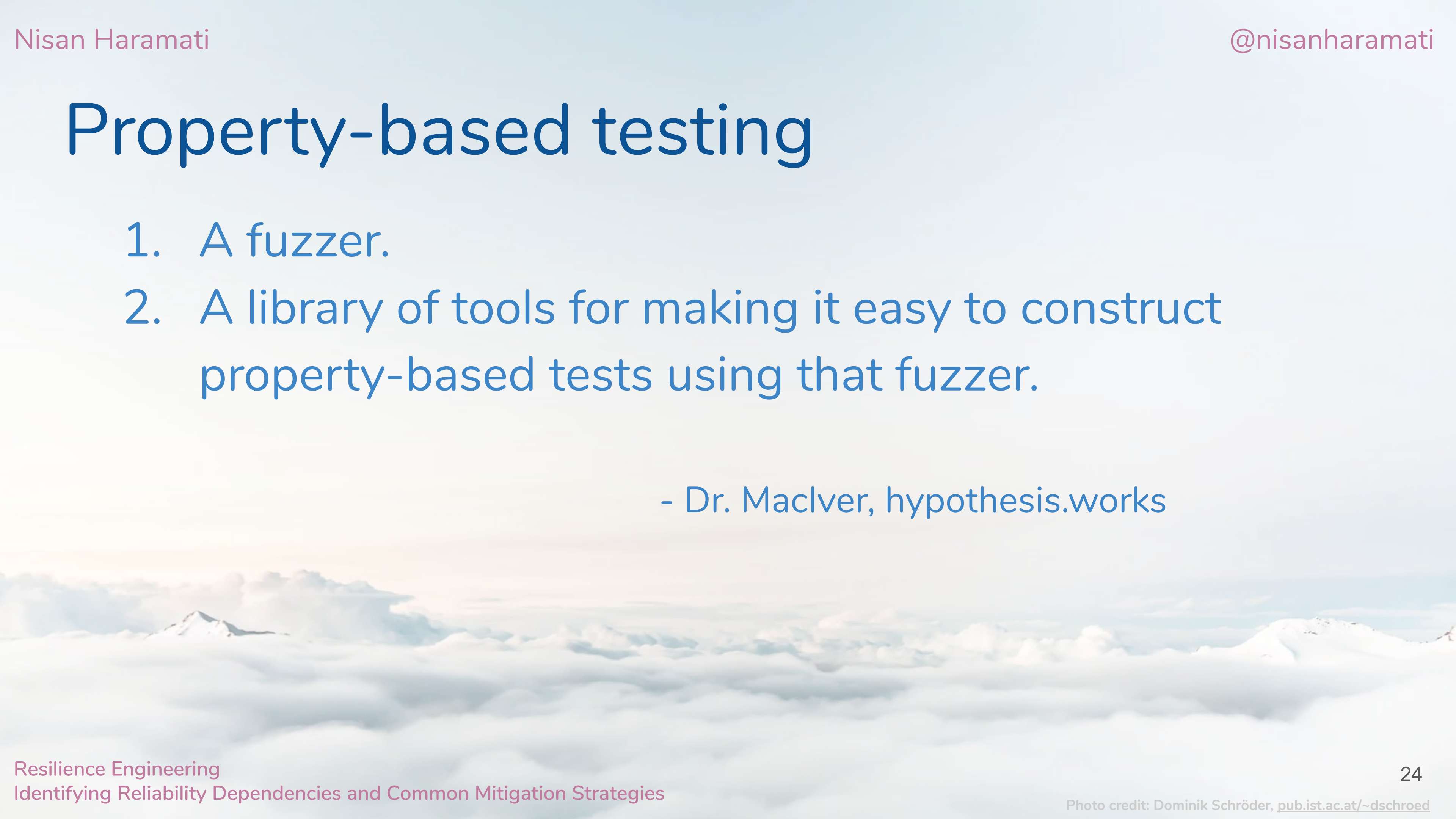
Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed

Property-based testing

Is this a unicorn?

- Has 1 horn
- Has 4 legs
- Has 1 tail
- Has 2 ears





Property-based testing

1. A fuzzer.
2. A library of tools for making it easy to construct property-based tests using that fuzzer.

- Dr. MacIver, hypothesis.works

Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed

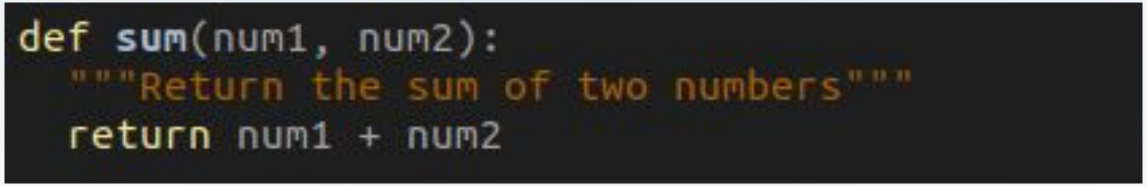
Fuzzer

- Produce input data for the test
- Possibly dynamically generated
- Possibly dependent on results of previous runs

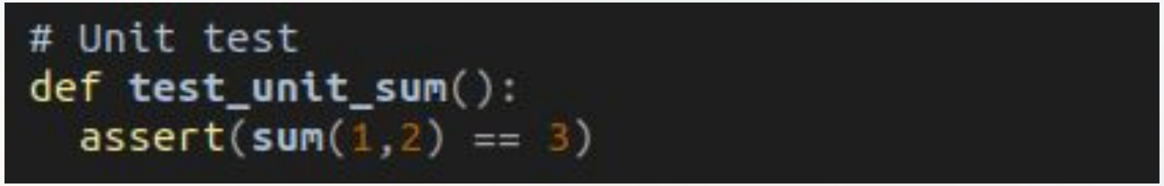
- Dr. MacIver, hypothesis.works

Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed

Property-based testing



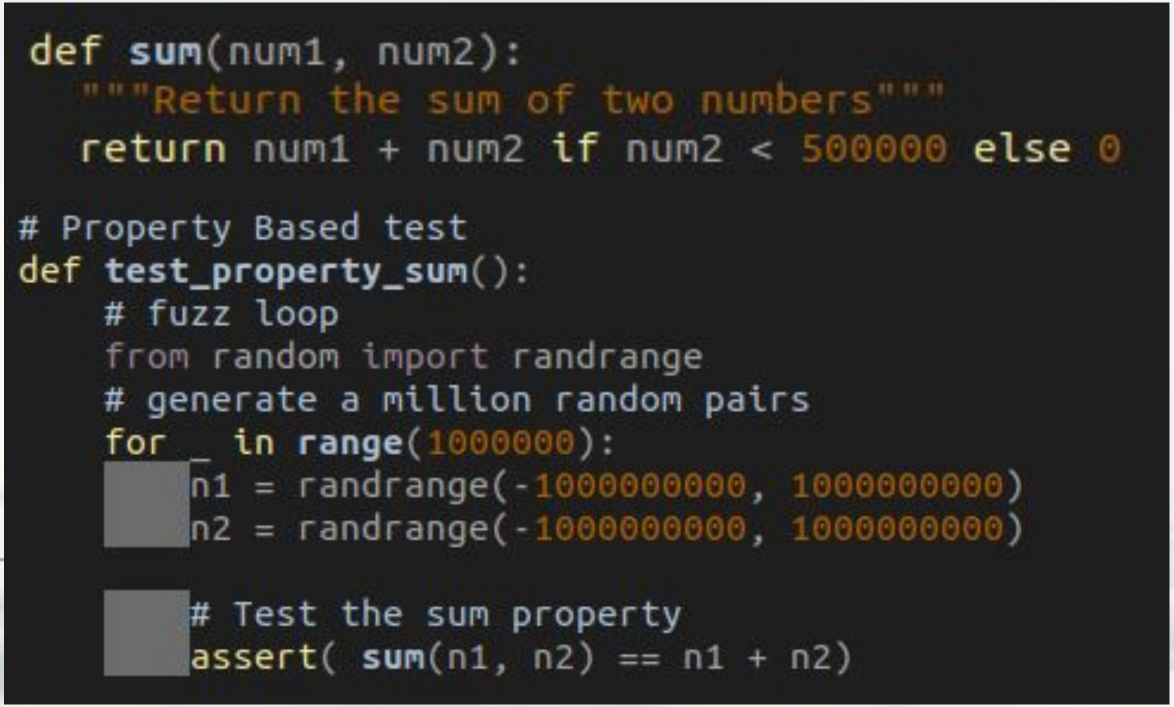
```
def sum(num1, num2):
  """Return the sum of two numbers"""
  return num1 + num2
```



```
# Unit test
def test_unit_sum():
  assert(sum(1,2) == 3)
```

Okay...

Property-based testing

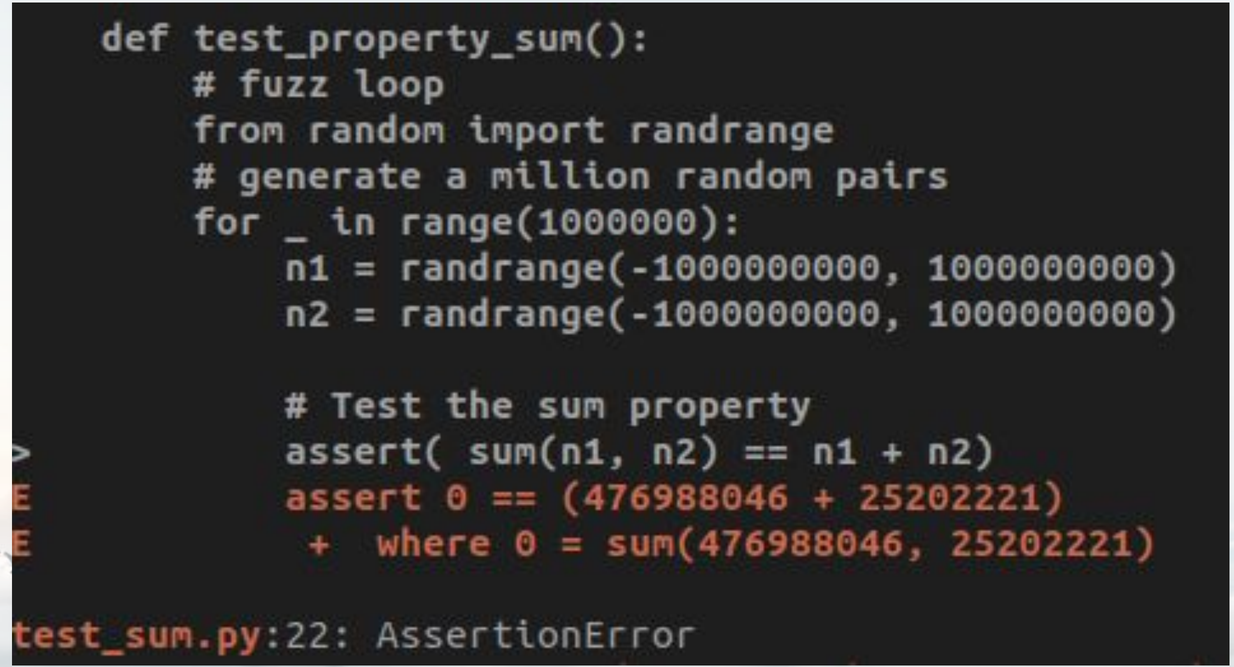


```python
 def sum(num1, num2):
   """Return the sum of two numbers"""
   return num1 + num2 if num2 < 500000 else 0

# Property Based test
def test_property_sum():
    # fuzz loop
    from random import randrange
    # generate a million random pairs
    for _ in range(1000000):
        n1 = randrange(-1000000000, 1000000000)
        n2 = randrange(-1000000000, 1000000000)

        # Test the sum property
        assert( sum(n1, n2) == n1 + n2)
```

Property-based testing



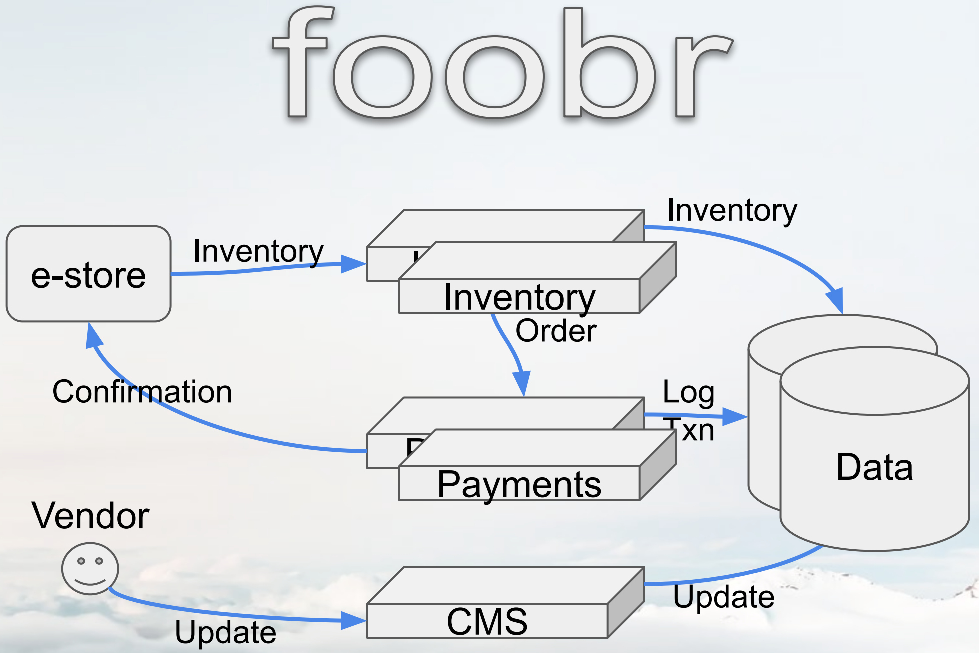
```
    def test_property_sum():
        # fuzz loop
        from random import randrange
        # generate a million random pairs
        for _ in range(1000000):
            n1 = randrange(-1000000000, 1000000000)
            n2 = randrange(-1000000000, 1000000000)

            # Test the sum property
>           assert( sum(n1, n2) == n1 + n2)
E           assert 0 == (476988046 + 25202221)
E            +  where 0 = sum(476988046, 25202221)

test_sum.py:22: AssertionError
```

Fuzzing Foobr

- Input data
  - Shoppers
  - Vendors
- System Events
  - Network faults & recovery
  - Service faults & recovery

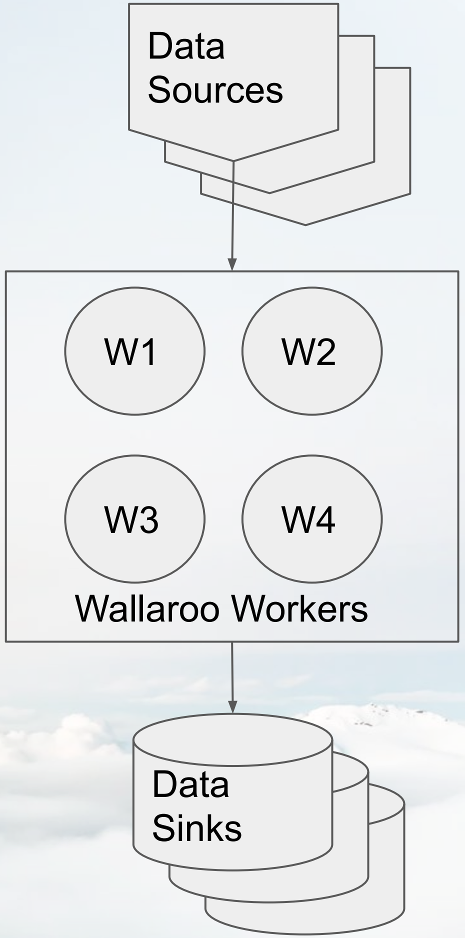


Fuzzing in Wallaroo Tests

- Input Data

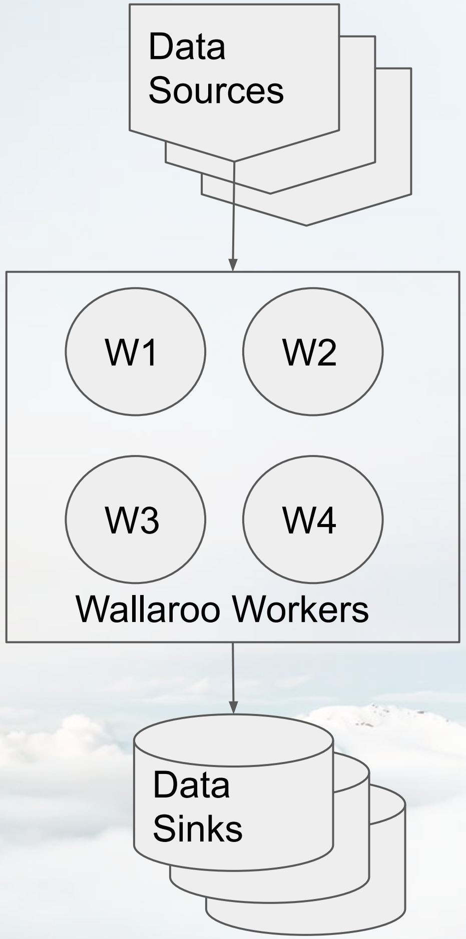
```
[(key:1, data:[1,2,3,4,…]),
 (key:2, data:[1,2,3,4,…]),…]
```

- Output Validation
  - Each key ⇒ a sequence
    no loss, no duplication,
    no reordering



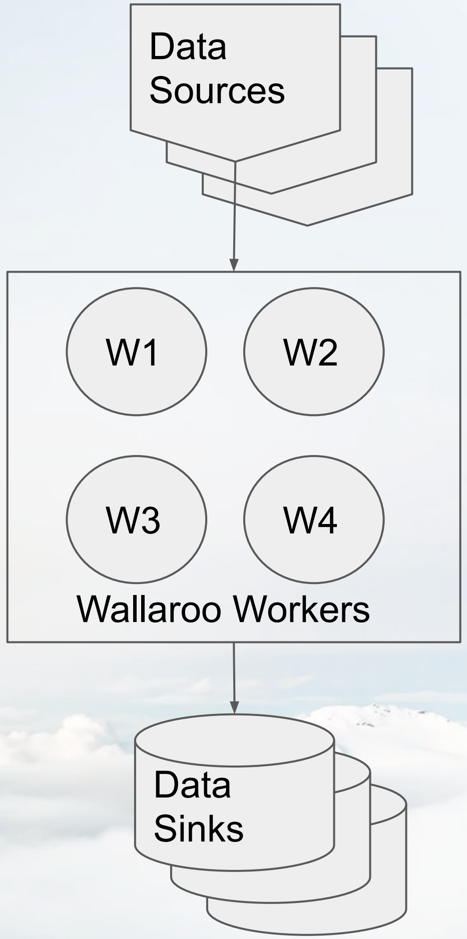
Fuzzing in Wallaroo Tests

- System events
  - Add Nodes
  - Remove nodes
  - Crash Nodes
  - Recover Nodes
  - Network faults



Fuzzing in Wallaroo Tests

- Application Topologies
  - Sources and Sinks
  - Computations with and without state
  - Key_by's (partitioning)
  - Filters and Multipliers (output cardinality)



Key Approaches

1. Map dependencies
2. Isolate, hedge, and degrade gracefully
3. Destructive testing
4. End-to-end testing
5. Fuzzing

Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed

Failure is Gonna Happen!

- Understanding it helps us live with it
- Resilience is the property of being able to sustain failure (internally) without loss of quality (externally)

Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed

Questions?

Nisan Haramati
@nisanharamati
haramati.ca

Photo credit: Dominik Schröder, pub.ist.ac.at/~dschroed