TRUST BUT VERIFY

QUIZ TIME

Does the following function work as expected?

```python
def is_weekend(d: datetime.date) -> bool:
    return d.weekday() in (6, 7)
```

a) Yes

b) No

c) It depends

😳 QUIZ TIME - ROUND 2

What do you know about unit testing?

a) I've written a unit test in the past month
b) I used to write unit tests in a galaxy far far away
c) What is a unit test?

WHAT WILL WE COVER?

1. Why is testing a topic we should care about?
2. What is hard about testing optimization models in particular?
3. Unit testing optimization code
4. Performance testing optimization code

WHY CARE ABOUT TESTING?

CRASH COURSE IN UNIT TESTING

Arrange - Act - Assert

System under test:

```python
def get_distance(origin, destination):
    return np.sqrt((origin['x'] - destination['x'])**2
          + (origin['y'] - destination['y'])**2) / 1000
```

Simple unit test:

```python
def test_get_distance():
    origin = {'x': 0, 'y': 0}
    destination = {'x': 1000, 'y': 0}
    assert get_distance(origin, destination) == 1
```

PROPERTY-BASED TESTING

Instead of writing specific test cases, what if we could **check general properties** instead?

```
def value():
        return st.floats(min_value=-1e6, max_value=1e6,
                  allow_nan=False)

@given(
    o=st.fixed_dictionaries({'x': value(), 'y': value()}),
    d=st.fixed_dictionaries({'x': value(), 'y': value()})
 )
def test_get_distance(o, d):
      distance = get_distance(o, d)
      assert distance >= 0
```

WHEN TO USE WHAT?

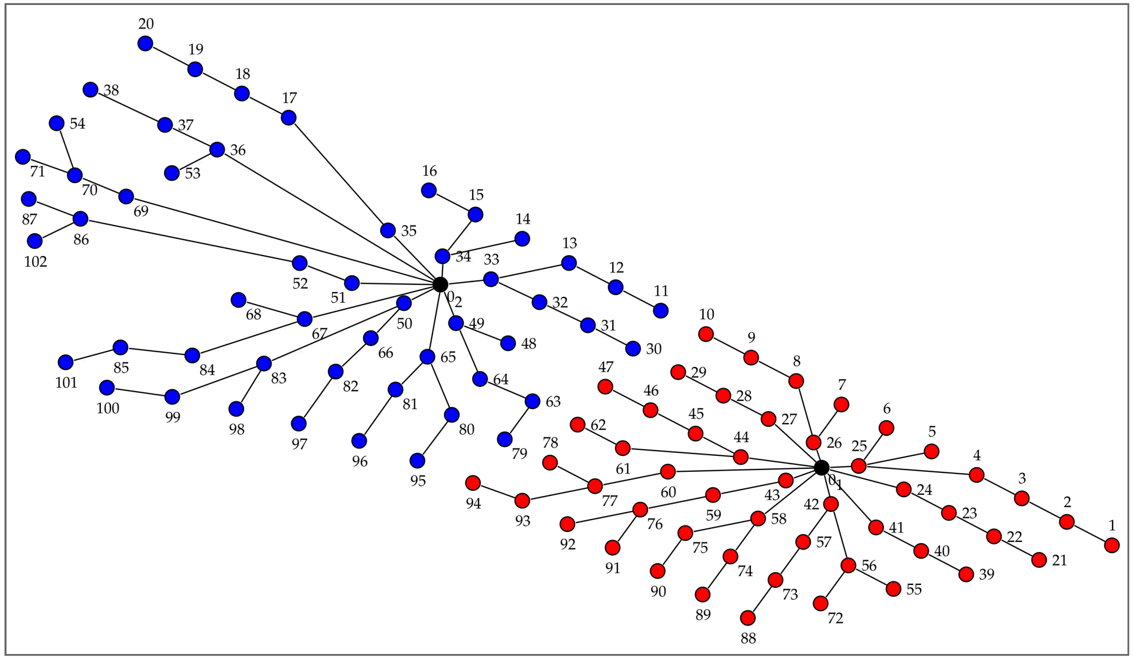
Classical	Property-based
Easy to setup and understand	Pain to setup
Hard to find all edge-cases	Edge-cases covered "for free"
Works especially well for "simple" functions	Great for complex patterns

WHY IS TESTING OPTIMIZATION MODELS HARD?

🛹 OR people write code, but rarely learn testing principles
💿 Mathematical **models** only "make sense" as a whole
🫷 You may have licensing restrictions from a commercial solver
⏳ Setting up test instances can be time consuming

THE TEST CASE TODAY #DADJOKE

How to connect wind turbines ("WTGs") to offshore substations ("OSS")



MODEL BASICS

- Minimize cost of cables: $\sum p_c x_c$
- Subject to:
- All turbines connected: $x_{out} = 1$
- Flow balance: $f_{in} + P = f_{out}$
- Flow limit on cable type: $f_c \leq M_c$
- Cables cannot cross: $x_1 + x_2 \leq 1 >$
- Max number of cable types: $\sum_c z_c \leq L$

> Indices omitted 😎 just because I can

LET'S LOOK AT THE CODE

UNIT TESTING THE CODE

CORE PRINCIPLE #1

Move as much of the logic out of model-building as possible.

EXAMPLE

```
for cable_type in cable_types:
  for c in connections:
    if c.cable_type == cable_type:
      model.add_linear_constraint(x[c] <= z[cable_type])
```

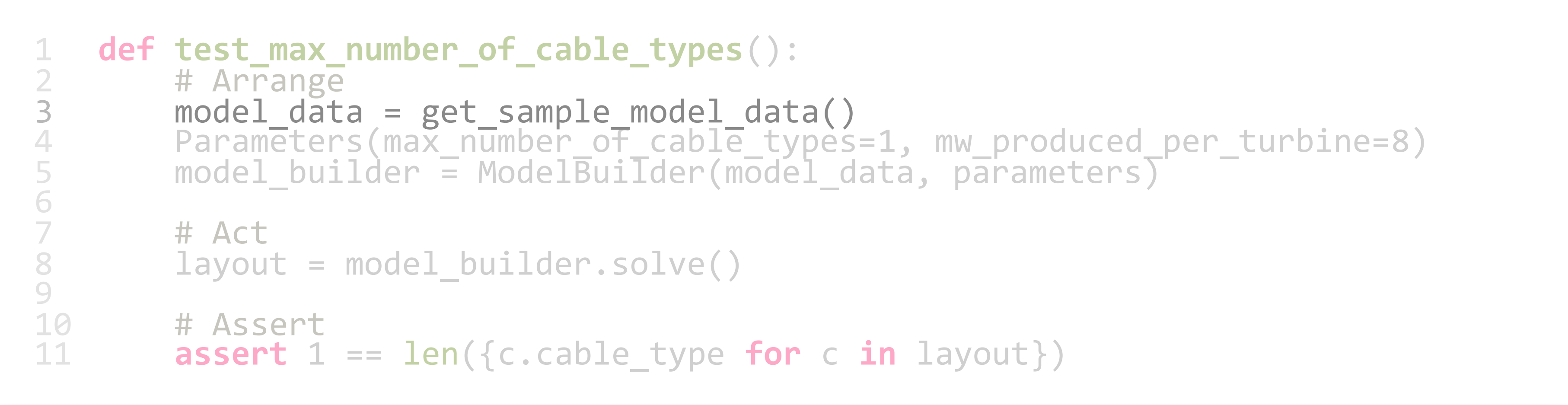
```
def get_connections_with_same_cable_type(connections, cable_type):
    return {c for c in self.connections if c.cable_type == cable_type}

for cable_type in cable_types:
  for c in get_connections_with_same_cable_type(connections, cable_type):
    model.add_linear_constraint(x[c] <= z[cable_type])
```

CORE PRINCIPLE #2

A model encodes a business problem. Test solutions against business rules which don't know optimization, and should come from the business.

EXAMPLE

"A valid layout will have a limited number of cable types"



```python
def test_max_number_of_cable_types():
    # Arrange
    model_data = get_sample_model_data()
    Parameters(max_number_of_cable_types=1, mw_produced_per_turbine=8)
    model_builder = ModelBuilder(model_data, parameters)

    # Act
    layout = model_builder.solve()

    # Assert
    assert 1 == len({c.cable_type for c in layout})
```

USING PROPERTY-BASED TESTING

```python
@given(model_data=model_data_st())
@settings(deadline=None, max_examples=500)
def test_max_number_of_cable_types(model_data):
    # Arrange
    parameters = Parameters(max_number_of_cable_types=1, mw_produced_per_turbine=8)
    model_builder = ModelBuilder(model_data, parameters)

    # Act
    layout = model_builder.solve()

    # Assert
    if layout is None:  # handle infeasible instances
        return
    assert 1 == len({c.cable_type for c in layout})
```

SOME THOUGHTS

- Instances must be **sensible** (not all infeasible/trivial)
- Instances must be **small/fast** to solve
- LLMs are really good at boilerplate code
- Testing constraints individually ⇒ mostly tests the **modeling library**, not correctness
- Golden LP files do not prove correctness

PERFORMANCE TESTING

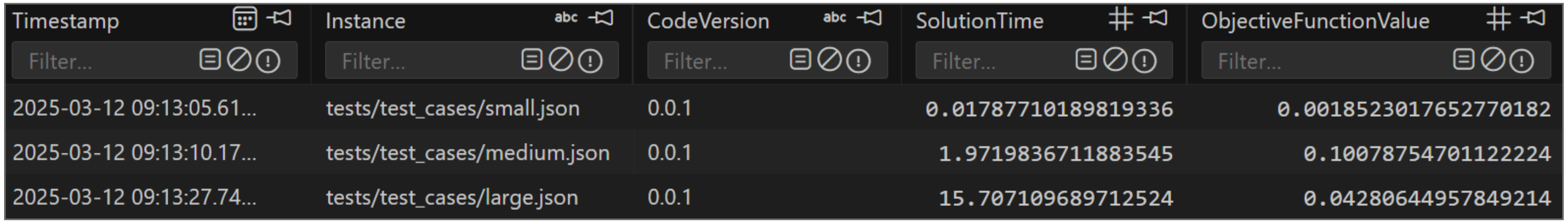
> Performance testing is (almost) as important as unit testing

CORE PRINCIPLE #3

Real instances, real hardware, real setup

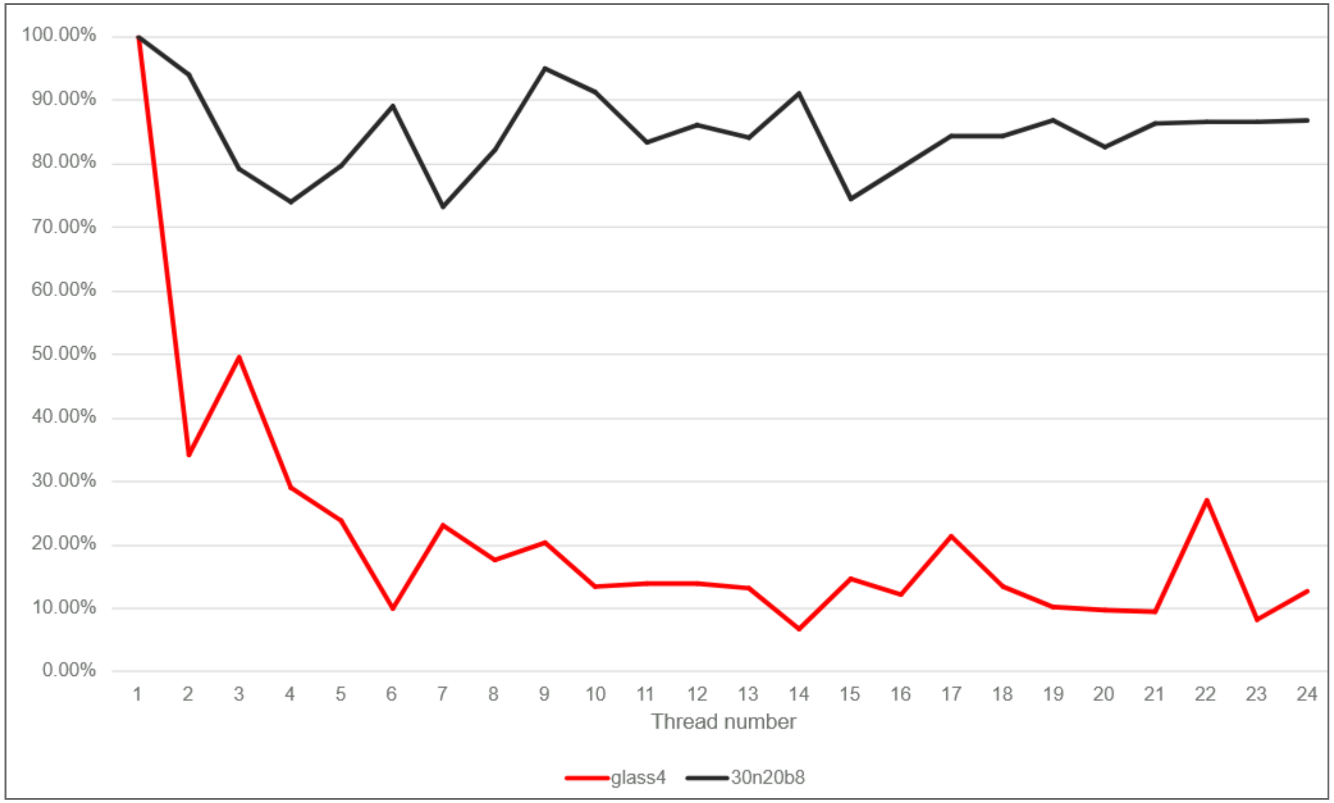
REAL INSTANCES

```
with sqlite3.connect("performance.db") as connection:
  connection.execute("""CREATE TABLE IF NOT EXISTS performance
    (Timestamp timestamp, Instance text, CodeVersion text,
      SolutionTime float, ObjectiveFunctionValue float)""")
```



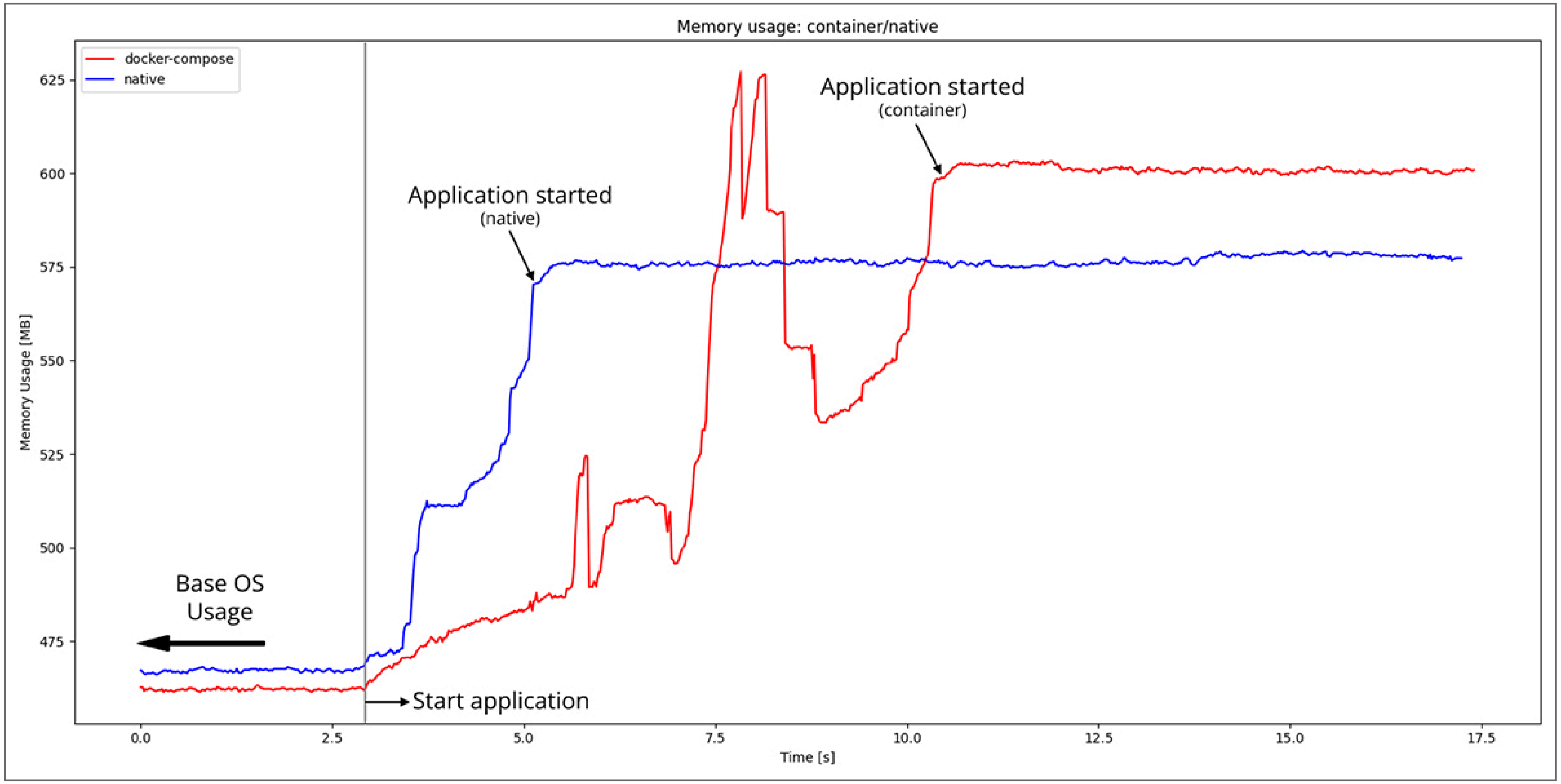
Timestamp	Instance	CodeVersion	SolutionTime	ObjectiveFunctionValue
Filter...	Filter...	Filter...	Filter...	Filter...
2025-03-12 09:13:05.61...	tests/test_cases/small.json	0.0.1	0.01787710189819336	0.0018523017652770182
2025-03-12 09:13:10.17...	tests/test_cases/medium.json	0.0.1	1.9719836711883545	0.10078754701122224
2025-03-12 09:13:27.74...	tests/test_cases/large.json	0.0.1	15.707109689712524	0.04280644957849214

REAL HARDWARE



Taken from "Tech Talk: Choosing the hardware that optimizes your Gurobi performance"

REAL SETUP


Memory usage: container/native
docker-compose
native
Memory Usage [MB]
625
600
575
550
525
500
475
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Time [s]
Application started
(container)
Application started
(native)
Base OS
Usage
Start application

CORE PRINCIPLES REVISITED

1. Move as much of the logic out of model-building as possible
2. A model encodes a business problem. Test solutions against business rules which don't know optimization, and should come from the business
3. Real instances, real hardware, real setup

TRUST BUT VERIFY

Speaker notes