Two Neat Decorators for Testing & Caching

Patrick Schemitz
Senior Scientist
solute GmbH

pytest

Tests with Multiple Values

```
def test_one():
    assert myfunc(1) == 1

def test_two():
    assert myfunc(2) == 4

def test_three():
    assert myfunc(3) == 9
```

Tests with Multiple Values

```
def test_one():
    assert myfunc(1) == 1

def test_two():
    assert my

def test_thre
    assert my
```

pytest.mark.parametrize

pytest supports multiple value tests:

```python
# content of test_expectation.py
import pytest
@pytest.mark.parametrize("test_input,expected", [
    ("3+5", 8),
    ("2+4", 6),
    ("6*9", 42),
])
def test_eval(test_input, expected):
    assert eval(test_input) == expected
```

“`test_input,expected`” redundant.

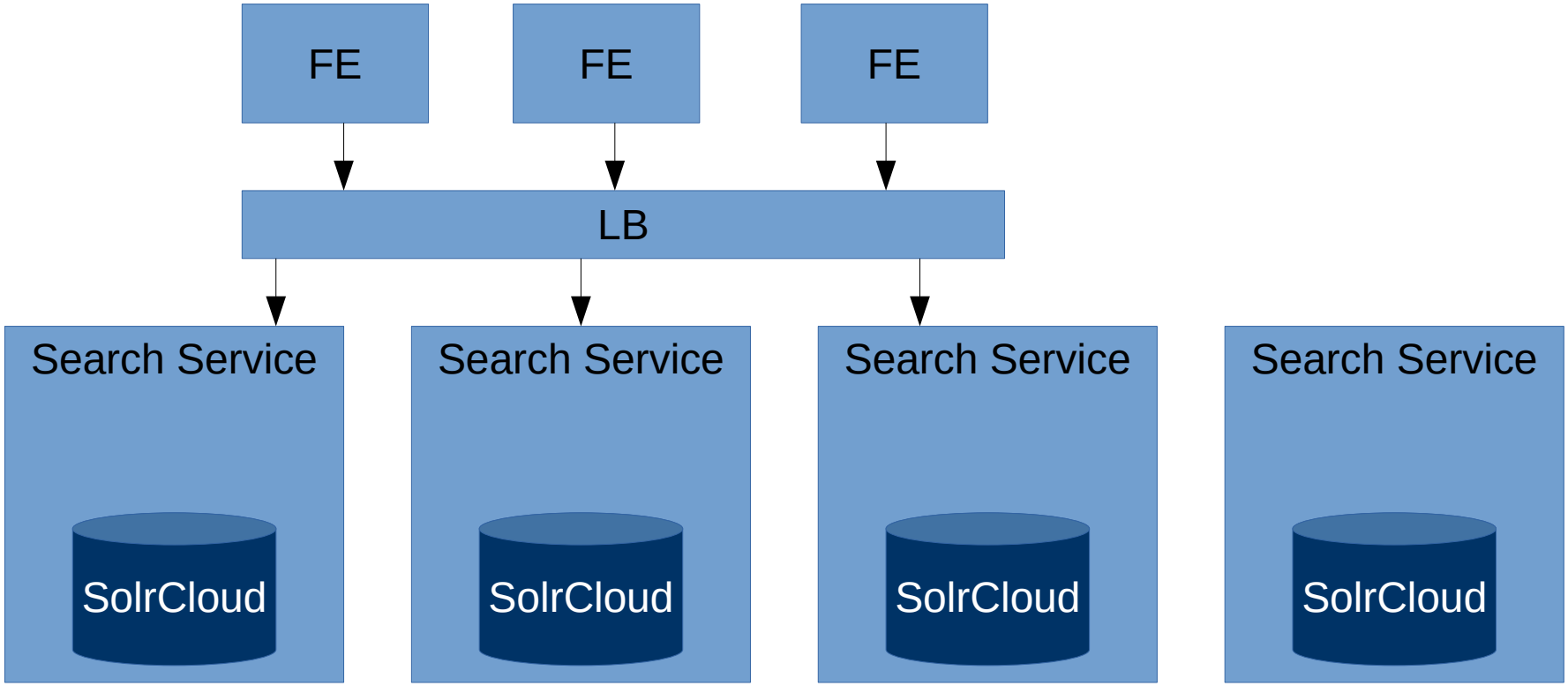
@multivalue_test

```
def multivalue_test(*values):
    def decorator(fn):
        args = fn.__code__.co_varnames[
            : fn.__code__.co_argcount]
        return pytest.mark.parametrize(args, values)(fn)
    return decorator


@multivalue_test([(1, 1), (2, 4), (3, 9)])
def test_n(value, expected):
    assert myfunc(value) == expected
```

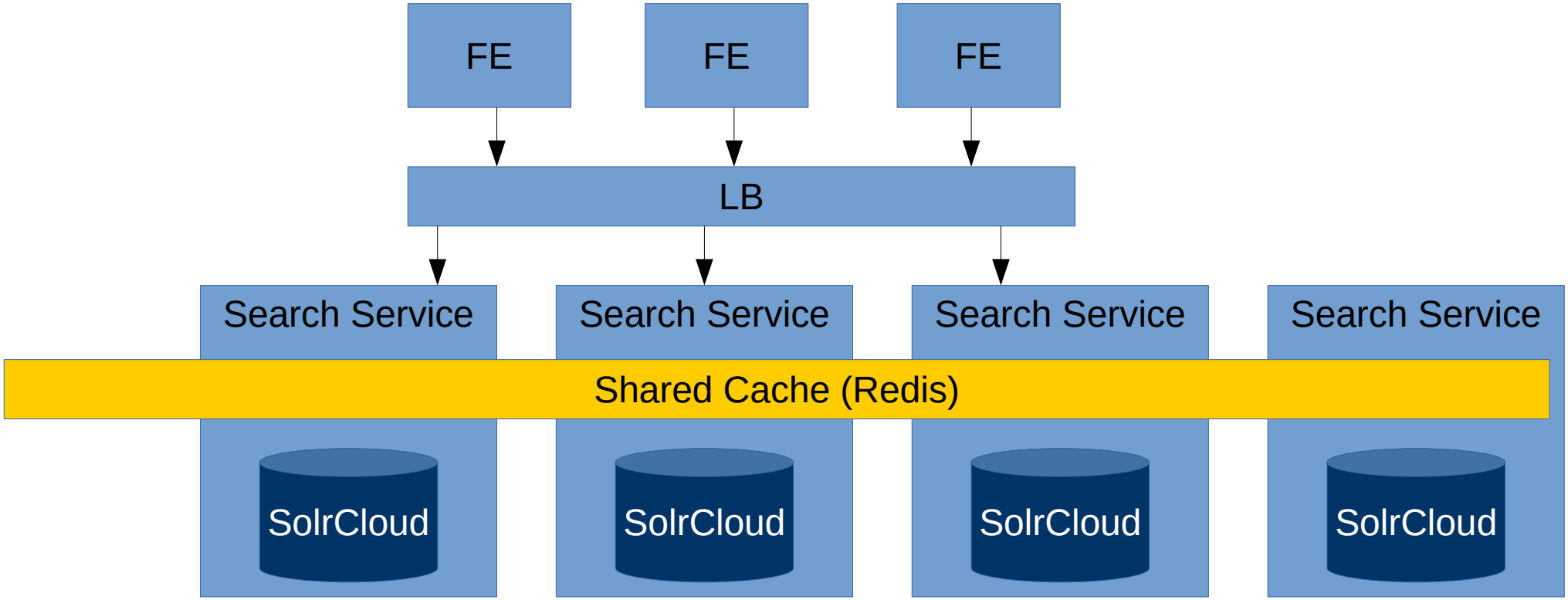
Caching Decorators

- Scenario: frontends accessing search servers through non-sticky loadbalancer; searches vary in duration (10ms .. 3000ms).



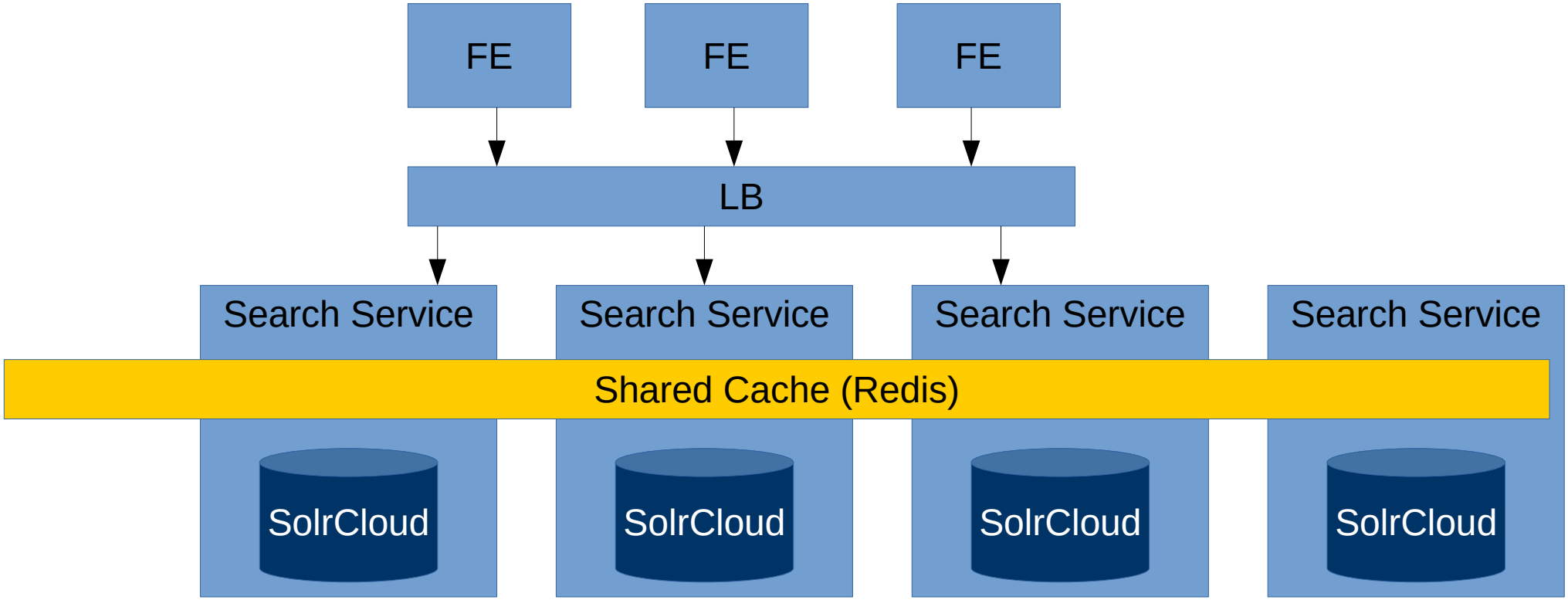
Caching Decorators

- Scenario: frontends accessing search servers through non-sticky loadbalancer; searches vary in duration (10ms .. 3000ms).



Caching Decorators

- Cache hits may be slower than a search!
  (But typically much faster.)
  Don’t waste cache space on fast queries.
  → Cache when it’s worth it!



@cache_when_worth_it

- Decorator that keeps track of costs for both cache misses *and* cache hits:

```
def my_cache_key(a, b, c):
  return "{!r}-{!r}-{!r}".format(a, b, c)

@cache_when_worth_it(my_cache_key)
def myfunc(a, b, c):
    result = 0
    for x in range(a):
        for y in range(b):
            for z in range(c):
                result += x*y*z
    return result
```

Neat Things to Remember

- time.monotonic()
- collections.deque.append() / deque.popleft()
- Arg names of fn:
  fn.__code__.co_varnames[: fn.__code__.co_argcount]
- __getitem__() / __setitem__() / KeyError protocol
  (also: __contains__())
- functools.wraps()
- Decorators can add attributes
  to decorated fn:
  fn.cache = CacheProxy(...)