

NPRG065: Programming in Python

Lecture 12

<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Tomas Bures

Petr Hnetynka

{bures, hnetynka}@d3s.mff.cuni.cz



CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Descriptors

- Descriptor ~ an object attribute with the methods
 - `__get__(self, instance, owner)`
 - `__set__(self, instance, value)`
 - `__delete__(self, instance)`

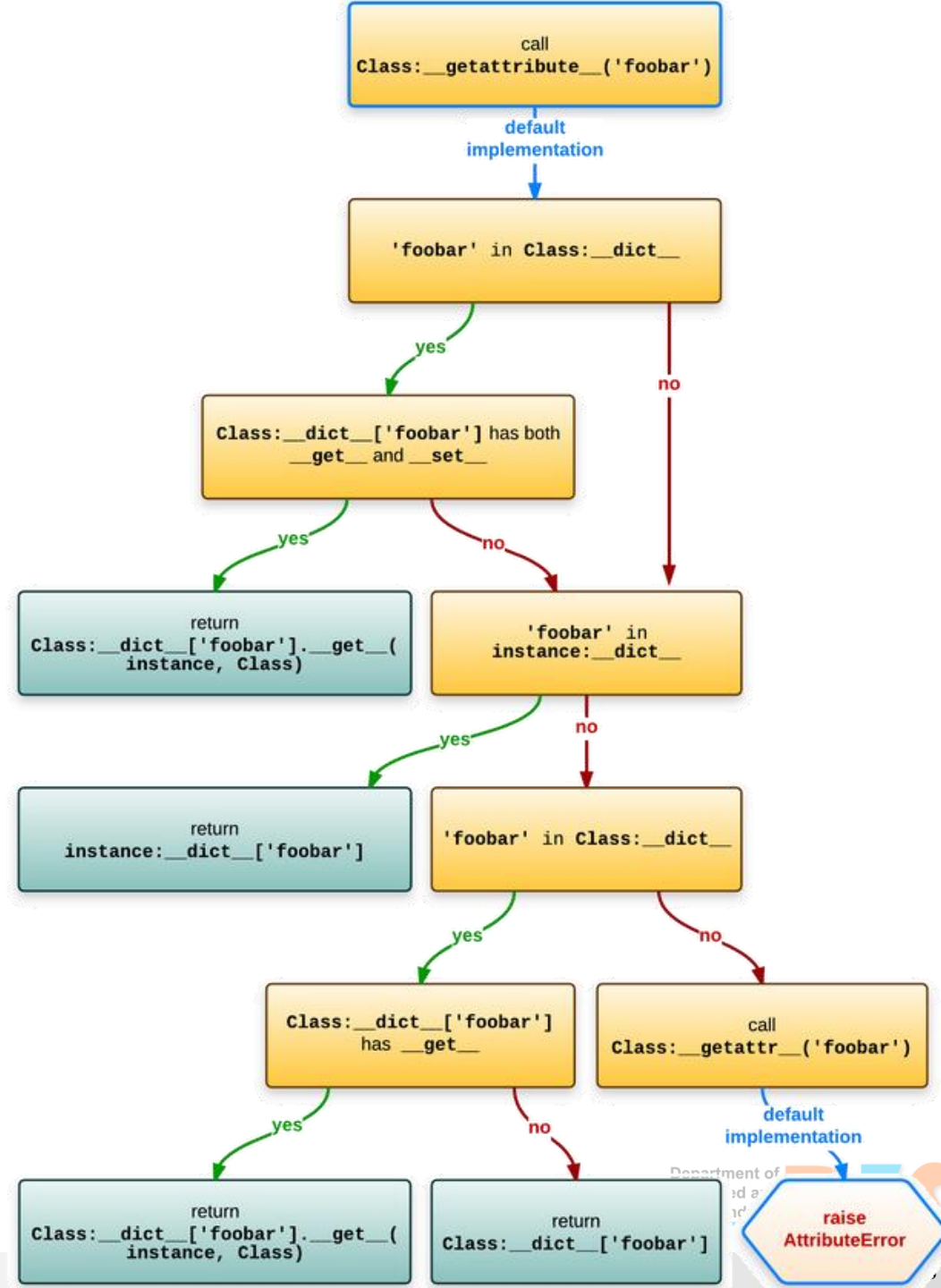
 - methods called when the attribute is accessed
- Compared to `__getattr__`, etc.
 - `__getattr__`, etc. defined on the class with attribute
 - `__get__`, etc. defined on the attribute's class

See
[descriptors.py](#)

Instance attribute lookup

```
class Class:  
    ...
```

```
instance = Class()  
instance.foobar
```



`__new__(cls[, ...])`

- the real “constructor” (create the object)
 - `__init__` only initializes the object, it does not create it
- a class method
- creates a new instance of class `cls`
- remaining arguments are those passed to the object constructor expression
- if `__new__()` returns an instance of `cls`, then the new instance's `__init__()` will be invoked like `__init__(self[, ...])`, where `self` is the new instance and the remaining arguments are the same as were passed to `__new__()`
- allows subclasses of immutable types (like `int`, `str`, or `tuple`) to customize instance creation

See
[new_immutable.py](#)

Metaclasses

- Factories for creating classes
- “Common” class definition

```
class Spam:  
    eggs = 'my eggs'
```

- Procedural definition via metaclass

```
Spam = type('Spam', (object,), dict(eggs='my eggs'))
```

- These two definitions are completely equivalent
 - In fact, Python transforms the first one into the second one
- **type** is a metaclass

See
[meta_basic.py](#)

Metaclasses

- Even in “common” definition, we can prescribe the metaclass

```
class Spam:  
    eggs = 'my eggs'
```

- is equivalent to

```
class Spam(metaclass=type) :  
    eggs = 'my eggs'
```

- We can define own metaclasses
 - as subclasses of **type**

See
[meta_basic_own.py](#)

Metaclasses

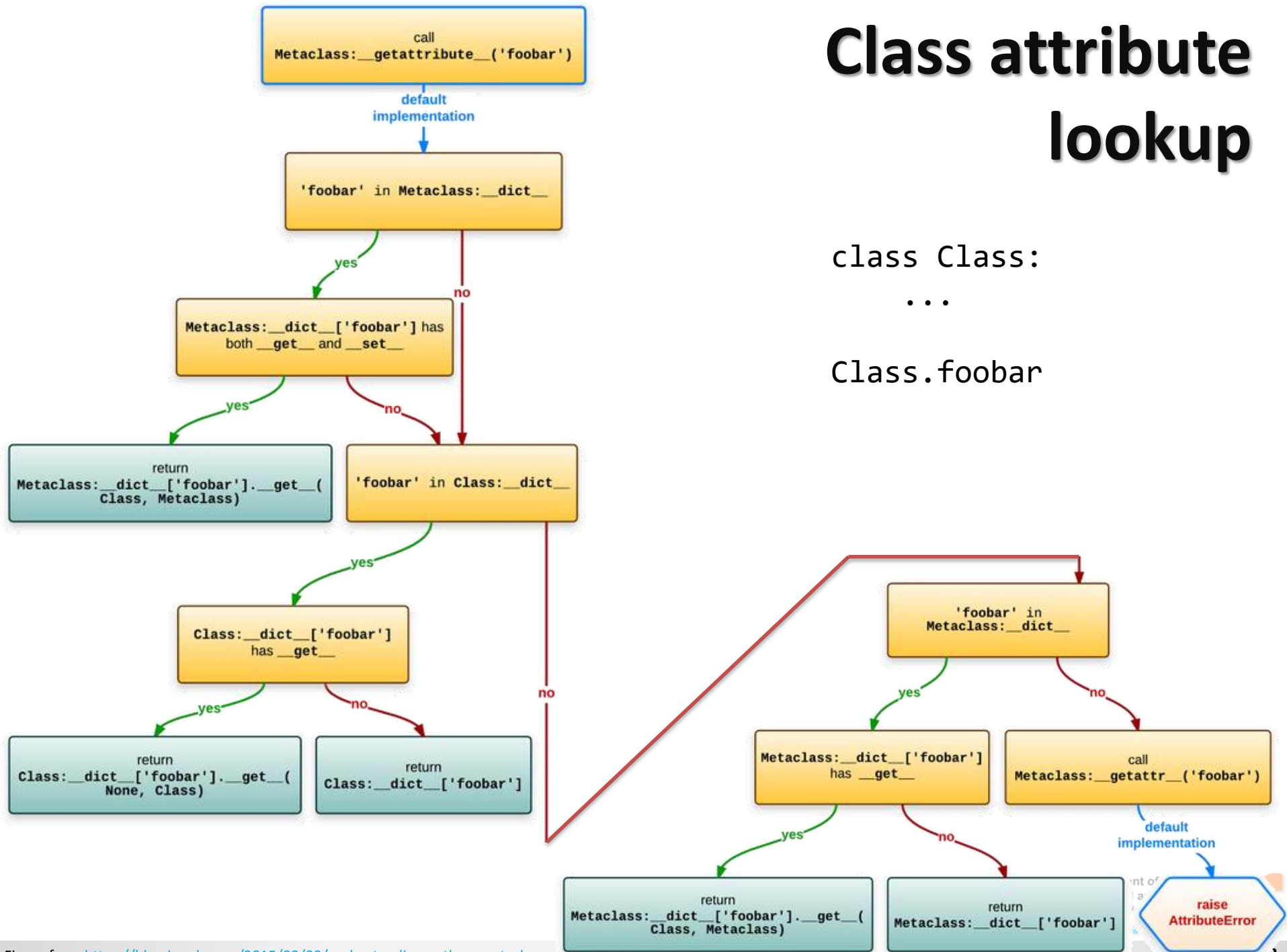
See
`meta_examples.py`
for more examples

Class attribute lookup

```
class Class:
```

```
...
```

```
Class.foobar
```



Metaclasses

- Metaclasses are used within the implementation of Abstract Base Classes (see lecture 8)

```
import abc

class PluginBase(abc.ABC):
    @abc.abstractmethod
    def process(self, input):
        pass

class ToUpperPlugin(PluginBase):
    def process(self, input):
        return input.upper()
```

- ABC class has ABCMeta metaclass
 - the following definition is equivalent

```
class PluginBase(metaclass=abc.ABCMeta):
    ...
```

See
[meta_abc.py](#)



The slides are licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).