Metaclasses

Robert Smallshire
COFOUNDER – SIXTY NORTH
@robsmallshire   rob@sixty-north.com

Summary

What is the class of a class object?

Default metaclass – `type`

Specifying a metaclass

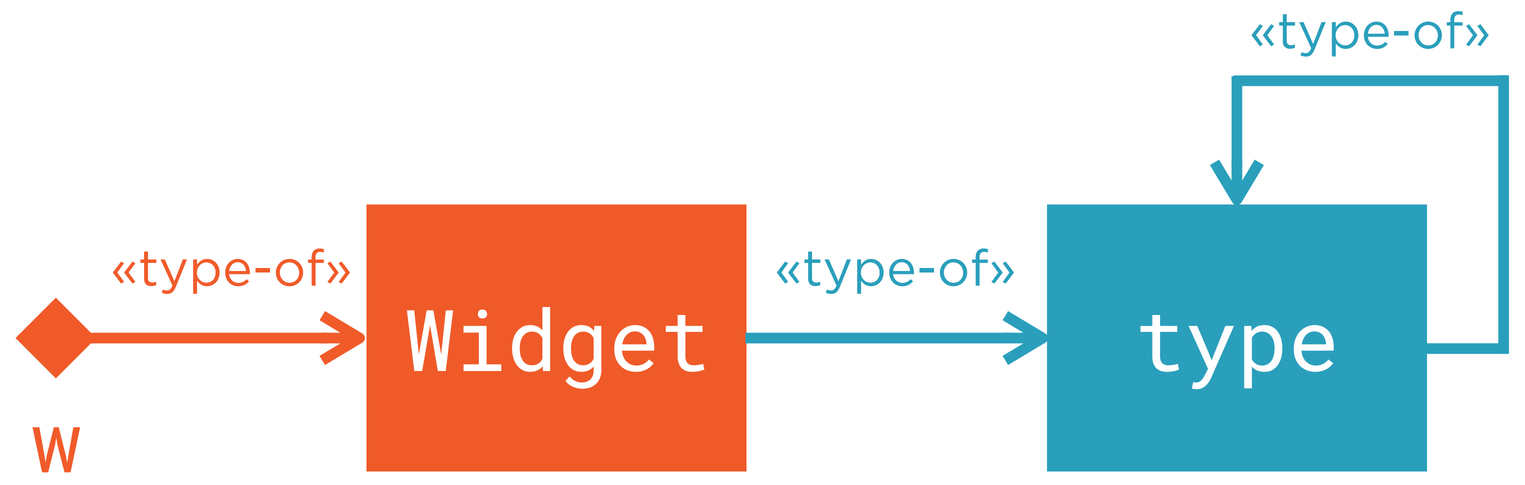
Defining a metaclass

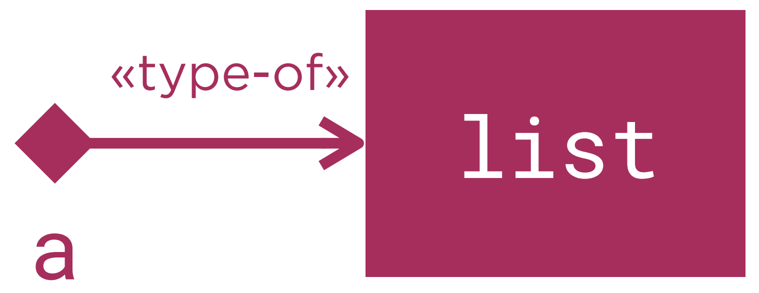
Special methods of metaclasses

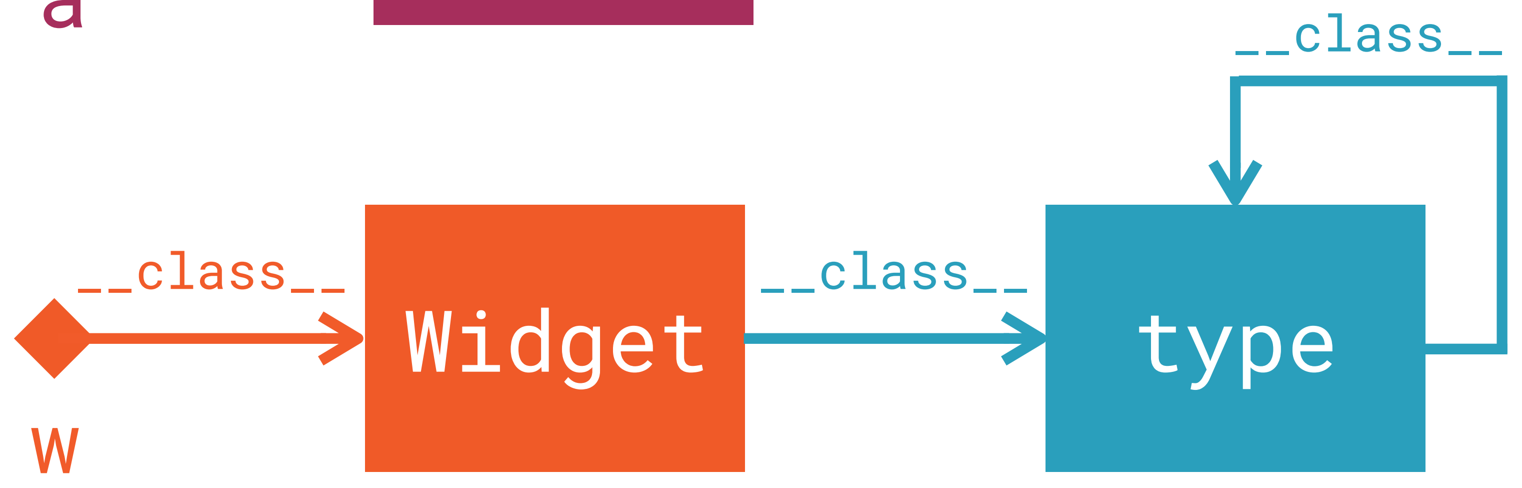
Practical uses

Metaclasses and inheritance

The Class of a Class







Class Definition

```python
class Widget:
    pass
```

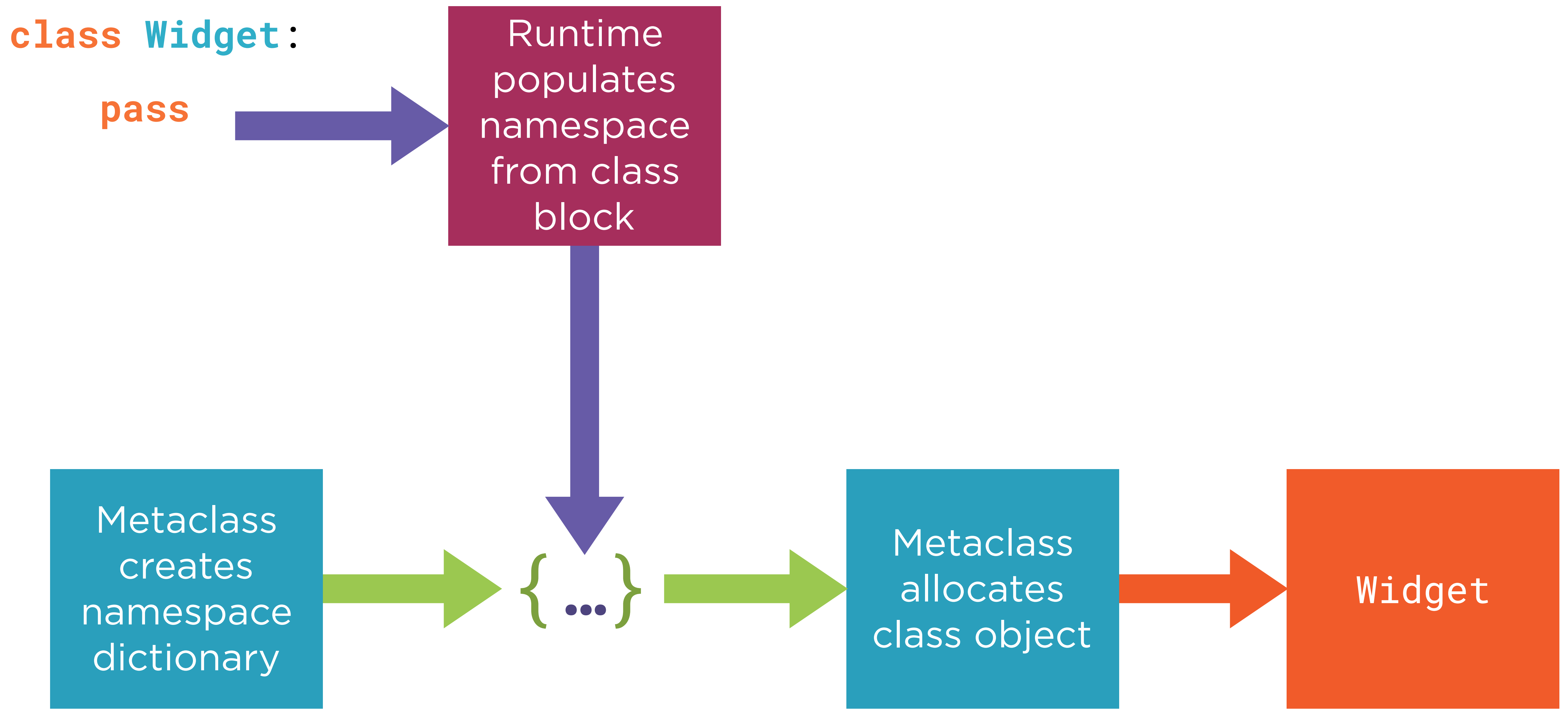
Default Base Class

Default Metaclass

```python
class Widget(object, metaclass=type):
    pass
```

Class Allocation and Initialisation

Class Definition


class Widget:
pass
Runtime populates namespace from class block
Metaclass creates namespace dictionary
{ ... }
Metaclass allocates class object
Widget

Class Definition

```
class Widget:
    pass
```

```
name = 'Widget'
metaclass = type
bases = ()
kwargs = {}
namespace = metaclass.__prepare__(name, bases, **kwargs)
Widget = metaclass.__new__(metaclass, name, bases, namespace, **kwargs)
metaclass.__init__(Widget, name, bases, namespace, **kwargs)
```

Which Metaclass Methods to Override?

`__prepare__`	`__new__`	`__init__`
Customise the type or initial value of the namespace mapping	Allocate and optionally configure new class object	Configure class object

Metaclass Keyword Arguments

Class Definition

positional arguments
base classes

keyword arguments
forwarded to metaclass

```
class Widget(object, metaclass=type, more=1, keyword=2, args=3):
    pass
```

metaclass keyword
specifies metaclass

Class Statement as a Class Factory

keyword arguments
forwarded to metaclass

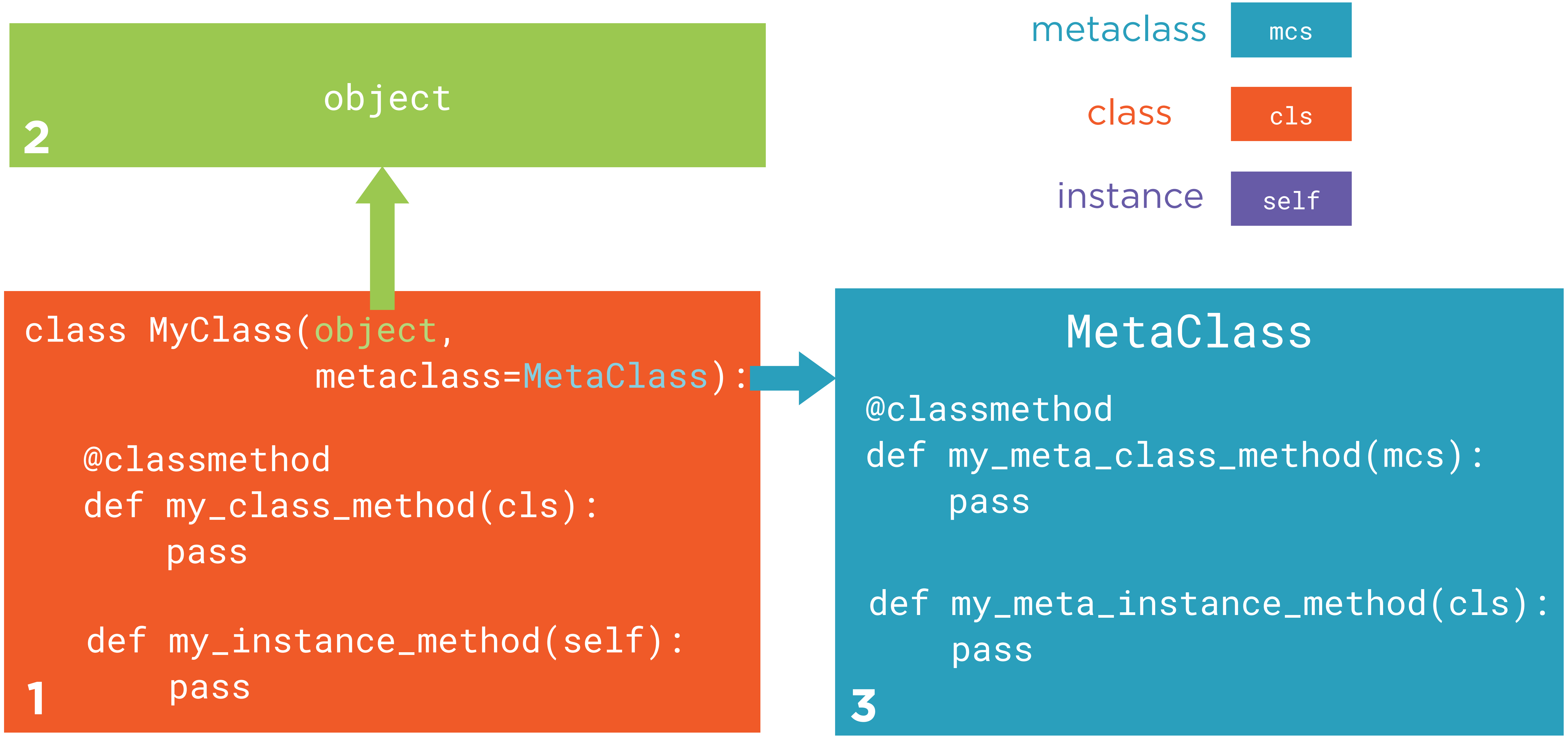
```
class Widget(object, metaclass=type, more=1, keyword=2, args=3):
    pass
```

Runtime parameterisation of class construction –

a **class factory**

Metaclass Method Visibility

Method Arguments

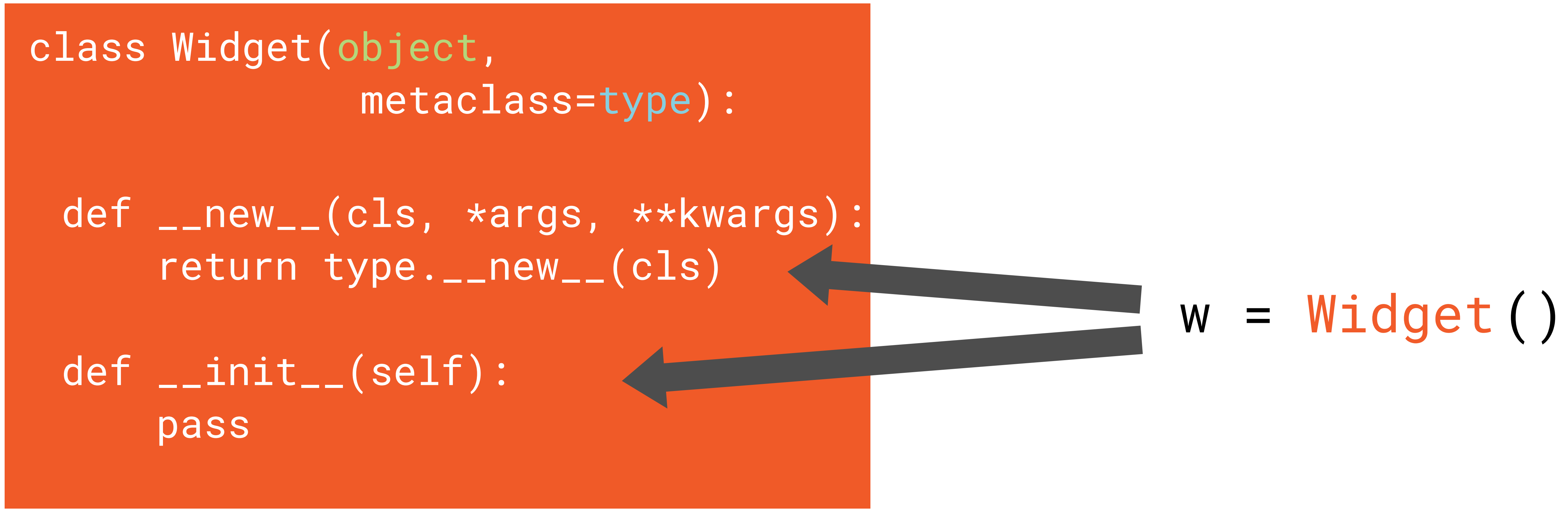


```
class MyClass(object,
              metaclass=MetaClass):

   @classmethod
   def my_class_method(cls):
       pass

   def my_instance_method(self):
       pass
```

```
@classmethod
def my_meta_class_method(mcs):
    pass

def my_meta_instance_method(cls):
    pass
```

Metaclass `__call__` : The Instance Constructor

The Instance Constructor



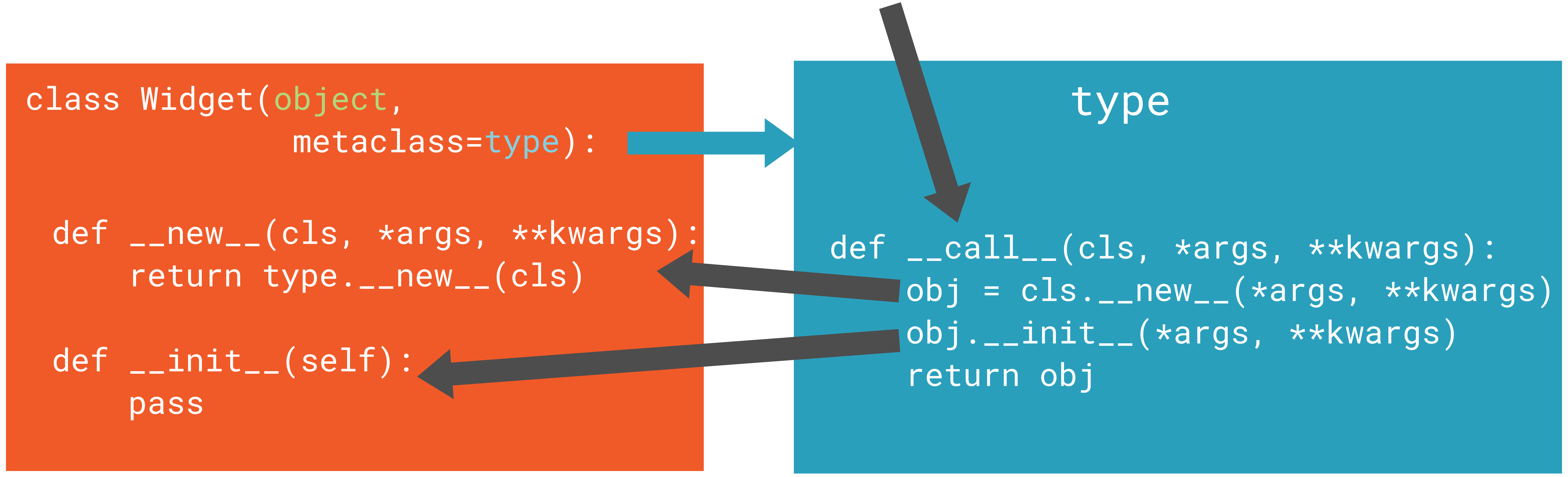
```python
class Widget(object,
             metaclass=type):

  def __new__(cls, *args, **kwargs):
      return type.__new__(cls)

  def __init__(self):
      pass
```

```python
w = Widget()
```

Metaclass __call__ : The Instance Constructor

Calling the regular class invokes `metaclass.__call__()`

```
w = Widget()
```



```
class Widget(object,
             metaclass=type):

  def __new__(cls, *args, **kwargs):
      return type.__new__(cls)

  def __init__(self):
      pass
```

```
def __call__(cls, *args, **kwargs):
    obj = cls.__new__(*args, **kwargs)
    obj.__init__(*args, **kwargs)
    return obj
```

`metaclass.__call__()` invokes regular class `__new__()` and `__init__()`

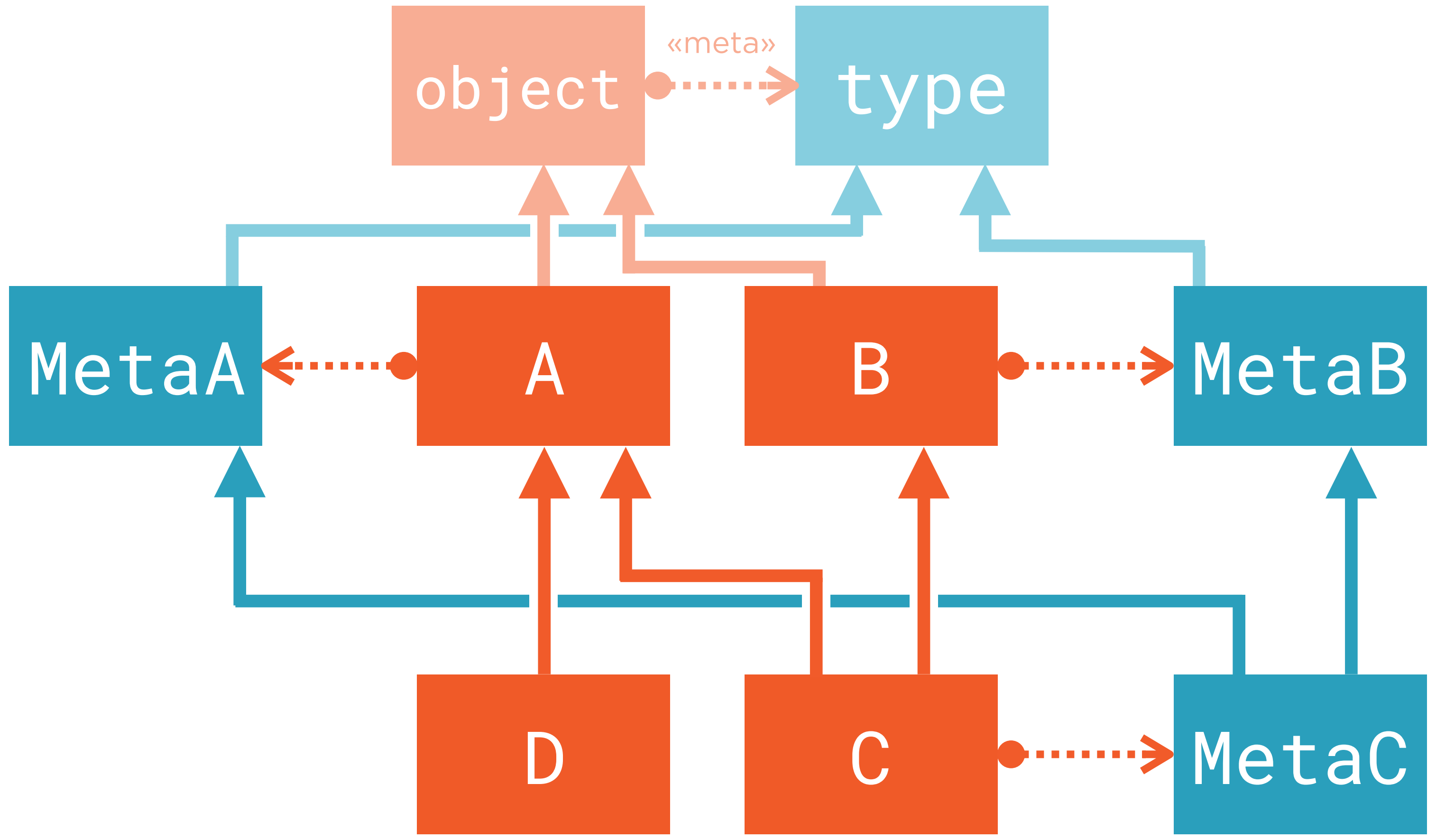
A Practical Metaclass Example

Idea!

Use a namespace dictionary which forbids re-assignment to existing keys

Naming Descriptors Using Metaclasses

Metaclasses and Inheritance


object
«meta»
type
MetaA
A
B
MetaB
D
C
MetaC

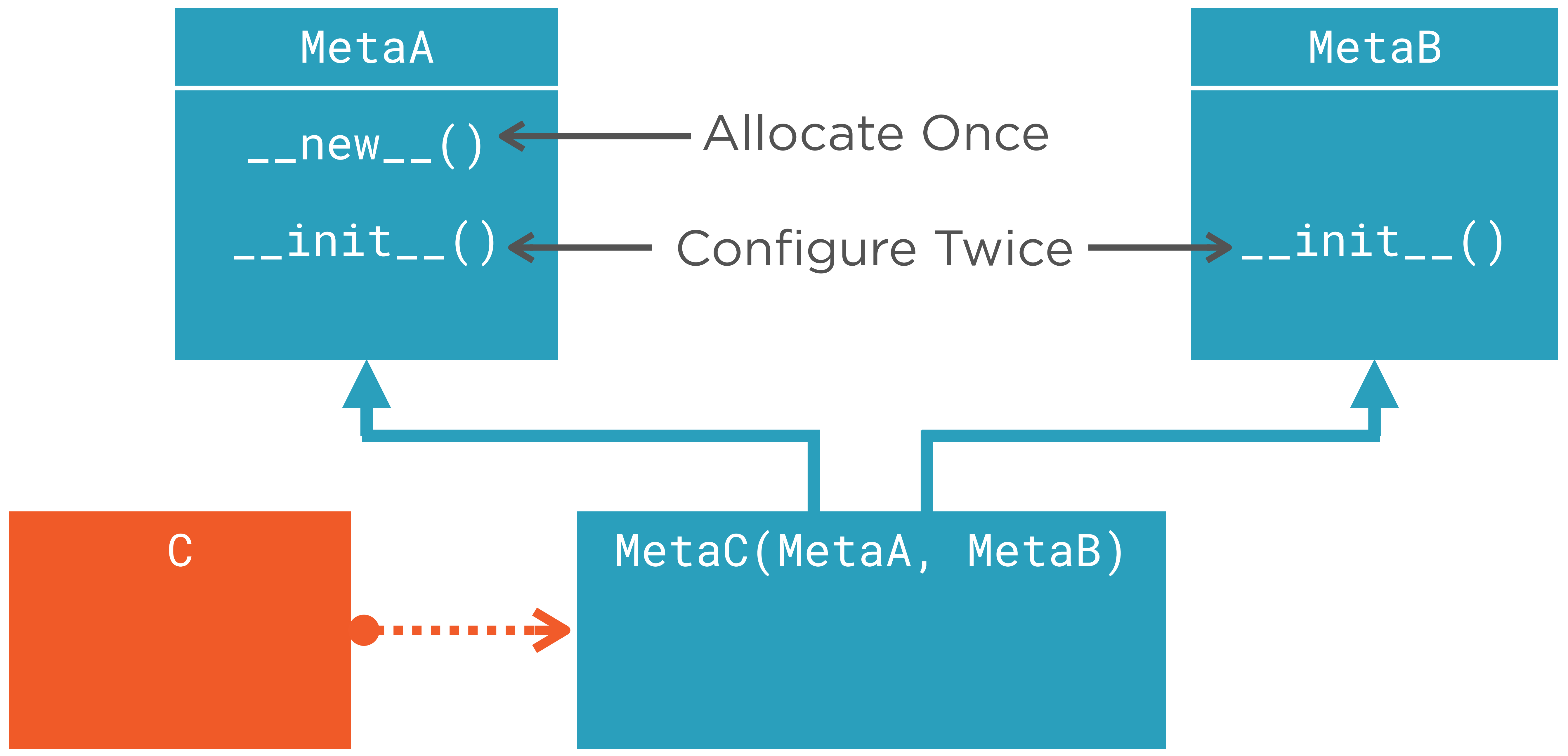
Cooperative Metaclasses

```python
class ProhibitDuplicatesMeta(type):

    @classmethod
    def __prepare__(mcs, name, bases):
        return OneShotClassNamespace(name)
```

```python
class KeywordsOnlyMeta(type):

    def __call__(cls, *args, **kwargs):
        if args:
            raise TypeError(
             "Constructor for class {!r} does "
             "not accept positional arguments.".format(cls))
        return super().__call__(cls, **kwargs)
```

```python
class ProhibitDuplicatesAndKeyWordsOnlyMeta(
    ProhibitDuplicatesMeta,
    KeyWordsOnlyMeta):
pass
```

Prefer `__init__()` to `__new__()` for Configuration



Use **`super()`** diligently for composable metaclasses

Summary

All classes have a metaclass

The default metaclass is `type`

Metaclasses convert parsed class namespaces into a class objects

Summary

`__prepare__()` **must return a mapping to hold the namespace contents**

`__new__()` **must return a class object**

`__init__()` **can configure a class object**

`__call__()` **on metaclasses is the instance constructor**

Summary

Metaclasses can be used to implement so-called named descriptors

Strict rules control the interaction of metaclasses with inheritance

Use `super()` wisely for cooperative metaclasses