single inheritance

```
class SubClass(BaseClass)
```

- Subclasses will want to initialize base classes
- Base class initializer will only be called automatically if subclass initializer is undefined

Calling base class initializer

- Other languages automatically call base class initializers
- Python treats `__init__()` like any other method
- Base class `__init__()` is not called if overridden
- Use `super()` to call base class `__init__()`

Multiple inheritance in Python is not much more complex than single inheritance.

isinstance()

determines if an object is of a specified type

`isinstance()`

determines if an object is of a specified type

Use `isinstance()` for runtime type checking

`issubclass()`

determines if one type is a subclass of another

multiple inheritance

defining a class with more than one base class

```
class SubClass(Base1, Base2, . . .)
```

Multiple inheritance

- Subclasses inherit methods of all bases
- Without conflict, names resolve in the obvious way
- Method Resolution Order (MRO) determines name lookup in all cases

If a class

A. has **multiple** base classes

B. defines **no initializer**

then **only** the initializer of the **first** base class is automatically called

__bases__

a tuple of base classes

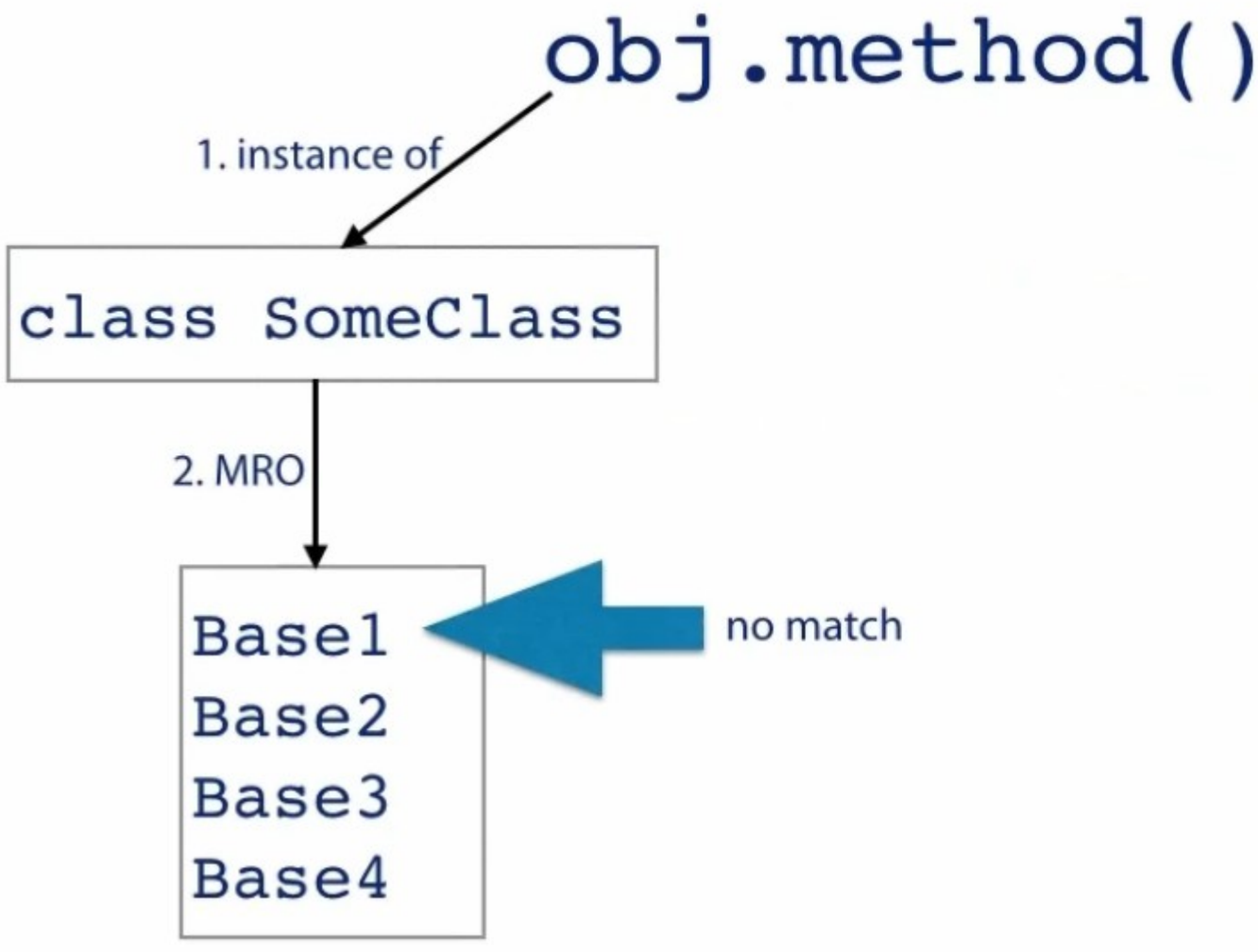
method resolution order

ordering that determines method name lookup

- Commonly called “MRO”
- Methods may be defined in multiple places
- MRO is an ordering of the inheritance graph
- Actually quite simple

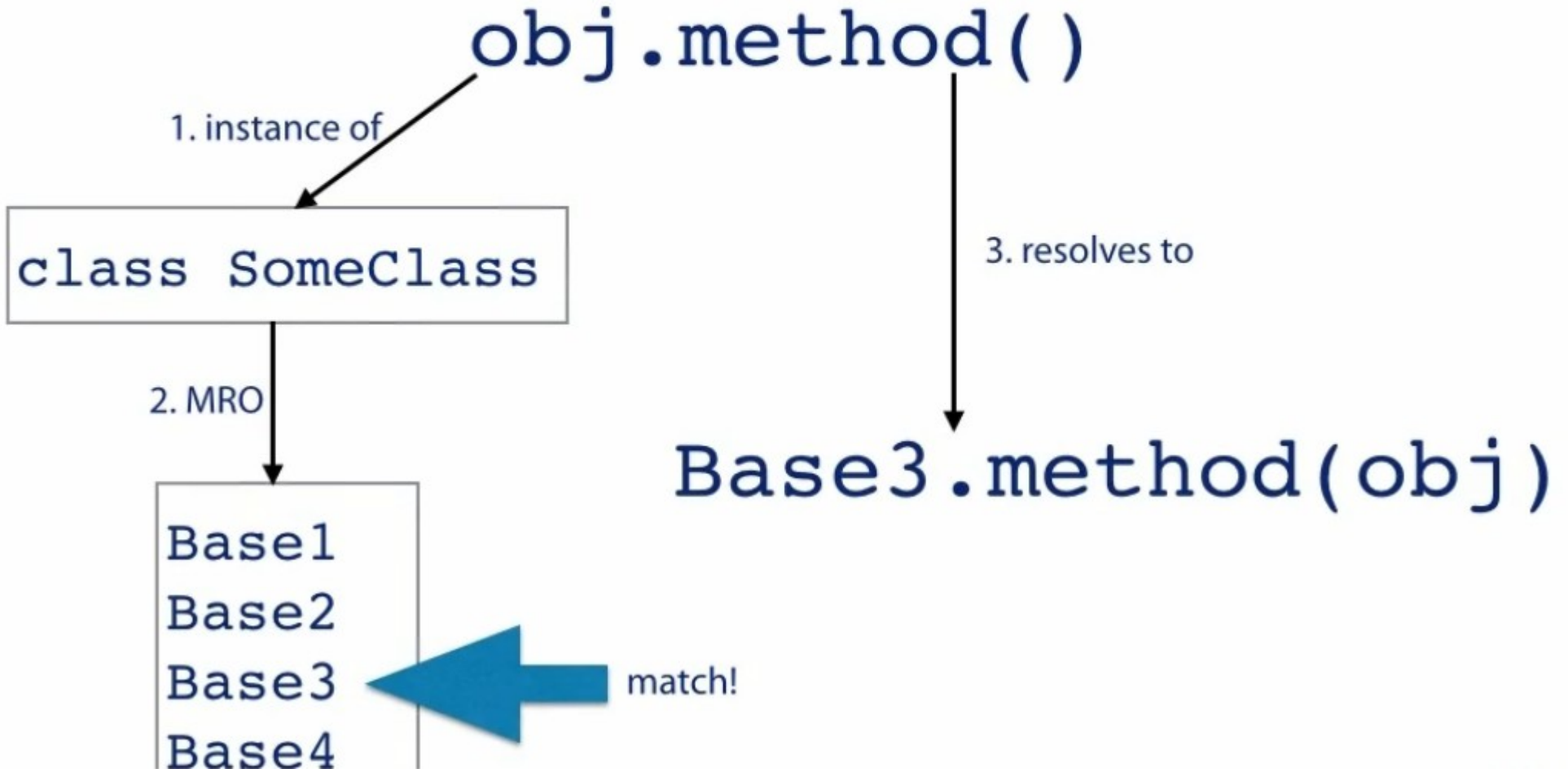
python™

How is MRO used?



python™

How is MRO used?



C3

Not all inheritance declarations are **allowed!**

algorithm for calculating MRO in Python

- Subclasses come before base classes
- Base class order from class definition is preserved
- First two qualities are preserved no matter where you start in the inheritance graph

super()

Given a ***method resolution order*** and a class C, `super()` gives you an object which resolves methods using only the part of the ***MRO*** which comes after C.

`super()` **returns a proxy object which routes method calls.**

Bound proxy

bound to a specific class or instance

Unbound proxy

not bound to a class or instance

There are two types of bound proxies:

instance-bound

and

class-bound

python™

Class-bound proxy

subclass of
first argument

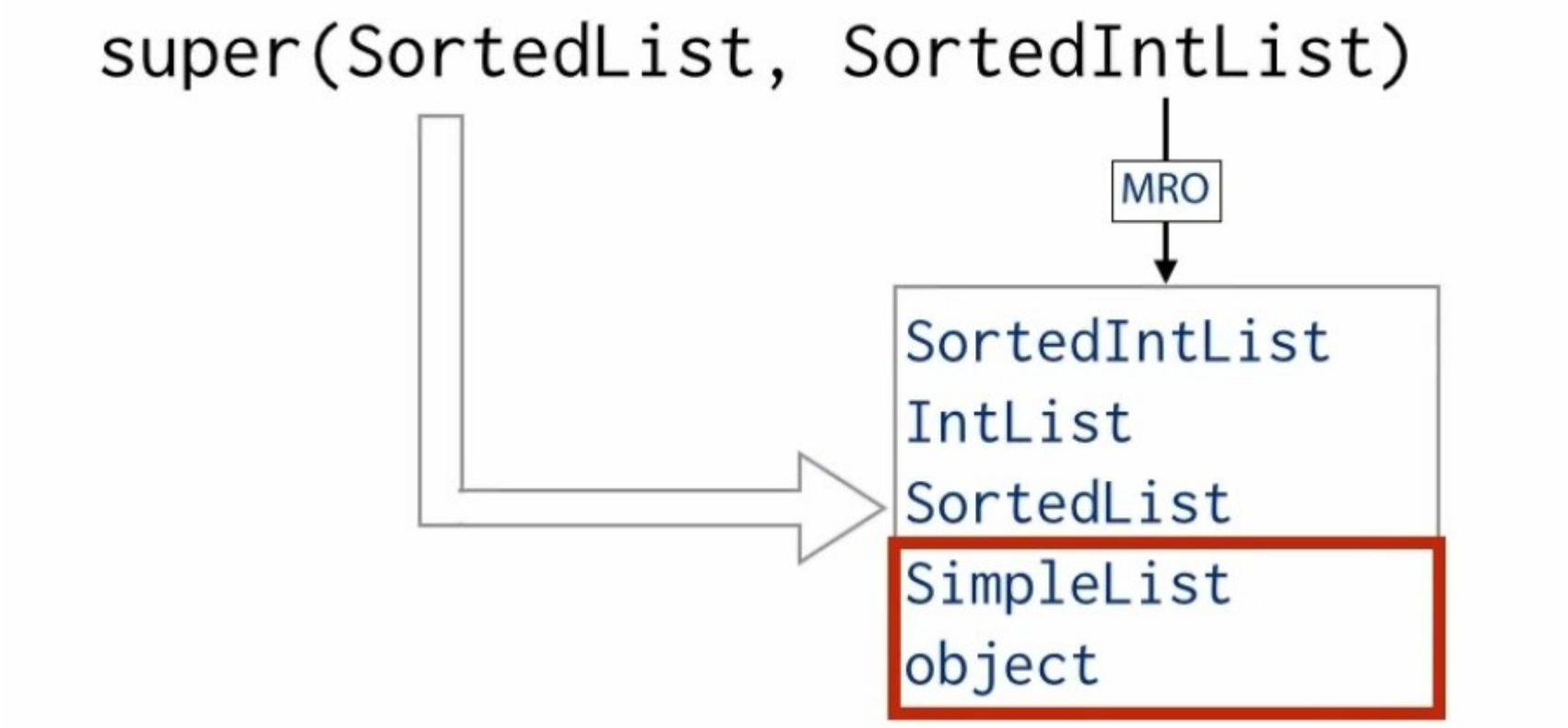
```
super(base-class, derived-class)
```

class object

Class-bound proxy

`super(base-class, derived-class)`

- Python finds MRO for `derived-class`
- It then finds `base-class` in that MRO
- It take everything ***after*** `base-class` in the MRO, and finds the first class in that sequence with a matching method name


super(SortedList, SortedIntList)
MRO
SortedIntList
IntList
SortedList
SimpleList
object

python™

Instance-bound proxy

instance of first argument

```
super(class, instance-of-class)
```

class object

Instance-bound proxy

`super(class, instance-of-class)`

- Finds the MRO for the type of the second argument
- Finds the location of the first argument in the MRO
- Uses everything ***after*** that for resolving methods

super()

instance method

```
super(class-of-method, self)
```

class method

```
super(class-of-method, class)
```