Module 22

Subclasses & Inheritance

An Application

- **Goal**: Presentation program (e.g. PowerPoint)
- **Problem**: There are many types of content
  - **Examples**: text box, rectangle, image, etc.
  - Have to write code to display each one
- **Solution**: Use object oriented features
  - Define class for every type of content
  - Make sure each has a `draw` method:

```
for x in slide[i].contents:
    x.draw(window)
```

Sharing Work

- These classes will have a lot in common

  - Drawing handles for selection

  - Background and foreground color

  - Current size and position

  - And more (see the formatting bar in PowerPoint)

- **Result**: A lot of repetitive code

- **Solution**: Create one class with shared code

  - All content are *subclasses* of the *parent* class

Defining a Subclass

```
class SlideContent(object):
    """Any object on a slide."""
    def __init__(self, x, y, w, h): ...
    def draw_frame(self): ...
    def select(self): ...


class TextBox(SlideContent):
    """An object containing text."""
    def __init__(self, x, y, text): ...
    def draw(self): ...


class Image(SlideContent):
    """An image."""
    def __init__(self, x, y, image_file): ...
    def draw(self): ...
```

Superclass
Parent class
Base class

Subclass
Child class
Derived class

SlideContent

TextBox   Image

SC

__init__(self,x,y,w,h)
draw_frame(self)
select(self)

TextBox(SC)

__init__(self,x,y,text)
draw(self)

Image(SC)

__init__(self,x,y,img_f)
draw(self)

Subclasses & Inheritance

Class Definition: Revisited

```
class <name>(<superclass>):

    """Class specification"""

    getters and setters

    initializer (__init__)

    definition of operators

    definition of methods

    anything else
```
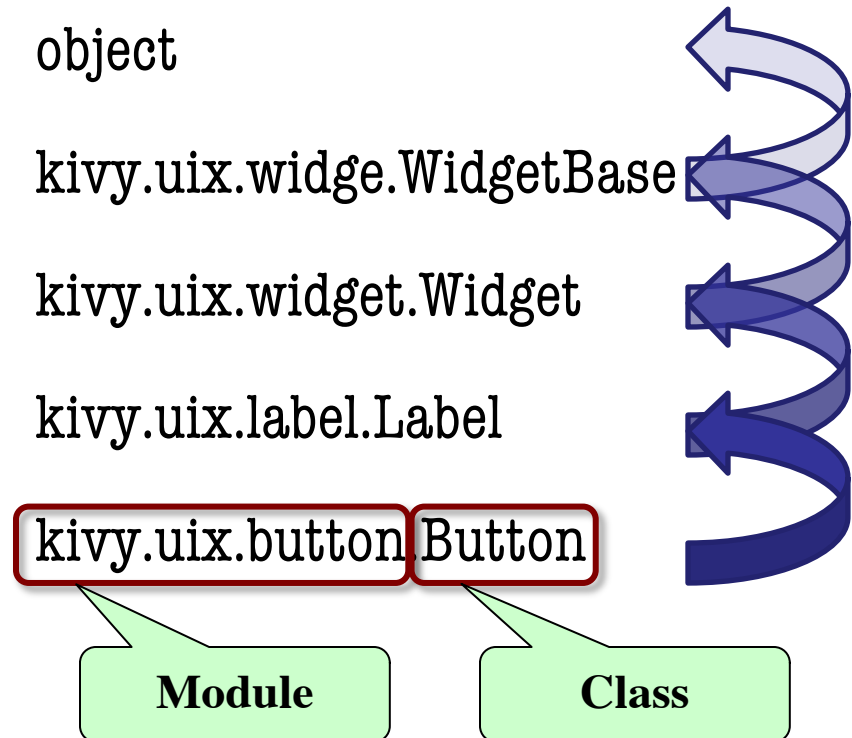
Class type to extend
(may need module name)

- Every class must extend *something*
- Previous classes all extended object

object and the Subclass Hierarcy

- Subclassing creates a **hierarchy** of classes
  - Each class has its own super class or parent
  - Until object at the "top"
- object has many features
  - Special built-in fields: `__class__`, `__dict__`
  - Special built-in methods: `__str__`, `__repr__`

Kivy Example

object

kivy.uix.widge.WidgetBase

kivy.uix.widget.Widget

kivy.uix.label.Label

kivy.uix.button Button

Module   Class

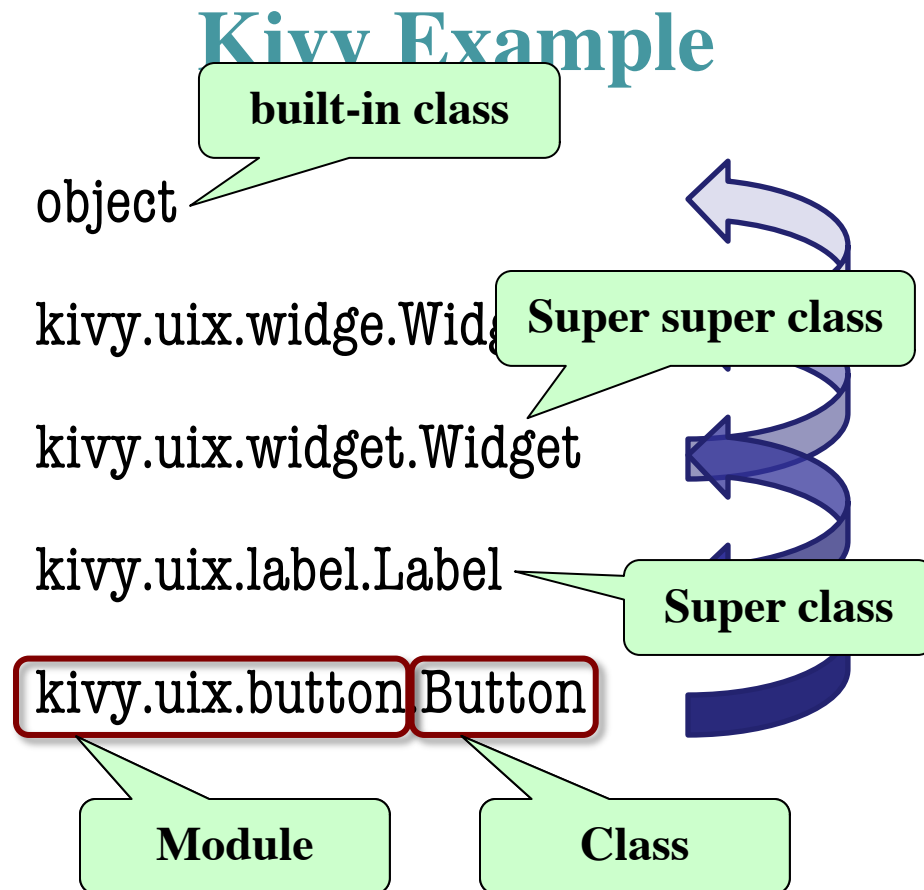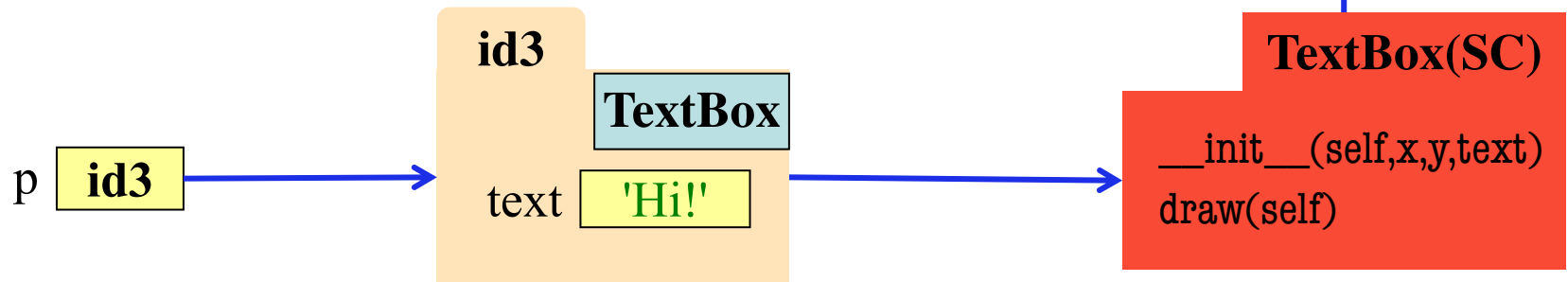# object and the Subclass Hierarcy

- Subclassing creates a **hierarchy** of classes
  - Each class has its own super class or parent
  - Until object at the "top"
- object has many features
  - Special built-in fields: `__class__`, `__dict__`
  - Special built-in methods: `__str__`, `__repr__`

Kivy Example

built-in class

object

kivy.uix.widge.Wid

Super super class

kivy.uix.widget.Widget

kivy.uix.label.Label

Super class

kivy.uix.button Button

Module

Class

Name Resolution Revisited

- To look up attribute/method name
  1. Look first in instance (object folder)
  2. Then look in the class (folder)
- Subclasses add two more rules:
  3. Look in the superclass
  4. Repeat 3. until reach `object`

object

????

SC(object)

__init__(self,x,y,w,h)

draw_frame(self)

select(self)

id3

TextBox

text | 'Hi!'

p | **id3**

TextBox(SC)

__init__(self,x,y,text)
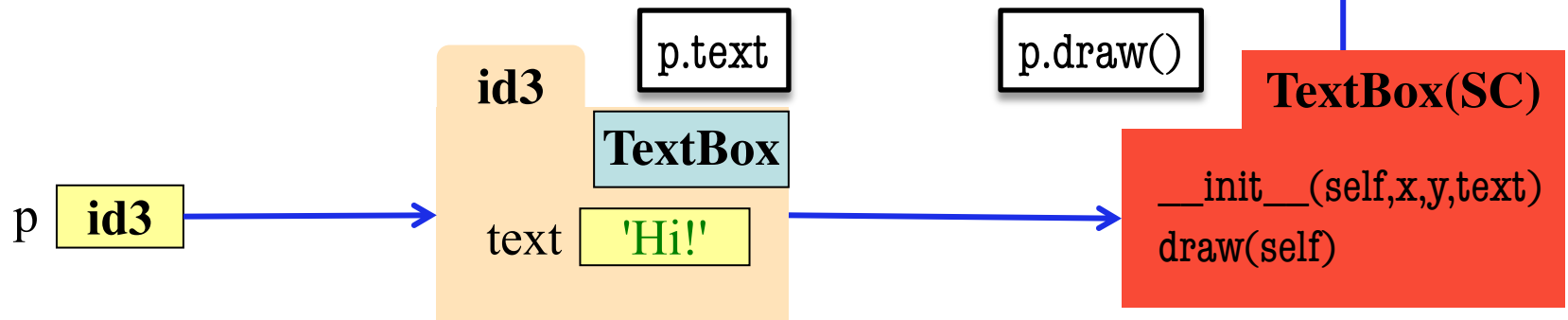
draw(self)

Name Resolution Revisited

- To look up attribute/method name
  1. Look first in instance (object folder)
  2. Then look in the class (folder)
- Subclasses add two more rules:
  3. Look in the superclass
  4. Repeat 3. until reach `object`

object

????

p.select()

SC(object)

__init__(self,x,y,w,h)

draw_frame(self)

select(self)

p.text

id3

TextBox

text 'Hi!'

p id3

p.draw()

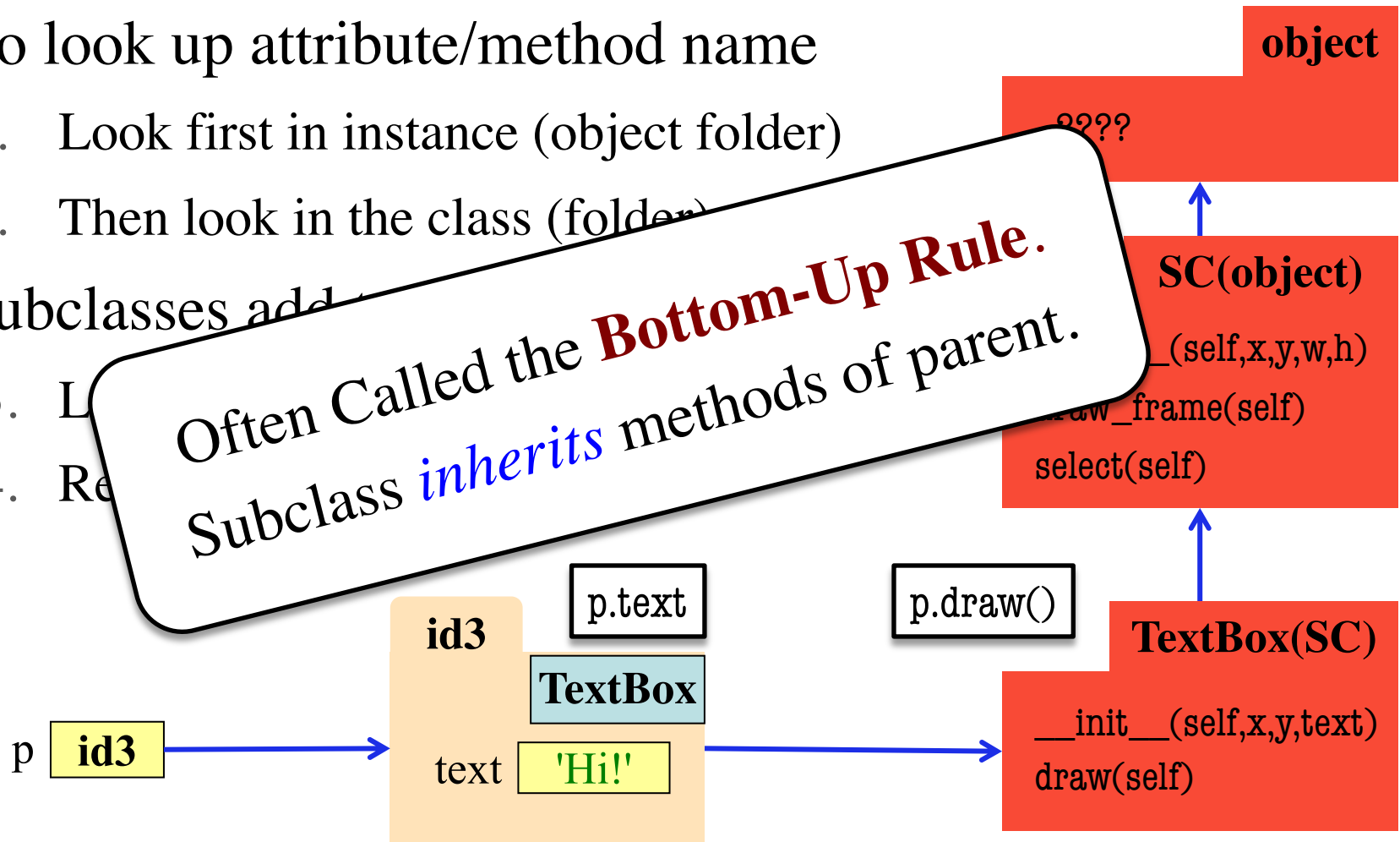**TextBox(SC)**

__init__(self,x,y,text)

draw(self)

Name Resolution Revisited

- To look up attribute/method name
  1. Look first in instance (object folder)
  2. Then look in the class (folder)

- Subclasses add

  3. L
  4. R

*Often Called the **Bottom-Up Rule**.*
Subclass inherits methods of parent.

object

????

SC(object)

__init__(self,x,y,w,h)
draw_frame(self)
select(self)

TextBox(SC)

__init__(self,x,y,text)
draw(self)

p | **id3**

id3

TextBox

text | 'Hi!'

p.text

p.draw()

Also Works With Class Attributes

Class Attribute: Assigned outside of any method definition

```python
class Employee(object):
    """Instance is salaried worker"""
    # Class Attribute
    STD_SALARY = 50000.0
```

```python
class Executive(Employee):
    """An Employee with a bonus."""
    # Class Attribute
    STD_BONUS = 10000.0
```
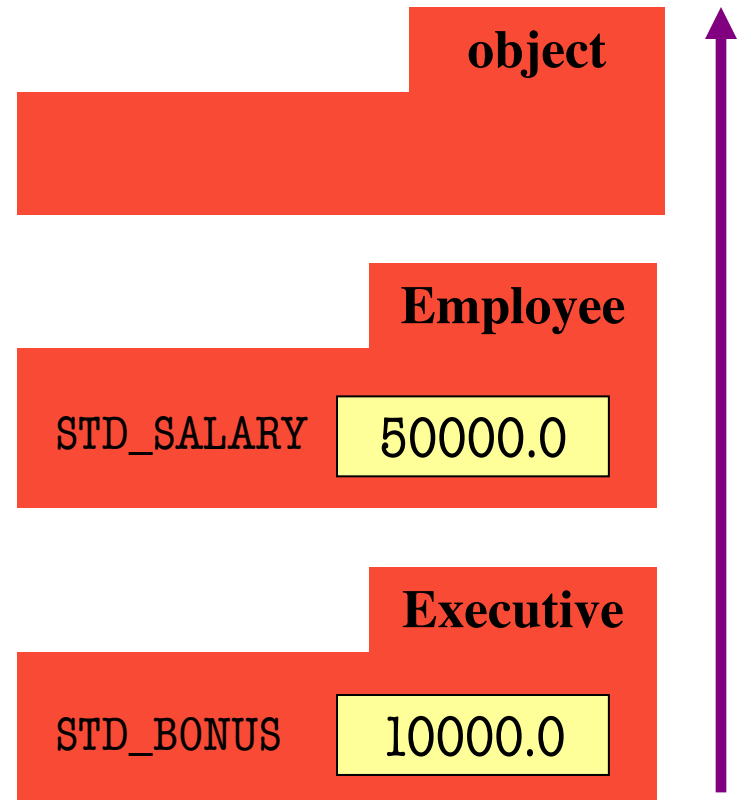
object

Employee

STD_SALARY   50000.0

Executive

STD_BONUS   10000.0

A Simpler Example

```
class Employee(object):
    """Instance is salaried worker"""
    # INSTANCE ATTRIBUTES:
    # _name:  full name, a string
    # _start:   first year hired,
    #    an int ≥ -1, -1 if unknown
    # _salary: yearly wage, a float


class Executive(Employee):
    """An Employee with a bonus"""
    # INSTANCE ATTRIBUTES:
    #_bonus: annual bonus, a float
```

object

__init__(self)
__str__(self)
__repr__(self)

Employee

__init__(self,n,d,s)
__str__(self)
__repr__(self)

Executive

__init__(self,n,d,b)
__str__(self)
__repr__(self)

A Simpler Example

```
class Employee(object):
    """Instance is salaried worker"""
    # INSTANCE ATTRIBUTES:
    # _name:  full name, a string
    # _start:   first year hired,
    #    an int ≥ -1, -1 if unknown
    # _salary: yearly wage, a float


class Executive(Employee):
    """An Employee with a bonus"""
    # INSTANCE ATTRIBUTES:
    #_bonus: annual bonus, a float
```

object

__init__(self
__str__(self)
__repr__(self

> All double underscore methods are in class object

Employee

__init__(self,n,d,s)
__str__(self)
__repr__(self)

Executive

__init__(self,n,d,b)
__str__(self)
__repr__(self)

Method Overriding

- Which __str__ do we use?
  - Start at bottom class folder
  - Find first method with name
  - Use that definition
- New method definitions **override** those of parent
  - Access to old version is **lost**
  - New version used instead
  - **Example**: __init__
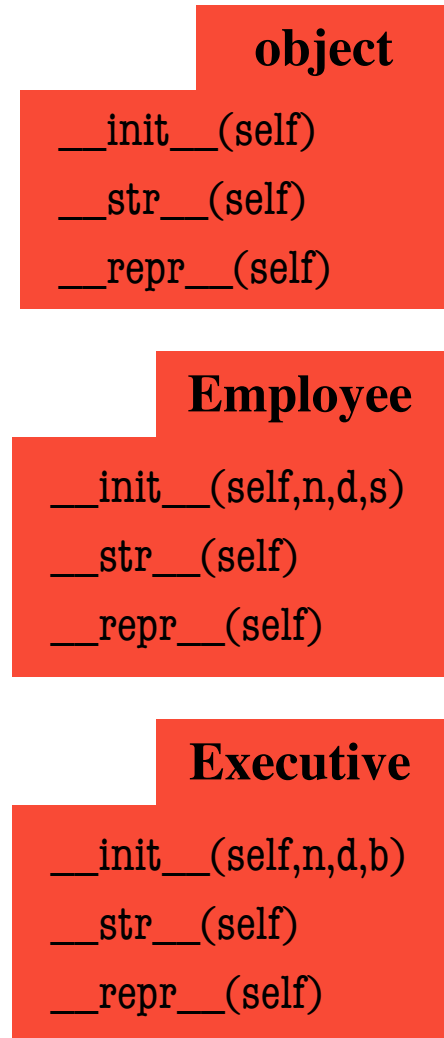
object

__init__(self)

__str__(self)

__repr__(self)

Employee

__init__(self,n,d,s)

__str__(self)

__repr__(self)

Executive

__init__(self,n,d,b)

__str__(self)

__repr__(self)

Accessing the "Previous" Method

- What if you want to use the original version method?
  - New method = original+more
  - Do not want to repeat code from the original version

- Use the function `super()`
  - "Converts" type to parent class
  - Now methods go to the class

- **Example**:

  `super().__str__()`

  > **In Python 2 self goes here**

object

```
__init__(self)
__str__(self)
__eq__(self)
```

Employee

```
__init__(self,n,d,s)
__str__(self)
__eq__(self)
```

Executive

```
__init__(self,n,d,b)
__str__(self)
__eq__(self)
```

Accessing the "Previous" Method

- What if you want to use the original version method?
  - New method = original+more
  - Do not want to repeat code from the original version
- Use the function super()
  - "Converts" type to parent class
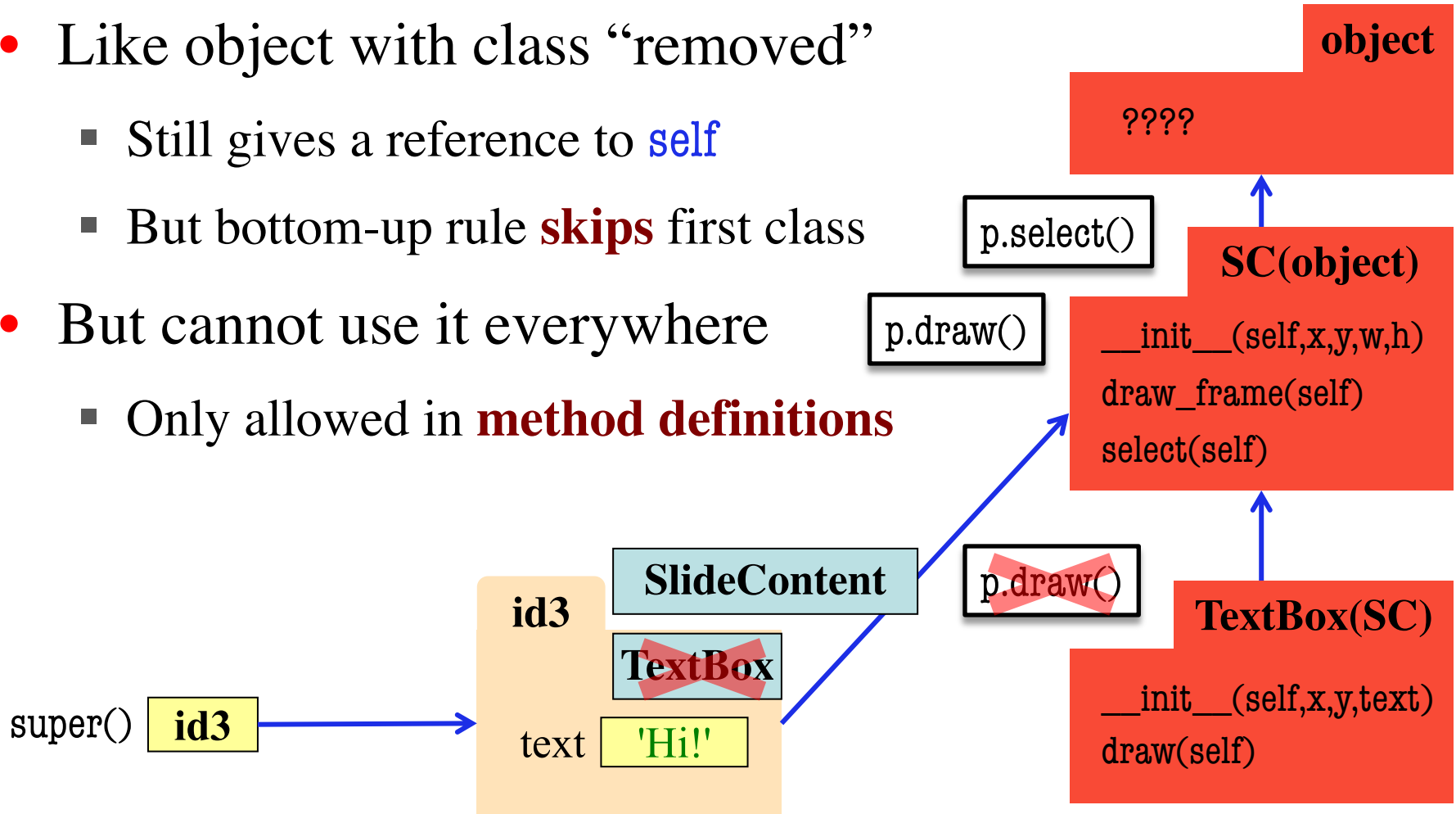 - Now methods go to the class
- **Example**:

  super().__str__()

  > **self** is implied

```python
class Employee(object):
    """An Employee with a salary"""

    ...

    def __str__(self):
        return (self._name +
                ', year ' + str(self._start) +
                ', salary ' + str(self._salary))


class Executive(Employee):
    """An Employee with a bonus."""

    ...

    def __str__(self):
        return (super().__str__()
                + ', bonus ' + str(self._bonus) )
```

What is **super()**?

- Like object with class "removed"
  - Still gives a reference to self
  - But bottom-up rule **skips** first class
- But cannot use it everywhere
  - Only allowed in **method definitions**
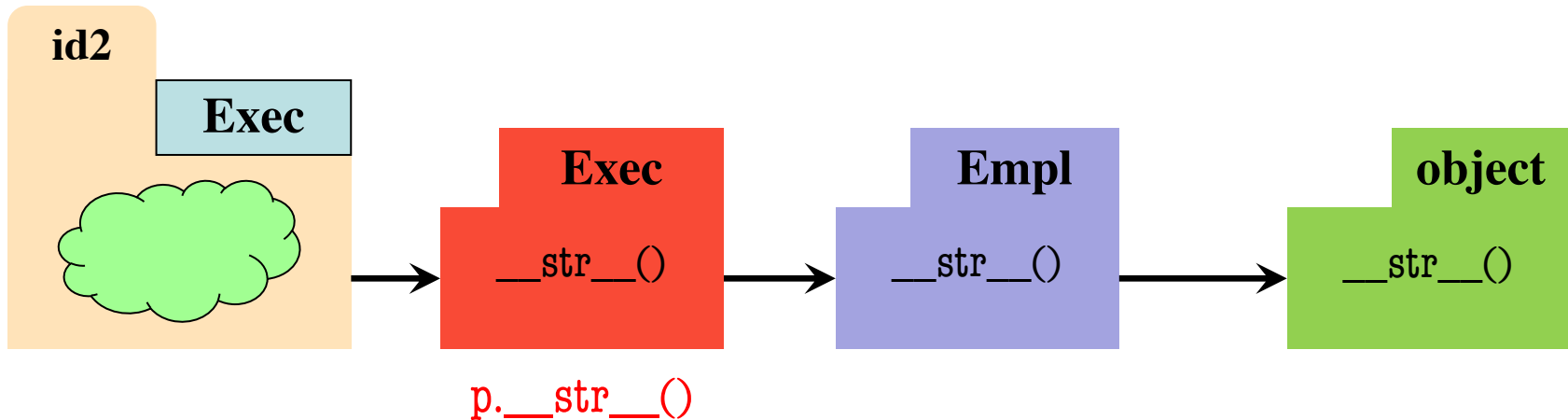
object

????

SC(object)

__init__(self,x,y,w,h)

draw_frame(self)

select(self)

p.select()

p.draw()

TextBox(SC)

__init__(self,x,y,text)

draw(self)

p.draw()

SlideContent

id3

TextBox

text | 'Hi!'

super() | **id3**

What is **super()**?

- super() is very limited
  - Can only go one level
  - **BAD**: super().super()

  p **id2**

- Need arguments for more
  - super(class,self)

  What class to **skip** over

  Object in the method

id2

Exec

Exec

__str__()

Empl

__str__()

object

__str__()

p.__str__()

What is **super()**?

- super() is very limited
  - Can only go one level
  - **BAD**: super().super()

  p [**id2**]

- Need arguments for more
  - super(class,self)

  What class to **skip** over

  Object in the method

id2

Exec

Exec

__str__()

Empl

__str__()

object

__str__()

p.__str__()

What is **super()**?

- super() is very limited
  - Can only go one level
  - **BAD**: super().super()

p id2

- Need arguments for more
  - super(class,self)

What class to **skip** over
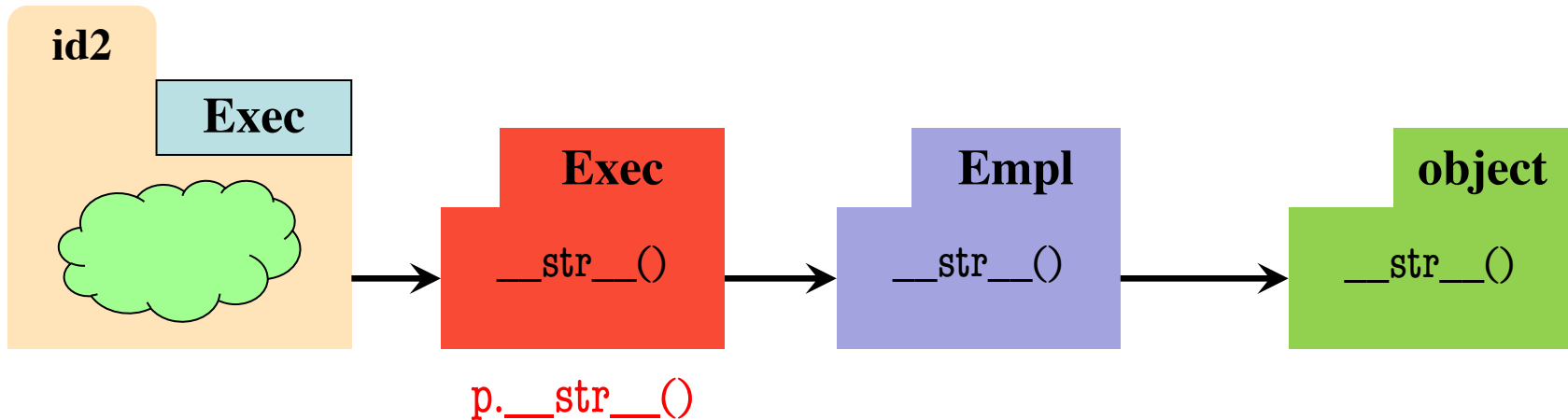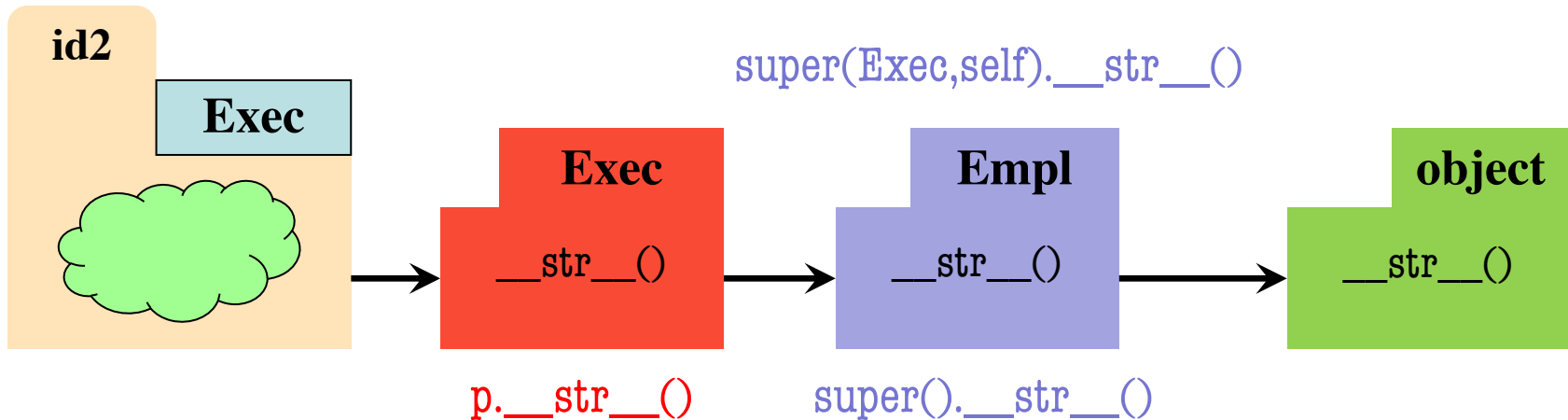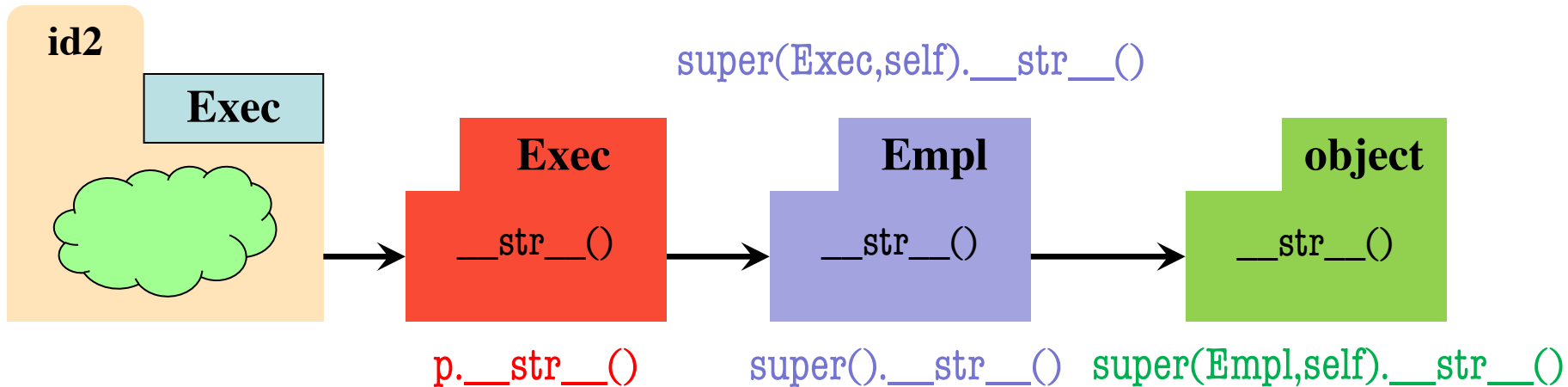
Object in the method

id2

Exec

super(Exec,self).__str__()

Exec

__str__()

Empl

__str__()

object

__str__()

p.__str__()

super().__str__()

Subclasses & Inheritance

What is **super()**?

- super() is very limited
  - Can only go one level
  - **BAD**: super().super()

p [**id2**]

- Need arguments for more
  - super(class,self)

What class to **skip** over

Object in the method

```
id2
  Exec

        Exec            super(Exec,self).__str__()

        Exec              Empl              object

        __str__()        __str__()        __str__()

      p.__str__()    super().__str__()  super(Empl,self).__str__()
```

Primary Application: Initializers

```python
class Employee(object):
    ...
    def __init__(self,n,d,s=50000.0):
        self._name = n
        self._start = d
        self._salary = s
```

```python
class Executive(Employee):
    ...
    def __init__(self,n,d,b=0.0):
        super().__init__(n,d)
        self._bonus = b
```

object

__init__(self)

__str__(self)

__repr__(self)

Employee

__init__(self,n,d,s)

__str__(self)

__repr__(self)

Executive

__init__(self,n,d,b)

__str__(self)

__repr__(self)

Instance Attributes are (Often) Inherited

```python
class Employee(object):
    ...
    def __init__(self,n,d,s=50000.0):
        self._name = n
        self._start = d
        self._salary = s
```

```python
class Executive(Employee):
    ...
    def __init__(self,n,d,b=0.0):
        super().__init__(n,d)
        self._bonus = b
```

id4

Executive

_name	'Fred'
_start	2012
_salary	50000.0

Created in **Employee** initializer

| _bonus | 0.0 |

Created in **Executive** initializer