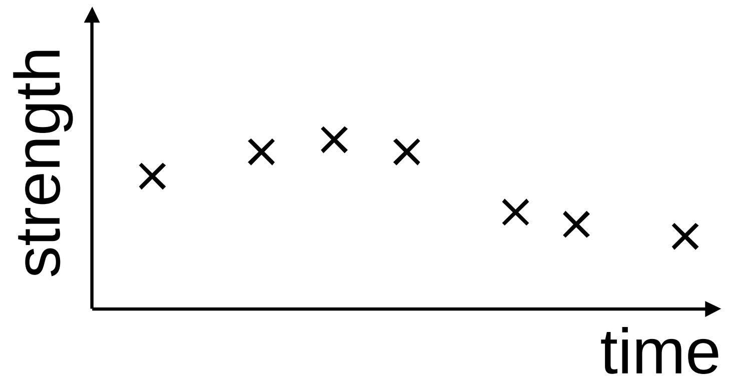
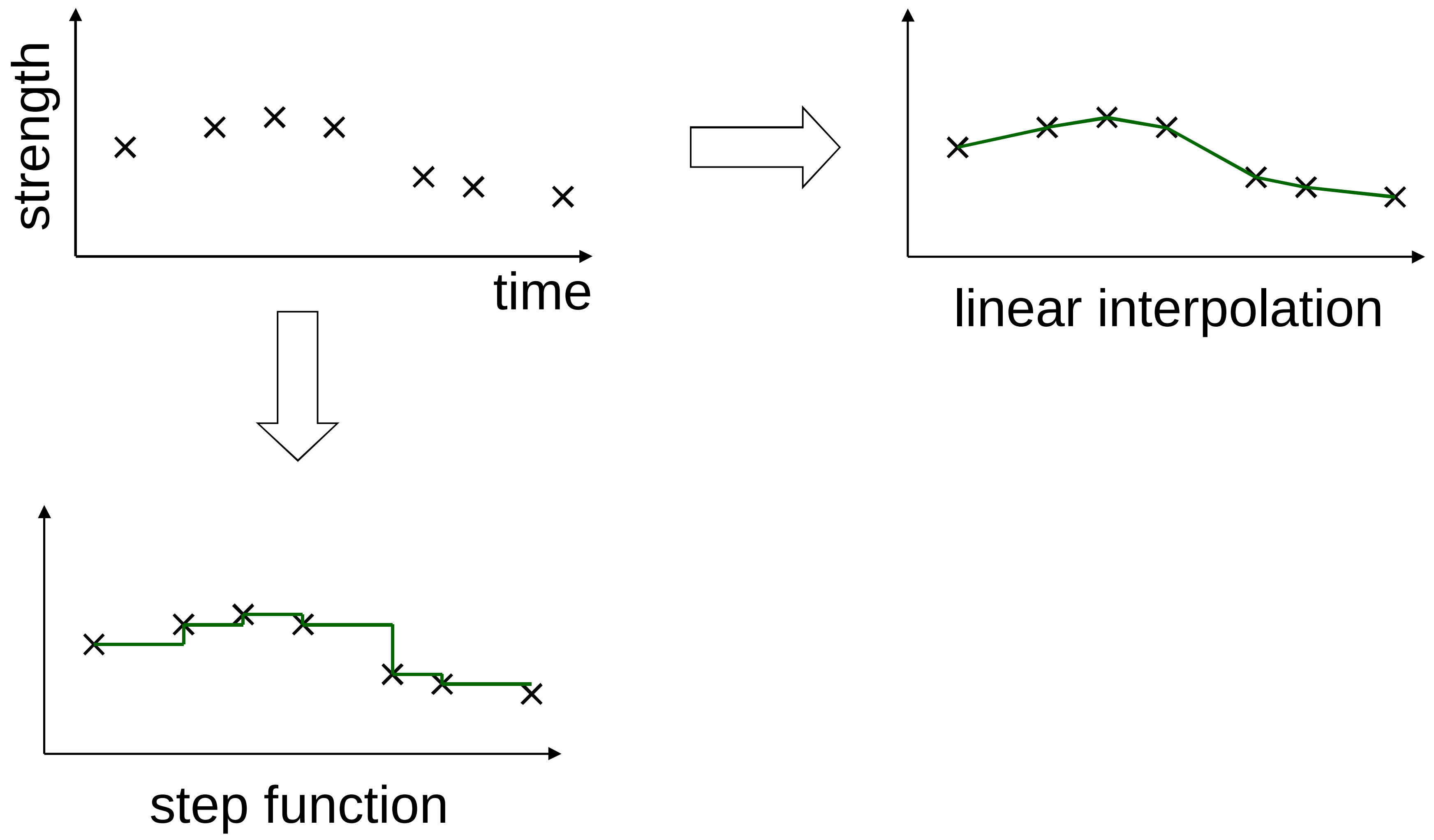
Classes and Objects

Inheritance

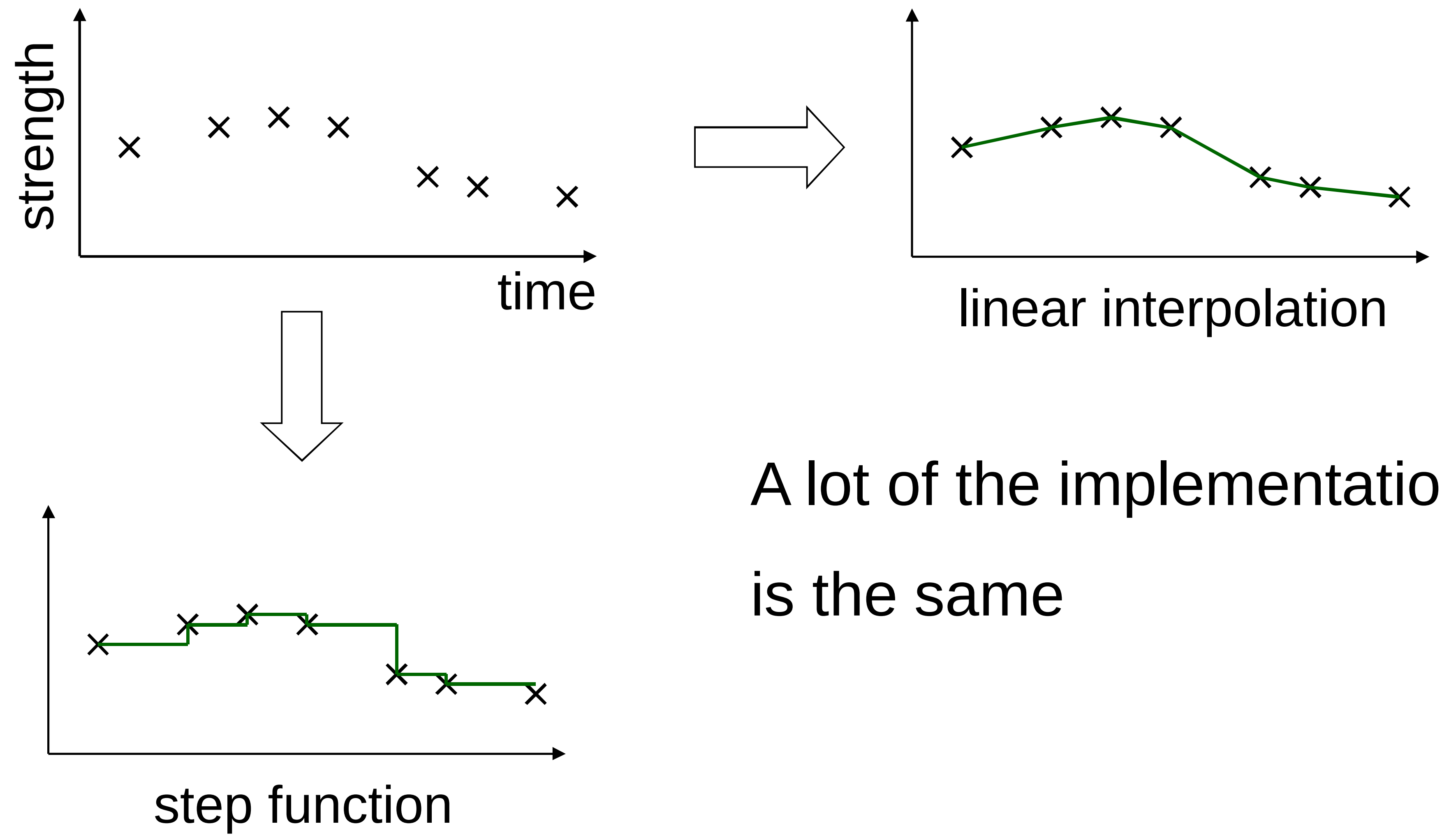
Interpolating time series signals



Interpolating time series signals

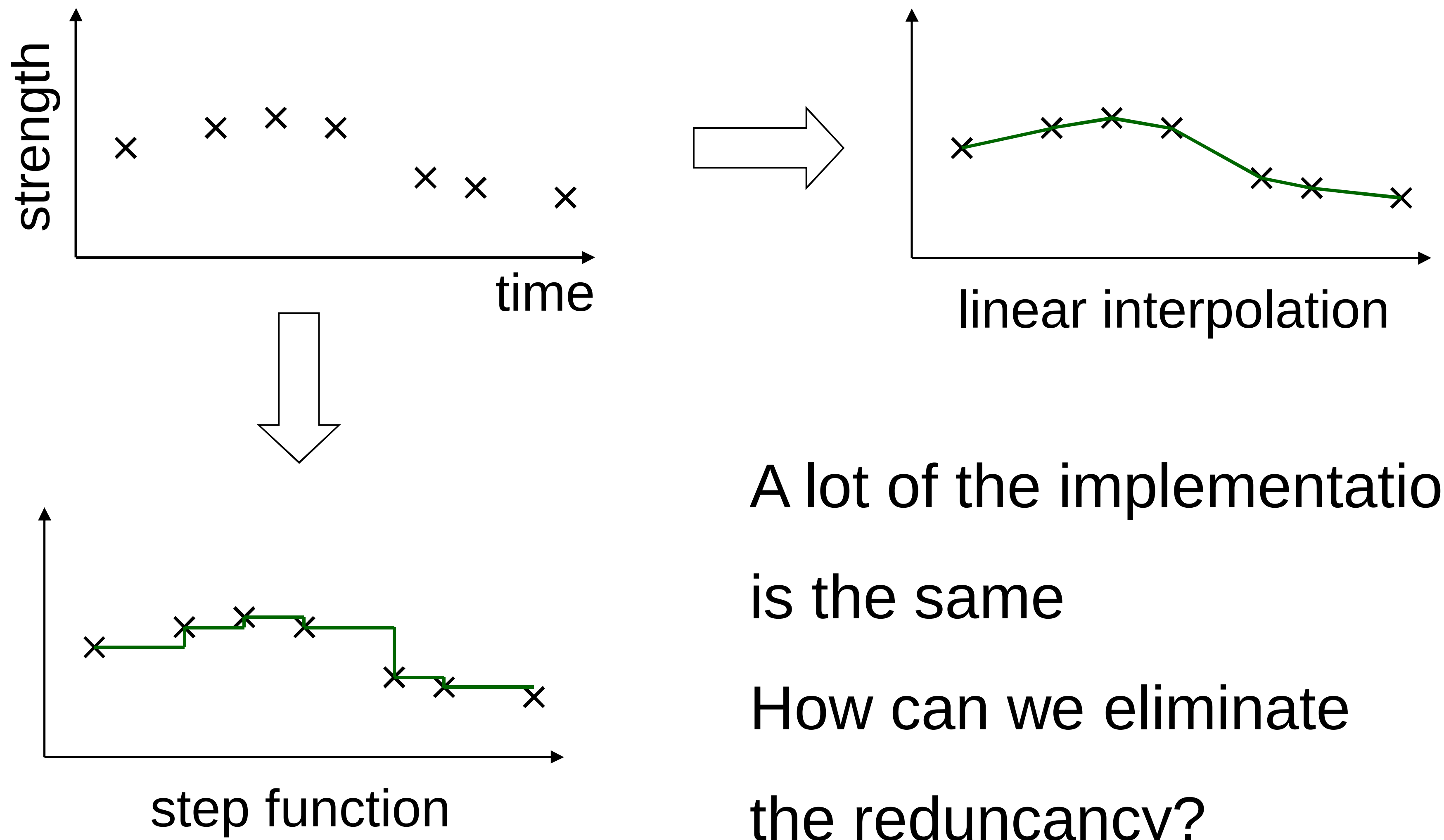


Interpolating time series signals



A lot of the implementation is the same

Interpolating time series signals



A lot of the implementation

is the same

How can we eliminate

the reduncancy?

First implementation

```
class StepSignal(object):
  ...
  def get(self, where):
    if where < self.values[0][0]:
      raise IndexError, '%f too low' % where
    for i in range(len(self.values)-1):
      x0, y0 = self.values[i]
      x1, y1 = self.values[i+1]
      if x0 <= where <= x1:
        return y0
    raise IndexError, '%f too high' % where
```

Second implementation

```
class LinearSignal(object):
  ...
  def get(self, where):
    if where < self.values[0][0]:
      raise IndexError, '%f too low' % where
    for i in range(len(self.values)-1):
      x0, y0 = self.values[i]
      x1, y1 = self.values[i+1]
      if x0 <= where <= x1:
        return y0 + (y1-y0) * (where-x0) /
   (x1-x0)
    raise IndexError, '%f too high' % where
```

Second implementation

```
class LinearSignal(object):
  ...
  def get(self, where):
    if where < self.values[0][0]:
      raise IndexError, '%f too low' % where
    for i in range(len(self.values)-1):
      x0, y0 = self.values[i]
      x1, y1 = self.values[i+1]
      if x0 <= where <= x1:
        return y0 + (y1-y0) * (where-x0) /
 (x1-x0)
    raise IndexError, '%f too high' % where
```

Refactor

```
class StepSignal(object):
  def get(self, where):
    i = self.find(self, where)
    return self.values[i][1]

class LinearSignal(object):
  def get(self, where):
    i = self.find(self, where)
    x0, y0 = self.values[i]
    x1, y1 = self.values[i+1]
    return y0 + (y1-y0) * (where-x0)/(x1-x0)
```

Refactor

```
class StepSignal(object):
  def get(self, where):
    i = self.find(self, where)
    return self.values[i][1]

class LinearSignal(object):
  def get(self, where):
    i = self.find(self, where)
    x0, y0 = self.values[i]
    x1, y1 = self.values[i+1]
    return y0 + (y1-y0) * (where-x0)/(x1-x0)
```

Where to put find?

Refactor

```
class StepSignal(object):
  def get(self, where):
    i = self.find(self, where)
    return self.values[i][1]

class LinearSignal(object):
  def get(self, where):
    i = self.find(self, where)
    x0, y0 = self.values[i]
    x1, y1 = self.values[i+1]
    return y0 + (y1-y0) * (where-x0)/(x1-x0)
```

Where to put find?

Don't want to duplicate

Use *inheritance*

Use *inheritance*

```
class Parent(object):
  def hello(self):
    print 'hello'
```

Use *inheritance*

```
class Parent(object):
  def hello(self):
    print 'hello'

class Child(Parent):
  def goodbye(self):
    print 'goodbye'
```

Use *inheritance*

```
class Parent(object):
  def hello(self):
    print 'hello'

class Child(Parent):
  def goodbye(self):
    print 'goodbye'
```

`Child` inherits from `Parent`

Use *inheritance*

```
class Parent(object):     c = Child()
  def hello(self):        c.goodbye()
    print 'hello'         goodbye

class Child(Parent):
  def goodbye(self):
    print 'goodbye'
```

`Child` inherits from `Parent`

Use *inheritance*

```
class Parent(object):
  def hello(self):
    print 'hello'

class Child(Parent):
  def goodbye(self):
    print 'goodbye'
```

```
c = Child()
c.goodbye()
goodbye
c.hello()
hello
```

`Child` inherits from `Parent`

Use *inheritance*

```
class Parent(object):
  def hello(self):
    print 'hello'

class Child(Parent):
  def goodbye(self):
    print 'goodbye'
```

```
c = Child()
c.goodbye()
goodbye
c.hello()
hello
p = Parent()
p.hello()
hello
```

`Child` inherits from `Parent`

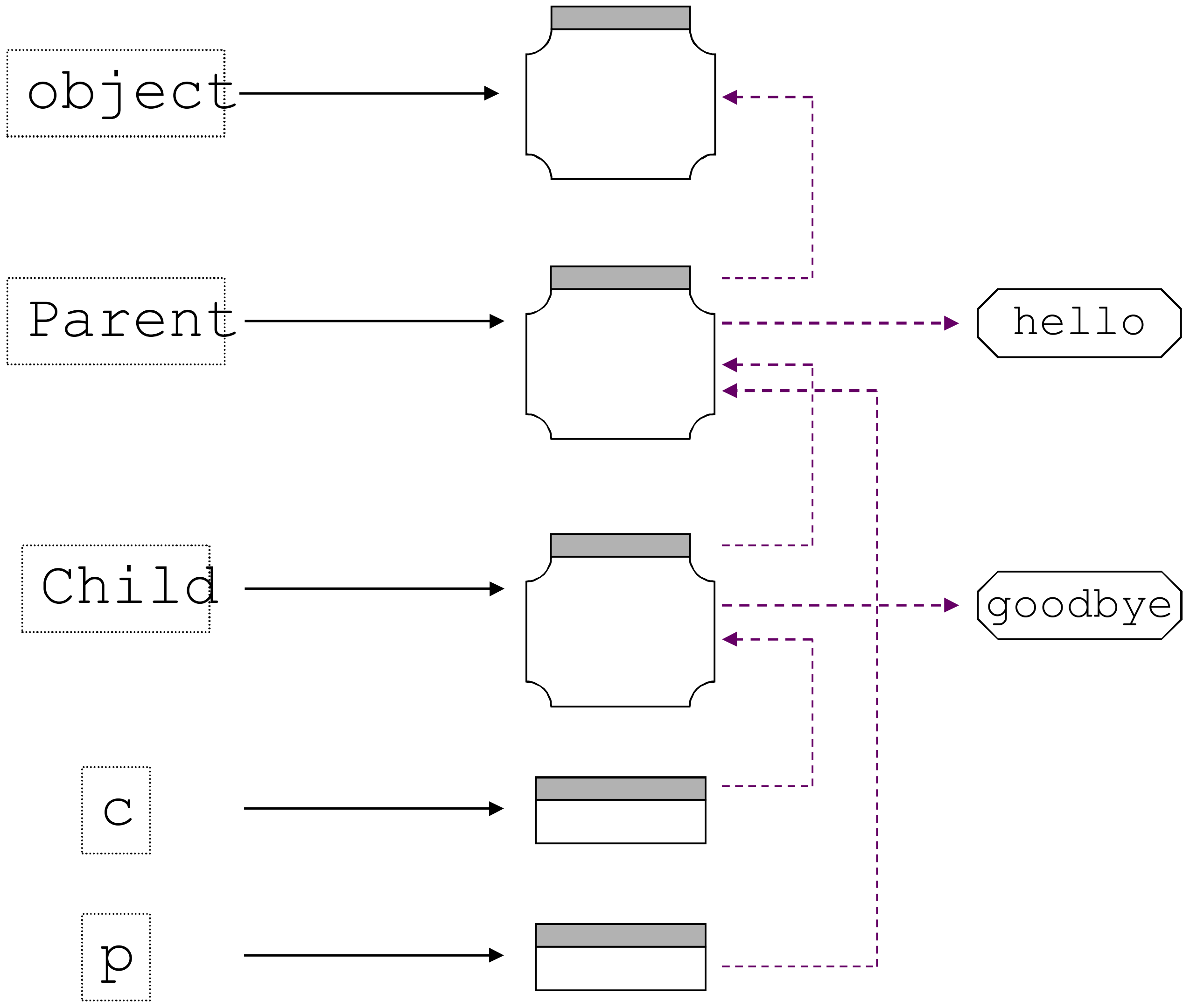
Use *inheritance*

```
class Parent(object):
   def hello(self):
      print 'hello'

class Child(Parent):
   def goodbye(self):
      print 'goodbye'
```

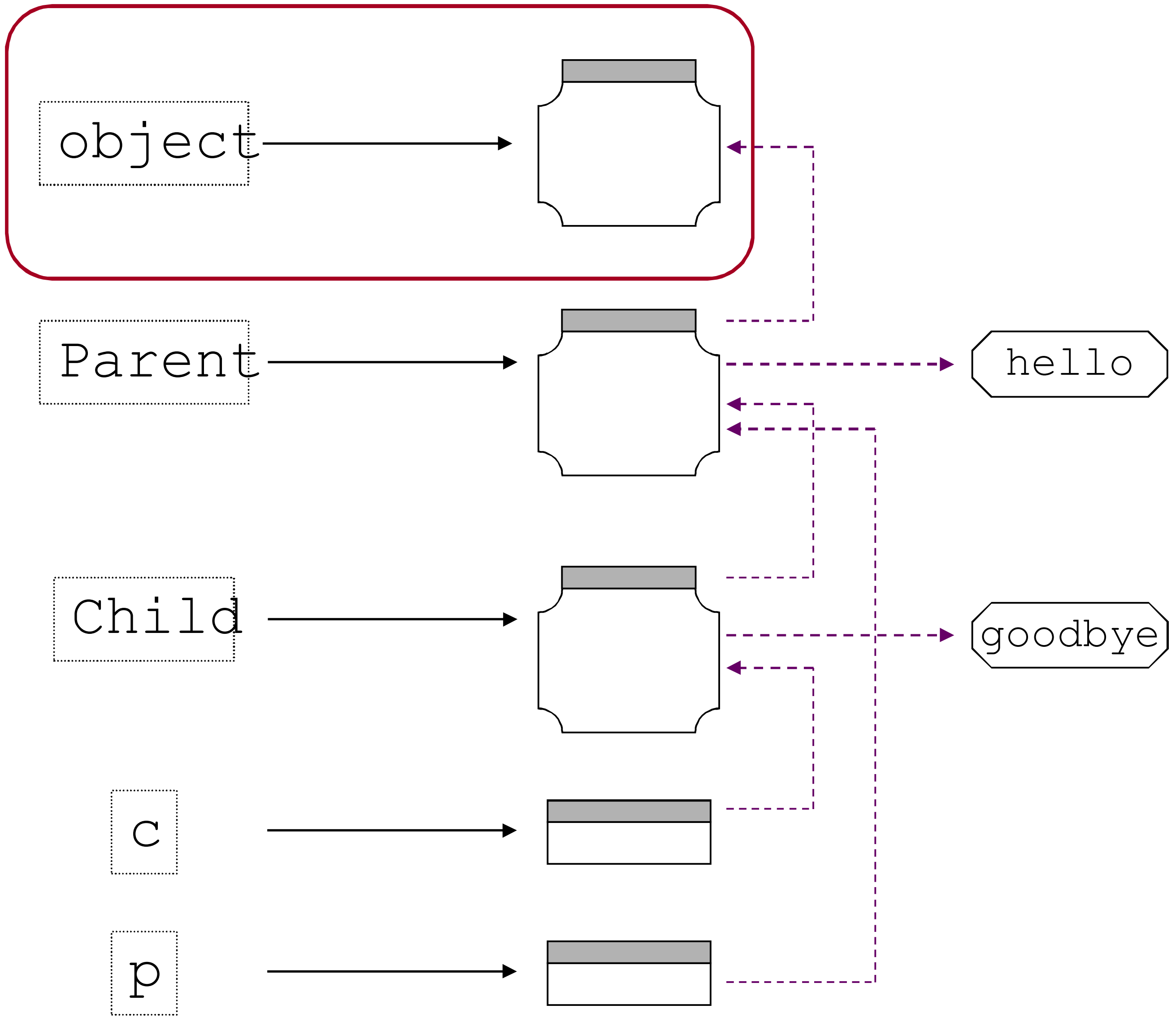
Child inherits from Parent

```
c = Child()
c.goodbye()
goodbye
c.hello()
hello
p = Parent()
p.hello()
hello
p.goodbye()
AttributeError: 'Parent'
   object
has no attribute 'goodbye'
```

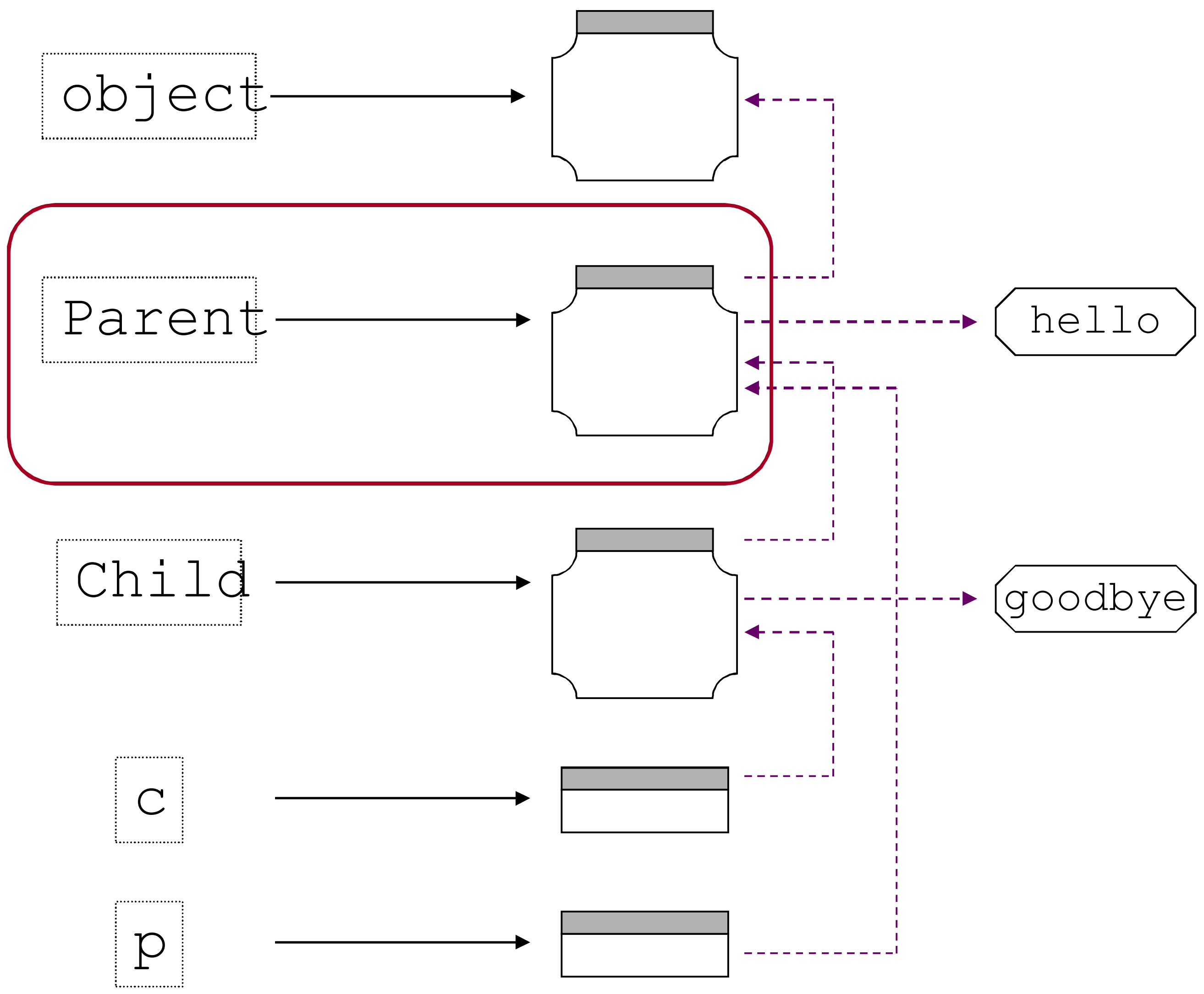
Contents of memory



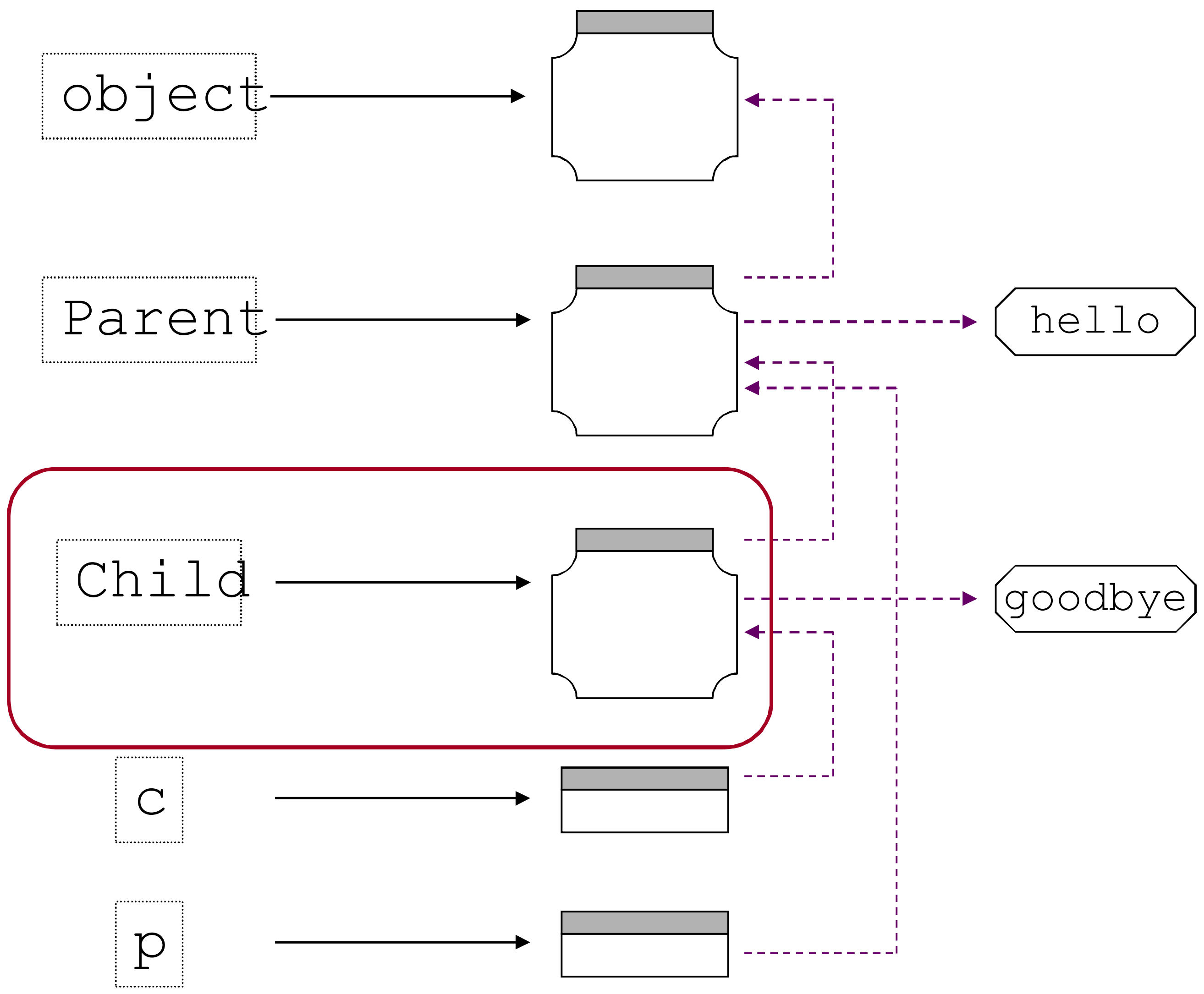
Contents of memory



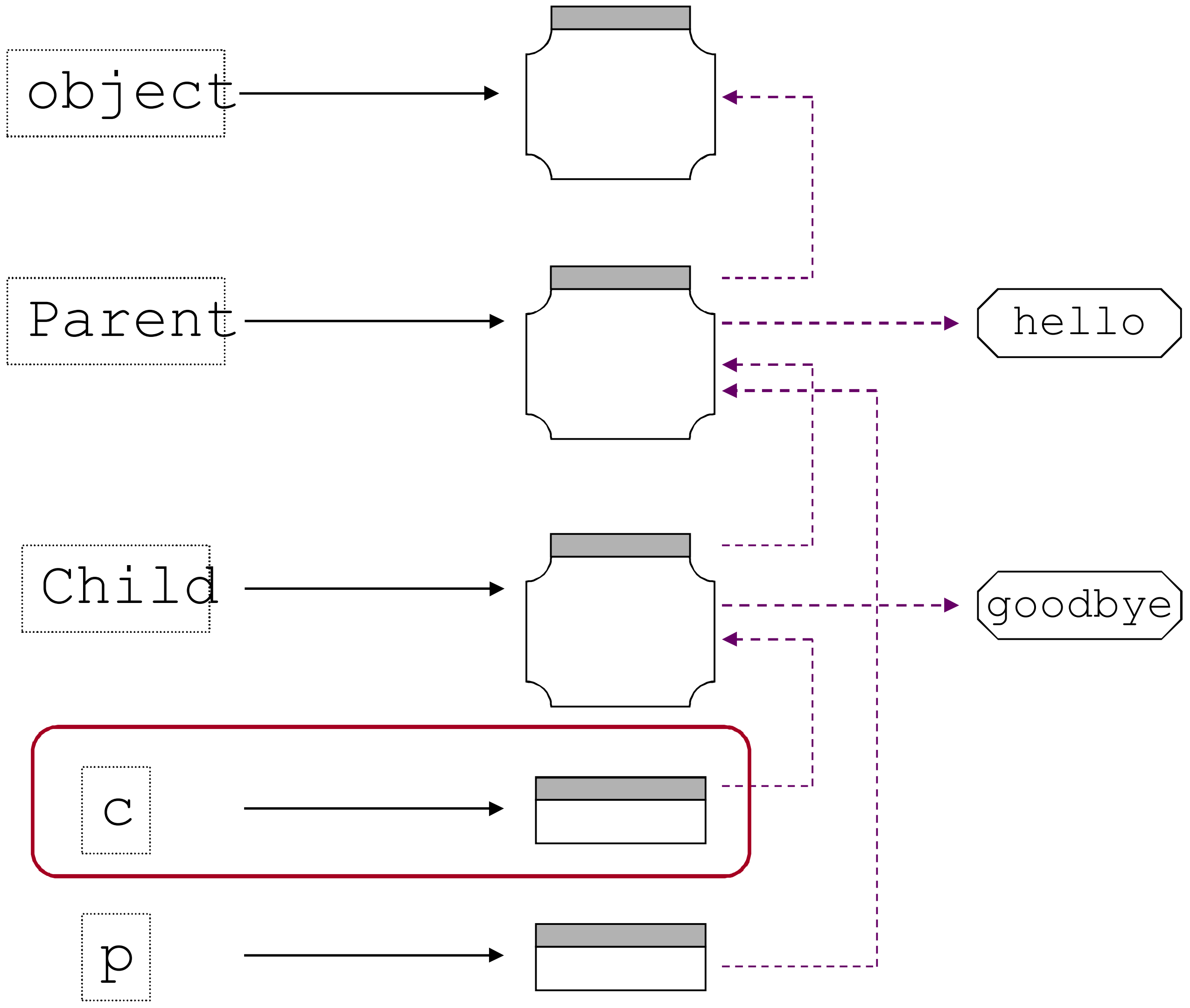
Contents of memory



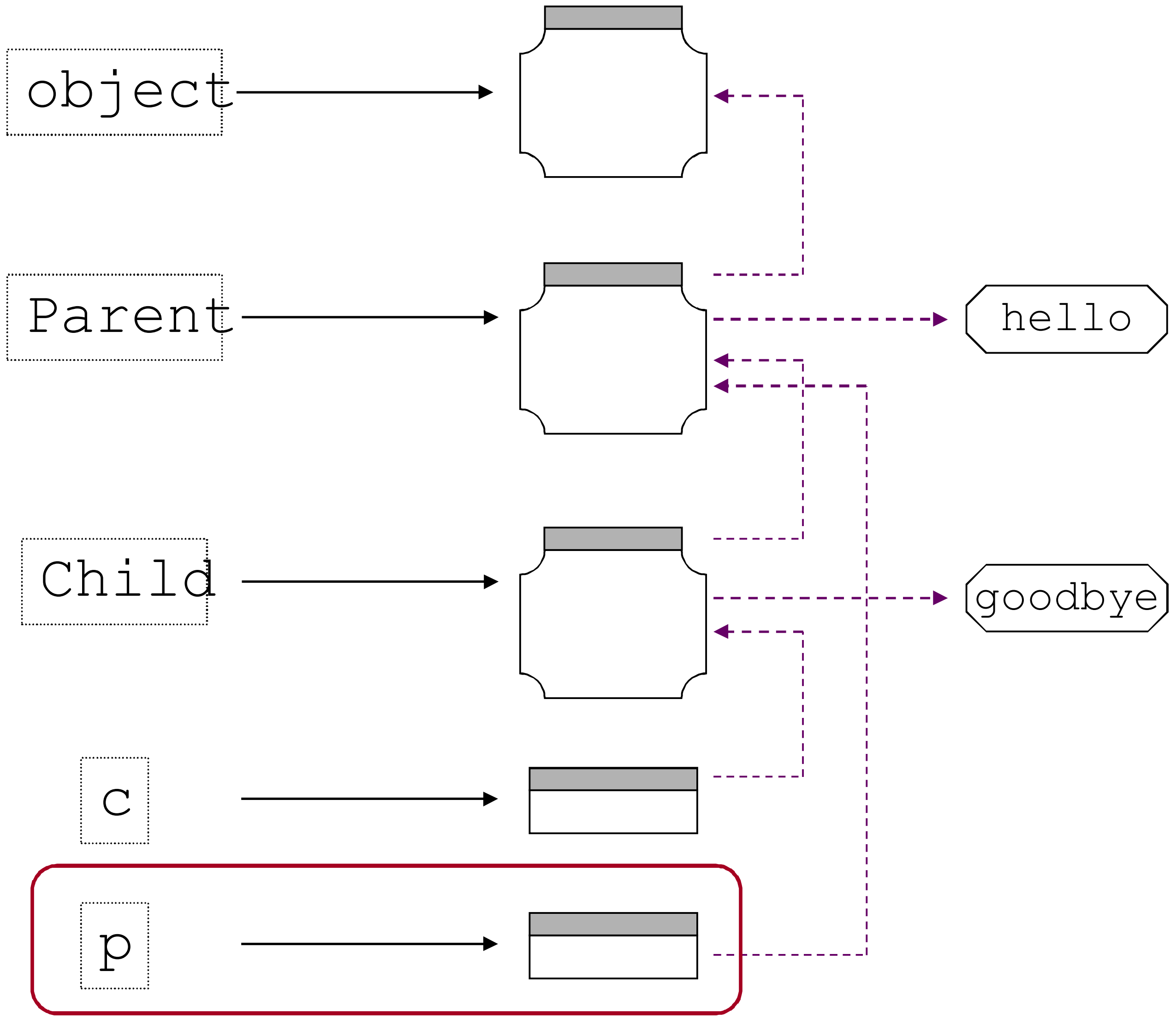
Contents of memory



Contents of memory



Contents of memory



```
class InterpolatedSignal(object):

  def find(self, where):
    if where < self.values[0][0]:
      raise IndexError, '%f too low' % where
    for i in range(len(self.values)-1):
      x0, y0 = self.values[i]
      x1, y1 = self.values[i+1]
      if x0 <= where <= x1:
        return i
    raise IndexError, '%f too high' % where
```

```
class InterpolatedSignal(object):

  def find(self, where):
    if where < self.values[0][0]:
      raise IndexError, '%f too low' % where
    for i in range(len(self.values)-1):
      x0, y0 = self.values[i]
      x1, y1 = self.values[i+1]
      if x0 <= where <= x1:
        return i
    raise IndexError, '%f too high' % where
```

Not much use on its own

```
class InterpolatedSignal(object):

  def find(self, where):
    if where < self.values[0][0]:
      raise IndexError, '%f too low' % where
    for i in range(len(self.values)-1):
      x0, y0 = self.values[i]
      x1, y1 = self.values[i+1]
      if x0 <= where <= x1:
        return i
    raise IndexError, '%f too high' % where
```

Where does this come from?

Not much use on its own

Derive specific interpolators

Derive specific interpolators

```
class StepSignal(InterpolatedSignal):

  def __init__(self, values):
    self.values = values[:]

  def get(self, where):
    i = self.find(where)
    return self.values[i][1]
```

Derive specific interpolators

```
class StepSignal(InterpolatedSignal):

  def __init__(self, values):
    self.values = values[:]

  def get(self, where):
    i = self.find(where)
    return self.values[i][1]
```

Derive specific interpolators

```
class StepSignal(InterpolatedSignal):

  def __init__(self, values):
    self.values = values[:]

  def get(self, where):
    i = self.find(where)
    return self.values[i][1]
```

Derive specific interpolators

```
class StepSignal(InterpolatedSignal):

  def __init__(self, values):
    self.values = values[:]

  def get(self, where):
    i = self.find(where)
    return self.values[i][1]
```

Derive specific interpolators

```
class StepSignal(InterpolatedSignal):

  def __init__(self, values):
    self.values = values[:]

  def get(self, where):
    i = self.find(where)
    return self.values[i][1]
```

Fragile

Derive specific interpolators

```
class StepSignal(InterpolatedSignal):

  def __init__(self, values):
    self.values = values[:]

  def get(self, where):
    i = self.find(where)
    return self.values[i][1]
```

Dependencies between classes should be explicit

Have the parent class store the values

Have the parent class store the values

```
class InterpolatedSignal(object):

  def __init__(self, values):
    self.values = values[:]

  def get(self, where):
    raise NotImplementedError('Must provide
 get!')

  def find(self, where):
    ...as before...
```

Have the parent class store the values

```
class InterpolatedSignal(object):

  def __init__(self, values):
    self.values = values[:]

  def get(self, where):
    raise NotImplementedError('Must provide
 get!')

  def find(self, where):
    ...as before...
```

The child's constructor relies on the parent's

```
class StepSignal(InterpolatedSignal):

  def __init__(self, values):
    InterpolatedSignal.__init__(self, values)

  def get(self, where):
    i = self.find(where)
    return self.values[i][1]
```

The child's constructor relies on the parent's

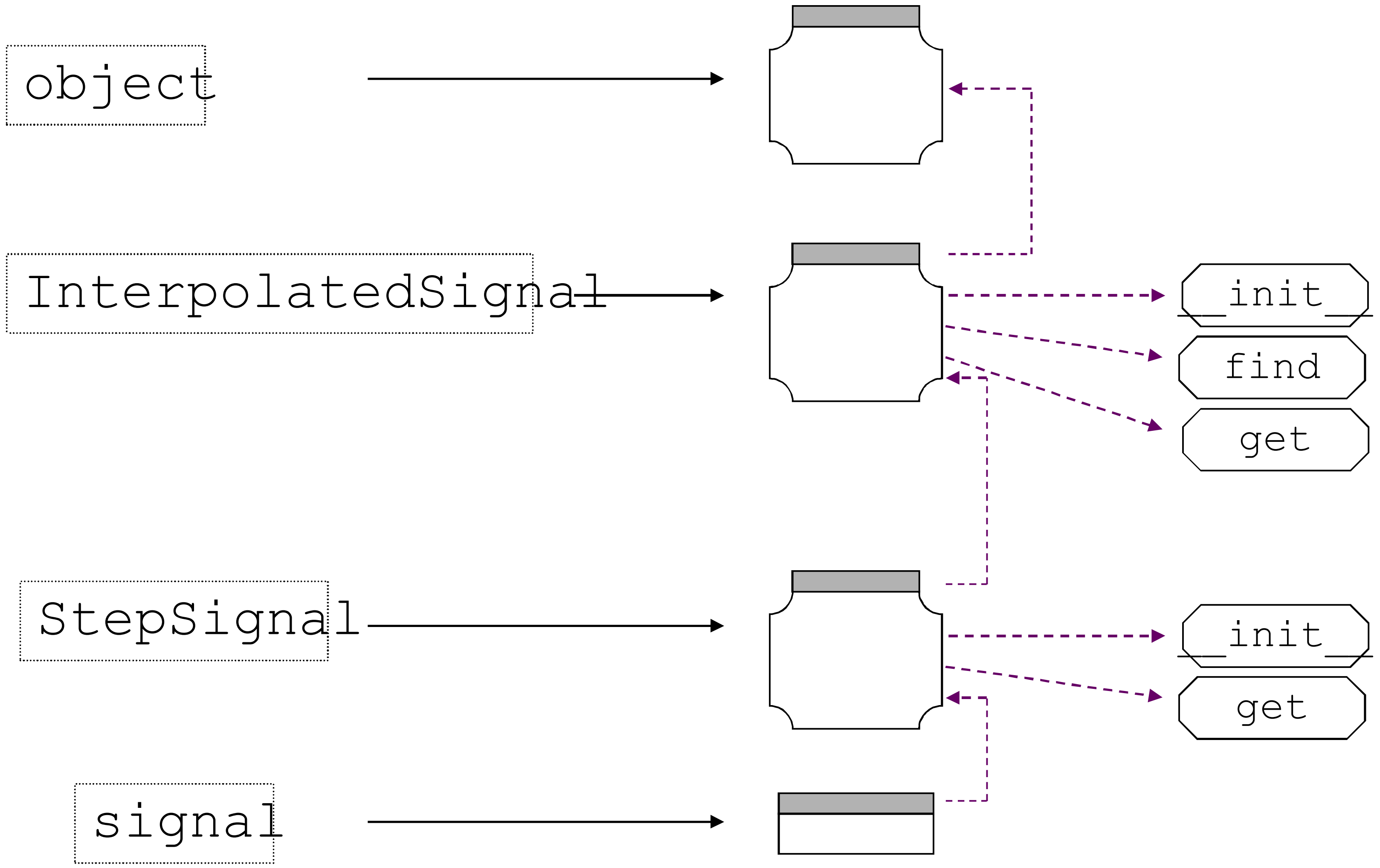
```
class StepSignal(InterpolatedSignal):

  def __init__(self, values):
    InterpolatedSignal.__init__(self, values)

  def get(self, where):
    i = self.find(where)
    return self.values[i][1]
```

Other classes are just as easy

```
Class LinearSignal(InterpolatedSignal):

  def __init__(self, values):
    InterpolatedSignal.__init__(self, values)

  def get(self, where):
    i = self.find(where)
    return y0 + (y1-y0) * (where-x0)/(x1-x0)
```

```
class InterpolatedSignal(object):
  def __init__(self, values):
    assert len(values) > 0, 'Must have some pairs'
    for i in range(len(values)):
      assert len(values[i]) == 2, 'Entries must be pairs'
    for i in range(len(values)-1)):
      x0 = values[i][0]
      x1 = values[i][1]
      assert x0 < x1, 'Samples must increase on x'


>>> signal = StepSignal([ [1., 0.], [0., 2] ])
AssertionError: Samples must increase on x
```

Child *overrides* parent method



Overriding in action

Overriding in action

```
class Parent(object):
  def hello(self):
    print 'hello'
  def goodbye(self):
    print 'goodbye'
```

Overriding in action

```
class Parent(object):
  def hello(self):
    print 'hello'
  def goodbye(self):
    print 'goodbye'

class Child(Parent):
  def goodbye(self):
    print 'au revoir'
```

Overriding in action

```
class Parent(object):
  def hello(self):
    print 'hello'
  def goodbye(self):
    print 'goodbye'

class Child(Parent):
  def goodbye(self):
    print 'au revoir'
```

`Child` overrides `Parent`

Overriding in action

```
class Parent(object):    p = Parent()
  def hello(self):       p.hello()
    print 'hello'        hello
  def goodbye(self):     p.goodbye()
    print 'goodbye'      goodbye

class Child(Parent):
  def goodbye(self):
    print 'au revoir'
```

Overriding in action

```
class Parent(object):
  def hello(self):
    print 'hello'
  def goodbye(self):
    print 'goodbye'

class Child(Parent):
  def goodbye(self):
    print 'au revoir'
```

```
p = Parent()
p.hello()
hello
p.goodbye()
goodbye
C = child()
c.hello()
hello
```

Overriding in action

```
class Parent(object):
  def hello(self):
    print 'hello'
  def goodbye(self):
    print 'goodbye'

class Child(Parent):
  def goodbye(self):
    print 'au revoir'
```

```
p = Parent()
p.hello()
hello
p.goodbye()
goodbye
C = child()
c.hello()
hello
c.goodbye()
au revoir
```

created by

Greg Wilson

January 2011