Ghent University
Faculty of Sciences

Python Programming

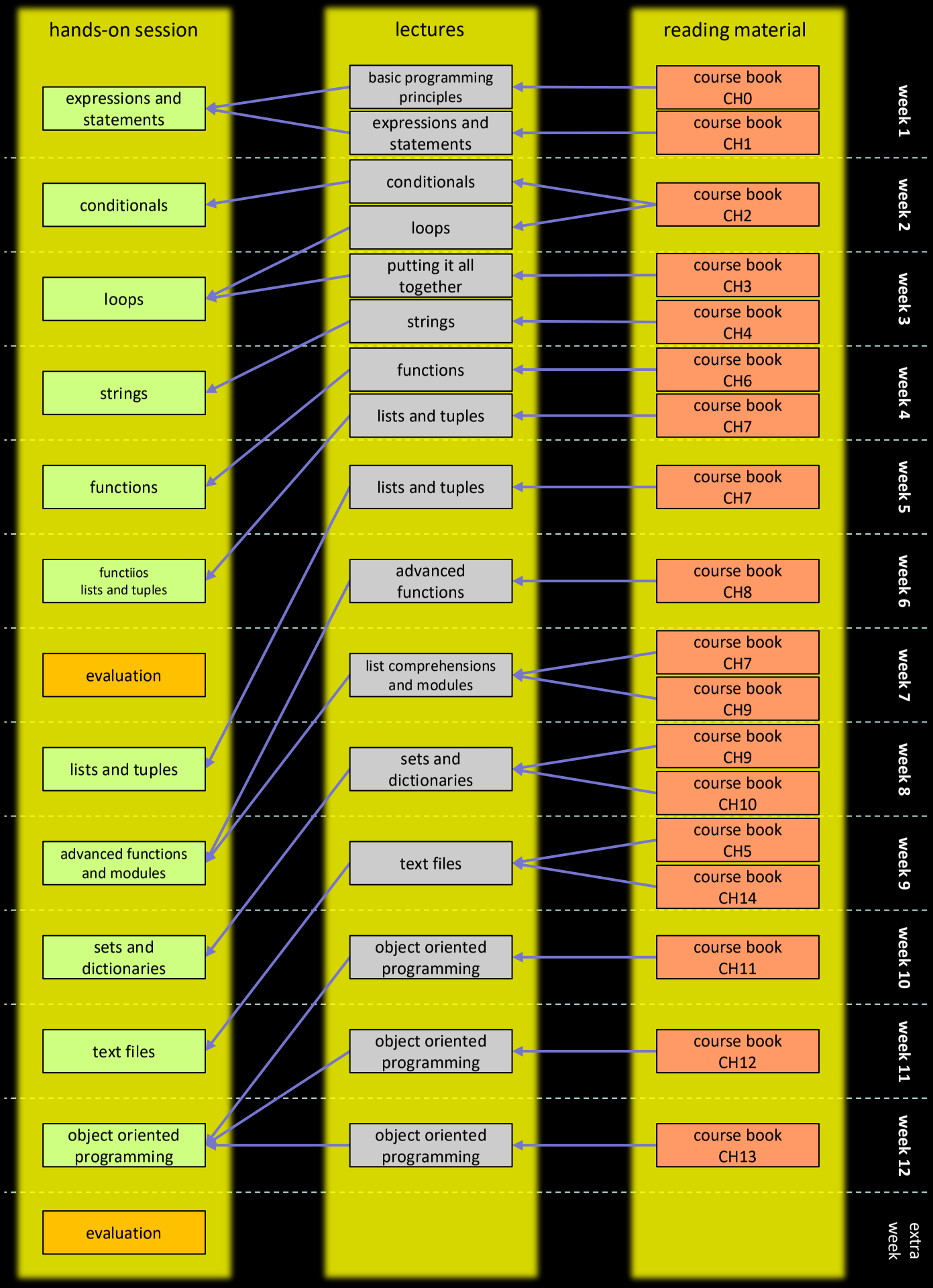
inheritance

Prof. Dr. Peter Dawyndt

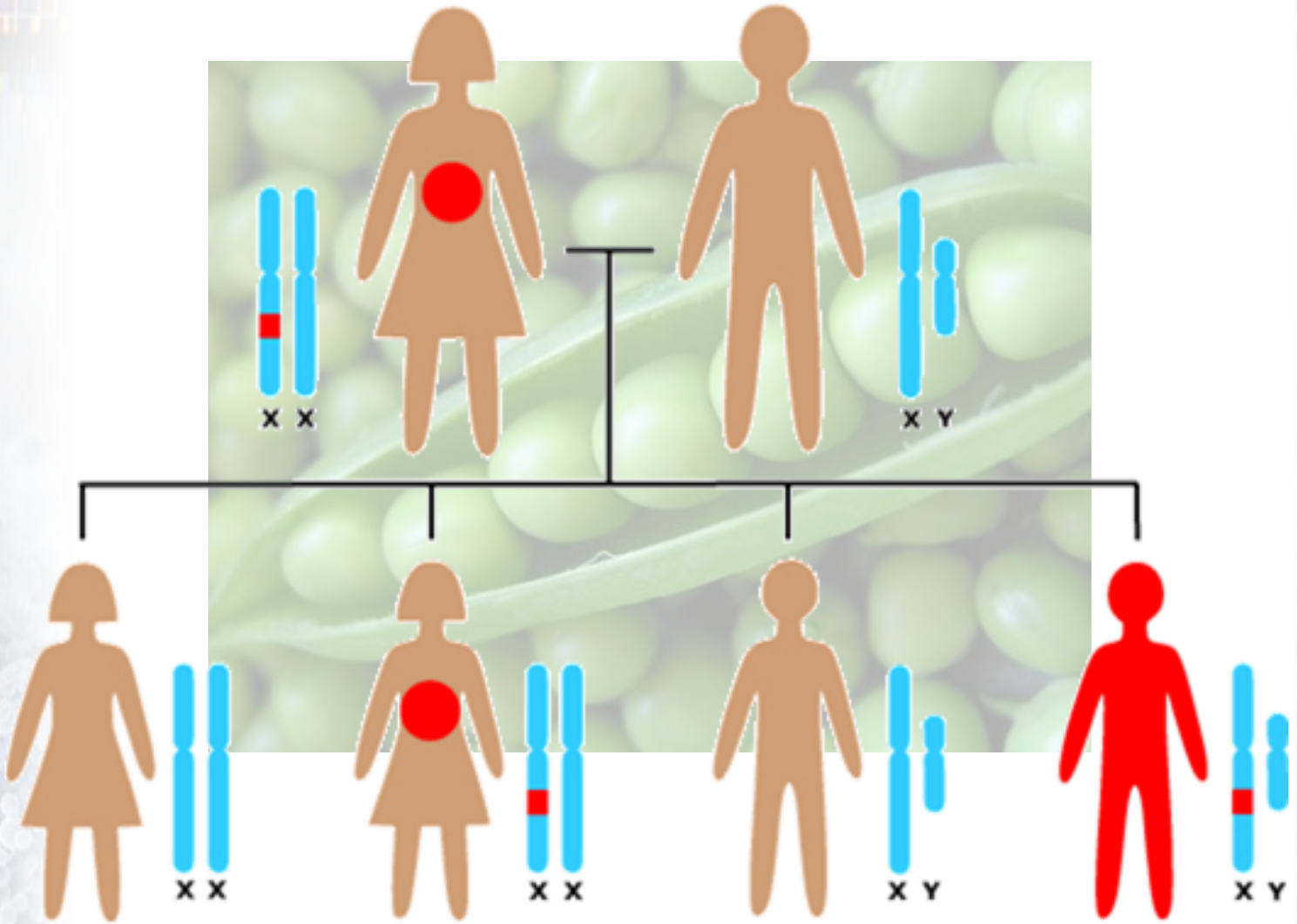
peter.dawyndt@ugent.be

@dawyndt

UNIVERSITEIT GENT


hands-on session
lectures
reading material
week 1
week 2
week 3
week 4
week 5
week 6
week 7
week 8
week 9
week 10
week 11
week 12
extra week
expressions and statements
conditionals
loops
strings
functions
functiios lists and tuples
evaluation
lists and tuples
advanced functions and modules
sets and dictionaries
text files
object oriented programming
evaluation
basic programming principles
expressions and statements
conditionals
loops
putting it all together
strings
functions
lists and tuples
lists and tuples
advanced functions
list comprehensions and modules
sets and dictionaries
text files
object oriented programming
object oriented programming
object oriented programming
course book CH0
course book CH1
course book CH2
course book CH3
course book CH4
course book CH6
course book CH7
course book CH7
course book CH8
course book CH7
course book CH9
course book CH9
course book CH10
course book CH5
course book CH14
course book CH11
course book CH12
course book CH13

Inheritance

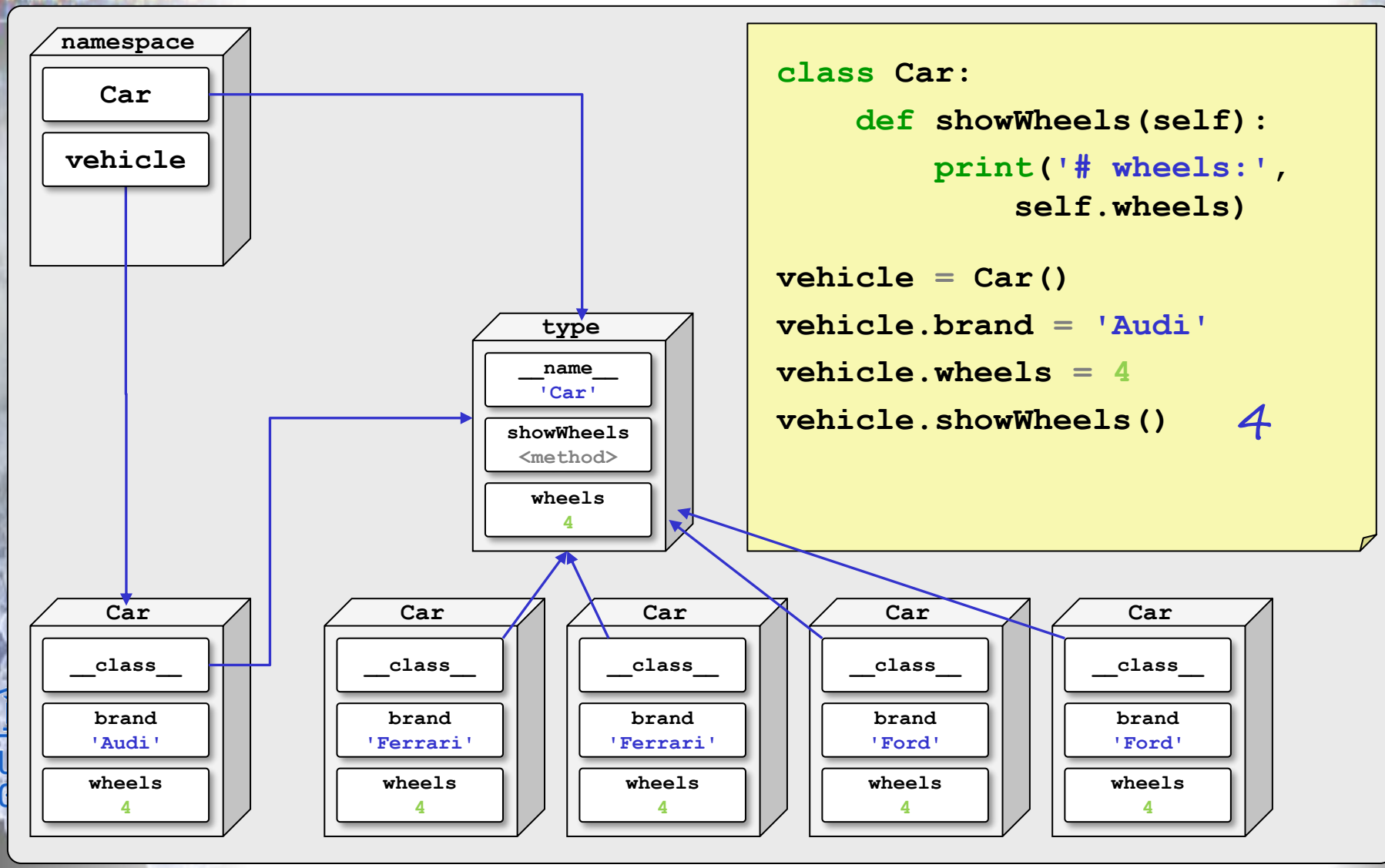


UNIVERSITEIT GENT

OOP characteristics

Classes and instances

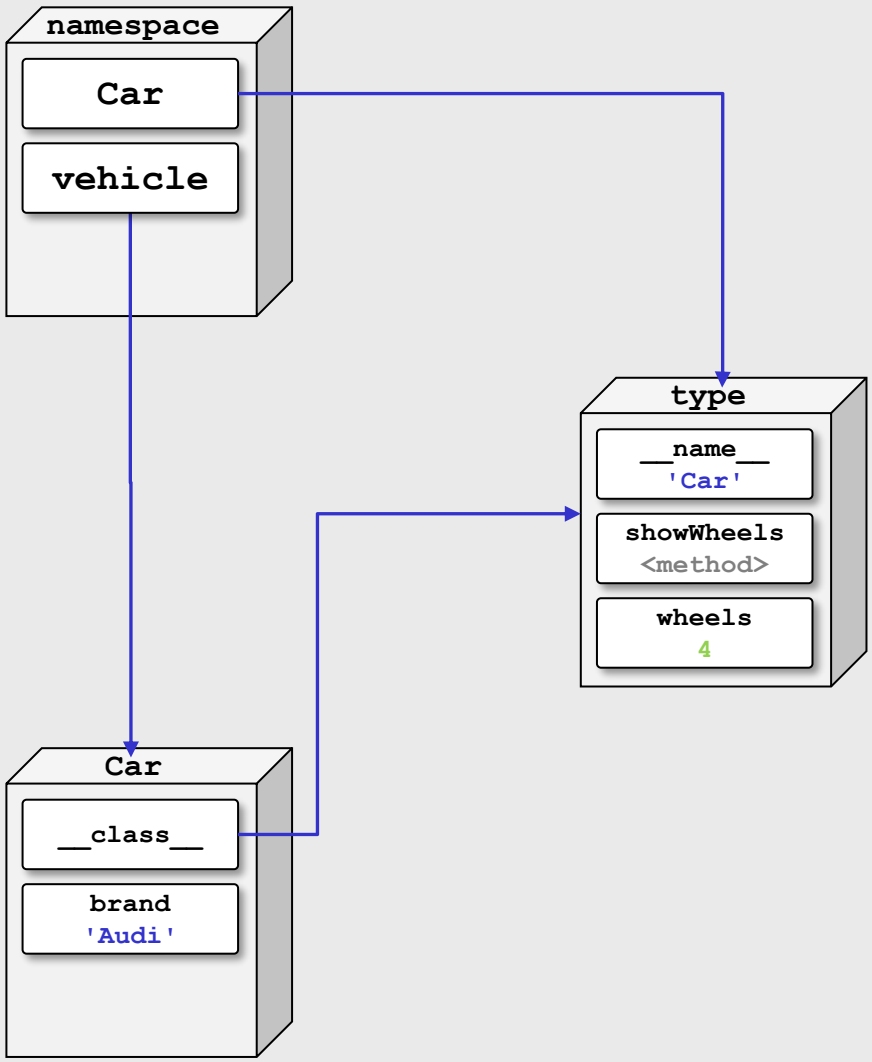
objects (main memory)




```
class Car:
    def showWheels(self):
        print('# wheels:',
              self.wheels)

vehicle = Car()
vehicle.brand = 'Audi'
vehicle.wheels = 4
vehicle.showWheels()    4
```

Classes and instances

objects (main memory)

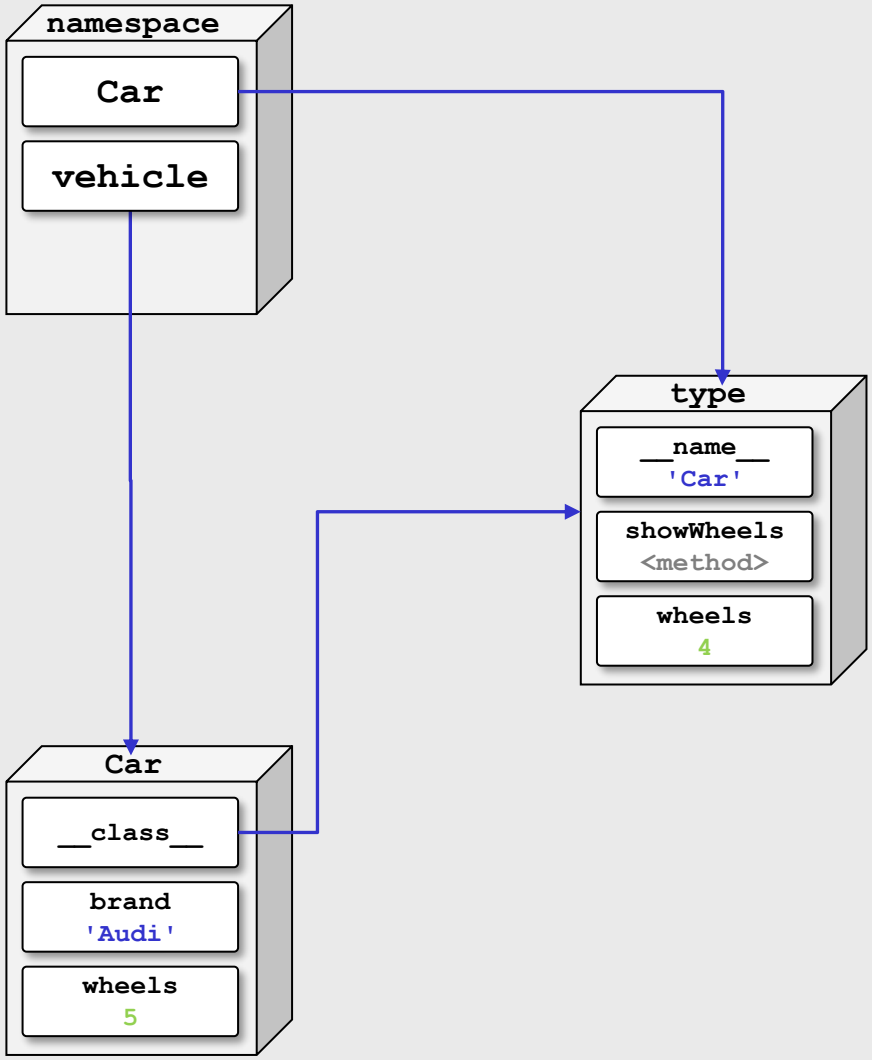



```
class Car:
    wheels = 4
    def showWheels(self):
        print('# wheels:',
              self.wheels)

vehicle = Car()
vehicle.brand = 'Audi'
vehicle.showWheels()
```

4

Classes and instances

objects (main memory)




```
class Car:
    wheels = 4
    def showWheels(self):
        print('# wheels:',
              self.wheels)

vehicle = Car()
vehicle.brand = 'Audi'
vehicle.wheels = 5
vehicle.showWheels()
```

5

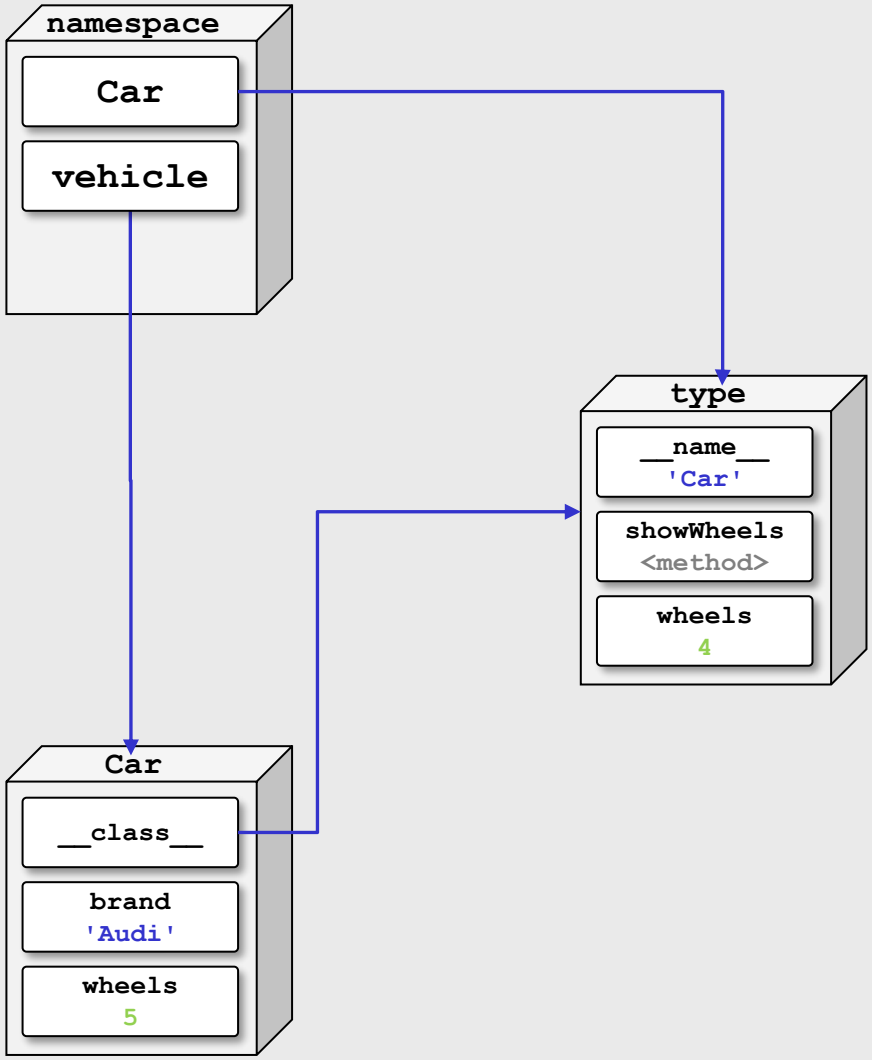
Roman numerals

I = 1
II = 2
III = 3
IV = 4
V = 5
VI = 6
VII = 7
VIII = 8
IX = 9
X = 10

XL = 40
L = 50
LX = 60
LXX = 70
LXXX = 80
XC = 90
C = 100
D = 500
M = 1000

Classes and instances

objects (main memory)

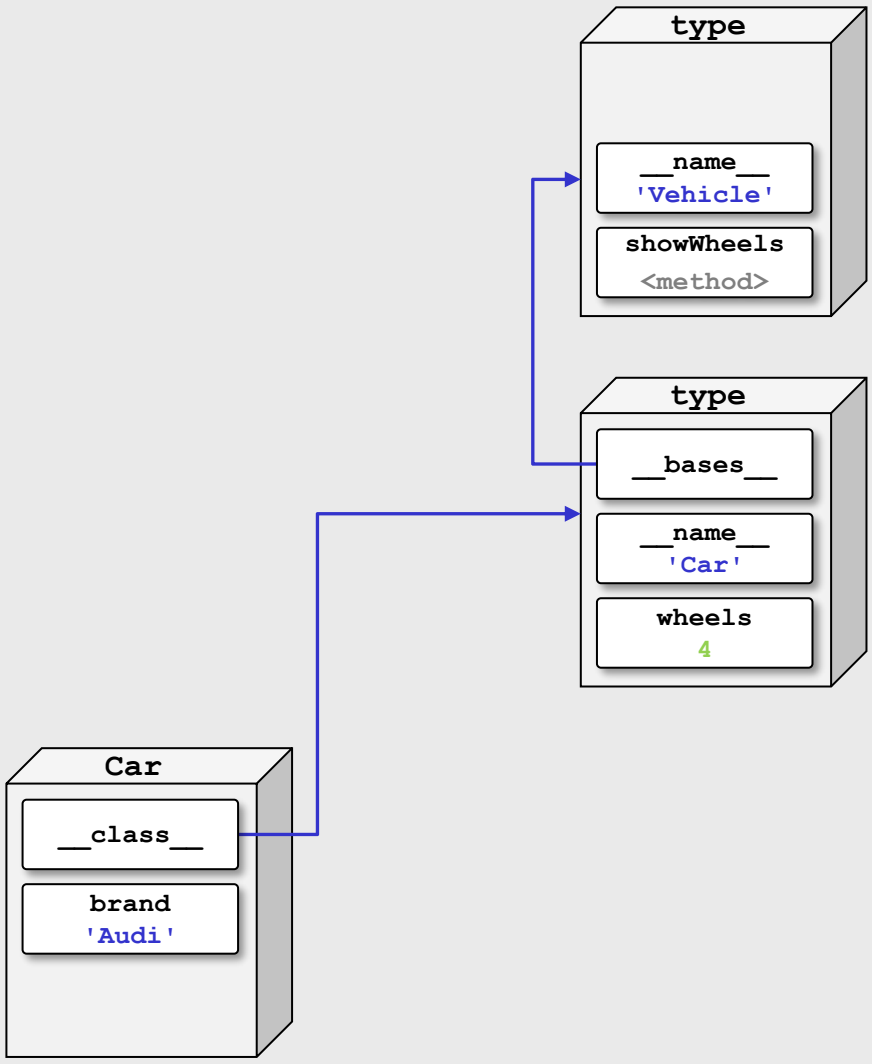



```
class Car:
    wheels = 4
    def showWheels(self):
        print('# wheels:',
              self.wheels)

vehicle = Car()
vehicle.brand = 'Audi'
vehicle.wheels = 5
vehicle.showWheels()
```

5

Inheritance

objects (main memory)

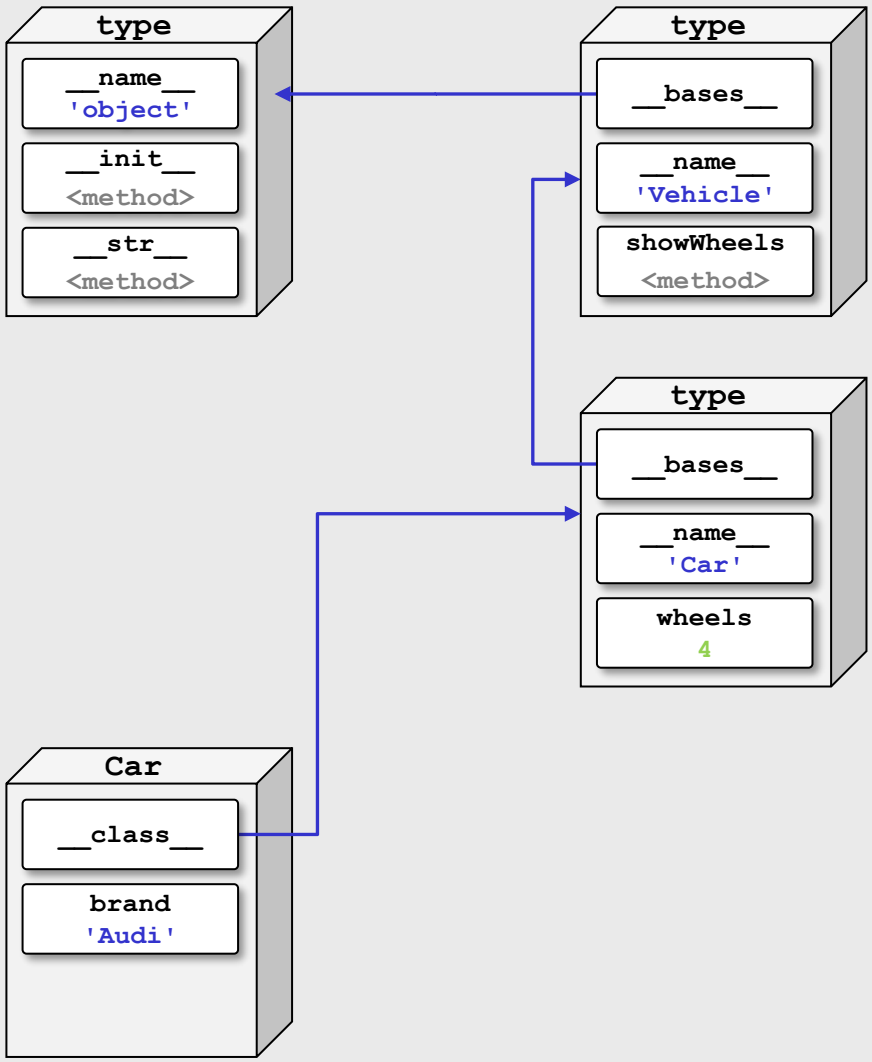



```
class Vehicle:
    def showWheels(self):
        print('# wheels:',
              self.wheels)

class Car(Vehicle):
    wheels = 4

vehicle = Car()
vehicle.brand = 'Audi'
vehicle.showWheels()
```

4

Inheritance

objects (main memory)




```
class Vehicle(object):
    def showWheels(self):
        print('# wheels:',
              self.wheels)

class Car(Vehicle):
    wheels = 4

vehicle = Car()
vehicle.brand = 'Audi'
vehicle.showWheels()
```

4

Classes: 'new style' vs 'old style'

- Python 2.x
  - differentiates between 'new style' and 'old style' classes
  - most differences are in advanced (technical) details

```
class Point:
    pass
```
old style

```
class Point(object):
    pass
```
new style

- Python 3.x
  - no difference between 'new style' and 'old style' classes

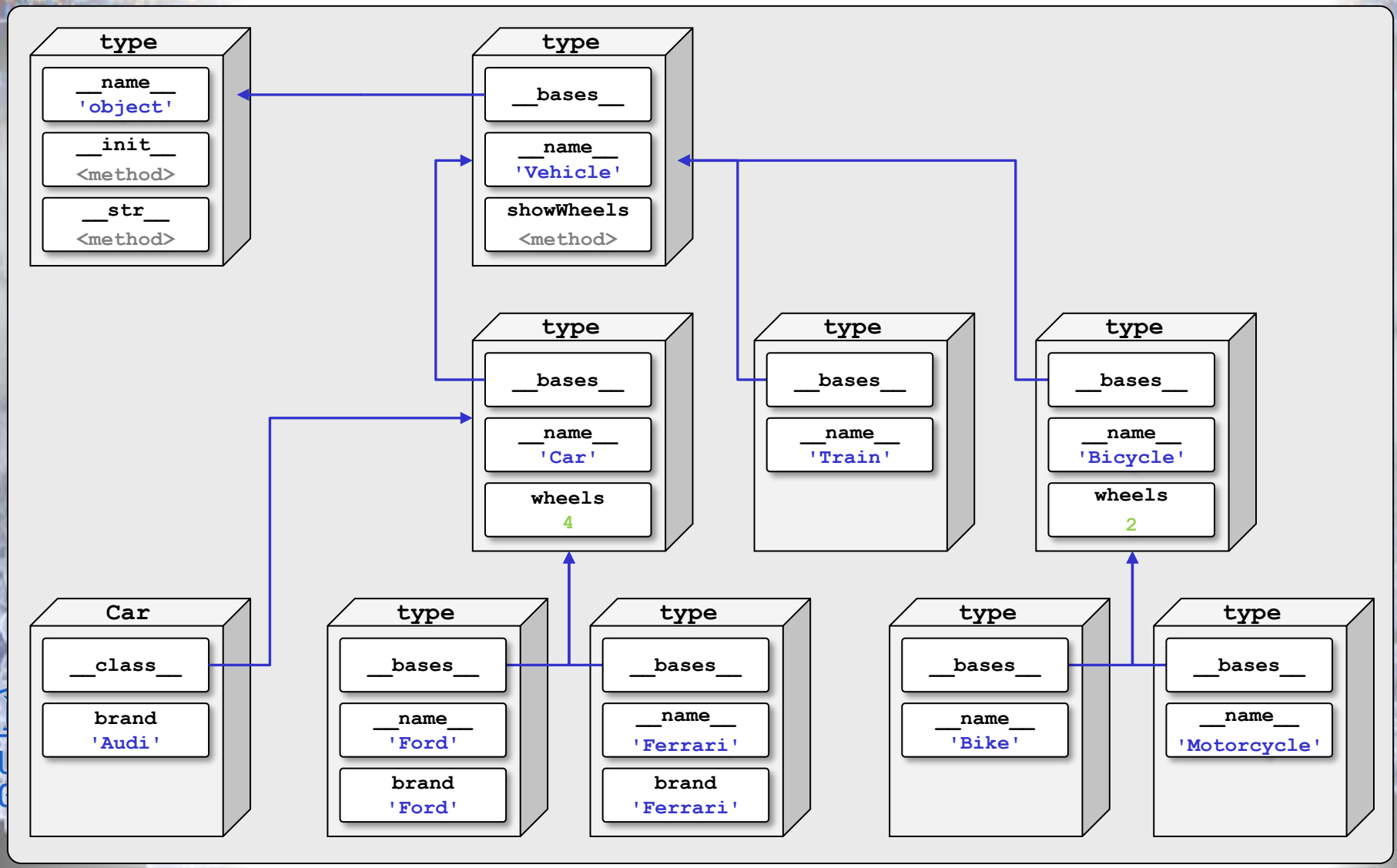
```
class Point:
    pass
```
new style

```
class Point(object):
    pass
```
new style

UNIVERSITEIT GENT

Inheritance

objects (main memory)



Inheritance

```
class Vehicle:
  "base class for vehicles"

  brand = 'unknown'
  wheels = None

  def __init__(self, brand=None, wheels=None):
    if not brand is None: self.brand = brand
    if not wheels is None: self.wheels = wheels

  def __str__(self):
    return 'A {} of brand {} with {} wheels'.format(
      self.__class__.__name__, self.brand, self.wheels)

  def showWheels(self):
    if self.wheels is None:
      print('Unknown number of wheels')
    else:
      print('Number of wheels: {}'.format(self.wheels))
```

```
class Car(Vehicle):
    wheels = 4

class Ford(Car):
    brand = 'Ford'

class Ferrari(Car):
    brand = 'Ferrari'

class Train(Vehicle):
    pass

class Bicycle(Vehicle):
    wheels = 2

class Motorcycle(Bicycle):
    pass

class Bike(Bicycle):
    pass
```

```
>>> bike = Bike(brand='Merckx')
>>> bike.brand
'Merckx'
>>> bike.showWheels()
Number of wheels: 2
>>> print(bike)
A Bike of brand Merckx with 2 wheels
```

UNIVERSITEIT GENT

Inheritance

```
class Vehicle:
  "base class for vehicles"

  brand = 'unknown'
  wheels = None

  def __init__(self, brand=None, wheels=None):
    if not brand is None: self.brand = brand
    if not wheels is None: self.wheels = wheels

  def __str__(self):
    return 'A {} of brand {} with {} wheels'.format(
      self.__class__.__name__, self.brand, self.wheels)

  def showWheels(self):
    if self.wheels is None:
      print('Unknown number of wheels')
    else:
      print('Number of wheels: {}'.format(self.wheels))
```

```
class Car(Vehicle):
    wheels = 4

class Ford(Car):
    brand = 'Ford'

class Ferrari(Car):
    brand = 'Ferrari'

class Train(Vehicle):
    pass

class Bicycle(Vehicle):
    wheels = 2

class Motorcycle(Bicycle):
    pass

class Bike(Bicycle):
    pass
```

```
>>> Car = Ferrari()
>>> Car.brand
'Ferrari'
>>> Car.showWheels()
Number of wheels: 4
>>> print(Car)
A Ferrari of brand Ferrari with 4 wheels
```

UNIVERSITEIT GENT

Inheritance

```
class Vehicle:
  "base class for vehicles"

  brand = 'unknown'
  wheels = None

  def __init__(self, brand=None, wheels=None):
    if not brand is None: self.brand = brand
    if not wheels is None: self.wheels = wheels

  def __str__(self):
    return 'A {} of brand {} with {} wheels'.format(
      self.__class__.__name__, self.brand, self.wheels)

  def showWheels(self):
    if self.wheels is None:
      print('Unknown number of wheels')
    else:
      print('Number of wheels: {}'.format(self.wheels))
```

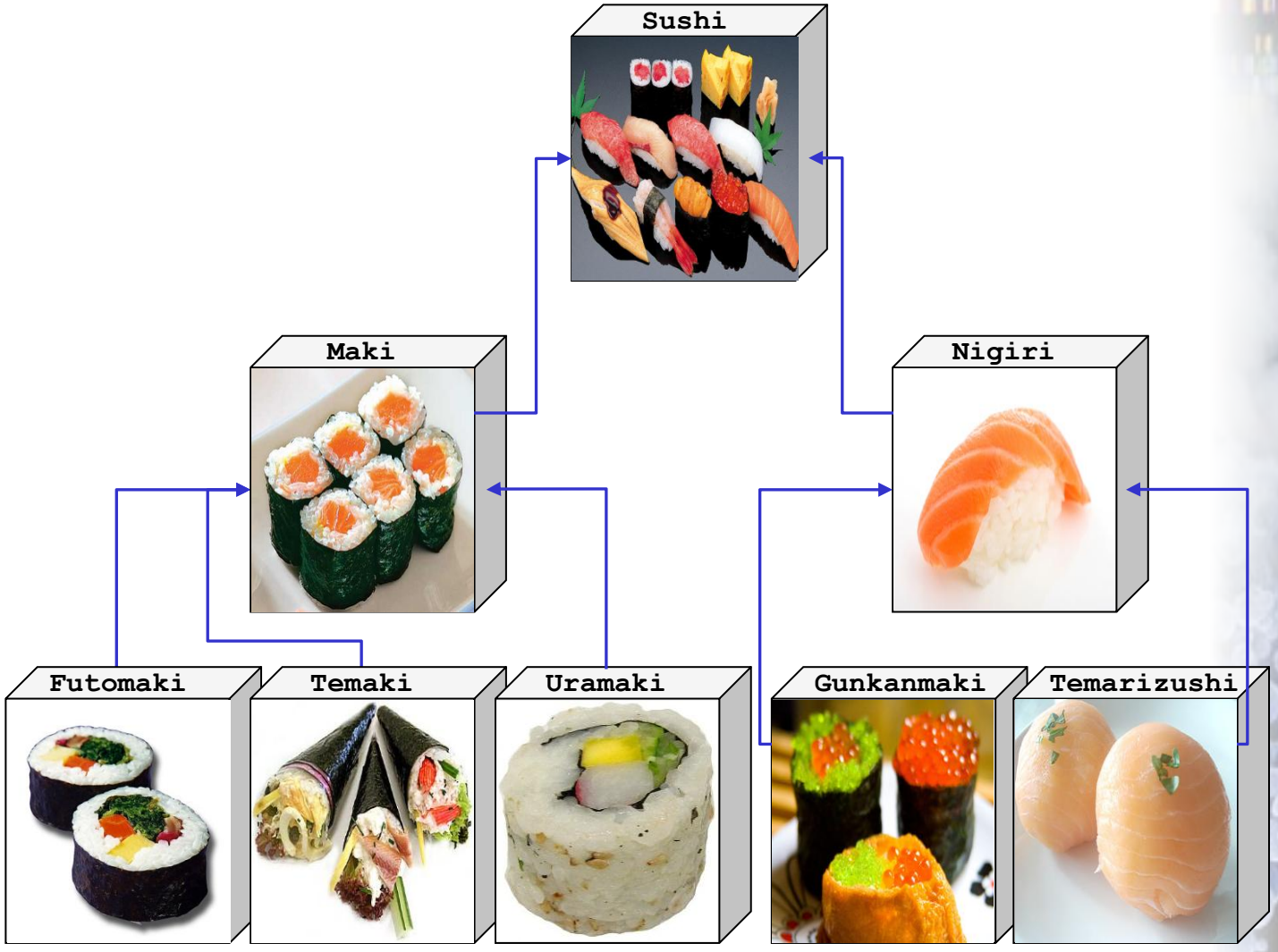
```
class Car(Vehicle):
    wheels = 4

class Ford(Car):
    brand = 'Ford'

class Ferrari(Car):
    brand = 'Ferrari'

class Train(Vehicle):
    pass

class Bicycle(Vehicle):
    wheels = 2

class Motorcycle(Bicycle):
    pass

class Bike(Bicycle):
    pass
```

```
>>> boat = Vehicle(brand='Ocean', wheels=0)
>>> boat.brand
'Ocean'
>>> boat.showWheels()
Number of wheels: 0
>>> print(boat)
A Vehicle of brand Ocean with 0 wheels
```

UNIVERSITEIT GENT

Sushi

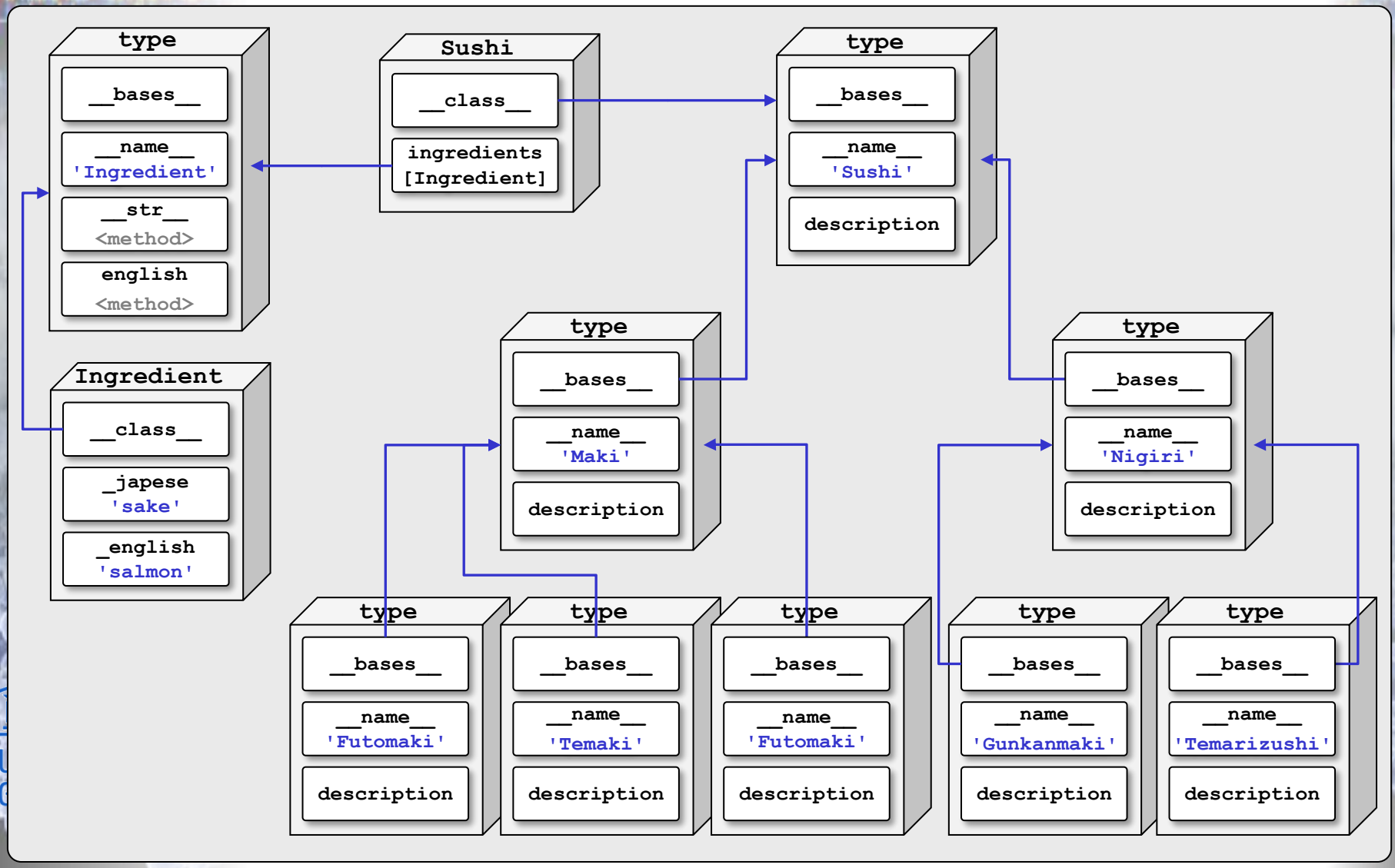
Sushi hierarchy



UNIVERSITEIT GENT

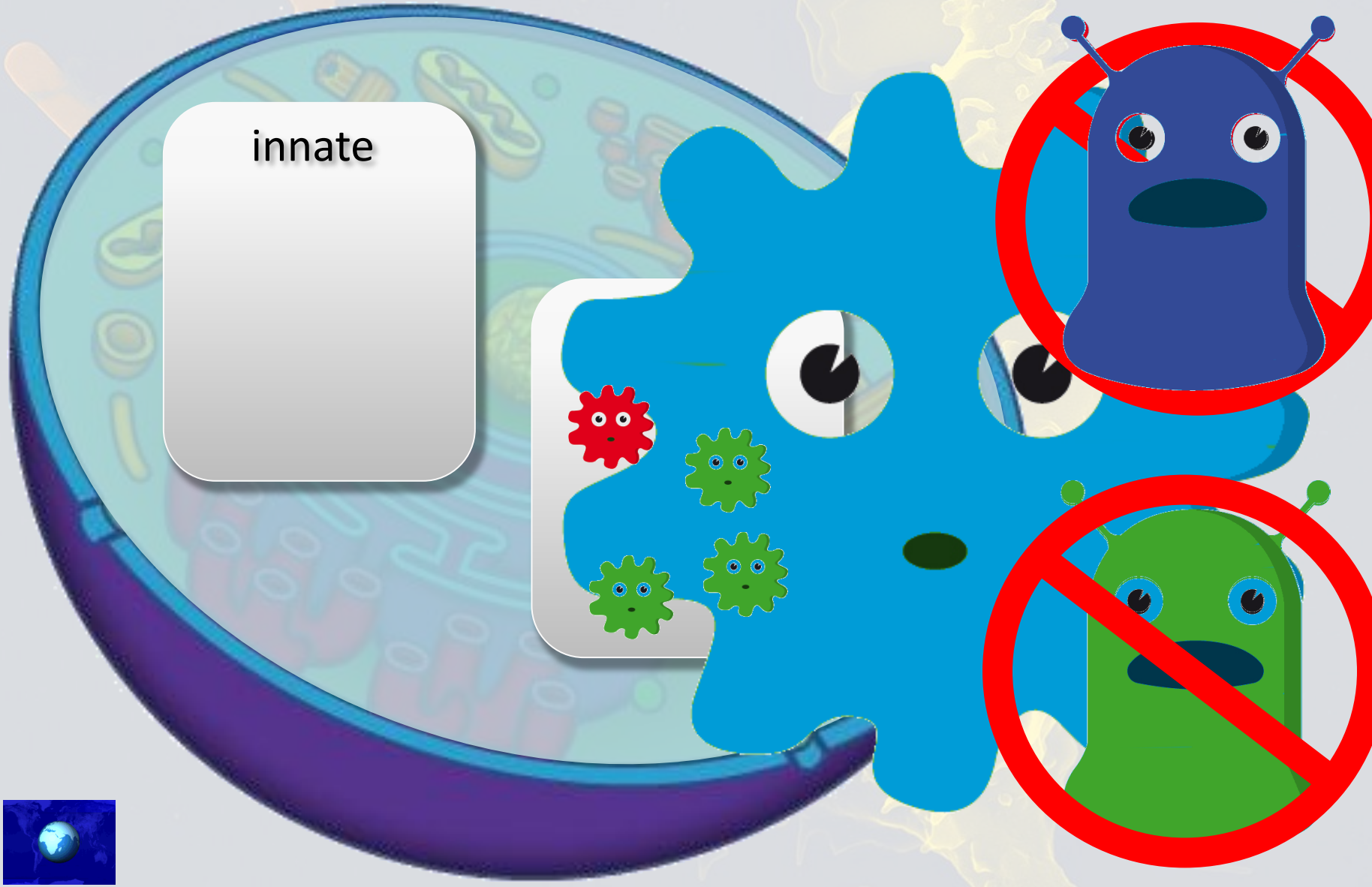
Sushi hierarchy

objects (main memory)



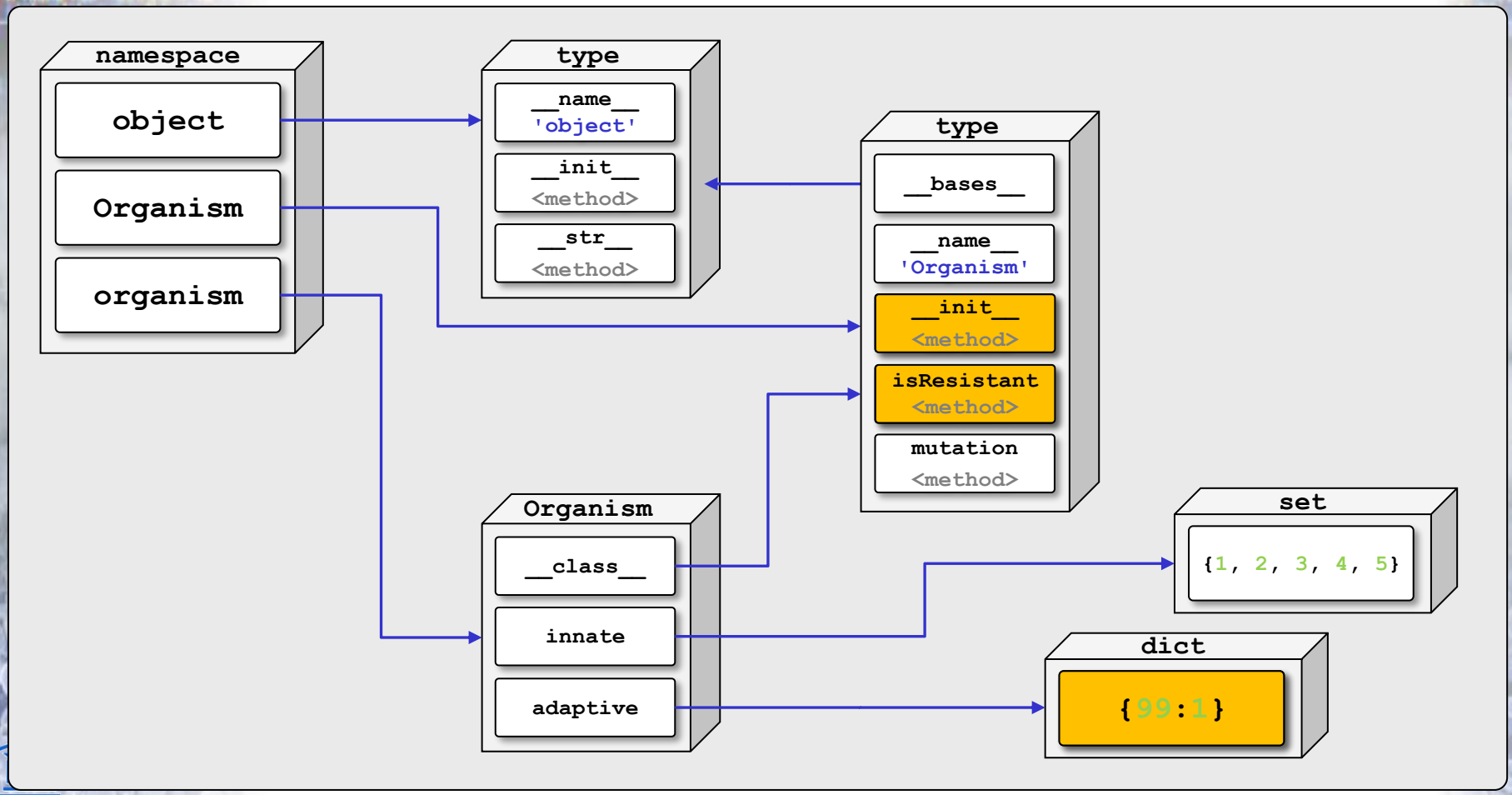
Immune system

innate



Immune system

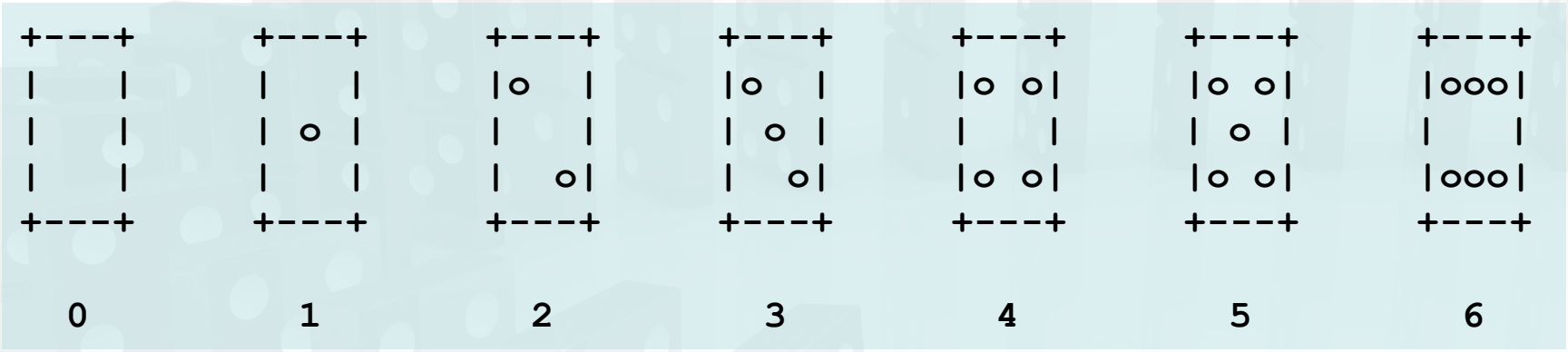
objects (main memory)

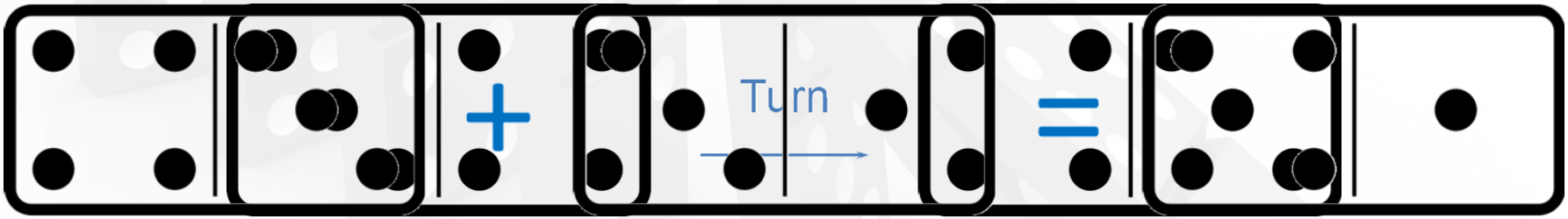


UNIVERSITEIT GENT

```
>>> organism.isResistant(virus=99)
```

Domino tiles





Quilts

`Quilt(2, 2, '//++')`

/	/
+	+

+

`Quilt(2, 2, '-\\||')`

-	\
|	|

=

`Quilt(2, 4, '//-\\++||')`

/	/	-	\
+	+	|	|

turn

`Quilt(4, 2, '+\\+\\-|-/')`

+	\
+	\
-	|
-	/

Questions or remarks?

UNIVERSITEIT GENT

The sky is the limit …

"When a task cannot be partitioned because of sequential constraints, the application of more effort has no effect on the schedule.

The bearing of a child takes nine months, no matter how many women are assigned.

Many software tasks have this characteristic because of the sequential nature of debugging."

— Frederick P. Brooks, Jr.
The Mythical Man-Month

UNIVERSITEIT GENT