

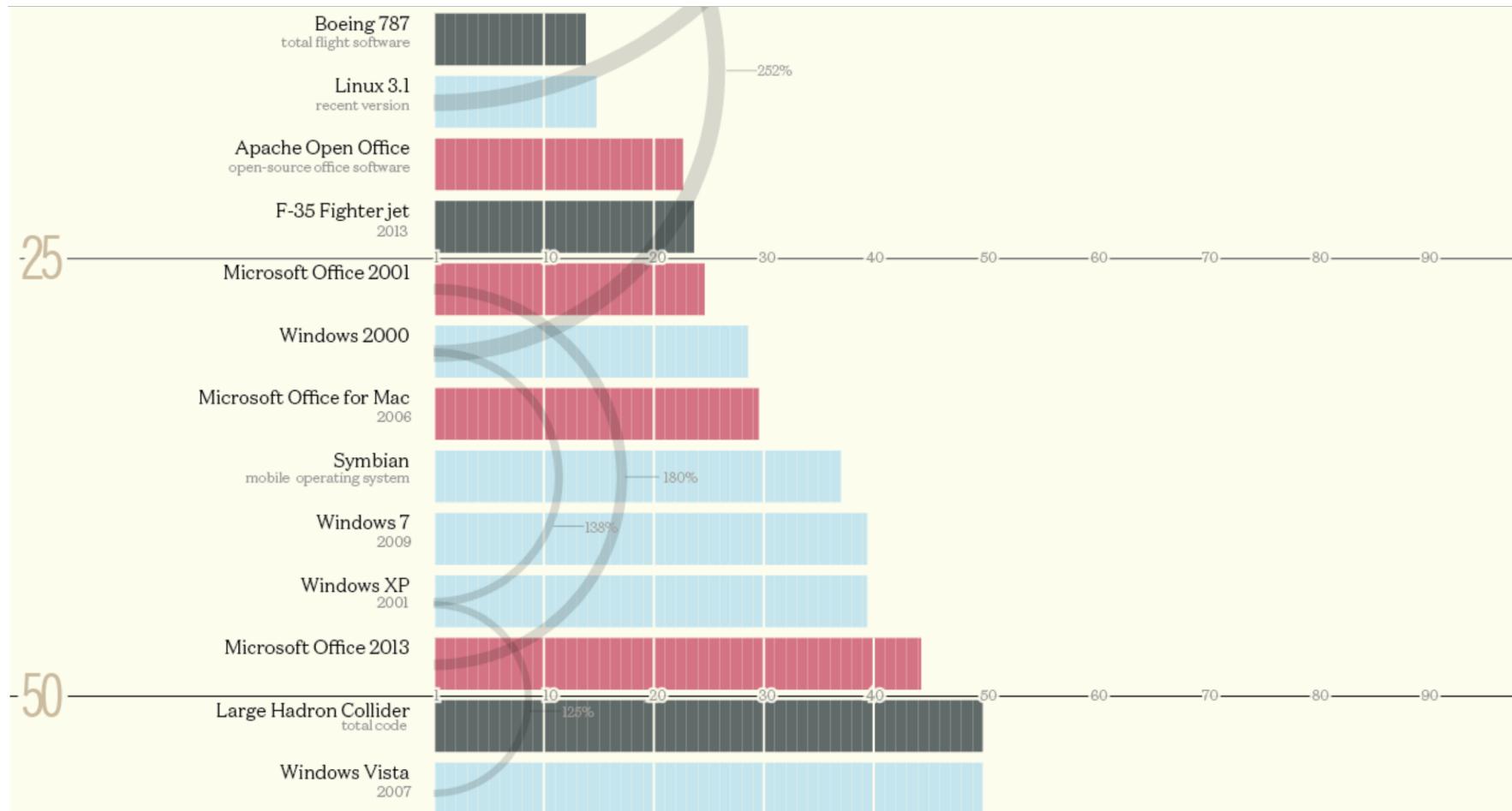
Introduction to Object Oriented Programming

Università di Modena e Reggio Emilia

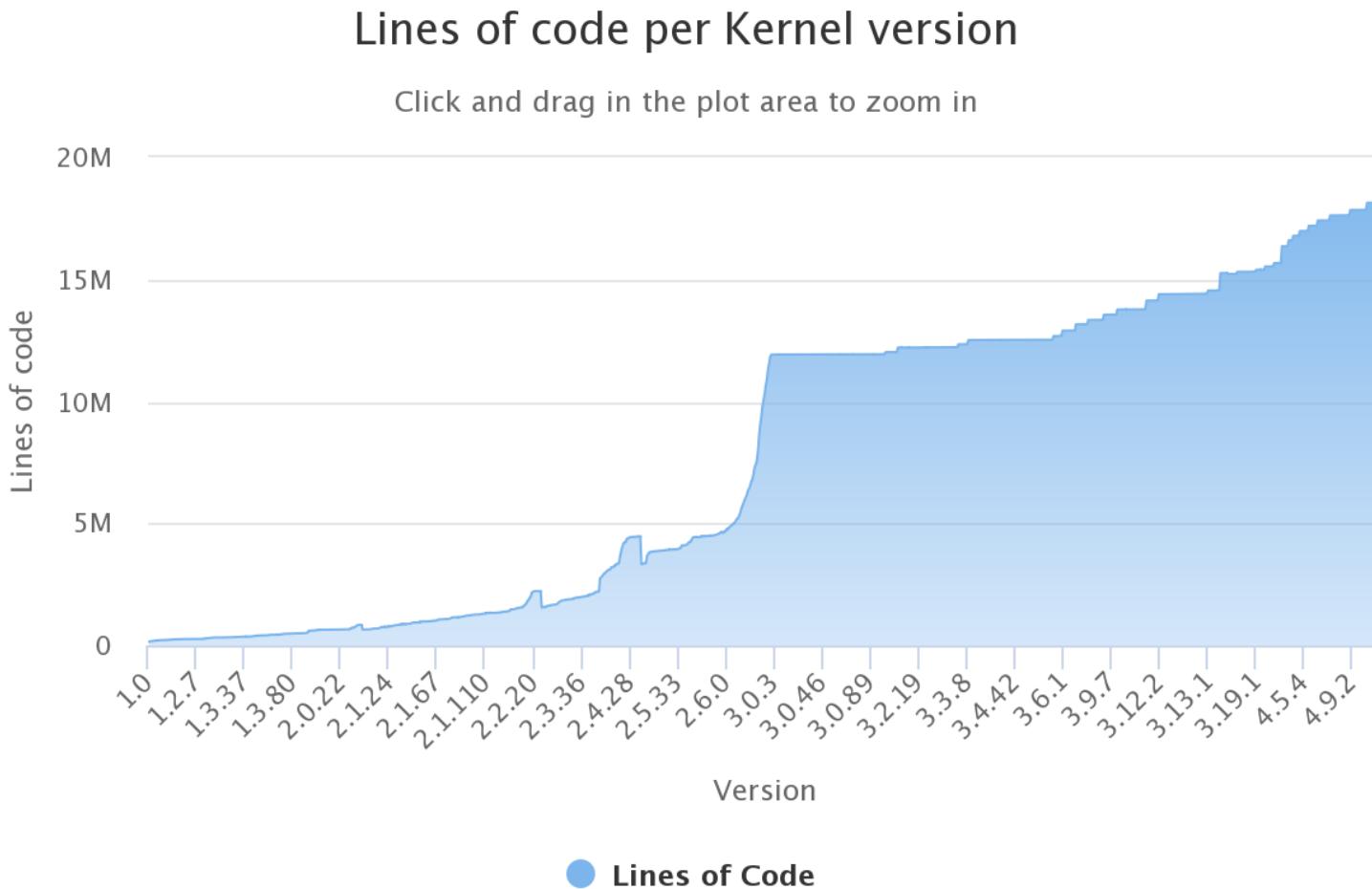
Prof. Nicola Bicocchi (nicola.bicocchi@unimore.it)



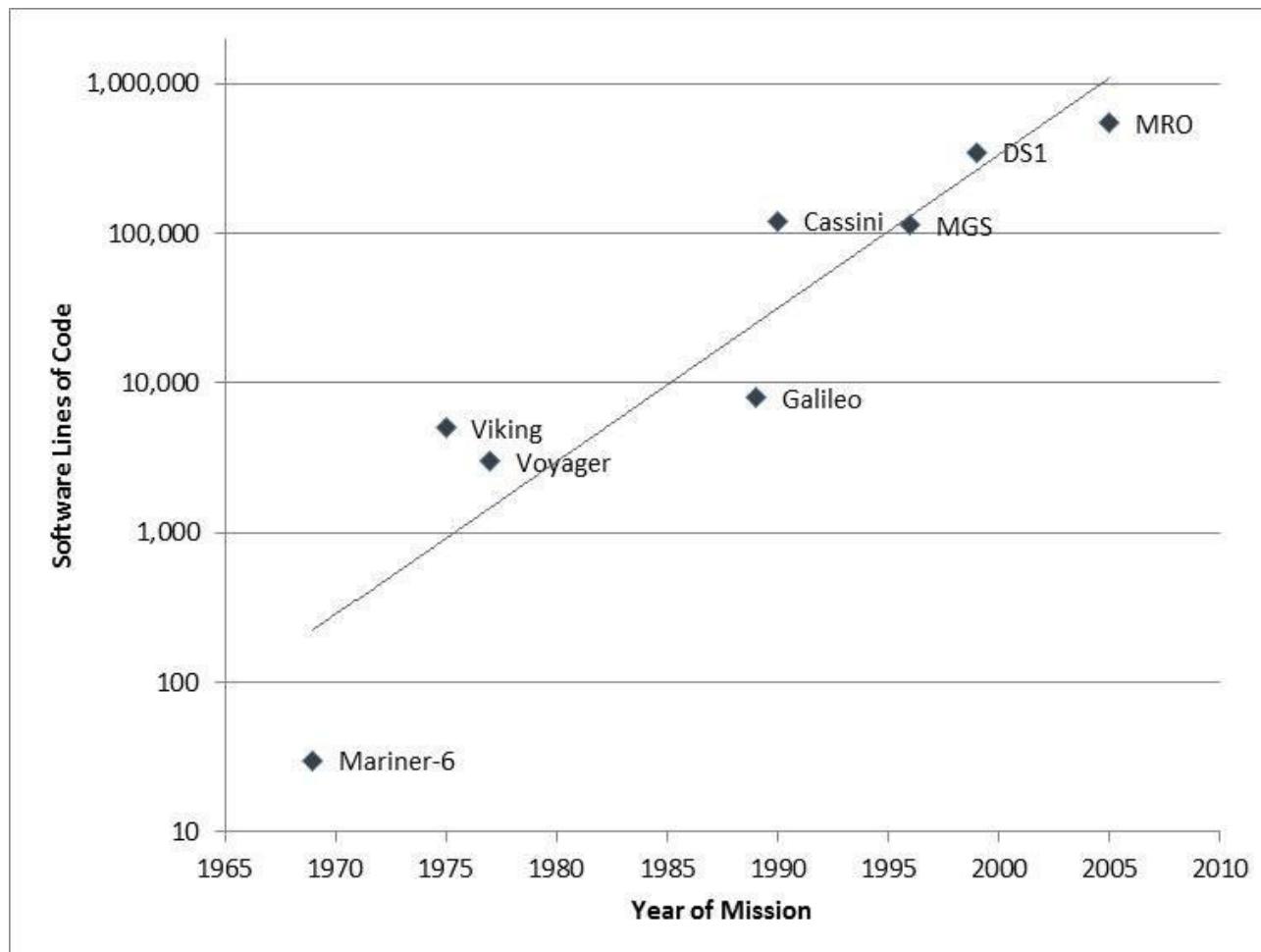
Software Size



Software Size



Software Size



Why OOP?



Why OOP?

- Procedural programming languages (e.g., Pascal, C) are **not suitable for building large software infrastructures**
- OOP addresses this issue and **reduces development and maintenance costs for large and complex software projects**

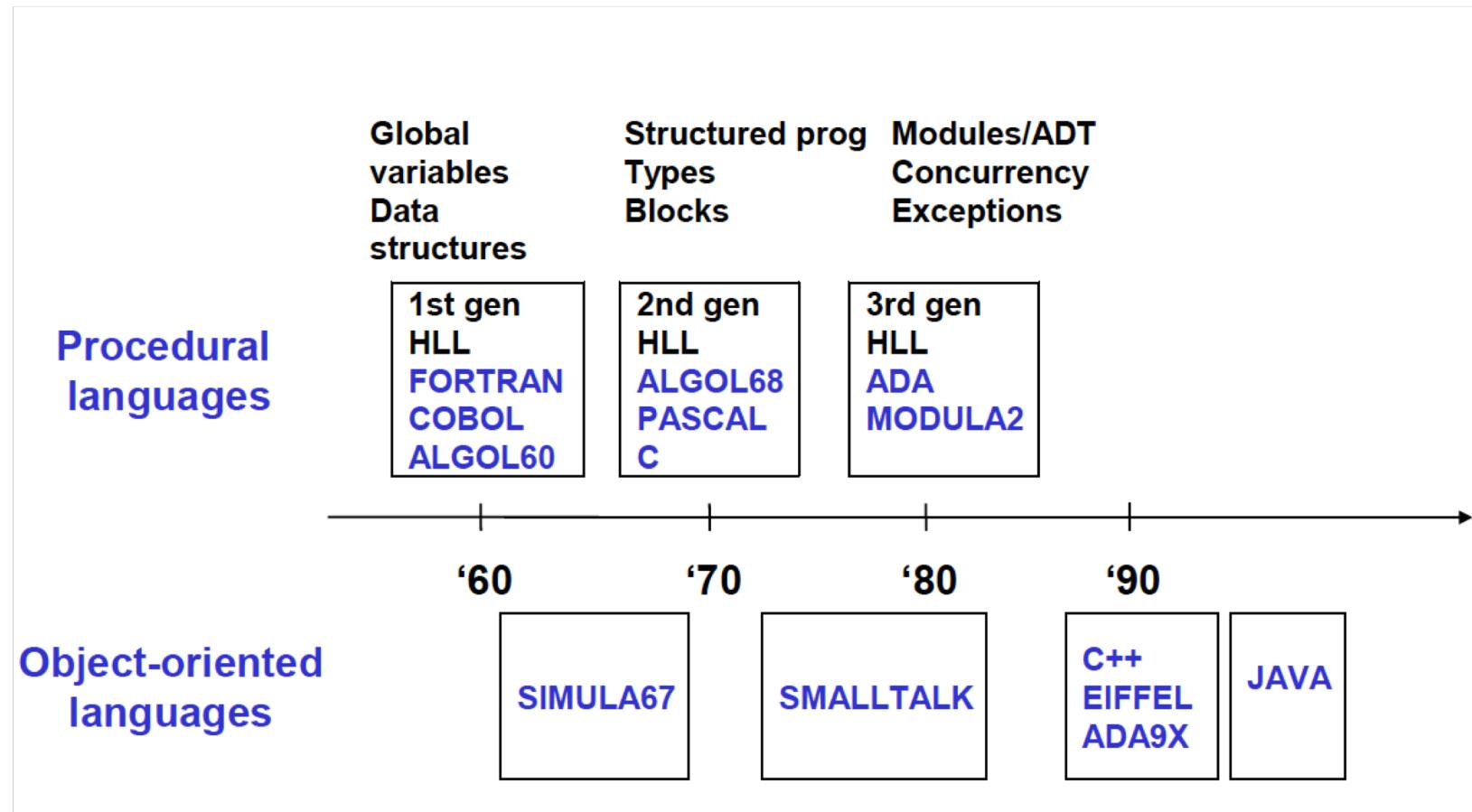
Errors / 1K SLOC

- **Industry Average**
 - 25 errors / 1K SLOC
- **Corporate Applications**
 - 5 errors / 1K SLOC
- **Cleanroom development technique**
 - 0.5 errors / 1K SLOC

Software crisis (1970)

- The causes of the software crisis were linked to the **overall complexity of hardware and the software development process**. The crisis manifested itself in several ways:
 - Projects running over-budget
 - Projects running over-time
 - Software was inefficient
 - Software was difficult to maintain

Languages Timeline



OOP Goal

- Build software being secure, re-usable, flexible, documentable, extensible
- Instead of focusing on algorithms, optimization and efficiency, **OOP focus on programming techniques**
- OOP considers software as a set of well-defined entities containing both data and functions

Procedural Programming

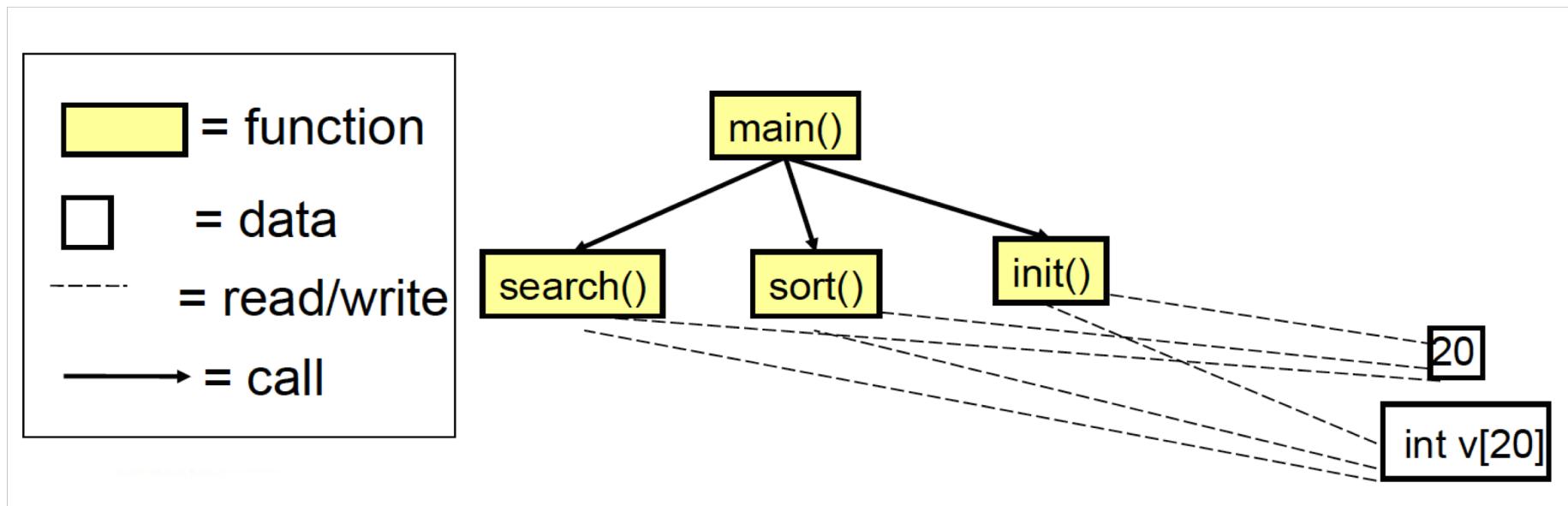
```
int vect[20];
int i;

void sort()      { /* sort */ }
int search(int n) { /* search */ }
void init()       { /* init */ }

void main() {
    init();
    sort();
    search(13);
}
```

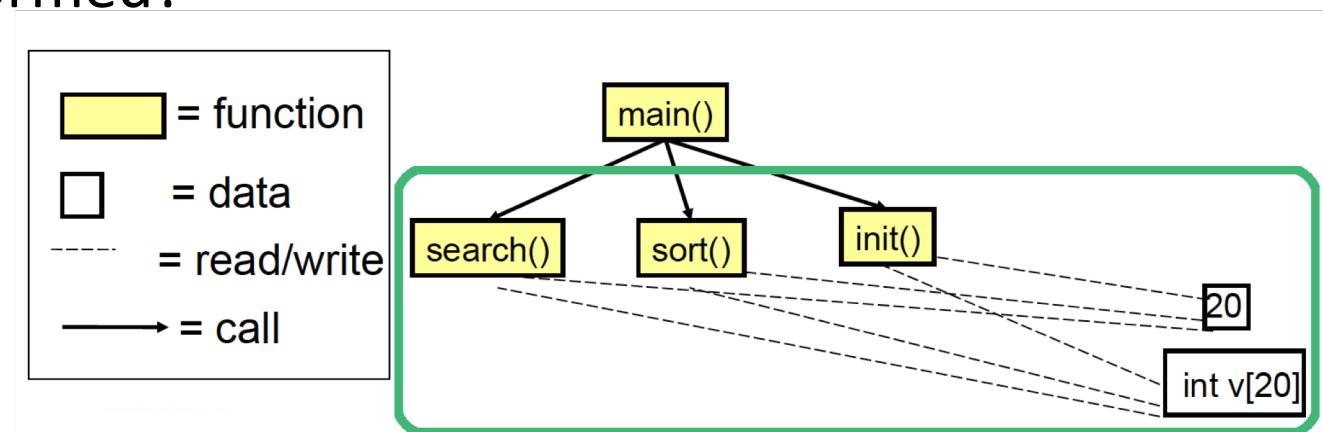


Modules and relationships



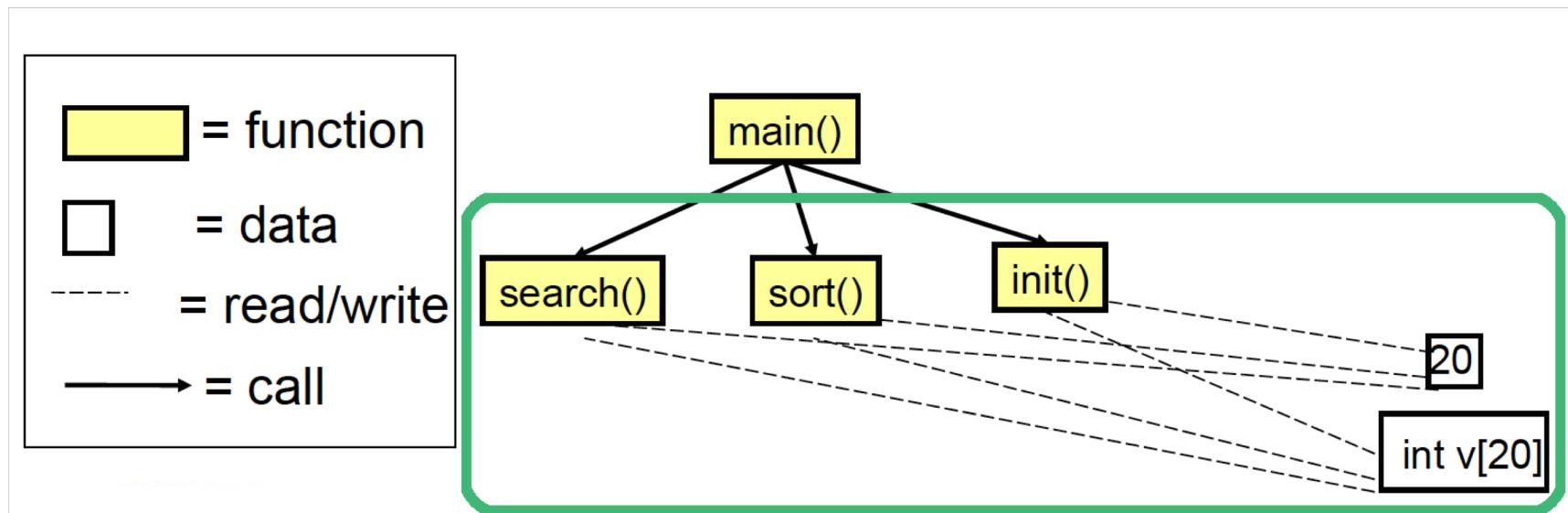
Issues

- There is no clear relationship between:
 - Vectors (`int vect[20]`)
 - Operations on vectors (`search`, `sort`, `init`)
- There is no control over size:
 - `for (i=0; i<=20; i++) { vect[i]=0; };`
- Initialization
 - Actually performed?



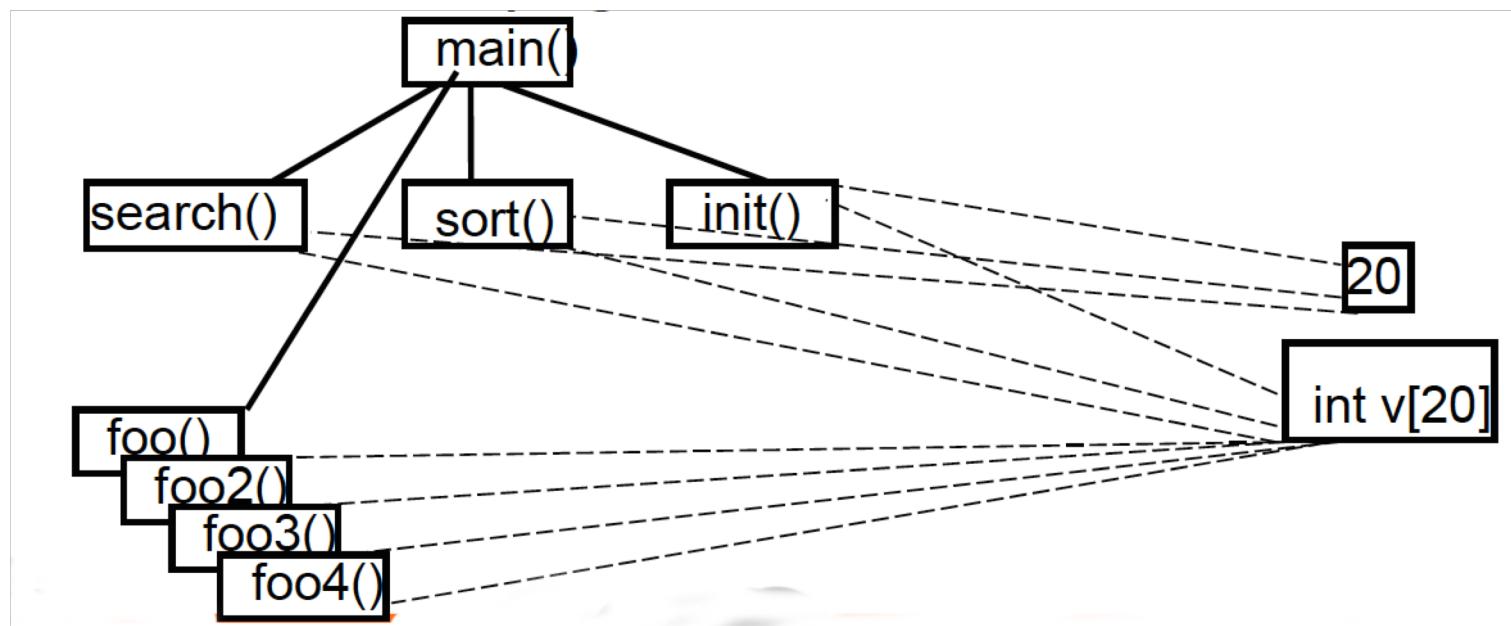
Issues

- It's not possible to consider a vector as a primitive concept
- Data and functions are not modularized



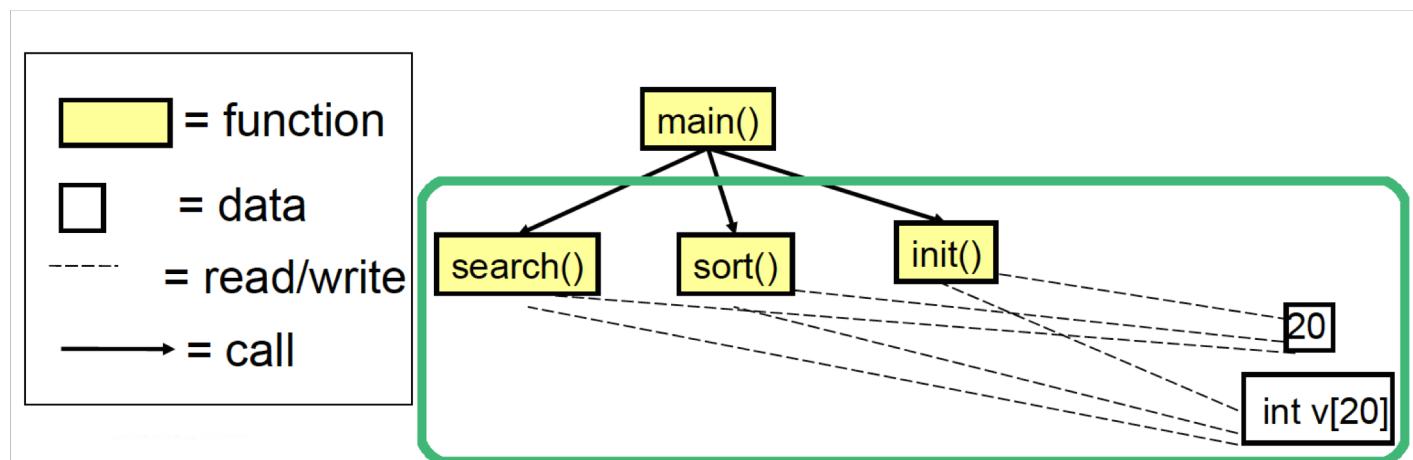
Issues, the long run

- External functions can read/write vector's data, leading to a **growing number of relationships**
- Source code becomes **difficult to understand and maintain**
- **Spaghetti Code**



Issues

- **Reuse of code limited**
 - Data and procedures (functions) are separate. This makes it complex to reuse existing code in other projects
- **Data protection limited**
 - Unprotected data accessible from vast portions of the source code. After a certain stage, debug becomes a nightmare!
- **Unsuitable for decomposition**
 - Large scale projects require large scale working force (many teams). Unprotected data, separate from functions makes it hard to decompose



Object-Oriented approach

```
class Vector {  
    // internal data  
    private int v[20];  
  
    // external interface  
    public Vector() {  
        /* init */  
        for(int i=0; i<20; i++) v[i]=0;  
    }  
  
    public sort()          { /*sort*/ }  
    public search(int c) { /*search*/ }  
}
```



Object-Oriented approach

```
Vector v1 = new Vector();  
Vector v2 = new Vector();
```

```
v1.sort();  
v1.search(22);
```

```
v1++;           // Error: not an integer  
v1.v[2] = 47;  // Error: v[] is encapsulated
```

An engineering approach

- Client-Server model
- OOP implies a paradigm shift
 - Do not use functions for processing data entities
 - Ask entities to deliver services

operation(object, params) *object.operation(params)*

For example:

insert(vector, element)

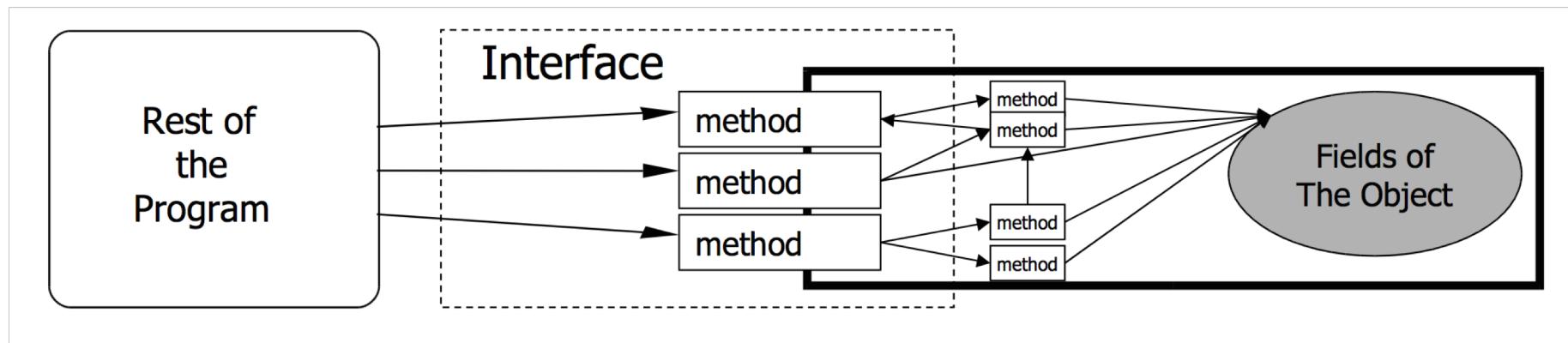
For example:

vector.insert(element)



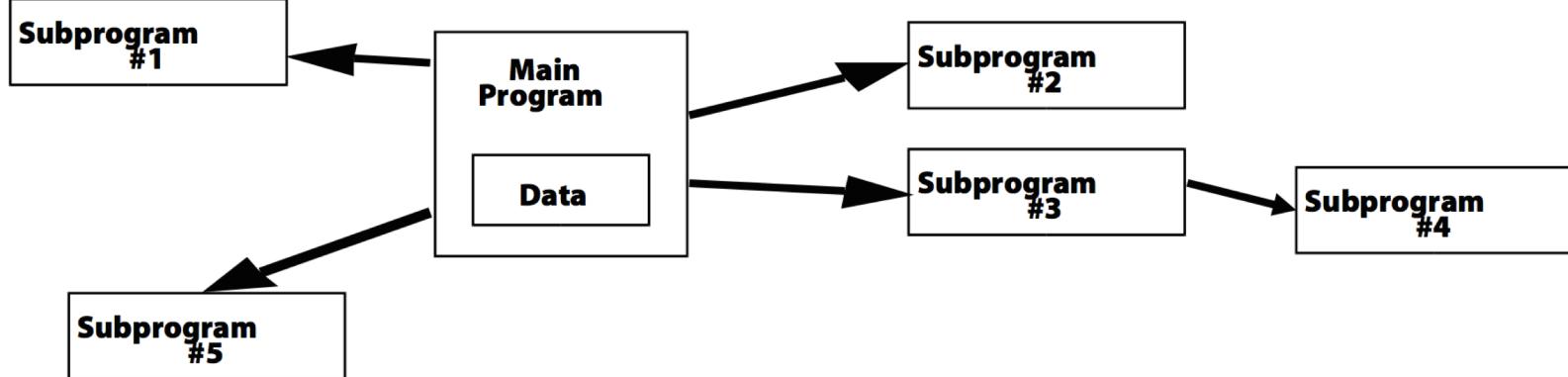
An engineering approach

- Given a system, we have to:
 - Identify the components
 - Define component interfaces
 - Define how components interact each other through their interfaces
 - Minimize relationships among components

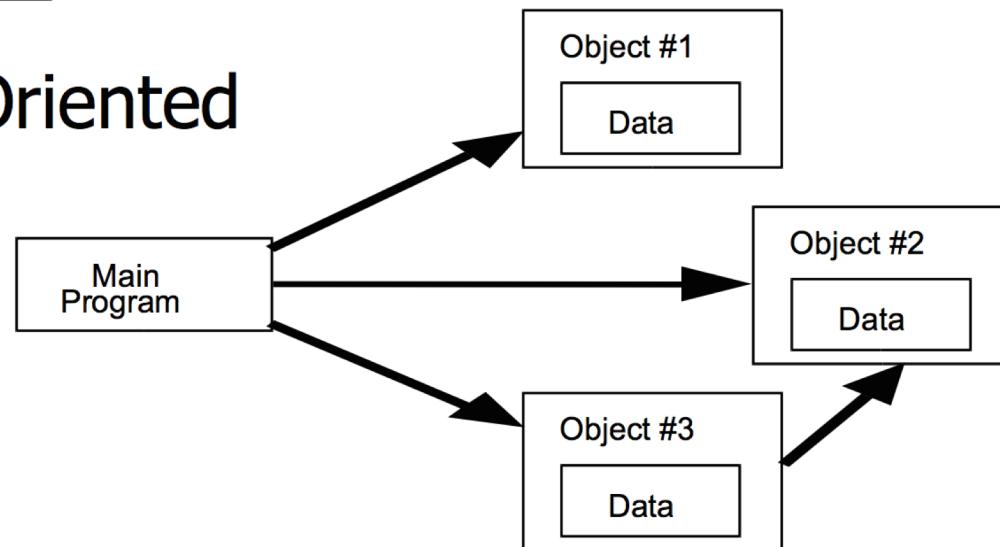


An engineering approach

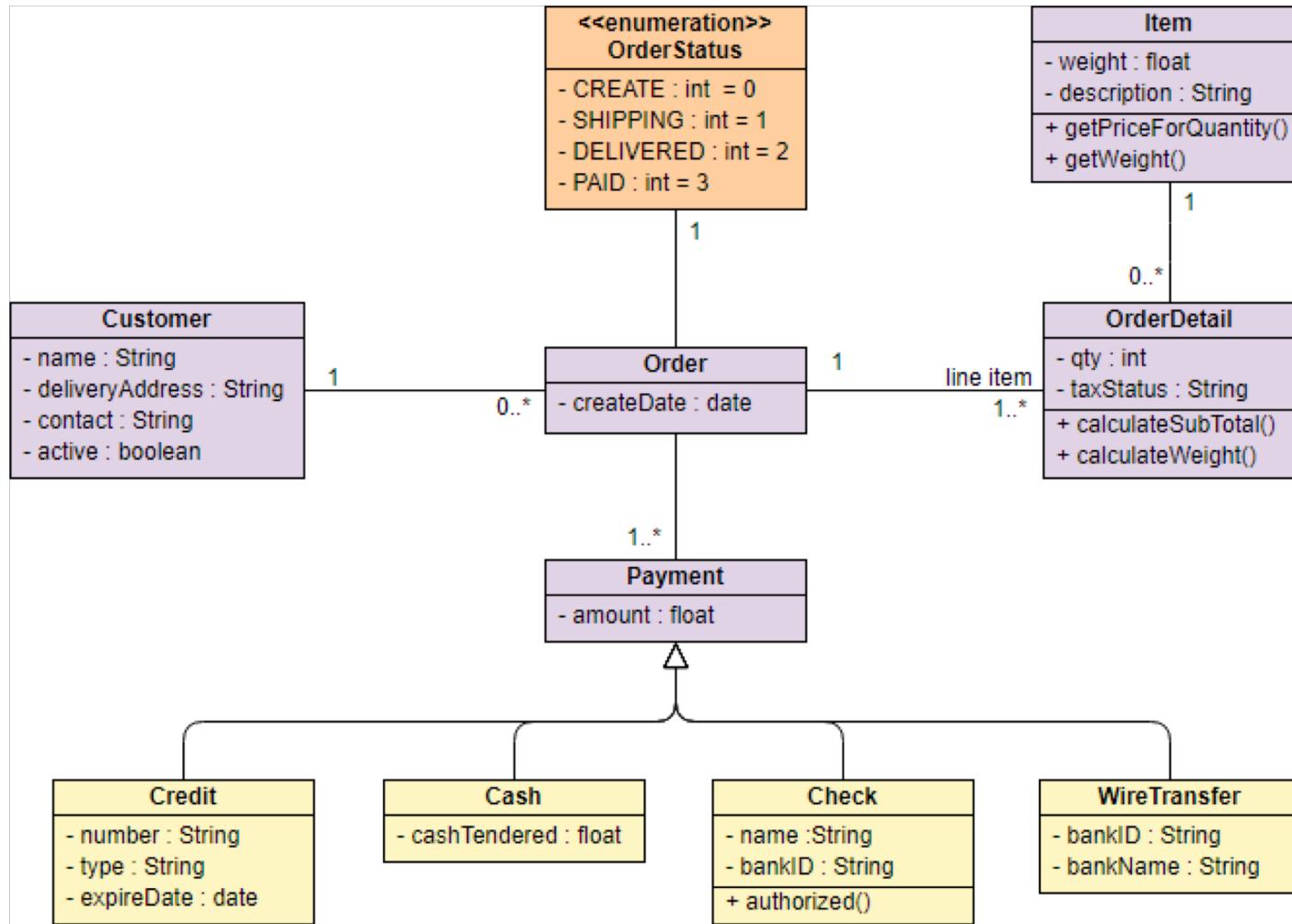
Procedural



Object Oriented

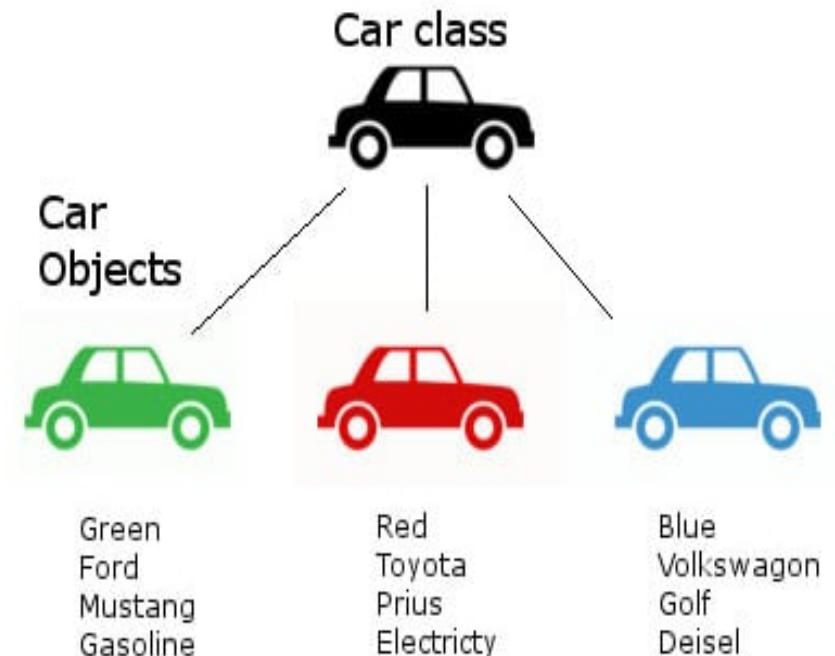


An engineering approach



Class and object

```
public class Car {  
    String color;  
    String brand;  
    String model;  
    String fuel;  
  
    public Car(color, brand, model, fuel) {  
        this.color = color;  
        this.brand = brand;  
        this.model = model;  
        this.fuel = fuel;  
    }  
    /* ... */  
}  
  
Car c1 = new Car(Green, Ford, Mustang, Gasoline);  
Car c2 = new Car(Red, Toyota, Prius, Electricity);  
Car c3 = new Car(Blue, VW, Golf, Diesal);
```



Class and object

- **Class** (the description of object structure, i.e. type):
 - Data (**ATTRIBUTES** or **FIELDS**)
 - Functions (**METHODS** or **OPERATIONS**)
 - Creation methods (**CONSTRUCTORS**)
- **Object** (class instance)
 - Identity
 - State

Class and object

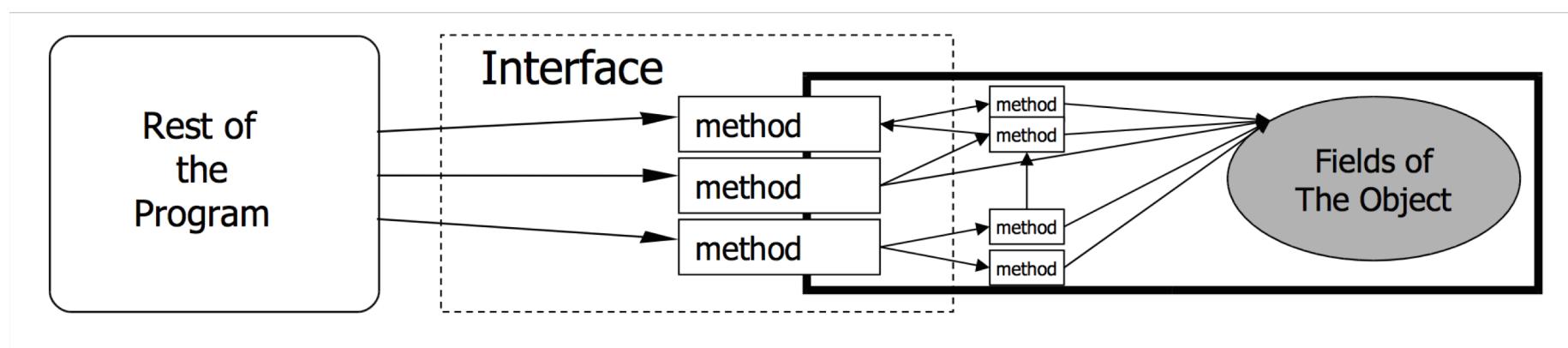
- A **class is like a type definition**. No data is allocated until an object is created from the class
- The creation of an object is called **instantiation**. The created object is often called an instance
- No limit to the number of objects that can be created from a class
- **Each object is independent**. Changing one object doesn't change the others

OOP Key Features

- Encapsulation
- Inheritance
- Polymorphism

Encapsulation

- Each object “wraps” code and data
- Each object handles its own data
- Other objects can use the object’s interface to require services



Encapsulation

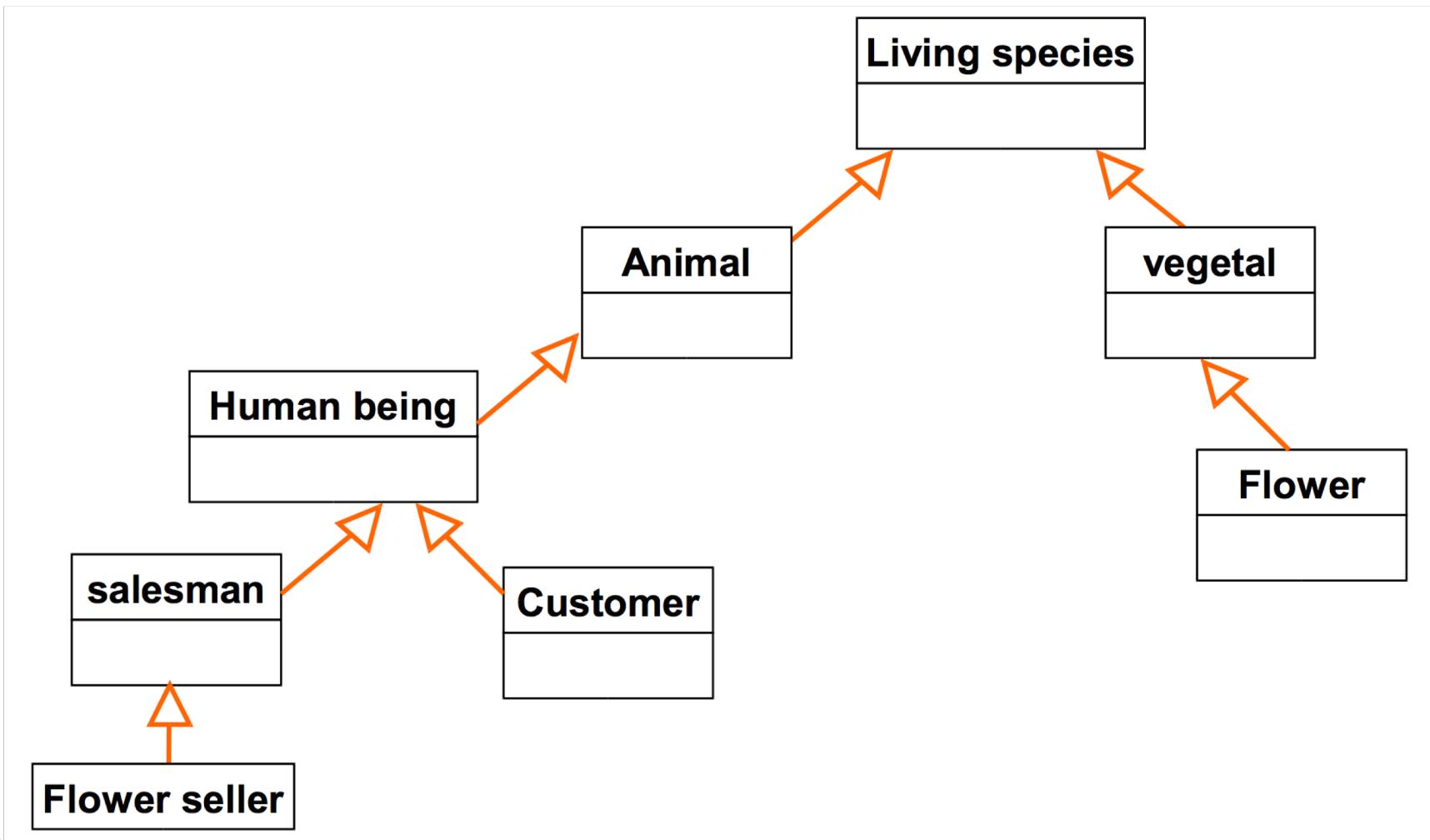
```
class MyVector {  
    // internal data (encapsulated data)  
    private int v[20];  
  
    // external interface (methods)  
    public MyVector() {  
        for(int i=0; i<20; i++) v[i]=0;  
    }  
  
    public sort()          { /*sort*/ }  
    public search(int c) { /*search*/ }  
}
```



Inheritance

- A class can derive from another class by extending it
- The derived class inherits variables and methods of the base class. The child class adds its own variables and methods
- Simplify the code reuse, build relations among classes

Inheritance



Inheritance

```
public class LivingSpecies {  
    private boolean isAlive;  
}  
  
public class Animal extends LivingSpecies {  
    . . .  
}  
  
public class HumanBeing extends Animal {  
    . . .  
}
```



Polymorphism

- Same requests for a method may lead to different behaviors depending on:
 - The actual object performing the operation (e.g., base class vs derived class)
 - Type of parameters passed as argument
 - Execution context

Polymorphism

```
class MyVector {  
    // internal data (encapsulated data)  
    private int v[20];  
  
    public MyVector() {  
        for(int i=0; i<20; i++) v[i]=0;  
    }  
  
    public MyVector(int[] v) {  
        for(int i=0; i<20; i++) this.v[i]=v[i];  
    }  
  
    public sort()          { /*sort*/ }  
    public search(int c) { /*search*/ }  
}
```



Advantages of OOP

- Simplify the process of building software in a cooperative manner:
 - Different people developing different classes (on the same project!)
- Simplify code management
 - Bugs on object data are easy to spot. Since data are not visible from the outside, errors mostly occurs in the object handling the data
 - Changes on a specific class do not impact other classes (unless the interface is not modified)
- Support incremental design and development
 - Define new classes by extending the existing ones

Costs of OOP

- Needs a Object Oriented way of thinking
- Complex design (e.g., Which classes?, How many classes?)
- Benefits only occur in large programs
 - Programs < 100 lines, spaghetti is understandable and faster to write
 - Programs > 1K lines, spaghetti is incomprehensible, probably doesn't work, not maintainable