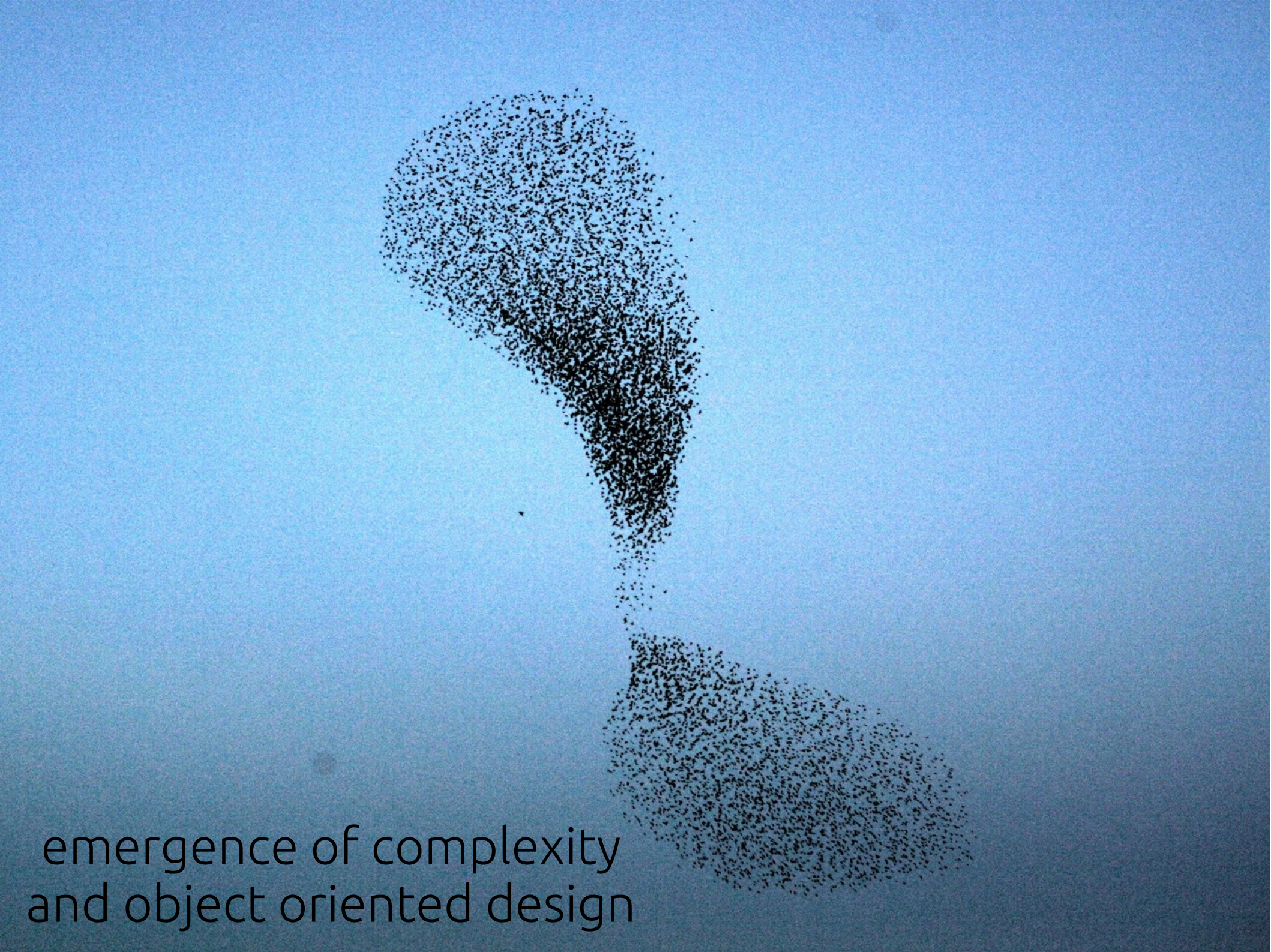
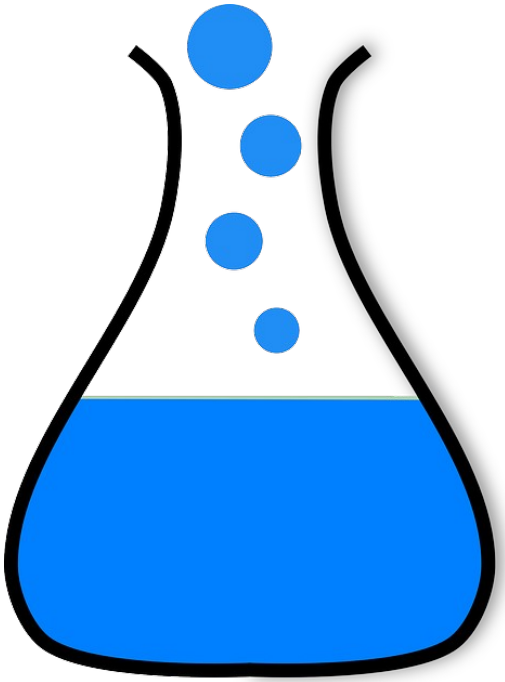

emergence of complexity
and object oriented design

term 2

- news? update from projects?
- more complex topics
- more open-ended lab assignments
- work from home on prototypes
- demo in class
- mocap



bird flocks

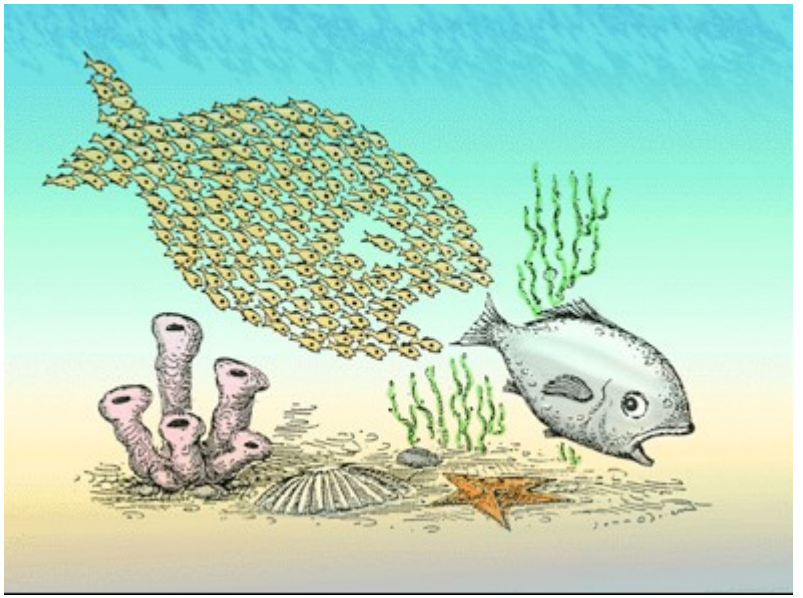
- individuals are not aware what the group is doing
- there is no leader
- the forms of the group are complex but they are based on a small number of simple rules on “bird level”:
  - watch what your neighbours are doing
  - have a tendency towards the center
  - avoid incoming birds

emergence of complexity

- simple rule at a low level
- organized complexity at higher levels
- resembles the Aristotelian:

  “the whole is greater than the sum of its parts”

- Complex systems can appear which are based on a great number of simple/small interactions



termite nest
termite level

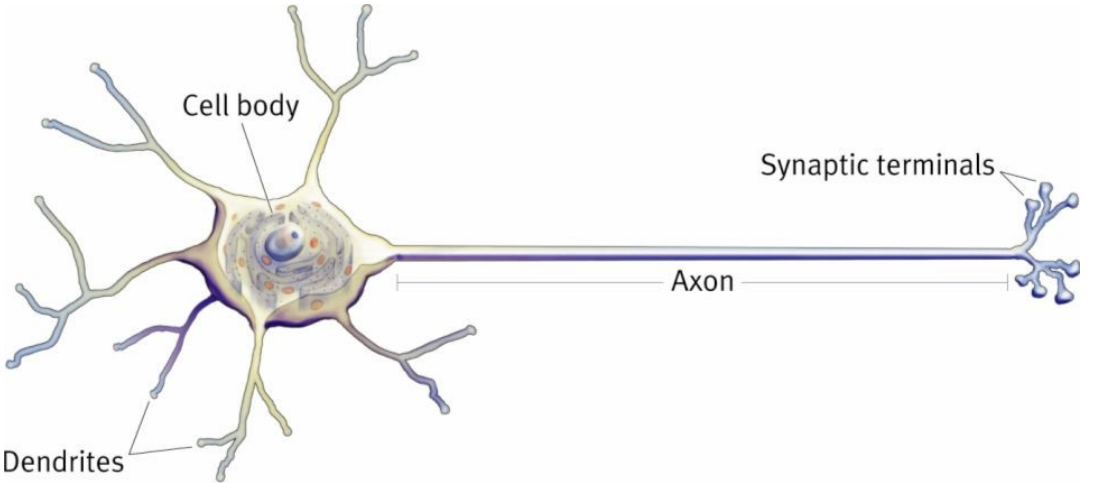
- an “organism” with clear and logical behaviors on two levels:
- termite level:
  - each termite has its needs
  - each termite has its abilities
  - each termite reacts to pheromones that influence its behaviors individually

termite nest
nest level

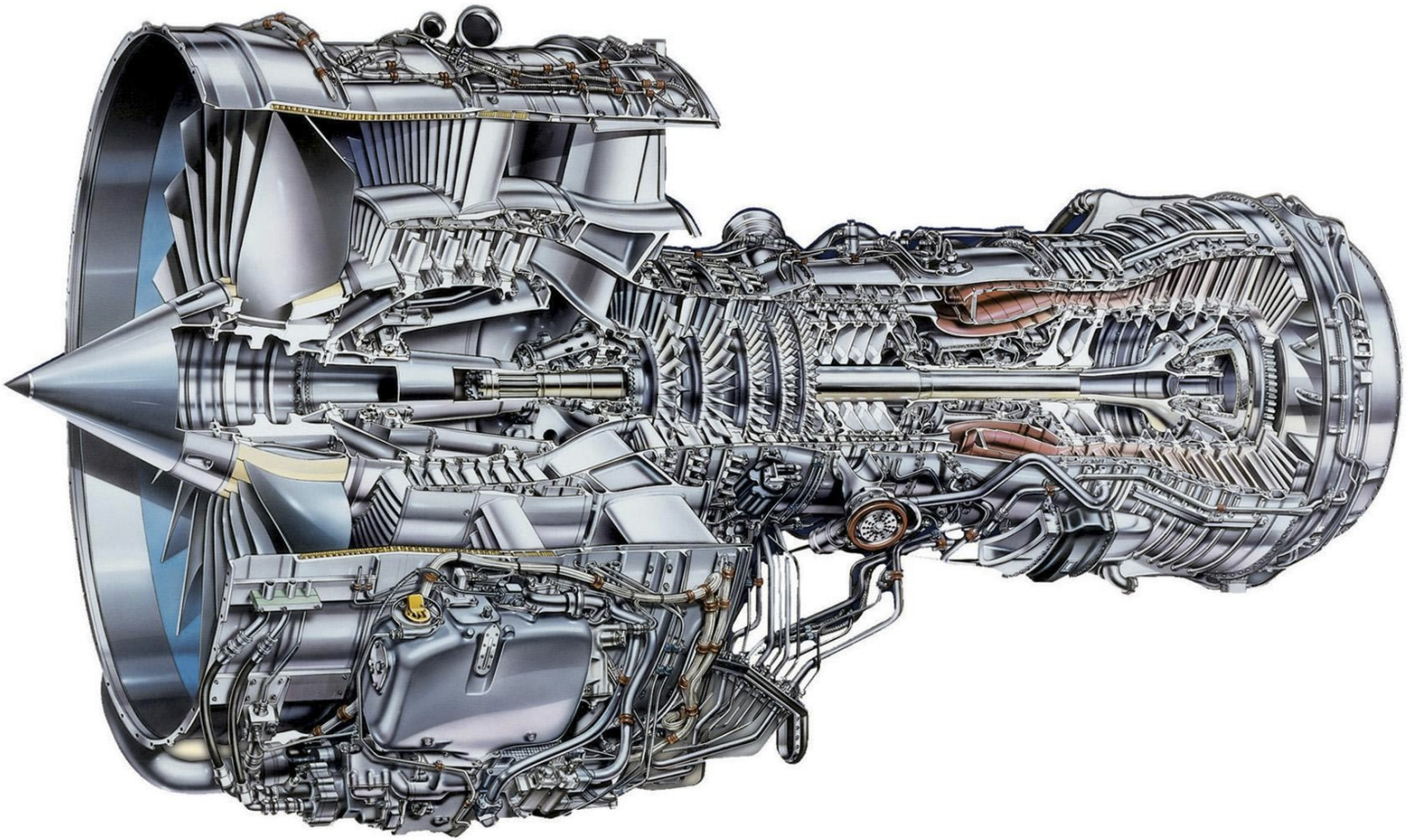
- nest level - a city with:
  - factory
  - defences
  - cleaning crews
- Complex systems can appear based on a great number of simple interactions

consciousness

- we know perfectly well how neurons work
- consciousness appears when 100 billion neurons interact
- fear, aspiration, hope, altruistic behaviour
  - impossible to predict by studying the neuron
  - they are a product of emerging complexity

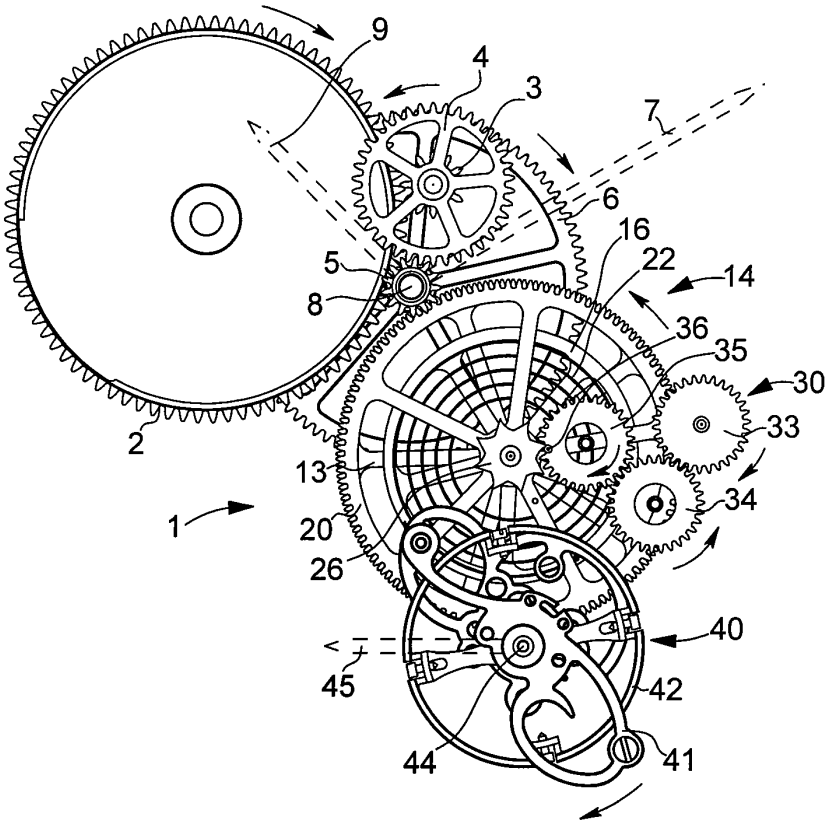


jet engine



complex vs. complicated

- complex systems:
  - give birth to a new structure
  - are unpredictable
- complicated systems:
  - are completely predictable



emergence in society

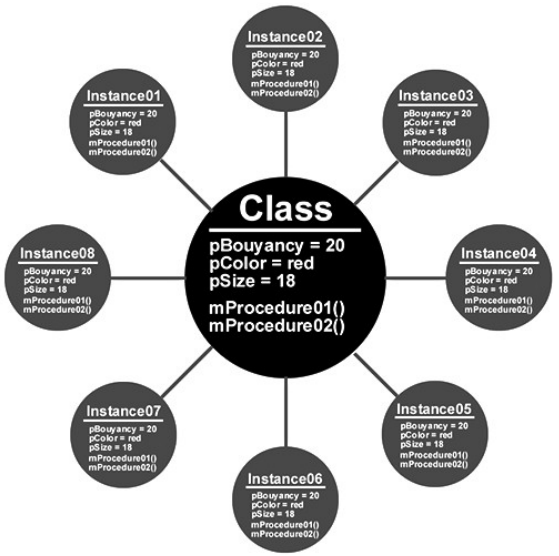
- “far left” idea during the 50's
- “visionary” during 1970-1990
- today many examples of it on the internet & it's becoming acceptable
  - NSA wants to predict revolutions through Twitter
  - Google predicts flu spreads 15 days faster than the centers of disease control
  - Big Data

How do we program agents?

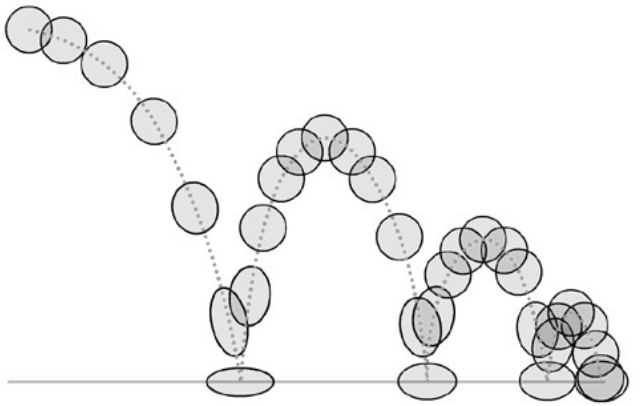
object oriented design

a conceptual leap forward

- a conceptual change in programming
- not a new computing technique, just a change in the way we think & organize our code
- objects around us have properties and behaviors



Class Ball



- properties
  - shape
  - color
  - weight
  - material
  - content
- behaviours
  - movement
  - bounce
  - inflating
  - deflating
  - blowing up

Class vs. object

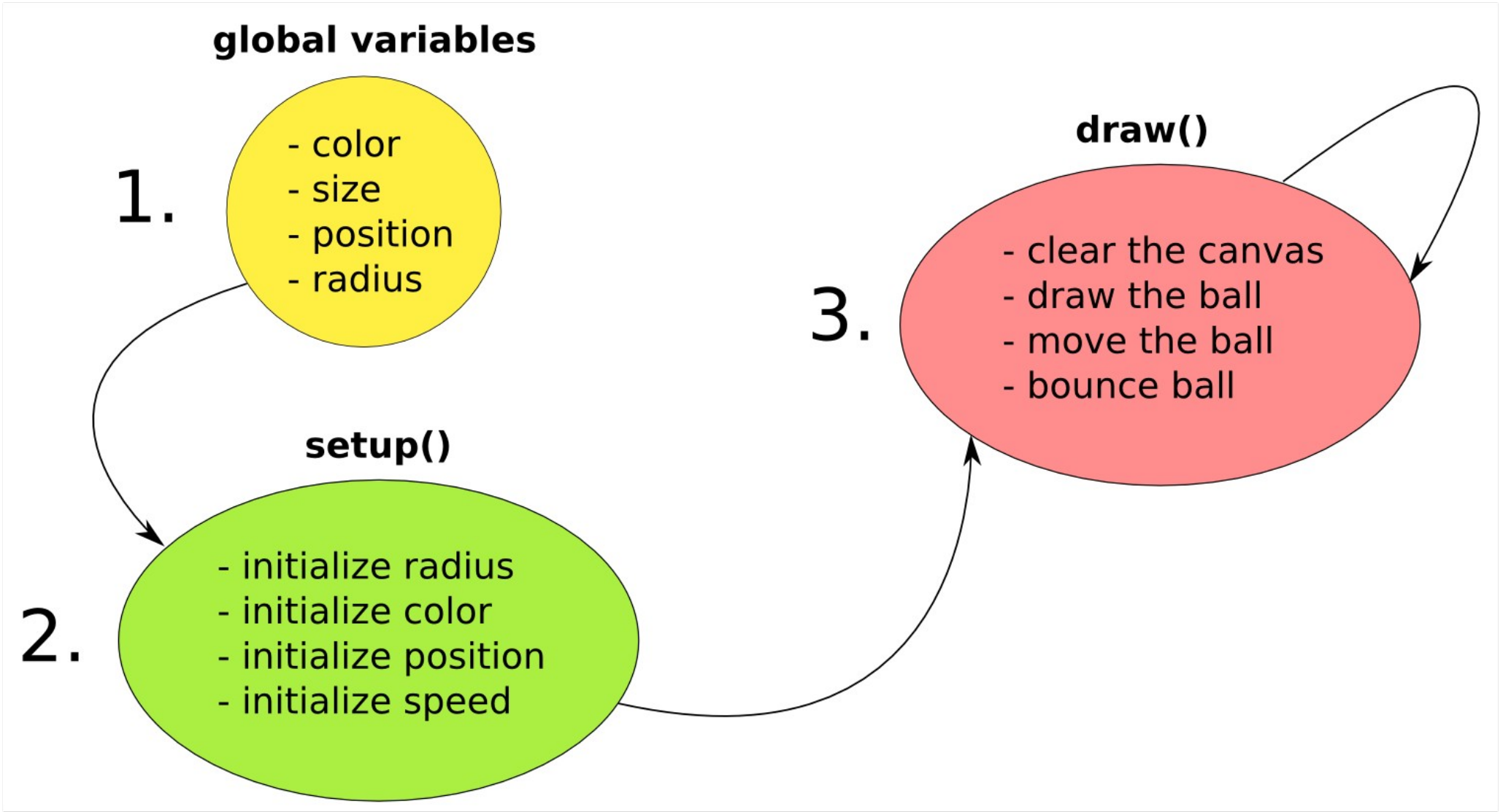
Class	Object
mobile phone	iphone 4G, Samsung Galaxy, HTC N1
computer	lenovo z61m, DELL 1120,
singer	Leonard Cohen, Lennon, Brel
mountain	Olympus, Everest
man	Garbo, Socrates, Freud
country	France, England, Germany
painter	Dali, Pollock, Van Gogh

objects in C++

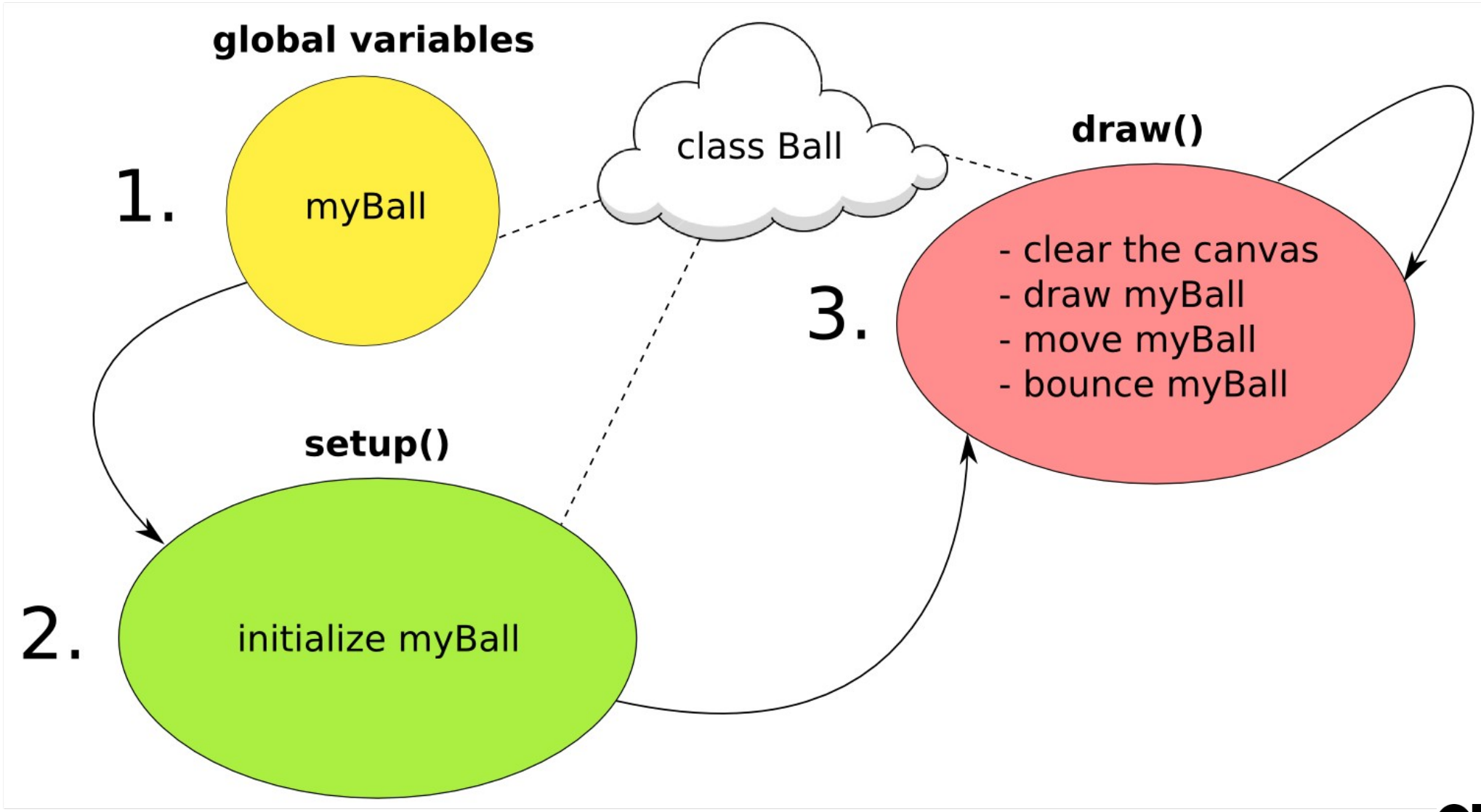
- class vs. object (Ball vs. myBeachBall)
  - myBeachBall is a example of an object with:
    - properties → variables in c++
    - behaviors → functions in c++

bouncingBall

in pseudocode

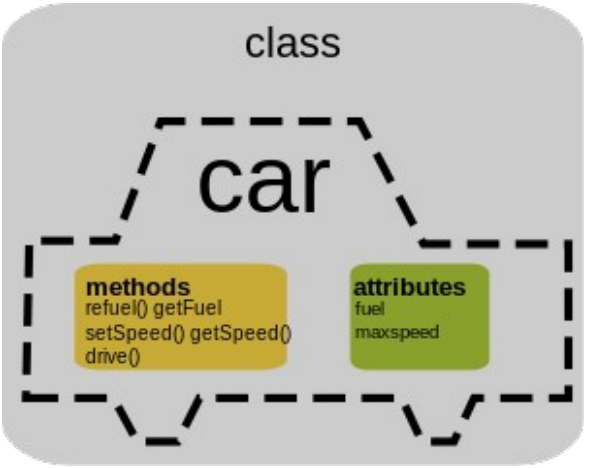


bouncingBall
in OOP



a class has

- a name
- variables
- constructor
- methods



```
float posX;
float posY;
float directionX;
float directionY;
float diam;
float colorR;
float colorG;
float colorB;

void setup()
{
  size(500, 500);

  posX = 10;
  posY = 10;
  directionX = 3;
  directionY = 4;
  diam = 20;
  colorR = 255;
  colorG = 0;
  colorB = 0;
}

void draw()
{
  background(255);
  drawBall();
  moveBall();
  bounceBall();
}

void drawBall()
{
  stroke(0);
  fill(colorR, colorG, colorB);
  ellipse(posX, posY, diam, diam);
}

void moveBall()
{
  posX = posX + directionX;
  posY = posY + directionY;
}

void bounceBall()
{
  if (( posX > width) || (posX < 0)) directionX = directionX * -1;
  if ((posY > height) || (posY < 0)) directionY = directionY * -1;
}
```

algorithmic
vs
OOP

```
class Ball
{
  float posX;
  float posY;
  float directionX;
  float directionY;
  float diam;
  float colorR;
  float colorG;
  float colorB;

  Ball()
  {
    posX = random(0, width);
    posY = random(0, height);
    directionX = random(-5, 5);
    directionY = random(-5, 5);
    diam = random(5, 30);
    colorR = random(255);
    colorG = random(255);
    colorB = random(255);
  }

  void drawBall()
  {
    stroke(0);
    fill(colorR, colorG, colorB);
    ellipse(posX, posY, diam, diam);
  }

  void moveBall()
  {
    posX = posX + directionX;
    posY = posY + directionY;
  }

  void bounceBall()
  {
    if (( posX > width) || (posX < 0)) directionX = directionX * -1;
    if ((posY > height) || (posY < 0)) directionY = directionY * -1;
  }
}
```

colorBall

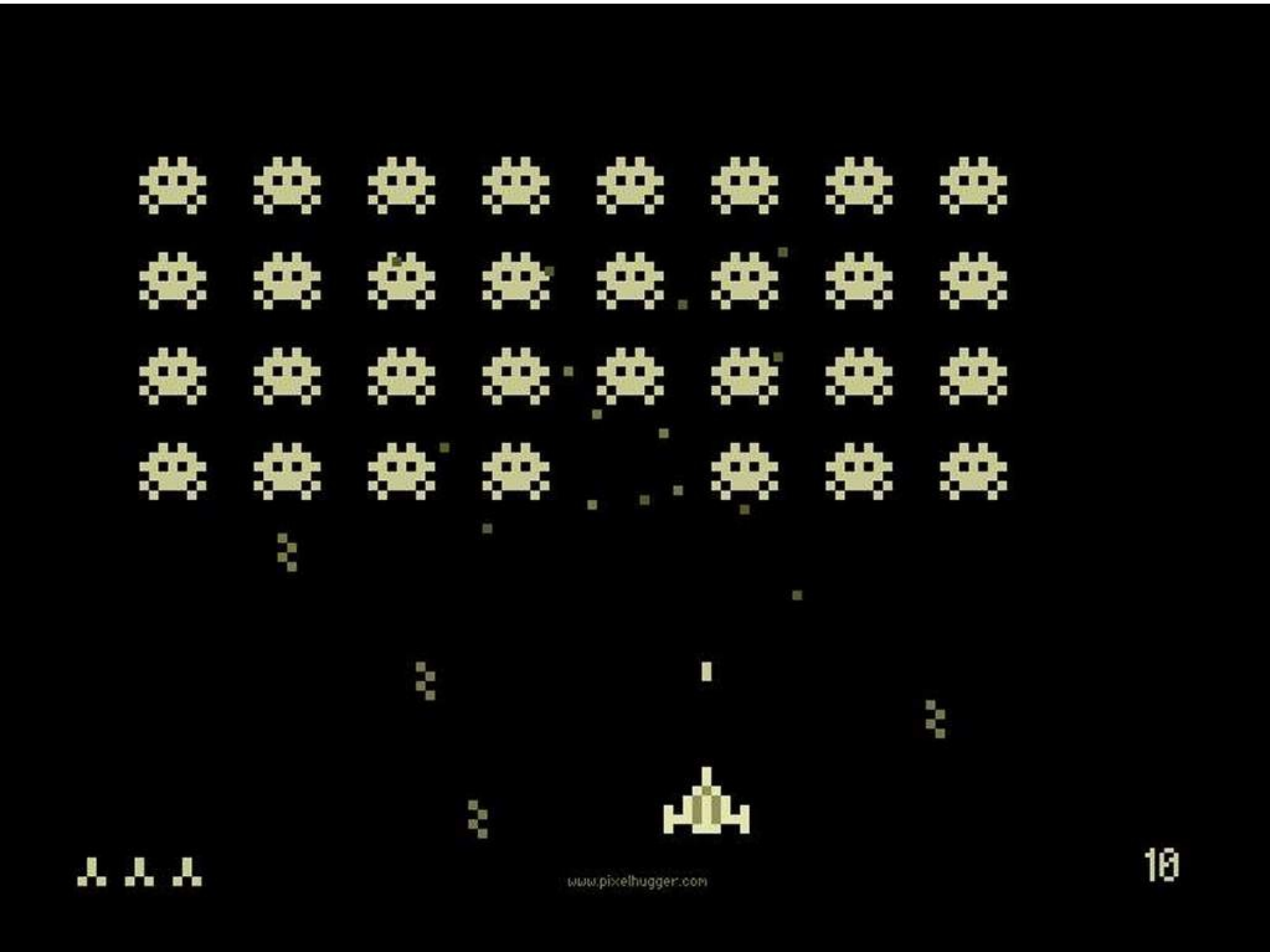
using vectors/arrays and OOP

- going crazy with the cookie-cutter

bouncingBall_OOP_array

using classes within classes

- Bike object contains:
  - wheel obj. (x2)
  - seat obj.
  - handlebar obj.
  - light obj. (x2)
  - frame obj.
  - chain obj.

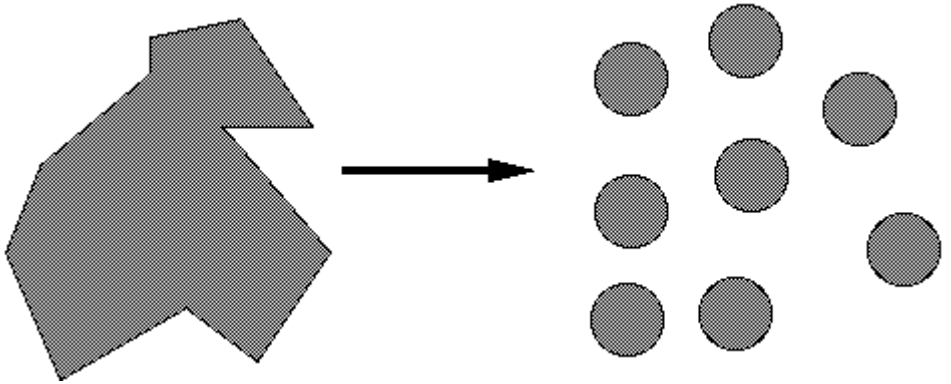

www.pixelhugger.com
10

in pseudocode

- space invaders update/draw functions
  - **run alien system**
    - move each alien
    - check each alien's collisions / remove dead ones
    - fire bullets
  - **run spaceship system**
    - move spaceship
    - check collisions
    - fire bullets
  - **update scores and lives**

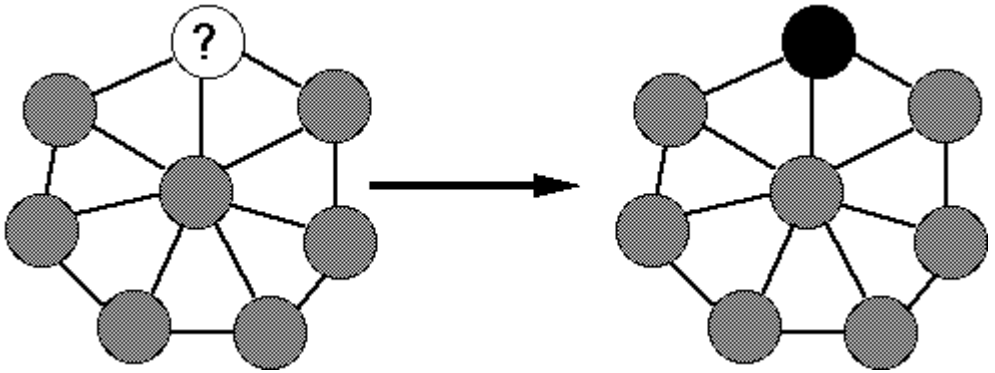
Why OOP?

- **Modularity:** To break the problem into smaller, manageable pieces



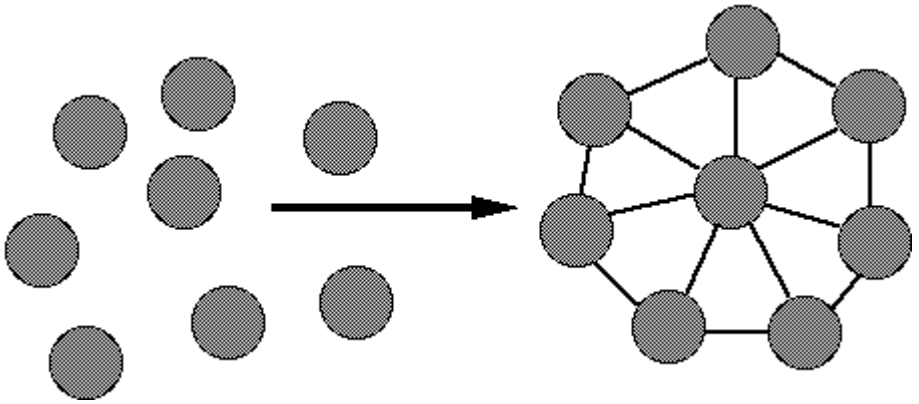
Why OOP?

- **Abstraction:** In order to make our code more understandable by taking about objects and behaviors



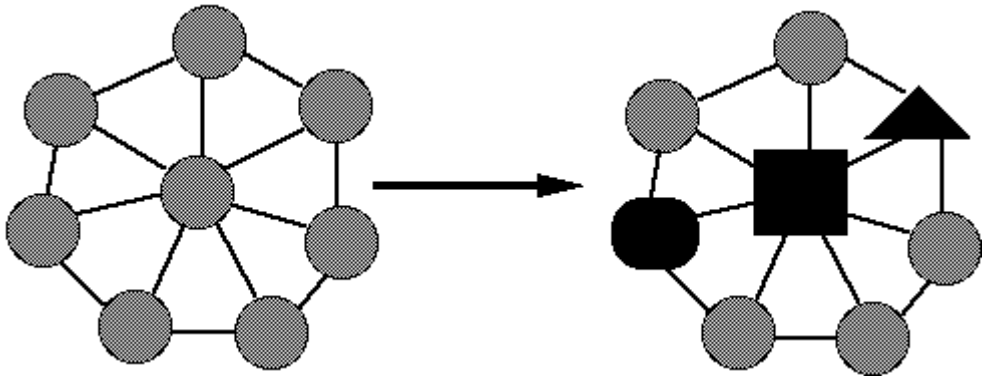
Why OOP?

- **Composability:** So that we can combine the parts into a new system.



Why OOP?

- **Continuity:** So that it's easier to maintain and extend our code.



when should you use OOP?

as often as you can!