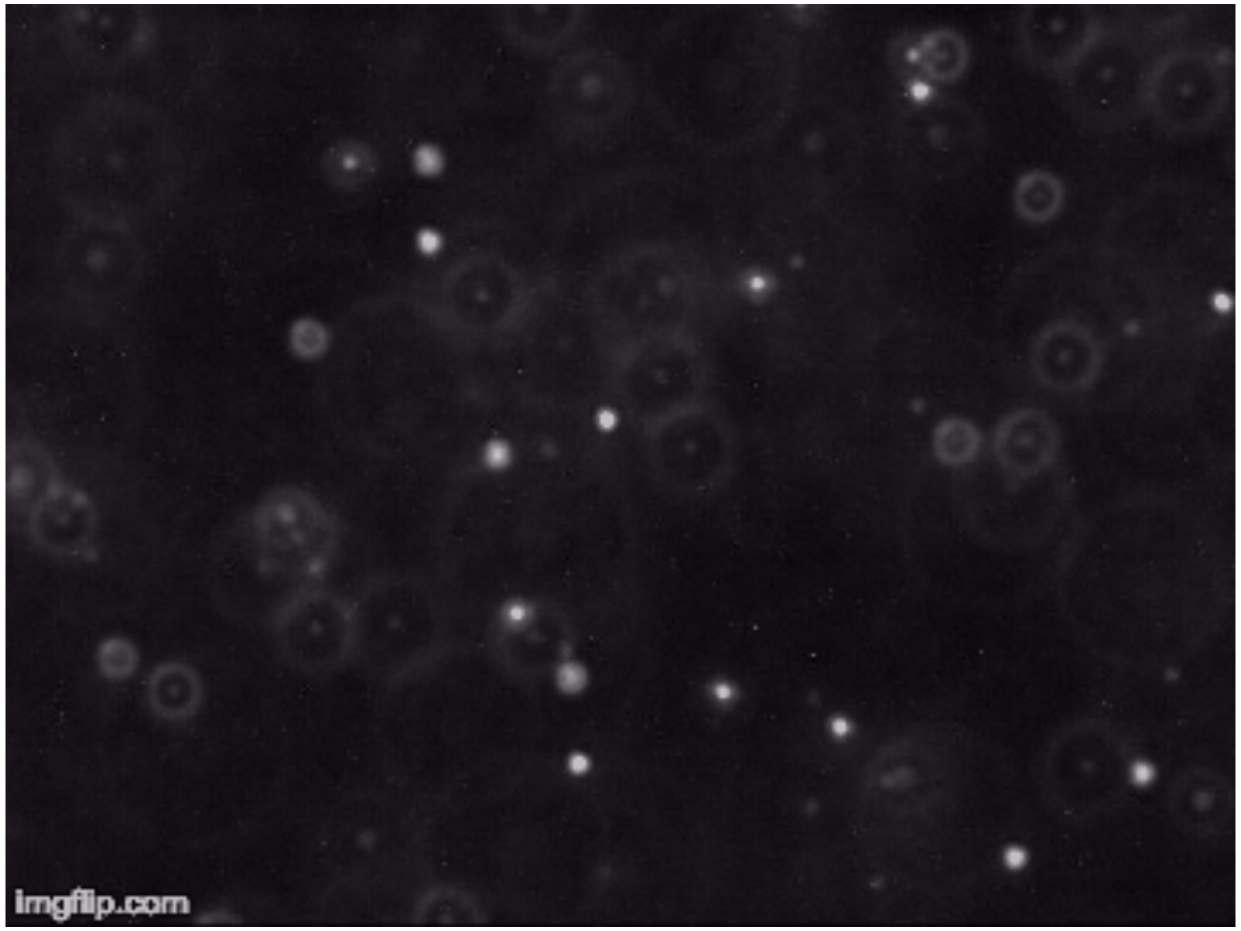
Final Project "Tips & Tricks"

COS126 - Spring 2017

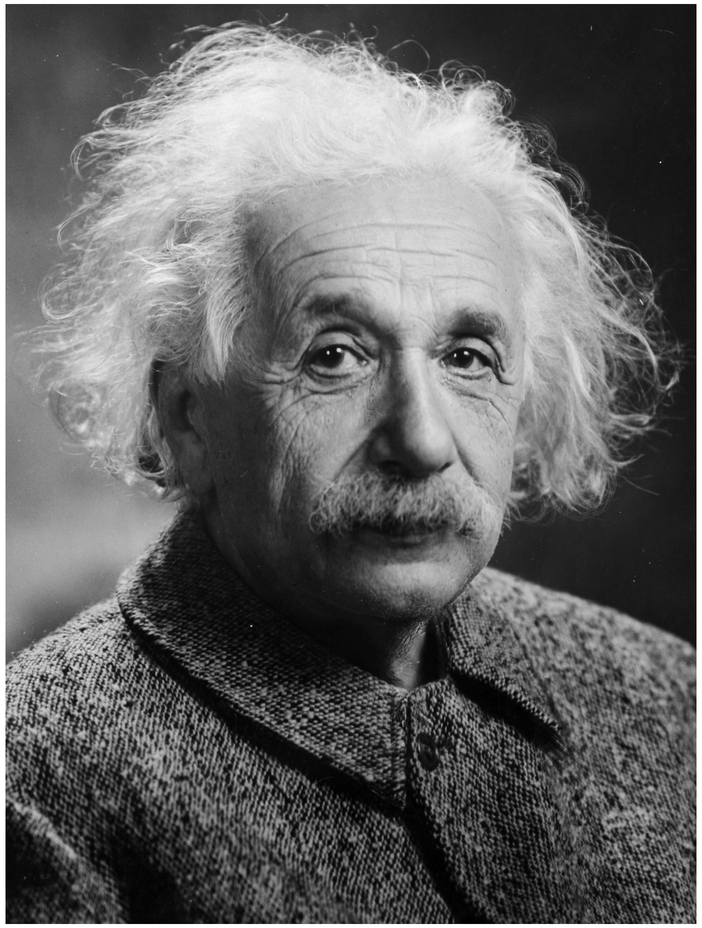
Project Context 1827

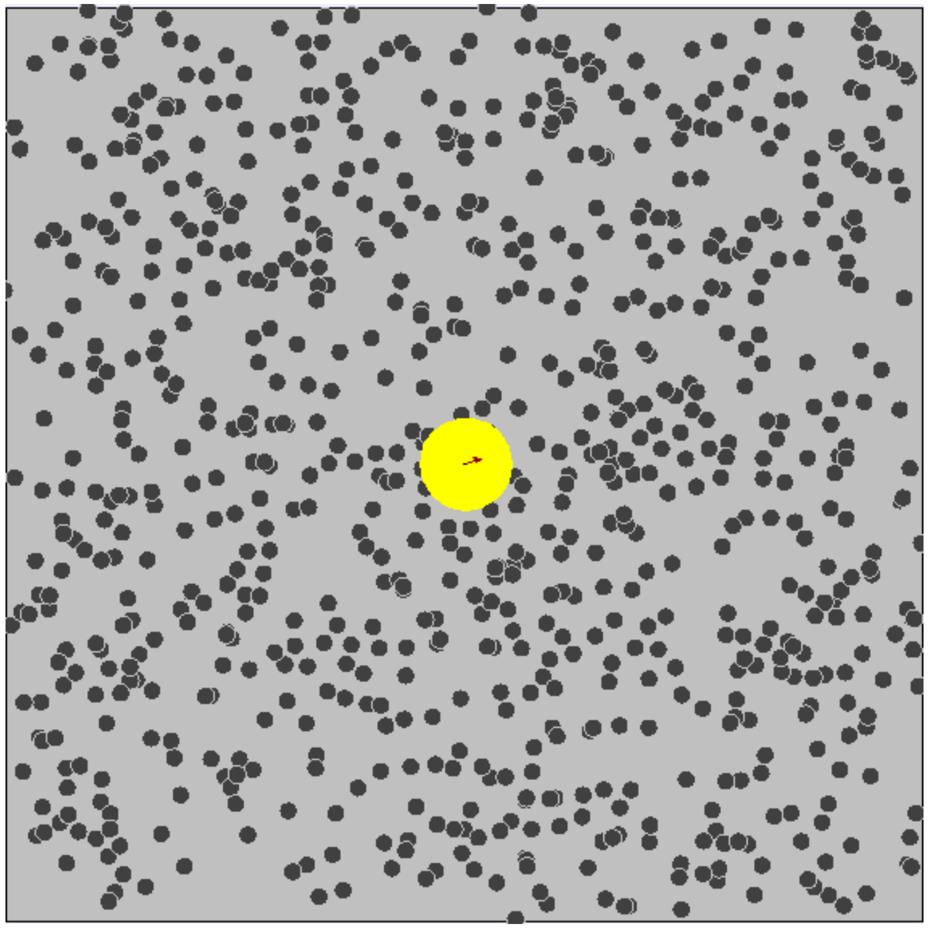


- No universal acceptance of the atomic nature of matter
- Botanist Robert **Brown** notices erratic motion of pollen grains in water. This motion is later called: **Brownian motion.**

Project Context

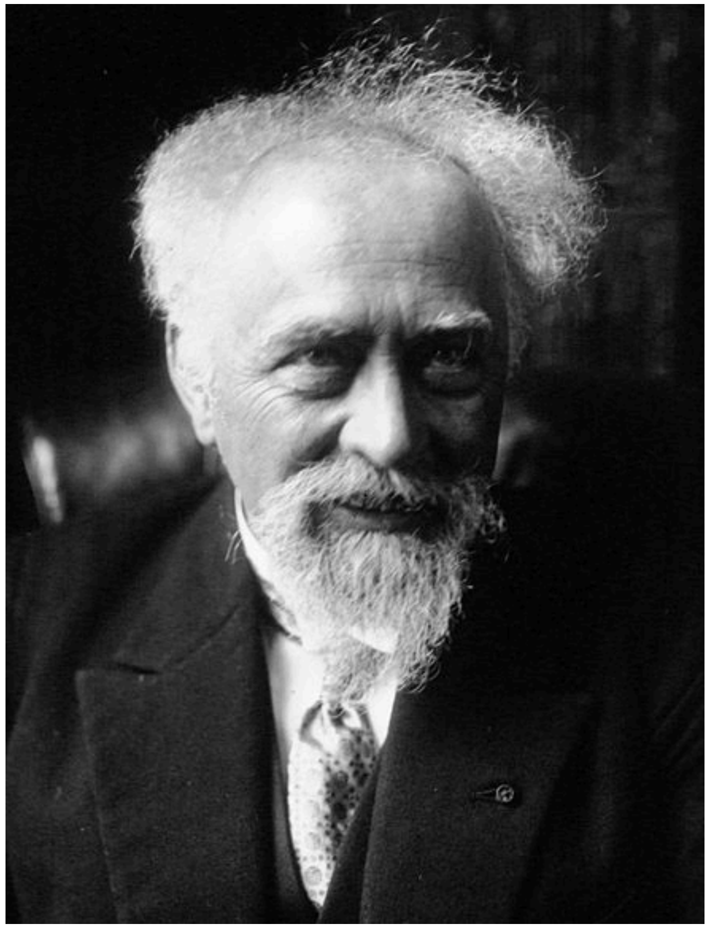
1905

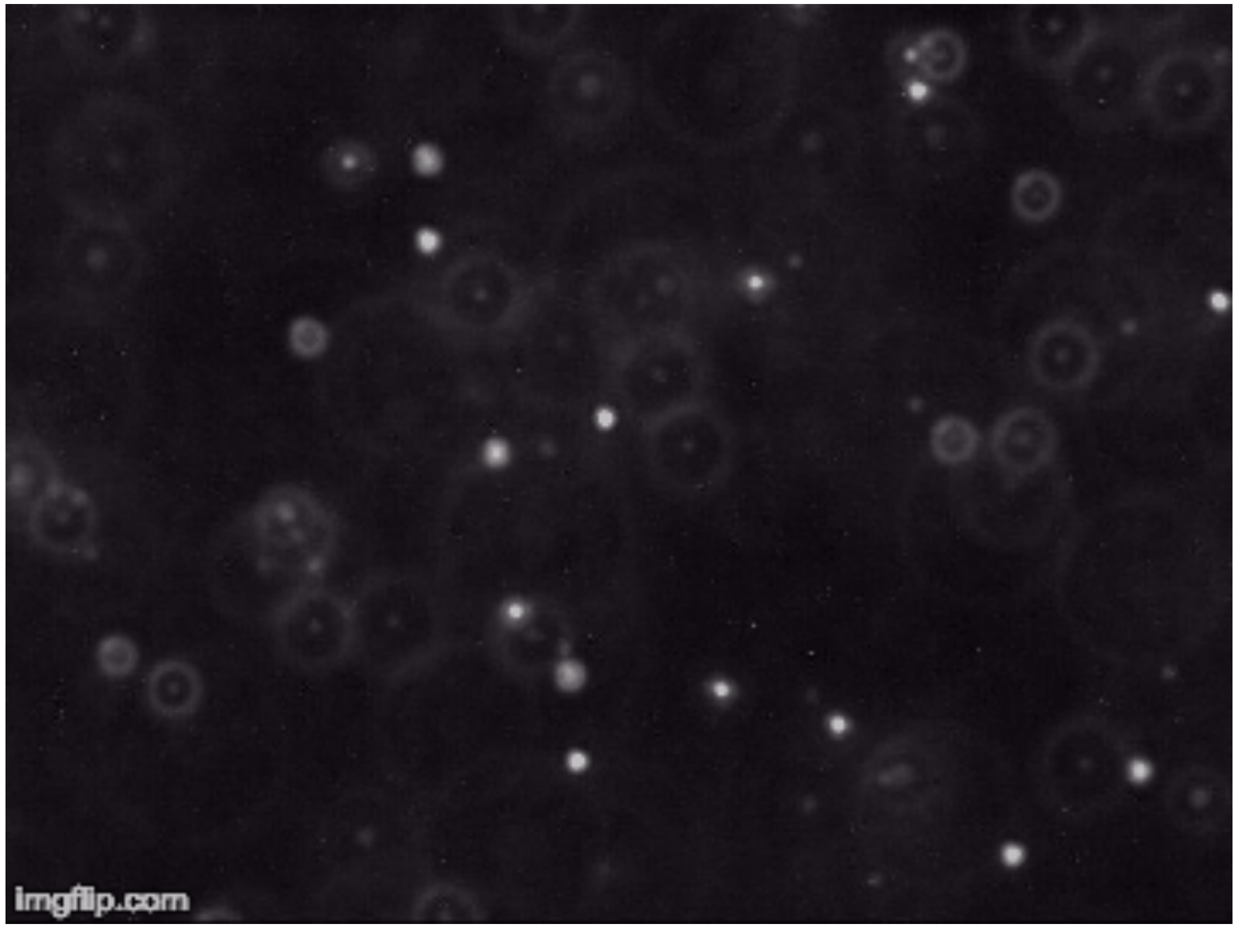




- Einstein publishes a revolutionary paper:
- Brownian motion is caused by **smaller** moving particles **colliding** with the **larger** pollen grains**.**
- Density of particles affects **displacement** in Brownian motion.

Project Context 1908



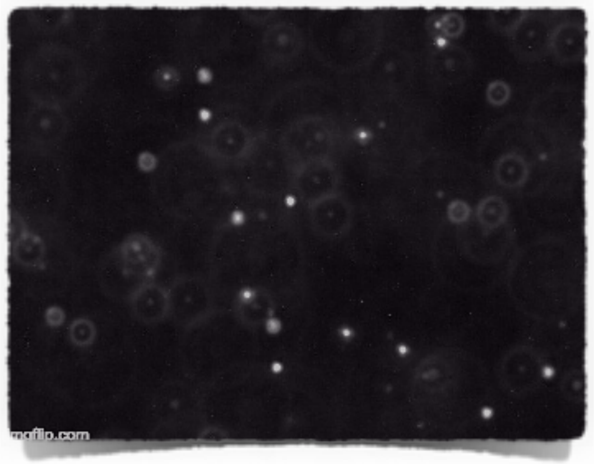


Jean Baptist Perrin experimentally validated Einstein's theory and equations.

Your Task: Redo Perrin's experiments!

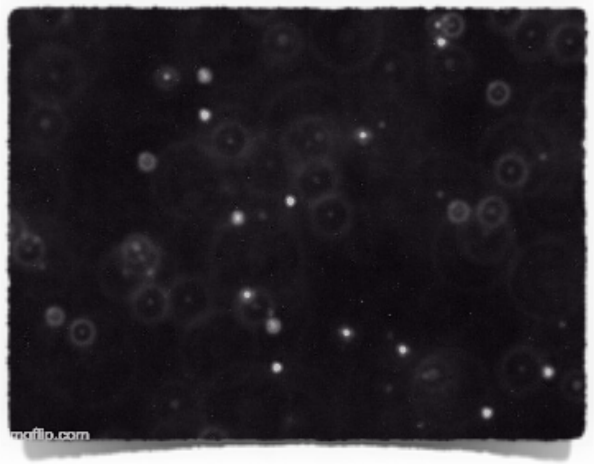
Not so difficult with computers and your **COS126** skills!

Experiment Overview

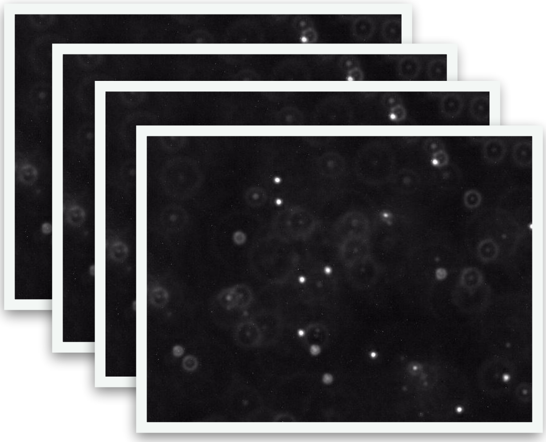


1 Record a microscopic **video** of particles undergoing Brownian motion

Experiment Overview



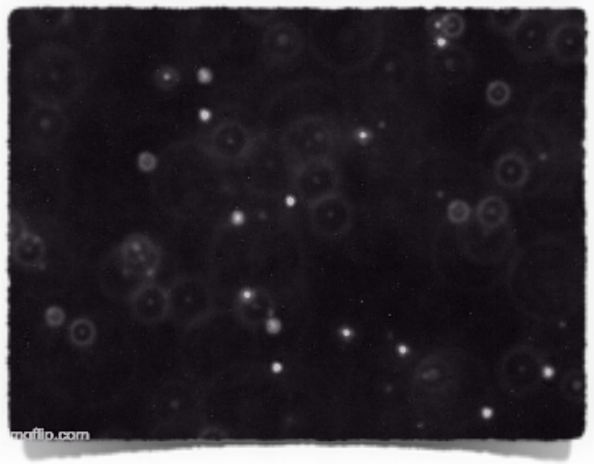
video



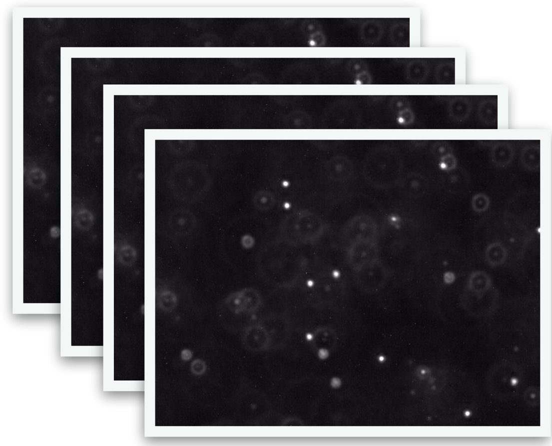
2

Convert the video into a set of **frames**

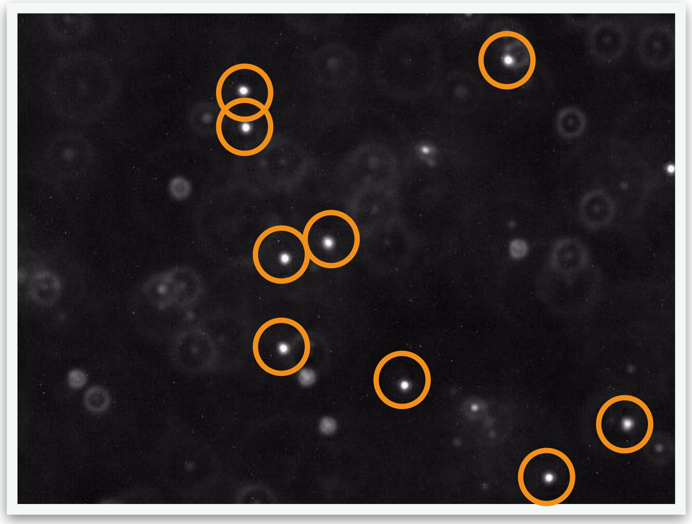
Experiment Overview



video



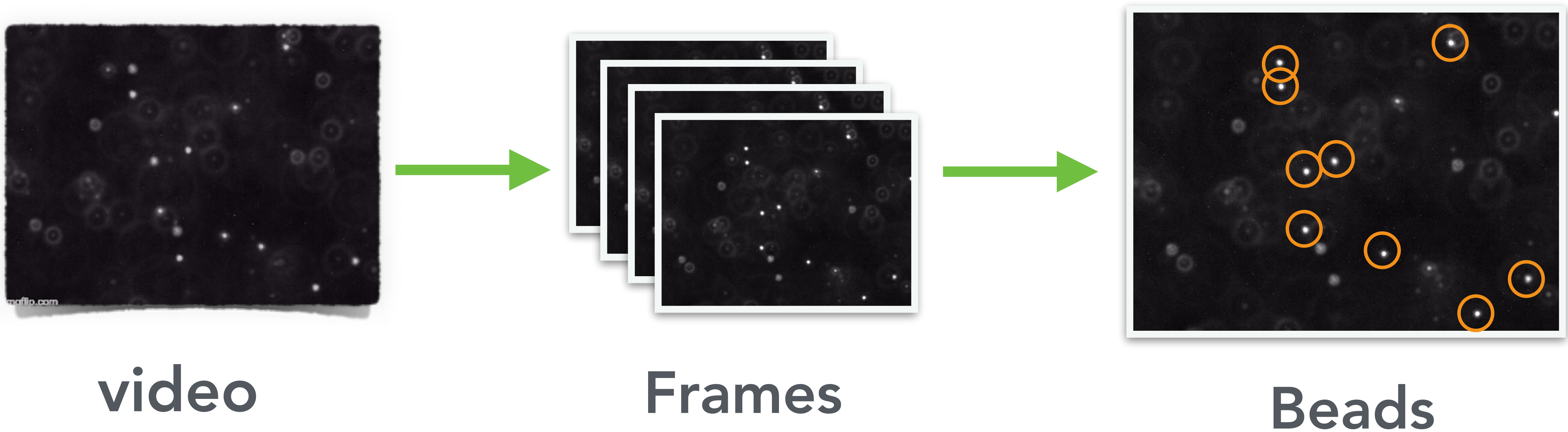
Frames



3

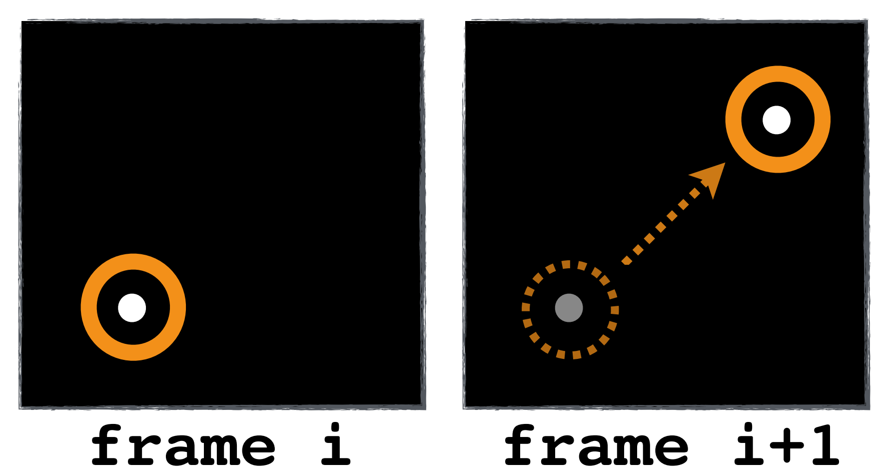
Detect **Beads** in every frame

Experiment Overview

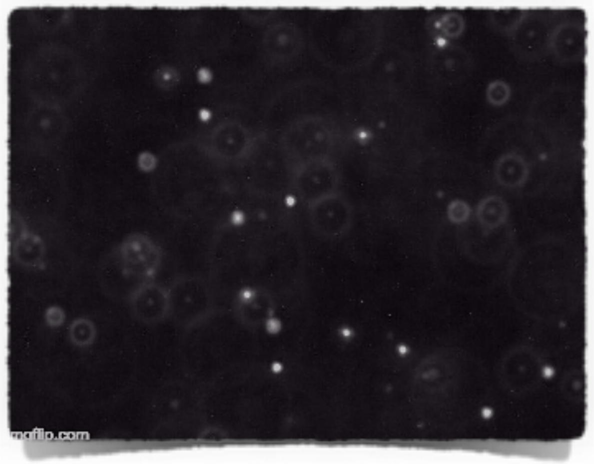


4

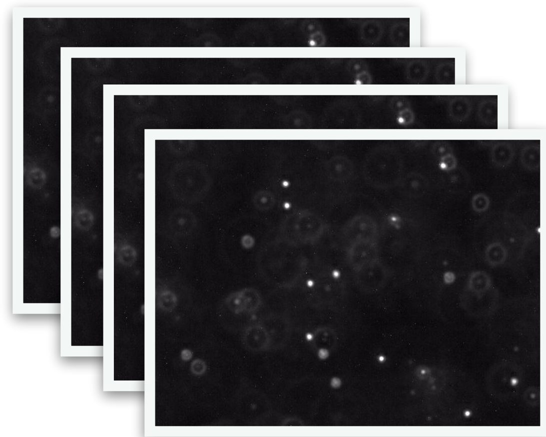
Compare **positions** of **beads** in every two **consecutive frames**



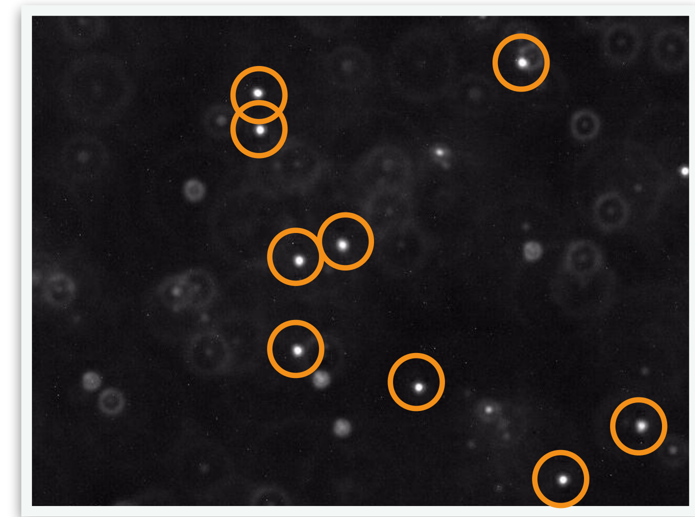
Experiment Overview



video



Frames

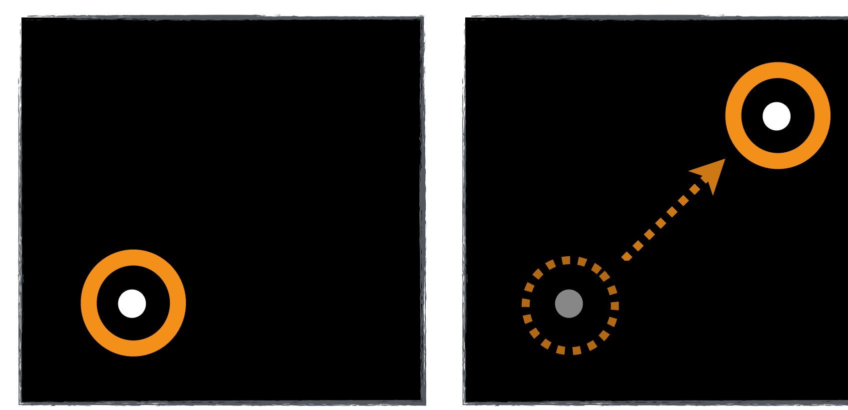


Beads

5

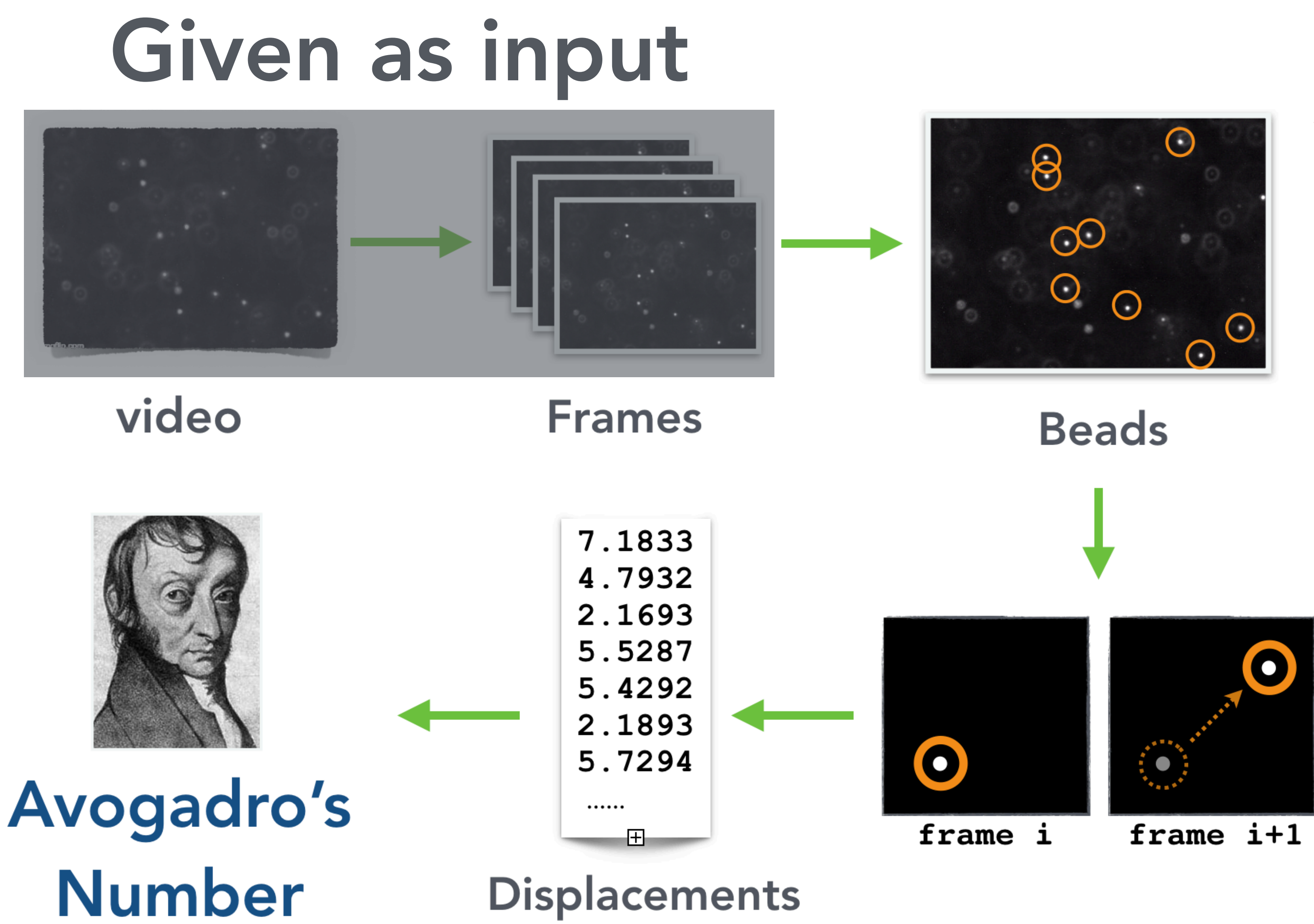
Record all bead **displacements** across consecutive fames.

7.1833
4.7932
2.1693
5.5287
5.4292
2.1893
5.7294
3.1141
4.5576
1.9898
......



Experiment Overview

Project Overview



BeadFinder.java
Detects all the "Beads" in a given frame.

BeadTracker.java
Outputs displacements of beads over successive frames.

Avogadro.java
Computes Avogadro's number from a given set of displacements.

+ **Blob.java** Represents a set of adjacent pixels.

+ **readme.txt** Shows performance analysis.

Project Requirements

Implement the following:

(1) `Blob.java`
Represents a set of adjacent pixels.

(2) `BeadFinder.java`
Detects all the “Beads” in a given picture.

(3) `BeadTracker.java`
Outputs displacements of beads over consecutive frames.

(4) `Avogadro.java`
Computes Avogadro’s number from a given set of displacements.

(5) Readme File
Shows performance analysis.

Project Requirements

Implement the following:

(1) `Blob.java`
Represents a set of adjacent pixels.

(2) `BeadFinder.java`
Detects all the "Beads" in a given picture.

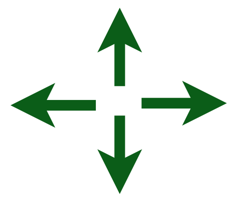
(3) `BeadTracker.java`
Outputs displacements of beads over consecutive frames.

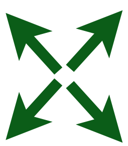
(4) `Avogadro.java`
Computes Avogadro's number from a given set of displacements.

(5) Readme File
Shows performance analysis.

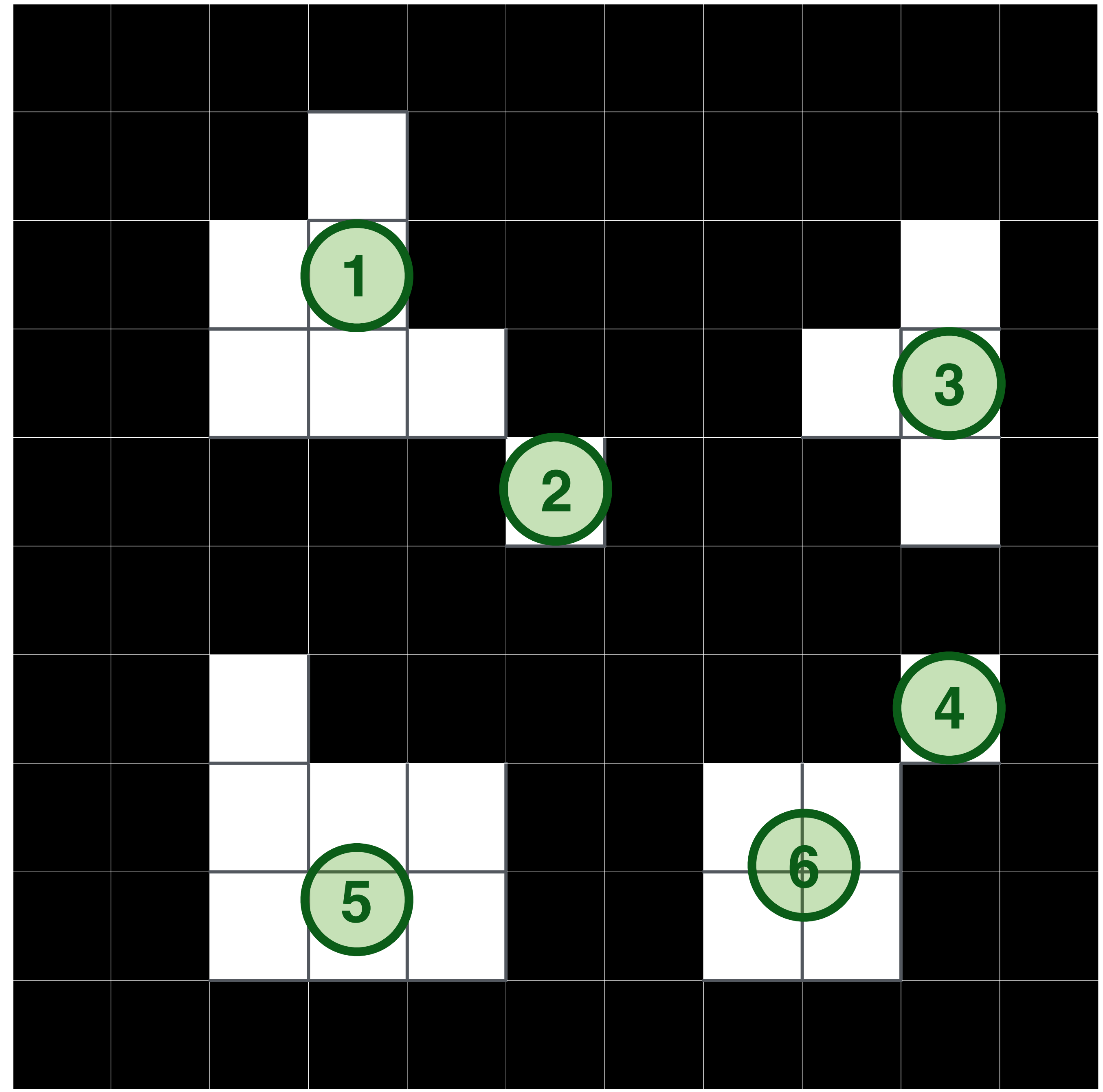
Blob.java

Blob: Any group of adjacent light pixels.

Adjacency is based on 

not 

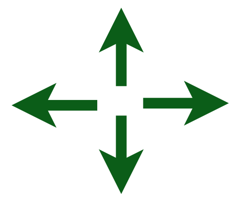
- **How many blobs are there?**

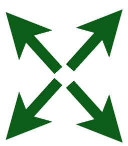


Assume each block to be a pixel

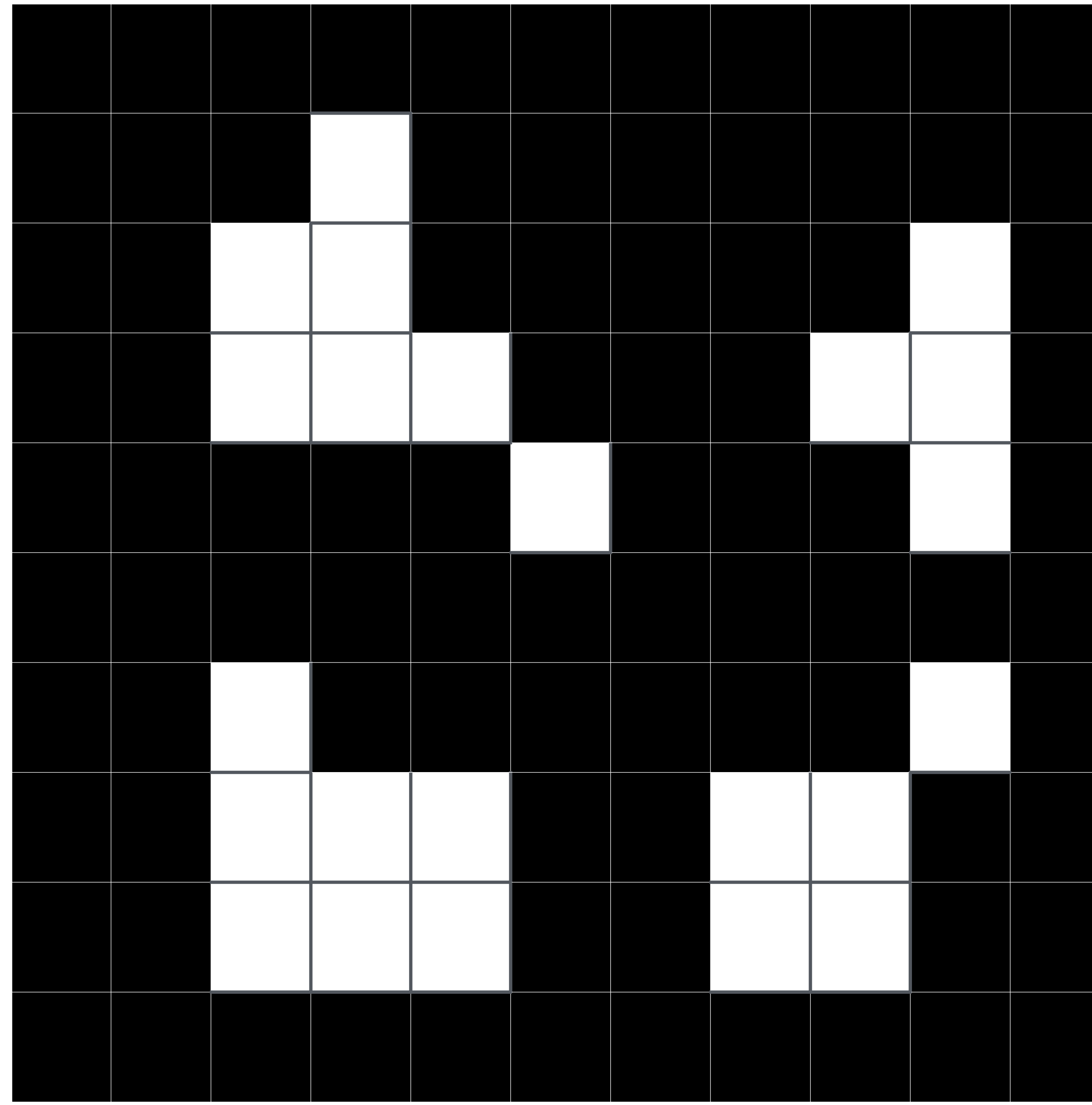
Blob.java

Blob: Any group of adjacent light pixels.

Adjacency is based on 

not 

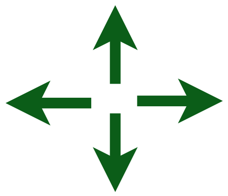
- **How many blobs are there?**

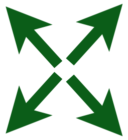


Assume each block to be a pixel

Blob.java

Blob: Any group of adjacent light pixels.

Adjacency is based on 

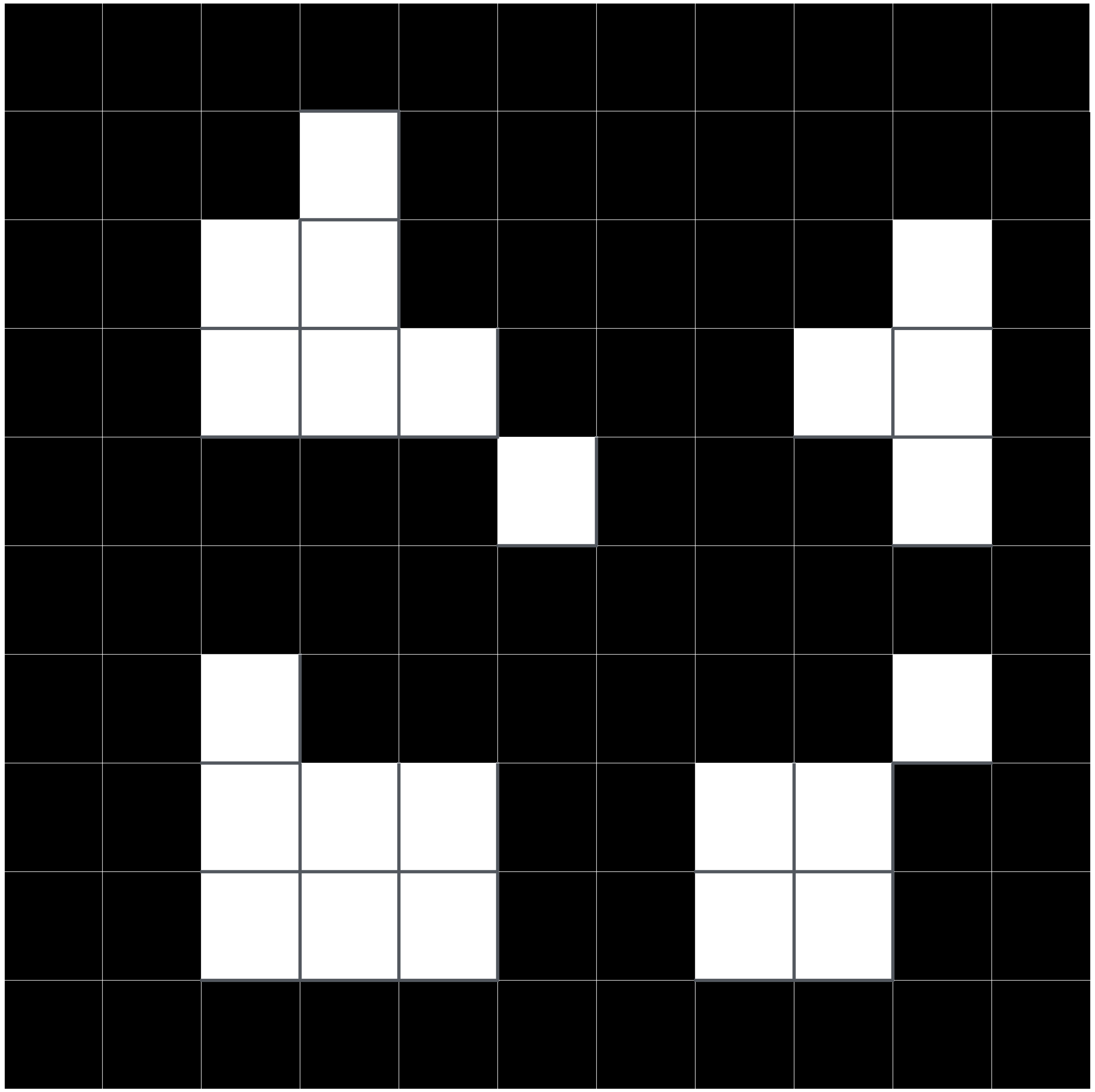
not 

- **How many blobs are there?**

Bead: A blob with a number of pixels that is at least ***min***.

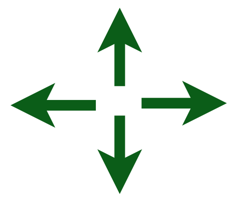
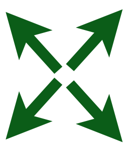
How many beads are there?

(assume *min=5*)



Assume each block to be a pixel

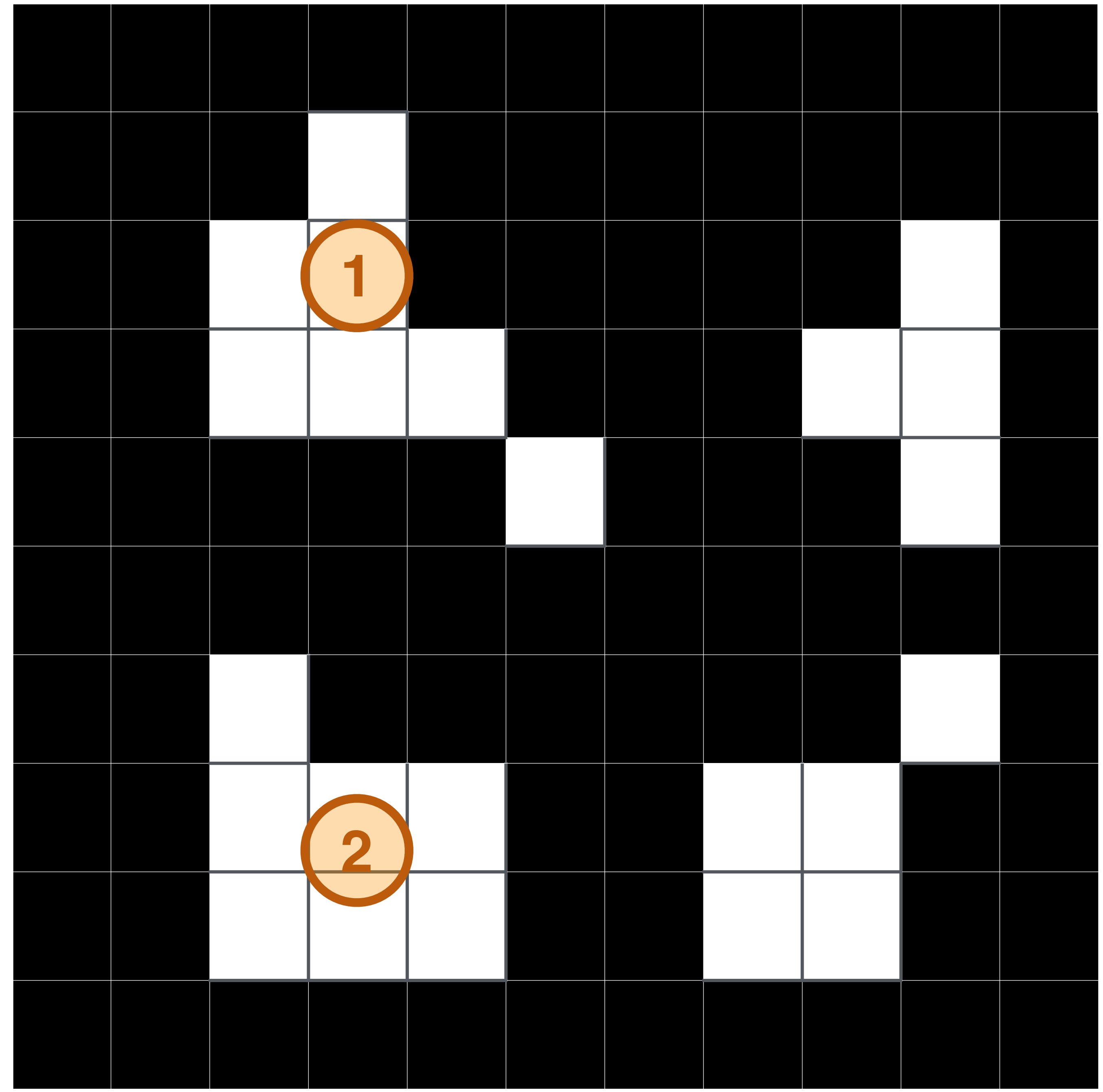
Blob.java

Blob: Any group of adjacent light pixels.
Adjacency is based on 
not 

- **How many blobs are there?**

Bead: A blob with a number of pixels that is at least ***min***.

How many beads are there?
(assume *min=5*)



Assume each block to be a pixel

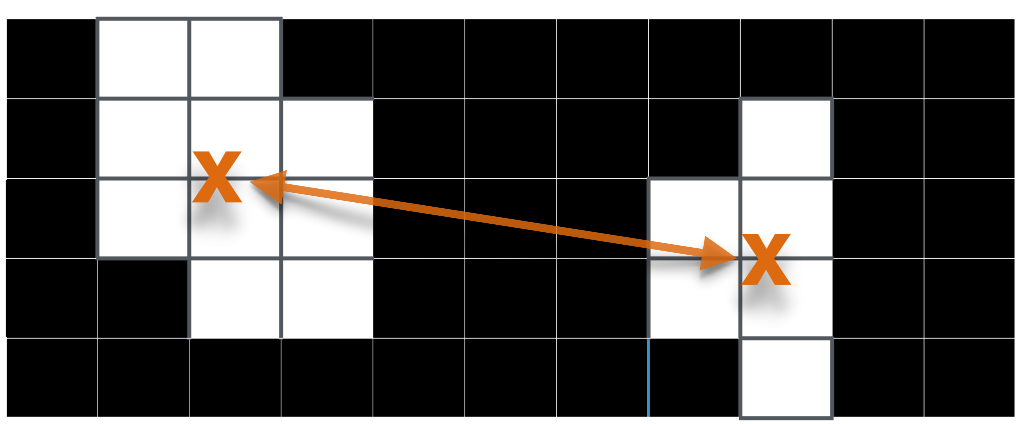
Blob.java

```
public class Blob {
    …
    public   void add(int x, int y)

    public     int mass()

    public double distanceTo(Blob that)
    …
}
```

adds a point to the blob

returns # of points in the blob

measures distance between this and that blob



Distance is measured between the **centers of mass** (`avgX, avgY`)

Blob.java

Test in main every public method.

```
public class Blob {
    public            Blob()
    public        int mass()
    public       void add(int x, int y)
    public     double distanceTo(Blob that)
    public     String toString()
    public static void main(String[] args)
}
```

- **Do not store every** added **point**. We are only interested in the **center of mass** of the points.
- **Checklist** has tips for implementing **`toString()`** and for handling corner cases.

Project Requirements

Implement the following:

(1) `Blob.java`
Represents a set of adjacent pixels.

(2) `BeadFinder.java`
Detects all the "Beads" in a given picture.

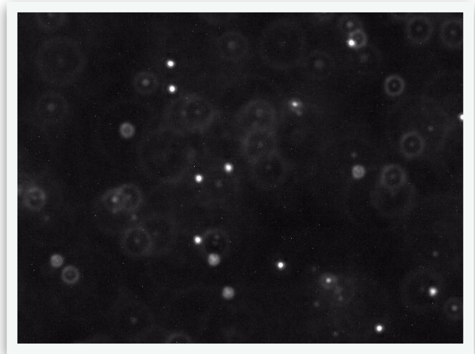
(3) `BeadTracker.java`
Outputs displacements of beads over consecutive frames.

(4) `Avogadro.java`
Computes Avogadro's number from a given set of displacements.

(5) **Readme File**
Shows performance analysis.

Input:

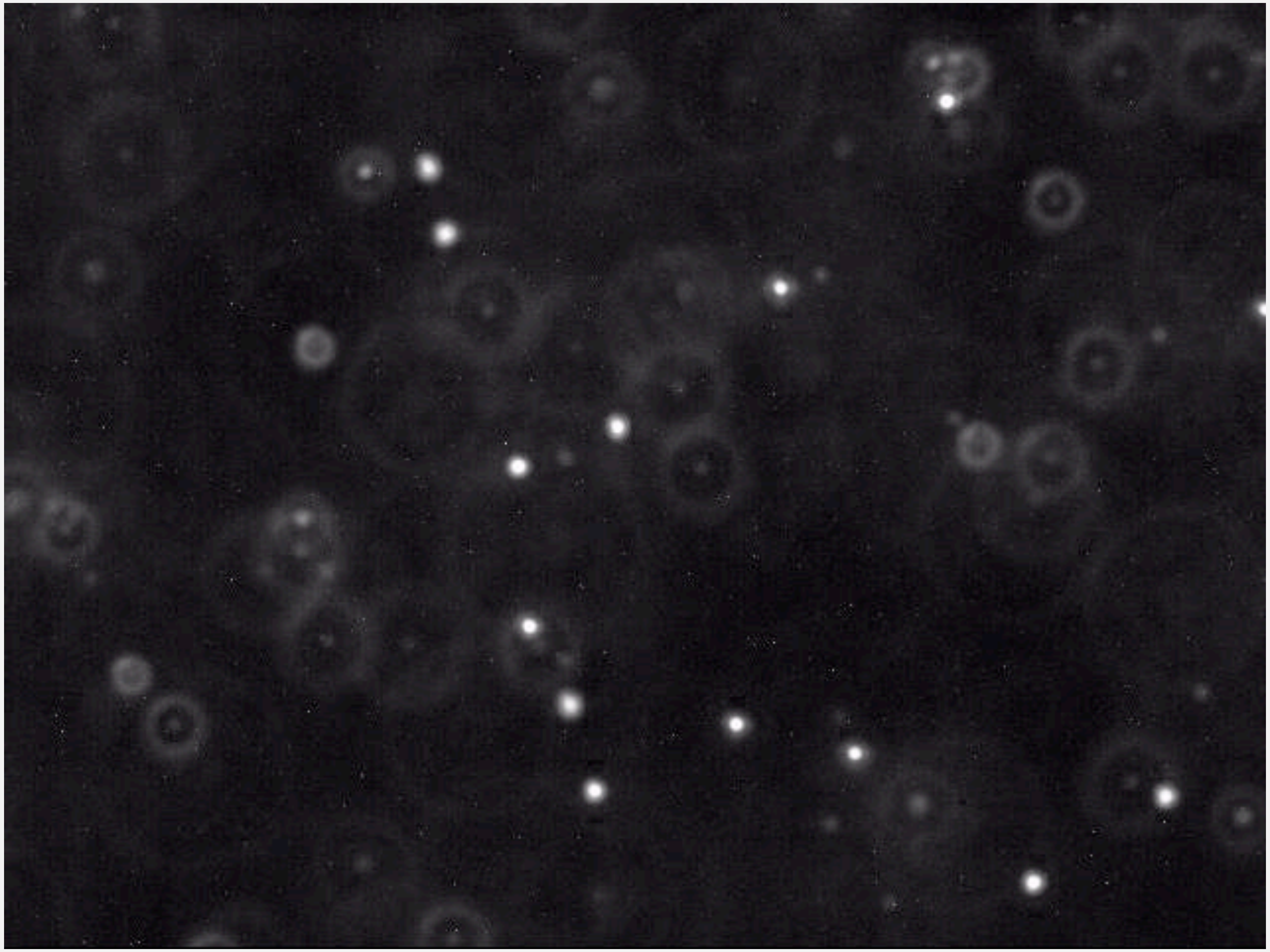
An Image



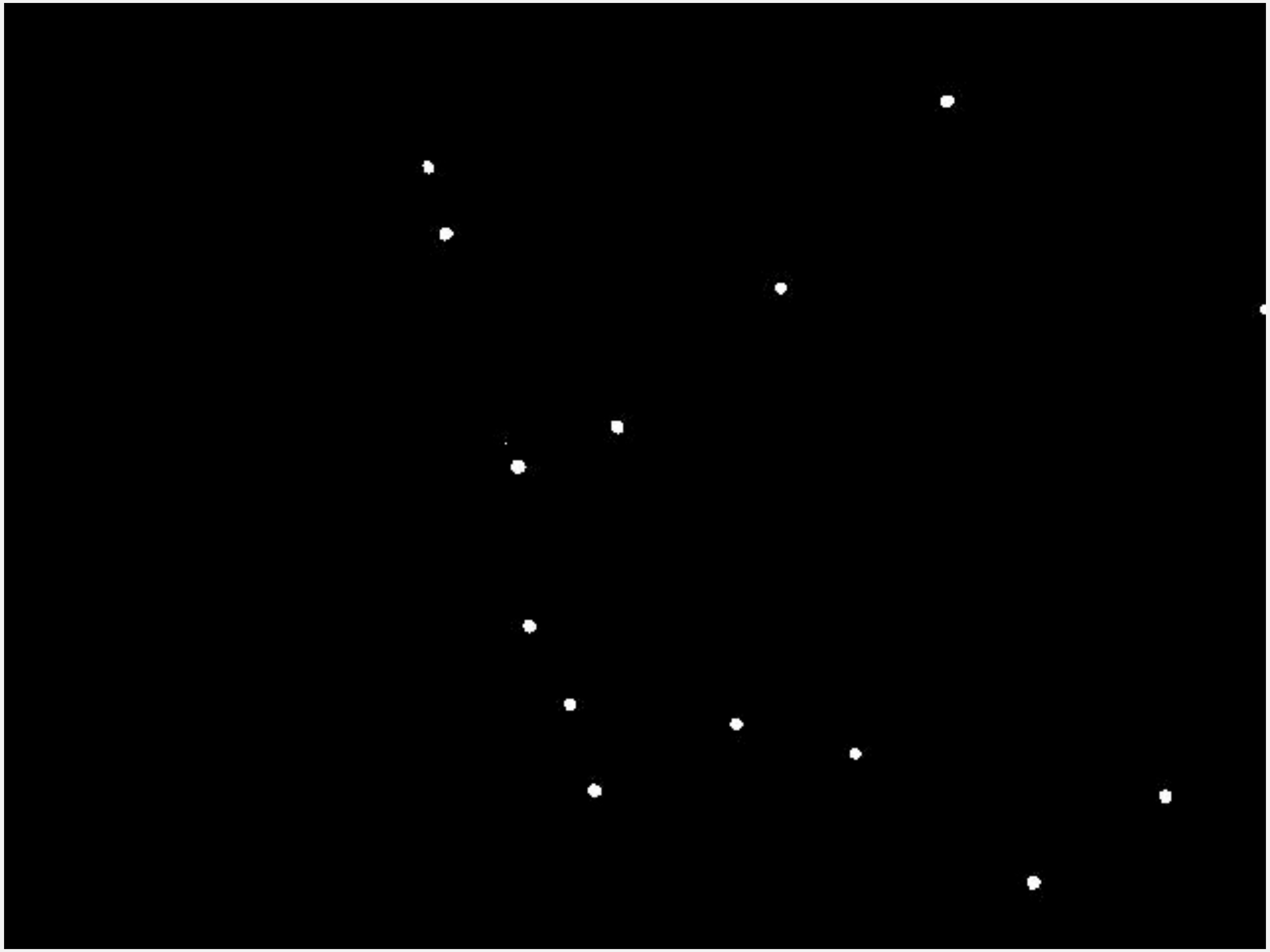
A Luminance Threshold ***tau***

BeadFinder.java

Original Image



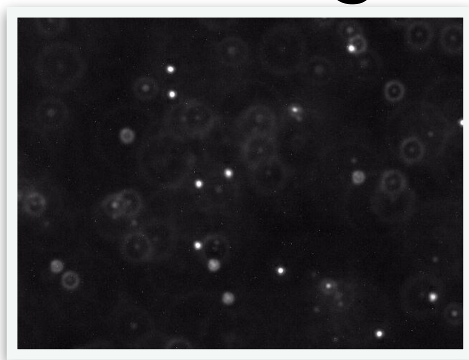
After Applying a Luminance Threshold ***tau***



threshold (monochrome luminance > 180)

Input:

An Image

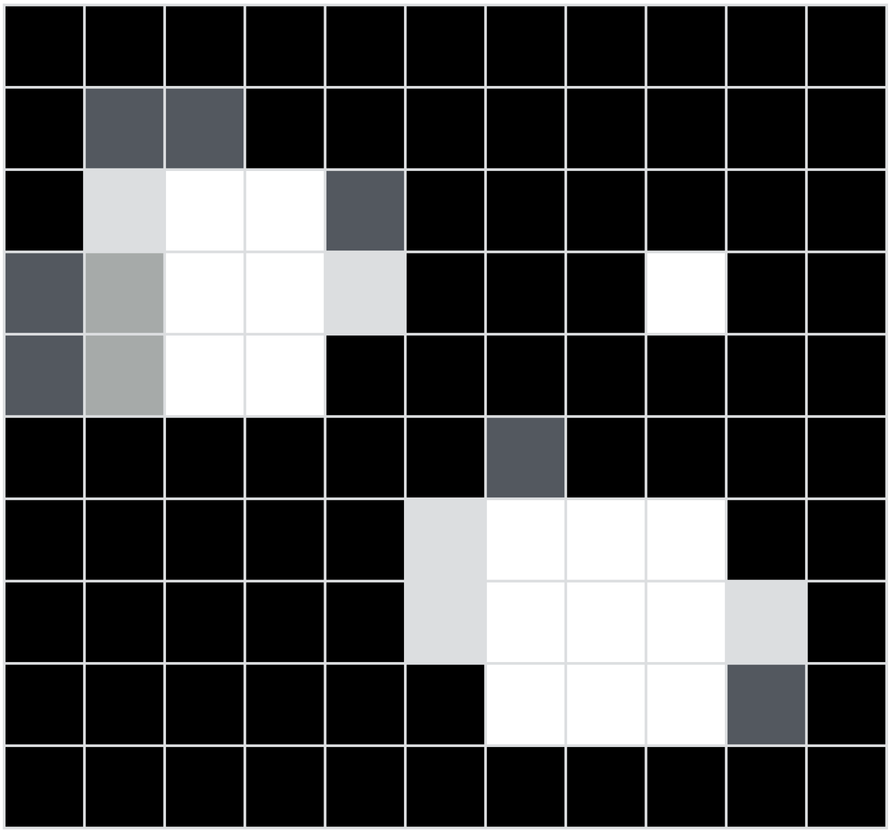


A Luminance Threshold ***tau***

BeadFinder.java

- Constructor stores ***all*** blobs in the image
- Returns through **`Blob [] getBeads(int min)`** all blobs that have at least ***min*** points

Detecting All Blobs



Pixels

F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F

visited[][]

For each pixel *p*:

If it is white enough

Create a blob from all its adjacent light pixels

For each pixel ***p***

If ***p*** is *white enough*

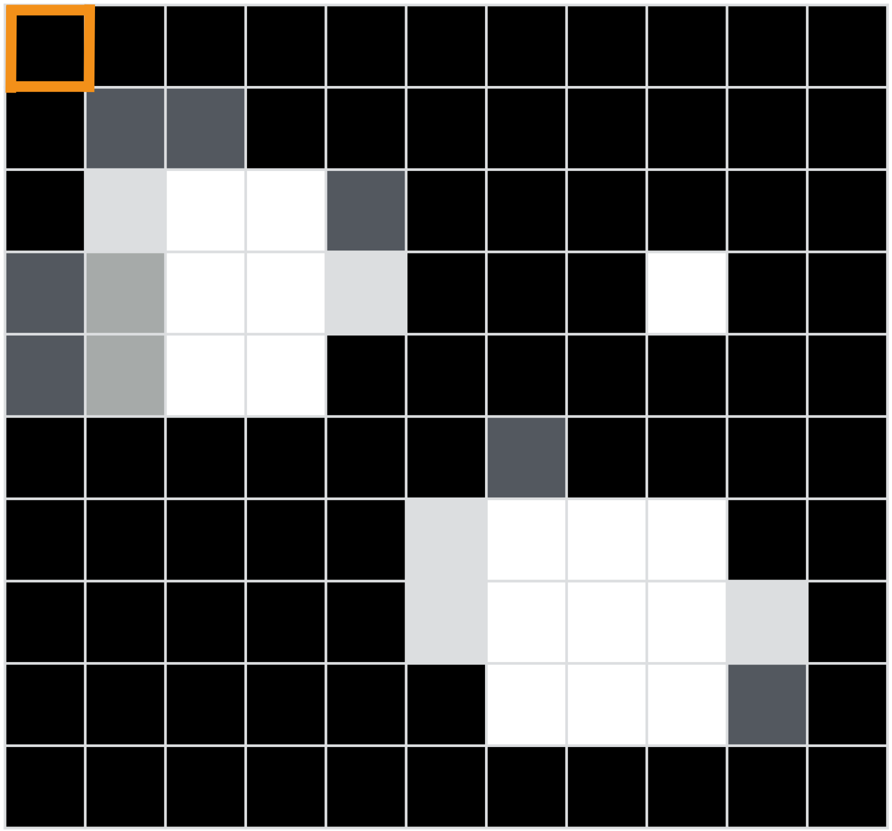
AND ***p*** *is not visited*

Create a **new Blob**

Start **DFS** from ***p***

Mark *p as* visited

Detecting All Blobs

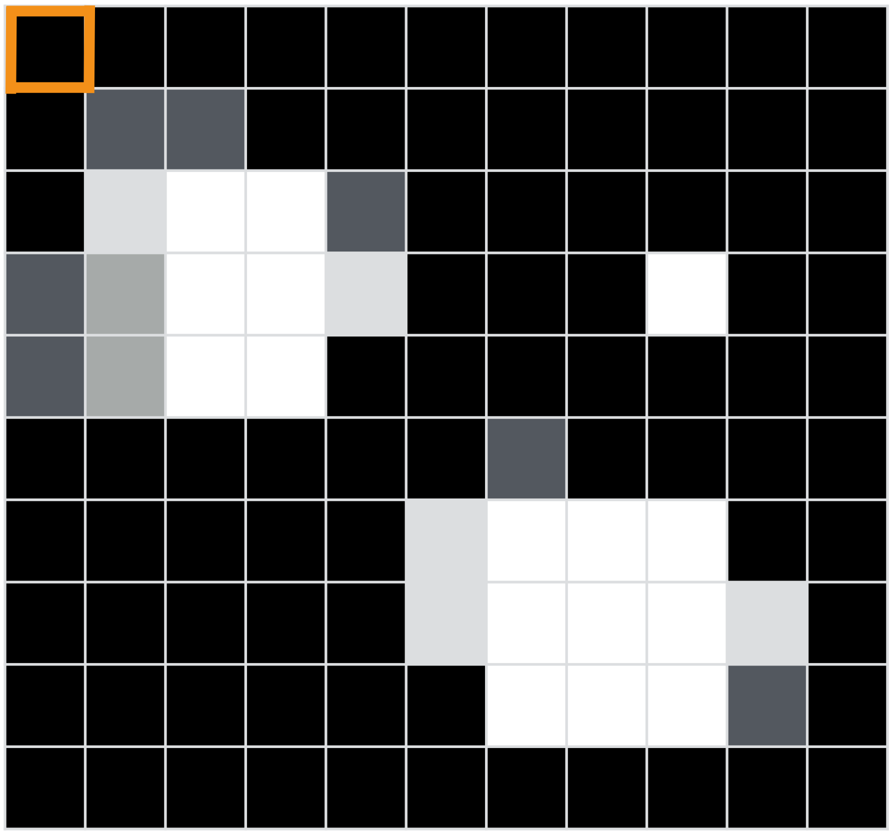


Detected Blobs

For each pixel ***p***

- **If** ***p*** is *white enough*
  AND ***p*** *is not visited*
  - Create a **new Blob**
  - Start **DFS** from ***p***
- **Mark** *p as* visited

Detecting All Blobs



T	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F

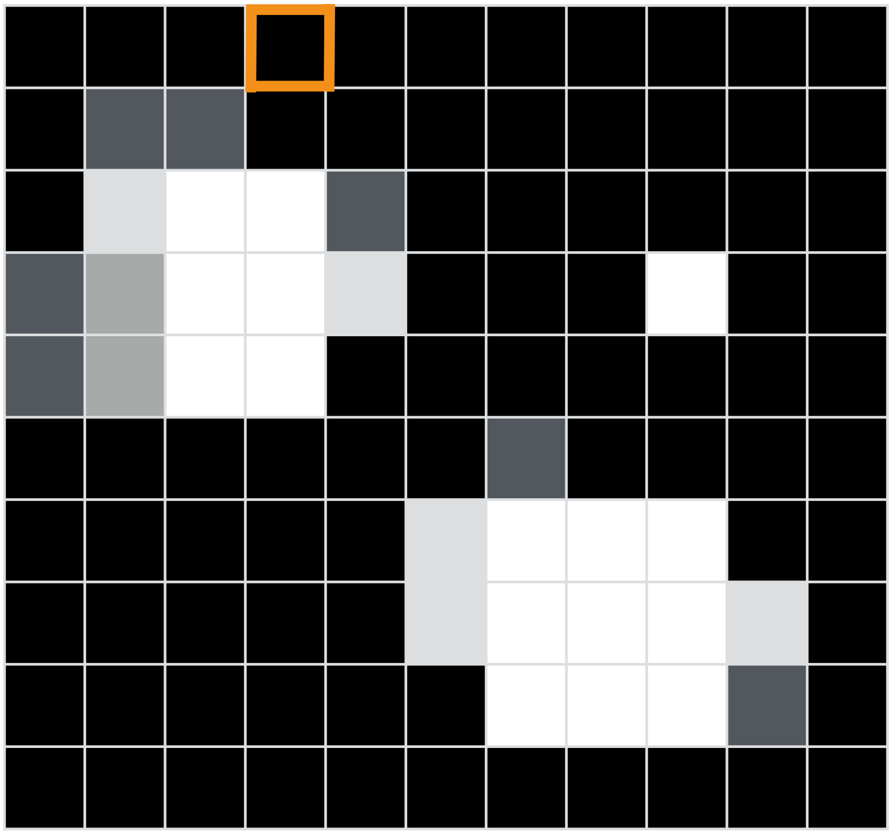
Detected Blobs

For each pixel ***p***

- **If** ***p*** is *white enough*
  AND ***p*** *is not visited*
  - Create a **new Blob**
  - Start **DFS** from ***p***
- **Mark** *p as* visited

First pixel is dark

Detecting All Blobs



T	T	T	T	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F

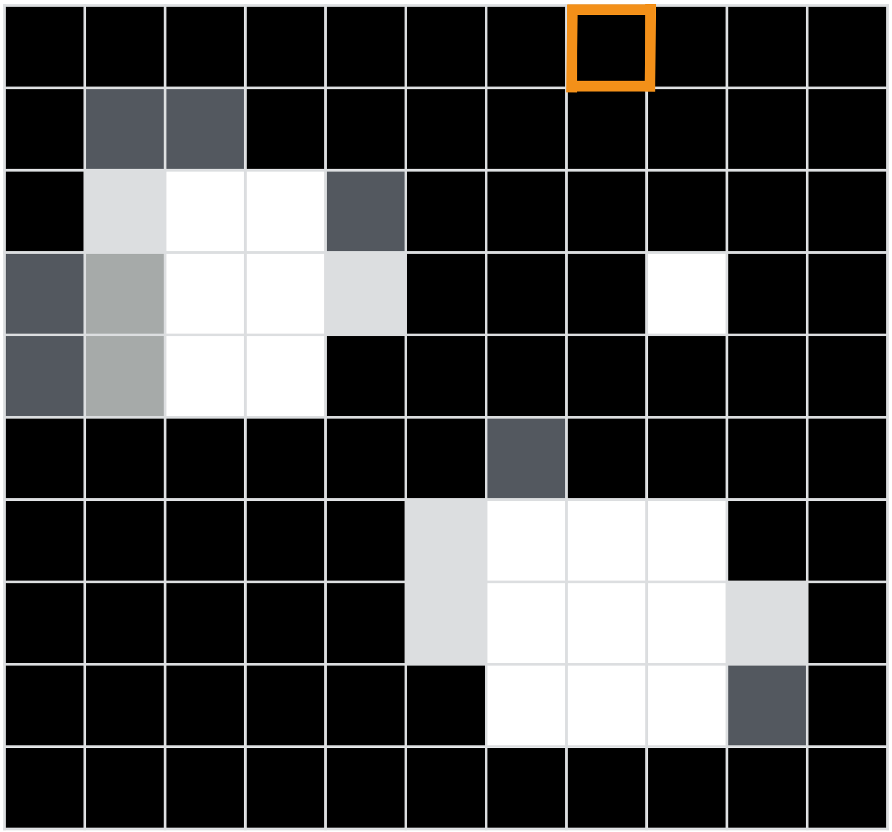
Detected Blobs

For each pixel ***p***

- **If** ***p*** is *white enough* AND ***p*** *is not visited*
  - Create a **new Blob**
  - Start **DFS** from ***p***
- **Mark** *p as* visited

Skip dark pixels and mark them as visited

Detecting All Blobs



T	T	T	T	T	T	T	T	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F

Detected Blobs

For each pixel ***p***

If ***p*** is *white enough*

AND ***p*** *is not visited*

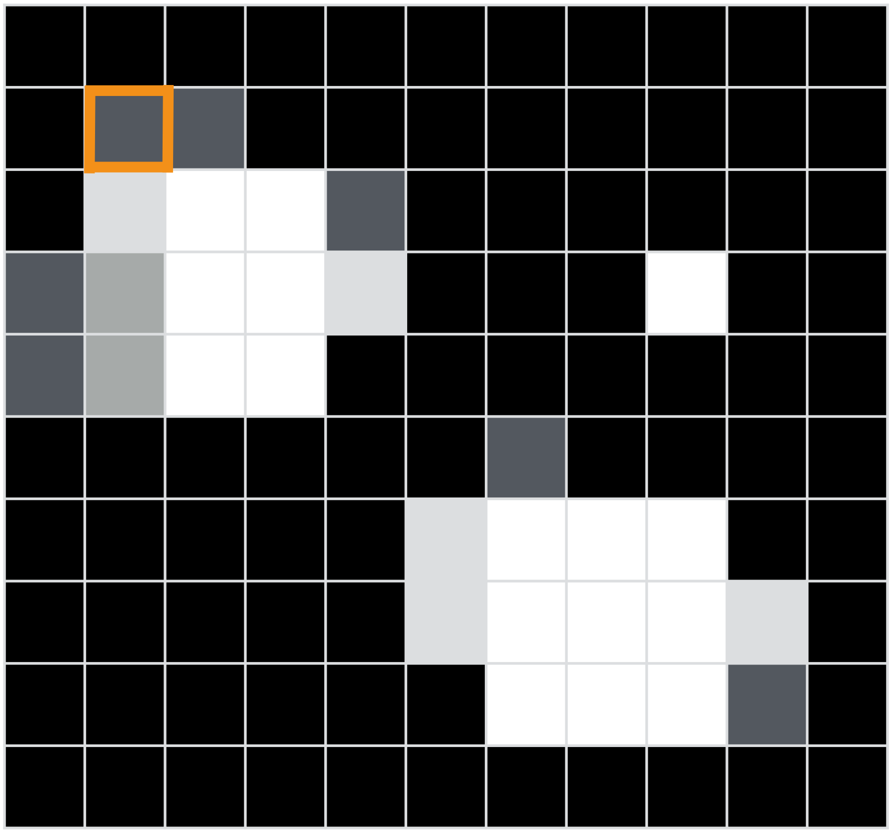
Create a **new Blob**

Start **DFS** from ***p***

Mark *p as* visited

Skip dark pixels and mark them as visited

Detecting All Blobs



T	T	T	T	T	T	T	T	T	T	T
T	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F

Detected Blobs

For each pixel ***p***

If ***p*** is *white enough*

AND ***p*** *is not visited*

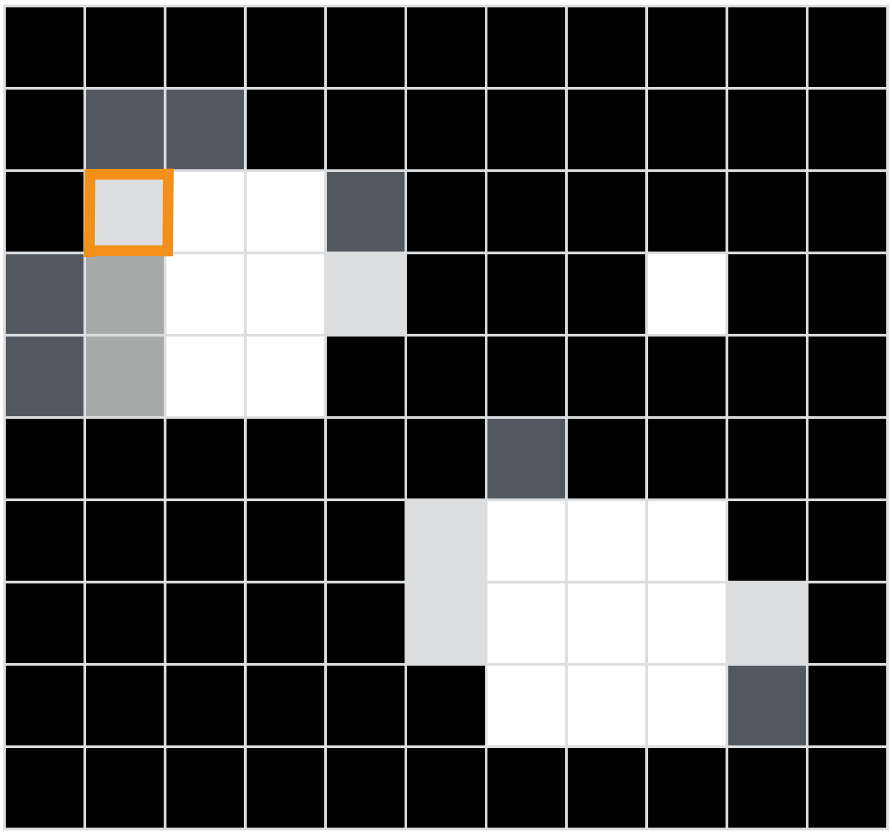
Create a **new Blob**

Start **DFS** from ***p***

Mark *p as* visited

Skip dark pixels and mark them as visited

Detecting All Blobs



T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F

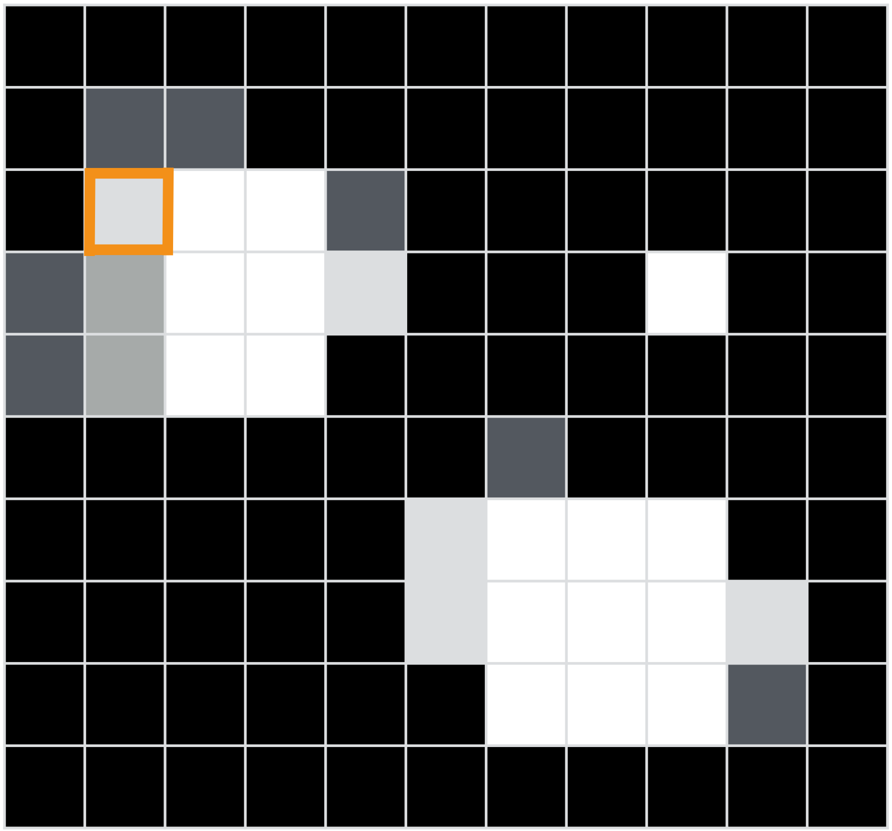
Detected Blobs

For each pixel ***p***

- **If** ***p*** is *white enough*
  - AND ***p*** *is not visited*
    - Create a **new Blob**
    - Start **DFS** from ***p***
- **Mark** *p as* visited

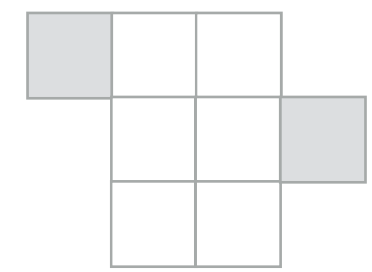
This pixel is light and has not been visited before. Iteration in the for loop is suspended and DFS starts

Detecting All Blobs



T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	F	F	F	F	F	F
F	T	T	T	T	T	F	F	F	F	F
F	T	T	T	T	F	F	F	F	F	F
F	F	T	T	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F

Detected Blobs



For each pixel ***p***

If ***p*** is *white enough*

AND ***p*** *is not visited*

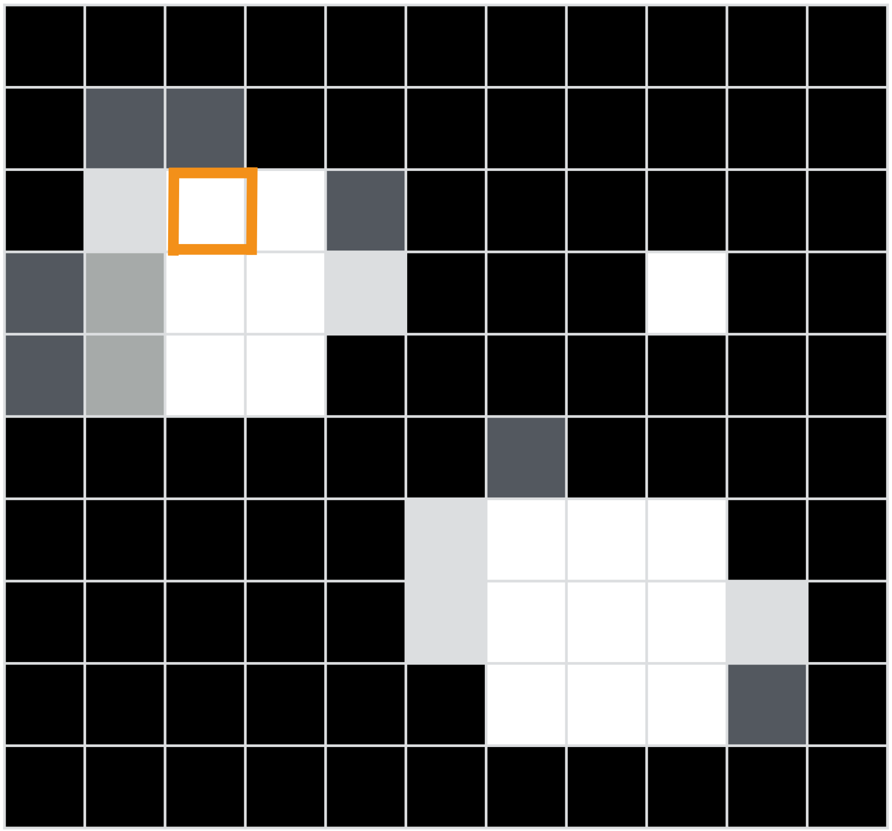
Create a **new Blob**

Start **DFS** from ***p***

Mark *p as* visited

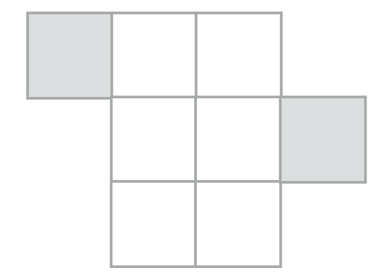
DFS adds all 'light' pixels adjacent to this pixel to a blob. All pixels visited by the DFS are marked as visited

Detecting All Blobs



T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	F	F	F	F	F	F
F	T	T	T	T	T	F	F	F	F	F
F	T	T	T	T	F	F	F	F	F	F
F	F	T	T	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F

Detected Blobs



For each pixel ***p***

If ***p*** is *white enough*

AND ***p*** *is not visited*

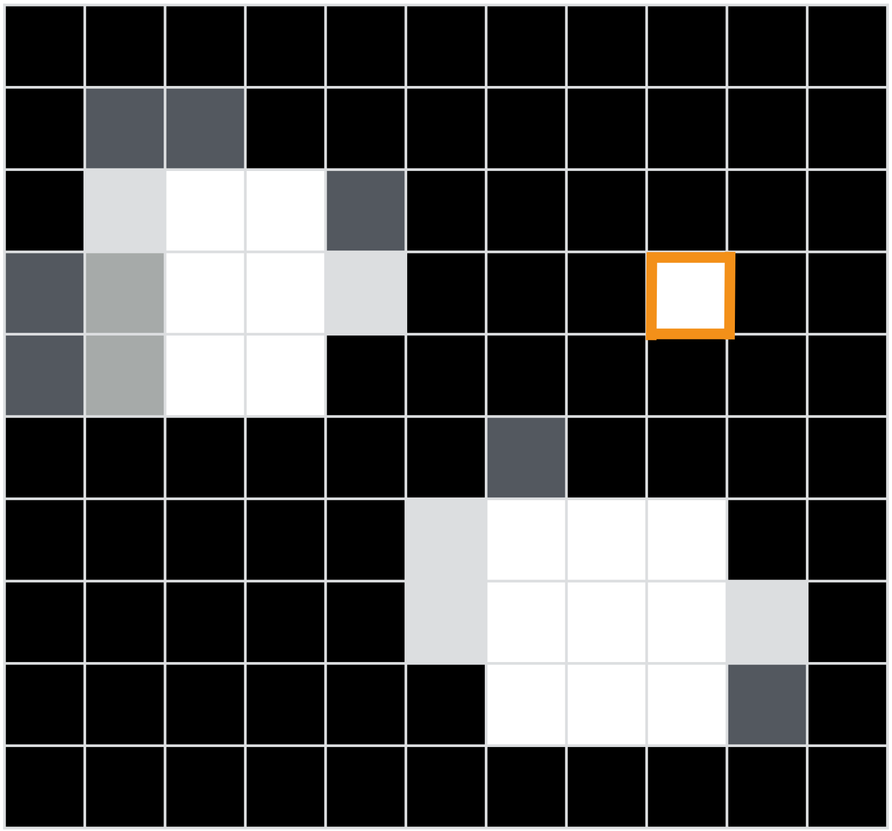
Create a **new Blob**

Start **DFS** from ***p***

Mark *p as* visited

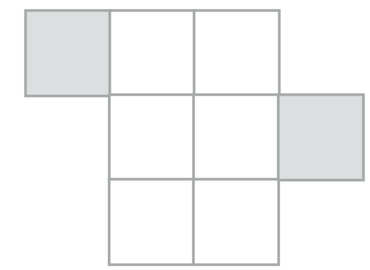
For loop proceeds and ignores this pixel because it is marked as visited

Detecting All Blobs



T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	F	F	F
F	T	T	T	T	F	F	F	F	F	F
F	F	T	T	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F

Detected Blobs

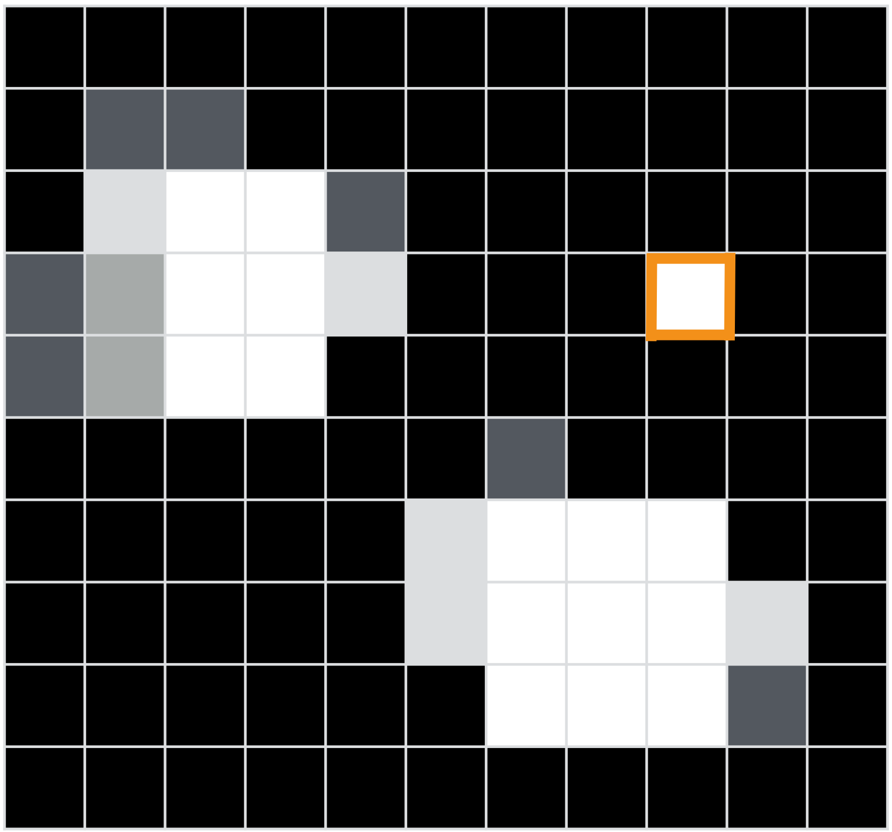


For each pixel ***p***

- **If** ***p*** is *white enough*
  AND ***p*** *is not visited*
  - Create a **new Blob**
  - Start **DFS** from ***p***
- **Mark** *p as* visited

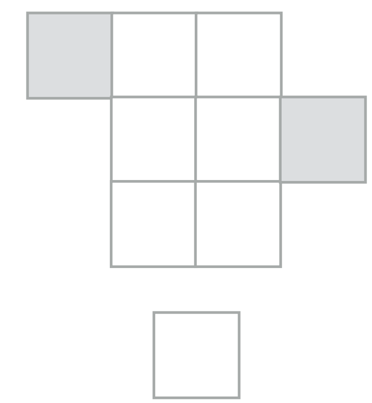
light and not visited!
Create a new blob and start
a new DFS

Detecting All Blobs



T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	F	F
F	T	T	T	T	F	F	F	F	F	F
F	F	T	T	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F

Detected Blobs



For each pixel ***p***

If ***p*** is *white enough*

AND ***p*** *is not visited*

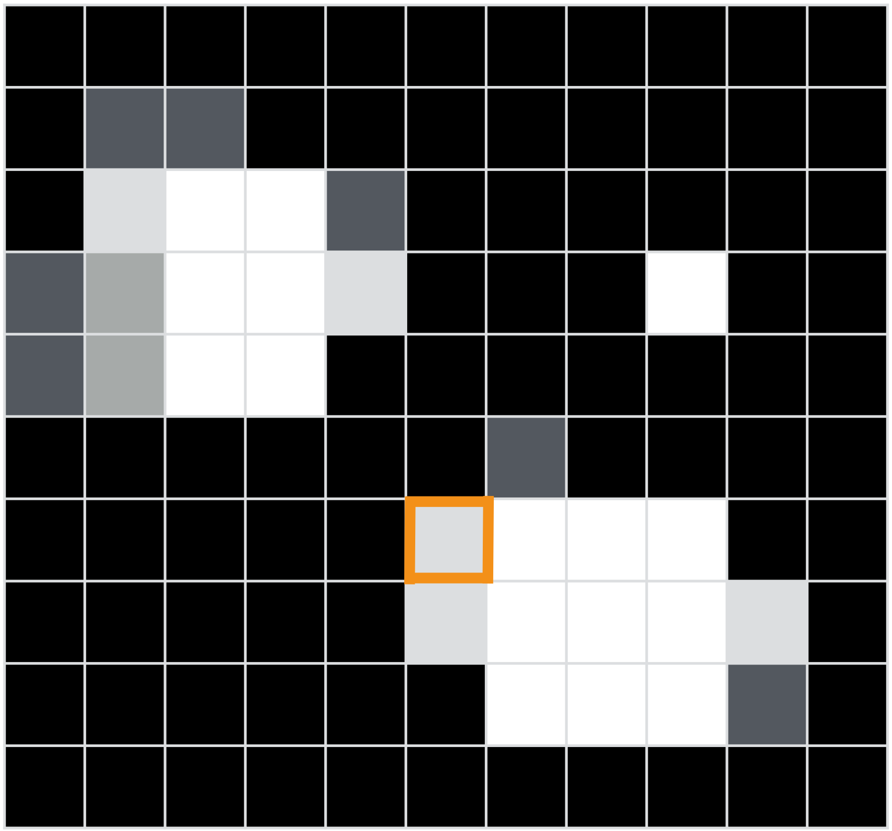
Create a **new Blob**

Start **DFS** from ***p***

Mark *p as* visited

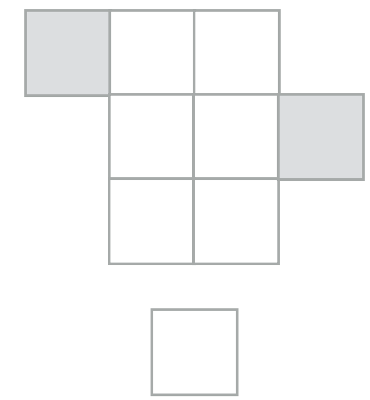
A new blob with only one pixel!

Detecting All Blobs



T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F

Detected Blobs

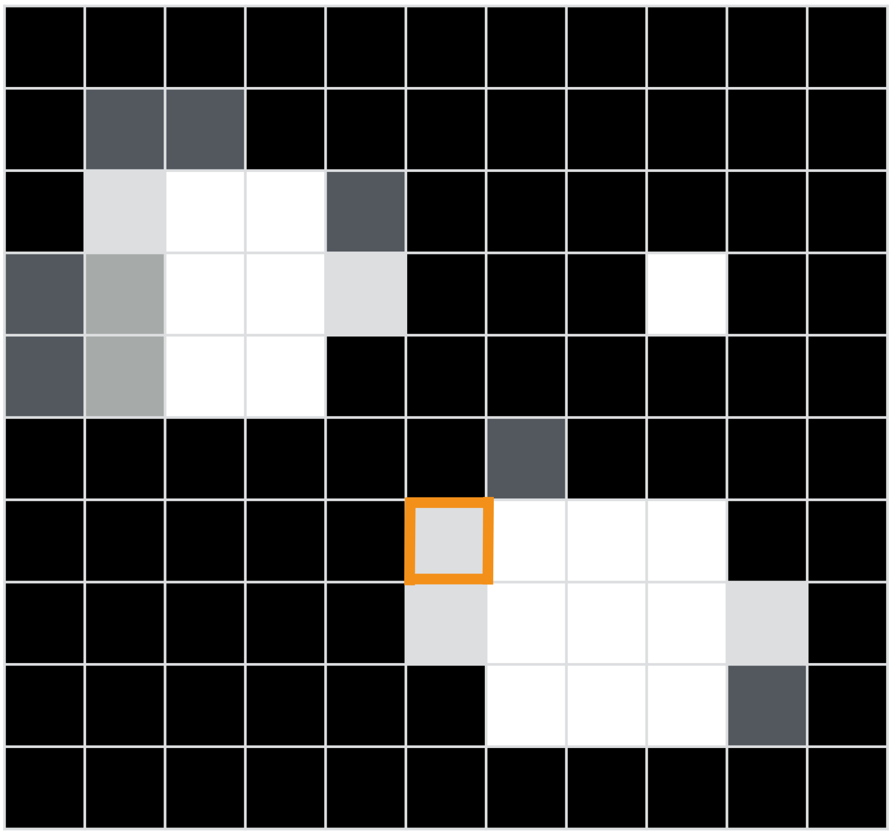


For each pixel ***p***

- **If** ***p*** is *white enough*
  AND ***p*** *is not visited*
  - Create a **new Blob**
  - Start **DFS** from ***p***
- **Mark** *p as* visited

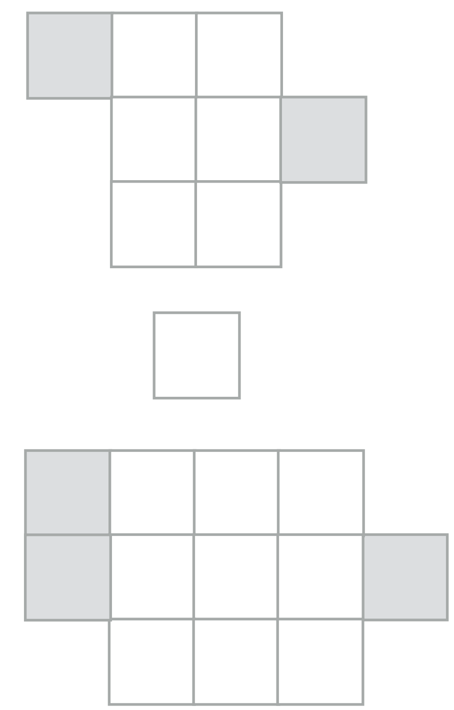
light and not visited!
Create a new blob and start
a new DFS

Detecting All Blobs



T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
F	F	F	F	T	T	T	T	T	T	T
F	F	F	F	F	T	T	T	T	T	T
F	F	F	F	F	F	T	T	T	T	F

Detected Blobs

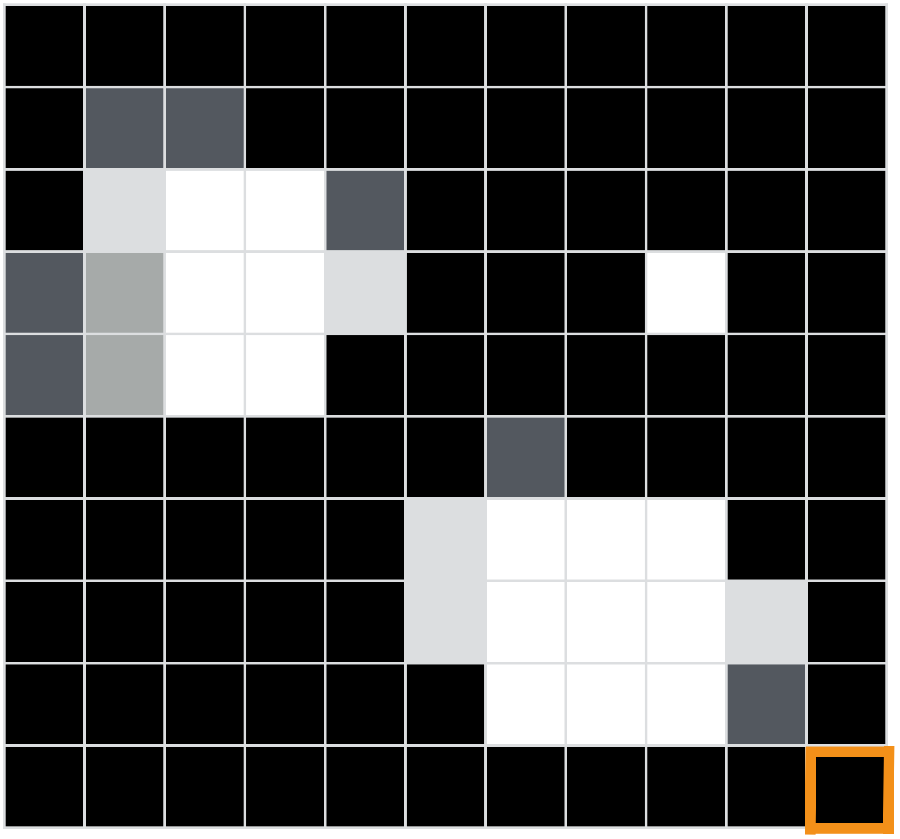


For each pixel ***p***

- **If** ***p*** is *white enough*
  AND ***p*** *is not visited*
  - Create a **new Blob**
  - Start **DFS** from ***p***
- **Mark** *p as* visited

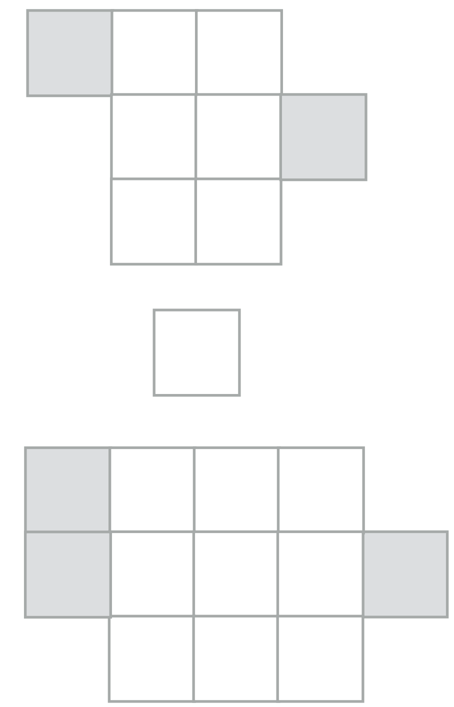
DFS adds all 'light' pixels adjacent to this pixel to a blob. All pixels visited by the DFS are marked as visited

Detecting All Blobs



T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T

Detected Blobs



For each pixel ***p***

If ***p*** is *white enough*

AND ***p*** *is not visited*

Create a **new Blob**

Start **DFS** from ***p***

Mark *p as* visited

Algorithm ends when all pixels have been marked as visited

Depth-First Search

DFS starting at *p*

Base cases?

Mark *p* as **visited**

Add *p* to the **blob**

DFS →

DFS ←

DFS ↑

DFS ↓

Depth-First Search

DFS starting at *p*

Base cases?

- Pixel out of bounds
- Pixel is dark
- Pixel is visited

Mark *p* as **visited**

Add *p* to the **blob**

DFS →

DFS ←

DFS ↑

DFS ↓

Bead Finder Notes

Number of Blobs is not known ahead of time. What **data structure** will you use to store them?

- **Array of Blobs?**
  - What should the size of the array be?
- **Linked List of Blobs?**
  - More implementation work!
  - Be careful not to traverse the whole list to add a blob!
- **java.util?**
  - Not allowed!
- **Others?**
  - You can assume access to Stack.java, Queue.java and ST.java. BUT, make sure to make a choice that is **efficient** and **makes sense!**

Bead Finder Notes

- Images are 640 x 480
  - Don’t hardwire! Your code should work for any image size.
- Private helper methods?
  - You will definitely need at least one! You can’t do recursion in a constructor!

Project Requirements

Implement the following:

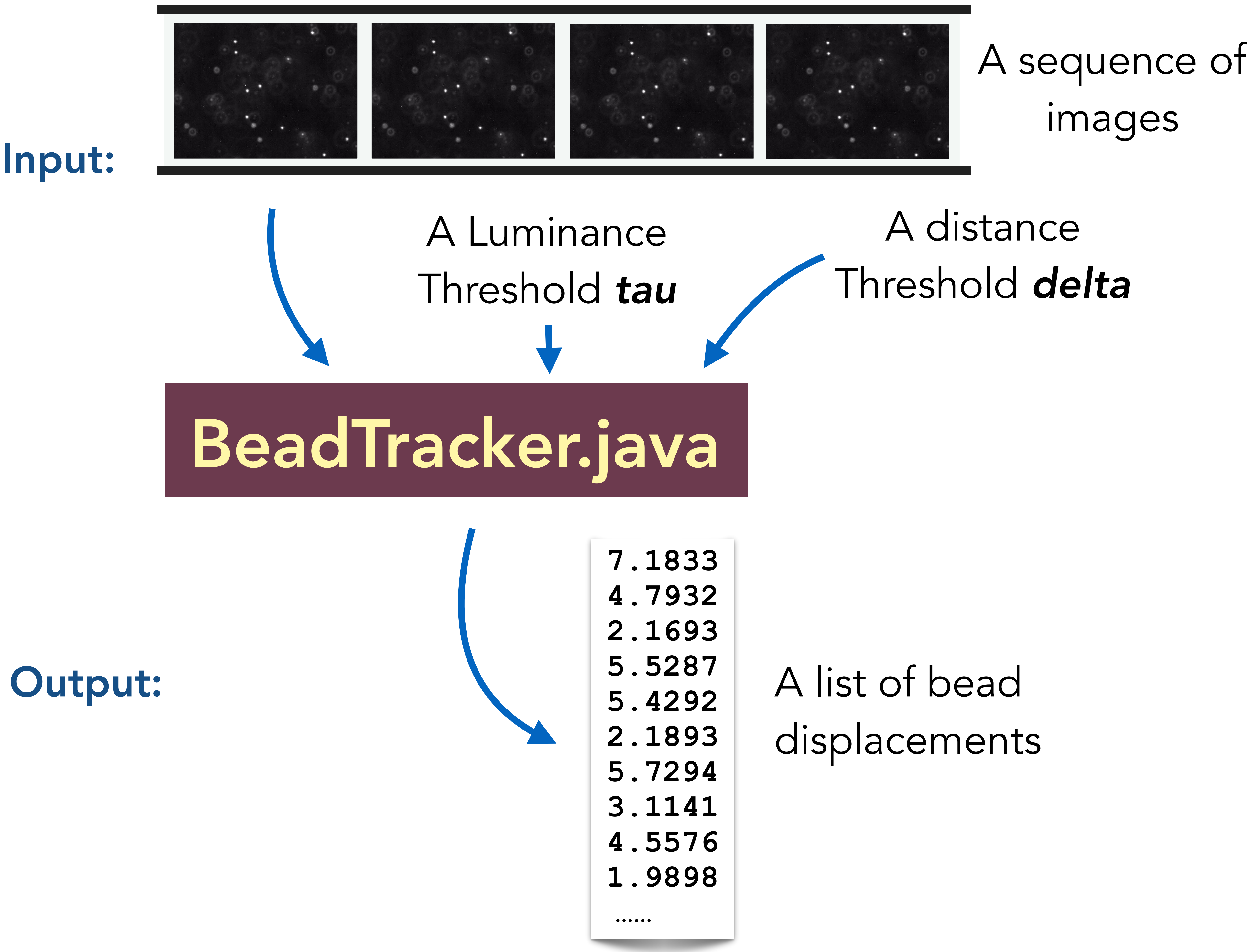
(1) `Blob.java`
Represents a set of adjacent pixels.

(2) `BeadFinder.java`
Detects all the "Beads" in a given picture.

(3) `BeadTracker.java`
Outputs displacements of beads over consecutive frames.

(4) `Avogadro.java`
Computes Avogadro's number from a given set of displacements.

(5) Readme File
Shows performance analysis.

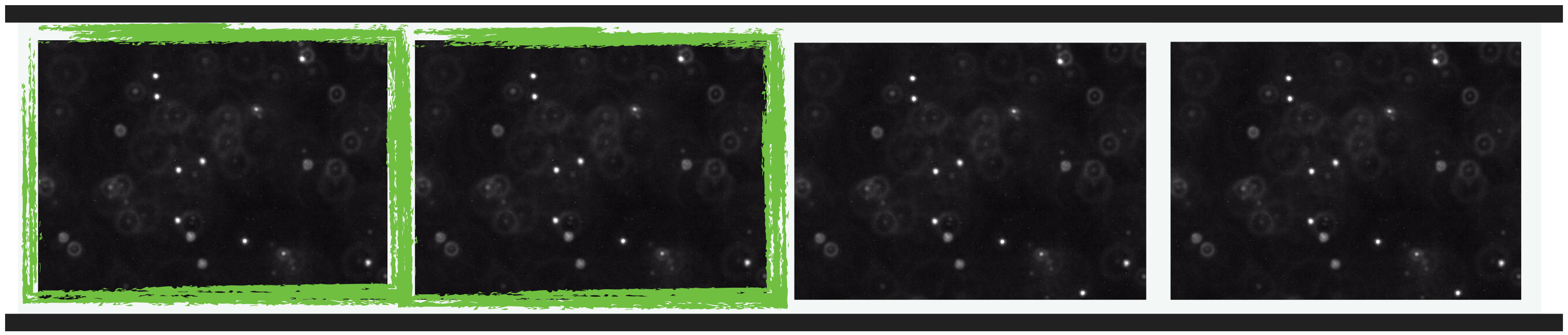

Input:
A sequence of images
A Luminance Threshold ***tau***
A distance Threshold ***delta***
BeadTracker.java
Output:
7.1833
4.7932
2.1693
5.5287
5.4292
2.1893
5.7294
3.1141
4.5576
1.9898
......
A list of bead displacements

BeadTracker.java

Displacements

```
7.1833
4.7932
2.1693
5.5287
5.4292
```

Image 1 Image 2



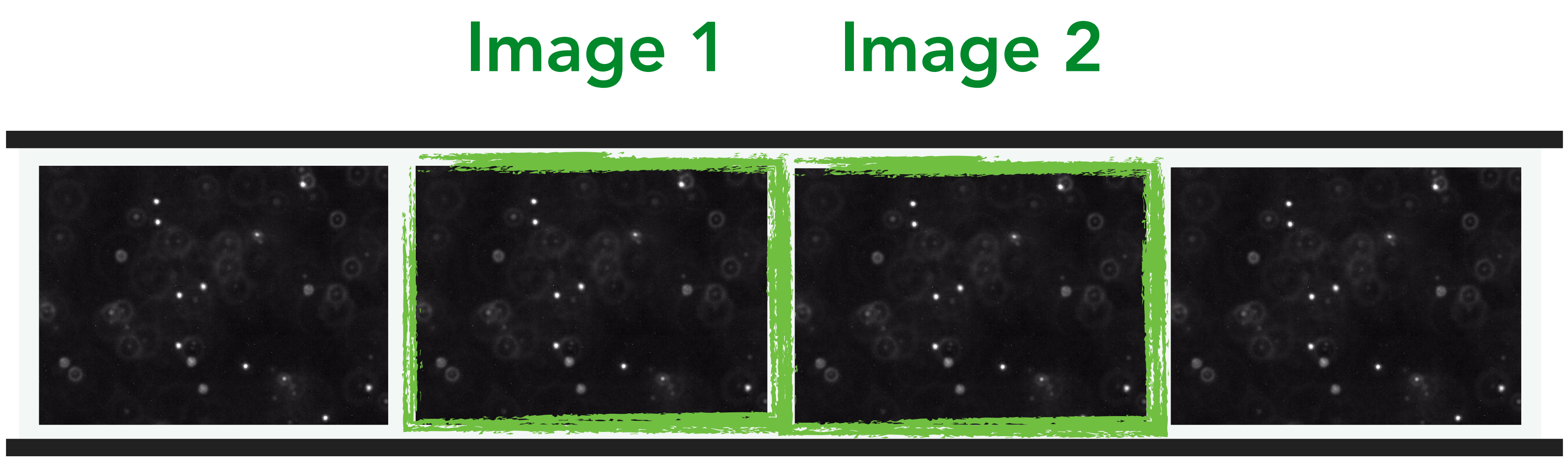
For every **pair** of consecutive images ***img1*** and ***img2***

Output how much every bead in img2 has **moved** from its position in img1

BeadTracker.java

Displacements

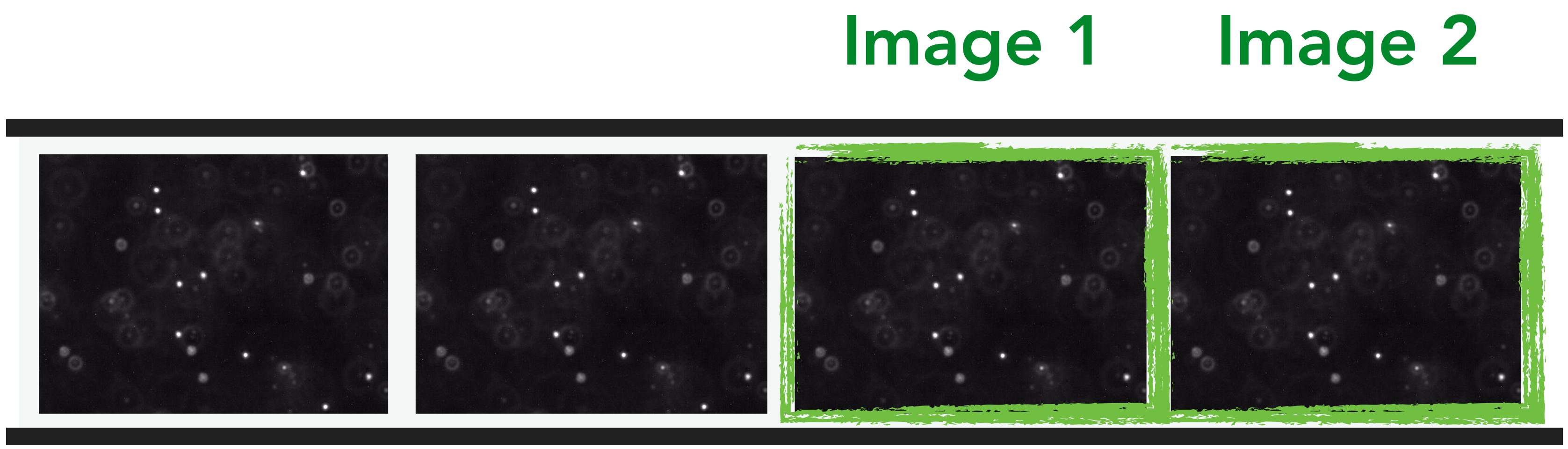
```
7.1833
4.7932
2.1693
5.5287
5.4292
7.1833
4.7932
2.1693
7.1833
4.7932
```



For every **pair** of consecutive images ***img1*** and ***img2***

Output how much every bead in img2 has **moved** from its position in img1

BeadTracker.java



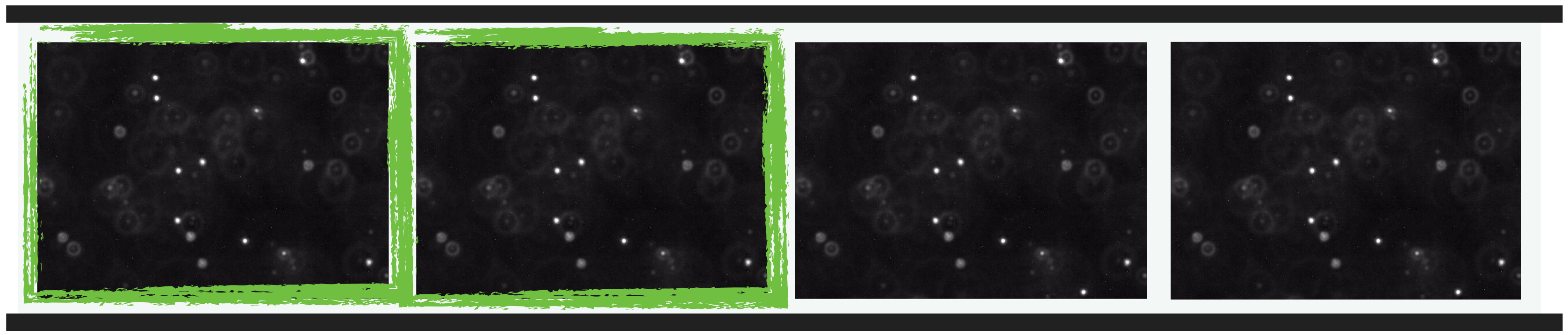
Displacements

```
7.1833
4.7932
2.1693
5.5287
5.4292
7.1833
4.7932
2.1693
7.1833
4.7932
2.1693
5.5287
5.4292
5.5287
5.4292
```

For every **pair** of consecutive images ***img1*** and ***img2***

Output how much every bead in img2 has **moved** from its position in img1

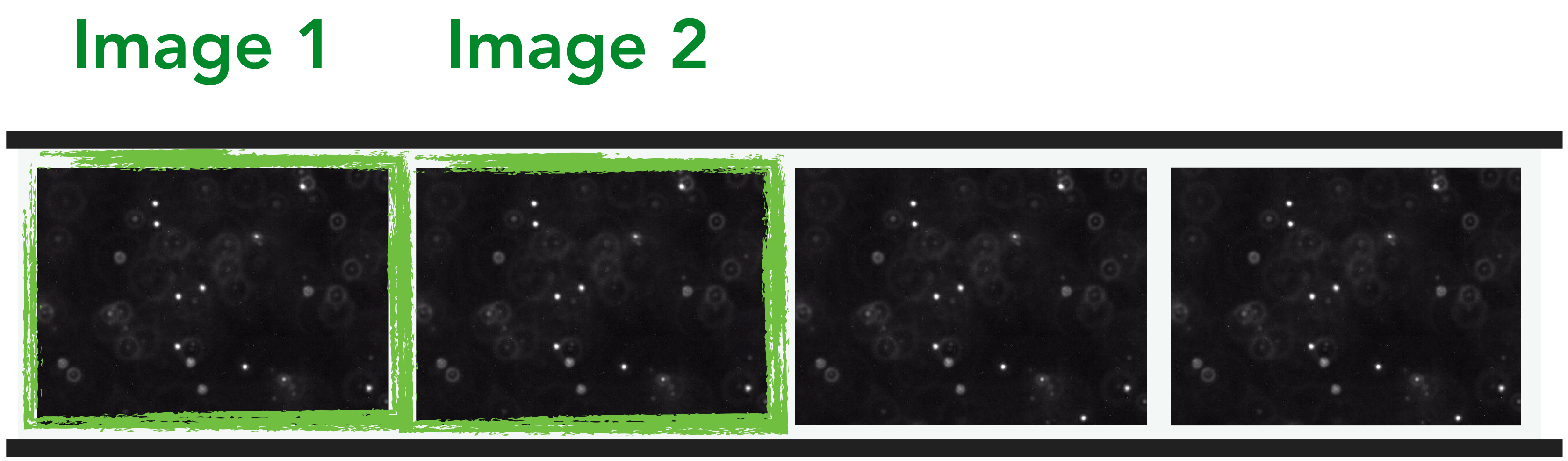
BeadTracker.java



img1 —input to—> BeadFinder —produces—> Blob [] beads1

img2 —input to—> BeadFinder —produces—> Blob [] beads2

BeadTracker.java



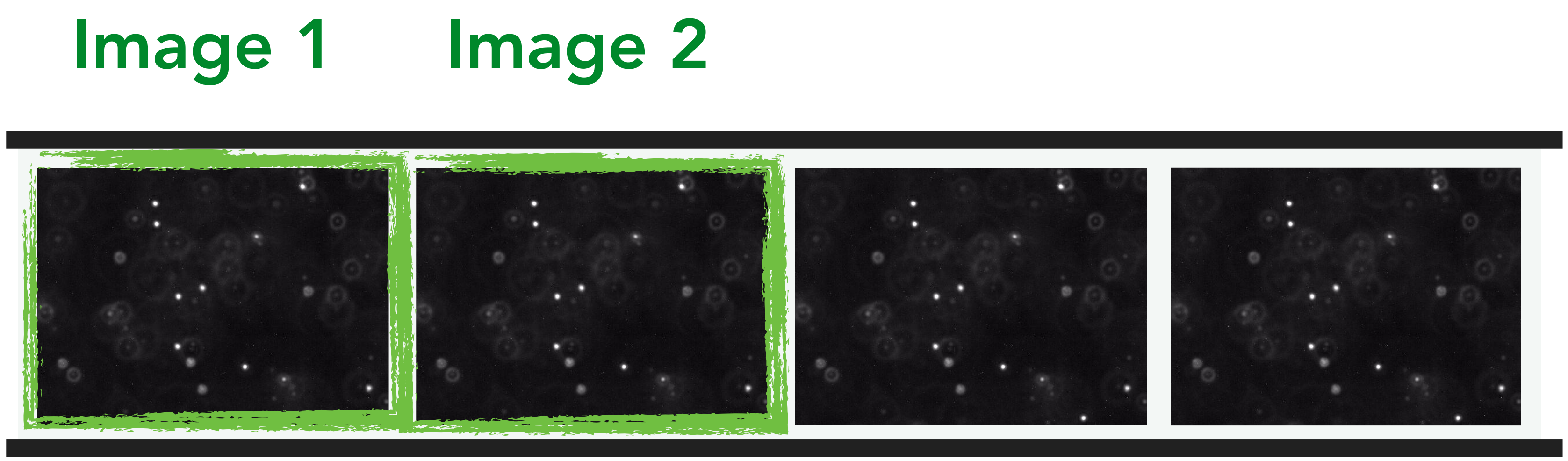
img1 —input to—> **BeadFinder** —produces—> **Blob [] beads1**
img2 —input to—> **BeadFinder** —produces—> **Blob [] beads2**

For each bead ***b*** in **beads2**

  Find ***closest*** bead in **beads1**

  output distance between ***b*** and ***closest***

BeadTracker.java

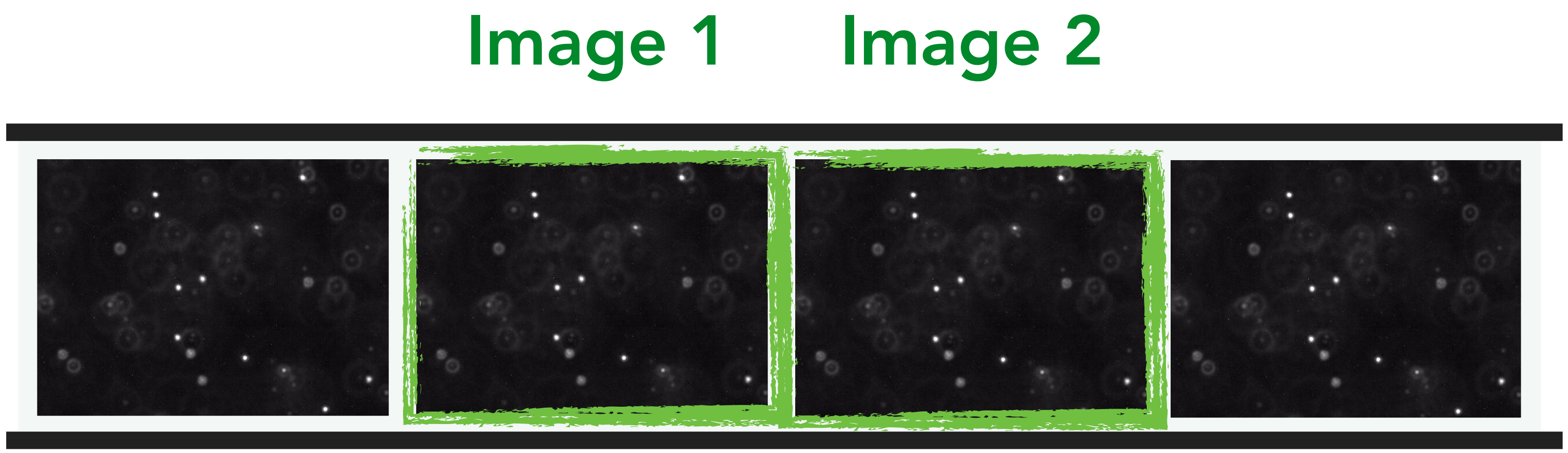


For each pair of images **img1** and **img2**

img1 —input to—> BeadFinder —produces—> **Blob [] beads1**
img2 —input to—> BeadFinder —produces—> **Blob [] beads2**

For each bead ***b*** in **beads2**
Find closest bead in **beads1**
output distance between ***b*** and ***closest***

BeadTracker.java



For each pair of images **img1** and **img2**

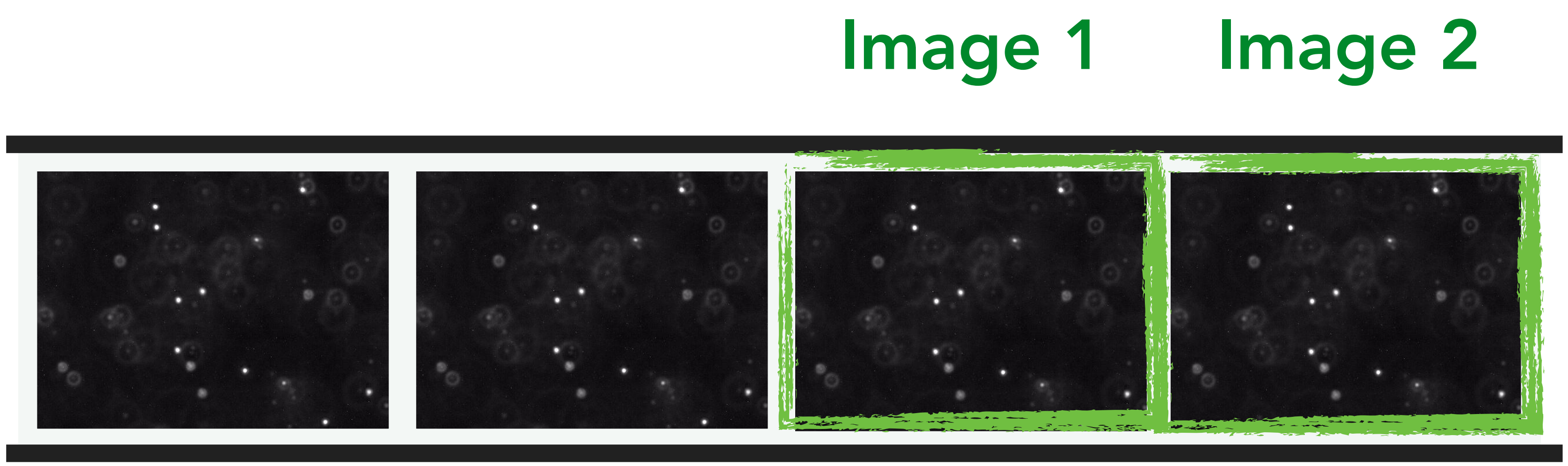
img1 —input to—> BeadFinder —produces—> **Blob [] beads1**
img2 —input to—> BeadFinder —produces—> **Blob [] beads2**

For each bead ***b*** in **beads2**

Find closest bead in **beads1**

output distance between ***b*** and ***closest***

BeadTracker.java



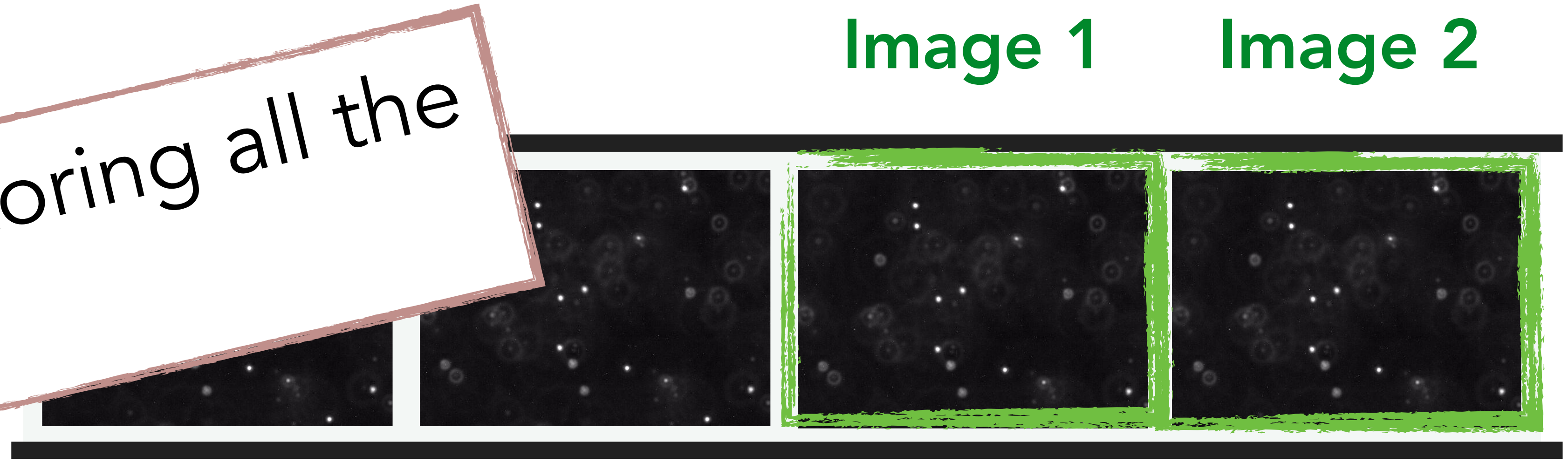
For each pair of images **img1** and **img2**

img1 —input to—> BeadFinder —produces—> **Blob [] beads1**
img2 —input to—> BeadFinder —produces—> **Blob [] beads2**

For each bead ***b*** in **beads2**
Find closest bead in **beads1**
output distance between ***b*** and ***closest***

BeadTracker.java

Avoid storing all the images!



For each pair of images **img1** and **img2**

img1 —input to—> BeadFinder —produ

img2 —input to—> BeadFinder —produ ads2

Avoid finding beads for the same image more than one

For each bead ***b*** in **beads2**

Find closest bead in **beads1**

output distance between ***b*** and ***closest***

Project Requirements

Implement the following:

(1) `Blob.java`
Represents a set of adjacent pixels.

(2) `BeadFinder.java`
Detects all the "Beads" in a given picture.

(3) `BeadTracker.java`
Outputs displacements of beads over consecutive frames.

(4) `Avogadro.java`
Computes Avogadro's number from a given set of displacements.

(5) Readme File
Shows performance analysis.

Avogadro.java

Receives as input a sequence of displacements.

Avogadro's Number is $\boxed{N_A = R / k}$

Where R is given and K can be computed using:

$$\boxed{D = kT / 6\pi\eta\rho}$$

Where T, π, η and ρ are given and D can be computed using:

$$\boxed{\sigma^2 = 2D\Delta t}$$

Where Δt is given and σ^2 is your job to compute!

Final Tips

- Be careful about units. **Convert** every read displacement from **pixels to meters** before using it in any formula.

- **Avogadro** can be implemented and tested **independently**.

- **Constants**! No magic numbers + No cryptic names!

- **Timing Tests!** Read Checklist + Use StopWatch.java + Redirect output to a file.

Image Sources

- **Slide 2:**
  - https://upload.wikimedia.org/wikipedia/commons/thumb/3/32/Robert_Brown_(botanist).jpg/220px-Robert_Brown_(botanist).jpg
- **Slide 3:**
  - https://cdn.miniphysics.com/wp-content/uploads/2011/01/brownianmotion.gif
  - https://upload.wikimedia.org/wikipedia/commons/d/d3/Albert_Einstein_Head.jpg
- **Slide 4:**
  - https://en.wikipedia.org/wiki/Jean_Baptiste_Perrin#/media/File:Jean_Perrin_1926.jpg