

# Введение в фотограмметрию

## Сопоставление ключевых точек



### Фотограмметрия. Лекция 3

- Brute-Force & ANN feature matching
  - HNSW, FAISS
  - Guided matching
  - Преселекция пар для сопоставления
  - Фильтрация сопоставлений
  - RANSAC
- Полярный Николай  
[polarnick239@gmail.com](mailto:polarnick239@gmail.com)

# Задача сопоставления фотографий

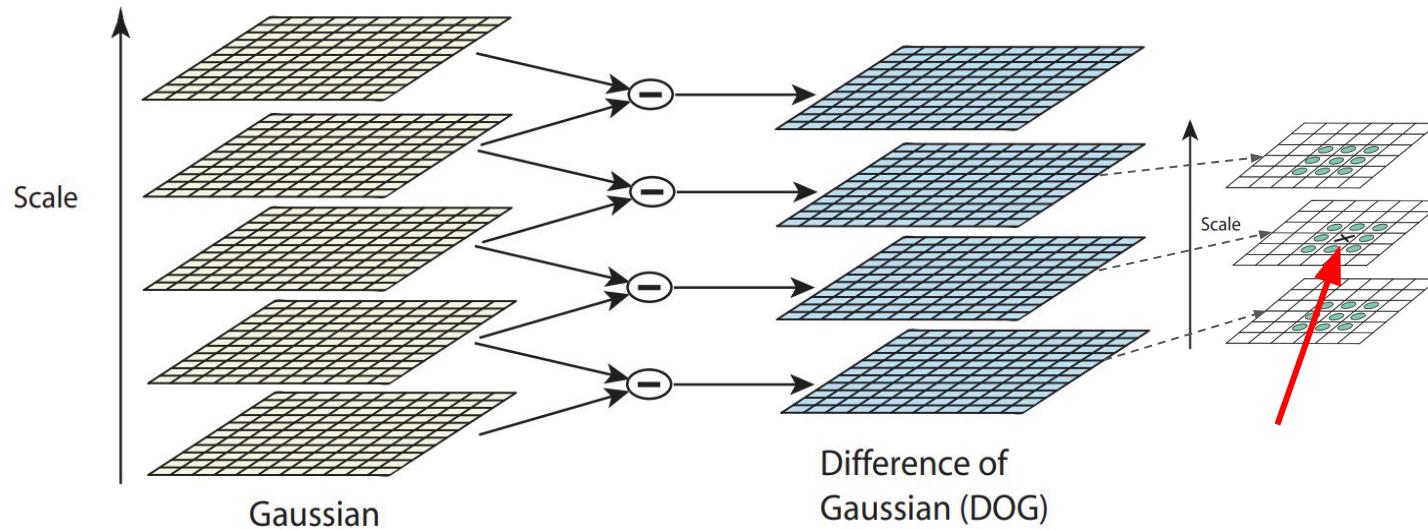


# Задача сопоставления фотографий



- 1) **Detector:** нашли на каждой фотографии ключевые точки

# Задача сопоставления фотографий



- 1) **Detector:** нашли на каждой фотографии ключевые точки

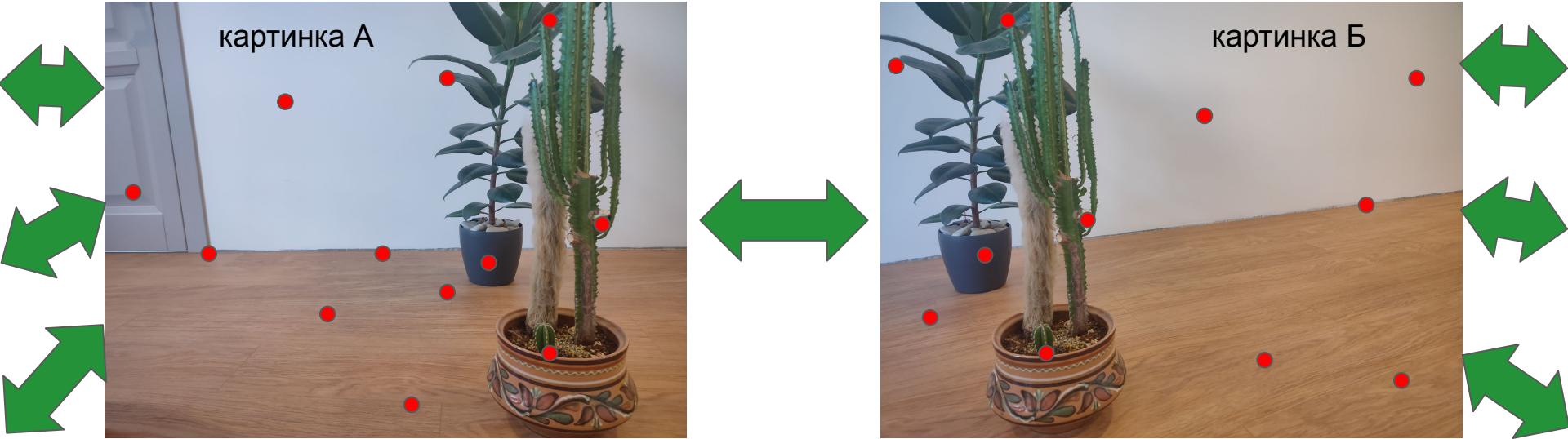
# Задача сопоставления фотографий



- 1) **Detector:** нашли на каждой (**независимо!**) фотографии ключевые точки

**Почему важно что мы на каждой фотографии детектируем точки независимо?**

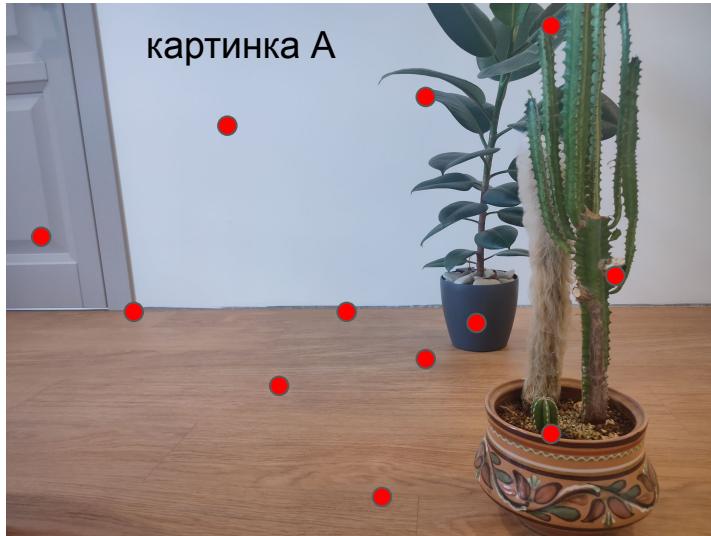
# Задача сопоставления фотографий



1) **Detector:** нашли на каждой (**независимо!**) фотографии ключевые точки

**Почему важно что мы на каждой фотографии детектируем точки независимо?**

# Задача сопоставления фотографий



1) **Detector:** нашли на каждой (**независимо!**) фотографии ключевые точки

Почему важно что мы на каждой фотографии детектируем точки независимо?

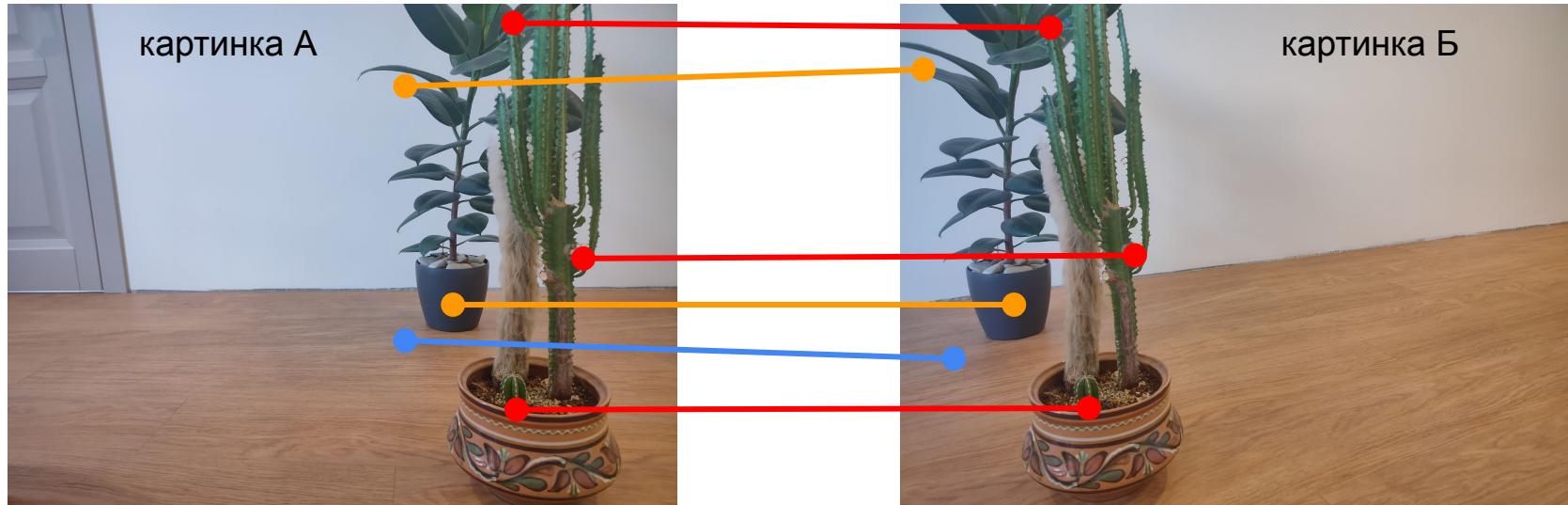
**А за счет чего тогда точки детектированные на одной фотографии - будут детектированы на другой?**

# Задача сопоставления фотографий



- 1) **Detector:** нашли на каждой (**независимо!**) фотографии ключевые точки
- 2) **Descriptor:** построили дескрипторы (**128 x float32**) для каждой точки

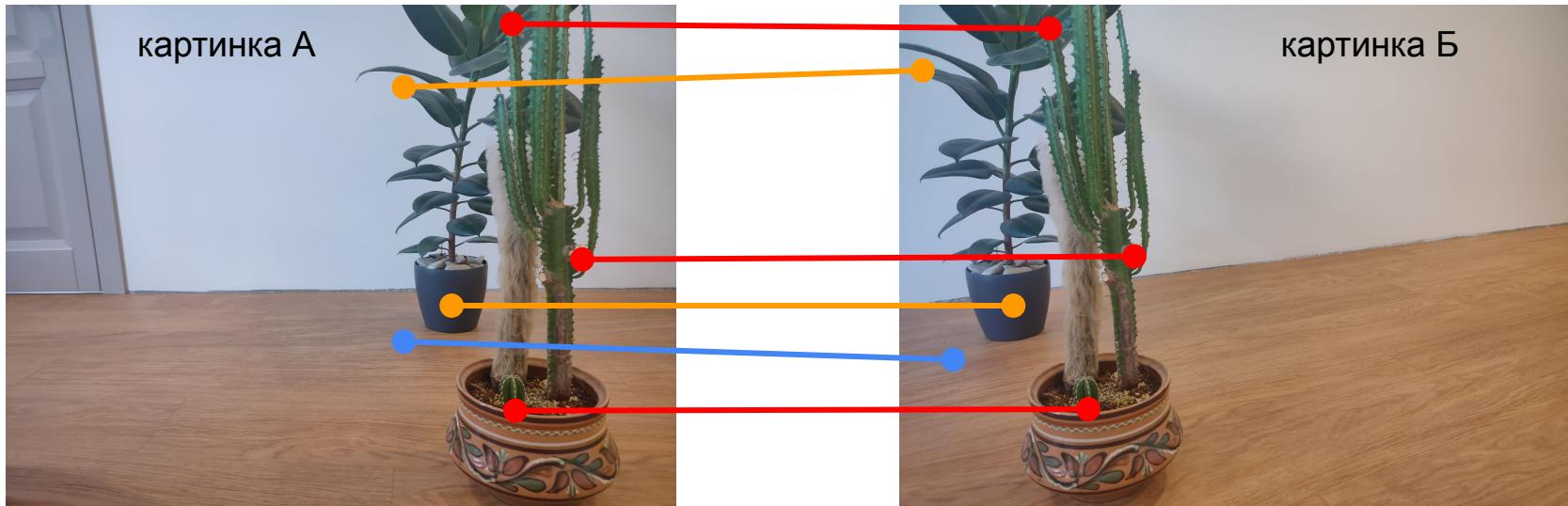
# Задача сопоставления фотографий



- 1) **Detector:** нашли на каждой (**независимо!**) фотографии ключевые точки
- 2) **Descriptor:** построили дескрипторы (**128 x float32**) для каждой точки
- 3) **Как теперь сопоставить ключевые точки пары изображений?**

$$dist(A, B) = \sqrt{\sum_{i=1}^{128} (A_i - B_i)^2}$$

# Задача сопоставления фотографий



- 1) **Detector:** нашли на каждой (**независимо!**) фотографии ключевые точки
- 2) **Descriptor:** построили дескрипторы (**128 x float32**) для каждой точки
- 3) **Сопоставляем:** ищем для каждой точки из **А** - самую похожую точку из **Б**

# Сопоставление точек: Brute-Force matching

Вход:

- **M картинок**
- **N точек на каждой картинке**
- у каждой точки **(128 x float32) SIFT** дескриптор

# Сопоставление точек: Brute-Force matching

Вход:

- **M картинок**
- **N точек на каждой картинке**
- у каждой точки **(128 x float32) SIFT** дескриптор

Выход (для каждой пары картинок А и Б):

- **Nearest-Neighbor:** для каждой точки из А - самая похожая точка из Б

# Сопоставление точек: Brute-Force matching

Вход:

- **M картинок**
- **N точек на каждой картинке**
- у каждой точки **(128 x float32) SIFT** дескриптор

Выход (для каждой пары картинок А и Б):

- **Nearest-Neighbor:** для каждой точки из А - самая похожая точка из Б

**Какая общая асимптотика?**

# Сопоставление точек: Brute-Force matching

Вход:

- **M картинок**
- **N точек на каждой картинке**
- у каждой точки (**128 x float32**) **SIFT** дескриптор

Выход (для каждой пары картинок А и Б):

- **Nearest-Neighbor:** для каждой точки из А - самая похожая точка из Б

**Brute-Force matching:**  $O(M^2 * N^2)$

# Сопоставление точек: Brute-Force matching

Вход:

- **M картинок**
- **N точек на каждой картинке**
- у каждой точки **(128 x float32) SIFT** дескриптор

Выход (для каждой пары картинок А и Б):

- **Nearest-Neighbor:** для каждой точки из А - самая похожая точка из Б

Brute-Force matching:  $O(M^2 * N^2)$

Как можно ускорить второй множитель?

# Сопоставление точек: Brute-Force matching

Brute-Force matching:  $O(M^2 * N^2)$

- у каждой точки (**128 x float32**) SIFT дескриптор  
это (**128 x const**) Flops на каждый расчет расстояния Евклида  
**как ускорить?**

# Сопоставление точек: Brute-Force matching

Brute-Force matching:  $O(M^2 * N^2)$

- у каждой точки (**128 x float32**) SIFT дескриптор - это (**128 x const**) Flops
- можно заменить на (**512 x bool**) MLDB - это (**16 x popcount32**) операций

# Сопоставление точек: Brute-Force matching

Brute-Force matching:  $O(M^2 * N^2)$

- у каждой точки (**128 x float32**) SIFT дескриптор - это (**128 x const**) Flops
- можно заменить на (**512 x bool**) MLDB - это (**16 x popcount32**) операций  
**как ускорить?**

# Сопоставление точек: Brute-Force matching

Brute-Force matching:  $O(M^2 * N^2)$

- у каждой точки (**128 x float32**) SIFT дескриптор - это (**128 x const**) Flops
- можно заменить на (**512 x bool**) MLDB - это (**16 x popcount32**) операций
- можно считать на видеокартах (**GPU**) - это, условно, x100 ускорение

# Сопоставление точек: Brute-Force matching

Brute-Force matching:  $O(M^2 * N^2)$

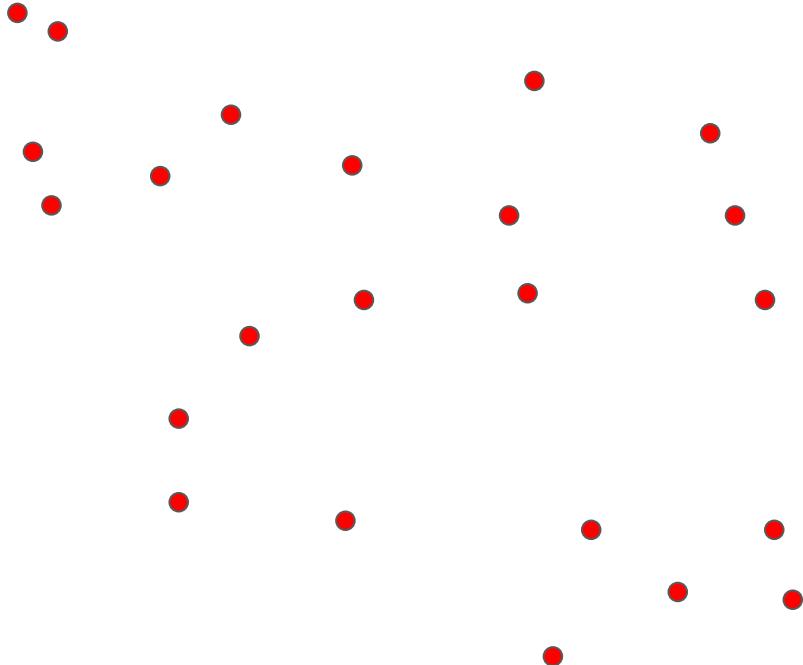
- у каждой точки (**128 x float32**) SIFT дескриптор - это (**128 x const**) Flops
- можно заменить на (**512 x bool**) MLDB - это (**16 x popcount32**) операций
- можно считать на видеокартах (**GPU**) - это, условно, x100 ускорение  
**как ускорить? как ускоряют поиск ближайшего соседа?**

# Сопоставление точек: Brute-Force matching

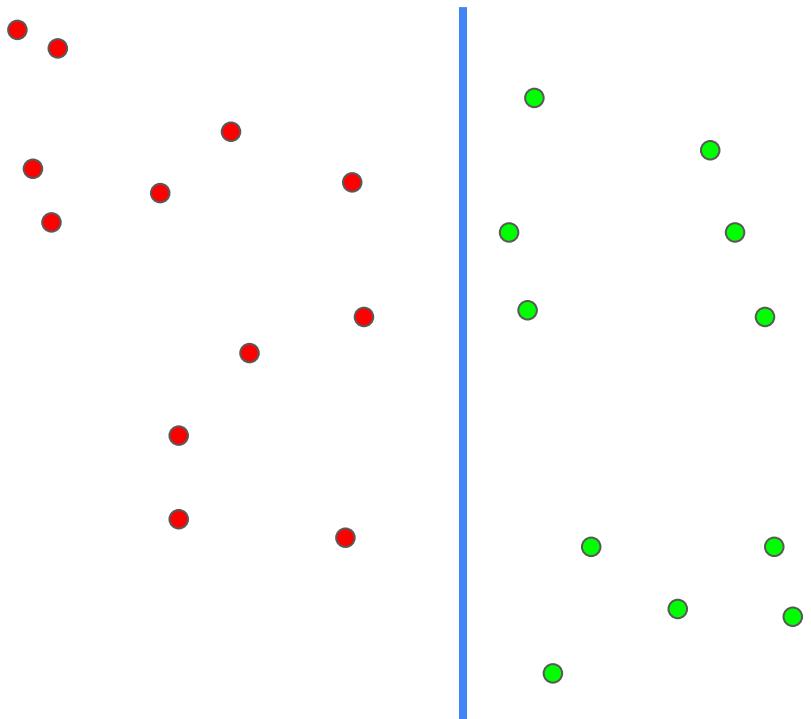
Brute-Force matching:  $O(M^2 * N^2)$

- у каждой точки (**128 x float32**) SIFT дескриптор - это (**128 x const**) Flops
- можно заменить на (**512 x bool**) MLDB - это (**16 x popcount32**) операций
- можно считать на видеокартах (**GPU**) - это, условно, x100 ускорение
- использовать **структуры поиска** (приближенного?) ближайшего соседа

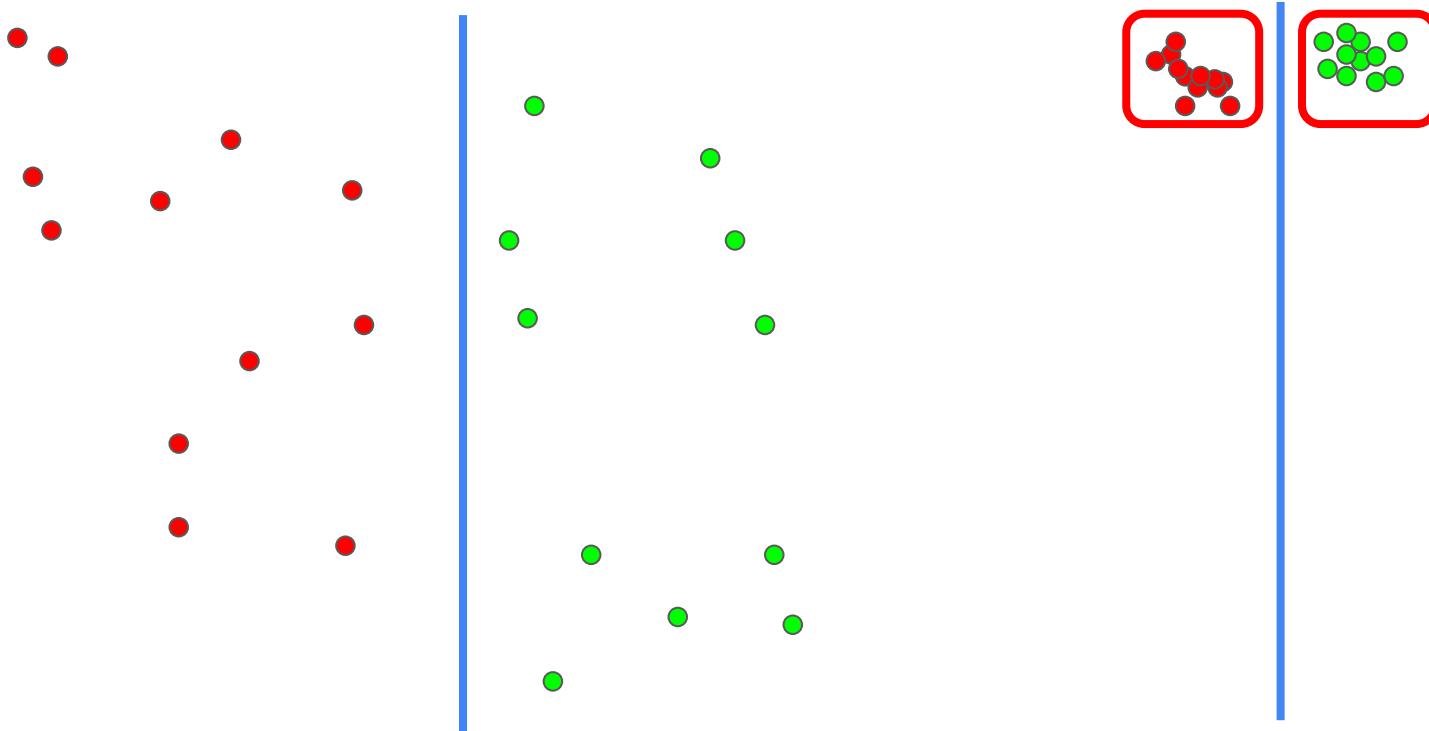
# Сопоставление точек: KD-tree



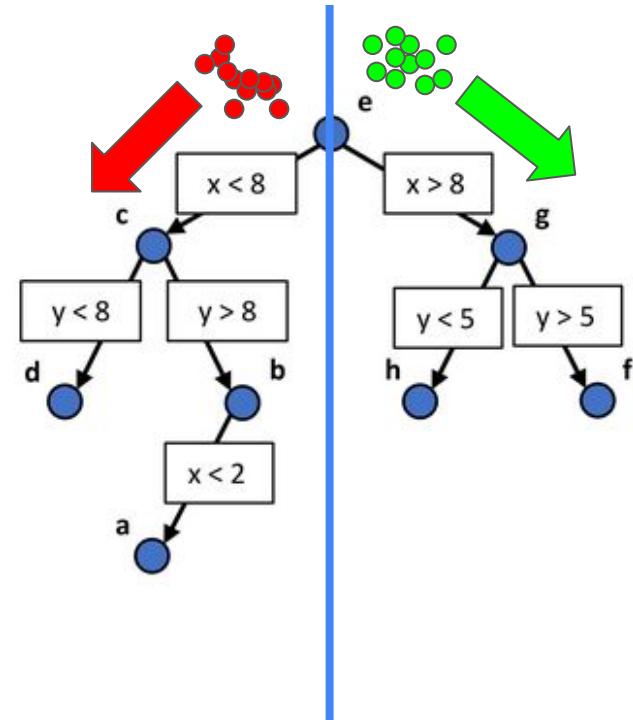
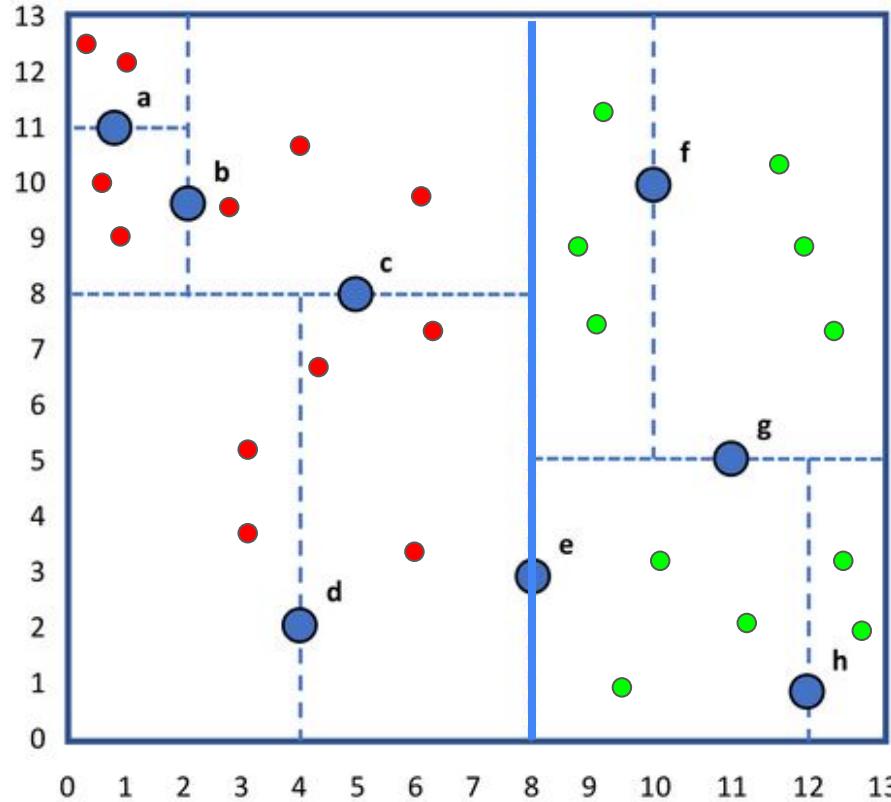
# Сопоставление точек: KD-tree



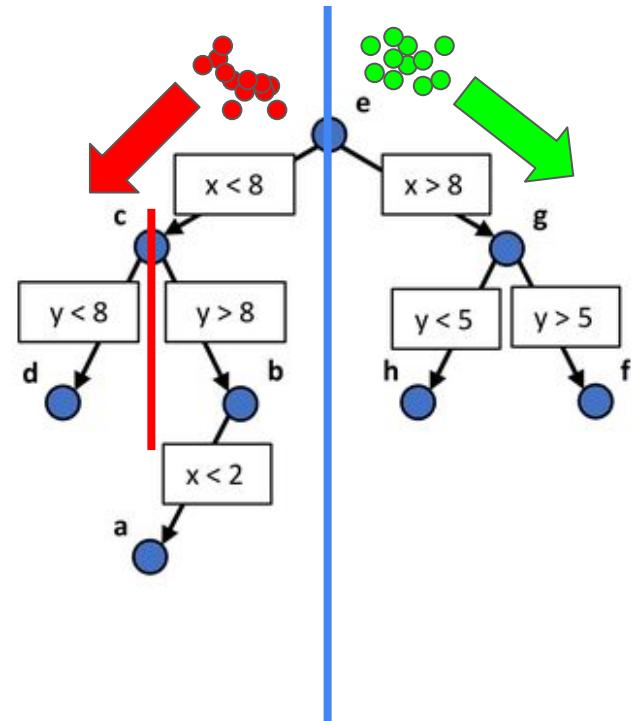
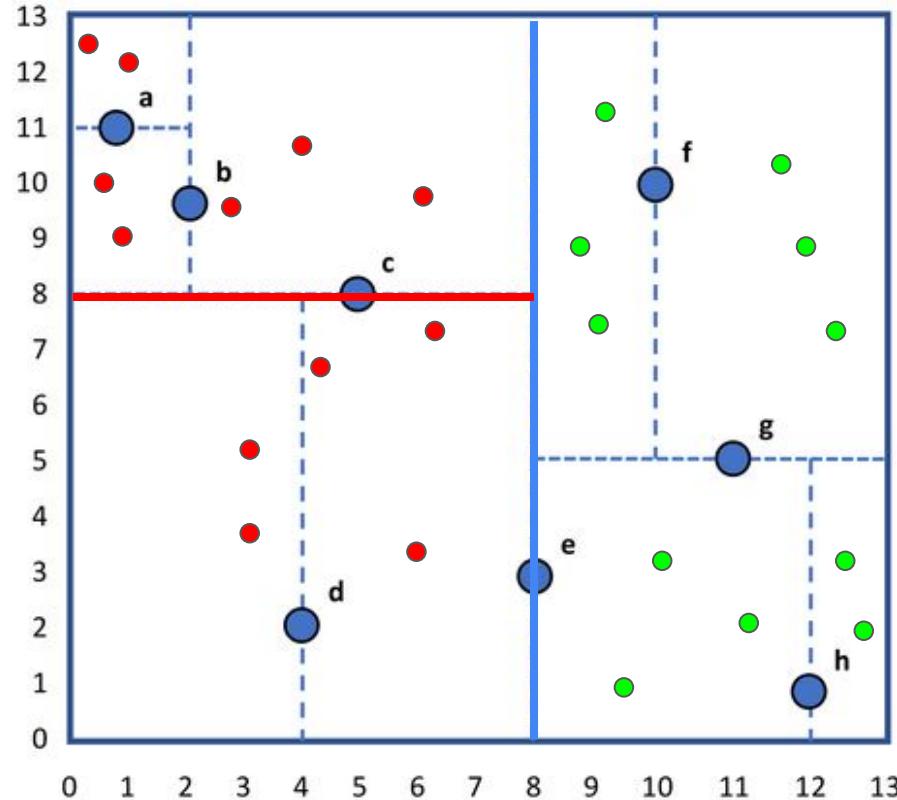
# Сопоставление точек: KD-tree



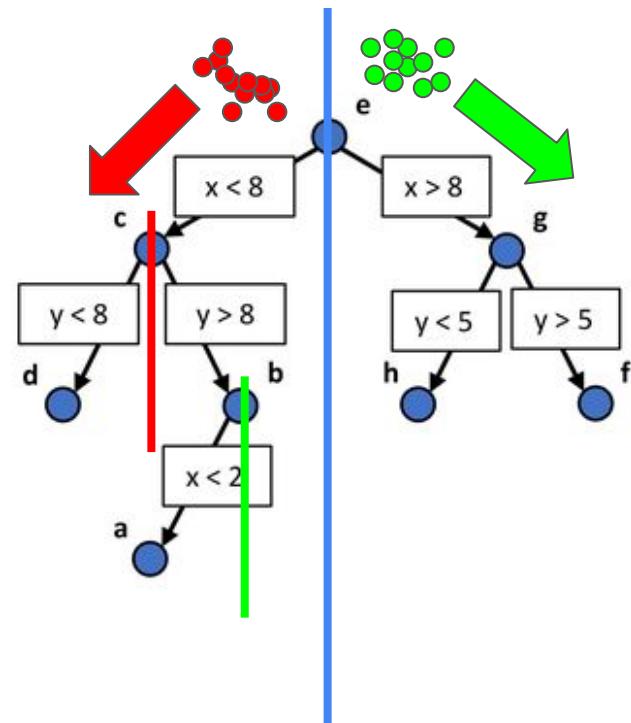
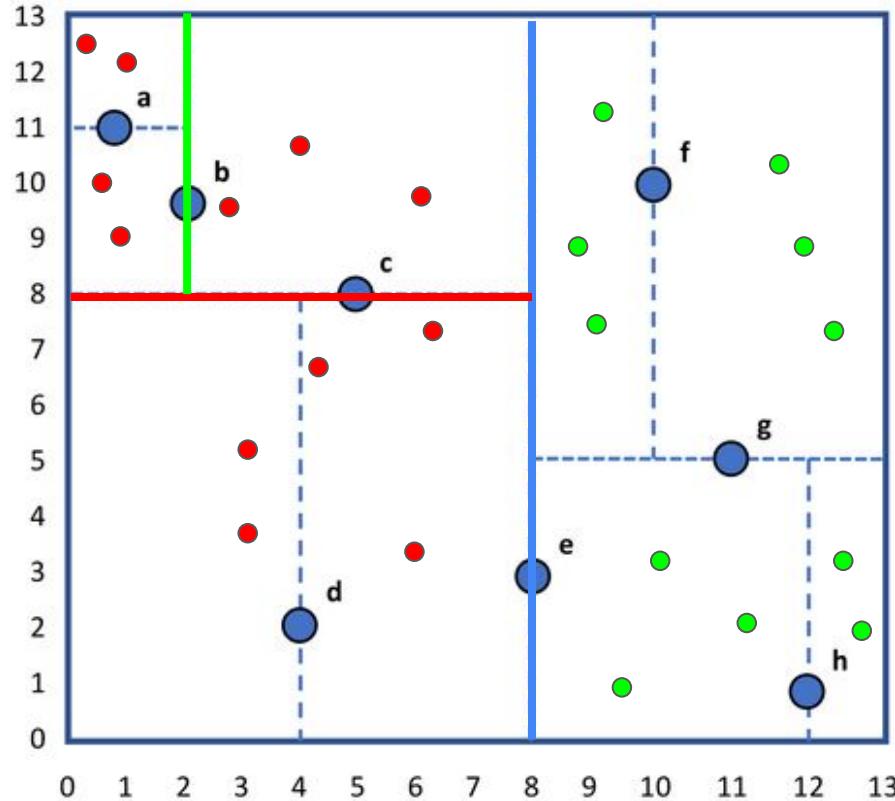
# Сопоставление точек: KD-tree



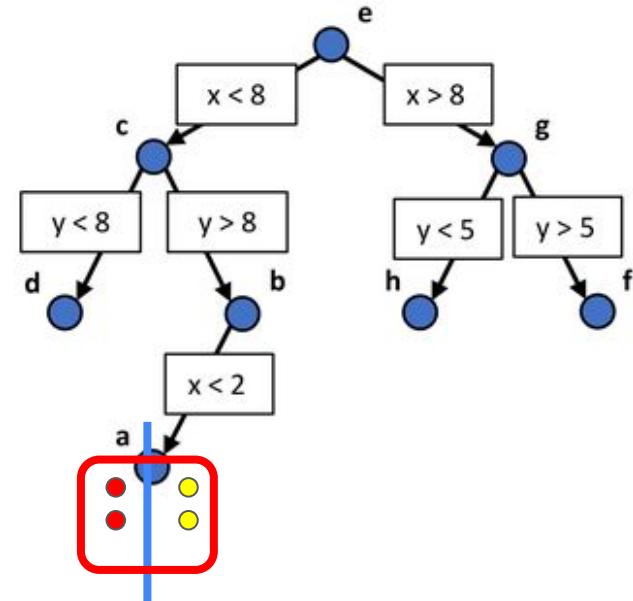
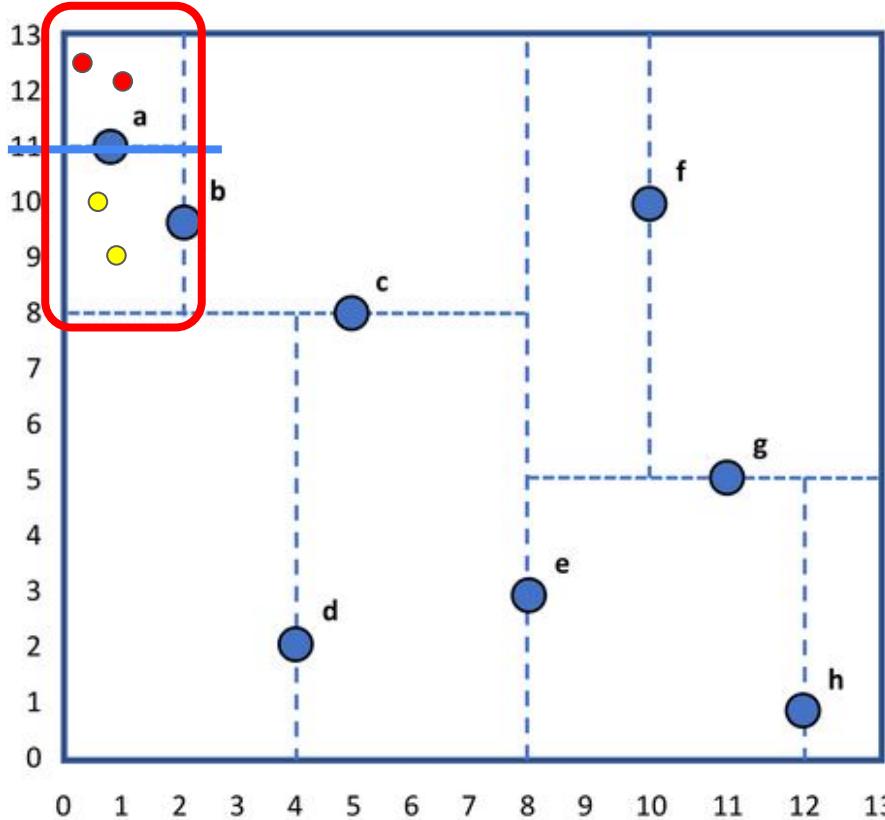
# Сопоставление точек: KD-tree



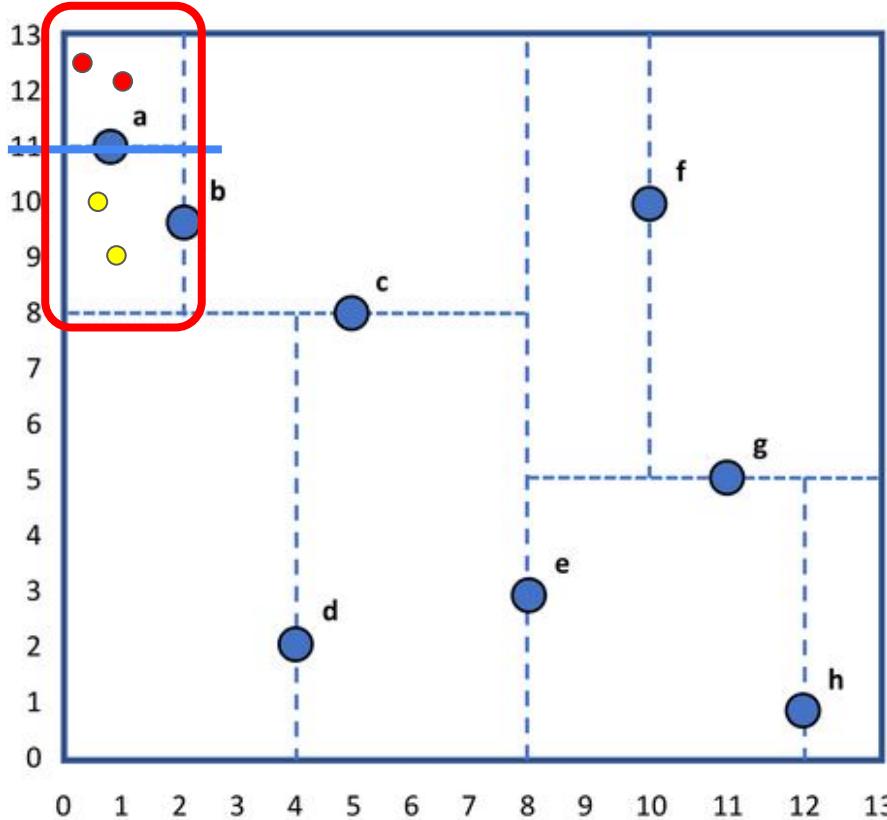
# Сопоставление точек: KD-tree



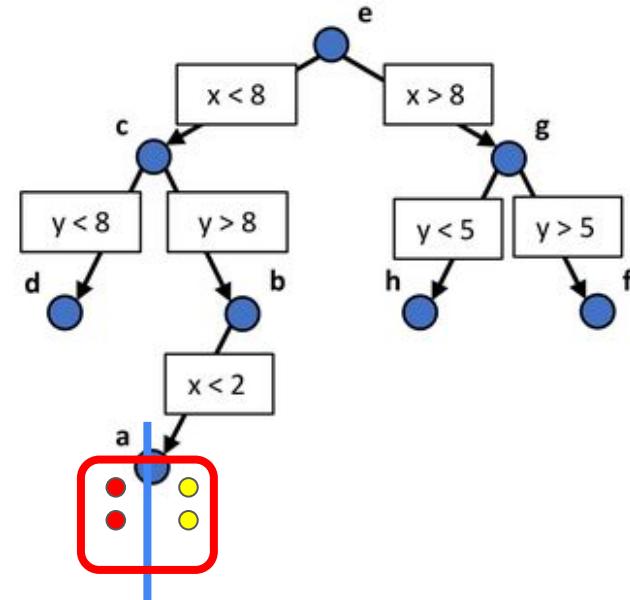
# Сопоставление точек: KD-tree



# Сопоставление точек: KD-tree

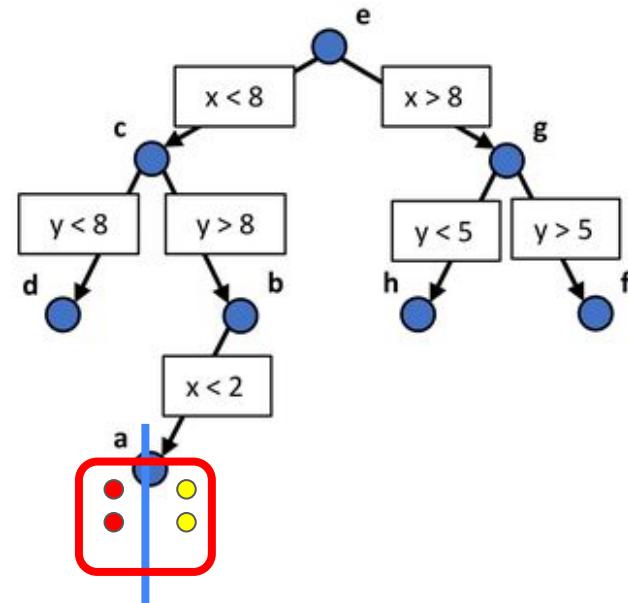
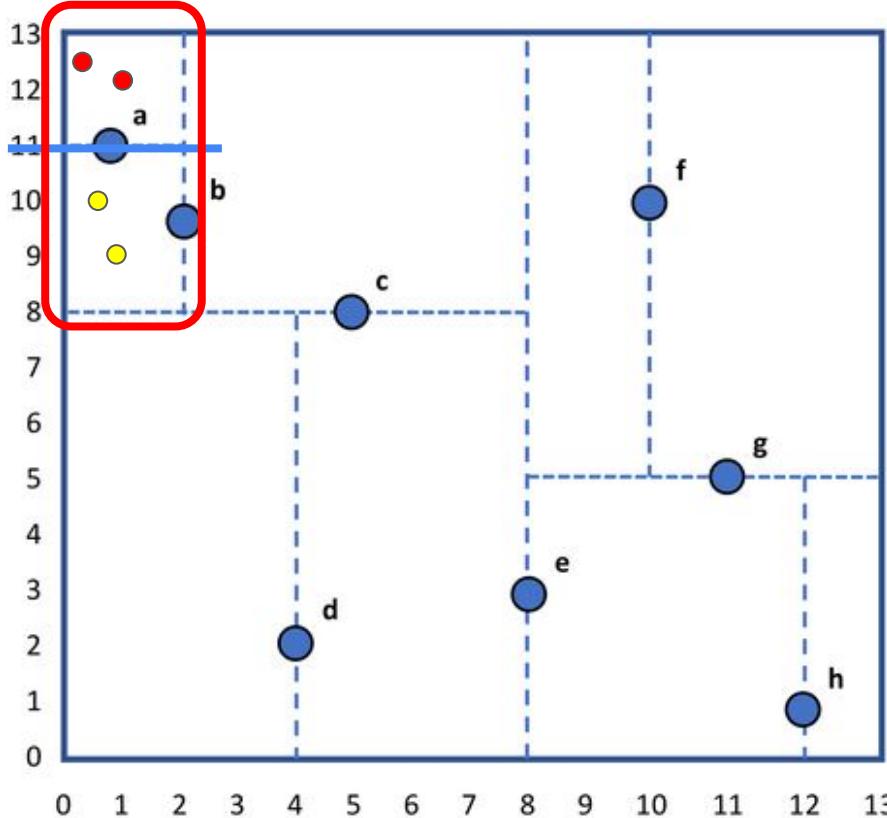


Как обработать запрос  
поиска по точке?

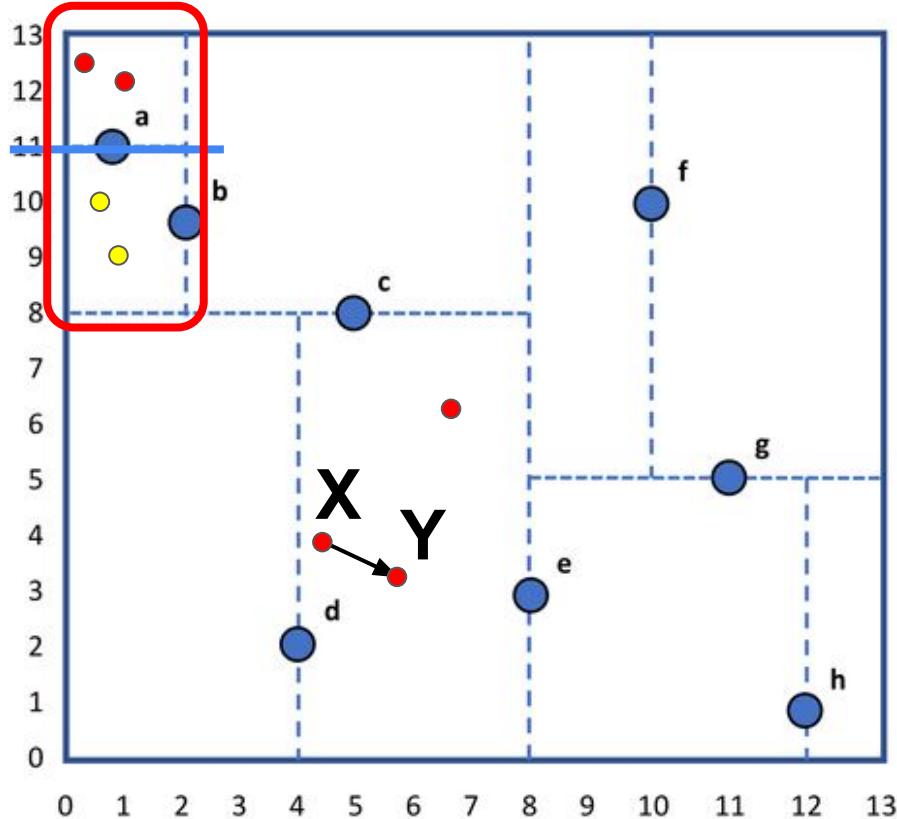


# Сопоставление точек: KD-tree

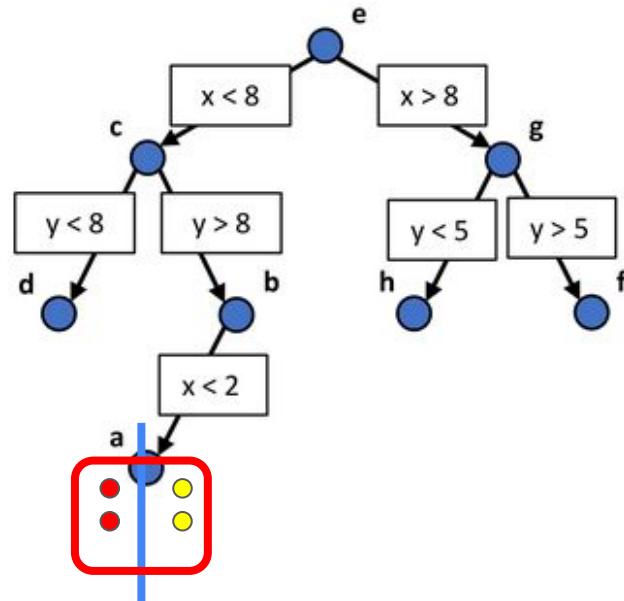
- 1) Запрос - точка X
- 2) Заходим в лист A содержащий X
- 3) Ищем ближайшего соседа Y из A



# Сопоставление точек: KD-tree

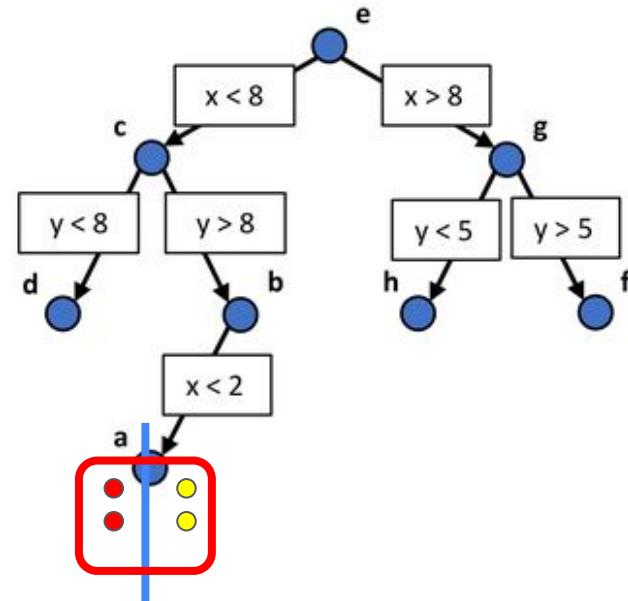
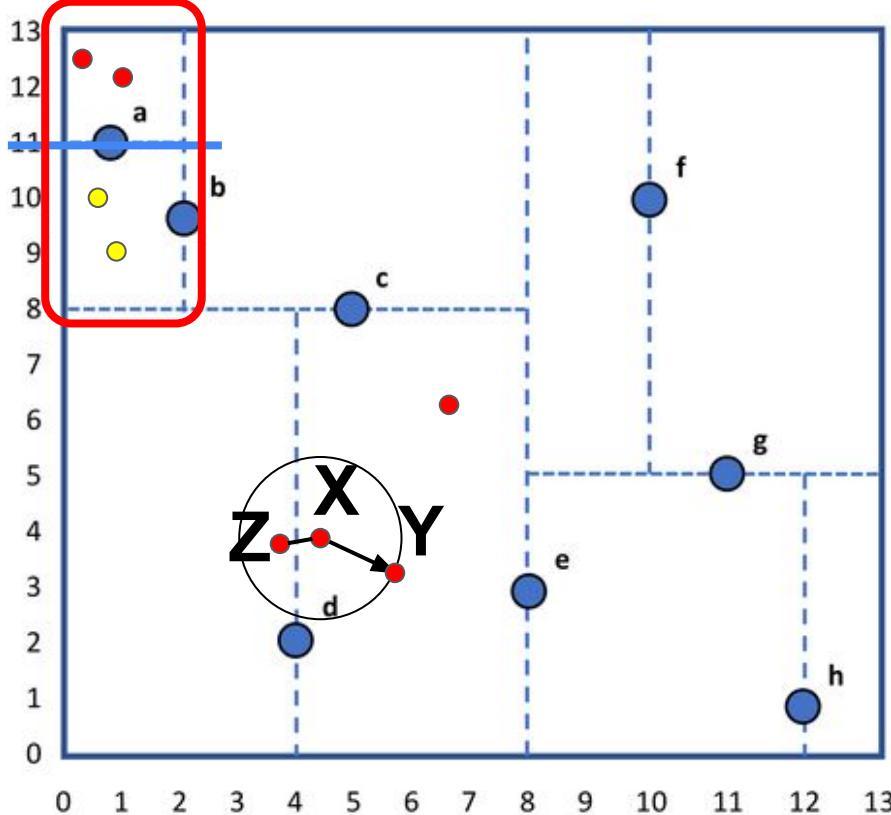


- 1) Запрос - точка **X**
  - 2) Заходим в лист **A** содержащий **X**
  - 3) Ищем ближайшего соседа **Y** из **A**
- Правильный ли это ответ?**

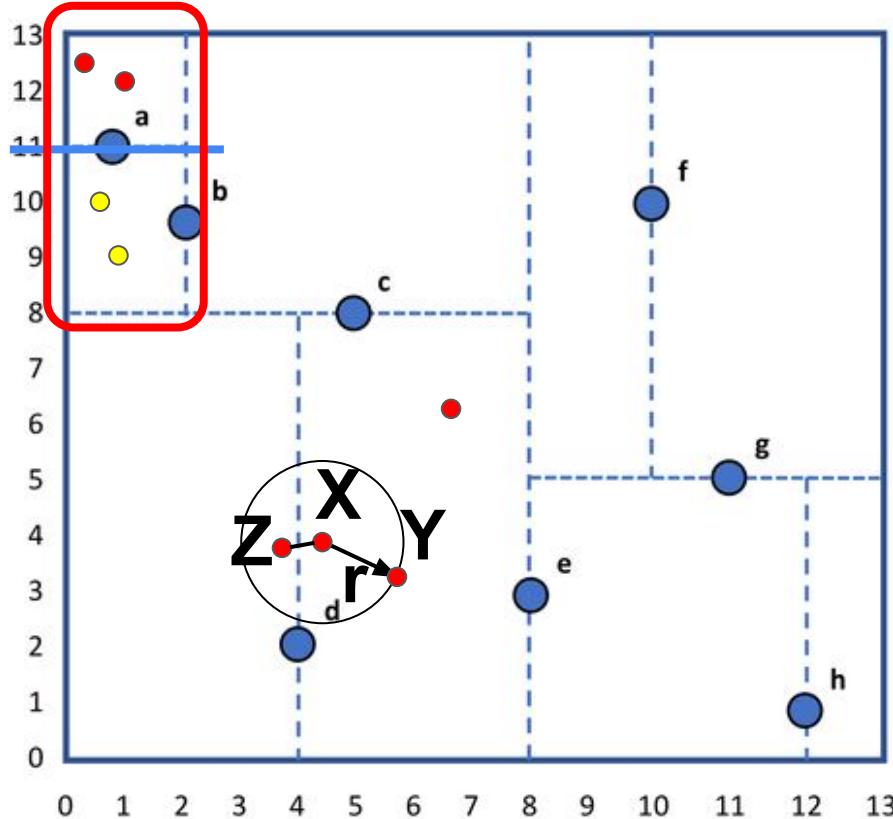


# Сопоставление точек: KD-tree

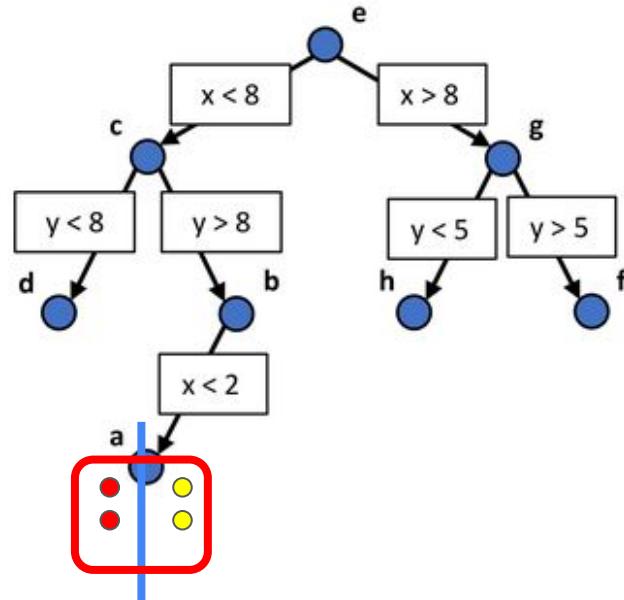
- 1) Запрос - точка X
- 2) Заходим в лист A содержащий X
- 3) Ищем ближайшего соседа Y из A  
**Правильный ли это ответ?**



# Сопоставление точек: KD-tree

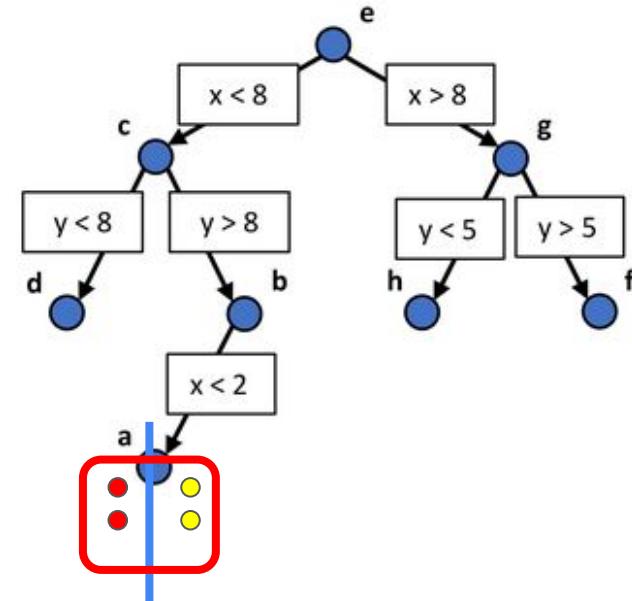
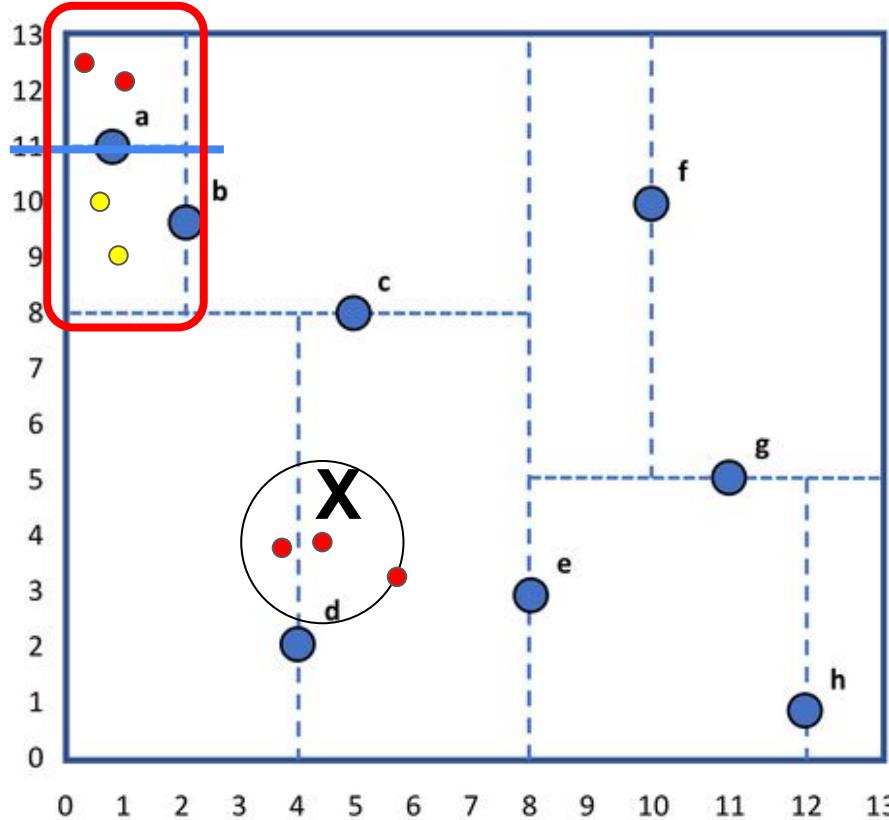


- 1) Запрос - точка **X**
- 2) Заходим в лист **A** содержащий **X**
- 3) Ищем ближайшего соседа **Y** из **A**
- 4) Радиус поиска  $r = \text{dist}(X, Y)$



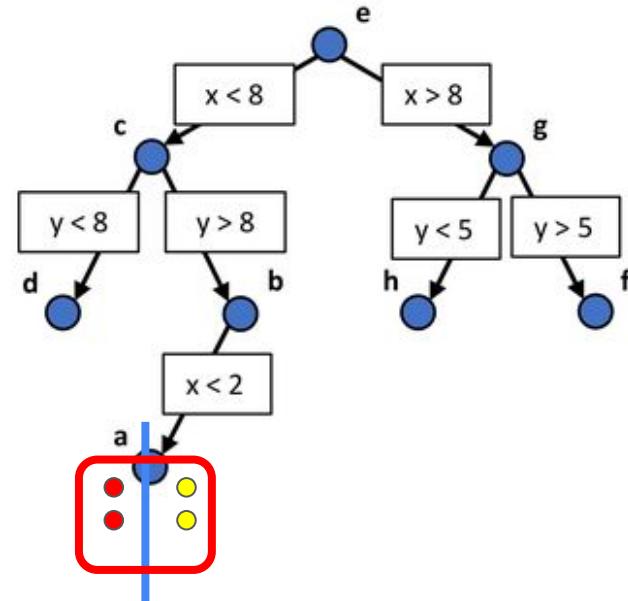
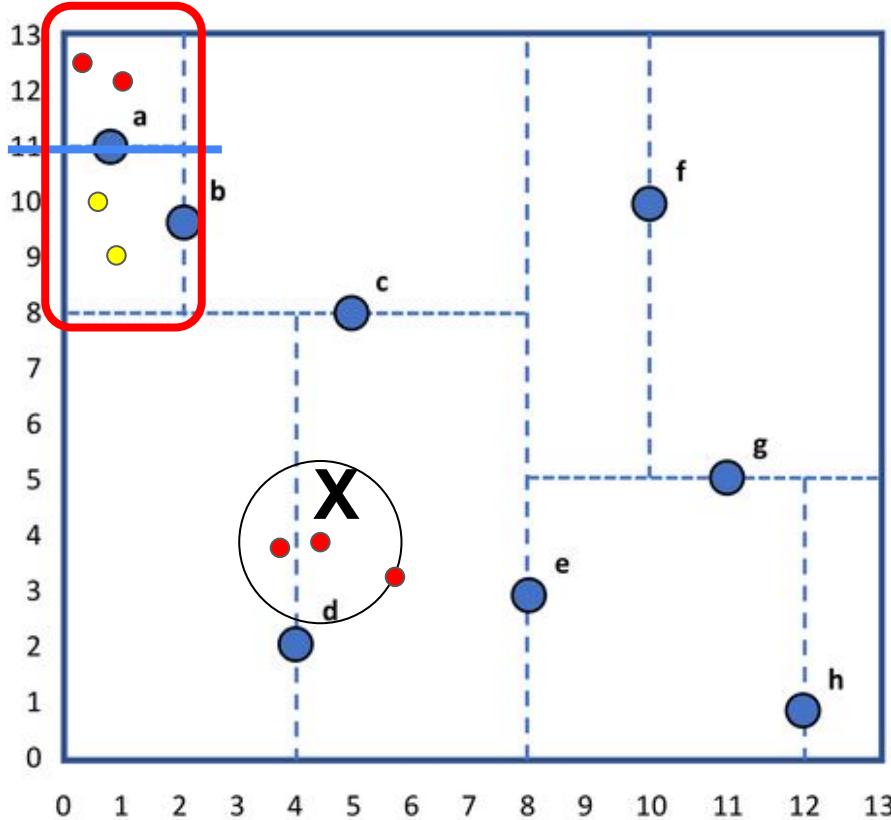
1) Запрос - точка  $X + r$  (радиус поиска)

## Сопоставление точек: KD-tree



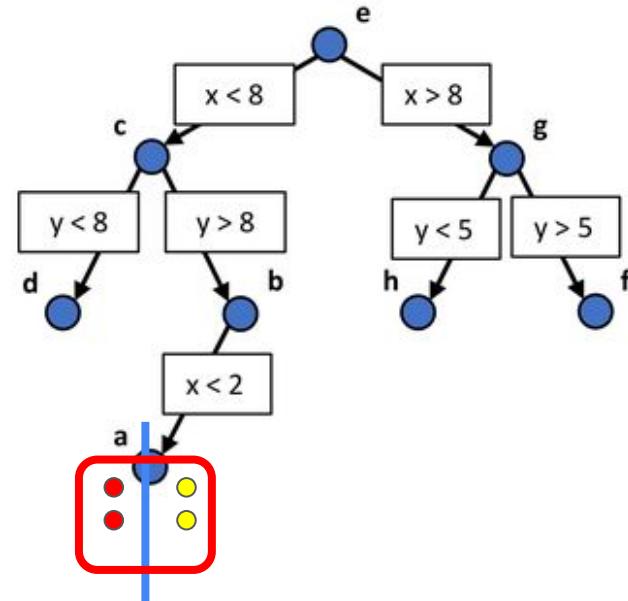
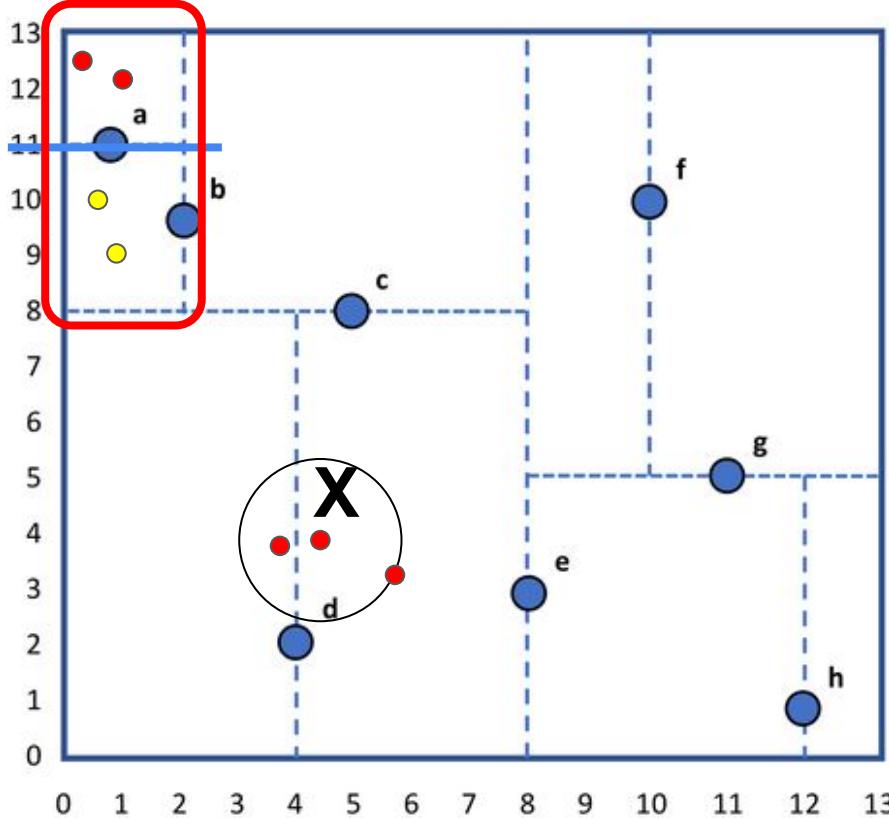
# Сопоставление точек: KD-tree

- 1) Запрос - точка  $X + r$  (радиус поиска)
- 2) Заходим во все релевантные узлы



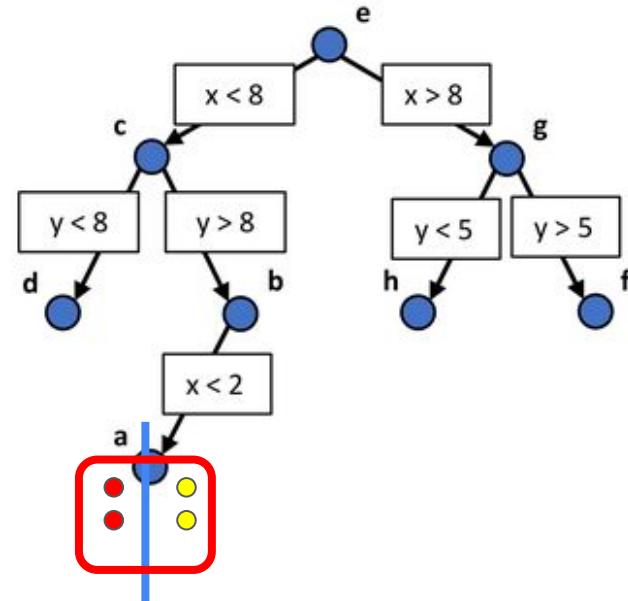
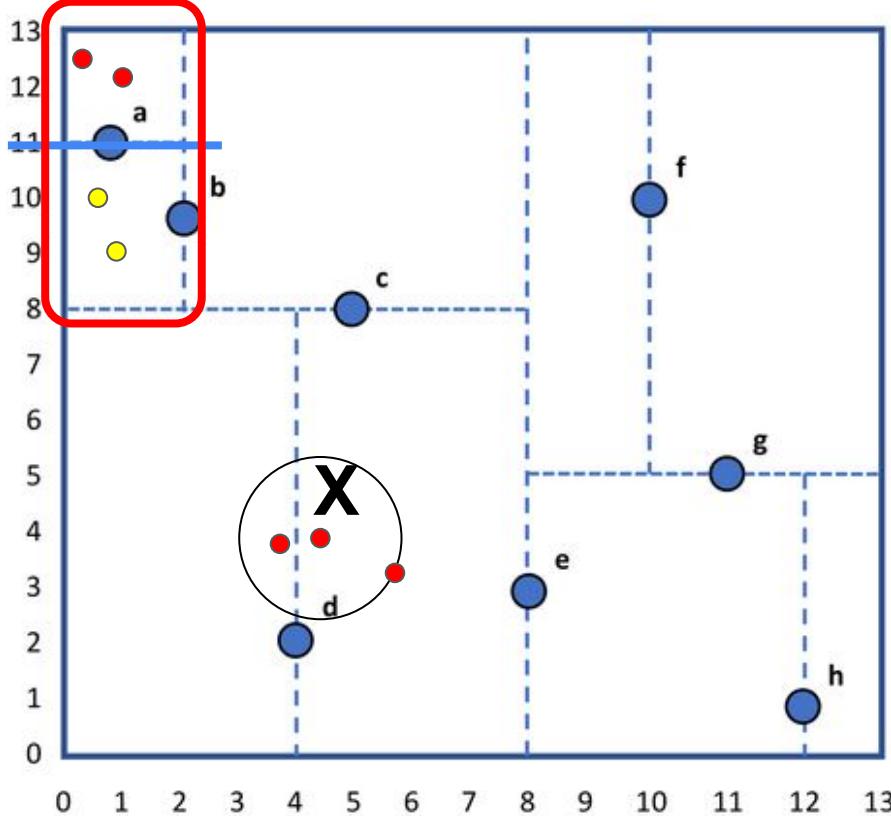
# Сопоставление точек: KD-tree

- 1) Запрос - точка  $X + r$  (радиус поиска)
- 2) Заходим во все релевантные узлы
- 3) Обновляя по мере шагов  $R$

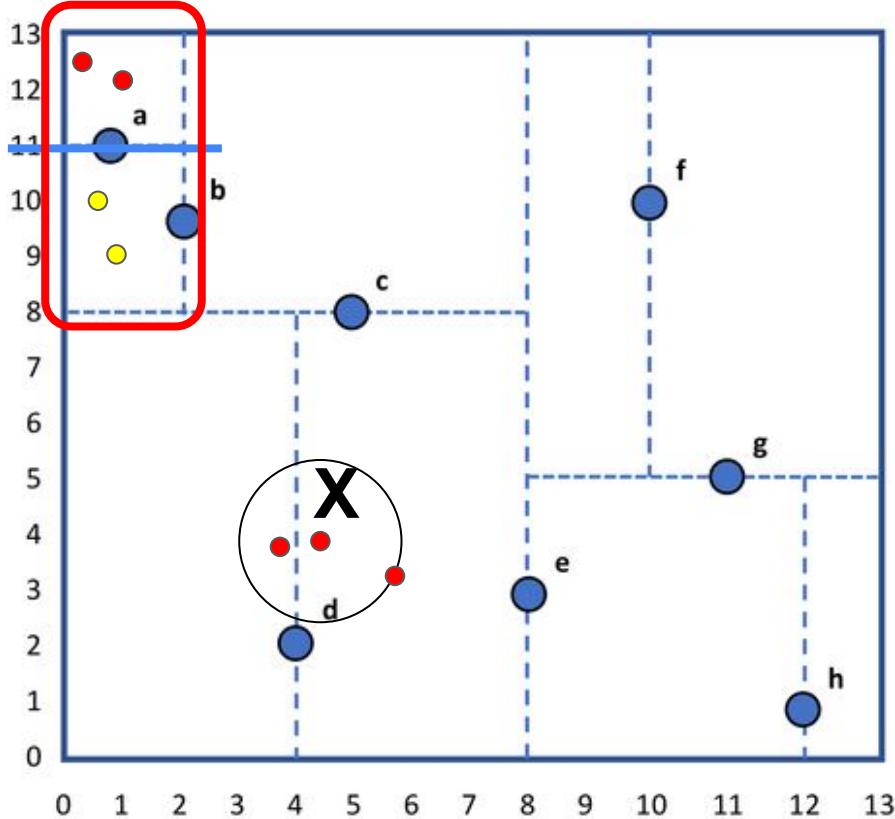


# Сопоставление точек: KD-tree

- 1) Запрос - точка  $X + r$  (радиус поиска)
  - 2) Заходим во все релевантные узлы
  - 3) Обновляя по мере шагов  $R$
- Зачем?**

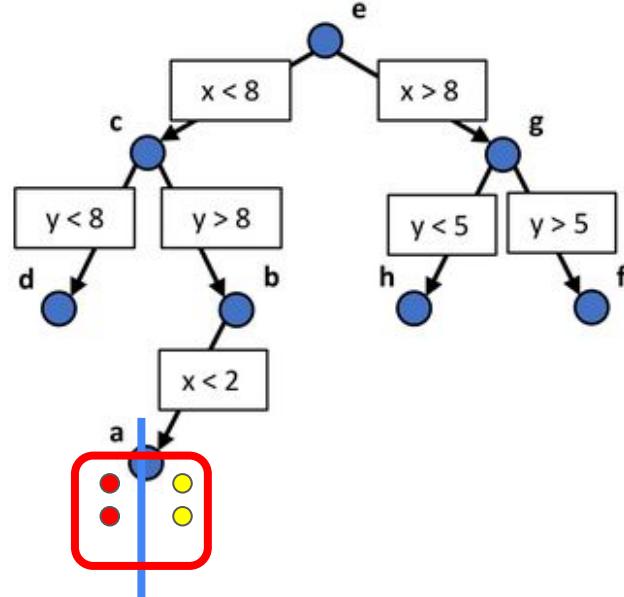


# Сопоставление точек: KD-tree

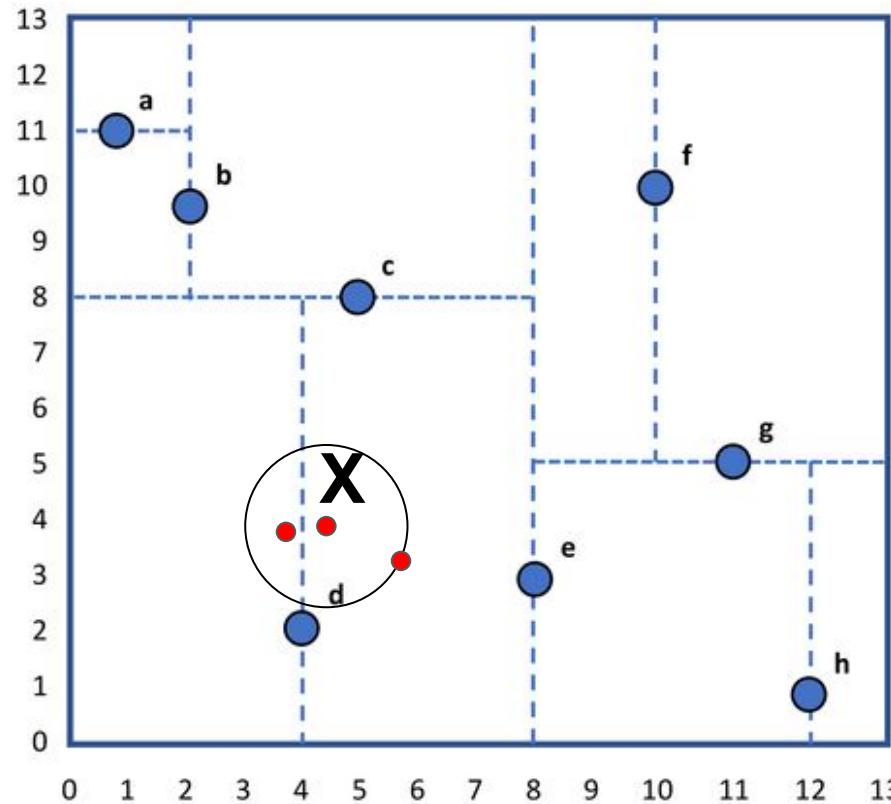


- 1) Запрос - точка  $X + r$  (радиус поиска)
- 2) Заходим во все релевантные узлы
- 3) Обновляя по мере шагов  $R$  ради ускорения

Но у нас 128-мерное пространство!  
Тормозим из-за проклятия размерности!

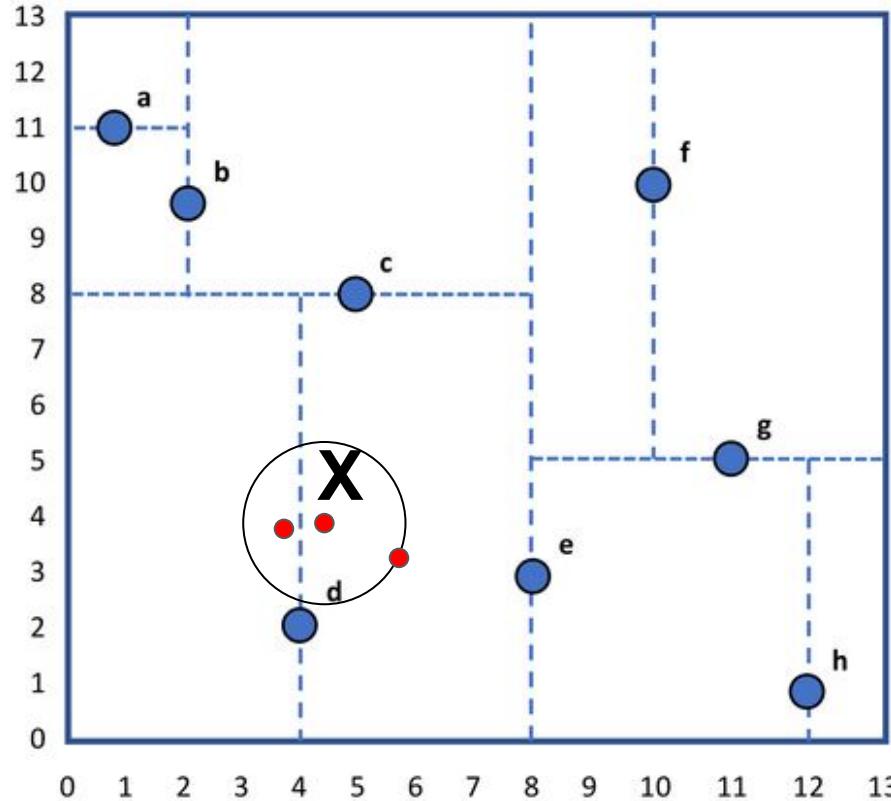


# KD-tree и проклятие размерности (интуиция 1)

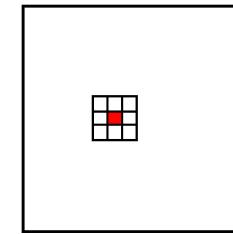


1) Сколько листьев посетим если 2D?

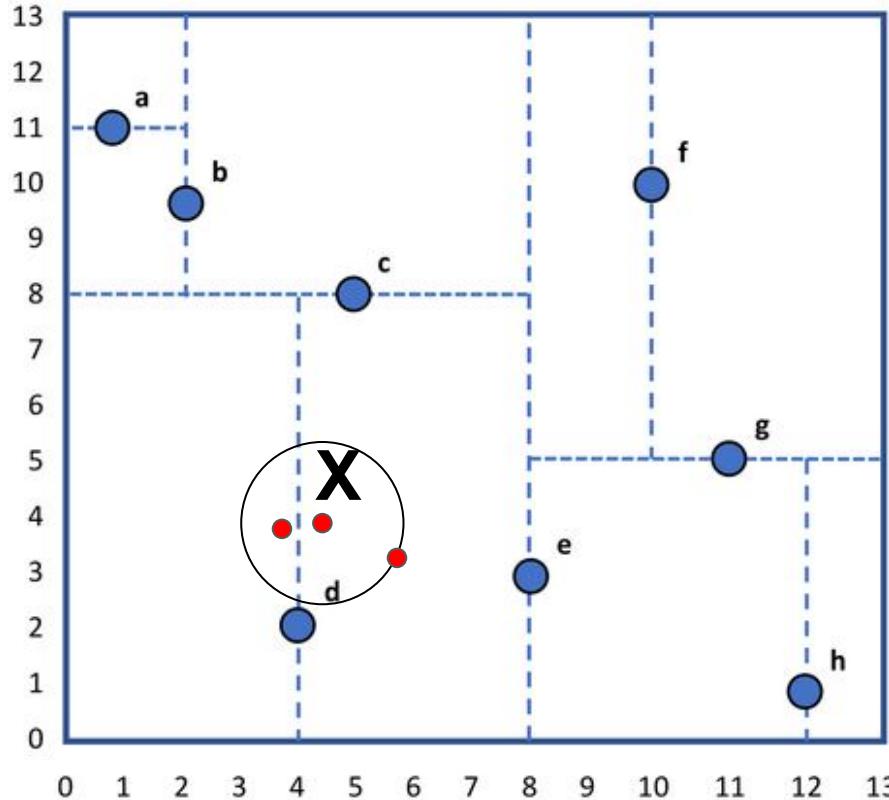
# KD-tree и проклятие размерности (интуиция 1)



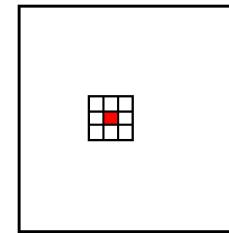
- 1) В **2D** случае на регулярной решетке посетим не более 9 листьев:



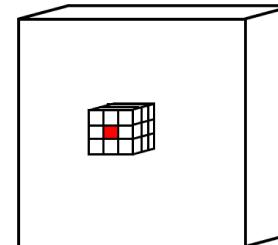
# KD-tree и проклятие размерности (интуиция 1)



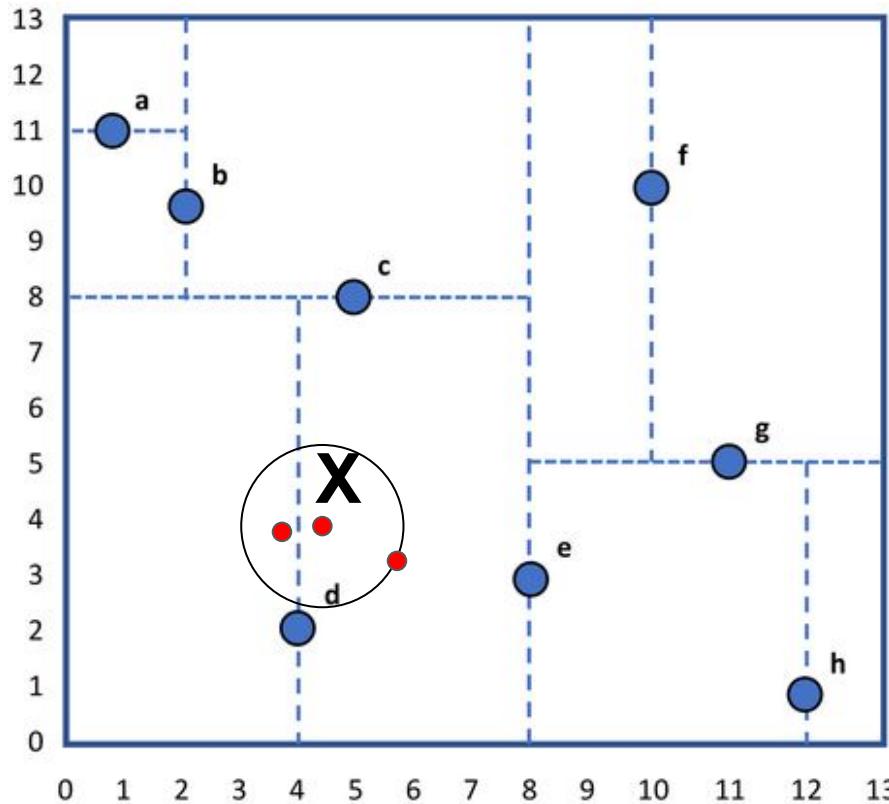
1) В **2D** случае на регулярной решетке посетим не более 9 листьев:



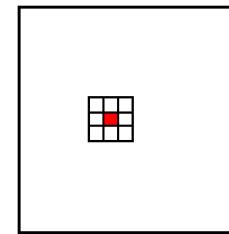
2) В **3D** случае - 27 листьев:



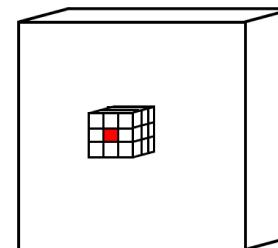
# KD-tree и проклятие размерности (интуиция 1)



1) В **2D** случае на регулярной решетке посетим не более 9 листьев:

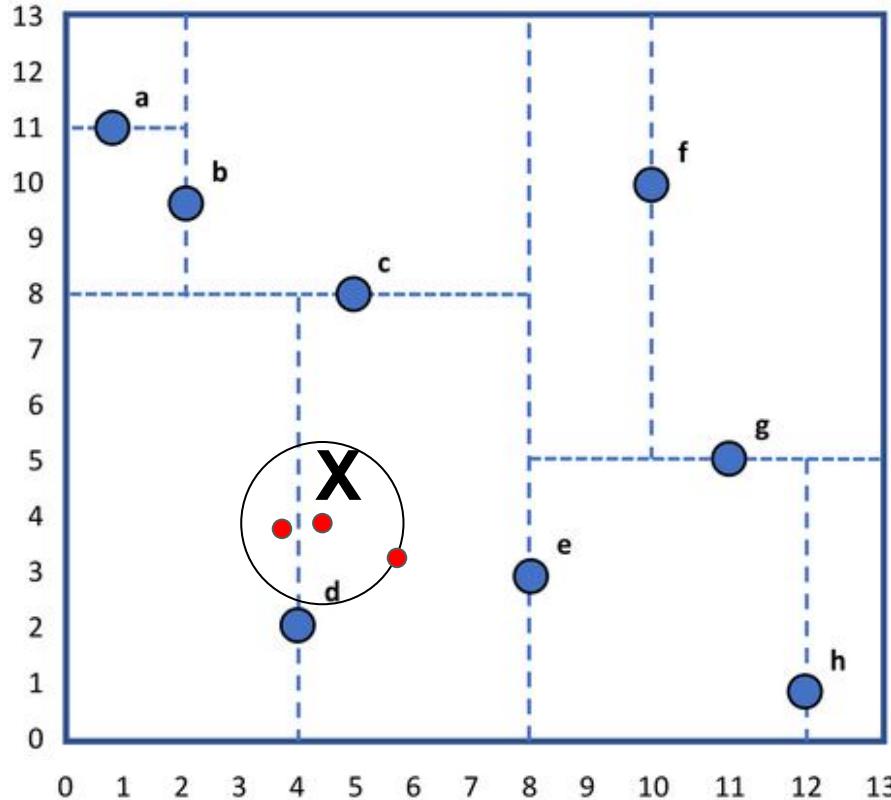


2) В **3D** случае - 27 листьев:

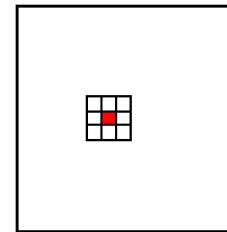


3) А в **128-мерном** случае?

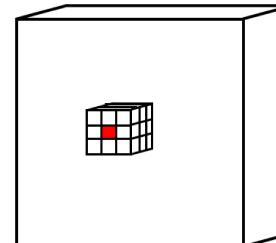
# KD-tree и проклятие размерности (интуиция 1)



1) В **2D** случае на регулярной решетке посетим не более 9 листьев:



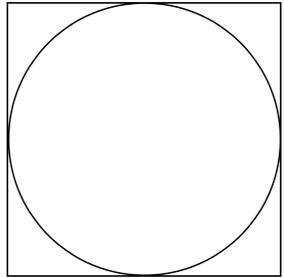
2) В **3D** случае - 27 листьев:



3) В **128D** случае -  $3^{127}$

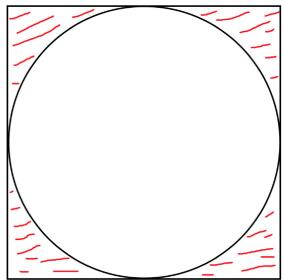
# KD-tree и проклятие размерности (интуиция 2)

1) 2D - какая вероятность случайной точке в квадрате оказаться внутри круга?



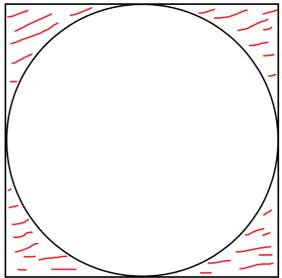
# KD-tree и проклятие размерности (интуиция 2)

- 1) 2D - какая вероятность случайной точке в квадрате оказаться внутри круга?

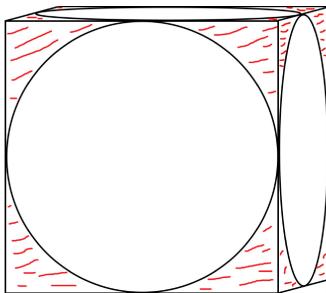


# KD-tree и проклятие размерности (интуиция 2)

1) **2D** - какая вероятность случайной точке в квадрате оказаться внутри круга?



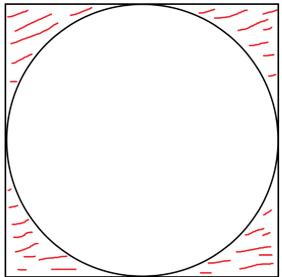
2) **3D** - какая вероятность если речь уже про шар в кубике?



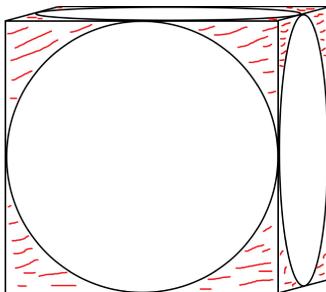
Более строгий анализ проблемы

# KD-tree и проклятие размерности (интуиция 2)

1) **2D** - какая вероятность случайной точке в квадрате оказаться внутри круга?



2) **3D** - какая вероятность если речь уже про шар в кубике?

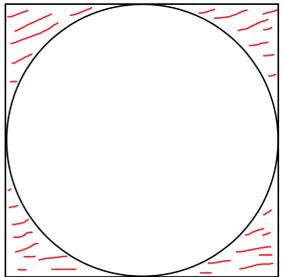


3) **128D** - какая вероятность?

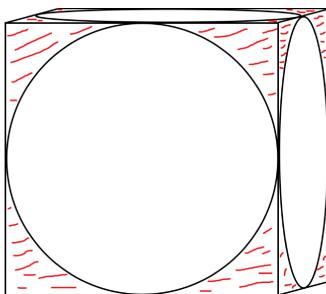
Более строгий анализ проблемы

# KD-tree и проклятие размерности (интуиция 2)

- 1) **2D** - какая вероятность случайной точке в квадрате оказаться внутри круга?



- 2) **3D** - какая вероятность если речь уже про шар в кубике?

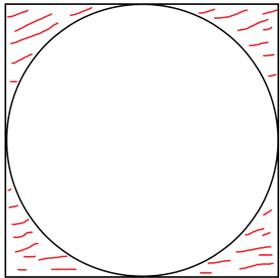


- 3) **128D** - какая вероятность? Почти все пространство сосредоточено в углах куба!  
**При чем здесь KD-tree?**

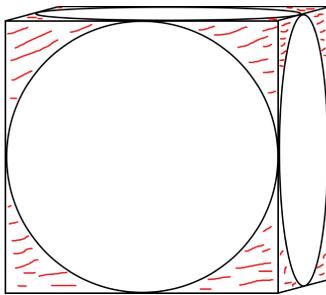
Более строгий анализ проблемы

# KD-tree и проклятие размерности (интуиция 2)

- 1) **2D** - какая вероятность случайной точке в квадрате оказаться внутри круга?



- 2) **3D** - какая вероятность если речь уже про шар в кубике?

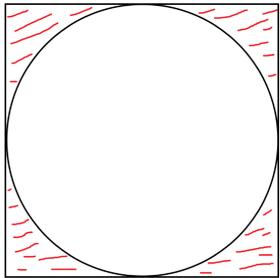


- 3) **128D** - какая вероятность? Почти все пространство сосредоточено в углах куба!  
Потенциальный **ответ** - пространство-шар в радиусе вокруг нас.  
**А какое количество листьев мы обходим? Какой объем пространства мы учитываем?**

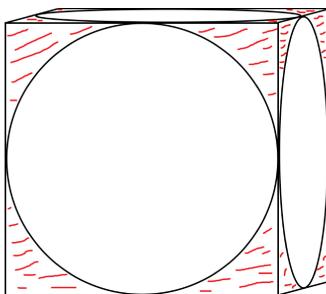
Более строгий анализ проблемы

# KD-tree и проклятие размерности (интуиция 2)

- 1) **2D** - какая вероятность случайной точке в квадрате оказаться внутри круга?



- 2) **3D** - какая вероятность если речь уже про шар в кубике?



- 3) **128D** - какая вероятность? Почти все пространство сосредоточено в углах куба!  
Потенциальный **ответ** - пространство-шар в радиусе вокруг нас.  
Но приходится учитывать все листья пересекающие пространство-куб вокруг нас!

Более строгий анализ проблемы

# KD-tree и проклятие размерности (интуиция 3)

- 1) 1D - какое мат. ожидание расстояния до ближайшего соседа на 10 точках?

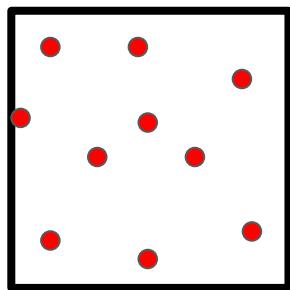


# KD-tree и проклятие размерности (интуиция 3)

1) 1D - какое мат. ожидание расстояния до ближайшего соседа на 10 точках?



2) 2D - какое мат. ожидание расстояния до ближайшего соседа на 10 точках?

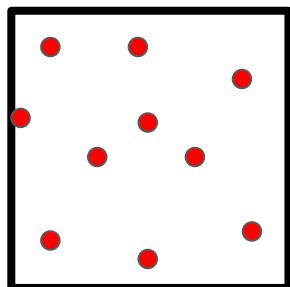


# KD-tree и проклятие размерности (интуиция 3)

- 1) 1D - какое мат. ожидание расстояния до ближайшего соседа на 10 точках?



- 2) 2D - какое мат. ожидание расстояния до ближайшего соседа на 10 точках?



- 3) 128D - мат. ожидание совсем большое.  
К чему это приведет в случае KD-tree?

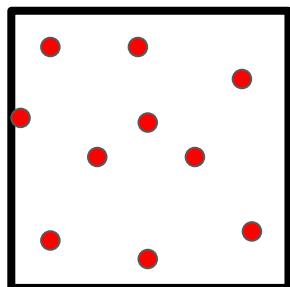
Более строгий анализ проблемы

# KD-tree и проклятие размерности (интуиция 3)

1) 1D - какое мат. ожидание расстояния до ближайшего соседа на 10 точках?



2) 2D - какое мат. ожидание расстояния до ближайшего соседа на 10 точках?



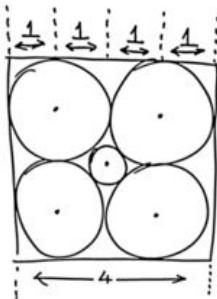
3) 128D - мат. ожидание совсем большое.

Значит радиус отсечения большой и мы посетим много узлов дерева.

Более строгий анализ проблемы

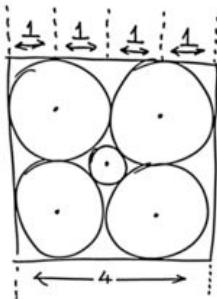
# Высокие размерности - это непонятно и можно уколоться

1) 2D - какого радиуса влезает окружность в центре?  $\sqrt{2} - 1 = \sim 0.4142$  (для ND:  $\sqrt{n} - 1$ )



# Высокие размерности - это непонятно и можно уколоться

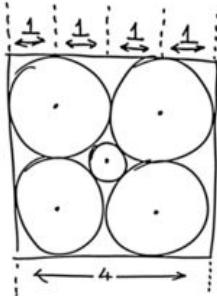
- 1) **2D** - какого радиуса влезает окружность в центре?  $\sqrt{2} - 1 = \sim 0.4142$  (для **ND**:  $\sqrt{n} - 1$ )



- 2) **3D** - какого радиуса влезает сфера в центре зажатая 8 сферами?  $\sqrt{3} - 1 = \sim 0.732$

# Высокие размерности - это непонятно и можно уколоться

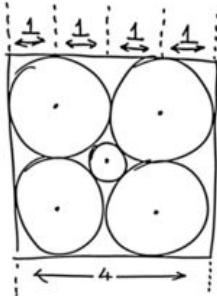
- 1) **2D** - какого радиуса влезает окружность в центре?  $\sqrt{2} - 1 = \sim 0.4142$  (для **ND**:  $\sqrt{n} - 1$ )



- 2) **3D** - какого радиуса влезает сфера в центре зажатая 8 сферами?  $\sqrt{3} - 1 = \sim 0.732$
- 3) **9D** - какой радиус сферы? Есть ли в этом радиусе что-то любопытное?

# Высокие размерности - это непонятно и можно уколоться

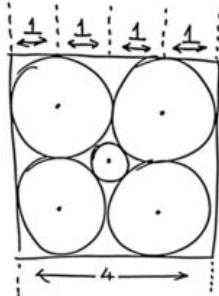
- 1) **2D** - какого радиуса влезает окружность в центре?  $\sqrt{2} - 1 = \sim 0.4142$  (для **ND**:  $\sqrt{n} - 1$ )



- 2) **3D** - какого радиуса влезает сфера в центре зажатая 8 сферами?  $\sqrt{3} - 1 = \sim 0.732$
- 3) **9D** - радиус гипер-сферы равен  $\sqrt{9} - 1 = 2$ , т.е. диаметр равен стороне гипер-куба!  
Т.е. сфера касается сторон куба!

# Высокие размерности - это непонятно и можно уколоться

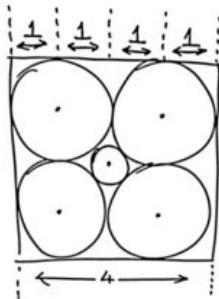
- 1) **2D** - какого радиуса влезает окружность в центре?  $\sqrt{2} - 1 = \sim 0.4142$  (для **ND**:  $\sqrt{n} - 1$ )



- 2) **3D** - какого радиуса влезает сфера в центре зажатая 8 сферами?  $\sqrt{3} - 1 = \sim 0.732$
- 3) **9D** - радиус гипер-сферы равен  $\sqrt{9} - 1 = 2$ , т.е. диаметр равен стороне гипер-куба! Т.е. сфера касается сторон куба!
- 4) **10D** - **какой радиус сферы? Есть ли в этом радиусе что-то любопытное?**

# Высокие размерности - это непонятно и можно уколоться

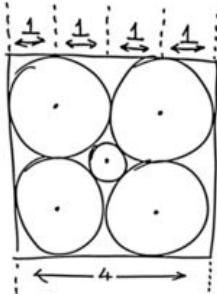
- 1) **2D** - какого радиуса влезает окружность в центре?  $\sqrt{2} - 1 = \sim 0.4142$  (для **ND**:  $\sqrt{n} - 1$ )



- 2) **3D** - какого радиуса влезает сфера в центре зажатая 8 сферами?  $\sqrt{3} - 1 = \sim 0.732$
- 3) **9D** - радиус гипер-сферы равен  $\sqrt{9} - 1 = 2$ , т.е. диаметр равен стороне гипер-куба!  
Т.е. сфера касается сторон куба!
- 4) **10D** - радиус гипер-сферы больше 2!  
Она просочилась между остальных сфер и торчит наружу!

# Высокие размерности - это непонятно и можно уколоться

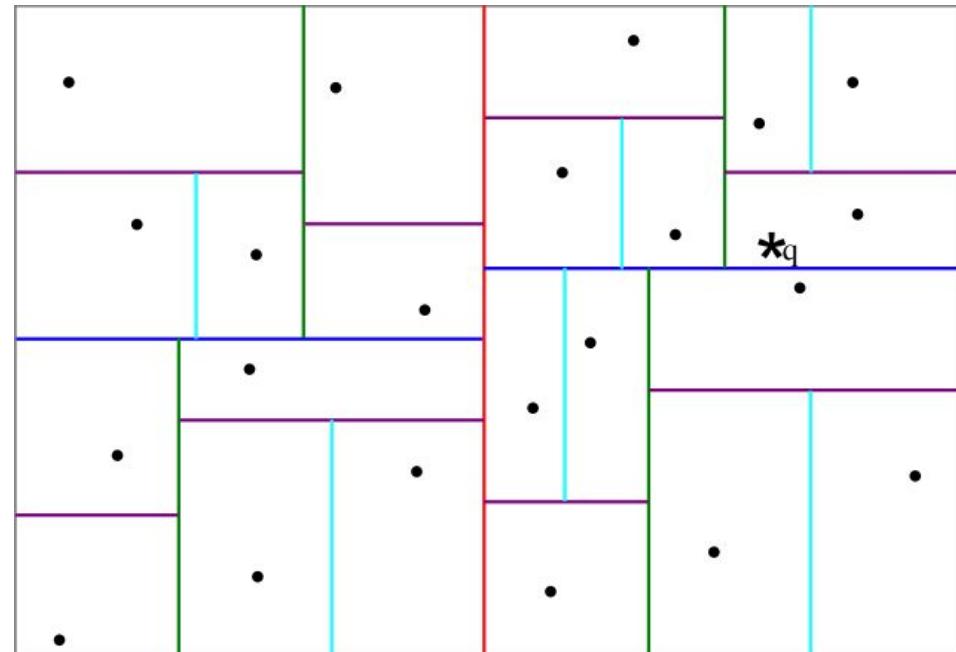
- 1) **2D** - какого радиуса влезает окружность в центре?  $\sqrt{2} - 1 = \sim 0.4142$  (для **ND**:  $\sqrt{n} - 1$ )



- 2) **3D** - какого радиуса влезает сфера в центре зажатая 8 сферами?  $\sqrt{3} - 1 = \sim 0.732$
- 3) **9D** - радиус гипер-сферы равен  $\sqrt{9} - 1 = 2$ , т.е. диаметр равен стороне гипер-куба!  
Т.е. сфера касается сторон куба!
- 4) **10D** - радиус гипер-сферы больше 2!  
Она просочилась между остальных сфер и торчит наружу!  
Т.е. сфера это ежик, но сфера симметрична, значит каждая точка на ее поверхности - иголка. И большая часть объема сферы расположена в этих иглах - вдоль поверхности.

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ

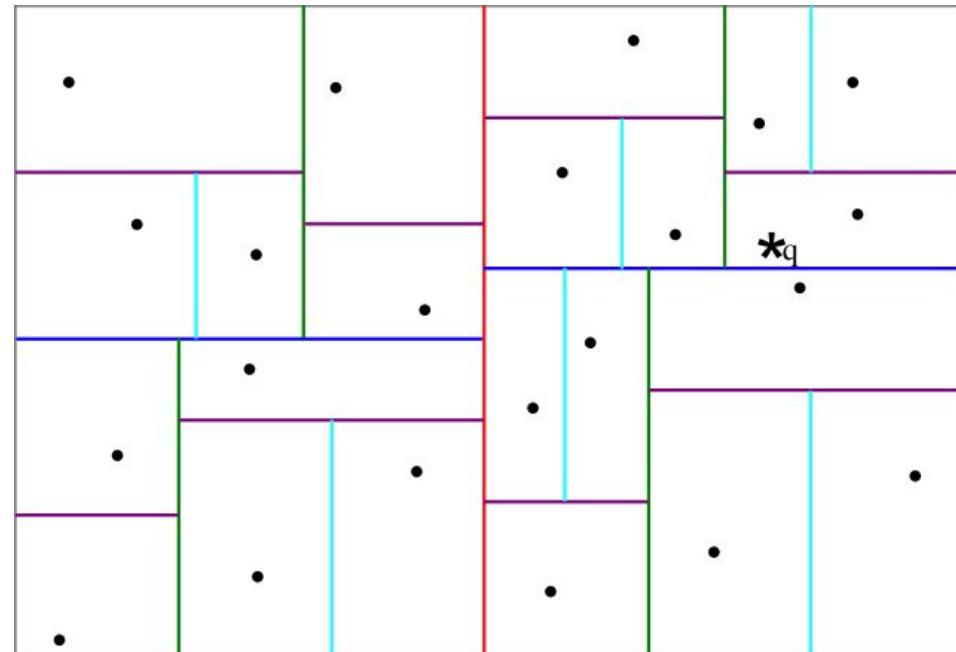


[Scalable Nearest Neighbor Algorithms for High Dimensional Data](#)

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ

Чем плохо?

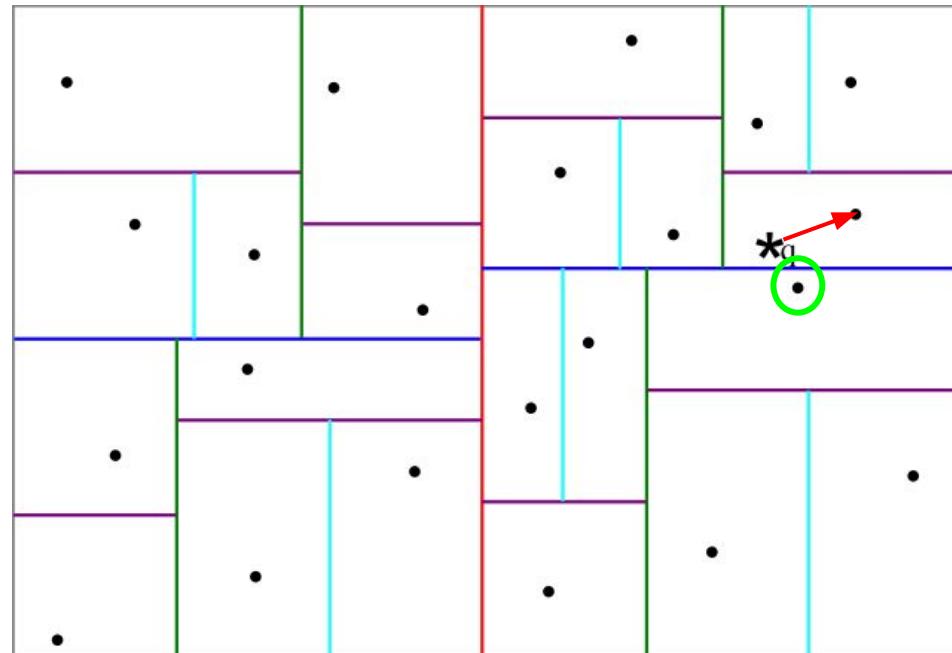


Scalable Nearest Neighbor Algorithms for High Dimensional Data

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ

Как исправить?

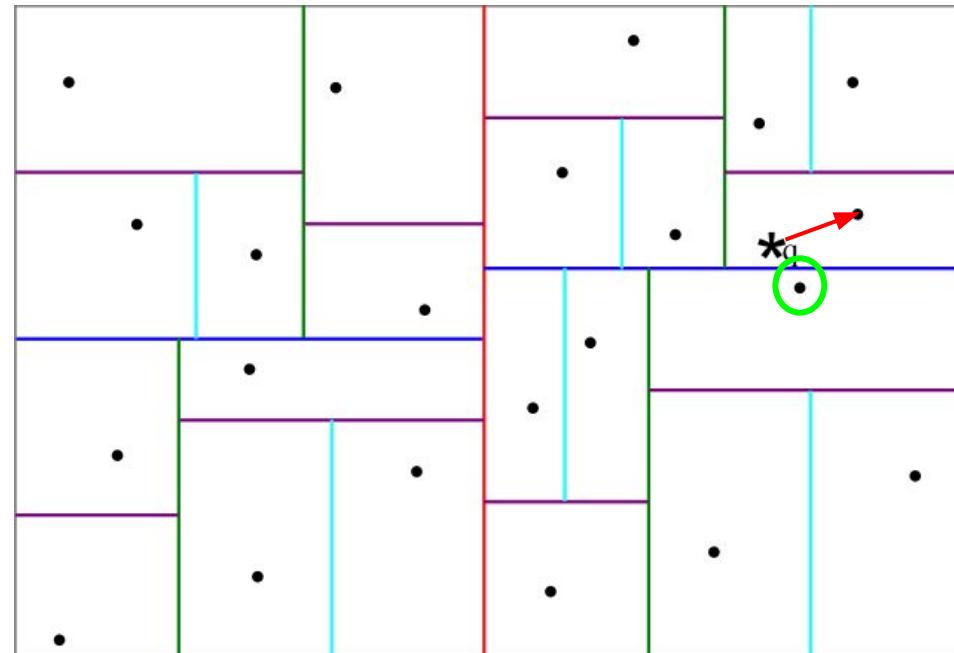


Правильный ответ в другом листе

Scalable Nearest Neighbor Algorithms for High Dimensional Data

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ
- 2) Посещаем остальные листья чтобы уточнить ответ.  
С ускорением через поддержание расстояния до “текущего ответа”.



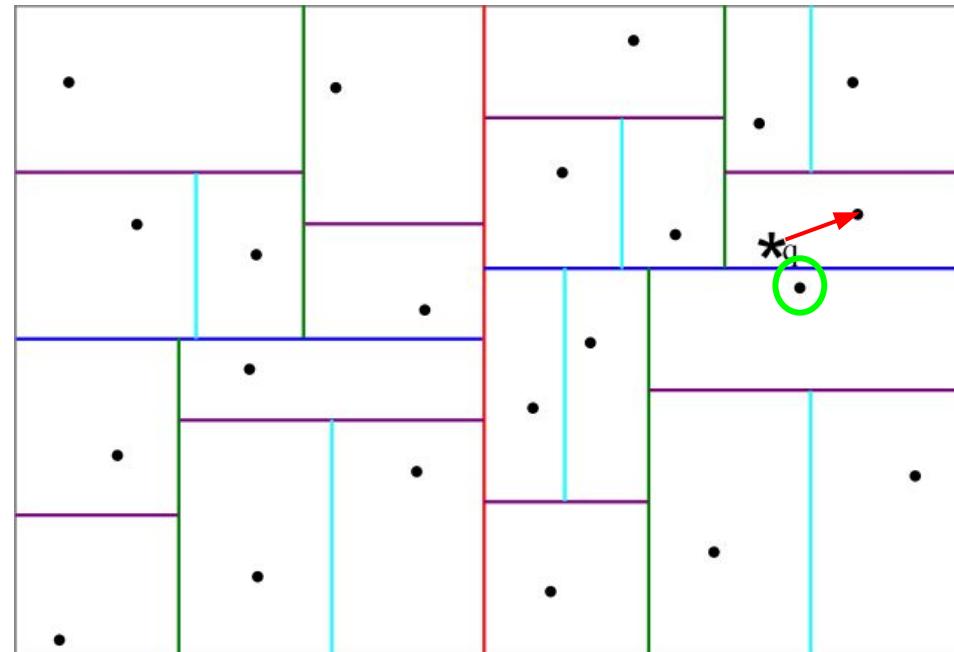
Правильный ответ в другом листе

Scalable Nearest Neighbor Algorithms for High Dimensional Data

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ
- 2) Посещаем остальные листья чтобы уточнить ответ.  
С ускорением через поддержание расстояния до “текущего ответа”.

Как лучше обходить узлы?

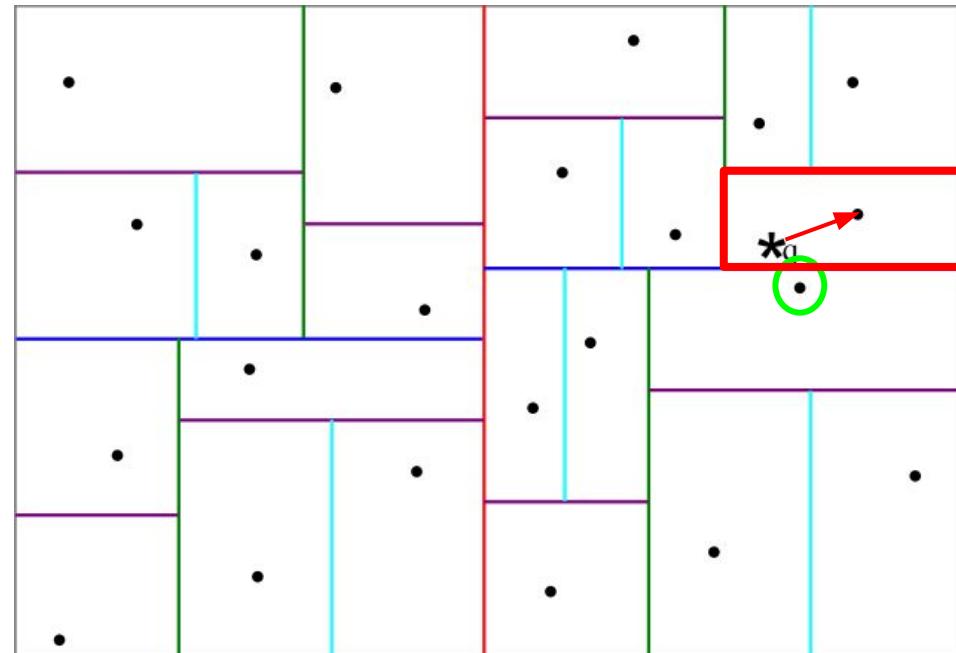


Правильный ответ в другом листе

Scalable Nearest Neighbor Algorithms for High Dimensional Data

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ
- 2) Посещаем остальные листья чтобы уточнить ответ.  
С ускорением через поддержание расстояния до “текущего ответа”.

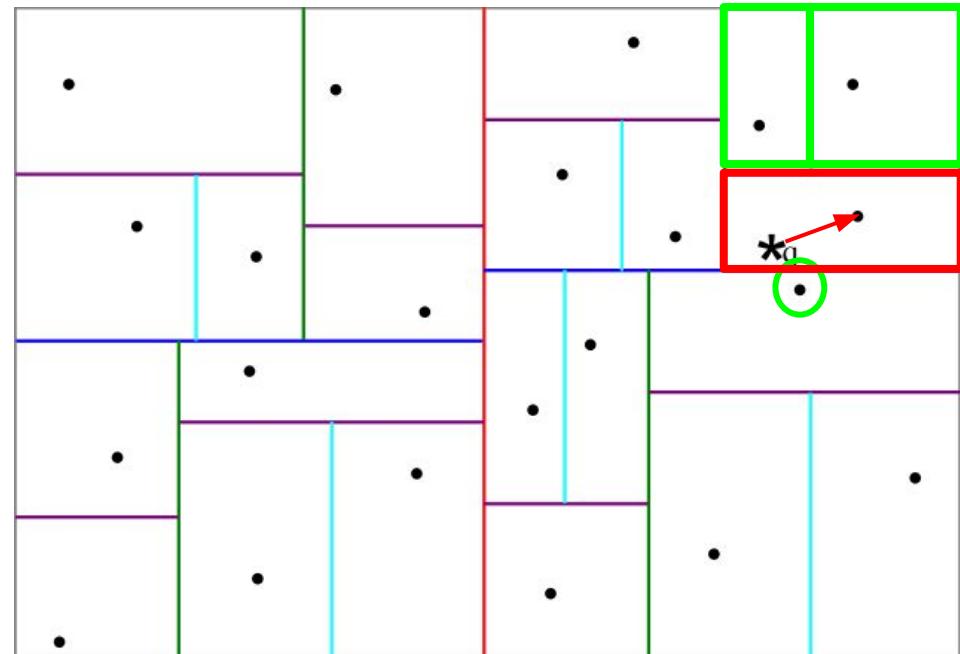


Правильный ответ в другом листе

Scalable Nearest Neighbor Algorithms for High Dimensional Data

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ
- 2) Посещаем остальные листья чтобы уточнить ответ.  
С ускорением через поддержание расстояния до “текущего ответа”.

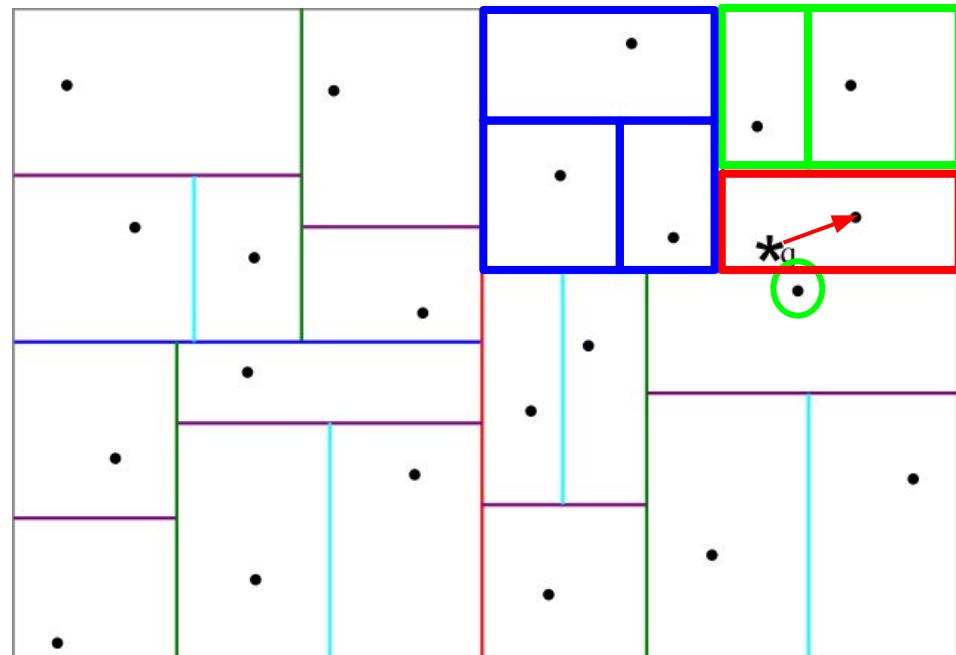


Правильный ответ в другом листе

Scalable Nearest Neighbor Algorithms for High Dimensional Data

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ
- 2) Посещаем остальные листья чтобы уточнить ответ.  
С ускорением через поддержание расстояния до “текущего ответа”.

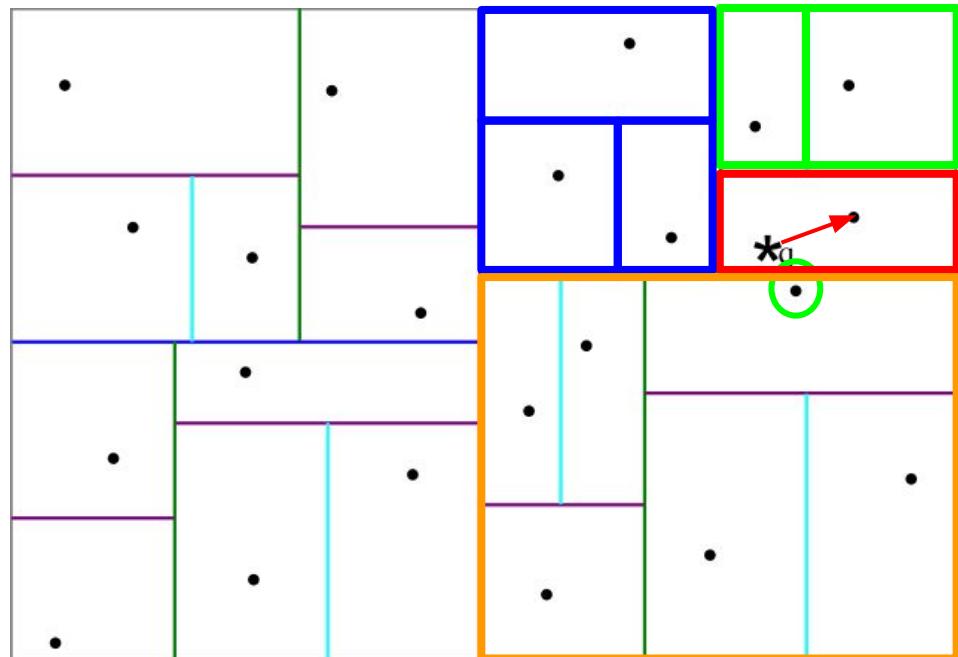


Правильный ответ в другом листе

Scalable Nearest Neighbor Algorithms for High Dimensional Data

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ
- 2) Посещаем остальные листья чтобы уточнить ответ.  
С ускорением через поддержание расстояния до “текущего ответа”.

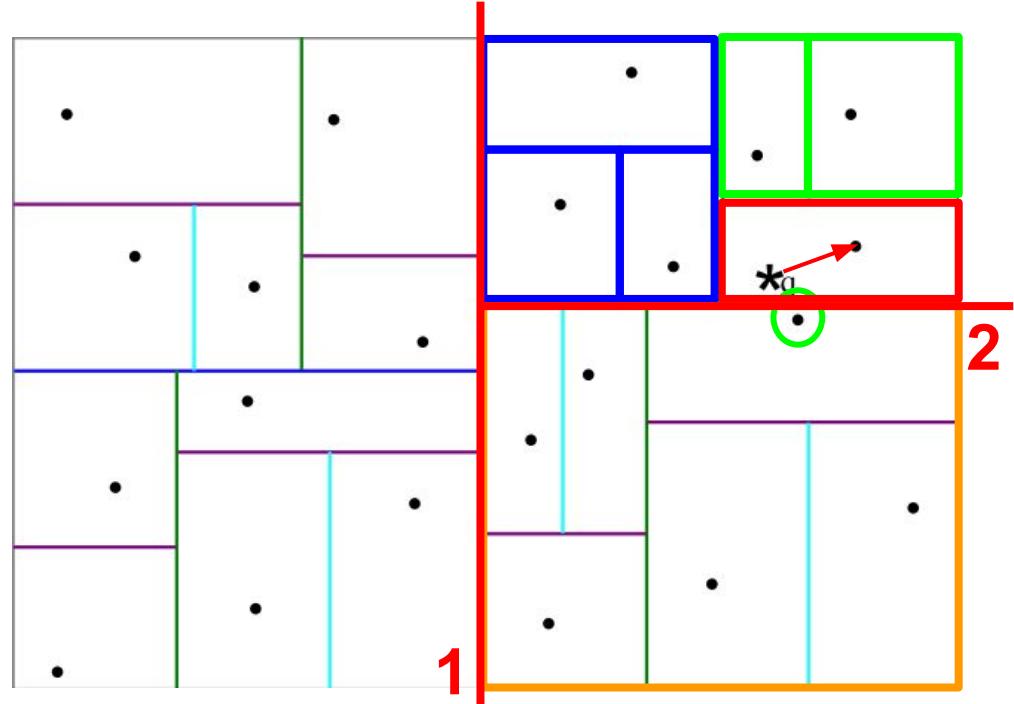


Правильный ответ в другом листе

Scalable Nearest Neighbor Algorithms for High Dimensional Data

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ
- 2) Посещаем остальные листья чтобы уточнить ответ.  
С ускорением через поддержание расстояния до “текущего ответа”.



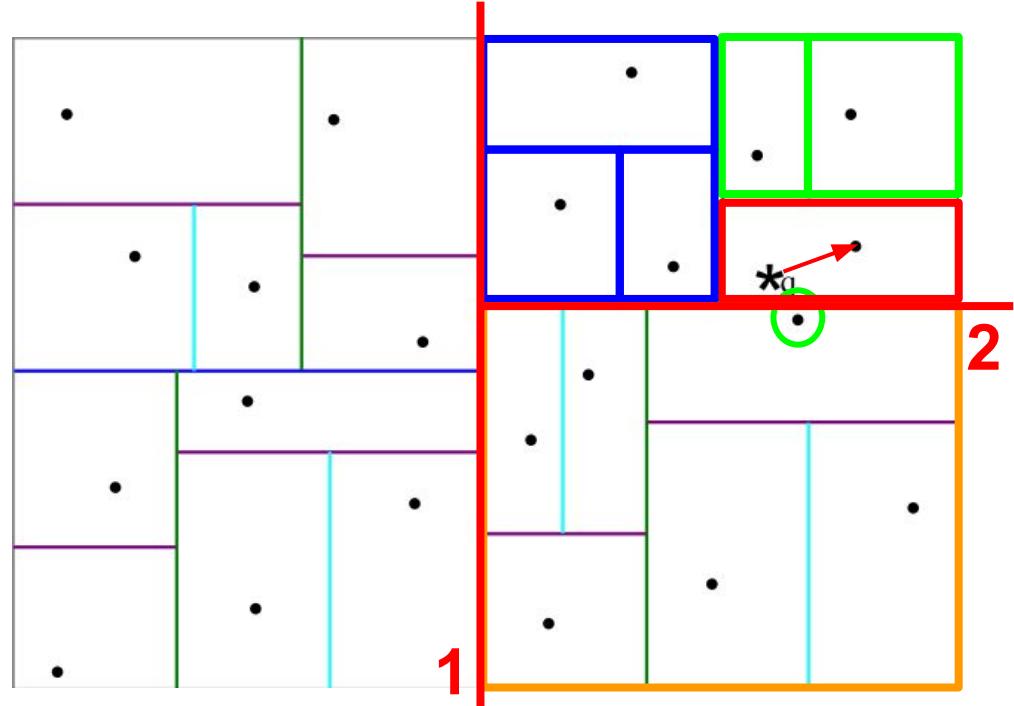
Правильный ответ был отсечен в начале!

Scalable Nearest Neighbor Algorithms for High Dimensional Data

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ
- 2) Посещаем остальные листья чтобы уточнить ответ.  
С ускорением через поддержание расстояния до “текущего ответа”.
- 3) Можно было бы ограничить число рассматриваемых листьев, но часто правильный ответ может быть далеко в другой ветке (отсечен у корня).

**Что делать?**



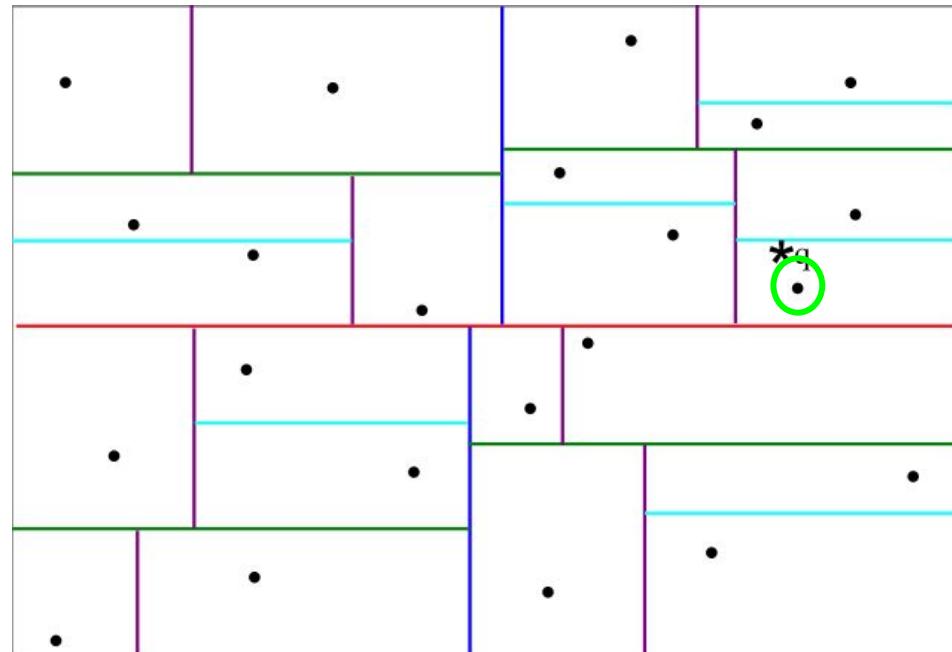
Правильный ответ был отсечен в начале!

Scalable Nearest Neighbor Algorithms for High Dimensional Data

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ
- 2) Посещаем остальные листья чтобы уточнить ответ.  
С ускорением через поддержание расстояния до “текущего ответа”.
- 3) Можно было бы ограничить число рассматриваемых листьев, но часто правильный ответ может быть далеко в другой ветке (отсечен у корня).

**Что делать?**

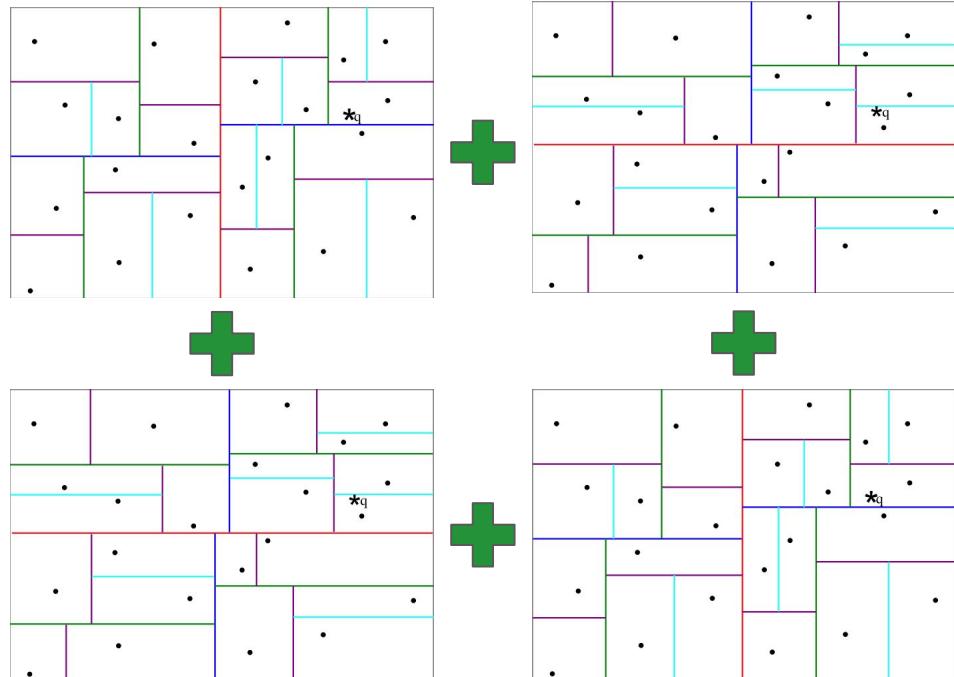


Scalable Nearest Neighbor Algorithms for High Dimensional Data

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ
- 2) Посещаем остальные листья чтобы уточнить ответ.  
С ускорением через поддержание расстояния до “текущего ответа”.
- 3) Ограничим число рассматриваемых листьев.
- 4) Используем несколько деревьев.

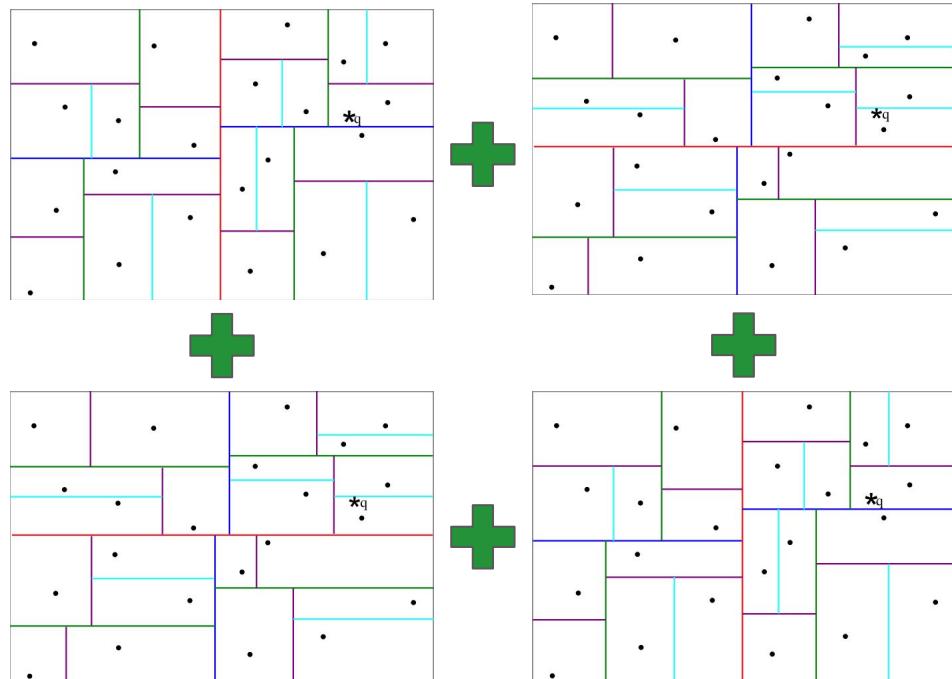
Но ведь KD-tree выбирает размерность рассечения на базе наибольшей дисперсии!  
Т.е. дерево однозначно. Что делать?



[Scalable Nearest Neighbor Algorithms for High Dimensional Data](#)

# Приближенный поиск ближайшего соседа (FLANN)

- 1) Строим KD-tree, заходим в лист, берем его ближайшую точку как ответ
- 2) Посещаем остальные листья чтобы уточнить ответ.  
С ускорением через поддержание расстояния до “текущего ответа”.
- 3) Ограничим число рассматриваемых листьев.
- 4) Используем несколько деревьев, (размерность рассечения выбирается случайно).



[Scalable Nearest Neighbor Algorithms for High Dimensional Data](#)

# Сопоставление точек

Вход:

- **M картинок**
- **N точек на каждой картинке**
- у каждой точки (**128 x float32**) **SIFT** дескриптор

Выход (для каждой пары картинок А и Б):

- **Nearest-Neighbor:** для каждой точки из А - самая похожая точка из Б

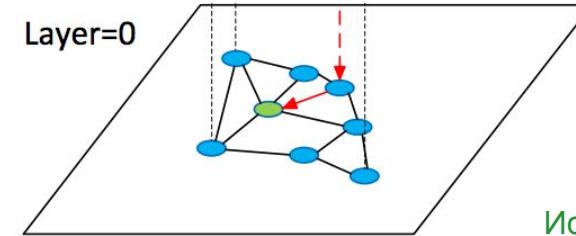
**Brute-Force matching:**  $O(M^2 * N^2)$

Как можно ускорить все вместе?

Например кинув все дескрипторы всех картинок в одну кучу...

# Сопоставление точек: HNSW, FAISS

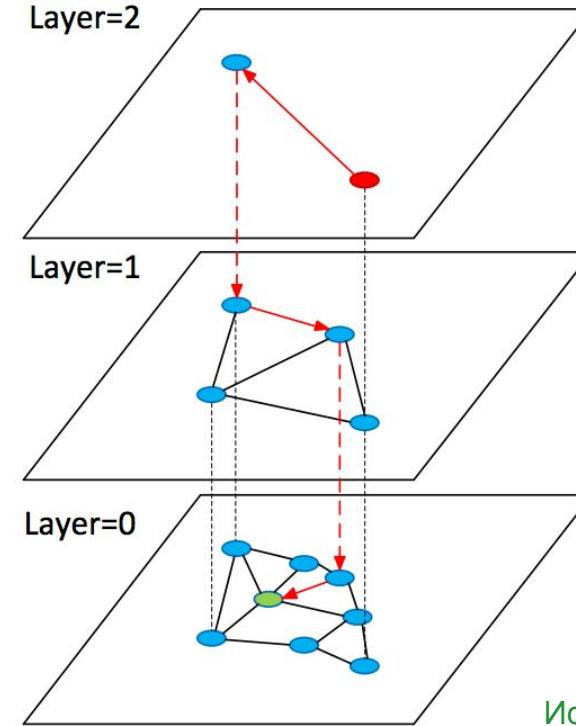
Кинем все дескрипторы со всех фотографий в одну огромную кучу:



# Сопоставление точек: HNSW, FAISS

Кинем все дескрипторы со всех фотографий в одну огромную кучу (Layer=0).

Через прореживание построим coarse-to-fine слои.

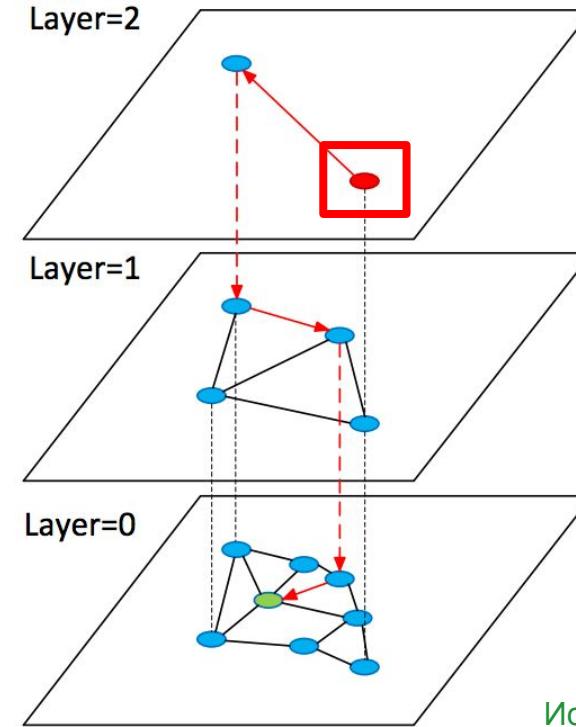


# Сопоставление точек: HNSW, FAISS

Кинем все дескрипторы со всех фотографий в одну огромную кучу (Layer=0).

Через прореживание построим coarse-to-fine слои.

Стартуем в случайной точке Layer=2.



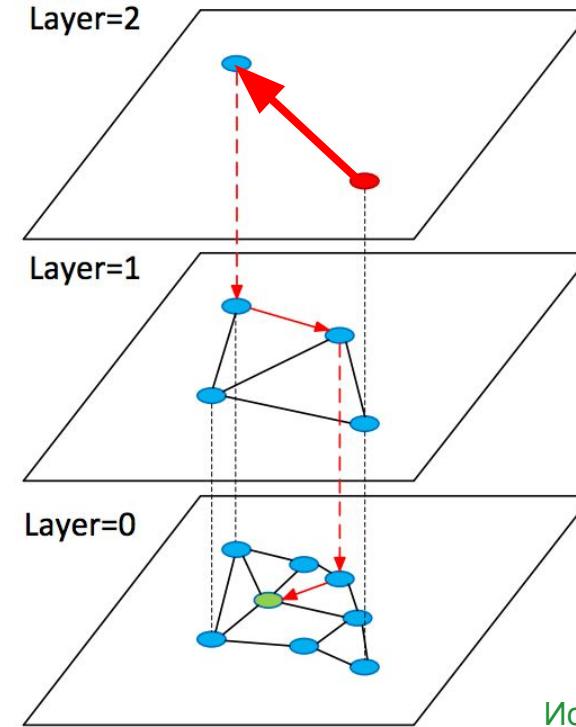
# Сопоставление точек: HNSW, FAISS

Кинем все дескрипторы со всех фотографий в одну огромную кучу (Layer=0).

Через прореживание построим coarse-to-fine слои.

Стартуем в случайной точке Layer=2.

Пока можем улучшить расстояние - ходим по слою через соседей.



# Сопоставление точек: HNSW, FAISS

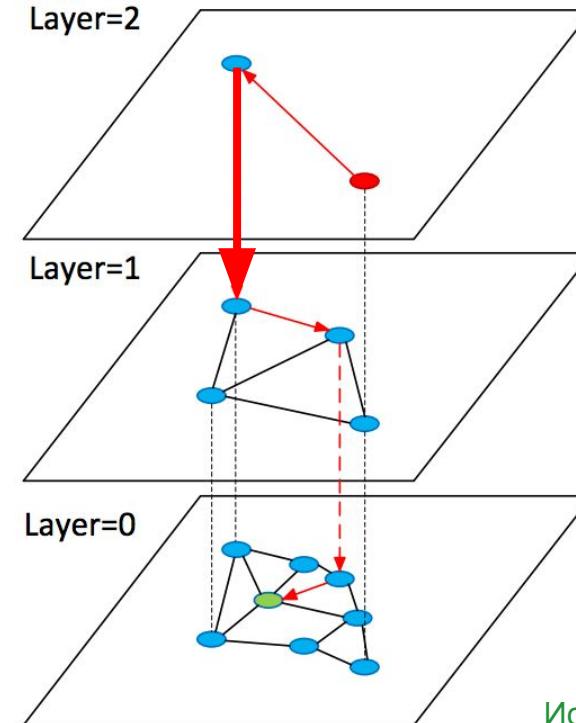
Кинем все дескрипторы со всех фотографий в одну огромную кучу (Layer=0).

Через прореживание построим coarse-to-fine слои.

Стартуем в случайной точке Layer=2.

Пока можем улучшить расстояние - ходим по слою через соседей.

Затем переходим на более детальный уровень.



# Сопоставление точек: HNSW, FAISS

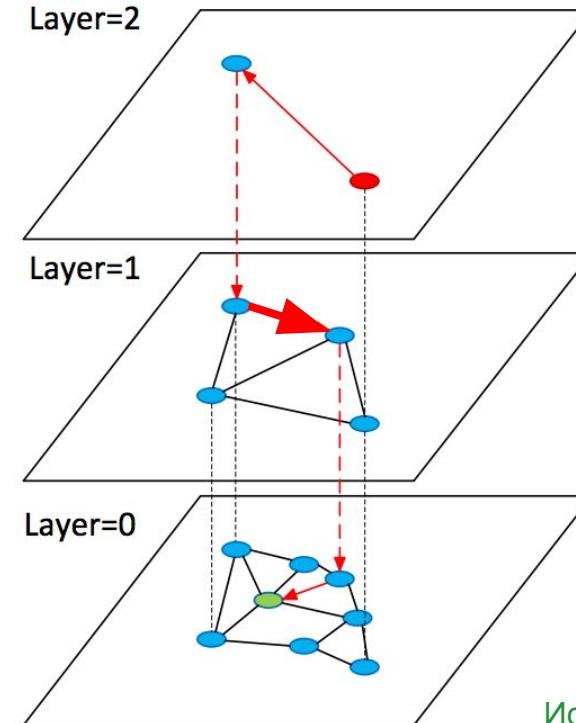
Кинем все дескрипторы со всех фотографий в одну огромную кучу (Layer=0).

Через прореживание построим coarse-to-fine слои.

Стартуем в случайной точке Layer=2.

Пока можем улучшить расстояние - ходим по слою через соседей.

Затем переходим на более детальный уровень.



# Сопоставление точек: HNSW, FAISS

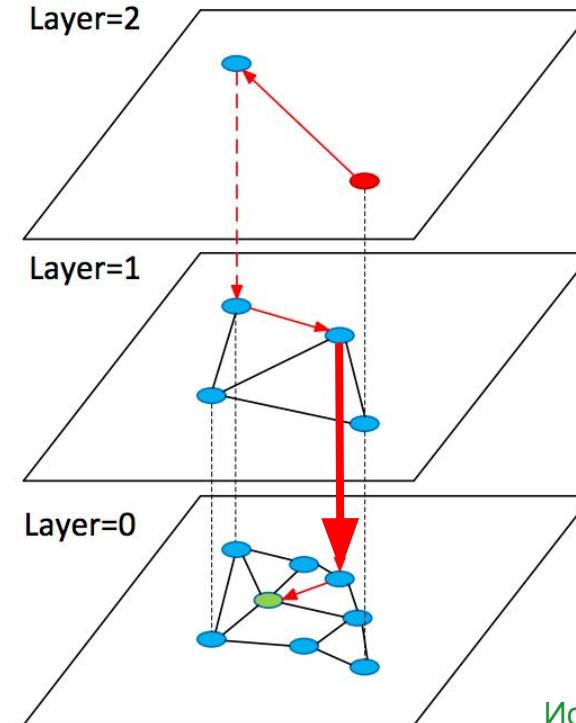
Кинем все дескрипторы со всех фотографий в одну огромную кучу (Layer=0).

Через прореживание построим coarse-to-fine слои.

Стартуем в случайной точке Layer=2.

Пока можем улучшить расстояние - ходим по слою через соседей.

Затем переходим на более детальный уровень.



# Сопоставление точек: HNSW, FAISS

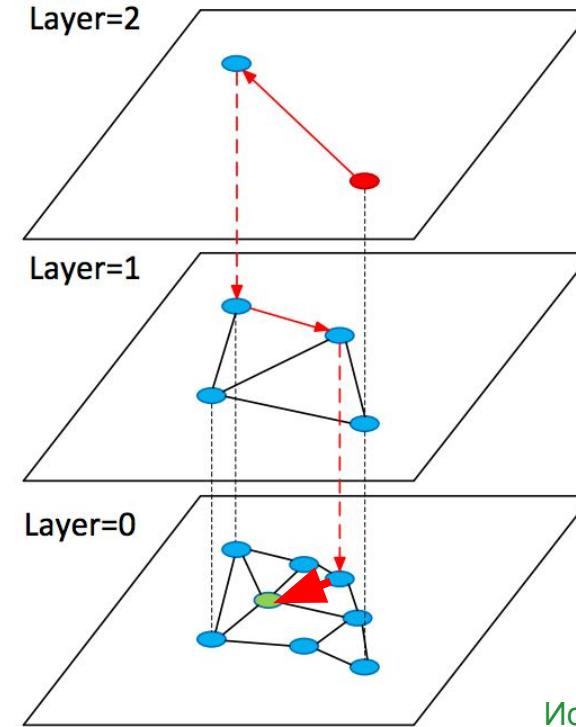
Кинем все дескрипторы со всех фотографий в одну огромную кучу (Layer=0).

Через прореживание построим coarse-to-fine слои.

Стартуем в случайной точке Layer=2.

Пока можем улучшить расстояние - ходим по слою через соседей.

Затем переходим на более детальный уровень.



# Сопоставление точек: HNSW, FAISS

Кинем все дескрипторы со всех фотографий в одну огромную кучу (Layer=0).

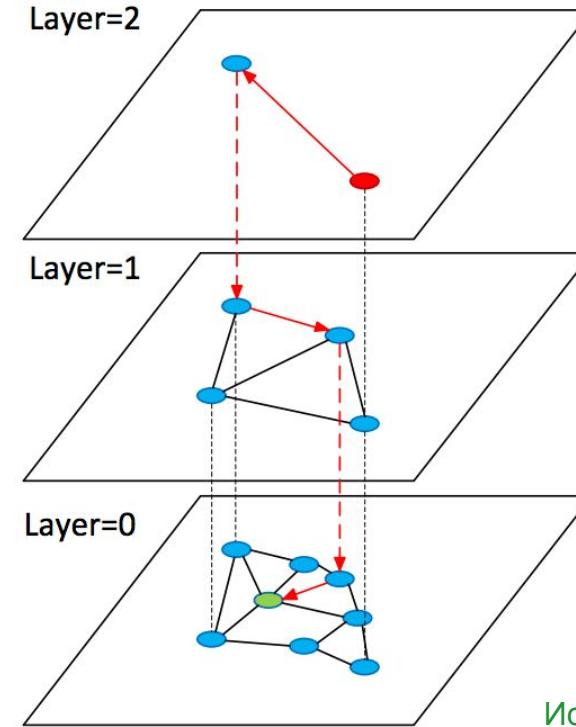
Через прореживание построим coarse-to-fine слои.

Стартуем в случайной точке Layer=2.

Пока можем улучшить расстояние - ходим по слою через соседей.

Затем переходим на более детальный уровень.

**FAISS** = идея кластеризации **k-means** + квантование + ускорение на GPU.



# Сопоставление точек

Вход:

- **M картинок**
- **N точек** на каждой картинке
- у каждой точки (**128 x float32**) **SIFT** дескриптор

Выход (для каждой пары картинок А и Б):

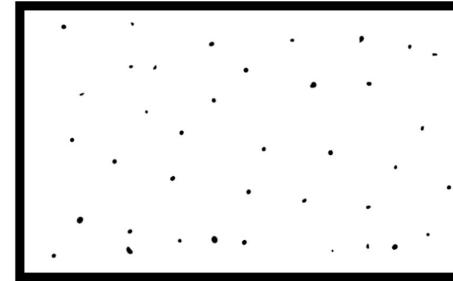
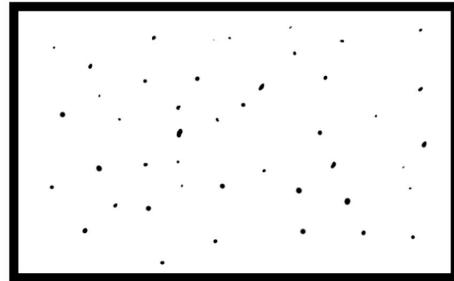
- **Nearest-Neighbor**: для каждой точки из А - самая похожая точка из Б

**Brute-Force matching**:  $O(M^2 * N^2)$

Что если пара больших фотографий (например 400 мегапикселей) и на каждой много ключевых точек (сотни тысяч)?

# Сопоставление точек: Guided Matching

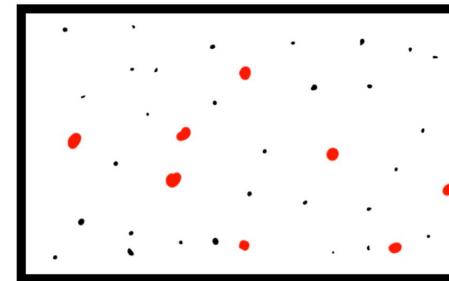
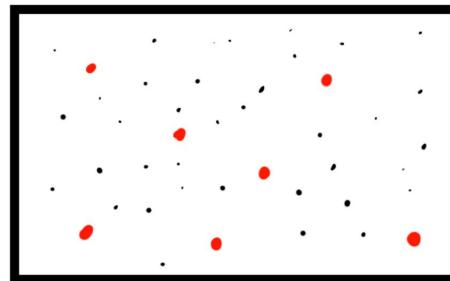
Пусть пара больших фотографий (например 400 мегапикселей) и на каждой много ключевых точек (сотни тысяч).



# Сопоставление точек: Guided Matching

Пусть пара больших фотографий (например 400 мегапикселей) и на каждой много ключевых точек (сотни тысяч).

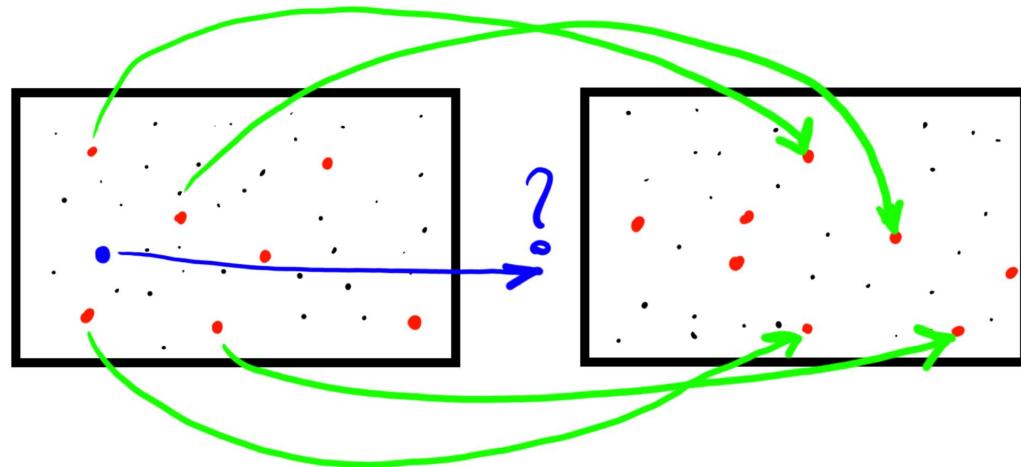
Оставим в каждой из фотографий часть ключевых точек (тысячу).



# Сопоставление точек: Guided Matching

Пусть пара больших фотографий (например 400 мегапикселей) и на каждой много ключевых точек (сотни тысяч).

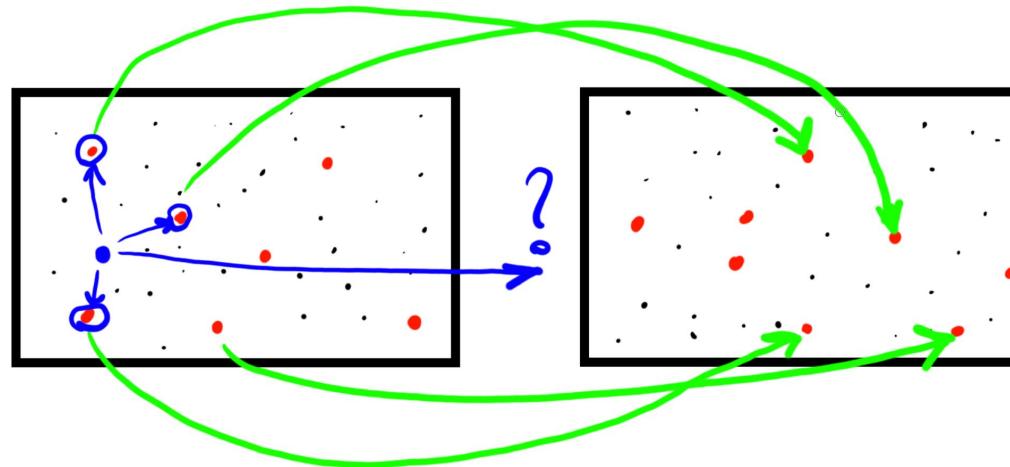
Оставим в каждой из фотографий часть ключевых точек (тысячу) и сопоставим их. **Как это поможет теперь найти ответ для сотен тысяч ключевых точек?**



# Сопоставление точек: Guided Matching

Пусть пара больших фотографий (например 400 мегапикселей) и на каждой много ключевых точек (сотни тысяч).

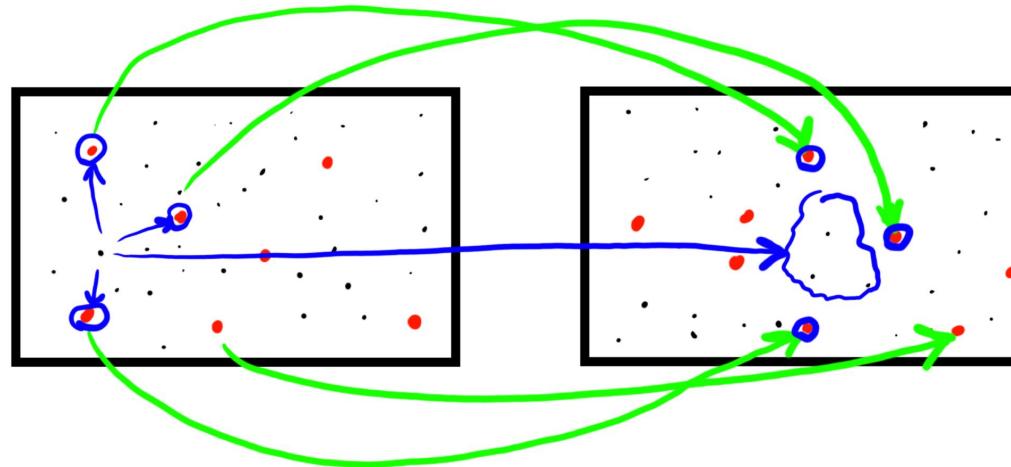
Оставим в каждой из фотографий часть ключевых точек (тысячу) и сопоставим их. **Как это поможет теперь найти ответ для сотен тысяч ключевых точек?**



# Сопоставление точек: Guided Matching

Пусть пара больших фотографий (например 400 мегапикселей) и на каждой много ключевых точек (сотни тысяч).

Оставим в каждой из фотографий часть ключевых точек (тысячу) и сопоставим их. На их базе сузим перечень кандидатов для сопоставления.

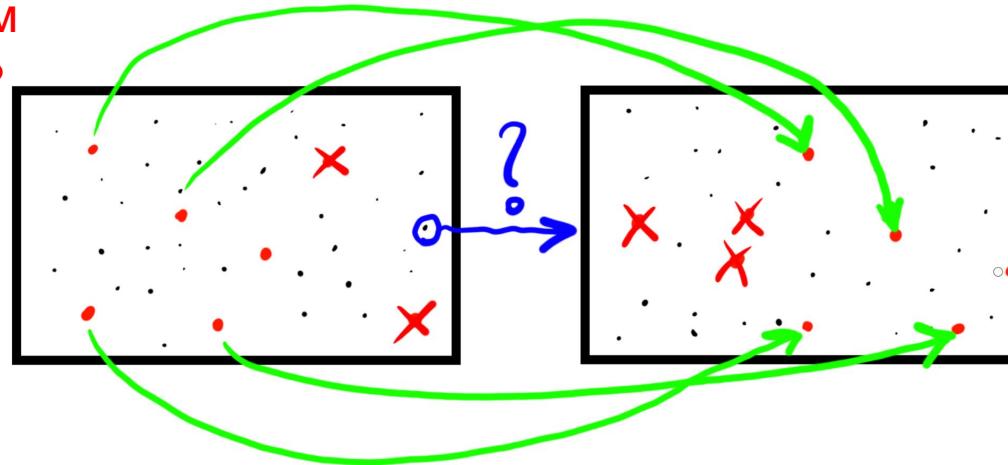


# Сопоставление точек: Guided Matching

Пусть пара больших фотографий (например 400 мегапикселей) и на каждой много ключевых точек (сотни тысяч).

Оставим в каждой из фотографий часть ключевых точек (тысячу) и сопоставим их. На их базе сузим перечень кандидатов для сопоставления.

А что если рядом не сопоставило?

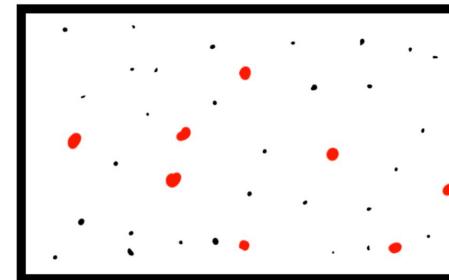
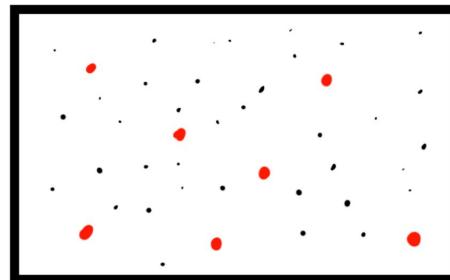


# Сопоставление точек: Guided Matching

Пусть пара больших фотографий (например 400 мегапикселей) и на каждой много ключевых точек (сотни тысяч).

Оставим в каждой из фотографий часть ключевых точек (тысячу) и сопоставим их. На их базе сузим перечень кандидатов для сопоставления.

**А какую тысячу точек выбрать? Какие требования к критерию выбора?**

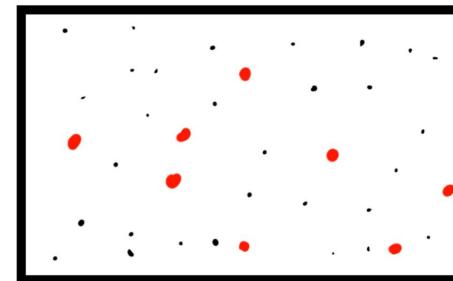
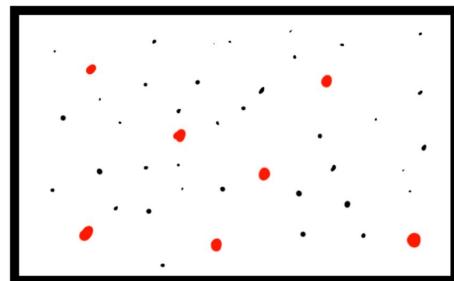


# Сопоставление точек: Guided Matching

Пусть пара больших фотографий (например 400 мегапикселей) и на каждой много ключевых точек (сотни тысяч).

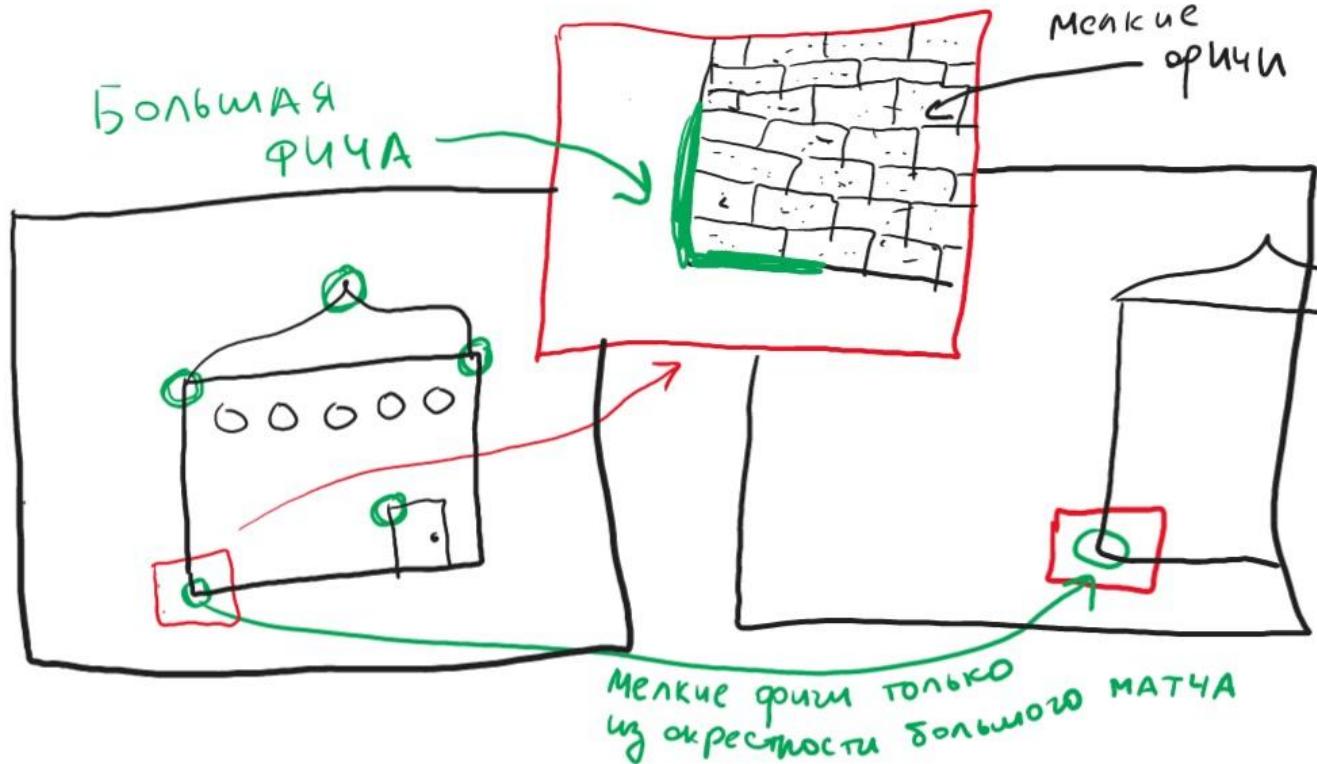
Оставим в каждой из фотографий часть ключевых точек (тысячу) и сопоставим их. На их базе сузим перечень кандидатов для сопоставления.

Нужно выбрать точки инвариантно (чтобы они повторялись) - **какие?**



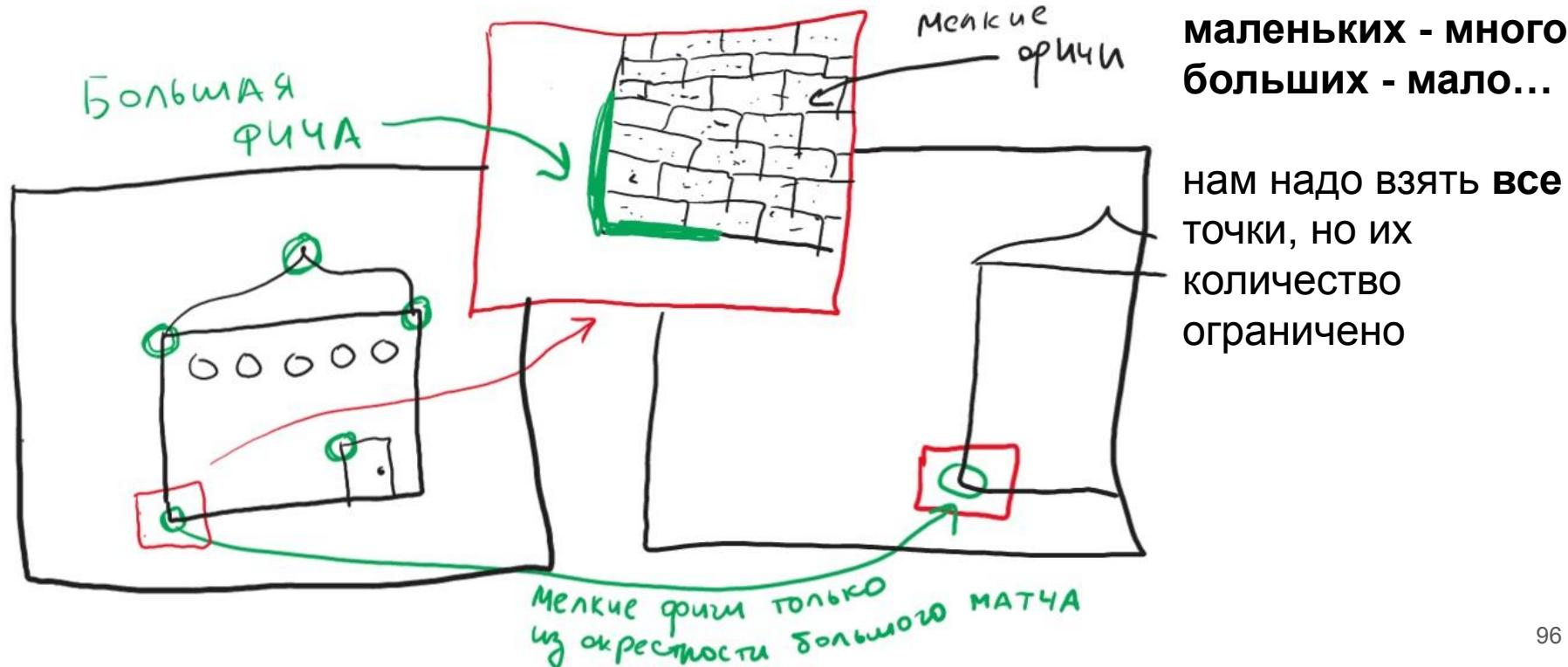
# Сопоставление точек: Guided Matching

Нужно выбрать точки инвариантно - самые большие или самые маленькие?



# Сопоставление точек: Guided Matching

Нужно выбрать точки инвариантно - самые большие или самые маленькие?

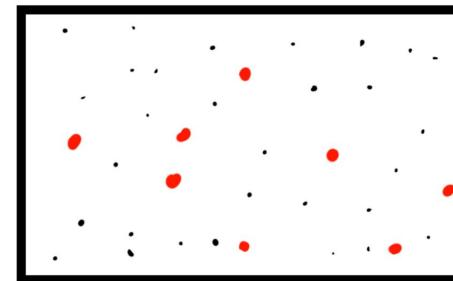
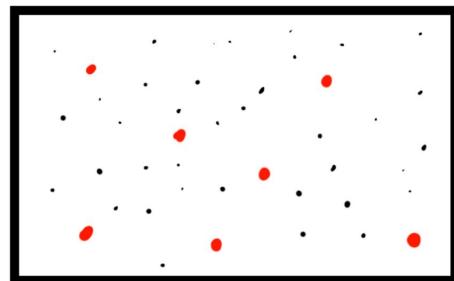


# Сопоставление точек: Guided Matching

Пусть пара больших фотографий (например 400 мегапикселей) и на каждой много ключевых точек (сотни тысяч).

Оставим в каждой из фотографий часть ключевых точек (тысячу) и сопоставим их. На их базе сузим перечень кандидатов для сопоставления.

Нужно выбрать точки инвариантно (чтобы они повторялись) - **самые большие!**



# Сопоставление точек

Вход:

- **M картинок**
- **N точек на каждой картинке**
- у каждой точки (**128 x float32**) **SIFT** дескриптор

Выход (для каждой пары картинок А и Б):

- **Nearest-Neighbor:** для каждой точки из А - самая похожая точка из Б

Brute-Force matching:  $O(M^2 * N^2)$

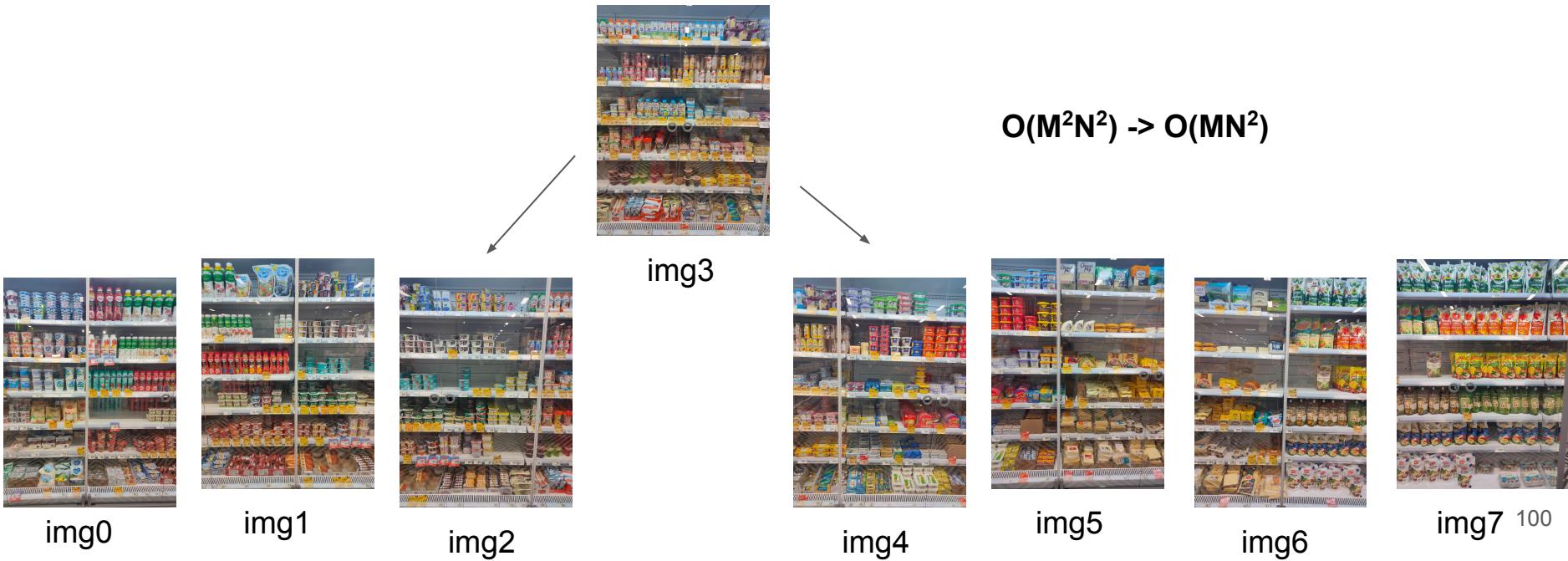
Как можно ускорить первый множитель?

# Преселекция по ближайшим кадрам



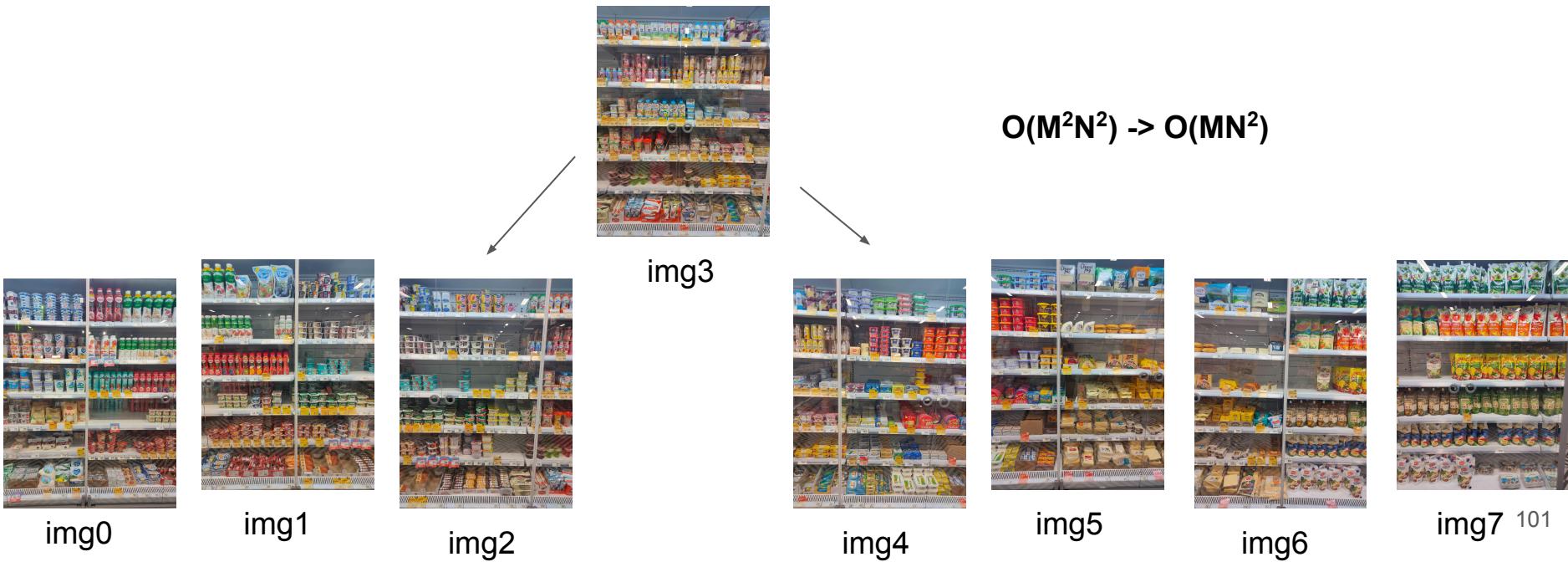
# Преселекция по ближайшим кадрам

- Если снимки идут подряд:



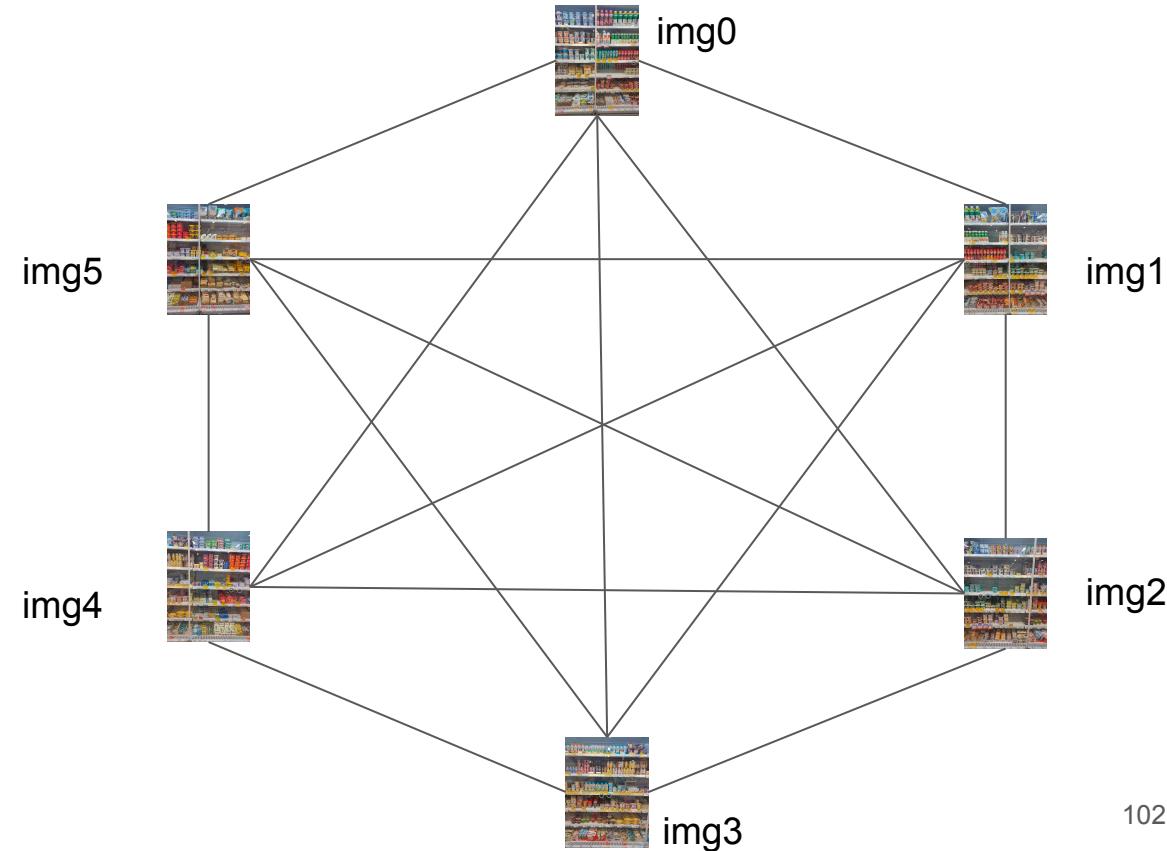
# Преселекция по ближайшим кадрам

- Если снимки идут подряд или известна близость по GPS координатам:



# Преселекция по грубым сопоставлениям

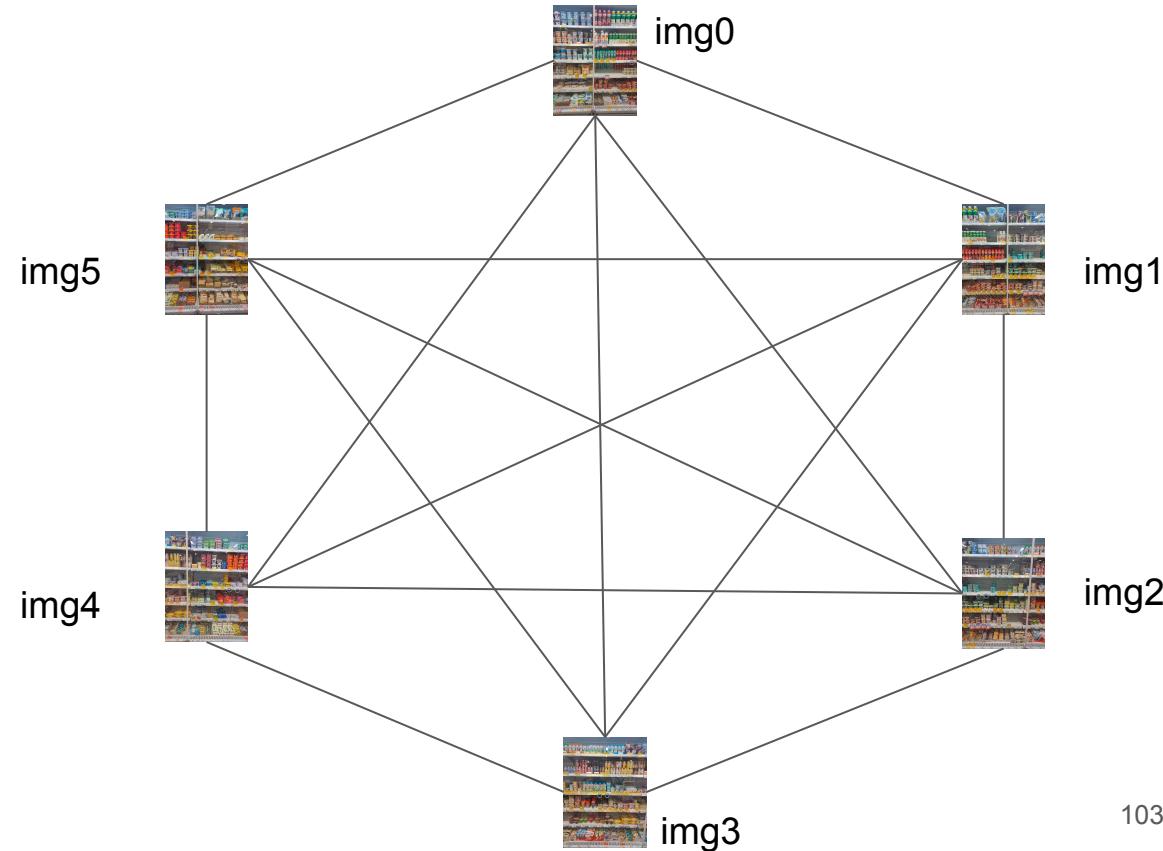
- $O(M^2)$  пар матчинга с маленьким количеством крупных фичей ( $N_{\text{strong}} = 1000$ )



# Преселекция по грубым сопоставлениям

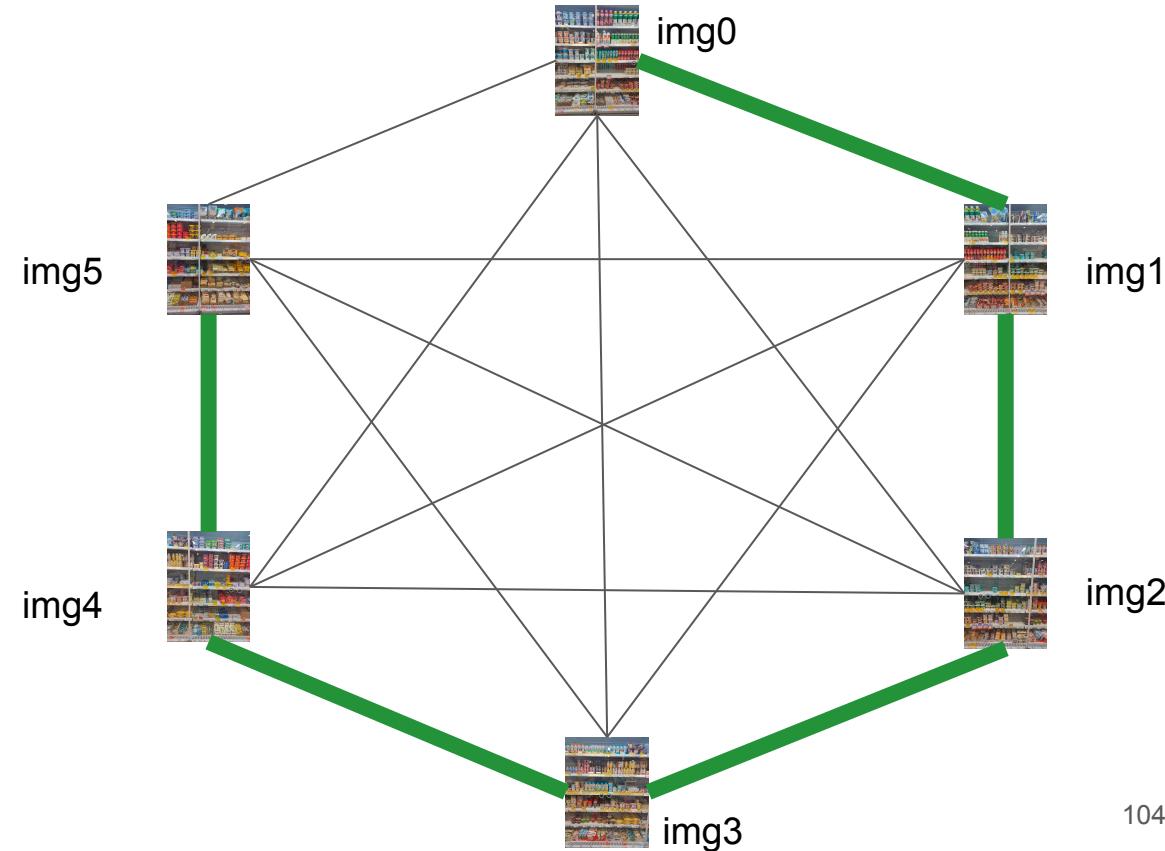
- $O(M^2)$  пар матчинга с маленьким количеством крупных фичей ( $N_{\text{strong}} = 1000$ )

Какие пары фотографий  
дальше сопоставлять?  
М пар с наибольшим  
числом общих точек?



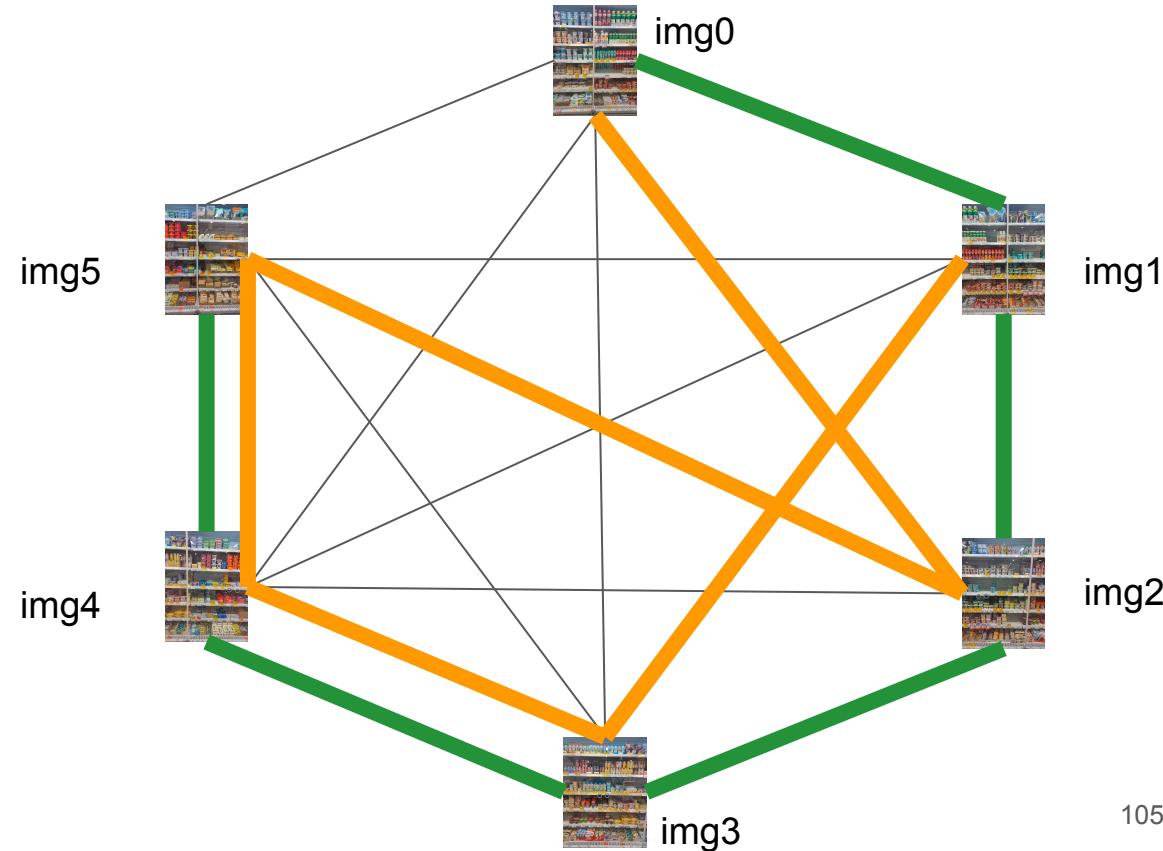
# Преселекция по грубым сопоставлениям

- $O(M^2)$  пар матчинга с маленьким количеством крупных фичей ( $N_{\text{strong}} = 1000$ )
- Найдем два остовных дерева максимальных по количеству матчей



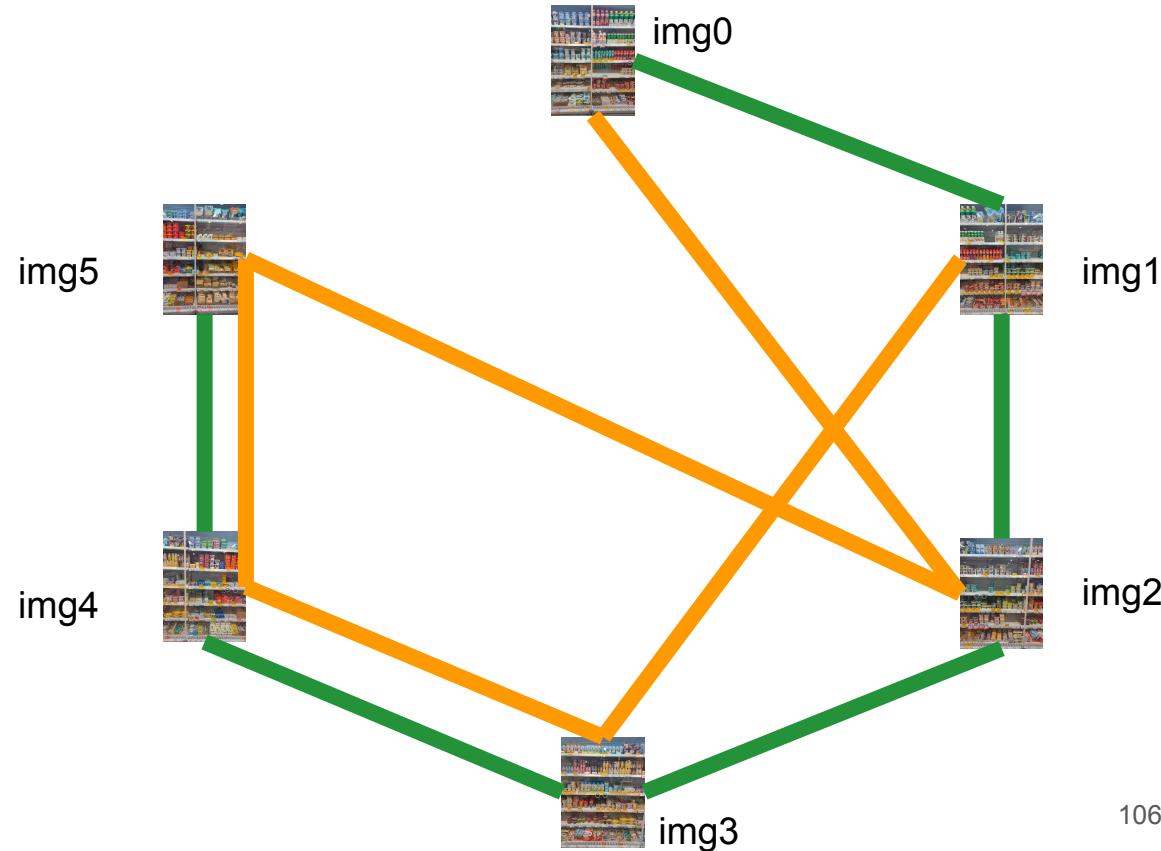
# Преселекция по грубым сопоставлениям

- $O(M^2)$  пар матчинга с маленьким количеством крупных фичей ( $N_{\text{strong}} = 1000$ )
- Найдем два остовных дерева максимальных по количеству матчей



# Преселекция по грубым сопоставлениям

- $O(M^2)$  пар матчинга с маленьким количеством крупных фичей ( $N_{\text{strong}} = 1000$ )
- Найдем два остовных дерева максимальных по количеству матчей
- $O(M)$  пар матчинга с полным количеством фичей ( $N=40000$ )
- Суммарная сложность (только матчинг):  
 $O(M^2N_{\text{strong}}^2) + O(MN^2)$

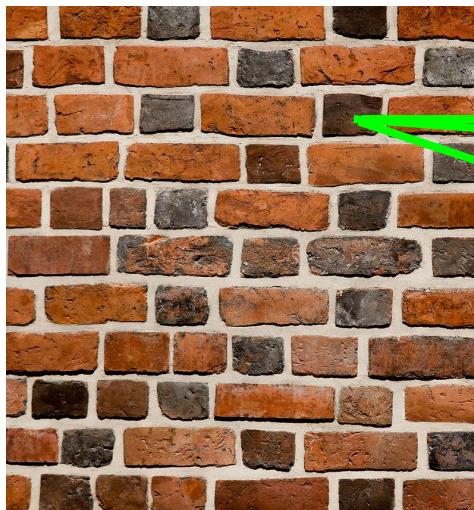


# Фильтрация сопоставлений

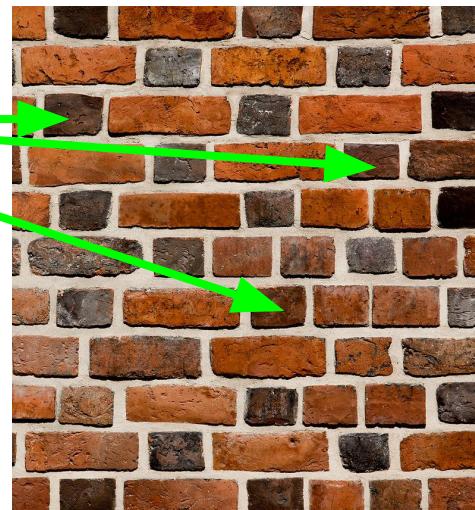
- 1) Научились быстро сопоставлять похожие ключевые точки
- 2) Ложные сопоставления? Какой может быть пример когда их много?

# Фильтрация сопоставлений

- 1) Научились быстро сопоставлять похожие ключевые точки
- 2) Ложные сопоставления? Какой может быть пример когда их много?

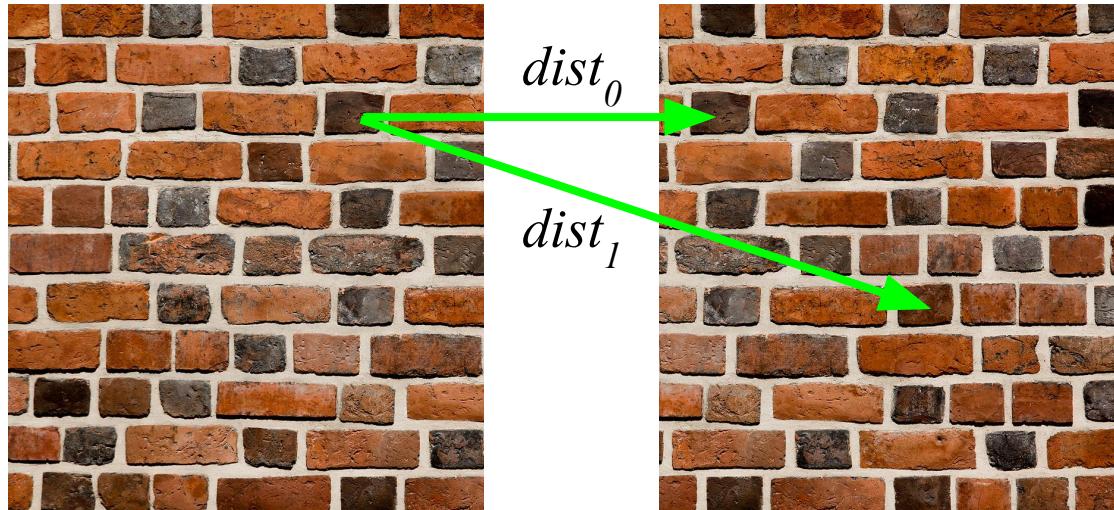


?



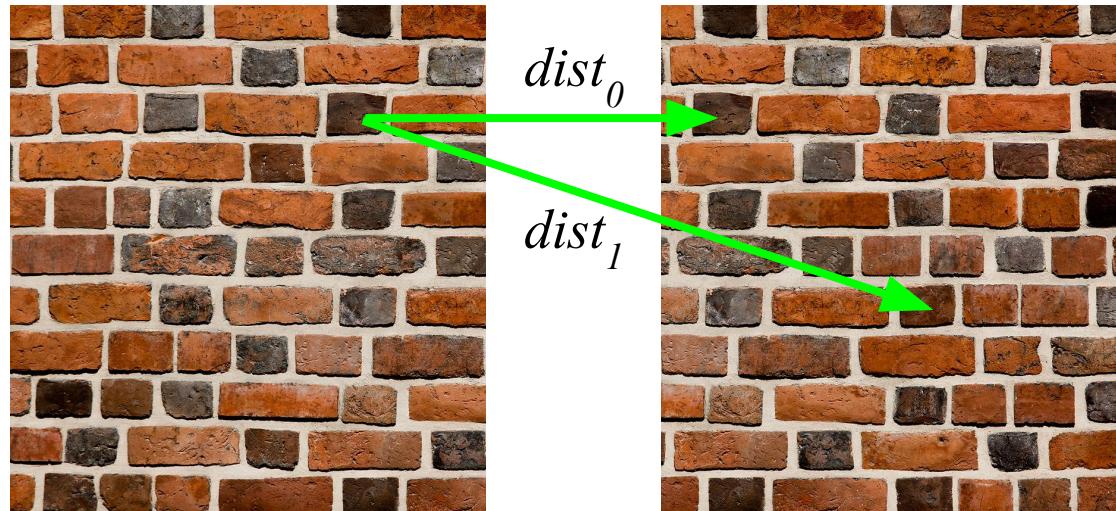
# Фильтрация сопоставлений

- 1) Научились быстро сопоставлять похожие ключевые точки
- 2) Ложные сопоставления? Какой может быть пример когда их много?
- 3) Как отличаются первый и второй дескрипторы в хорошем и плохом случае?



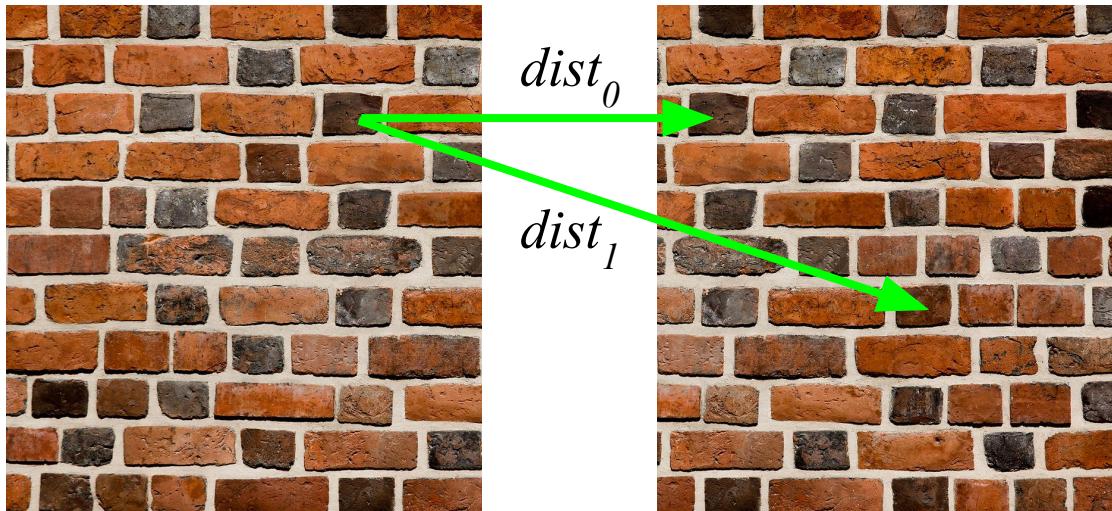
# Фильтрация сопоставлений. K-ratio test

1) Ищем не только ближайшего соседа ( $dist_0$ ) но и второго ( $dist_1$ ) по близости



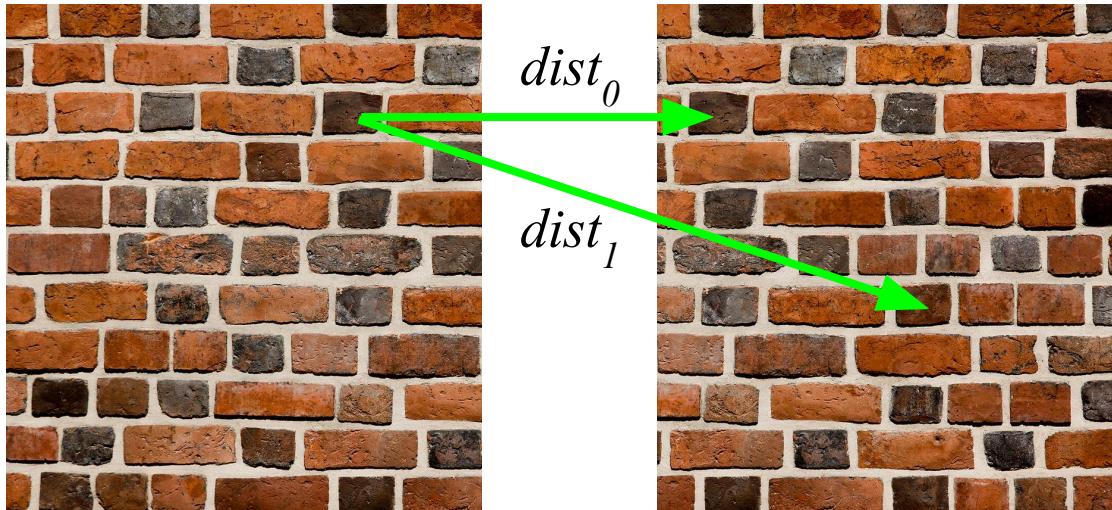
# Фильтрация сопоставлений. K-ratio test

- 1) Ищем не только ближайшего соседа ( $dist_0$ ) но и второго ( $dist_1$ ) по близости
- 2) Если отличается слабо ( $dist_0 * 0.8 < dist_1$ ) - риск ошибки (неоднозначность)

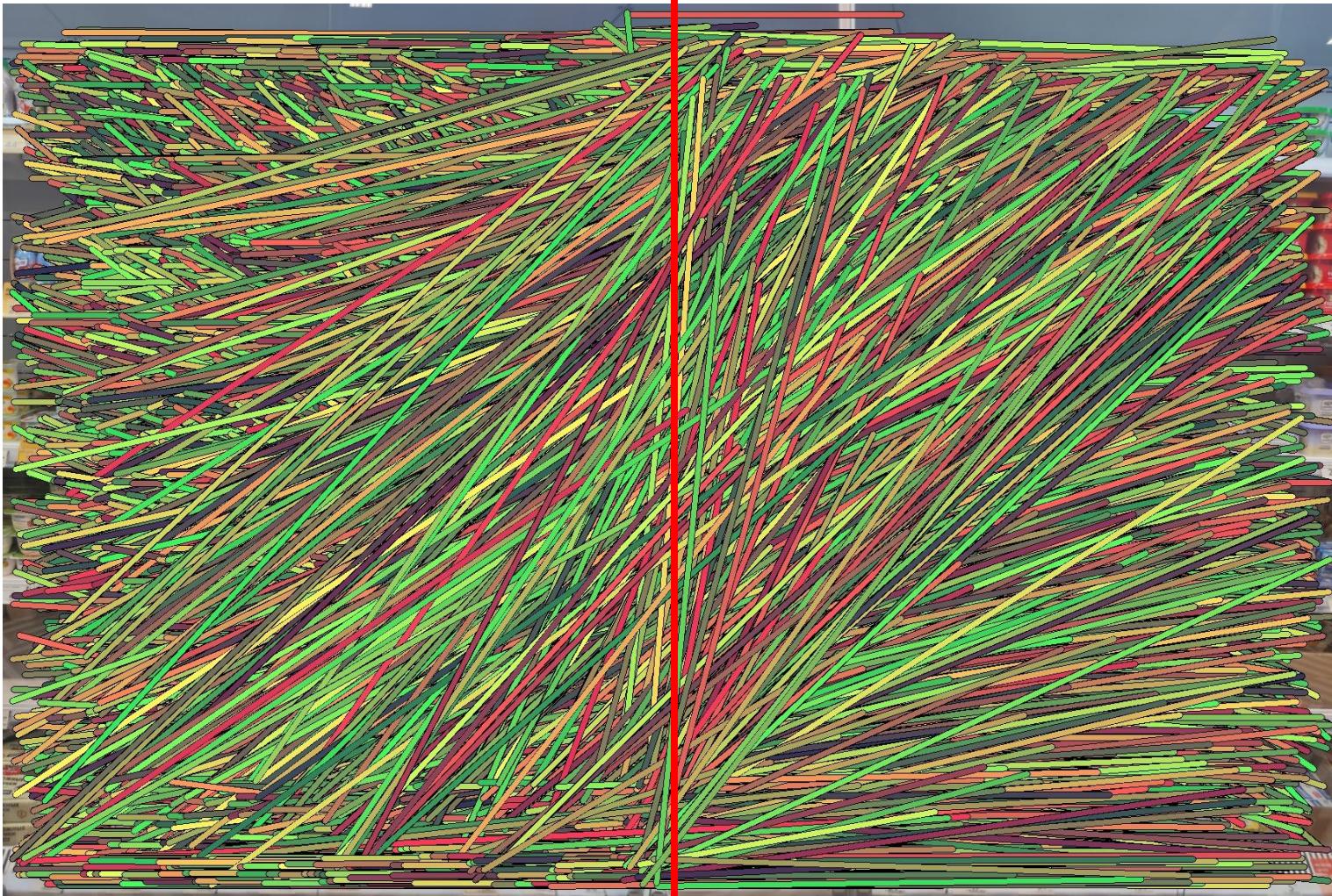


# Фильтрация сопоставлений. K-ratio test

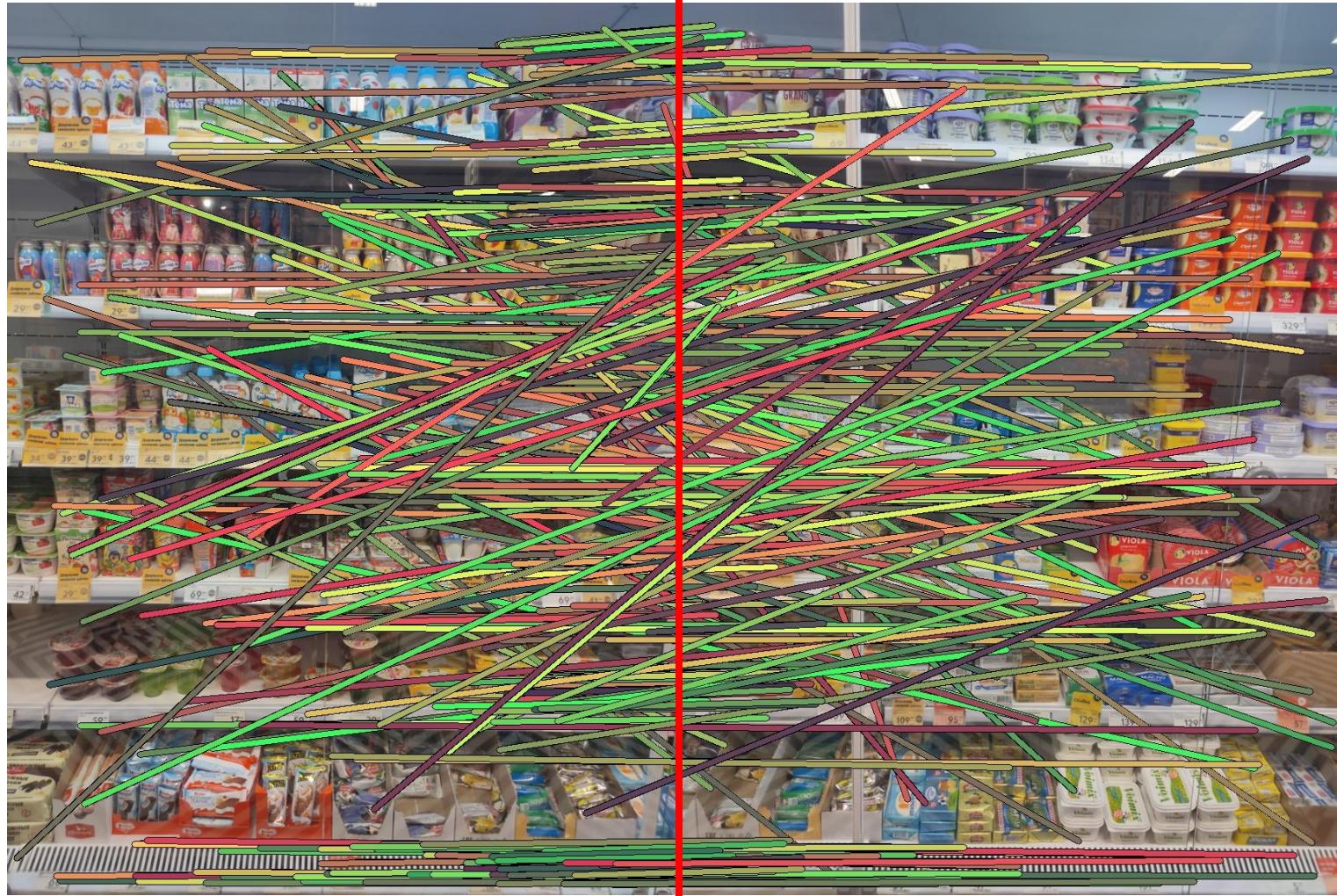
- 1) Ищем не только ближайшего соседа ( $dist_0$ ) но и второго ( $dist_1$ ) по близости
- 2) Если отличается слабо ( $dist_0 * 0.8 < dist_1$ ) - риск ошибки (неоднозначность)
- 3) Если отличается сильно ( $dist_0 * 0.8 > dist_1$ ) - сопоставление надежное



# 1. Nearest Neighbors



## 2. K-ratio test



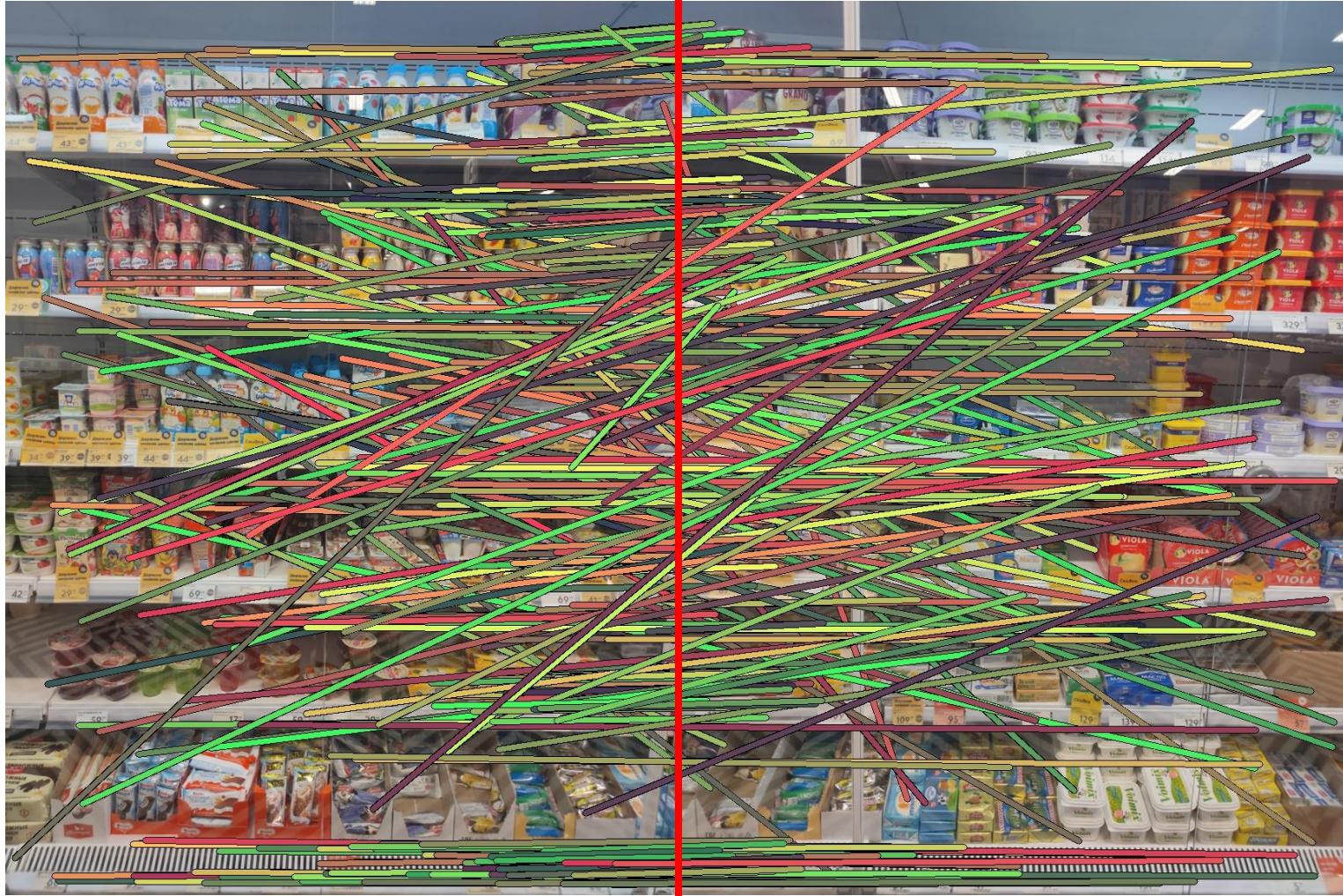
# **Фильтрация сопоставлений**

- 1) Кроме дубликатов есть и просто шумы, которые достаточно уникальны, но не являются хорошими матчами

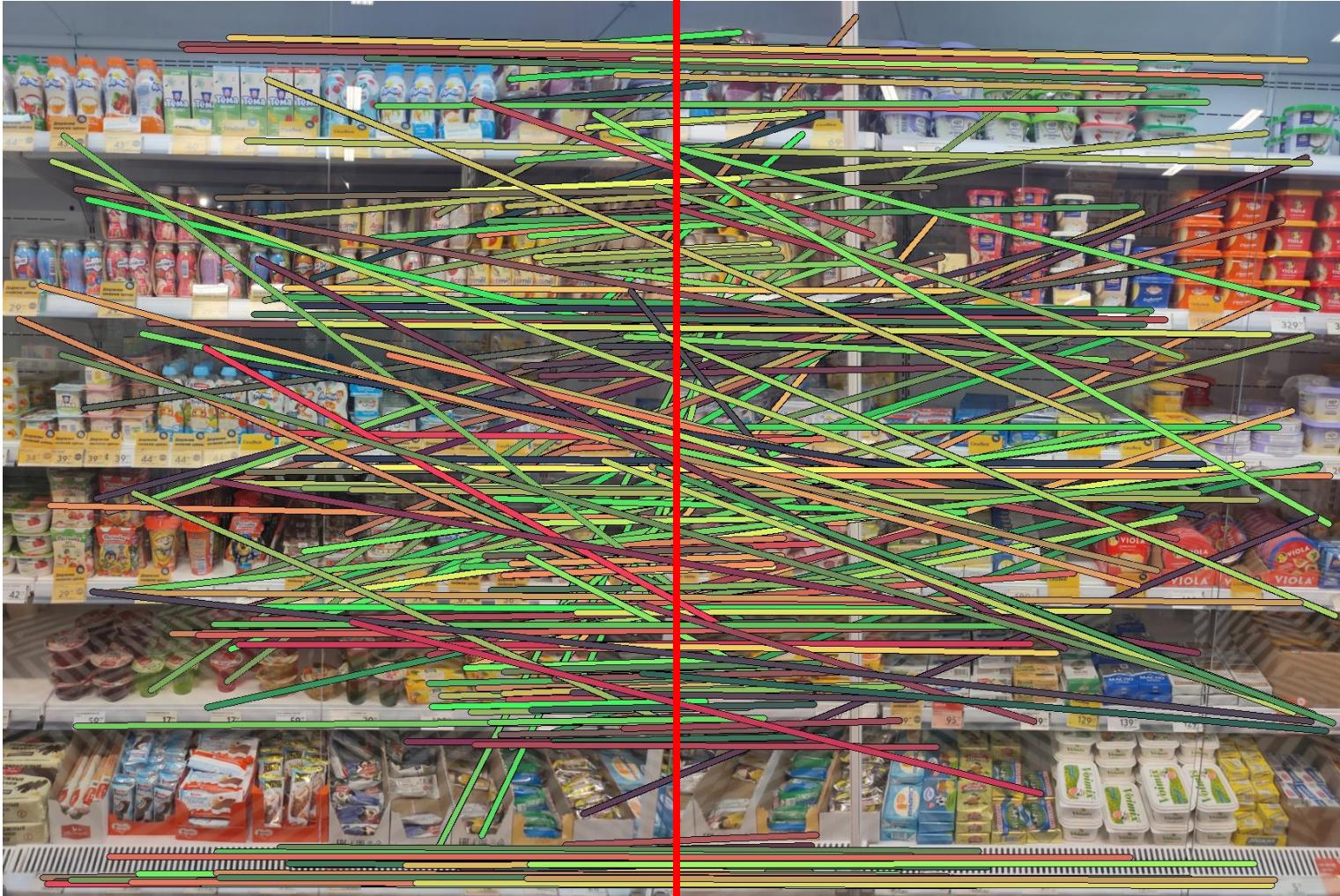
# Фильтрация сопоставлений

- 1) Кроме дубликатов есть и просто шумы, которые достаточно уникальны, но не являются хорошими матчами
- 2) Можно ли как-то использовать информацию о том, какие сопоставления нашлись при поиске ближайших матчей со второй картинки на первую?

## 2. K-ratio test

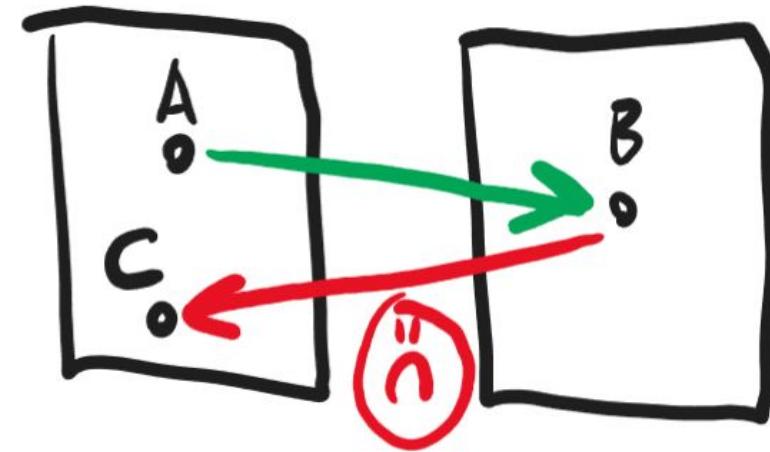
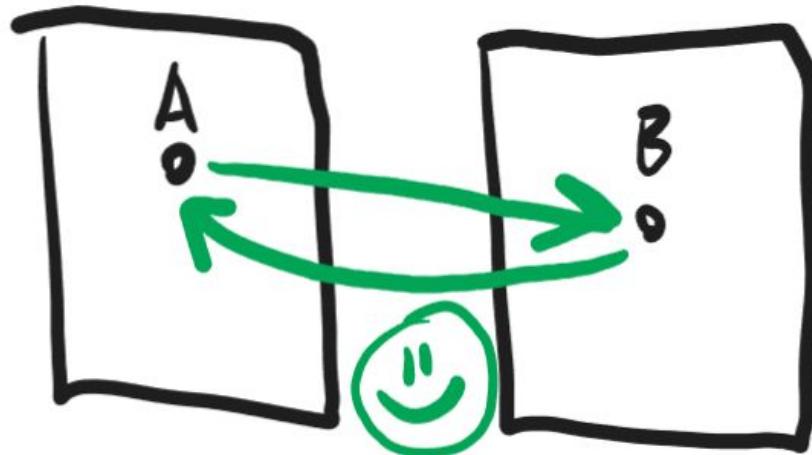


## 2. K-ratio test (right-to-left)

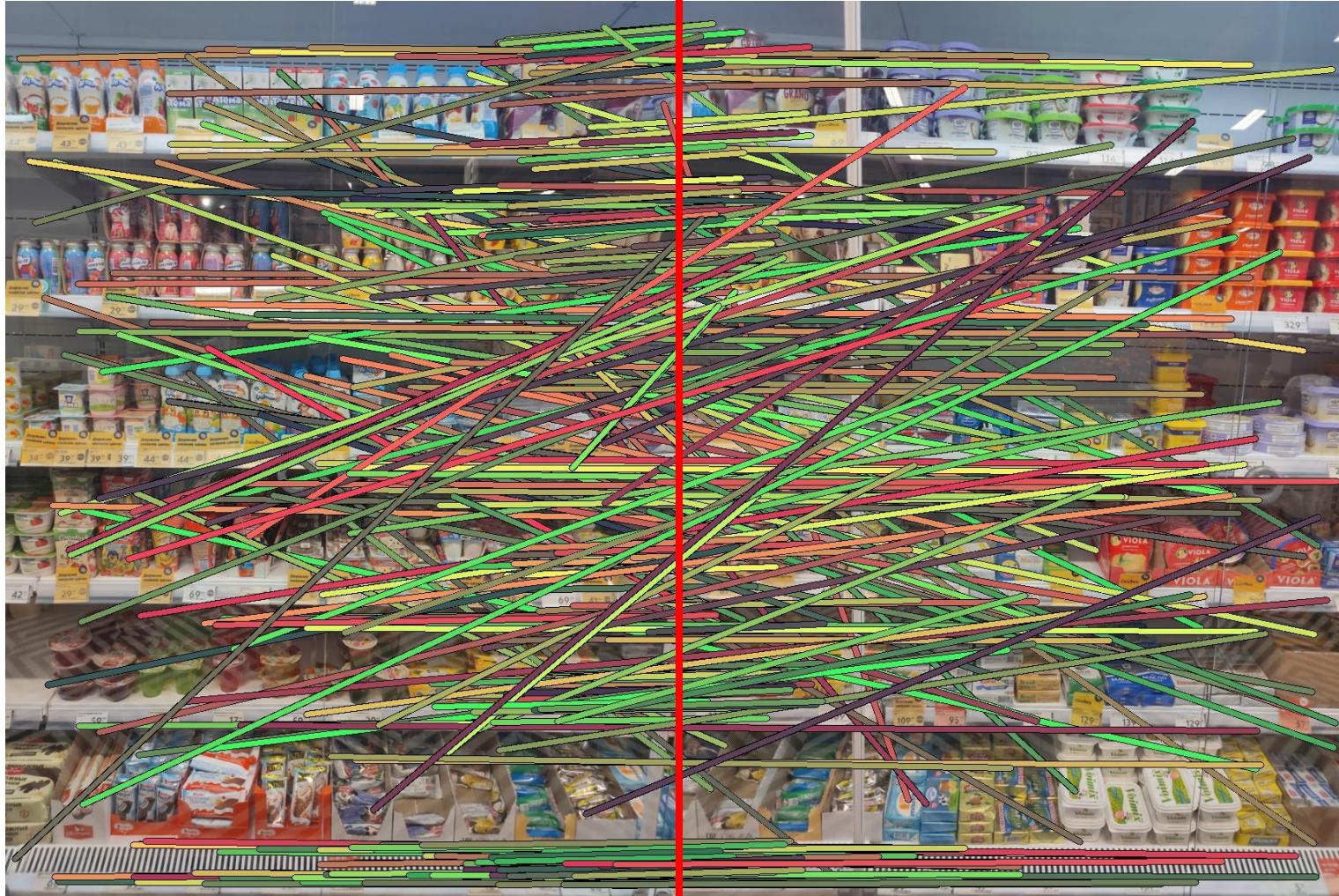


# Фильтрация сопоставлений. Left-right check

- 1) Кроме дубликатов есть и просто шумы, которые достаточно уникальны, но не являются хорошими матчами
- 2) Можно ли как-то использовать информацию о том, какие сопоставления нашлись при поиске ближайших матчей со второй картинки на первую?
- 3) Матчим справа-налево и слева-направо, оставляем только согласующиеся



## 2. K-ratio test

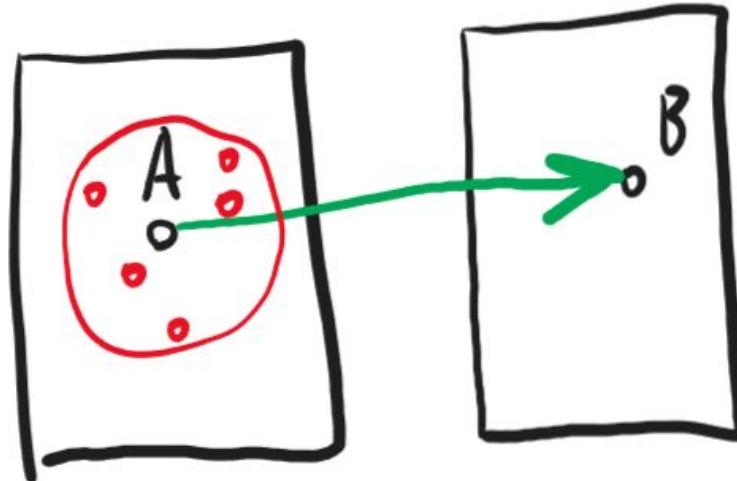


### 3. Left-right check



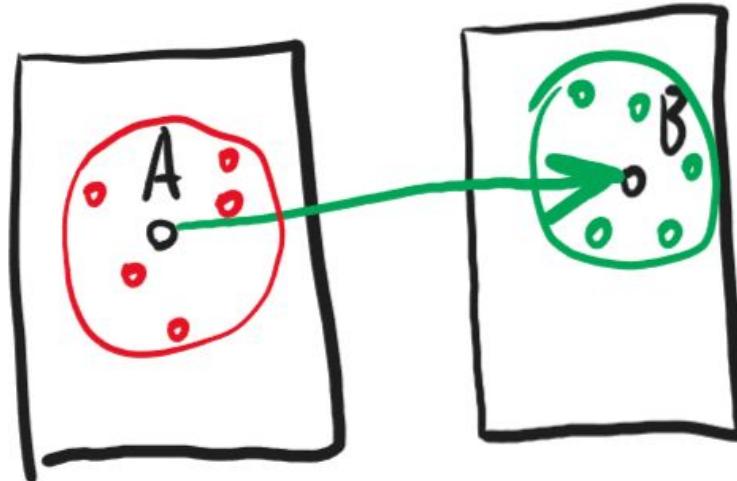
# Фильтрация сопоставлений

- 1) Кроме дубликатов есть и просто шумы, которые достаточно уникальны, но не являются хорошими матчами
- 2) Как должна перейти кучка ключевых точек с первой картинки на вторую?



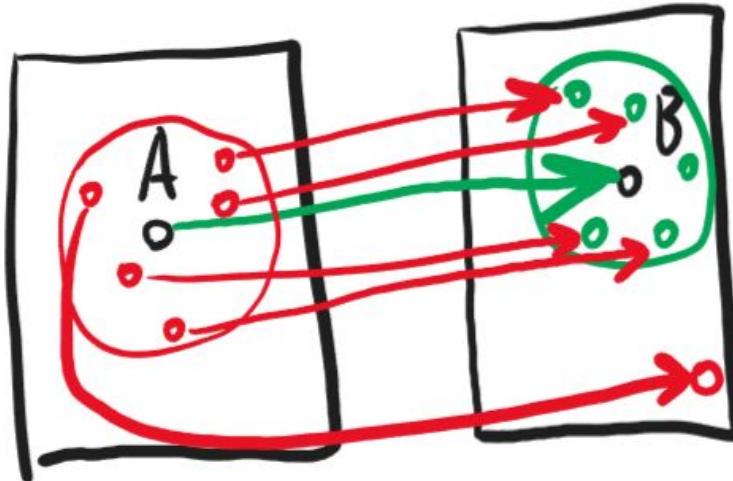
# Фильтрация сопоставлений

- 1) Кроме дубликатов есть и просто шумы, которые достаточно уникальны, но не являются хорошими матчами
- 2) Как должна перейти кучка ключевых точек с первой картинки на вторую?



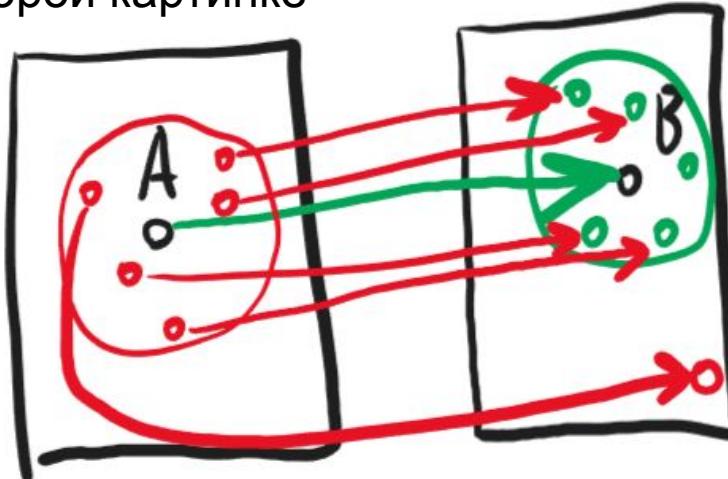
# Фильтрация сопоставлений

- 1) Кроме дубликатов есть и просто шумы, которые достаточно уникальны, но не являются хорошими матчами
- 2) Как должна перейти кучка ключевых точек с первой картинки на вторую?



# Фильтрация сопоставлений. Cluster filtering

- 1) Кроме дубликатов есть и просто шумы, которые достаточно уникальны, но не являются хорошими матчами
- 2) Как должна перейти кучка ключевых точек с первой картинки на вторую?
- 3) Из нашей **k**-окрестности **больше половины** точек должны перейти в **k**-окрестность на второй картинке



### 3. Left-right check



## 4. Cluster filtering

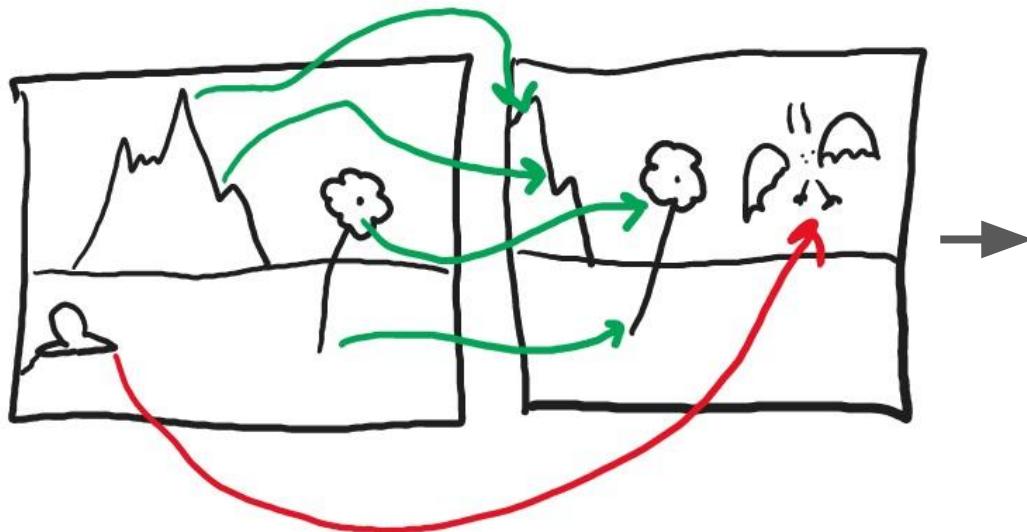


# **Фильтрация сопоставлений**

- 1) Попробуем как-нибудь использовать модель задачи которую решаем?

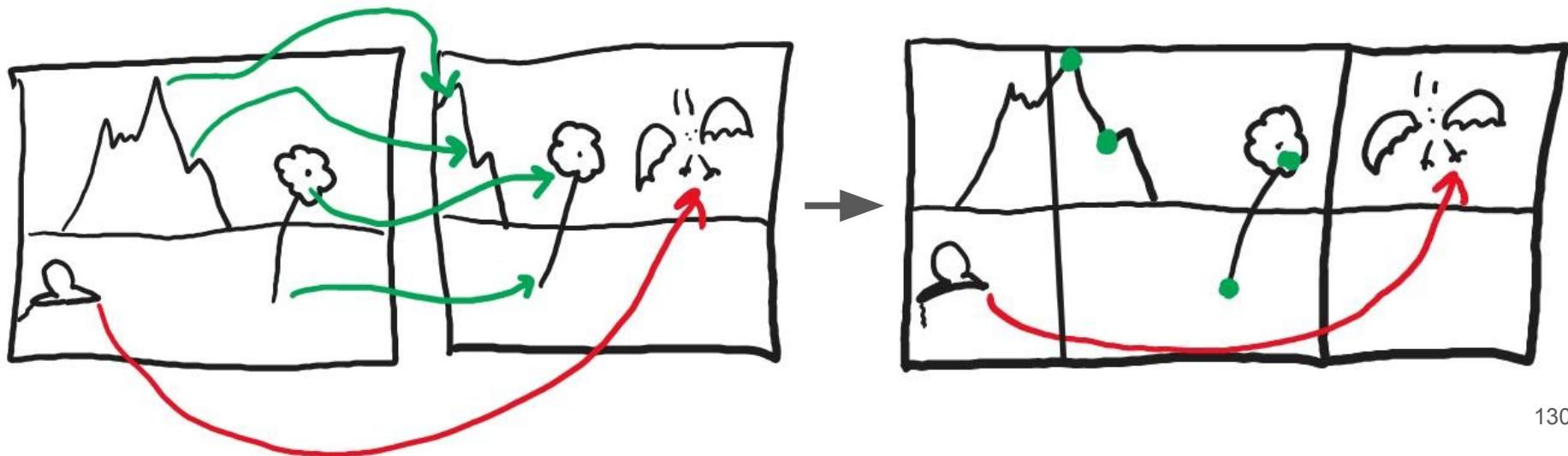
# Фильтрация сопоставлений

- 1) Попробуем как-нибудь использовать модель задачи которую решаем?
- 2) Пусть мы знаем, как одна картинка накладывается на другую. Что произойдет с координатами **плохих** и **хороших** сматченных точек со второй картинки после сдвига?



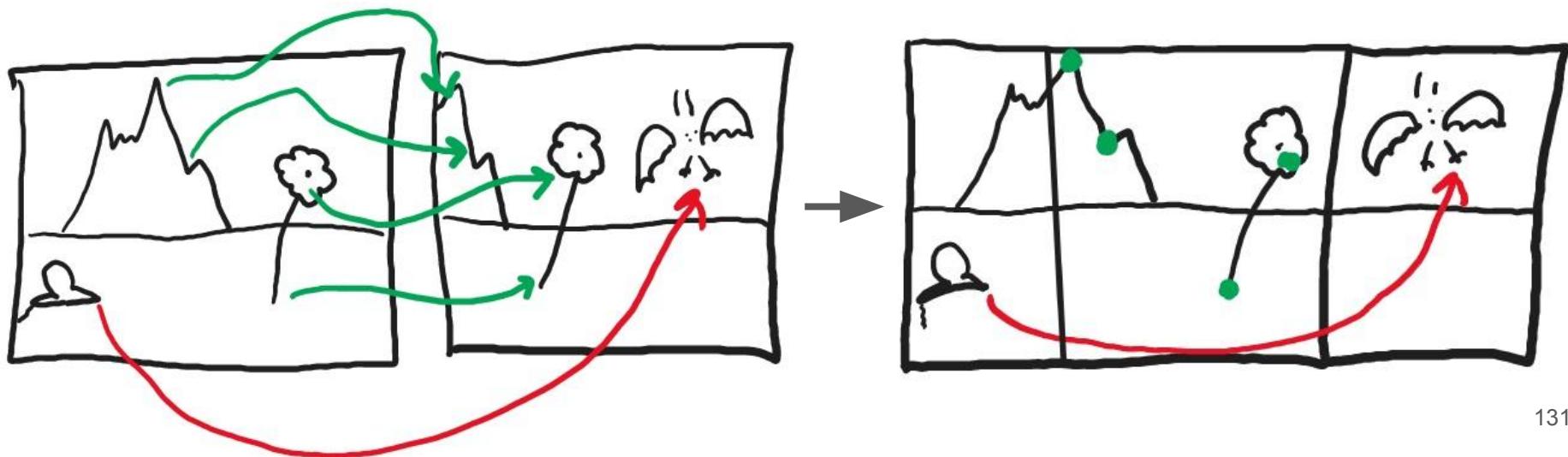
# Фильтрация сопоставлений

- 1) Попробуем как-нибудь использовать модель задачи которую решаем?
- 2) Пусть мы знаем, как одна картинка накладывается на другую. Что произойдет с координатами **плохих** и **хороших** сматченных точек со второй картинки после сдвига?



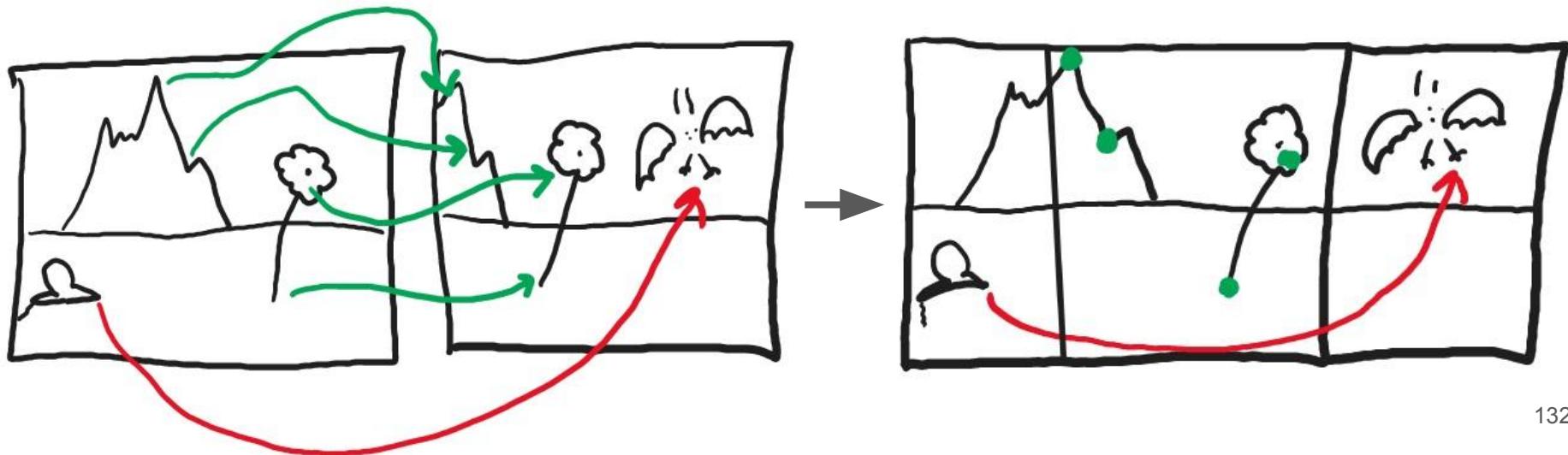
# Фильтрация сопоставлений

- 1) Попробуем как-нибудь использовать модель задачи которую решаем?
- 2) Пусть мы знаем, как одна картинка накладывается на другую. Тогда применим к точкам со второй картинки преобразование, и выбросим те матчи, точки с которых не сойдутся



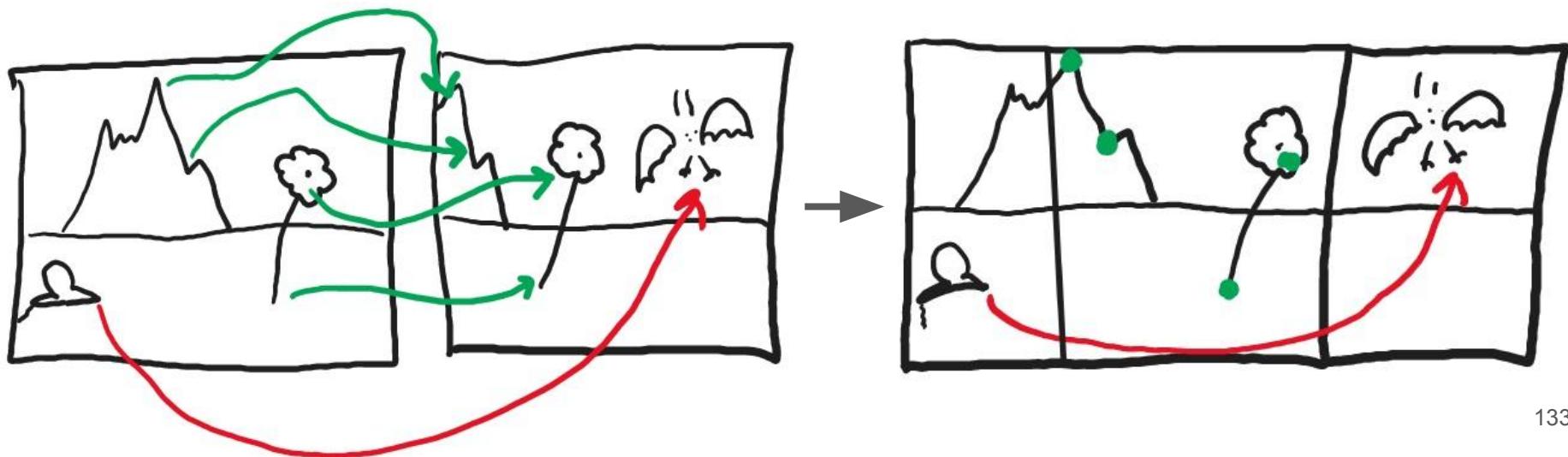
# Фильтрация сопоставлений

- 1) Попробуем как-нибудь использовать модель задачи которую решаем?
- 2) **Пусть мы знаем**, как одна картинка накладывается на другую. Тогда применим к точкам со второй картинки преобразование, и выбросим те матчи, точки с которых не сойдутся



# Фильтрация сопоставлений

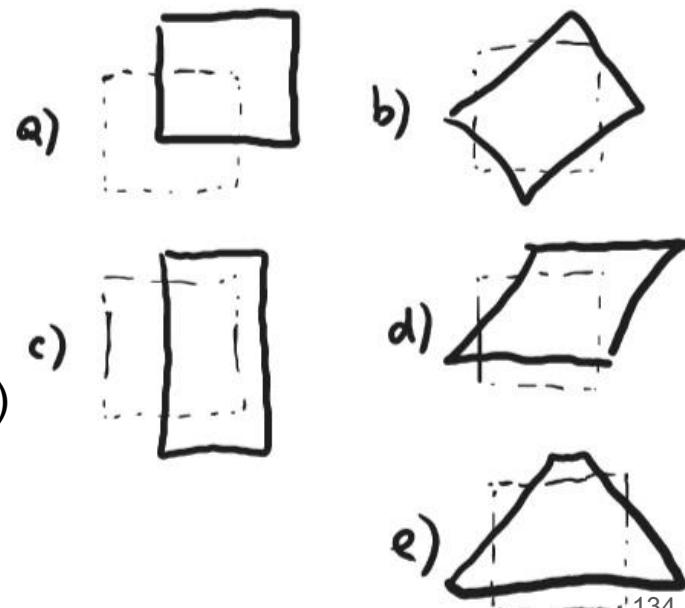
Проблема курицы и яйца, откуда узнаем как передвинуть вторую картинку?



# Справка: Гомография

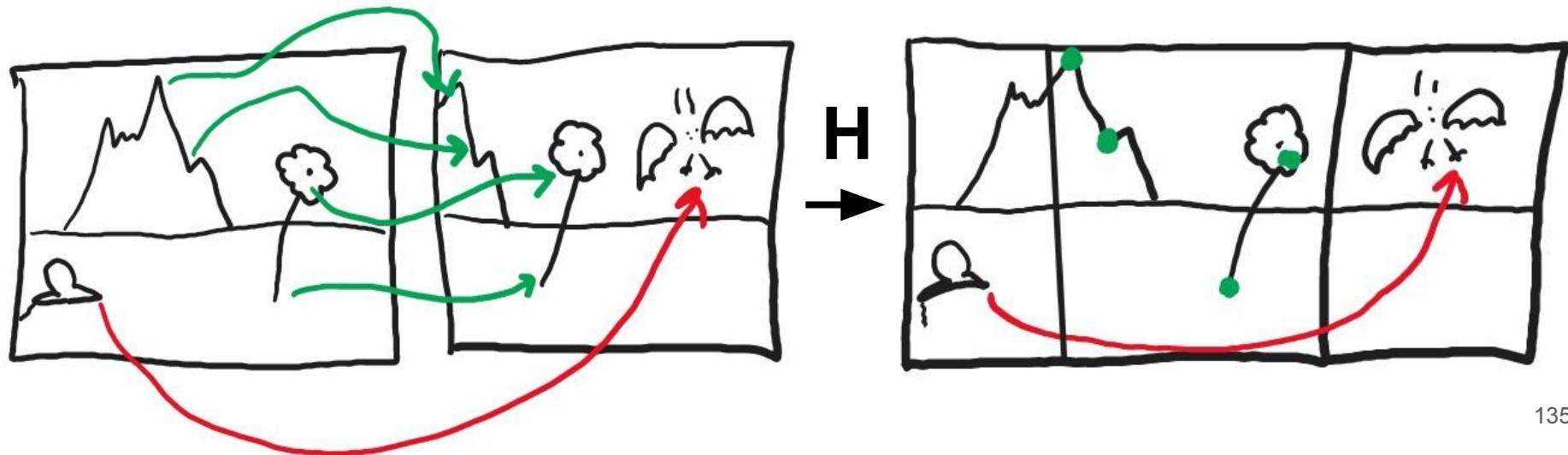
$$\hat{x} = Hx$$

- 1) **Непрерывное** преобразование пикселей со второй картинки в первую можно описать с помощью особой матрицы **H**, называемой гомографией
- 2) Гомография описывает:
  - a) Сдвиг
  - b) Поворот
  - c) Растяжение по осям X и Y
  - d) Shear
  - e) Перспективное преобразование
- 3) Переводит прямые линии в прямые
- 4) Можно рассчитать по 4 матчам (след. лекция)
- 5) Зная парные гомографии можно по цепочке склеить все картинки в одну панораму

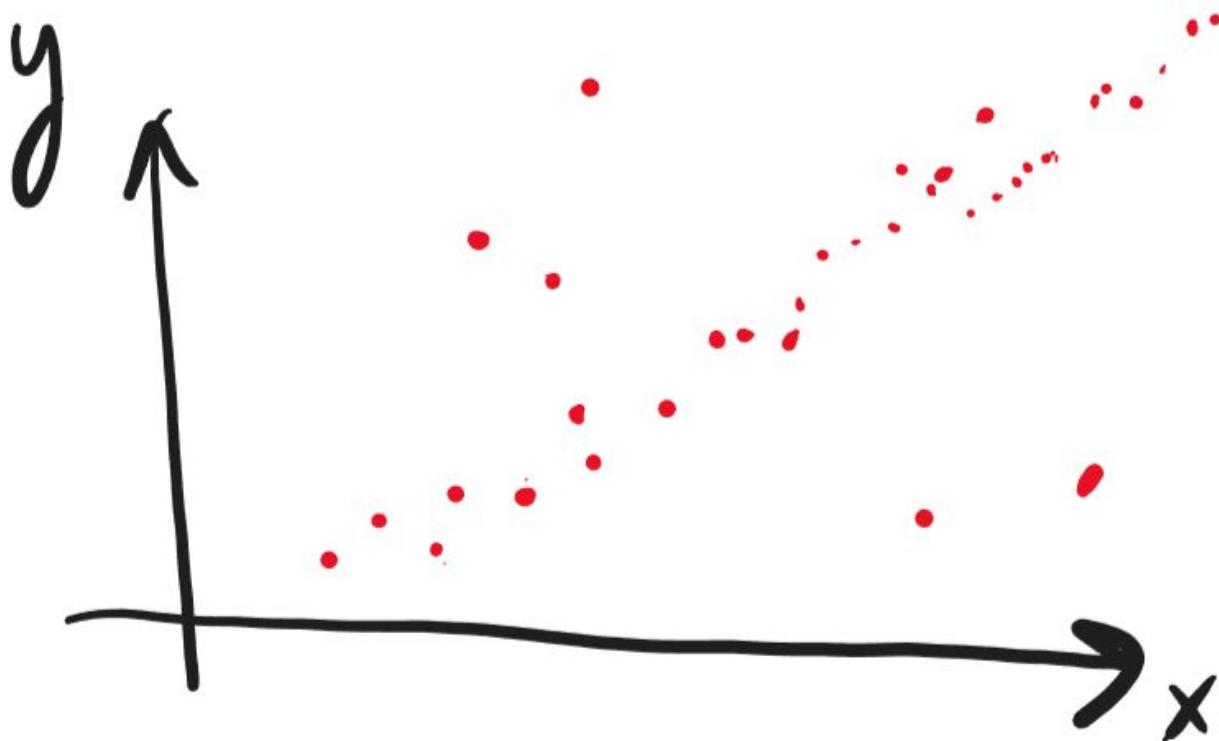


# Фильтрация сопоставлений

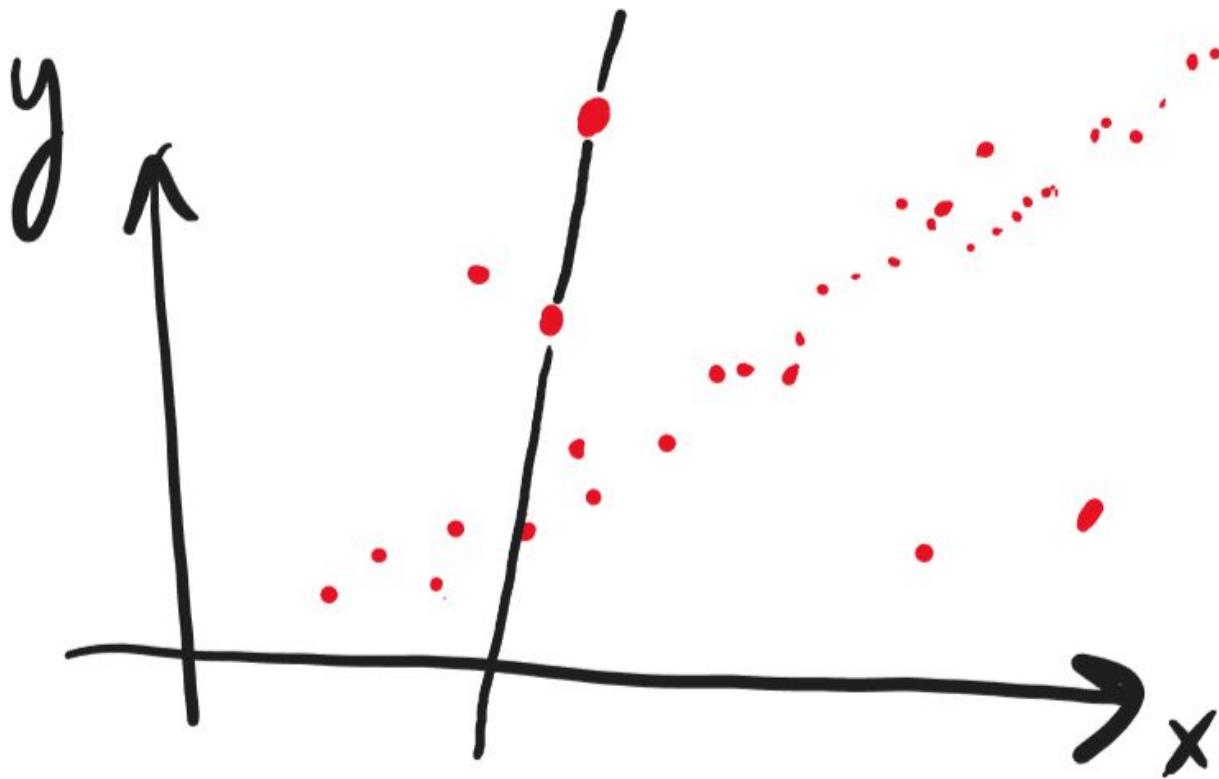
Проблема курицы и яйца, откуда узнаем гомографию?



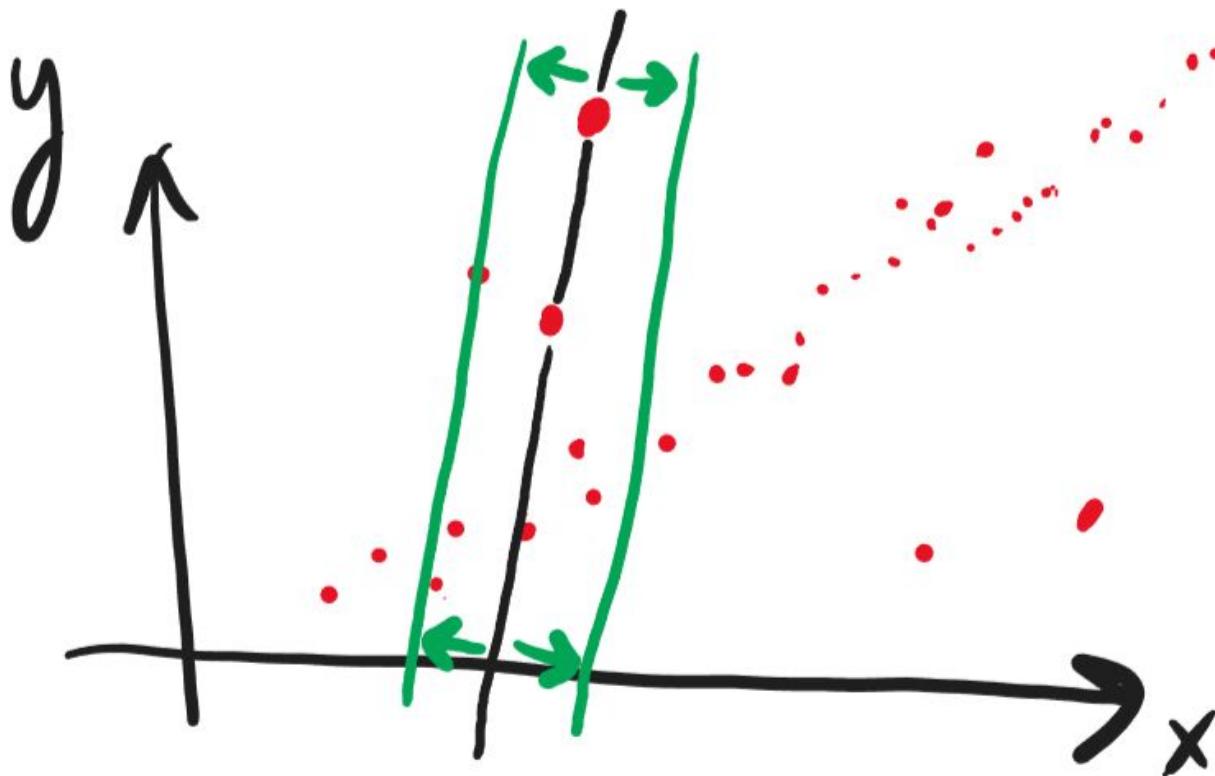
# RANSAC для поиска прямой среди точек



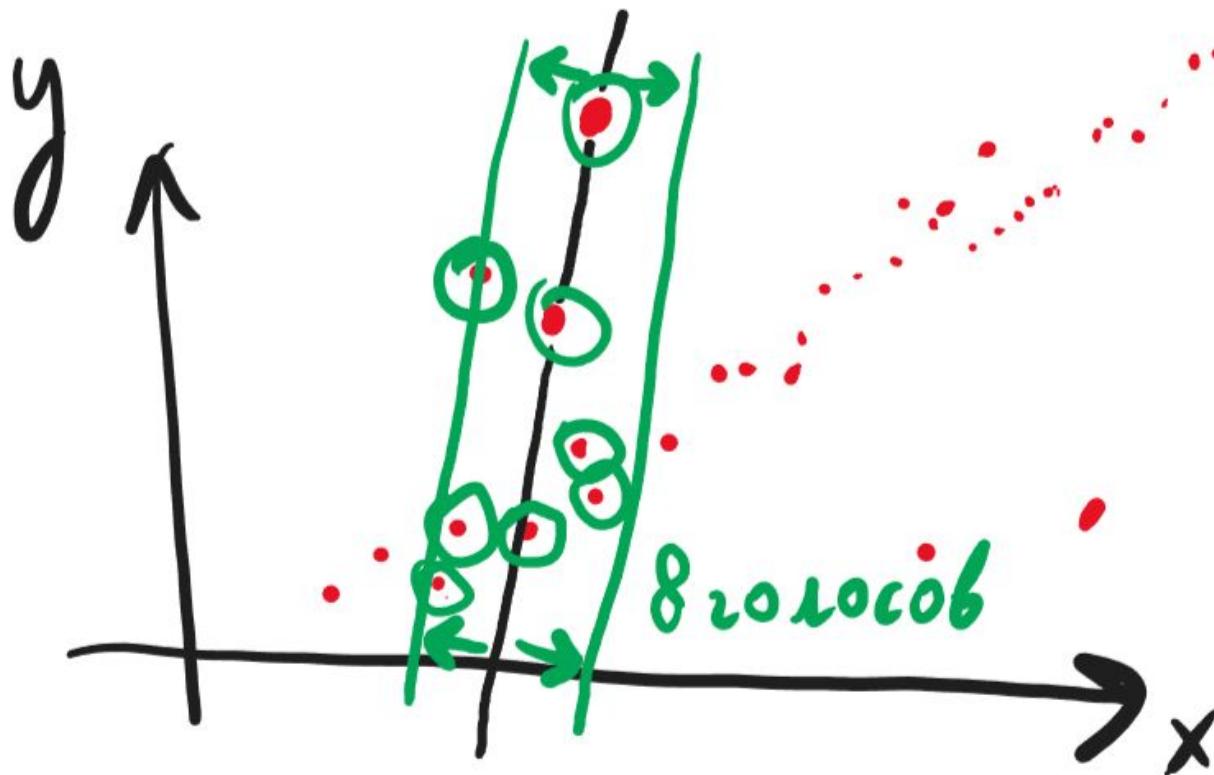
# RANSAC для поиска прямой среди точек



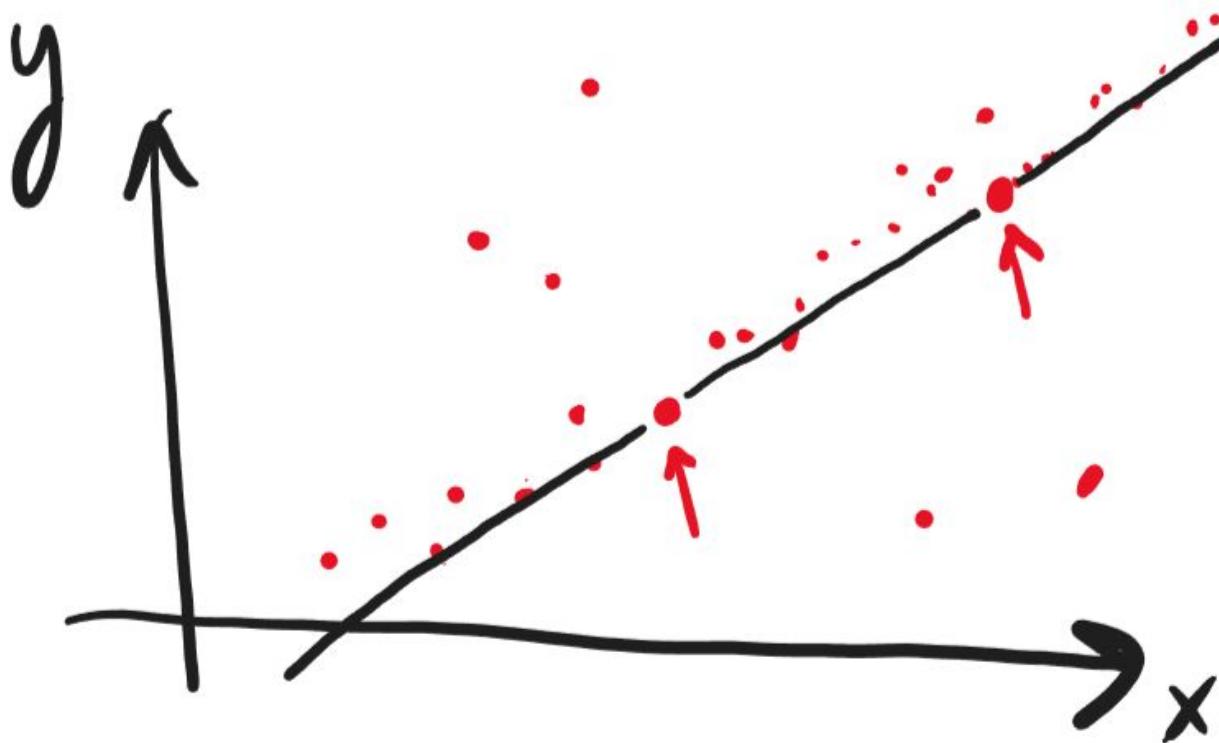
# RANSAC для поиска прямой среди точек



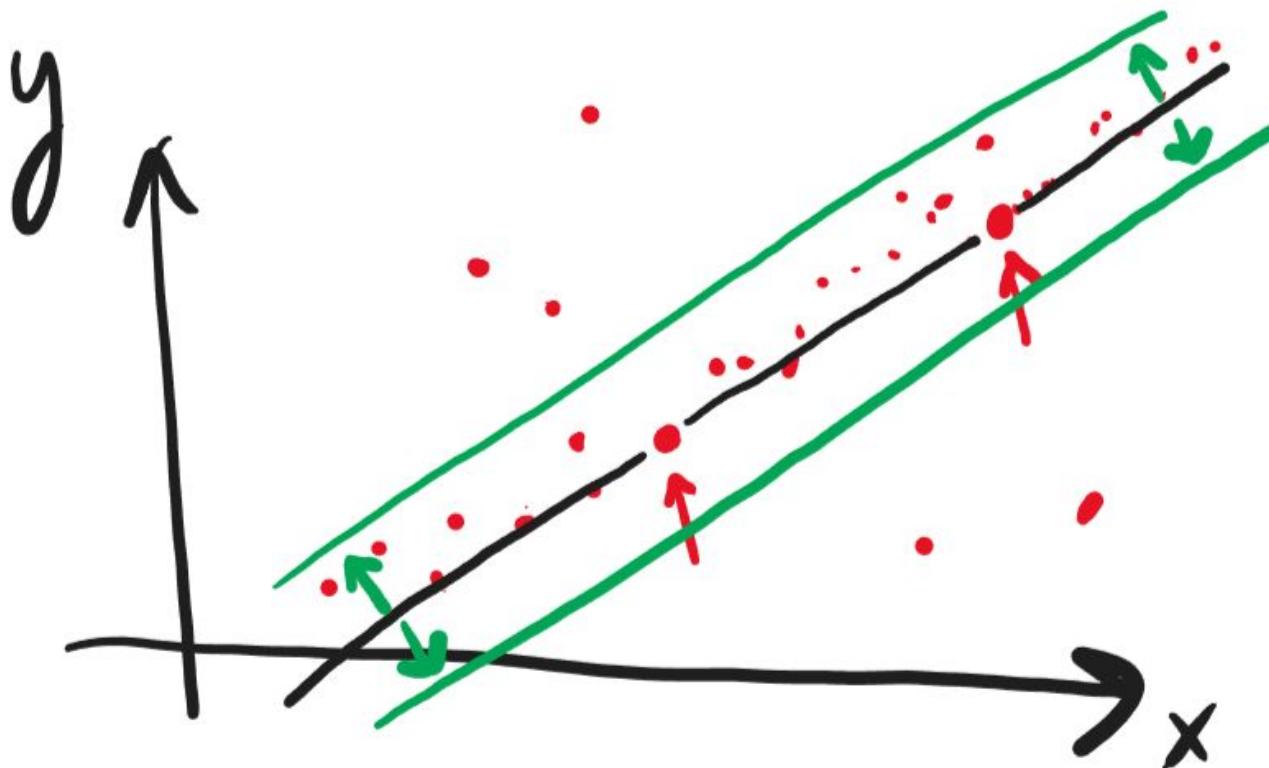
## RANSAC для поиска прямой среди точек



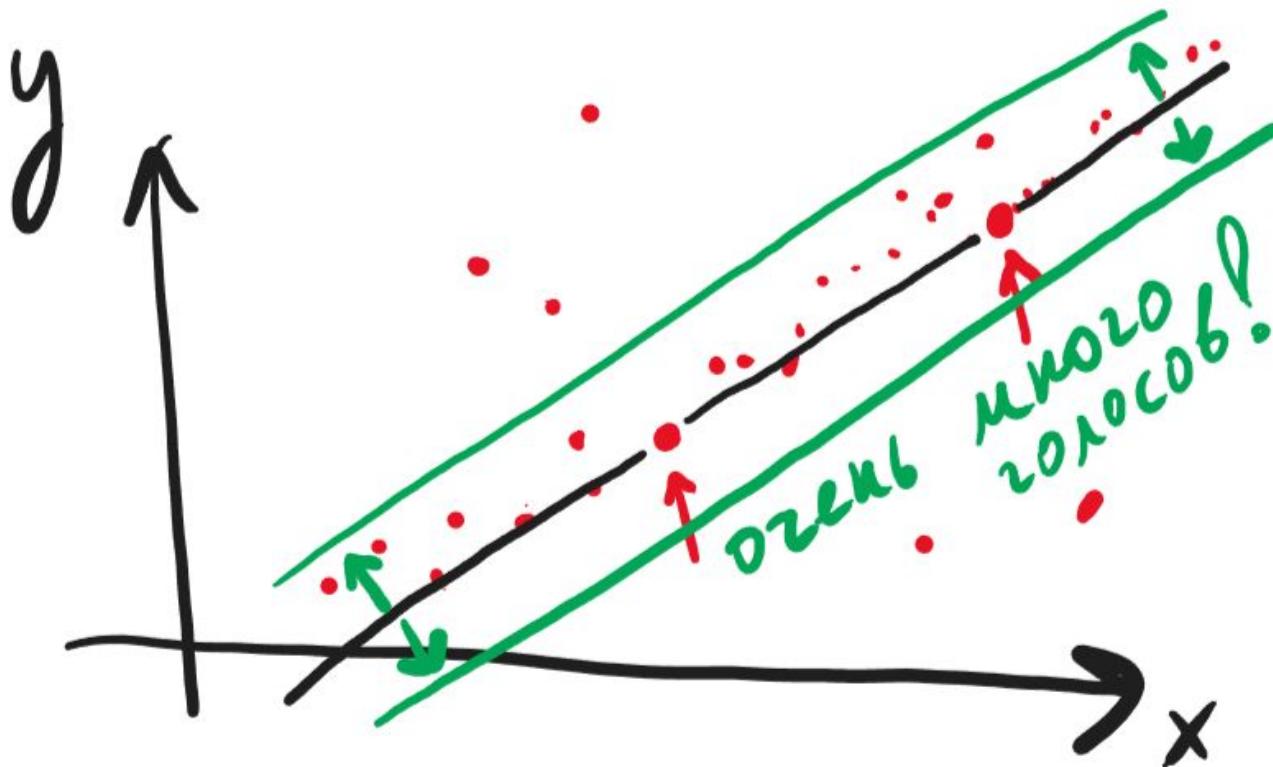
# RANSAC для поиска прямой среди точек



# RANSAC для поиска прямой среди точек



# RANSAC для поиска прямой среди точек



# RANSAC для поиска прямой среди точек

```
result_line = None
```

```
result_score = 0
```

# RANSAC для поиска прямой среди точек

```
result_line = None
result_score = 0
// количество итераций можно высчитать на основе ожидаемого процента
// выбросов
1000 итераций делаем:
A, B = randomSamples(points)
```

# RANSAC для поиска прямой среди точек

```
result_line = None
result_score = 0
// количество итераций можно высчитать на основе ожидаемого процента
// выбросов
1000 итераций делаем:
A, B = randomSamples(points)
line = fit(A, B)
```

# RANSAC для поиска прямой среди точек

```
result_line = None
```

```
result_score = 0
```

```
// количество итераций можно высчитать на основе ожидаемого процента
```

```
// выбросов
```

```
1000 итераций делаем:
```

```
A, B = randomSamples(points)
```

```
line = fit(A, B)
```

```
score = estimateScore(points, line)
```

# RANSAC для поиска прямой среди точек

```
result_line = None
result_score = 0
// количество итераций можно высчитать на основе ожидаемого процента
// выбросов
1000 итераций делаем:
A, B = randomSamples(points)
line = fit(A, B)
score = estimateScore(points, line)
if (score > result_score)
    result_line, result_score = line, score
```

# RANSAC для поиска прямой среди точек

- 1) А что изменится если бы мы искали параболу?  
(много точек лежащих около какой-то одной параболы + много выбросов)

# RANSAC для поиска прямой среди точек

- 1) А что изменится если бы мы искали параболу?  
(много точек лежащих около какой-то одной параболы + много выбросов)
- 2) А как решить задачу поиска двух парабол?  
(40% точек на одной параболе, 40% точек на другой параболе  
и 20% точек - выбросы)

# RANSAC для поиска плоскости

3) А что если точки в 3D и мы хотим найти плоскость на которой они лежат?  
Например это LIDAR-скан окружающего мира с беспилотного автомобиля и мы  
хотим найти где земля:

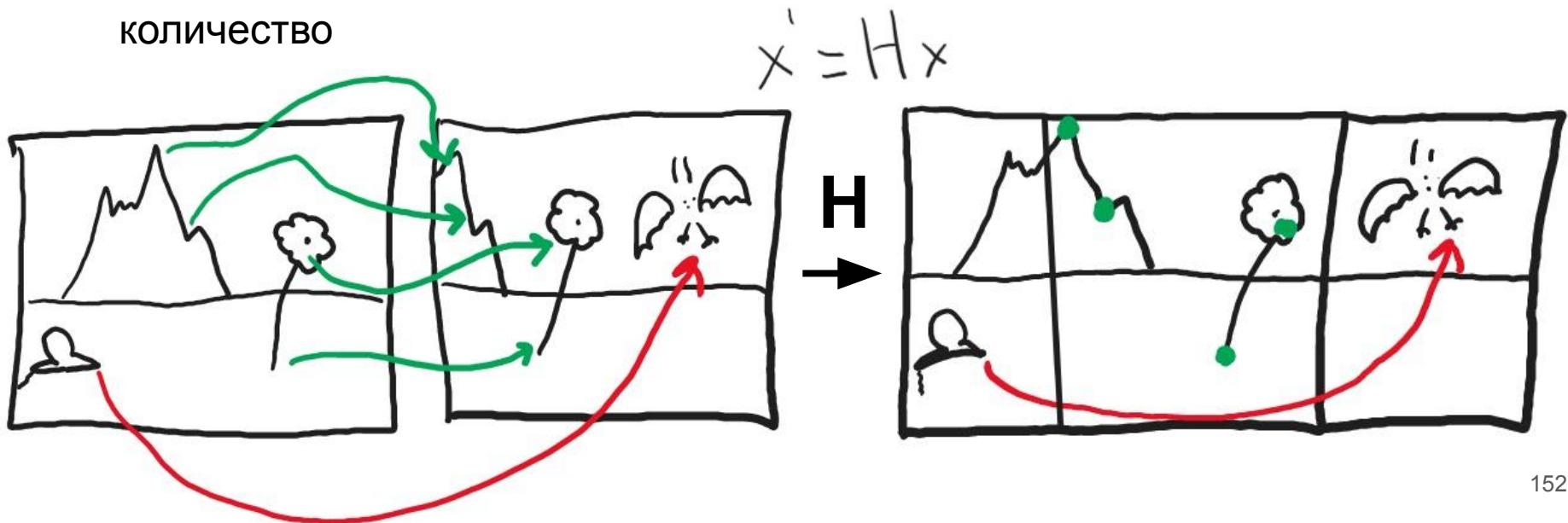


# RANSAC. Реклама

- 1) Устойчивый к шуму и выбросам способ оценить какую-то (какую угодно) модель по набору измерений, про которые не знаем, какие верные а какие - нет
- 2) Можно применять для фиттинга прямых, парабол, сложных нелинейных кривых, плоскостей, параболоидов, фотограмметрических матриц (гомографий, фундаментальных/вещественных матриц), поиска фигур в облаках точек и тд.....
- 3) Есть много вариаций, в том числе с автоматической оценкой порогов. см. MSAC, AC-RANSAC, PROSAC, MLSAC etc.
- 4) Можно накладывать ограничения, например ищем только плоскости без дырок
- 5) Хорошо параллелится, просто пишется
- 6) Сложно перехвалить...

# Фильтрация сопоставлений. RANSAC

- 1) На каждой итерации RANSAC по 4 матчам определяем гомографию и отфильтровываем выбросы
- 2) Оставляем матчи с итерации, где фильтрацию прошло наибольшее их количество



## 4. Cluster filtering



## 5. RANSAC



## 2. Сопоставление ключевых точек

### 1) **Nearest Neighbor (NN):**

- **BruteForce** (с разными преселекциями)
- **FLANN**
- **HNSW**

## 2. Сопоставление ключевых точек

### 1) **Nearest Neighbor (NN):**

- **BruteForce** (с разными преселекциями)
- **FLANN**
- **HNSW**

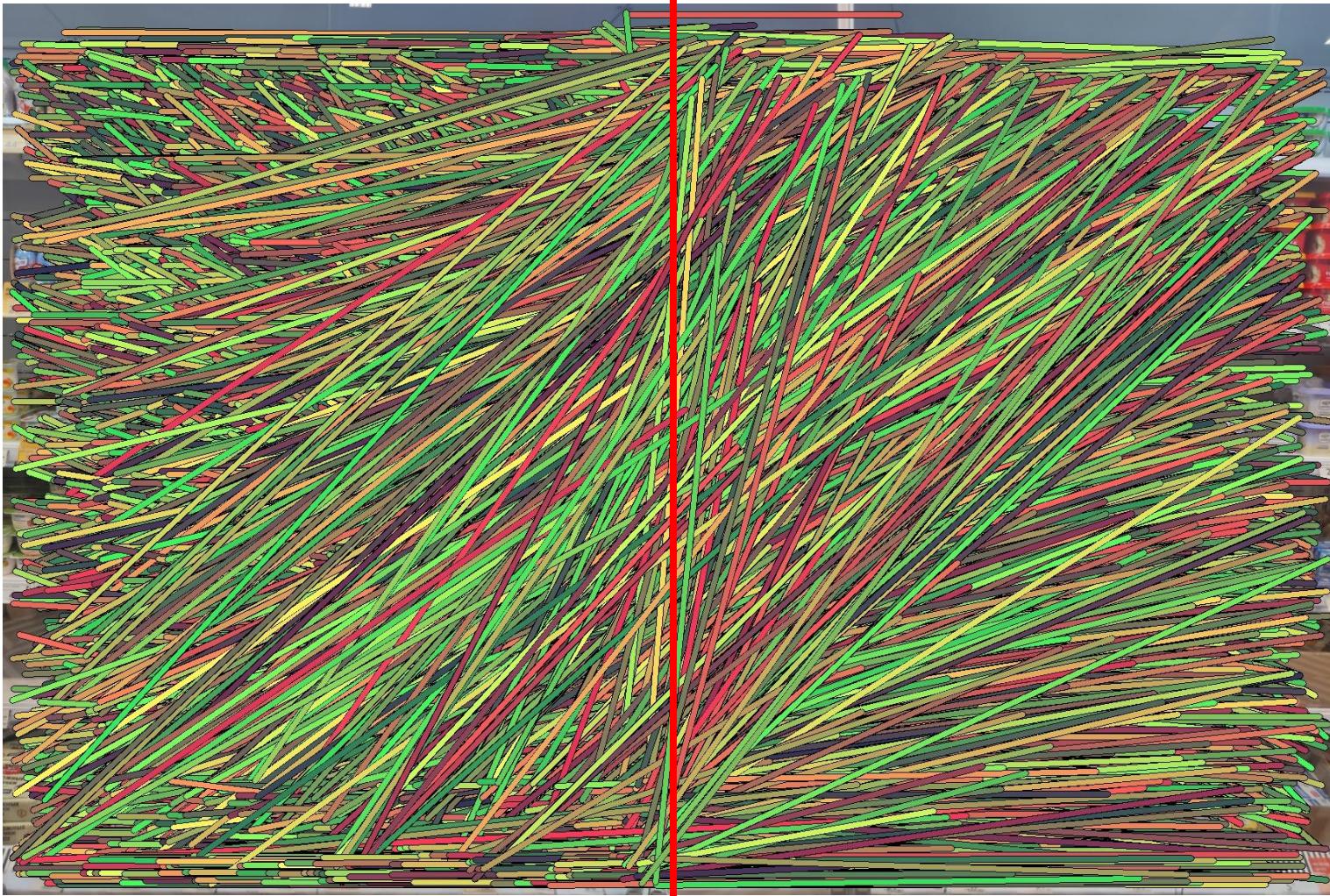
### 2) **K-ratio test**

### 3) **Left-right check**

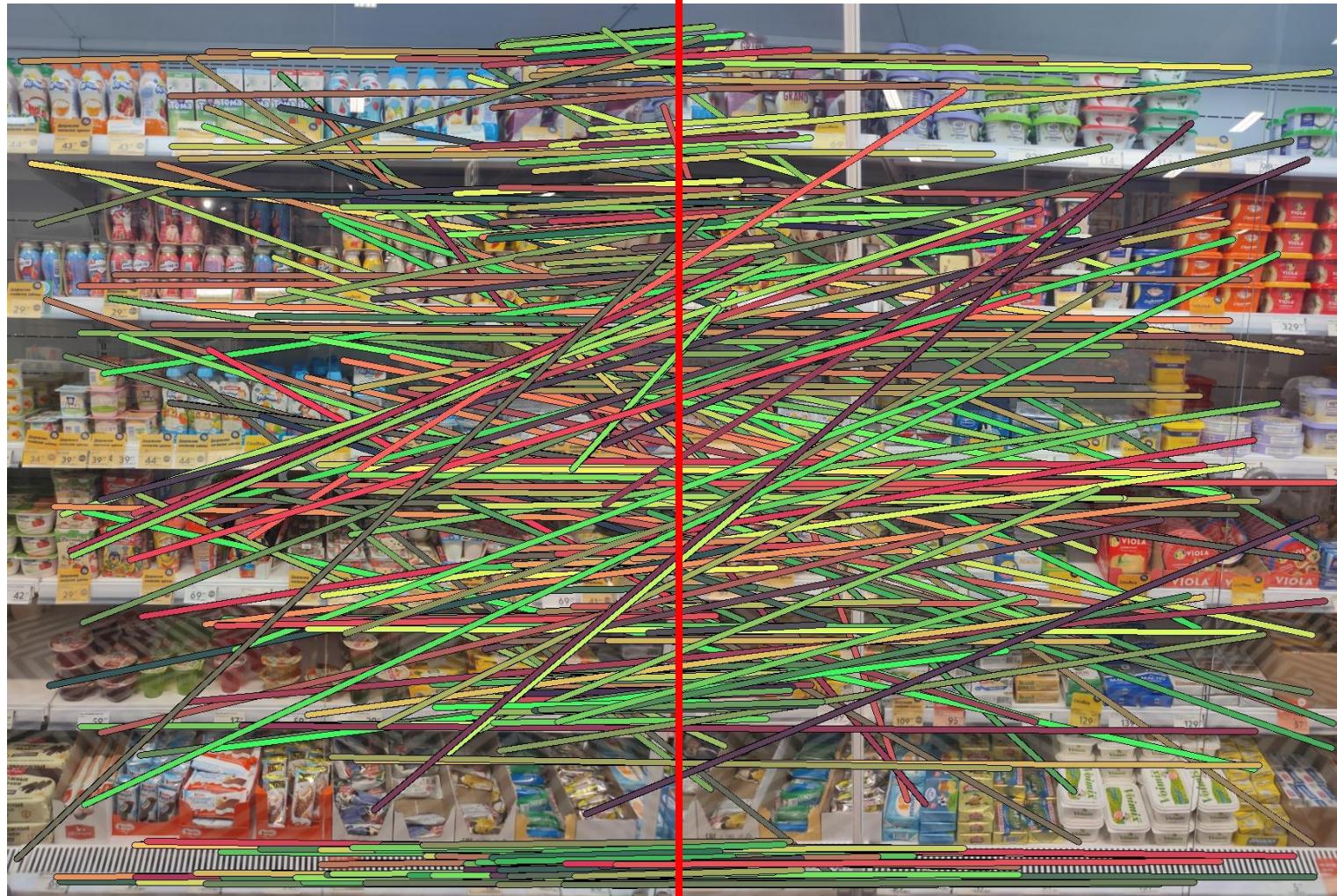
### 4) **Cluster filtering**

### 5) **RANSAC**

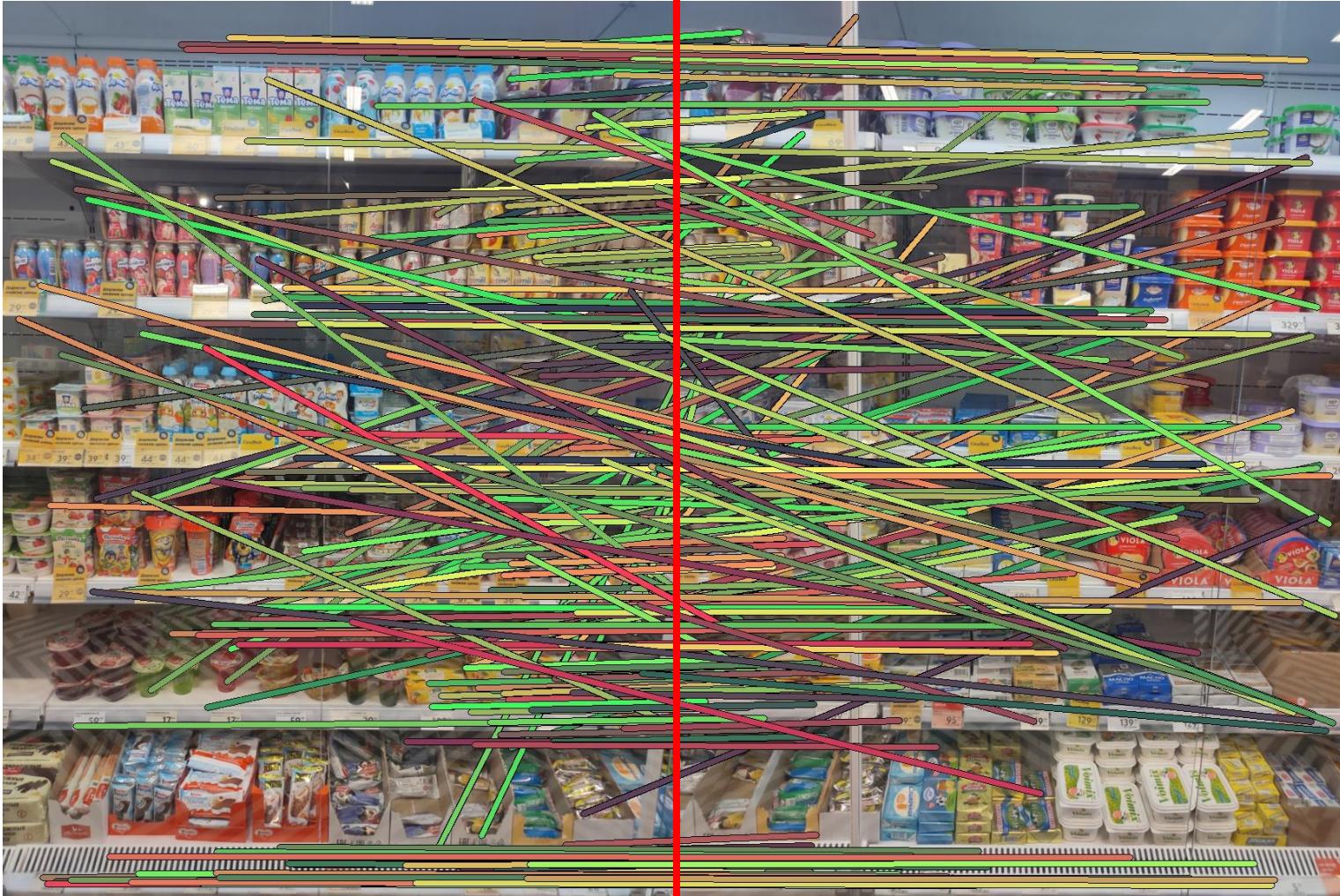
# 1. Nearest Neighbors



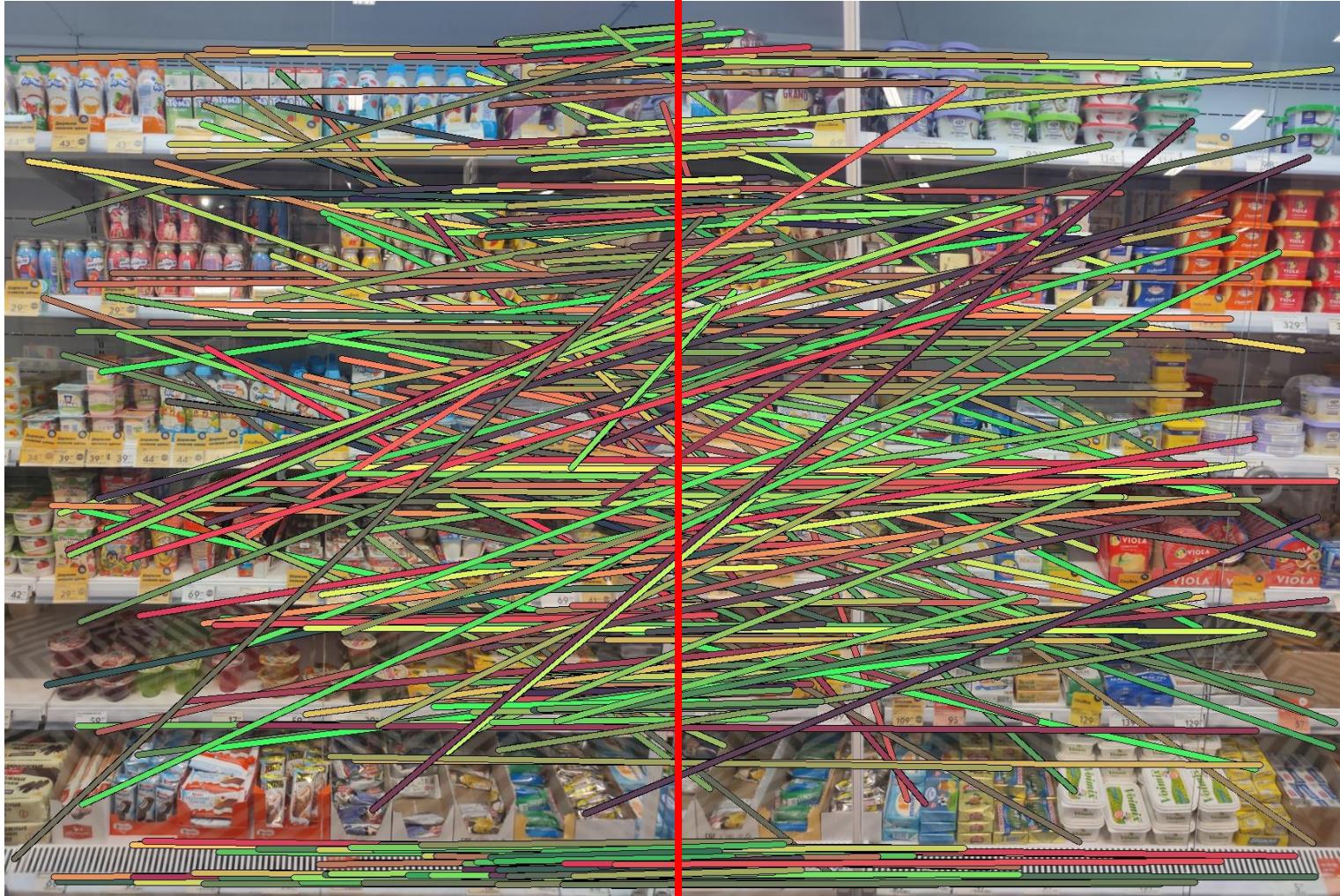
## 2. K-ratio test



## 2. K-ratio test (right-to-left)



## 2. K-ratio test



### 3. Left-right check



## 4. Cluster filtering

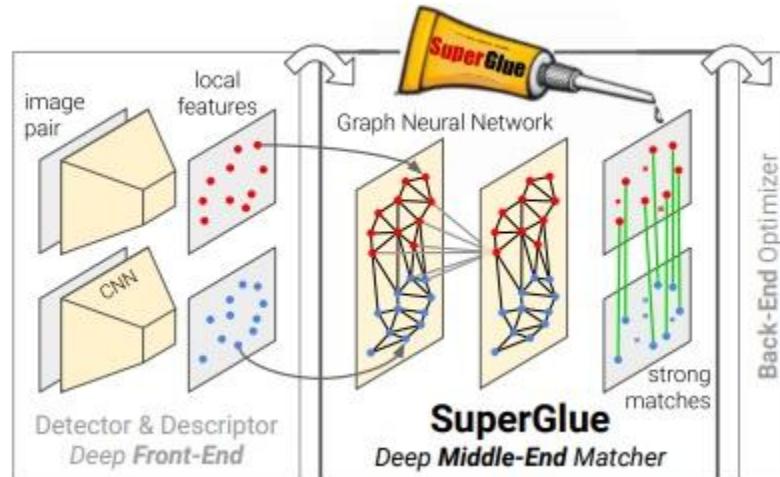


## 5. RANSAC



# Ссылки

- 1) [GMS matcher](#) (похоже на cluster filtering)
- 2) [SuperGlue](#) (нейронка, расширение cluster filtering, следим за паттернами)
- 3) [Бенчмарк](#) современных детекторов и дескрипторов и сравнение с SIFT
- 4) [Ежегодный большой контест](#) по feature matching



Вопросы?



Полярный Николай  
[polarnick239@gmail.com](mailto:polarnick239@gmail.com)