MLRF Lecture 03

J. Chazalon, LRDE/EPITA, 2019

Descriptors matching and indexing

Lecture 03 part 04

Matching

Introduction

Given those keypoints in image 1, what are the more similar ones in image 2?

This is a **nearest neighbor problem** in **descriptor space**.
This is also a **geometrical problem** in **coordinate space**.

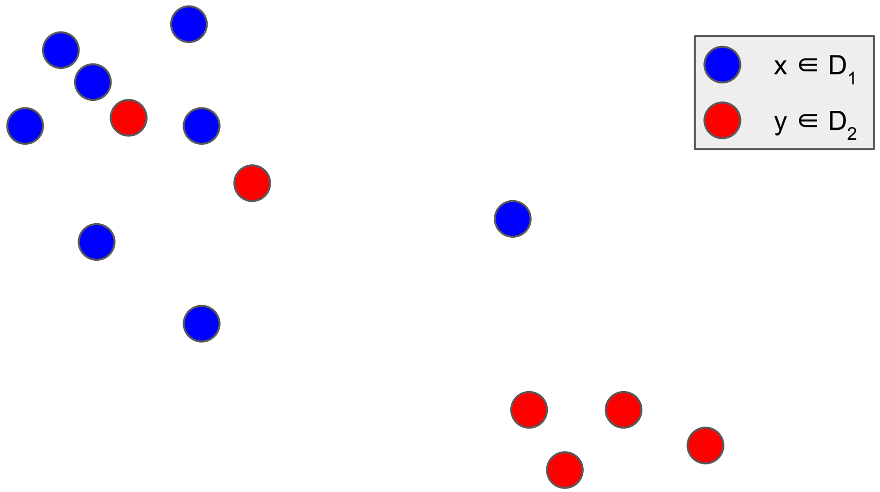
Let us start with the matching problem.

Need a **distance/norm**: depend on the descriptor

- Distribution? Statistics?
- Data type?
  - Float, integers: Euclidean, cosine…
  - Binary: Hamming…

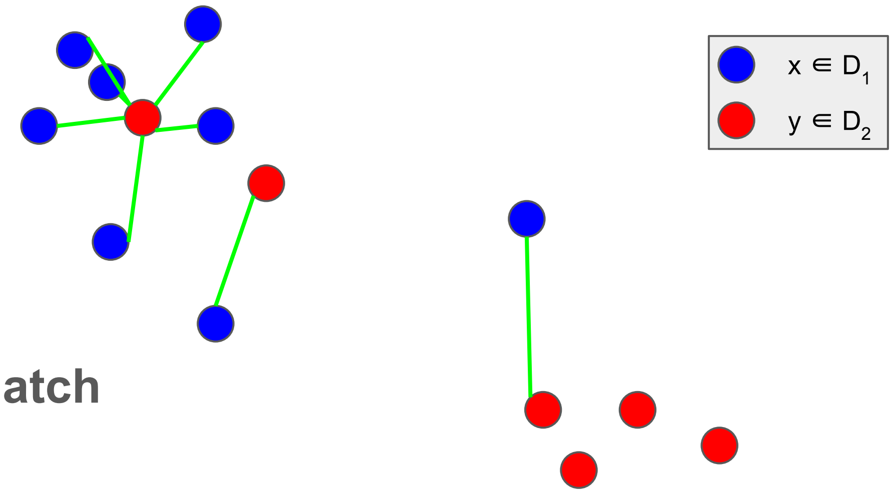
1-way matching

For each x_i in the set of descriptors D_1, find the closest element y_i in D_2.



1-way matching

For each x_i in the set of descriptors D_1, find the closest element y_i in D_2.

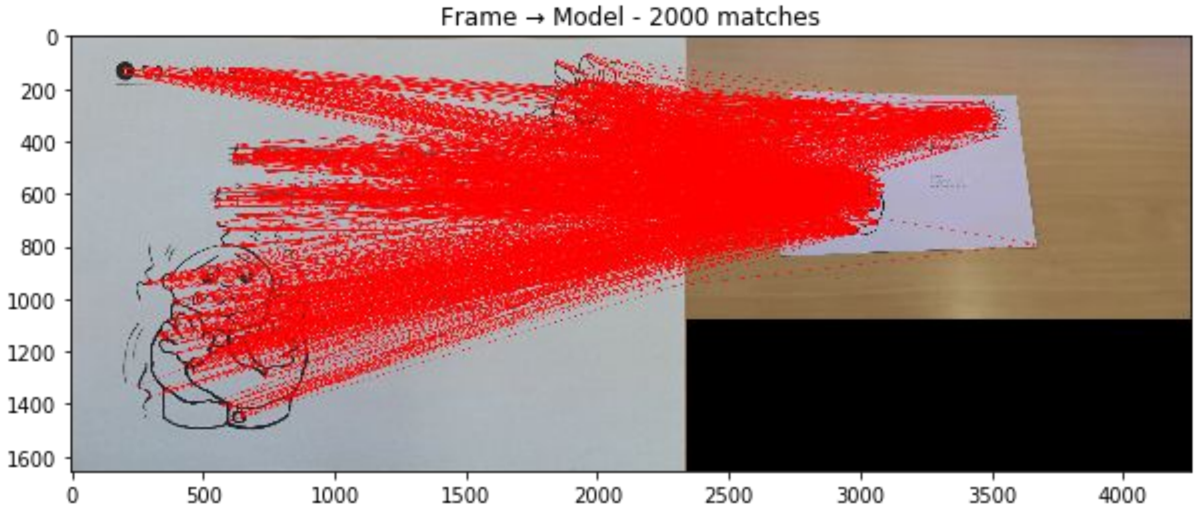


We have a match **m(x,y) for each x_i**

Many y in D_2 may **not belong to any match**

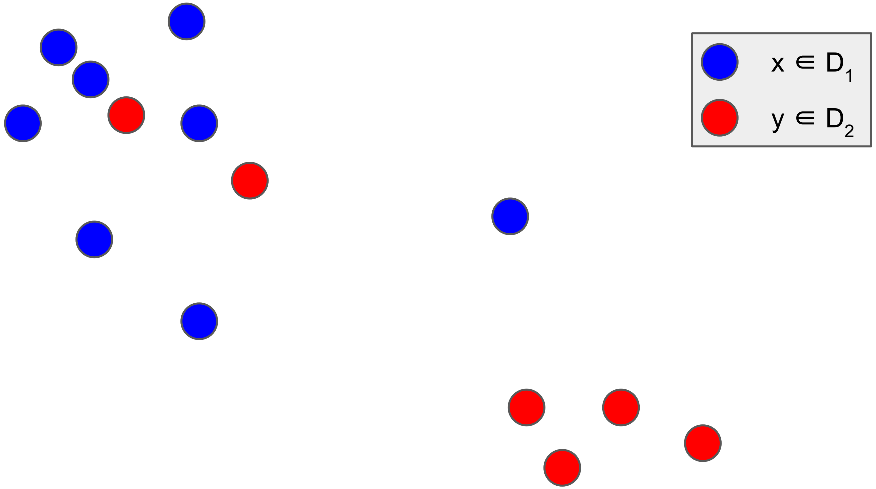
1-way matching

Example from next practice session



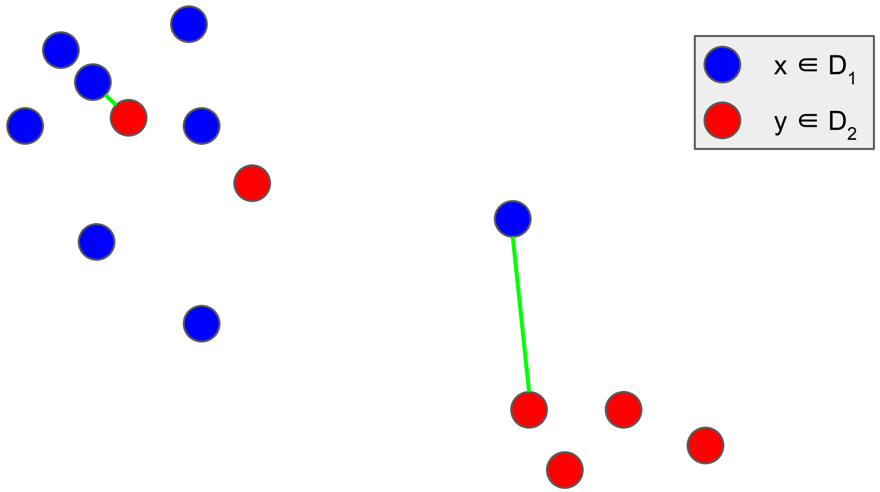
Symmetry test *aka* cross check *aka* 2-way matching

For each x_i in the set of descriptors D_1, find the closest element y_i in D_2 such as x_i is also the closest element to y_i.



Symmetry test *aka* cross check *aka* 2-way matching

For each x_i in the set of descriptors D_1, find the closest element y_i in D_2 such as x_i is also the closest element to y_i.

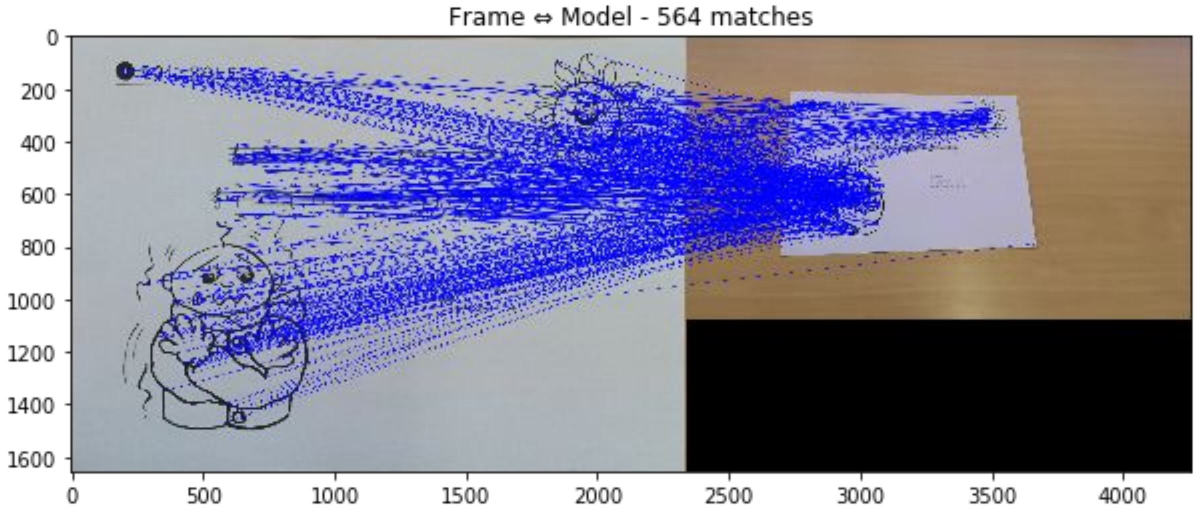


Filters a lot of noise.

Costly to compute.

Symmetry test *aka* cross check *aka* 2-way matching

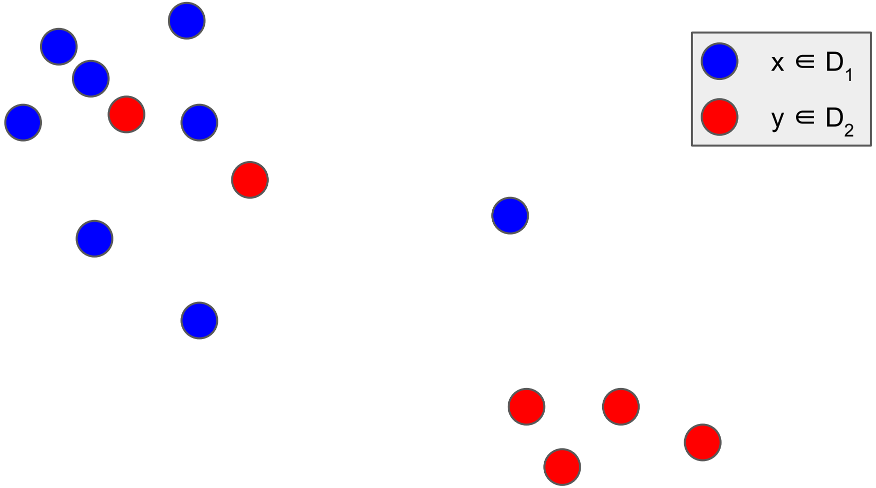
Example from next practice session



Ratio test

For each x_i in the set of descriptors D_1, find the 2 closest elements y_i and y_j in D_2.

Keep the match $\mathbf{m(x_i, y_i)}$ *only if* $\mathbf{dist(x_i, y_i) < RATIO * dist(x_i, y_j)}$

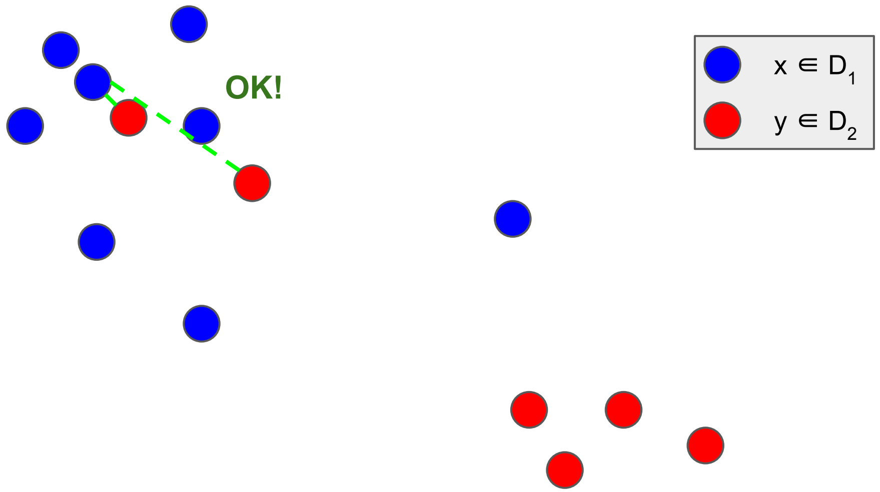


Ratio test

For each x_i in the set of descriptors D_1, find the 2 closest elements y_i and y_j in D_2.

Keep the match $\mathbf{m(x_i, y_i)}$ *only if* $\mathbf{dist(x_i, y_i) < RATIO * dist(x_i, y_j)}$

Assume RATIO = ⅔

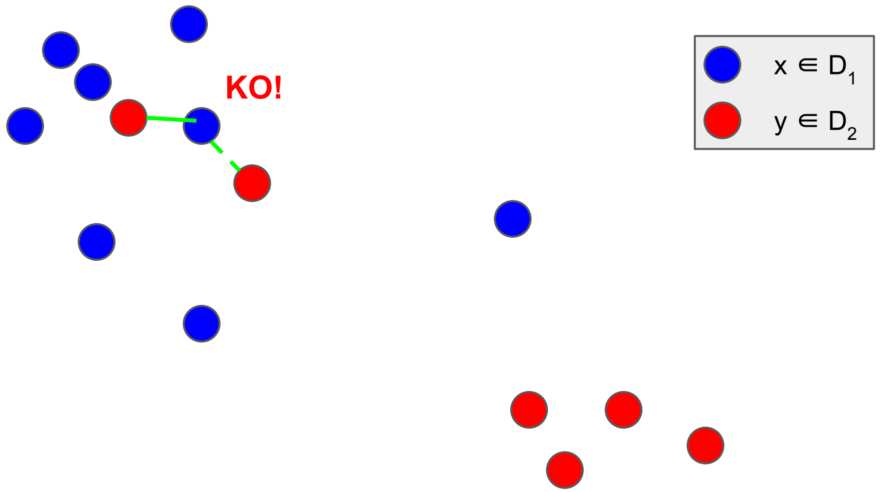


Ratio test

For each x_i in the set of descriptors D_1, find the 2 closest elements y_i and y_j in D_2.

Keep the match $\mathbf{m(x_i, y_i)}$ *only if* $\mathbf{dist(x_i, y_i) < RATIO * dist(x_i, y_j)}$

Assume RATIO = ⅔

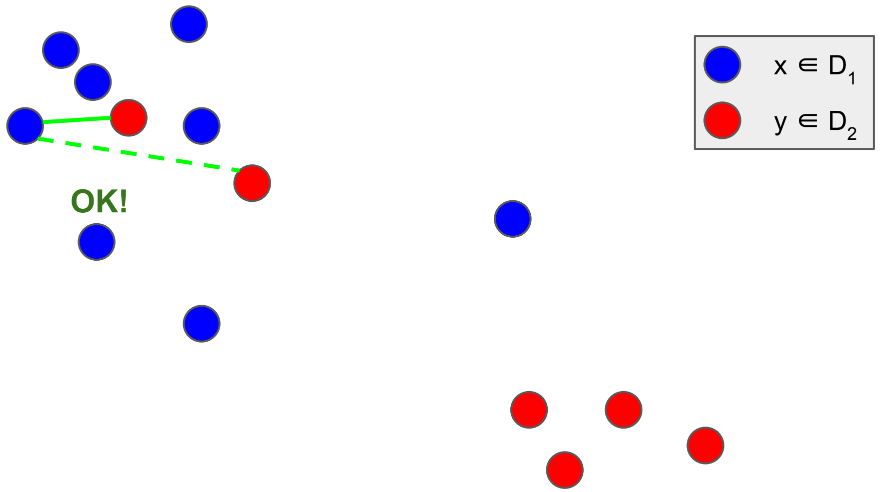


Ratio test

For each x_i in the set of descriptors D_1, find the 2 closest elements y_i and y_j in D_2.

Keep the match **m(x_i, y_i)** *only if* **dist(x_i, y_i) < RATIO * dist(x_i, y_j)**

Assume RATIO = ⅔

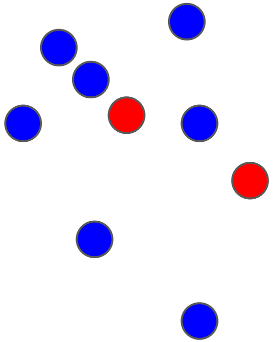


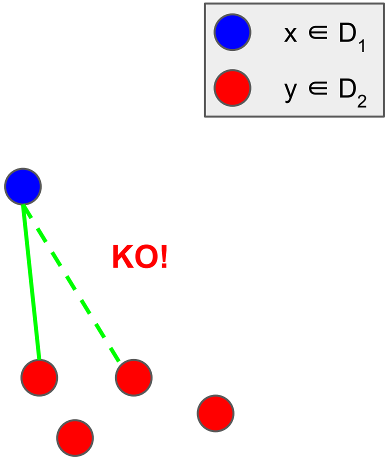
Ratio test

For each x_i in the set of descriptors D_1, find the 2 closest elements y_i and y_j in D_2.

Keep the match $\mathbf{m(x_i, y_i)}$ *only if* $\mathbf{dist(x_i, y_i) < RATIO * dist(x_i, y_j)}$

Assume RATIO = ⅔





Good filter which **removes ambiguous matches.**

Like 1-way matching, **potential y duplicates** in matches

Can be made symmetrical

Ratio test: calibrate the ratio

Adjust it on a training set!

For each correct/incorrect match in your annotated database, plot the next to next closest distance PDF.

What is a good ratio in D. Lowe's experiment? *→*

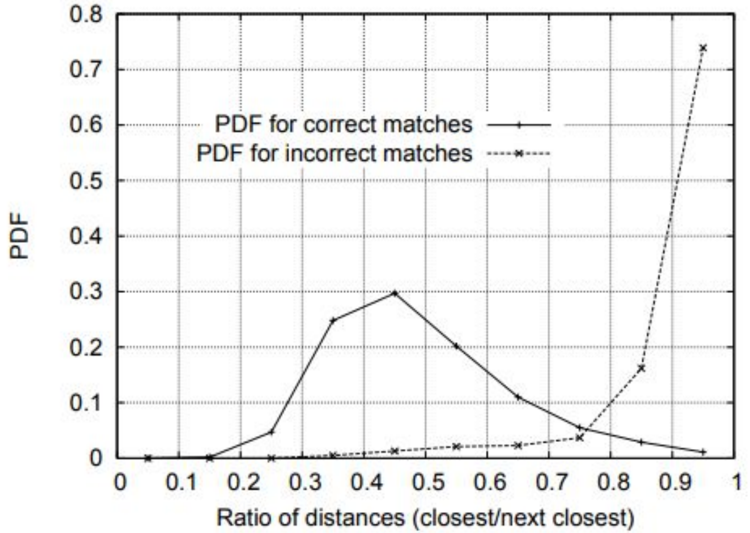
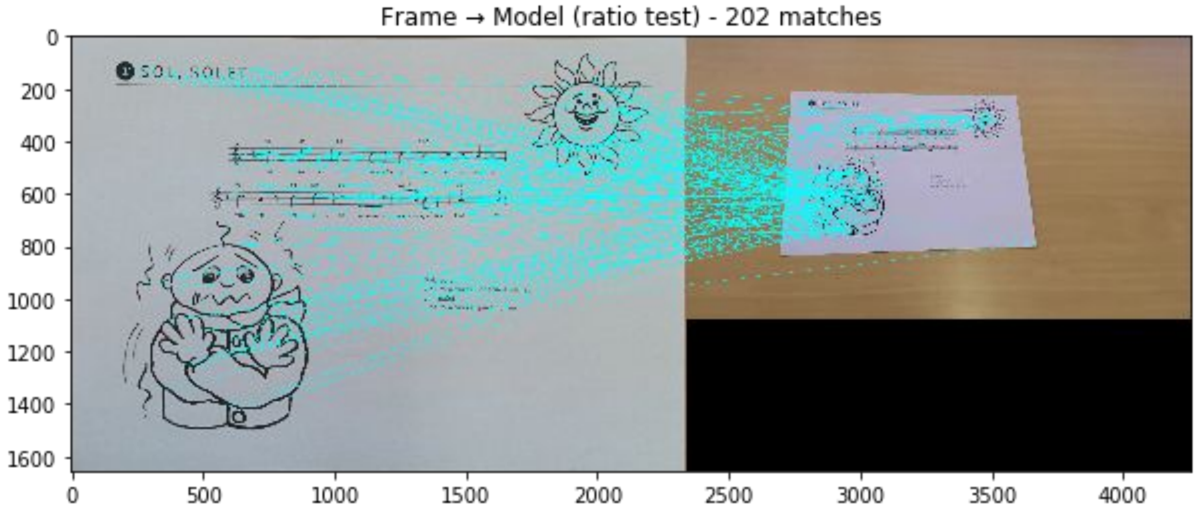


Figure 11: The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. Using a database of 40,000 keypoints, the solid line shows the PDF of this ratio for correct matches, while the dotted line is for matches that were incorrect.

David G. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.

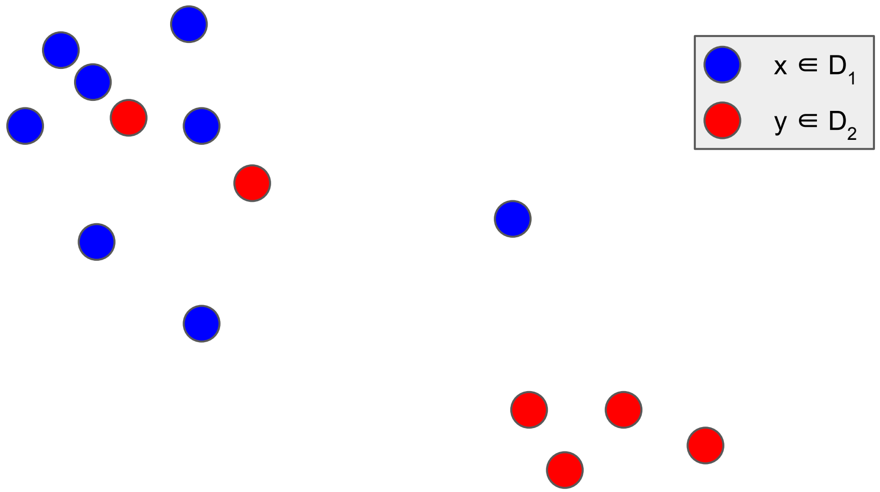
Ratio test

Example from next practice session



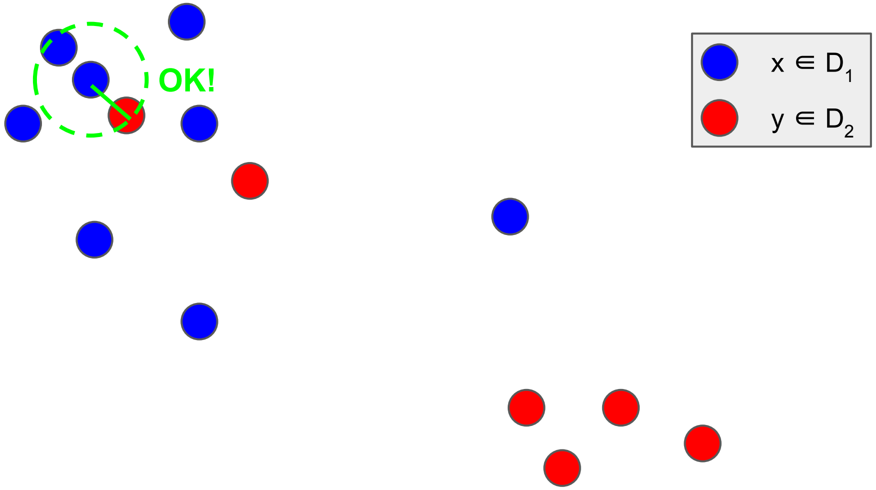
Radius threshold

For each x_i in the set of descriptors D_1, find the closest element y_i in D_2 and make sure **dist(x_i, y_i) < RADIUS**



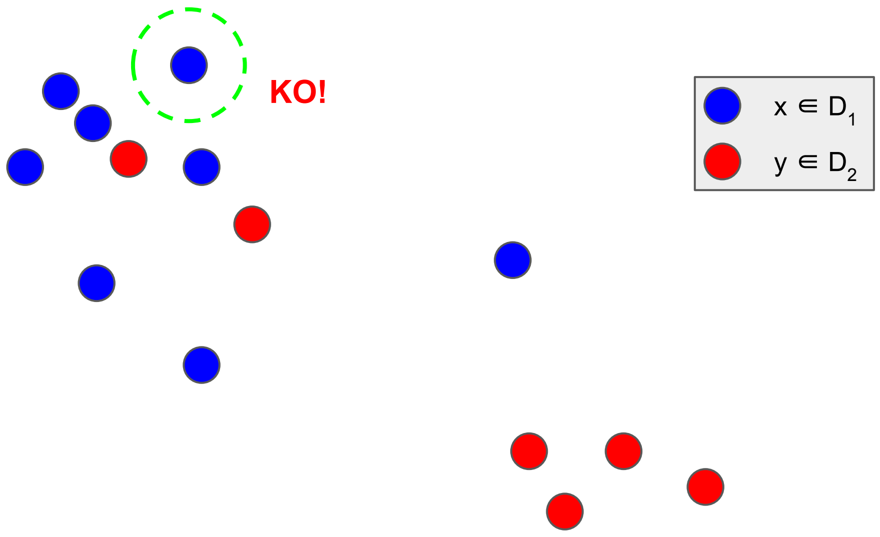
Radius threshold

For each x_i in the set of descriptors D_1, find the closest element y_i in D_2 and make sure **dist(x_i, y_i) < RADIUS**



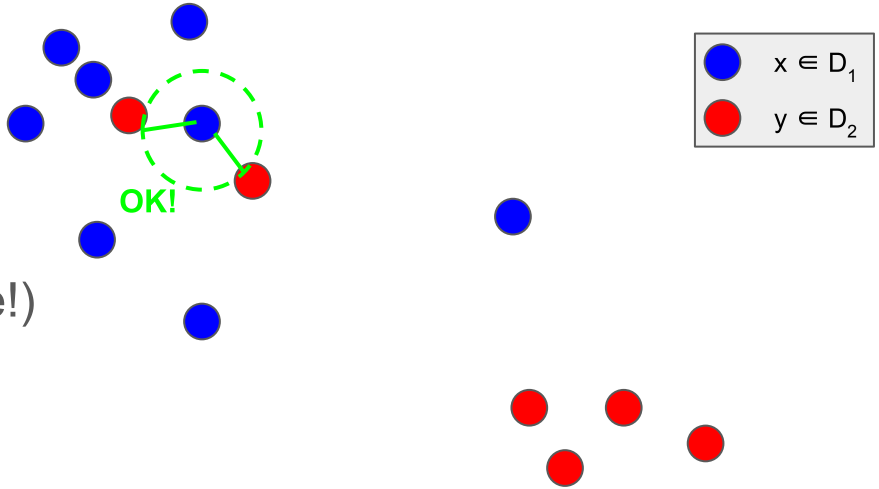
Radius threshold

For each x_i in the set of descriptors D_1, find the closest element y_i in D_2 and make sure **dist(x_i, y_i) < RADIUS**



Radius threshold

For each x_i in the set of descriptors D_1, find the closest element y_i in D_2 and make sure **dist(x_i, y_i) < RADIUS**



May allow *multiple good matches*

Harder to calibrate
(1 absolute value for all descriptor space!)

Good trick with “background model”
= set D_3 with **only incorrect matches**

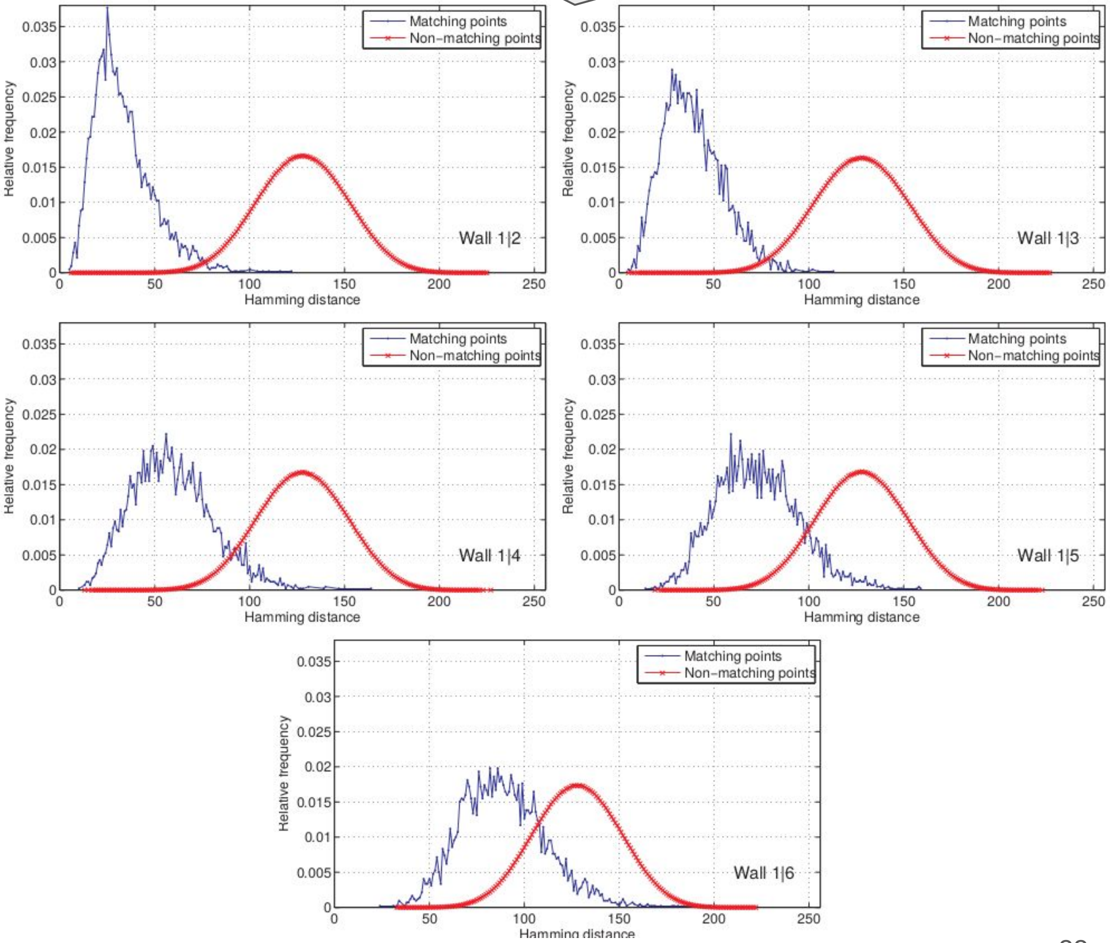
But how to query within a certain distance efficiently? **Indexing!**

Radius threshold

From BRIEF paper.

If we have a background model which give us the red curve for each case (not knowing the blue one), **can we choose a good radius?**

Beware: absolute values here!



M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “Brief: Binary robust independent elementary features,” in European conference on computer vision, 2010, pp. 778–792.

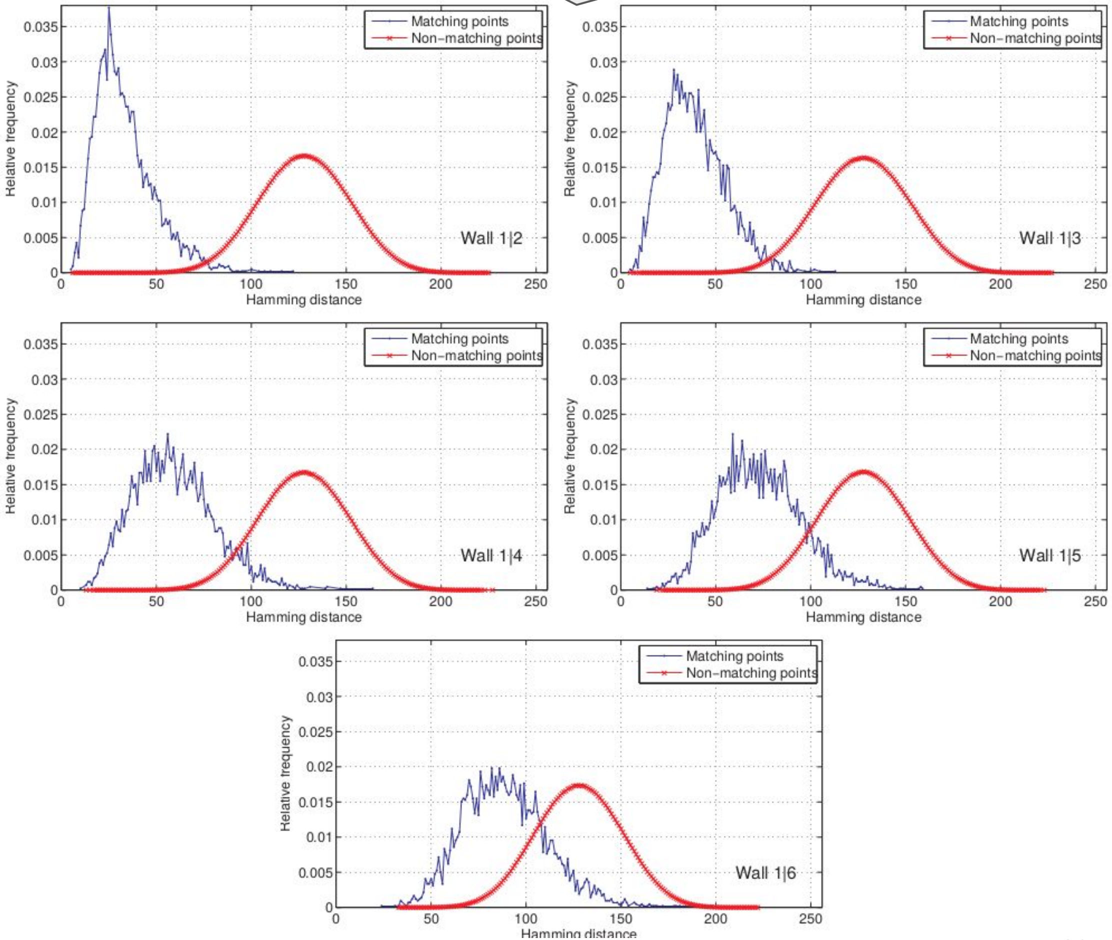
Beware: absolute values here!

Radius threshold

From BRIEF paper.

If we have a background model which give us the red curve for each case (not knowing the blue one), **can we choose a good radius?**

75 seems good here!



M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “Brief: Binary robust independent elementary features,” in European conference on computer vision, 2010, pp. 778–792.

Radius threshold

Example from next practice session

[Missing because not useful for this session]

[and tricky to handle multiple good matches]

Geometric validation

What about the coordinates of the keypoints?

Once we have a list of matches $\mathbf{m(x_i, y_i)}$,
we can check whether the coordinates of the keypoints of the matched descriptors describe a consistent mapping from one position to another.

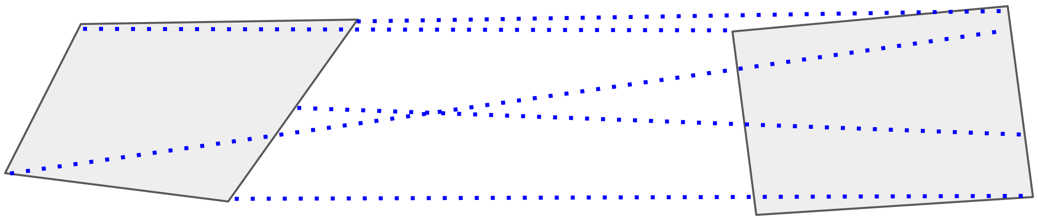


Image space!

Geometric validation

What about the coordinates of the keypoints?

Once we have a list of matches $\mathbf{m}(\mathbf{x}_i, \mathbf{y}_i)$,
we can check whether the coordinates of the keypoints of the matched descriptors describe a consistent mapping from one position to another.

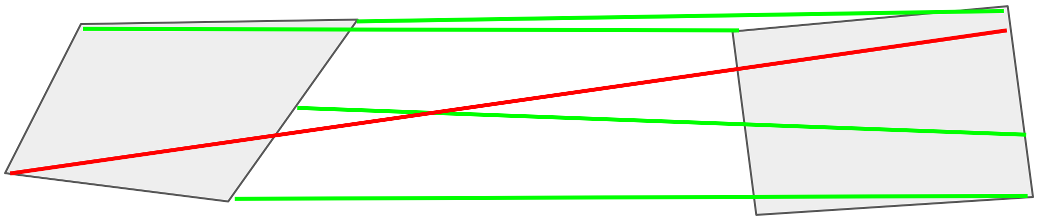
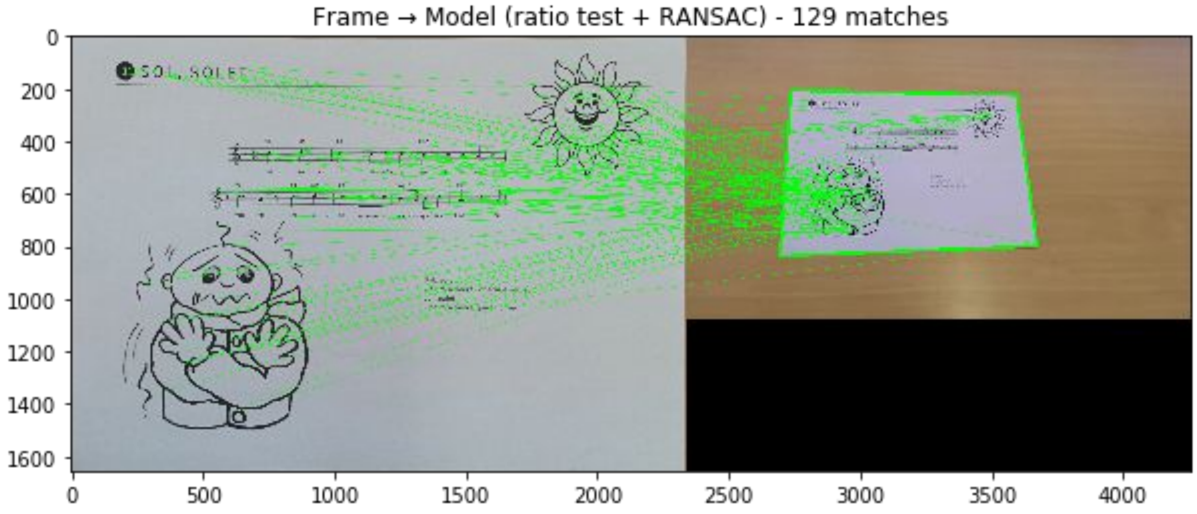


Image space!

Geometric validation

Example from next practice session



Geometric validation

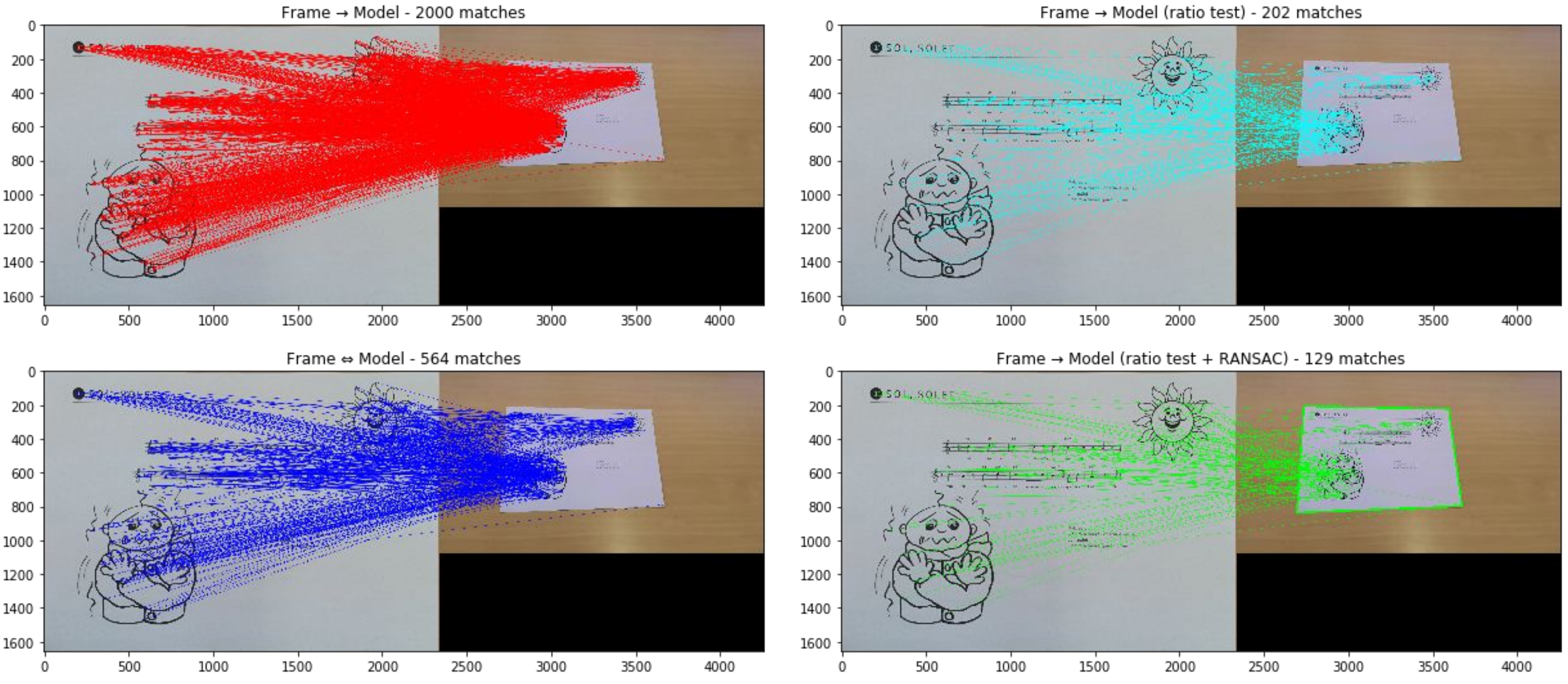
How to check the consistency of the transformation?

Difference classes of transformation.

Different methods to estimate them and check which matches agree and disagree.

⇒ Next lecture parts.

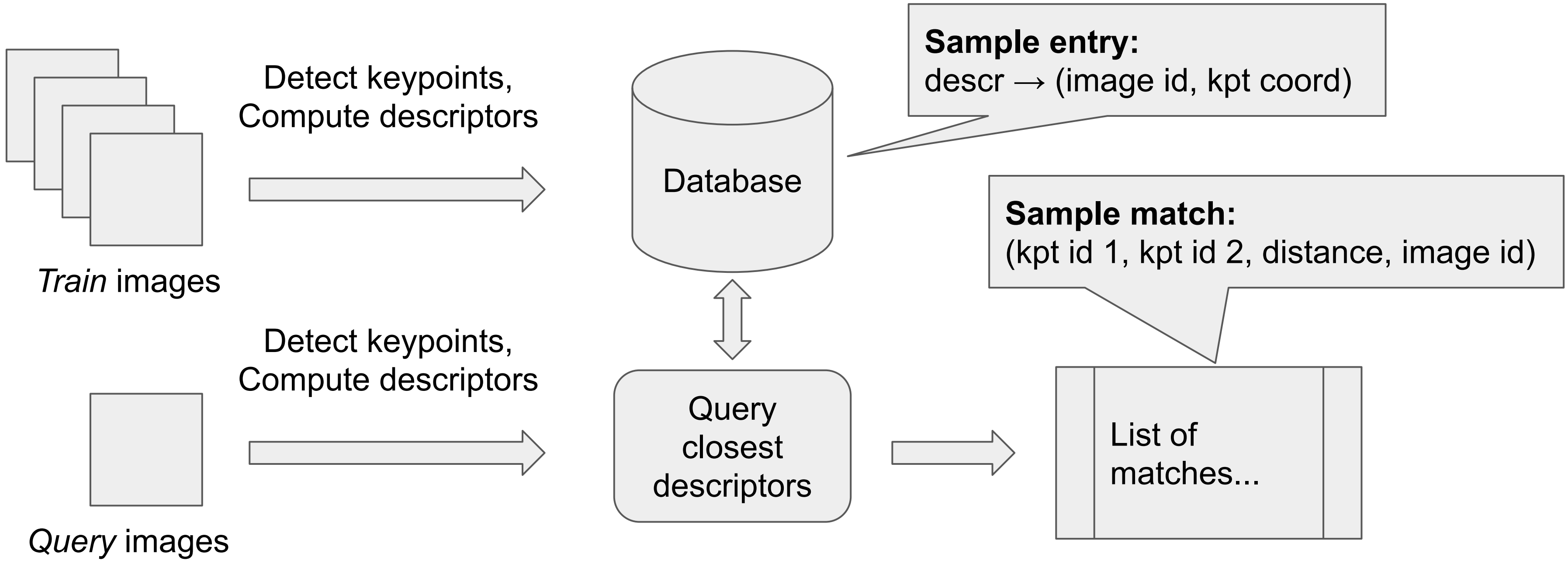
Summary of matching techniques



Indexing

Indexing pipeline

Often, we have a database of images, and we want to find an object from it.



Bruteforce matching *aka* linear matching

Simply scan all data and keep the closest elements.

Does not scale to large databases, but can be faster on small ones!
Especially with fast distance measures, like Hamming.

Exact matching. Global optimum guarantee.

Supports cross check (double scan).

Indexing

Build one+ indexes of descriptors → descriptor data

Indexing is often approximate (especially if asking for more than 1 neighbor), because :

1. Databases can grow very large
2. Descriptor spaces have many dimensions

Exact matching and global optimum are **not always guaranteed**.

Also, cross check usually does not make sense and is therefore not implemented.

Usually, we start by **reducing the dimension** / **encoding our features** *(next lecture)*

kD-Trees

The k-d tree is a binary tree in which every leaf node is a k-dimensional point.

Construction: for each dimension, recursively split the space to maximize data separation until a maximum size is reached

Retrieval: compute the leaf node of each query, then explore points in the leaf and in siblings / parents if not satisfying (boundaries not within radius of the query ball)

Complexity: asymptotic O(logN) when N>>2k

In practice, kD-trees do not work for searching in <u>high</u> dimensions.

Bentley, J. L. (1975). "Multidimensional binary search trees used for associative searching". Communications of the ACM. 18 (9): 509.

FLANN – Efficient indexing

Original version: hierarchical k-Means.

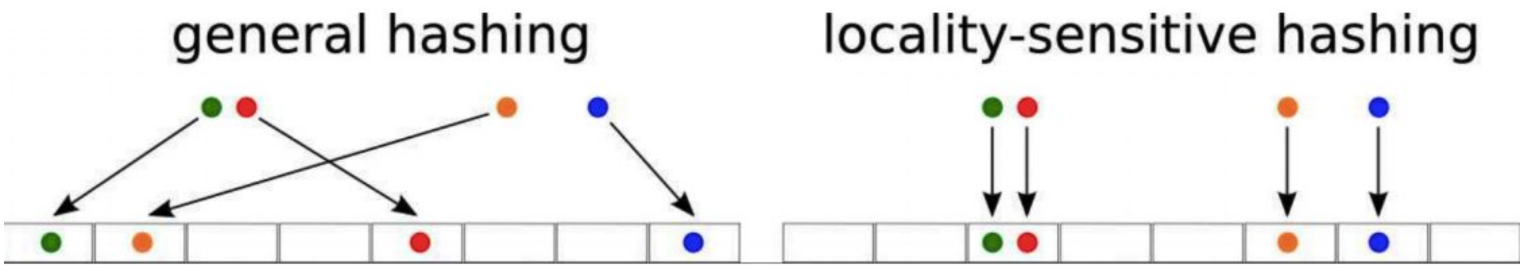
Construction: repetitive k-Means on data (then inside clusters) until minimum cluster size is reached.

Lookup: traverse the tree in a best-bin-first manner with backtrack queue, backtrack until enough points are returned

Marius Muja, David G. Lowe. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration, 2009

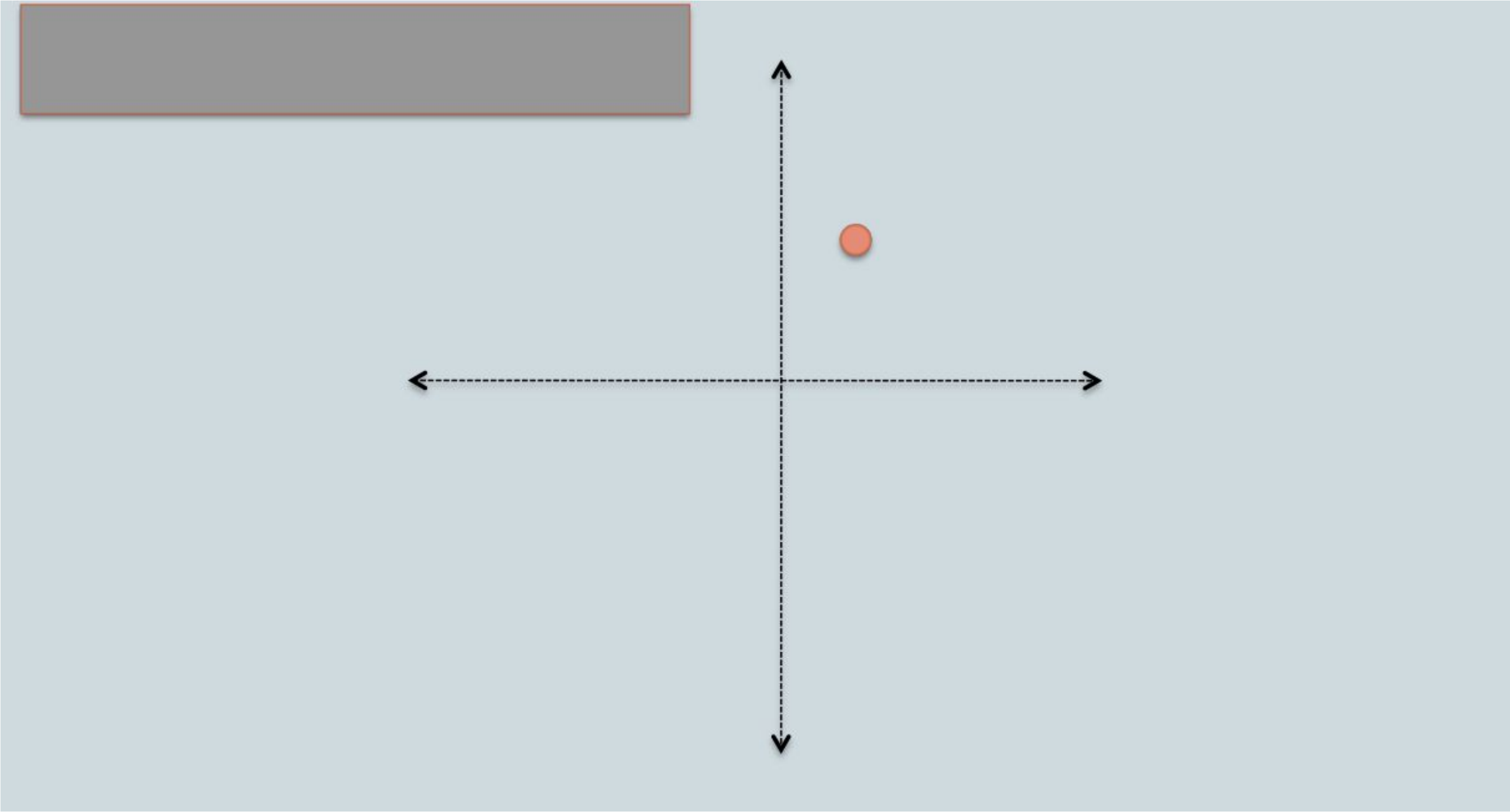
Locality Sensitive Hashing (LSH)

Hash items using a family of hash function which project similar items in the same bucket with high probability. **NOT cryptographic hashing!**



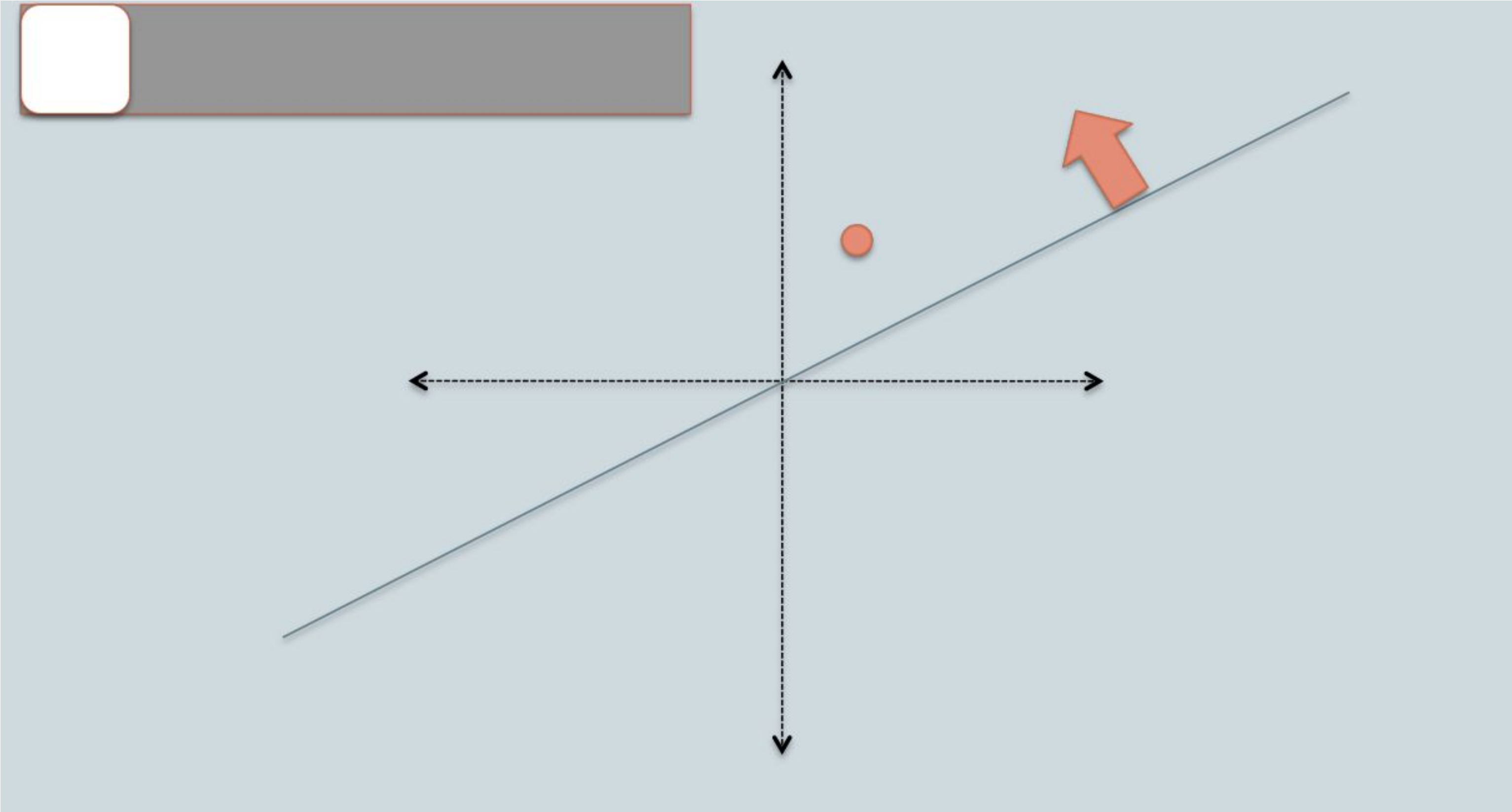
Illustrations: https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134

Locality Sensitive Hashing (LSH)



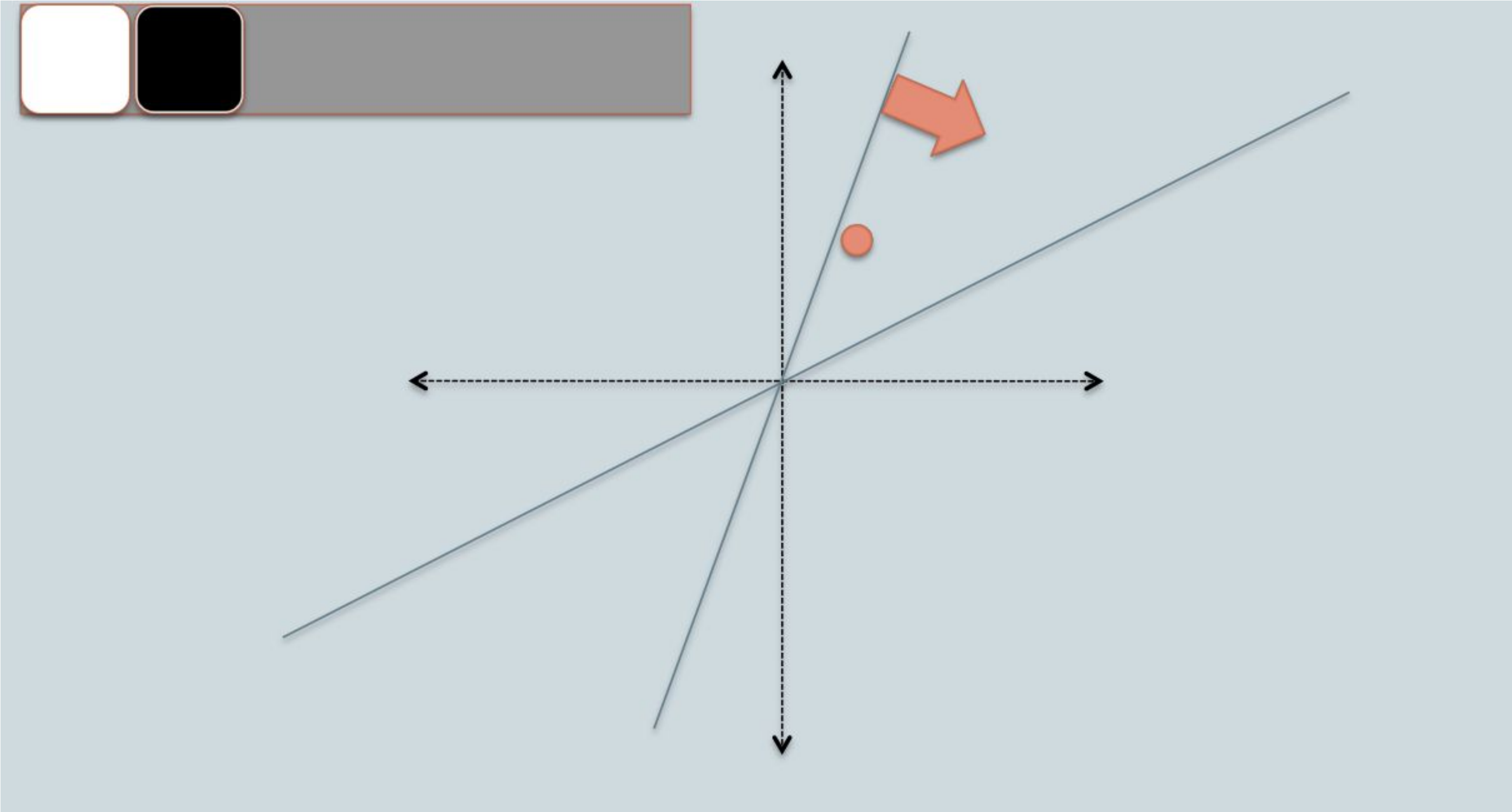
Illustrations: Benjamin Van Durme & Ashwin Lall

Locality Sensitive Hashing (LSH)



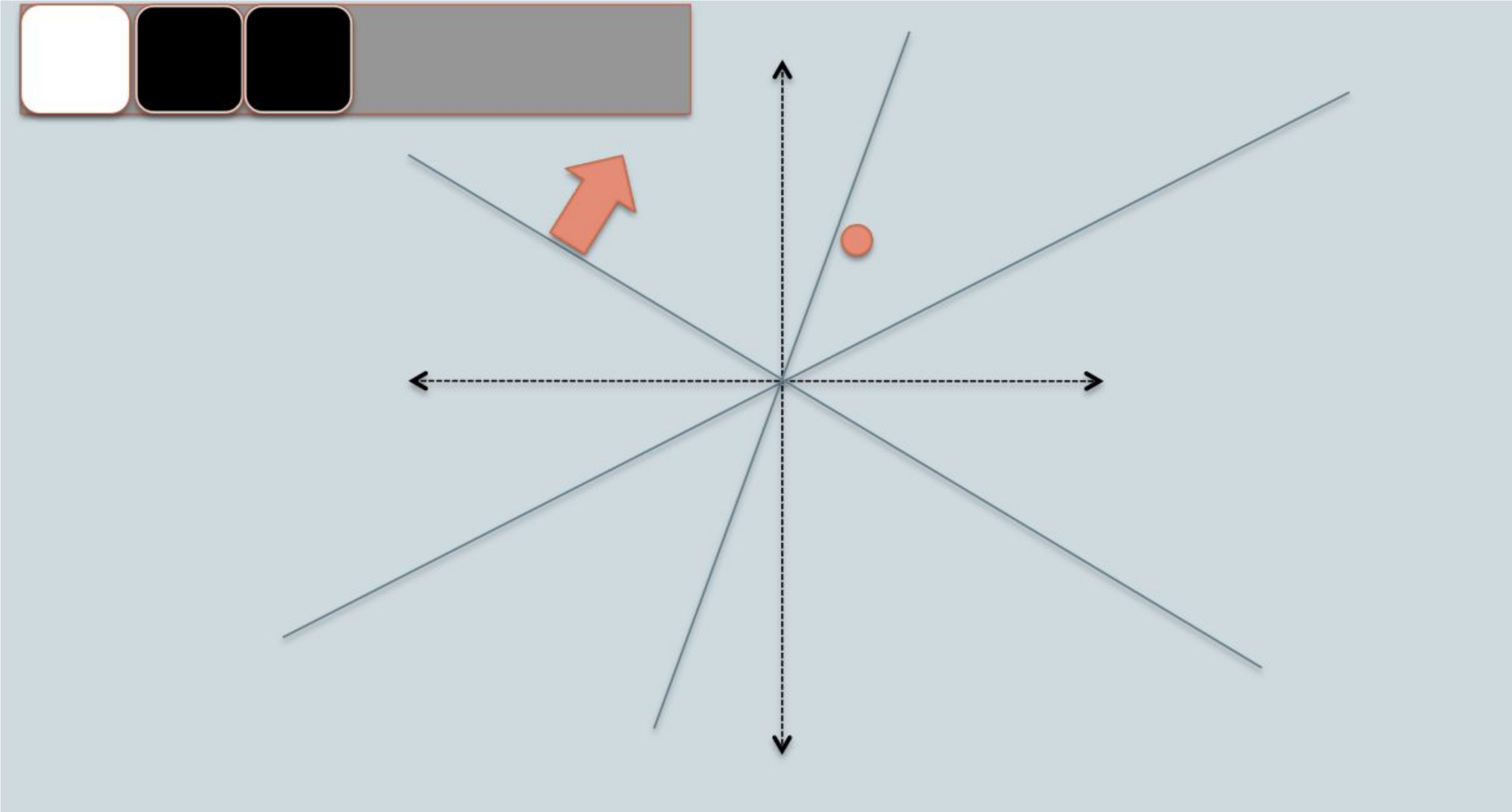
Illustrations: Benjamin Van Durme & Ashwin Lall

Locality Sensitive Hashing (LSH)



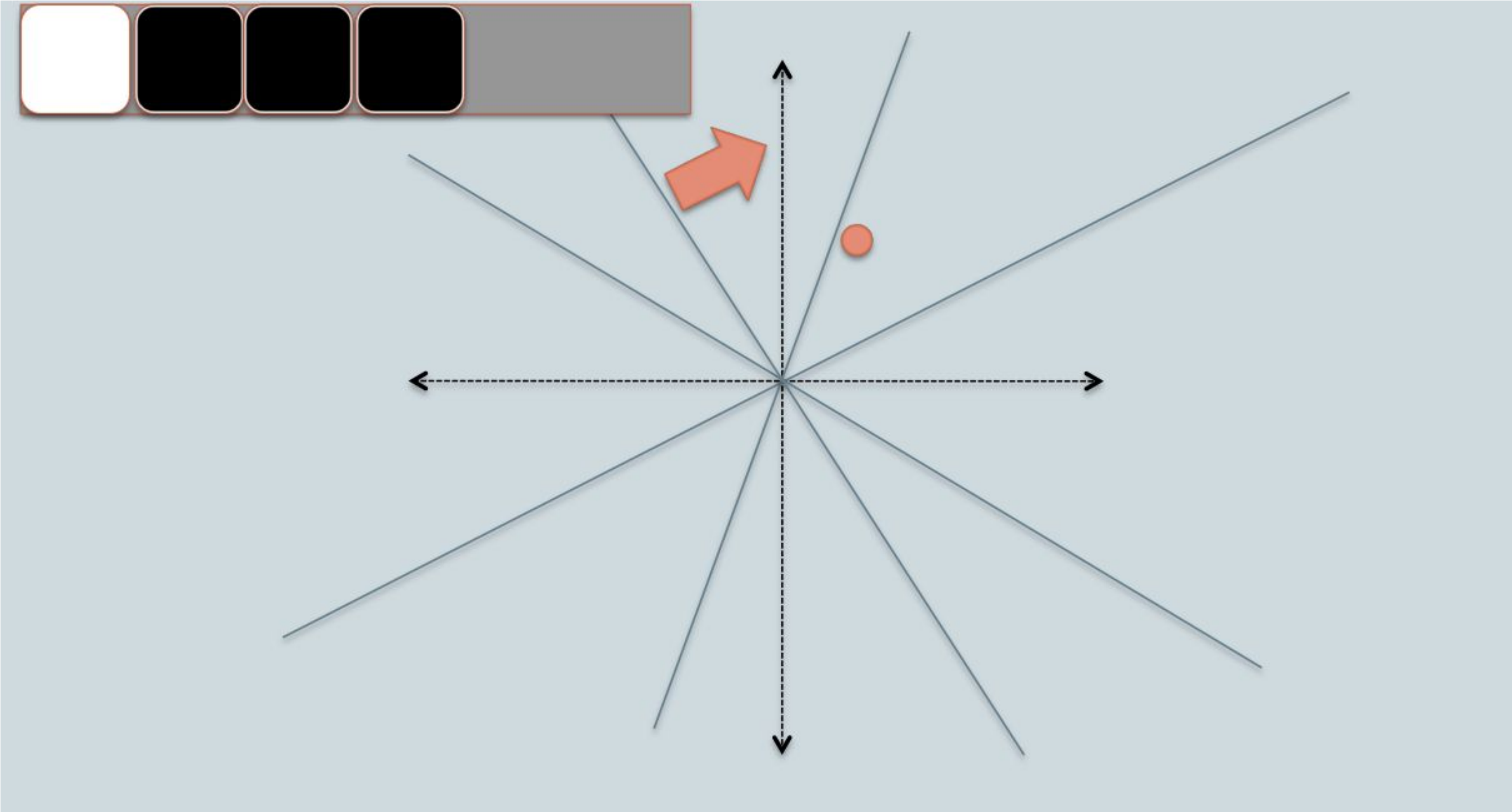
Illustrations: Benjamin Van Durme & Ashwin Lall

Locality Sensitive Hashing (LSH)



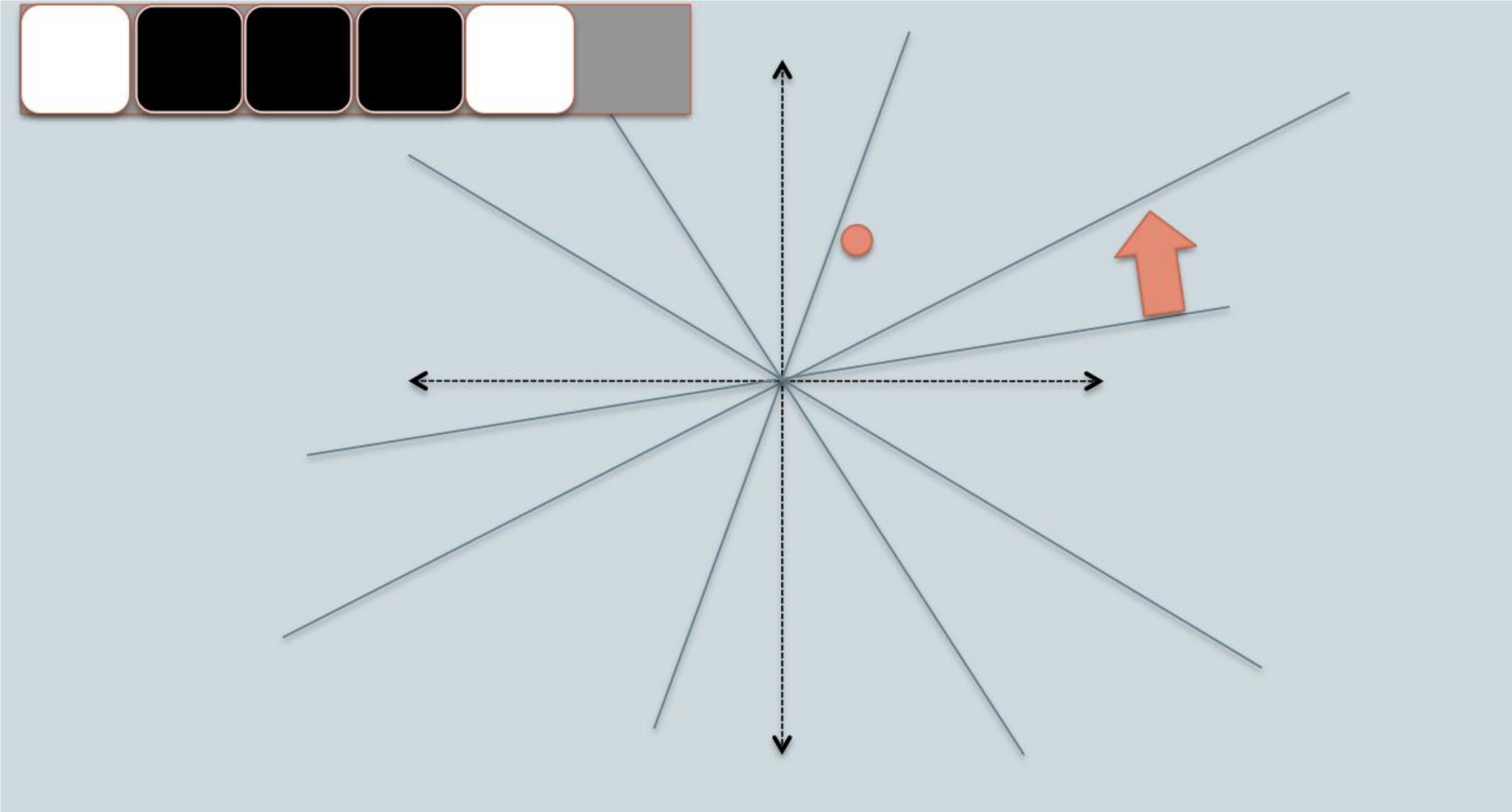
Illustrations: Benjamin Van Durme & Ashwin Lall

Locality Sensitive Hashing (LSH)



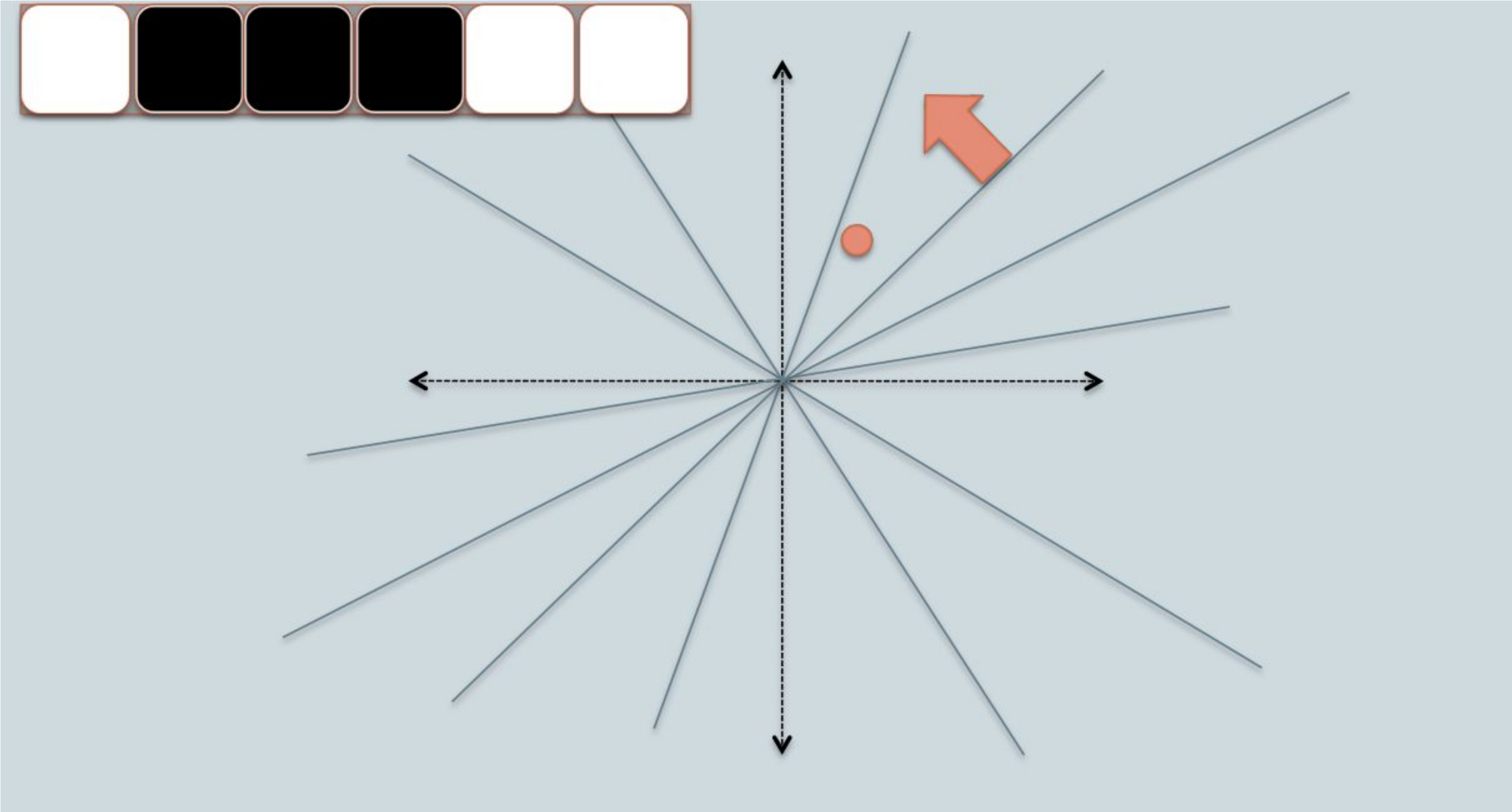
Illustrations: Benjamin Van Durme & Ashwin Lall

Locality Sensitive Hashing (LSH)



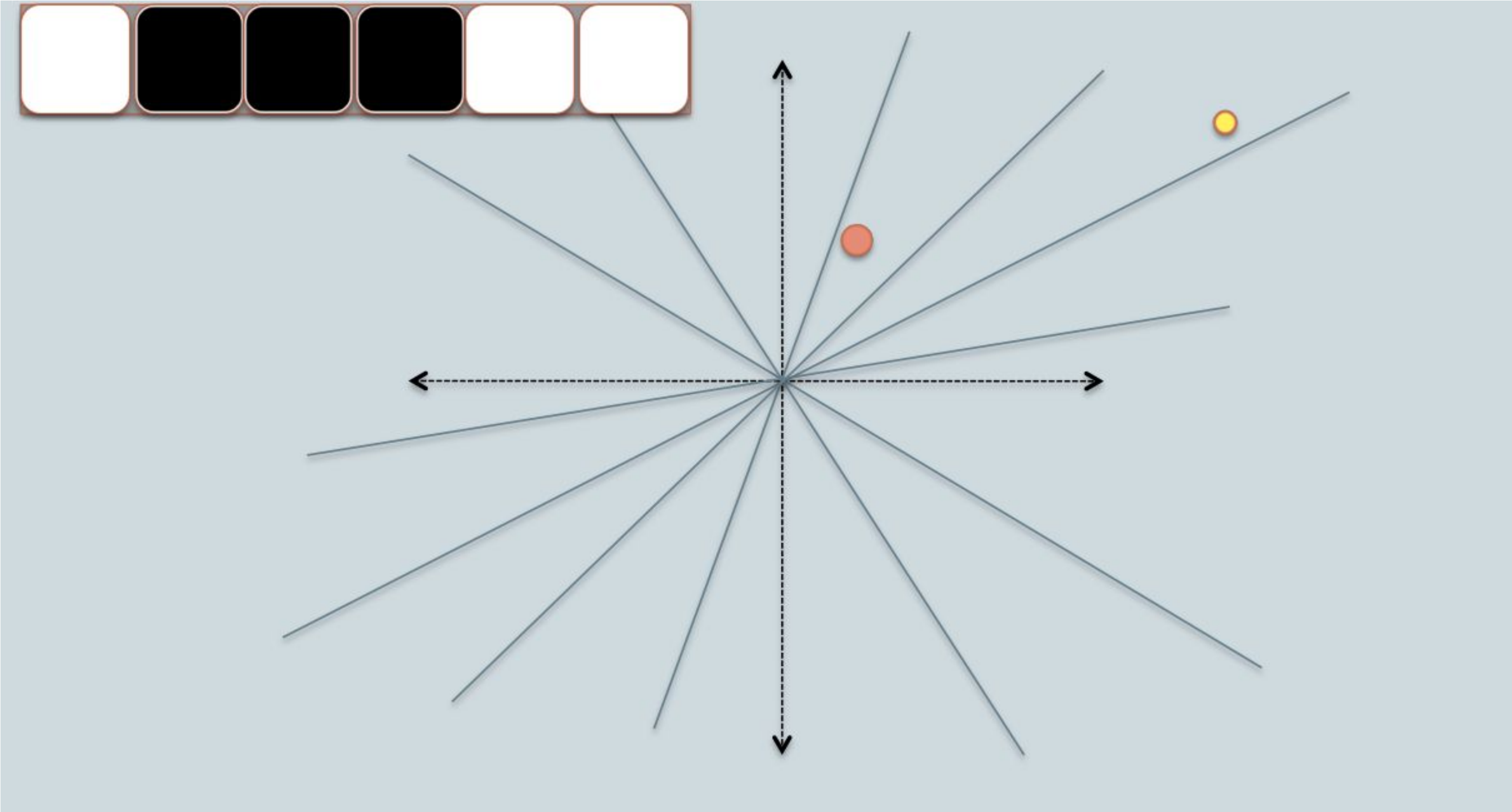
Illustrations: Benjamin Van Durme & Ashwin Lall

Locality Sensitive Hashing (LSH)



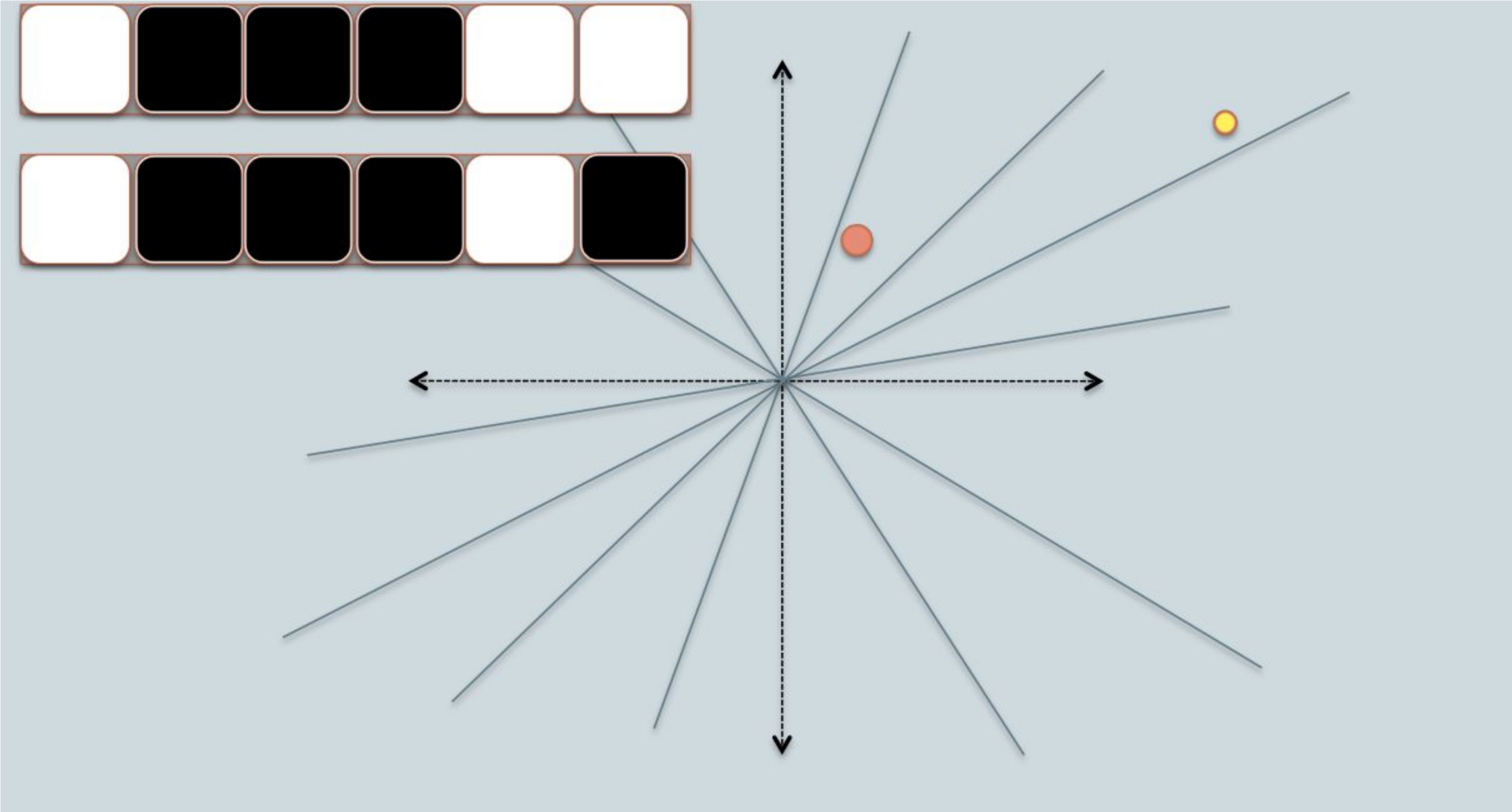
Illustrations: Benjamin Van Durme & Ashwin Lall

Locality Sensitive Hashing (LSH)



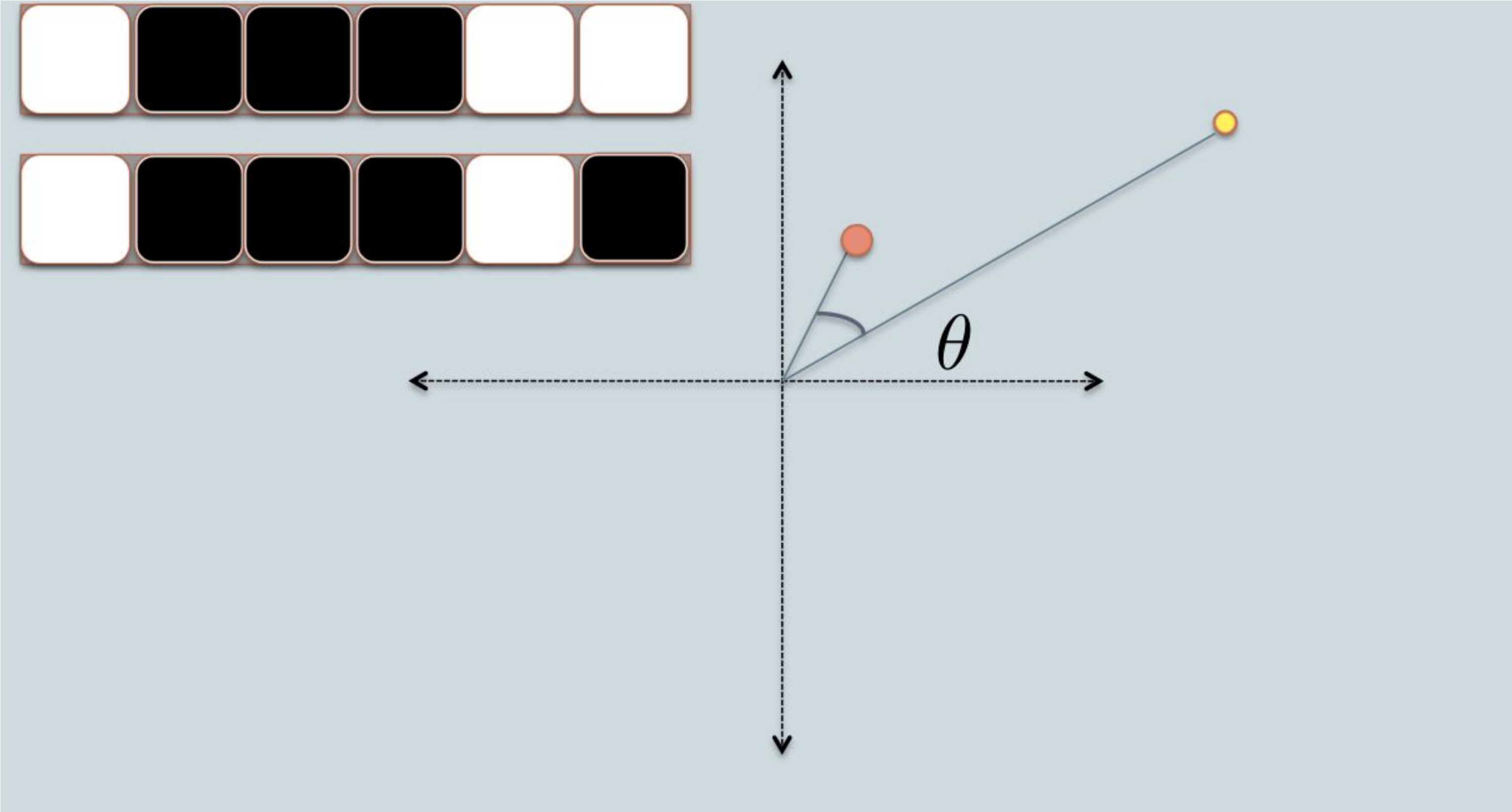
Illustrations: Benjamin Van Durme & Ashwin Lall

Locality Sensitive Hashing (LSH)



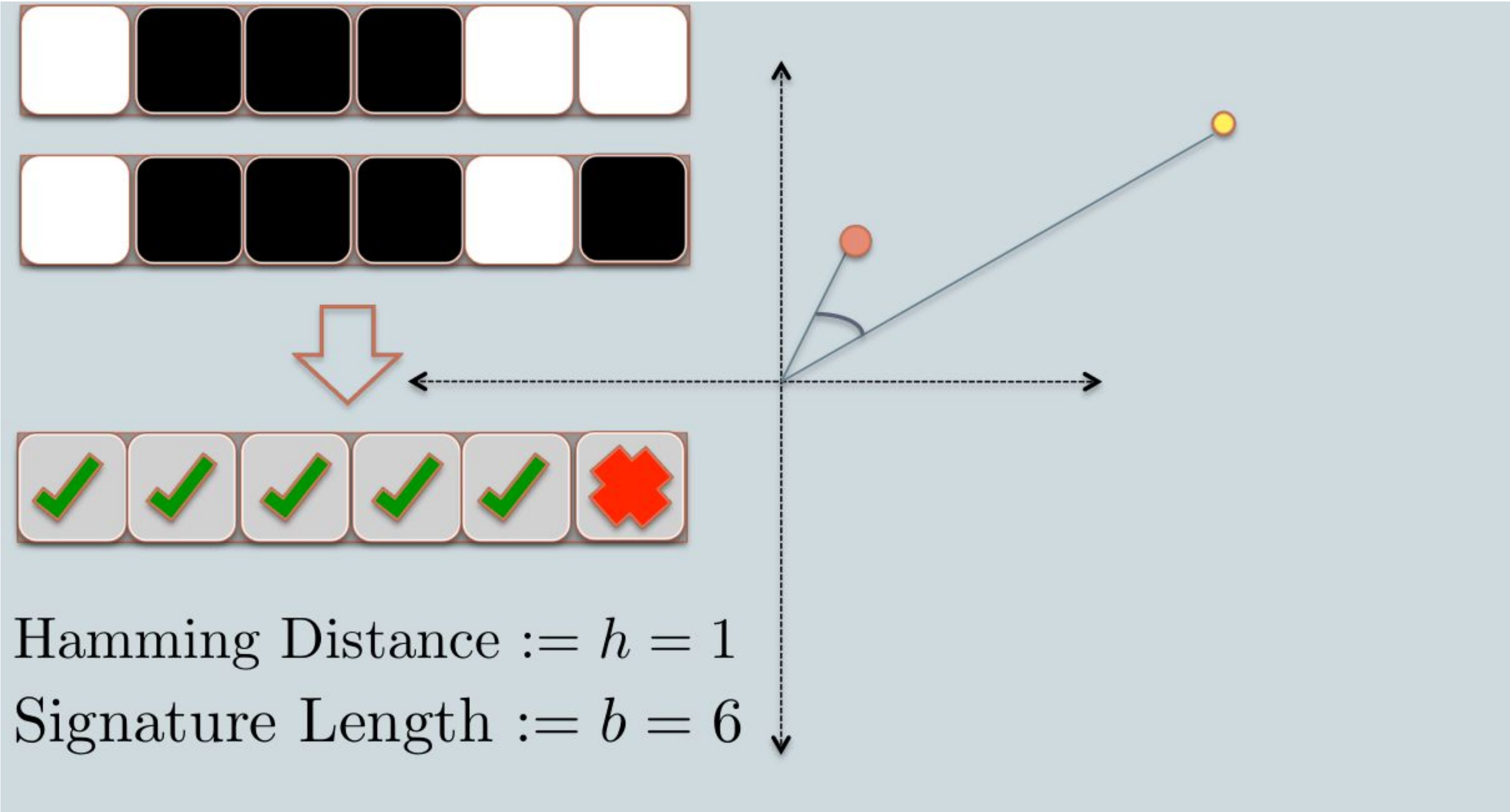
Illustrations: Benjamin Van Durme & Ashwin Lall

Locality Sensitive Hashing (LSH)



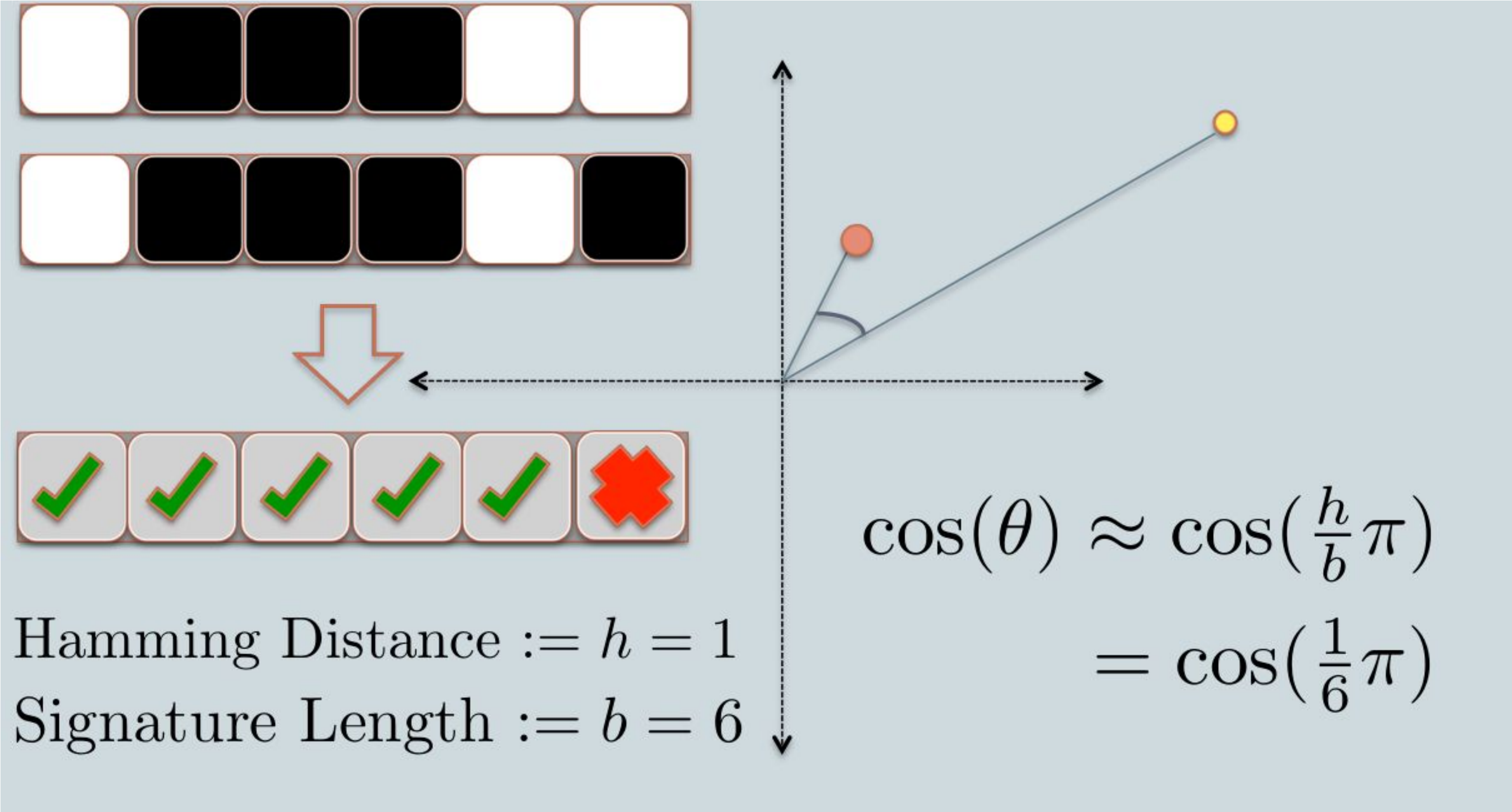
Illustrations: Benjamin Van Durme & Ashwin Lall

Locality Sensitive Hashing (LSH)



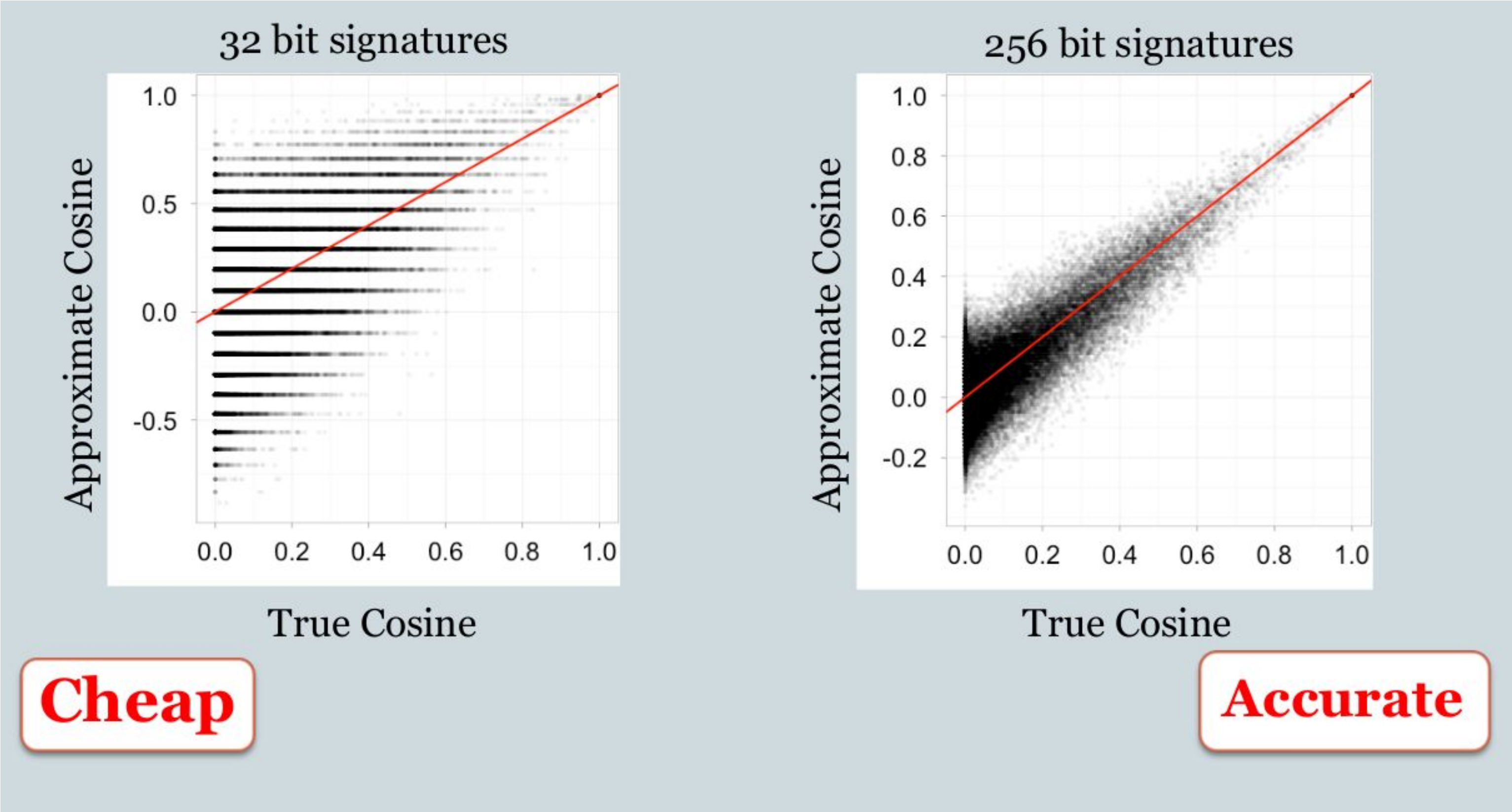
Illustrations: Benjamin Van Durme & Ashwin Lall

Locality Sensitive Hashing (LSH)



Illustrations: Benjamin Van Durme & Ashwin Lall

Locality Sensitive Hashing (LSH)



Illustrations: Benjamin Van Durme & Ashwin Lall

Locality Sensitive Hashing (LSH)

Fast and efficient with large spaces, lot of data

Return a “good match”, maybe not the best one.

kNN can be costly (scan other bins)

Indexes binary descriptors very straightforwardly using bit sampling (sample bits from the coordinates)

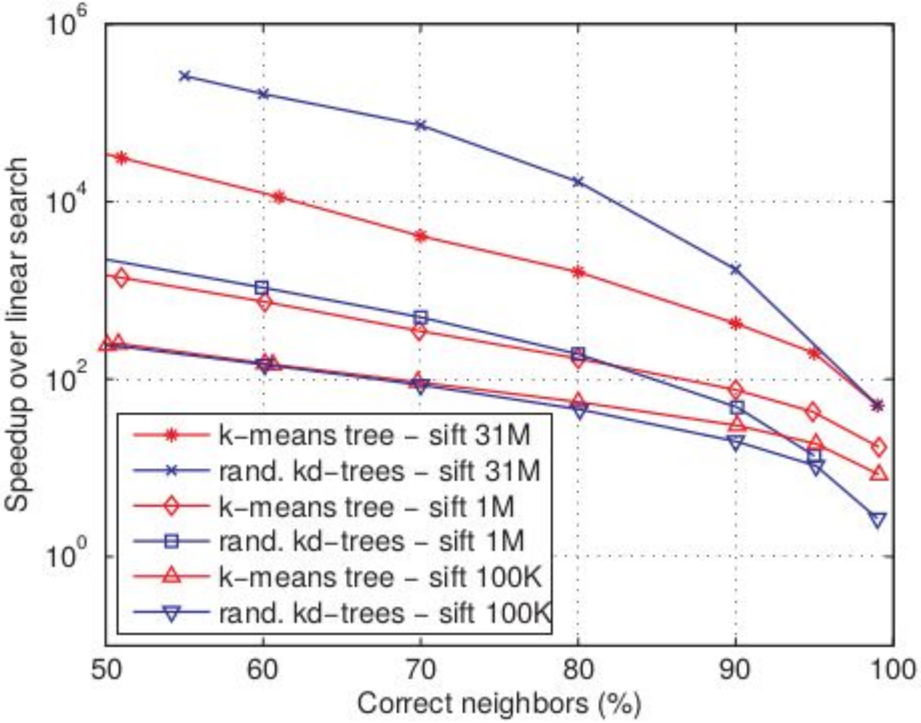
Random projections for other cases, or other techniques…

Which indexing?

Experiment.

Advices:

- Use LSH for binary descriptors like ORB
- Use randomized kD-Trees with SIFT (integer descr. with similar dimension) for moderate dataset size,
  k-Mean tree otherwise



Marius Muja, David G. Lowe. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration, 2009