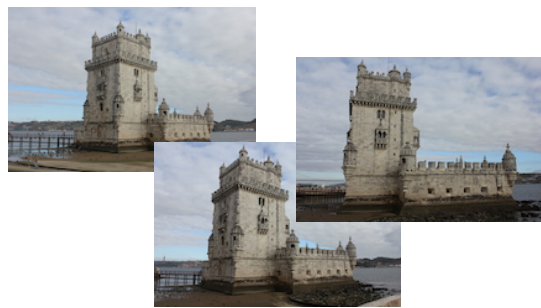


3D from Photographs: Automatic Matching of Images

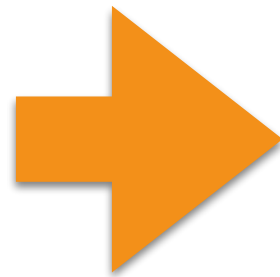
Dr Francesco Banterle

francesco.banterle@isti.cnr.it

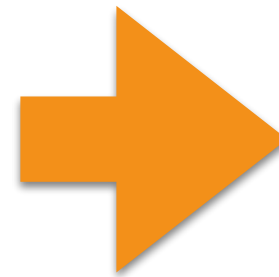
3D from Photographs



Photographs



Automatic
Matching of
Images



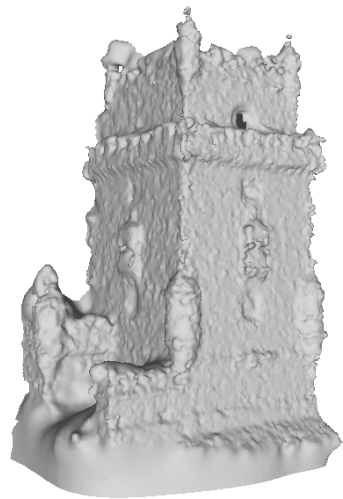
Camera
Calibration



Dense
Matching

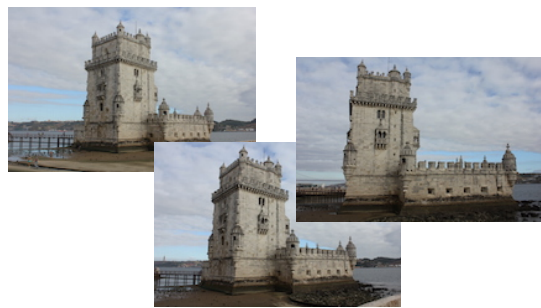


Surface
Reconstruction



3D model

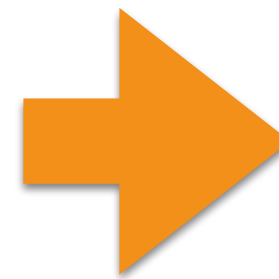
3D from Photographs



Photographs



Automatic
Matching of
Images



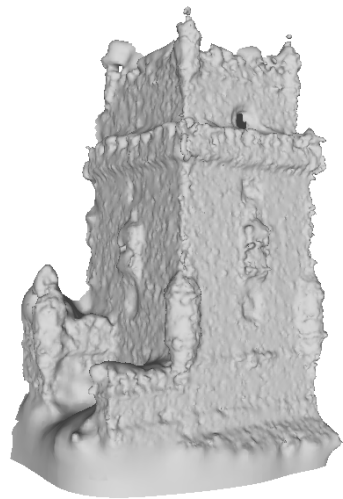
Camera
Calibration



Dense
Matching



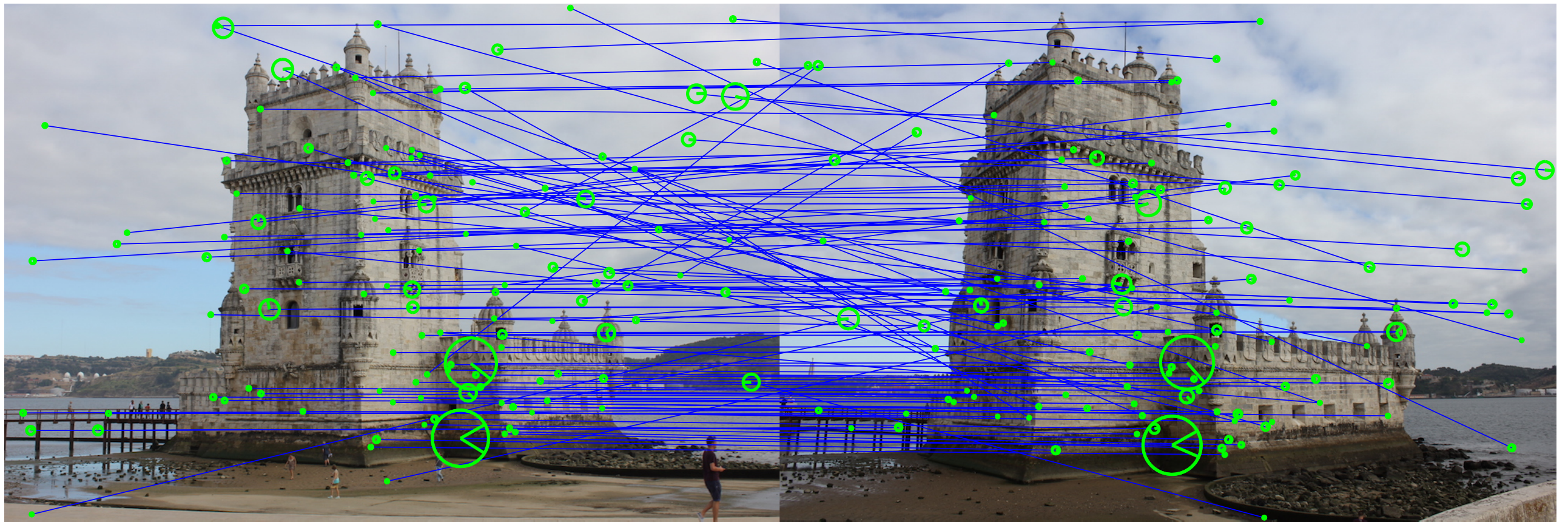
Surface
Reconstruction



3D model

The Matching Problem

- We need to find corresponding feature across two or more views:



The Matching Problem

- Why?
 - 3D Reconstruction.
 - Image Registration.
 - Visual Tracking.
 - Object Recognition.
 - etc.

The Matching Problem: Automatic Panorama Generation



Input
Photographs

The Matching Problem: Automatic Panorama Generation



Input
Photographs

The Matching Problem: Automatic Panorama Generation



Input
Photographs



Panorama

Extraction of Features

Features

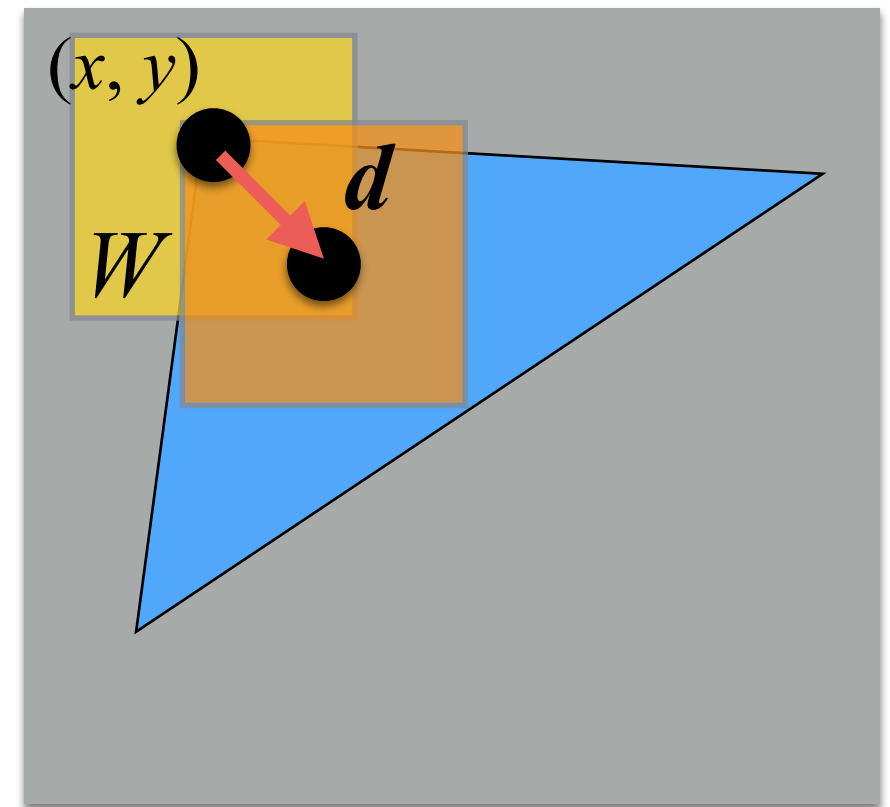
- A feature is a piece of the input image that is relevant for solving a given task.
- Features can be global or local.
- We will focus on local features that are more robust to occlusions and variations.

Extraction of Local Features

- We can extract different kind of features:
 - Flat regions or Blobs
 - Edges
 - Corners

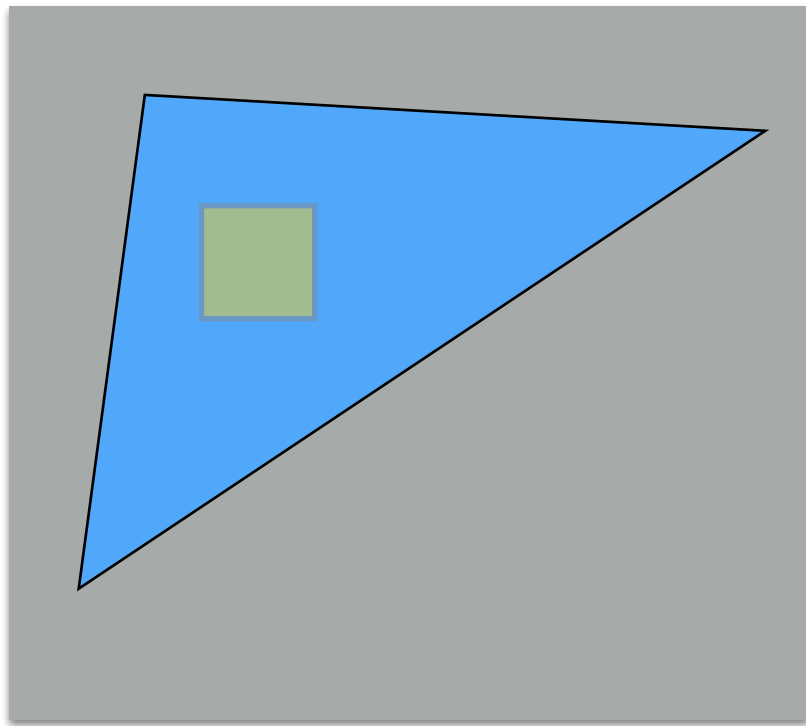
Harris Corner Detector

- Let's consider a window W centered in (x, y) :
 - how do pixels change from a window in (x, y) to another one with a shift $\mathbf{d} = (u, v)$?
 - Let's compare each pixel before and after moving W by $\mathbf{d} = (u, v)$ using the sum of squared differenced (SSD).

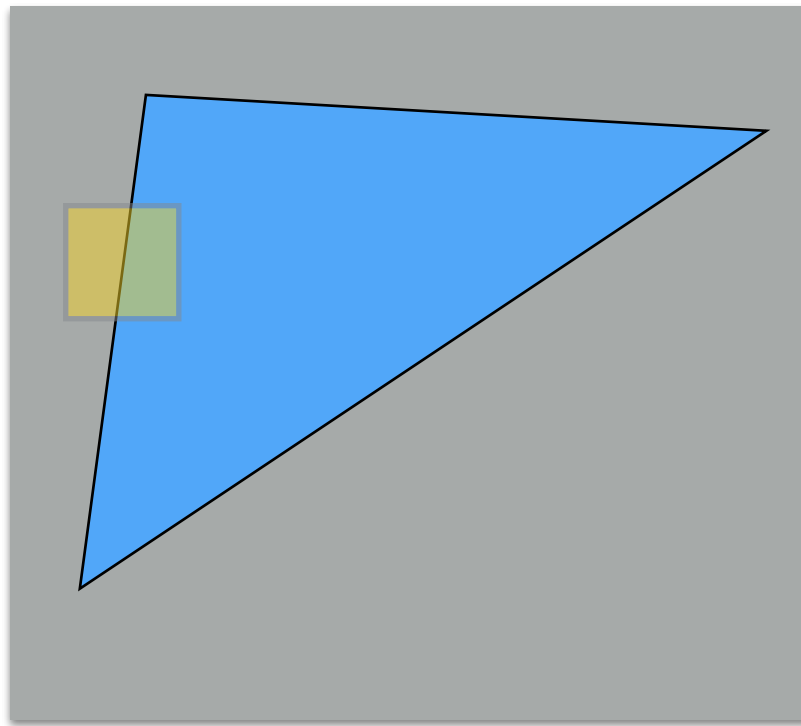


$$E(x, y) = \sum_{x_k, y_k \in W(x, y)} \left(I(x_k + u, y_k + v) - I(x_k, y_k) \right)^2$$

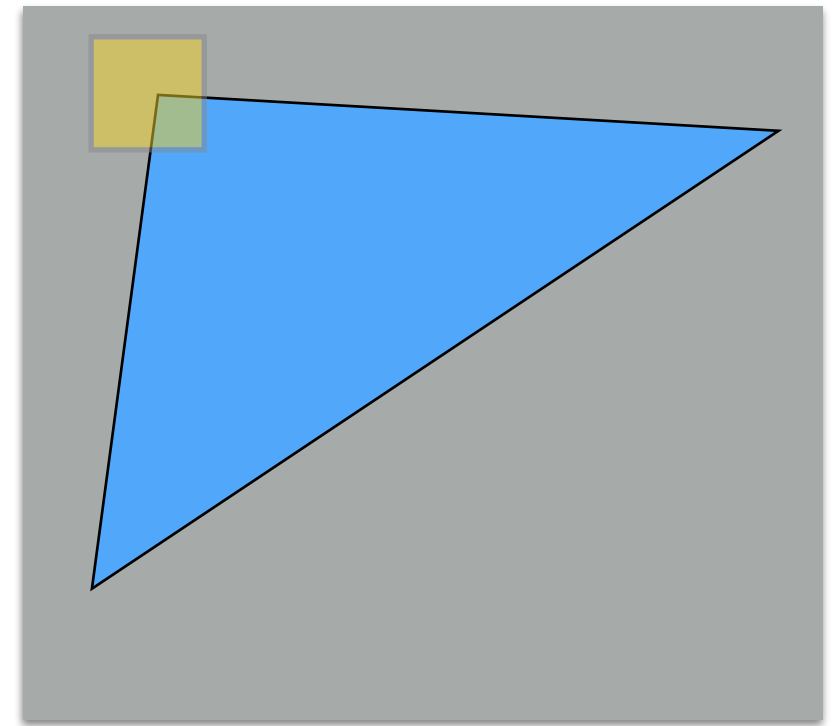
What a Corners is



Flat Region:
no change
in all directions.



Edge:
no change
along the edge.



Corner:
significant change
in all directions.

Harris Corner Detector: Small Motion Assumption

- Let's apply a first-order approximation, which provides good results for small motions:

$$\begin{aligned} I(x + u, y + v) &\approx I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v \\ &\approx I(x, y) + \begin{bmatrix} I_x & I_y \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$

Harris Corner Detector: Small Motion Assumption

$$\begin{aligned} E(x, y) &= \sum_{x_k, y_k \in W(x, y)} \left(I(x_k + u, y_k + v) - I(x_k, y_k) \right)^2 \\ &\approx \sum_{x_k, y_k \in W(x, y)} \left(I(x_k, y_k) + I_x(x_k, y_k)u + I_y(x_k, y_k)v - I(x_k, y_k) \right)^2 \\ &= \sum_{x_k, y_k \in W(x, y)} \left(I_x(x_k, y_k)u + I_y(x_k, y_k)v \right)^2 \\ &= \sum_{x_k, y_k \in W(x, y)} \left(I_x(x_k, y_k)^2 u^2 + 2I_x(x_k, y_k)I_y(x_k, y_k) + I_y(x_k, y_k)^2 v^2 \right) \end{aligned}$$

Harris Corner Detector: Small Motion Assumption

$$E(x, y) \approx \sum_{x_k, y_k \in W(x, y)} \left(I_x(x_k, y_k)^2 u^2 + 2I_x(x_k, y_k)I_y(x_k, y_k) + I_y(x_k, y_k)^2 v^2 \right) \\ = Au^2 + 2Buv + Cv^2$$

$$A = \sum_{x_k, y_k \in W(x, y)} I_x(x_k, y_k)^2$$

$$B = \sum_{x_k, y_k \in W(x, y)} I_x(x_k, y_k)I_y(x_k, y_k)$$

$$C = \sum_{x_k, y_k \in W(x, y)} I_y(x_k, y_k)^2$$

Harris Corner Detector: Small Motion Assumption

- The surface at (x, y) can be locally approximate by a quadratic form:

$$\begin{aligned} E(x, y) &\approx Au^2 + 2Buv + Cv^2 \\ &\approx \begin{bmatrix} u & v \end{bmatrix} \cdot \begin{bmatrix} A & B \\ B & C \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$

$$A = \sum_{x_k, y_k \in W(x, y)} I_x(x_k, y_k)^2$$

$$B = \sum_{x_k, y_k \in W(x, y)} I_x(x_k, y_k) I_y(x_k, y_k)$$

$$C = \sum_{x_k, y_k \in W(x, y)} I_y(x_k, y_k)^2$$

Harris Corner Detector: Small Motion Assumption

- $E(x,y)$ can be rewritten as

$$\begin{aligned} E(x, y) &\approx \sum_{x_k, y_k \in W(x, y)} \begin{bmatrix} u & v \end{bmatrix} \cdot \begin{bmatrix} I_x^2(x_k, y_k) & I_x(x_k, y_k)I_y(x_k, y_k) \\ I_x(x_k, y_k)I_y(x_k, y_k) & I_y^2(x_k, y_k) \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} \\ &= \begin{bmatrix} u & v \end{bmatrix} \cdot M \cdot \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$

$$M = \sum_{x_k, y_k \in W(x, y)} \begin{bmatrix} I_x^2(x_k, y_k) & I_x(x_k, y_k)I_y(x_k, y_k) \\ I_x(x_k, y_k)I_y(x_k, y_k) & I_y^2(x_k, y_k) \end{bmatrix}$$

Harris Corner Detector: Small Motion Assumption

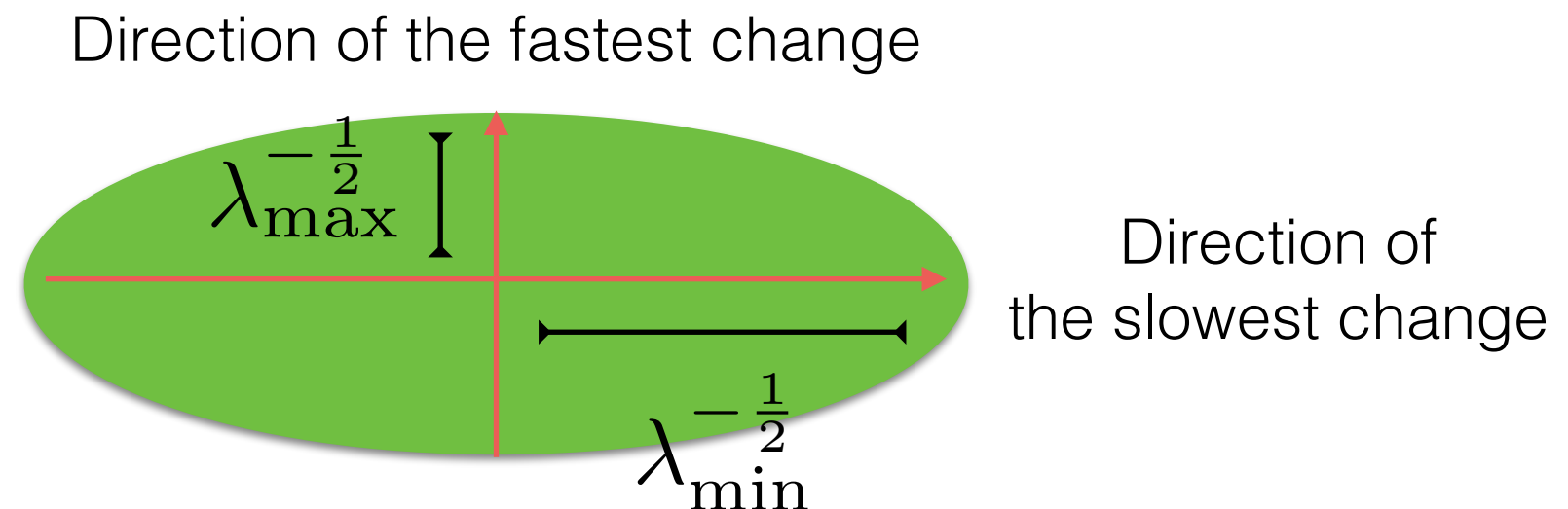
- $E(x,y)$ can be rewritten as

$$E(x, y) \approx \sum_{x_k, y_k \in W(x, y)} [u \quad v] \cdot \begin{bmatrix} I_x^2(x_k, y_k) & I_x(x_k, y_k)I_y(x_k, y_k) \\ I_x(x_k, y_k)I_y(x_k, y_k) & I_y^2(x_k, y_k) \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix}$$
$$= \boxed{[u \quad v] \cdot M \cdot \begin{bmatrix} u \\ v \end{bmatrix}} \quad \text{Ellipse Equation:}$$
$$E(u, v) = k$$

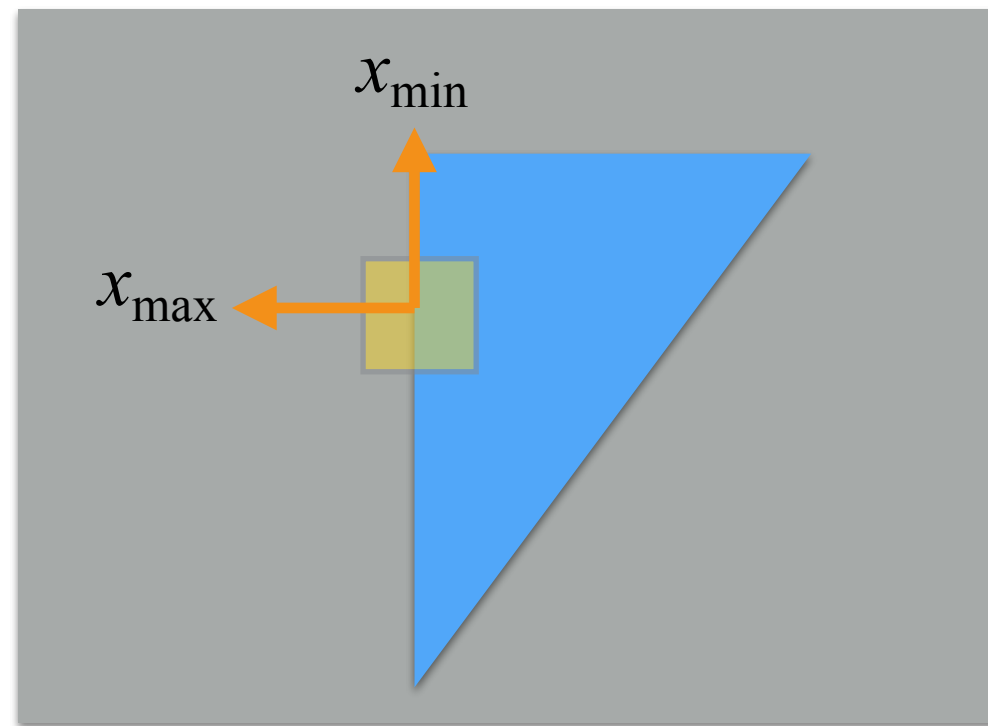
$$M = \sum_{x_k, y_k \in W(x, y)} \begin{bmatrix} I_x^2(x_k, y_k) & I_x(x_k, y_k)I_y(x_k, y_k) \\ I_x(x_k, y_k)I_y(x_k, y_k) & I_y^2(x_k, y_k) \end{bmatrix}$$

Harris Corner Detector: Second Moment Matrix

- M reveals information about the distribution of gradients around a pixel.
- The eigenvectors of M identify the directions of fastest and slowest change.



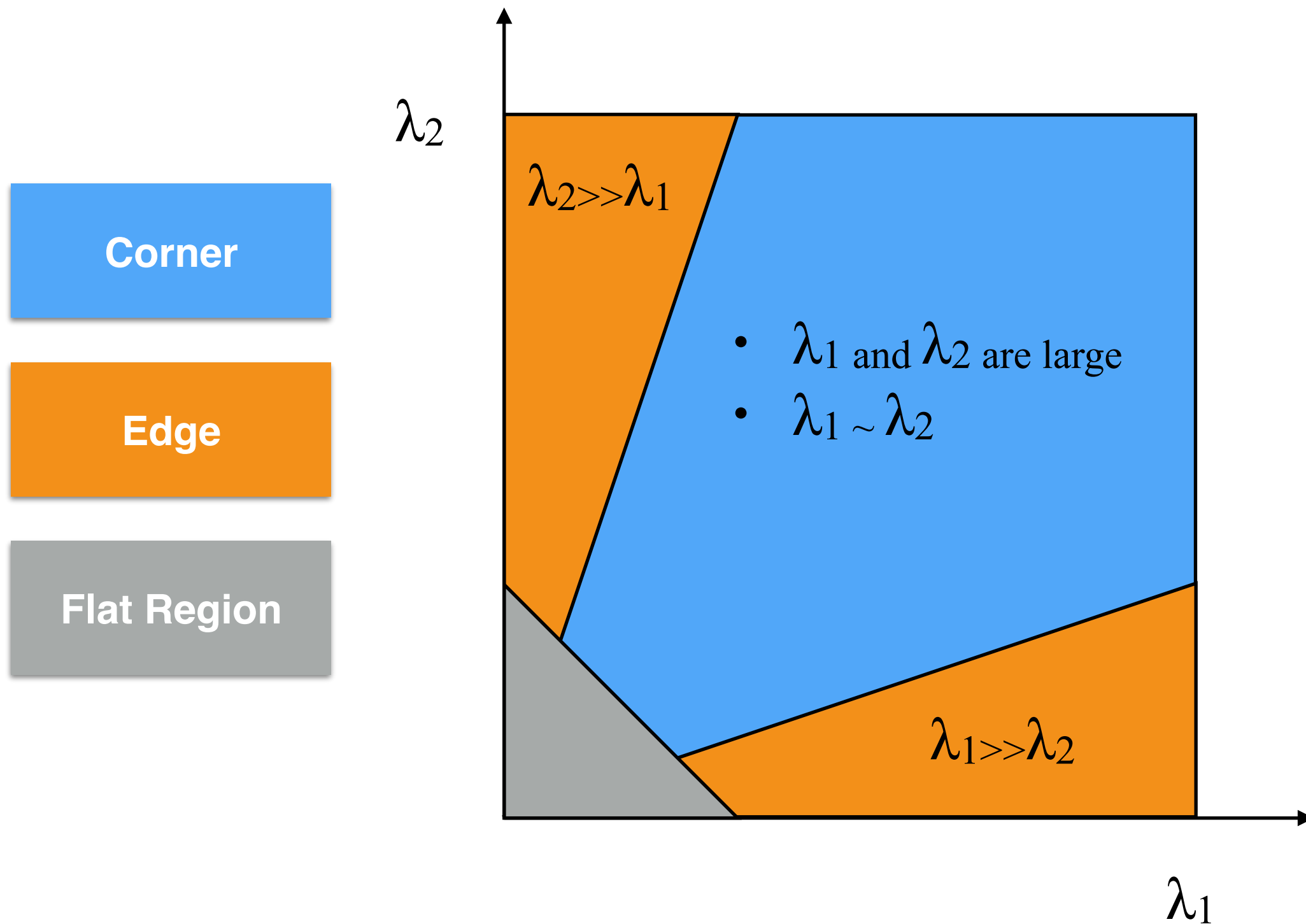
Harris Corner Detector: Second Moment Matrix



Eigenvalues and eigenvectors of M define shift directions with the smallest and largest change in E :

- $x_{\max}$ = direction of largest increase in E
- $\lambda_{\max}$ = amount of increase in direction $x_{\max}$
- $x_{\min}$ = direction of smallest increase in E
- $\lambda_{\min}$ = amount of increase in direction $x_{\min}$

Classification



Harris Corner Detector: Cornersness Measure

- Instead of directly computing the eigenvalues, we use a measure that determines the “***cornerness***” of a pixel (i.e., how close to be a corner is):

$$R = \text{Det}(M) - k\text{Tr}(M)^2$$

$$\text{Det}(M) = \lambda_1 \lambda_2$$

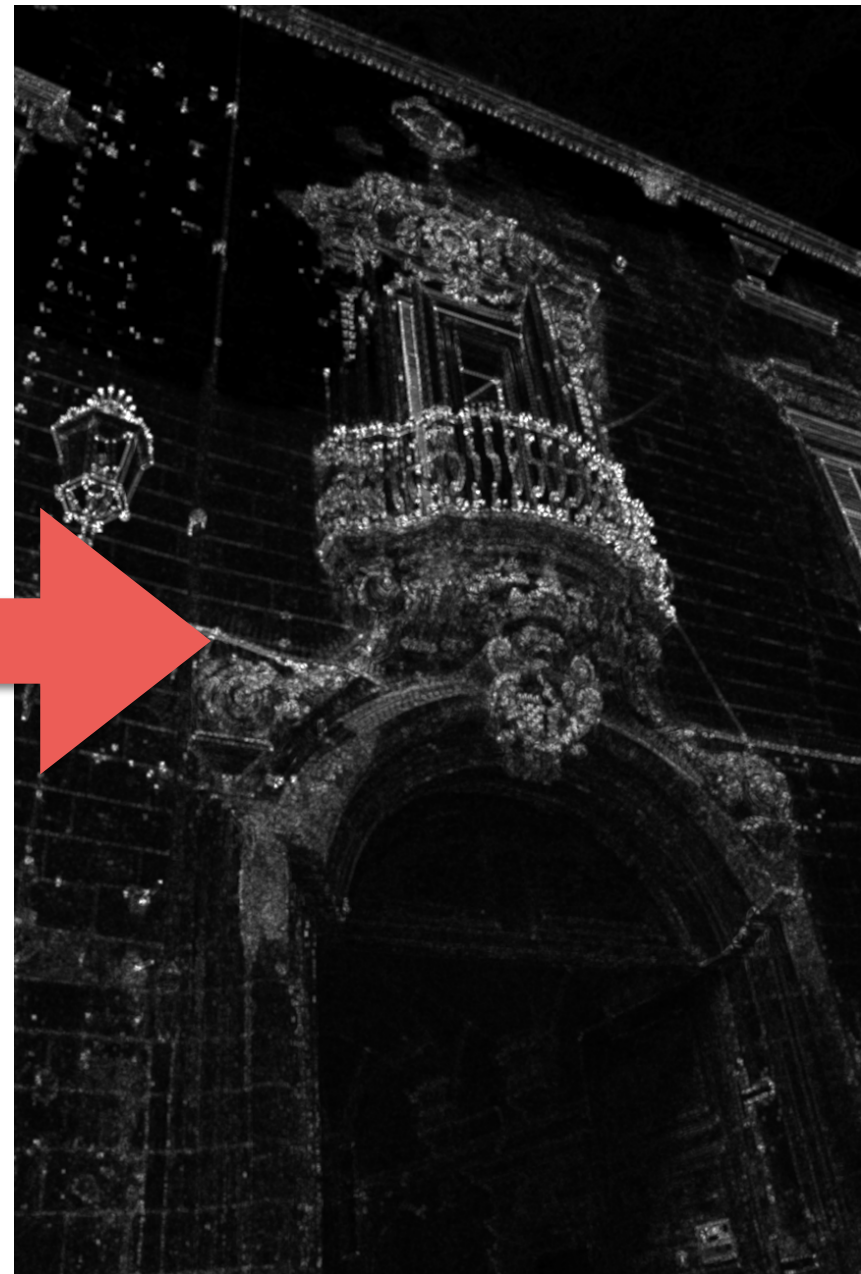
$$\text{Tr}(M) = \lambda_1 + \lambda_2$$

- k is an empiric constant with values $[0.04 \ 0.06]$.

Harris Corner Detector: Corneriness Measure



Input Image



R

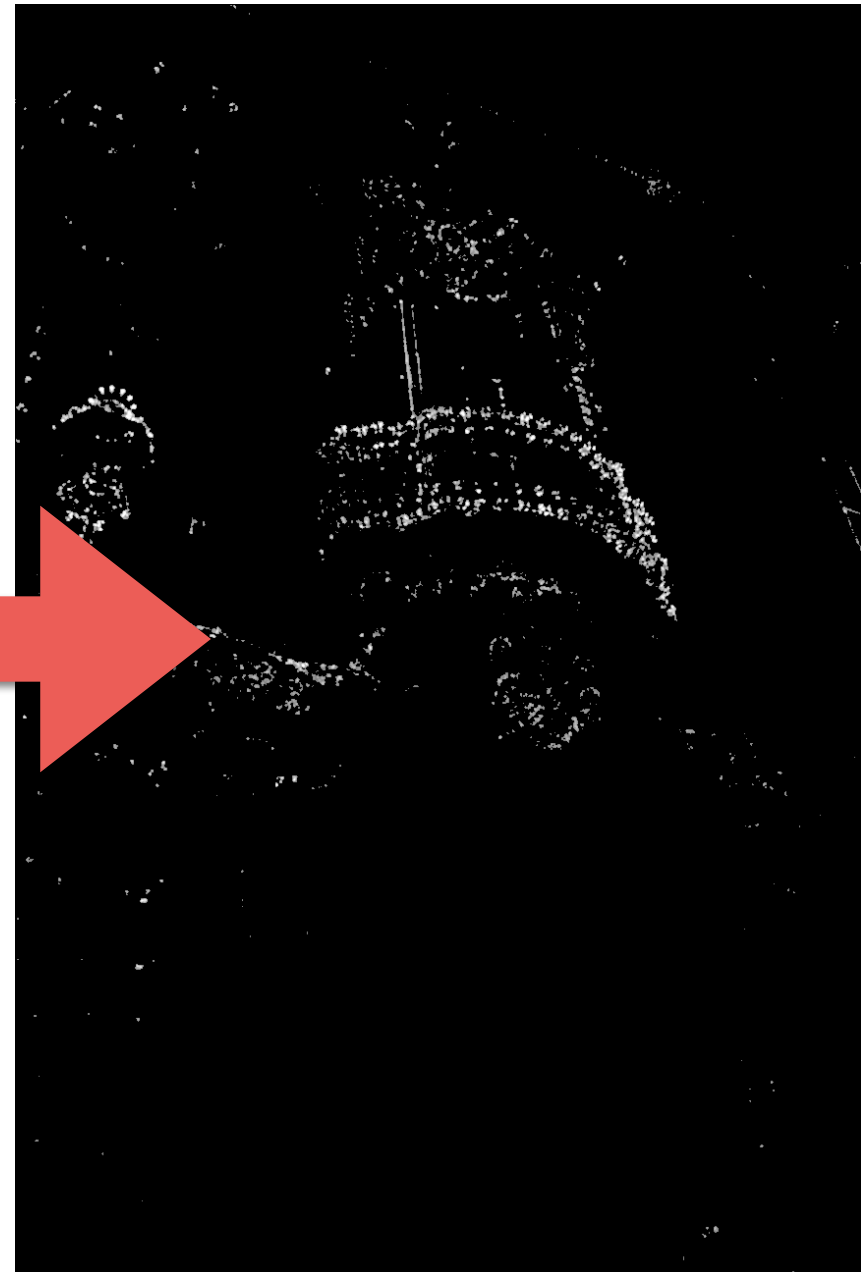
Harris Corner Detector: Pruning Corners

- We have to find pixels with large corner response, R , i.e., $R > T_0$.
- Typically, T_0 in $[0,1]$ depends on the number of points we want to extract; a default value is 0.01.

Harris Corner Detector: Thresholding



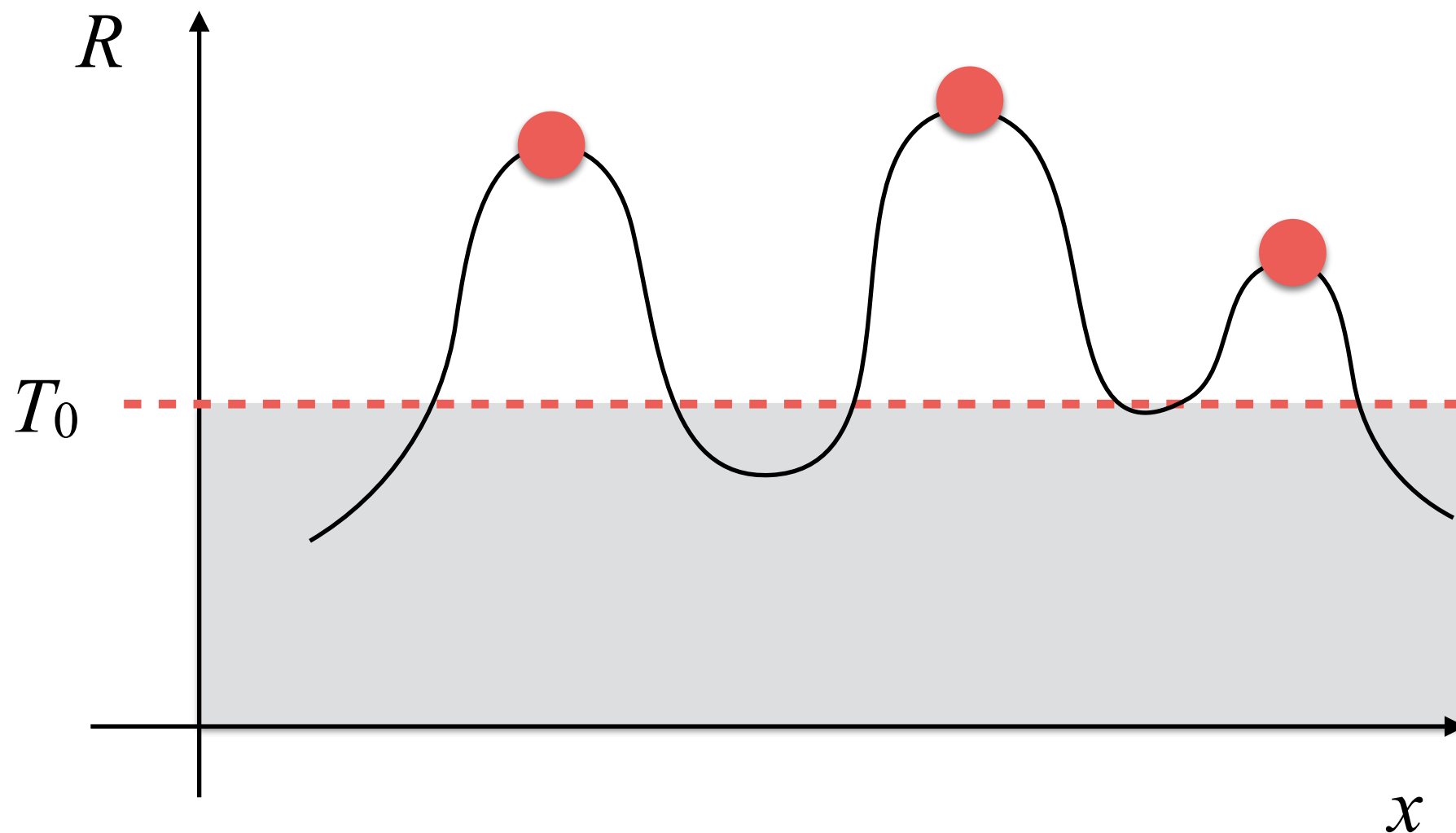
R



R after thresholding

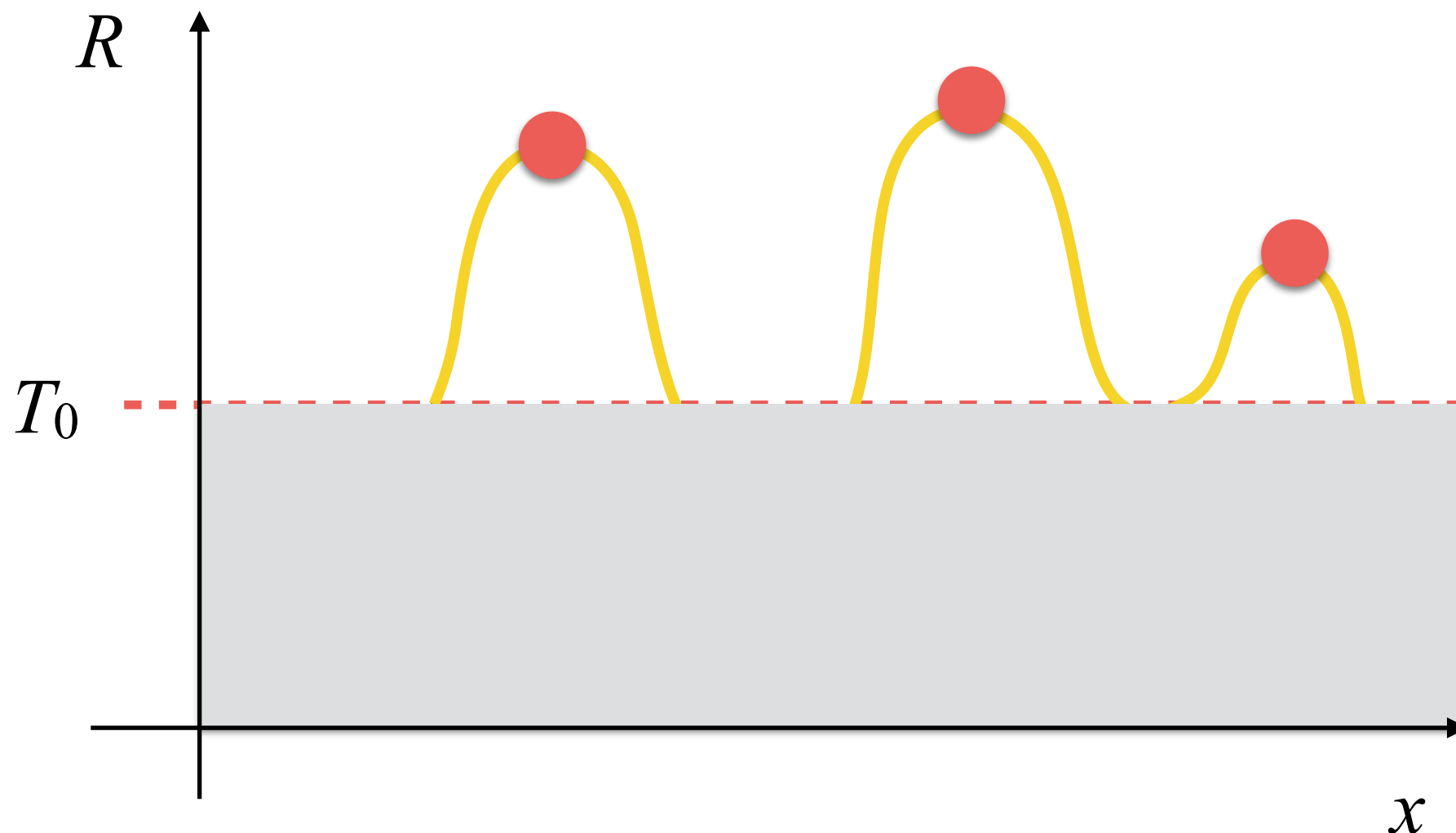
Harris Corner Detector: Pruning Corners

- At this point, we need to suppress/remove values that are not maxima.



Harris Corner Detector: Pruning Corners

- At this point, we need to suppress/remove values that are not maxima, but they are over the threshold (yellow pixels).



Harris Corner Detector: Pruning Corners

- We set a radius (in pixel) for suppressing non-maxima; e.g., 3-5.
- We apply to R a maximum filter; it is similar to a median filter, but it computes the maximum instead of the median. After this we obtain a filtered image called $R_{\max}$.
- A pixel at position (x, y) is a local maximum if and only if:

$$R_{\max}(x, y) = R(x, y) \quad \wedge \quad R(x, y) > T_0$$

Harris Corner Detector: Pruning Corners Example 1

100	0	30
40	20	20
0	0	0

The current pixel that we are evaluating is the central one!

$$T_0 = 5$$

Harris Corner Detector: Pruning Corners Example 1

<i>100</i>	0	30
40	20	20
0	0	0

The maximum is 100!

Harris Corner Detector: Pruning Corners Example 1

100	0	30
40	0	20
0	0	0

$20 < 100$ so it has to be suppressed; i.e., set to 0!

Harris Corner Detector: Pruning Corners Example 2

20	0	30
40	100	20
0	0	0

The current pixel that we are evaluating is the central one!

$$T_0 = 5$$

Harris Corner Detector: Pruning Corners Example 2

20	0	30
40	100	20
0	0	0

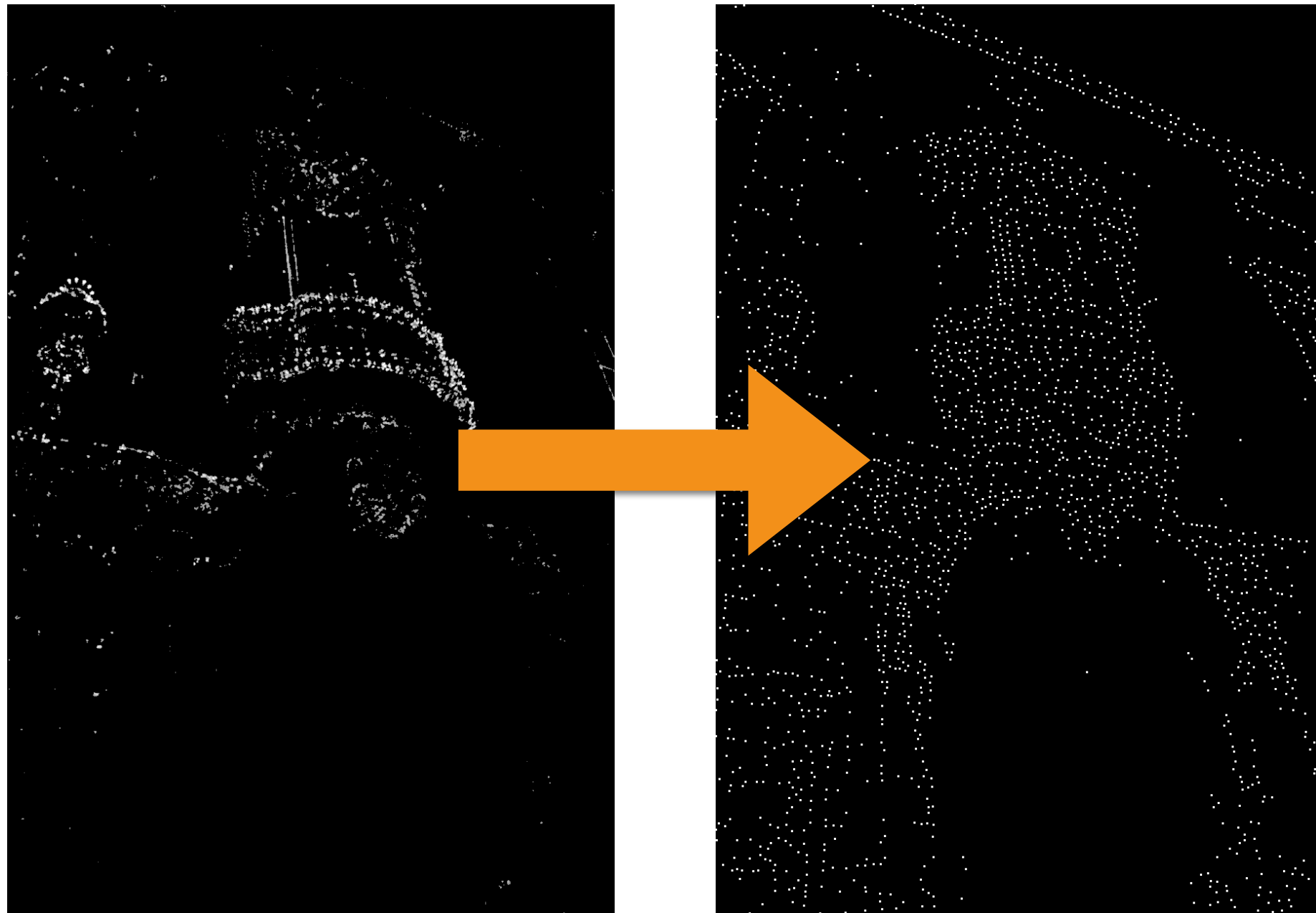
The maximum is 100!

Harris Corner Detector: Pruning Corners Example 2

20	0	30
40	100	20
0	0	0

$100 == 100$ so it has to be kept!

Harris Corner Detector: Non-Maximal Suppression



R after thresholding

Non-Maximal Suppression

Harris Corner Detector: Non-Maximal Suppression



Harris Corner Detector: Non-Maximal Suppression

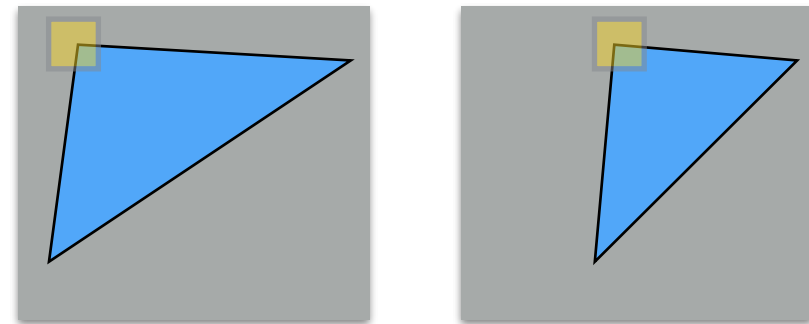


Harris Corner Detector: Non-Maximal Suppression

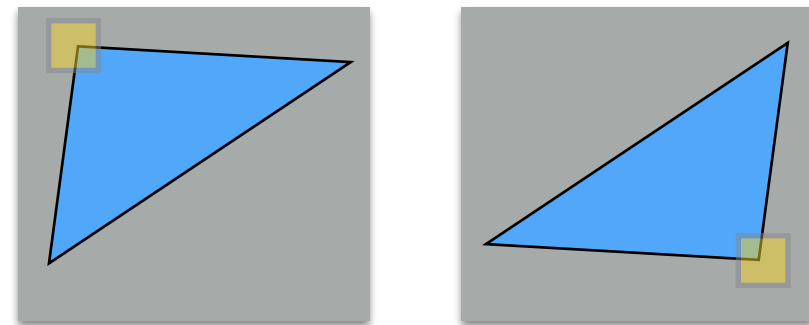


Harris Corner: Advantages

- Translational invariance:



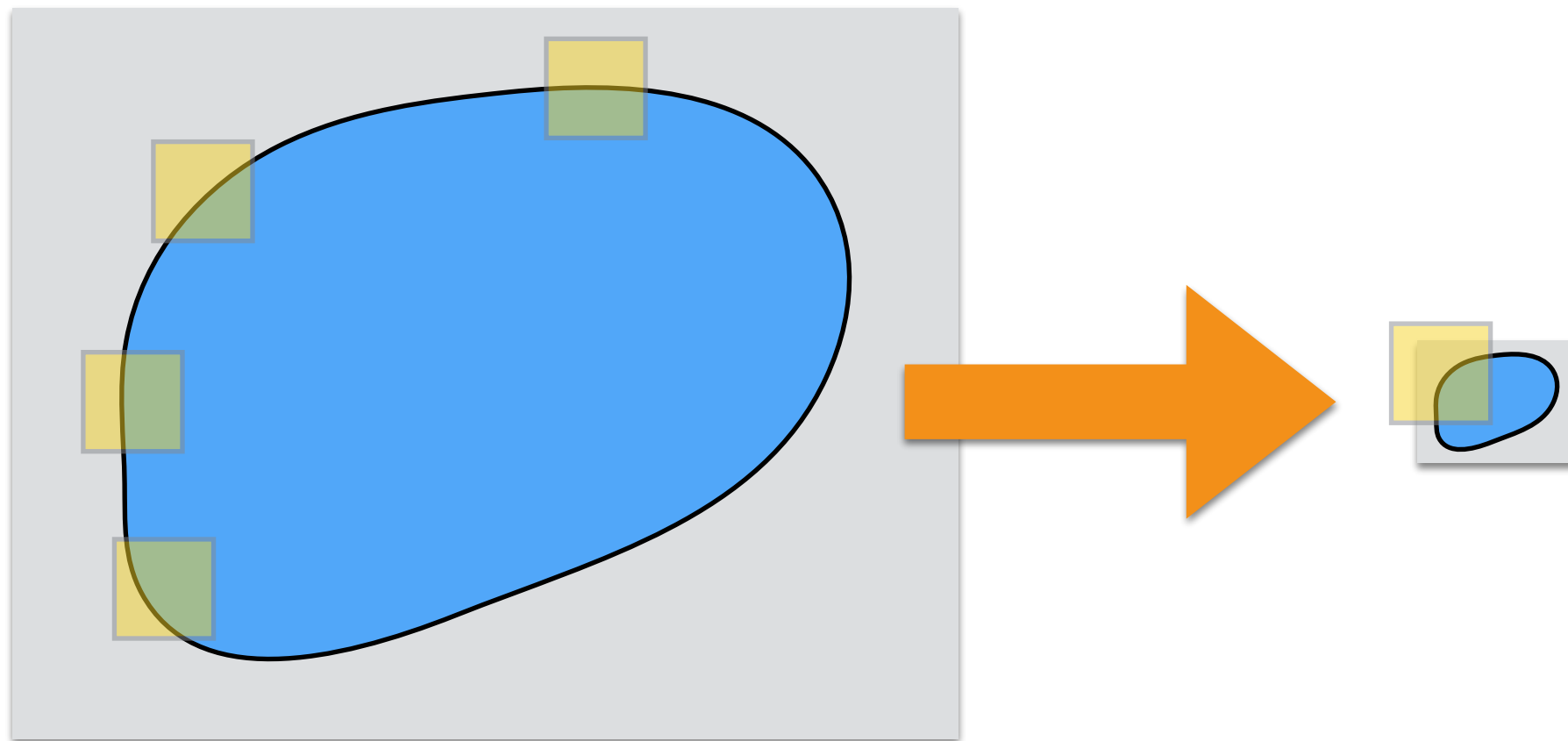
- Rotation invariance:



- Only derivatives are employed:
 - Intensity shift invariance: $I' = I + b$
 - Intensity scale invariance: $I' = I a$

Harris Corner: Disadvantage

- Not scale invariant!



All points are
classified as edges

It is now
a corner!

The same feature in
different images can have
different size!

The Scale Problem

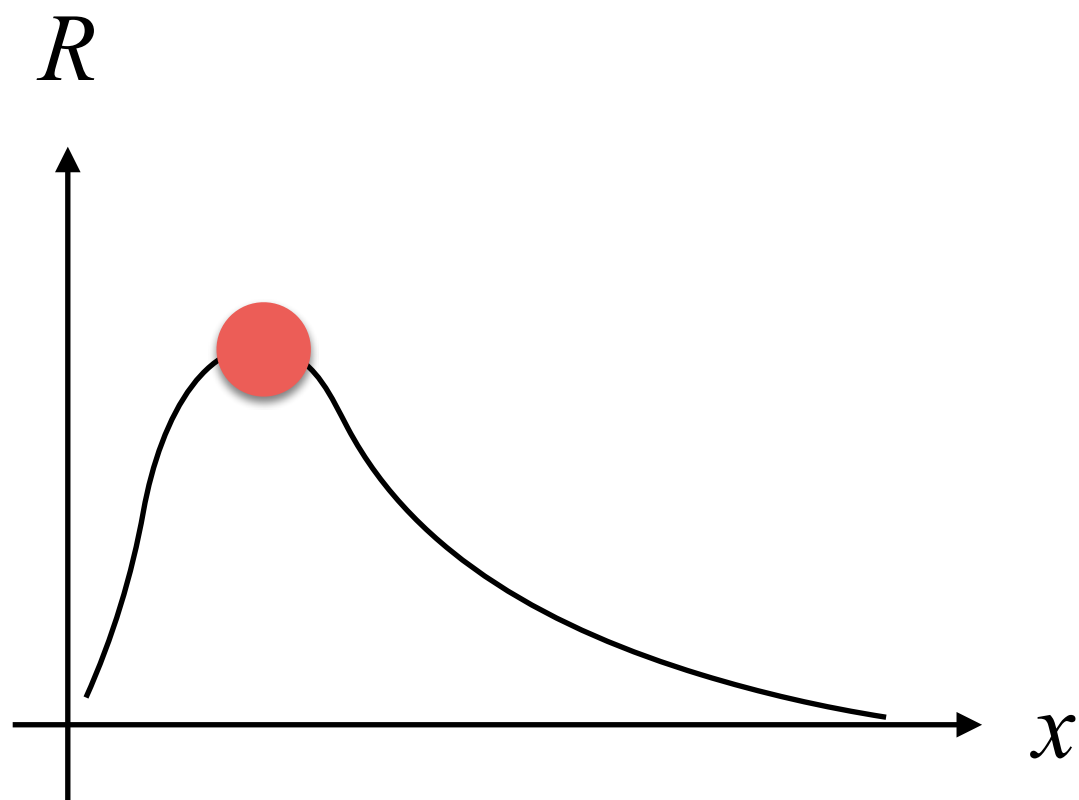


Near Object

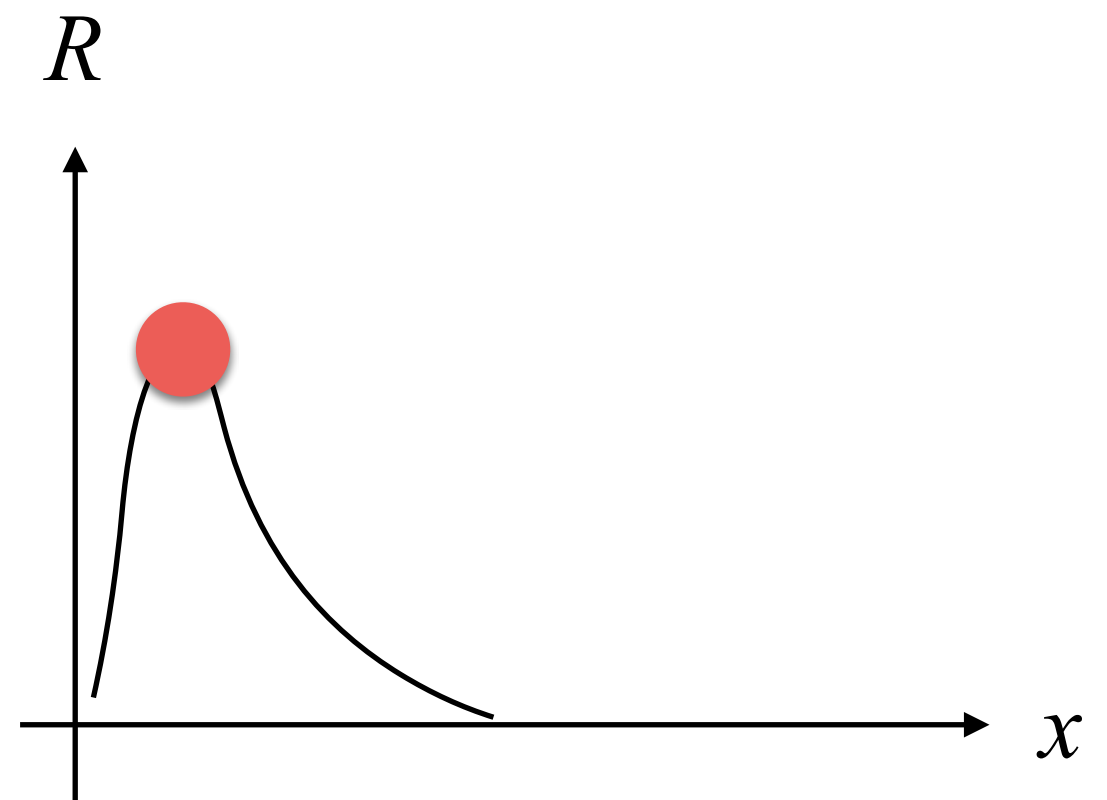
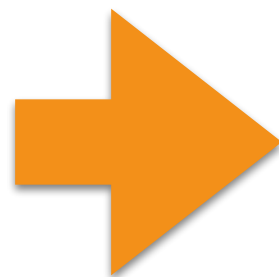


Far Object

Scale Invariant: Stable Corners

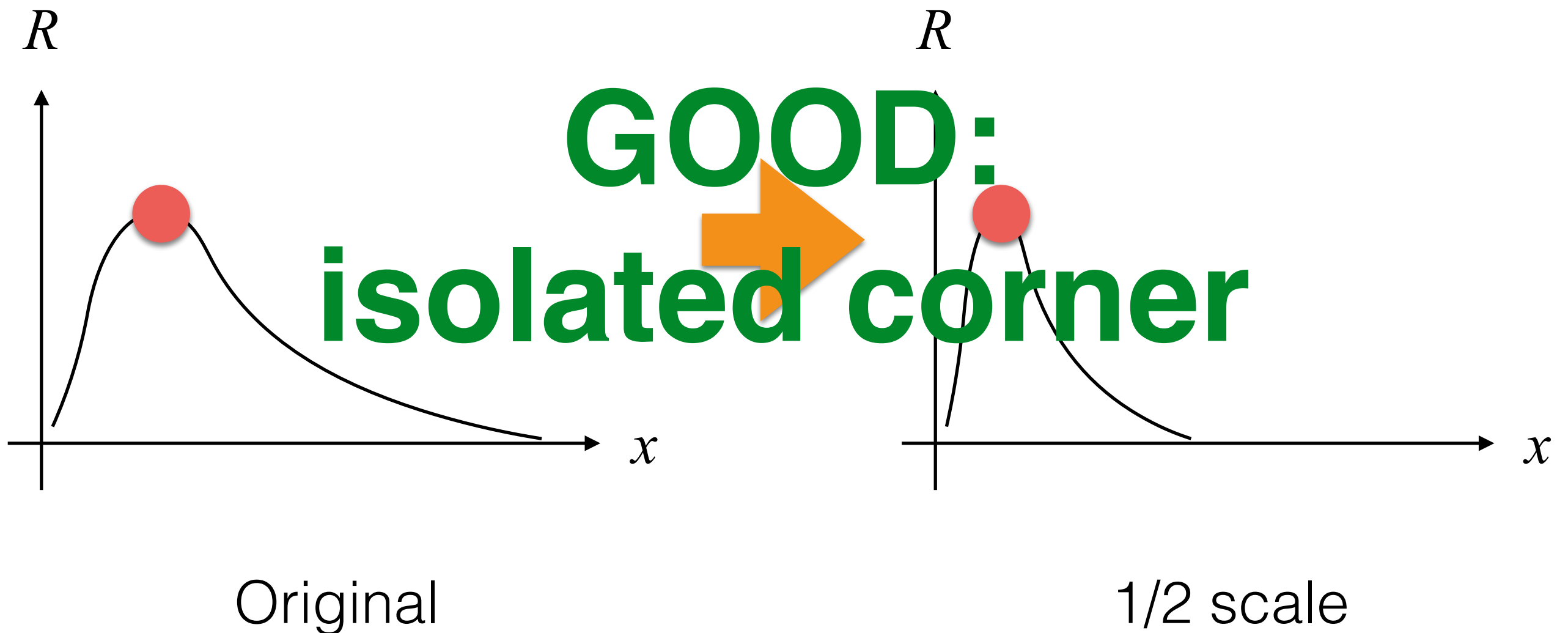


Original

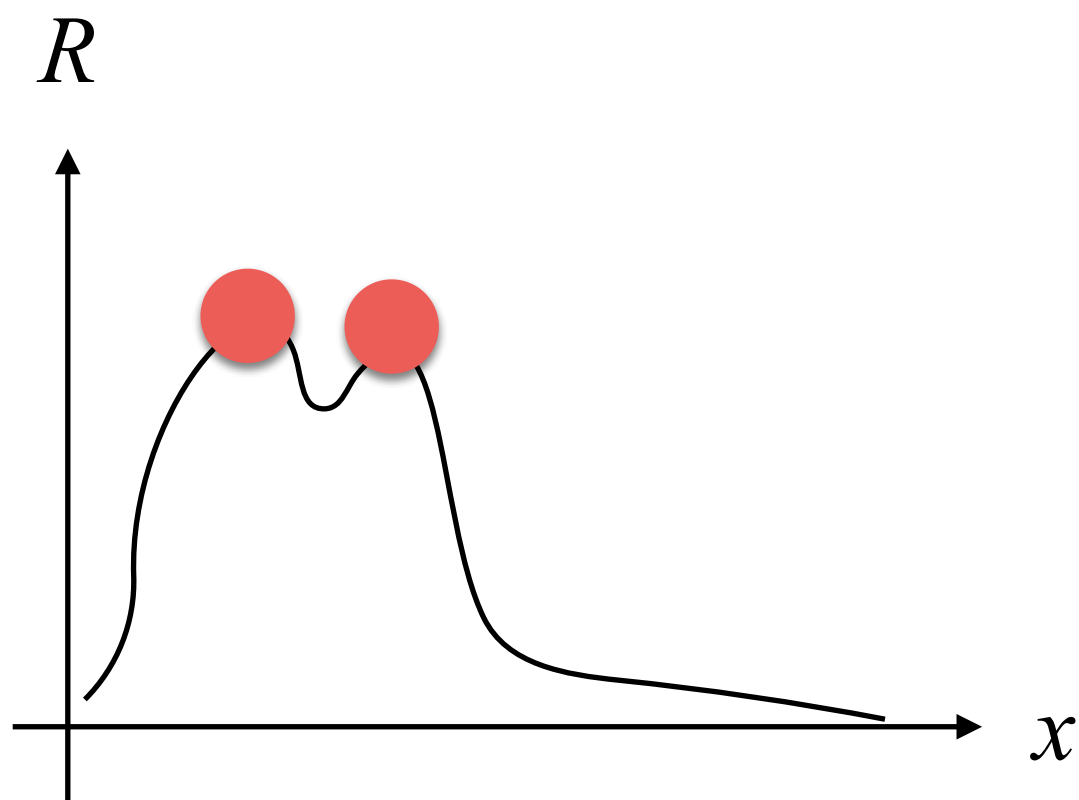


1/2 scale

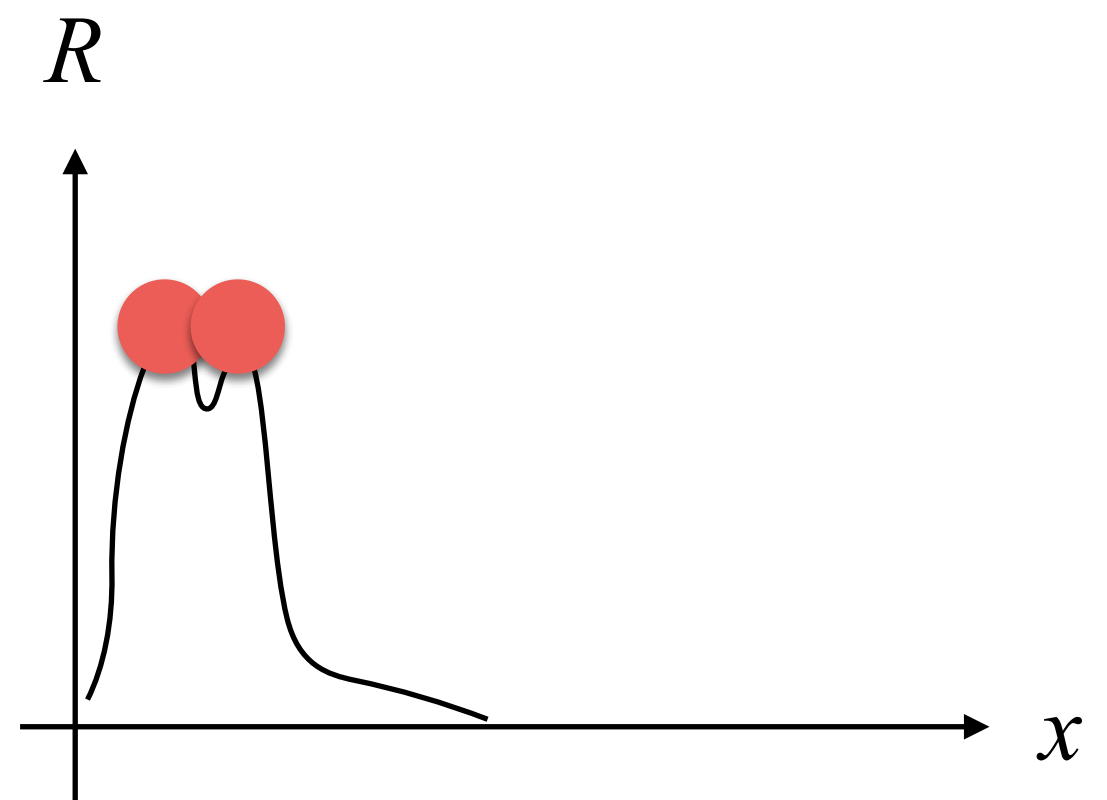
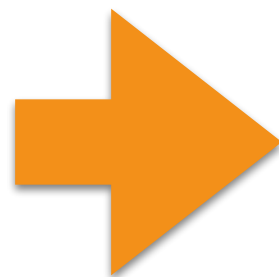
Scale Invariant: Stable Corners



Scale Invariant: Unstable Corners

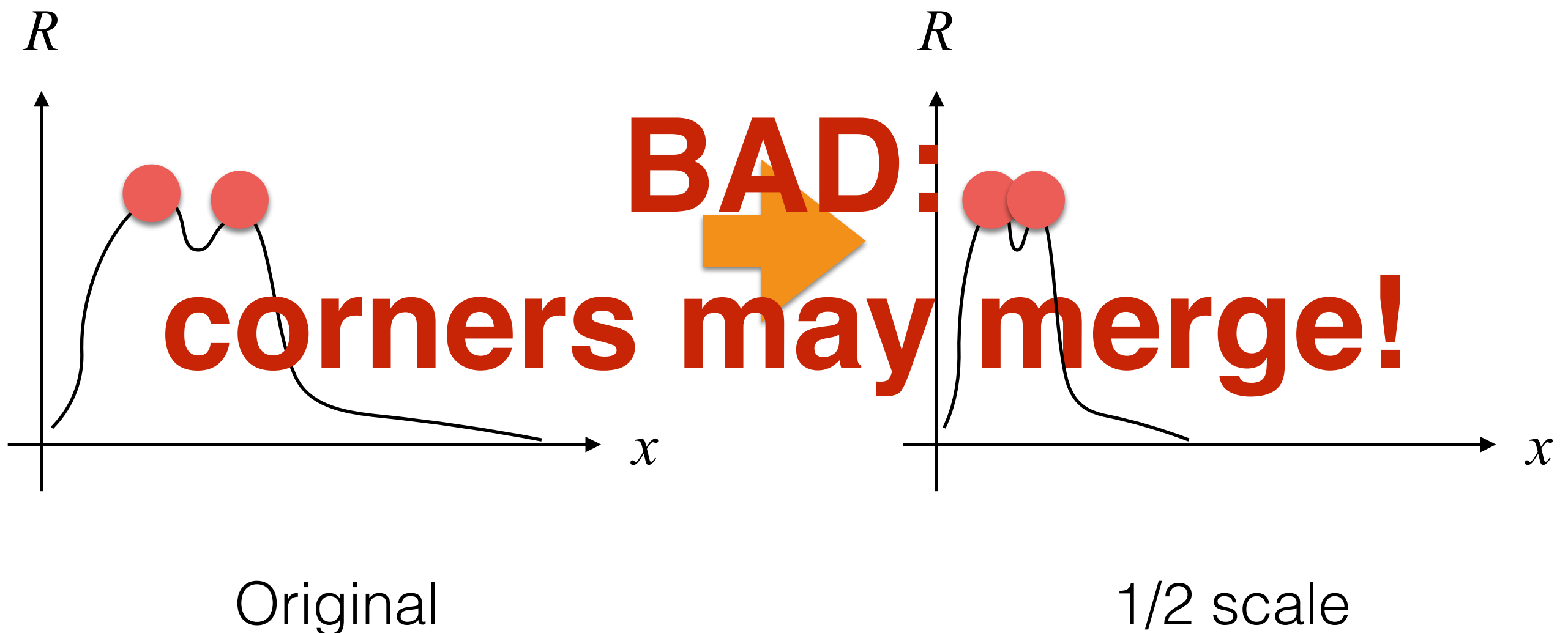


Original



1/2 scale

Scale Invariant: Unstable Corners



Scale Invariant: A Multi-Scale Approach

- Depending on the content of the image:
 - We need to detect the scale of corner.
 - We need to use its scale to vary the size of the window W for computing corners!

Scale Invariant: The Signature Function

- A signature function, s , is a function giving us an idea of the local content of the image, I , around a point with coordinates (x, y) at a given scale σ .
- An example of signature function is the Difference of Gaussians (DoG):

$$s(I, x, y, \sigma) = [I \otimes G(\sigma)](x, y) - [I \otimes G(\sigma \cdot 2)](x, y)$$

- where G is a Gaussian kernel.

Scale Invariant: The Signature Function



-



DoG

Scale Invariant: The Approach



We need to find the right scale for resizing W for each image!

Scale Invariant: The Approach

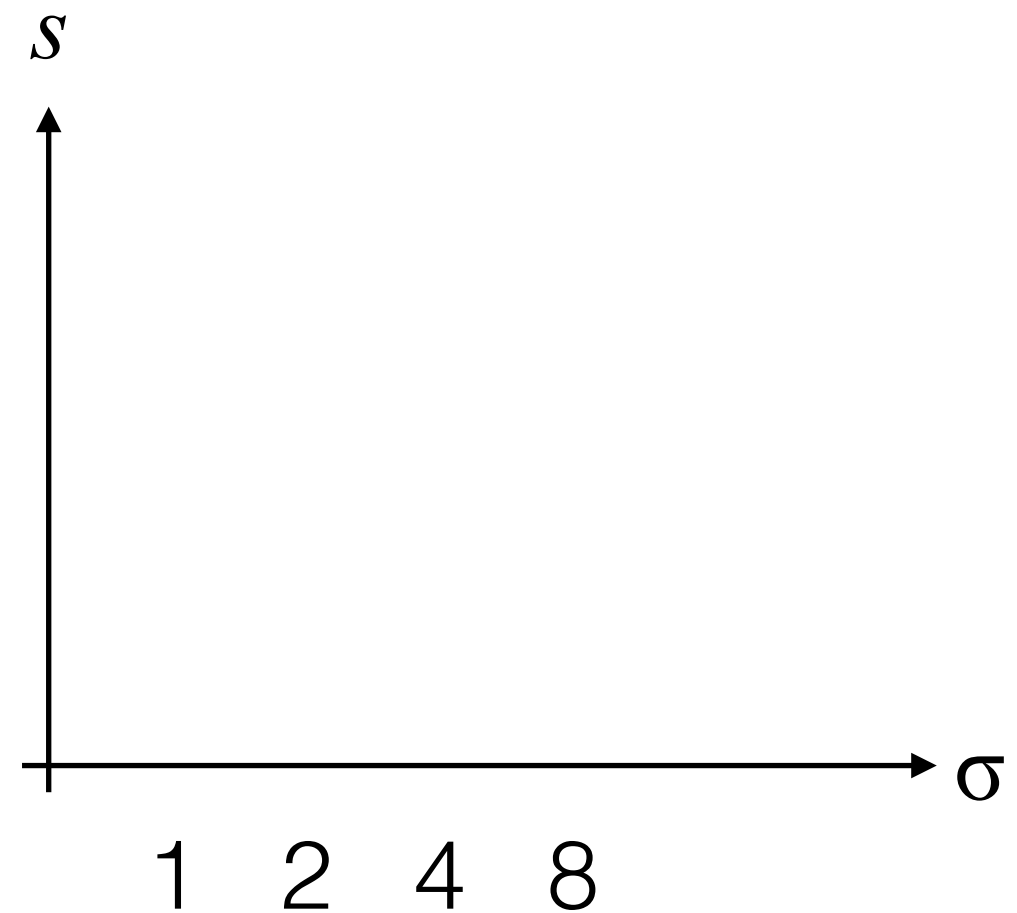
- The signature function, s , can give us an idea of the content of the image.
- Therefore, we need to find a maximum point of s for pixel of an input image!

Scale Invariant: The Approach



Let's build s at the red point!

Scale Invariant: The Approach

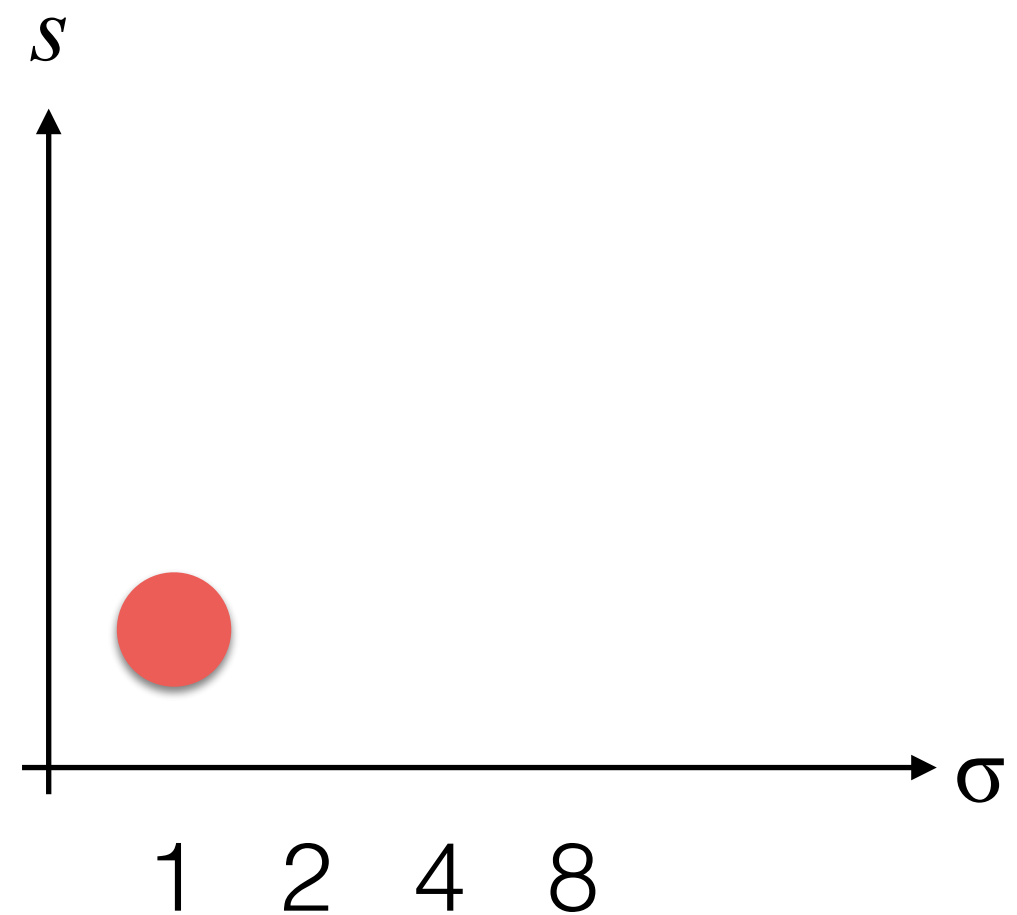


This is our start!

Scale Invariant: The Approach



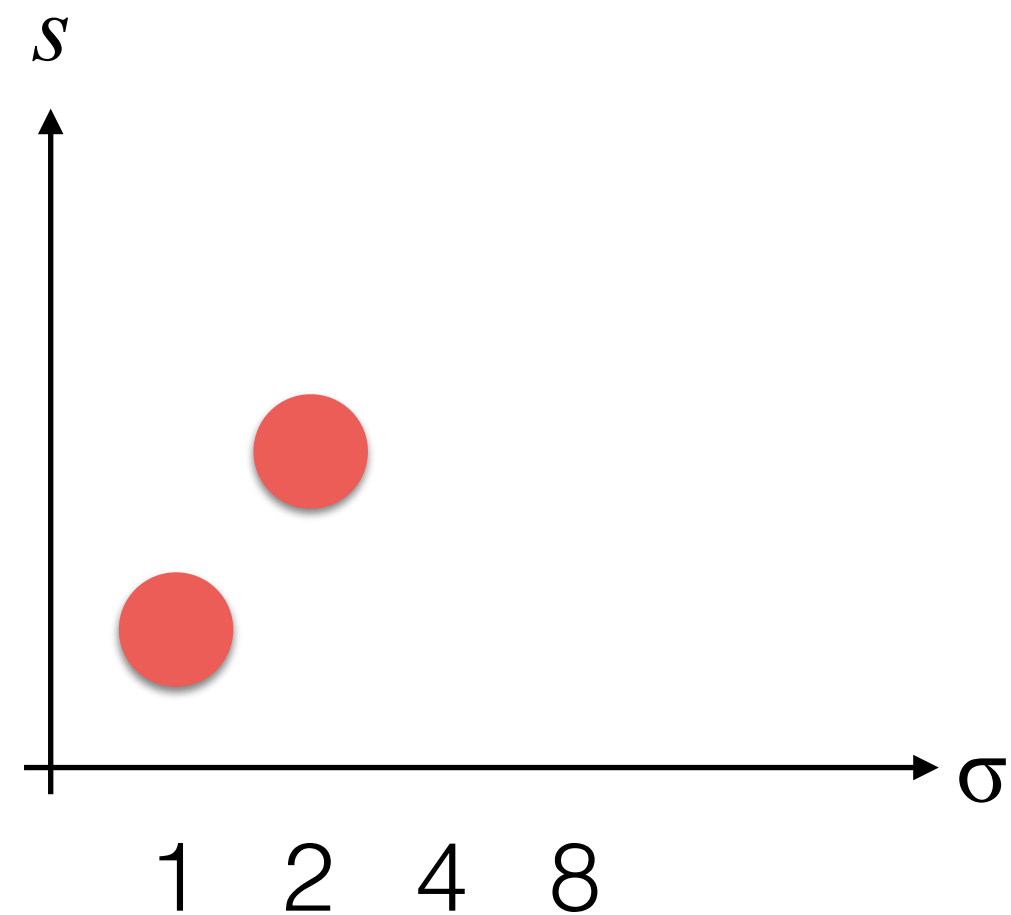
$$\sigma = 1$$



Scale Invariant: The Approach



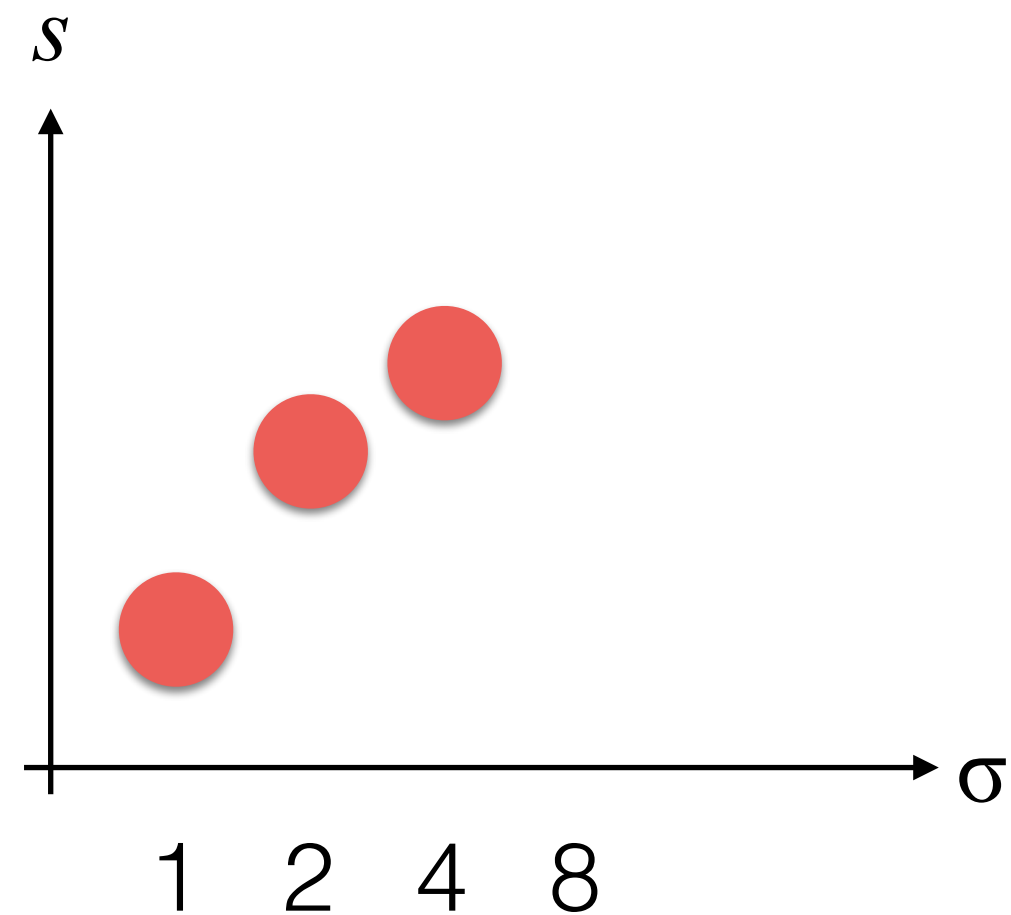
$$\sigma = 2$$



Scale Invariant: The Approach



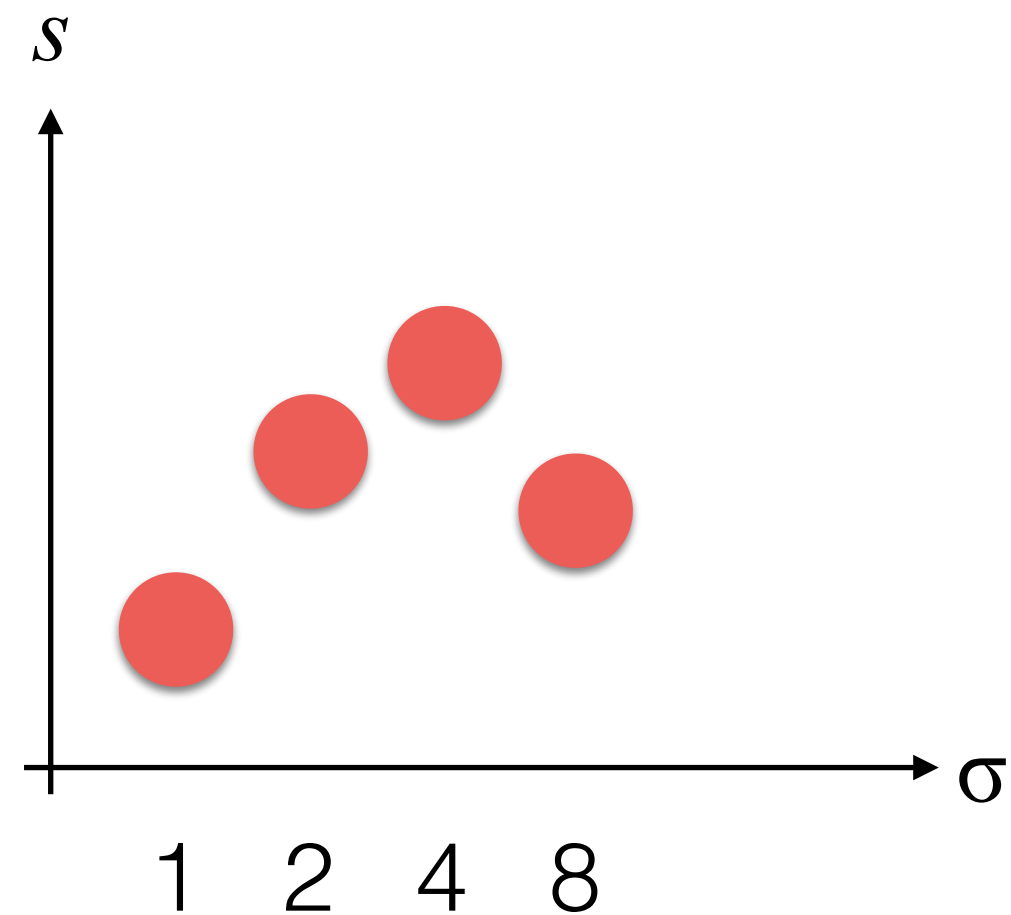
$$\sigma = 4$$



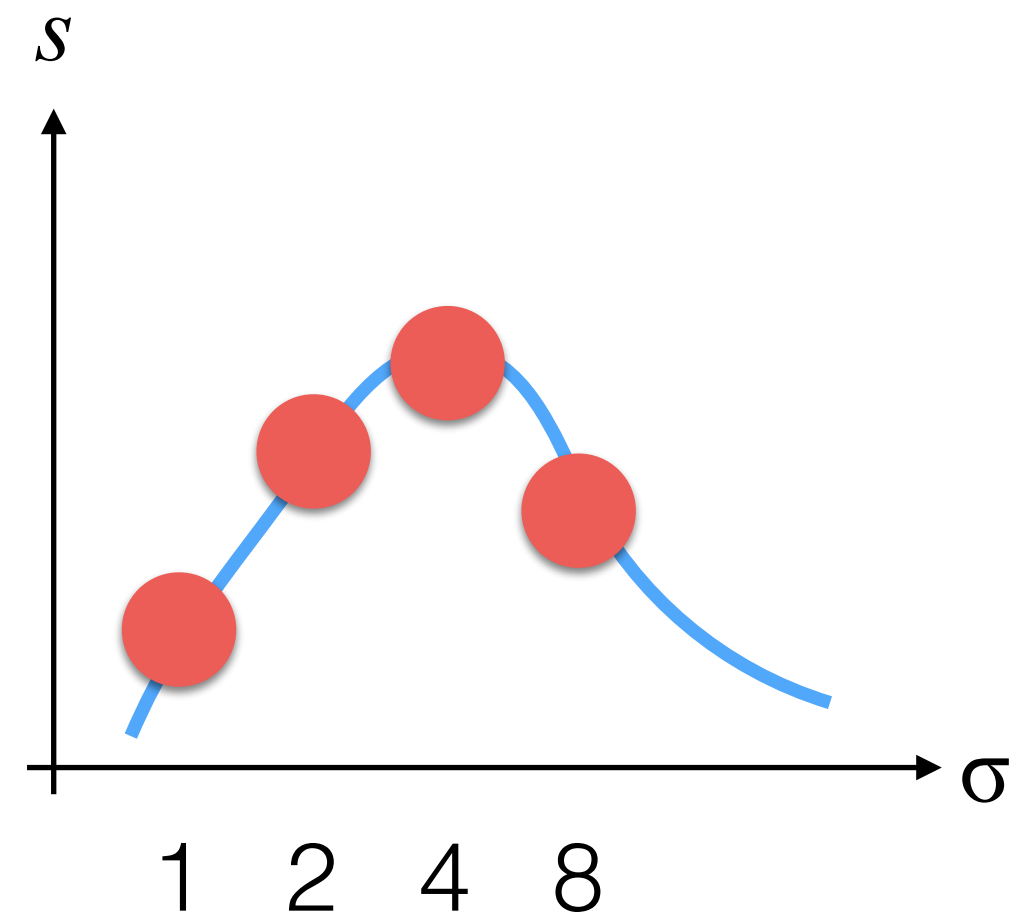
Scale Invariant: The Approach



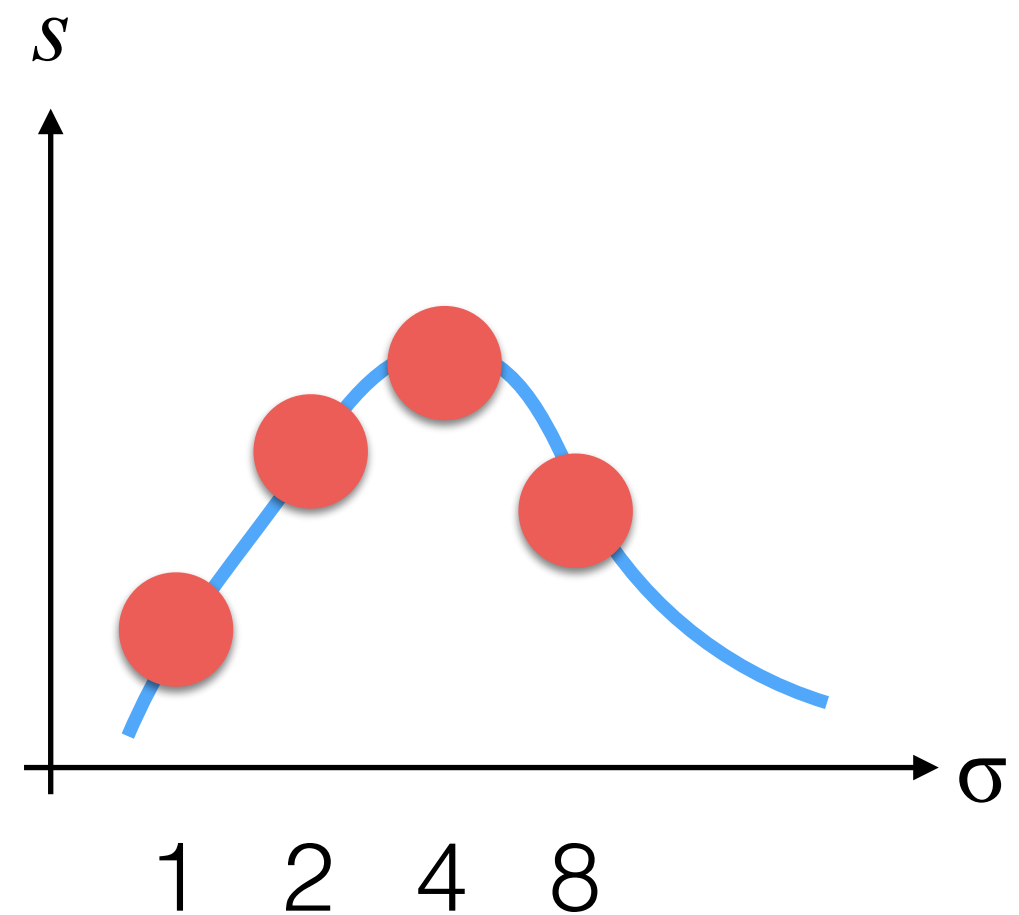
$$\sigma = 8$$



Scale Invariant: The Approach



Scale Invariant: The Approach

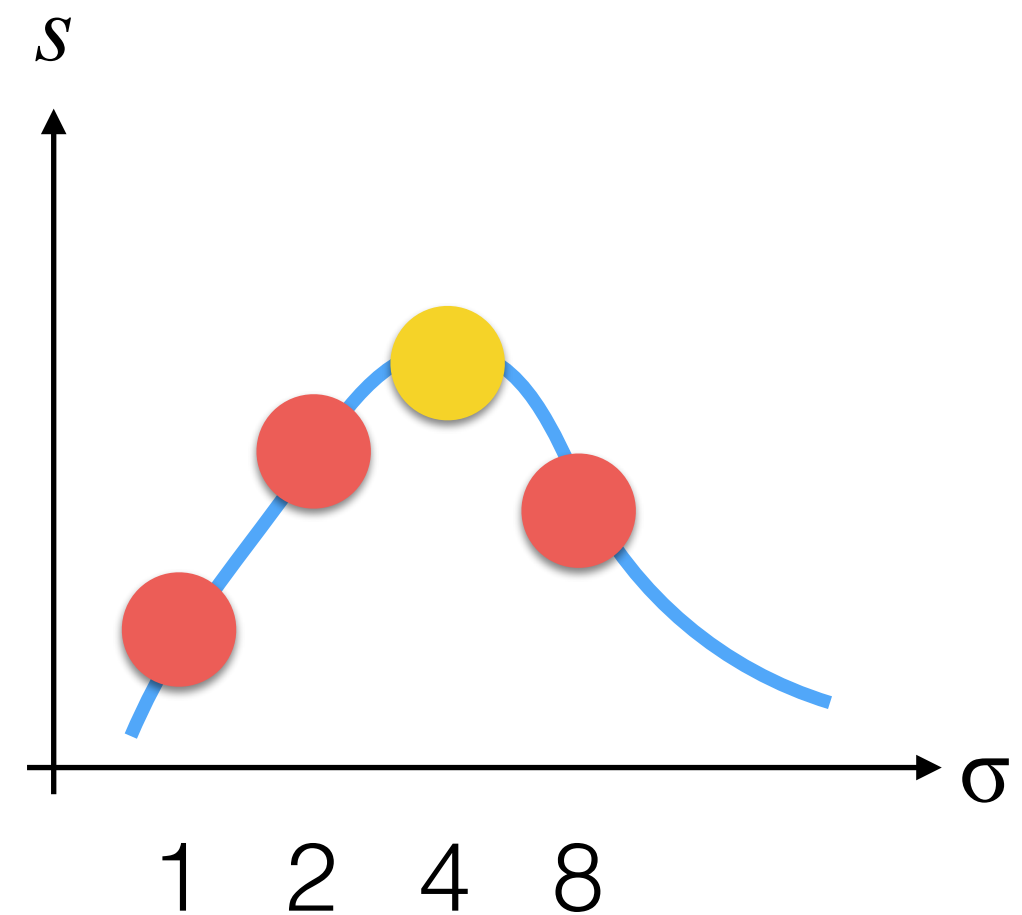


Which is σ for which s is the maximum?

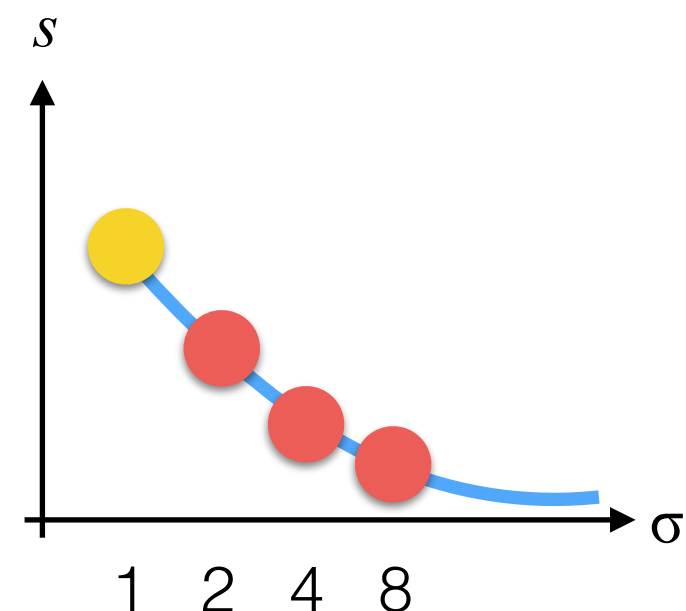
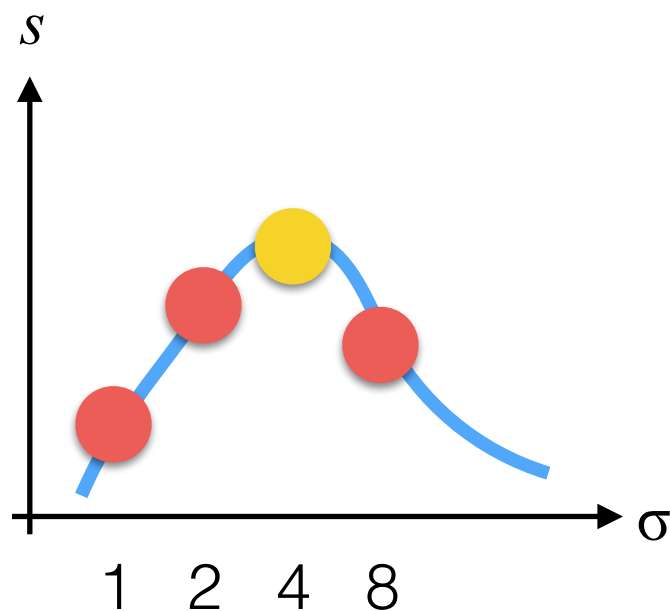
Scale Invariant: The Approach



It is $\sigma = 4$



Scale Invariant: The Approach



Extraction of Features

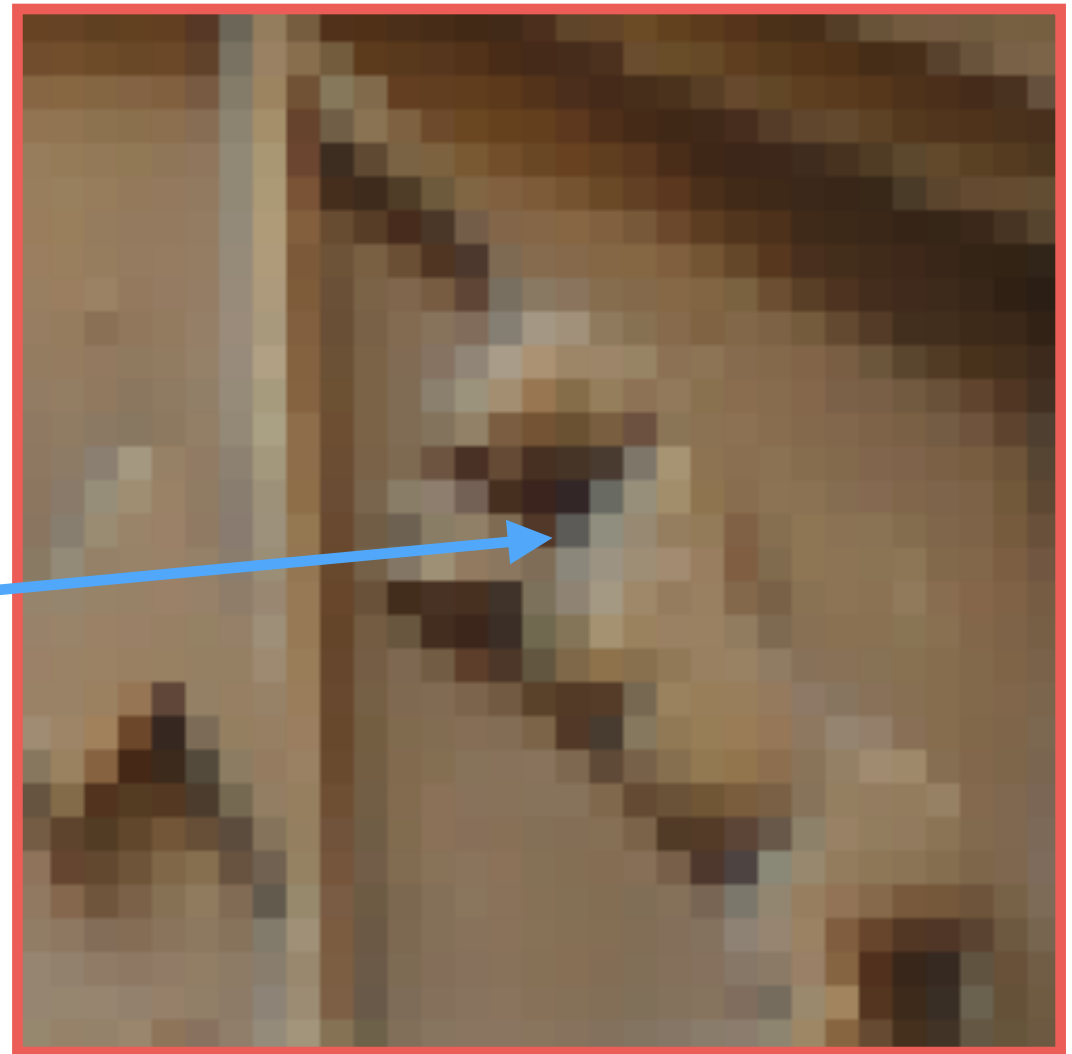
- General overview:
 - We compute the scale for each pixel using the sigma value at which we have the maximum value of the signature function.
 - We compute the Harris Corner using the scale to increase the size of the local window; i.e., the scale of the window will be multiplied by the sigma value.

Feature Descriptors

Feature Descriptors

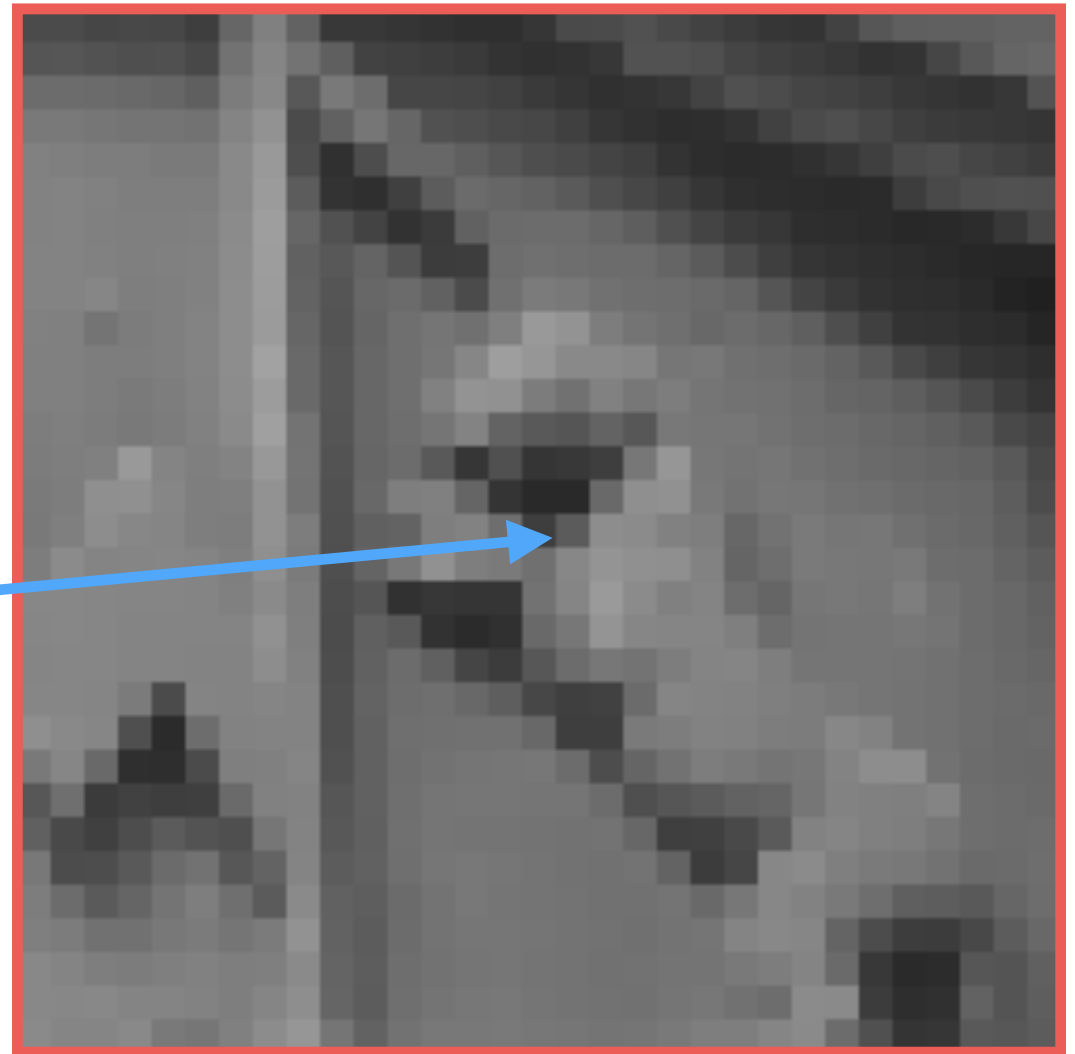
- Once we found our features (i.e., corners), we need to describe in a meaningful and possibly unique way.
- Why?
 - We want compare corners between images in order to find correspondences between images.

Feature Descriptors



A patch, P , is a sub-image centered in a given point (u, v) .

Feature Descriptors



A patch, P , is a sub-image centered in a given point (u, v) .

Feature Descriptors

- There are many local features descriptors in literature:
 - BRIEF/ORB descriptor.
 - SIFT descriptor.
 - SURF descriptor.
 - etc.

Feature Descriptors

- Good properties that we want are invariance to:
 - Illumination changes.
 - Rotation.

BRIEF Descriptor

- The descriptor creates a vector of n binary values:

$$\text{BRIEF}(P) = \mathbf{b} = [0, 1, 0, 0, \dots, 1]^\top$$

- For efficiency, it is encoded as a number:

$$n_{\mathbf{b}} = \sum_{I=1}^n 2^{i-1} b_i$$

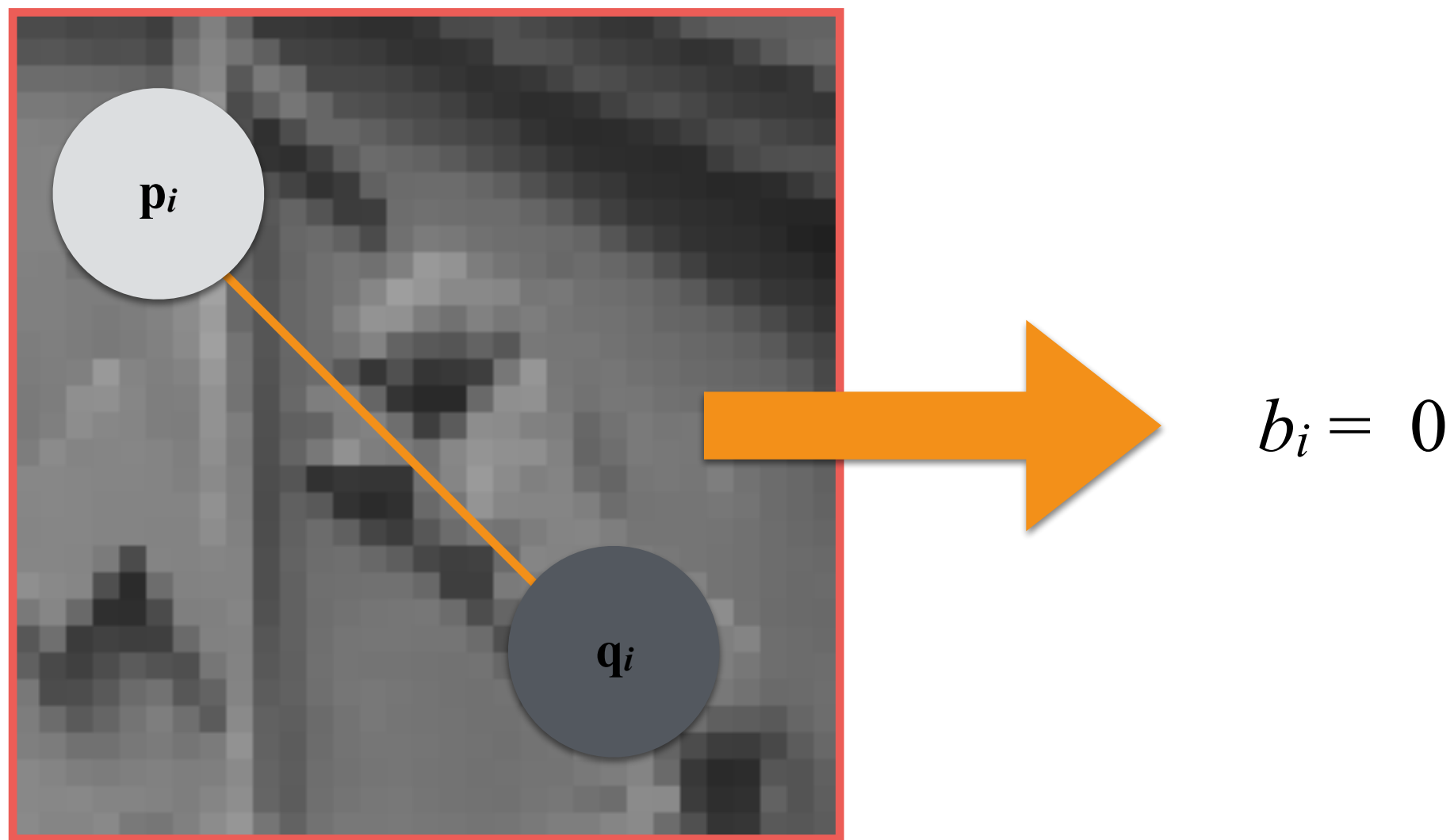
BRIEF Descriptor

- Given a patch, P , of size $S \times S$ an element of $\mathbf{b}$ is defined as

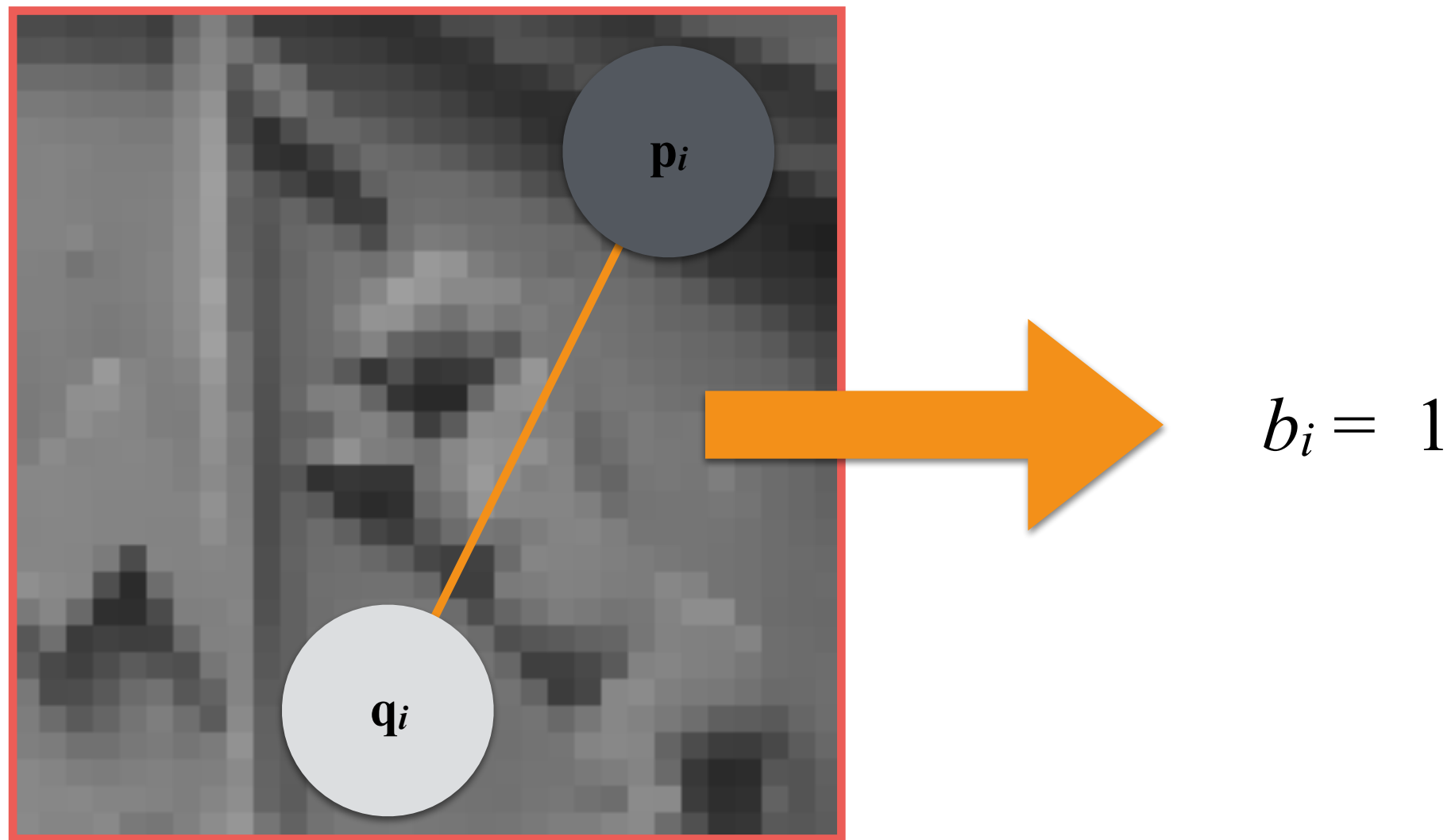
$$b_i(\mathbf{q}_i, \mathbf{p}_i) = \begin{cases} 1 & \text{if } P(\mathbf{p}_i) < P(\mathbf{q}_i), \\ 0 & \text{otherwise} \end{cases}$$

- where $\mathbf{p}_i$ and $\mathbf{q}_i$ are the coordinates (x, y) of two random points in P .

BRIEF Descriptor: Example



BRIEF Descriptor: Example



BRIEF Descriptor: Test

- Let's say we have two descriptor $\mathbf{b}^1$ and $\mathbf{b}^2$. How do we check if they are describing the same corner?
- We count the number of different bits in the two vectors (Hamming distance):

$$D_H(\mathbf{b}^1, \mathbf{b}^2) = \sum_{i=1}^n \neg \text{xor}(b_i^1, b_i^2)$$

- **Higher the closer:**
 - This is a very computationally efficient distance function.

BRIEF Descriptor: Test

A	B	A XOR B = (NOT A) AND B	NOT (A XOR B)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

BRIEF Descriptor: Evil Details

- Optimal n is 256; from experiments testing different lengths: 16, 32, 64, 128, 256, and 512.
- Points distribution:

- Uniform distribution in P .

- $(\mathbf{p}_i, \mathbf{q}_i) \sim \text{i.i.d. Gaussian}\left(0, \frac{S^2}{25}\right)$

- Points are pre-computed generating a set:

$$A = \begin{bmatrix} \mathbf{p}_0, & \mathbf{p}_1, & \dots & \mathbf{p}_n \\ \mathbf{q}_0, & \mathbf{q}_1, & \dots & \mathbf{q}_n \end{bmatrix}$$

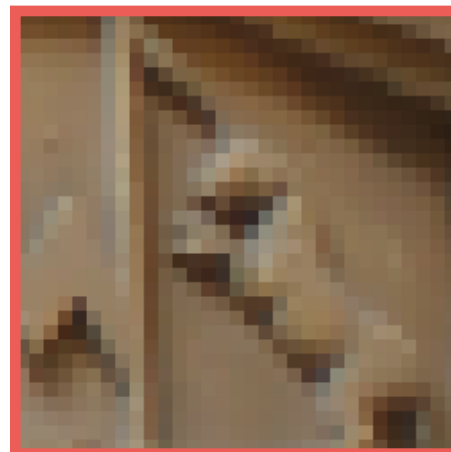
- This set is used for the extraction of all descriptors in all photos!
 - If this is not done, we cannot do comparisons because we are comparing different tests!

BRIEF Descriptor

- Advantages:
 - Computationally fast.
 - Invariant to illumination changes.
 - Compact!
 - Patent free.
- Disadvantage:
 - Rotation is an issue!

BRIEF Descriptor

- Advantages:
 - Computationally fast.
 - Invariant to illumination changes.
 - Compact!
 - Patent free.
- Disadvantage:
 - Rotation is an issue!



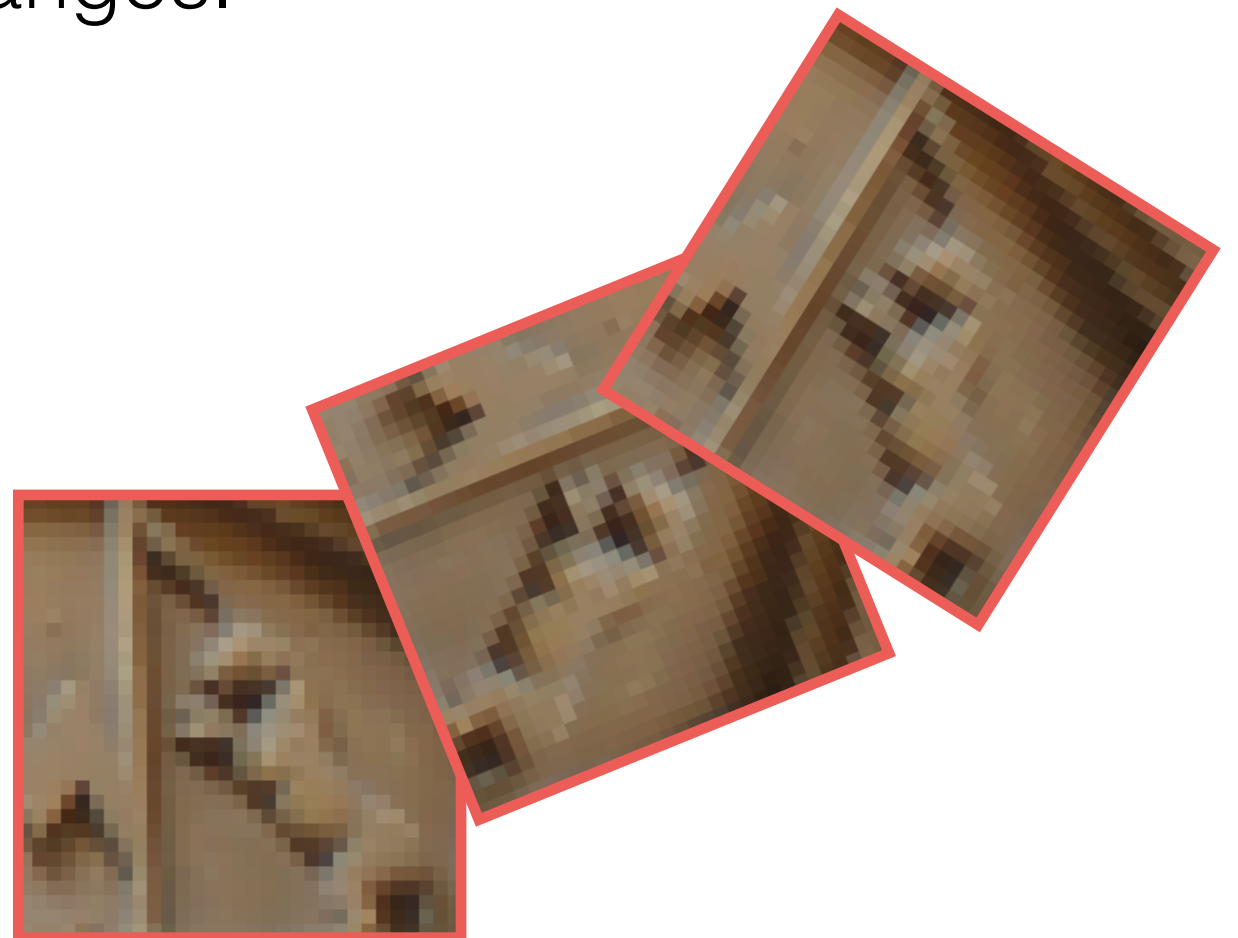
BRIEF Descriptor

- Advantages:
 - Computationally fast.
 - Invariant to illumination changes.
 - Compact!
 - Patent free.
- Disadvantage:
 - Rotation is an issue!



BRIEF Descriptor

- Advantages:
 - Computationally fast.
 - Invariant to illumination changes.
 - Compact!
 - Patent free.
- Disadvantage:
 - Rotation is an issue!



ORB Descriptor

- The descriptor is a modified version of BRIEF and it can handle rotations!
- The first step of the algorithm is to compute the orientation of the current patch P .

ORB Descriptor: Patch Orientation

- We compute the patch orientation using Rosin moments of a patch:

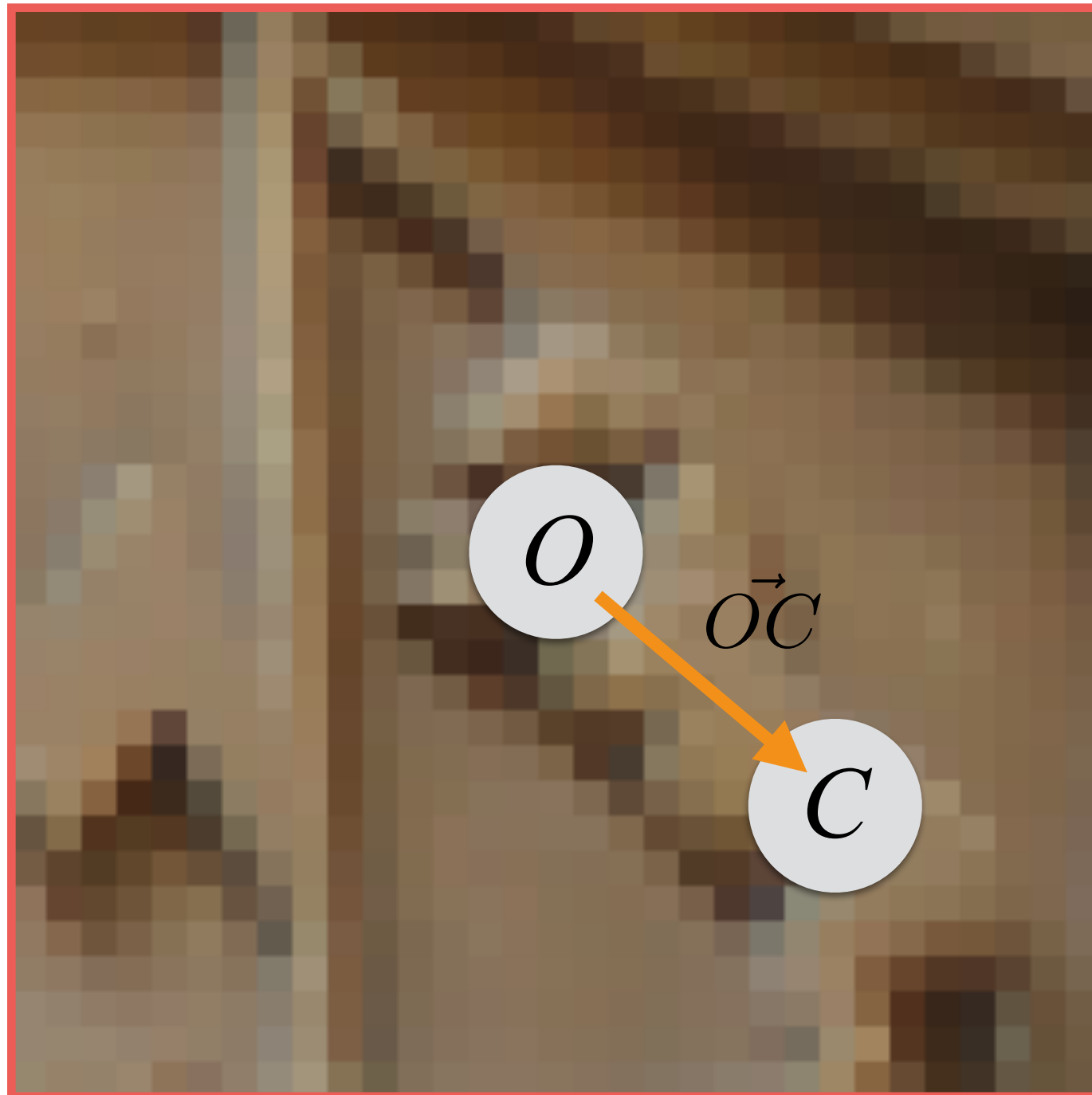
$$m_{a,b} = \sum_{x,y \in P} x^a y^b P(x,y)$$

- From this, we define the centroid, C , as

$$C = \left(\frac{m_{1,0}}{m_{0,0}}, \frac{m_{0,1}}{m_{0,0}} \right)$$

- Now, we can create a vector from corner's center, O , to the centroid, C .

ORB Descriptor: Patch Orientation



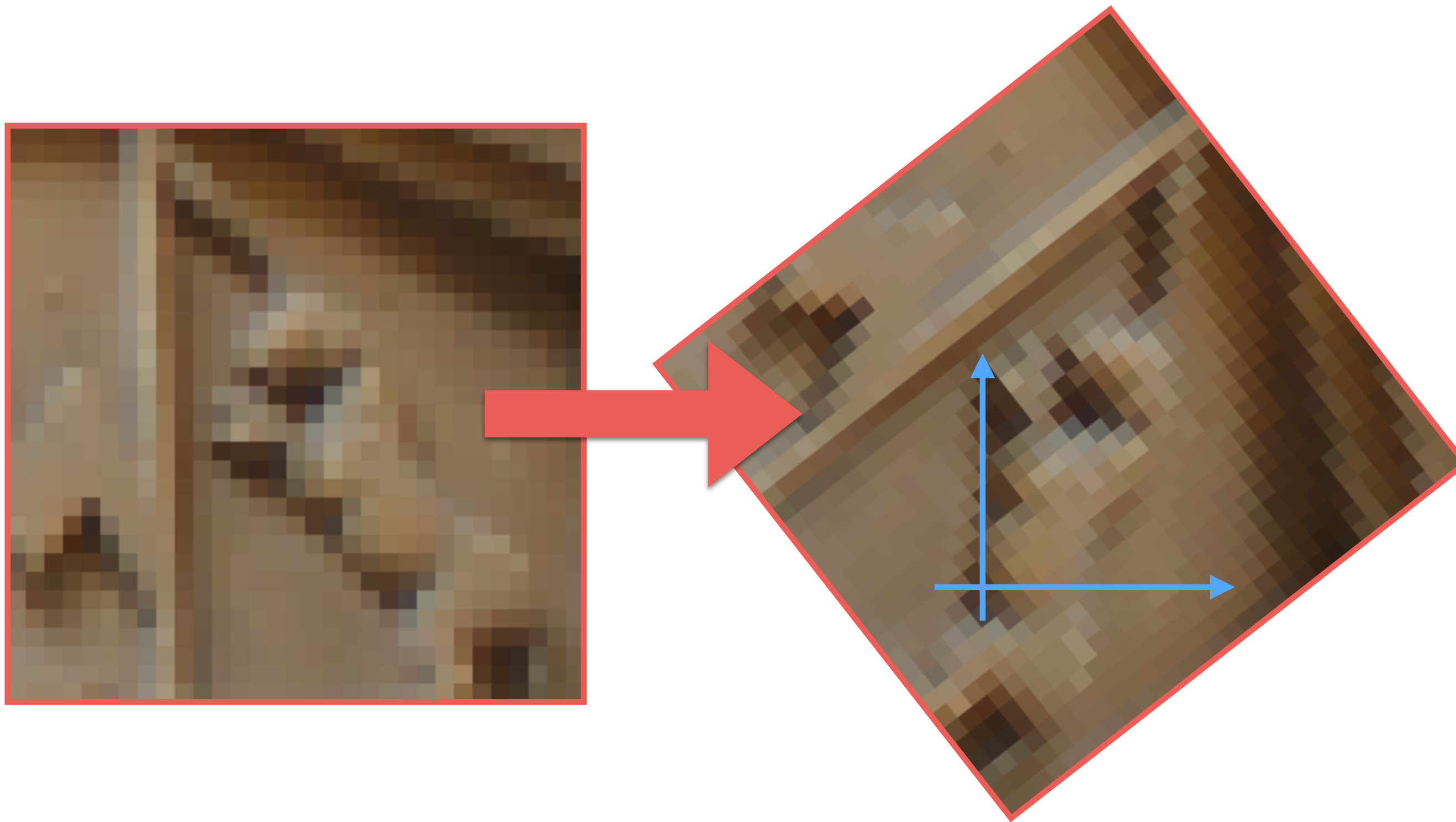
ORB Descriptor: Patch Orientation

- From this vector, the orientation of the patch can be computed simply as

$$\theta = \text{atan2}(m_{0,1}, m_{1,0})$$

- From this, we can rotate the patch P , but this operation is very computationally expensive:
 - We need to rotate each single point in the patch!

ORB Descriptor



ORB Descriptor: Patch Orientation

- Instead of rotating the whole patch, we can rotate only the points stored in A as

$$A_{\theta} = R_{\theta} \cdot A$$

- where R_{θ} is a 2D rotation matrix.
- **NOTE:** we need to rotate less points!

ORB Descriptor

- Advantages:
 - Computationally fast.
 - Invariant to illumination changes.
 - Compact!
 - Invariant to rotation.
 - Patent free.
- Disadvantage:
 - Not robust as SIFT.

SIFT Descriptor

- It is the state-of-the-art descriptor.
- It was introduced in 1999, but it is still the king.

SIFT Descriptor: Patch Orientation

- The first step is to compute the orientation of P .
- We compute the horizontal (P_x) and vertical (P_y) gradients of the P .
- For each pixel at coordinates (i, j) in the patch we compute its orientation and magnitude:

$$m(i, j) = \sqrt{P_x(i, j)^2 + P_y(i, j)^2}$$

$$\theta(i, j) = \text{atan2}\left(P_y(i, j), P_x(i, j)\right)$$

SIFT Descriptor: Patch Orientation

- A histogram, H , of directions (**18** bins) is created for each orientation taking into account its magnitude.
- Let's say we have a gradient with $m = 10$ and $\theta = 45^\circ$. How do we insert it in the histogram H ?

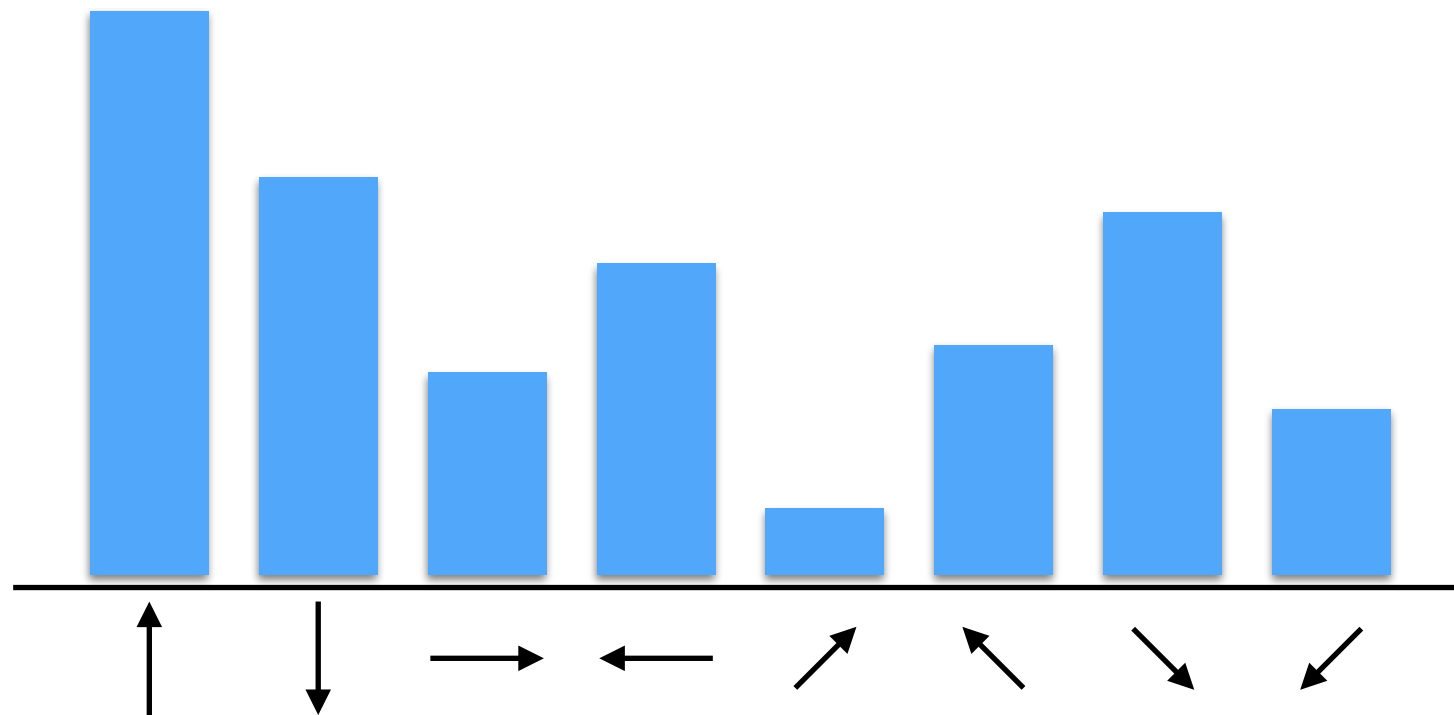
- First, we compute the index of the bin to update:

$$i = \left\lfloor \frac{45}{20} \right\rfloor = 2$$

- Then, we update H as $H(i) = H(i) + 10$
- We repeat this process for all gradients in the patch!

SIFT Descriptor: Patch Orientation

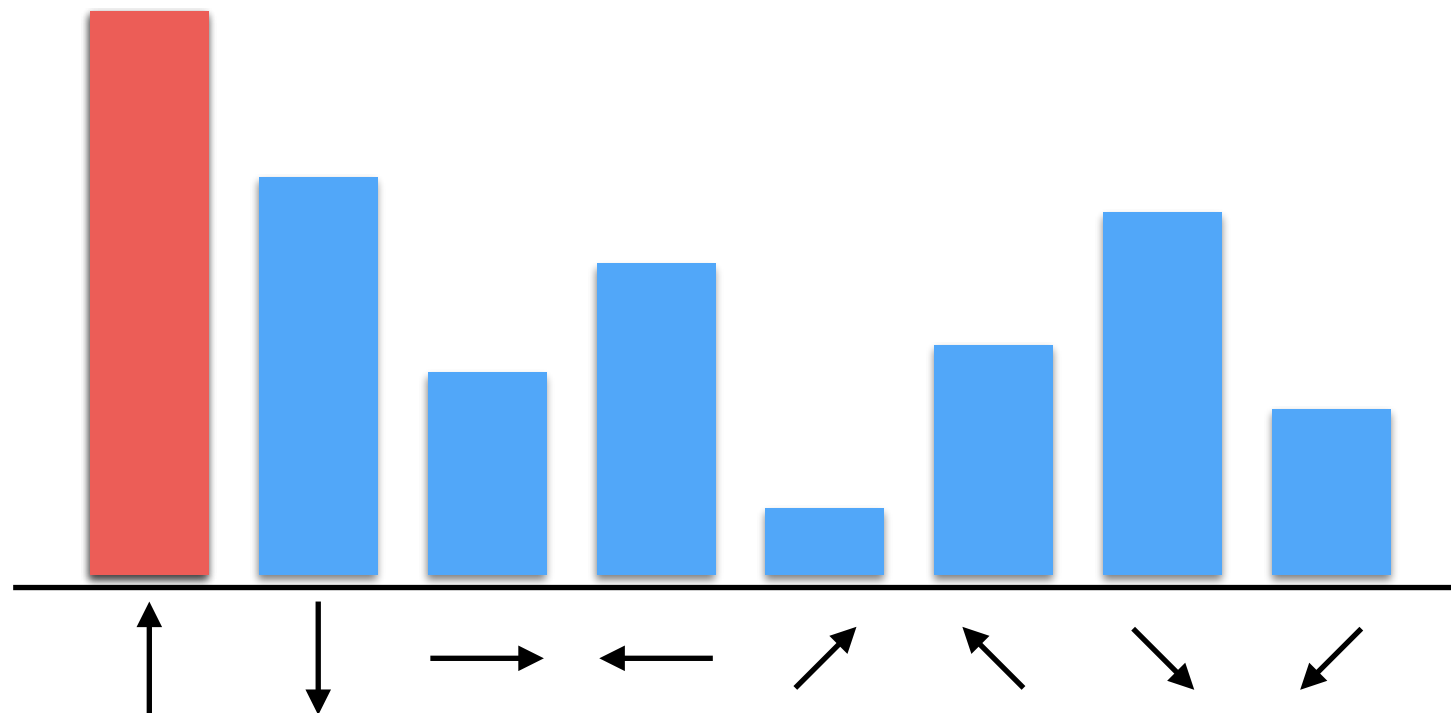
- Finally, we get this (an example with 8 bins; i.e., 8 directions):



- The patch orientation, θ , is given by the highest peak:
 - If we have two equal peaks, we take the as winner the first one in histogram.

SIFT Descriptor: Patch Orientation

- Finally, we get this (an example with 8 bins; i.e., 8 directions):

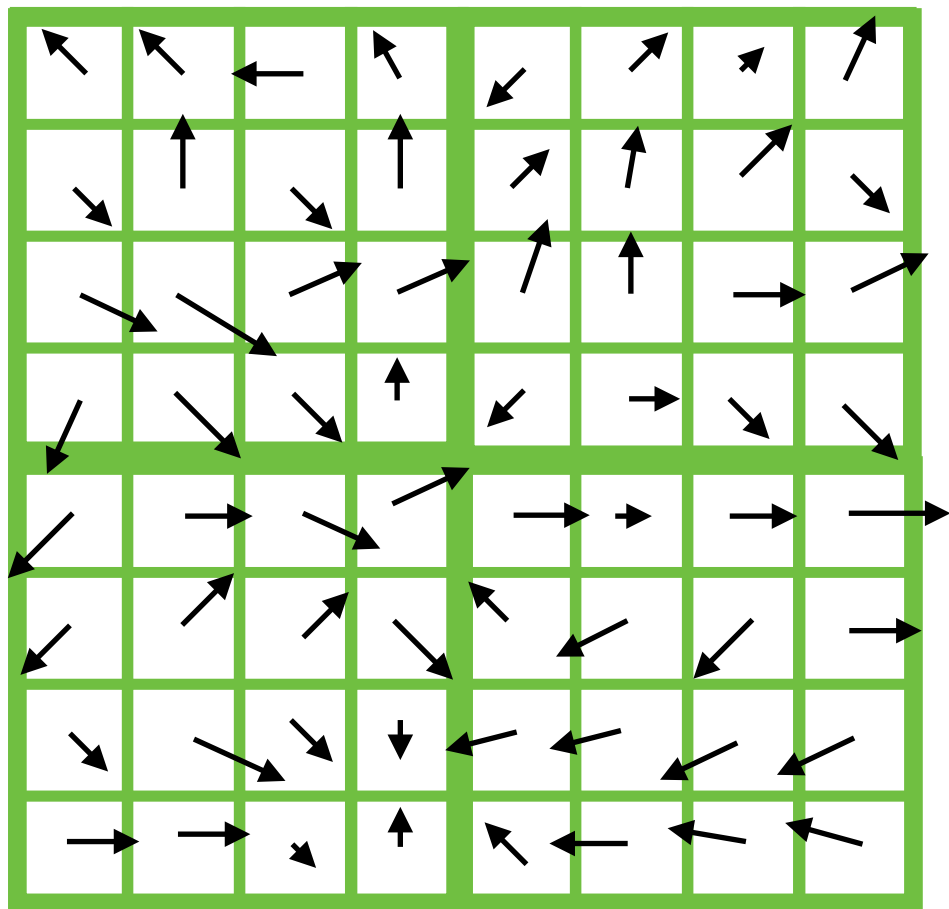


- The patch orientation, θ , is given by the highest peak:
 - If we have two equal peaks, we take the as winner the first one in histogram.

SIFT Descriptor

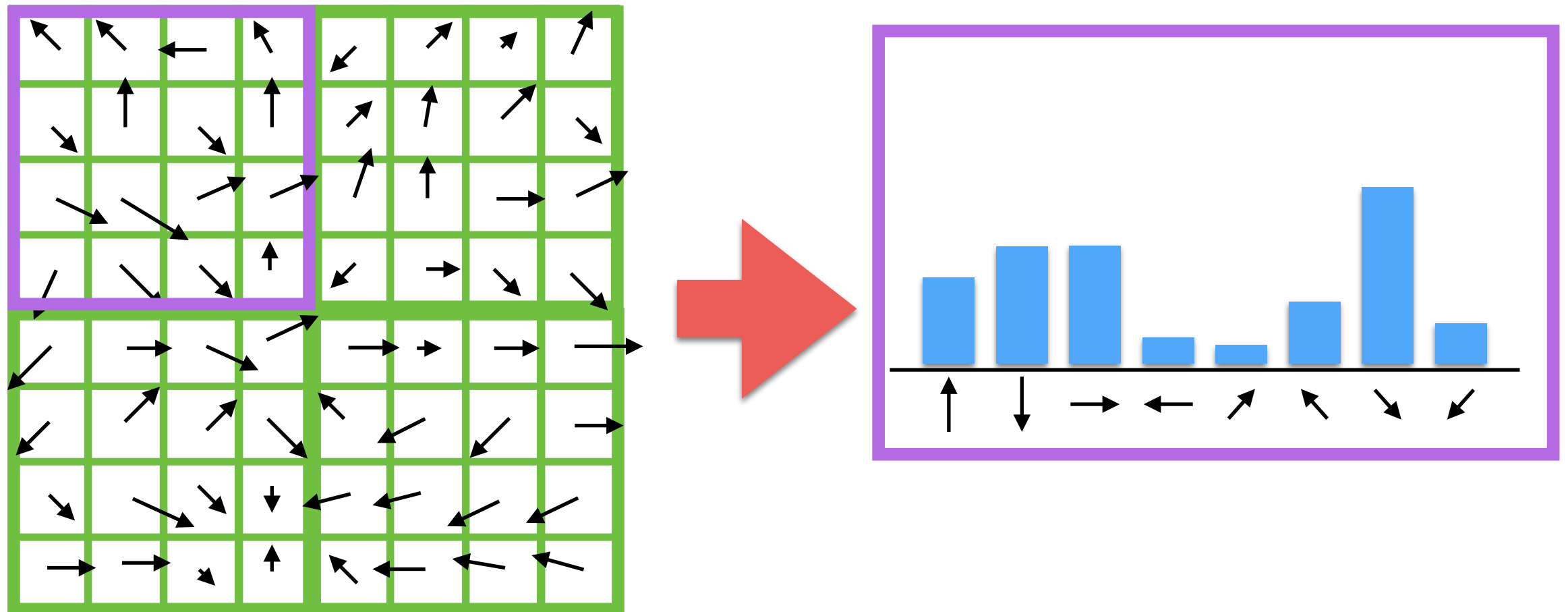
- Once we have θ , we can rotate all gradients in the patch using it.
 - This ensures to be invariant to rotations!
- At this point, we divide the patch into 4x4 blocks. For each block, we compute a histogram of directions as before for computing θ .
- The final SIFT descriptor is the concatenation (flattening) of all these histograms.

SIFT Descriptor: Example with 2x2 Blocks



Patch and its gradients

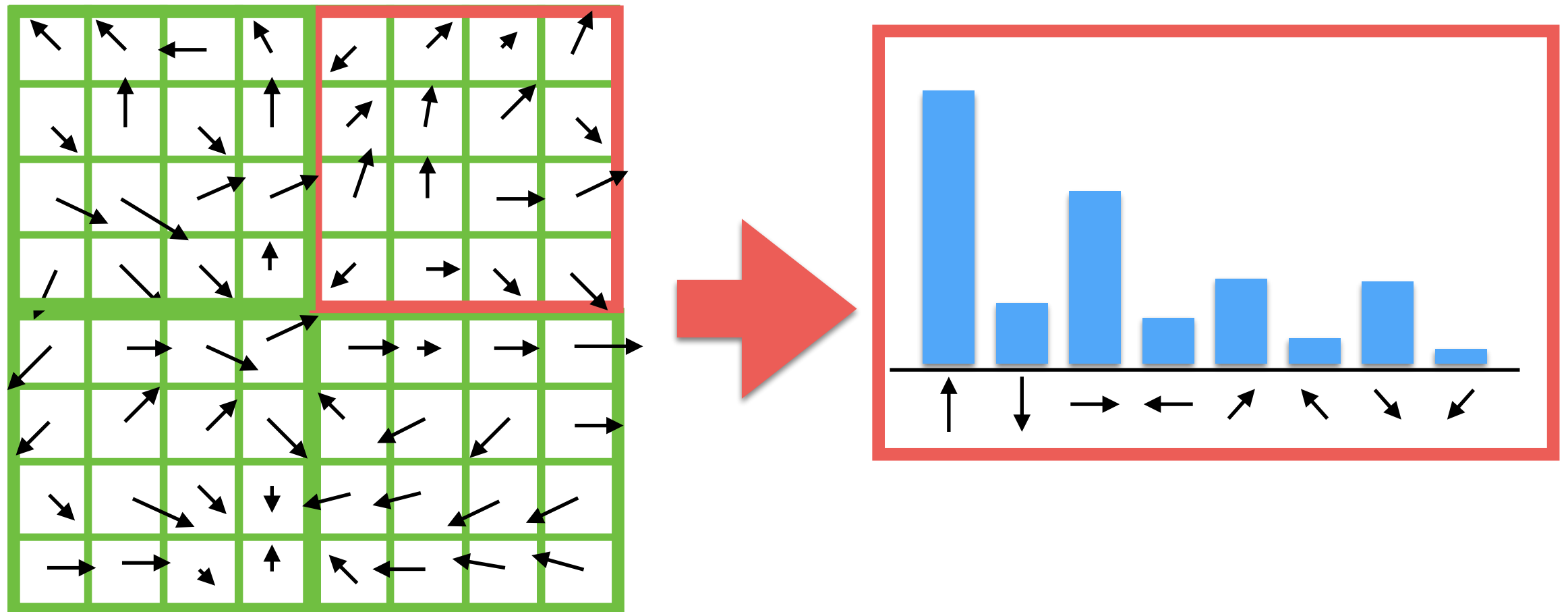
SIFT Descriptor: Example with 2x2 Blocks



Patch and its gradients

We compute the histogram for the first block in violet

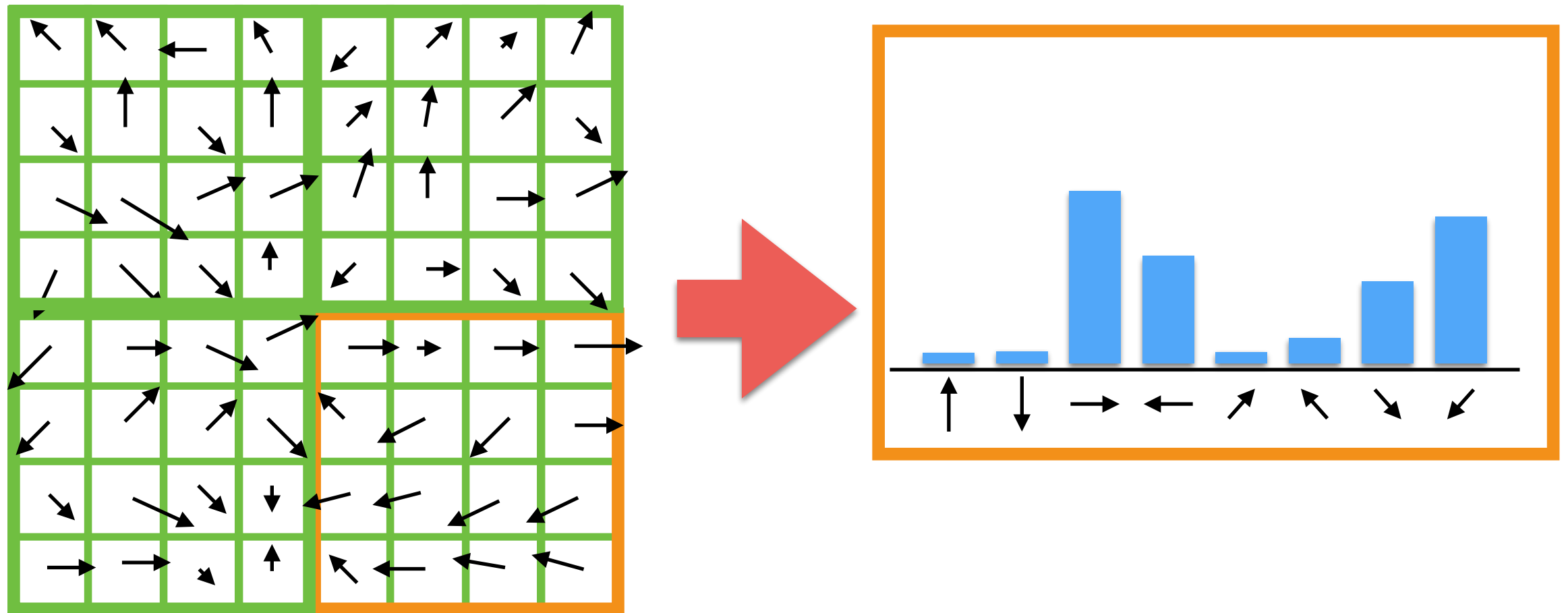
SIFT Descriptor: Example with 2x2 Blocks



Patch and its gradients

We compute the histogram for the second block in red

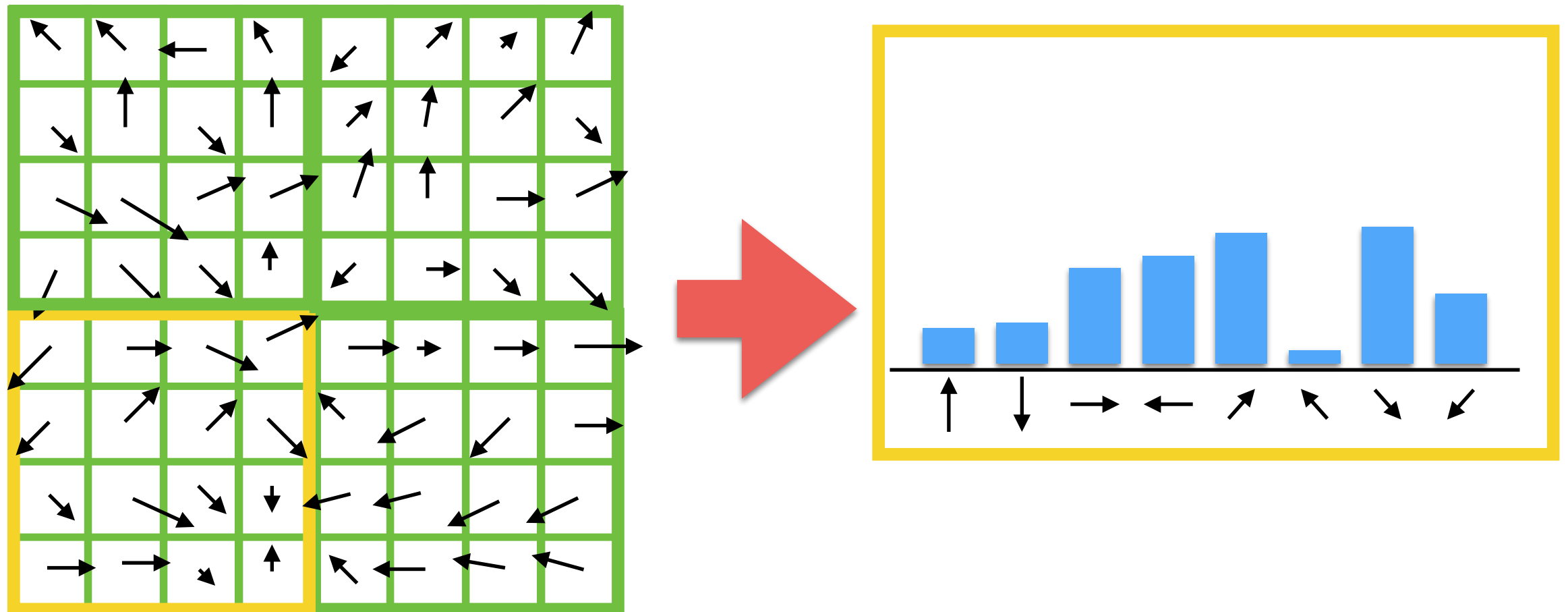
SIFT Descriptor: Example with 2x2 Blocks



Patch and its gradients

We compute the histogram for the third block in orange

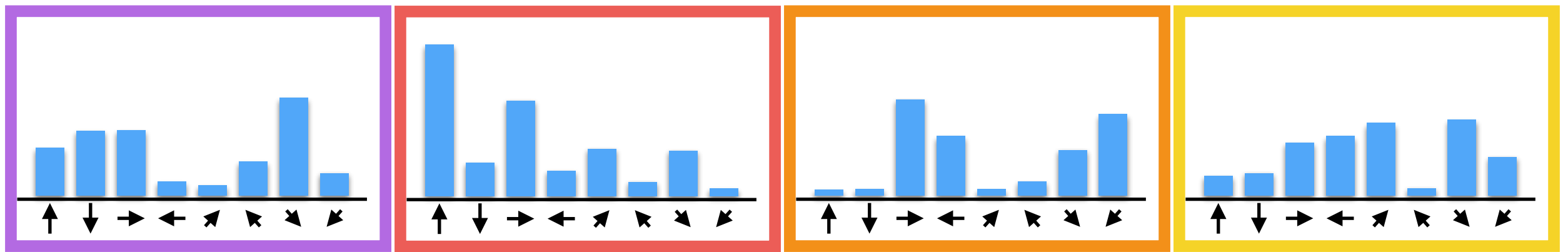
SIFT Descriptor: Example with 2x2 Blocks



Patch and its gradients

We compute the histogram for the four block in yellow

SIFT Descriptor: Example with 2x2 Blocks



The ***final descriptor*** is the concatenation of the histogram of all blocks.
Note that this can be encoded as a vector; in this example the vector has size equal to:

$$4 \times 8 = \mathbf{32}$$

4 because we have 2x2 Blocks

8 because we have 8 direction for each histogram.

SIFT Descriptor: Test

- We test the differences as distance between histograms:

$$D_2(\mathbf{h}^1, \mathbf{h}^2) = \sqrt{\sum_{i=1}^n (h_i^1 - h_i^2)^2}$$

- **The lower the closer:**
 - This is the opposite compared to BRIEF/ORB.

SIFT Descriptor

- Advantages:
 - Invariant to illumination changes.
 - Invariant to rotation.
- Disadvantages:
 - Slower than BRIEF/ORB.
 - More memory than binary methods.
 - Patented!

Matching Images

Matching: An Image Against Another One

- **Input:** two descriptor lists (*they can be of equal or different size*), **desc**₁ and **desc**₂, respectively of image I_1 and I_2 .
- **Output:** a vector with indices of matches for each list:
 - The output is called **M**₁₂ if we match I_1 against I_2
 - The output is called **M**₂₁ if we match I_2 against I_1

Matching:

How the Output is Encoded Example 1

- Let's say we have 4 descriptors in **desc**₁
- Let's say we have 3 descriptors in **desc**₂
- Let's say that we want to match I_1 against I_2 , this means that we want to compute **M**₁₂.

Matching: Example 1

$$\mathbf{desc}_1 = \begin{bmatrix} d_1^1 \\ d_2^1 \\ d_3^1 \\ d_4^1 \end{bmatrix}$$

$$\mathbf{desc}_2 = \begin{bmatrix} d_1^2 \\ d_2^2 \\ d_3^2 \end{bmatrix}$$

$$\mathbf{M}_{12} = [\]$$

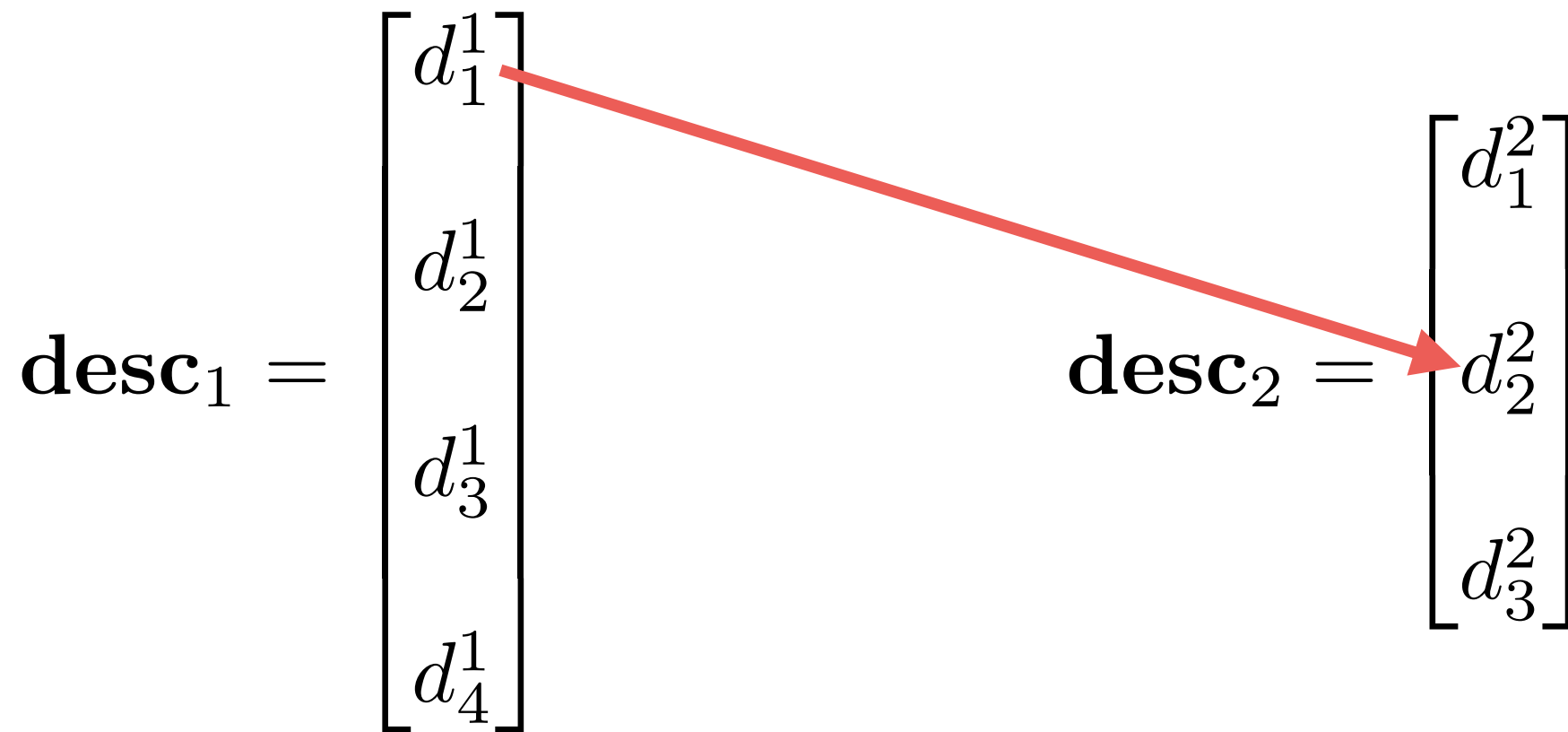
Matching: Example 1

$$\mathbf{desc}_1 = \begin{bmatrix} d_1^1 \\ d_2^1 \\ d_3^1 \\ d_4^1 \end{bmatrix} \qquad \mathbf{desc}_2 = \begin{bmatrix} d_1^2 \\ d_2^2 \\ d_3^2 \end{bmatrix}$$

We find out that the first descriptor of $\mathbf{desc}_1$ matches with the second descriptor of $\mathbf{desc}_2$.

$$\mathbf{M}_{12} = [\quad]$$

Matching: Example 1



We find out that the first descriptor of $\mathbf{desc}_1$ matches with the second descriptor of $\mathbf{desc}_2$.

$$\mathbf{M}_{12} = [2,]$$

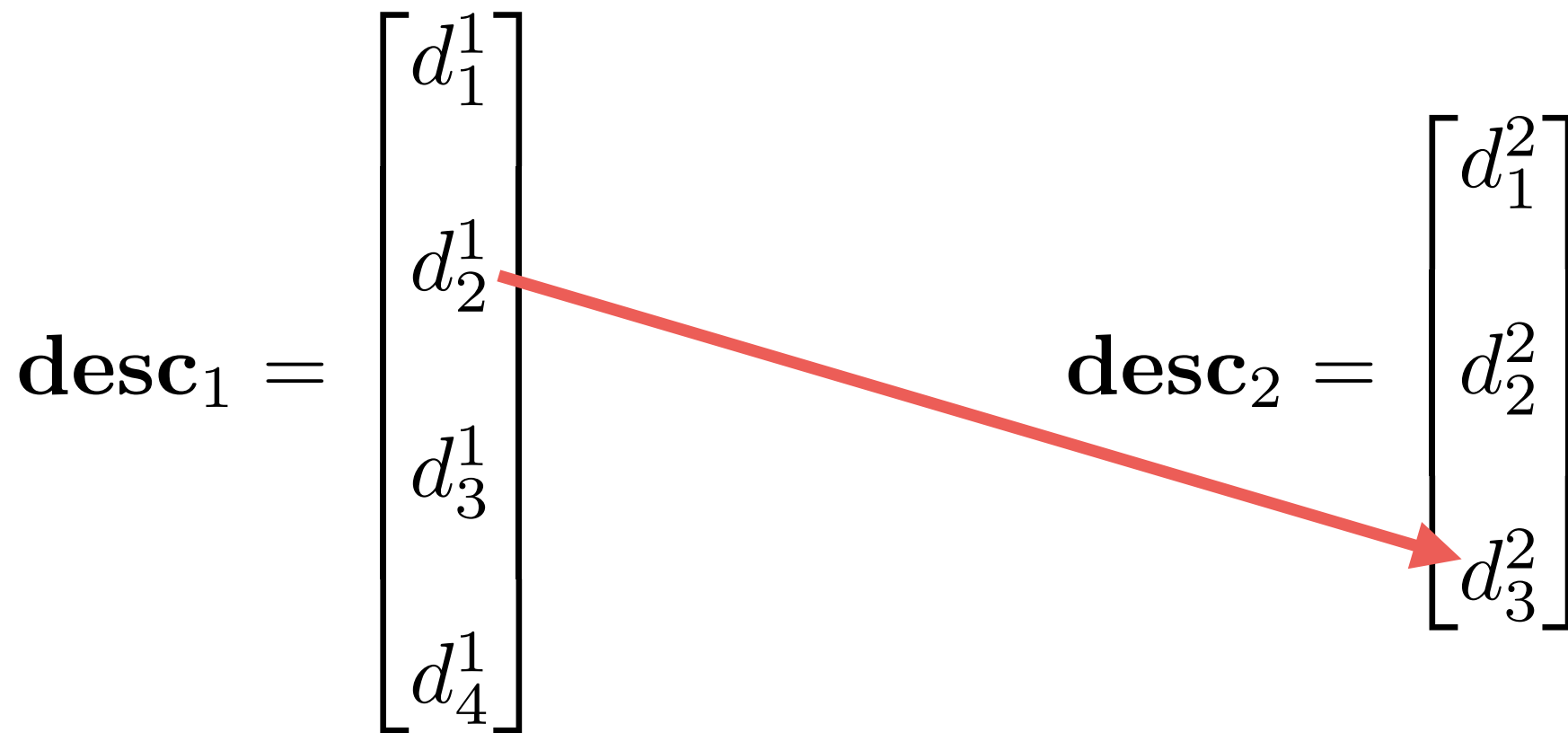
Matching: Example 1

$$\mathbf{desc}_1 = \begin{bmatrix} d_1^1 \\ d_2^1 \\ d_3^1 \\ d_4^1 \end{bmatrix} \qquad \mathbf{desc}_2 = \begin{bmatrix} d_1^2 \\ d_2^2 \\ d_3^2 \end{bmatrix}$$

We find out that the second descriptor of $\mathbf{desc}_1$ matches with the third descriptor of $\mathbf{desc}_2$.

$$\mathbf{M}_{12} = [2,]$$

Matching: Example 1



We find out that the second descriptor of $\mathbf{desc}_1$ matches with the third descriptor of $\mathbf{desc}_2$.

$$\mathbf{M}_{12} = [2, 3,]$$

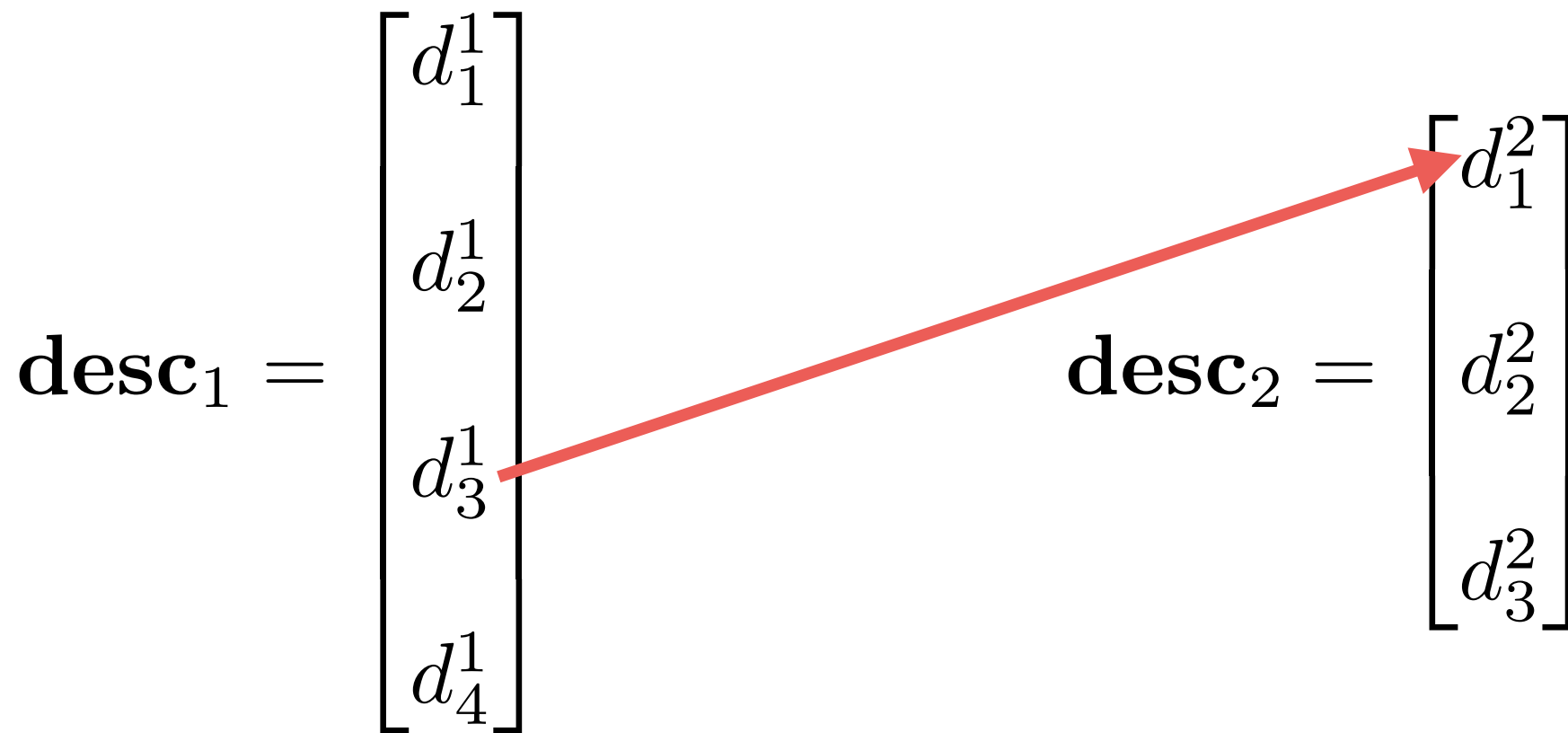
Matching: Example 1

$$\mathbf{desc}_1 = \begin{bmatrix} d_1^1 \\ d_2^1 \\ d_3^1 \\ d_4^1 \end{bmatrix} \qquad \mathbf{desc}_2 = \begin{bmatrix} d_1^2 \\ d_2^2 \\ d_3^2 \end{bmatrix}$$

We find out that the third descriptor of $\mathbf{desc}_1$ matches with the first descriptor of $\mathbf{desc}_2$.

$$\mathbf{M}_{12} = [2, 3,]$$

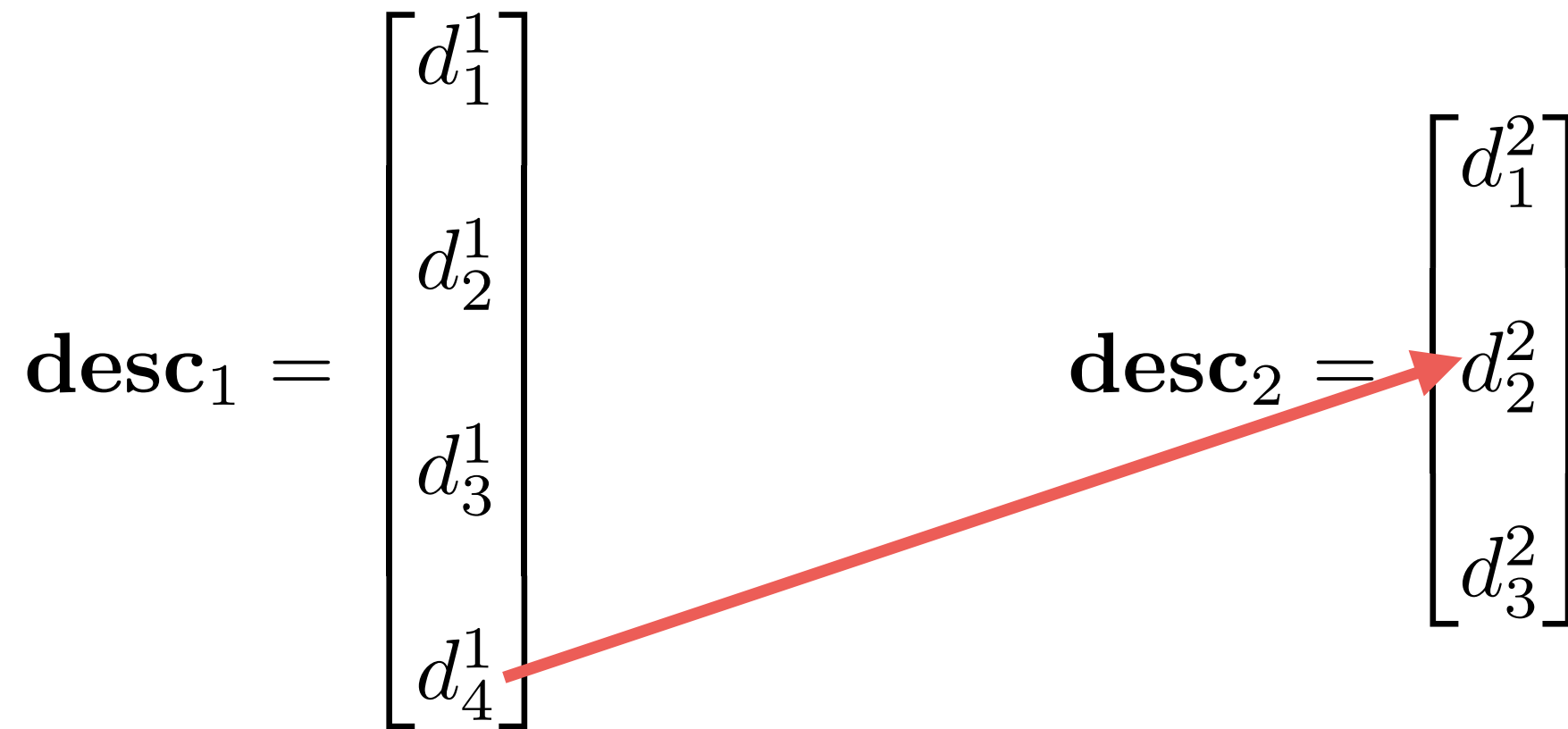
Matching: Example 1



We find out that the third descriptor of $\mathbf{desc}_1$ matches with the first descriptor of $\mathbf{desc}_2$.

$$\mathbf{M}_{12} = [2, 3, 1,]$$

Matching: Example 1



We find out that the fourth descriptor of $\mathbf{desc}_1$ matches with the second descriptor of $\mathbf{desc}_2$.

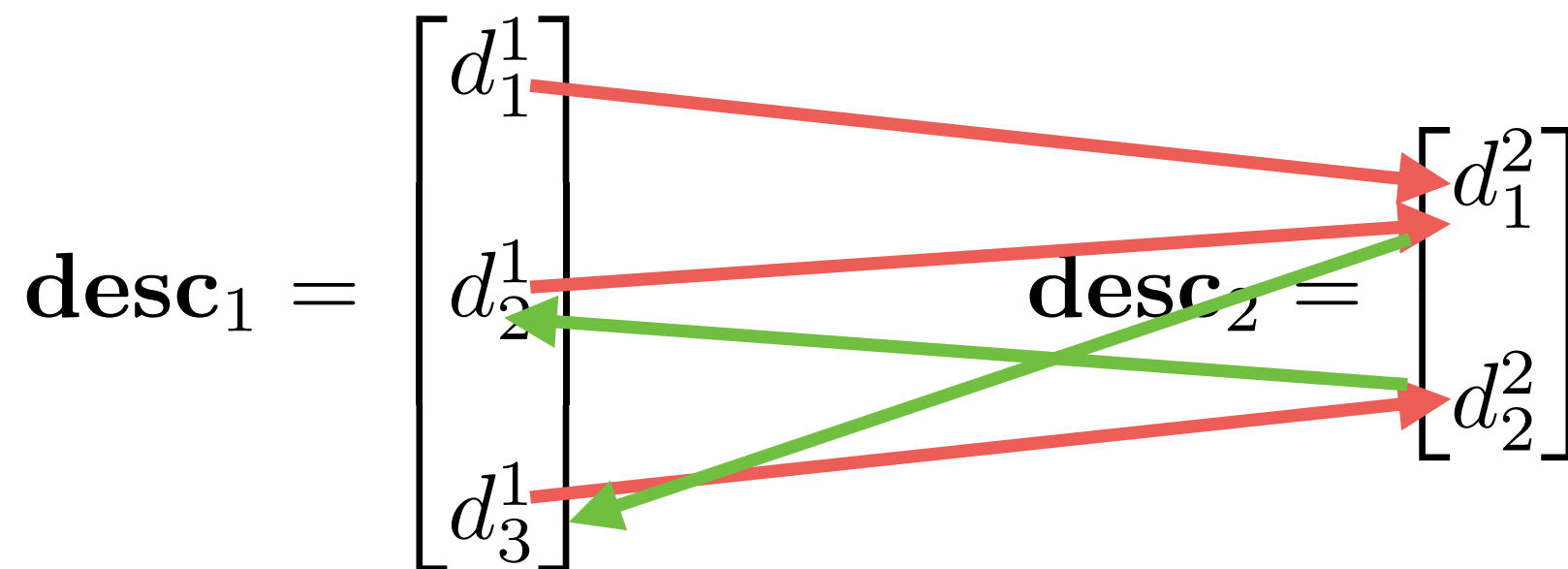
$$\mathbf{M}_{12} = [2, 3, 1, 2]$$

Matching:

How the Output is Encoded Example 2

- Let's say we have 3 descriptors in **desc₁**
- Let's say we have 2 descriptors in **desc₂**
- Let's say that we match I_1 against I_2 , obtaining **M₁₂**.
Then, we match I_2 against I_1 obtaining **M₂₁**.

Matching: Example 2



$$\text{desc}_{12} = [1, 1, 2]$$

$$\text{desc}_{21} = [3, 2]$$

Matching: Example 2

- From this example, we can notice that:
 - The matching operator is ***NOT*** an invertible function:
 - Therefore, $\mathbf{M}_{12}$ and $\mathbf{M}_{21}$ can be very different!
- Why? Let's see it!

Matching: Example 2



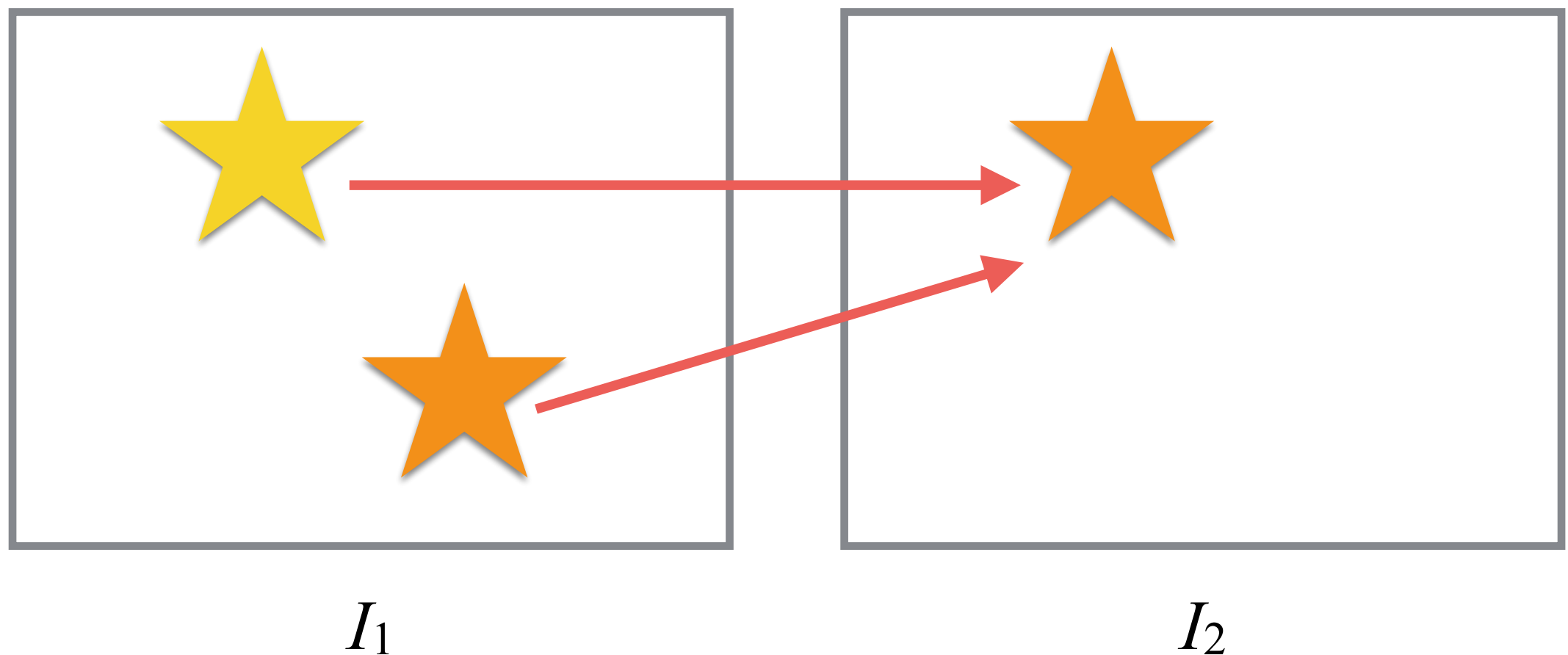
I_1



I_2

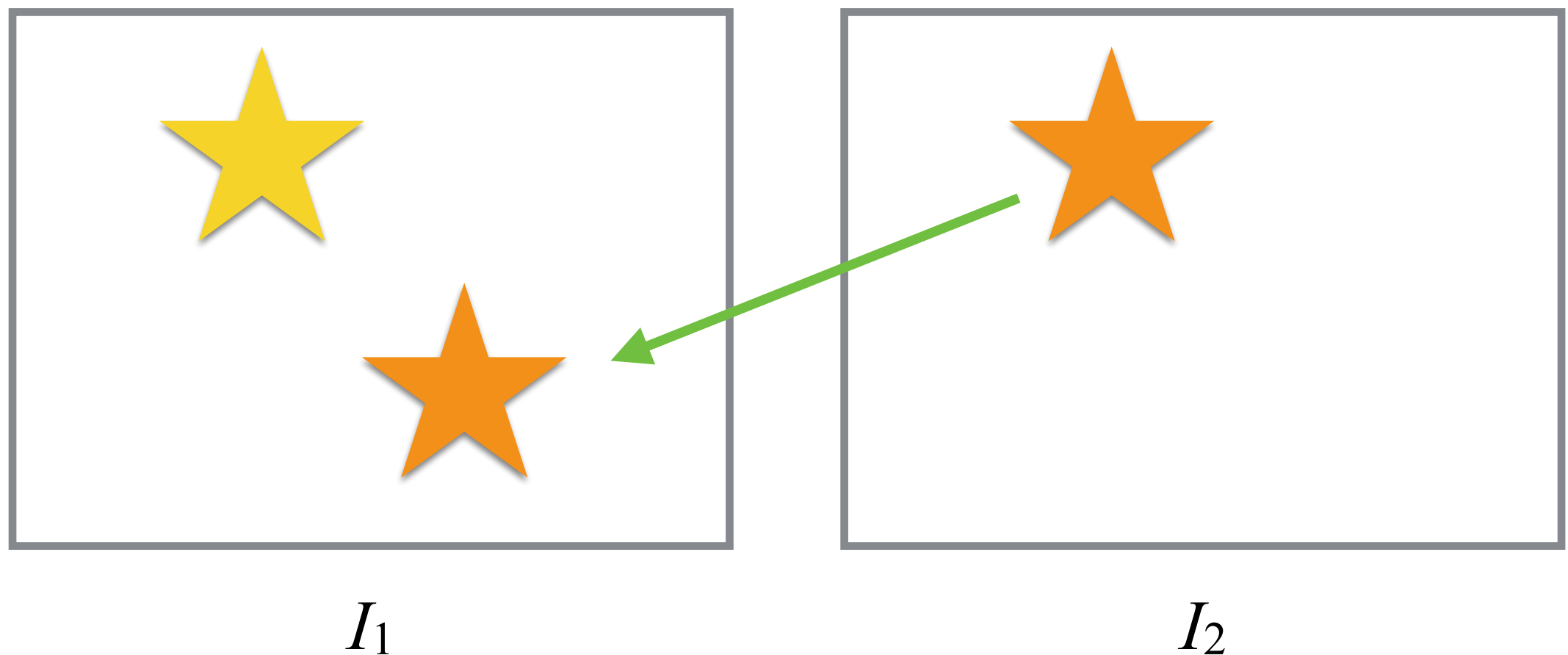
Let's say that stars in this example are feature-point.

Matching: Example 2



When we match I_1 against I_2 , we have two matches; between the two stars in I_1 and the one in I_2 . This is because we need to match the **yellow** star with something in the other image no matter what.

Matching: Example 2



When we match I_2 against I_1 , we do not have a match between the **yellow** star in I_1 and the **orange** one in I_2 because the other **orange** star in I_1 is closer than the **yellow** star!

Matching: Brute Force Algorithm

- A simple method to find a *matched descriptor* in **desc₂** for each descriptor in **desc₁**:
- For each descriptor in **desc₁**, we test it against all descriptors in **desc₂**, and we keep as matched one the *closest* (in terms of distance; either Hamming or Euclidean).

Matching: Brute Force Algorithm

For each descriptor $\mathbf{d}^1_i$ in $\mathbf{desc}_1$:

matched(i) = -1;

matched_dist = **BOTTOM**;

For each descriptor $\mathbf{d}^2_j$ in $\mathbf{desc}_2$:

if Closer($D(\mathbf{d}^1_i, \mathbf{d}^2_j)$, matched_dist)

matched(i) = j ;

matched_dist = $D(\mathbf{d}^1_i, \mathbf{d}^2_j)$;

endif

$D(,)$ is a distance function; it can be Hamming, Euclidean, etc.

Matching: Brute Force Algorithm

For each descriptor $\mathbf{d}^1_i$ in $\mathbf{desc}_1$:

matched(i) = -1;

matched_dist = **BOTTOM**;

For each descriptor $\mathbf{d}^2_j$ in $\mathbf{desc}_2$:

if Closer($D(\mathbf{d}^1_i, \mathbf{d}^2_j)$, matched_dist)

matched(i) = j ;

matched_dist = $D(\mathbf{d}^1_i, \mathbf{d}^2_j)$;

endif

$D(,)$ is a distance function; it can be Hamming, Euclidean, etc.

Matching: Brute Force Algorithm

For each descriptor $\mathbf{d}^1_i$ in $\mathbf{desc}_1$:

matched(i) = -1;

matched_dist = **BOTTOM**;

For each descriptor $\mathbf{d}^2_j$ in $\mathbf{desc}_2$:

if Closer($D(\mathbf{d}^1_i, \mathbf{d}^2_j)$, matched_dist)

matched(i) = j ;

matched_dist = $D(\mathbf{d}^1_i, \mathbf{d}^2_j)$;

endif

BOTTOM = +Inf for SIFT
BOTTOM = 0 for BRIEF/ORB

$D(.)$ is a distance function; it can be Hamming, Euclidean, etc.

Matching: Brute Force Algorithm

- Advantage:
 - It is exhaustive and finds the ***best solution!***
- Disadvantage:
 - This method is very slow:
 - Let's say we have n descriptors in **desc**₁ and n in **desc**₂. In the worst case, we need to compare roughly n^2 descriptors. This becomes an issue when we have more than 100 descriptors per image!

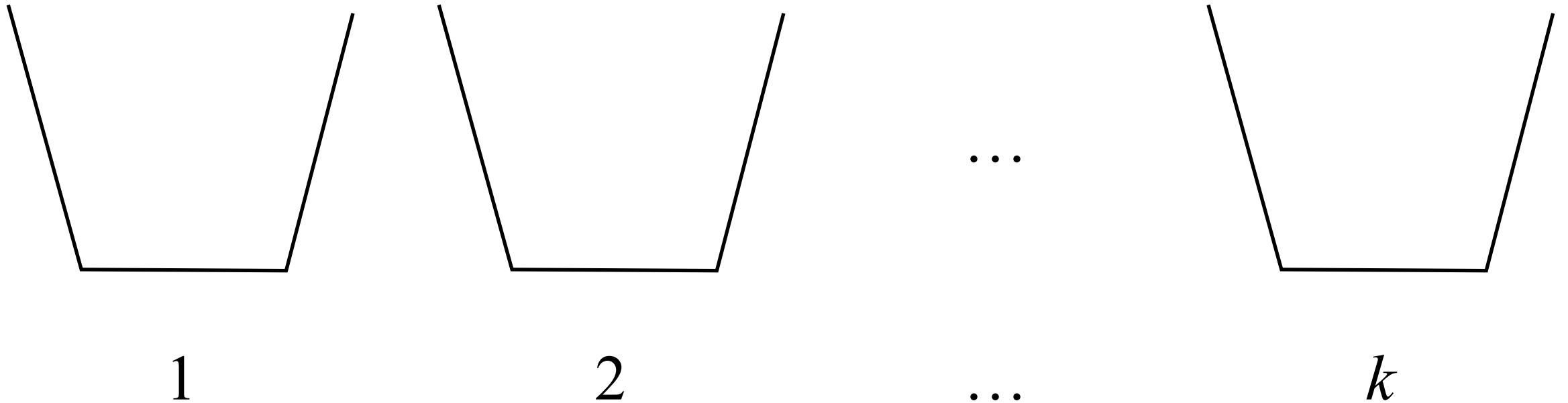
Matching: Improving Efficiency

- How can we improve (approximating results)?
- **Hashing**: the idea is to group similar descriptors in k groups or buckets that have a constant size.
- We create k bucket.
- Each **descriptor** in $\mathbf{desc}_2$ of I_2 is assigned to a bucket using a function f , called hash function. This is defined as:

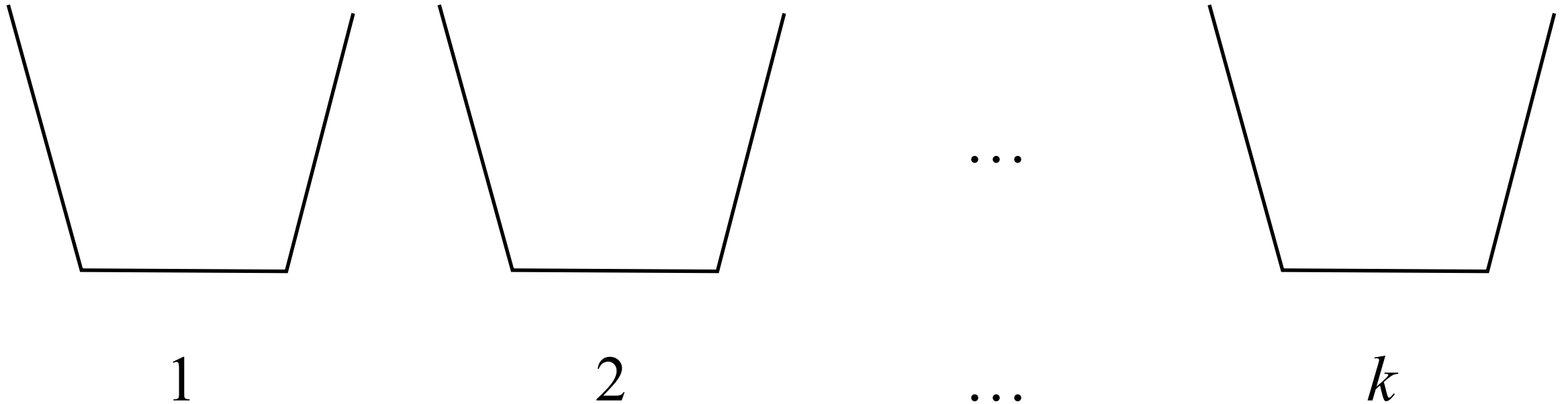
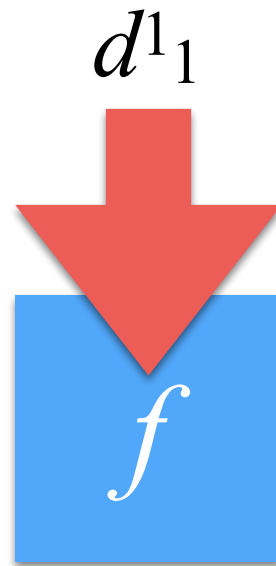
$$f: \mathbf{descriptor} \longrightarrow [1, k] \text{ (positive integer numbers!)}$$

- This means that f cover generates a number in $[1, k]$ given a descriptor.
- For example, an f for BRIEF/ORB, where the descriptor is a 256-bit number, is the modulo operation.

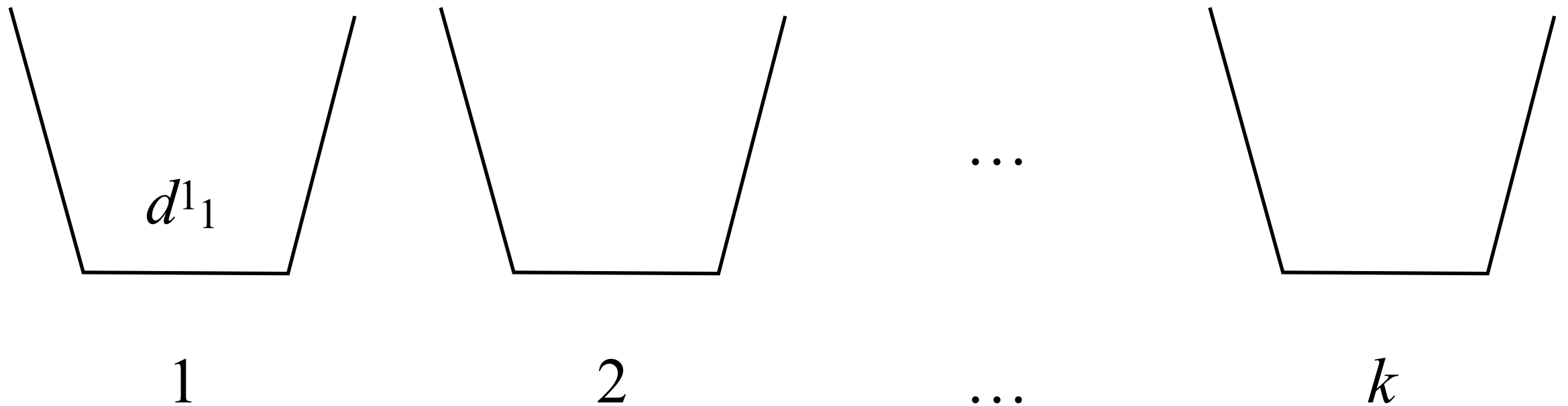
Matching: Improving Efficiency



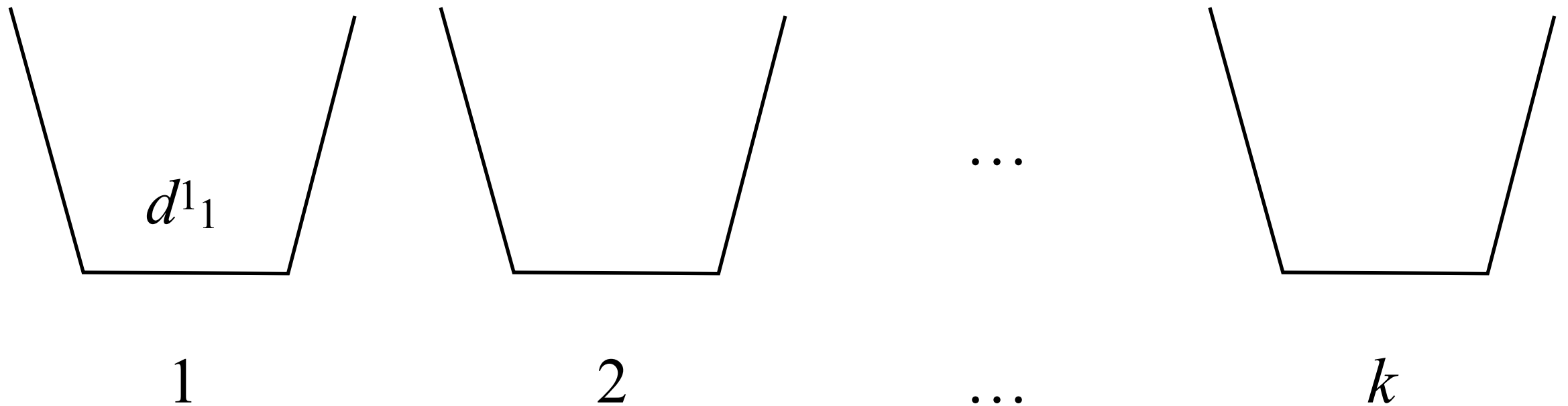
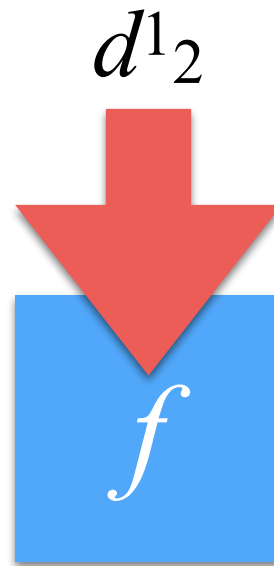
Matching: Improving Efficiency



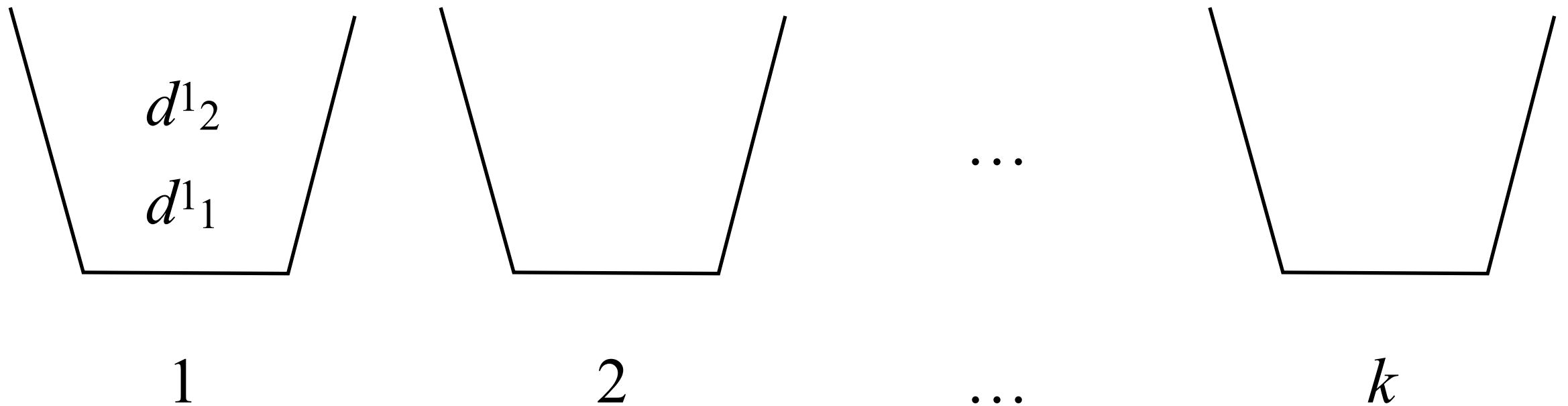
Matching: Improving Efficiency



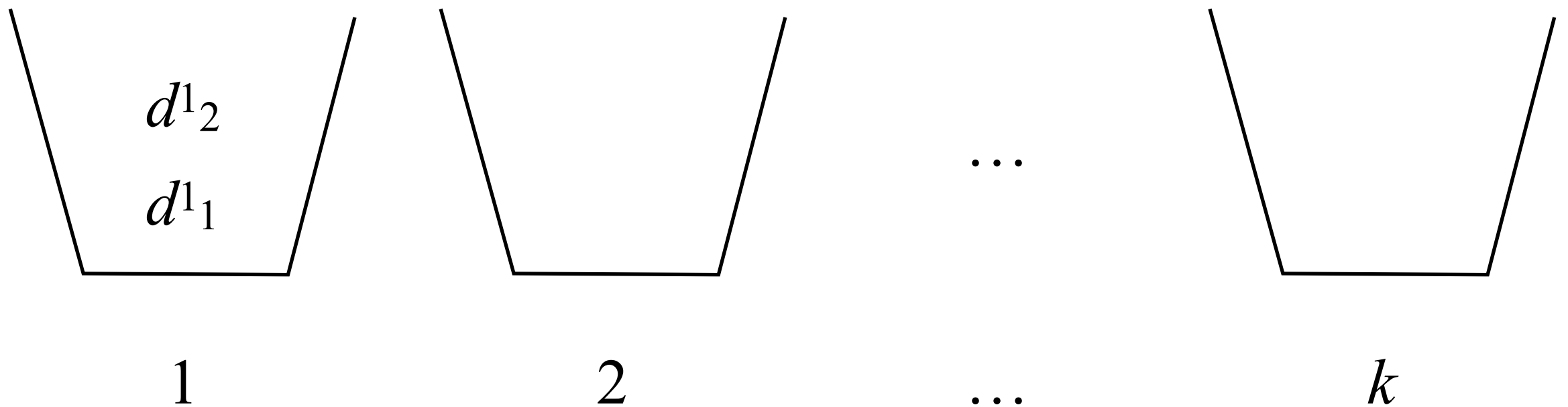
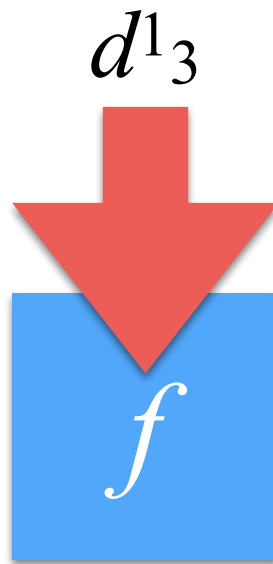
Matching: Improving Efficiency



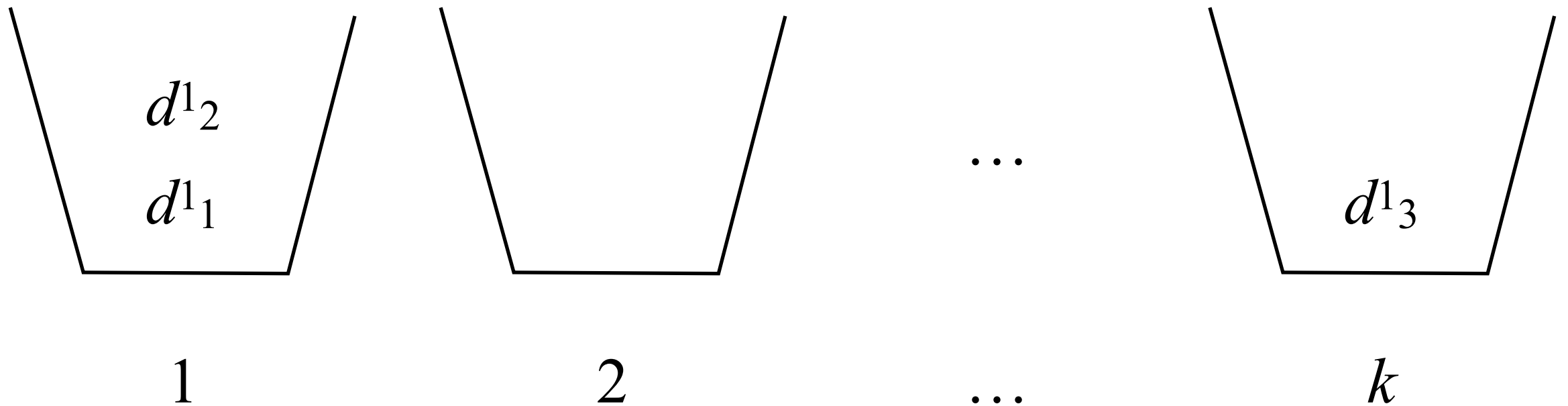
Matching: Improving Efficiency



Matching: Improving Efficiency



Matching: Improving Efficiency



etc.

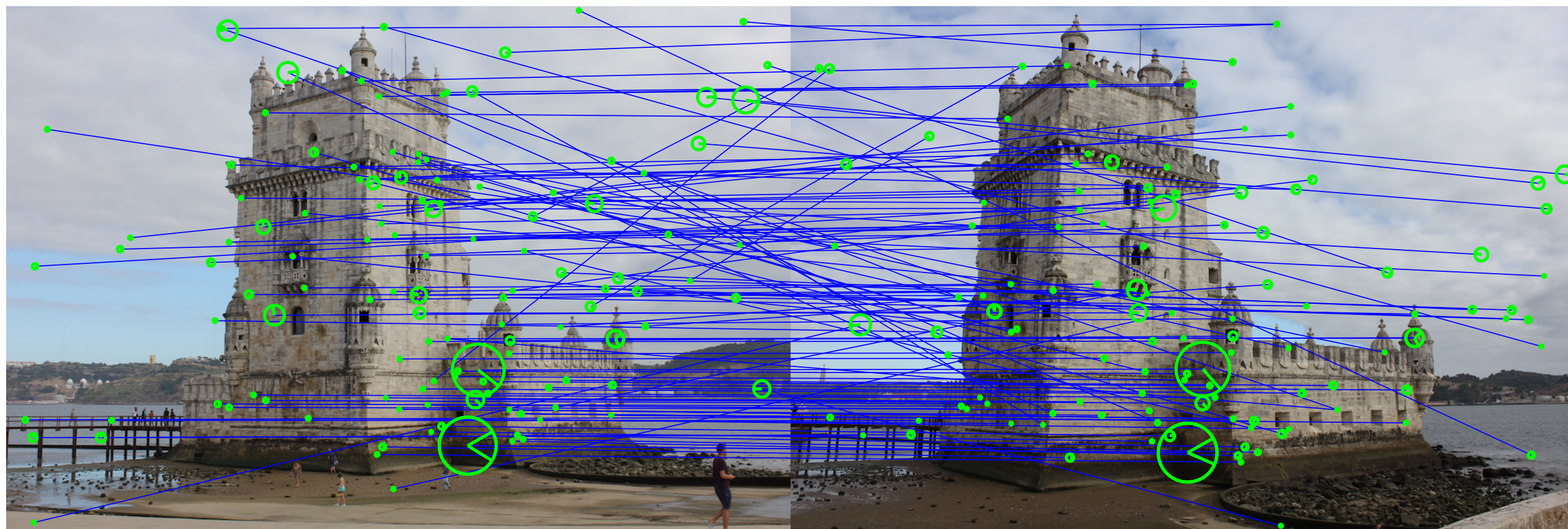
Matching: Improving Efficiency

- Now, we have all descriptors of I_2 into buckets.
- To find a match for a descriptor d^1_i of I_1 , we apply f to d^1_i . In this way, we obtain a bucket number, let's call it r .
- We run the brute force method between d^1_i and all the descriptors that are in r .

Matching: Improving Efficiency

- Advantages:
 - It is faster, we run the brute force method for a subset of descriptors.
- Disadvantages:
 - It is not exact, it is *approximate*; i.e., we test only a sub-set of descriptors.
 - If f is not well crafted, we may have distant descriptors in the same bucket.

Matching: Example



Matching

- Once we have know matches between images, we can understand which images are near each others!
- This is important for stable algorithms for triangulation of points and determining cameras' poses!

that's all folks!