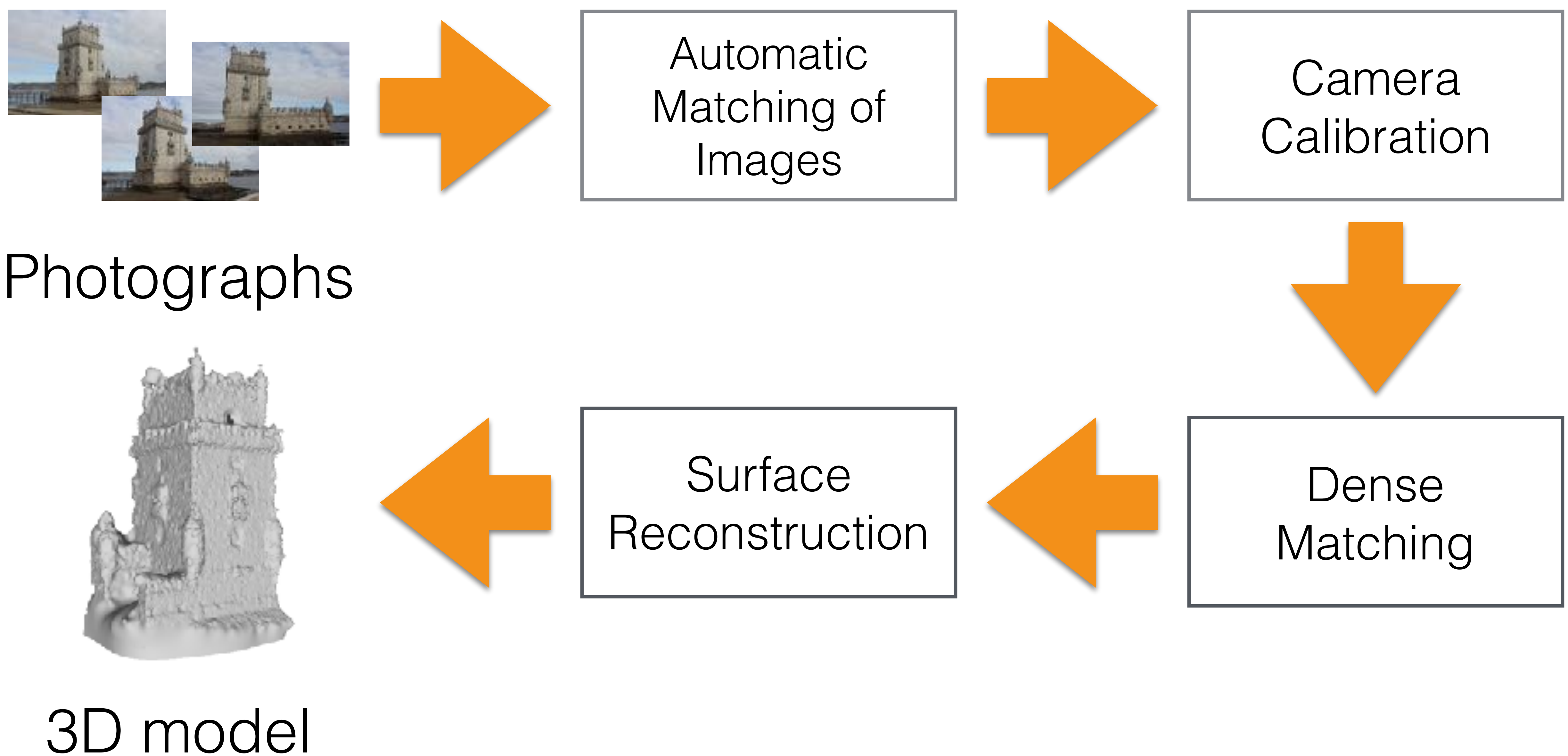
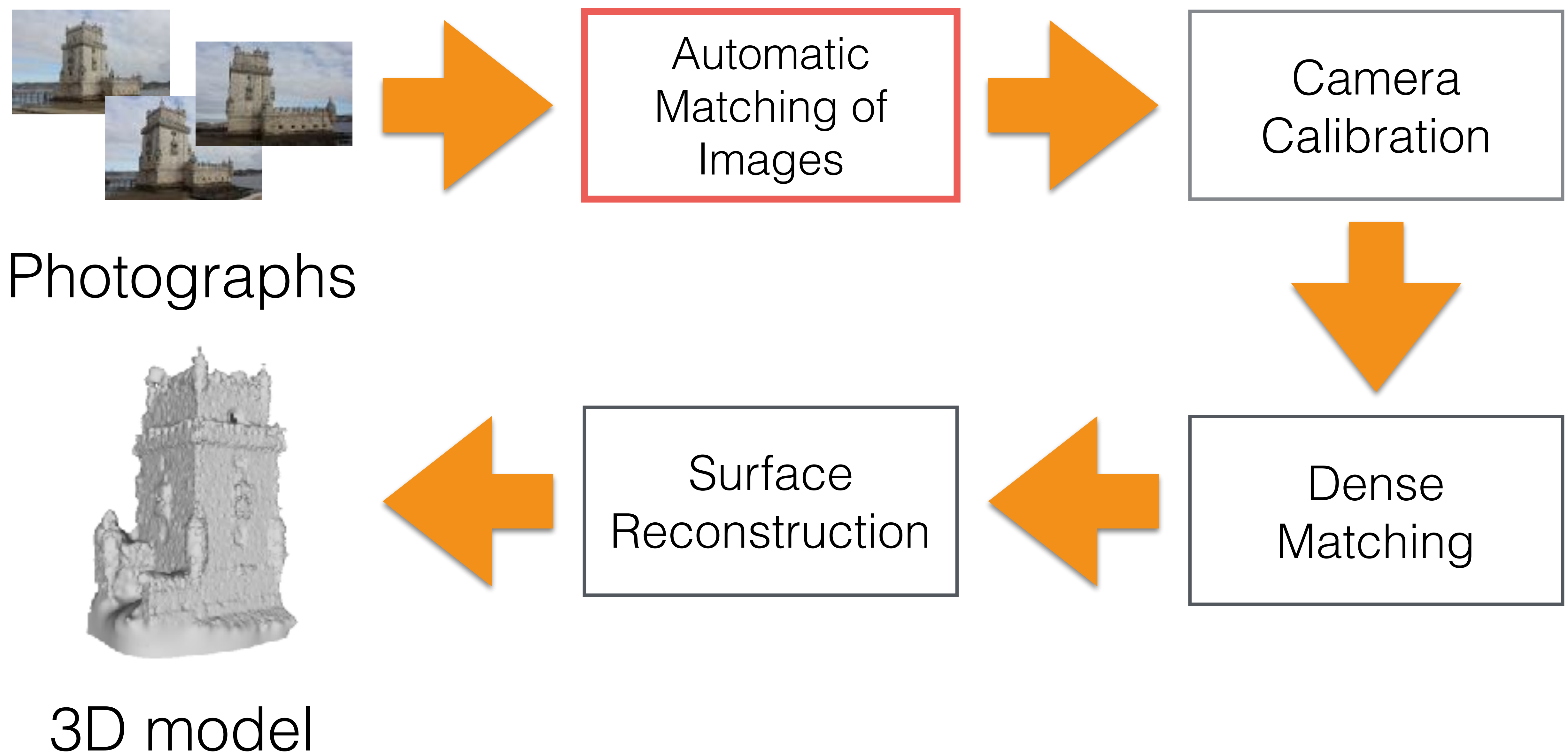
3D from Photographs: Automatic Matching of Images

Dr Francesco Banterle
francesco.banterle@isti.cnr.it

3D from Photographs

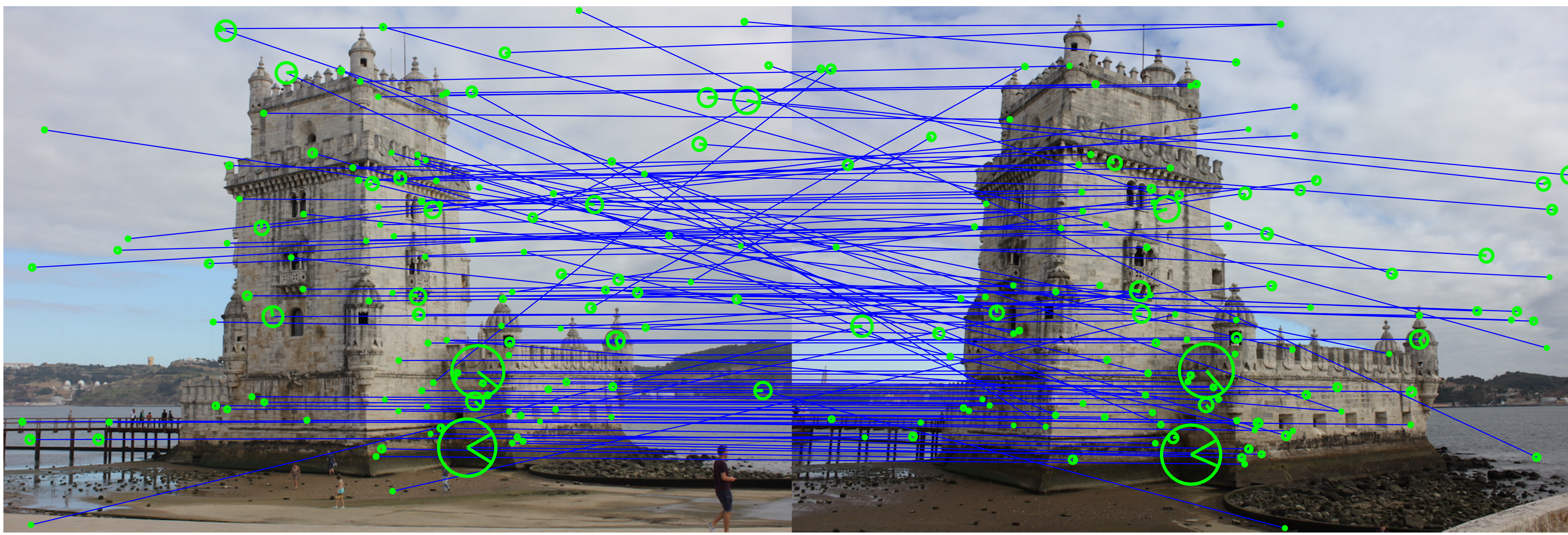


3D from Photographs



The Matching Problem

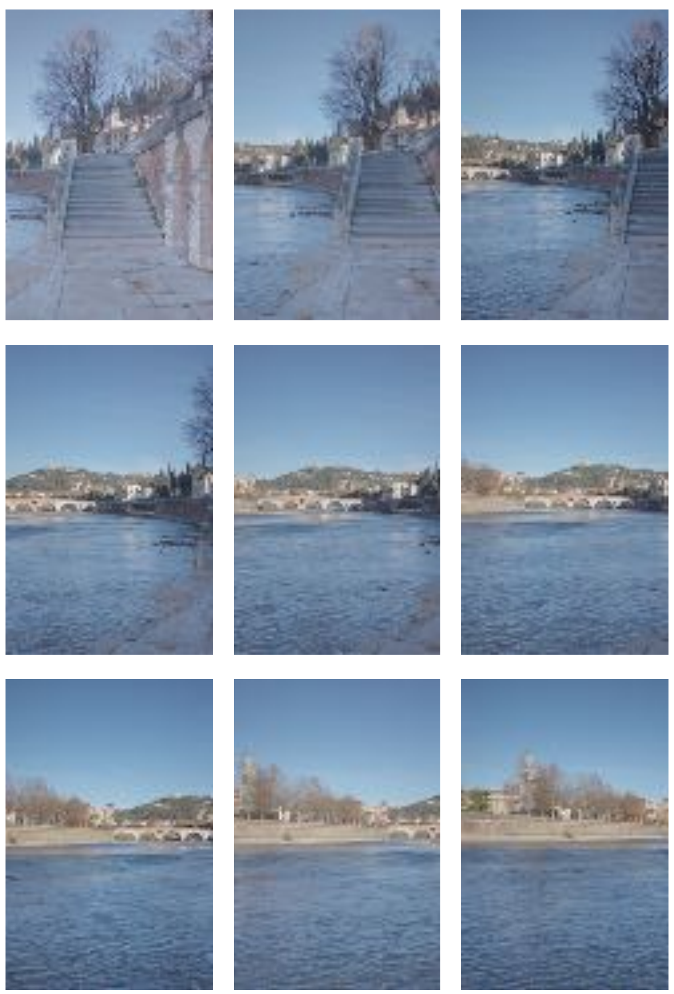
- We need to find corresponding feature across two or more views:



The Matching Problem

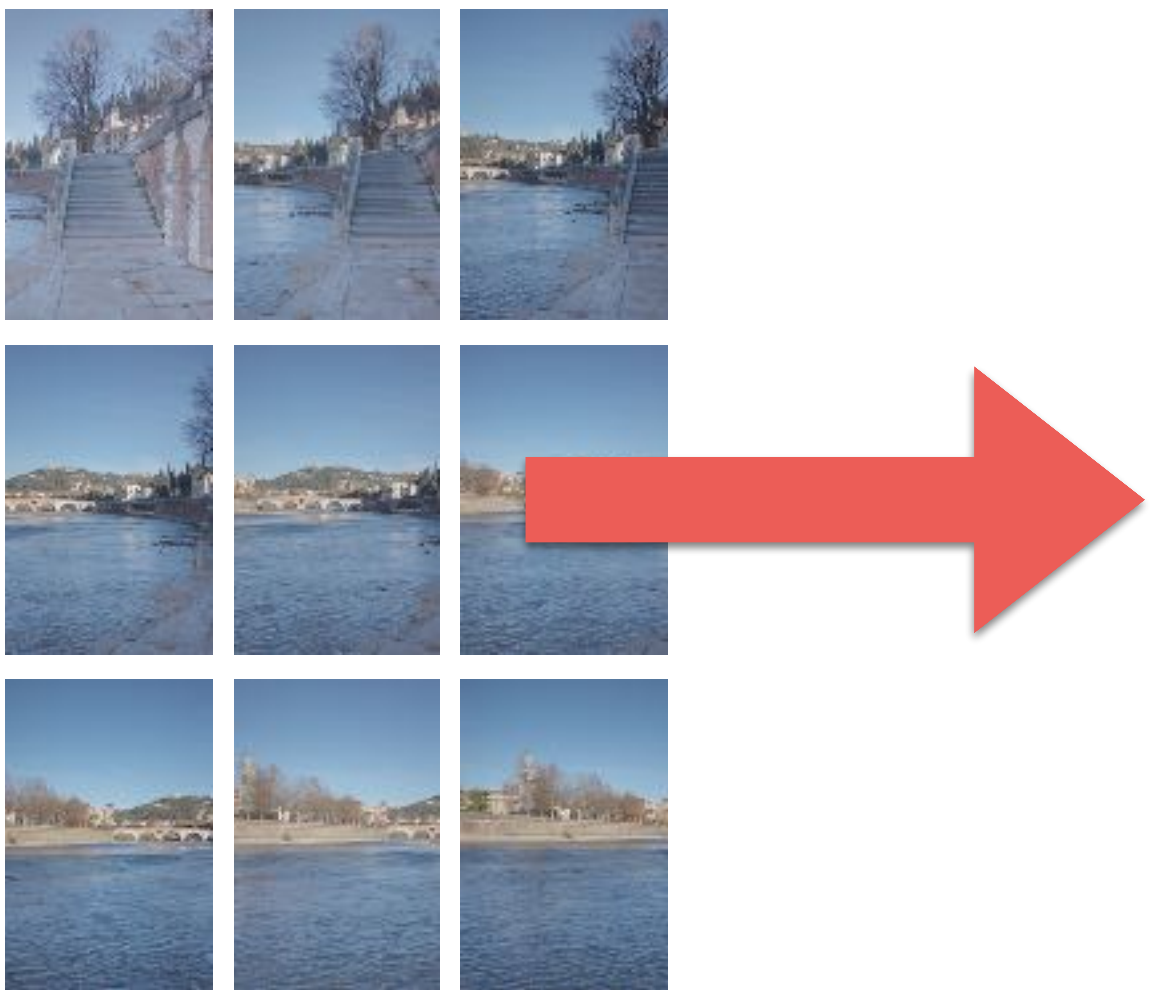
- Why?
  - 3D Reconstruction.
  - Image Registration.
  - Visual Tracking.
  - Object Recognition.
  - etc.

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

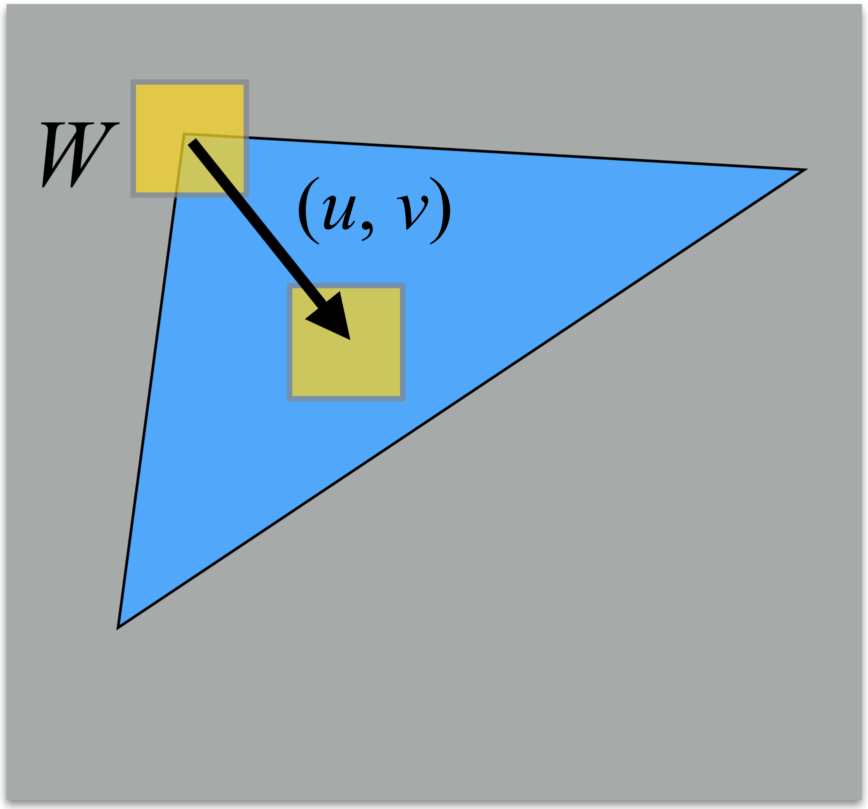
- A feature is a piece of the input image that is relevant for solving a given task.
- Features can be global or local.
- We will focus on local features that are more robust to occlusions and variations.

Extraction of Local Features

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

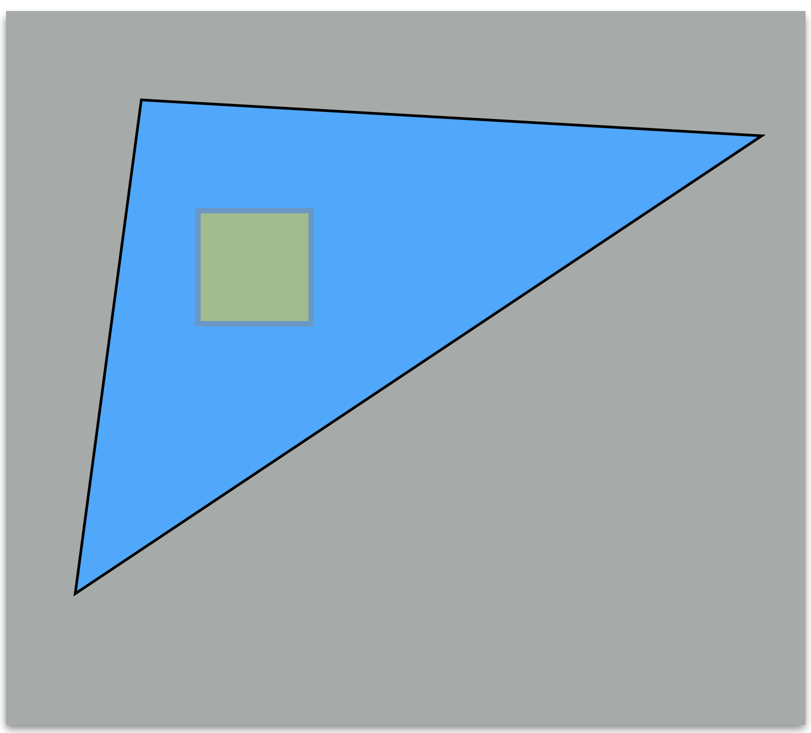
Harris Corner Detector

- Let's consider a window, W:
  - how do pixels change in W?
  - Let's compare each pixel before and after moving W by (u, v) using the sum of squared differenced (SSD).

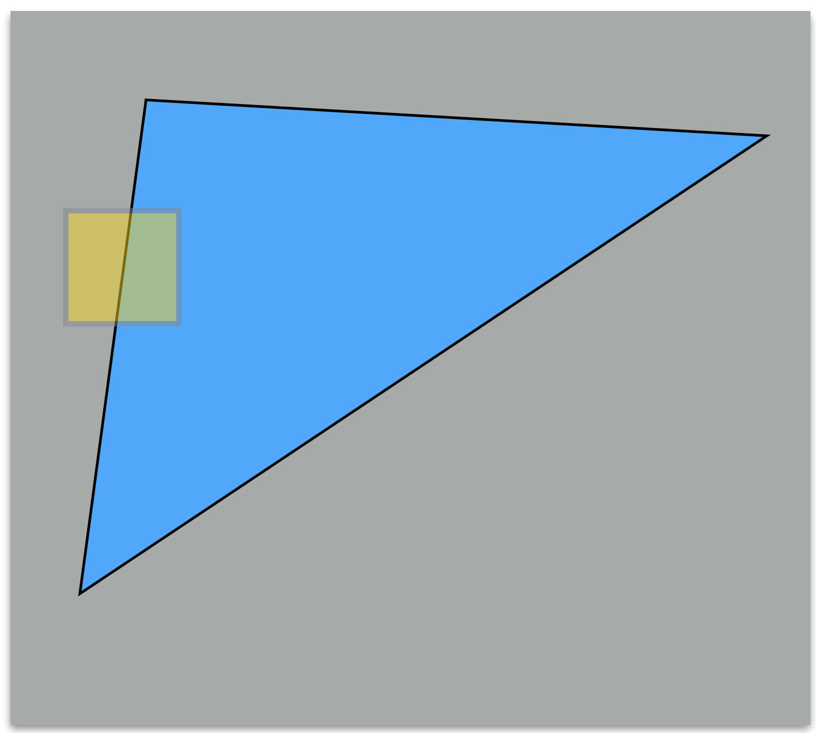


$$E(u,v) = \sum_{x,y \in W} \Big(I(x+u, y+v) - I(x,y) \Big)^2$$

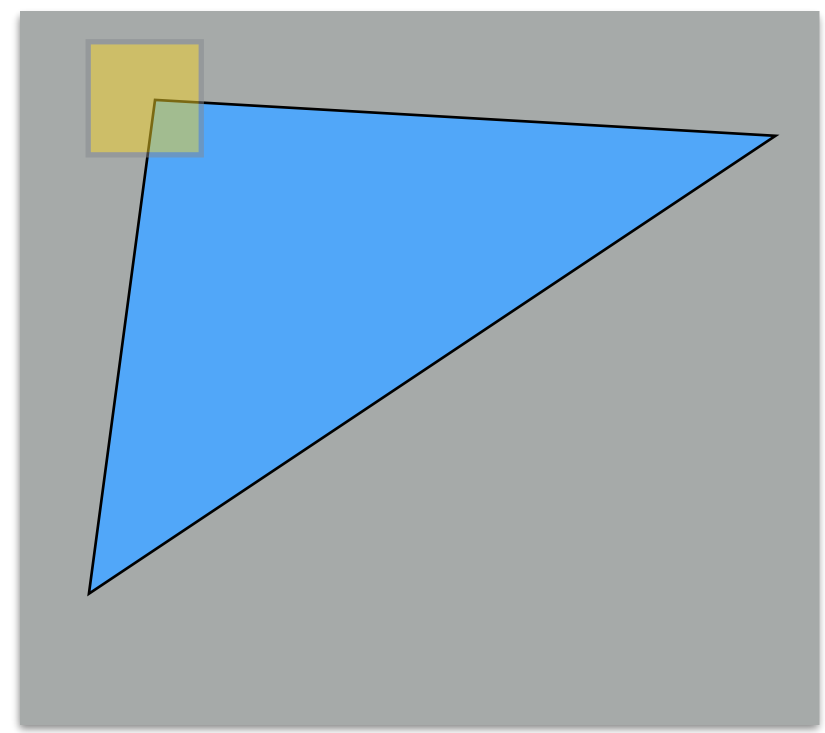
What a Corners is



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:

$$I(x+u, y+v) \approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

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

Harris Corner Detector: Small Motion Assumption

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

$$\approx \sum_{x,y \in W} \Big(I(x,y) + I_x(x,y)u + I_y(x,y)v - I(x,y) \Big)^2$$

$$\approx \sum_{x,y \in W} \Big(I_x(x,y)u + I_y(x,y)v \Big)^2$$

$$\approx \sum_{x,y \in W} \Big(I_x(x,y)^2 u^2 + 2 I_x(x,y) I_y(x,y) uv + I_y(x,y)^2 v^2 \Big)$$

Harris Corner Detector: Small Motion Assumption

$$E(u,v) \approx \sum_{x,y \in W} \left(I_x(x,y)^2 u^2 + 2 I_x(x,y) I_y(x,y) uv + I_y(x,y)^2 v^2 \right)$$

$$\approx Au^2 + 2Buv + Cv^2$$

$$A = \sum_{x,y \in W} I_x(x,y)^2 \quad B = \sum_{x,y \in W} I_x(x,y)^2 I_y(x,y)^2 \quad C = \sum_{x,y \in W} I_y(x,y)^2$$

Harris Corner Detector: Small Motion Assumption

- The surface (u, v) can be locally approximate by a quadratic form:

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

$$A = \sum_{x,y \in W} I_x(x,y)^2 \quad B = \sum_{x,y \in W} I_x(x,y)^2 I_y(x,y)^2 \quad C = \sum_{x,y \in W} I_y(x,y)^2$$

Harris Corner Detector: Small Motion Assumption

- E(u,v) can be rewritten as

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

$$\approx \begin{bmatrix} u & v \end{bmatrix} \cdot M \cdot \begin{bmatrix} u \\ v \end{bmatrix}$$

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

Harris Corner Detector: Small Motion Assumption

- E(u,v) can be rewritten as

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

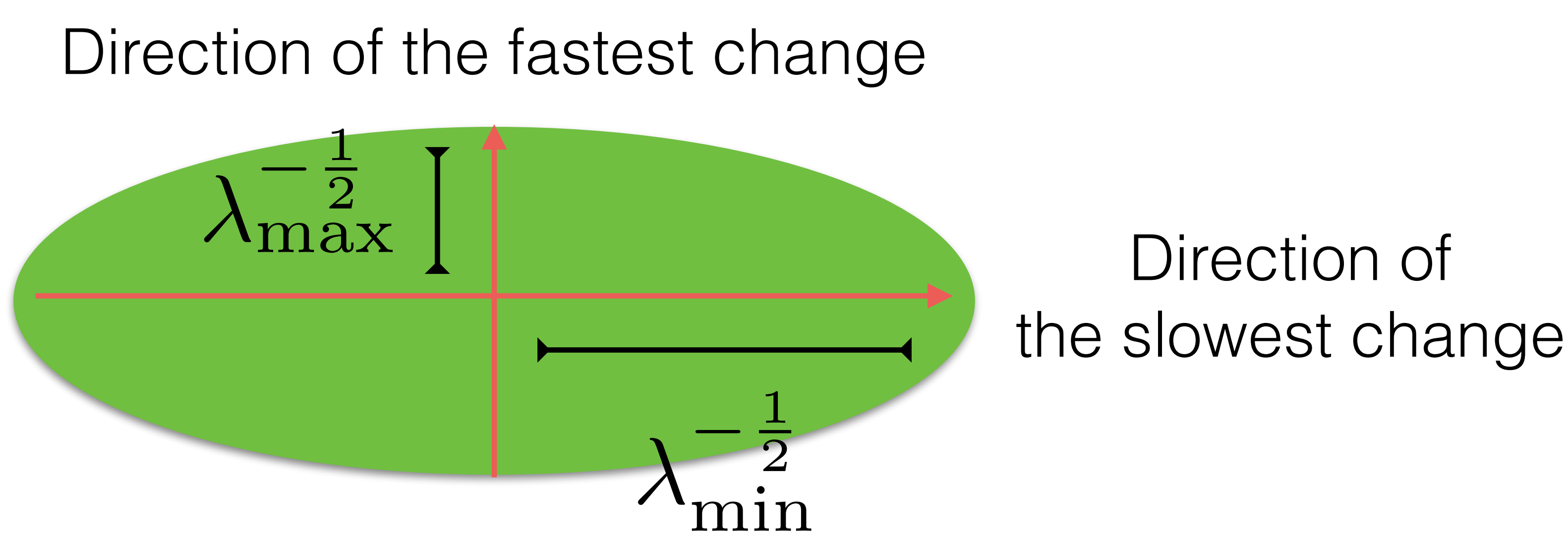
$$\approx \boxed{\begin{bmatrix} u & v \end{bmatrix} \cdot M \cdot \begin{bmatrix} u \\ v \end{bmatrix}}$$

Ellipse Equation:
$E(u, v) = k$

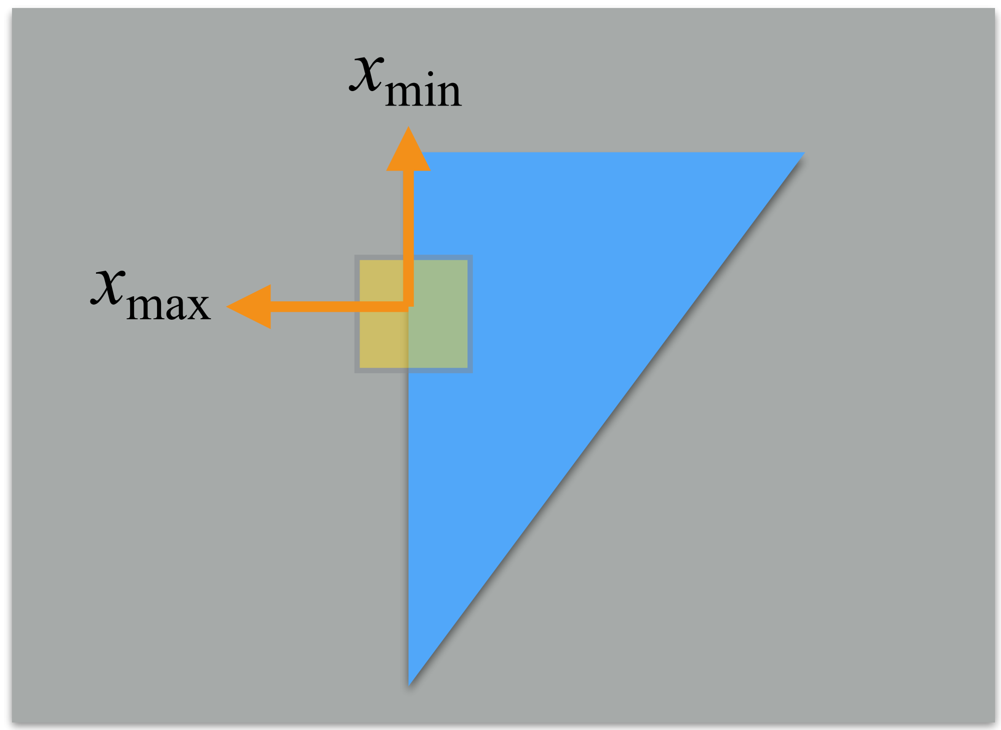
$$M = \sum_{x,y \in W} \begin{bmatrix} I_x^2(x,y) & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y^2(x,y) \end{bmatrix}$$

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.



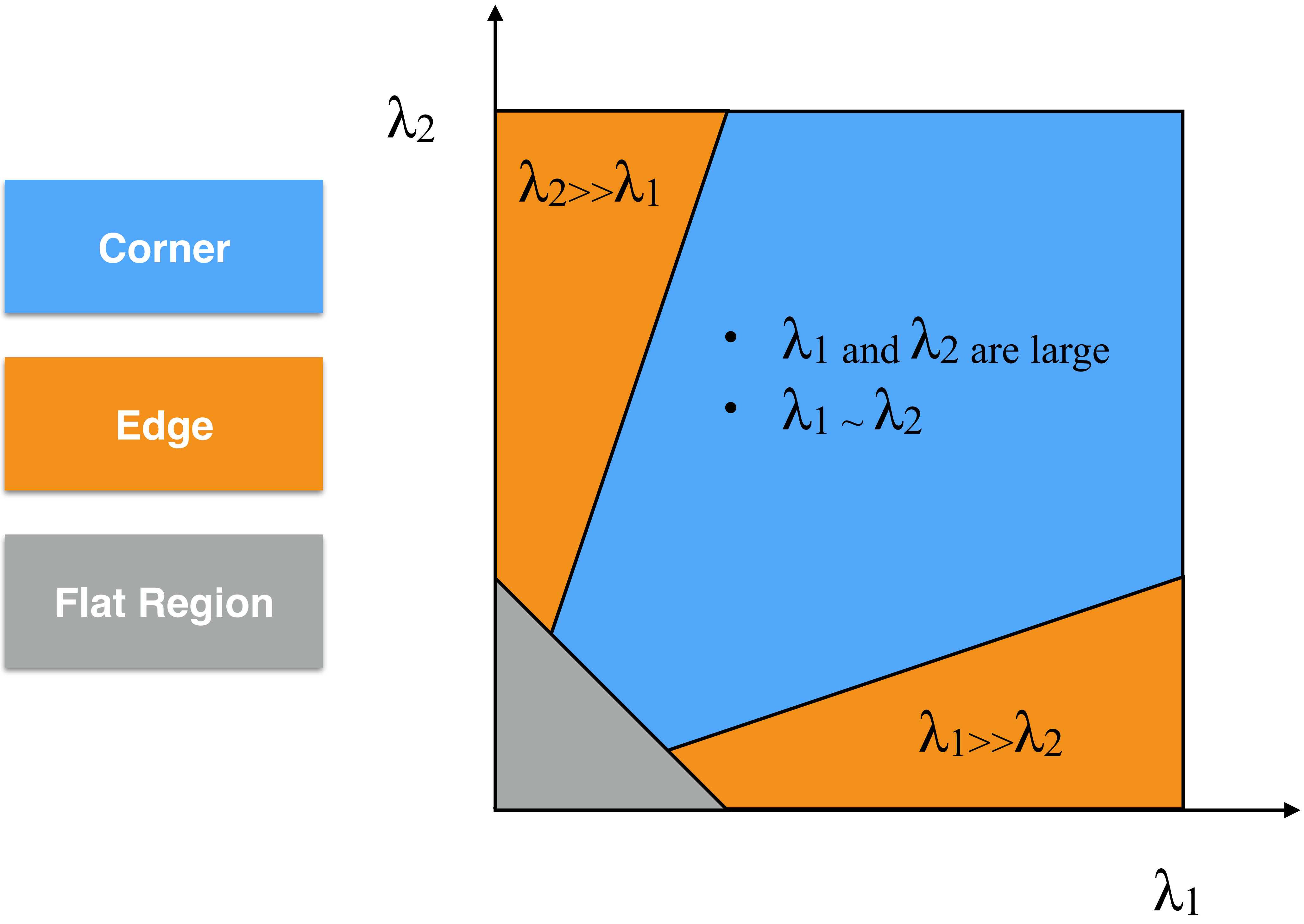
Harris Corner Detector: Second Moment Matrix



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

- x_{max} = direction of largest increase in E
- λ_{max} = amount of increase in direction x_{max}
- x_{min} = direction of smallest increase in E
- λ_{min} = amount of increase in direction x_{min}

Classification



Harris Corner Detector: Cornerness 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):

$$R = \mathrm{Det}(M) - k\mathrm{Tr}(M)^2$$
$$\mathrm{Det}(M) = \lambda_1 \lambda_2$$
$$\mathrm{Tr}(M) = \lambda_1 + \lambda_2$$

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

Harris Corner Detector: Cornerness Measure

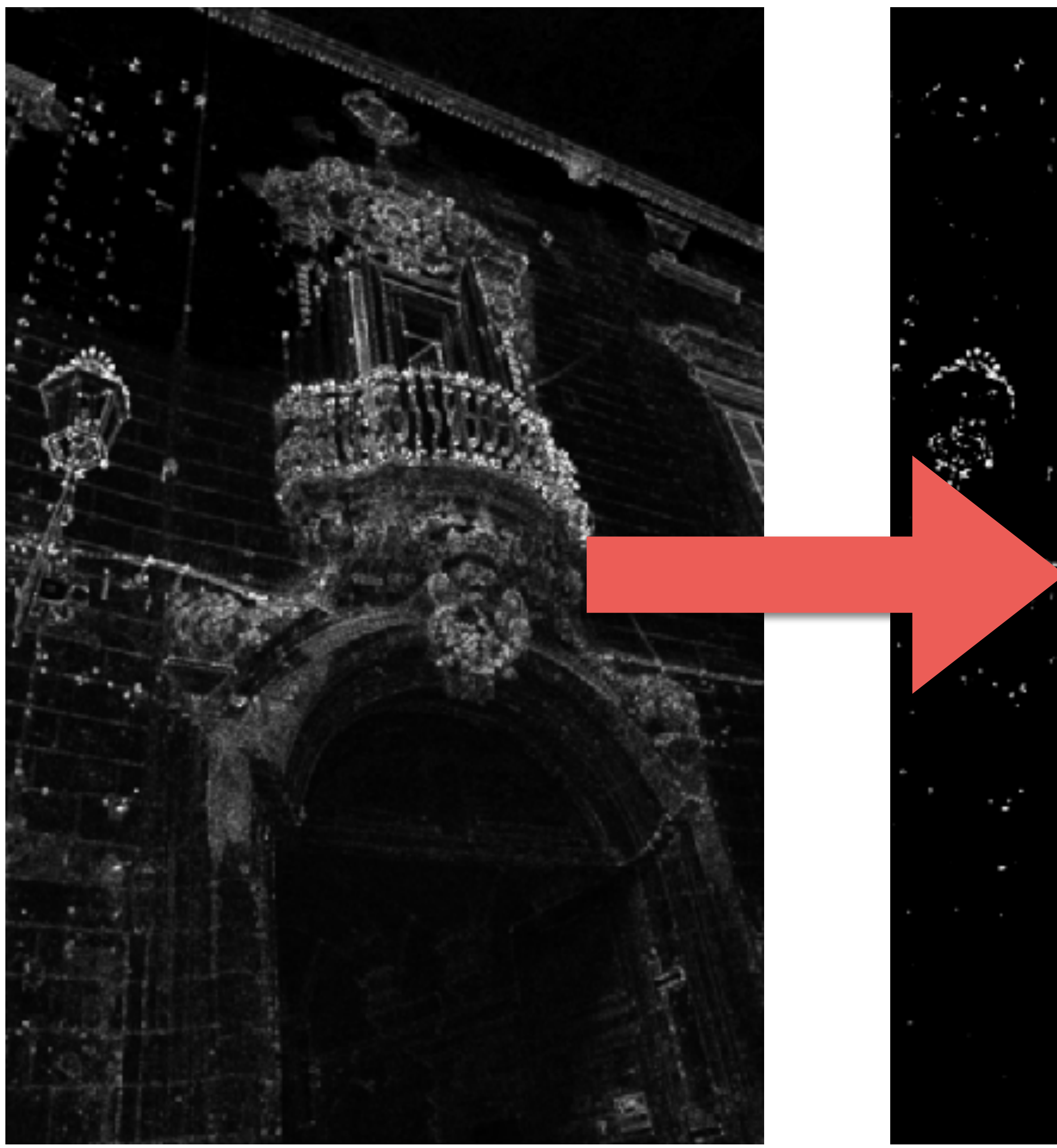
Input Image

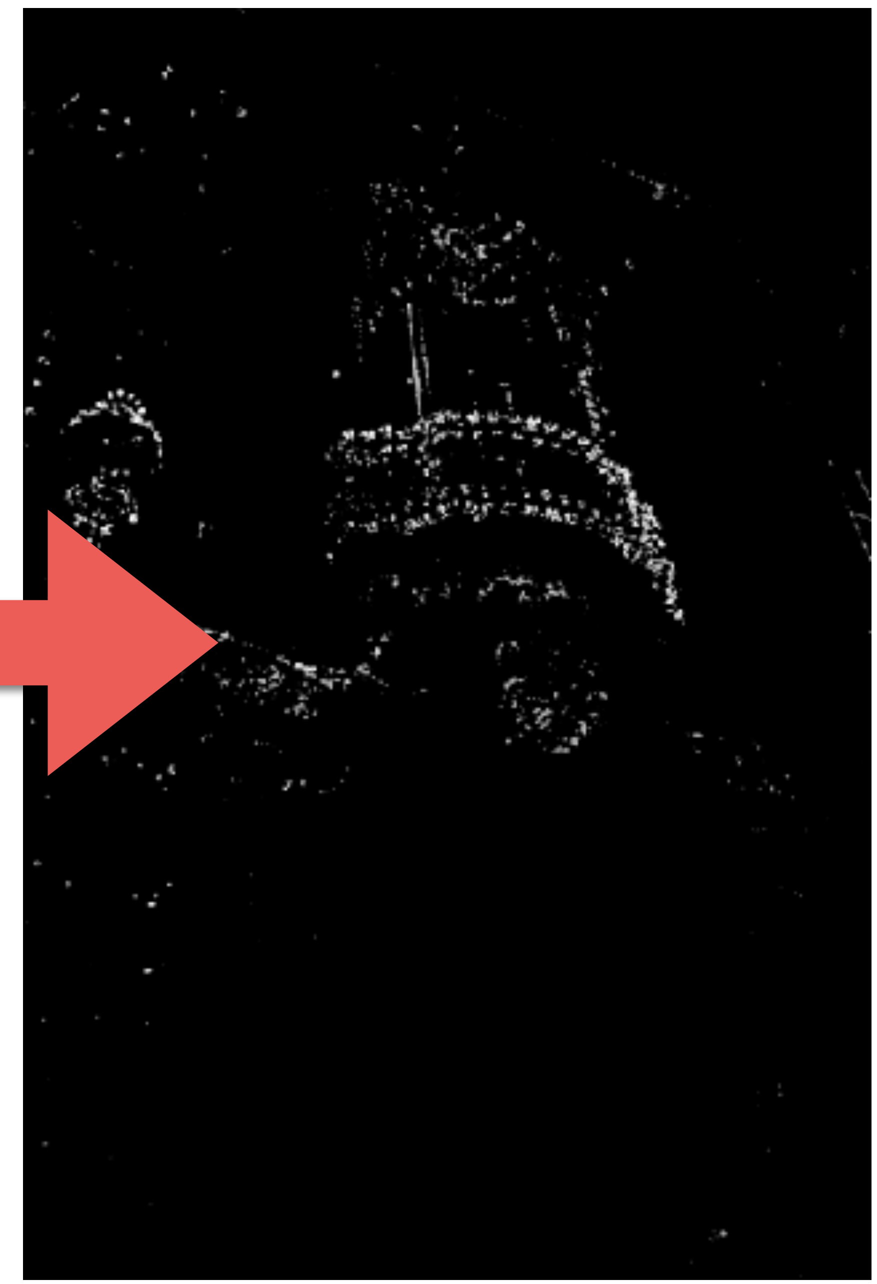
R

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



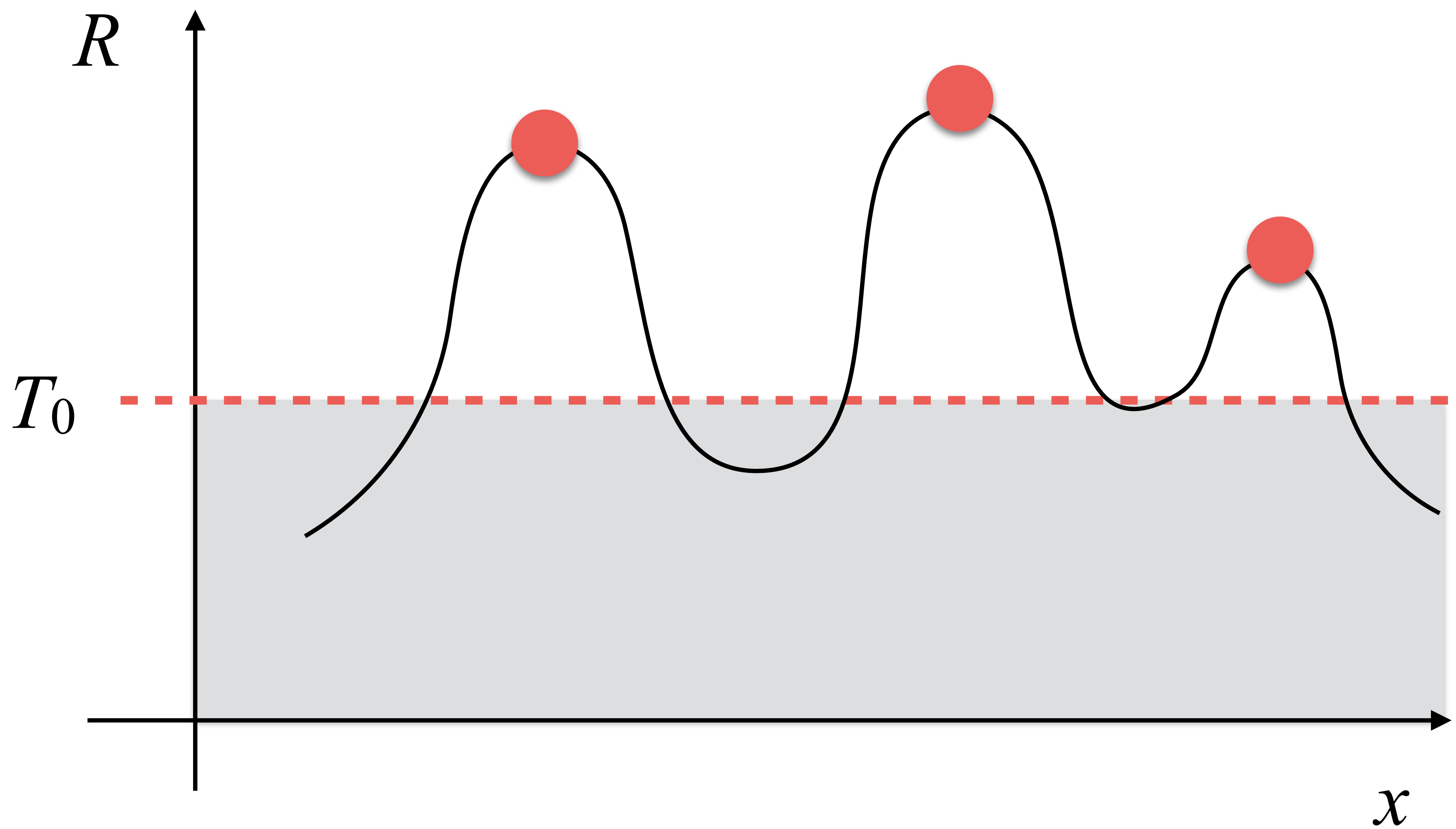


R

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

- We set a radius (in pixel) for suppressing non-maxima; e.g., 5-9.

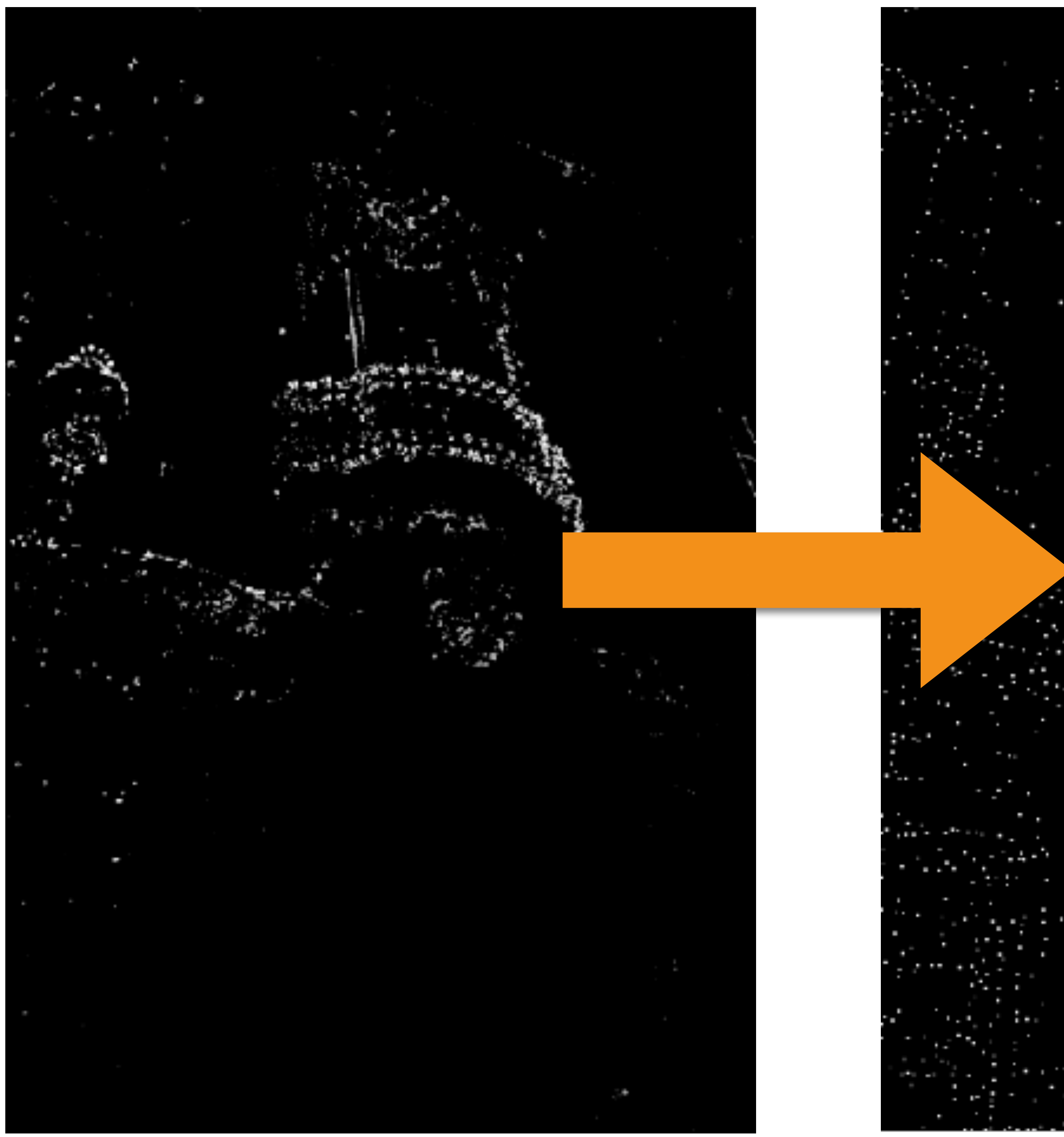
- We apply to R a maximum filter; it is similar to the median filter, but it sets the maximum to pixels:

  - We obtain $R_{\max}$.

- A local pixel is a local maximum if and if:

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

Harris Corner Detector: Non-Maximal Suppression



R after thresholding

Non-Maximal Suppression

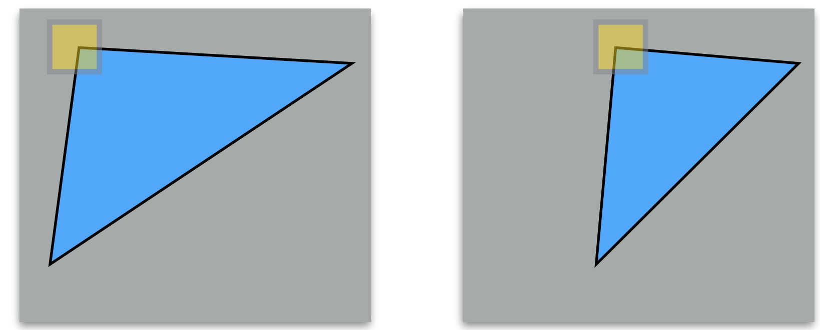
Harris Corner Detector: Non-Maximal Suppression

Harris Corner Detector: Non-Maximal Suppression

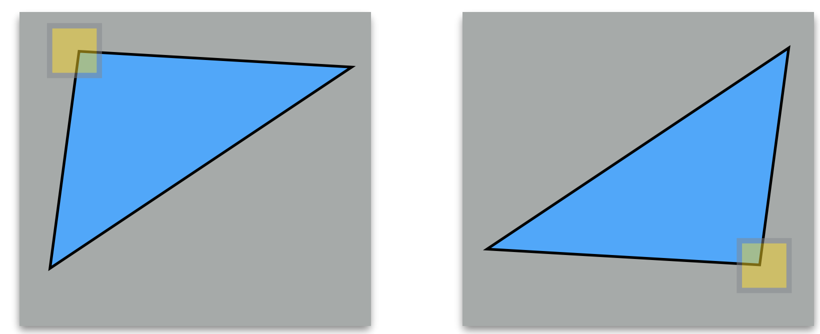
Harris Corner Detector: Non-Maximal Suppression

Harris Corner: Advantages

- Translational invariance:



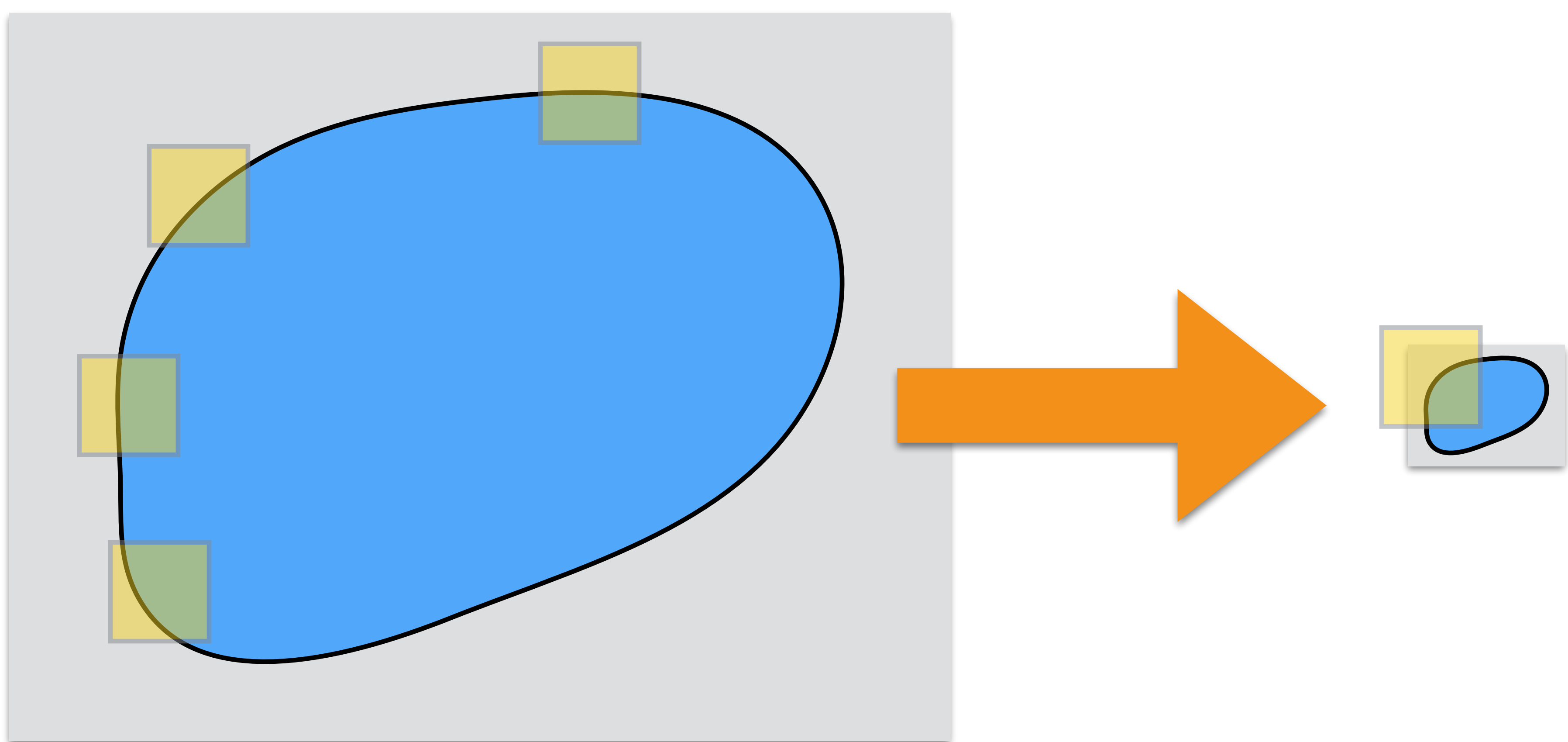
- Rotation invariance:



- 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!

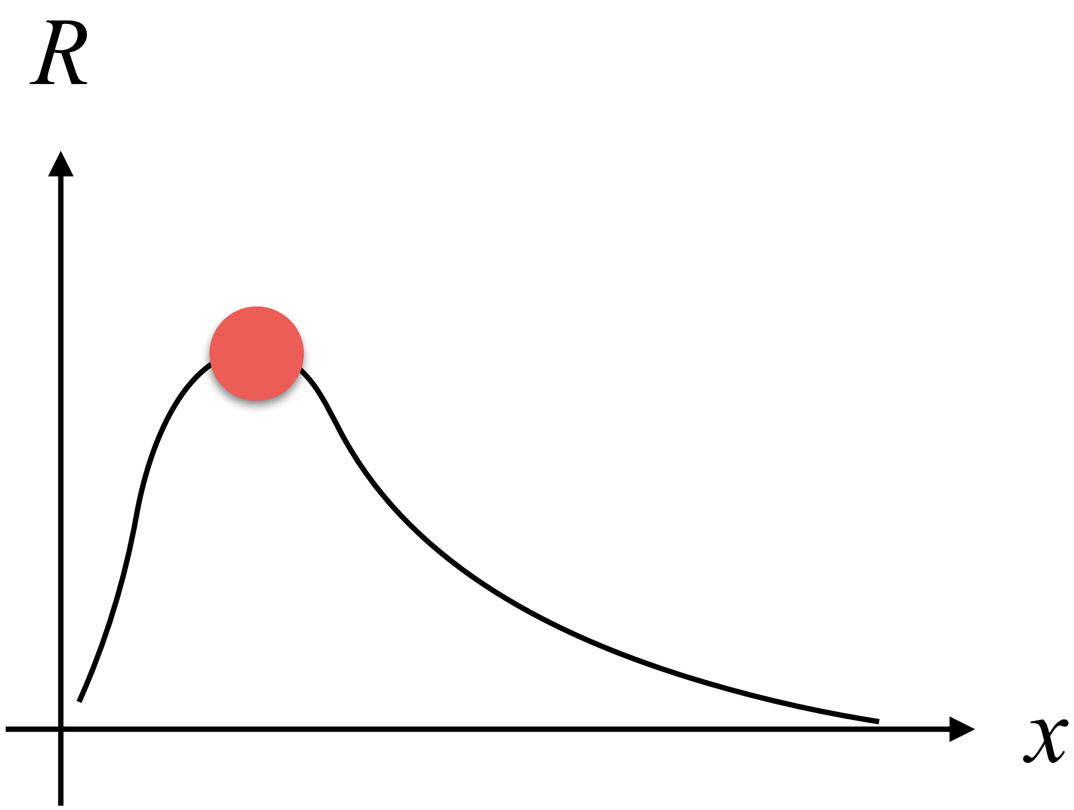
The same feature in different images can have different size!

The Scale Problem

Near Object

Far Object

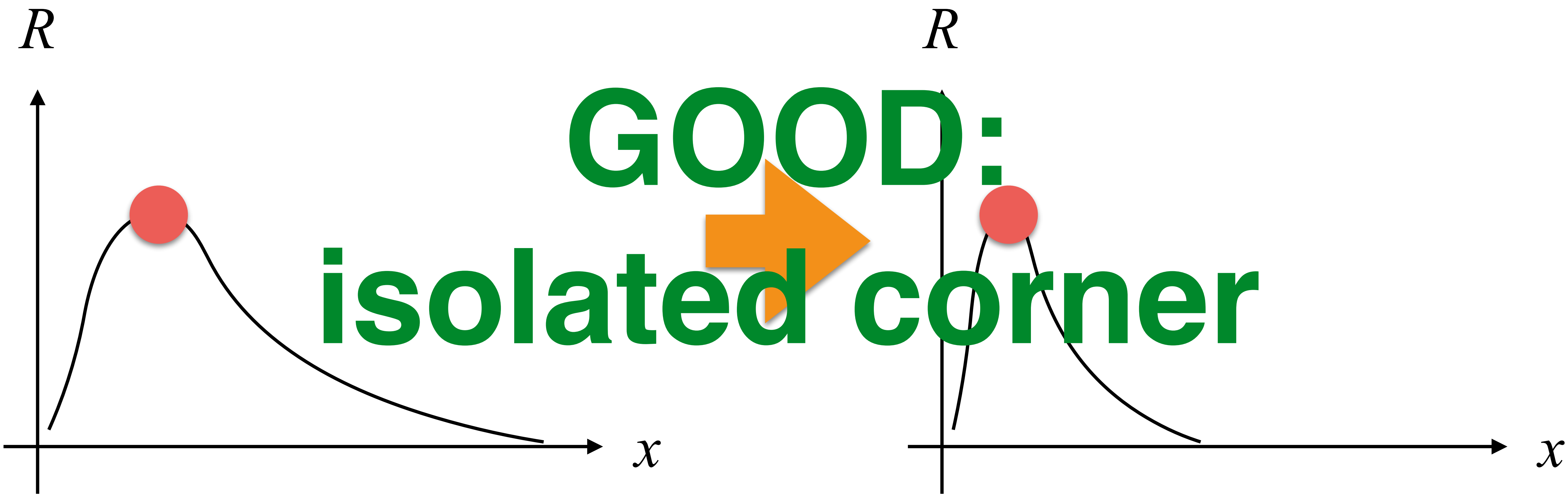
Scale Invariant: Stable Corners



Original

1/2 scale

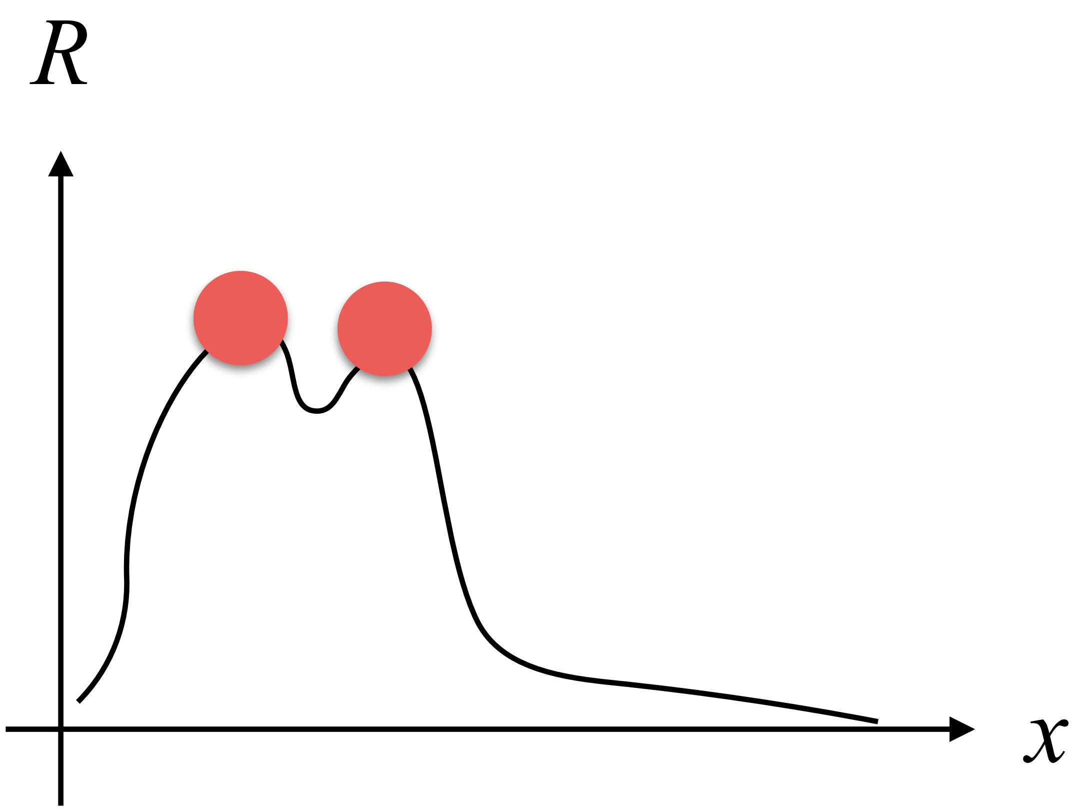
Scale Invariant: Stable Corners



Original

1/2 scale

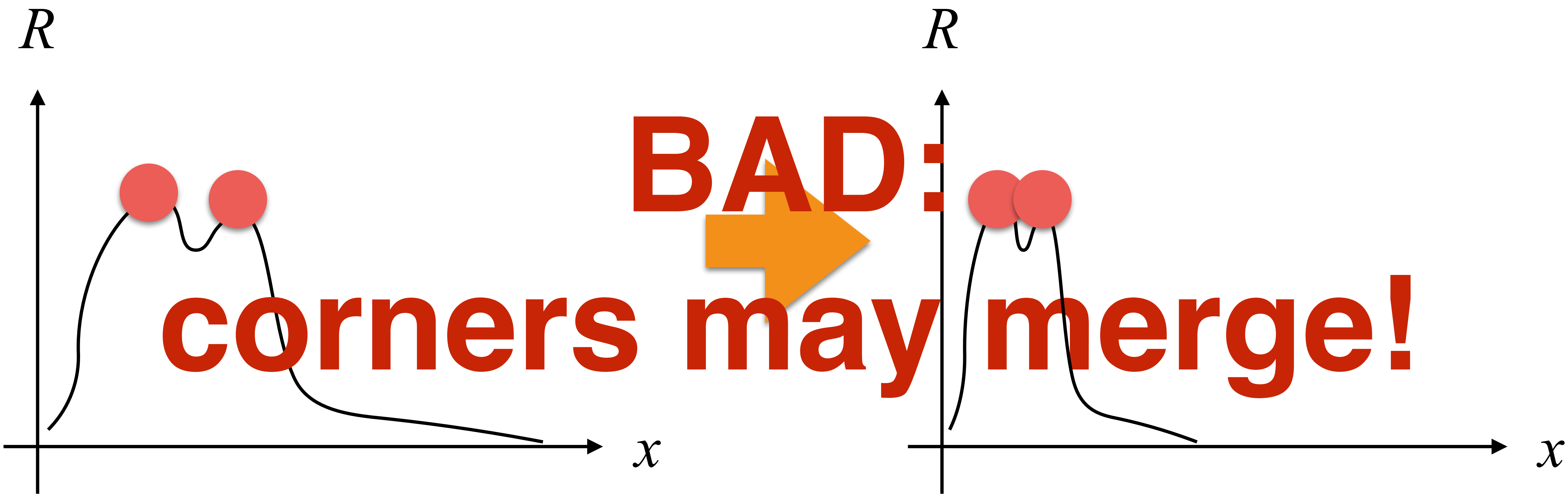
Scale Invariant: Unstable Corners



Original

1/2 scale

Scale Invariant: Unstable Corners



Original

1/2 scale

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

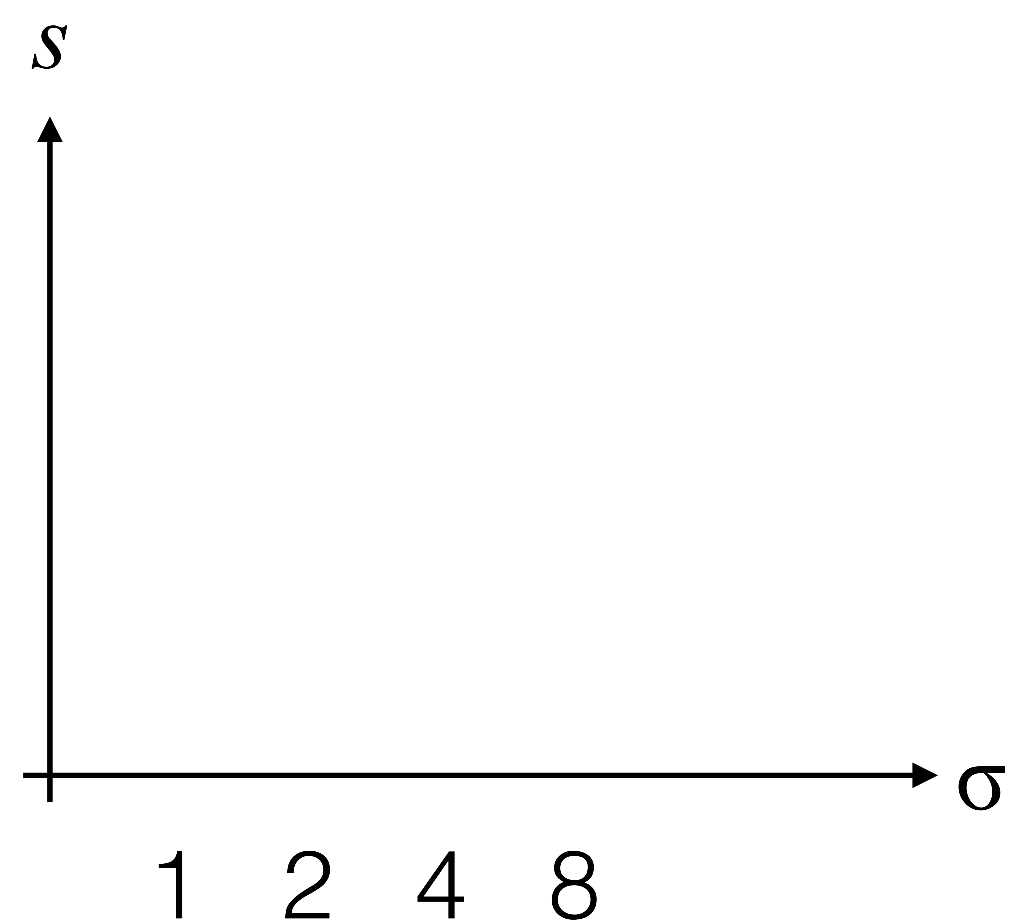
- The signature function, s, can give us an idea of the content of the image.
- Therefore, we need to find a maximum point of s for pixel of an input image!

Scale Invariant: The Approach

Let's build s at the red point!

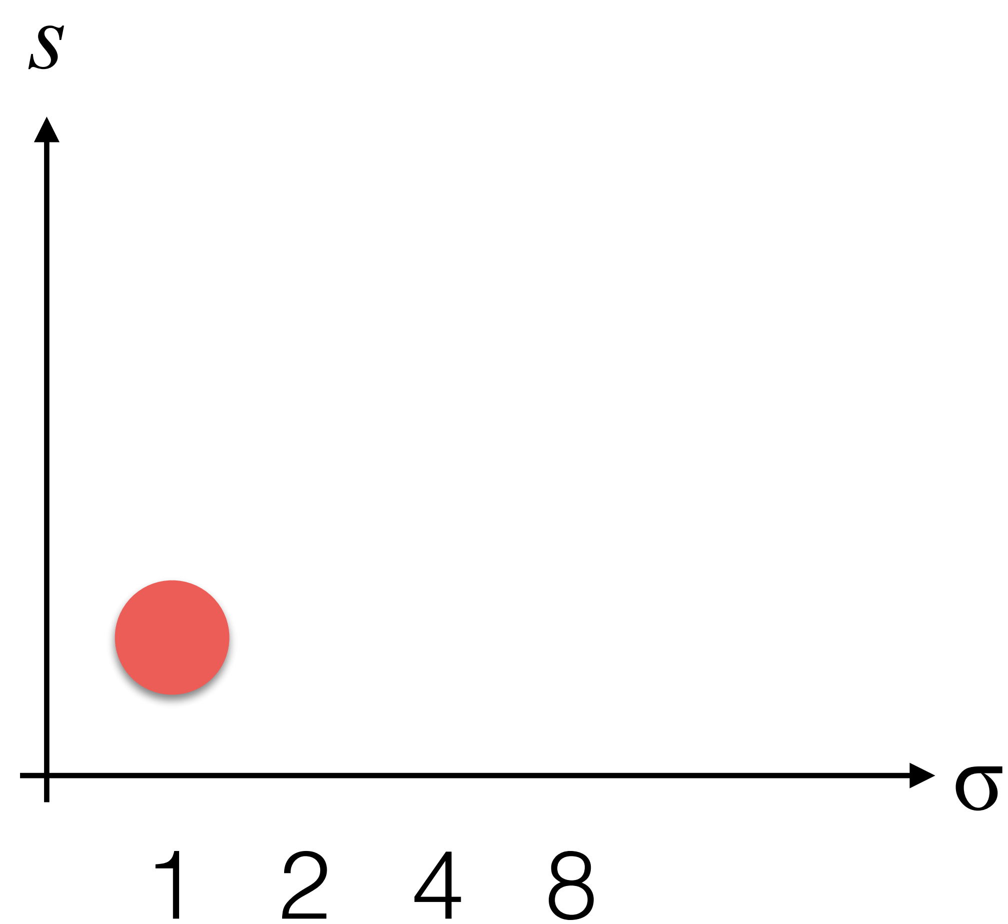
Scale Invariant: The Approach

This is our start!

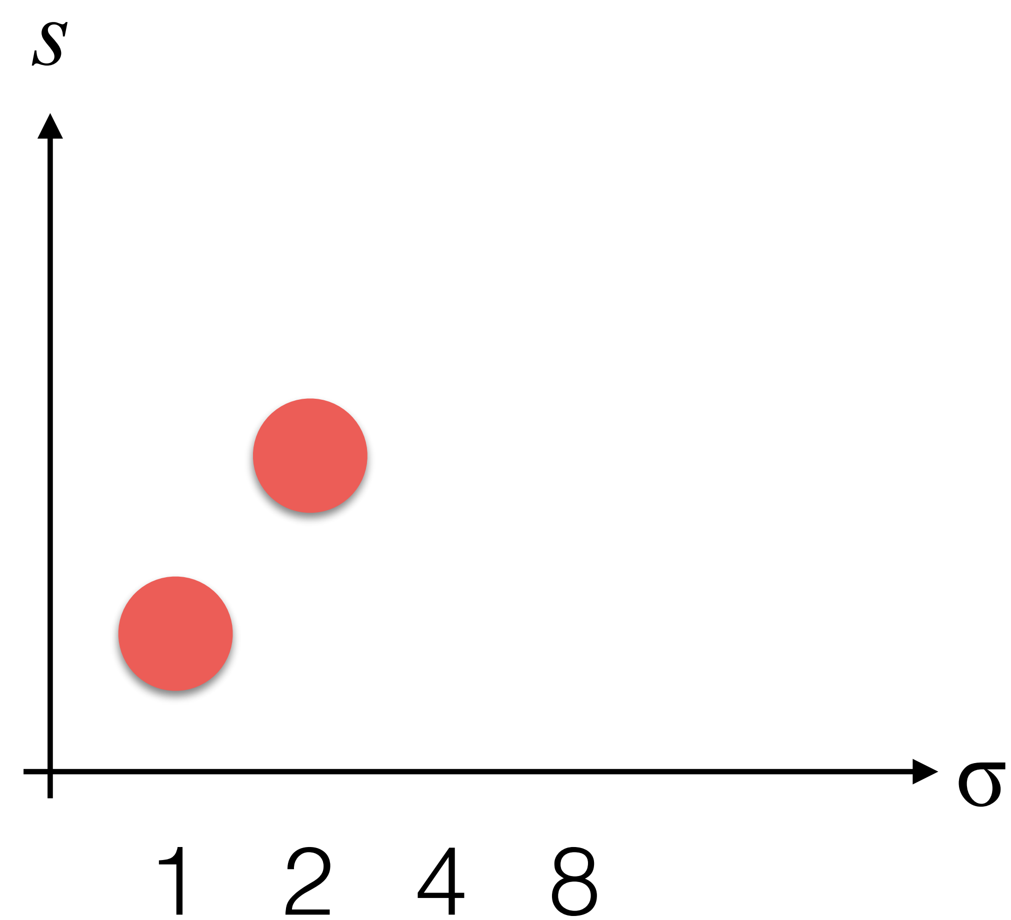


Scale Invariant: The Approach

$\sigma = 1$

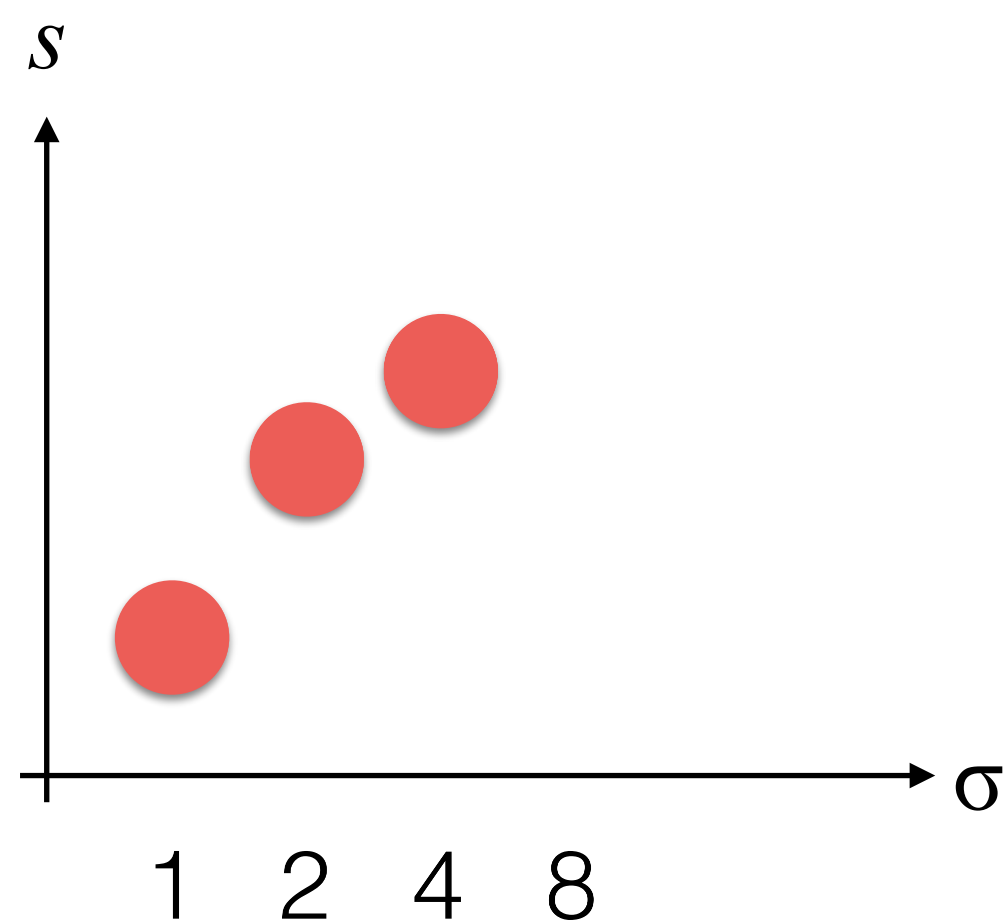


Scale Invariant: The Approach



$\sigma = 2$

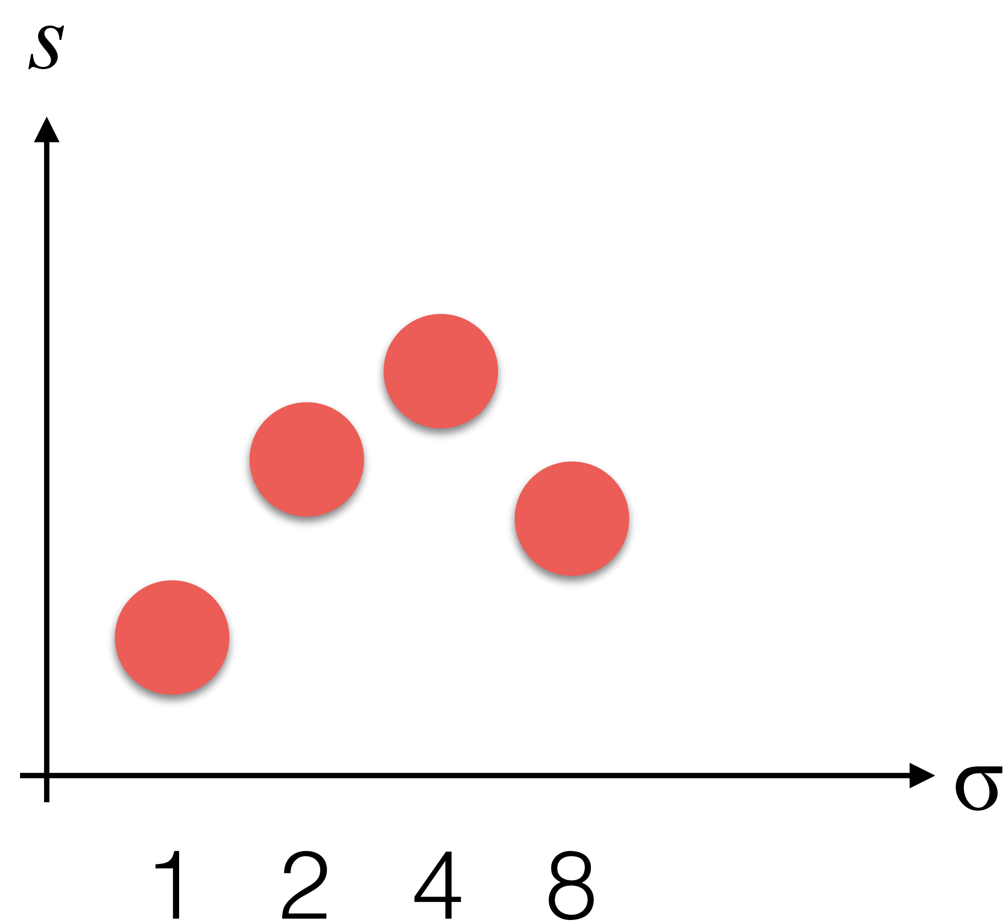
Scale Invariant: The Approach



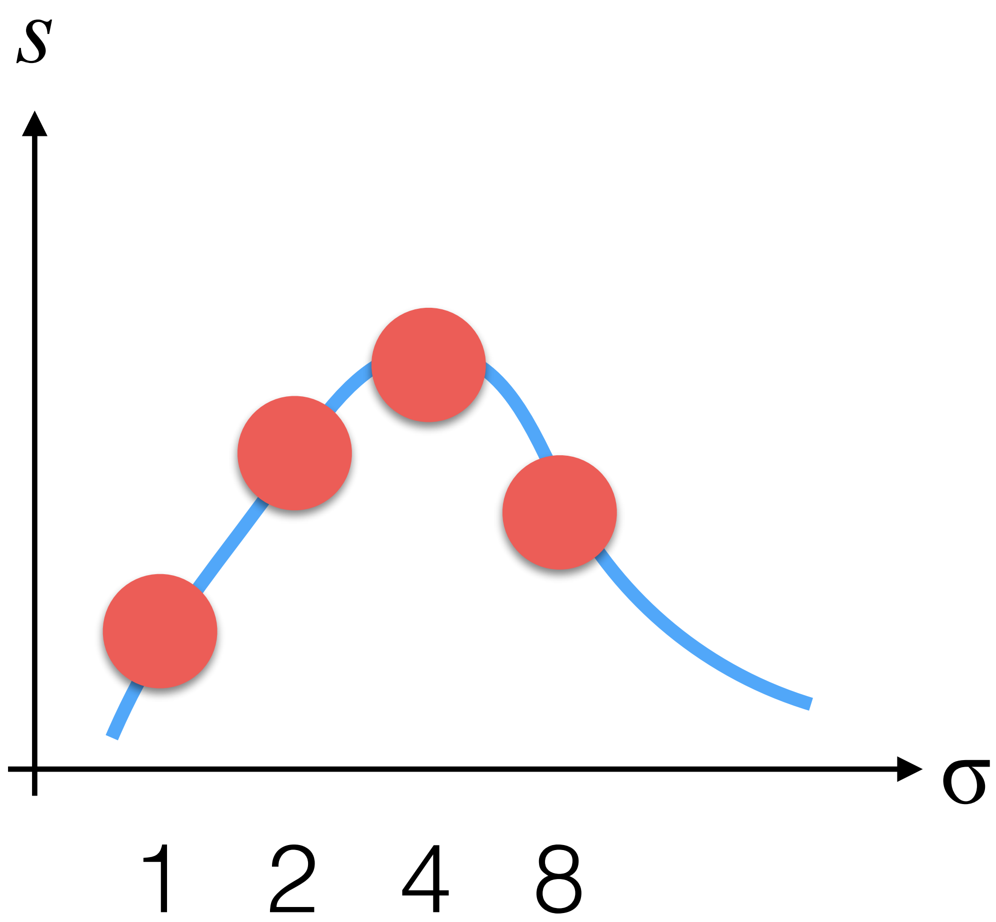
σ = 4

Scale Invariant: The Approach

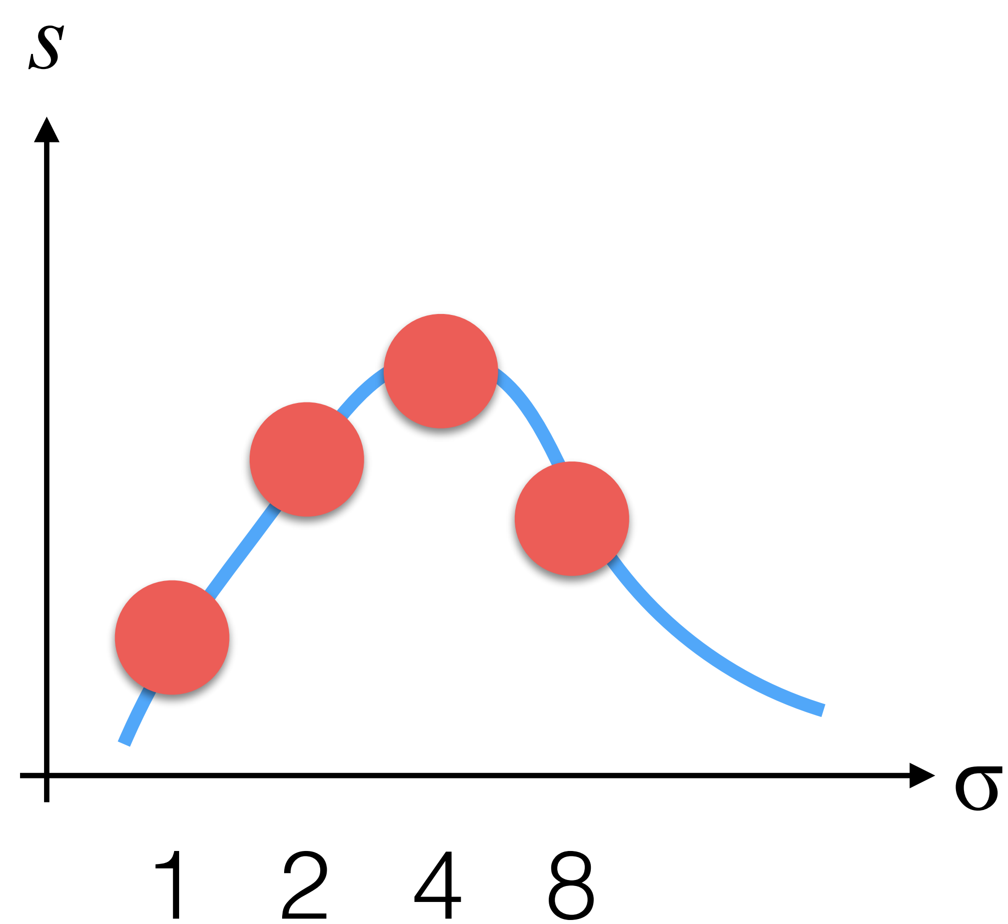
$\sigma = 8$



Scale Invariant: The Approach

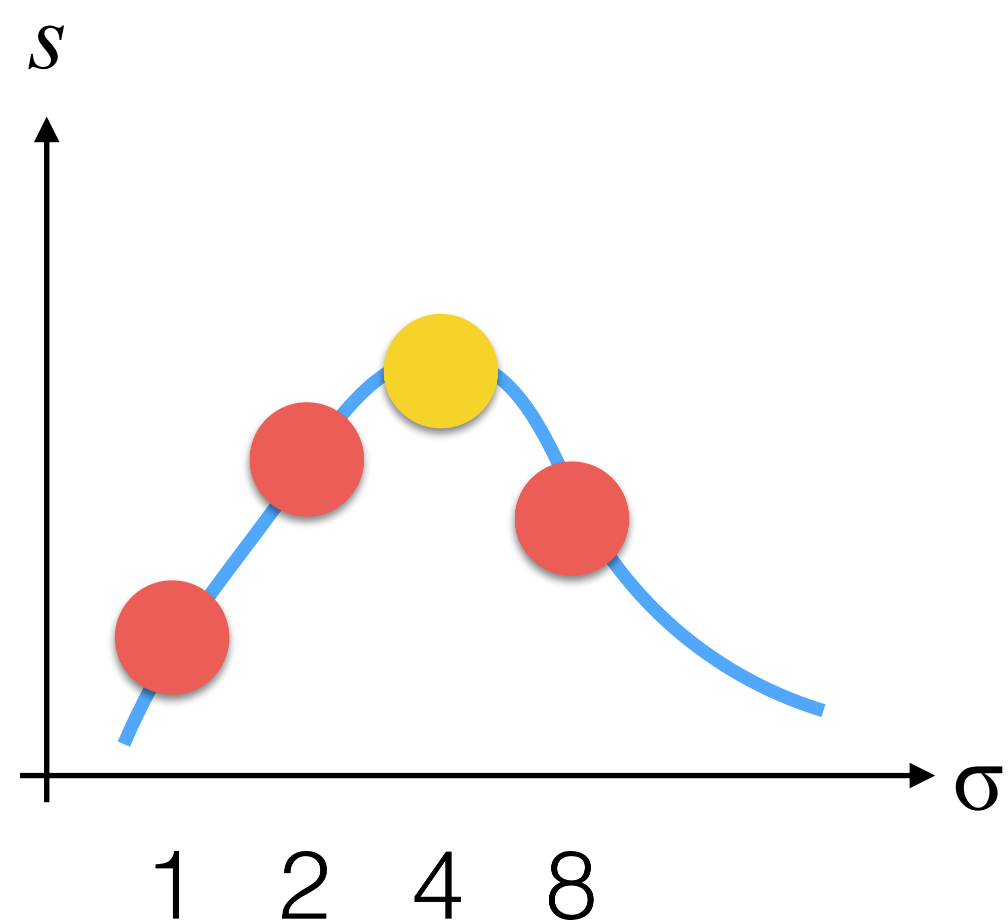


Scale Invariant: The Approach



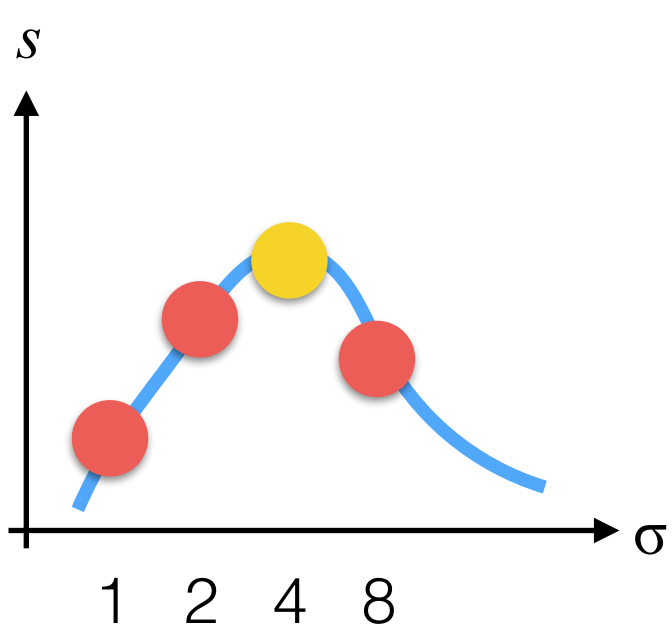
Which is σ for which *s* is the maximum?

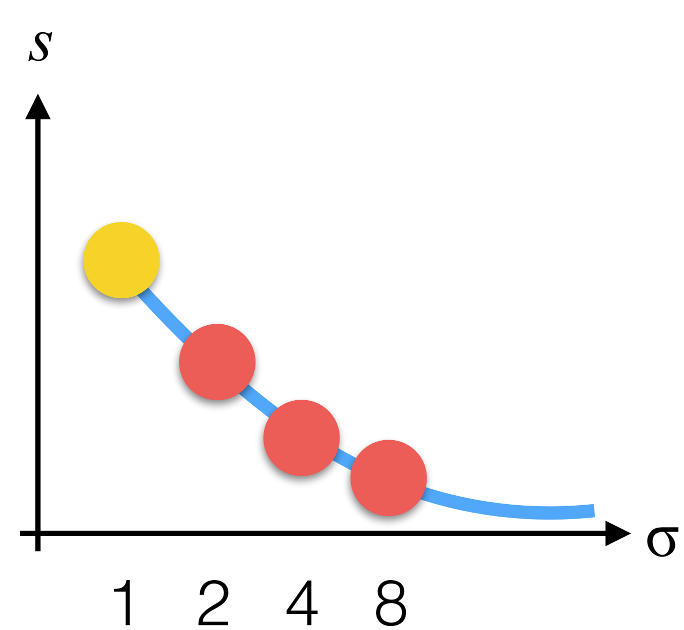
Scale Invariant: The Approach



It is $\sigma = 4$

Scale Invariant: The Approach





Extraction of Features

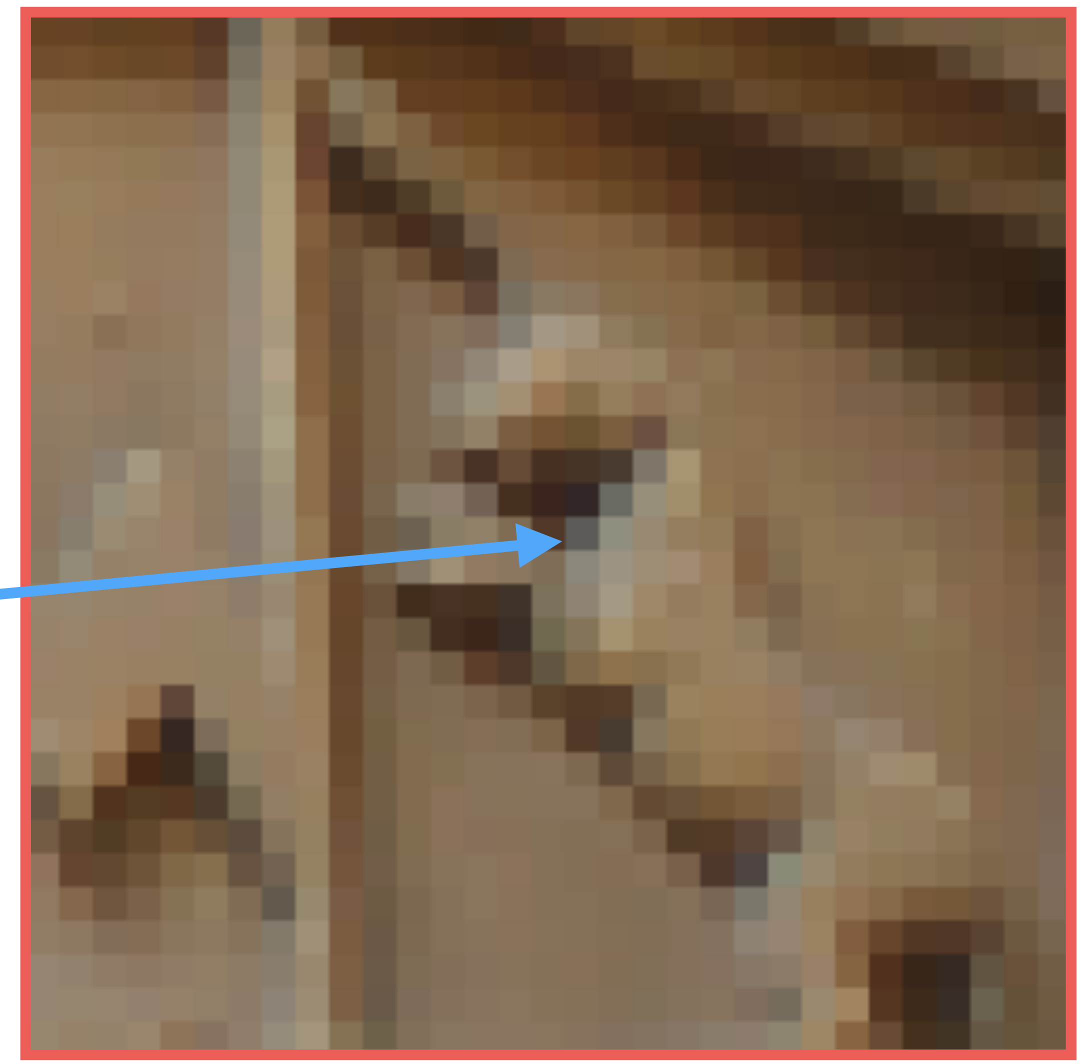
- General overview:
  - Computation of the scale for each pixel using the sigma value at which we have the maximum value of the signature function.
  - Computation of the Harris Corner using the scale to increase the size of the local window.

Feature Descriptors

Feature Descriptors

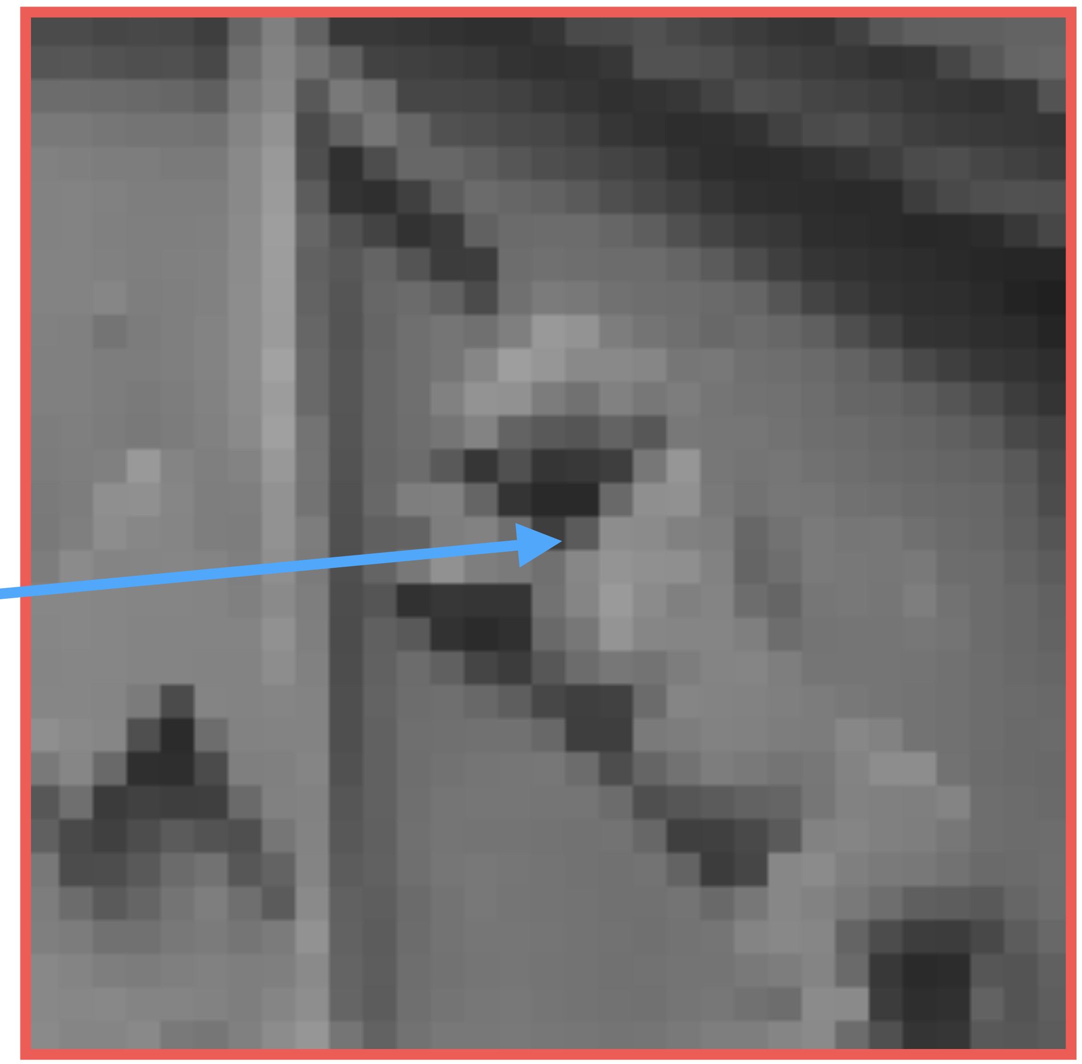
- Once we found our features (i.e., corners), we need to describe in a meaningful and possibly unique way.
  - Why?
    - We want compare corners between images in order to find correspondences between images.

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:

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

- For efficiency, it is encoded as a number:

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

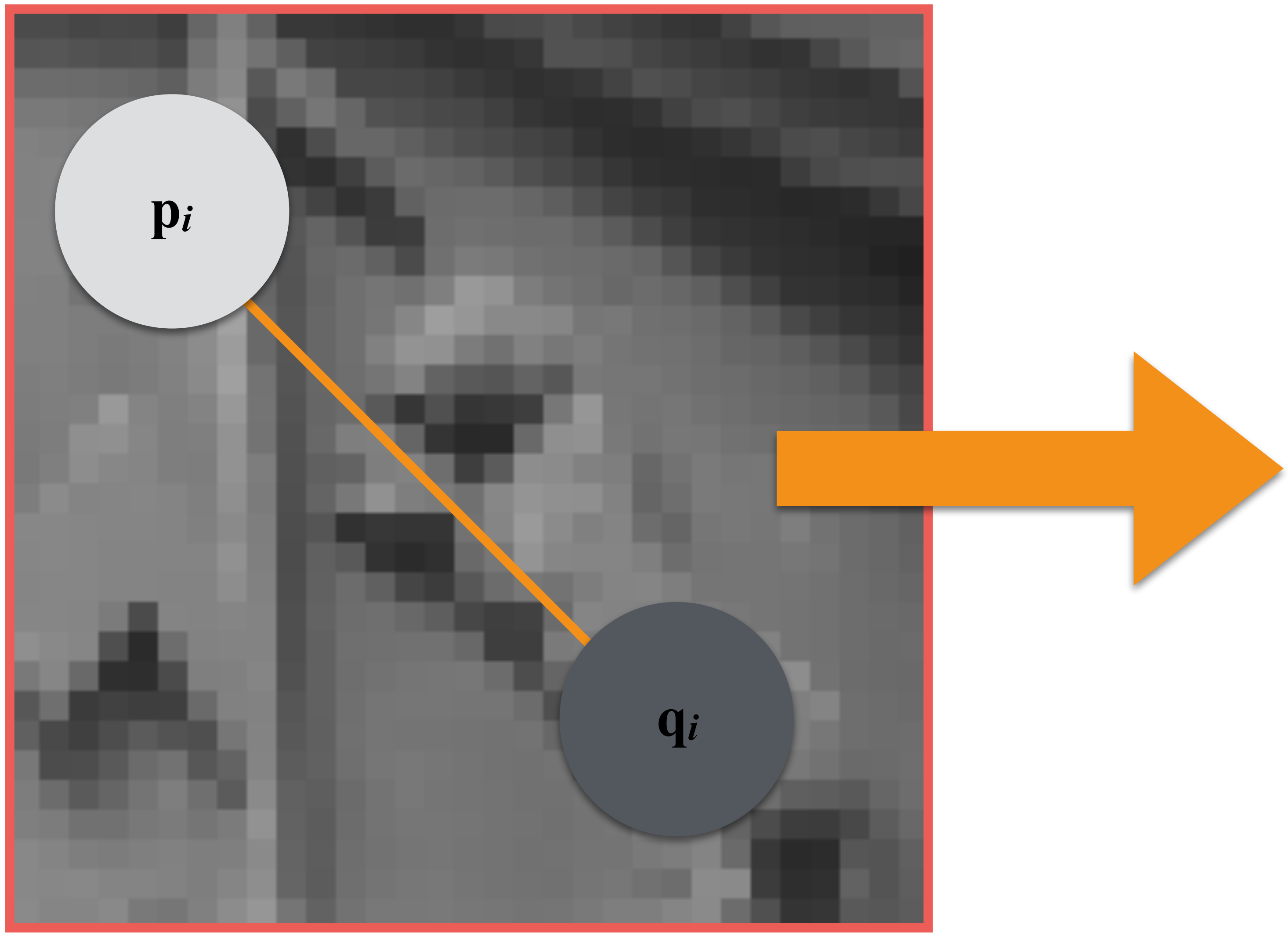
BRIEF Descriptor

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

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

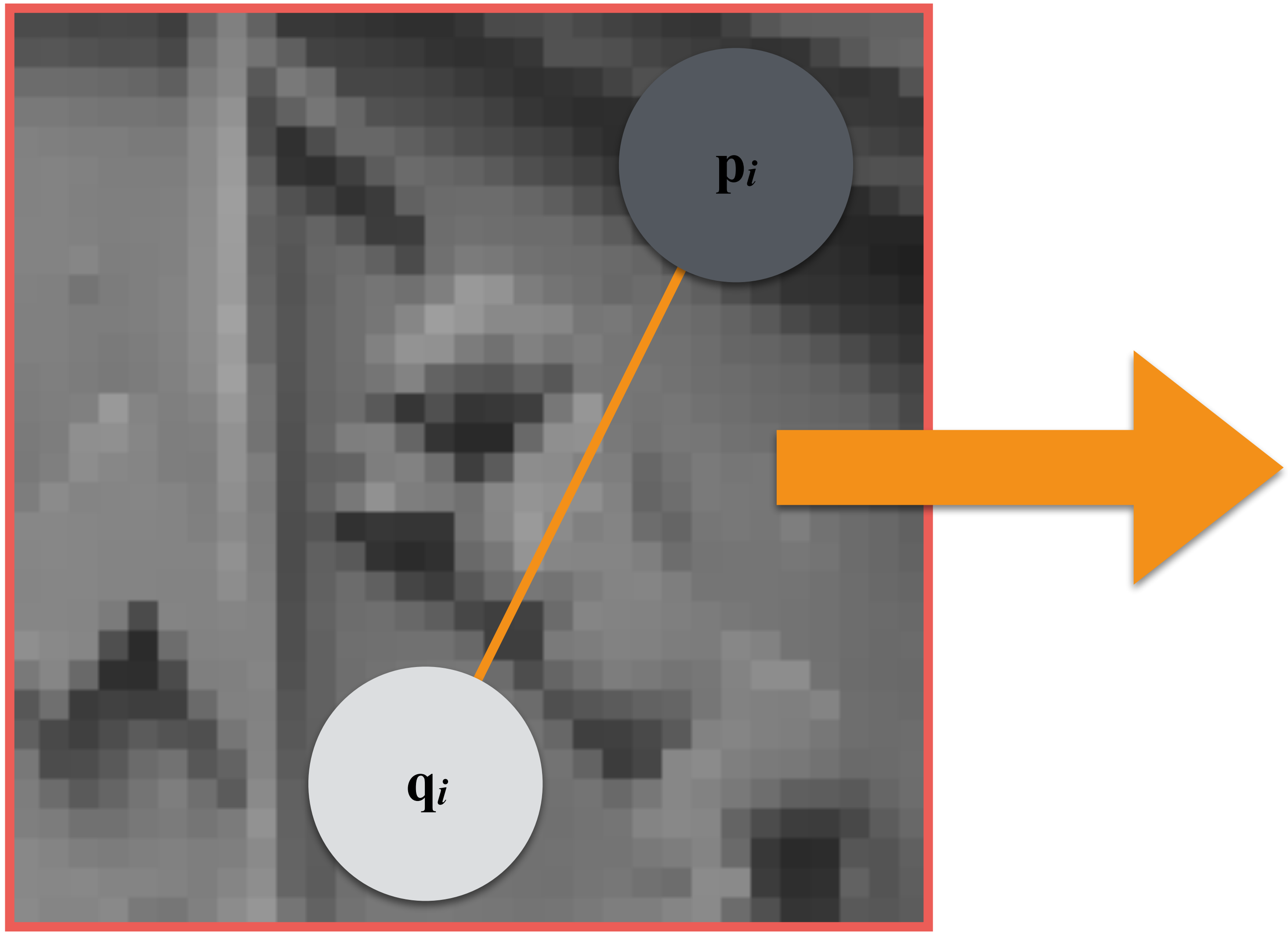
- where $\mathbf{p}_i$ and $\mathbf{q}_i$ are the coordinates (x, y) of two random points in P.

BRIEF Descriptor: Example



$b_i = 0$

BRIEF Descriptor: Example



$b_i = 1$

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):

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

- **Higher the closer**:

  - This is a very computationally efficient distance function.

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:

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

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:

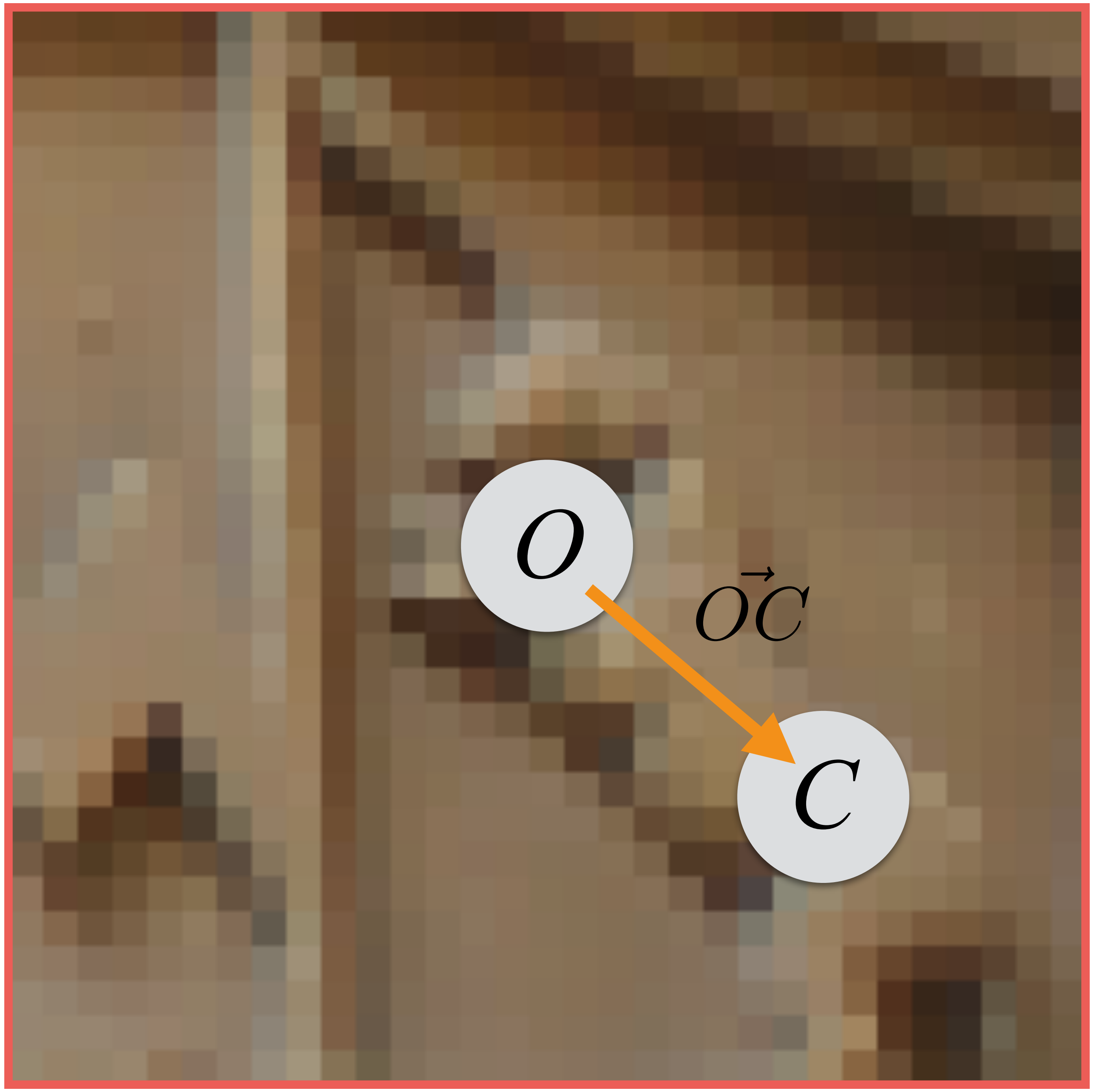
$$m_{a,b} = \sum_{x,y \in P} x^a y^b P(x, y)$$

- 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

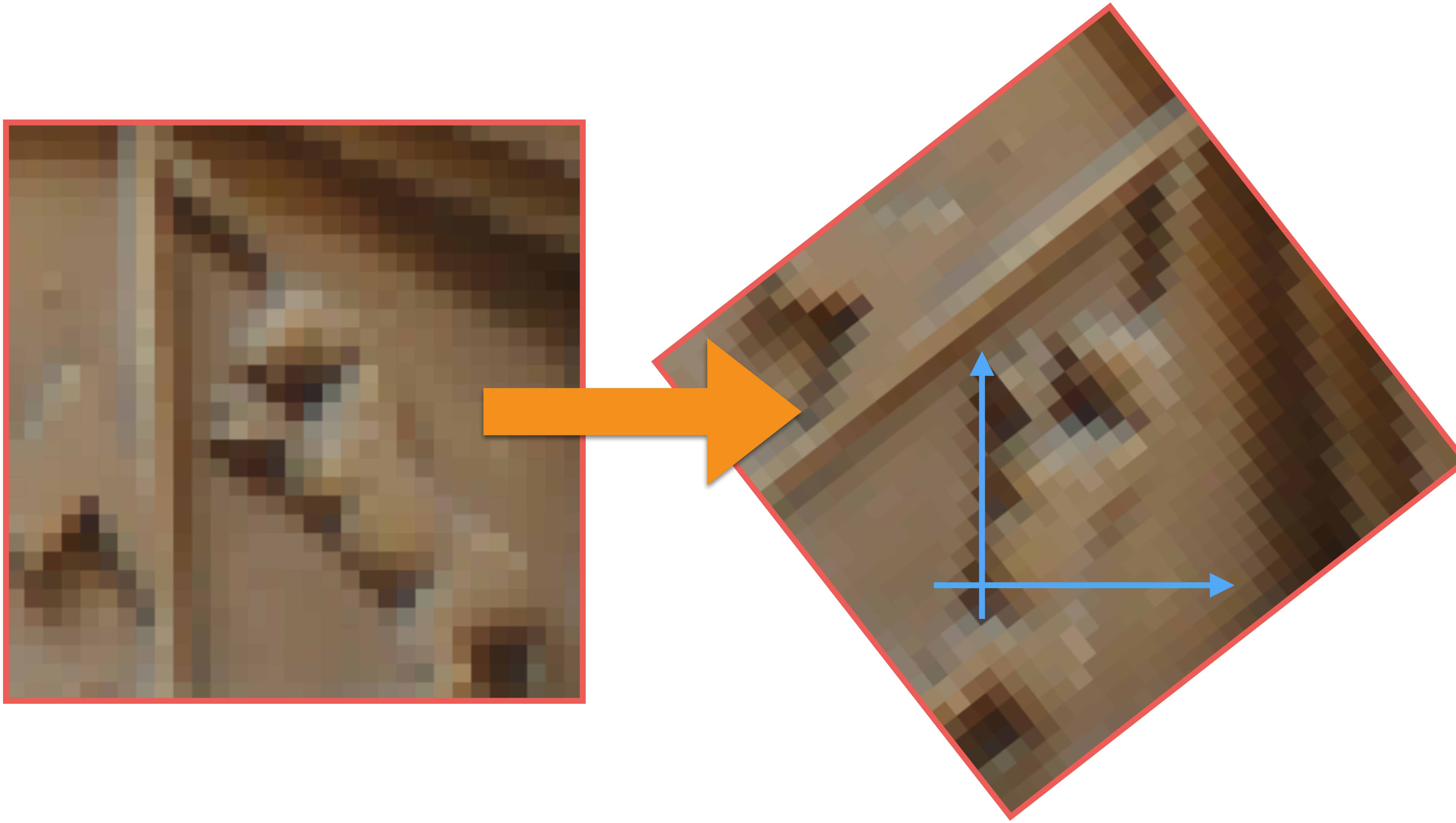
$$\theta = \mathrm{atan2}(m_{0,1}, m_{1,0})$$

- From this, we can rotate points stored in A as

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

- where R_θ is a 2D rotation matrix.

ORB Descriptor



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:

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

$$\theta(i,j) = \mathrm{atan2}\Big(P_y(i,j), P_x(i,j)\Big)$$

SIFT Descriptor: Patch Orientation

- A histogram, H, of directions (**18** bins) is created for each orientation taking into account magnitude.

- Let's say we have a gradient with $m = 10$ and $\theta = 45°$. How do we insert it in the histogram H?

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

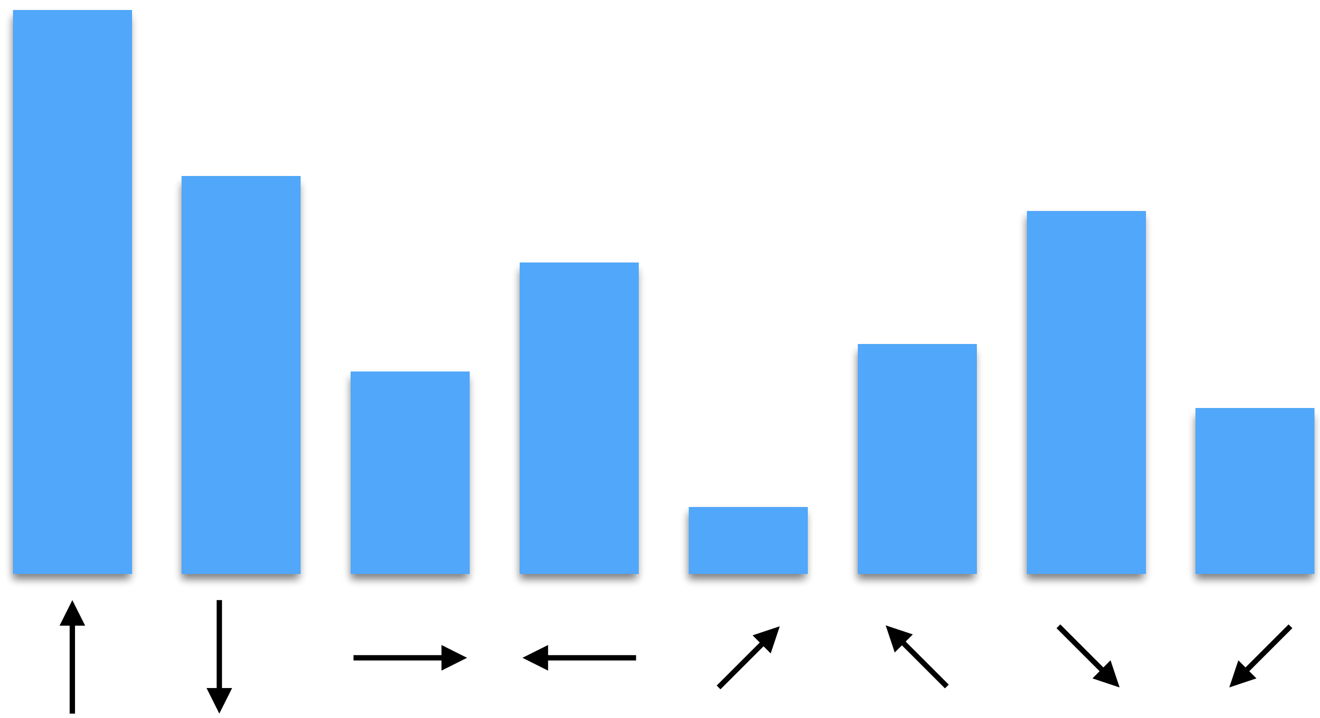
$$i = \left\lfloor \frac{45}{20} \right\rfloor = 2$$

  - Then, we update H as $\quad H(i) = H(i) + 10$

  - We repeat this process for all gradients in the patch!

SIFT Descriptor: Patch Orientation

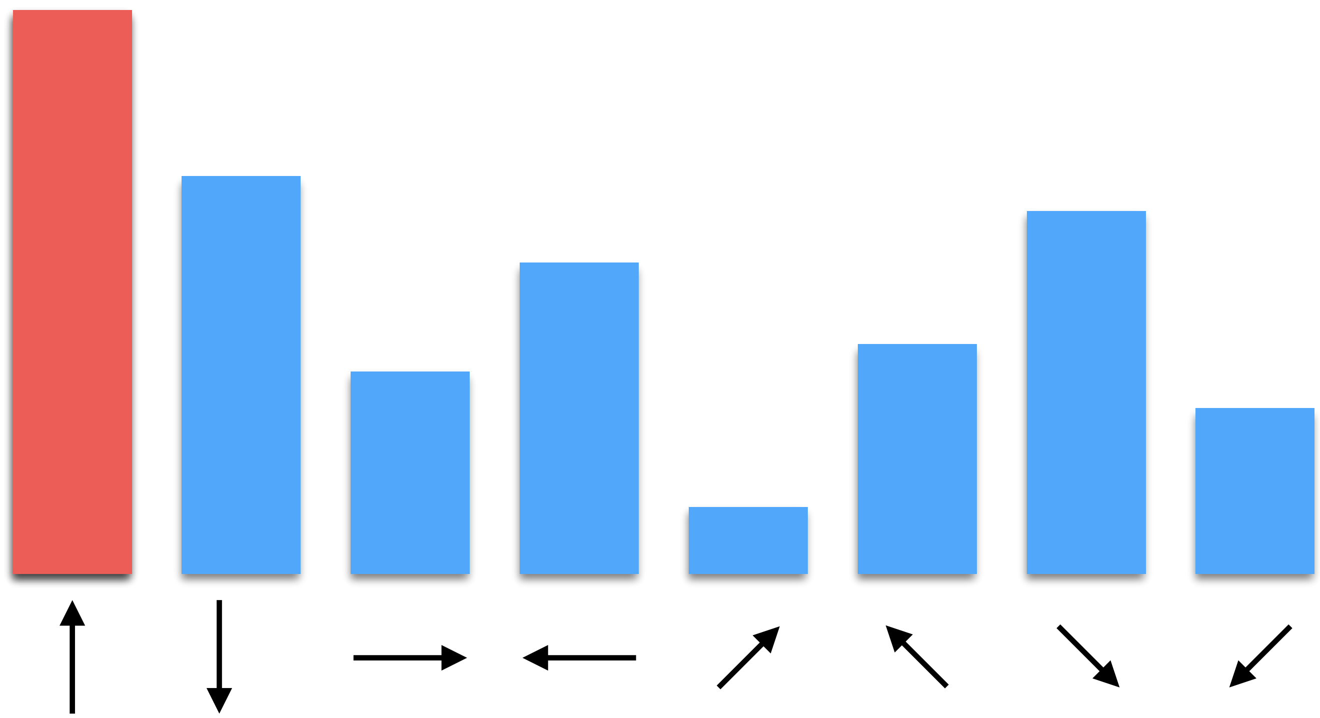
- Finally, we get this (a toy example with 8 bins!):



- 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 (a toy example with 8 bins!):

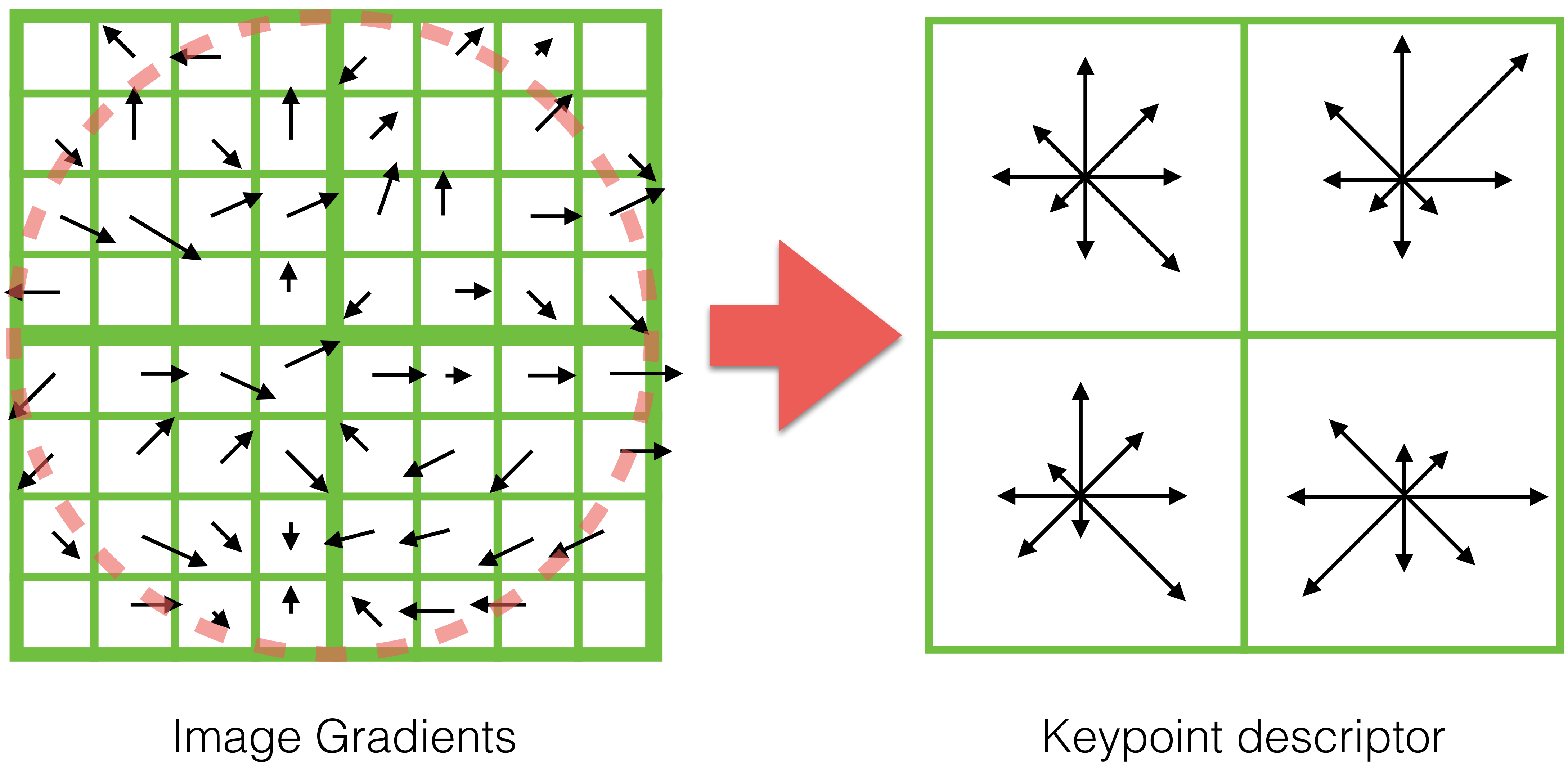


- The patch orientation, θ, is given by the highest peak:
  - If we have two equal peaks, we take the as winner the first one in histogram.

SIFT Descriptor

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

SIFT Descriptor: Example
Dividing the Patch into 2x2 Blocks



Note: when we compute gradients, we rotate them using the computed orientation!

SIFT Descriptor: Test

- We test the differences as distance between histograms:

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

- **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

- **Input**: two descriptor lists (with different lengths), $\mathbf{desc}_1$ and $\mathbf{desc}_2$, respectively of image I_1 and I_2.

- **Output**: two arrays with indices of matches for each list.

  - For $\mathbf{desc}_1$: $\mathbf{m}_1 = [10, 23, \ldots, 1]^\top$

  - For $\mathbf{desc}_2$: $\mathbf{m}_2 = [100, 4, \ldots, 2]^\top$

Matching: Example

- Let's say we have 5 descriptors in $\mathbf{desc}_1$
- Let's say we have 7 descriptors in $\mathbf{desc}_2$
- **Output**:
  - $\mathbf{m}_1 = [3, 5, 6, 7, 1]$
  - $\mathbf{m}_2 = [2, 3, 4, 5, 1, 1, 3]$

Matching: Example

- $\mathbf{m}_1$ = [3, 5, 6, 7, 1]
  - This means that the 1st descriptor in $\mathbf{desc}_1$ is matched with the 3rd in $\mathbf{desc}_2$.
  - This means that the 2nd descriptor in $\mathbf{desc}_1$ is matched with the 5th in $\mathbf{desc}_2$.
  - This means that the 3rd descriptor in $\mathbf{desc}_1$ is matched with the 6th in $\mathbf{desc}_2$.
  - This means that the 4th descriptor in $\mathbf{desc}_1$ is matched with the 7th in $\mathbf{desc}_2$.
  - This means that the 5th descriptor in $\mathbf{desc}_1$ is matched with the 1st in $\mathbf{desc}_2$.

Matching: Example

- $\mathbf{m}_2$ = [2, 3, 4, 5, 1, 1, 3]
  - This means that the 1st descriptor in $\mathbf{desc}_2$ is matched with the 2nd in $\mathbf{desc}_1$.
  - This means that the 2nd descriptor in $\mathbf{desc}_2$ is matched with the 3rd in $\mathbf{desc}_1$.
  - This means that the 3rd descriptor in $\mathbf{desc}_2$ is matched with the 4th in $\mathbf{desc}_1$.
  - This means that the 4th descriptor in $\mathbf{desc}_2$ is matched with the 5th in $\mathbf{desc}_1$.
  - This means that the 5th descriptor in $\mathbf{desc}_2$ is matched with the 1st in $\mathbf{desc}_1$.
  - This means that the 6th descriptor in $\mathbf{desc}_2$ is matched with the 1st in $\mathbf{desc}_1$.
  - This means that the 7th descriptor in $\mathbf{desc}_2$ is matched with the 3rd in $\mathbf{desc}_1$.

Matching: Brute Force Algorithm

- A simple method to find a ***matched descriptor*** in $\mathbf{desc}_2$ for each descriptor in $\mathbf{desc}_1$:
    - For each descriptor $\mathbf{d}_{1,i}$ in $\mathbf{desc}_1$ to test all descriptors $\mathbf{desc}_2$ and to keep as matched the ***closest*** (in terms of distance).

Matching: Brute Force Algorithm

For each descriptor $\mathbf{d}_{1,i}$ in $\mathbf{desc}_1$:

matched = -1;

dist_matched = BOTTOM;

For each descriptor $\mathbf{d}_{2,j}$ in $\mathbf{desc}_2$:

if Closer($D(\mathbf{d}_{1,i}, \mathbf{d}_{2,j})$, dist_matched)

matched = j;

dist_matched = $D(\mathbf{d}_i, \mathbf{d}_i)$;

endif

Matching: Brute Force Algorithm

For each descriptor $\mathbf{d}_{1,i}$ in $\mathbf{desc}_1$:

matched = -1;

dist_matched = BOTTOM;

For each descriptor $\mathbf{d}_{2,j}$ in $\mathbf{desc}_2$:

if Closer($\mathrm{D}(\mathbf{d}_{1,i}, \mathbf{d}_{2,j})$, dist_matched)

matched = j;

dist_matched = $\mathrm{D}(\mathbf{d}_i, \mathbf{d}_i)$;

endif

Matching: Brute Force Algorithm

For each descriptor $\mathbf{d}_{1,i}$ in $\mathbf{desc}_1$:

matched = -1;

dist_matched = BOTTOM;

For each descriptor $\mathbf{d}_{2,j}$ in $\mathbf{desc}_2$:

if Closer($D(\mathbf{d}_{1,i}, \mathbf{d}_{2,j})$, dist_matched)

matched = j;

dist_matched = $D(\mathbf{d}_i, \mathbf{d}_i)$;

endif

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

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 $\mathbf{desc}_2$ and n in $\mathbf{desc}_2$. In the worst case, we need to compare descriptors $n^2/2$.

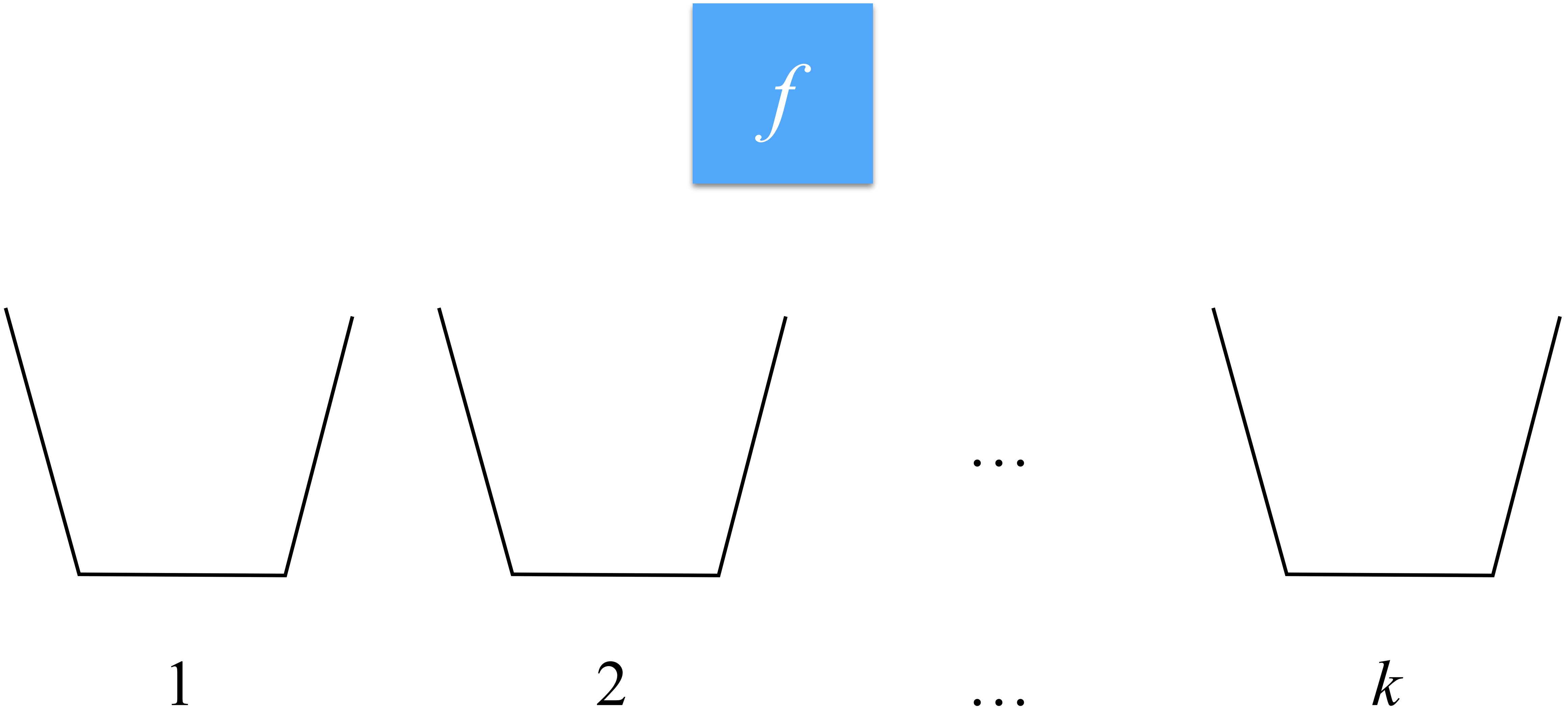
Matching: Improving Efficiency

- How can we improve (approximating results)?
- Hashing:
  - We create k bucket.
  - Each ***descriptor*** $\mathbf{d}_{2,i}$ of I_2 s assigned to a bucket using a function f, called hash function. This is defined as:

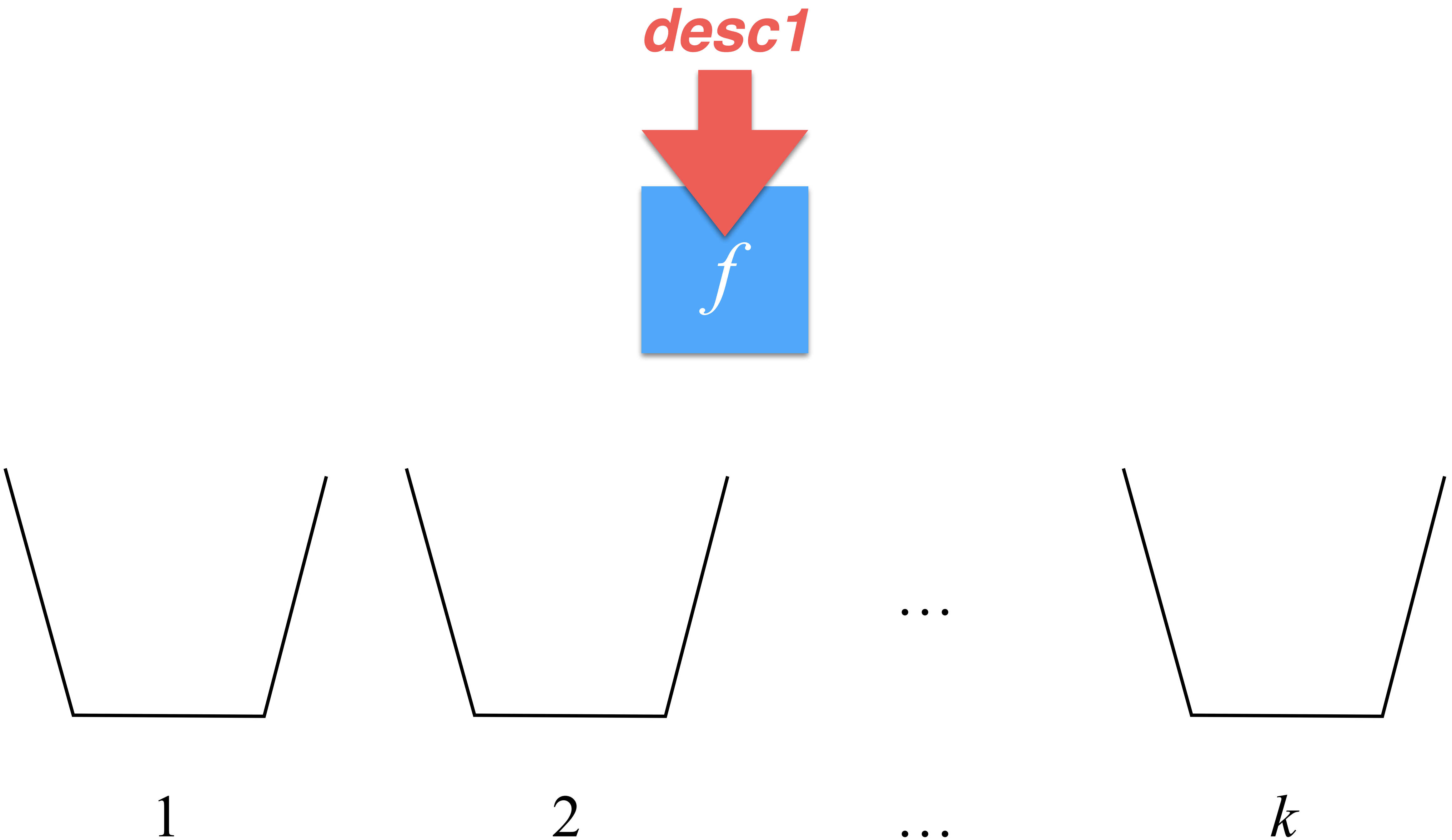
$$f\text{: } \textit{descriptor} \longrightarrow [1, k] \quad \text{(positive integer numbers!)}$$

  - 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.

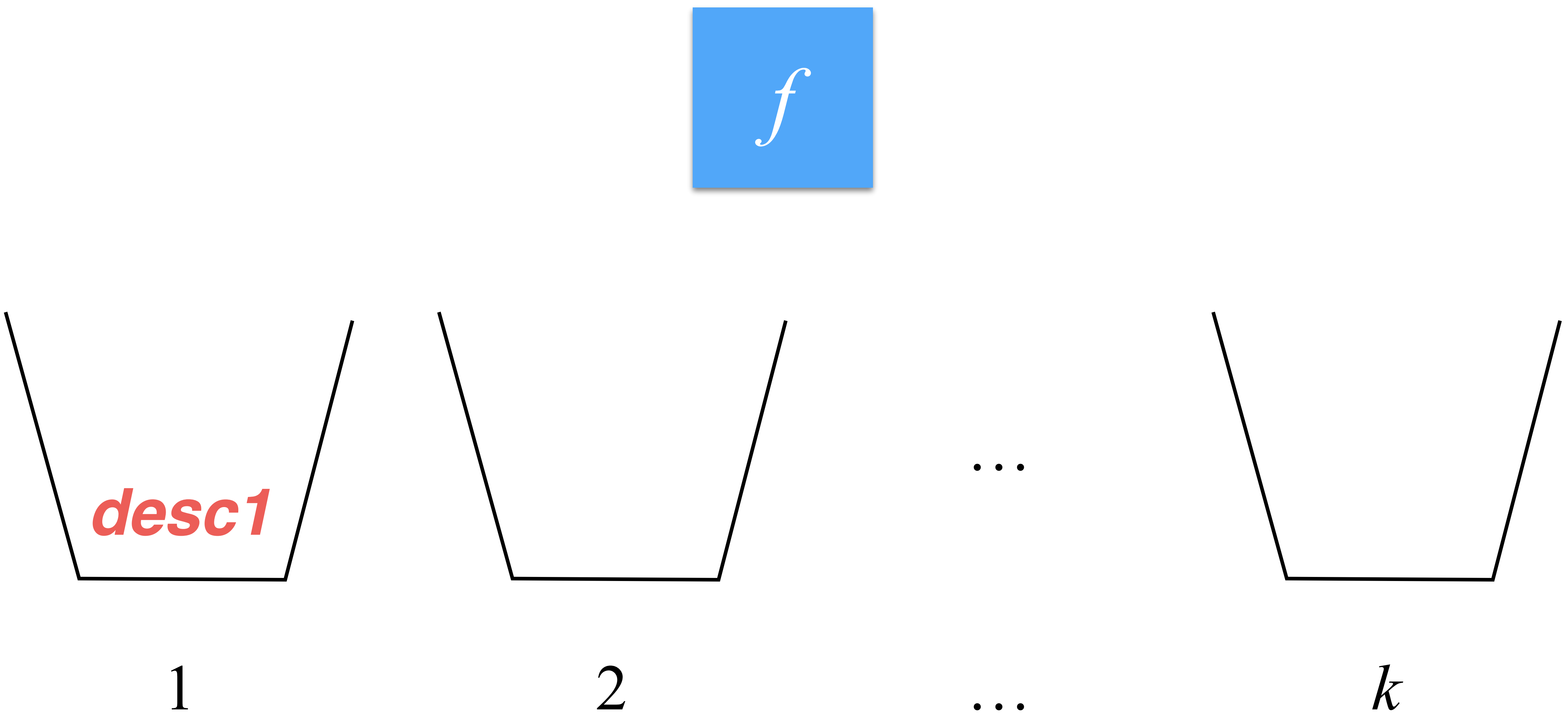
Matching:
Improving Efficiency



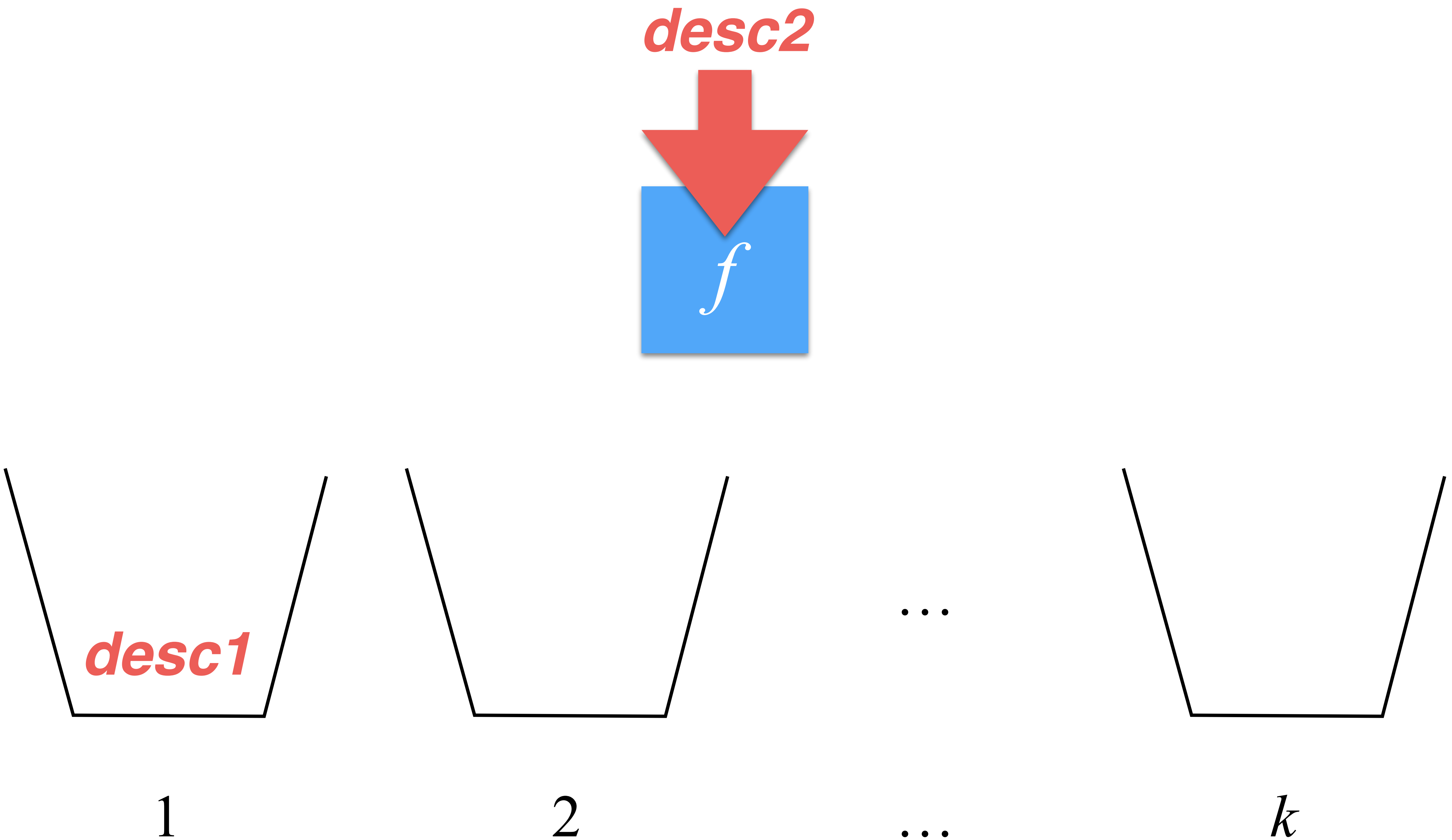
Matching:
Improving Efficiency



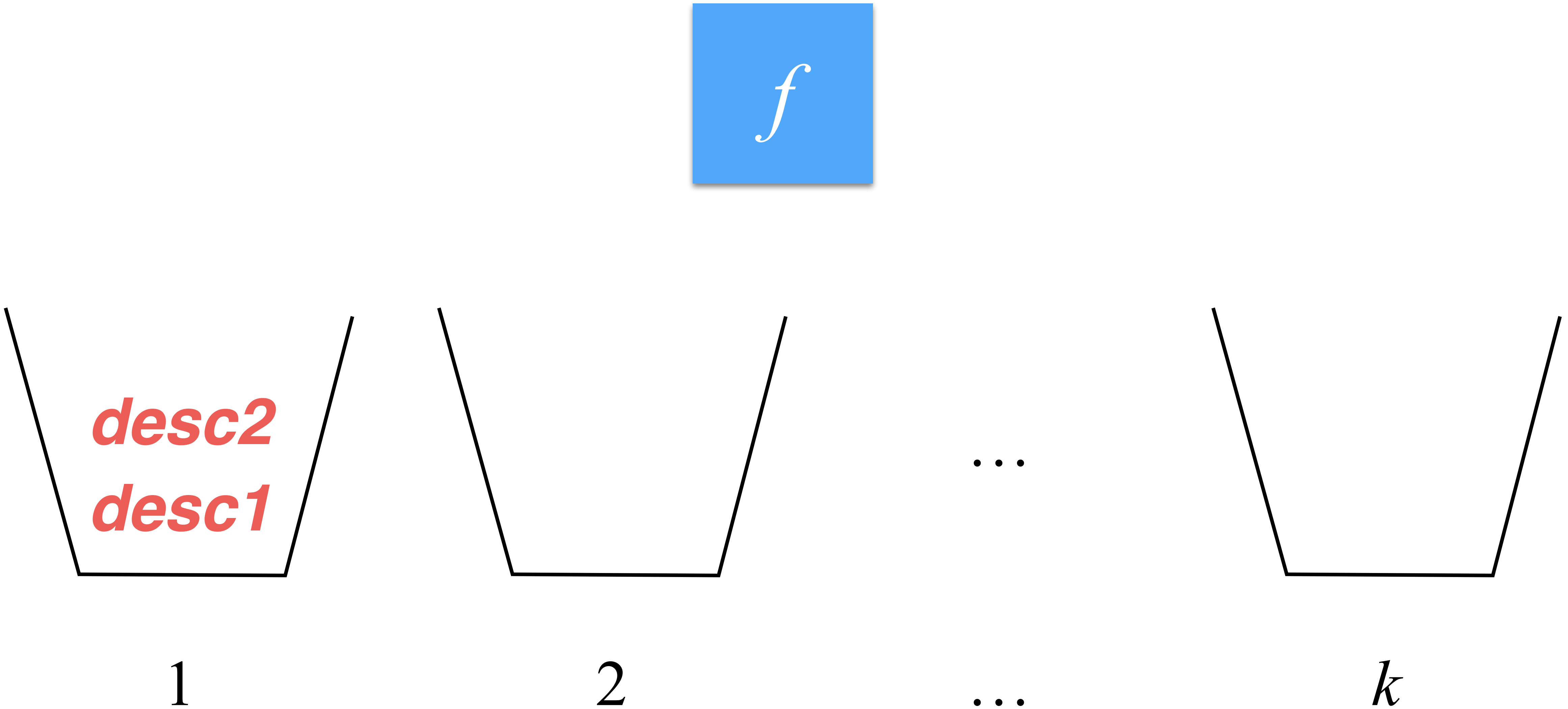
Matching: Improving Efficiency



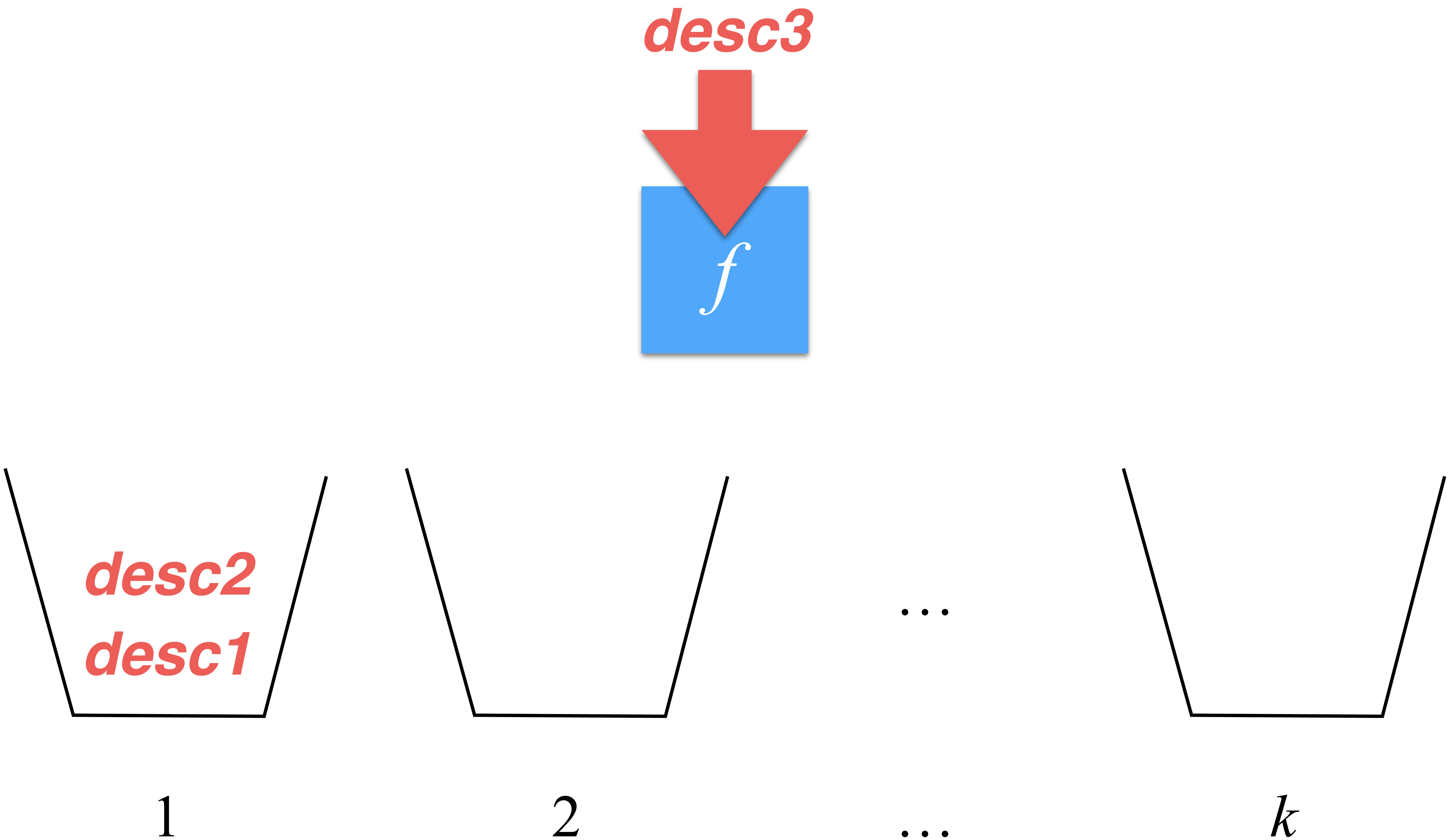
Matching: Improving Efficiency



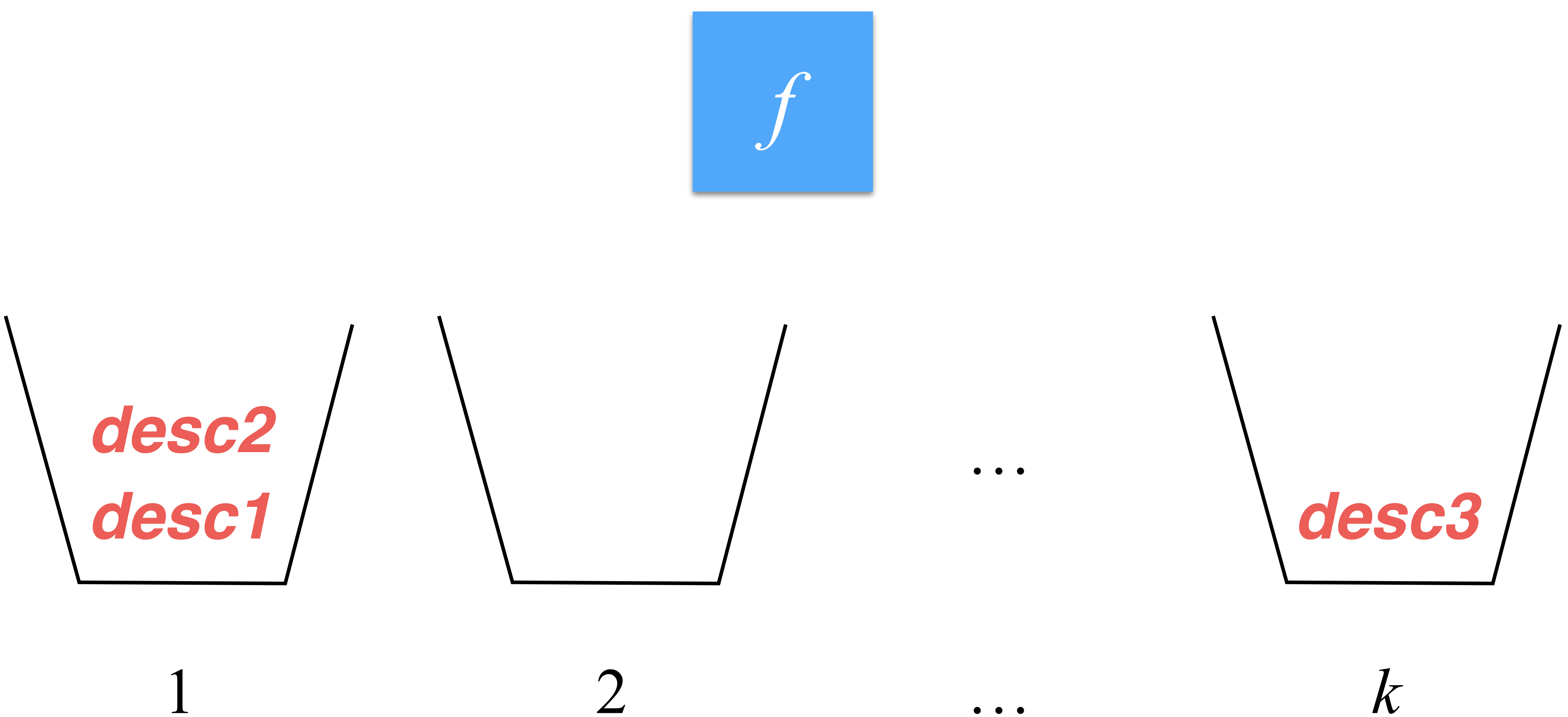
Matching:
Improving Efficiency



Matching: Improving Efficiency



Matching: Improving Efficiency



etc.

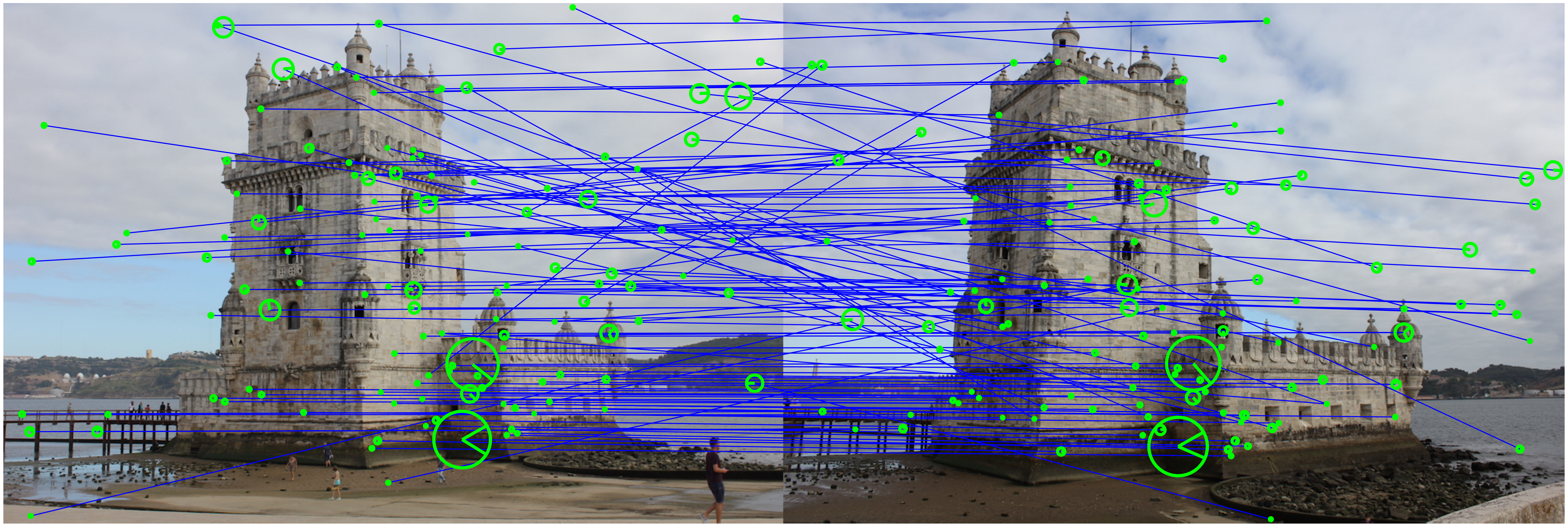
Matching: Improving Efficiency

- Now, we have all descriptors of I_2 into buckets.
- To find a match for a descriptor $\mathbf{d}_{1,i}$ of I_1, we apply f to $\mathbf{d}_{1,i}$. In this way, we obtain a bucket number, let's call it T.
- We run the brute force method for T.

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!