

# Lecture 20: Fitting geometric models

# Announcements

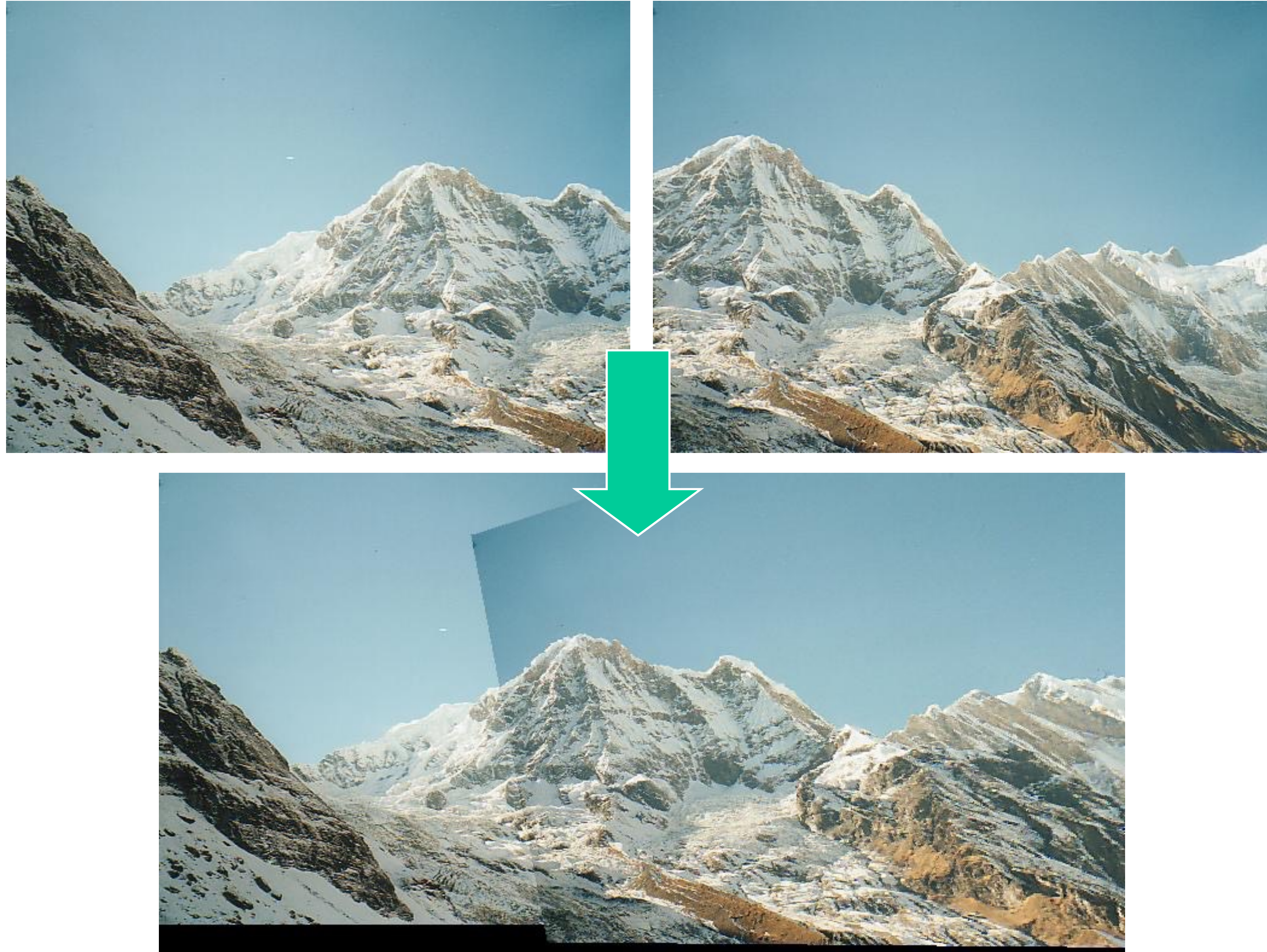
- PS8 - PS10 will be a bit easier (and PS10 will be short)
- Section this week: geometry tutorial
- My office hours next Monday (11/16) are cancelled
  - Can chat in Friday OH instead, or by appointment

# Today

- Finding correspondences
- Fitting a homography
- RANSAC

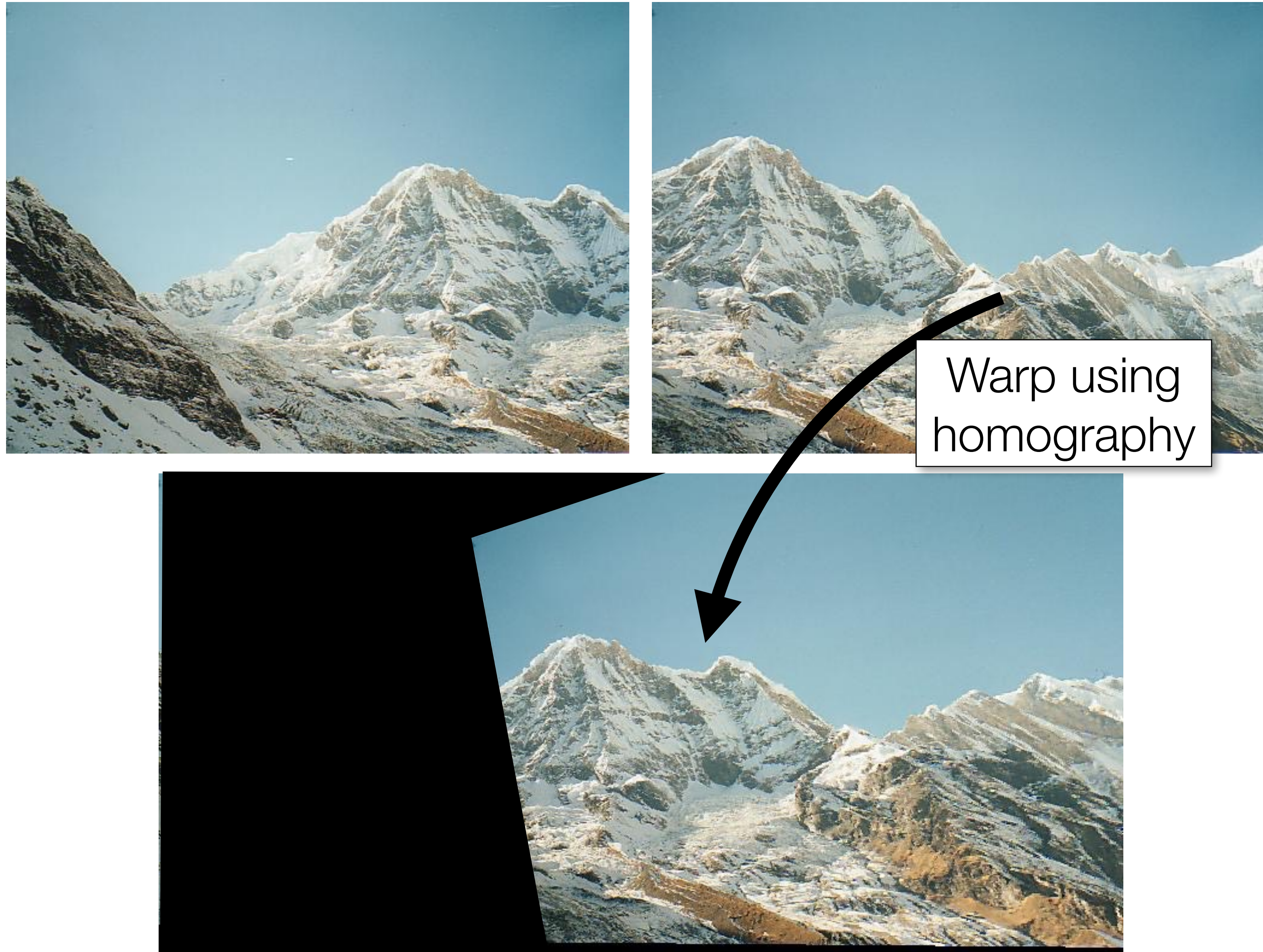


# Panorama stitching (PS9)



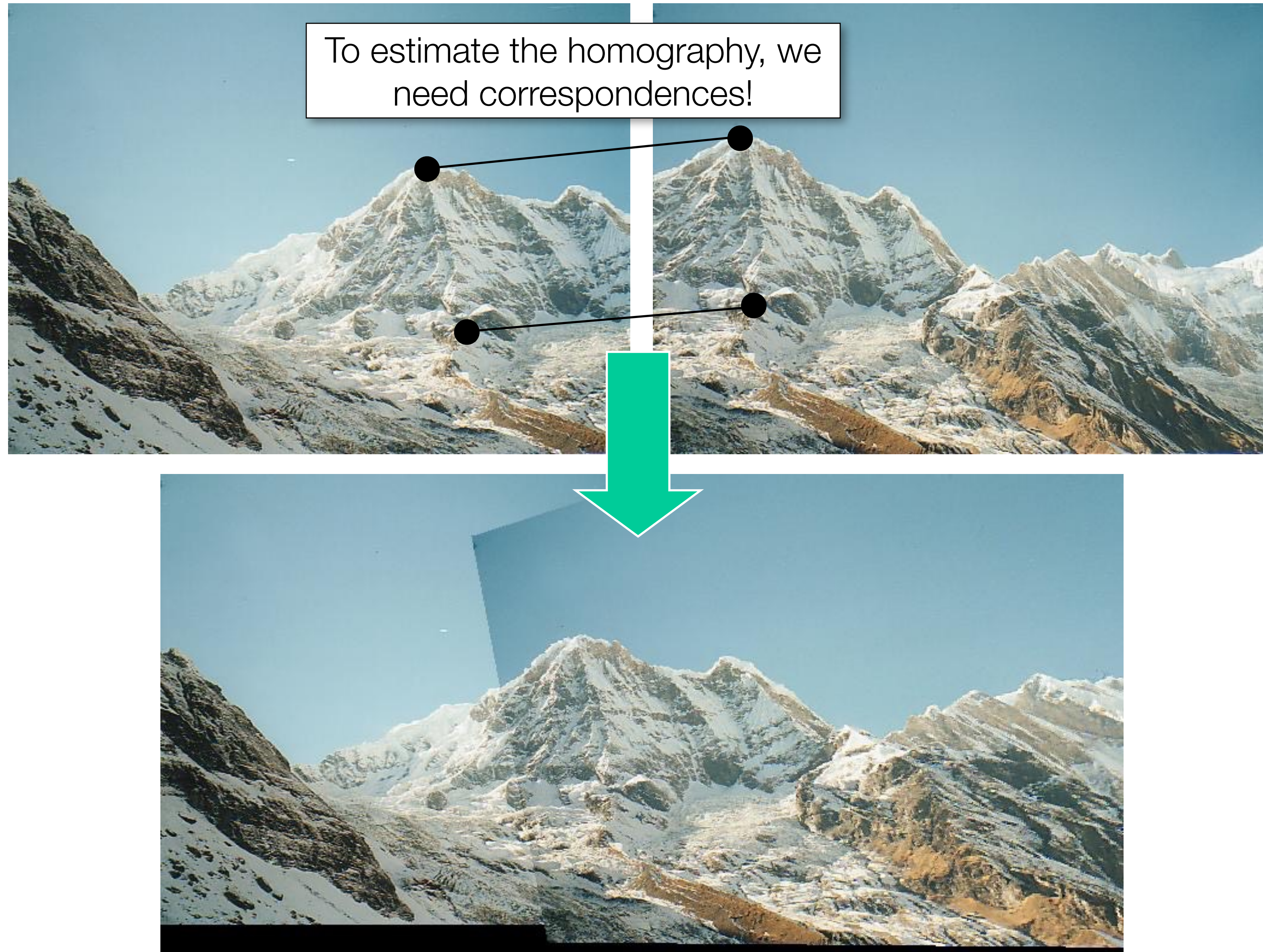


# Panorama stitching





# Panorama stitching



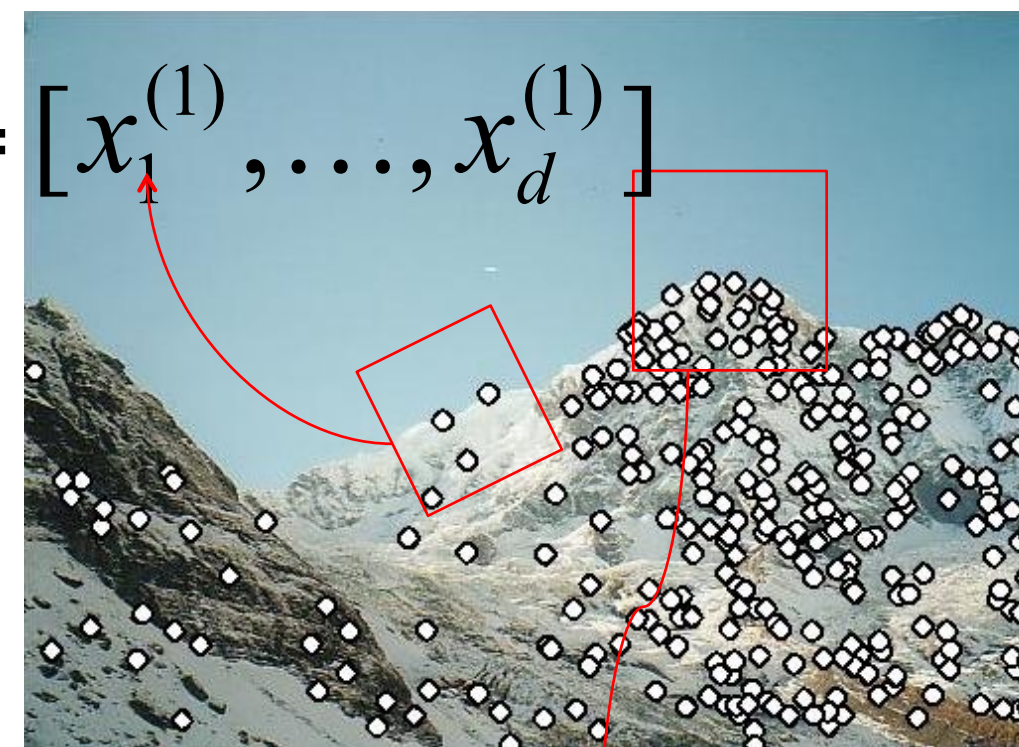


# Finding correspondences with local features

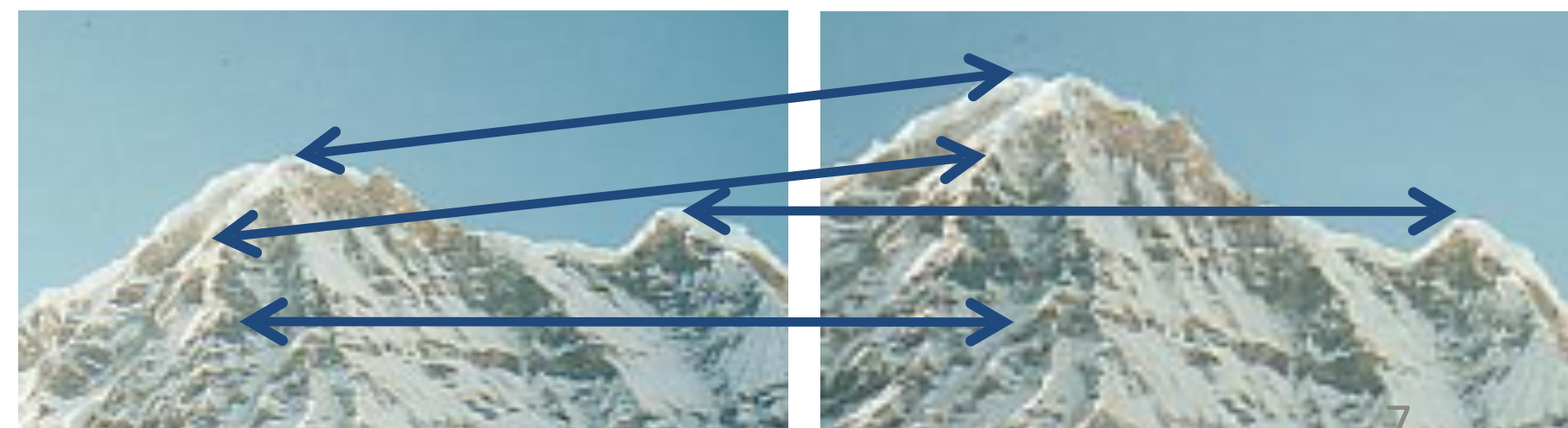
- 1) **Detection:** Identify the interest points, the candidate points to match
- 2) **Description:** Extract vector feature descriptor surrounding each interest point.
- 3) **Matching:** Determine correspondence between descriptors in two views



$$\mathbf{x}_1 = [x_1^{(1)}, \dots, x_d^{(1)}]$$

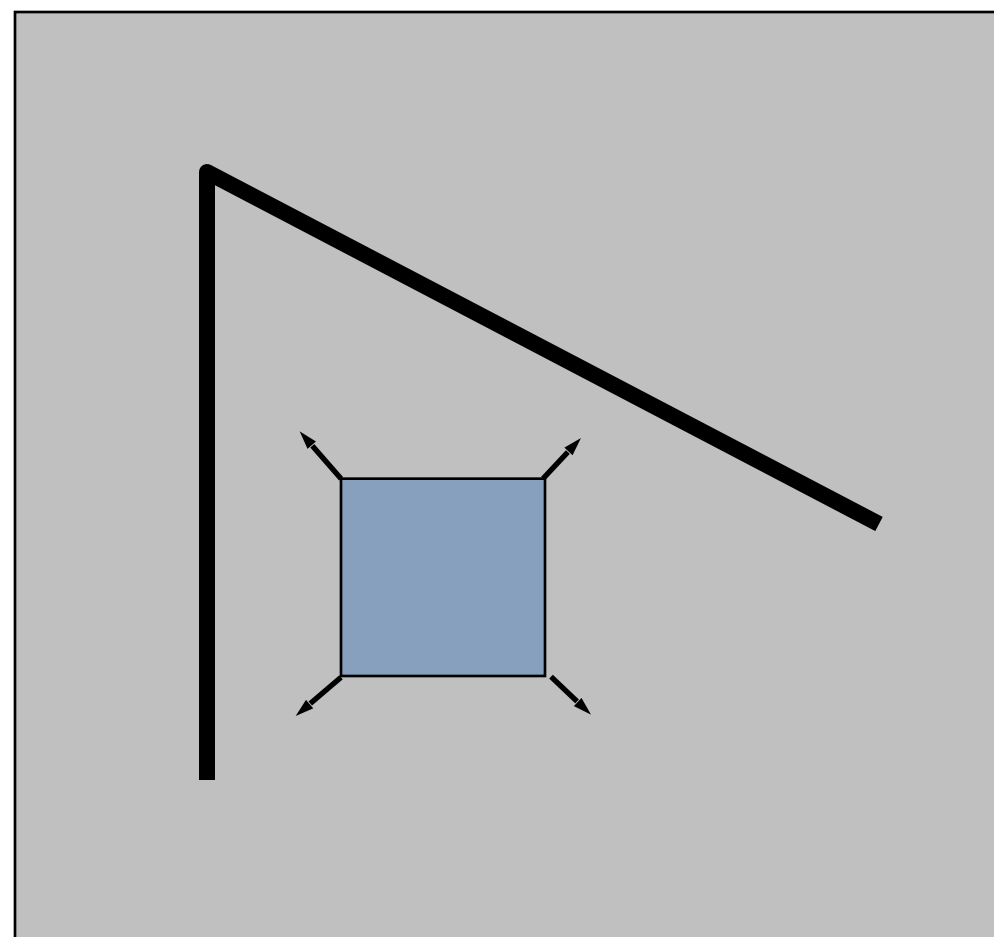


$$\mathbf{x}_2 = [x_1^{(2)}, \dots, x_d^{(2)}]$$

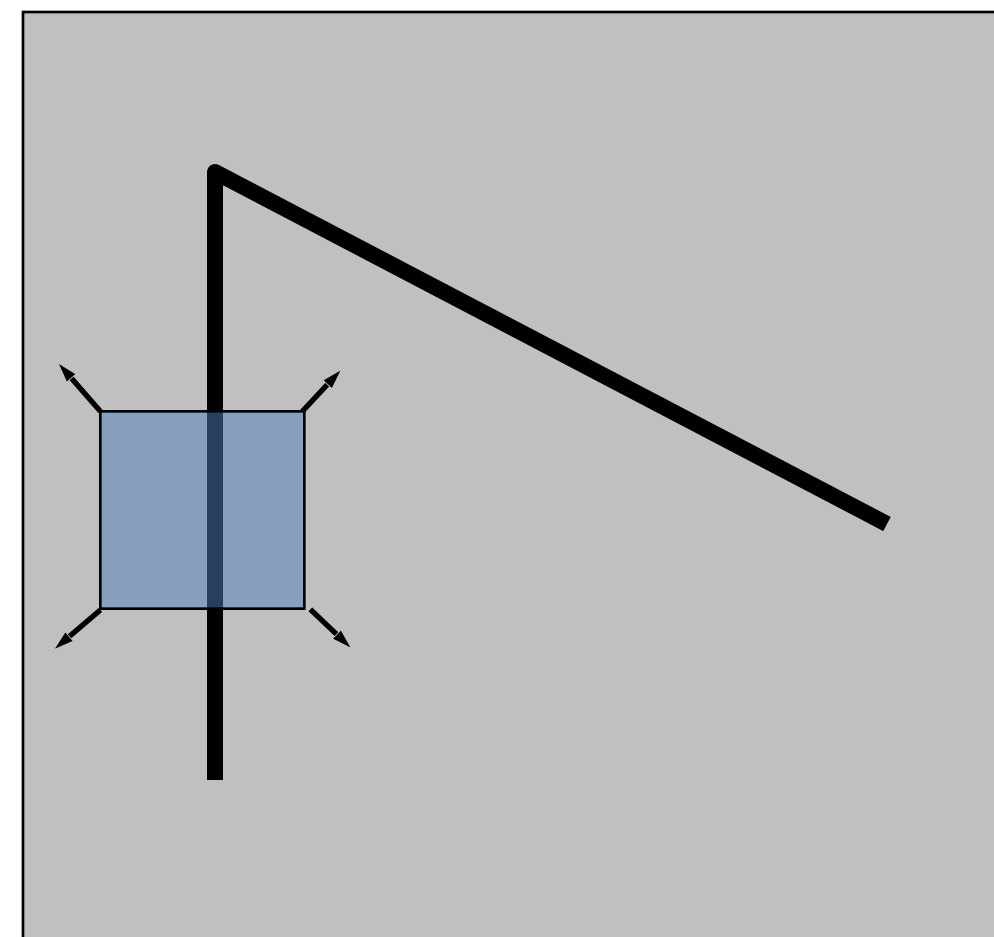


# What are good regions to match?

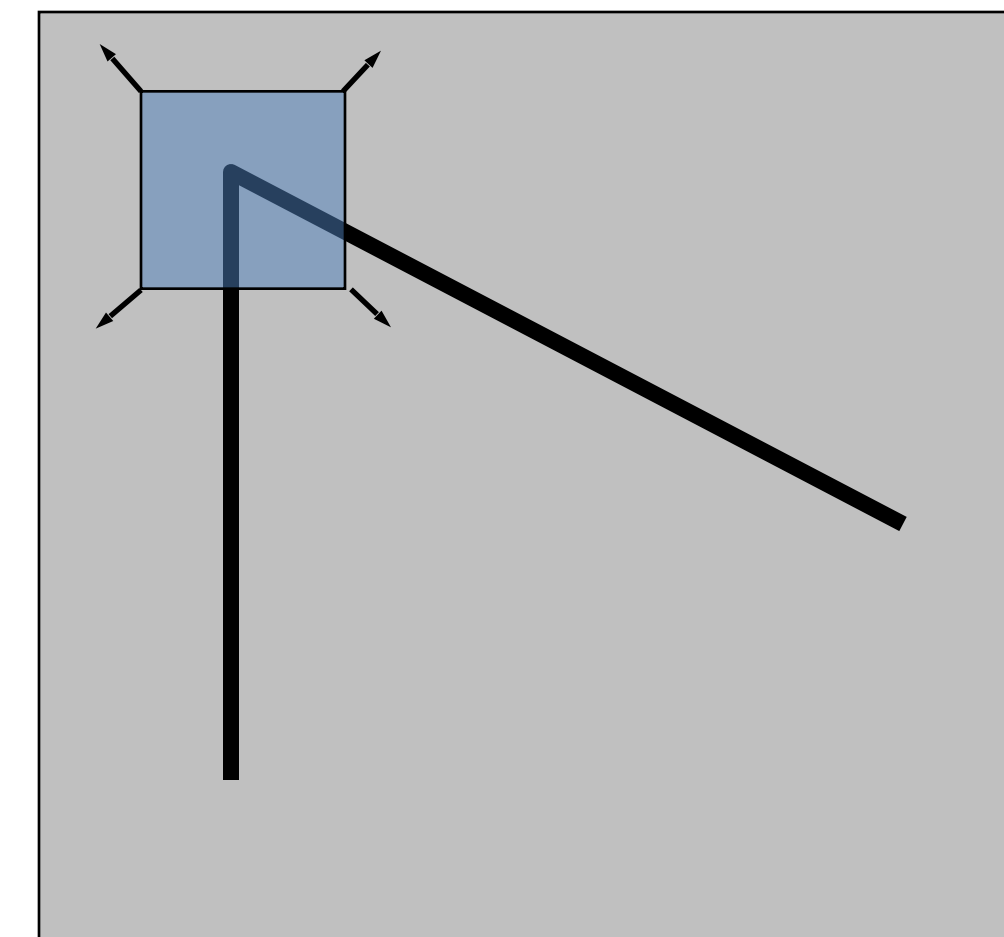
- How does the window change when you shift it?
- Shifting the window in any direction causes a big change



“flat” region:  
no change in all  
directions



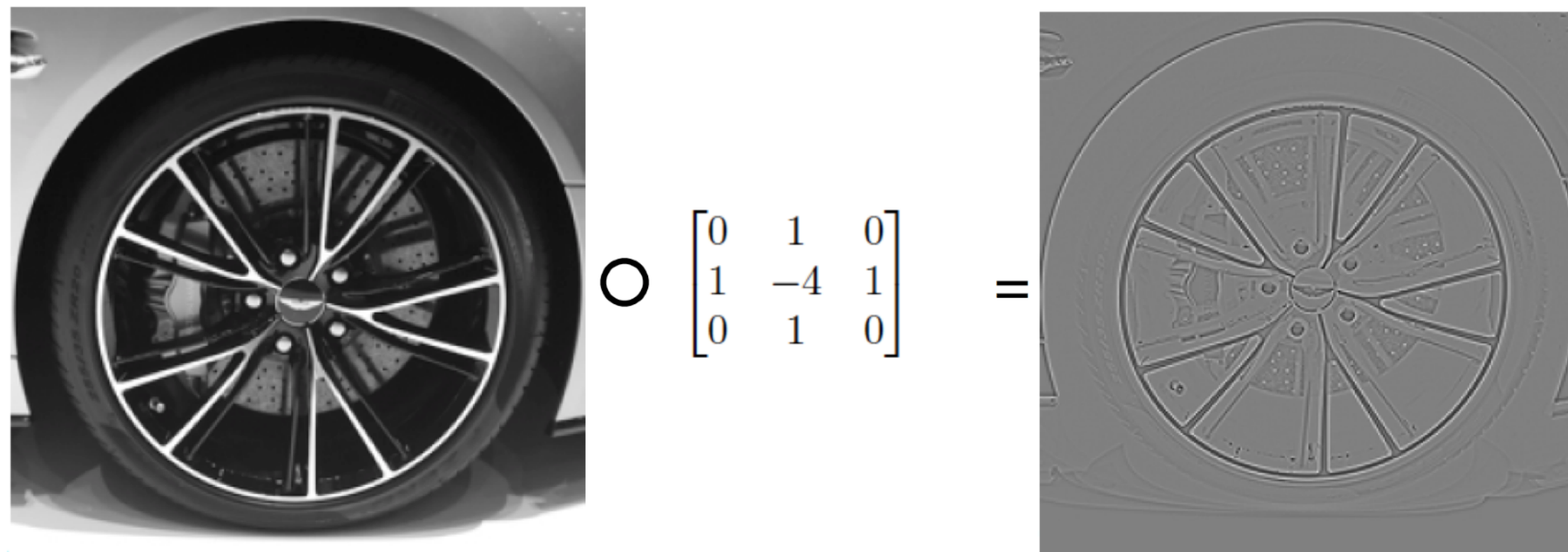
“edge”:  
no change along the  
edge direction



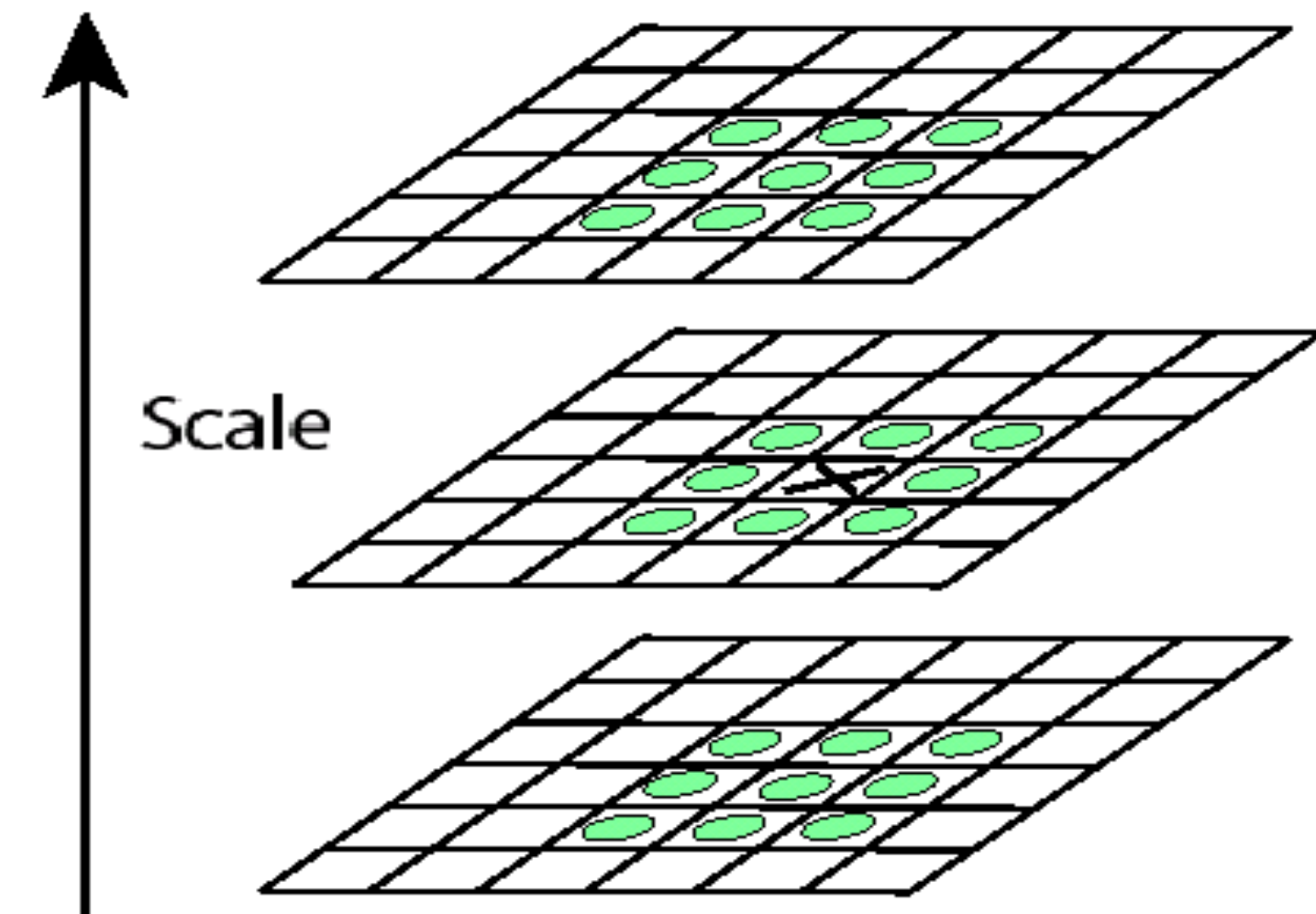
“corner”:  
significant change in  
all directions



# Finding good key points to match



Compute difference-of-Gaussians filter (approx. to Laplacian).



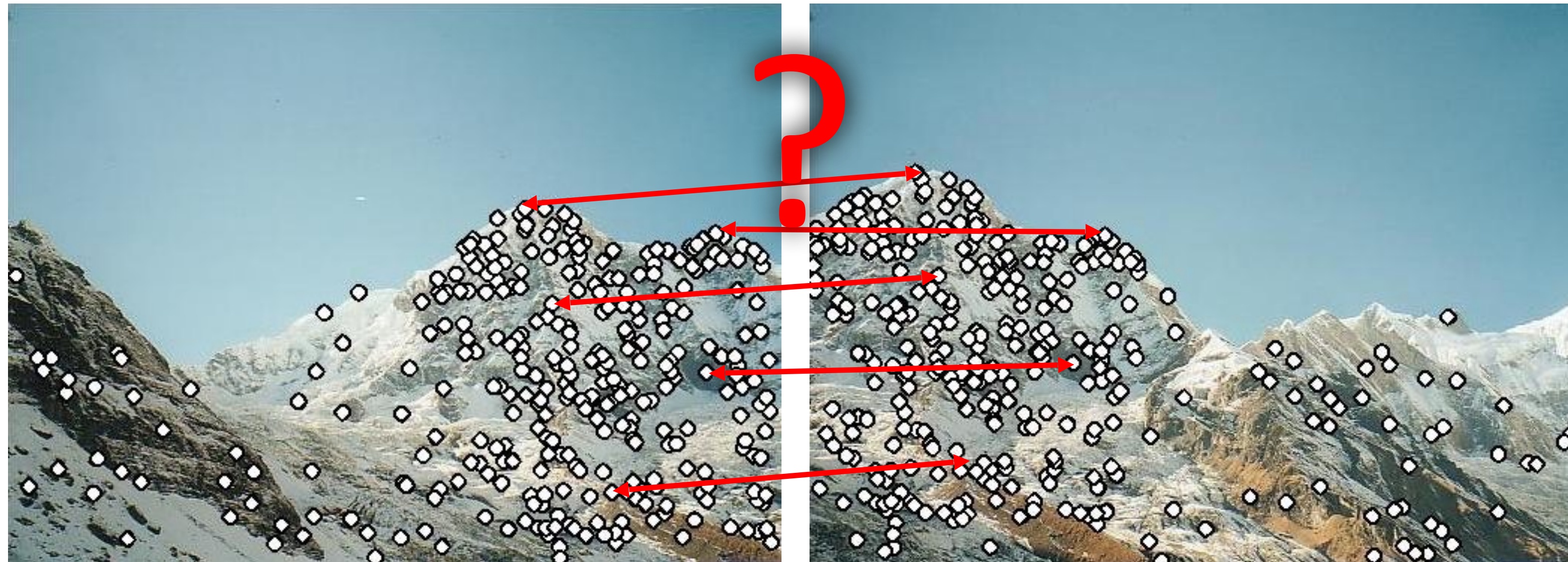
Find local optima in space and scale using Laplacian pyramid.



# Feature descriptors

We know how to detect good points

Next question: **How to match them?**



Come up with a *descriptor* (feature vector) for each point, find similar descriptors between the two images

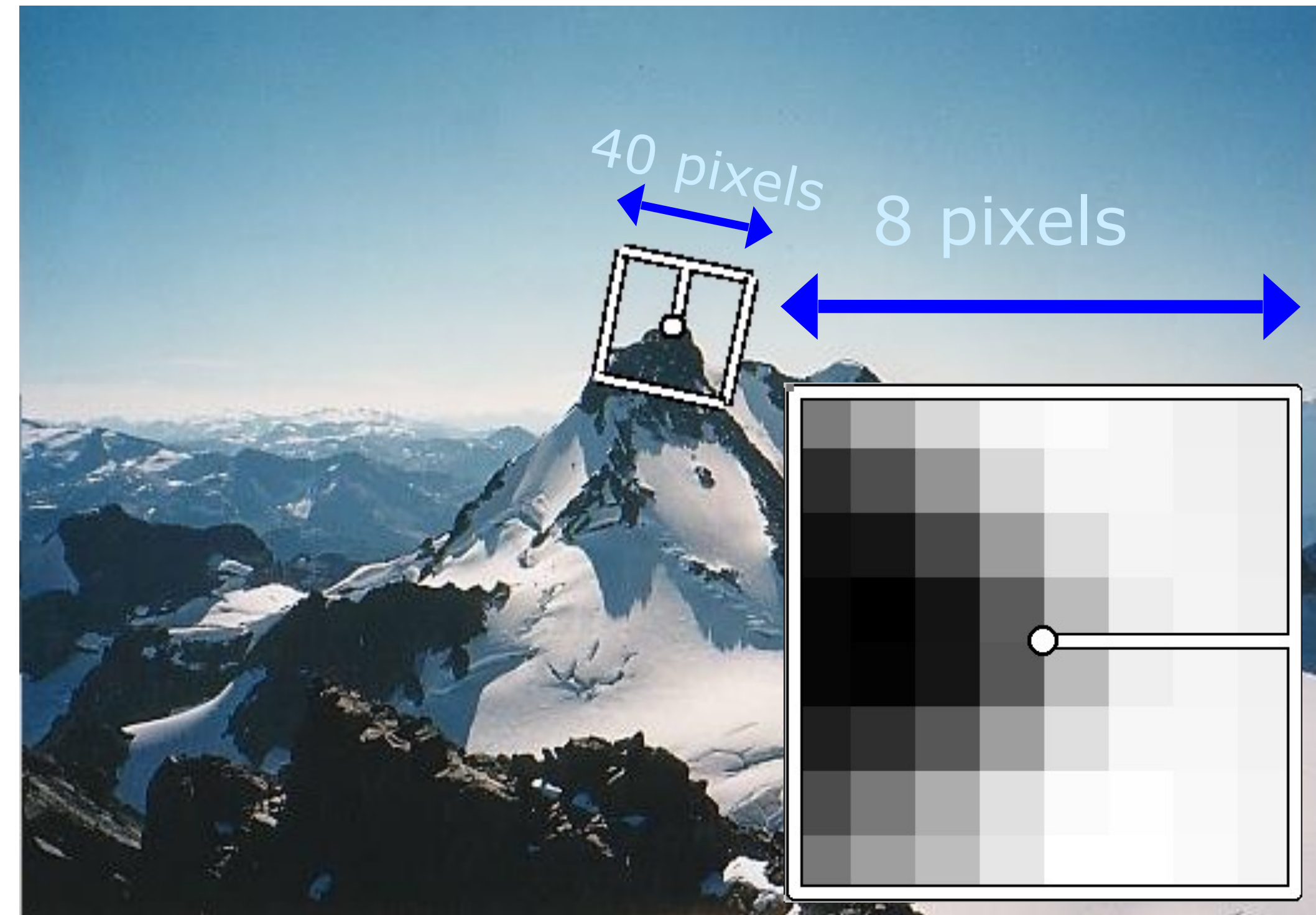


# Simple idea: normalized image patch

We want invariance to rotation, lighting, and tiny spatial shifts.

Take 40x40 window around feature

- Find dominant orientation
- Rotate to horizontal
- Downsample to 8x8
- Intensity normalize the window by subtracting the mean, dividing by the standard deviation in the window

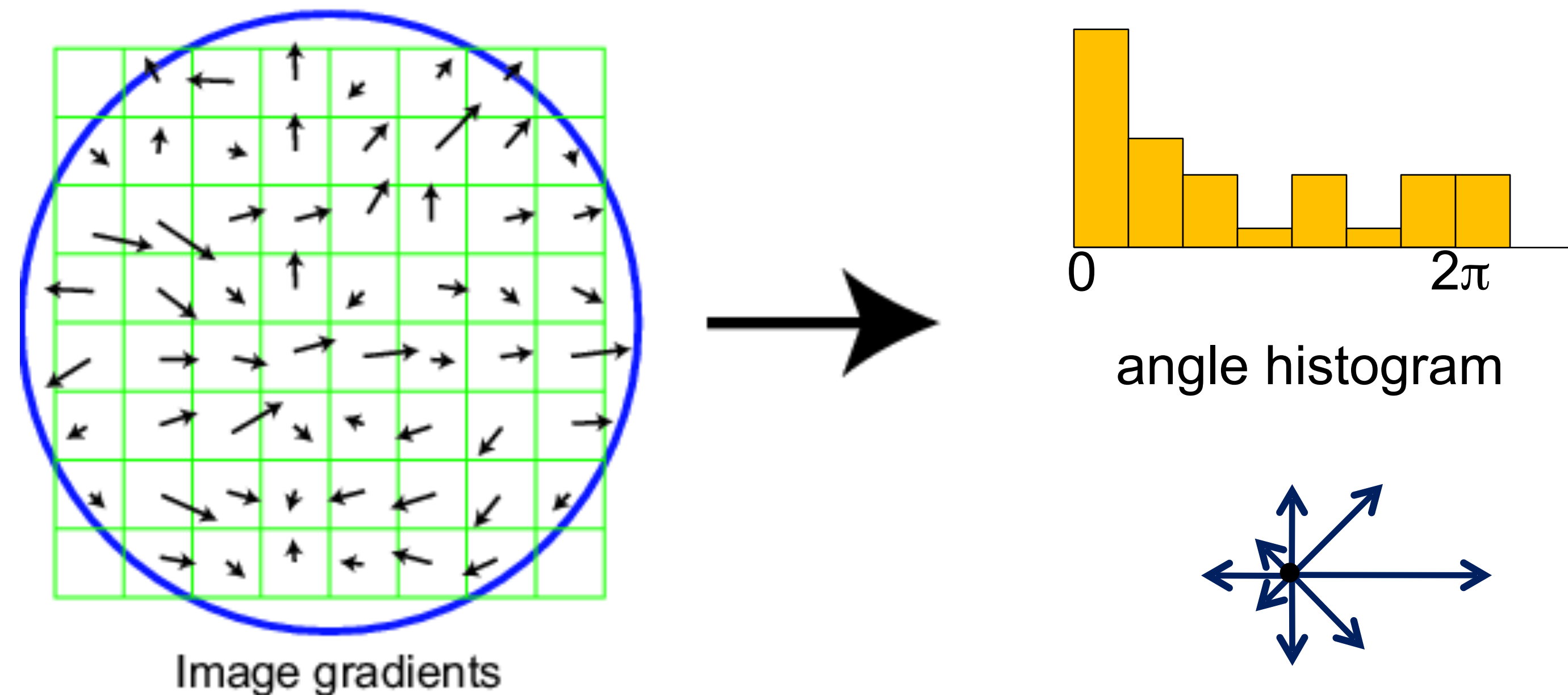




# Scale Invariant Feature Transform (SIFT)

Basic idea: looks like a hand-crafted CNN

- Take 16x16 square window around detected feature
- Compute edge orientation for each pixel
- Create histogram of edge orientations

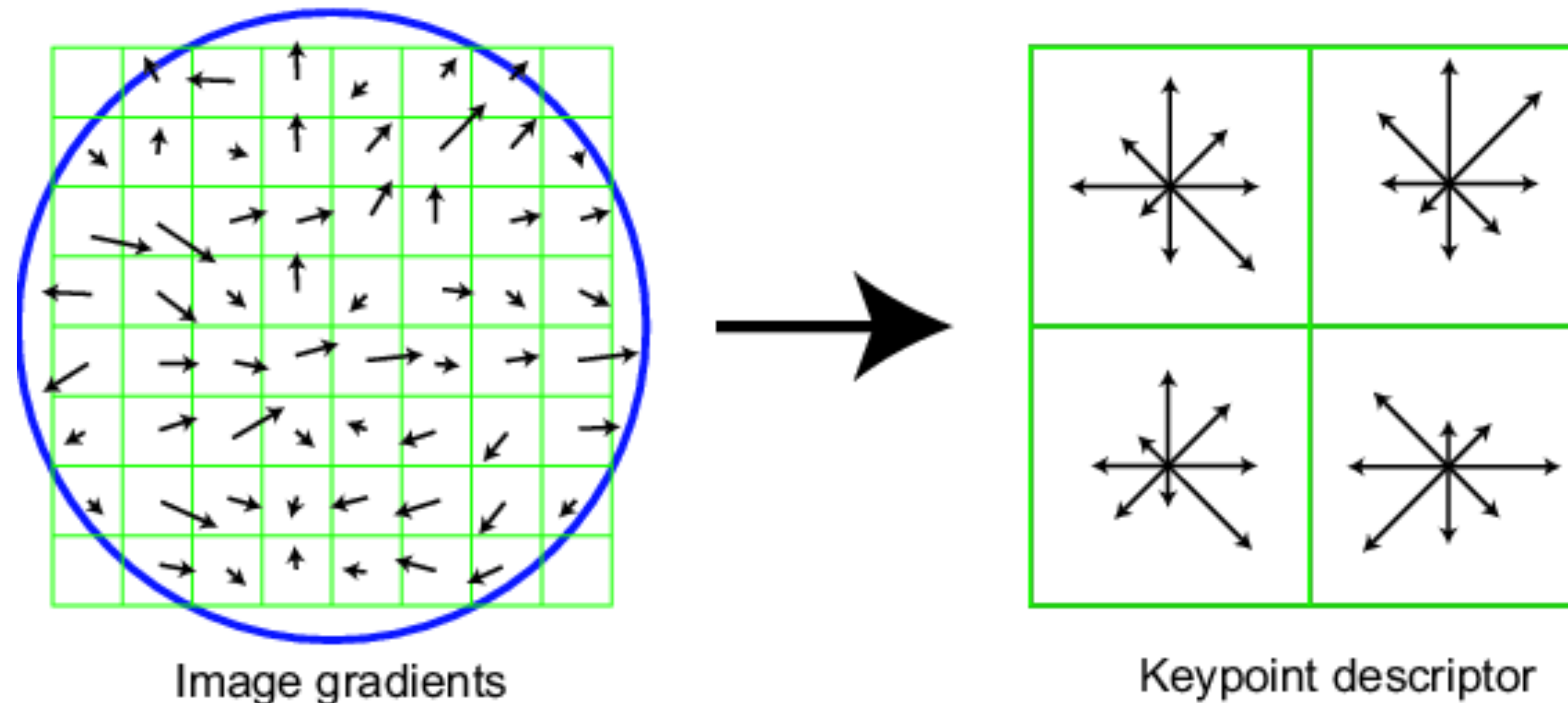




# Scale Invariant Feature Transform

Create the descriptor:

- Rotation invariance: rotate by “dominant” orientation
- Spatial invariance: spatial pool to 2x2
- Compute an orientation histogram for each cell
- $(4 \times 4)$  cells  $\times$  8 orientations = 128 dimensional descriptor





# SIFT invariances



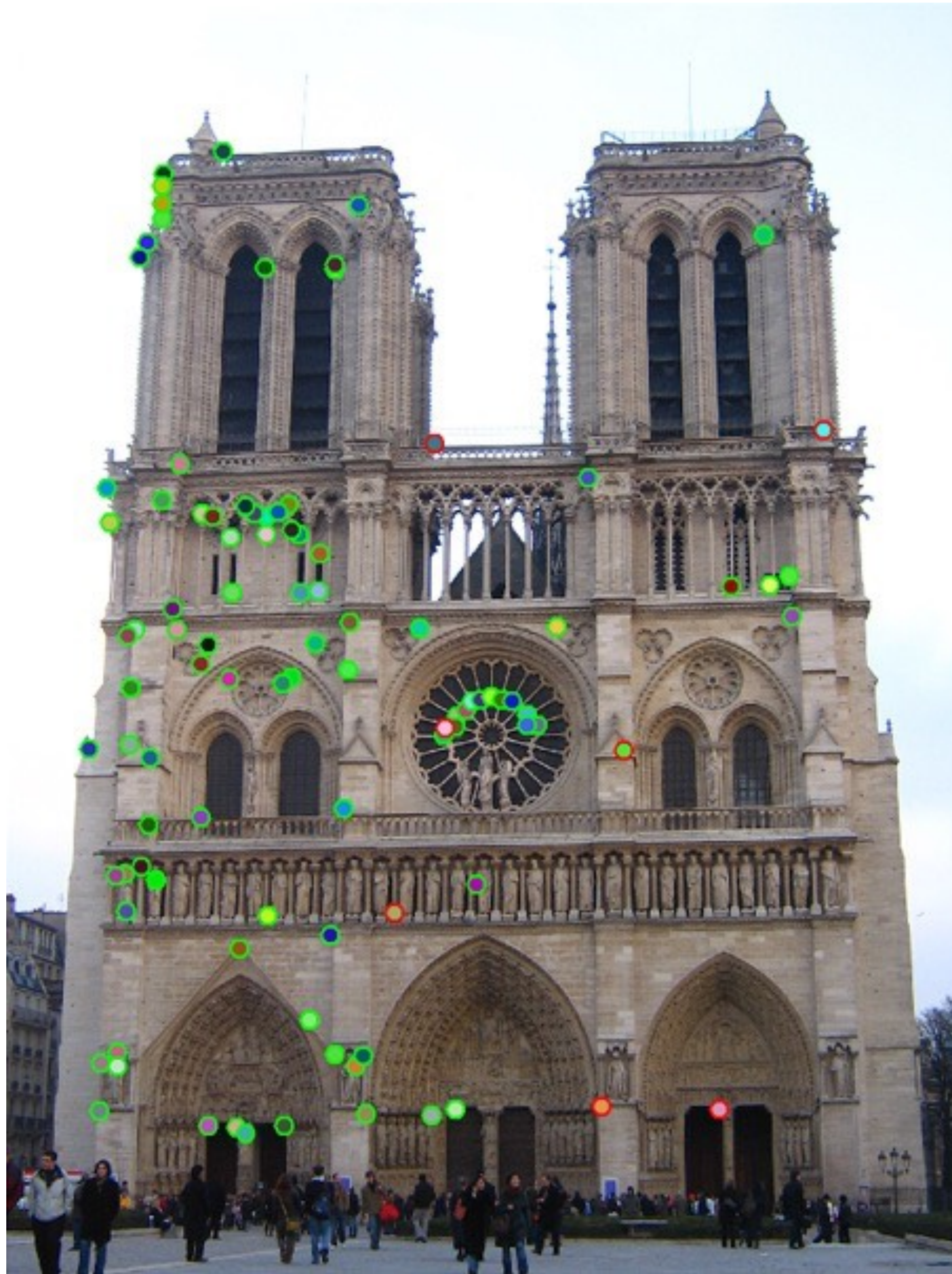


# Today

- Finding correspondences
  - Computing local features
  - **Matching**
- Fitting a homography
- RANSAC



# How can we tell if two features match?



Source: N. Snavely



# Finding matches

How do we know if two features match?

- Simple approach: are they the nearest neighbor in  $L_2$  distance,  $\|f_1 - f_2\|$ ?



$I_1$



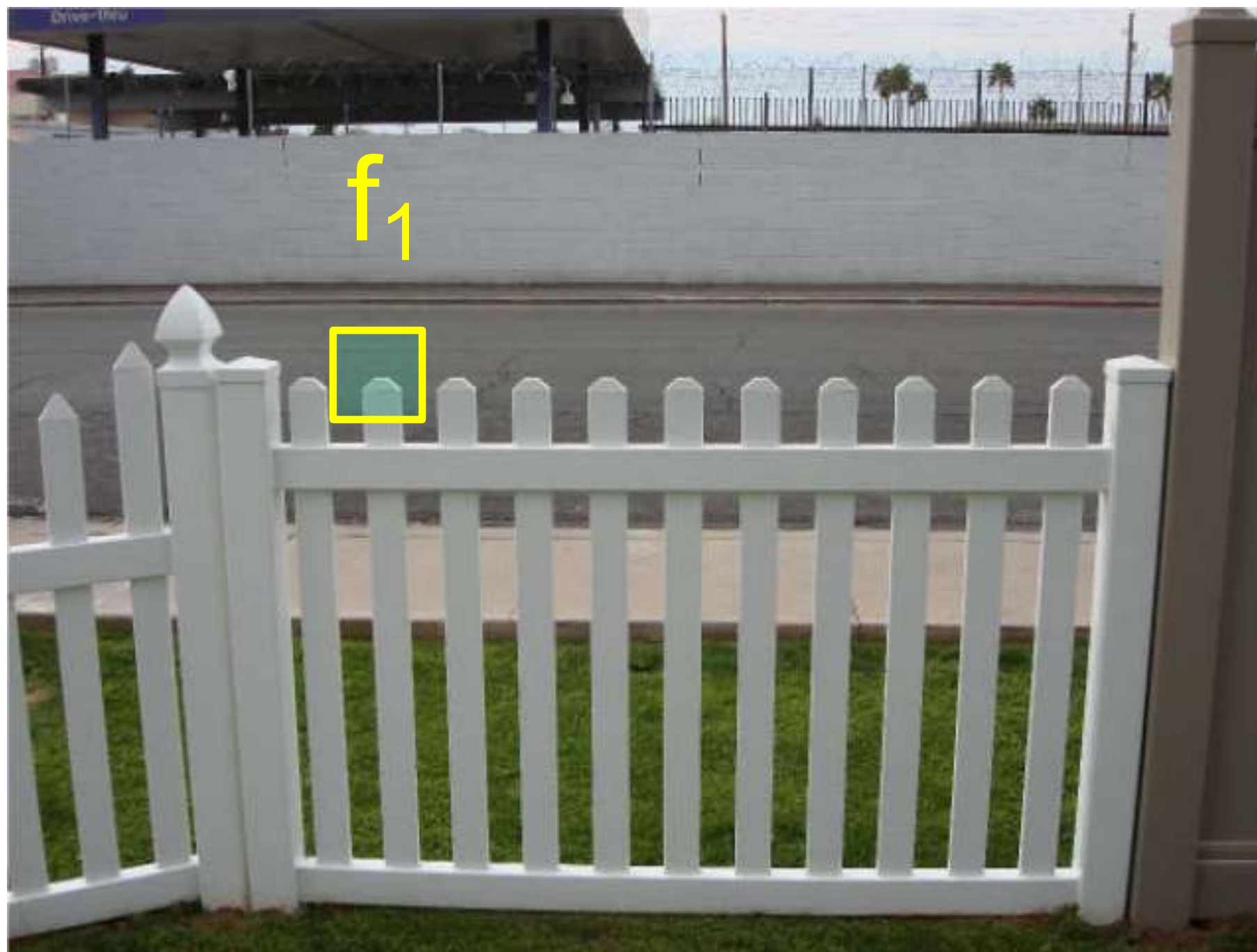
$I_2$



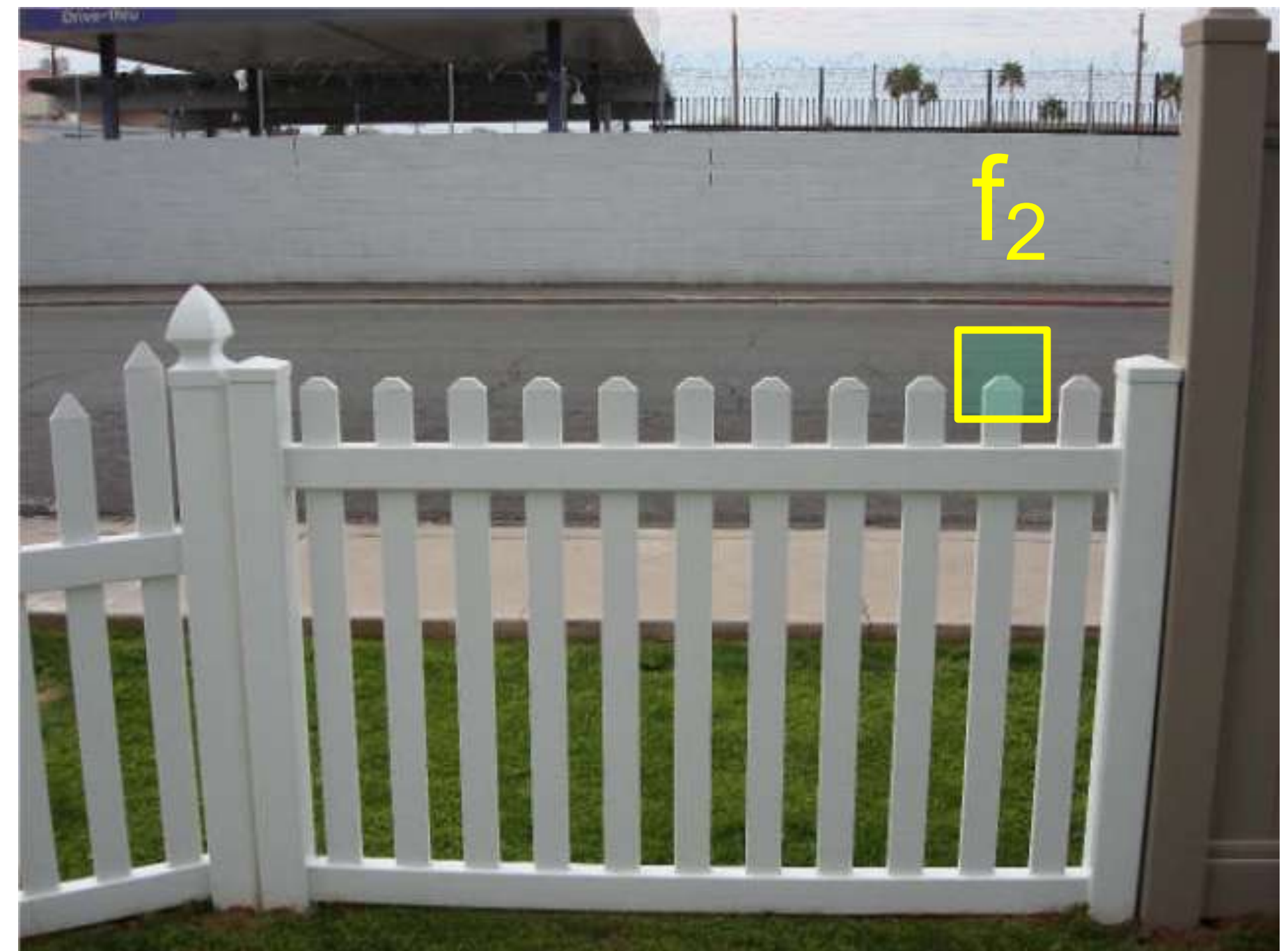
# Finding matches

How do we know if two features match?

- Simple approach: are they the nearest neighbor in  $L_2$  distance,  $\|f_1 - f_2\|$ ?
- Can give good scores to ambiguous (incorrect) matches



$I_1$

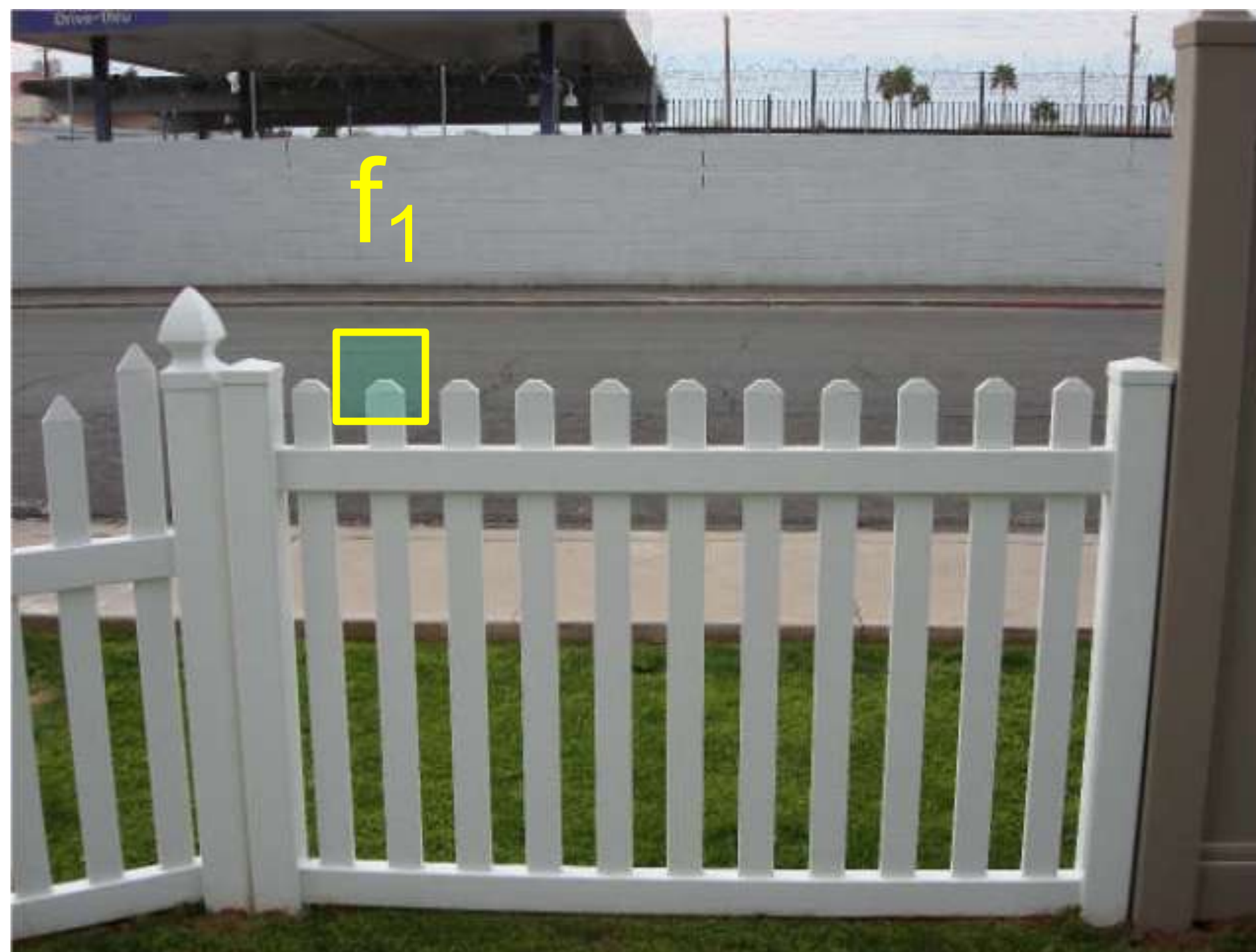


$I_2$

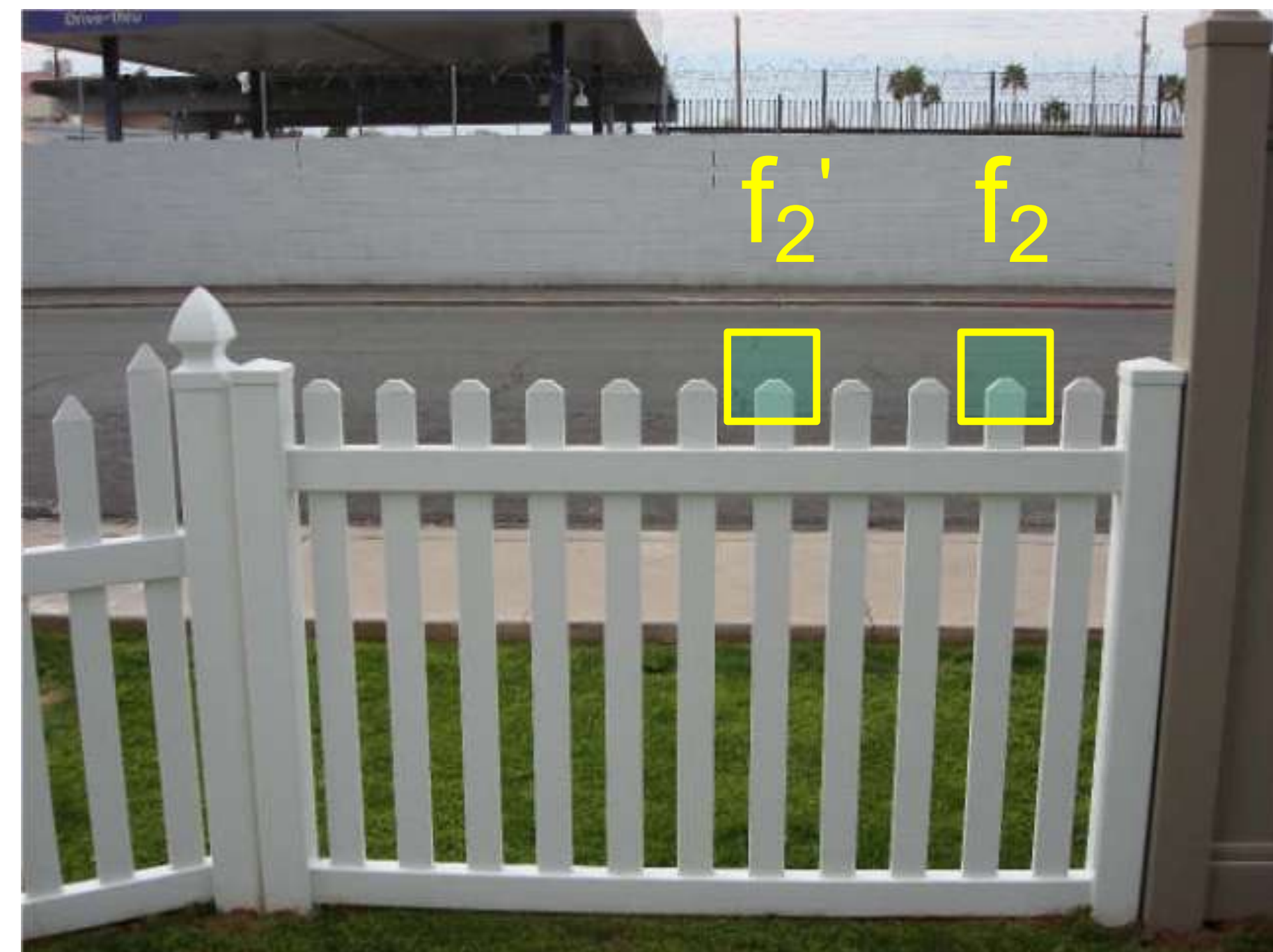
# Finding matches

Throw away matches that fail tests:

- **Ratio test:** this *by far* the best match?
  - Ratio distance =  $\|f_1 - f_2\| / \|f_1 - f_2'\|$
  - $f_2$  is best SSD match to  $f_1$  in  $I_2$
  - $f_2'$  is 2<sup>nd</sup> best SSD match to  $f_1$  in  $I_2$
- **Forward-backward consistency:**  $f_1$  should also be nearest neighbor of  $f_2$



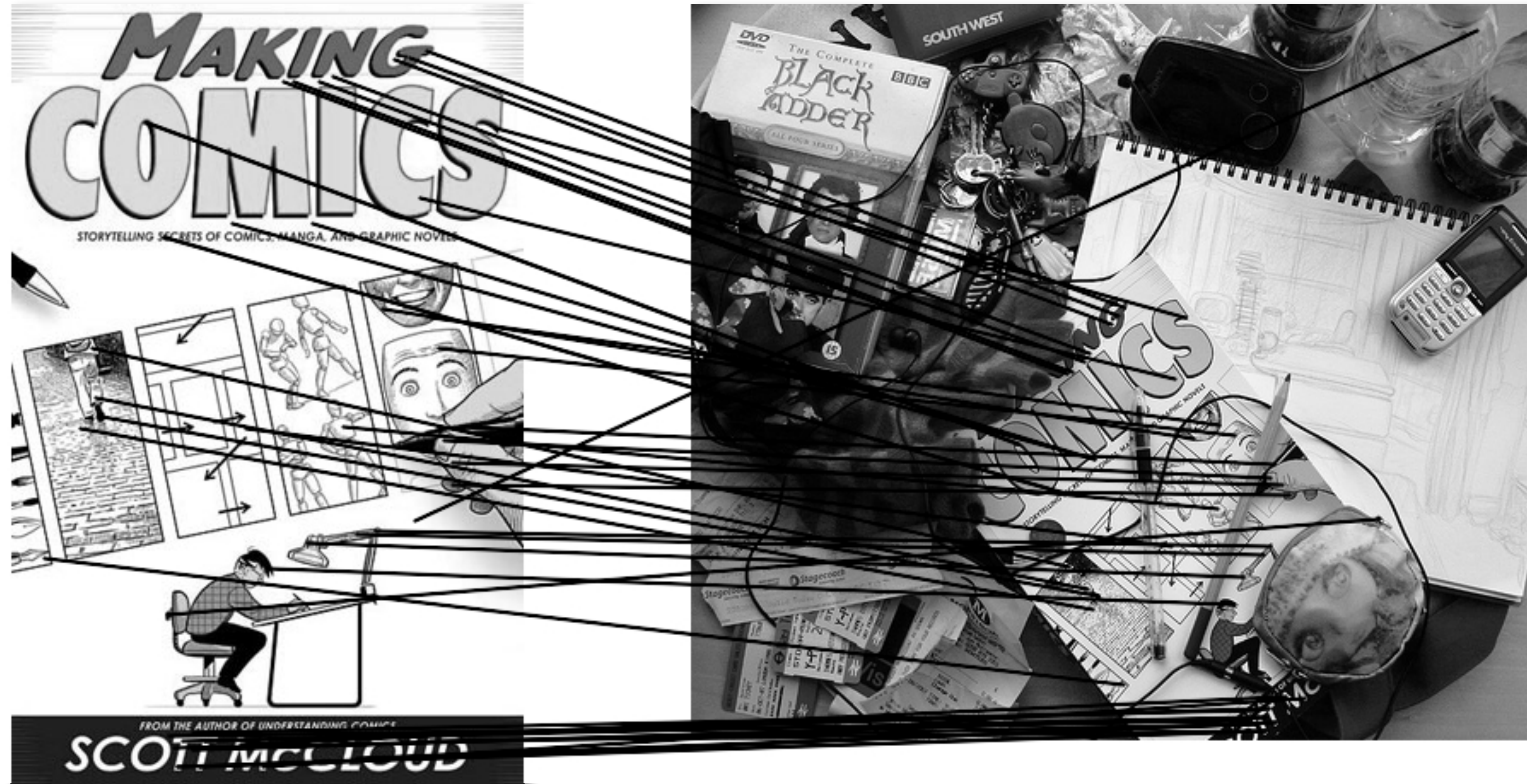
$I_1$



$I_2$



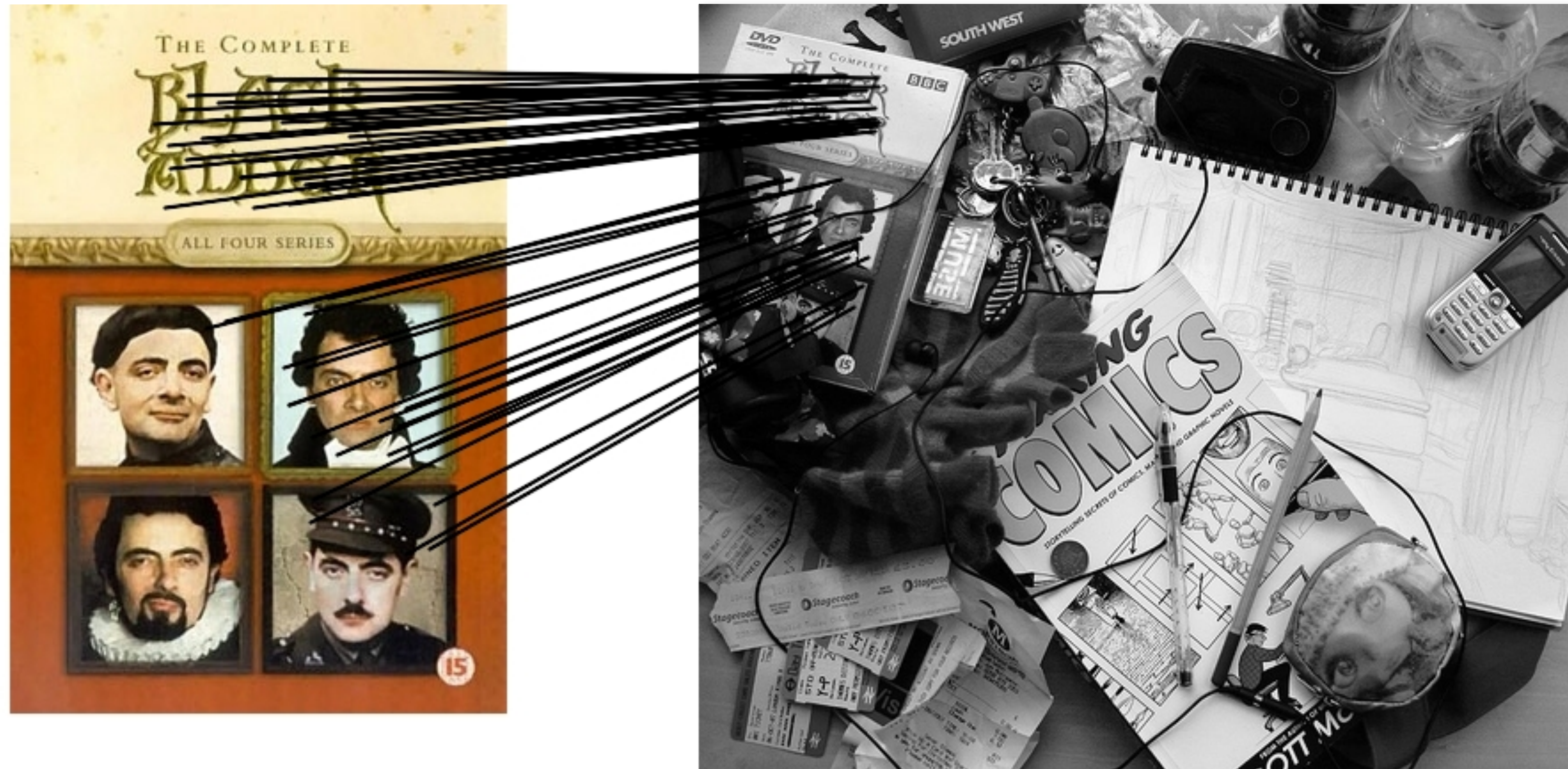
# Feature matching example



51 feature matches after ratio test



# Feature matching example



58 feature matches after ratio test

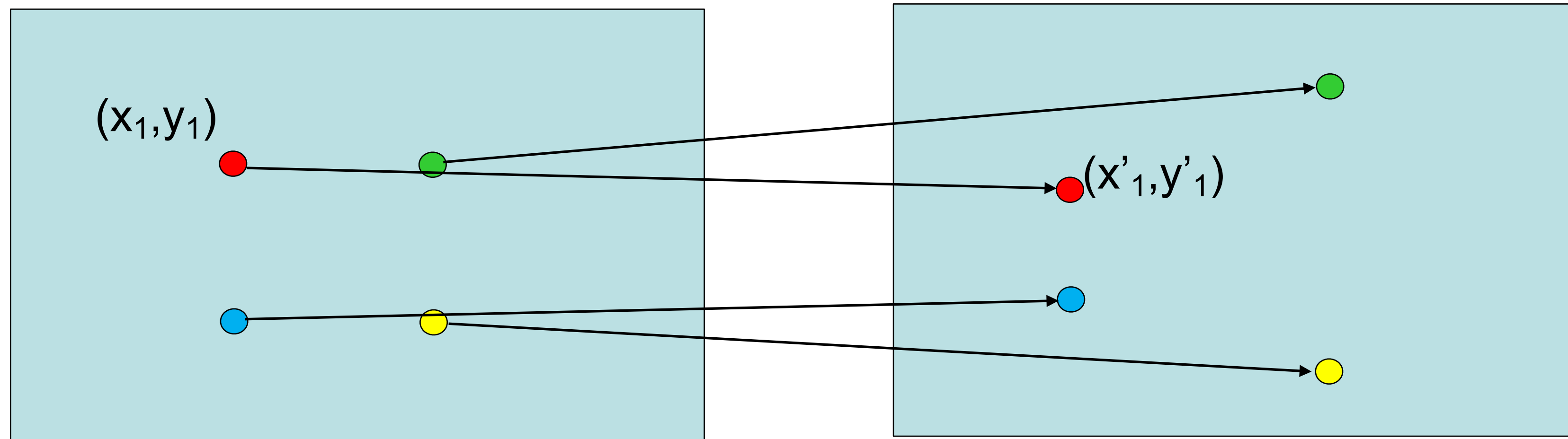


# Today

- Finding correspondences
  - Computing local features
  - Matching
- Fitting a homography
- RANSAC



# From matches to a homography



$$\begin{bmatrix} x'_1 \\ y'_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$



# From matches to a homography

Point in 1st image

Matched point in 2nd

$$\text{minimize } J(H) = \sum_i ||f_H(p_i) - p'_i||^2$$

where  $f_H(p_i) = Hp_i / (H_3^T p_i)$  applies homography  
(remember: homogenous coordinates)



# Option #1: Direct linear transform

$$\begin{bmatrix} x_1' \\ y_1' \\ w_1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Going to heterogeneous coordinates:

$$x_1' = \frac{ax_1 + by_1 + c}{gx_1 + hy_1 + i}$$

$$y_1' = \frac{dx_1 + ey_1 + f}{gx_1 + hy_1 + i}$$

Re-arranging the terms:

$$gx_1x_1' + hy_1x_1' + ix_1' = ax_1 + by_1 + c$$

$$gx_1y_1' + hy_1y_1' + ix_1' = dx_1 + ey_1 + f$$



# Option #1: Direct linear transform

$$gx_1x'_1 + hy_1x'_1 + ix'_1 = ax_1 + by_1 + c$$

$$gx_1y'_1 + hy_1y'_1 + iy'_1 = dx_1 + ey_1 + f$$

Re-arranging the terms:

$$gx_1x'_1 + hy_1x'_1 + ix'_1 - ax_1 - by_1 - c = 0$$

$$gx_1y'_1 + hy_1y'_1 + iy'_1 - dx_1 - ey_1 - f = 0$$

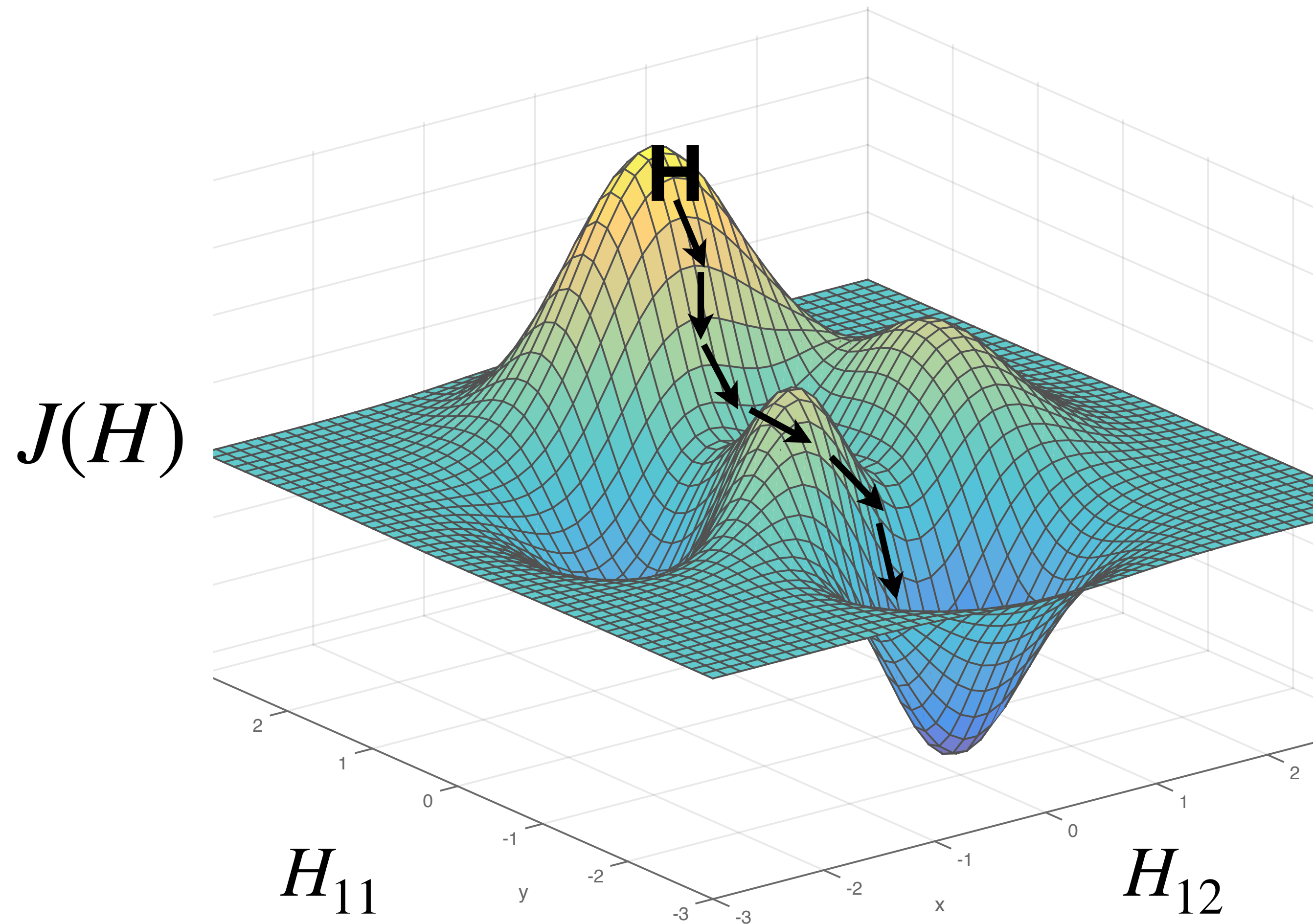
In matrix form. Can solve using Singular Value Decomposition (SVD).

$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x'_1 & y_1x'_1 & x'_1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y'_1 & y_1y'_1 & y'_1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Fast to solve (but not using “right” loss function). Uses an algebraic trick.  
Often used in practice for initial solutions!



# Option #2: Optimization



$$\text{minimize} \quad J(H) = \sum_i ||f_H(p_i) - p'_i||^2$$



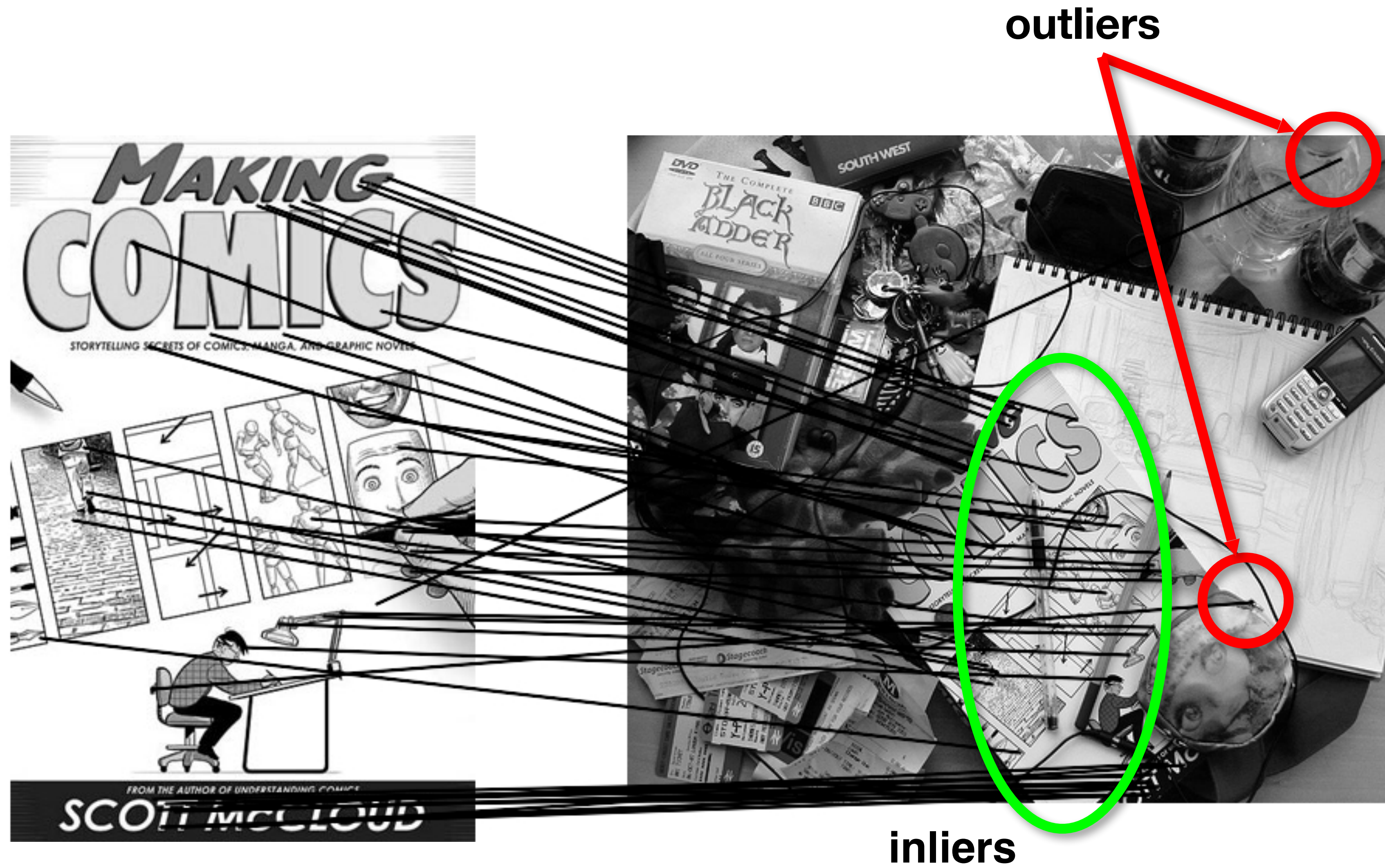
# Optimization

$$\text{minimize} \quad J(H) = \sum_i ||f_H(p_i) - p'_i||^2$$

- Can use gradient descent, just like when learning neural nets
- The problem is **smaller scale** than deep learning but has **more local optima**:
  - Use 2nd derivatives to improve optimization
  - Can use finite differences or autodiff
- Can use special-purpose **nonlinear least squares** methods.
  - Exploits structure in the problem for a sum-of-squares loss.



# Outliers





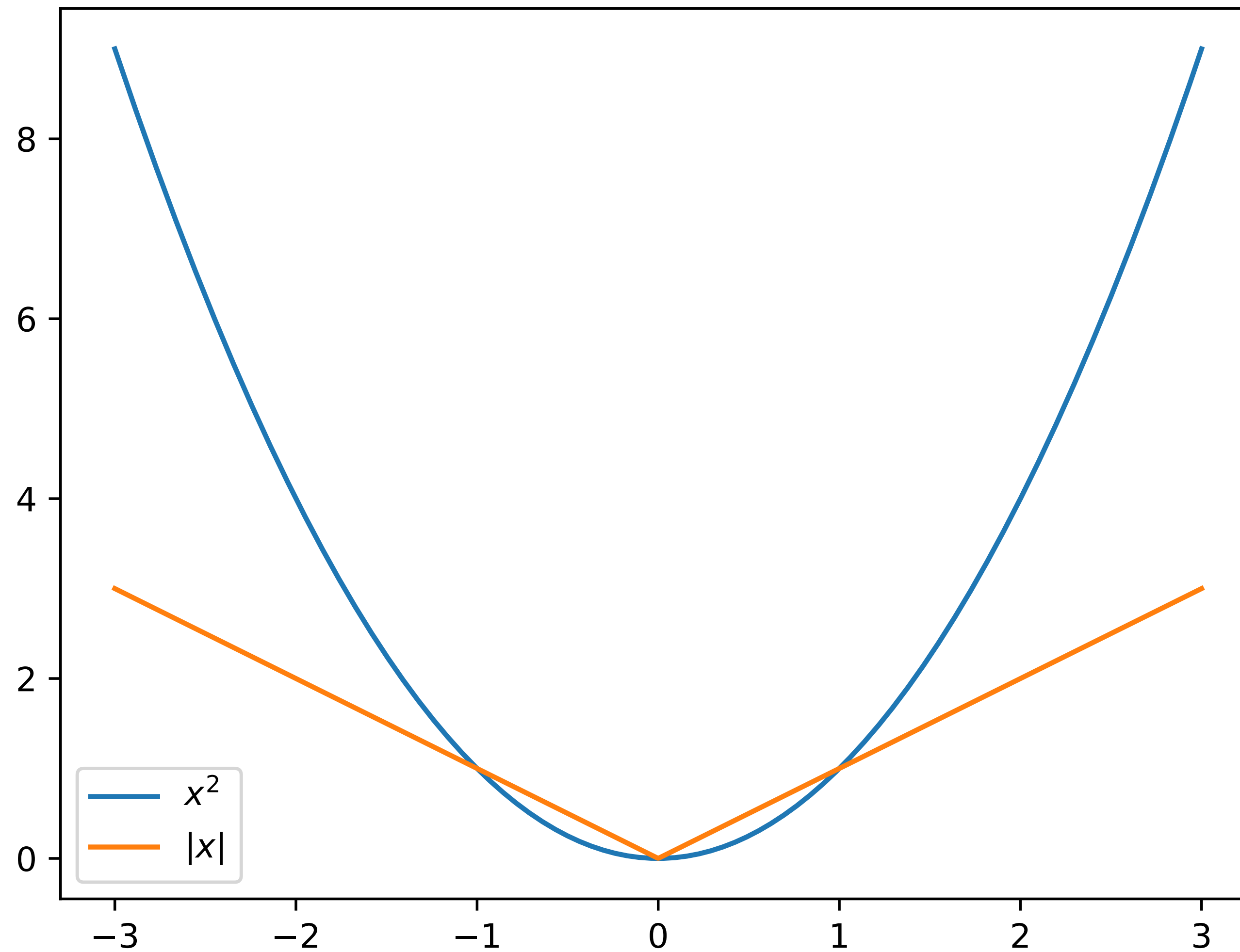
One idea: robust loss functions

$$\text{minimize } J(H) = \sum_{i=1}^N \sum_{j=1}^2 \rho(f_H(p_i)_j - p'_{ij})$$

where  $\rho(x)$  is a **robust** loss.

Special case:  $\rho(x) = x^2$  is L2 loss (same as before)

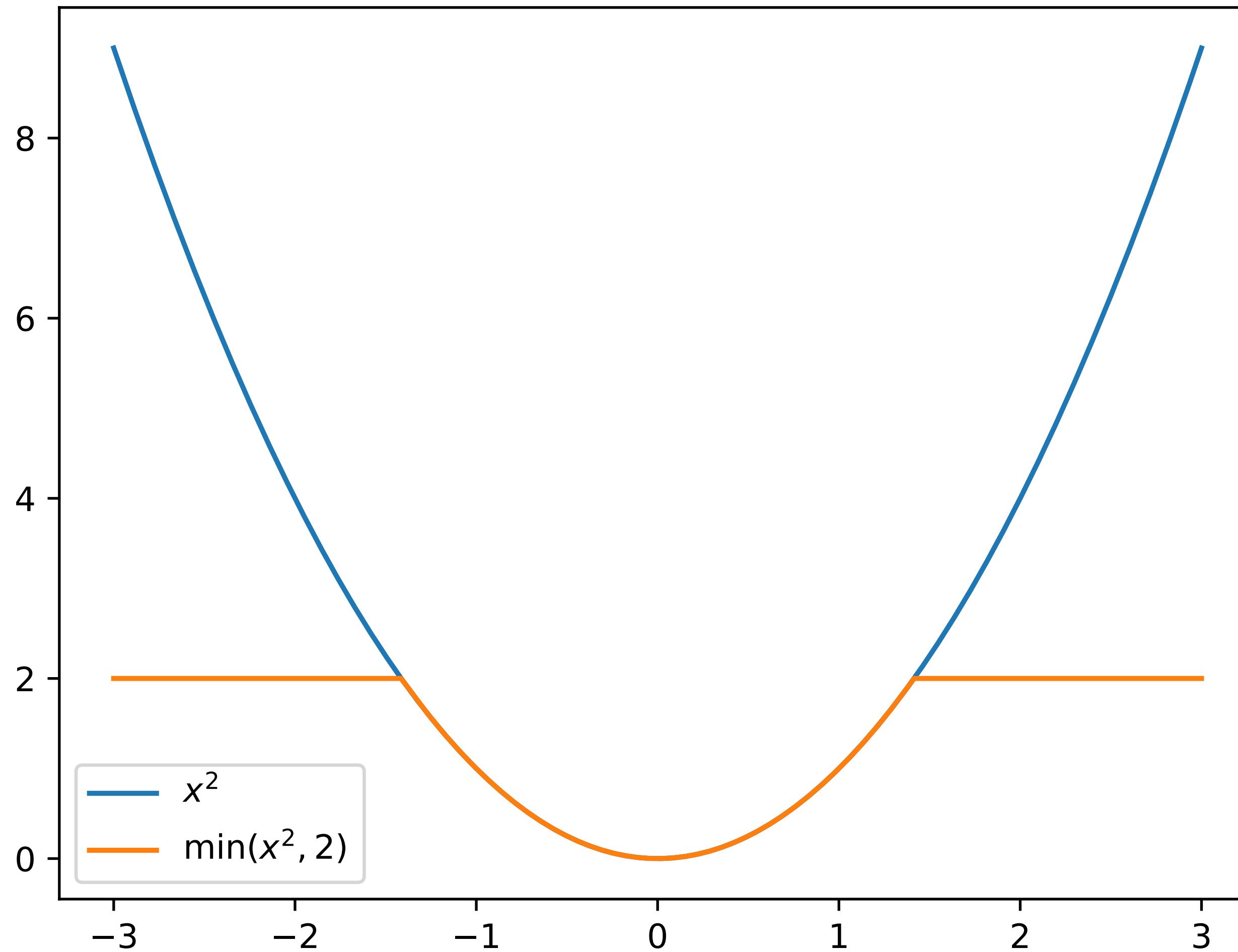
# Robust loss functions



L1 loss:  $\rho(x) = |x|$

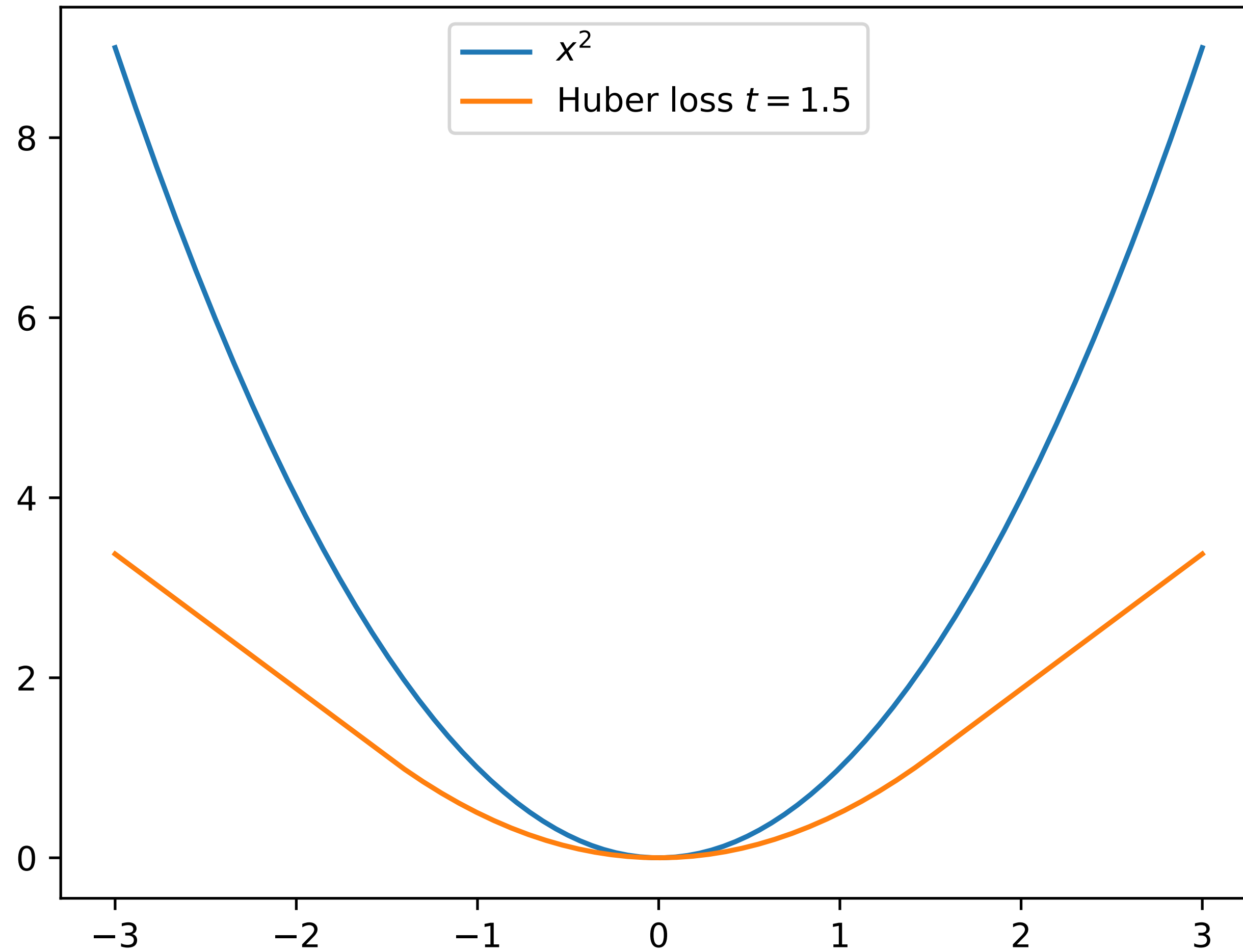


# Robust loss functions



Truncated quadratic:  $\rho(x) = \min(x^2, \tau)$

# Robust loss functions

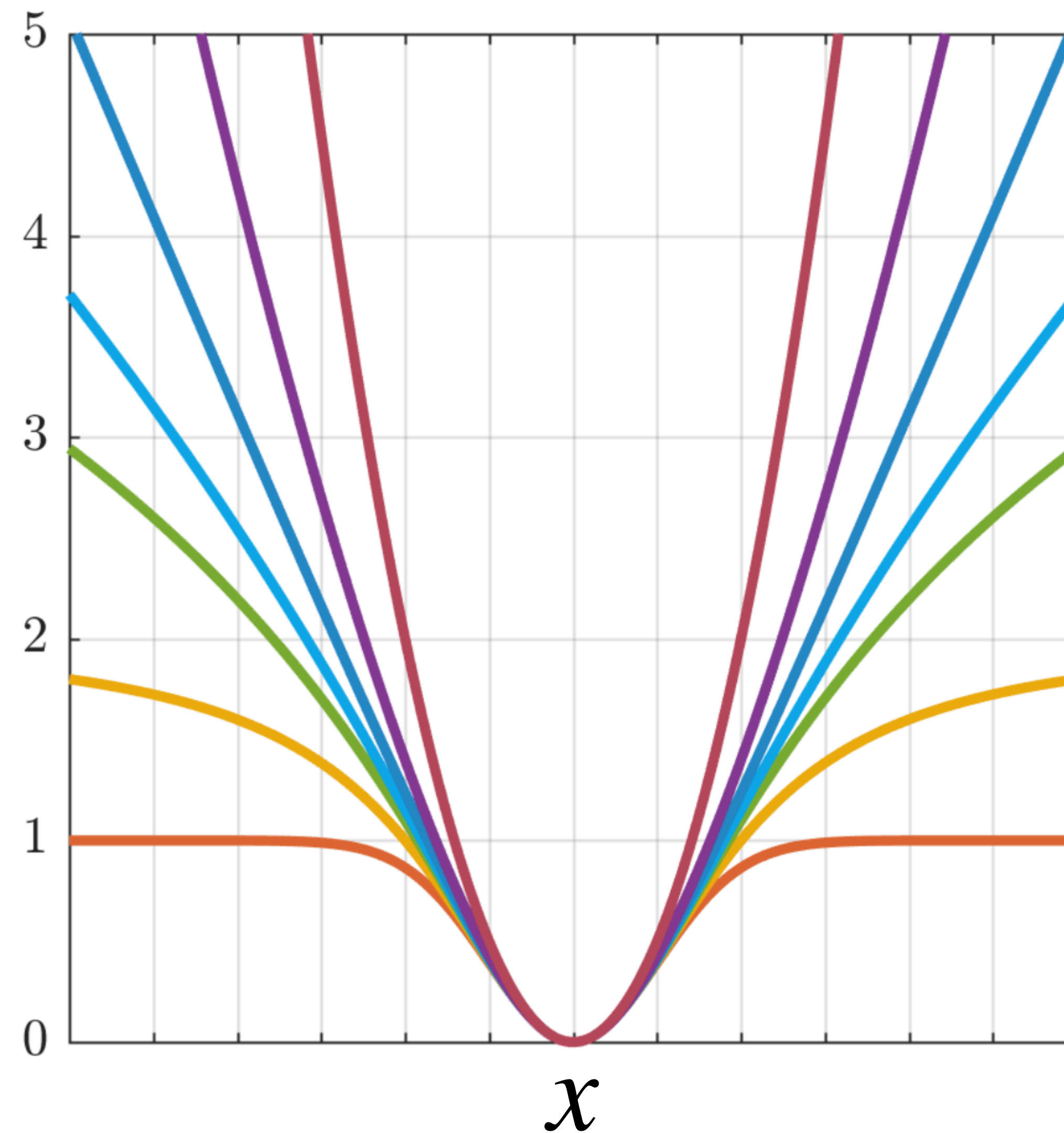


Huber loss:

$$\rho(x) = \begin{cases} \frac{1}{2}x^2 & \text{if } |x| \leq \tau, \\ \tau(|x| - \frac{1}{2}\tau), & \text{else} \end{cases}$$

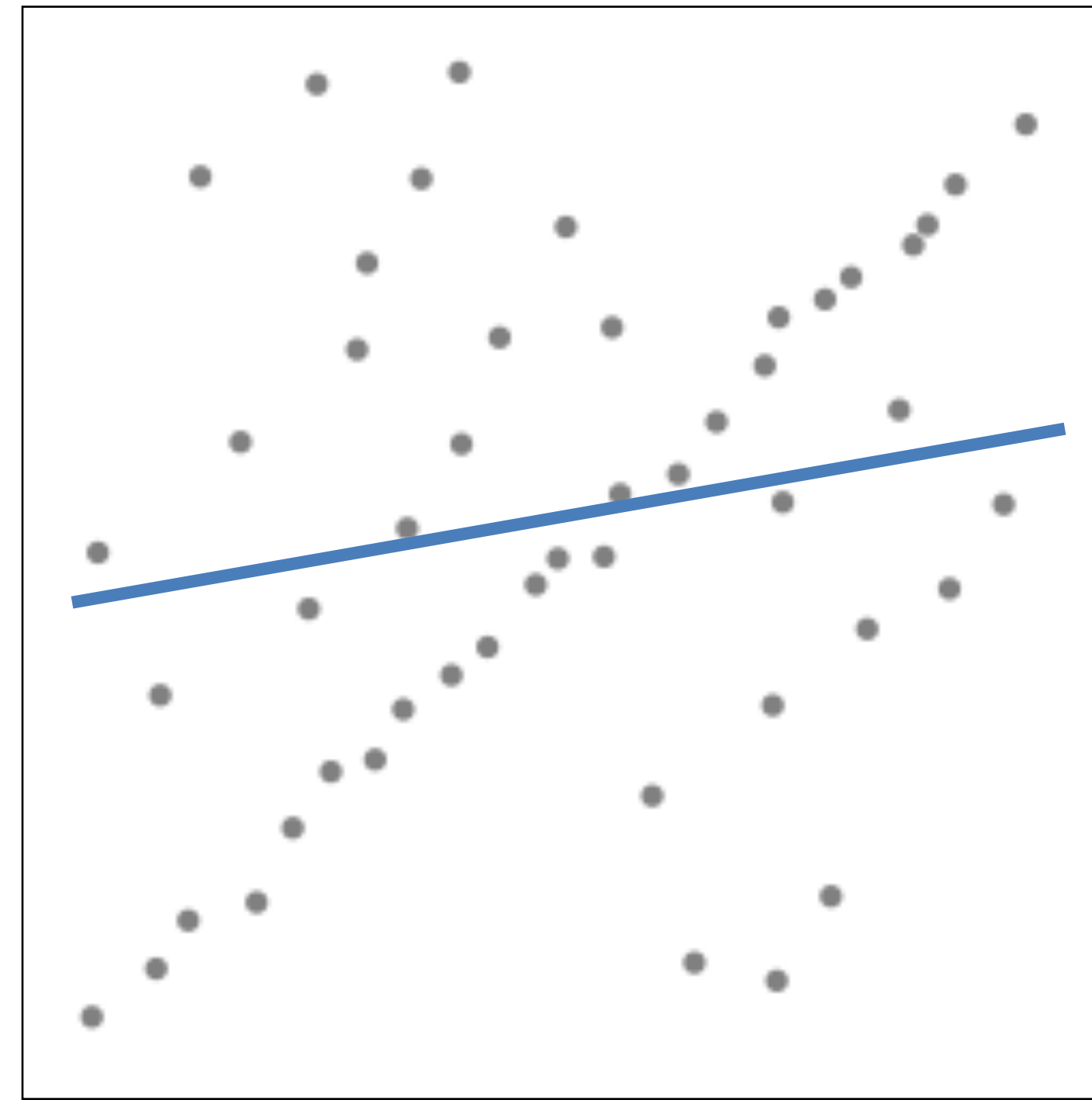
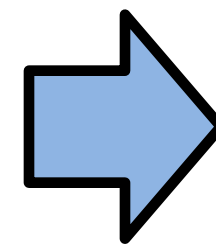
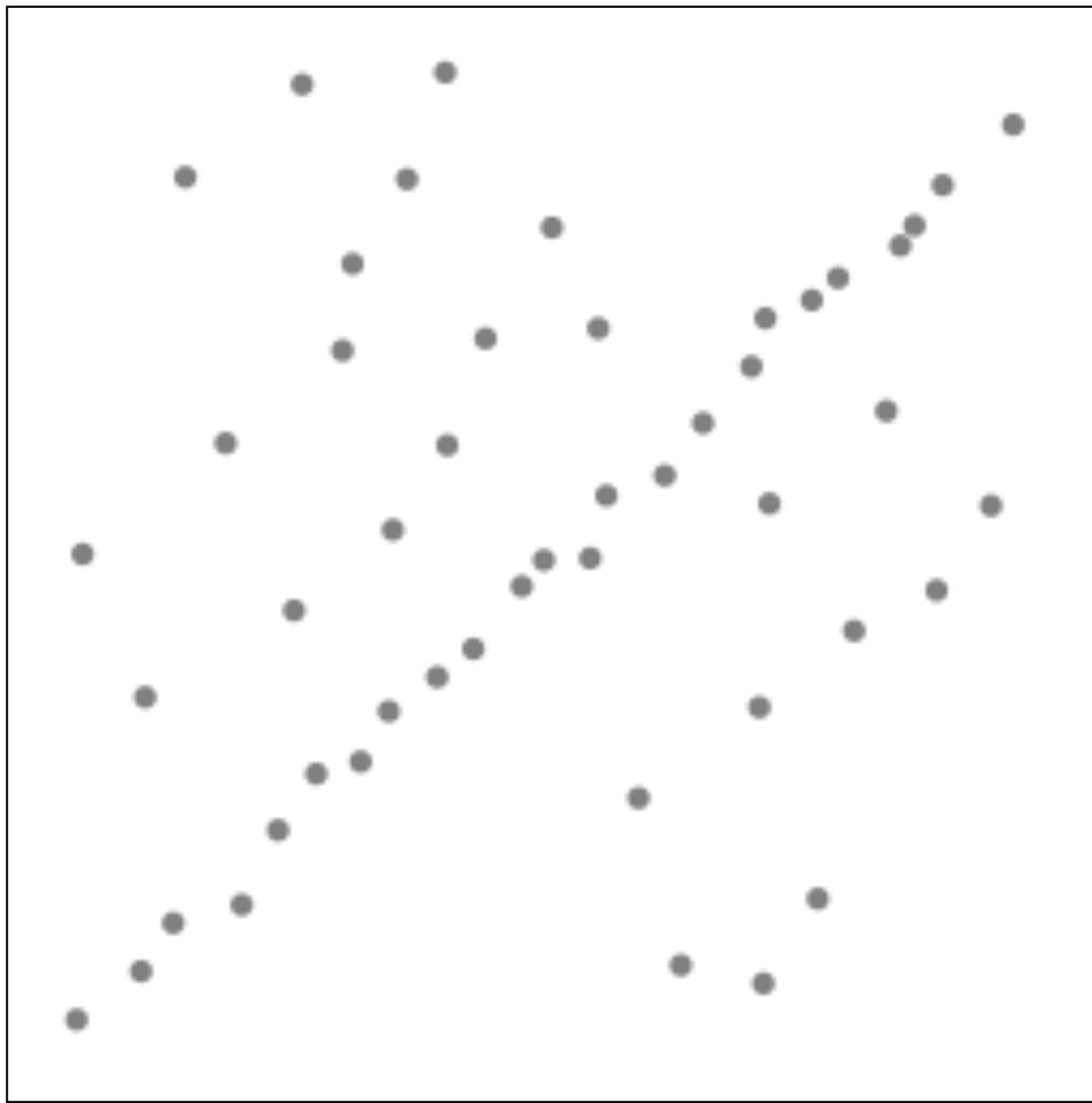


# Robust loss functions



# Handling outliers

- Can be hard to fit robust loss
  - Can be low, or get stuck in bad local minima
- Let's consider the problem of linear regression

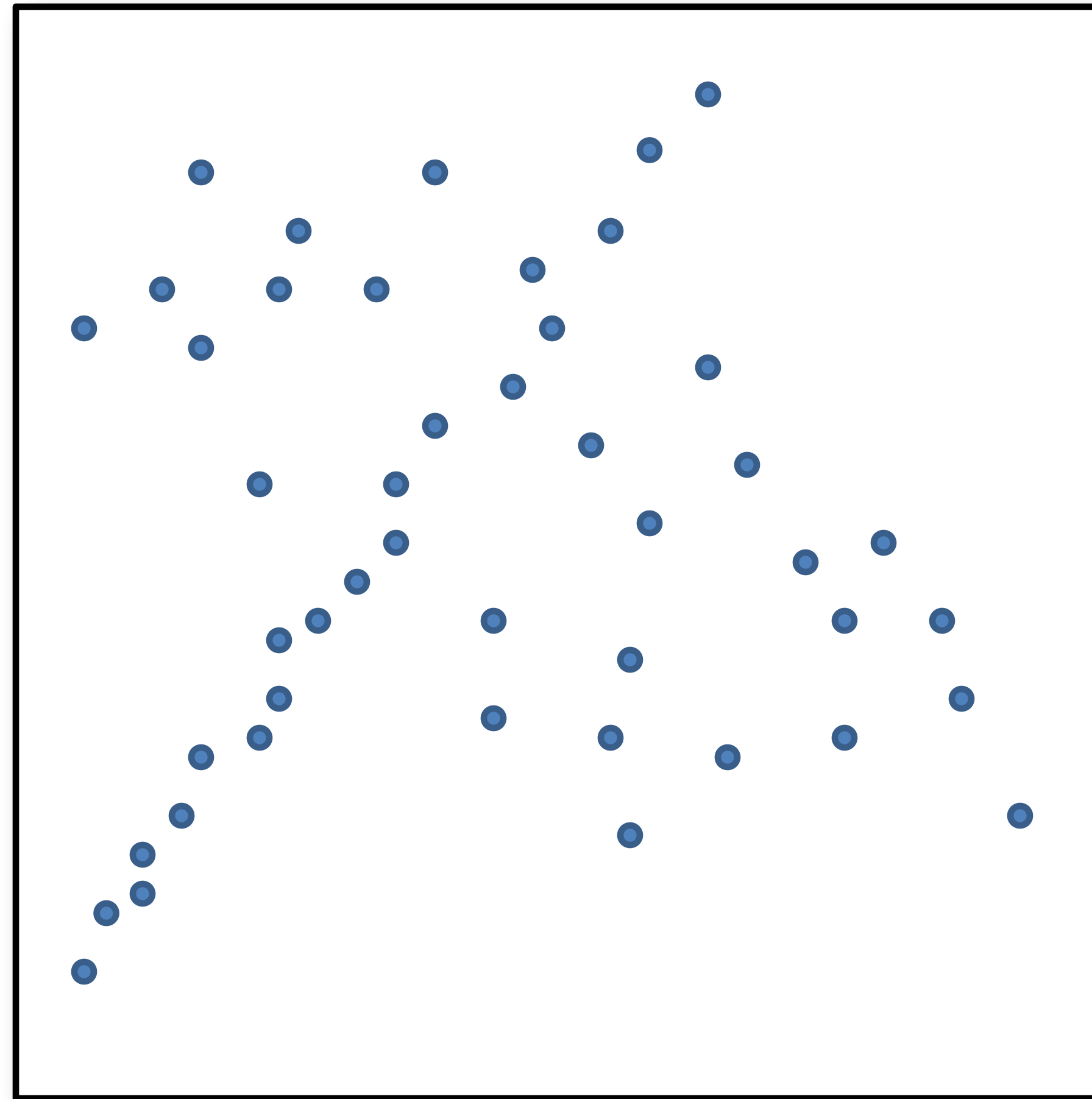


Problem: Fit a line to these data points

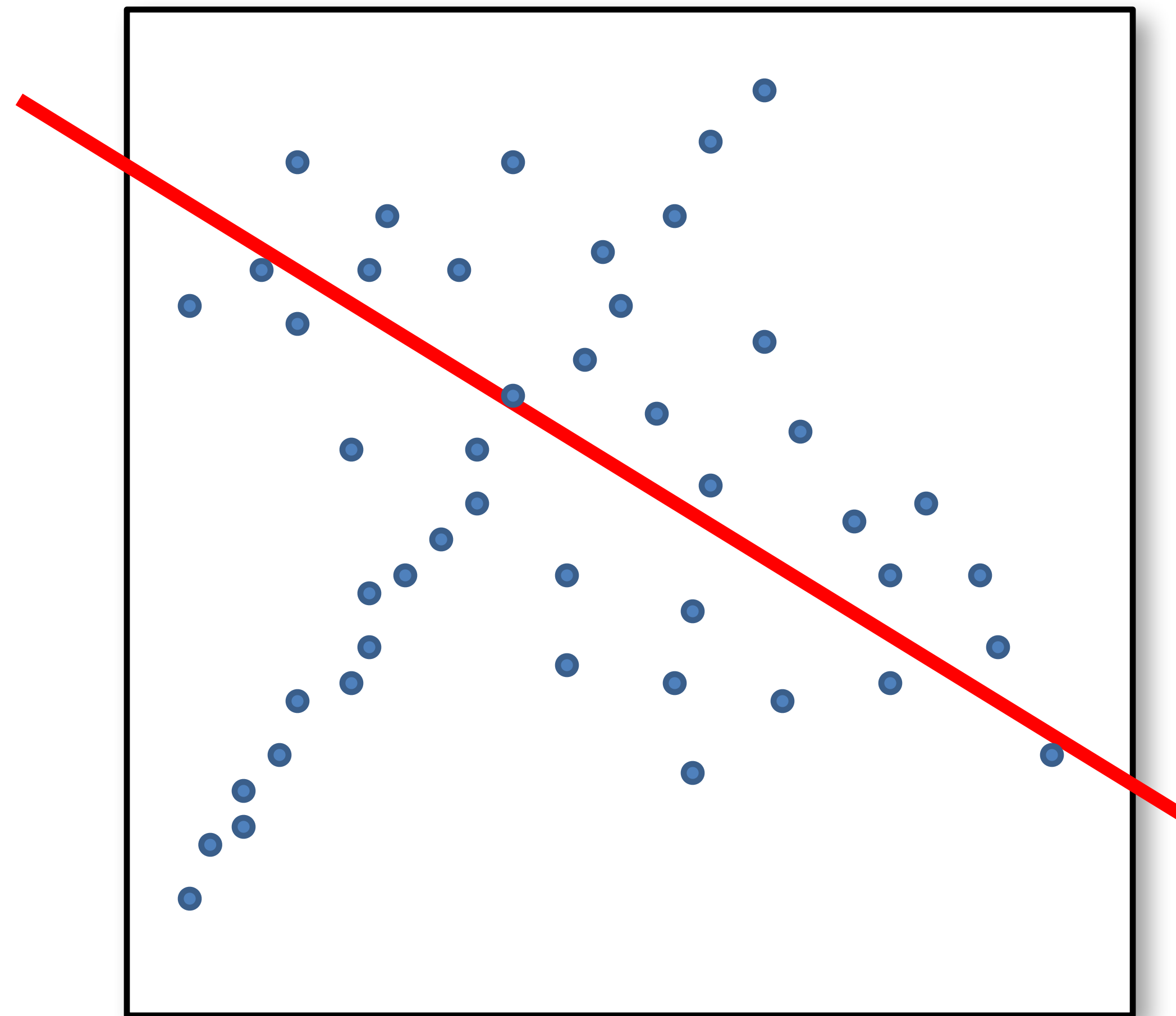
Least squares fit



# Counting inliers



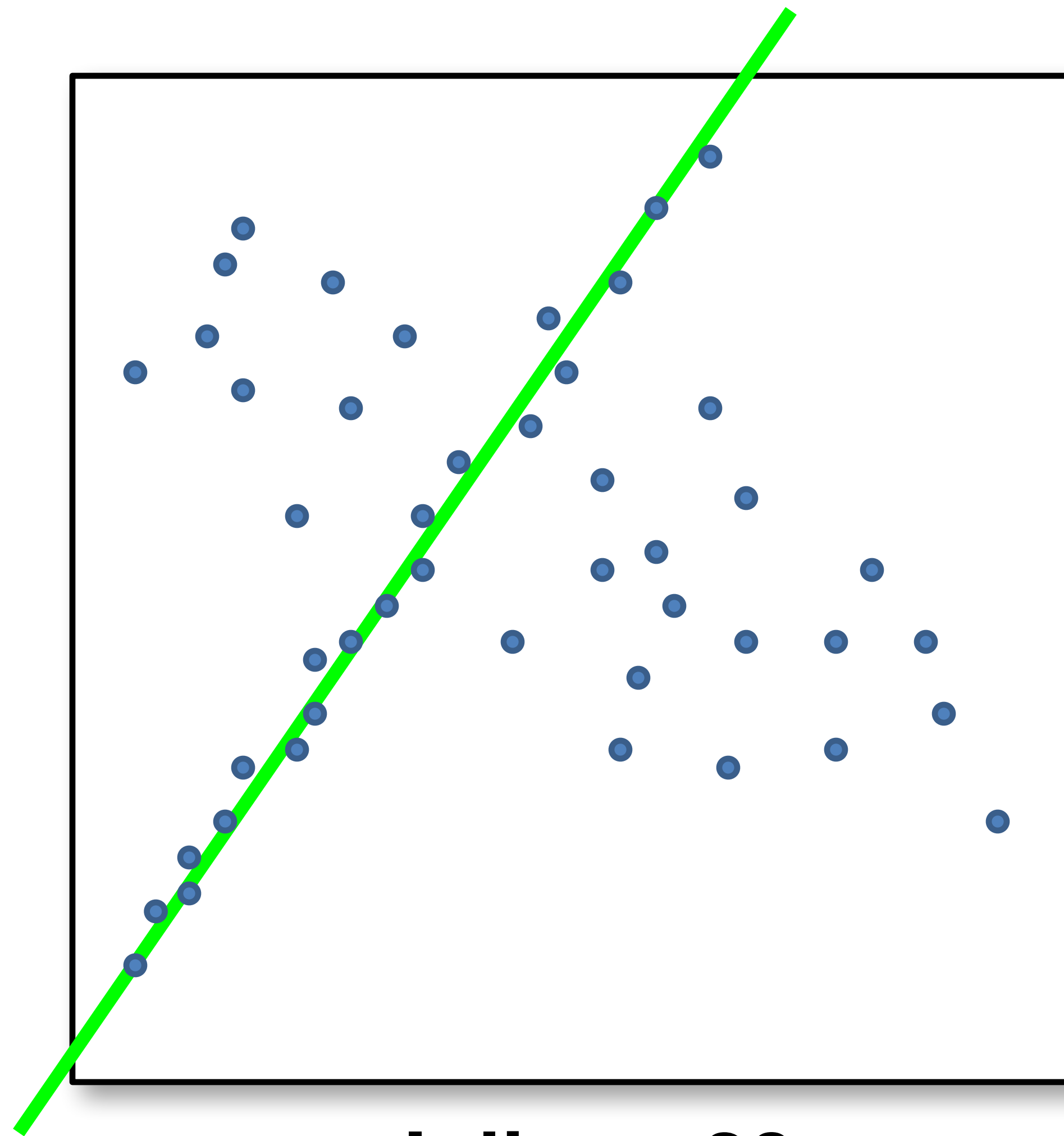
# Counting inliers



**Inliers: 3**



# Counting inliers



**Inliers: 20**

# RANSAC

- Idea:
  - All the inliers will agree with each other on the solution; the (hopefully small) number of outliers will (hopefully) disagree with each other
    - RANSAC only has guarantees if there are  $< 50\%$  outliers
  - “All good matches are alike; every bad match is bad in its own way.”
    - Tolstoy via Alyosha Efros



# RANSAC: random sample consensus

RANSAC loop (for N iterations):

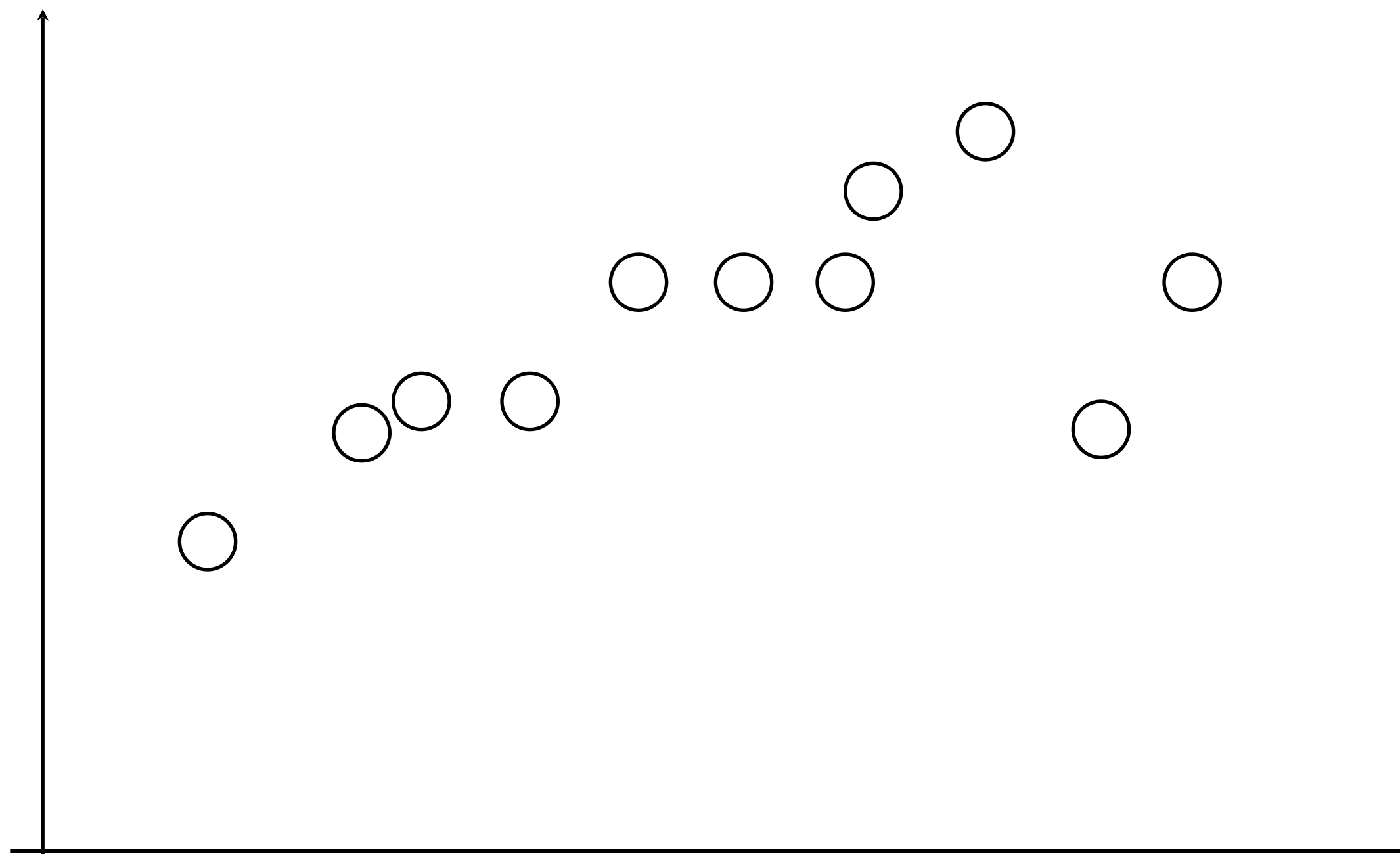
- Select four feature pairs (at random)
- Compute homography  $H$
- Count **inliers** where  $\|p_i' - \mathbf{H} p_i\| < \varepsilon$

Afterwards:

- Choose  $\mathbf{H}$  with largest set of inliers
- Recompute  $\mathbf{H}$  using only those inliers (often using high-quality nonlinear least squares)

# Simple example: fit a line

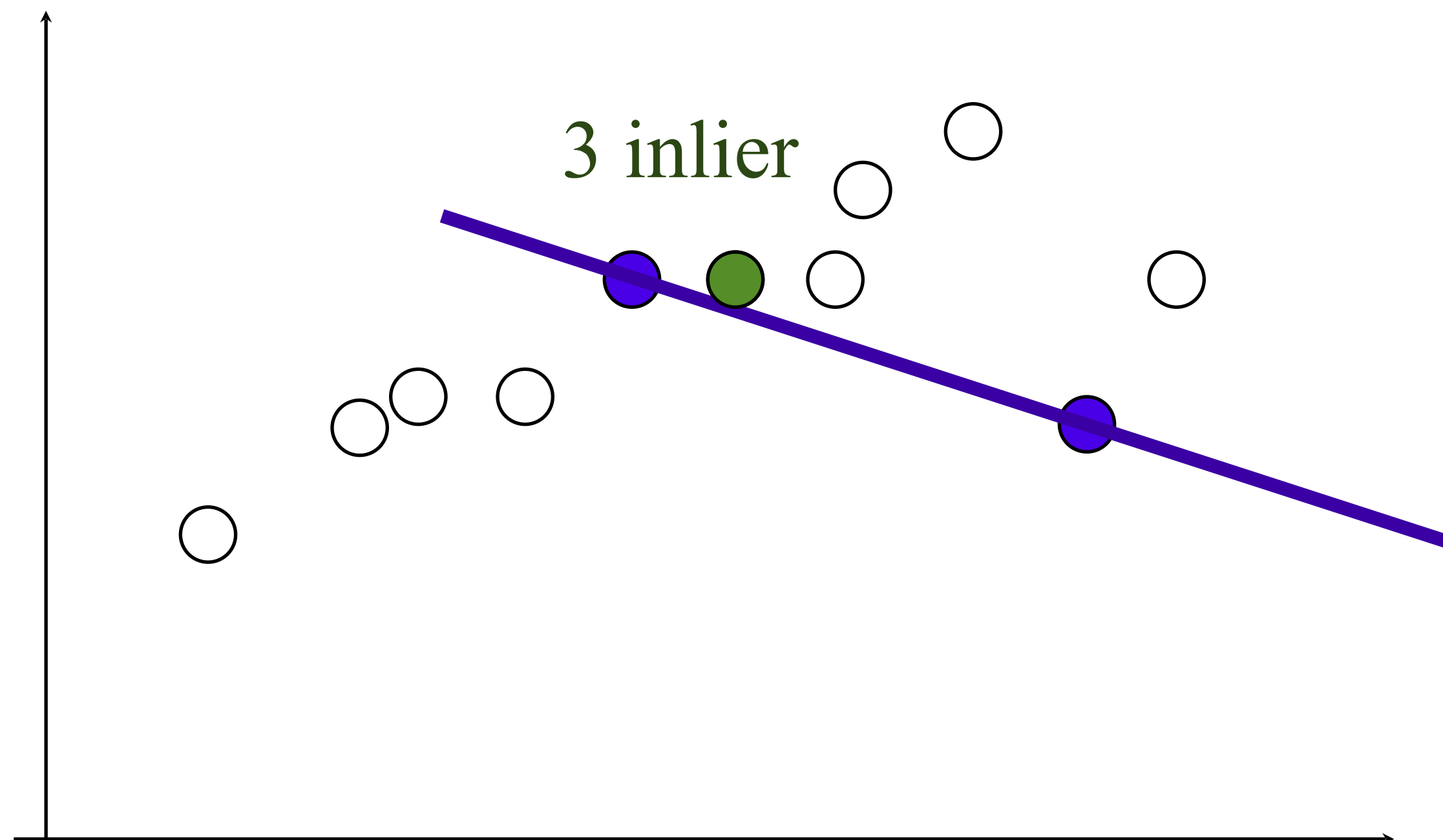
- Rather than homography  $H$  (8 numbers)  
fit  $y=ax+b$  (2 numbers  $a, b$ ) to 2D pairs





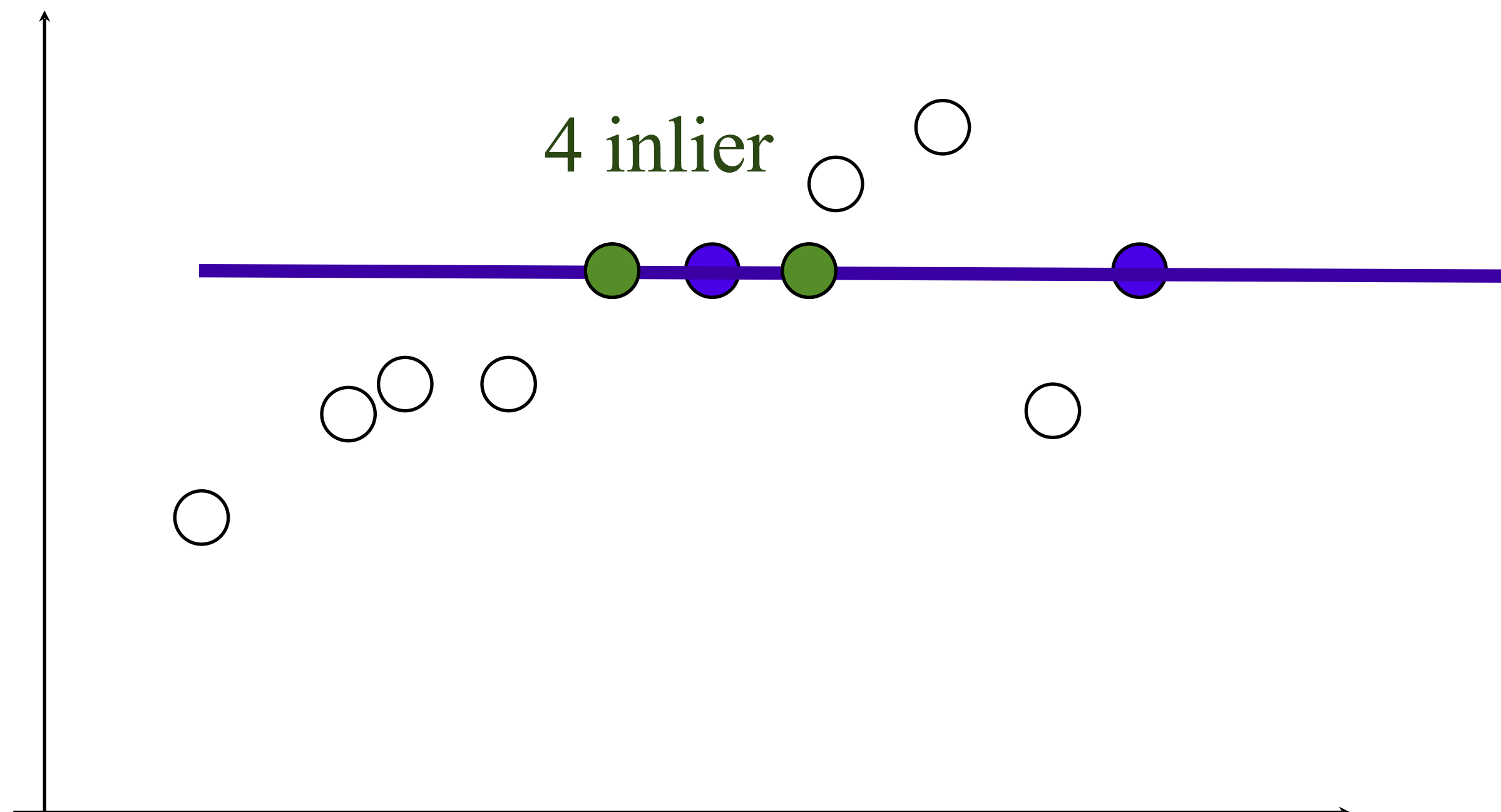
# Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers



# Simple example: fit a line

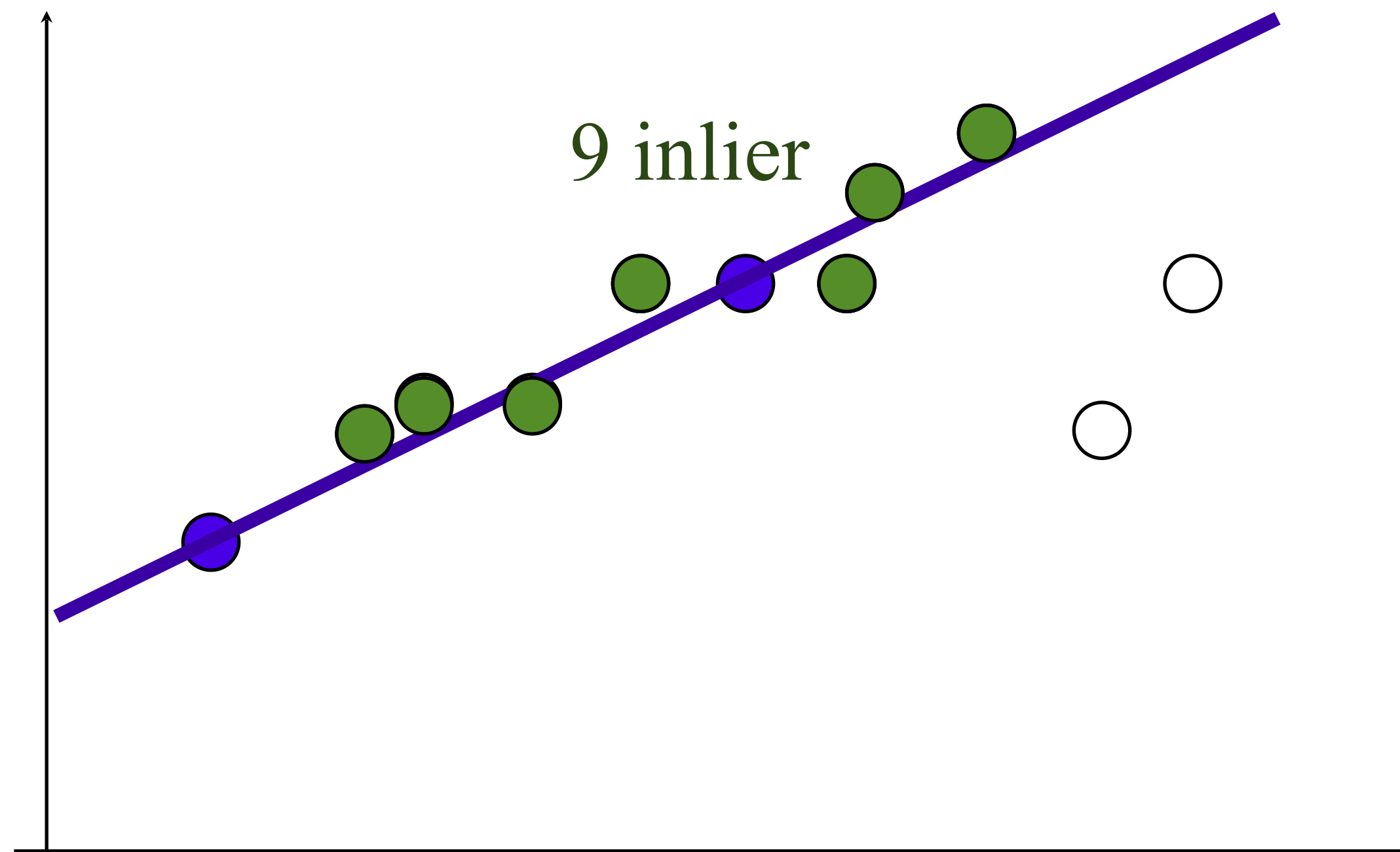
- Pick 2 points
- Fit line
- Count inliers





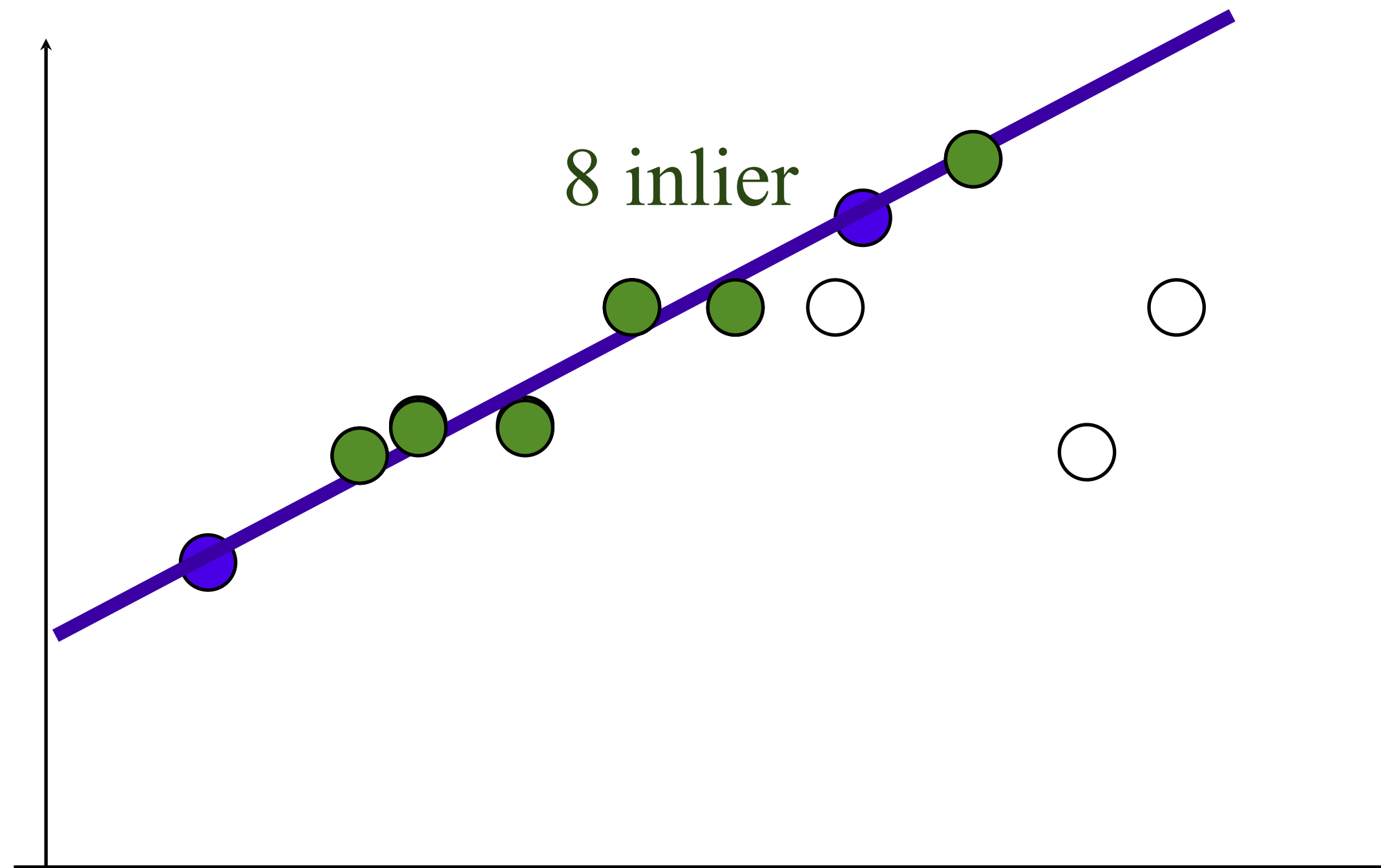
# Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers



# Simple example: fit a line

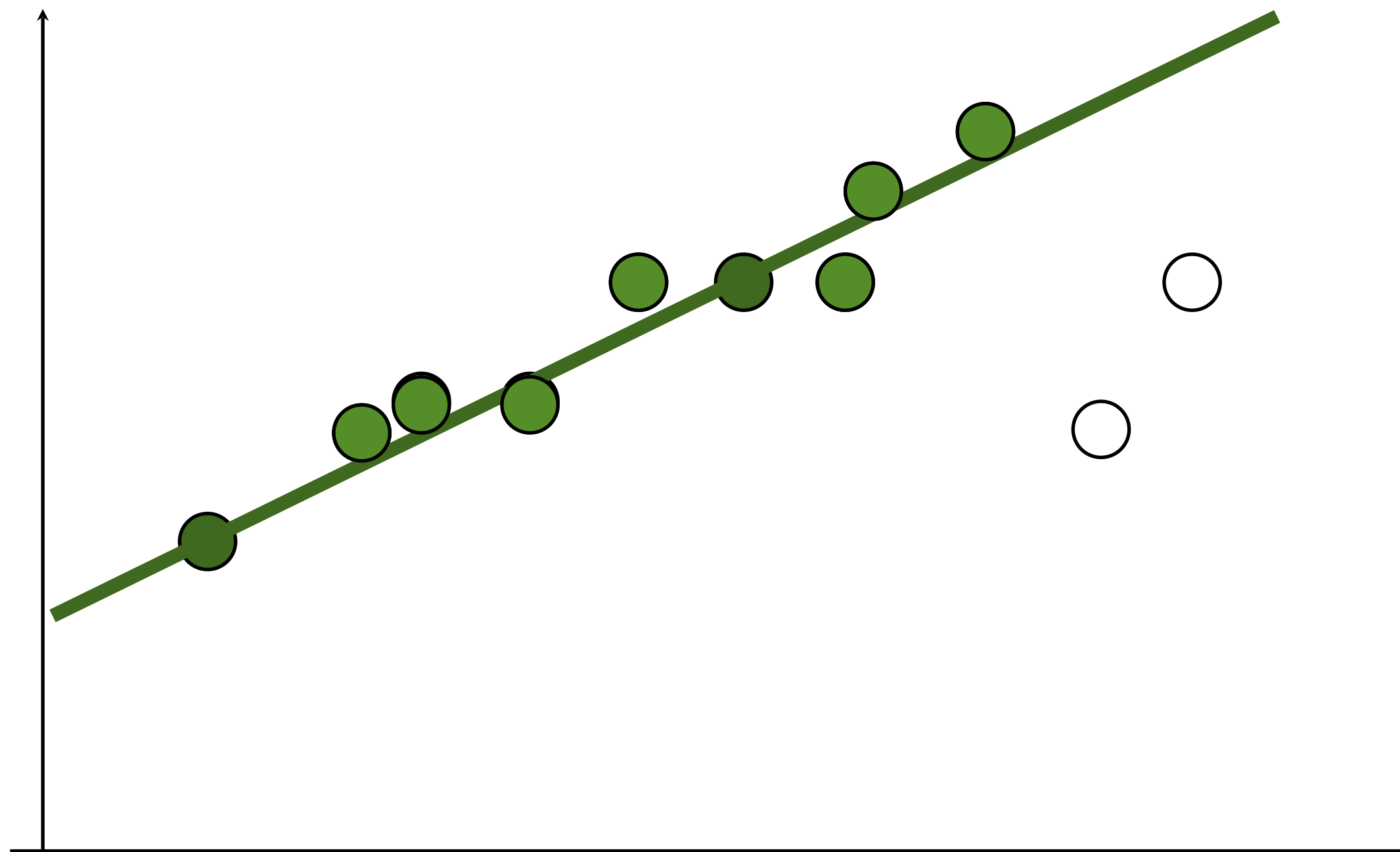
- Pick 2 points
- Fit line
- Count inliers



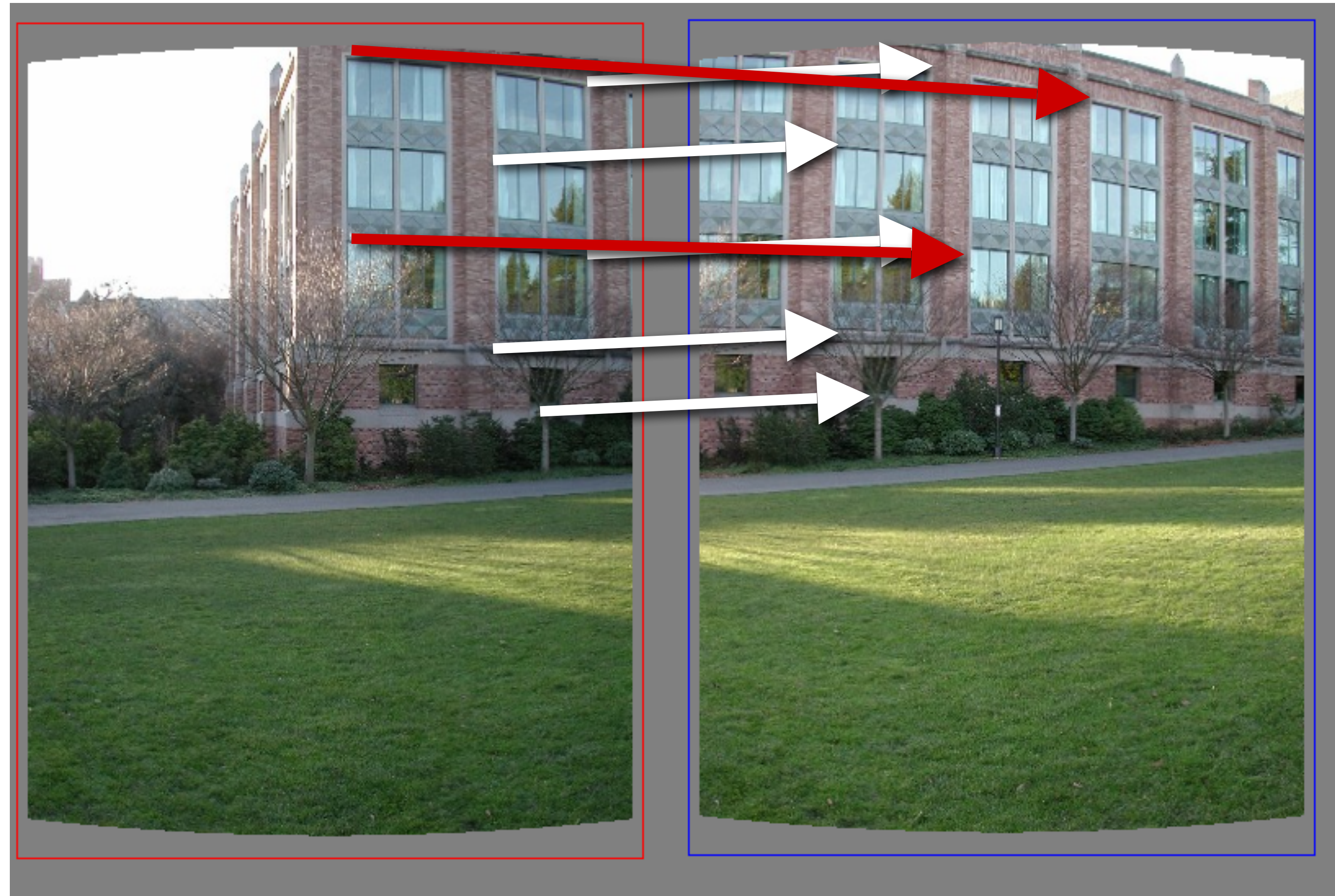


# Simple example: fit a line

- Use biggest set of inliers
- Do least-square fit

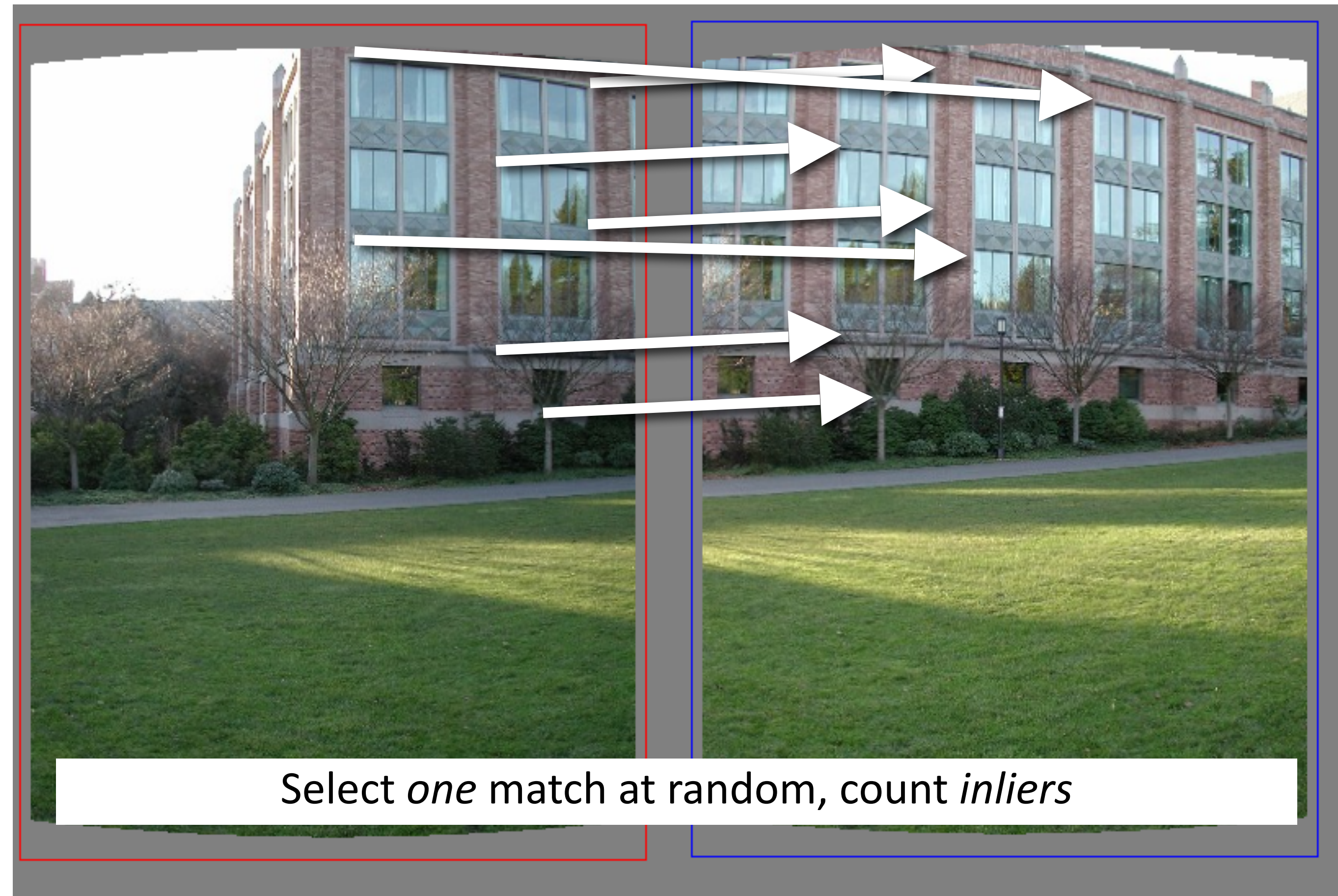


# Example: fitting a translation



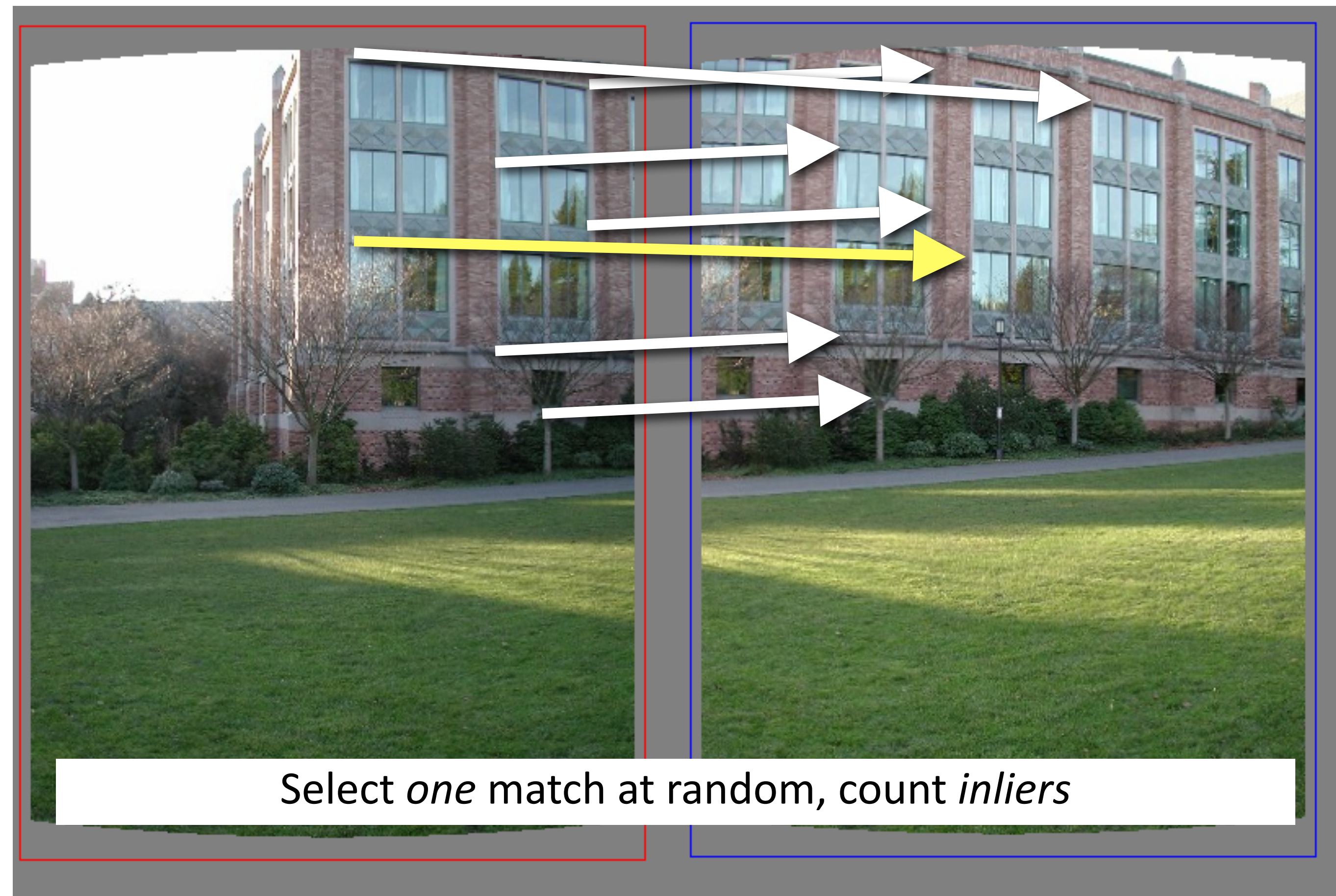


# Random Sample Consensus



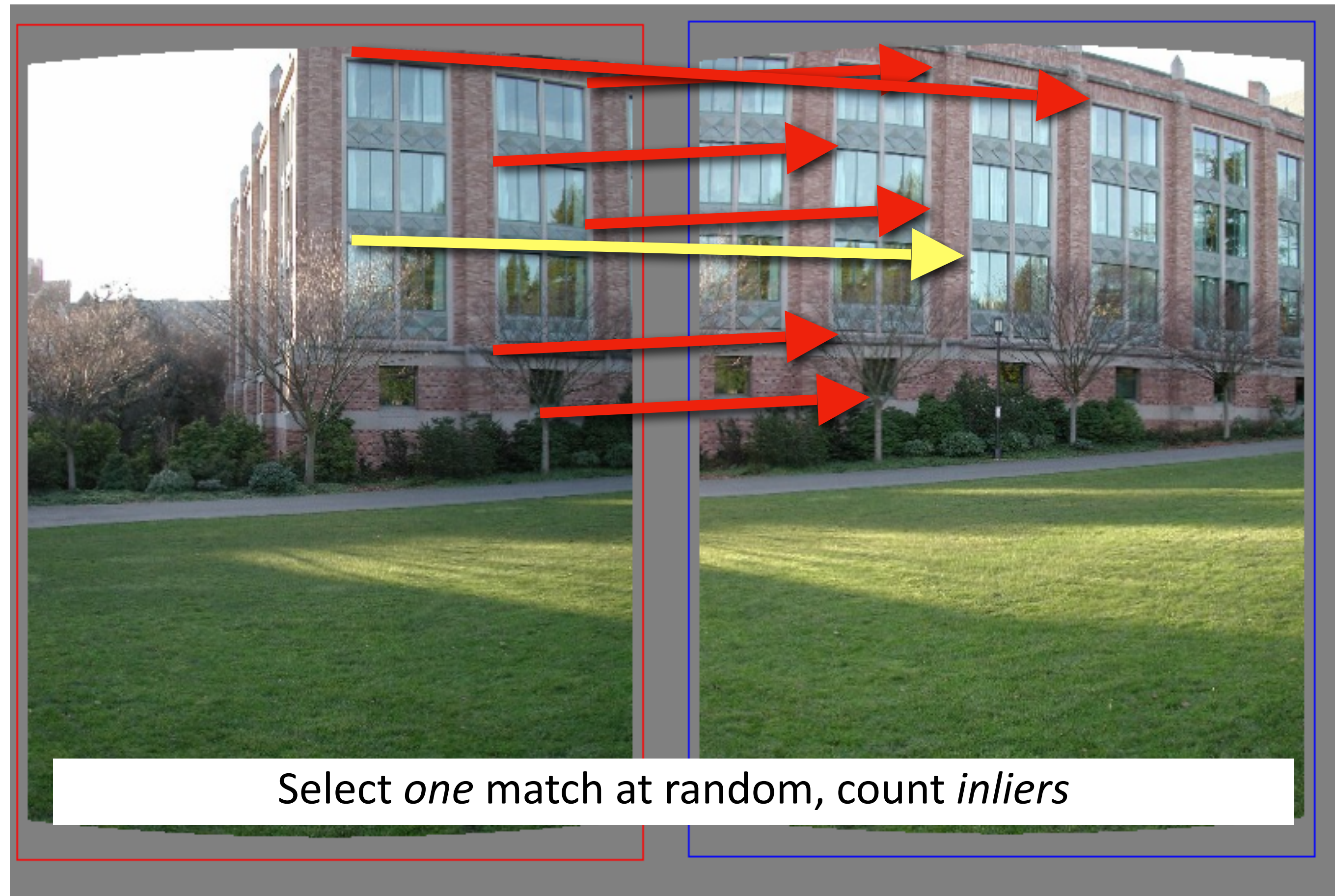


# Random Sample Consensus



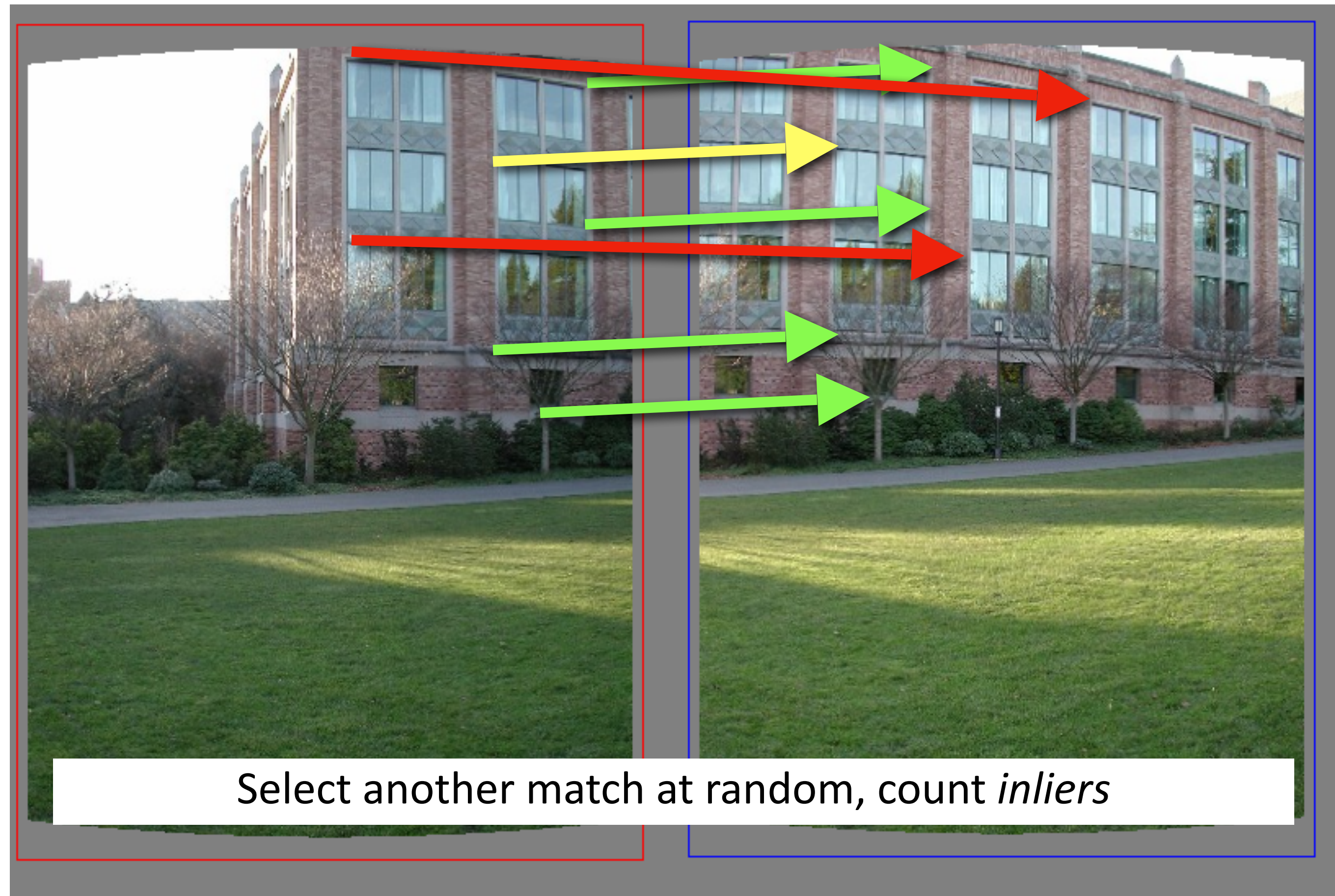


# Random Sample Consensus



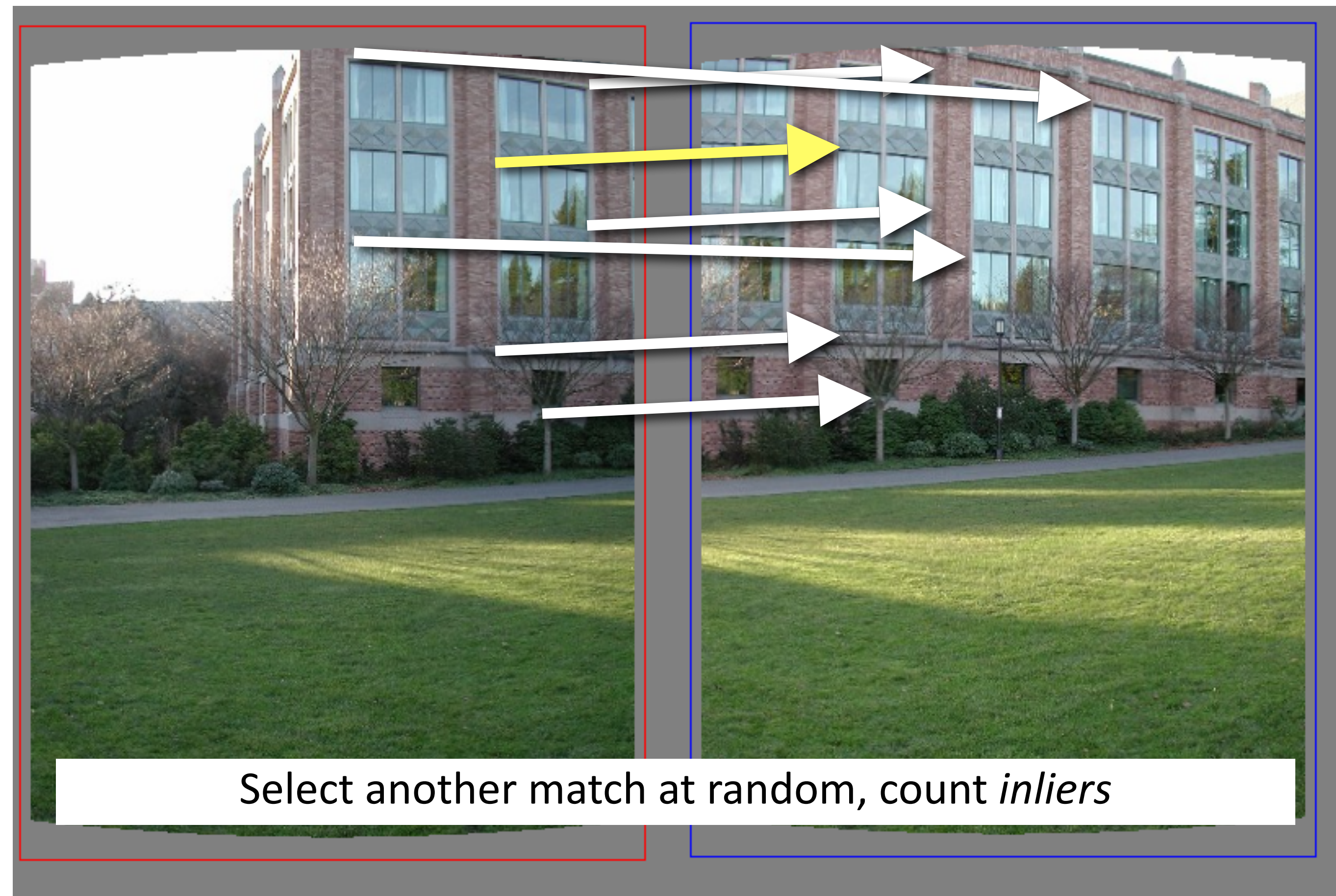


# Random Sample Consensus



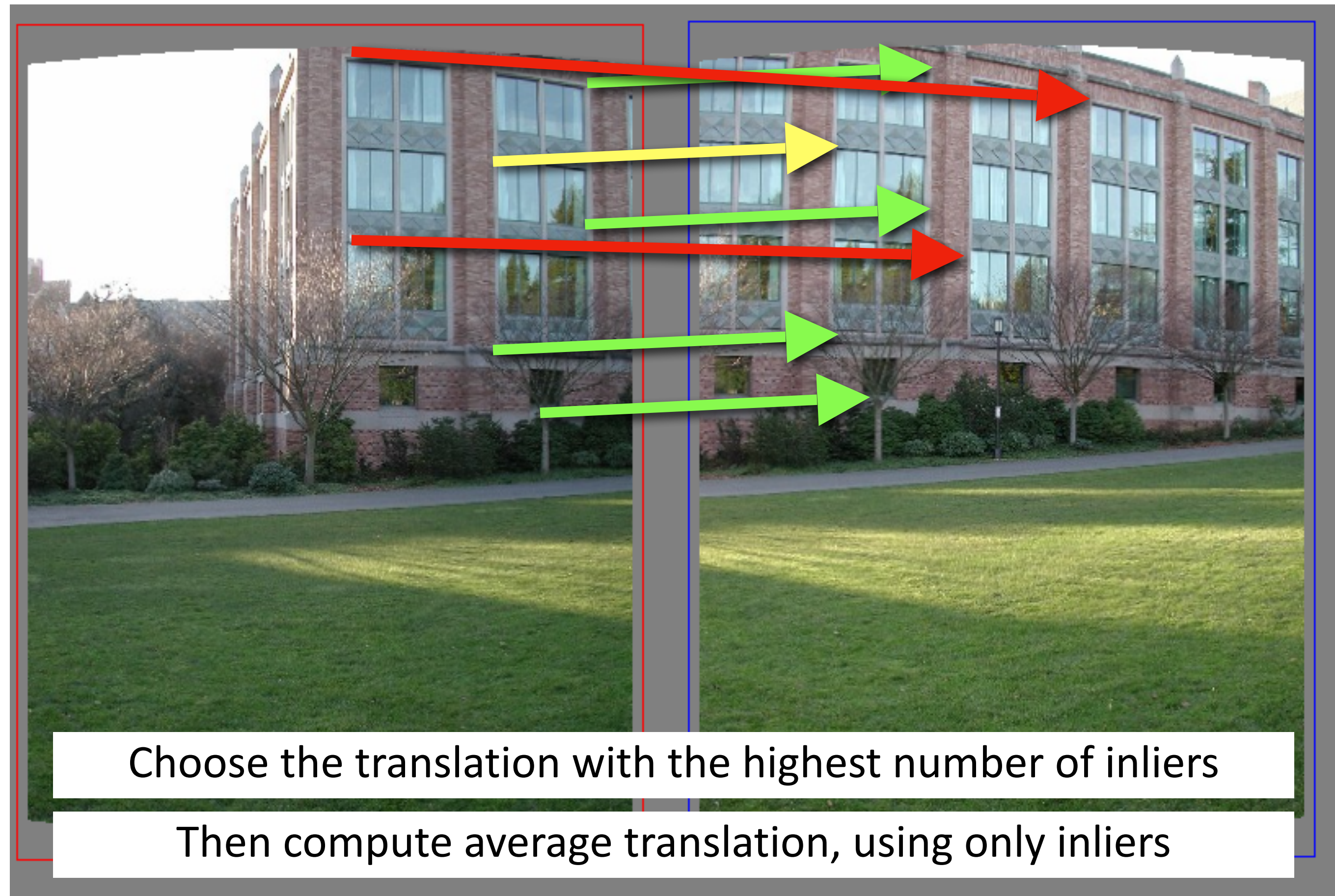


# Random Sample Consensus



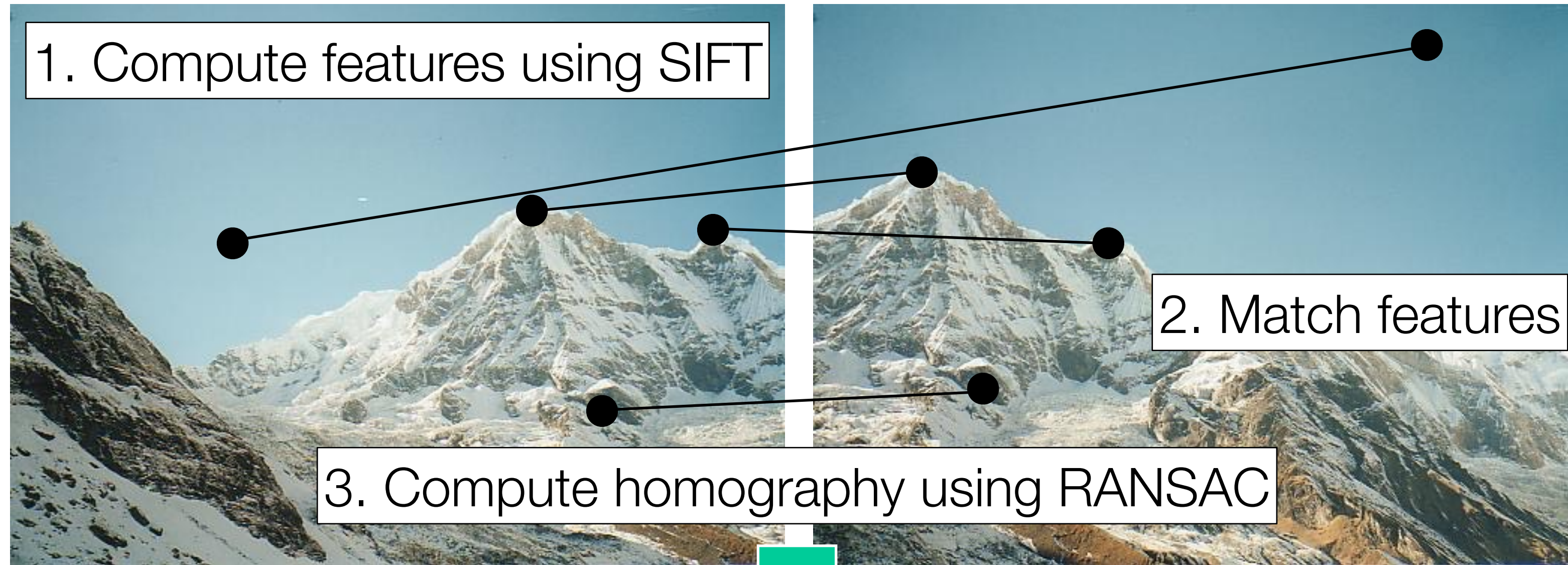


# Random Sample Consensus





# Warping with a homography (PS9)





Next class: more 3D