

# Kanade-Lucas-Tomasi (KLT) tracker



<https://www.youtube.com/watch?v=rwljkECpY0M>

# Feature-based tracking

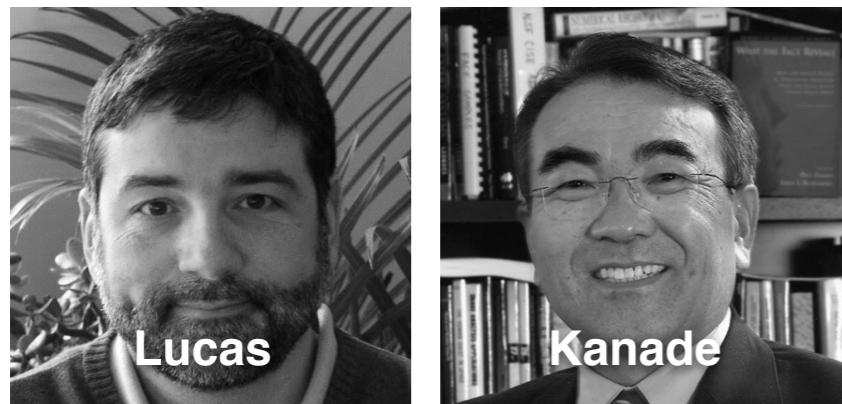
Up to now, we've been aligning entire images  
but we can also track just small image regions too!

(sometimes called sparse tracking or sparse alignment)

How should we select the ‘small images’ (features)?

How should we track them from frame to frame?

# History of the Kanade-Lucas-Tomasi (KLT) Tracker



An Iterative Image Registration Technique  
with an Application to Stereo Vision.

**1981**



**1991**

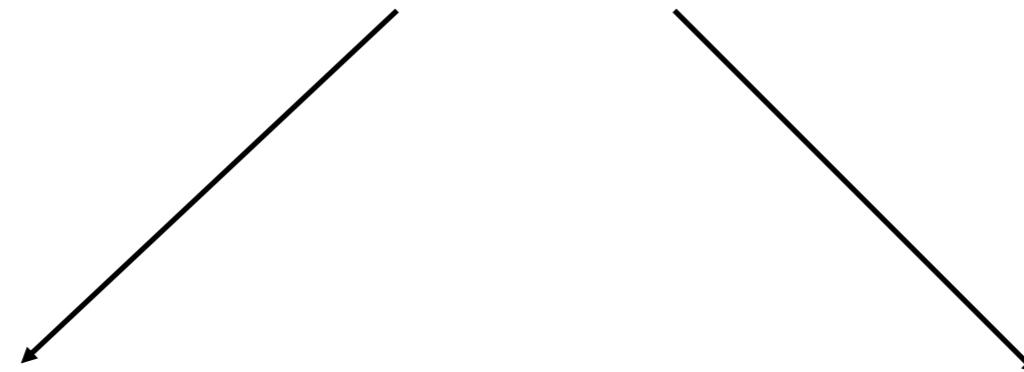
The original KLT algorithm



Good Features to Track.

**1994**

# Kanade-Lucas-Tomasi



**Lucas-Kanade**

Method for aligning  
(tracking) an image patch

**Tomasi-Kanade**

Method for choosing the  
best feature (image patch)  
for tracking

*What are good features for tracking?*

## *What are good features for tracking?*

Intuitively, we want to avoid smooth regions and edges.

But is there a more principled way to define good features?

## *What are good features for tracking?*

Can be derived from the tracking algorithm

## *What are good features for tracking?*

Can be derived from the tracking algorithm

*'A feature is good if it can be tracked well'*

Recall the Lucas-Kanade image alignment method:

error function (SSD) 
$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2$$

incremental update 
$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p})) - T(\mathbf{x})]^2$$

Recall the Lucas-Kanade image alignment method:

error function (SSD) 
$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2$$

incremental update 
$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p})) - T(\mathbf{x})]^2$$

linearize 
$$\sum_{\mathbf{x}} \left[ I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta \mathbf{p} - T(\mathbf{x}) \right]^2$$

Recall the Lucas-Kanade image alignment method:

error function (SSD) 
$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2$$

incremental update 
$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p})) - T(\mathbf{x})]^2$$

linearize 
$$\sum_{\mathbf{x}} \left[ I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta \mathbf{p} - T(\mathbf{x}) \right]^2$$

Gradient update 
$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^\top [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]$$

$$H = \sum_{\mathbf{x}} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^\top \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]$$

Recall the Lucas-Kanade image alignment method:

error function (SSD) 
$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2$$

incremental update 
$$\sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p})) - T(\mathbf{x})]^2$$

linearize 
$$\sum_{\mathbf{x}} \left[ I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2$$

Gradient update 
$$\Delta\mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^\top [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]$$

$$H = \sum_{\mathbf{x}} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^\top \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]$$

Update 
$$\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$$

Stability of gradient decent iterations depends on ...

$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^\top [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]$$

Stability of gradient decent iterations depends on ...

$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^\top [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]$$

Inverting the Hessian

$$H = \sum_{\mathbf{x}} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^\top \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]$$

*When does the inversion fail?*

Stability of gradient decent iterations depends on ...

$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^\top [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]$$

Inverting the Hessian

$$H = \sum_{\mathbf{x}} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^\top \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]$$

*When does the inversion fail?*

H is singular. But what does that mean?

Above the noise level

$$\lambda_1 \gg 0$$

$$\lambda_2 \gg 0$$

both Eigenvalues are large

Well-conditioned

both Eigenvalues have similar magnitude

Concrete example: Consider translation model

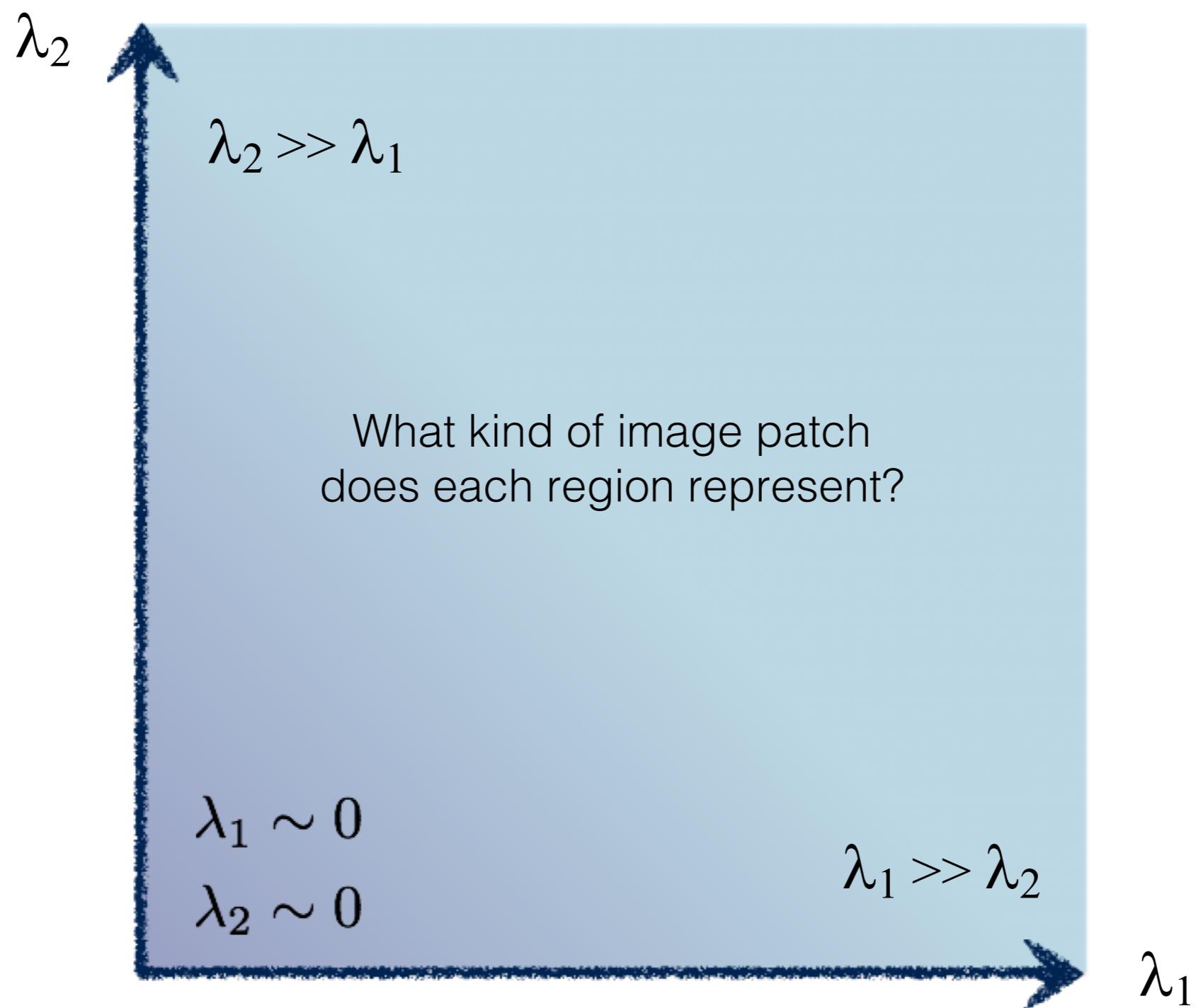
$$\mathbf{W}(\mathbf{x}; \mathbf{p}) = \begin{bmatrix} x + p_1 \\ y + p_2 \end{bmatrix} \quad \frac{\mathbf{W}}{\partial \mathbf{p}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Hessian

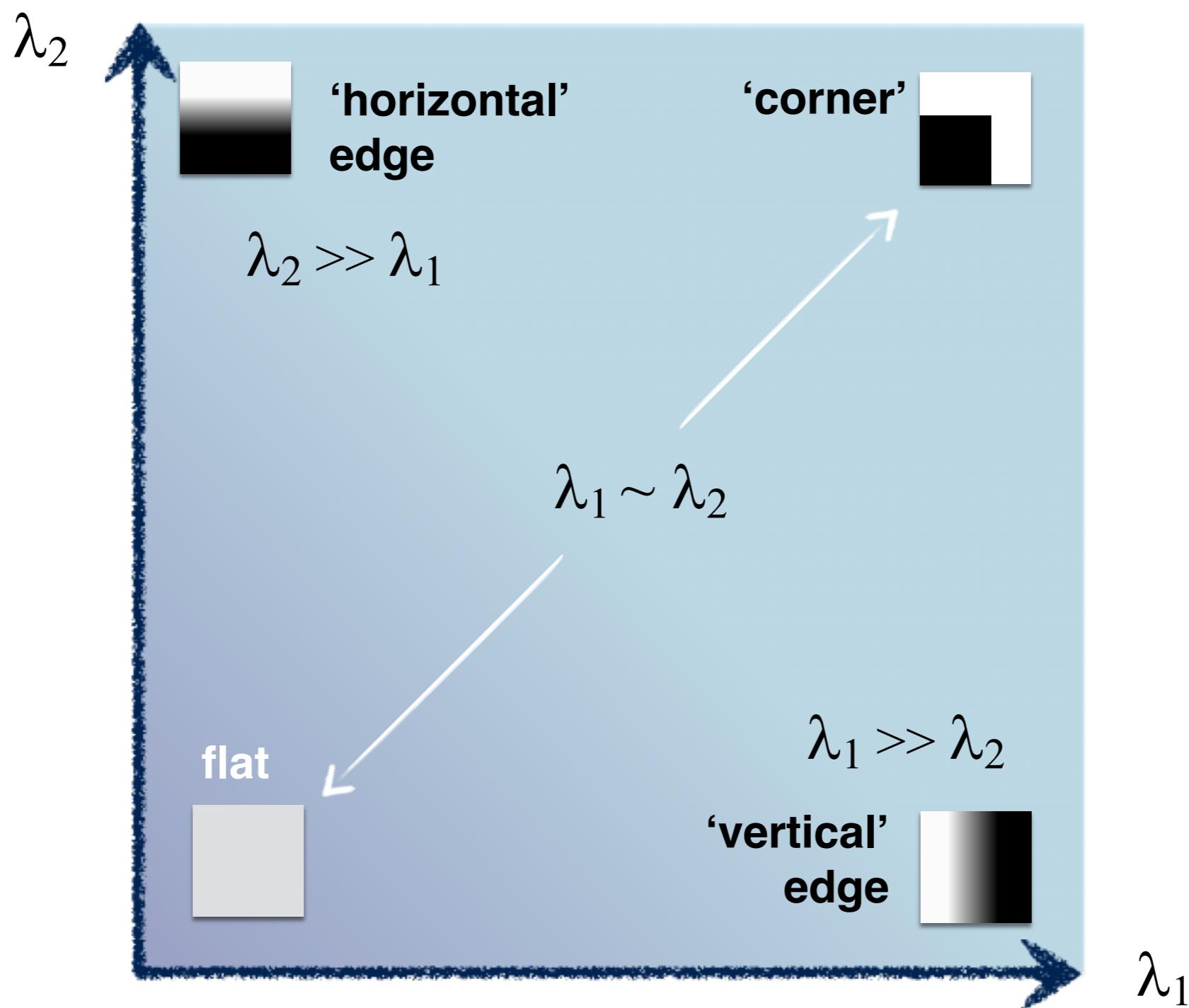
$$\begin{aligned} H &= \sum_{\mathbf{x}} \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^\top \left[ \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] \\ &= \sum_{\mathbf{x}} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \sum_{\mathbf{x}} I_x I_x & \sum_{\mathbf{x}} I_y I_x \\ \sum_{\mathbf{x}} I_x I_y & \sum_{\mathbf{x}} I_y I_y \end{bmatrix} \quad \leftarrow \text{when is this singular?} \end{aligned}$$

*How are the eigenvalues related to image content?*

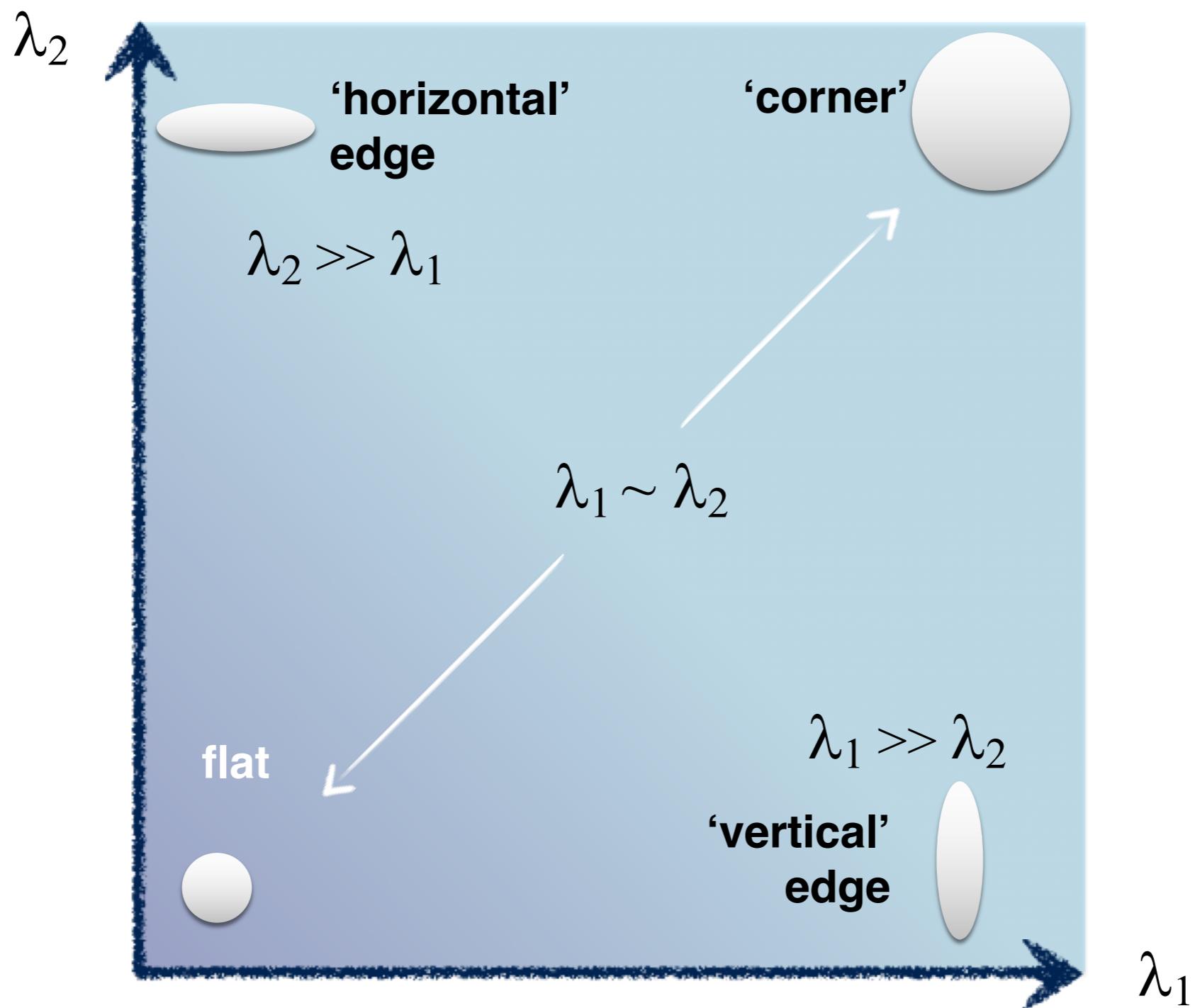
# interpreting eigenvalues



# interpreting eigenvalues



# interpreting eigenvalues



*What are good features for tracking?*

## *What are good features for tracking?*

$$\min(\lambda_1, \lambda_2) > \lambda$$

‘big Eigenvalues means good for tracking’

# KLT algorithm

1. Find corners satisfying  $\min(\lambda_1, \lambda_2) > \lambda$
2. For each corner compute displacement to next frame using the Lucas-Kanade method
3. Store displacement of each corner, update corner position
4. (optional) Add more corner points every M frames using 1
5. Repeat 2 to 3 (4)
6. Returns long trajectories for each corner point

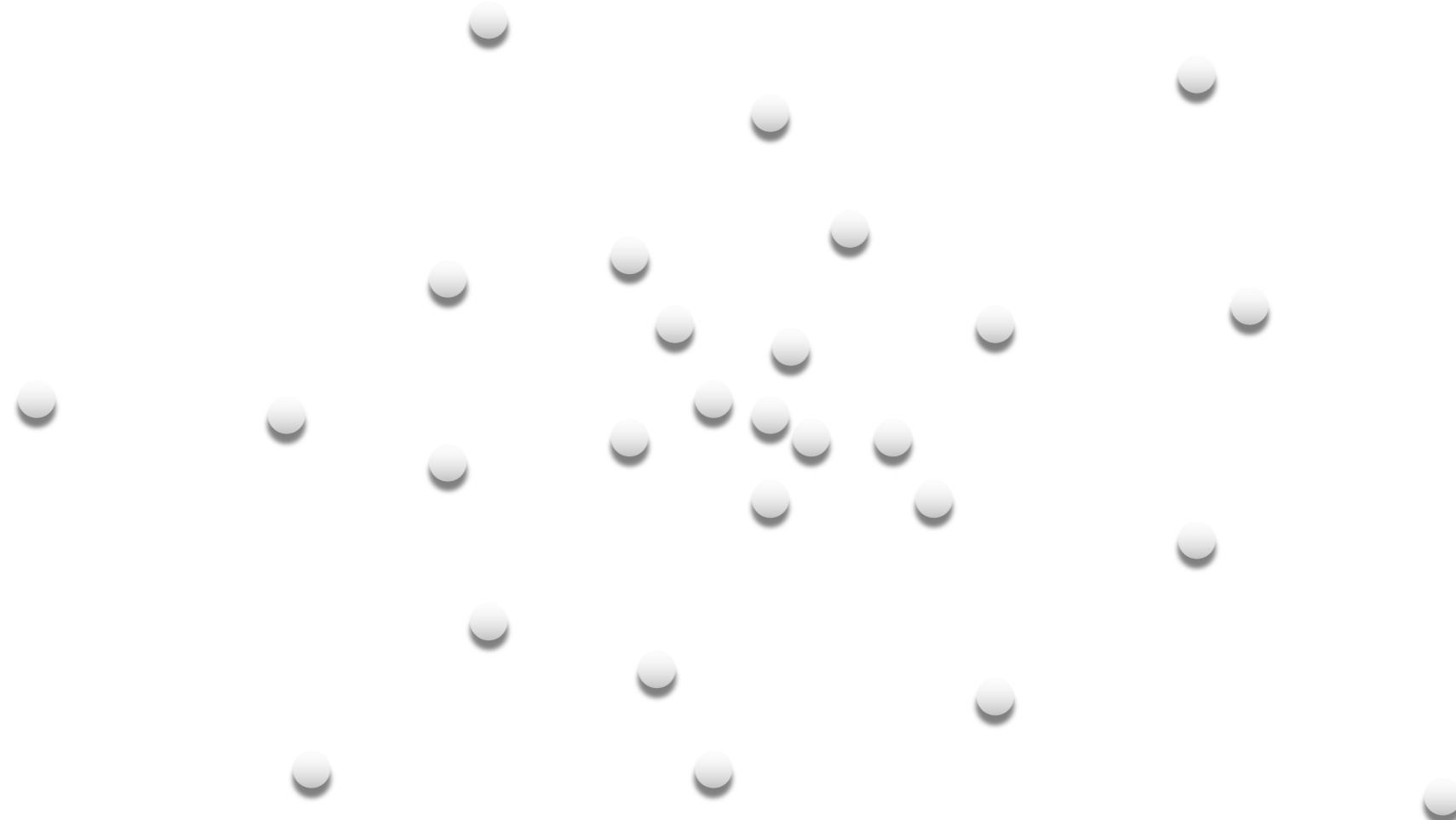
# Mean-shift algorithm



# Mean Shift Algorithm

A ‘mode seeking’ algorithm

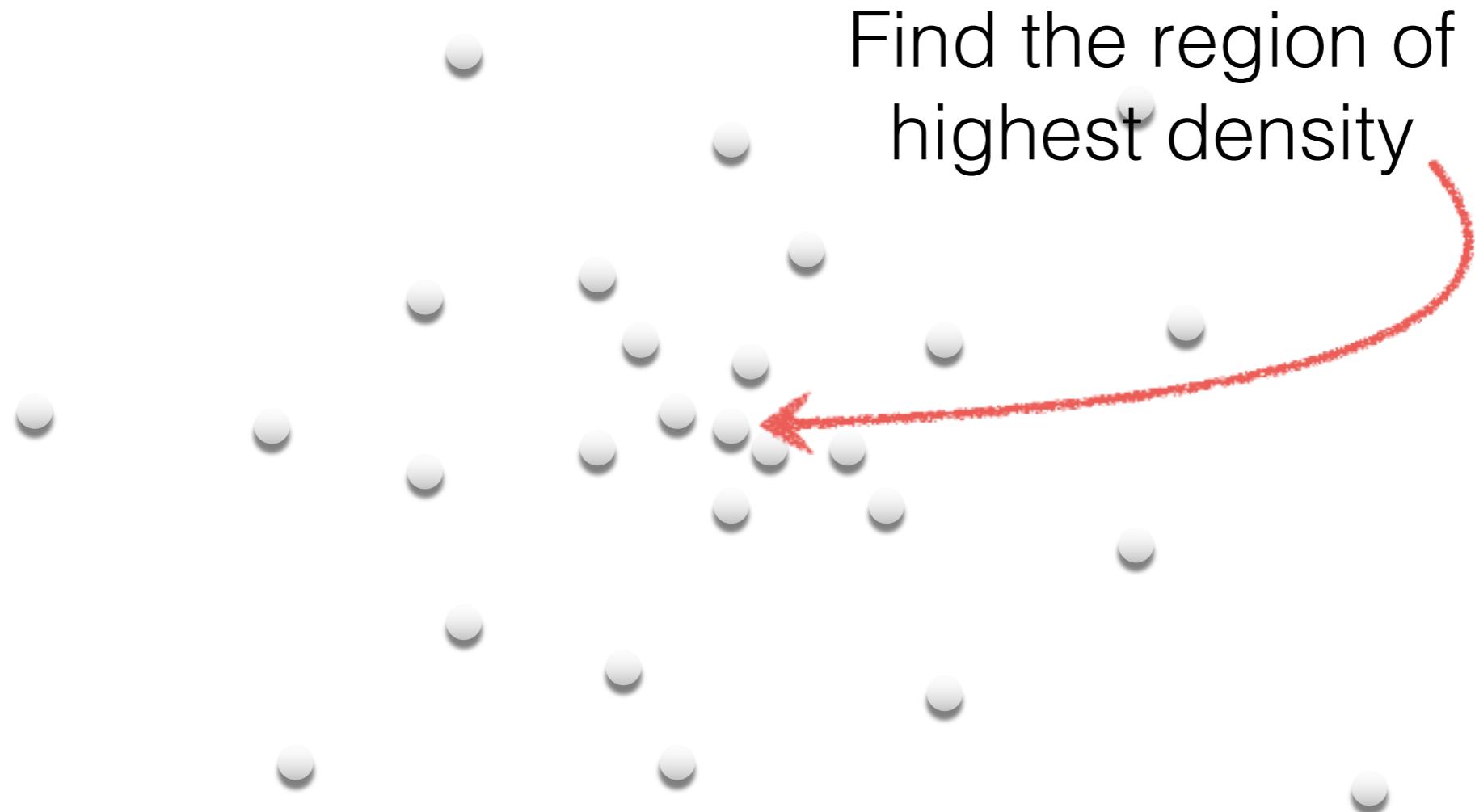
Fukunaga & Hostetler (1975)



# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

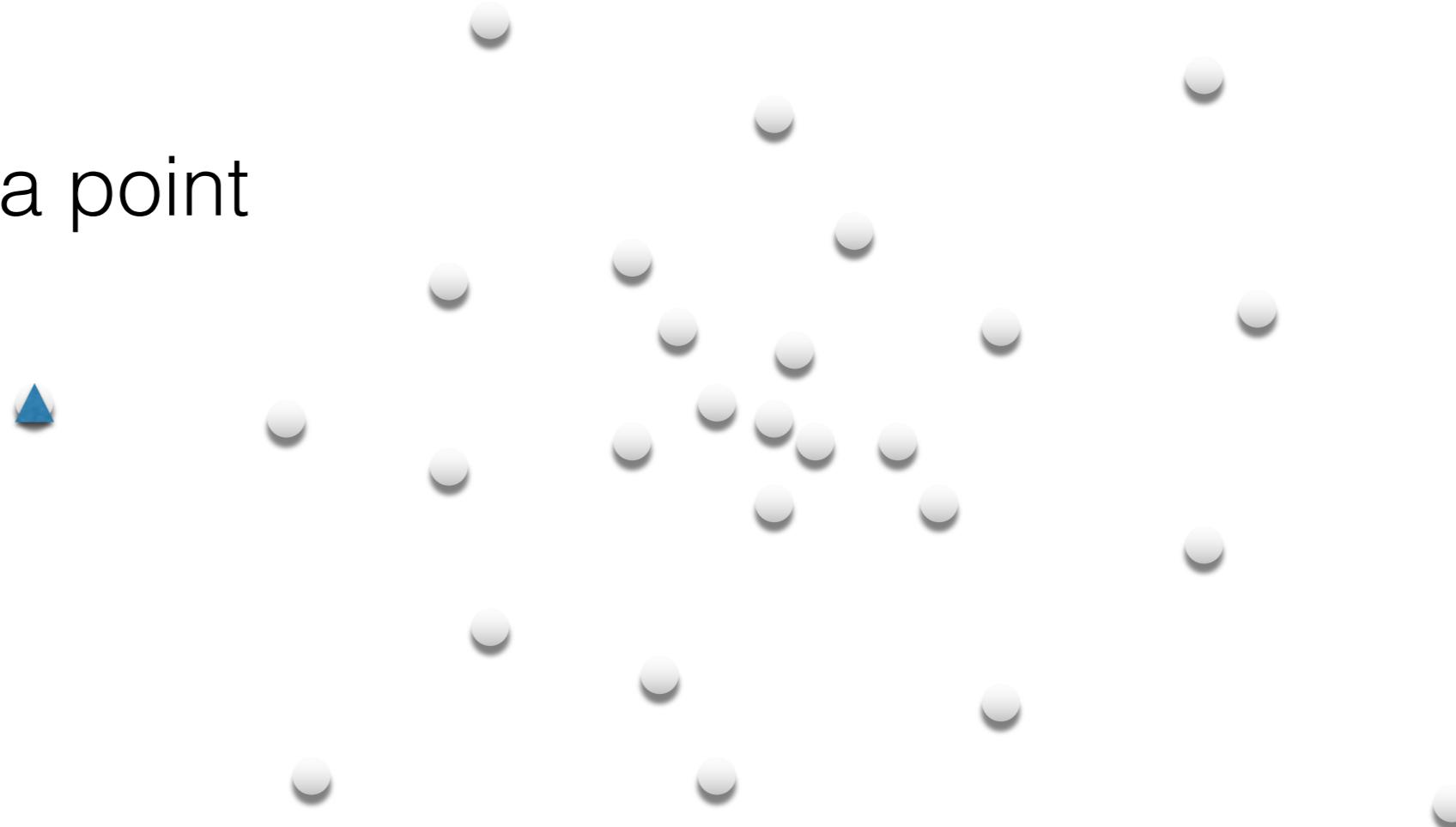


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

Pick a point

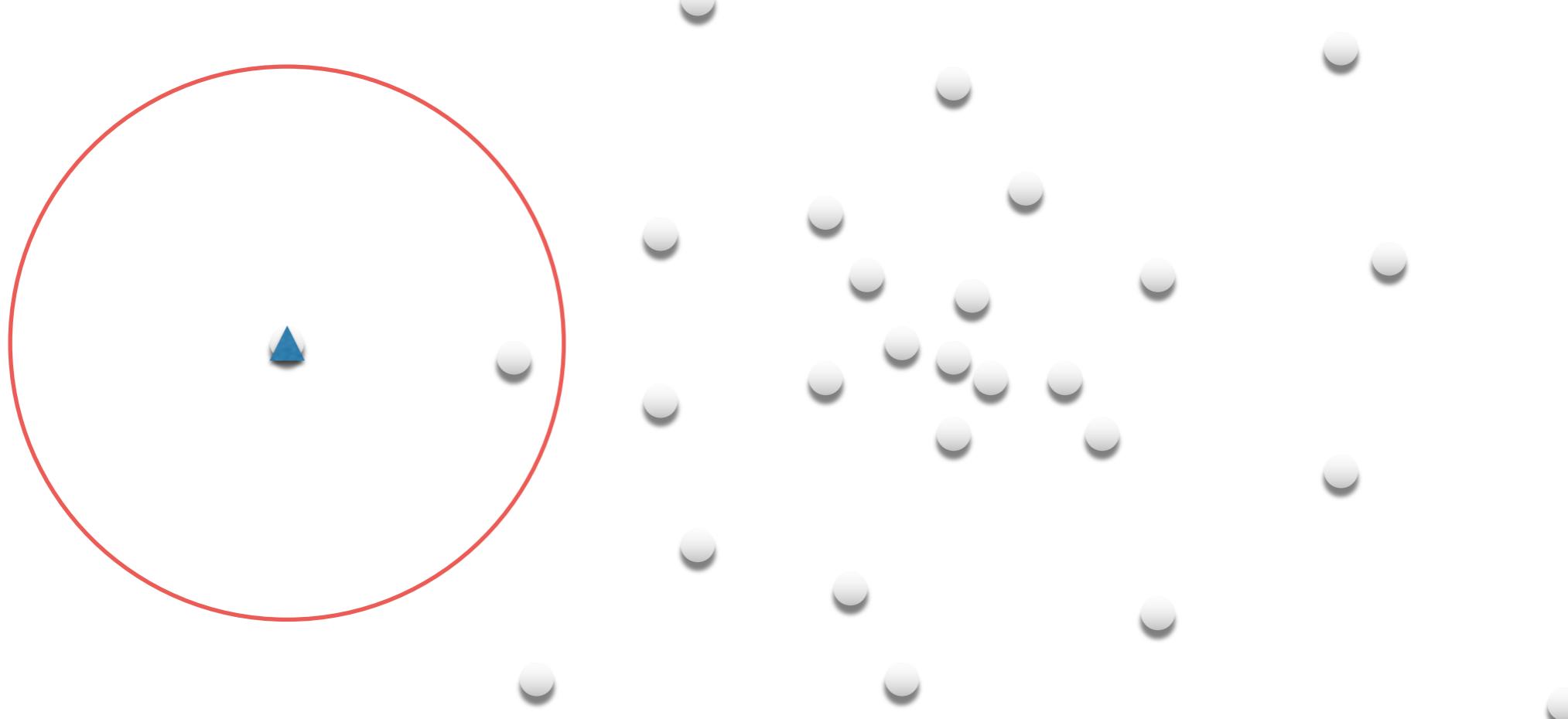


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

Draw a window

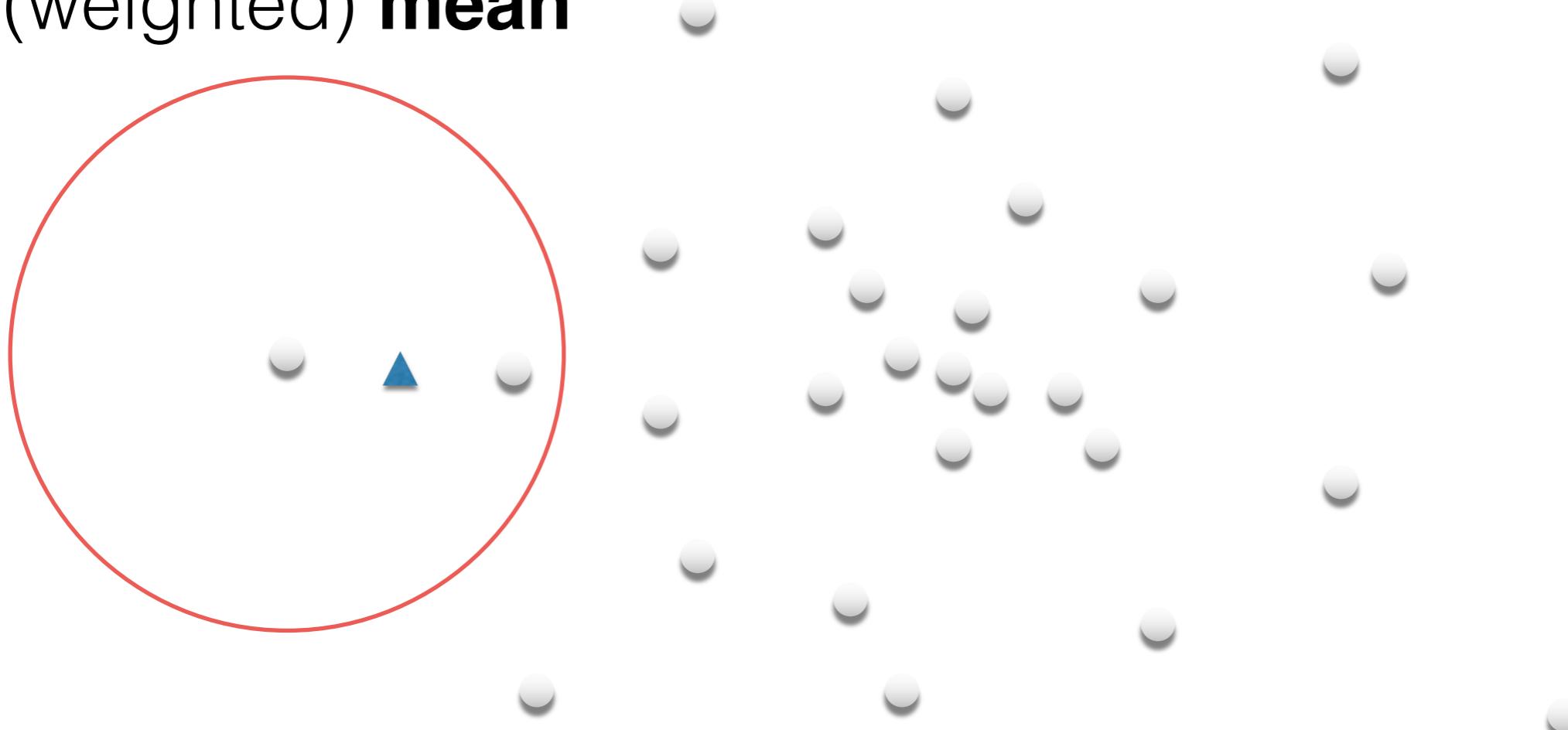


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

Compute the  
(weighted) **mean**

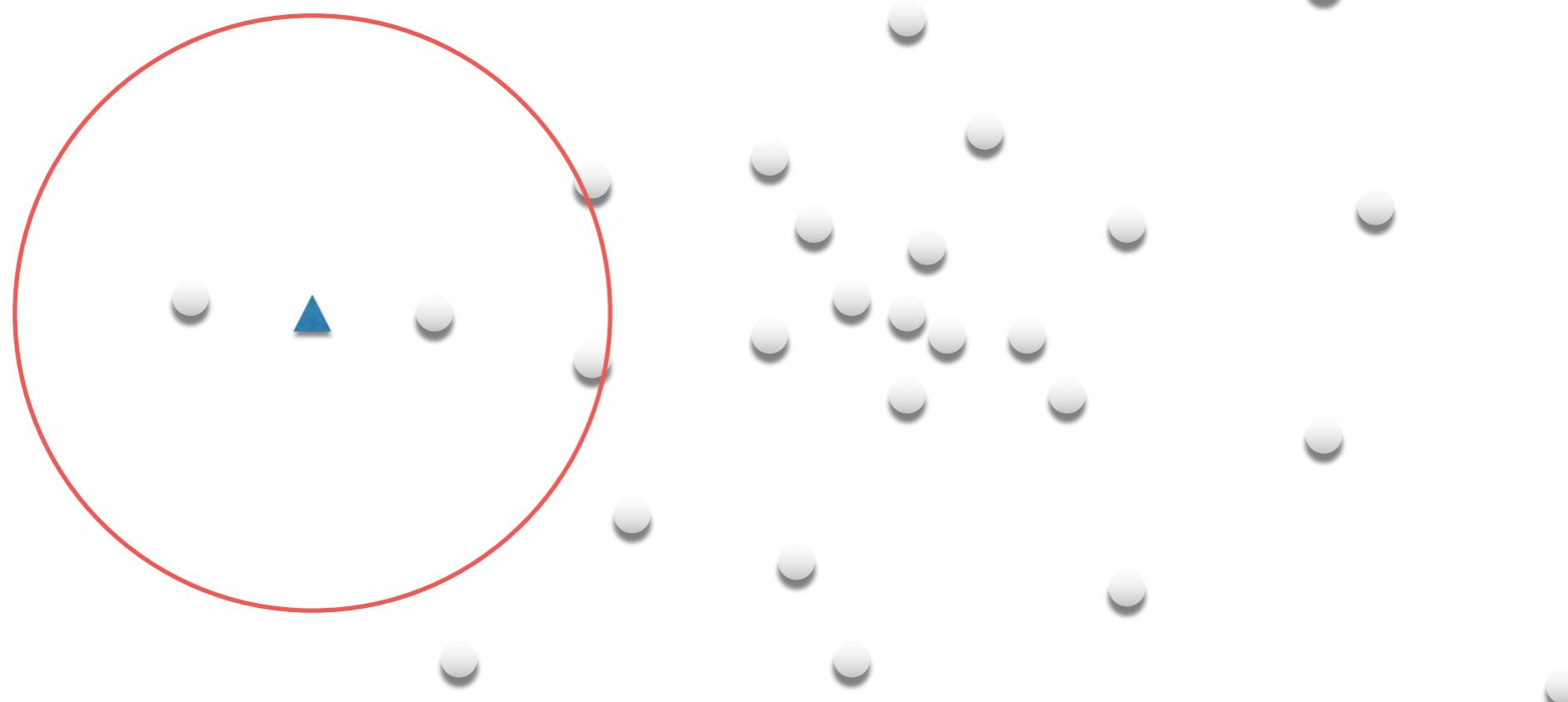


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

**Shift** the window

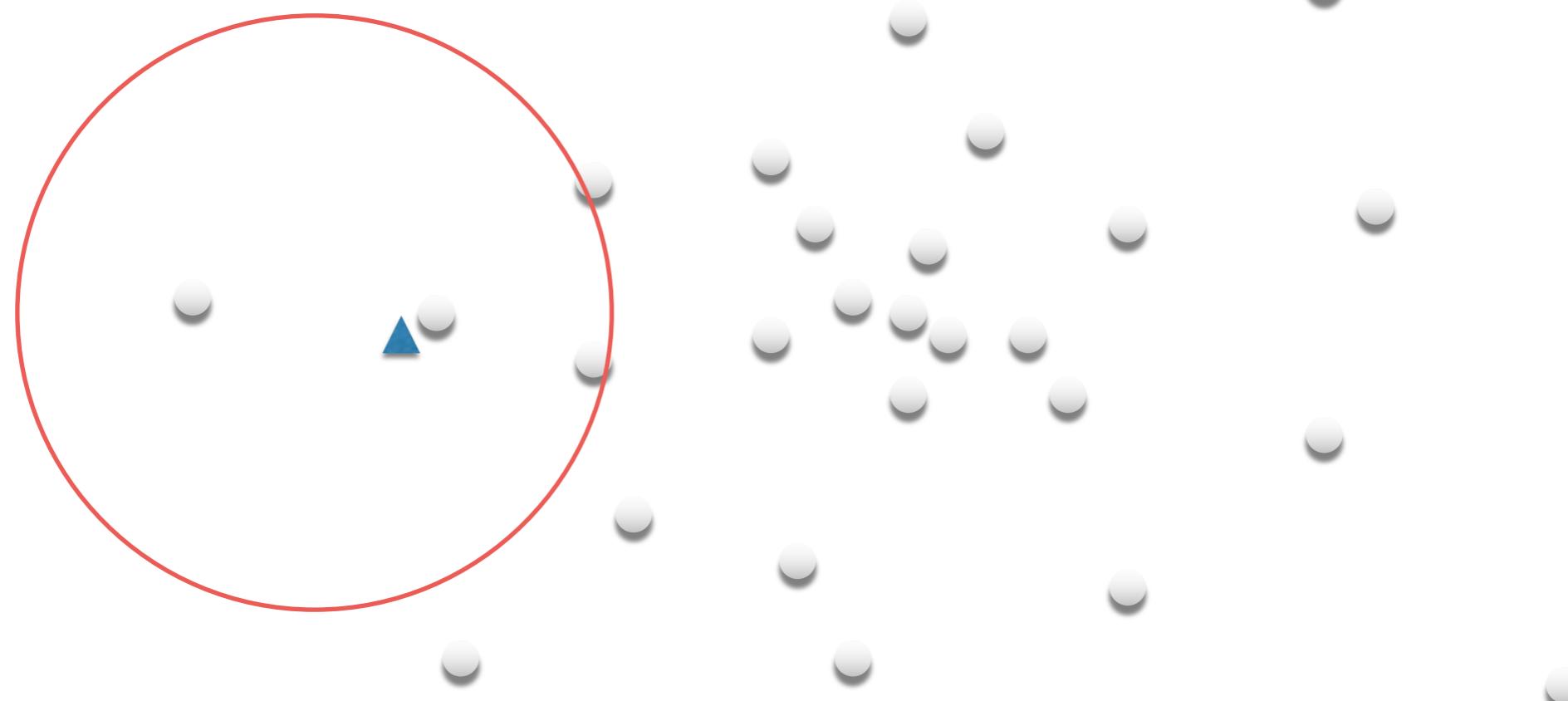


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

Compute the **mean**

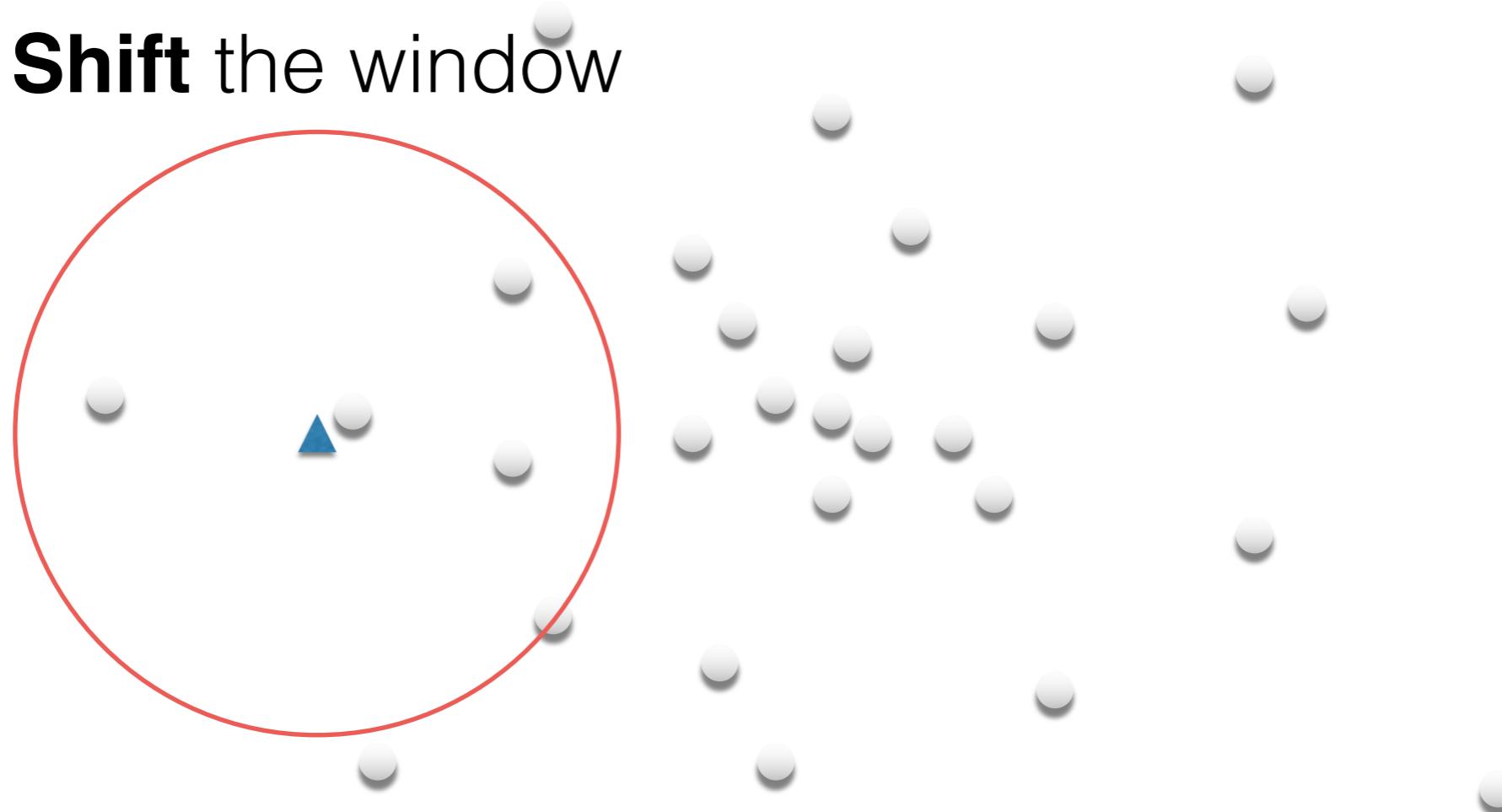


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

**Shift** the window

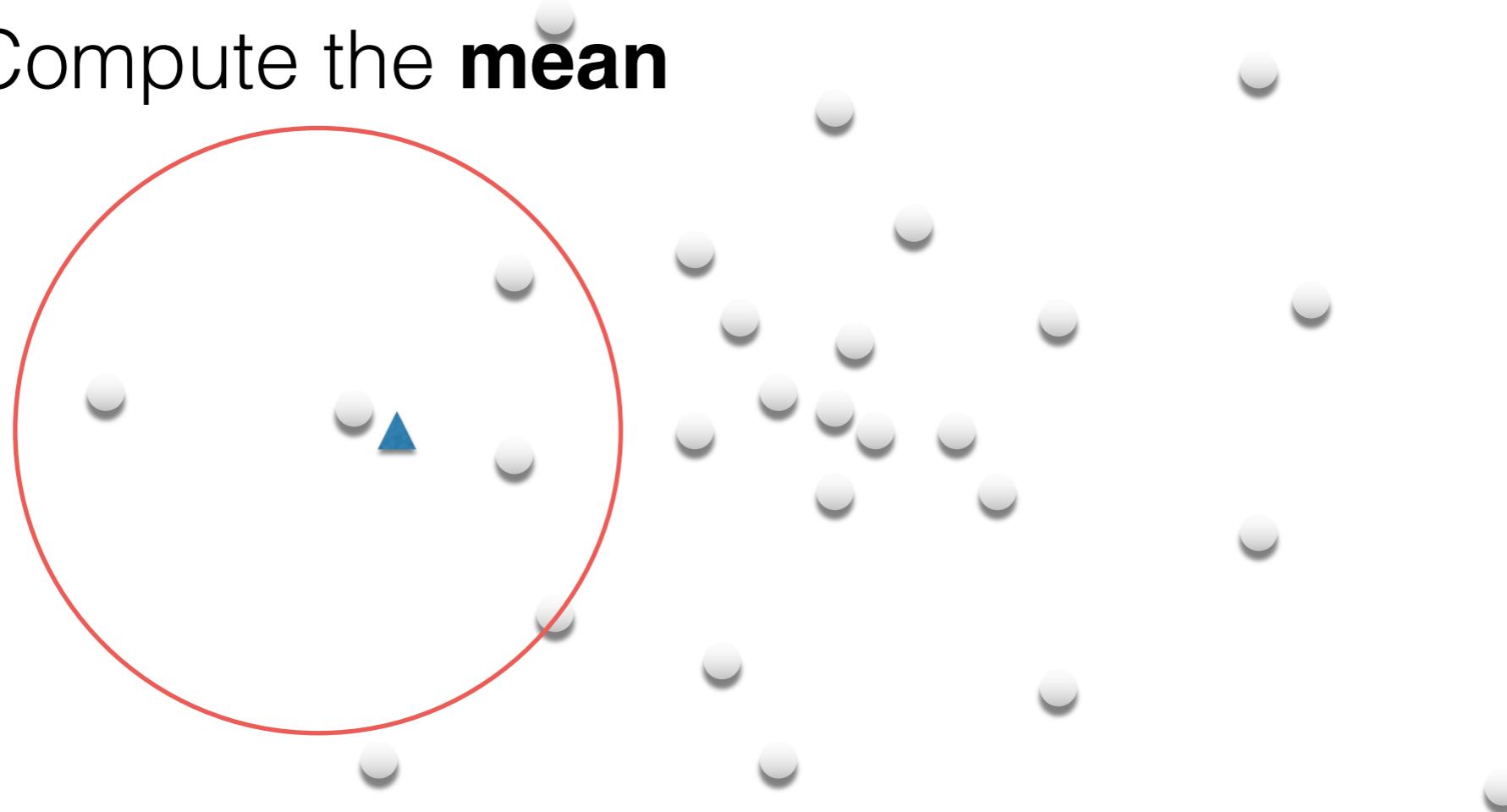


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

Compute the **mean**

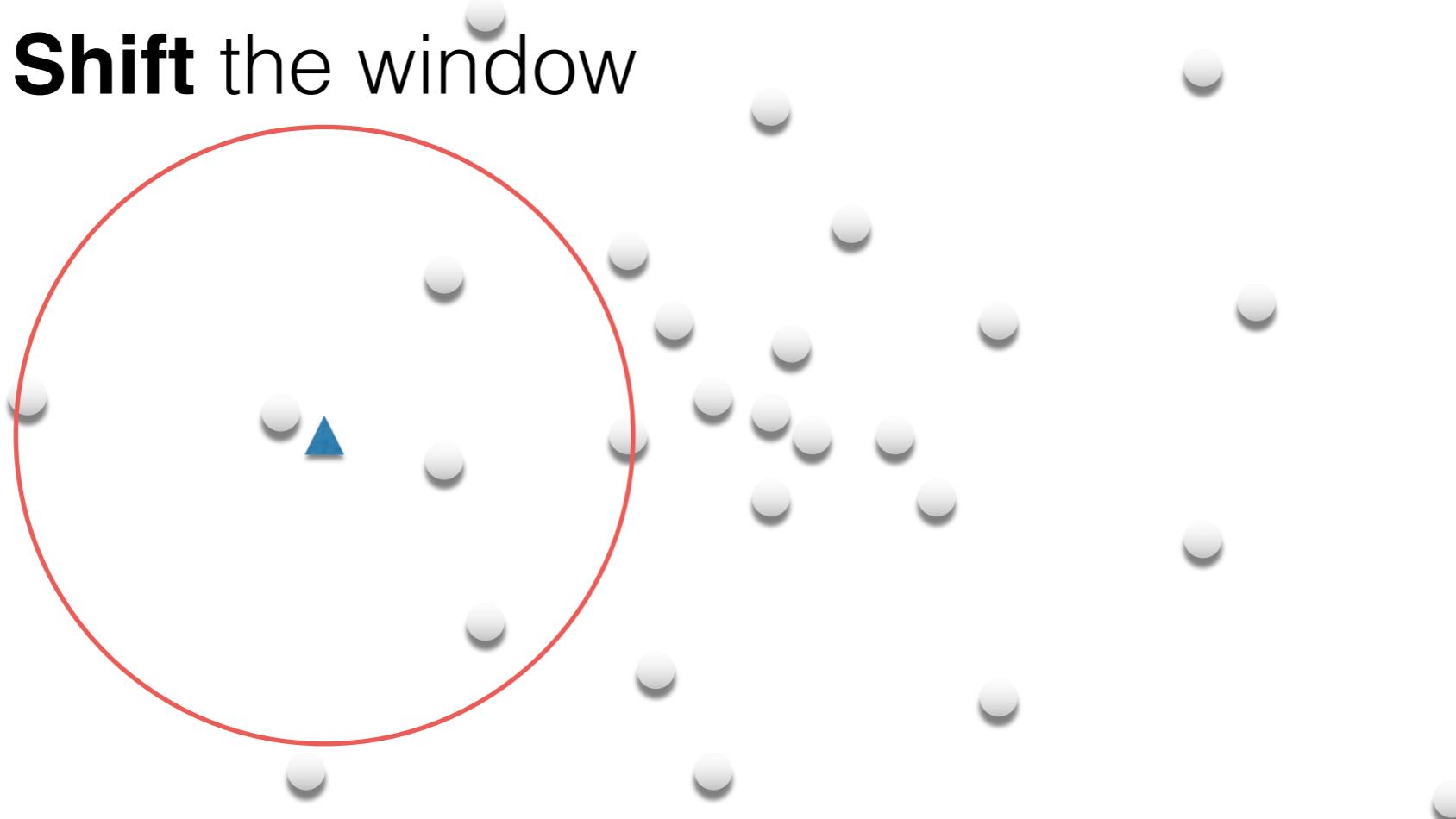


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

**Shift** the window

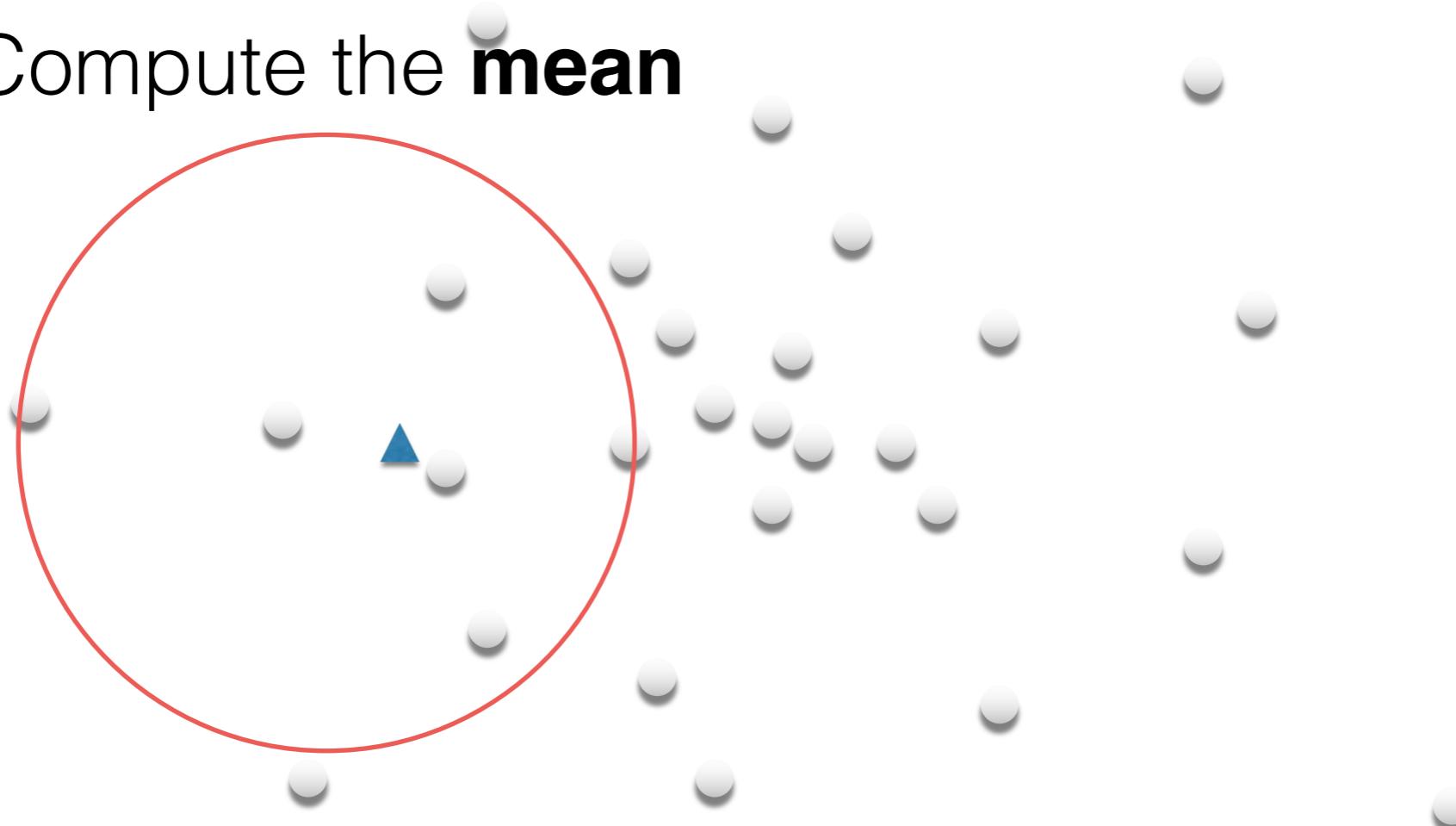


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

Compute the **mean**

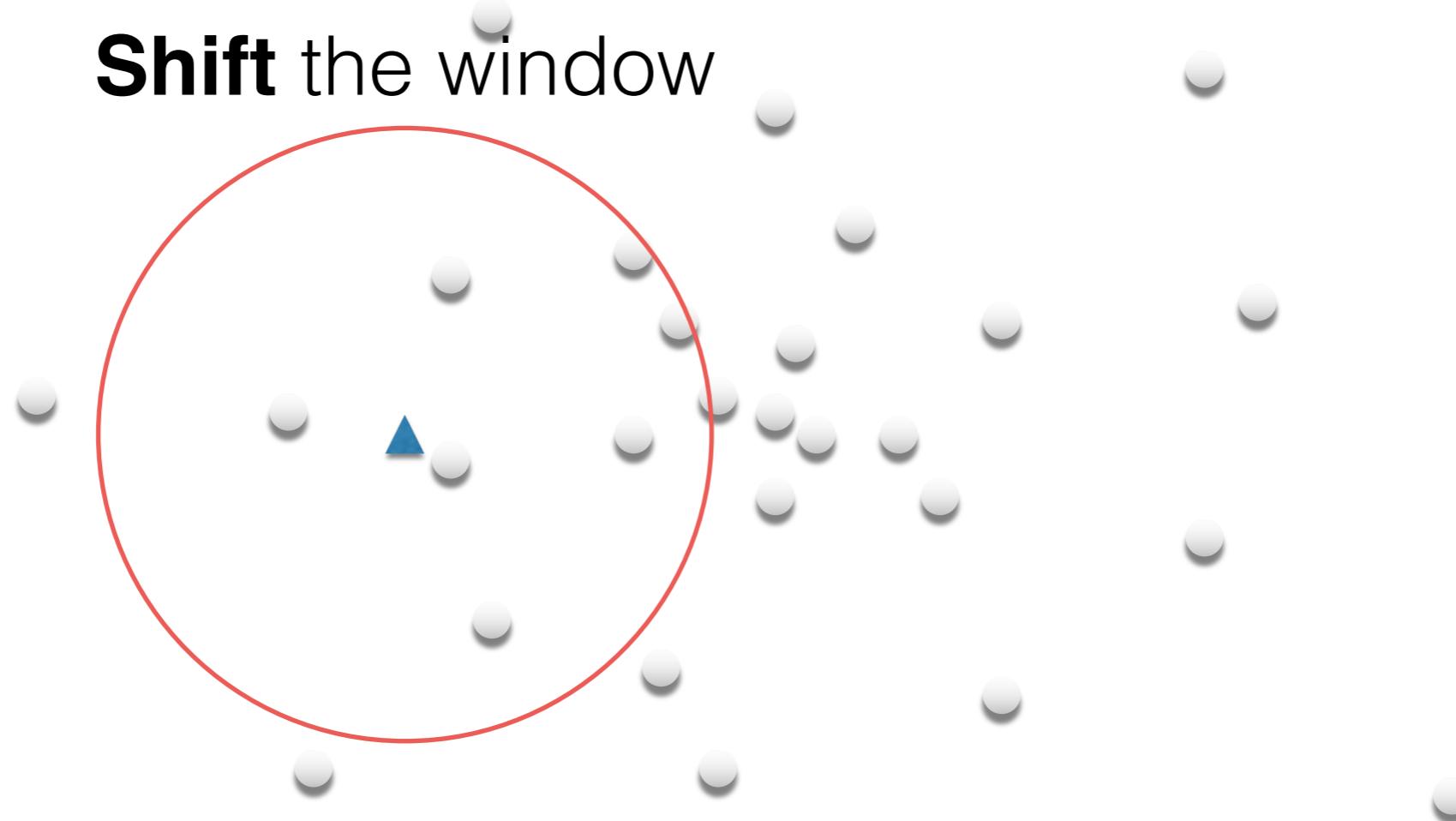


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

**Shift** the window

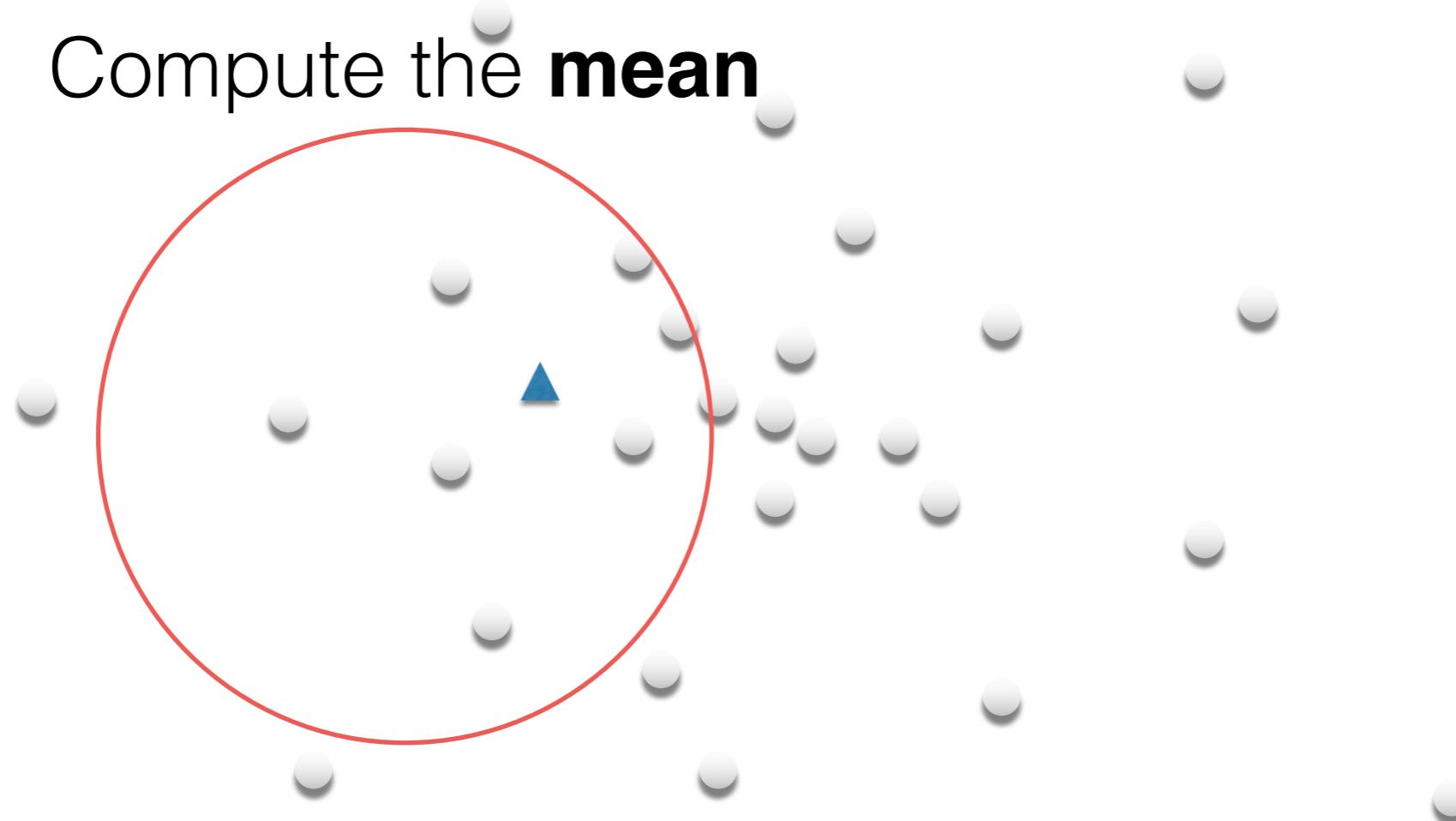


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

Compute the **mean**

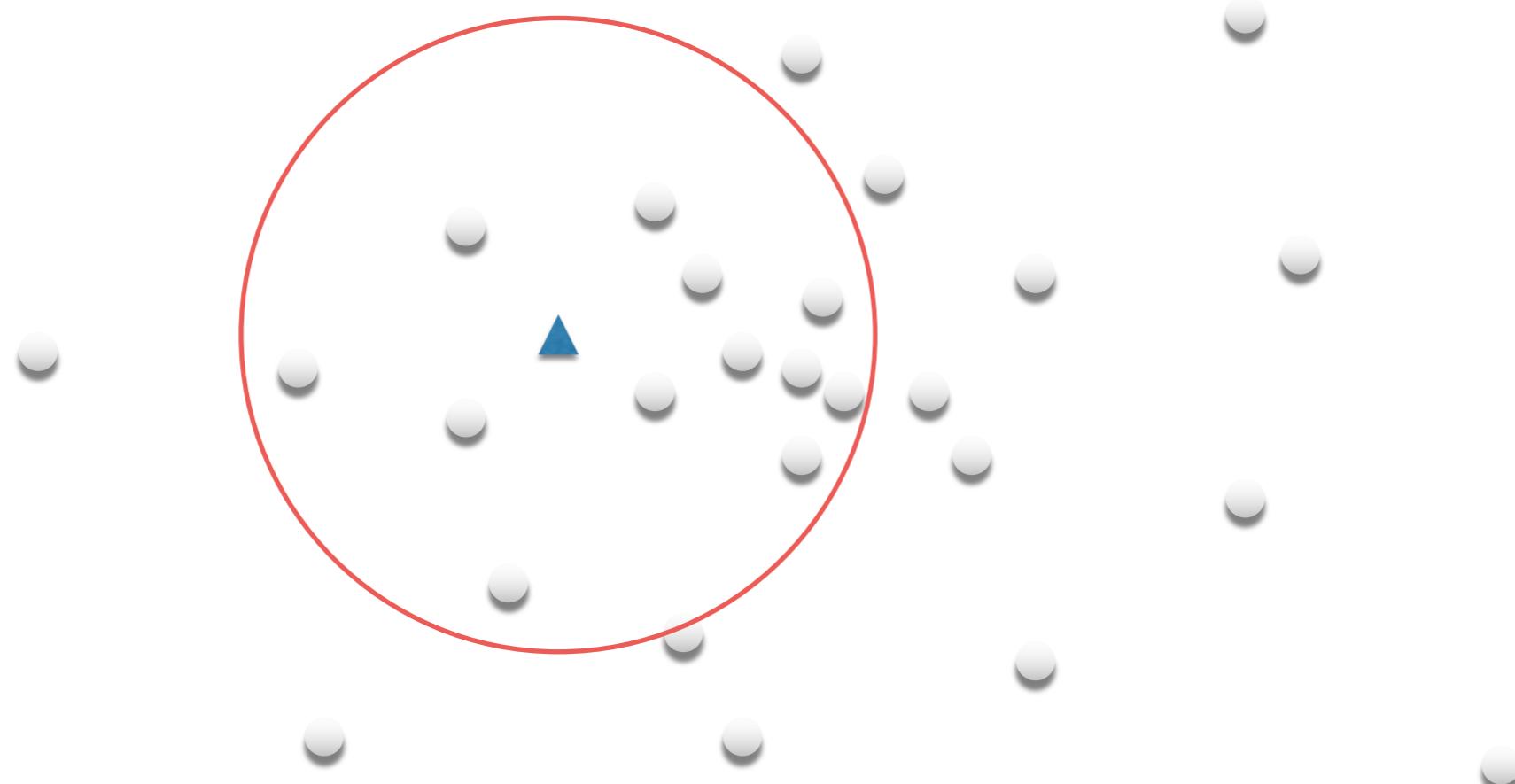


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

**Shift** the window

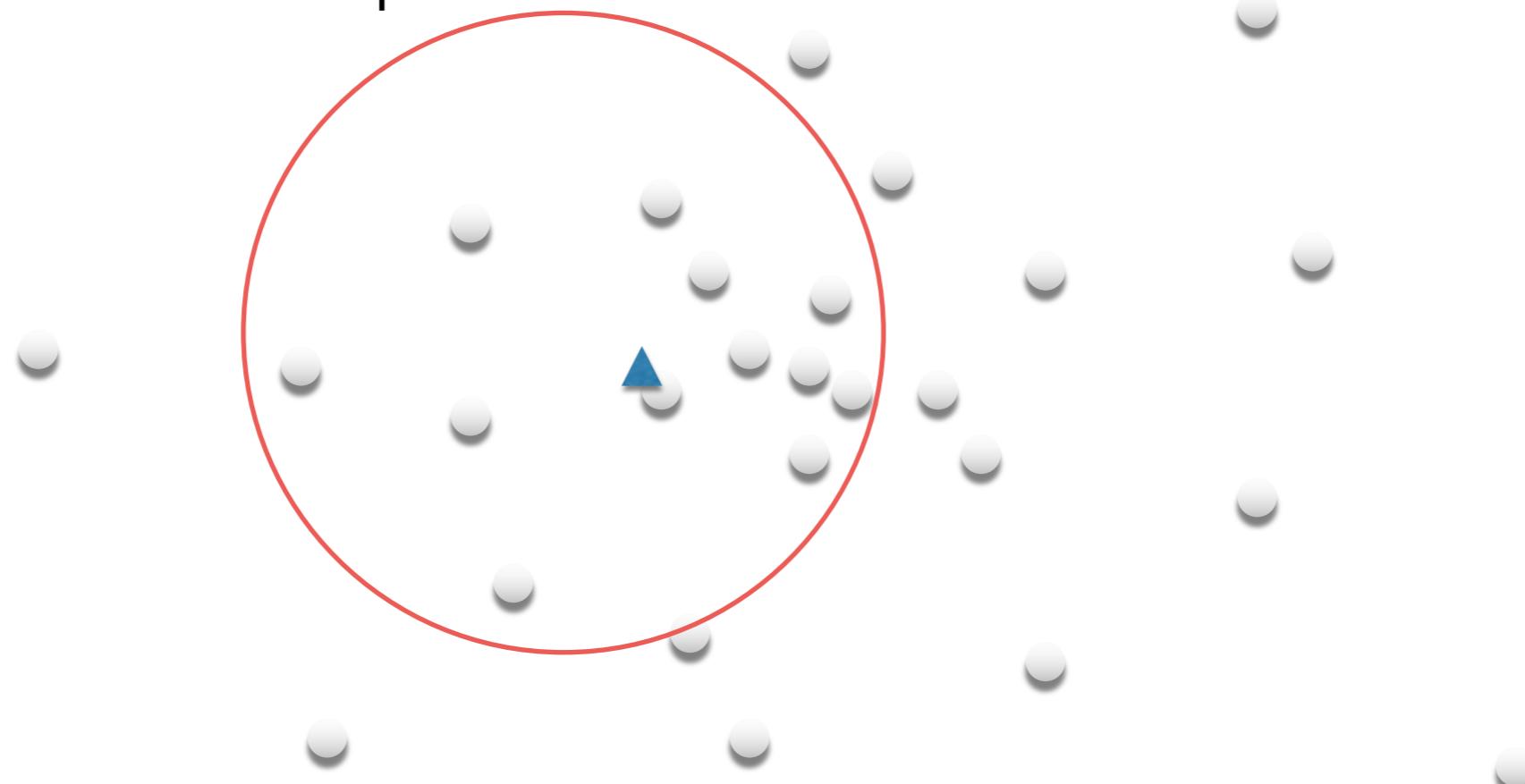


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

Compute the **mean**

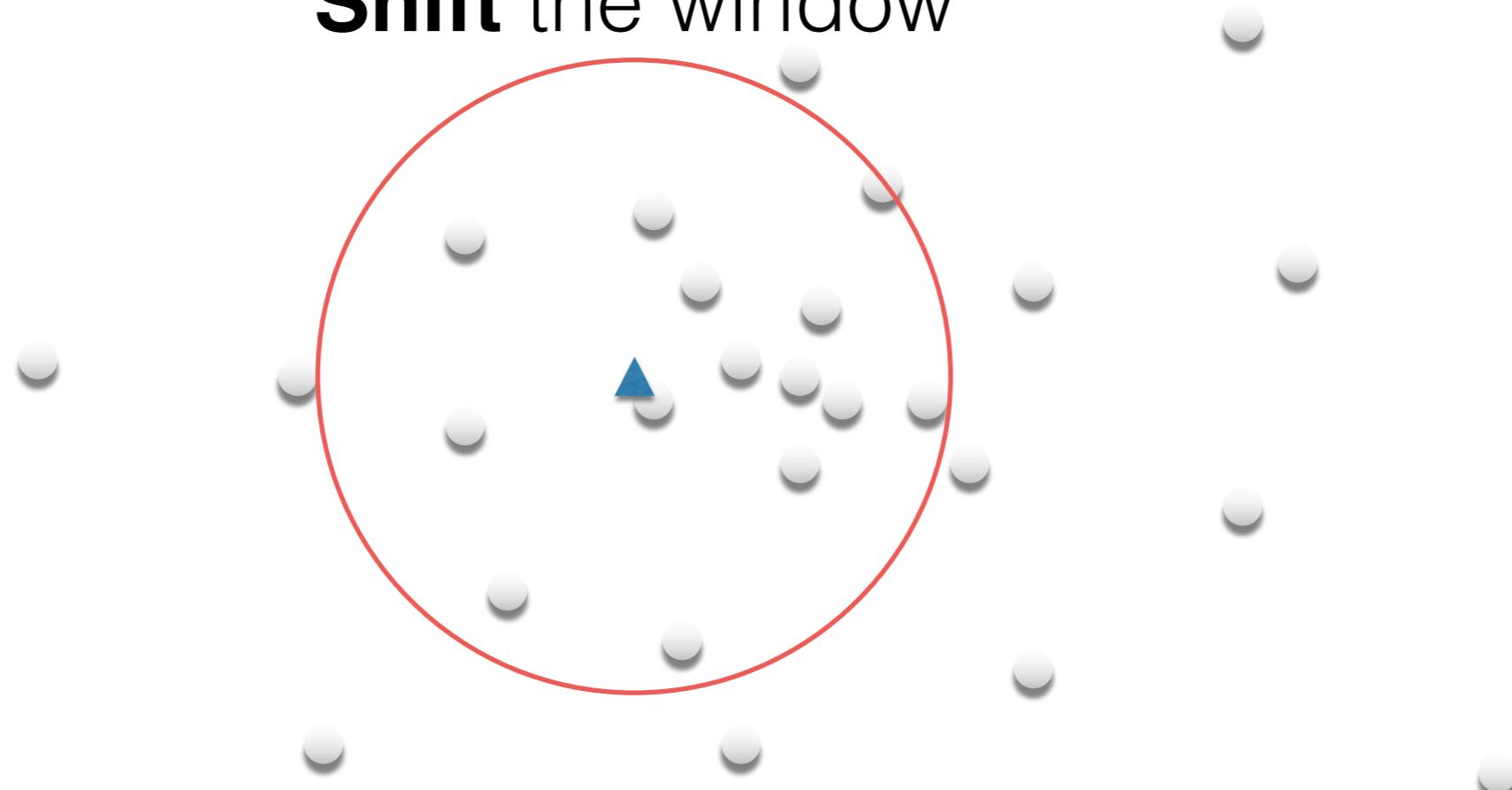


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

**Shift** the window

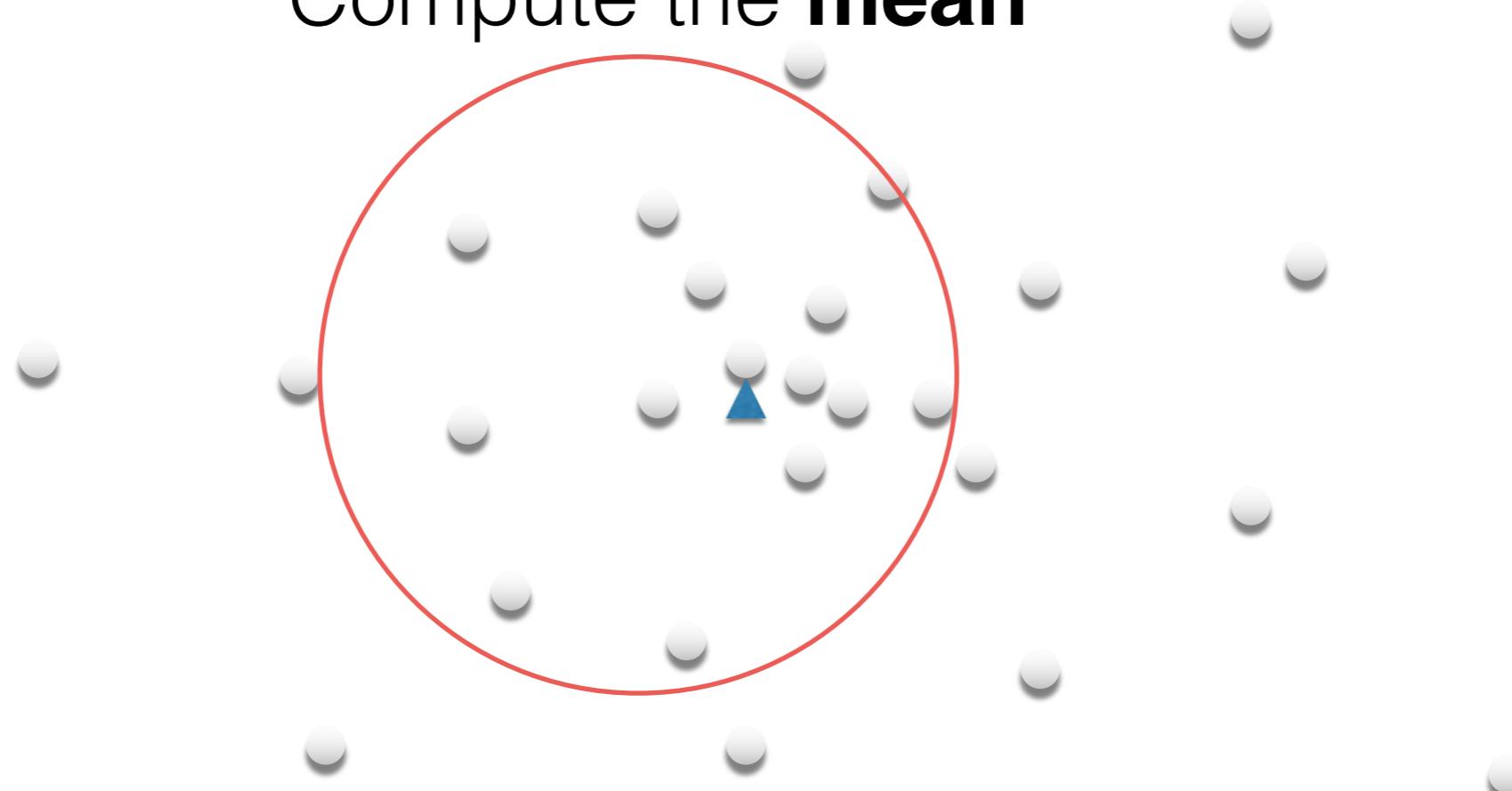


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

Compute the **mean**

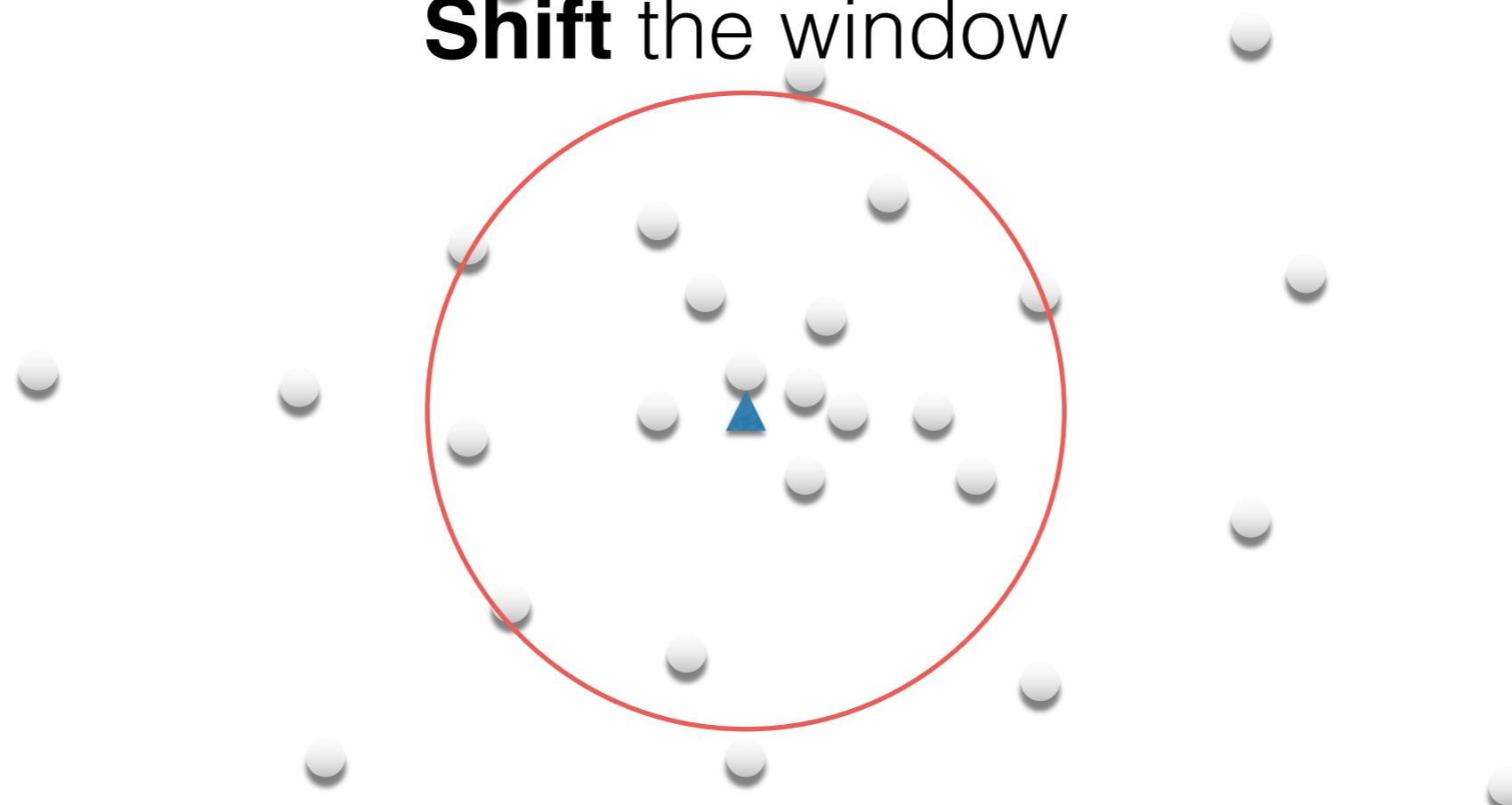


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

**Shift** the window

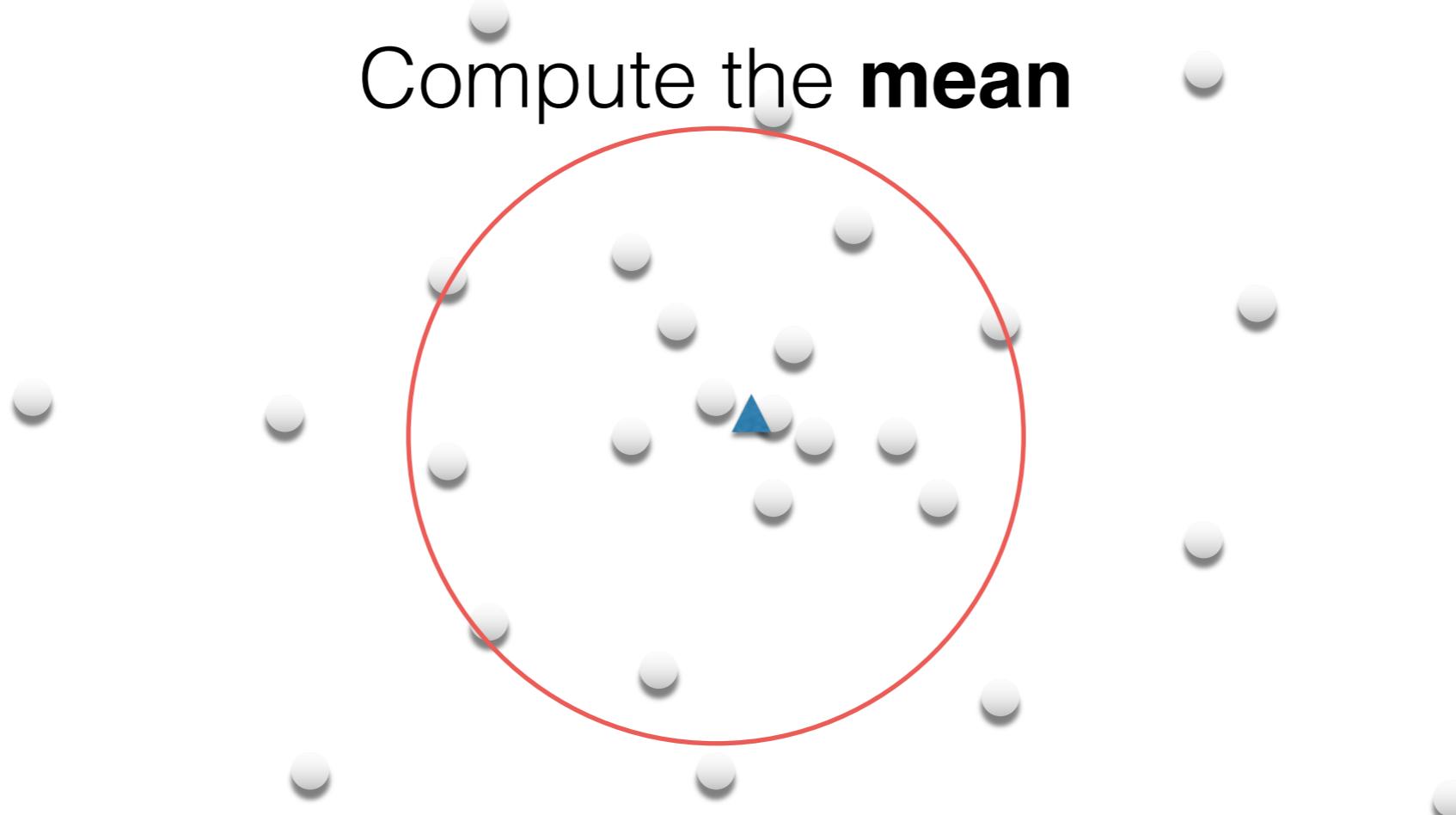


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

Compute the **mean**

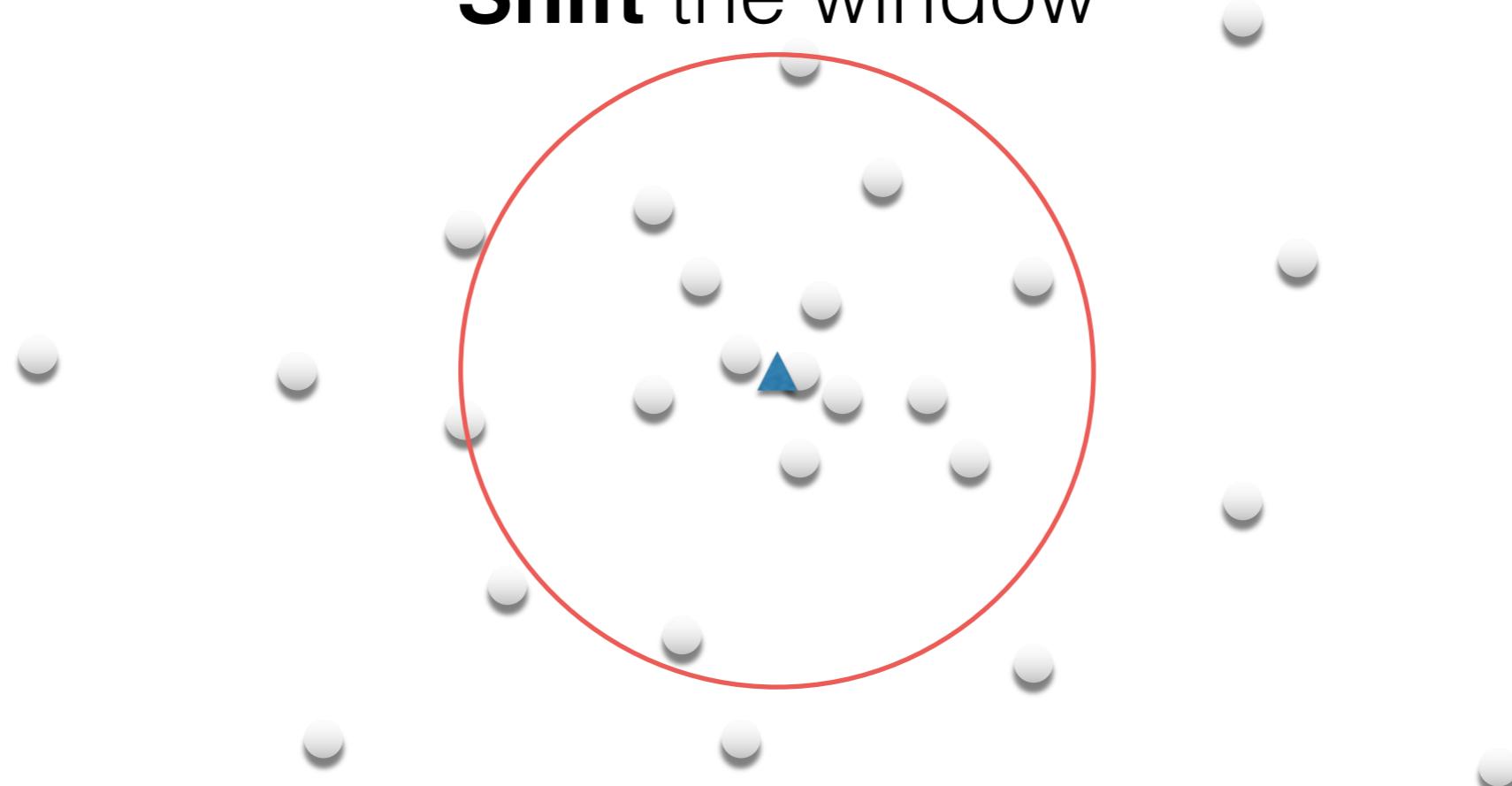


# Mean Shift Algorithm

A ‘mode seeking’ algorithm

Fukunaga & Hostetler (1975)

**Shift** the window



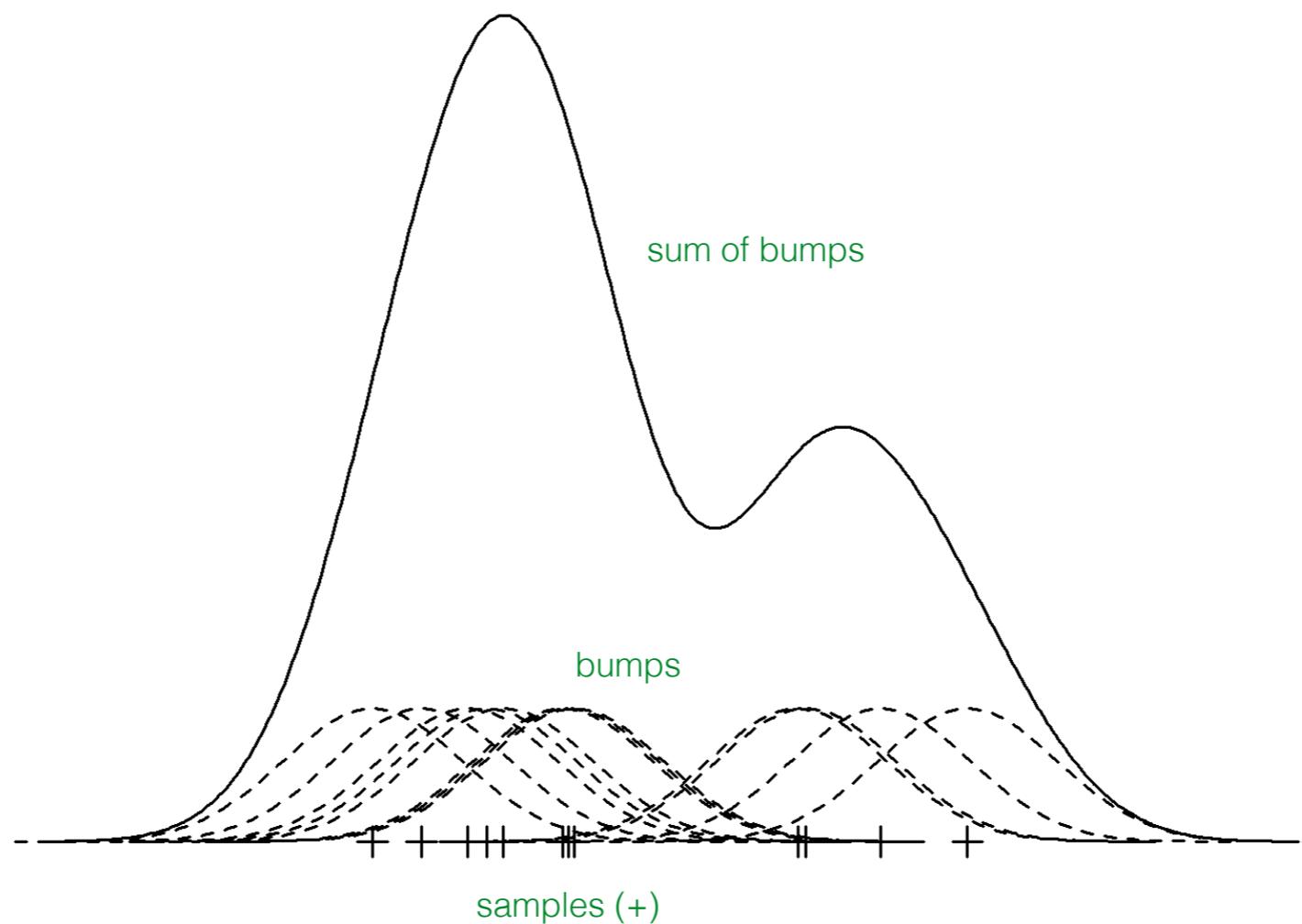
To understand the theory behind this we need to understand...

Kernel density  
estimation

To understand the mean shift algorithm ...

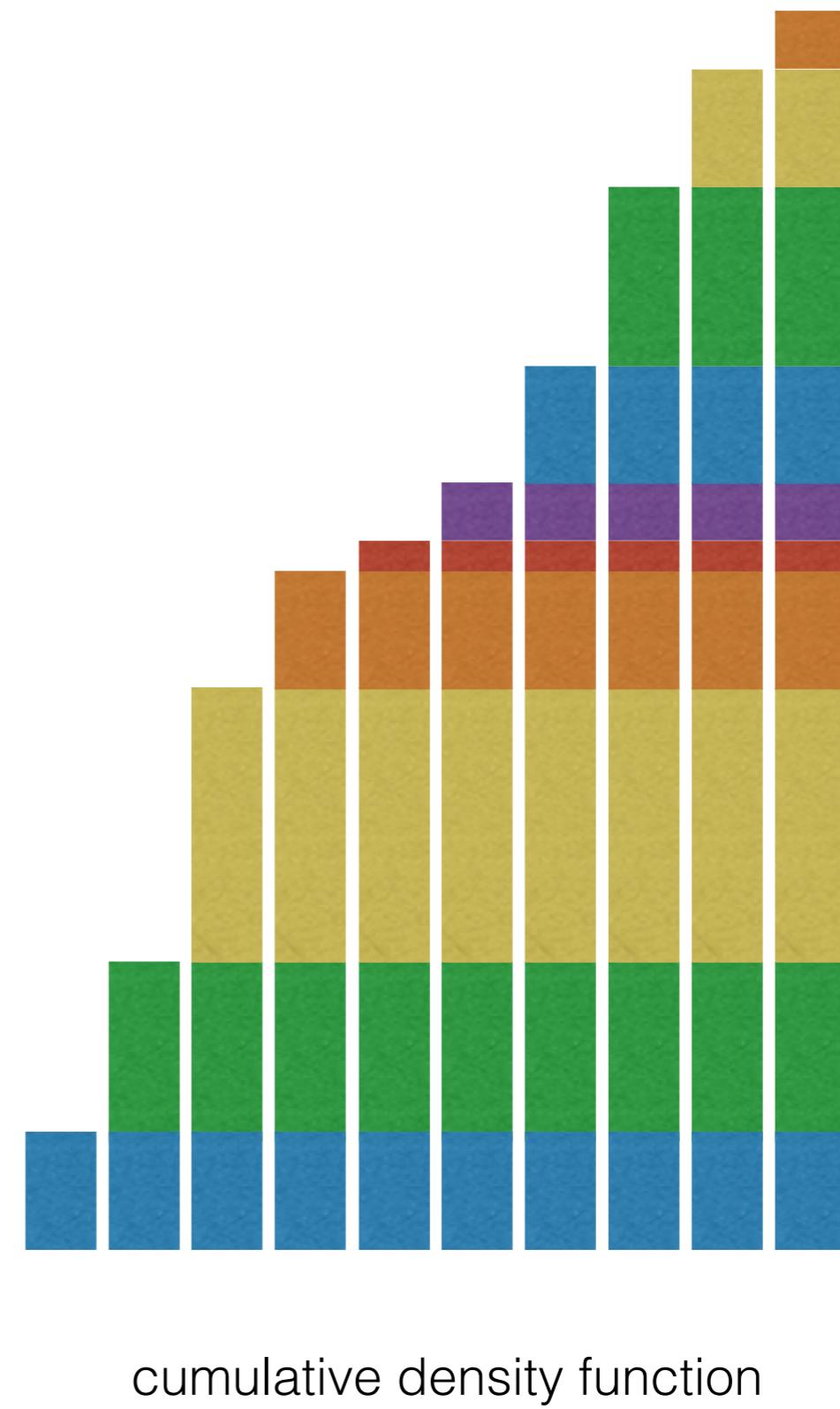
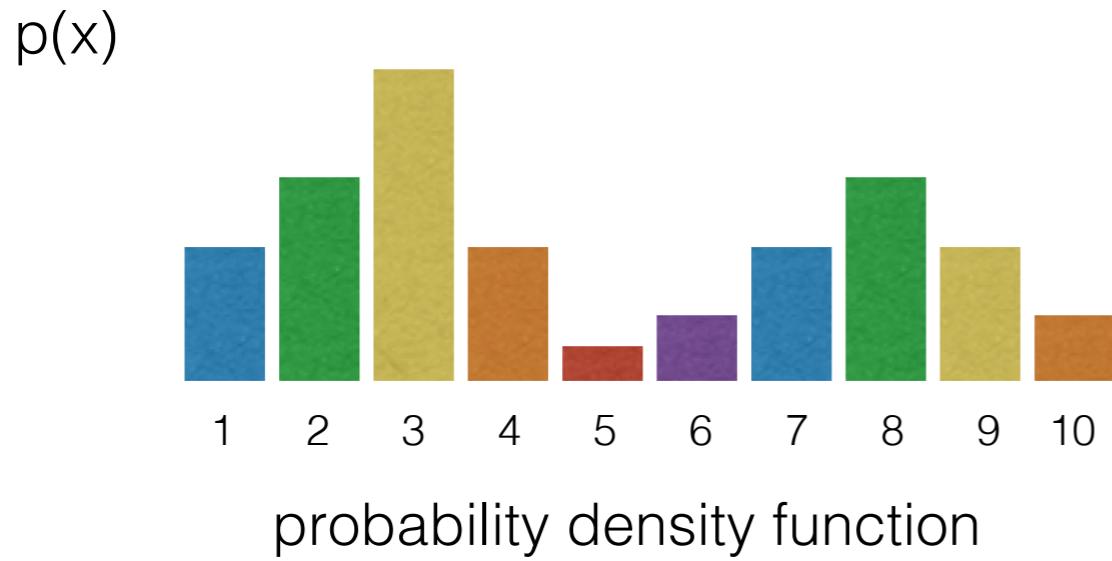
# Kernel Density Estimation

A method to approximate an underlying PDF from samples



Put 'bump' on every sample to approximate the PDF

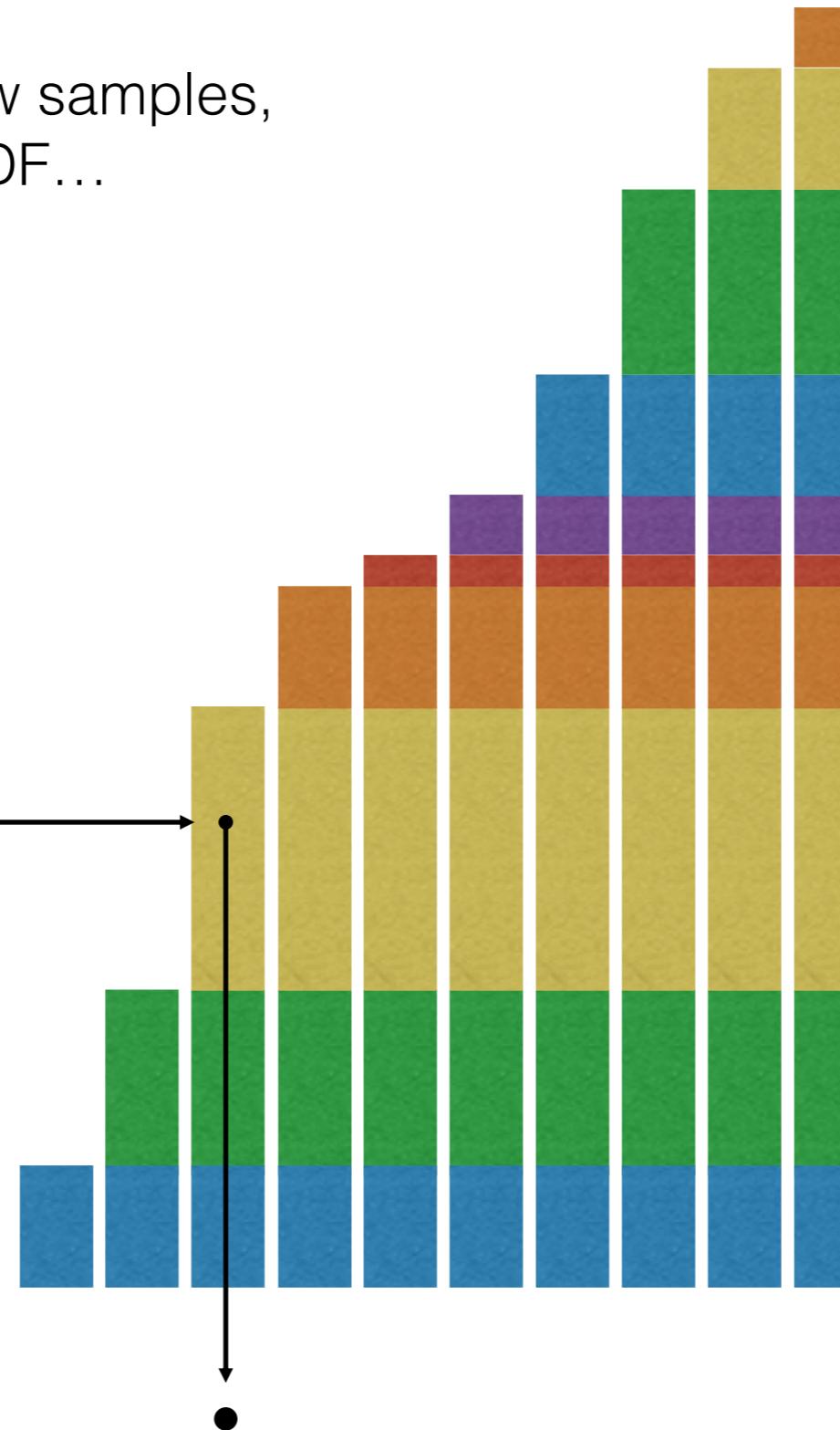
Say we have some hidden PDF...

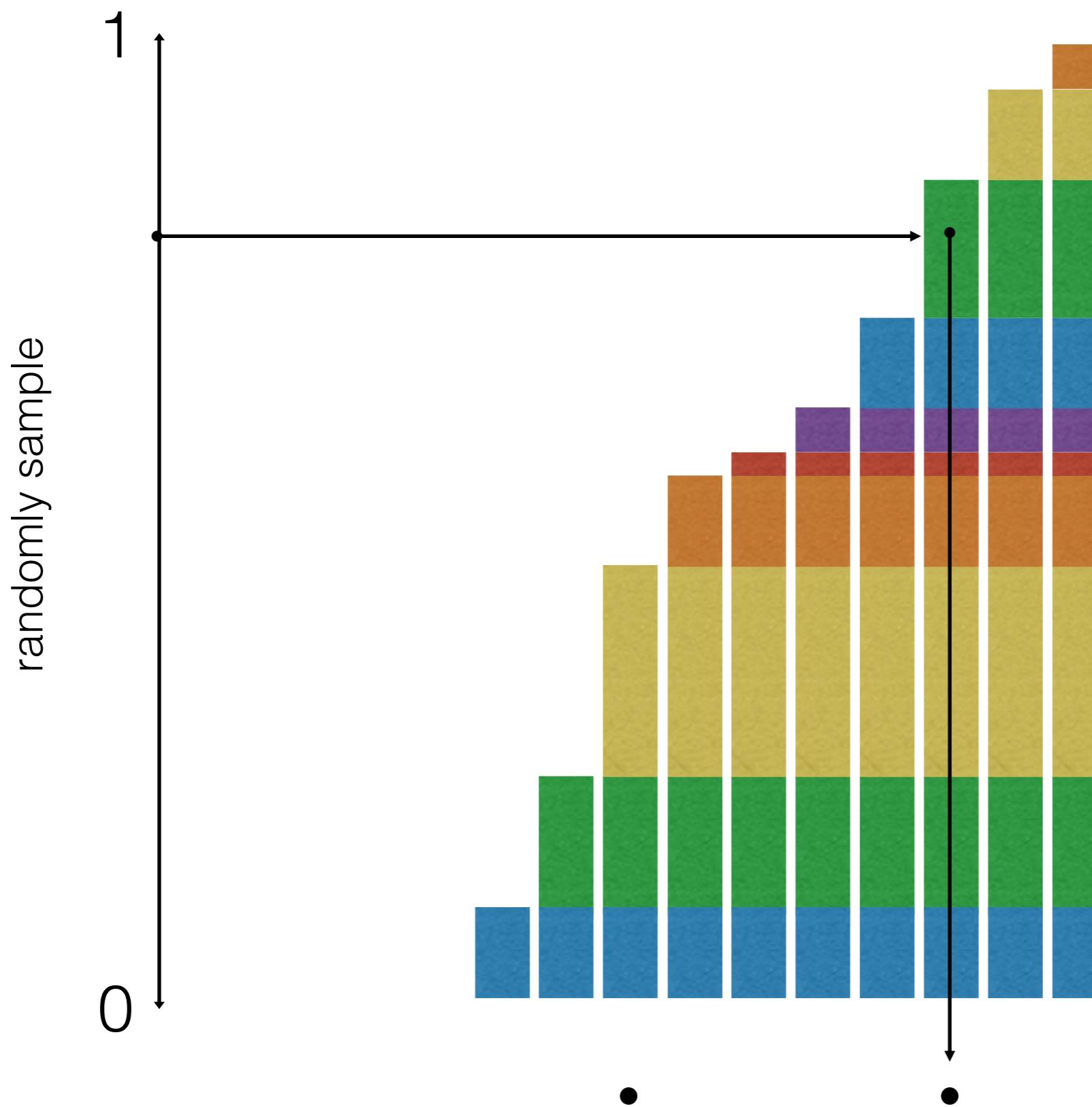


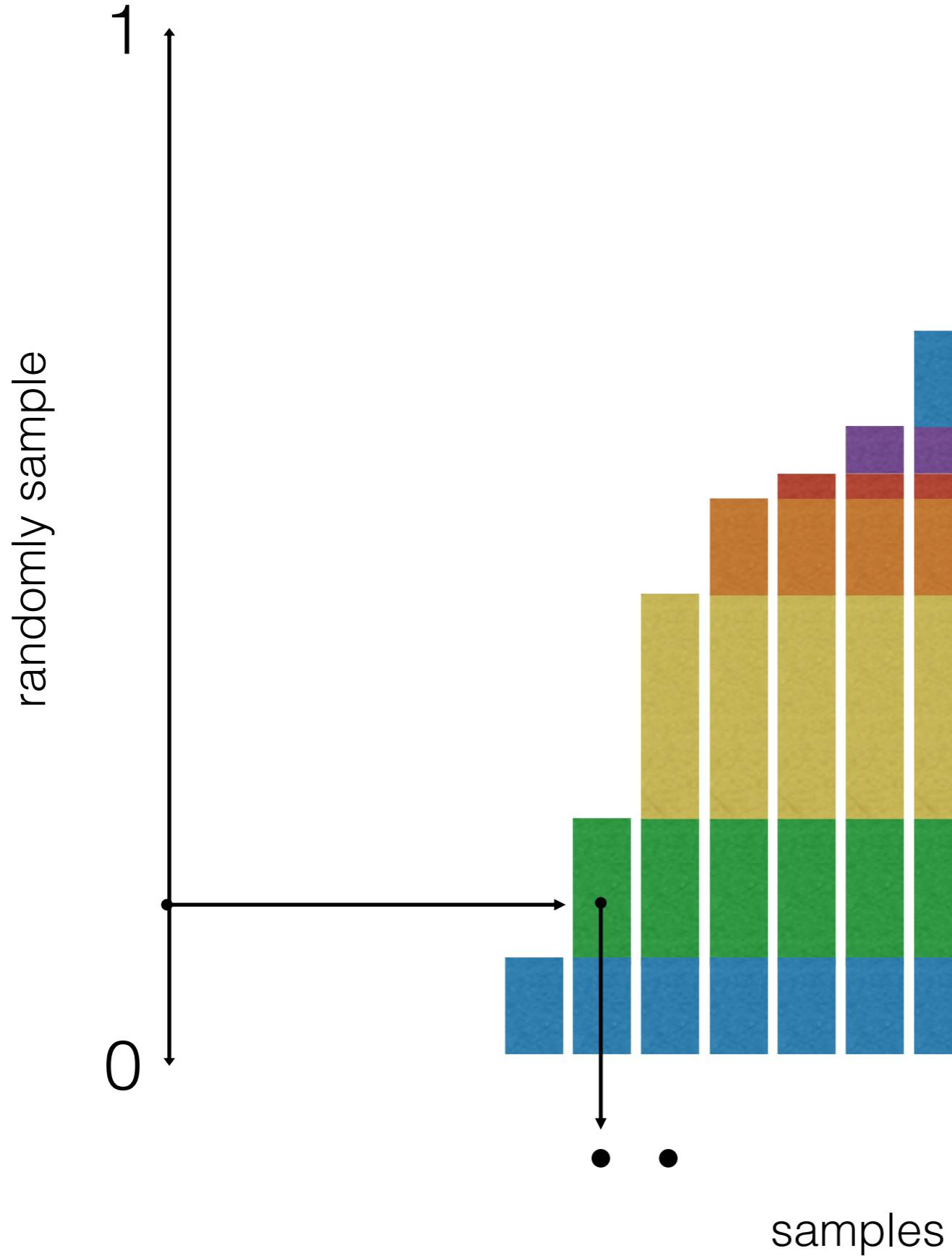
randomly sample

1

We can draw samples,  
using the CDF...



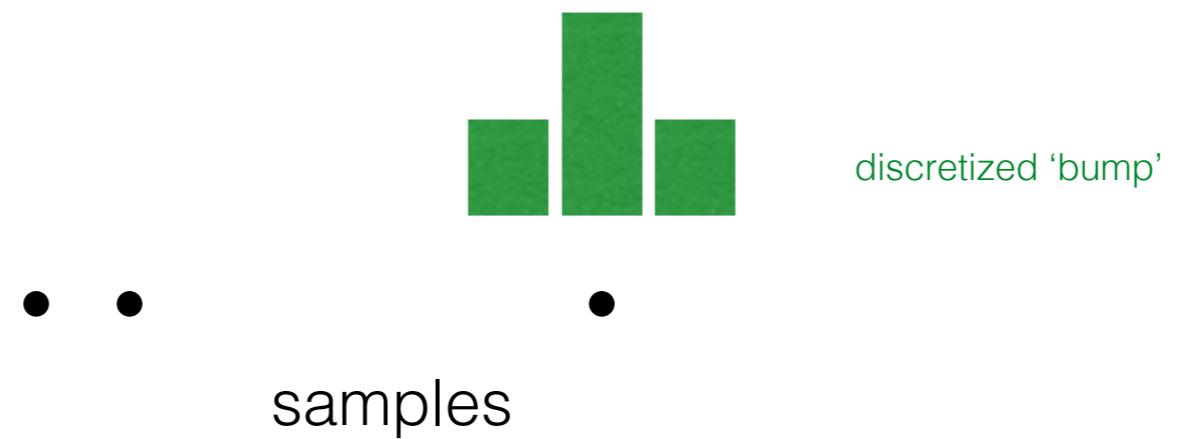


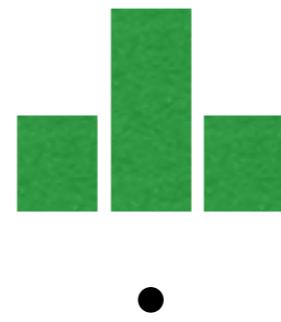
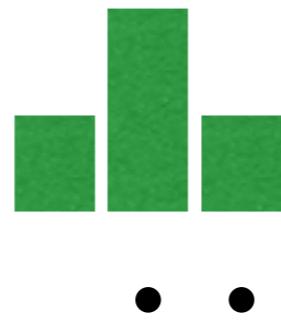


Now to estimate the ‘hidden’ PDF  
place Gaussian bumps on the samples...

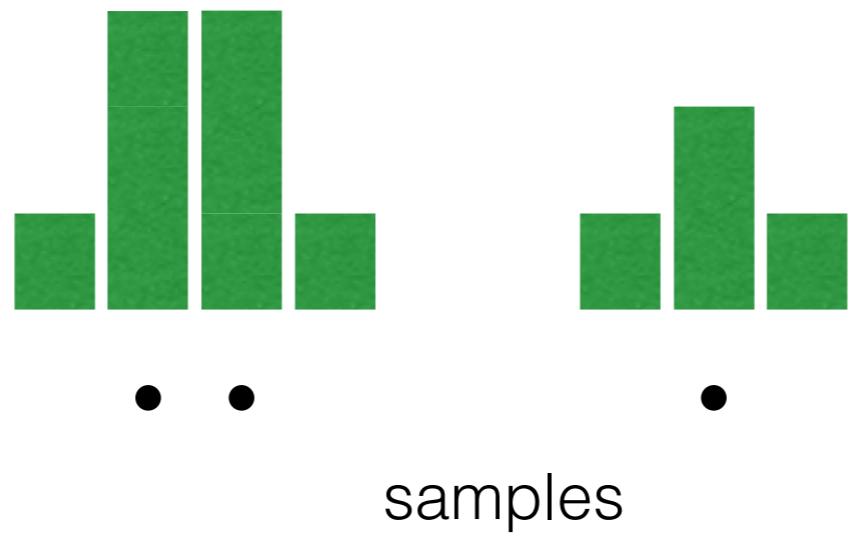
• • •

samples

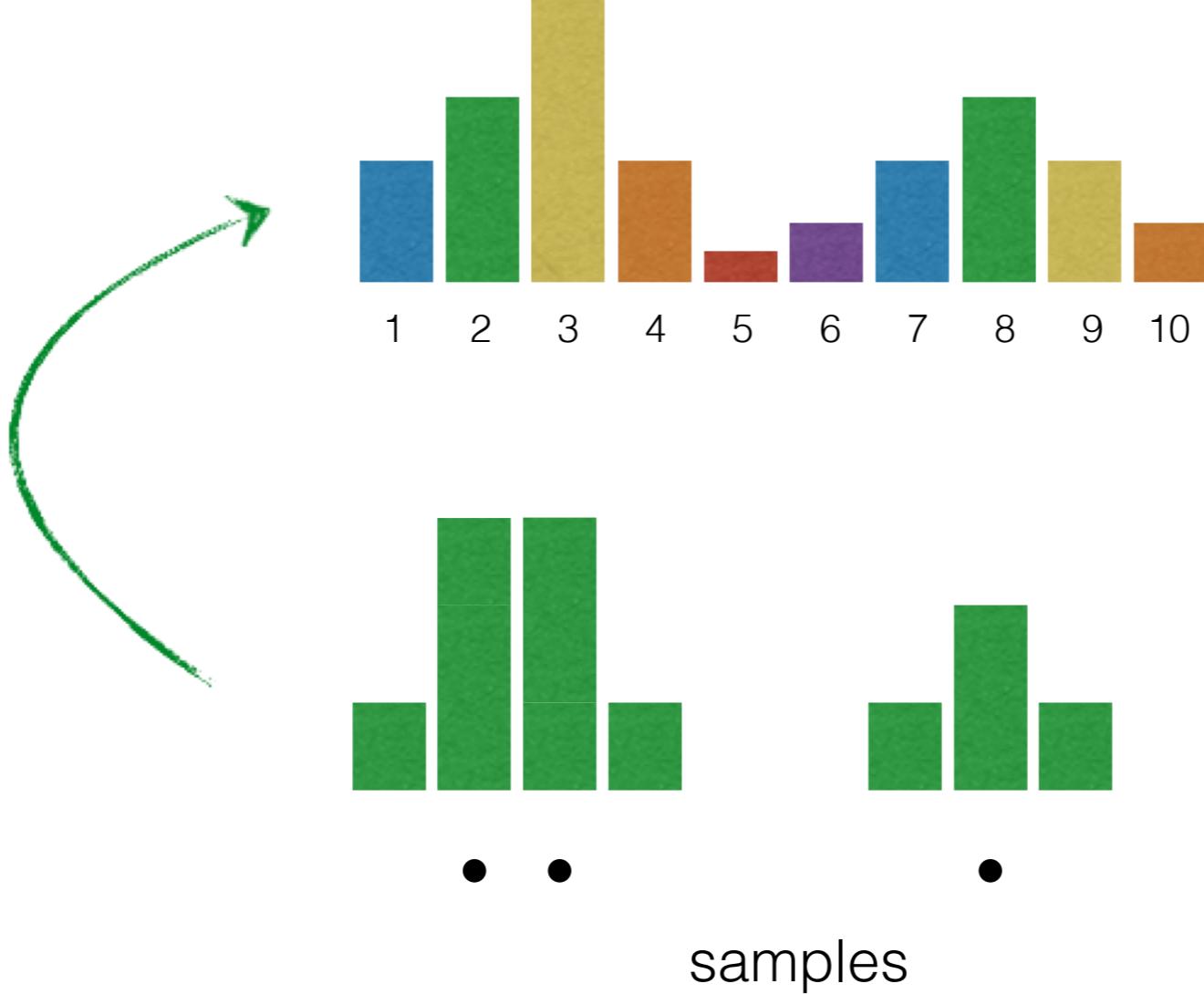




samples

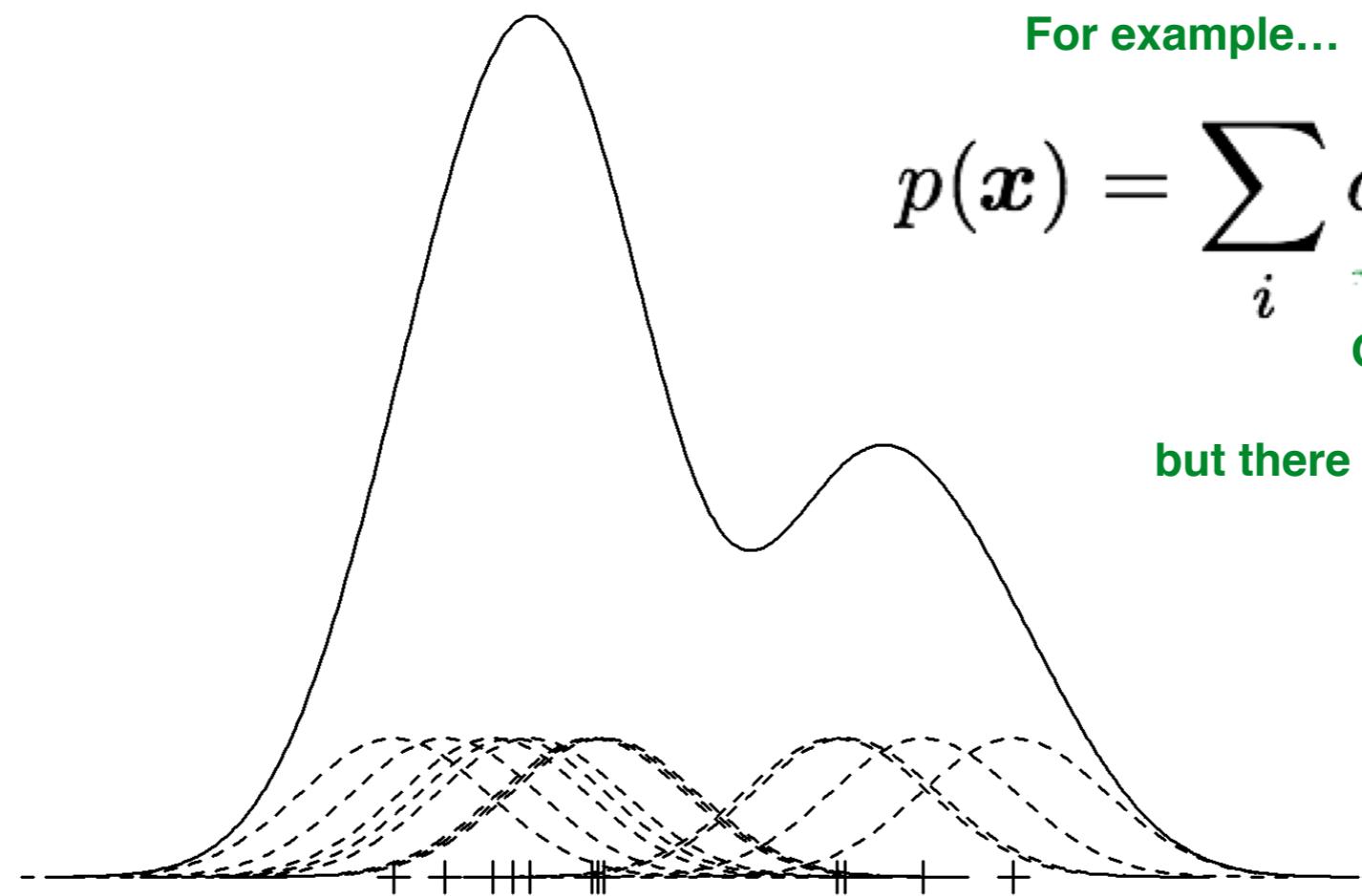


Kernel Density Estimate  
approximates the  
original PDF



# Kernel Density Estimation

Approximate the underlying PDF from samples from it



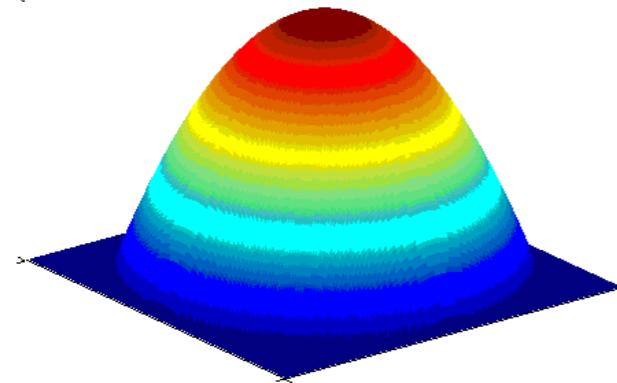
Put ‘bump’ on every sample to approximate the PDF

# Kernel Function

$$K(\mathbf{x}, \mathbf{x}')$$

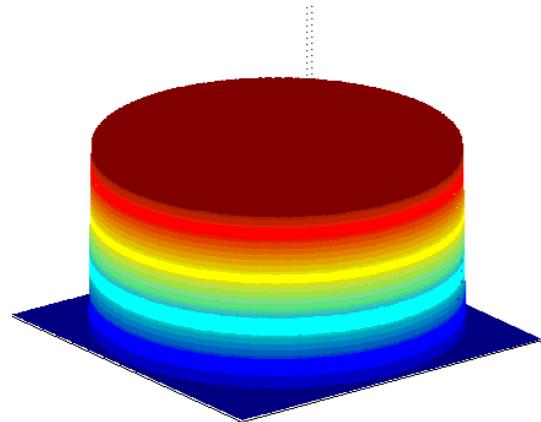
returns the ‘distance’ between two points

## Epanechnikov kernel



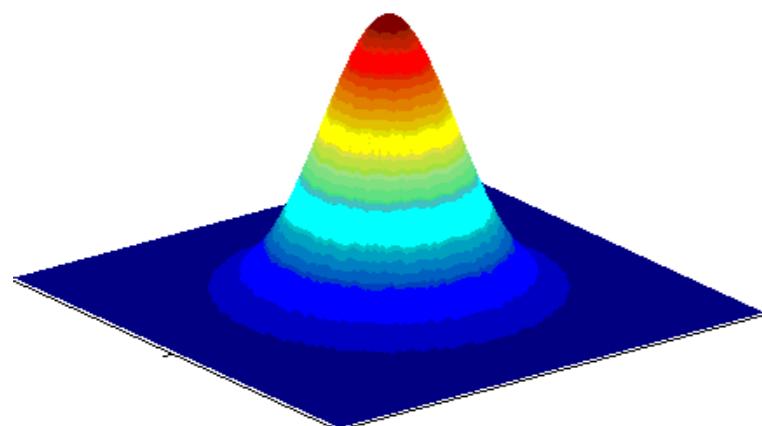
$$K(\mathbf{x}, \mathbf{x}') = \begin{cases} c(1 - \|\mathbf{x} - \mathbf{x}'\|^2) & \|\mathbf{x} - \mathbf{x}'\|^2 \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

## Uniform kernel



$$K(\mathbf{x}, \mathbf{x}') = \begin{cases} c & \|\mathbf{x} - \mathbf{x}'\|^2 \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

## Normal kernel



$$K(\mathbf{x}, \mathbf{x}') = c \exp\left(\frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|^2\right)$$

These are all radially symmetric kernels

# Radially symmetric kernels

...can be written in terms of its *profile*

$$K(\mathbf{x}, \mathbf{x}') = c \cdot k(\|\mathbf{x} - \mathbf{x}'\|^2)$$



# Connecting KDE and the Mean Shift Algorithm

# Mean-Shift Tracking

Given a set of points:

$$\{\mathbf{x}_s\}_{s=1}^S \quad \mathbf{x}_s \in \mathcal{R}^d$$

and a kernel:

$$K(\mathbf{x}, \mathbf{x}')$$

Find the mean sample point:

$$\mathbf{x}$$

# Mean-Shift Algorithm

Initialize  $\mathbf{x}$  place we start

While  $v(\mathbf{x}) > \epsilon$  shift values becomes really small

1. Compute mean-shift

$$m(\mathbf{x}) = \frac{\sum_s K(\mathbf{x}, \mathbf{x}_s) \mathbf{x}_s}{\sum_s K(\mathbf{x}, \mathbf{x}_s)}$$
compute the ‘mean’

$$v(\mathbf{x}) = m(\mathbf{x}) - \mathbf{x}$$
compute the ‘shift’

2. Update  $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{v}(\mathbf{x})$  update the point

*Where does this algorithm come from?*

# Mean-Shift Algorithm

Initialize  $\mathbf{x}$

While  $v(\mathbf{x}) > \epsilon$

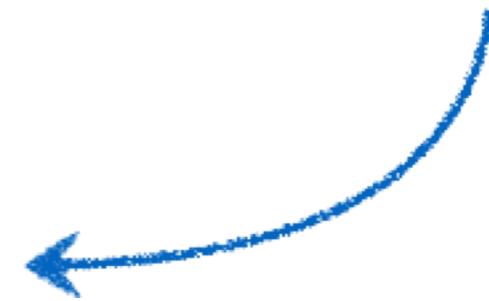
1. Compute mean-shift

$$m(\mathbf{x}) = \frac{\sum_s K(\mathbf{x}, \mathbf{x}_s) \mathbf{x}_s}{\sum_s K(\mathbf{x}, \mathbf{x}_s)}$$

$$v(\mathbf{x}) = m(\mathbf{x}) - \mathbf{x}$$

2. Update  $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{v}(\mathbf{x})$

*Where does this come from?*



*Where does this algorithm come from?*

## *How is the KDE related to the mean shift algorithm?*

**Recall:**

Kernel density estimate

(radially symmetric kernels)

$$P(\mathbf{x}) = \frac{1}{N} c \sum_n k(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

can compute probability for any point using the KDE!

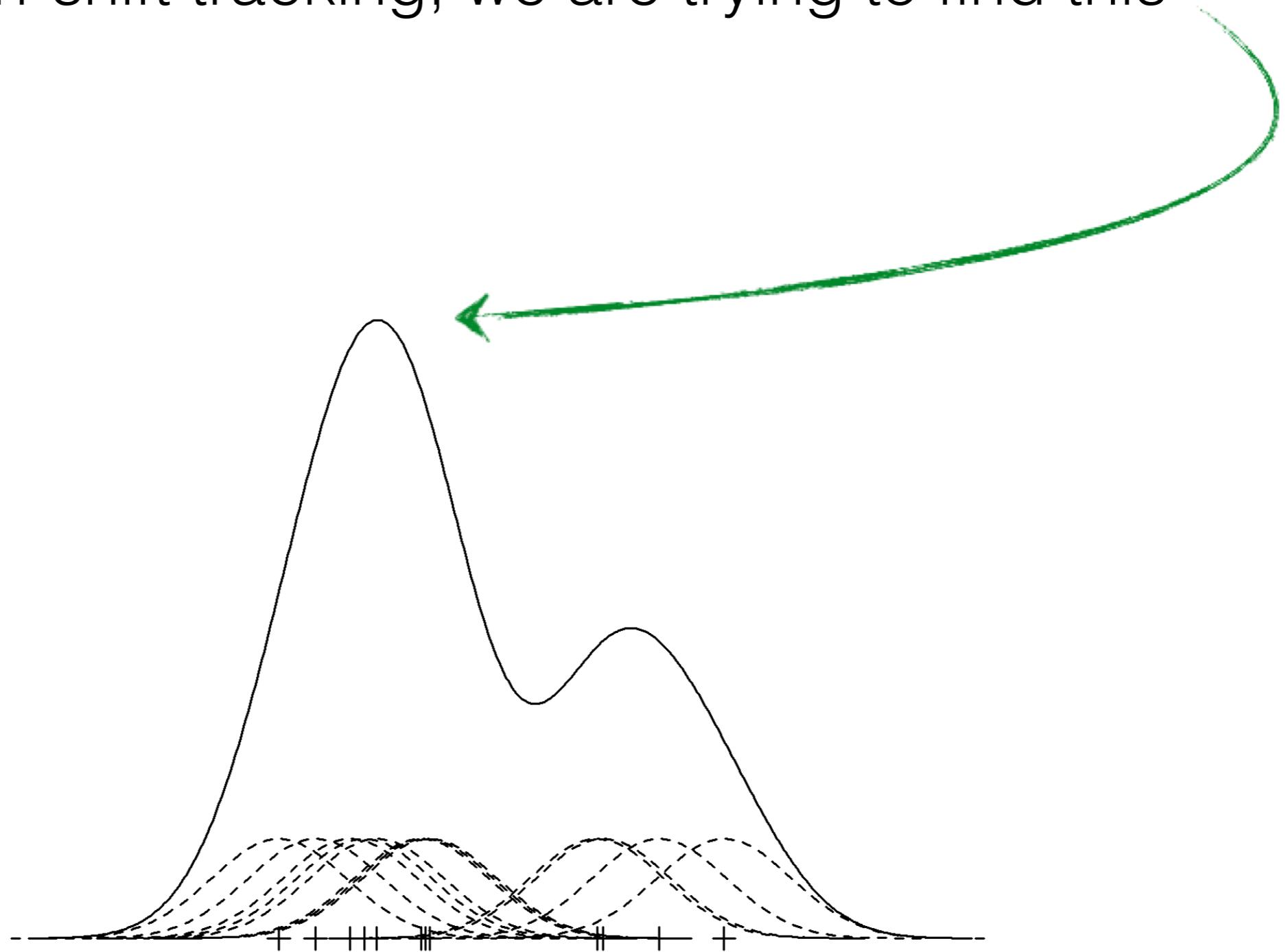
**We can show that:**

Gradient of the PDF is related to the mean shift vector

$$\nabla P(\mathbf{x}) \propto m(\mathbf{x})$$

The mean shift vector is a ‘step’ in the direction of the gradient of the KDE  
mean-shift algorithm is maximizing the objective function

In mean-shift tracking, we are trying to find this



which means we are trying to...

We are trying to optimize this:

$$\mathbf{x} = \arg \max_{\mathbf{x}} P(\mathbf{x}) \quad \text{find the solution that has the highest probability}$$

$$= \arg \max_{\mathbf{x}} \frac{1}{N} c \sum_n k(||\mathbf{x} - \mathbf{x}_n||^2)$$

↑  
usually non-linear      ↑  
non-parametric

*How do we optimize this non-linear function?*

We are trying to optimize this:

$$\begin{aligned}\mathbf{x} &= \arg \max_{\mathbf{x}} P(\mathbf{x}) \\ &= \arg \max_{\mathbf{x}} \frac{1}{N} c \sum_n k(||\mathbf{x} - \mathbf{x}_n||^2)\end{aligned}$$

↑  
usually non-linear      ↑  
non-parametric

*How do we optimize this non-linear function?*

compute partial derivatives ... **gradient descent!**

$$P(\mathbf{x}) = \frac{1}{N} c \sum_n k(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

Compute the gradient

$$P(\mathbf{x}) = \frac{1}{N} c \sum_n k(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

Gradient       $\nabla P(\mathbf{x}) = \frac{1}{N} c \sum_n \nabla k(\|\mathbf{x} - \mathbf{x}_n\|^2)$

Expand the gradient (algebra)

$$P(\mathbf{x}) = \frac{1}{N}c \sum_n k(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

Gradient

$$\nabla P(\mathbf{x}) = \frac{1}{N}c \sum_n \nabla k(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

Expand gradient

$$\nabla P(\mathbf{x}) = \frac{1}{N}2c \sum_n (\mathbf{x} - \mathbf{x}_n)k'(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

$$P(\mathbf{x}) = \frac{1}{N}c \sum_n k(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

Gradient

$$\nabla P(\mathbf{x}) = \frac{1}{N}c \sum_n \nabla k(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

Expand gradient

$$\nabla P(\mathbf{x}) = \frac{1}{N}2c \sum_n (\mathbf{x} - \mathbf{x}_n)k'(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

Call the gradient of the kernel function g

$$k'(\cdot) = -g(\cdot)$$

$$P(\mathbf{x}) = \frac{1}{N} c \sum_n k(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

Gradient

$$\nabla P(\mathbf{x}) = \frac{1}{N} c \sum_n \nabla k(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

Expand gradient

$$\nabla P(\mathbf{x}) = \frac{1}{N} 2c \sum_n (\mathbf{x} - \mathbf{x}_n) k'(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

change of notation  
(kernel-shadow pairs)

$$\nabla P(\mathbf{x}) = \frac{1}{N} 2c \sum_n (\mathbf{x}_n - \mathbf{x}) g(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

keep this in memory:  $k'(\cdot) = -g(\cdot)$

$$\nabla P(\mathbf{x}) = \frac{1}{N} 2c \sum_n (\mathbf{x}_n - \mathbf{x}) g(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

multiply it out

$$\nabla P(\mathbf{x}) = \frac{1}{N} 2c \sum_n \mathbf{x}_n g(\|\mathbf{x} - \mathbf{x}_n\|^2) - \frac{1}{N} 2c \sum_n \mathbf{x} g(\|\mathbf{x} - \mathbf{x}_n\|^2)$$

too long!  
(use short hand notation)

$$\nabla P(\mathbf{x}) = \frac{1}{N} 2c \sum_n \mathbf{x}_n g_n - \frac{1}{N} 2c \sum_n \mathbf{x} g_n$$

$$\nabla P(\mathbf{x}) = \frac{1}{N} 2c \sum_n \mathbf{x}_n g_n - \frac{1}{N} 2c \sum_n \mathbf{x} g_n$$

multiply by one!

$$\nabla P(\mathbf{x}) = \frac{1}{N} 2c \sum_n \mathbf{x}_n g_n \left( \frac{\sum_n g_n}{\sum_n g_n} \right) - \frac{1}{N} 2c \sum_n \mathbf{x} g_n$$

collecting like terms...

$$\nabla P(\mathbf{x}) = \frac{1}{N} 2c \sum_n g_n \left( \frac{\sum_n \mathbf{x}_n g_n}{\sum_n g_n} - \mathbf{x} \right)$$

*What's happening here?*

$$\nabla P(\mathbf{x}) = \underbrace{\frac{1}{N} 2c \sum_n g_n}_{\text{constant}} \left( \underbrace{\frac{\sum_n \mathbf{x}_n g_n}{\sum_n g_n} - \mathbf{x}}_{\text{mean shift!}} \right)$$

mean      shift

The **mean shift** is a ‘step’ in the direction of the gradient of the KDE

Let  $\mathbf{v}(\mathbf{x}) = \left( \frac{\sum_n \mathbf{x}_n g_n}{\sum_n g_n} - \mathbf{x} \right) = \frac{\nabla P(\mathbf{x})}{\frac{1}{N} 2c \sum_n g_n}$

Can interpret this to be  
gradient ascent with  
data dependent step size



# Mean-Shift Algorithm

Initialize  $\mathbf{x}$

While  $v(\mathbf{x}) > \epsilon$

1. Compute mean-shift

$$m(\mathbf{x}) = \frac{\sum_s K(\mathbf{x}, \mathbf{x}_s) \mathbf{x}_s}{\sum_s K(\mathbf{x}, \mathbf{x}_s)}$$

$$v(\mathbf{x}) = m(\mathbf{x}) - \mathbf{x}$$

2. Update  $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{v}(\mathbf{x})$

gradient with  
adaptive step size

$$\frac{\nabla P(\mathbf{x})}{\frac{1}{N} 2c \sum_n g_n}$$

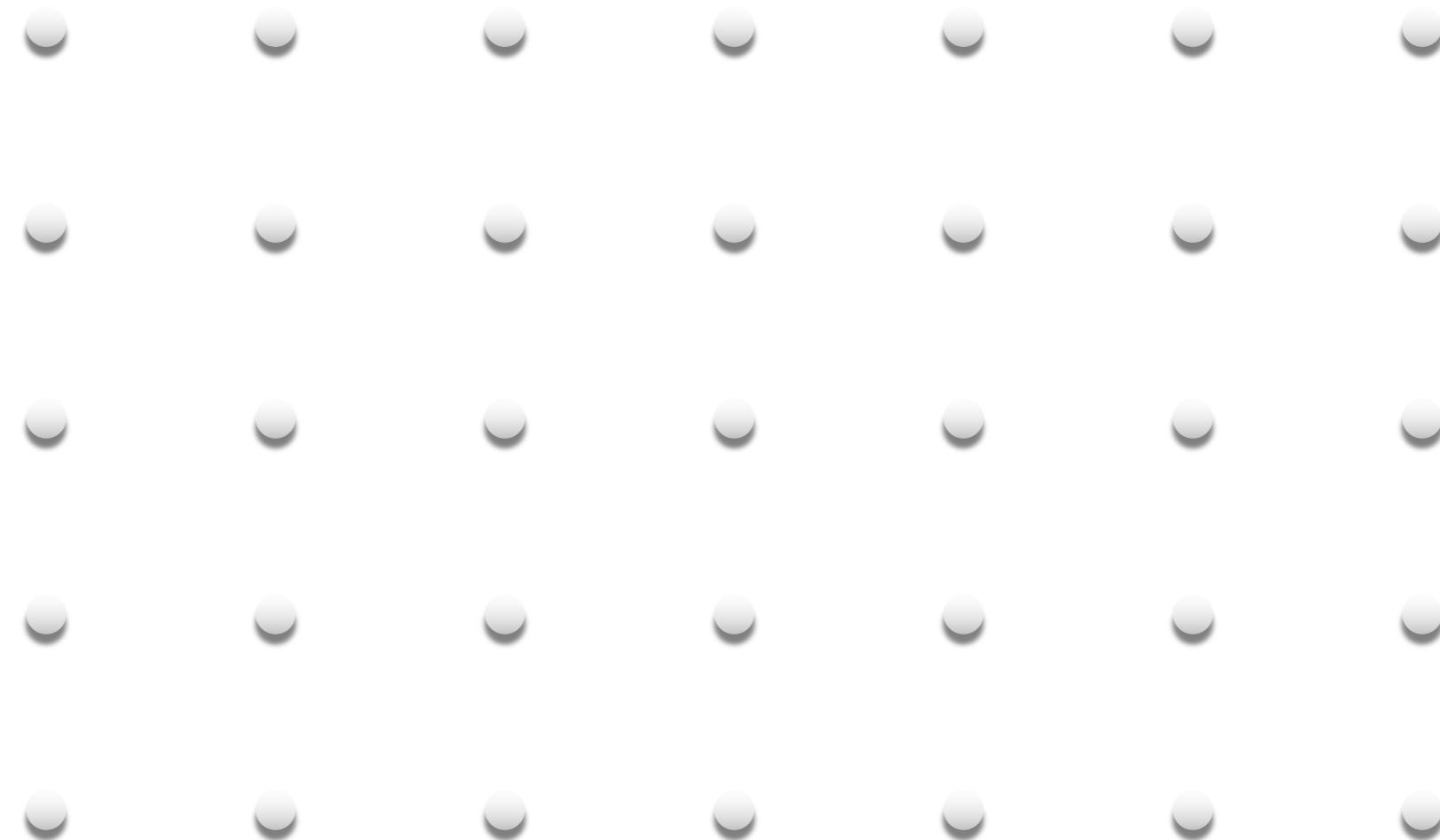
Just 5 lines of code!

Everything up to now has been about distributions over samples...

# Mean-shift tracker

# Dealing with images

Pixels for a lattice, spatial density is the same everywhere!



*What can we do?*

*same*  
Consider a set of points:  $\{\mathbf{x}_s\}_{s=1}^S \quad \mathbf{x}_s \in \mathcal{R}^d$

Associated weights:  $w(\mathbf{x}_s)$

Sample mean:

$$m(\mathbf{x}) = \frac{\sum_s K(\mathbf{x}, \mathbf{x}_s) w(\mathbf{x}_s) \mathbf{x}_s}{\sum_s K(\mathbf{x}, \mathbf{x}_s) w(\mathbf{x}_s)}$$

*same*  
Mean shift:  $m(\mathbf{x}) - \mathbf{x}$

# Mean-Shift Algorithm

(for images)

Initialize  $\mathbf{x}$

While  $v(\mathbf{x}) > \epsilon$

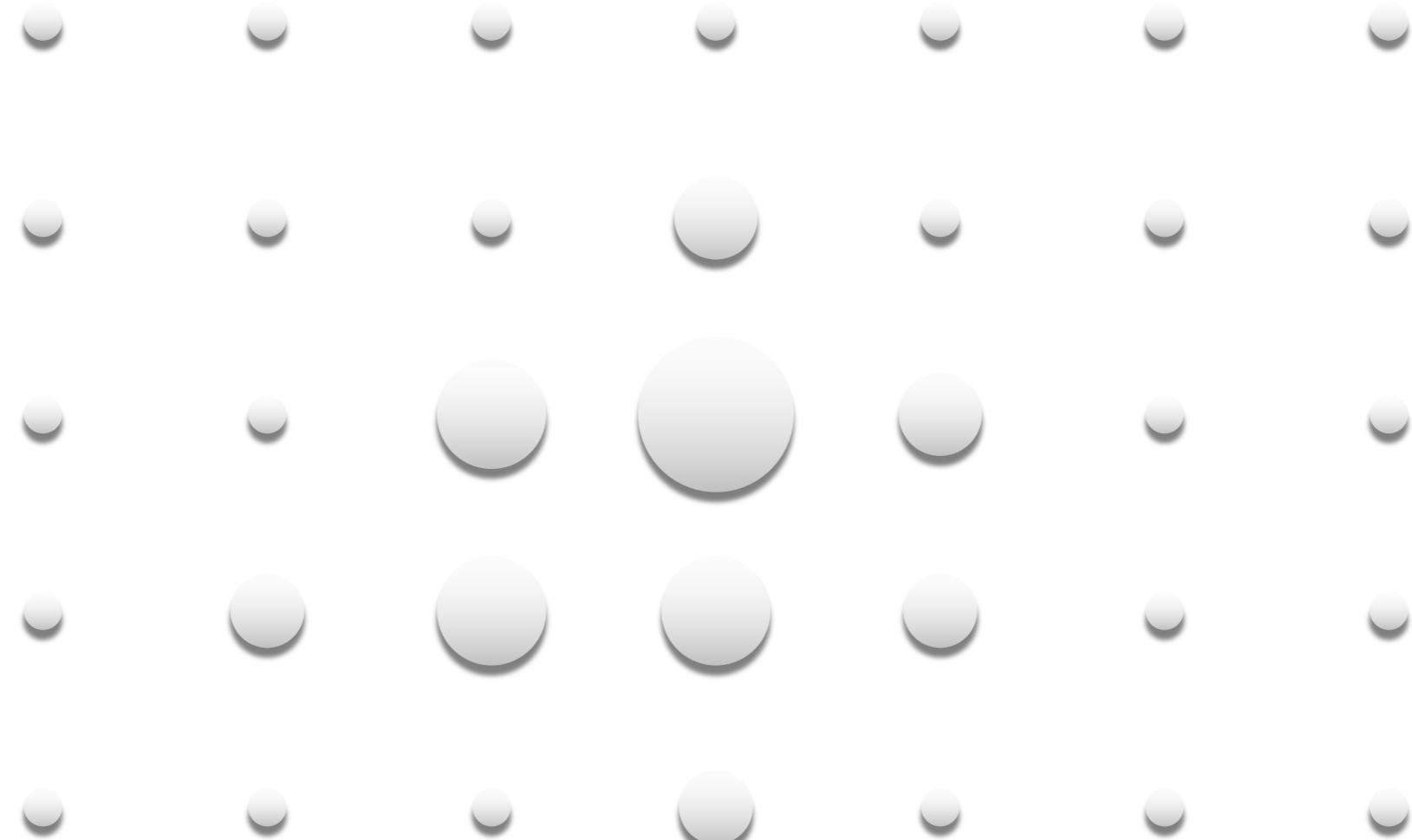
1. Compute mean-shift

$$m(\mathbf{x}) = \frac{\sum_s K(\mathbf{x}, \mathbf{x}_s) w(\mathbf{x}_s) \mathbf{x}_s}{\sum_s K(\mathbf{x}, \mathbf{x}_s) w(\mathbf{x}_s)}$$

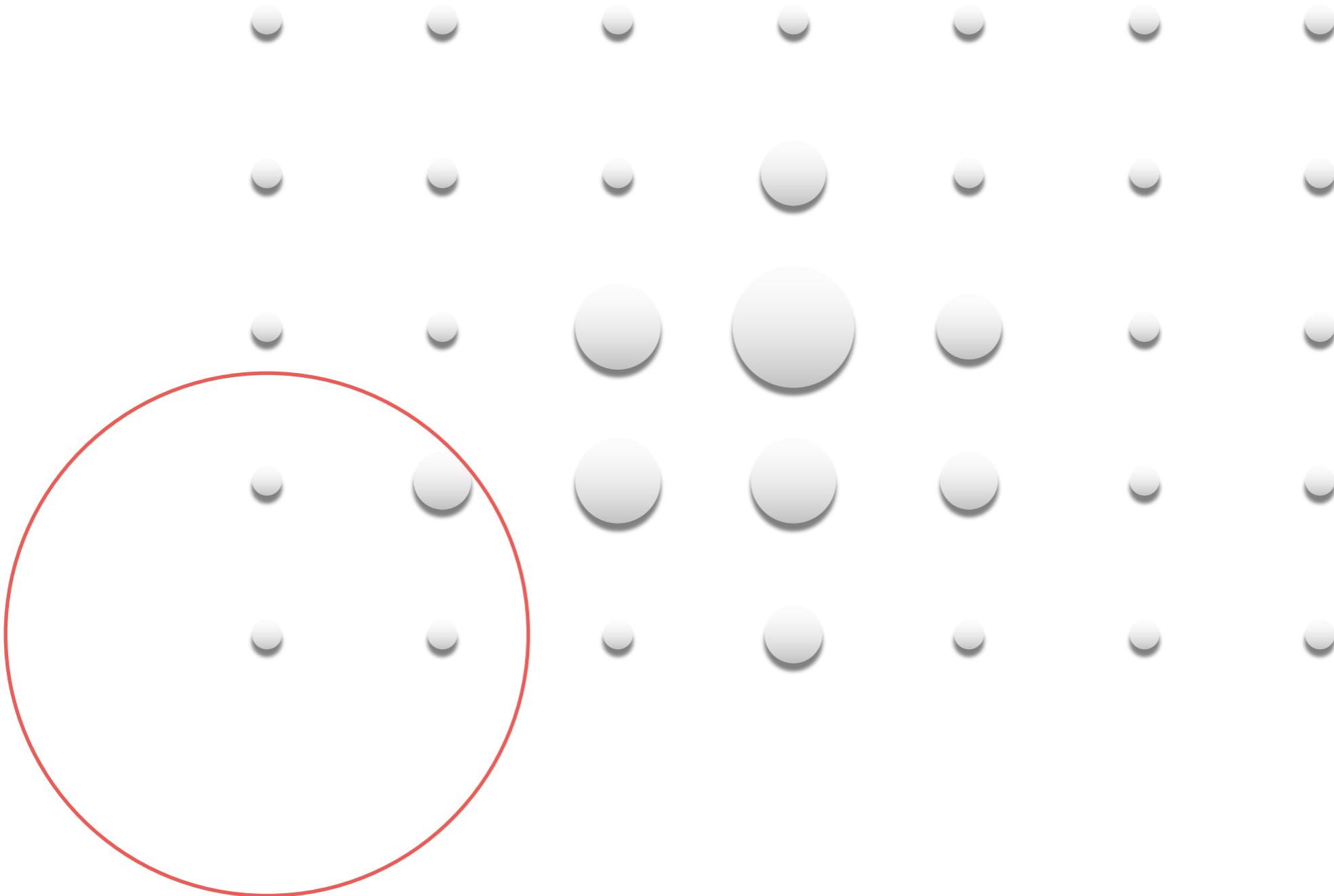
$$v(\mathbf{x}) = m(\mathbf{x}) - \mathbf{x}$$

2. Update  $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{v}(\mathbf{x})$

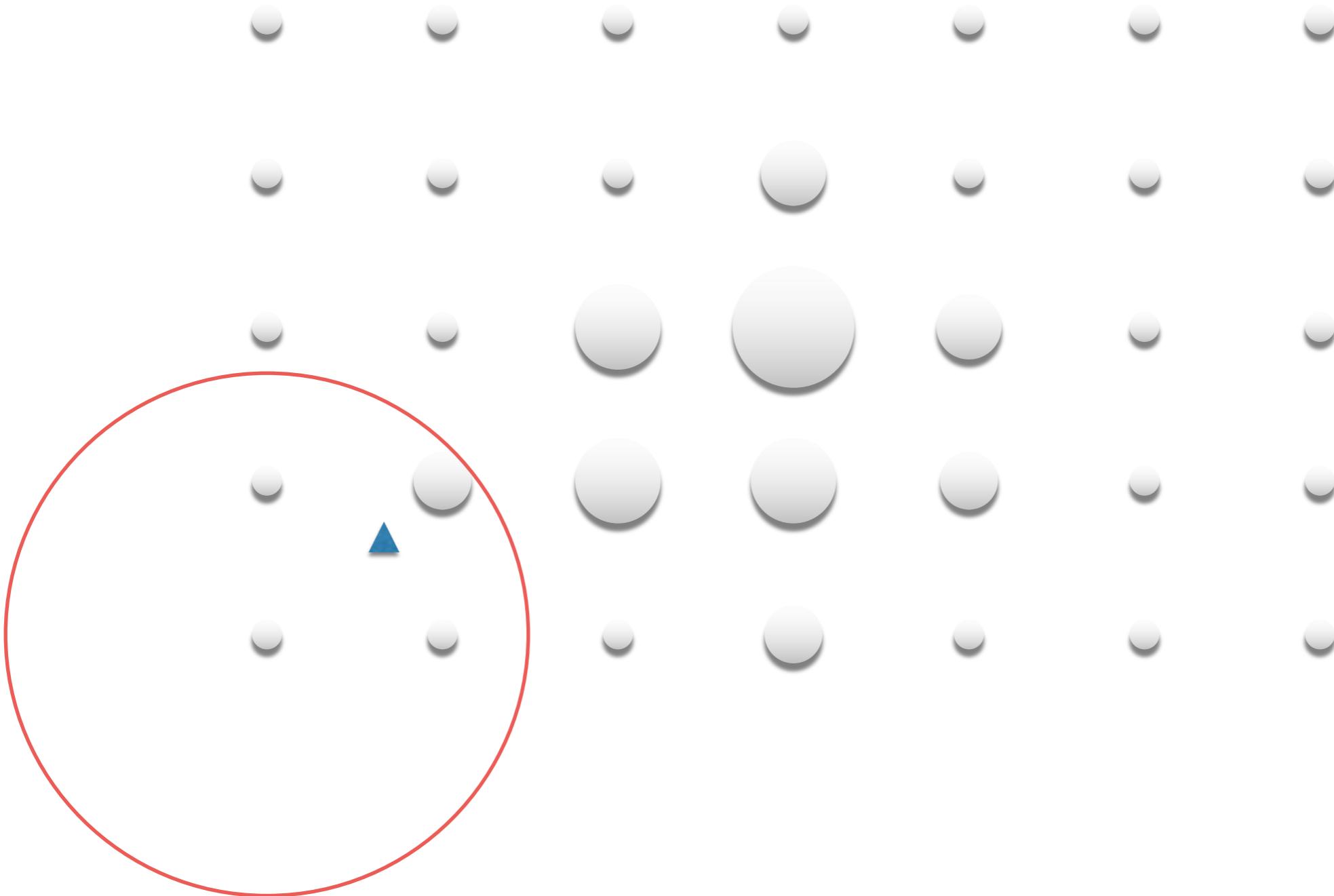
For images, each pixel is point with a weight



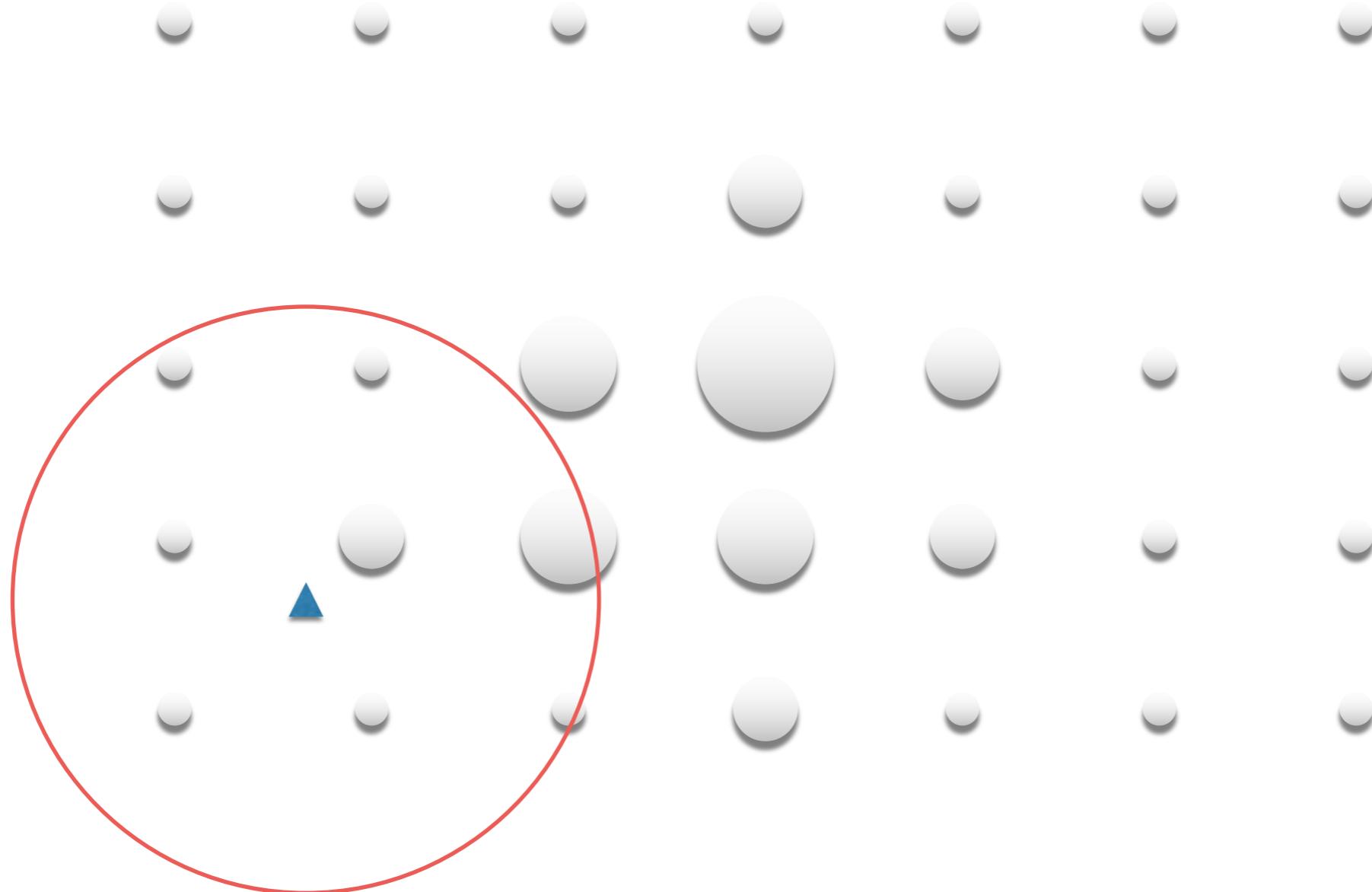
For images, each pixel is point with a weight



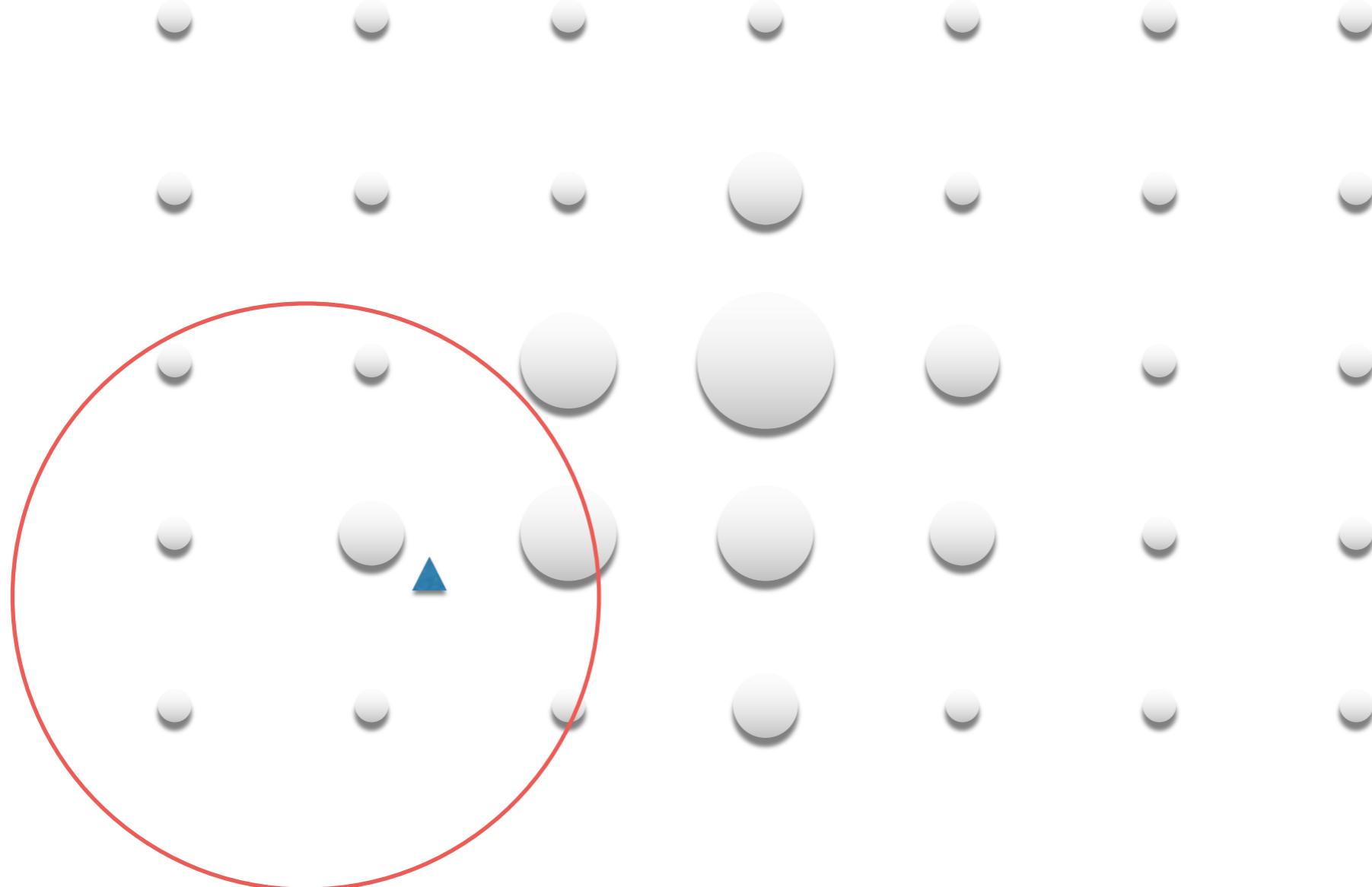
For images, each pixel is point with a weight



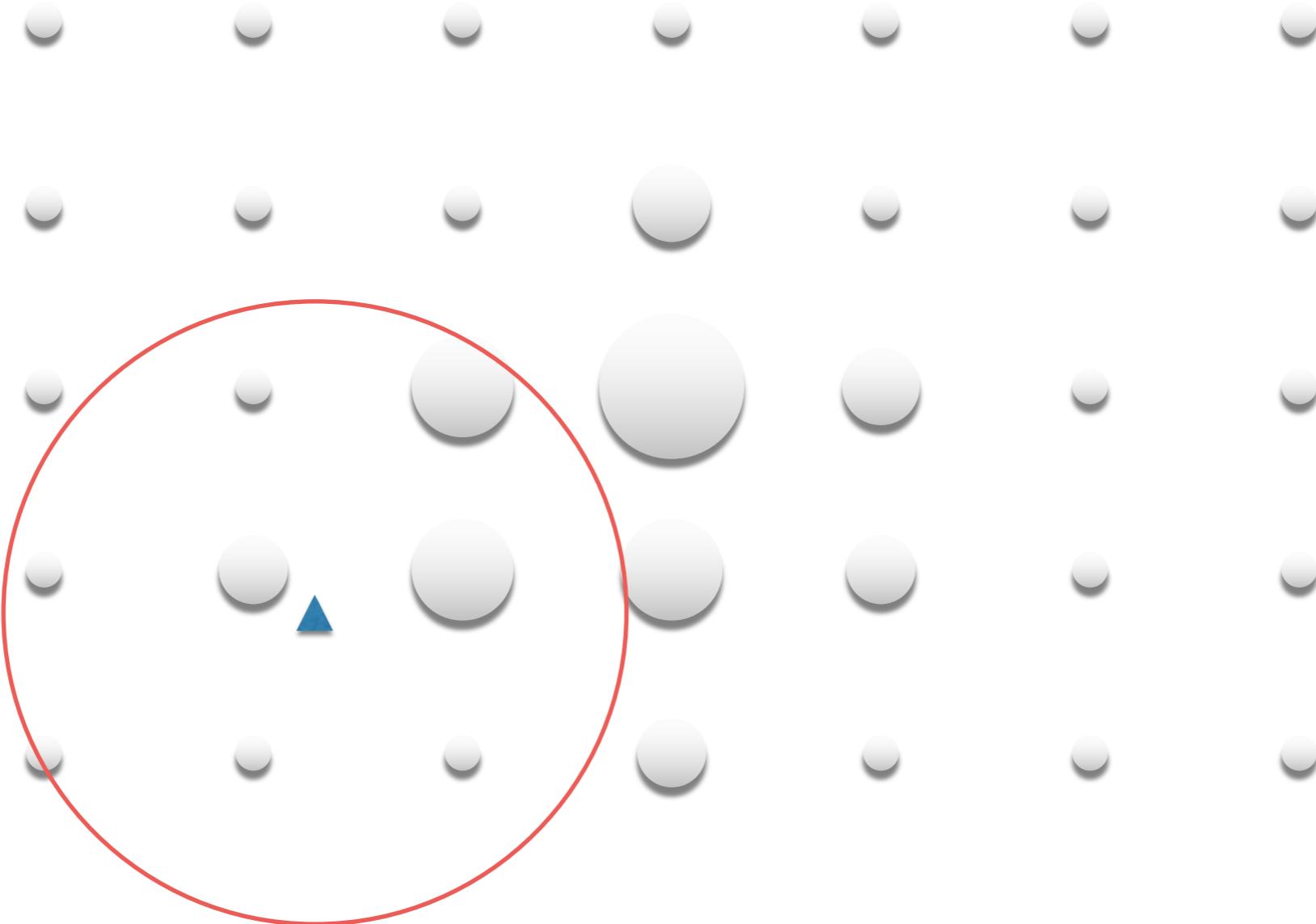
For images, each pixel is point with a weight



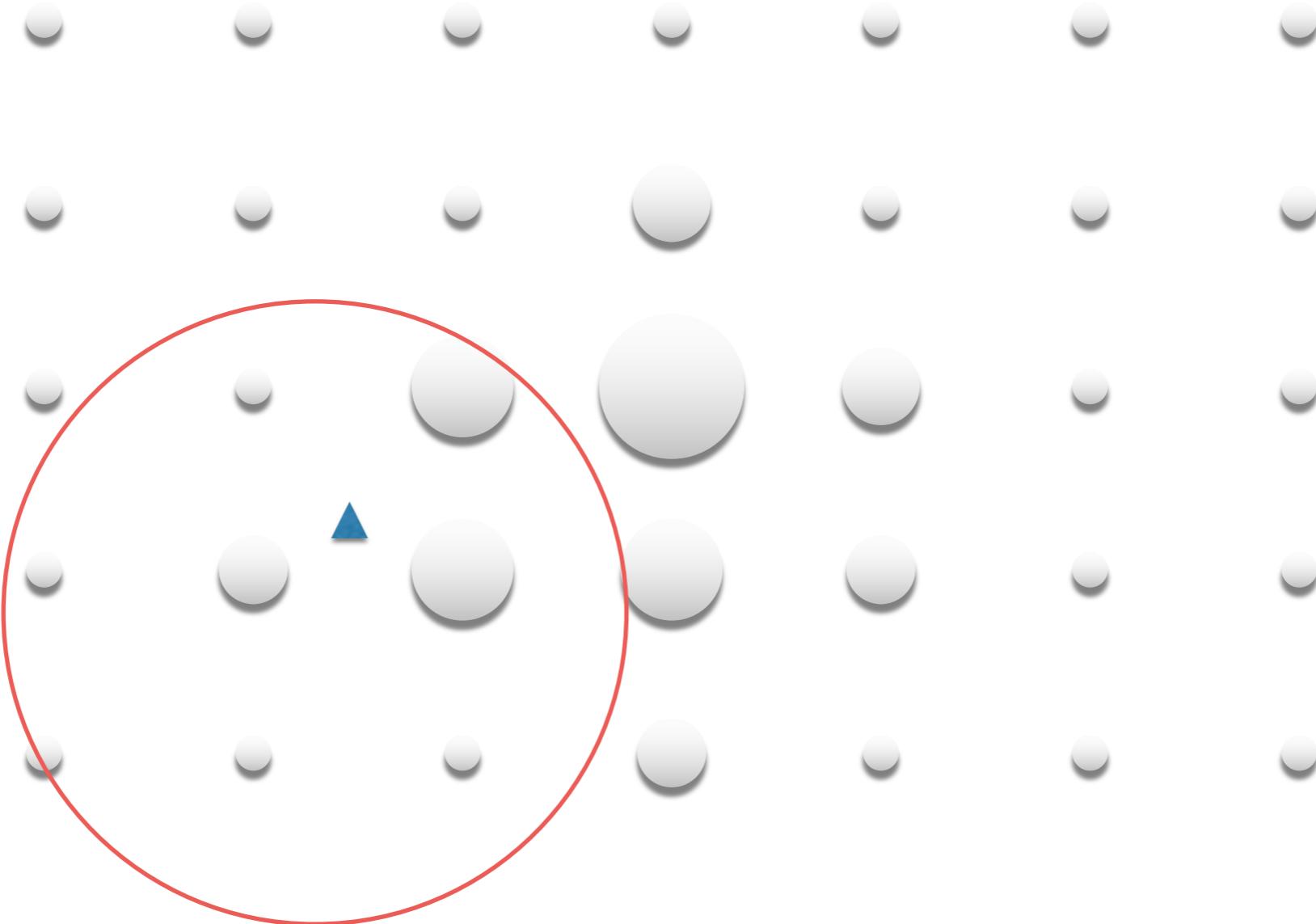
For images, each pixel is point with a weight



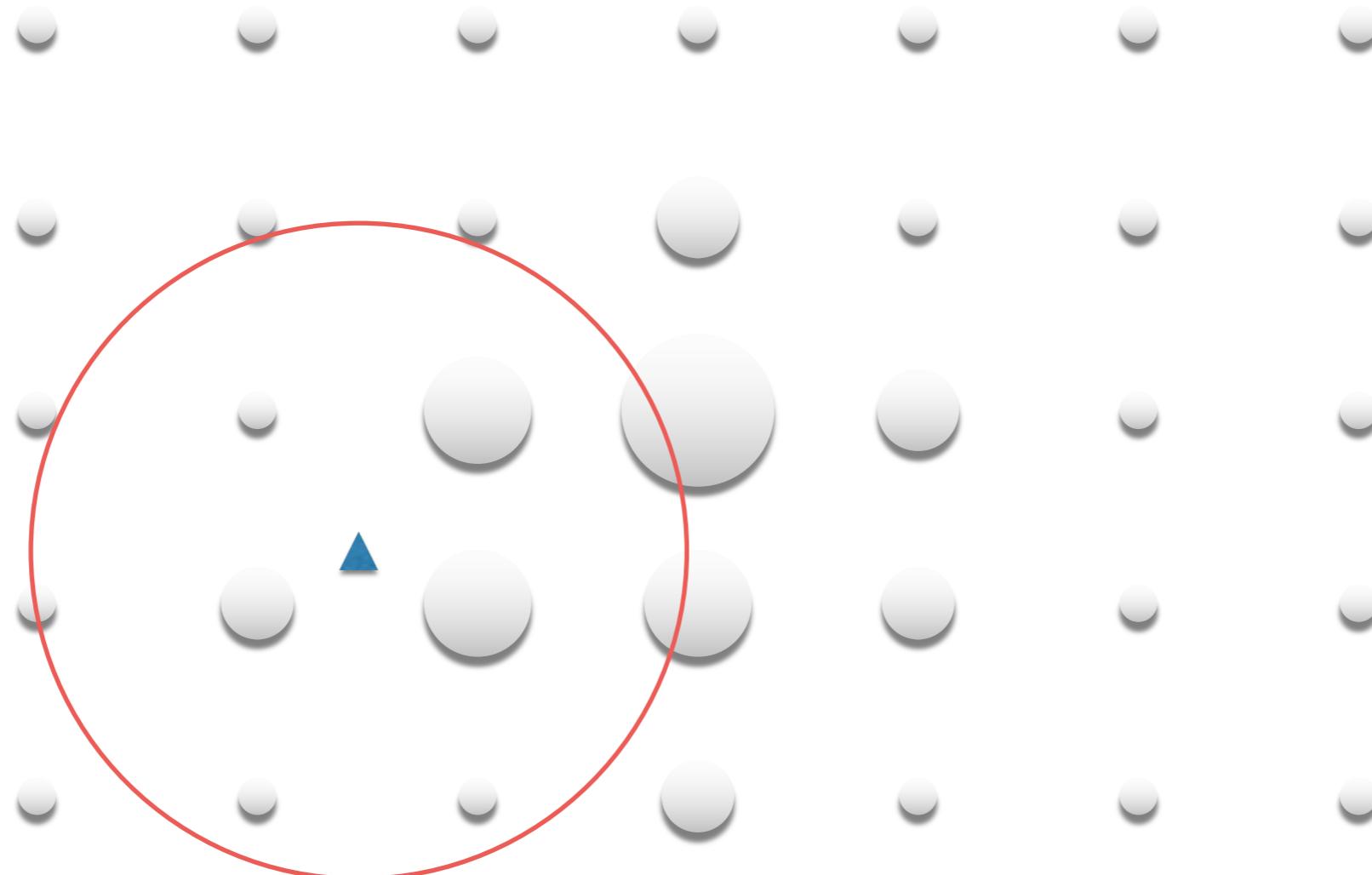
For images, each pixel is point with a weight



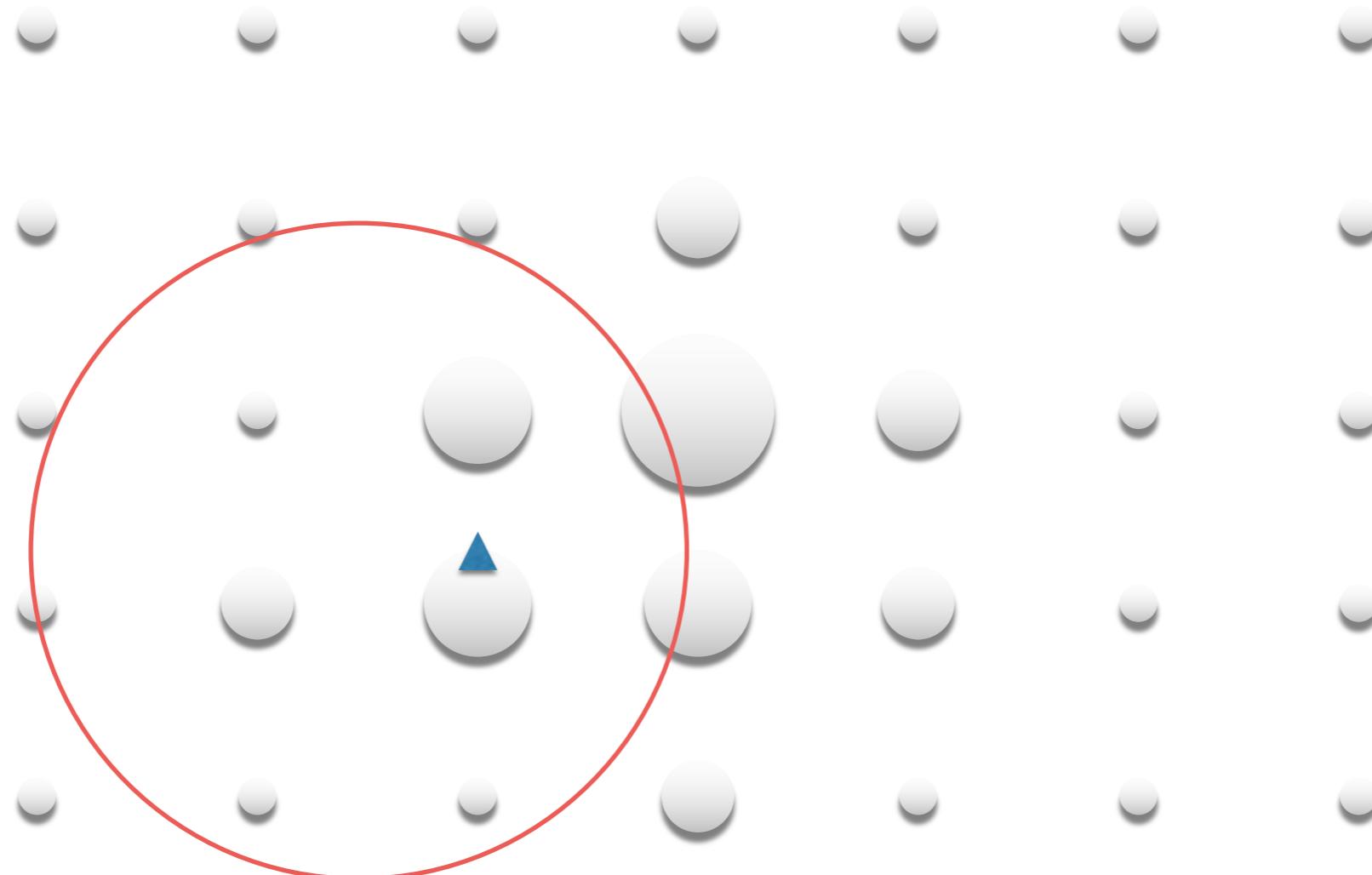
For images, each pixel is point with a weight



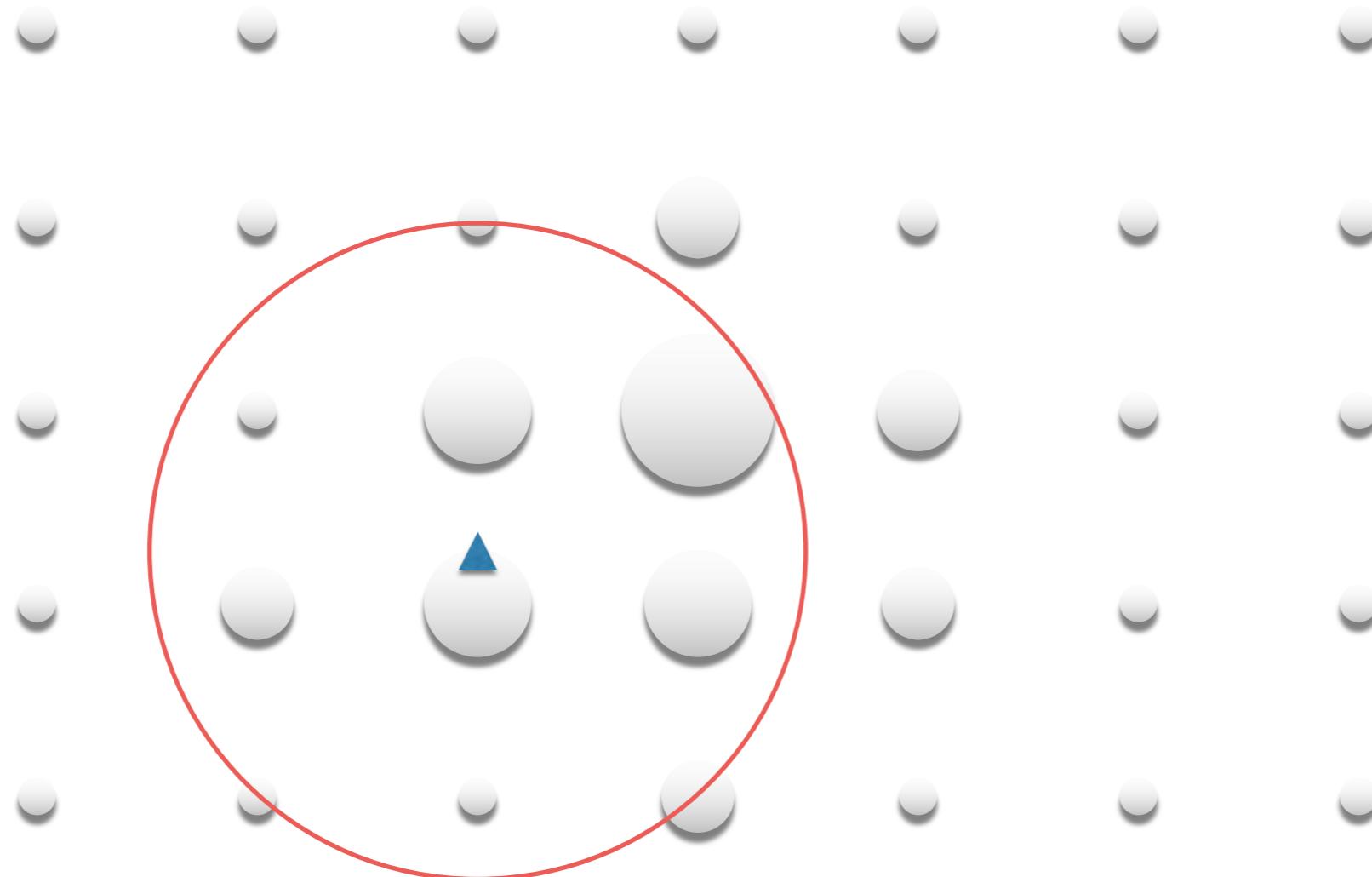
For images, each pixel is point with a weight



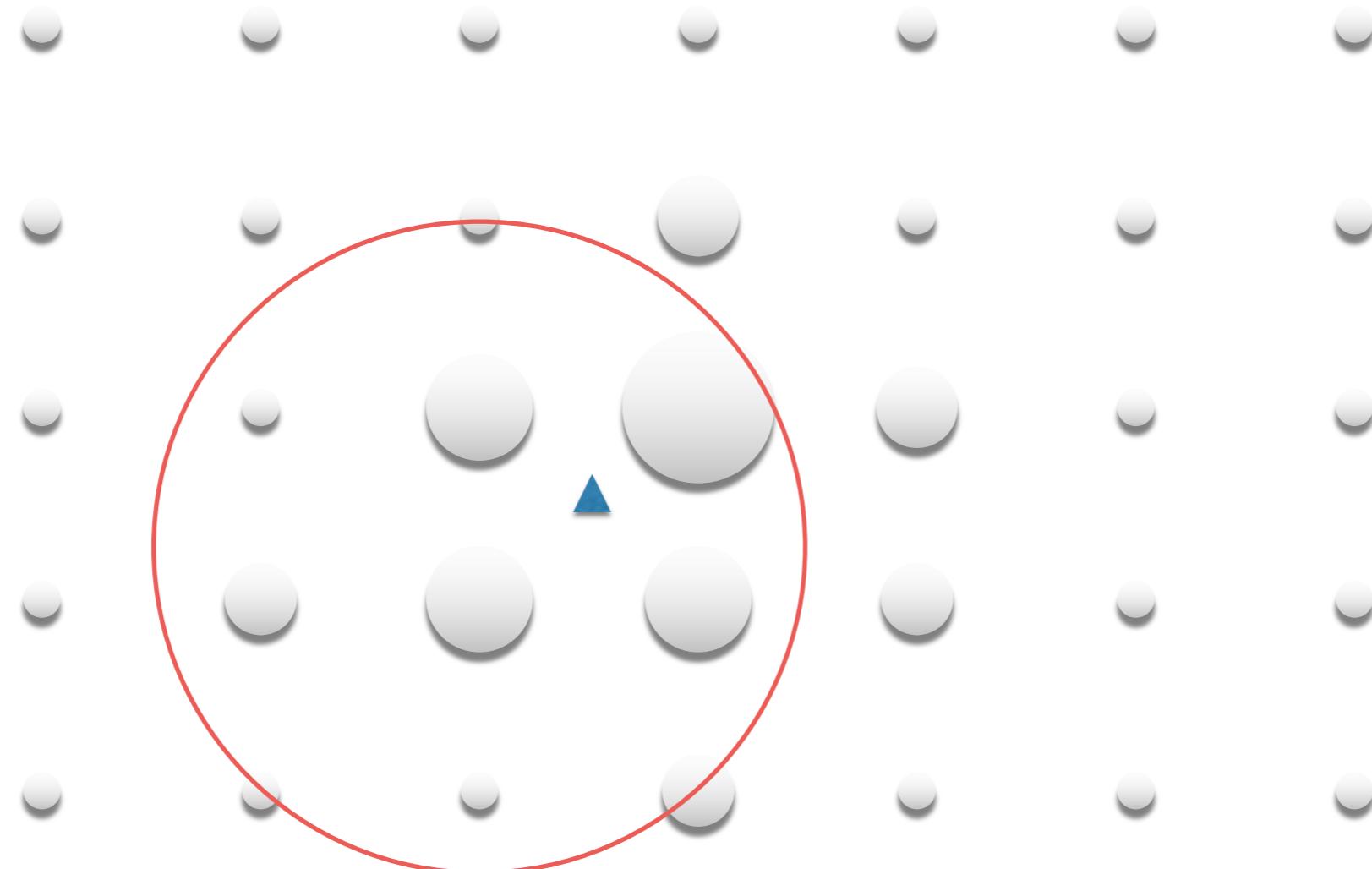
For images, each pixel is point with a weight



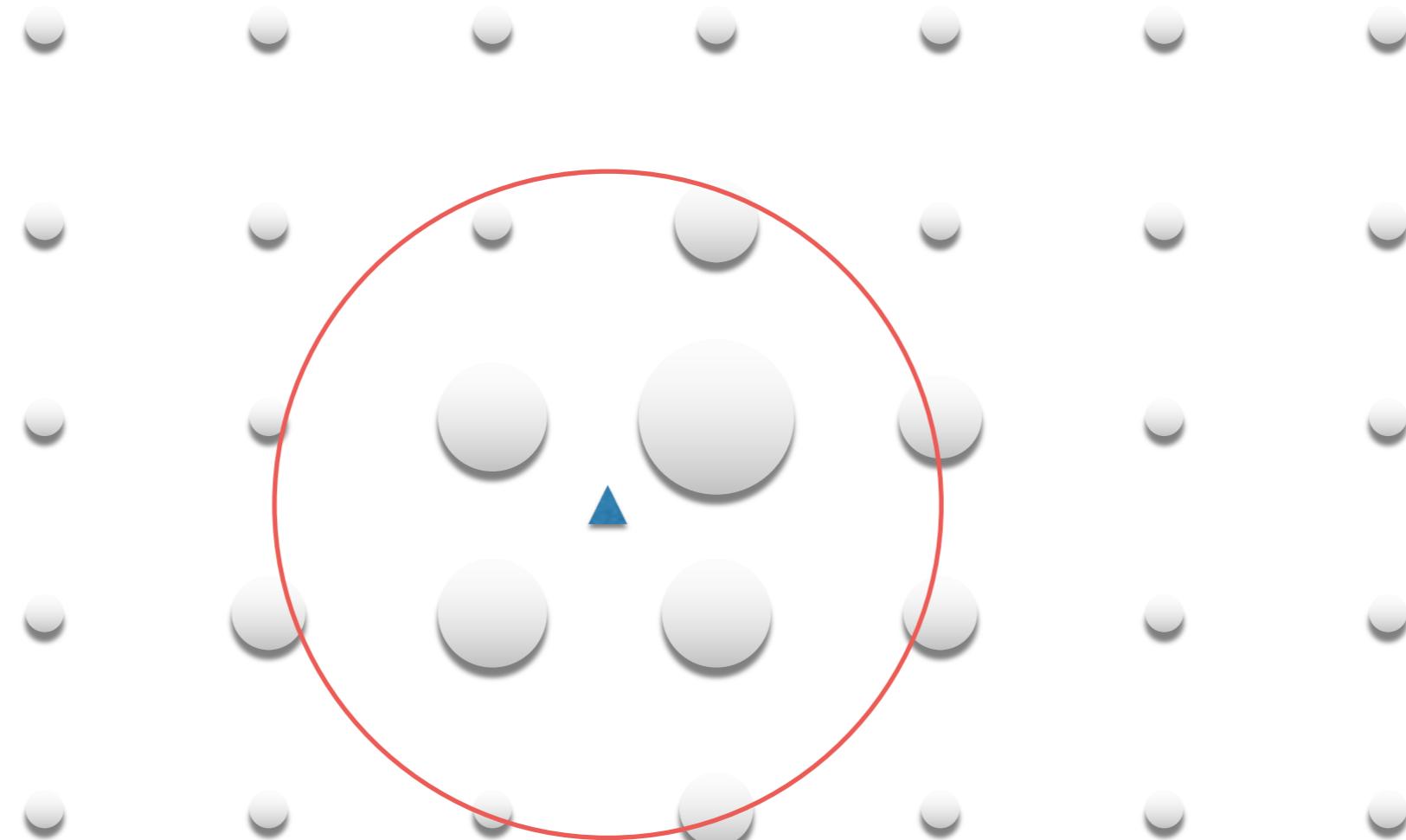
For images, each pixel is point with a weight



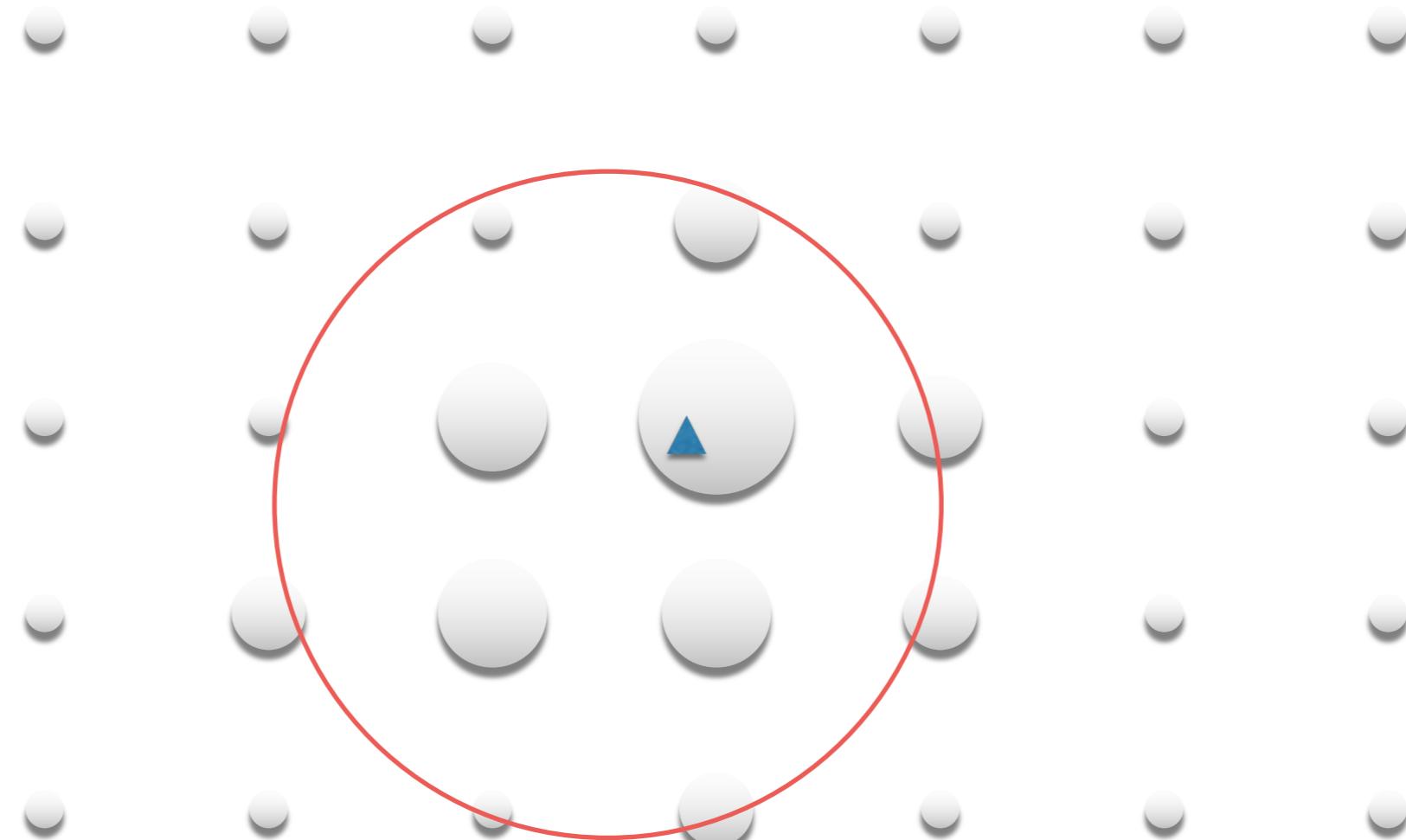
For images, each pixel is point with a weight



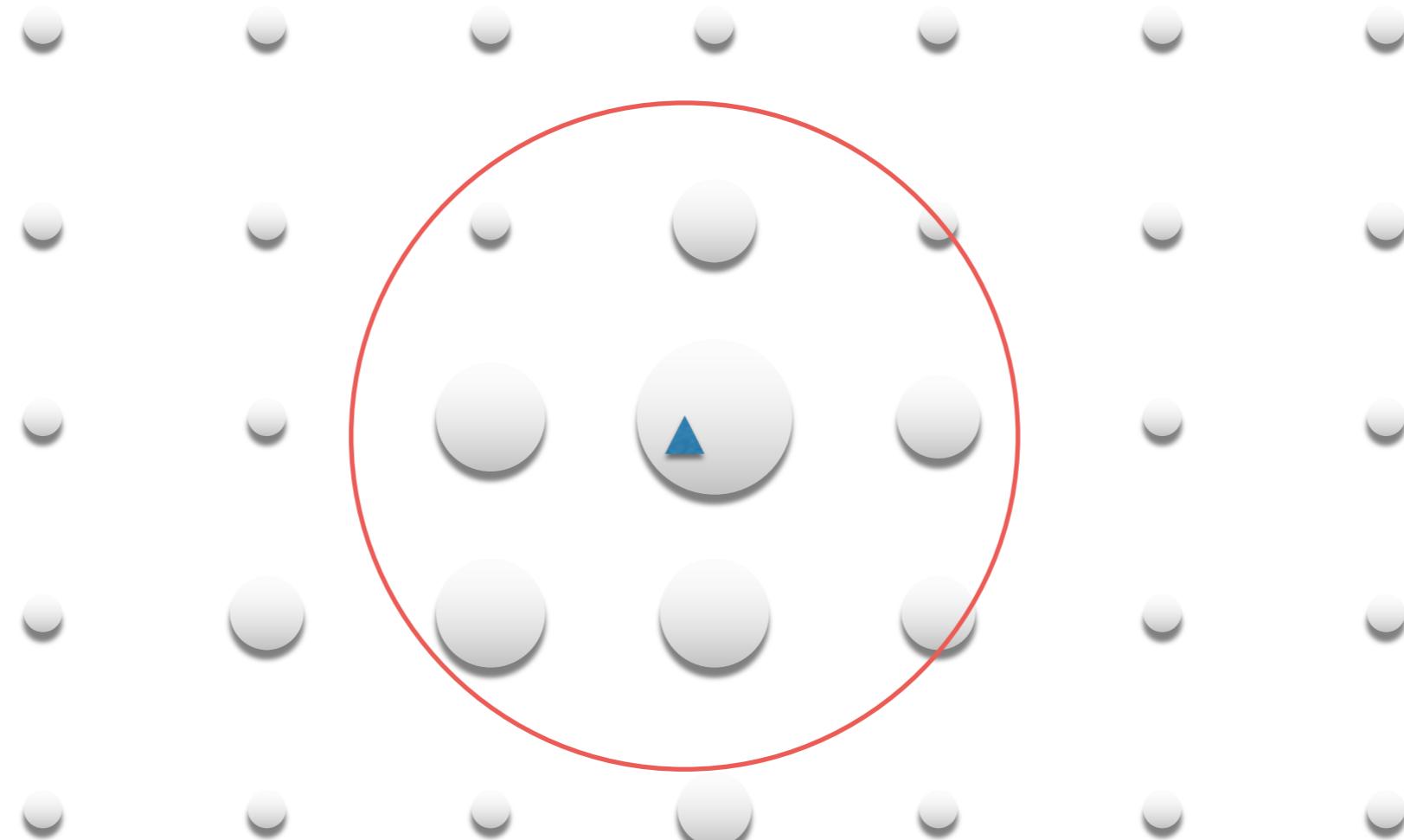
For images, each pixel is point with a weight



For images, each pixel is point with a weight

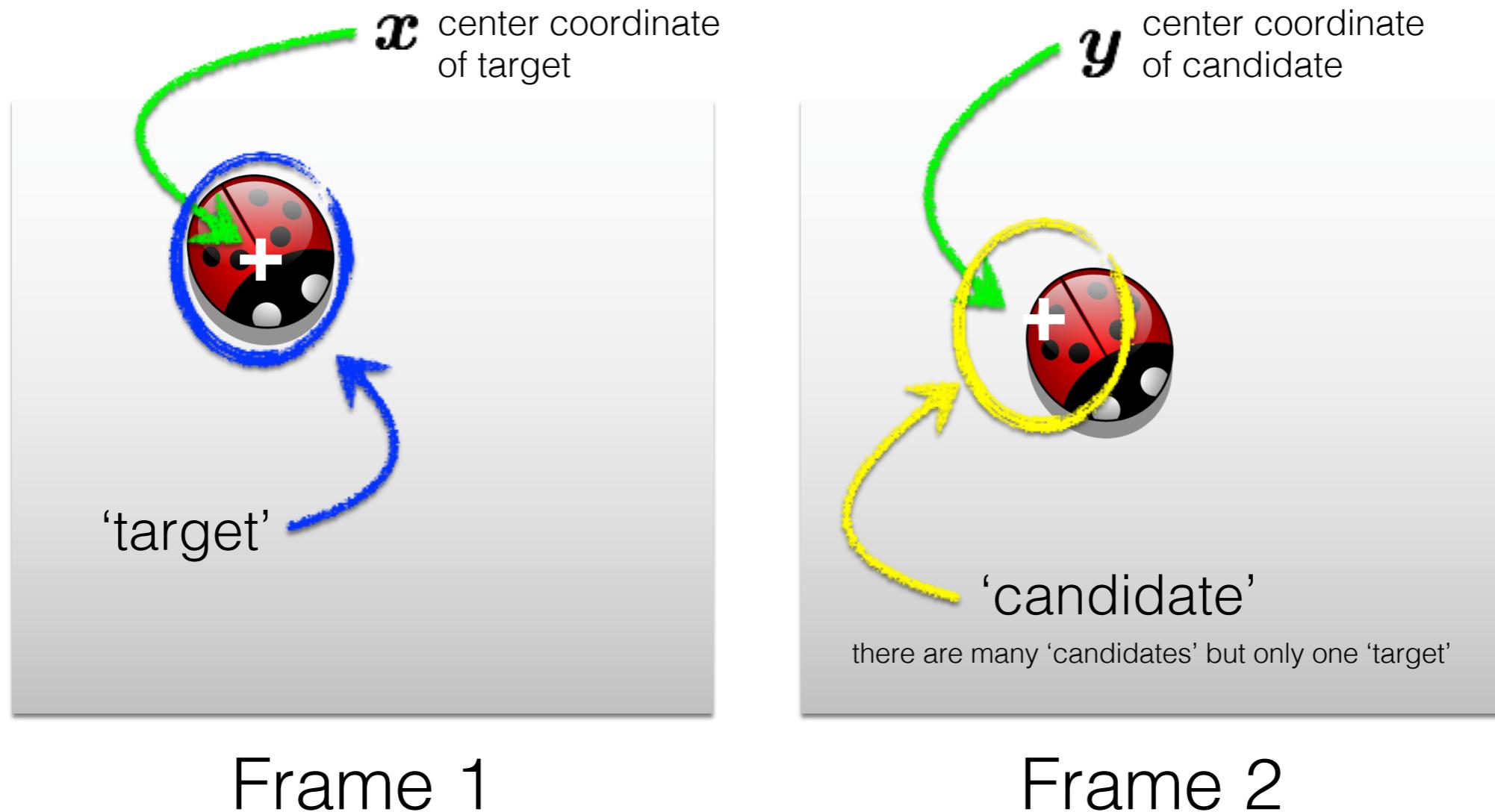


For images, each pixel is point with a weight



Finally... mean shift tracking in video!

# Goal: find the best candidate location in frame 2

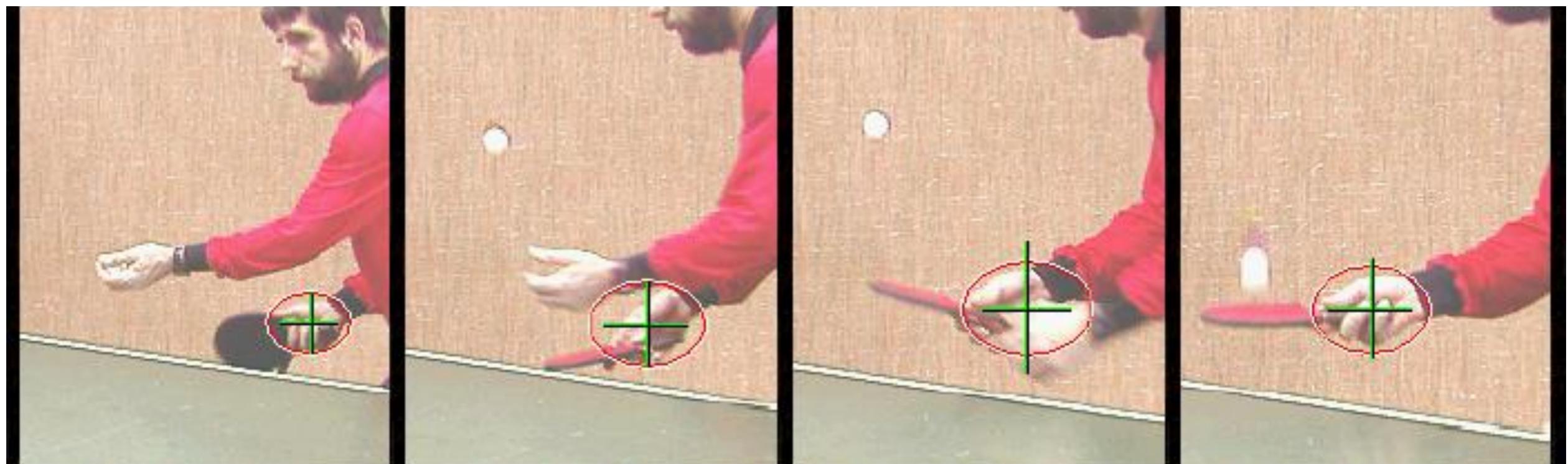


Frame 1

Frame 2

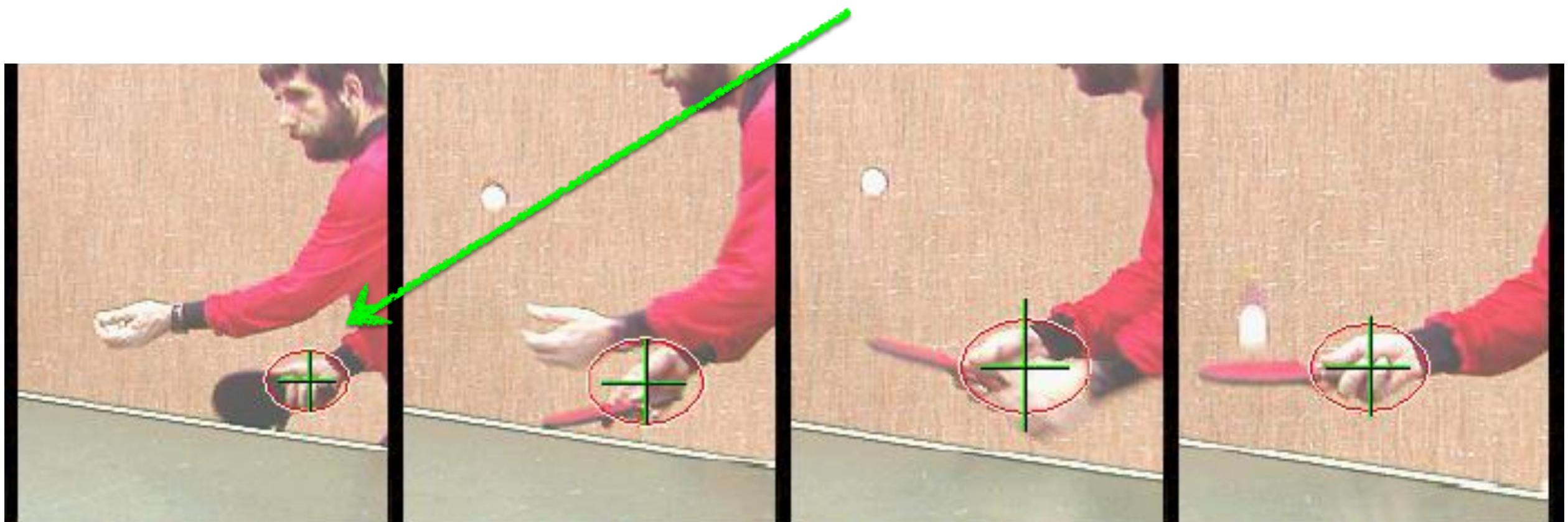
Use the mean shift algorithm  
to find the best candidate location

# Non-rigid object tracking



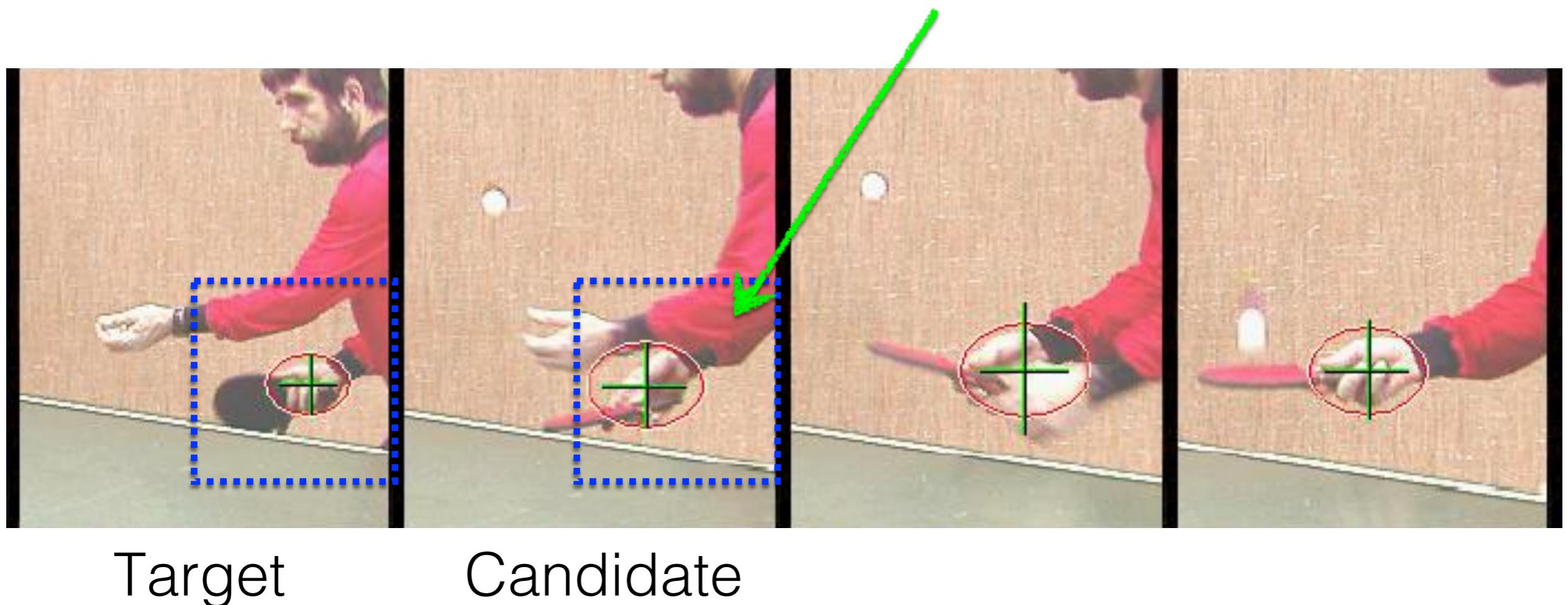
hand tracking

Compute a descriptor for the target

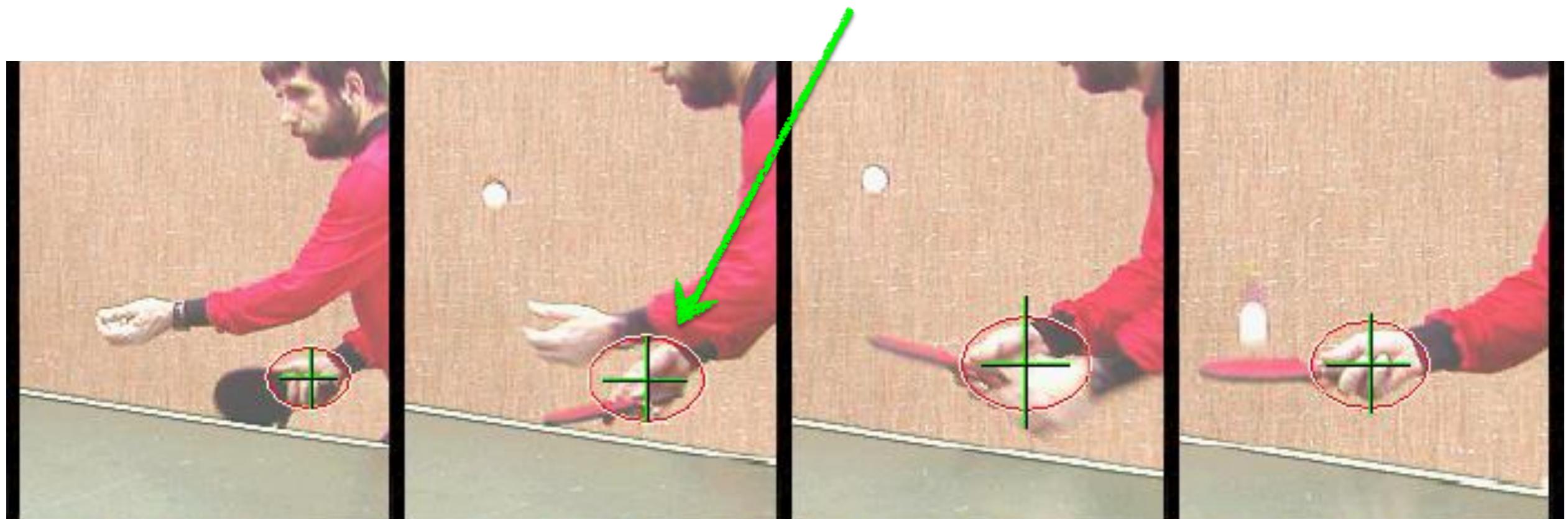


Target

Search for similar descriptor in neighborhood in next frame

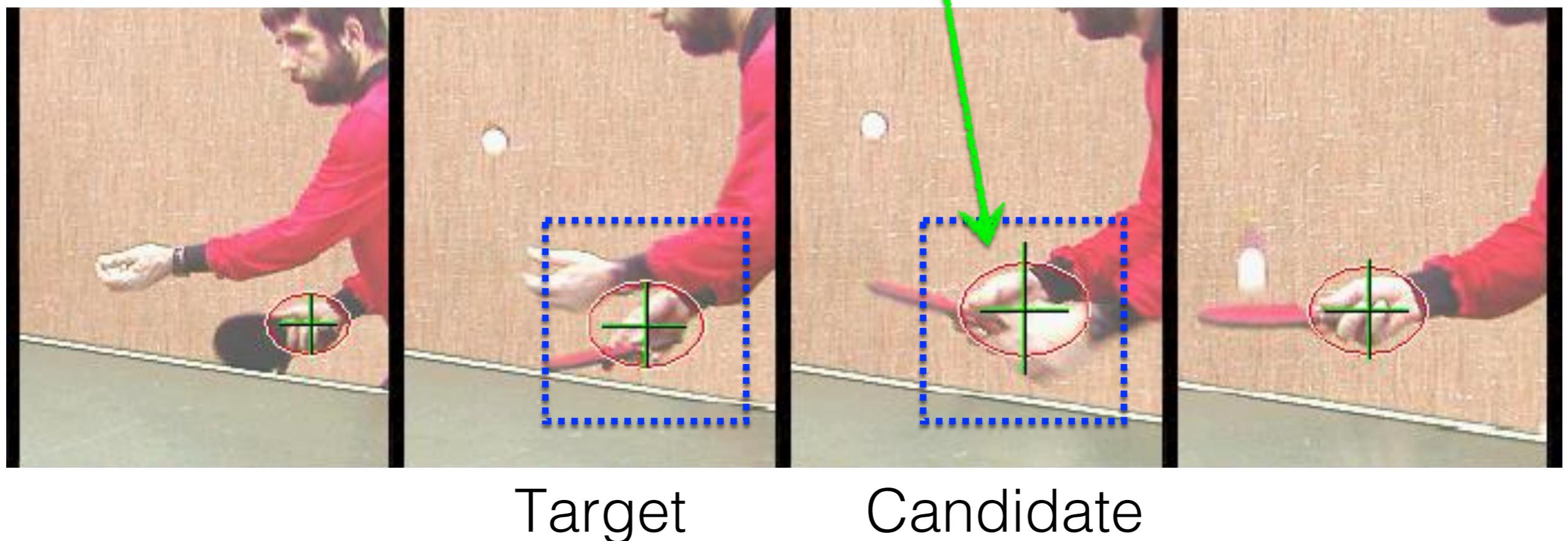


Compute a descriptor for the new target



Target

Search for similar descriptor in neighborhood in next frame



How do we model the target and candidate regions?

# Modeling the target



M-dimensional **target** descriptor

$$\mathbf{q} = \{q_1, \dots, q_M\}$$

(centered at target center)

a ‘fancy’ (confusing) way to write a weighted histogram

$$q_m = C \sum_n k(\|\mathbf{x}_n\|^2) \delta[b(\mathbf{x}_n) - m]$$

Normalization factor

$n$

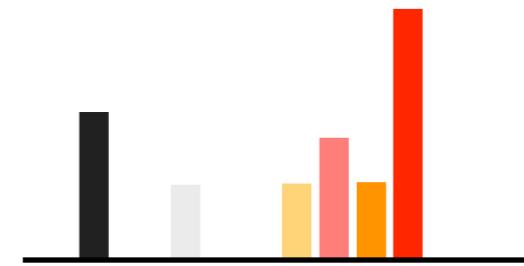
sum over all pixels

function of inverse distance (weight)

Kronecker delta function

quantization function

bin ID



A normalized color histogram (weighted by distance)

# Modeling the candidate

M-dimensional **candidate** descriptor

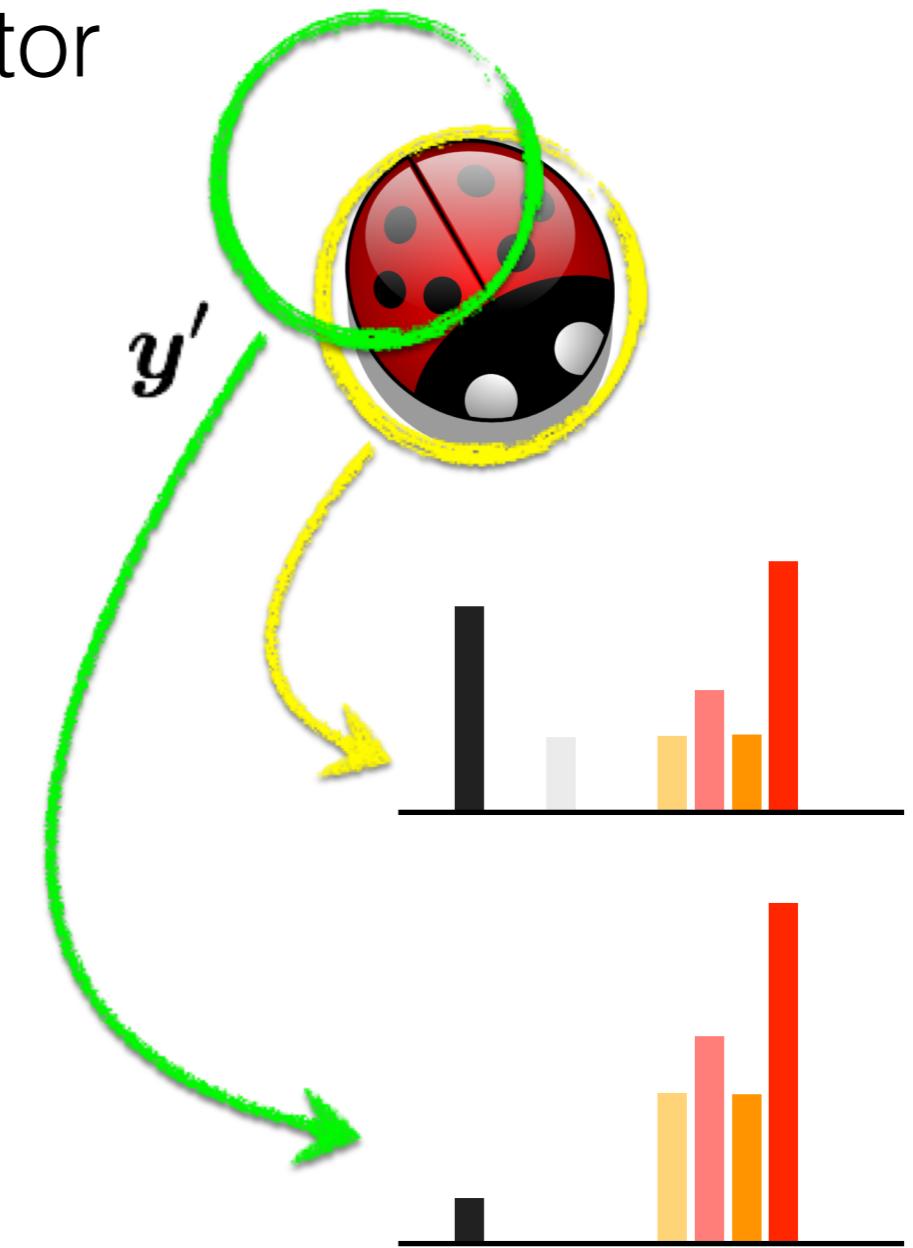
$$\mathbf{p}(\mathbf{y}) = \{p_1(\mathbf{y}), \dots, p_M(\mathbf{y})\}$$

(centered at location  $\mathbf{y}$ )

a weighted histogram at  $\mathbf{y}$

$$p_m = C_h \sum_n k \left( \left\| \frac{\mathbf{y} - \mathbf{x}_n}{h} \right\|^2 \right) \delta[b(\mathbf{x}_n) - m]$$

bandwidth



# Similarity between the target and candidate

Distance function

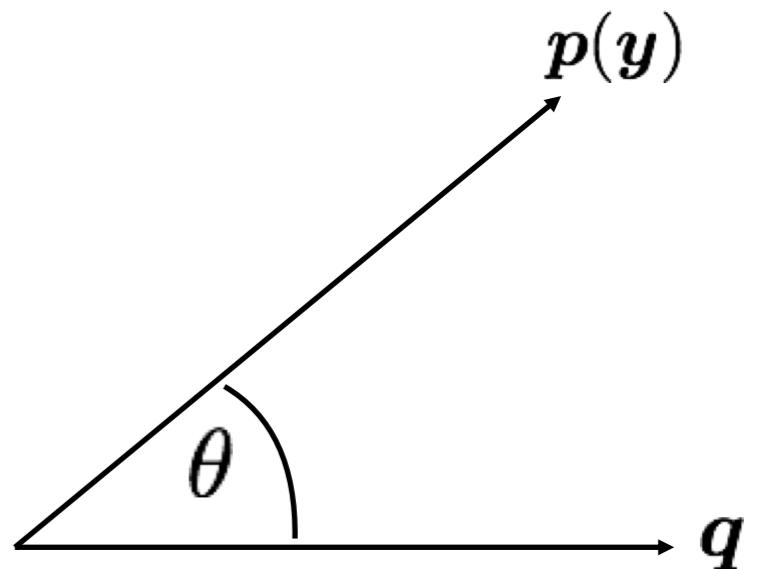
$$d(\mathbf{y}) = \sqrt{1 - \rho[\mathbf{p}(\mathbf{y}), \mathbf{q}]}$$

Bhattacharyya Coefficient

$$\rho(\mathbf{y}) \equiv \rho[\mathbf{p}(\mathbf{y}), \mathbf{q}] = \sum_m \sqrt{p_m(\mathbf{y})q_u}$$

Just the Cosine distance between two unit vectors

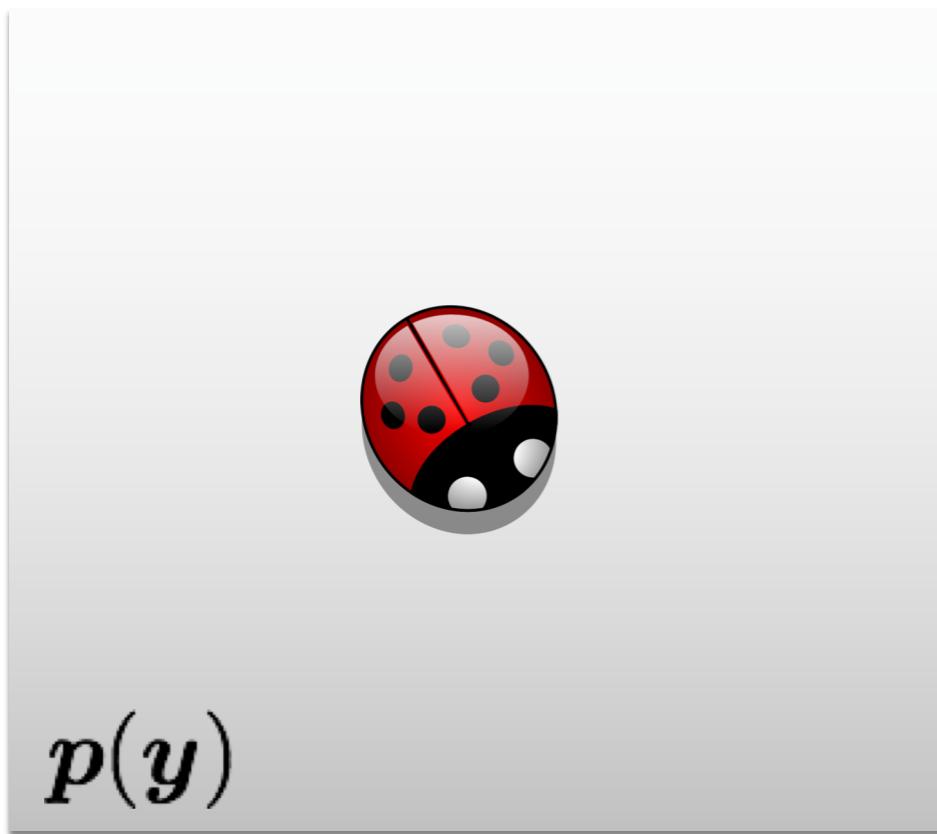
$$\rho(\mathbf{y}) = \cos \theta_{\mathbf{y}} = \frac{\mathbf{p}(\mathbf{y})^\top \mathbf{q}}{\|\mathbf{p}\| \|\mathbf{q}\|} = \sum_m \sqrt{p_m(\mathbf{y})q_m}$$



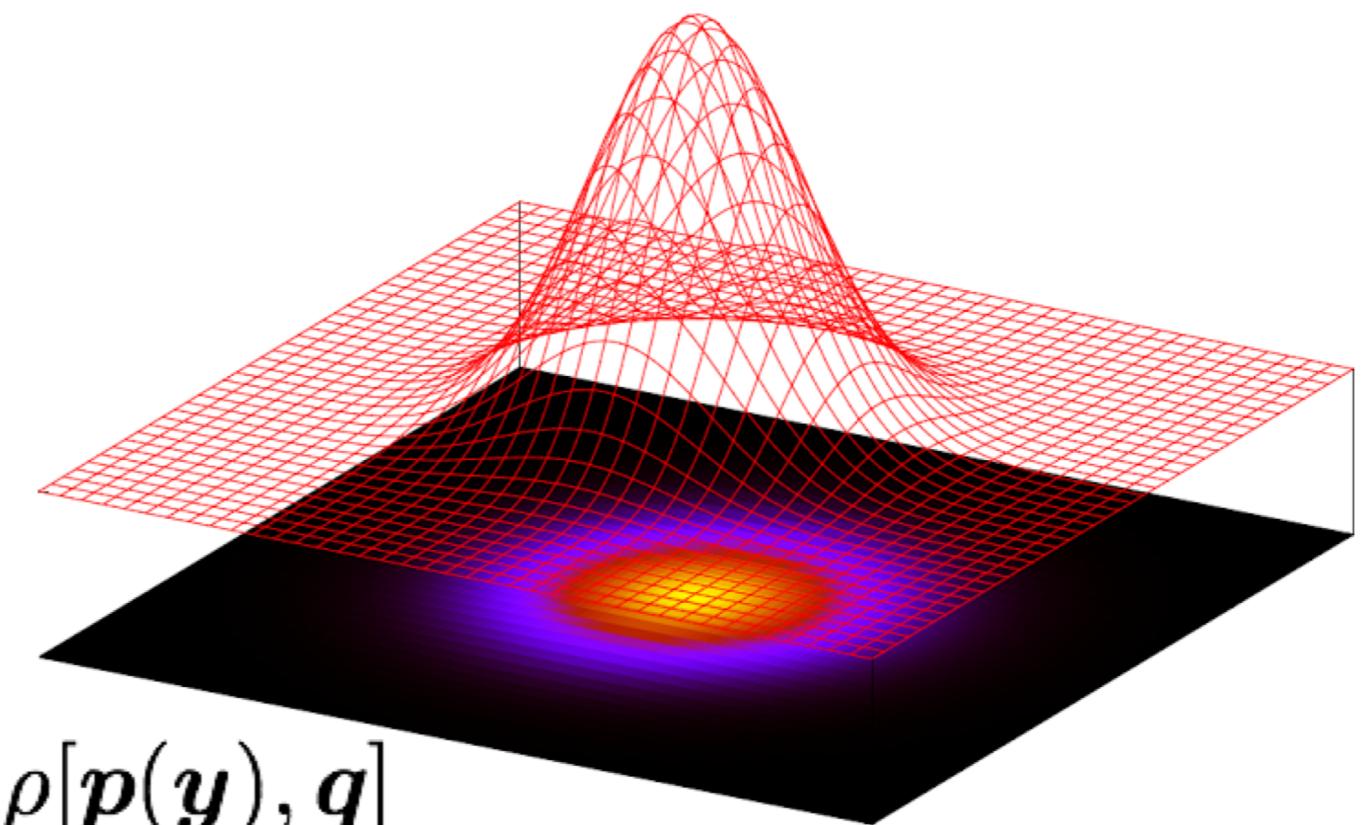
Now we can compute the similarity between  
a target and multiple candidate regions



target

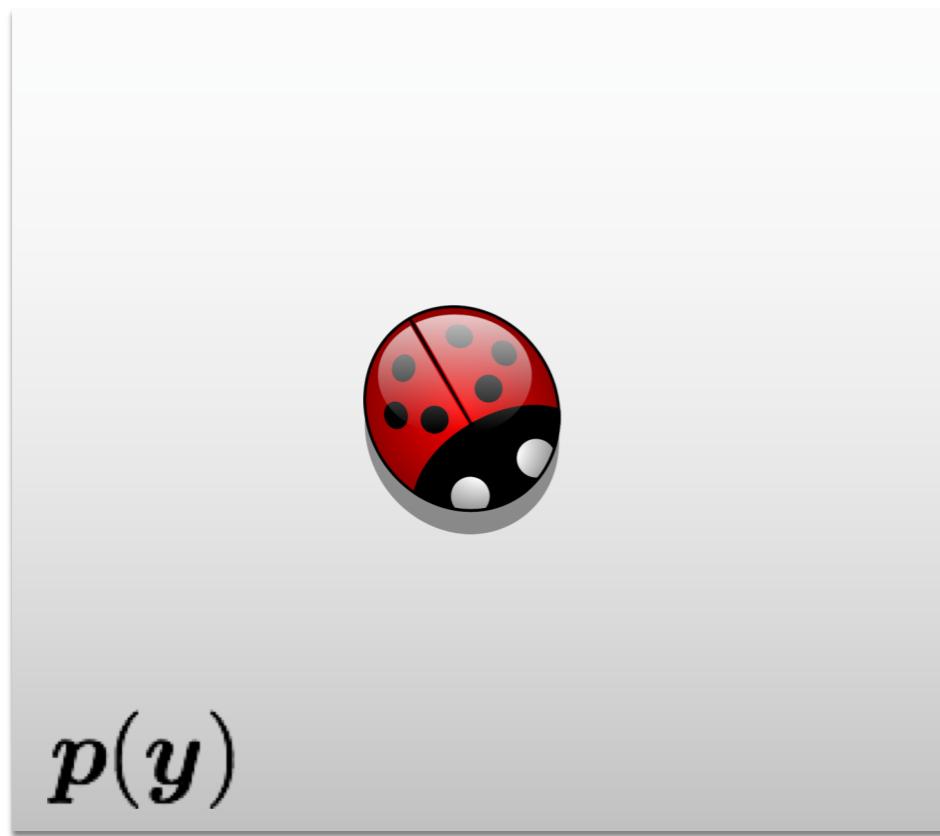


image



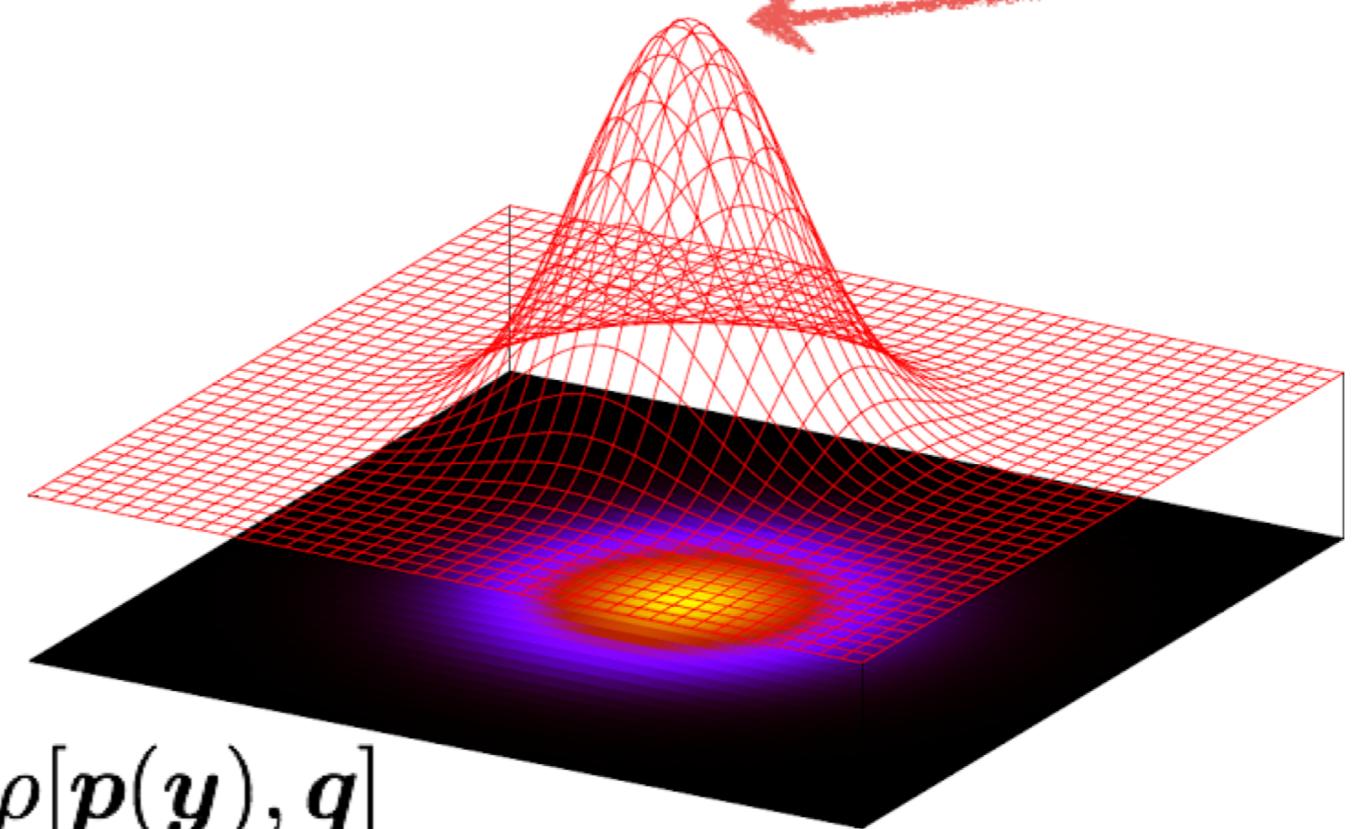
$$\rho[\mathbf{p}(\mathbf{y}), \mathbf{q}]$$

similarity over image



target

we want to find this peak



image

similarity over image

# Objective function

$$\min_{\mathbf{y}} d(\mathbf{y}) \quad \text{same as} \quad \max_{\mathbf{y}} \rho[p(\mathbf{y}), q]$$

Assuming a good initial guess

$$\rho[p(y_0 + y), q]$$

Linearize around the initial guess (Taylor series expansion)

## Linearized objective

$$\rho[\mathbf{p}(\mathbf{y}), \mathbf{q}] \approx \frac{1}{2} \sum_m \sqrt{p_m(\mathbf{y}_0) q_m} + \frac{1}{2} \sum_m p_m(\mathbf{y}) \sqrt{\frac{q_m}{p_m(\mathbf{y}_0)}}$$

$$p_m = C_h \sum_n k \left( \left\| \frac{\mathbf{y} - \mathbf{x}_n}{h} \right\|^2 \right) \delta[b(\mathbf{x}_n) - m]$$

Remember  
definition of this?

## Fully expanded

$$\rho[\mathbf{p}(\mathbf{y}), \mathbf{q}] \approx \frac{1}{2} \sum_m \sqrt{p_m(\mathbf{y}_0) q_m} + \frac{1}{2} \sum_m \left\{ C_h \sum_n k \left( \left\| \frac{\mathbf{y} - \mathbf{x}_n}{h} \right\|^2 \right) \delta[b(\mathbf{x}_n) - m] \right\} \sqrt{\frac{q_m}{p_m(\mathbf{y}_0)}}$$

## Fully expanded linearized objective

$$\rho[\mathbf{p}(\mathbf{y}), \mathbf{q}] \approx \frac{1}{2} \sum_m \sqrt{p_m(\mathbf{y}_0) q_m} + \frac{1}{2} \sum_m \left\{ C_h \sum_n k \left( \left\| \frac{\mathbf{y} - \mathbf{x}_n}{h} \right\|^2 \right) \delta[b(\mathbf{x}_n) - m] \right\} \sqrt{\frac{q_m}{p_m(\mathbf{y}_0)}}$$

Moving terms around...

$$\rho[\mathbf{p}(\mathbf{y}), \mathbf{q}] \approx \boxed{\frac{1}{2} \sum_m \sqrt{p_m(\mathbf{y}_0) q_m}} + \boxed{\frac{C_h}{2} \sum_n w_n k \left( \left\| \frac{\mathbf{y} - \mathbf{x}_n}{h} \right\|^2 \right)}$$

Does not depend on unknown  $\mathbf{y}$

Weighted kernel density estimate

$$\text{where } w_n = \sum_m \sqrt{\frac{q_m}{p_m(\mathbf{y}_0)}} \delta[b(\mathbf{x}_n) - m]$$

Weight is bigger when  $q_m > p_m(\mathbf{y}_0)$

OK, why are we doing all this math?

We want to maximize this

$$\max_{\boldsymbol{y}} \rho[\boldsymbol{p}(\boldsymbol{y}), \boldsymbol{q}]$$

We want to maximize this

$$\max_{\mathbf{y}} \rho[\mathbf{p}(\mathbf{y}), \mathbf{q}]$$

Fully expanded linearized objective

$$\rho[\mathbf{p}(\mathbf{y}), \mathbf{q}] \approx \frac{1}{2} \sum_m \sqrt{p_m(\mathbf{y}_0) q_m} + \frac{C_h}{2} \sum_n w_n k \left( \left\| \frac{\mathbf{y} - \mathbf{x}_n}{h} \right\|^2 \right)$$

where  $w_n = \sum_m \sqrt{\frac{q_m}{p_m(\mathbf{y}_0)}} \delta[b(\mathbf{x}_n) - m]$

We want to maximize this

$$\max_{\mathbf{y}} \rho[\mathbf{p}(\mathbf{y}), \mathbf{q}]$$

Fully expanded linearized objective

$$\rho[\mathbf{p}(\mathbf{y}), \mathbf{q}] \approx \frac{1}{2} \sum_m \sqrt{p_m(\mathbf{y}_0) q_m} + \frac{C_h}{2} \sum_n w_n k \left( \left\| \frac{\mathbf{y} - \mathbf{x}_n}{h} \right\|^2 \right)$$

doesn't depend on unknown  $\mathbf{y}$

where  $w_n = \sum_m \sqrt{\frac{q_m}{p_m(\mathbf{y}_0)}} \delta[b(\mathbf{x}_n) - m]$

We want to maximize this

$$\max_{\mathbf{y}} \rho[\mathbf{p}(\mathbf{y}), \mathbf{q}]$$

only need to  
maximize this!

Fully expanded linearized objective

$$\rho[\mathbf{p}(\mathbf{y}), \mathbf{q}] \approx \frac{1}{2} \sum_m \sqrt{p_m(\mathbf{y}_0) q_m} + \frac{C_h}{2} \sum_n w_n k \left( \left\| \frac{\mathbf{y} - \mathbf{x}_n}{h} \right\|^2 \right)$$

doesn't depend on unknown  $\mathbf{y}$

where  $w_n = \sum_m \sqrt{\frac{q_m}{p_m(\mathbf{y}_0)}} \delta[b(\mathbf{x}_n) - m]$

We want to maximize this

$$\max_{\mathbf{y}} \rho[\mathbf{p}(\mathbf{y}), \mathbf{q}]$$

Fully expanded linearized objective

$$\rho[\mathbf{p}(\mathbf{y}), \mathbf{q}] \approx \frac{1}{2} \sum_m \sqrt{p_m(\mathbf{y}_0) q_m} + \frac{C_h}{2} \sum_n w_n k \left( \left\| \frac{\mathbf{y} - \mathbf{x}_n}{h} \right\|^2 \right)$$

doesn't depend on unknown  $\mathbf{y}$

$$\text{where } w_n = \sum_m \sqrt{\frac{q_m}{p_m(\mathbf{y}_0)}} \delta[b(\mathbf{x}_n) - m]$$

what can we use to solve this weighted KDE?

**Mean Shift Algorithm!**

$$\frac{C_h}{2} \sum_n w_n k\left(\left\|\frac{\mathbf{y} - \mathbf{x}_n}{h}\right\|^2\right)$$

the new sample of mean of this KDE is

$$\mathbf{y}_1 = \frac{\sum_n \mathbf{x}_n w_n g\left(\left\|\frac{\mathbf{y}_0 - \mathbf{x}_n}{h}\right\|^2\right)}{\sum_n w_n g\left(\left\|\frac{\mathbf{y}_0 - \mathbf{x}_n}{h}\right\|^2\right)}$$

(this was derived earlier)

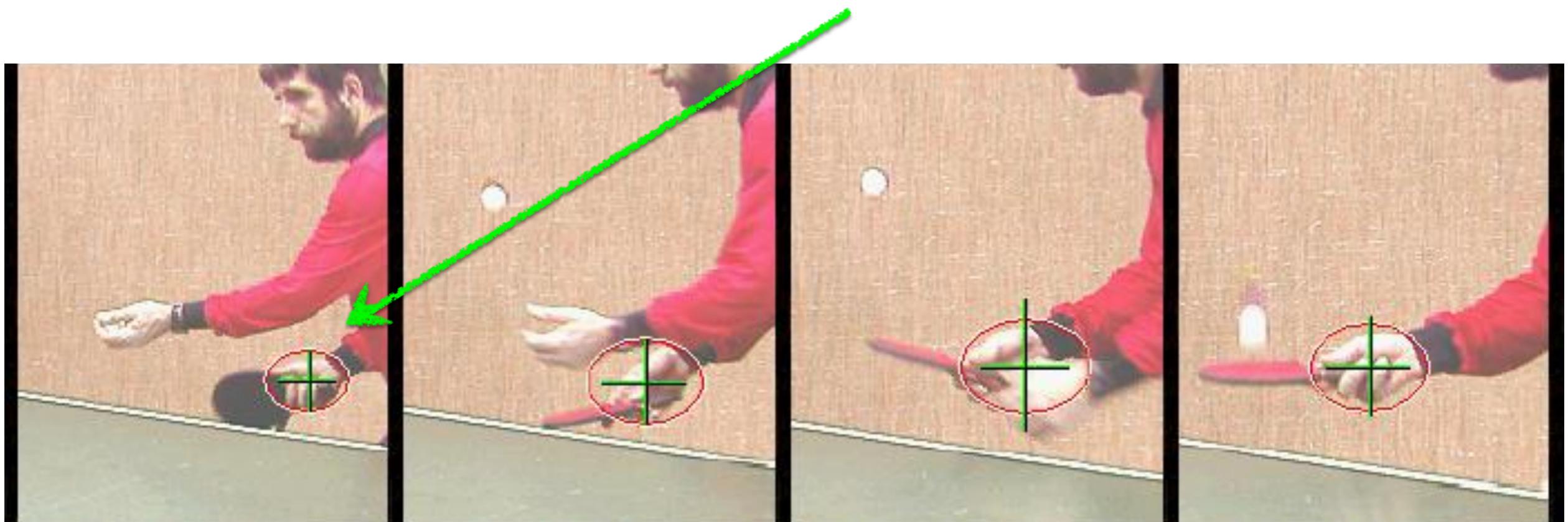
(new candidate location)

# Mean-Shift Object Tracking

For each frame:

1. Initialize location  $\mathbf{y}_0$   
Compute  $\mathbf{q}$   
Compute  $\mathbf{p}(\mathbf{y}_0)$
2. Derive weights  $w_n$
3. Shift to new candidate location (mean shift)  $\mathbf{y}_1$
4. Compute  $\mathbf{p}(\mathbf{y}_1)$
5. If  $\|\mathbf{y}_0 - \mathbf{y}_1\| < \epsilon$  return  
Otherwise  $\mathbf{y}_0 \leftarrow \mathbf{y}_1$  and go back to 2

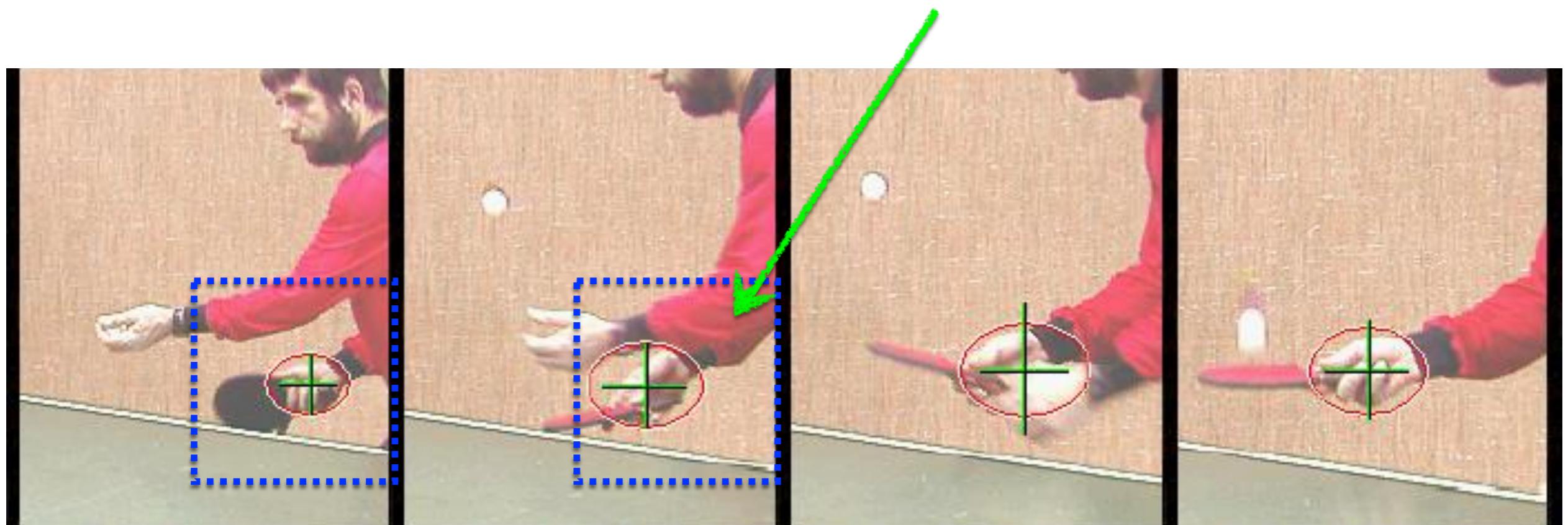
Compute a descriptor for the target



Target

$q$

Search for similar descriptor in neighborhood in next frame

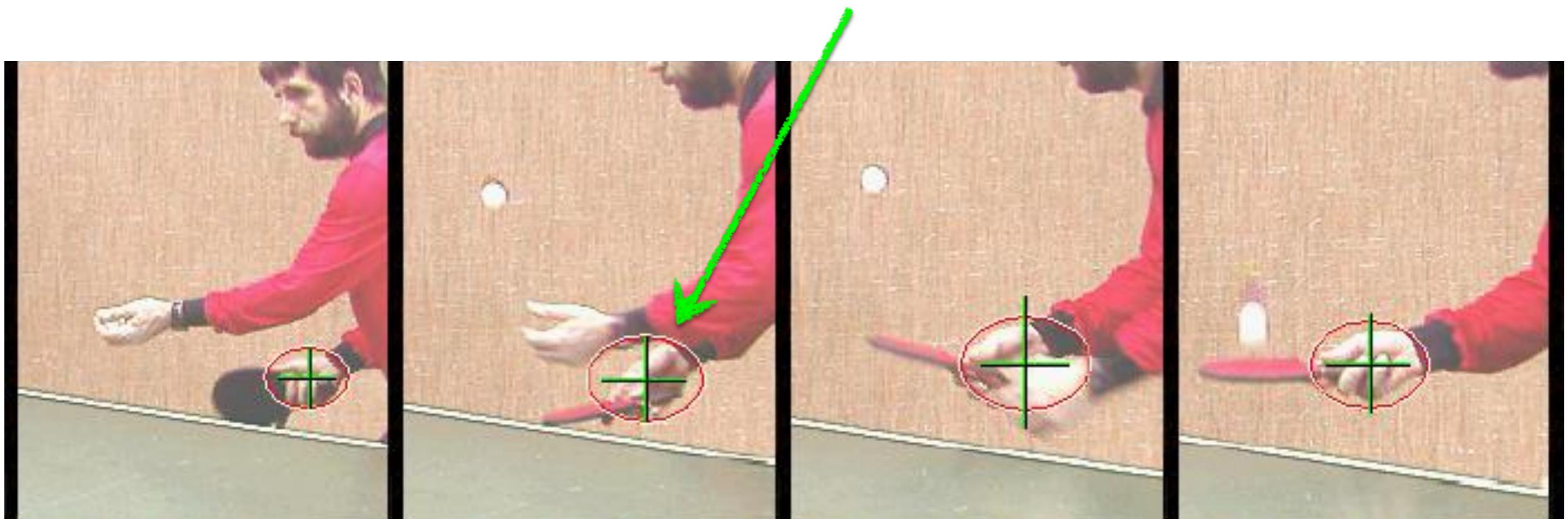


Target

Candidate

$$\max_{\mathbf{y}} \rho[\mathbf{p}(\mathbf{y}), \mathbf{q}]$$

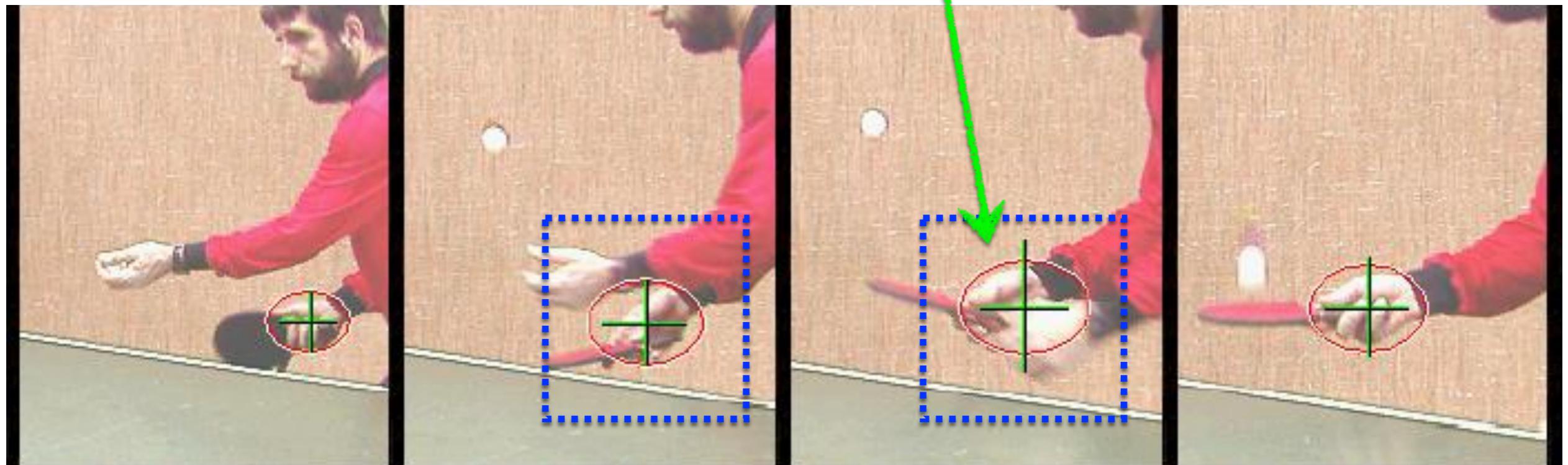
Compute a descriptor for the new target



Target

$q$

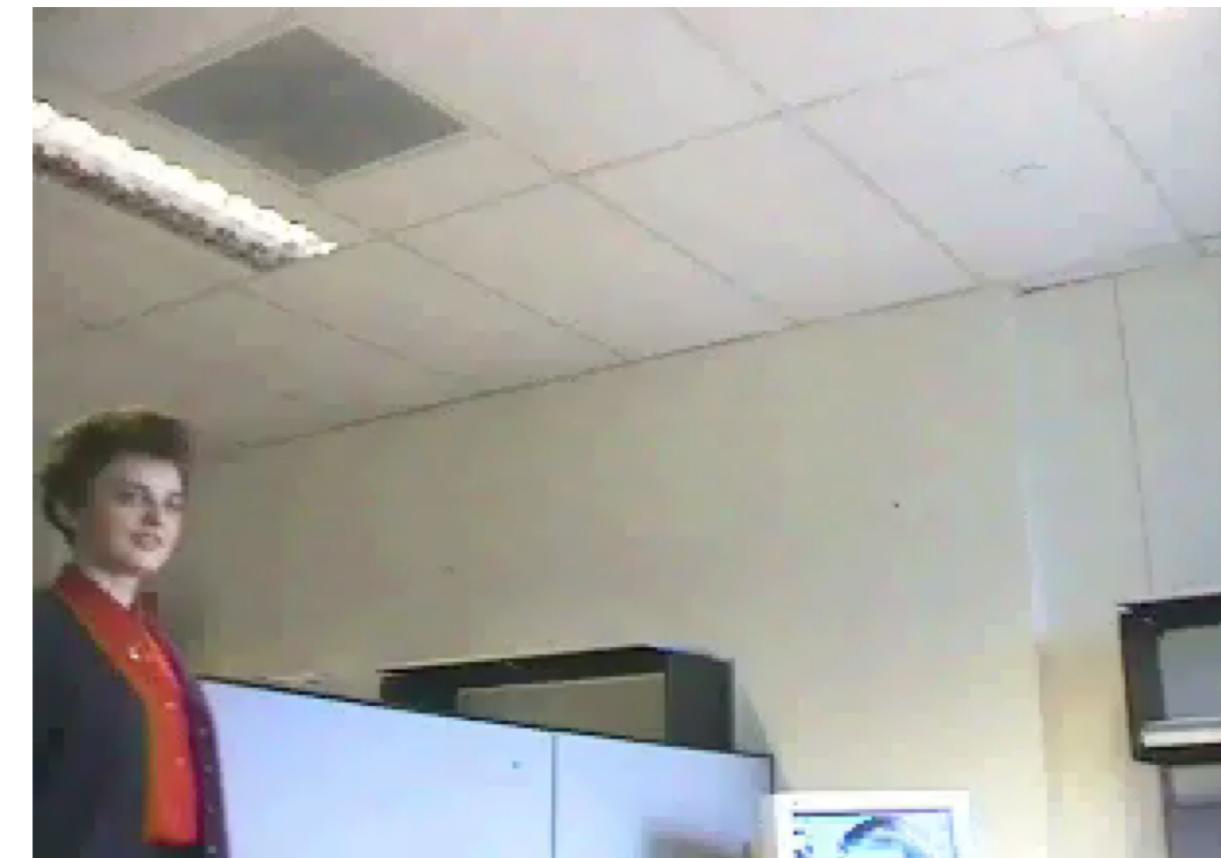
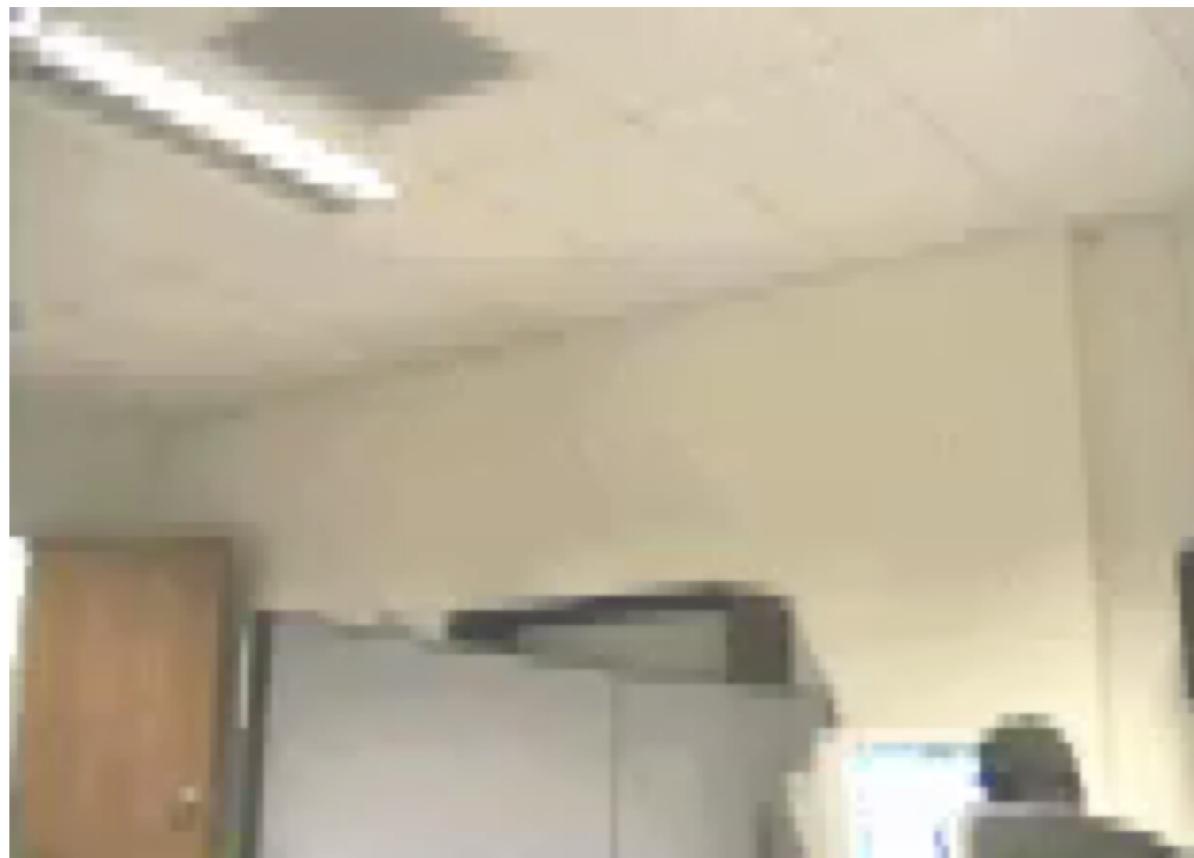
Search for similar descriptor in neighborhood in next frame



Target

Candidate

$$\max_{\mathbf{y}} \rho[\mathbf{p}(\mathbf{y}), \mathbf{q}]$$



# Modern trackers



# Learning Multi-Domain Convolutional Neural Networks for Visual Tracking

Hyeonseob Nam and Bohyung Han

# References

Basic reading:

- Szeliski, Sections 4.1.4, 5.3, 8.1.