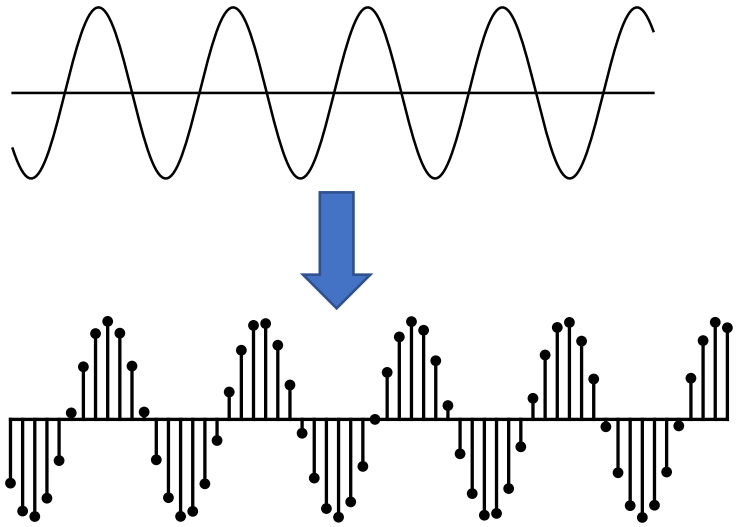
Image resizing

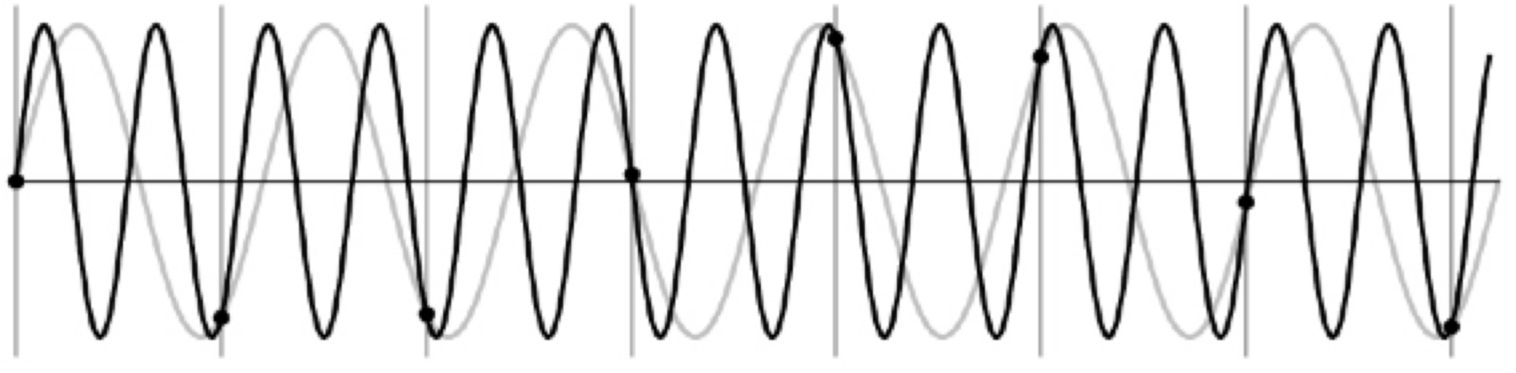
Image resizing

What is a (digital) image?

- True image is a function from R^2 to R
- Digital image is a sample from it
- 1D example:



Undersampling

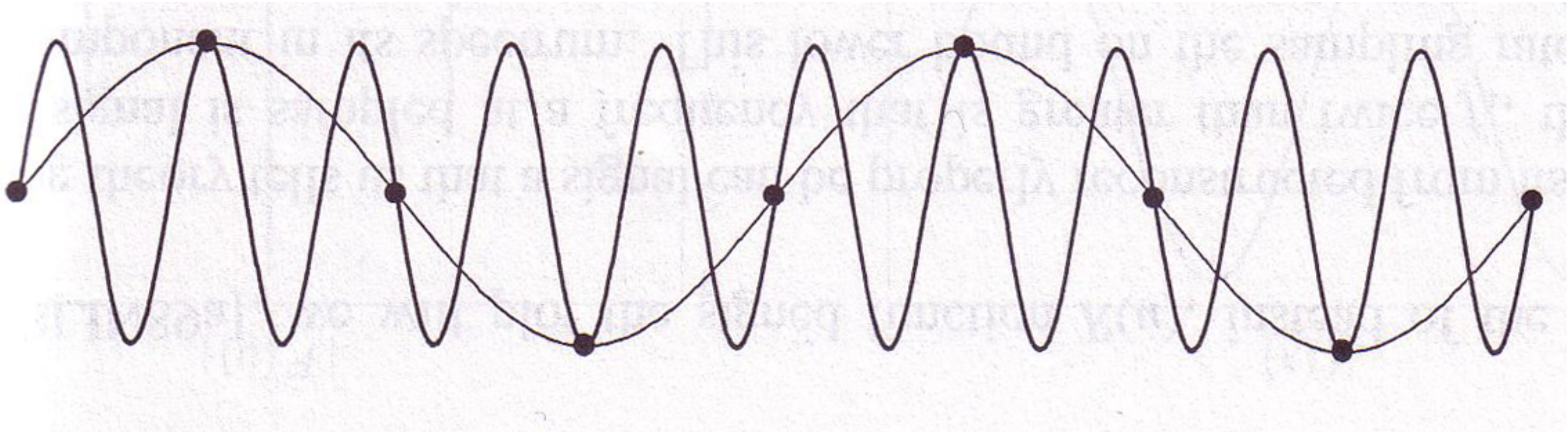


Undersampling

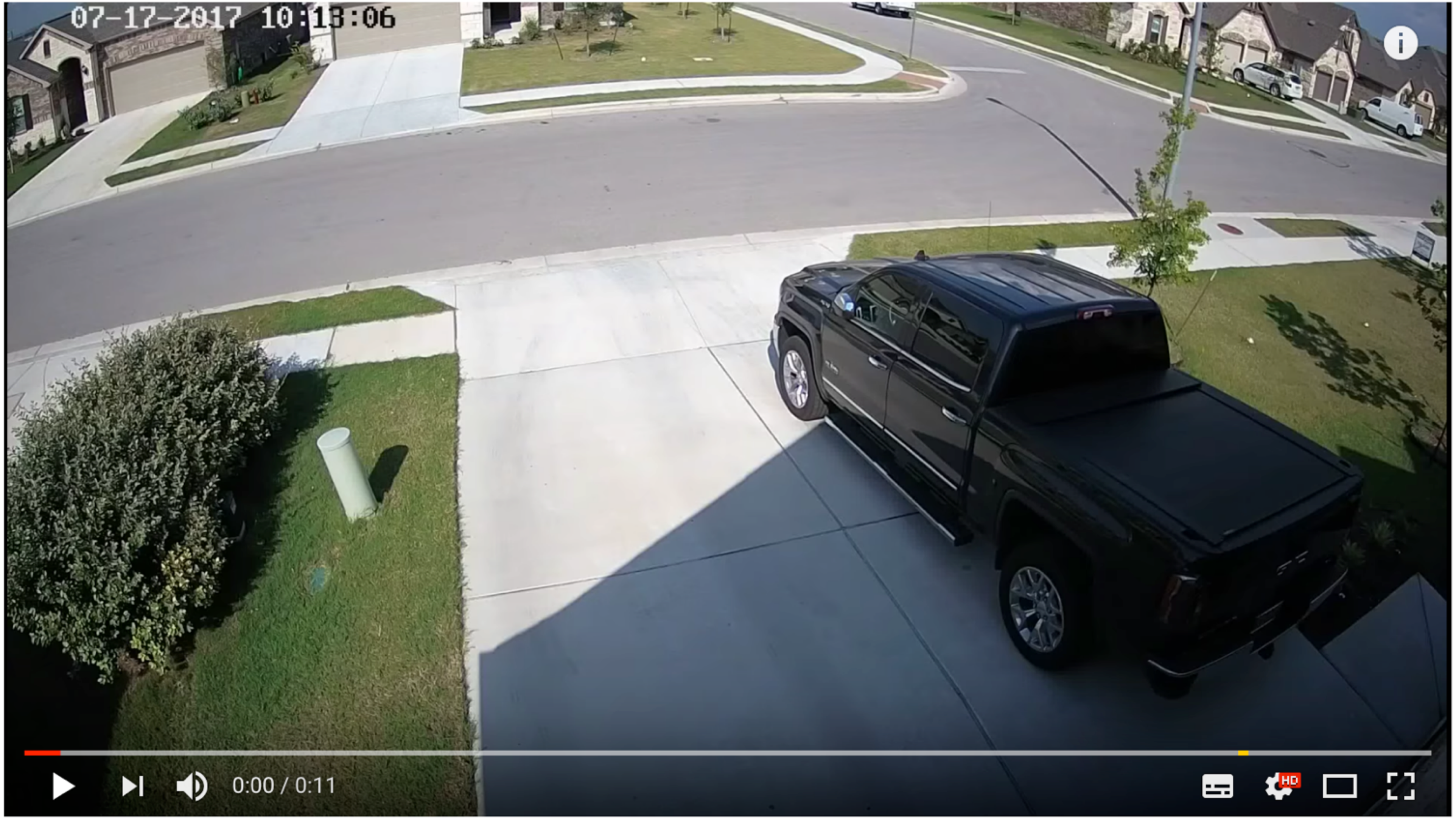
- What if we “missed” things between the samples?
- Simple example: undersampling a sine wave
  - unsurprising result: information is lost
  - surprising result: indistinguishable from lower frequency
  - also was always indistinguishable from higher frequencies
  - *aliasing*: signals “traveling in disguise” as other frequencies

Aliasing

- When sampling is not adequate, impossible to distinguish between low and high frequency signal



Aliasing in time

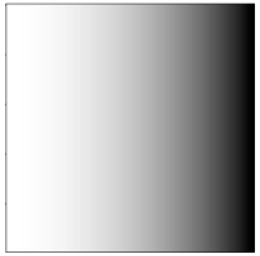


Beyond sines and cosines

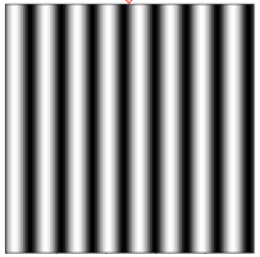
- Images are not sines and cosines
- But they can be written as a *sum* sines (**Fourier Transform**)

High frequency components

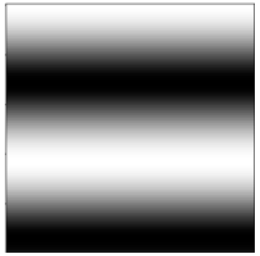
= 30*



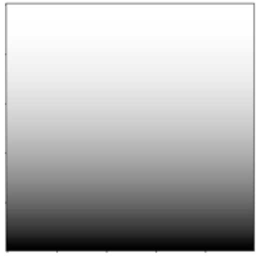
+ 0.5*



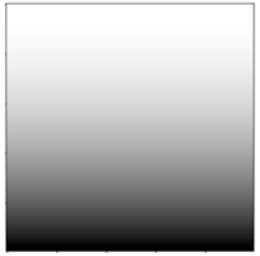
-



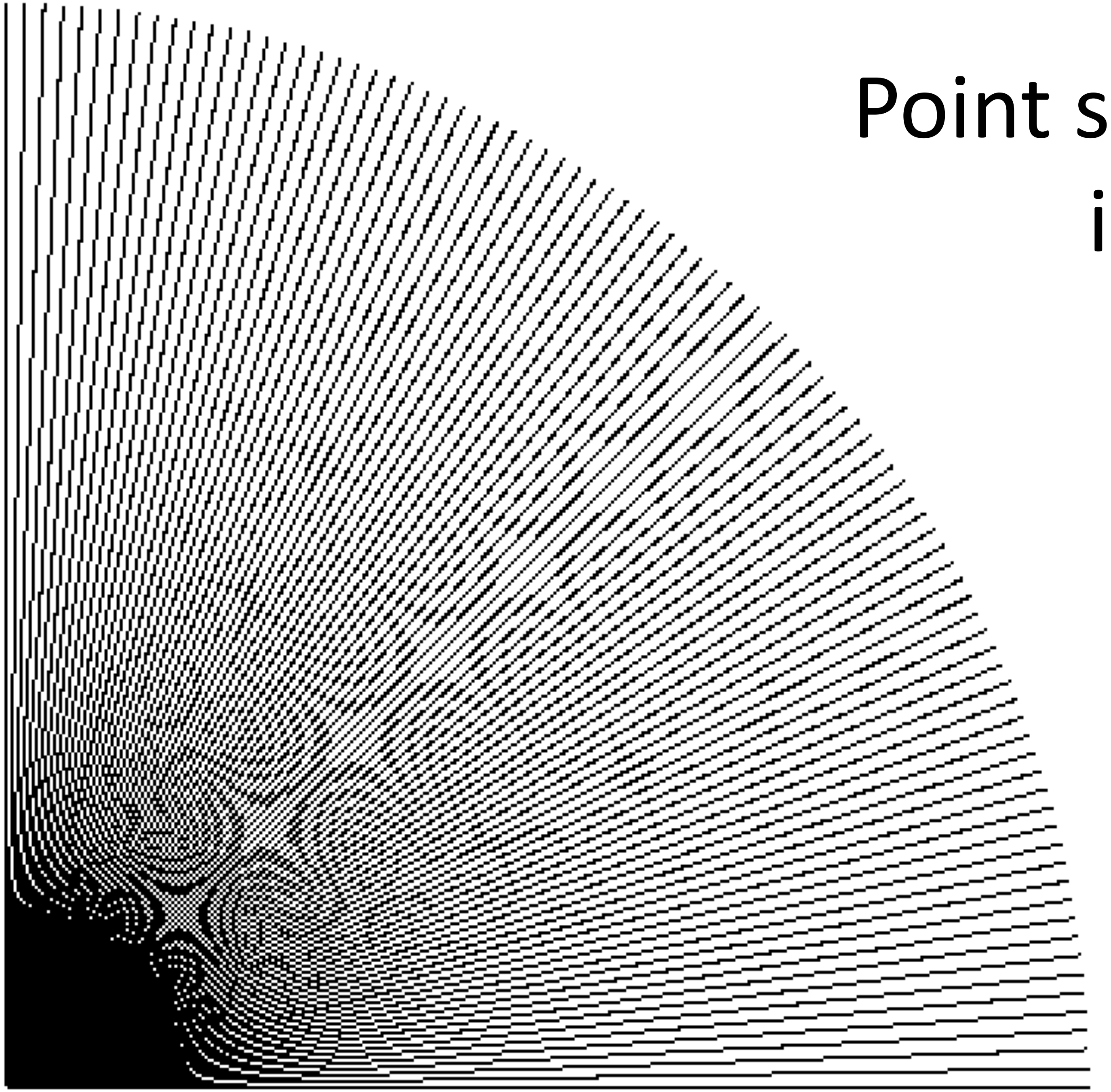
+ 100*



+10*



Point sampling in action



Aliasing

- High frequency components when downsampled *masquerade* as low frequency components
- Key step: remove high frequency components. But how?

Smoothing and image frequencies

- Smoothing makes a pixel more like its neighbors
- Image intensities change more slowly with x and y in smoothed image
- ➔ Smoothing *removes high frequencies*

Subsampling images

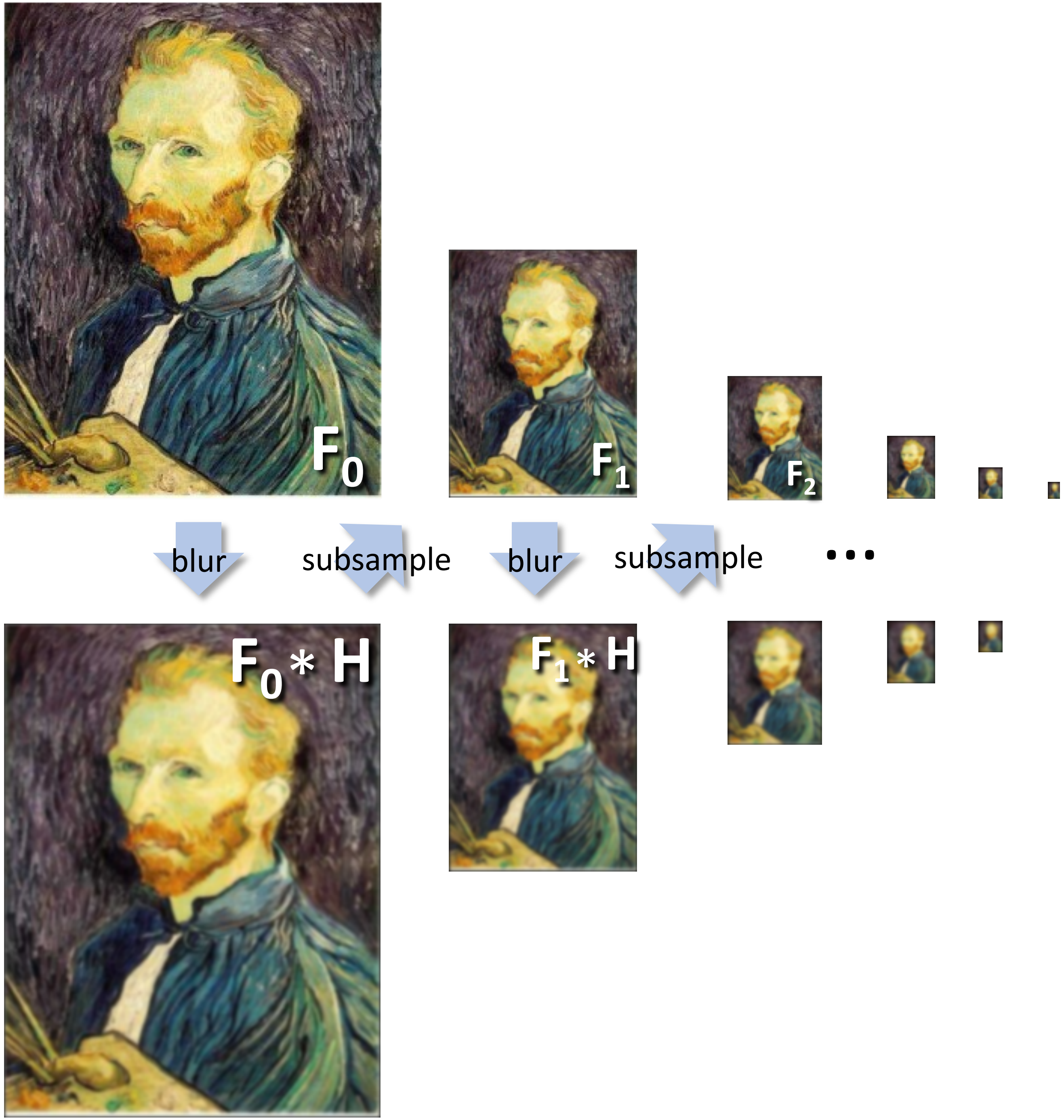
- Step 1: Convolve with Gaussian to eliminate high frequencies
- Step 2: Drop unneeded pixels

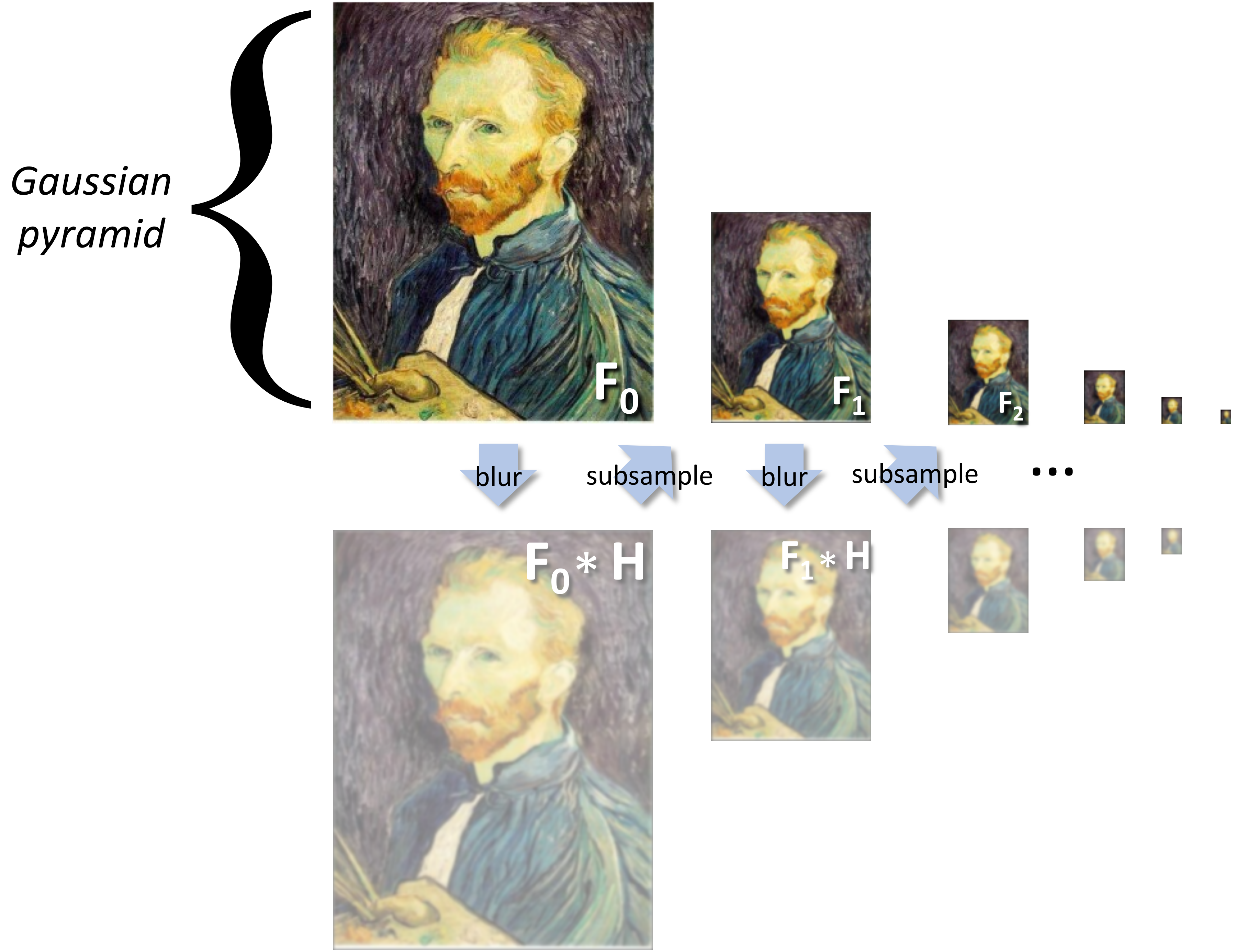
Subsampling without removing high frequencies

Subsampling after removing high frequencies

Gaussian pre-filtering

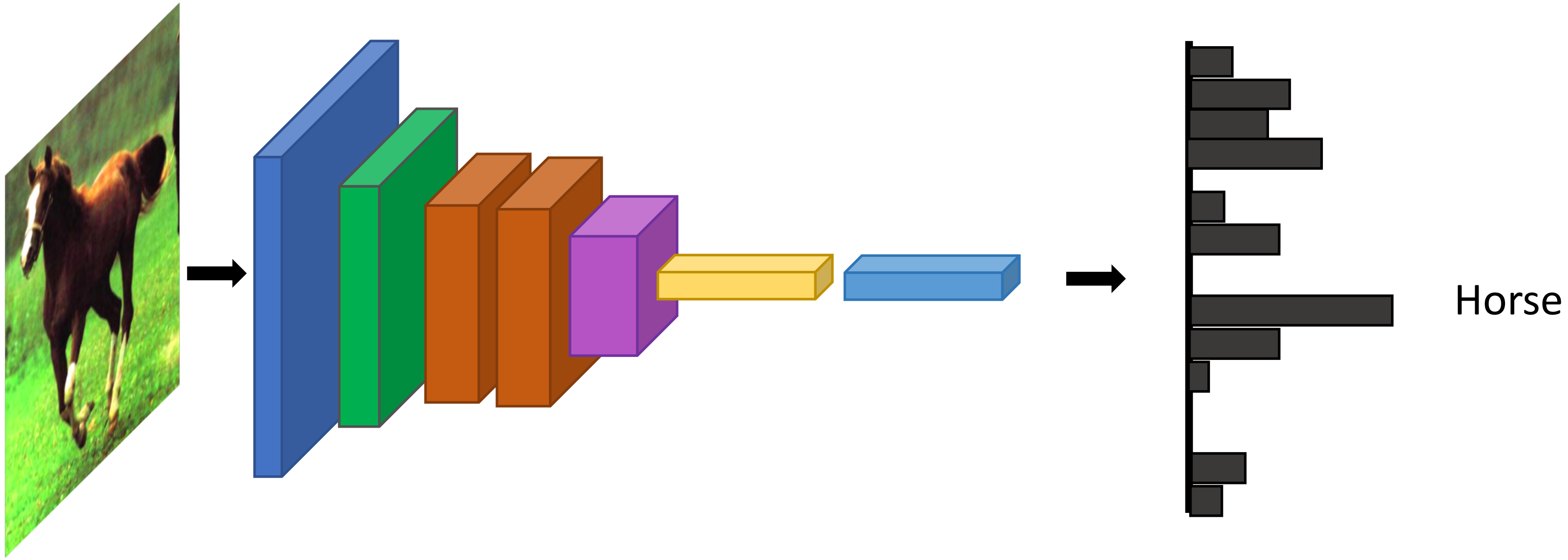
- Solution: filter the image, *then* subsample





Gaussian pyramid
F_0
F_1
F_2
blur
subsample
blur
subsample
...
$F_0 * H$
$F_1 * H$

Convolution and subsampling is familiar…

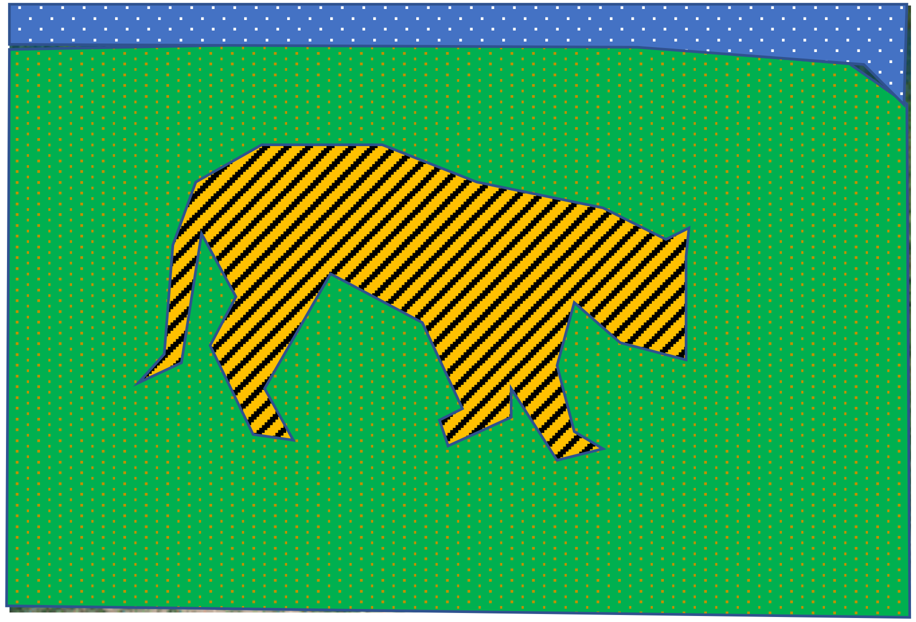


Key take-away

- A versatile tool : convolution
  - Any linear shift-invariant operation is a convolution
  - In particular edge detection, image smoothing
- A versatile structure : pyramids
  - Early layers capture low-level detail, higher layers capture global structure.

Grouping

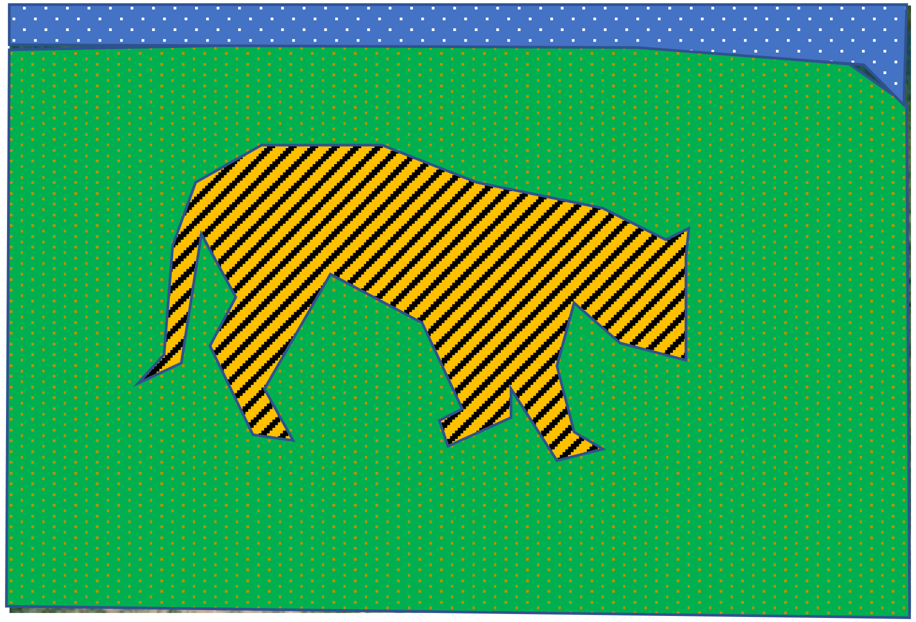
What is grouping?

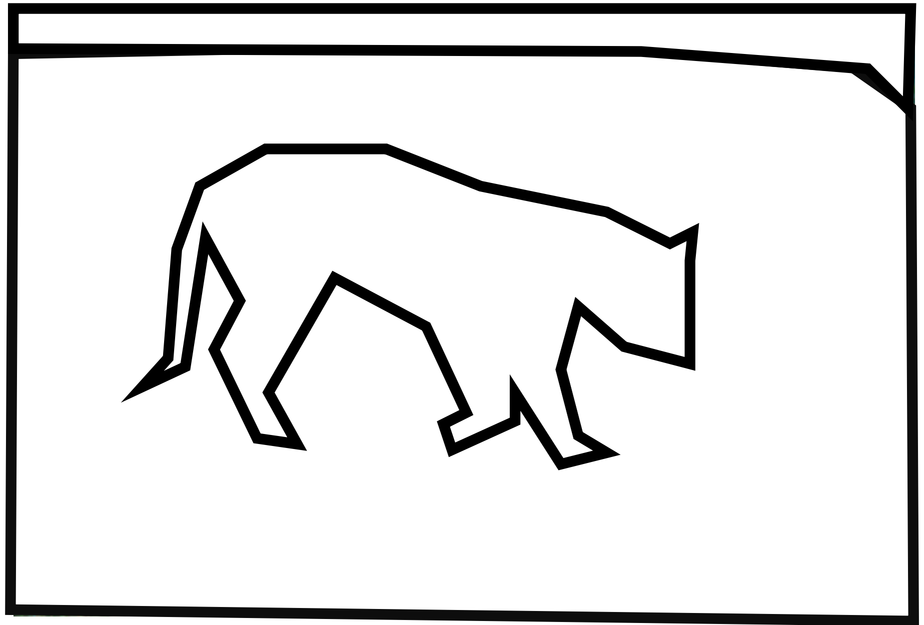


Why grouping?

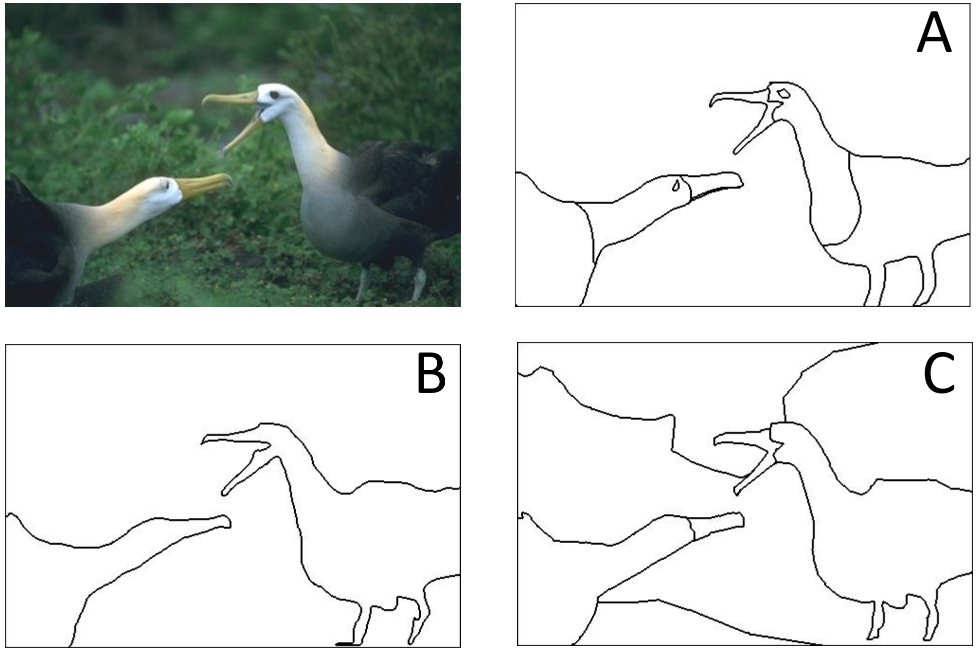
- Pixels property of sensor, not world
- Reasoning at object level (might) make things easy:
  - objects at consistent depth
  - objects can be recognized
  - objects move as one

Regions ⟷ Boundaries



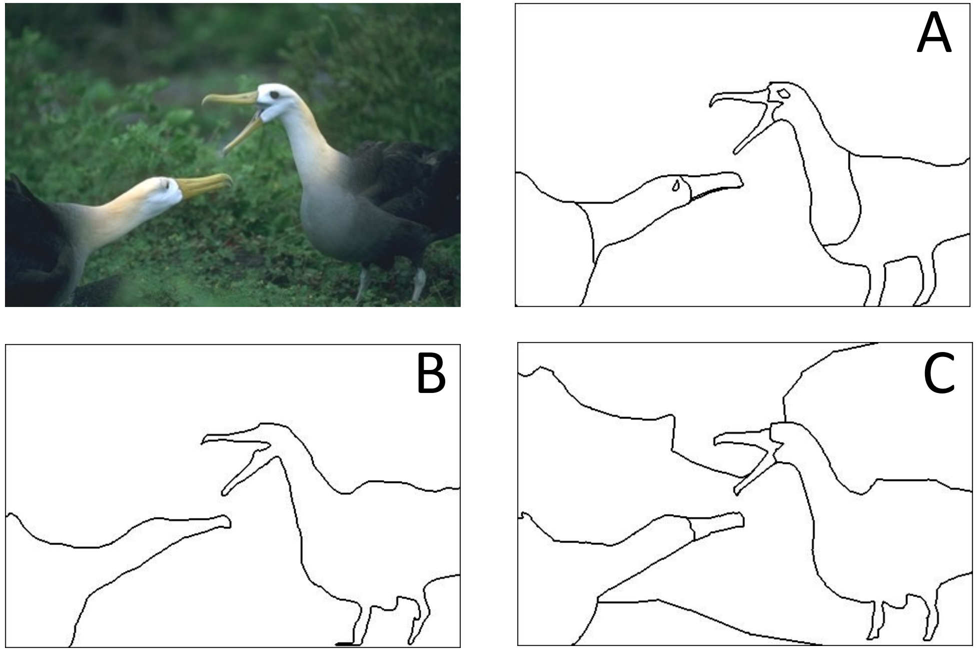


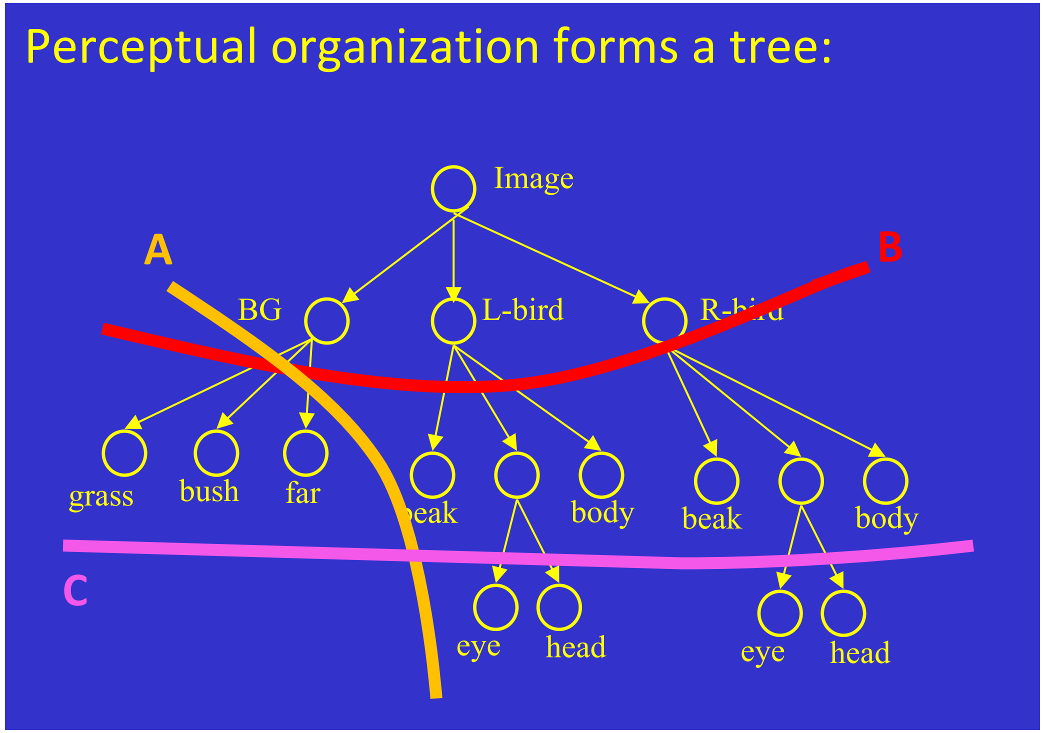
Is grouping well-defined?



- Depends on purpose
  - Object parts
  - Background segmentation

Is grouping well-defined?

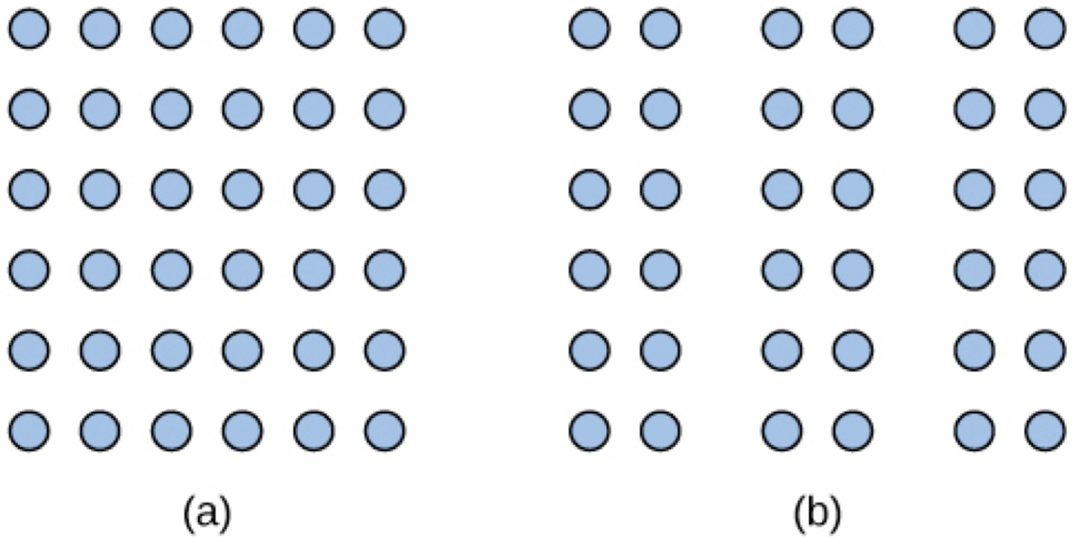




D. Martin, C. Fowlkes, D. Tal, J. Malik. "A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics", ICCV, 2001

How do we group things?

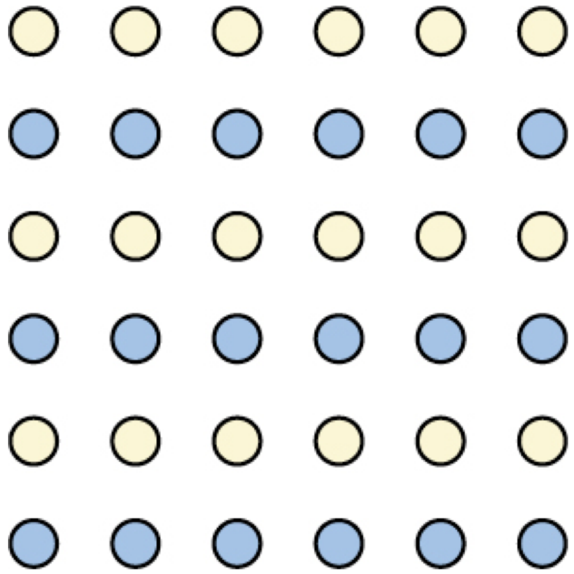
- *Gestalt* principles
- Principle of *proximity*



https://courses.lumenlearning.com/wsu-sandbox/chapter/gestalt-principles-of-perception/

How do we group things?

- Gestalt principles
- Principle of *similarity*



https://courses.lumenlearning.com/wsu-sandbox/chapter/gestalt-principles-of-perception/

How do we group things?

- Gestalt principles
- Principle of *continuity* and *closure*

https://courses.lumenlearning.com/wsu-sandbox/chapter/gestalt-principles-of-perception/

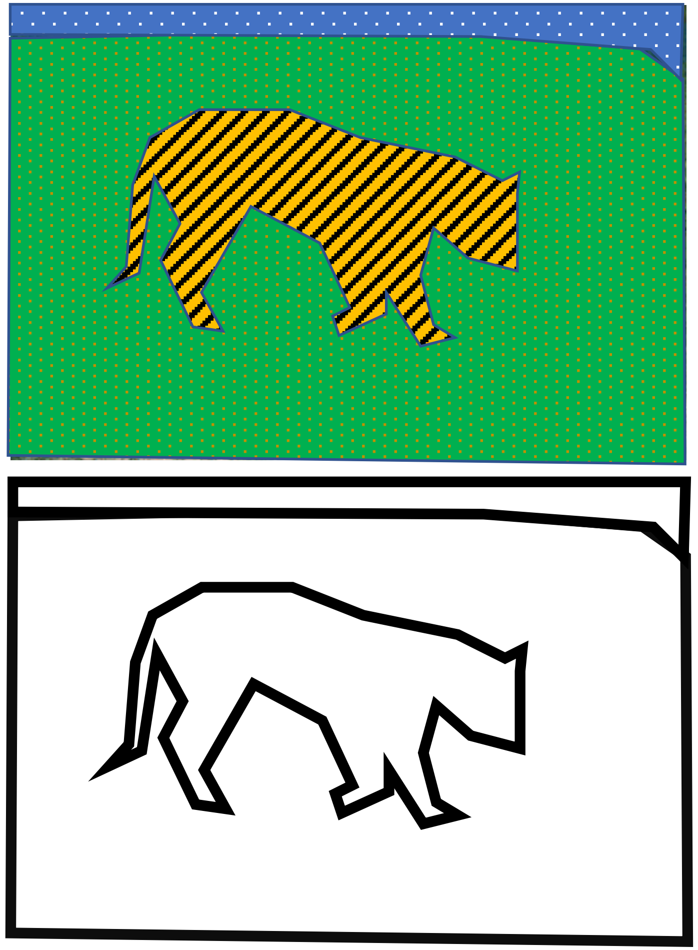
How do we group things?

- Gestalt principles
- Principle of *common fate*

Gestalt principles in the context of images

- Principle of proximity: nearby pixels are part of the same object
- Principle of similarity: similar pixels are part of the same object
  - Look for differences in color, intensity, or texture across the boundary
- Principle of closure and continuity: contours are likely to continue
- High-level knowledge?

Regions ⟷ Boundaries

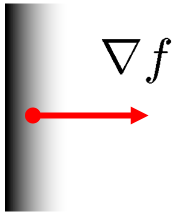


Discussion: Contour Detection and Hierarchical Image Segmentation

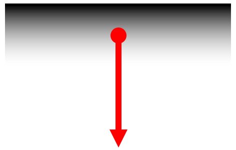
Image gradient

- The *gradient* of an image: $$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

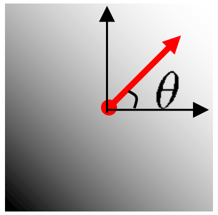
The gradient points in the direction of most rapid increase in intensity



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The *edge strength* is given by the gradient magnitude:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

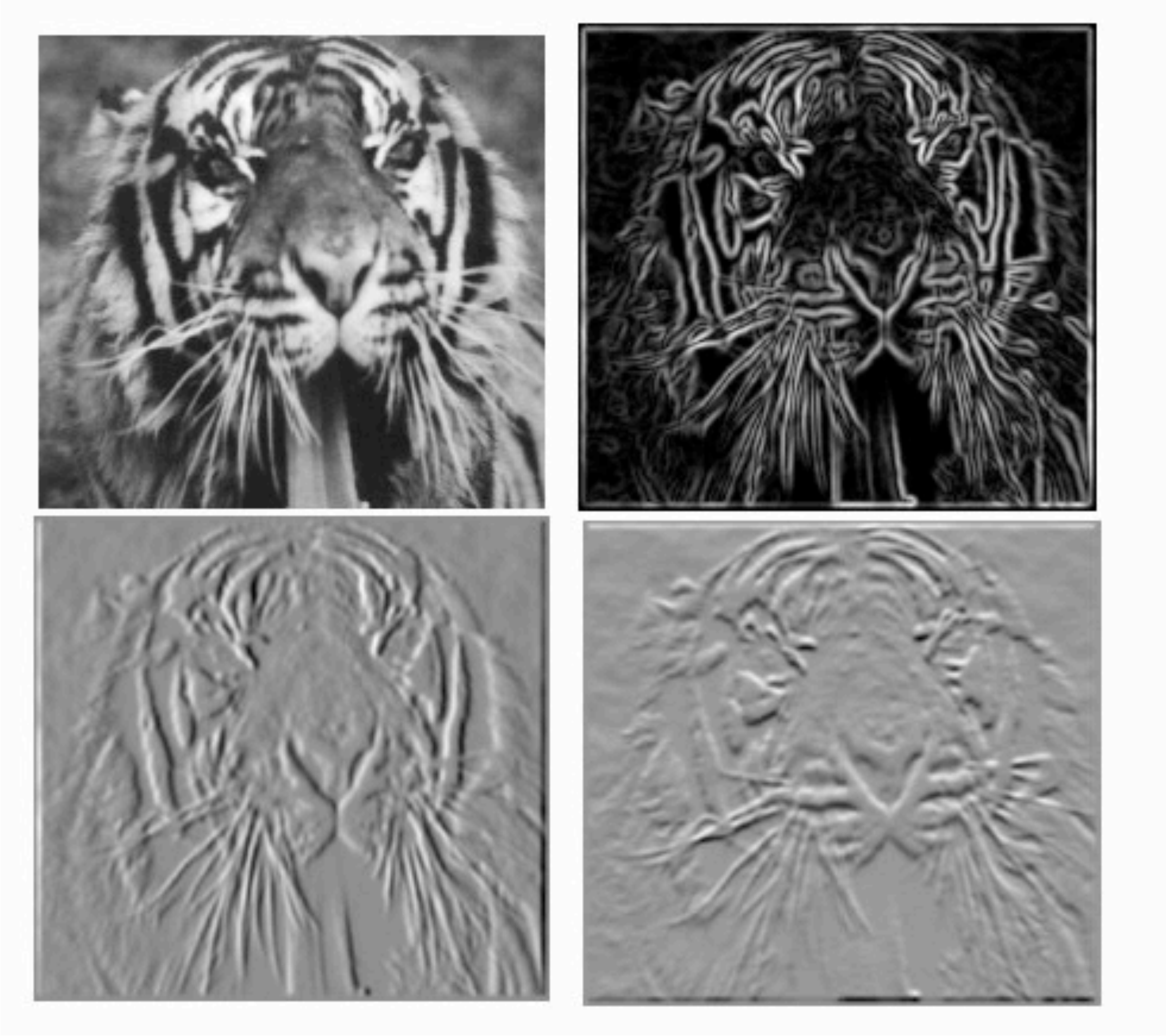
The gradient direction is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} \Big/ \frac{\partial f}{\partial x}\right)$$

- how does this relate to the direction of the edge?

Source: Steve Seitz

Image gradient



Source: L. Lazebnik

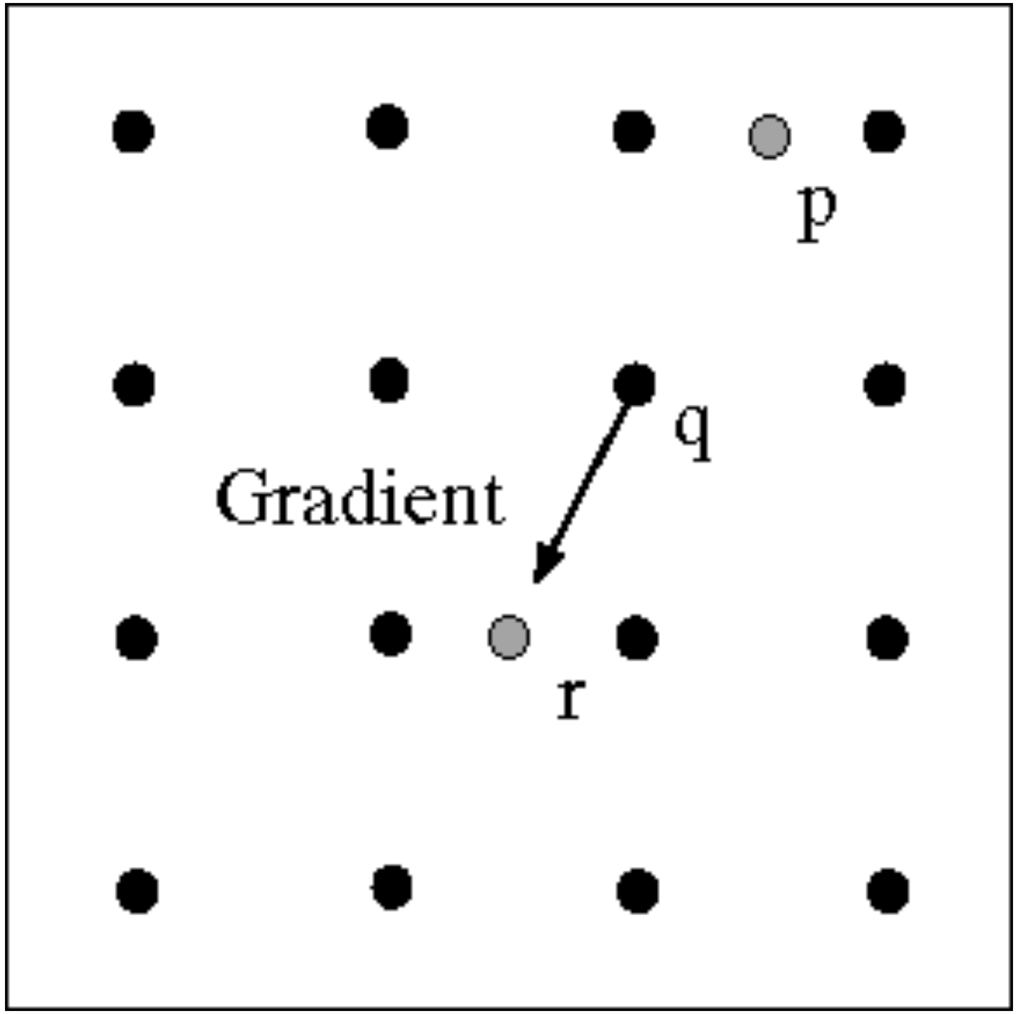
Gradient magnitude and orientation

- Orientation is undefined at pixels with 0 gradient

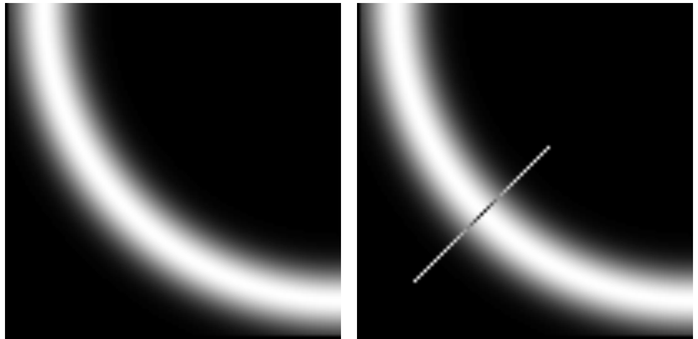
Magnitude

Orientation
theta = numpy.arctan2(gy, gx)

Non-maximum suppression for each orientation



At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.



Source: D. Forsyth

Before Non-max Suppression

After Non-max Suppression

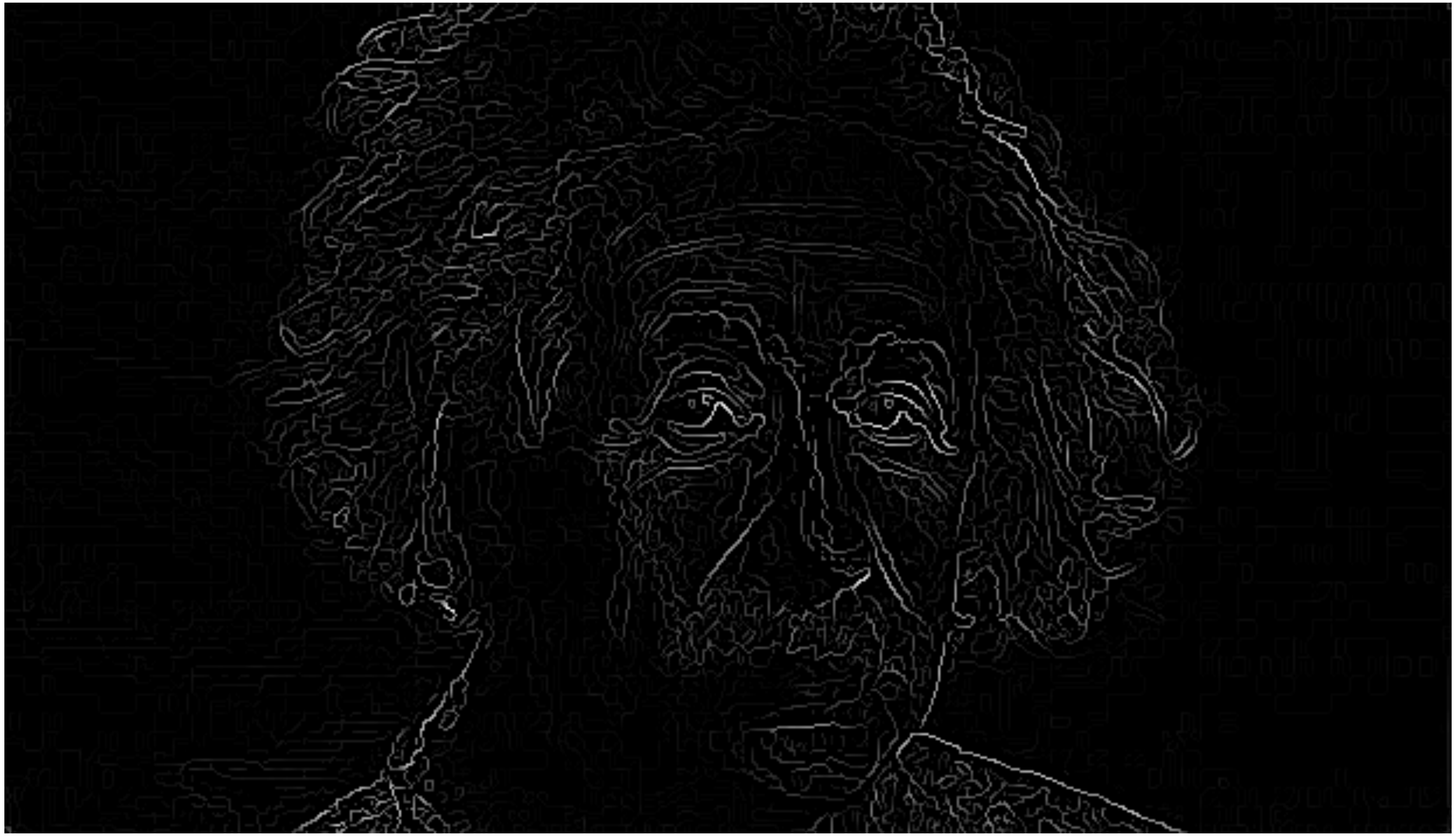
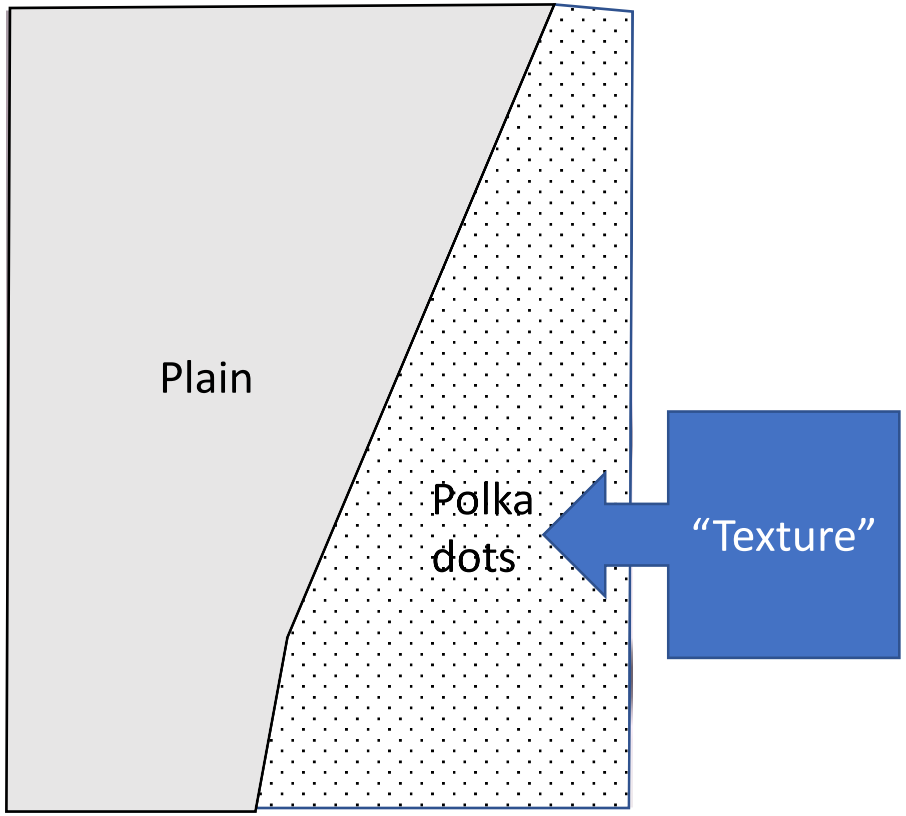


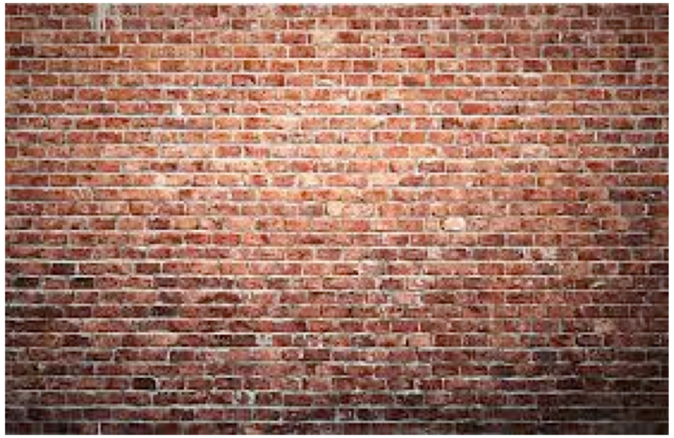
Image gradients are not enough

Image gradients are not enough



What is texture?




Same thing repeated over and over

What is texture?

Julesz's texton theory

- What is texture?
- Distributions of some elements
  - Elongated blobs of specific orientations, widths, lengths
  - Terminators (ends of line segments)
  - Crossings of line segments

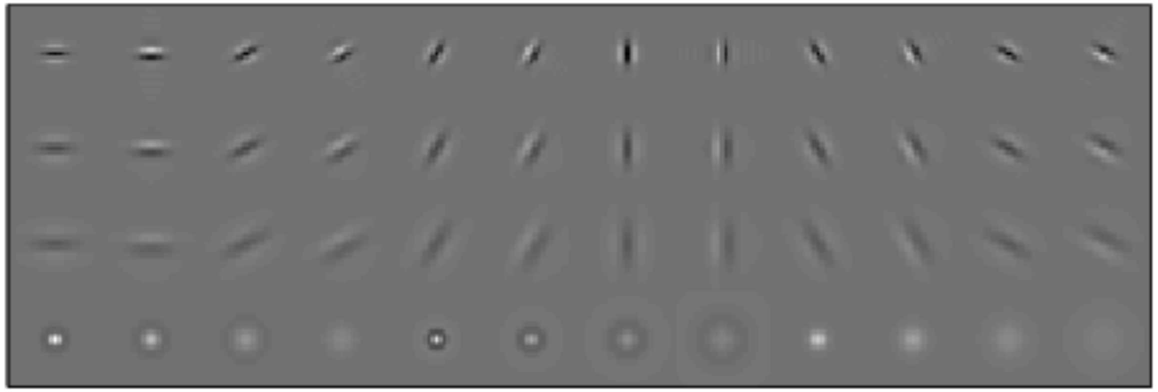
Textons

Bringing textons to computer vision

- Define a “vocabulary” of textons
- Describe texture by a distribution of different textons

Bringing textons to computer vision

- **Define a “vocabulary” of textons**
- Describe texture by a distribution of different textons

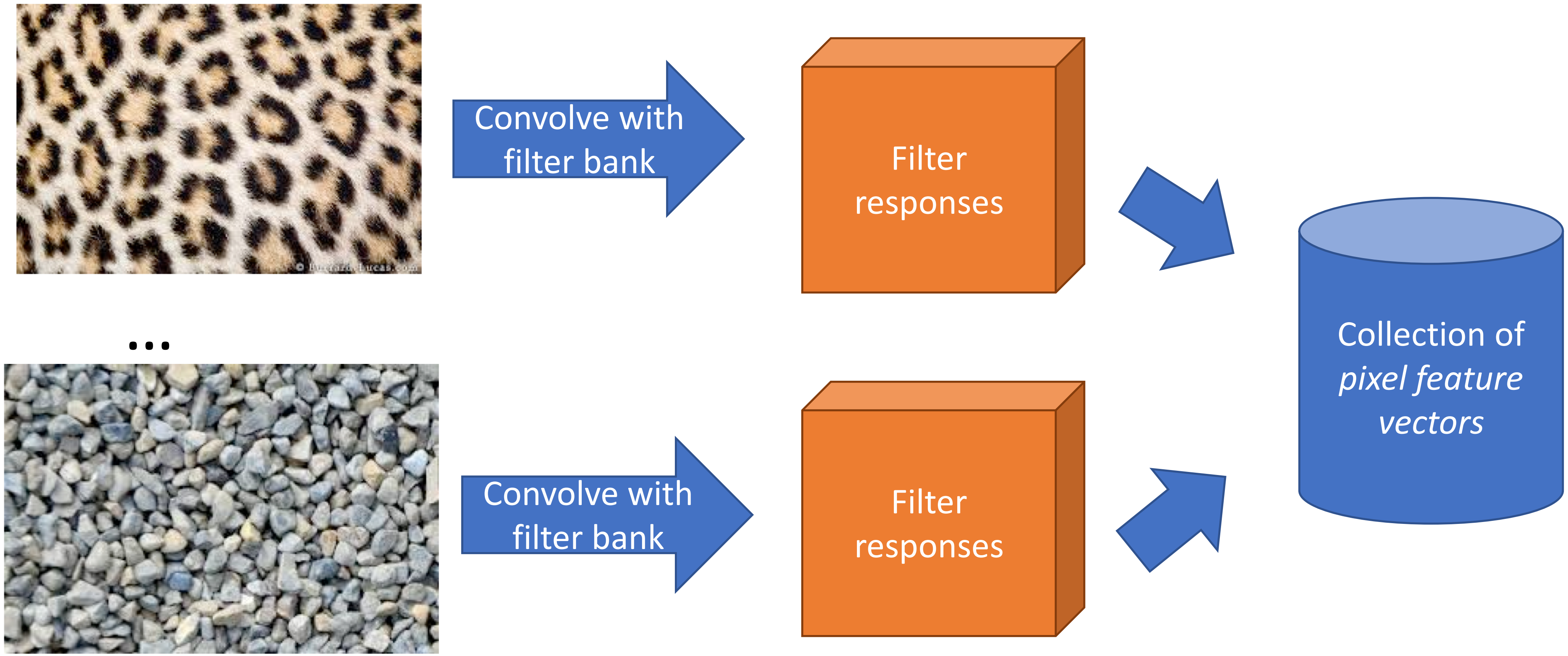


Convolve with filter bank

Filter responses

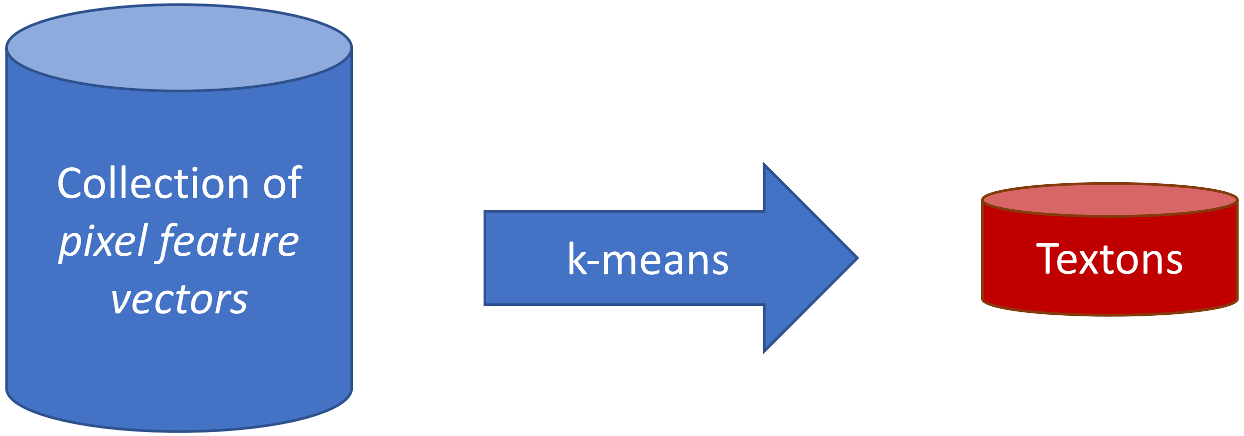
Bringing textons to computer vision

- **Define a "vocabulary" of textons**
- Describe texture by a distribution of different textons



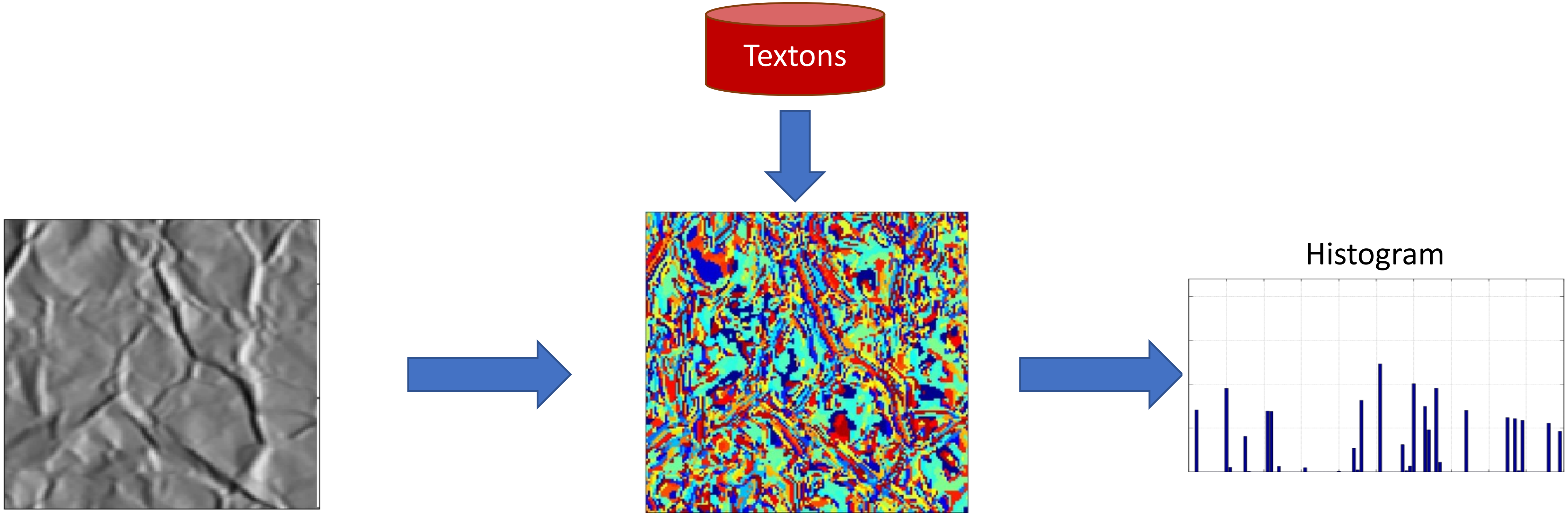
Bringing textons to computer vision

- **Define a “vocabulary” of textons**
- Describe texture by a distribution of different textons



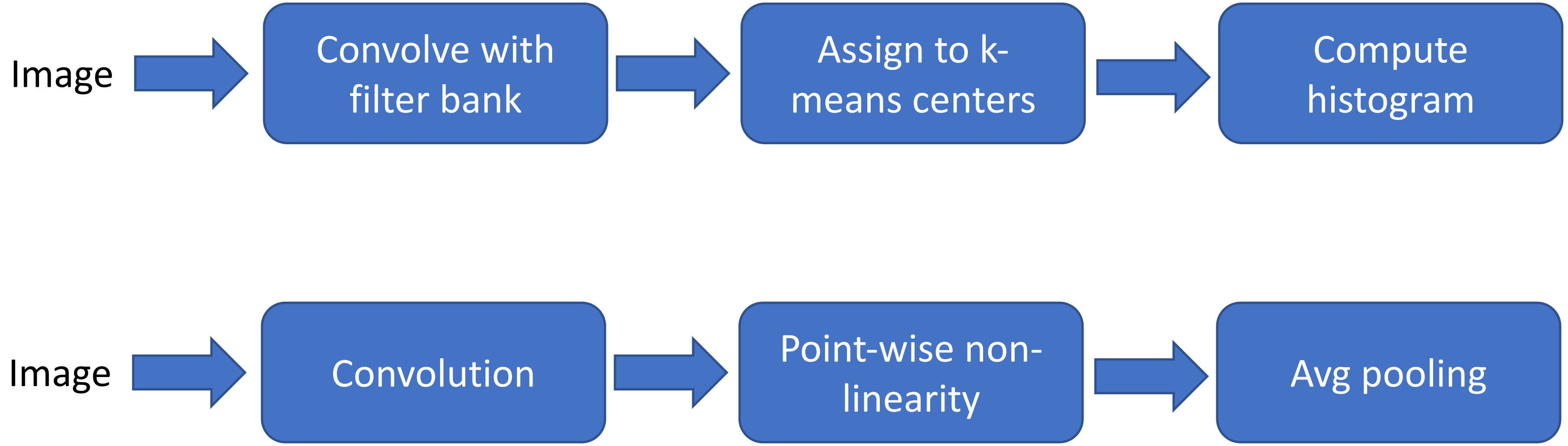
Bringing textons to computer vision

- Define a “vocabulary” of textons
- **Describe texture by a distribution of different textons**



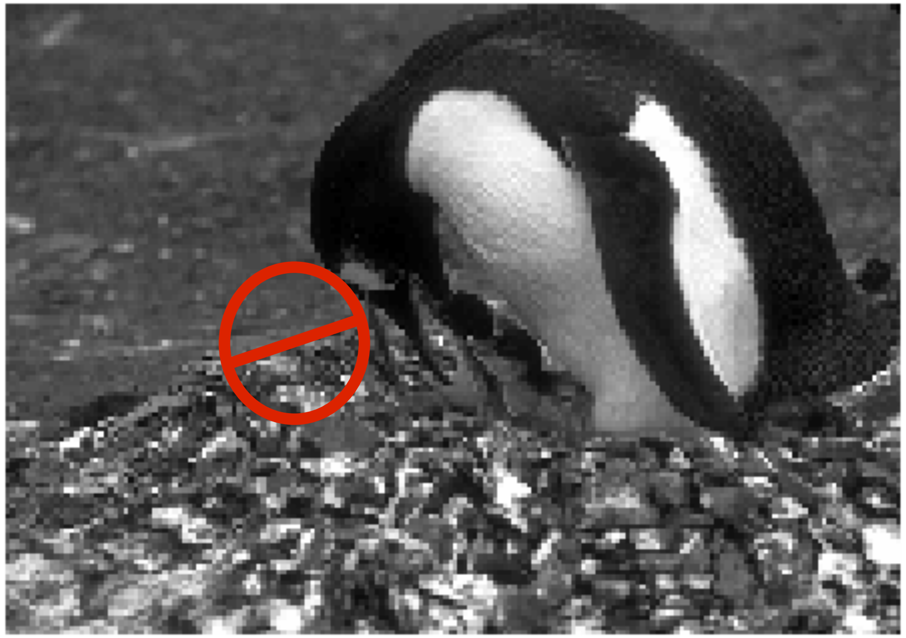
Leung, Thomas, and Jitendra Malik. "Representing and recognizing the visual appearance of materials using three-dimensional textons." *International journal of computer vision* 43.1 (2001): 29-44.

Textons in computer vision



Detecting texture boundaries

- Problem: gradient captures change from *pixel* to *pixel*
- *But texture property of region*



Martin, David R., Charless C. Fowlkes, and Jitendra Malik. "Learning to detect natural image boundaries using local brightness, color, and texture cues." *TPAMI (2004).*

Cue combination

Image

Boundary Cues

Brightness

Color

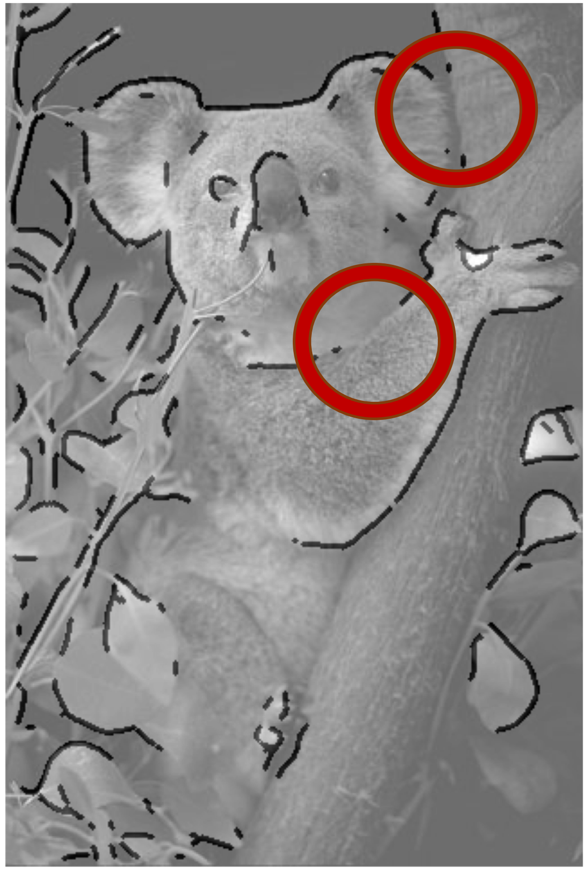
Texture

Average

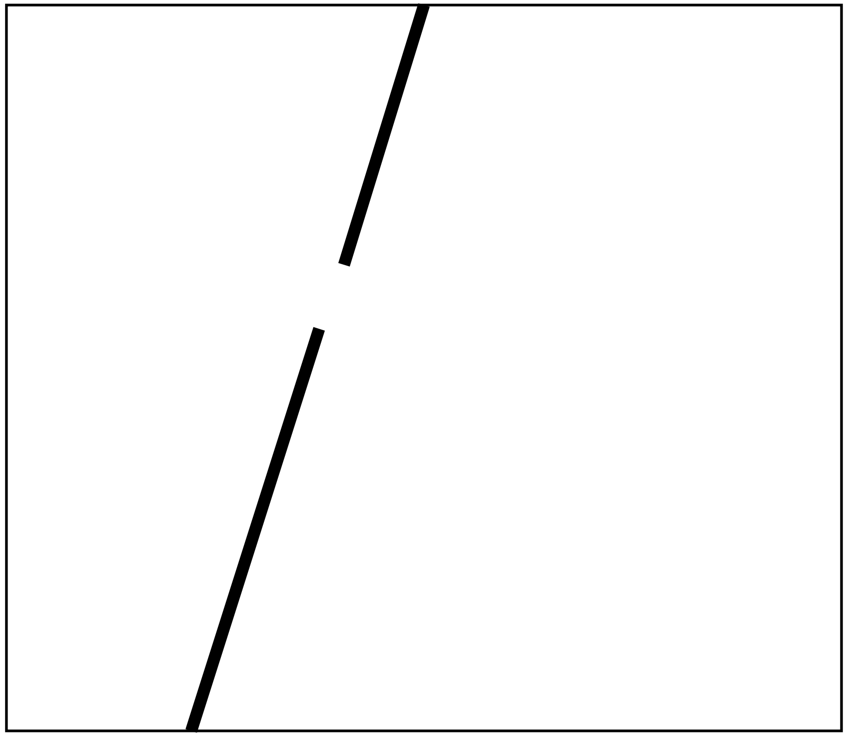
P_b

Martin, David R., Charless C. Fowlkes, and Jitendra Malik. "Learning to detect natural image boundaries using local brightness, color, and texture cues." *TPAMI (2004).*

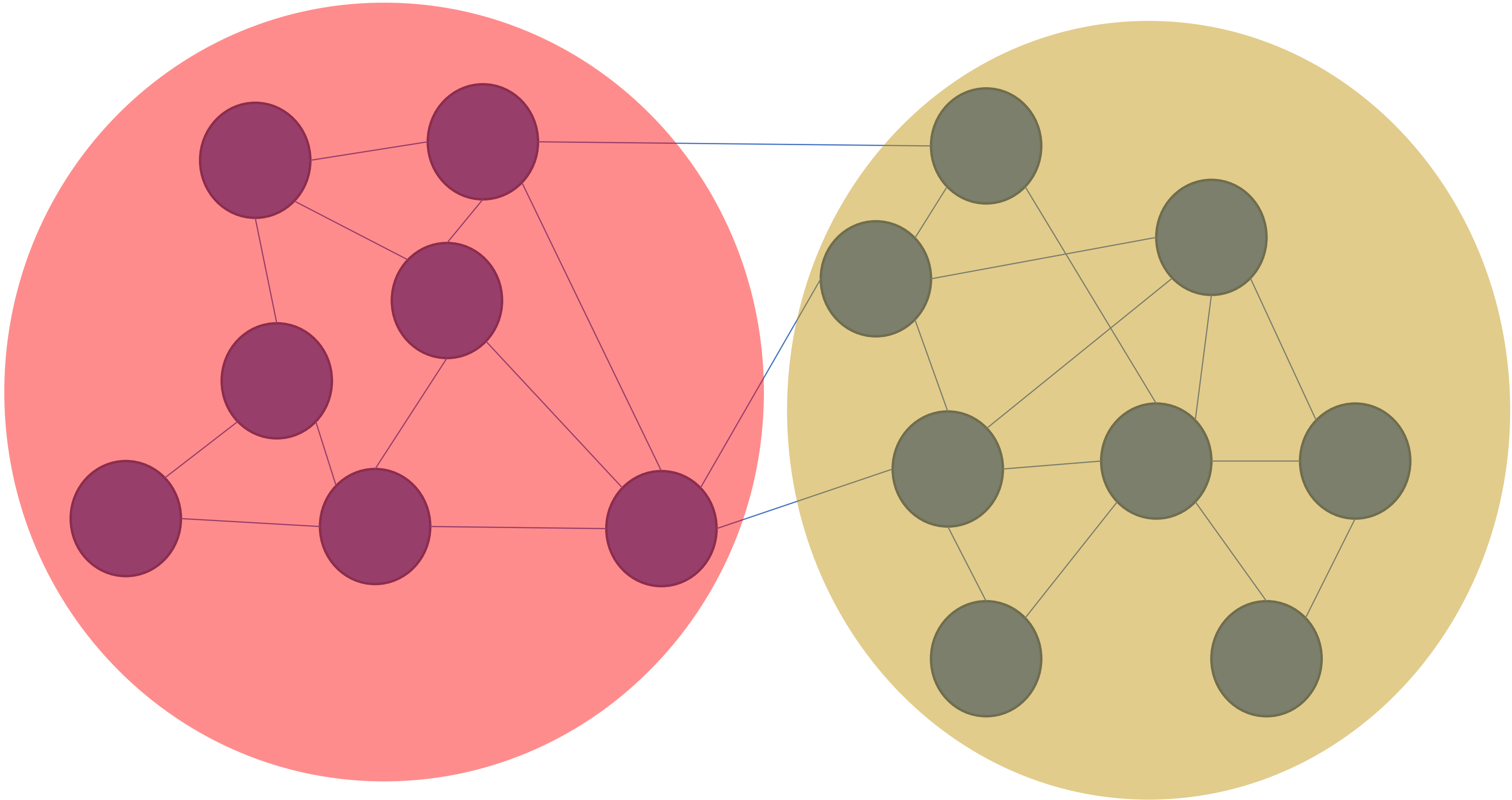
Local computation not enough



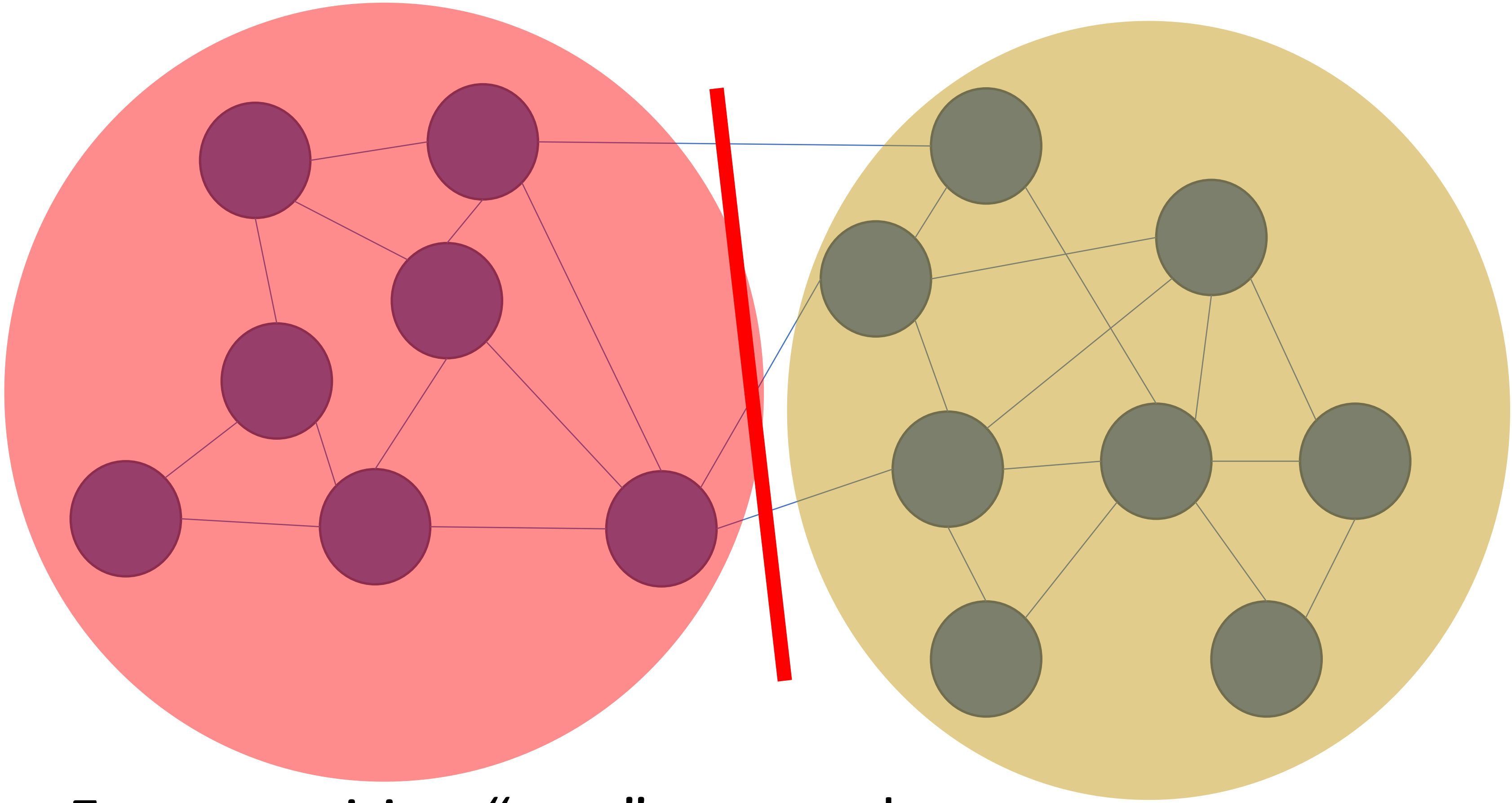
“Globalisation”



Segmentation is graph partitioning

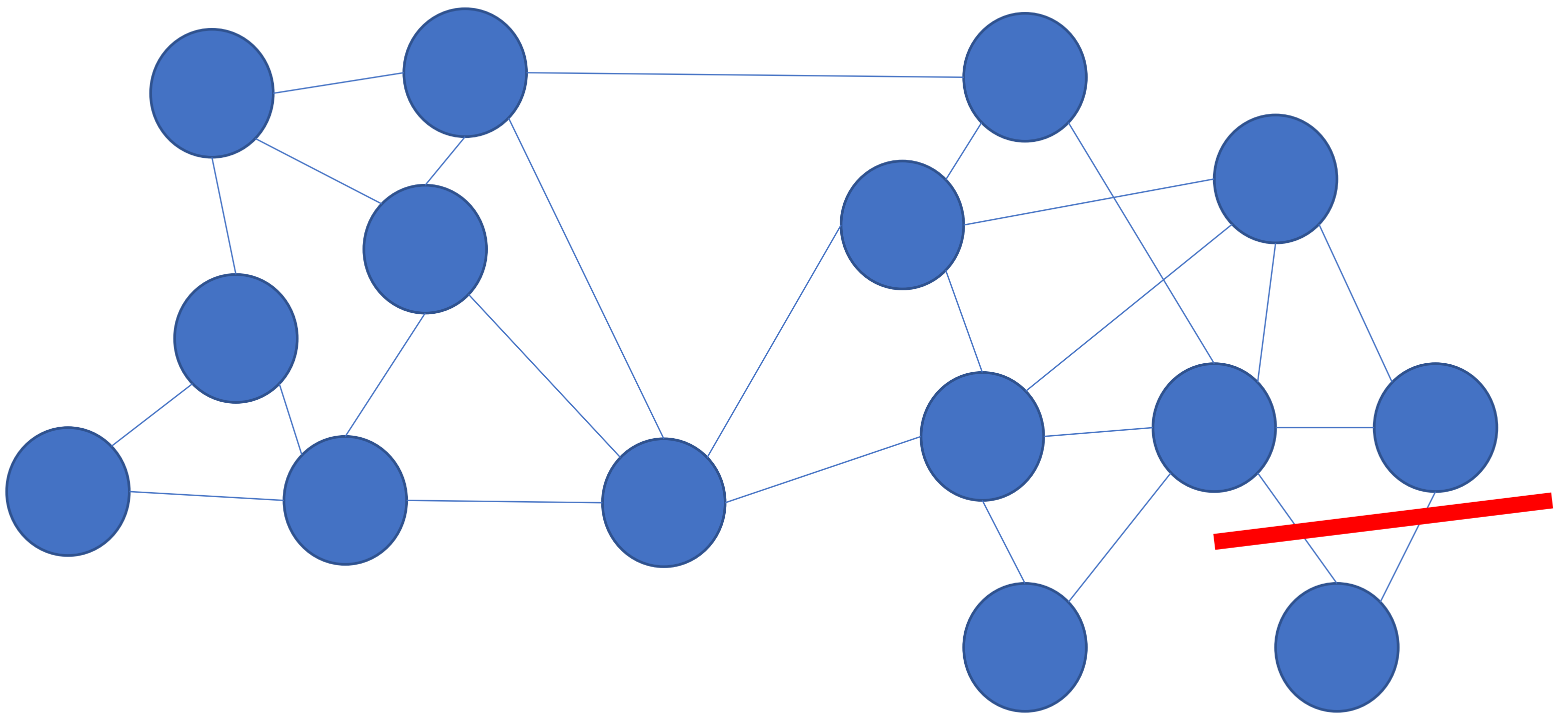


Segmentation is graph partitioning



- Every partition “cuts” some edges
- Idea: minimize total weight of edges cut!

Criterion: Min-cut?

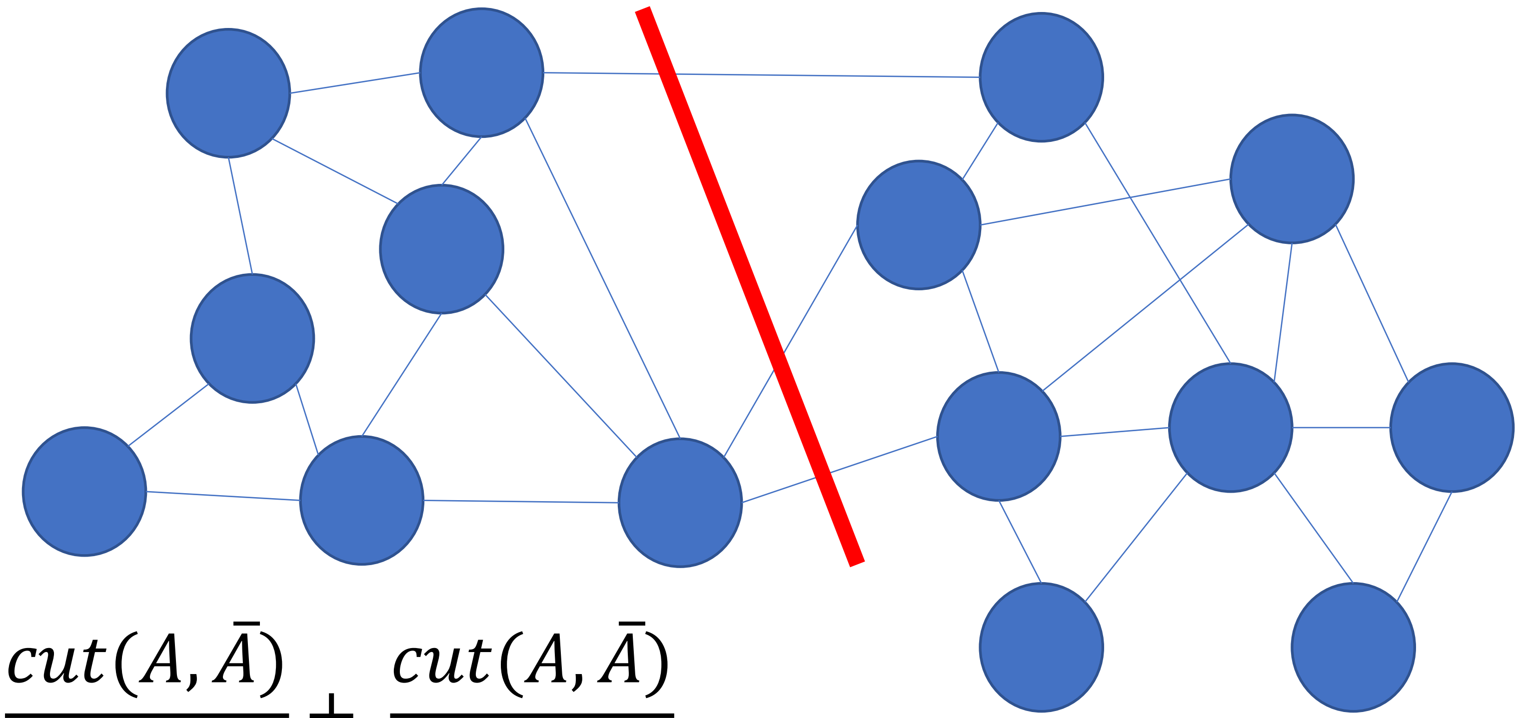


- Min-cut carves out small isolated parts of the graph
- In image segmentation: individual pixels

Normalized cuts

- "Cut" = total weight of cut edges
- Small cut means the groups don't "like" each other
- But need to normalize w.r.t how much they like *themselves*
- *"Volume"* of a subgraph = total weight of edges within the subgraph

Normalized cut



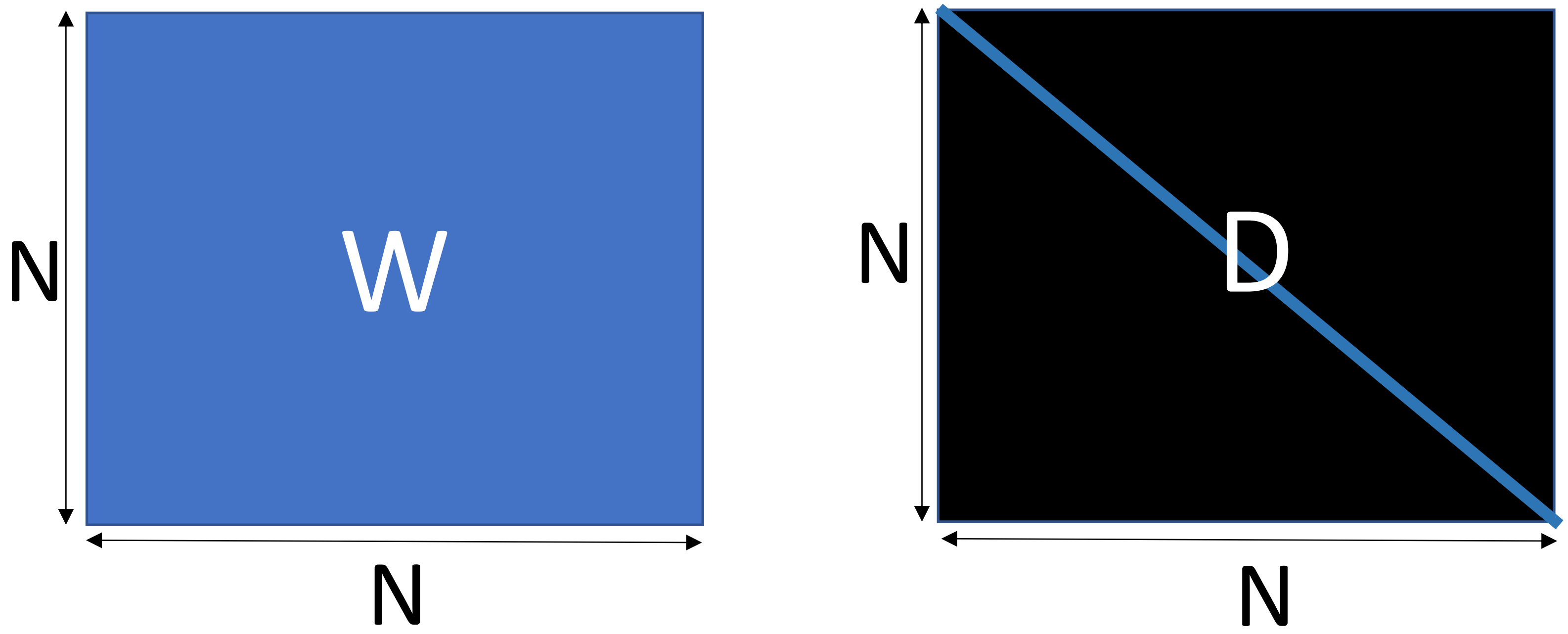
$$\frac{cut(A,\bar{A})}{vol(A)} + \frac{cut(A,\bar{A})}{vol(\bar{A})}$$

Min-cut vs normalized cut

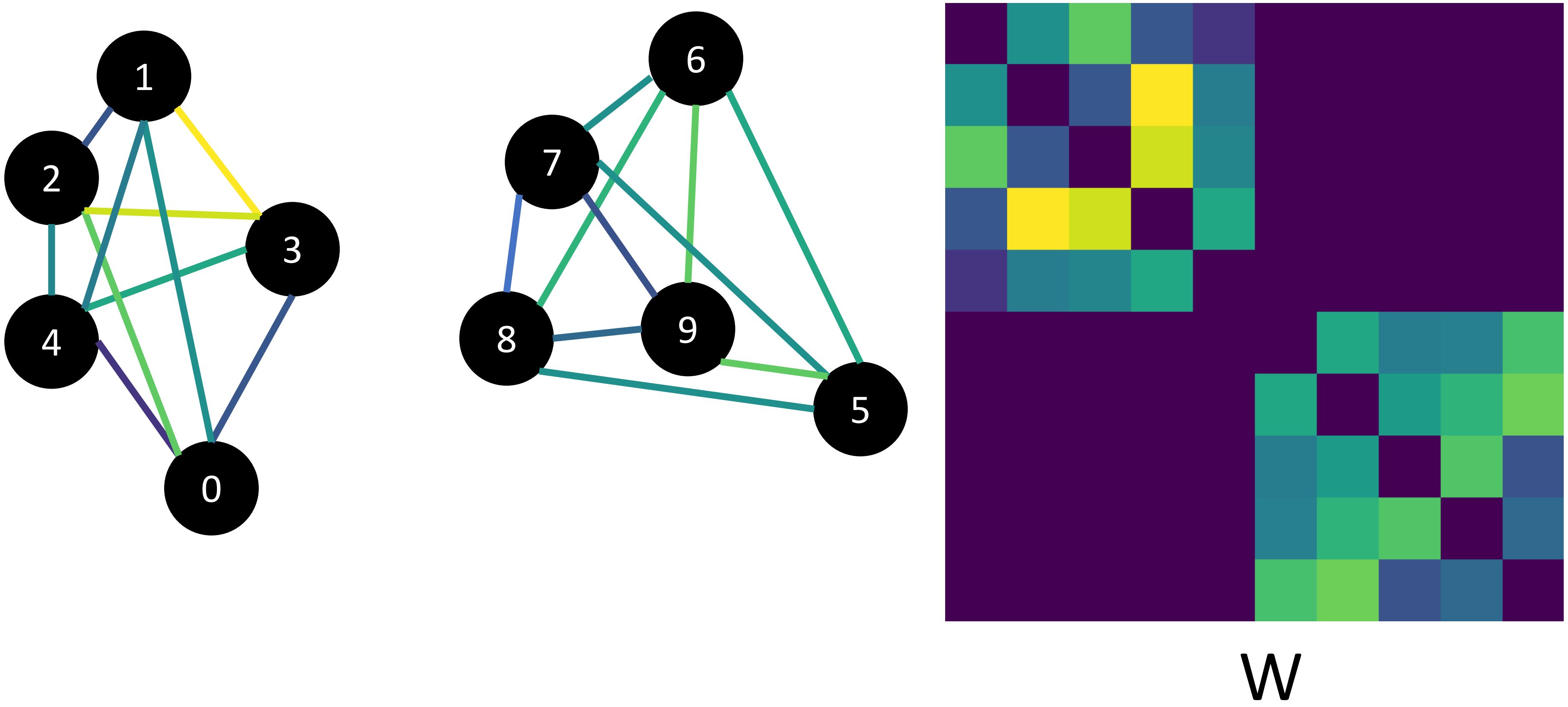
- Both rely on interpreting images as graphs
- By itself, min-cut gives small isolated pixels
  - But can work if we add other constraints
- min-cut can be solved in polynomial time
  - Dual of max-flow
- N-cut is NP-hard
  - But approximations exist!

Graphs and matrices

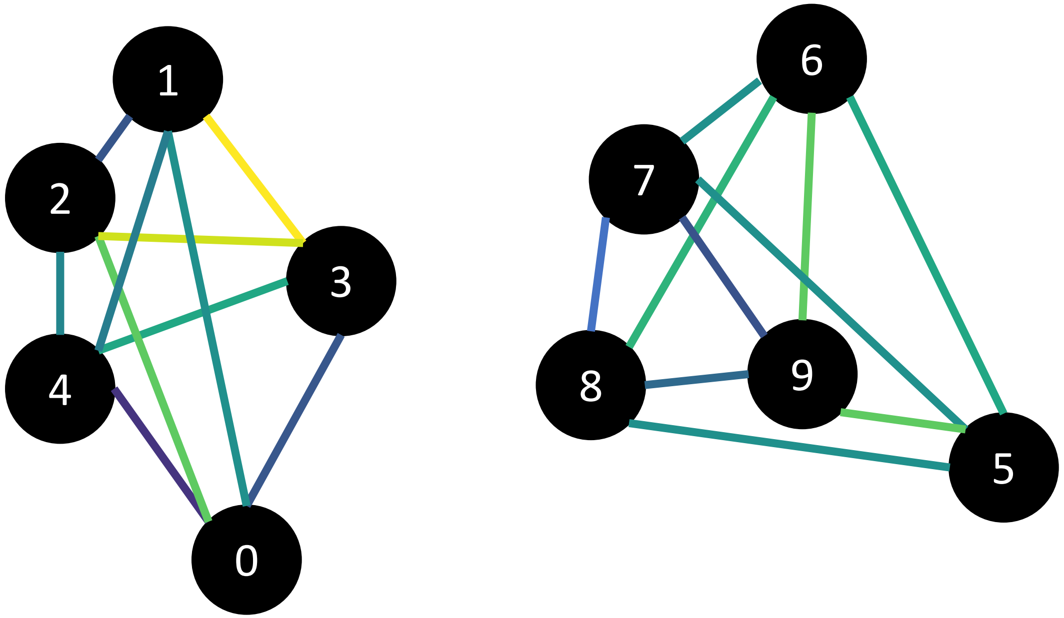
- w(i,j) = weight between i and j (*Affinity matrix)*
- d(i) = degree of i $=\sum_j w(i,j)$
- D = diagonal matrix with d(i) on diagonal



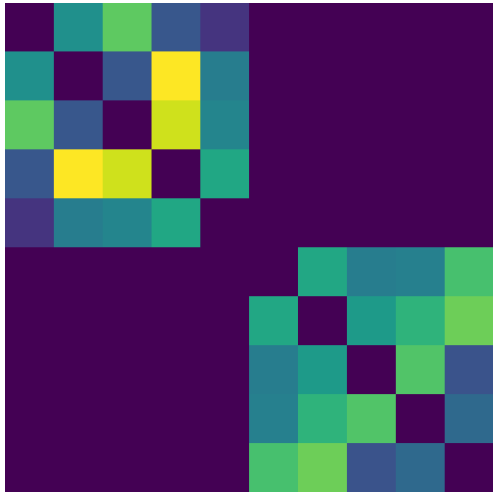
Graphs and matrices



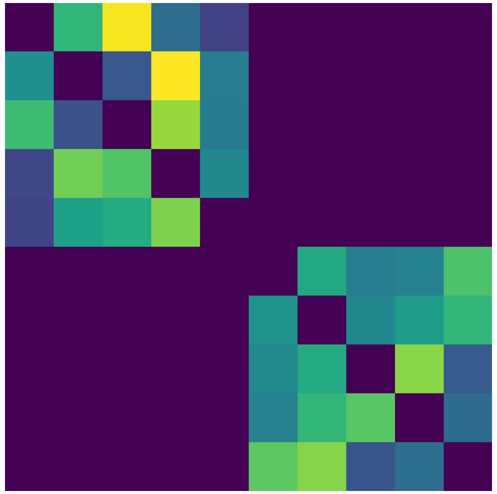
Graphs and matrices



$$E_{ij} = \frac{w_{ij}}{\sum_k w_{ik}}$$



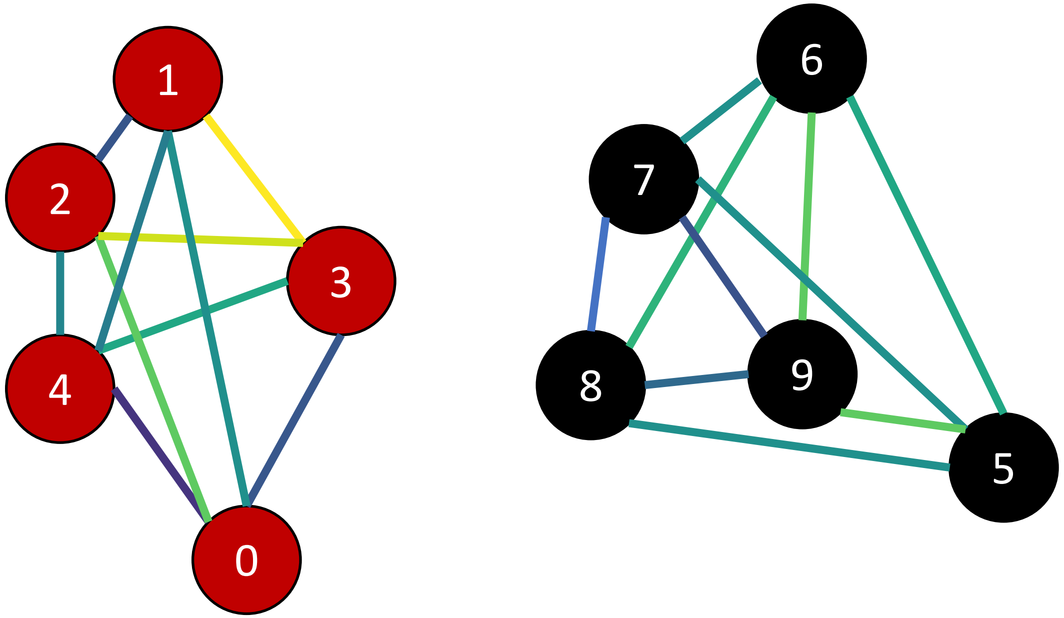
W



$E = D^{-1}W$

Graphs and matrices

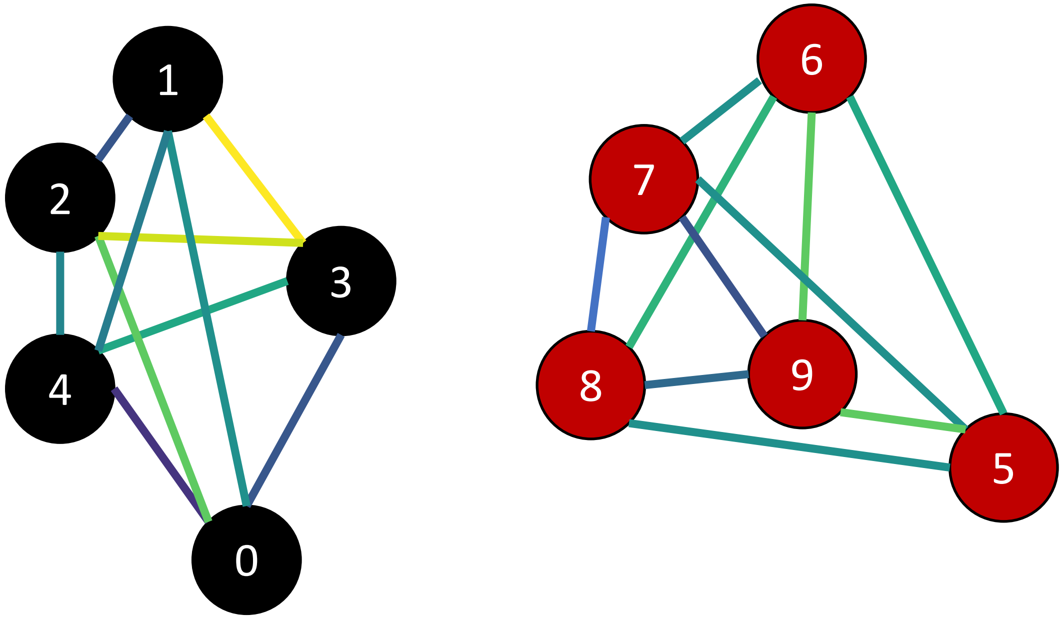
- How do we represent a clustering?
- A label for N nodes
  - 1 if part of cluster A, 0 otherwise
- An N-dimensional vector!



	v_1
0:	1
1:	1
2:	1
3:	1
4:	1
5:	0
6:	0
7:	0
8:	0
9:	0

Graphs and matrices

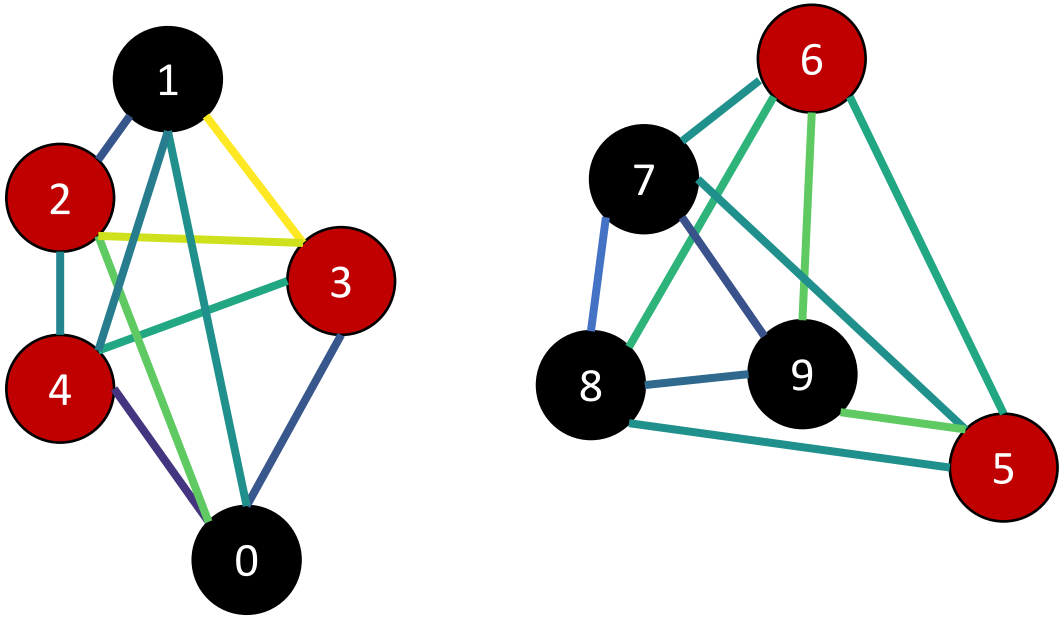
- How do we represent a clustering?
- A label for N nodes
  - 1 if part of cluster A, 0 otherwise
- An N-dimensional vector!



	v_1	v_2
0:	1	0
1:	1	0
2:	1	0
3:	1	0
4:	1	0
5:	0	1
6:	0	1
7:	0	1
8:	0	1
9:	0	1

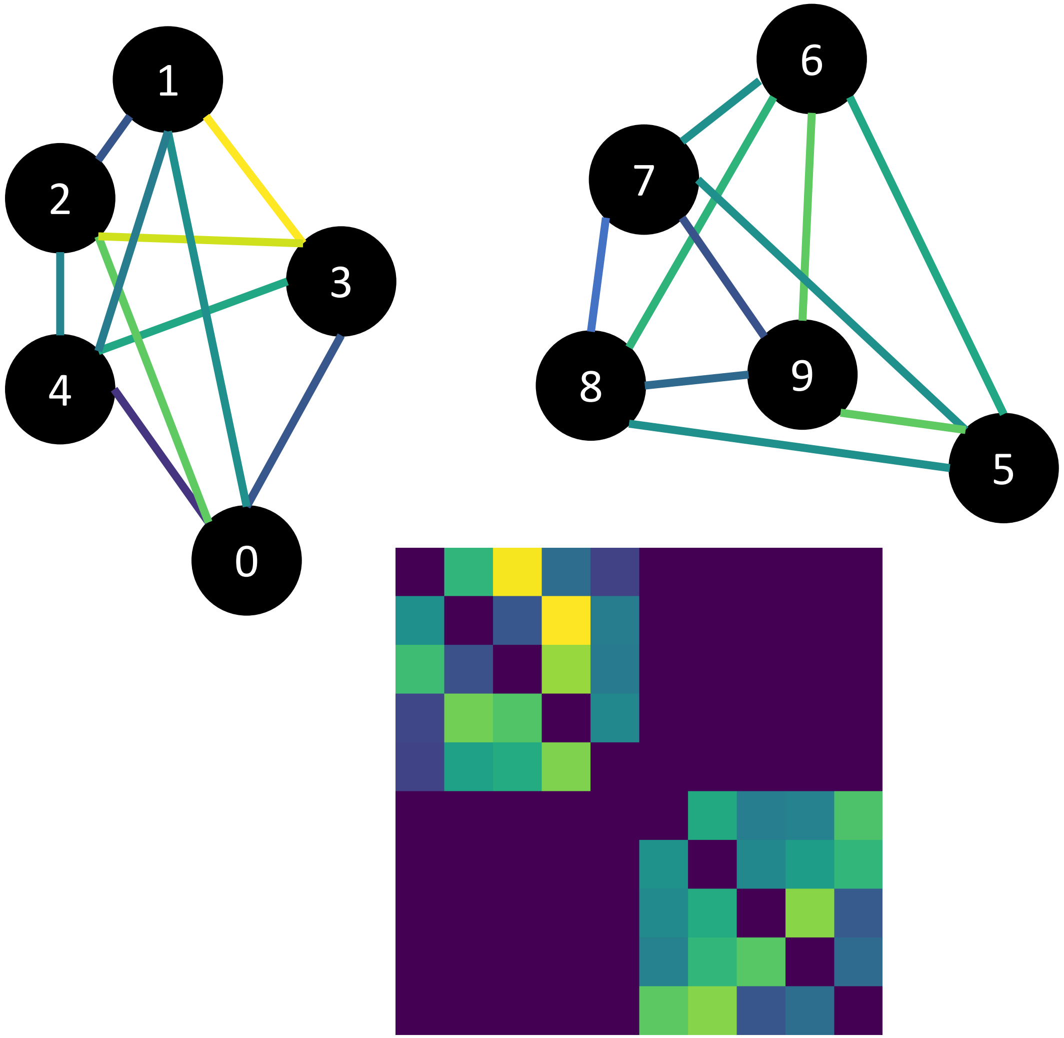
Graphs and matrices

- How do we represent a clustering?
- A label for N nodes
  - 1 if part of cluster A, 0 otherwise
- An N-dimensional vector!



	v_1	v_2	v_3
0:	1	0	0
1:	1	0	0
2:	1	1	1
3:	1	1	1
4:	1	1	1
5:	0	1	1
6:	0	1	1
7:	0	0	0
8:	0	0	0
9:	0	0	0

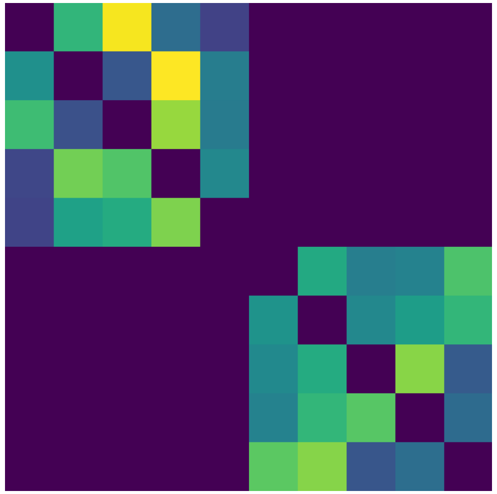
Graphs and matrices



$E = D^{-1}W$

	v_1
0:	1
1:	1
2:	1
3:	1
4:	1
5:	0
6:	0
7:	0
8:	0
9:	0

Graphs and matrices

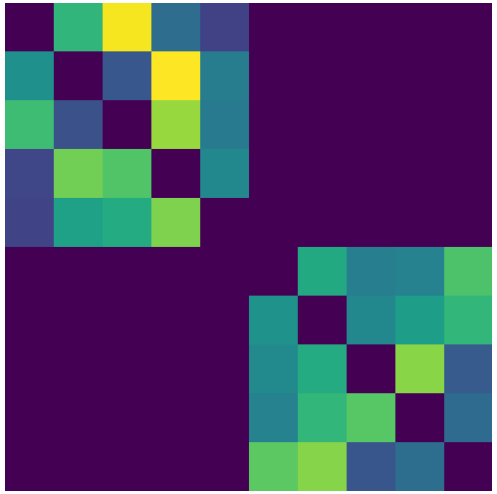


$E = D^{-1}W$

$$E_{ij} = \frac{w_{ij}}{\sum_k w_{ik}}$$

	v_1	$\mathbf{Ev_1}$
0:	1	1
1:	1	1
2:	1	1
3:	1	1
4:	1	1
5:	0	0
6:	0	0
7:	0	0
8:	0	0
9:	0	0

Graphs and matrices

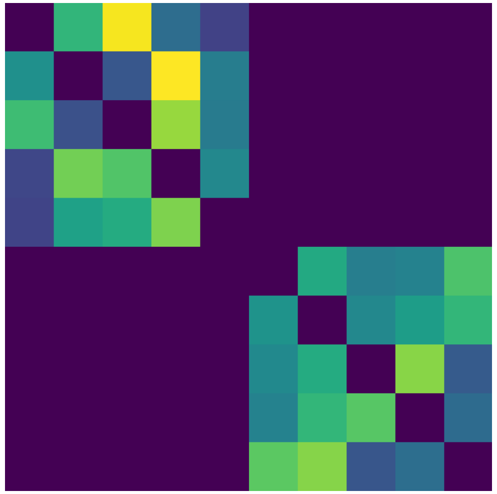


$E = D^{-1}W$

$$E_{ij} = \frac{w_{ij}}{\sum_k w_{ik}}$$

	v_2	Ev_2
0:	0	0
1:	0	0
2:	0	0
3:	0	0
4:	0	0
5:	1	1
6:	1	1
7:	1	1
8:	1	1
9:	1	1

Graphs and matrices

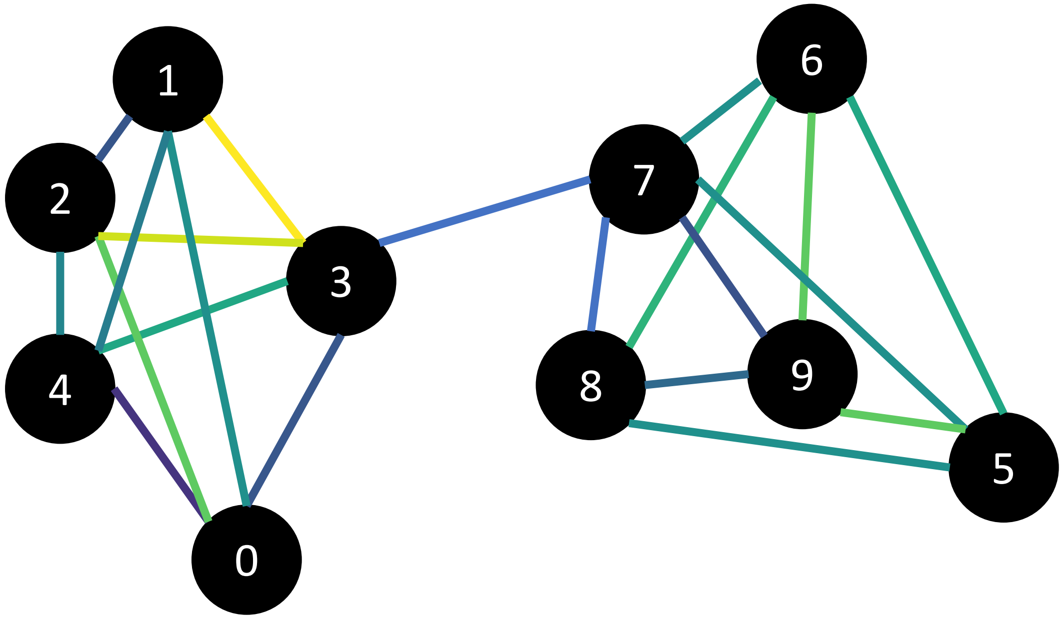


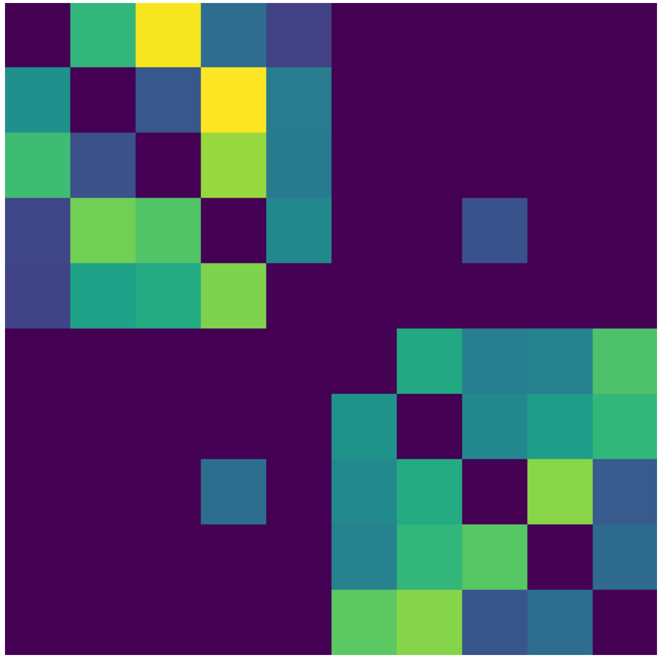
E = $D^{-1}W$

$$E_{ij} = \frac{w_{ij}}{\sum_k w_{ik}}$$

	v_3	Ev_3
0:	0	0.7
1:	0	0.8
2:	1	0.6
3:	1	0.5
4:	1	0.6
5:	1	0.3
6:	1	0.2
7:	0	0.5
8:	0	0.5
9:	0	0.7

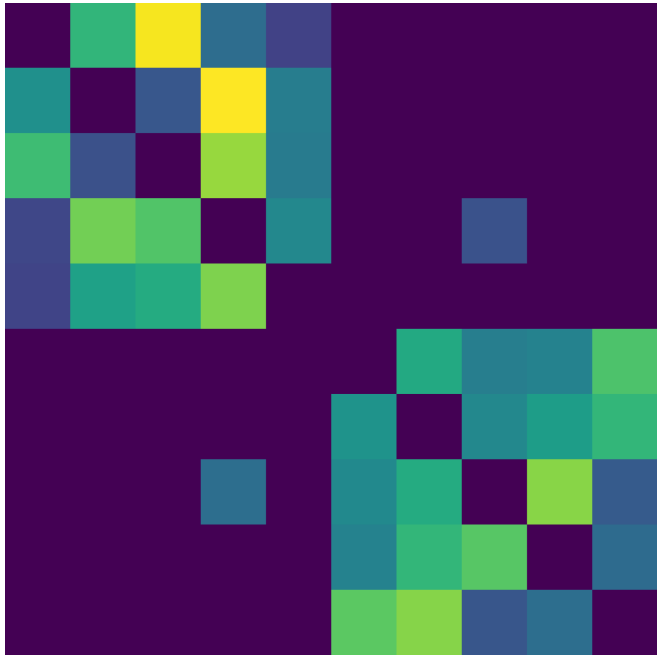
Graphs and matrices





$E = D^{-1}W$

Graphs and matrices



E = $D^{-1}W$

$$E_{ij} = \frac{w_{ij}}{\sum_k w_{ik}}$$

	v_1	**Ev_1**
0:	1	1
1:	1	1
2:	1	1
3:	1	1
4:	1	1
5:	0	0
6:	0	0
7:	0	0.2
8:	0	0
9:	0	0

Graphs and matrices

$$D^{-1}Wy \approx y$$

Define z so that $y = D^{-\frac{1}{2}}z$

$$D^{-1}WD^{-\frac{1}{2}}z \approx D^{-\frac{1}{2}}z$$

$$\Rightarrow D^{-\frac{1}{2}}WD^{-\frac{1}{2}}z \approx z$$

$$\Rightarrow (I - D^{-\frac{1}{2}}WD^{-\frac{1}{2}})z \approx 0$$

Graphs and matrices

$$\Rightarrow (I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}) z \approx 0$$

$$\Rightarrow \mathcal{L} z \approx 0$$

$$\mathcal{L} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

is called the
Normalized Graph
Laplacian

Graphs and matrices

$$\mathcal{L} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

- We want $\mathcal{L}z \approx 0$
- Trivial solution: all nodes of graph in one cluster, nothing in the other
- To avoid trivial solution, look for the *eigenvector with the second smallest eigenvalue*

$$\mathcal{L}z = \lambda z$$

$$\lambda_1 < \lambda_2 < \ldots < \lambda_N$$

- Find z s.t. $\mathcal{L}z = \lambda_2 z$

Normalized cuts

- Approximate solution to normalized cuts
- Construct matrix W and D
- Construct normalized graph laplacian

$$\mathcal{L} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

- Look for the second smallest eigenvector

$$\mathcal{L}z = \lambda_2 z$$

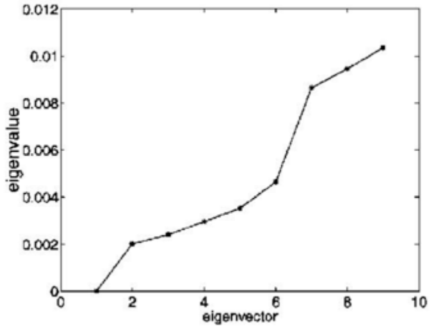
- Compute $y = D^{-\frac{1}{2}} z$
- *Threshold y to get clusters*
    - Ideally, sweep threshold to get lowest N-cut value

Eigenvectors of images

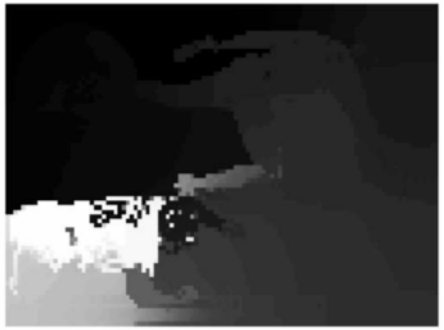
- The eigenvector has as many components as pixels in the image

Eigenvectors of images

- The eigenvector has as many components as pixels in the image



(a)



(b)

(c)

(d)

(e)

(f)

Another example

2^{nd} eigenvector

3^{rd} eigenvector

4^{th} eigenvector

Eigenvectors of images