What you'll be able to do!

What you'll be able to do!

machine translation "hello!" → "bonjour!"

document search "Can I get a refund?" →

- "What's your return policy?"
- ...
- "May I get my money back?"

Learning Objectives

- Transform vector
- “K nearest neighbors”
- Hash tables
- Divide vector space into regions
- Locality sensitive hashing
- Approximated nearest neighbors

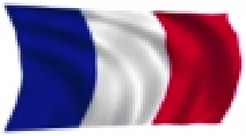
Learning Objectives

- Transform vector
- "K nearest neighbors"
- Hash tables
- Divide vector space into regions
- Locality sensitive hashing
- Approximated nearest neighbors

Machine translation

Document search

Overview of Translation

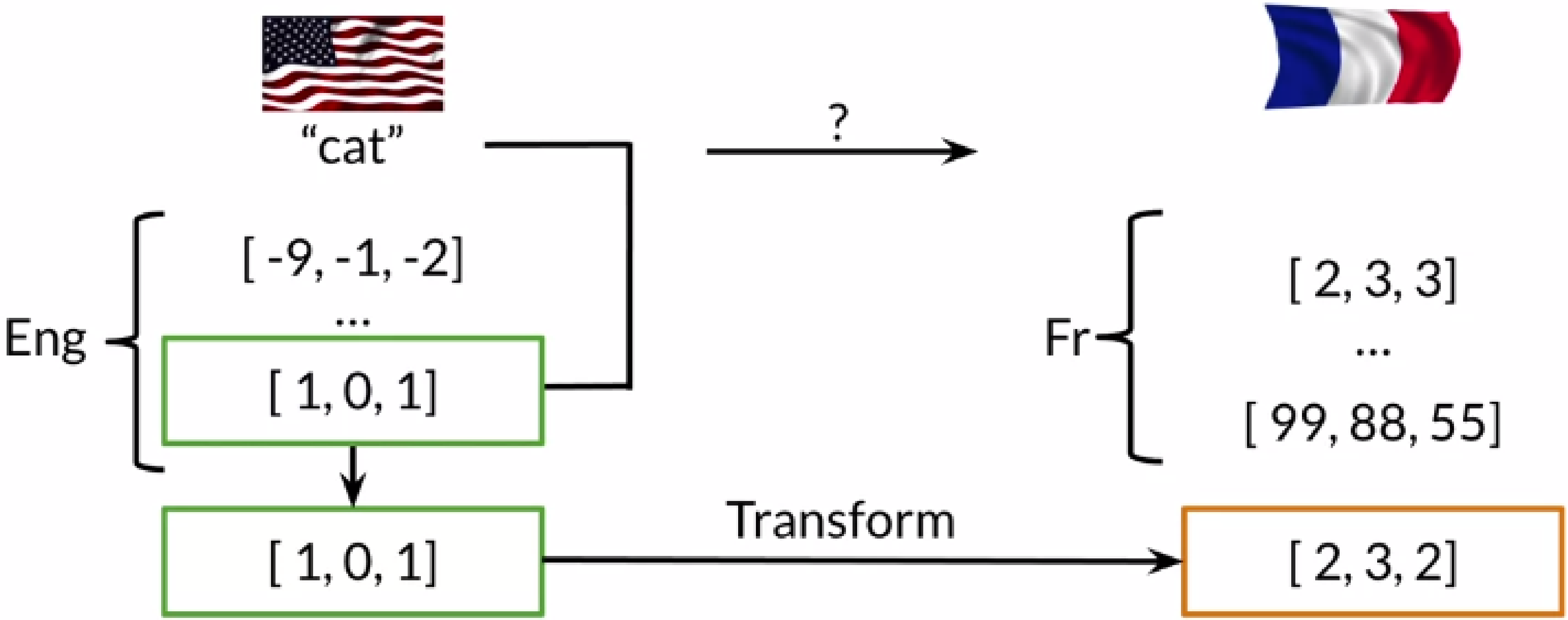


$$\xrightarrow{?}$$

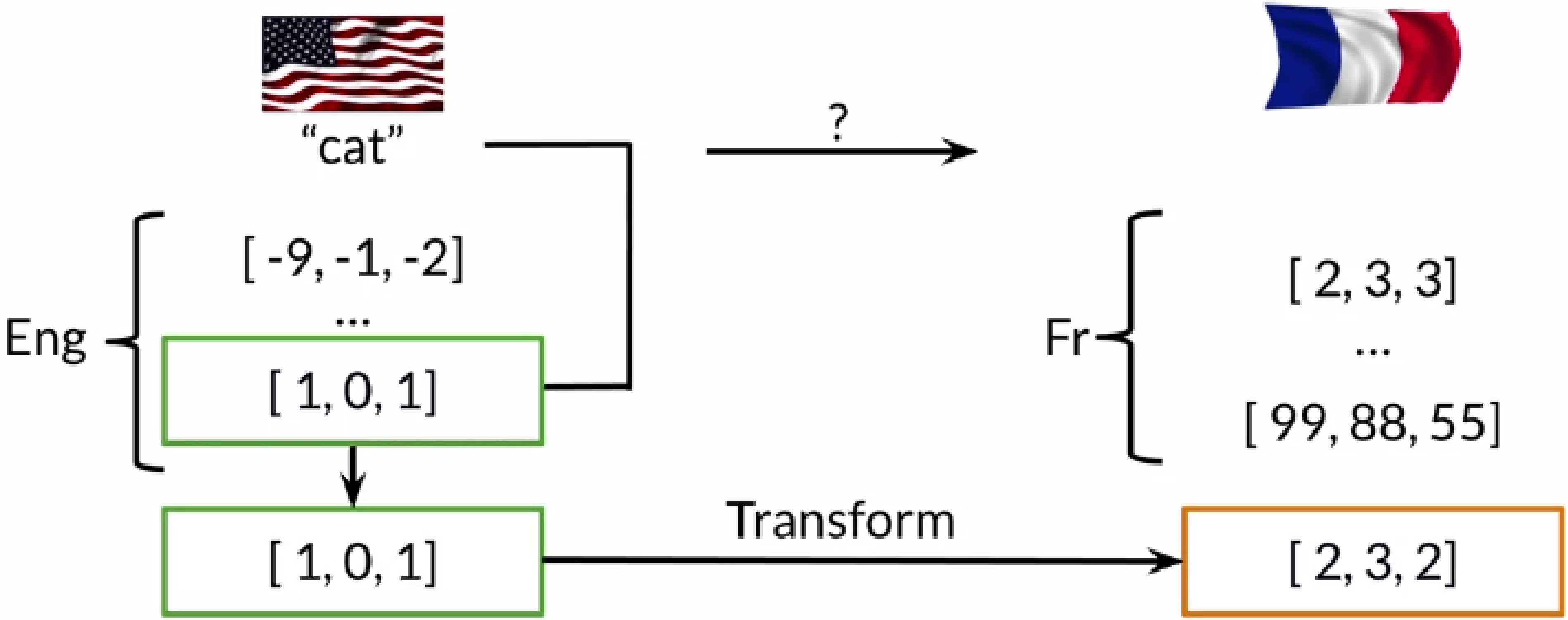
Eng
[-9, -1, -2]
...
[1, 0, 1]

Fr
[2, 3, 3]
...
[99, 88, 55]

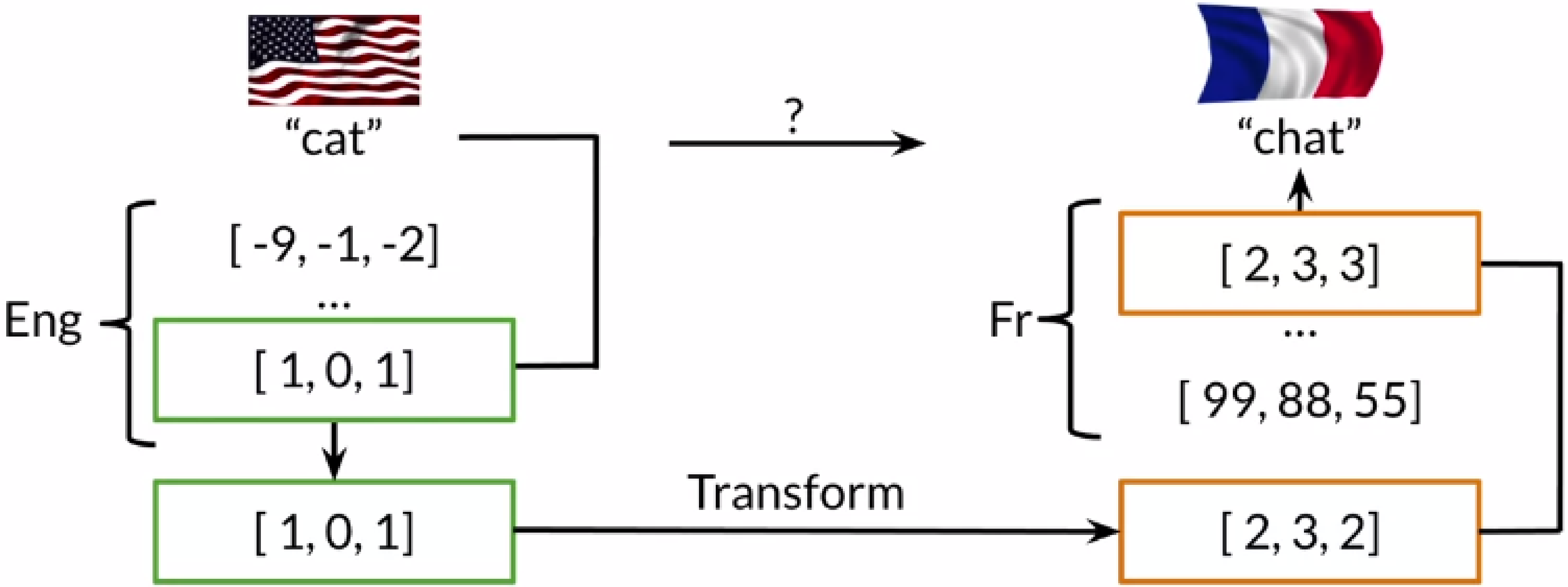
Overview of Translation



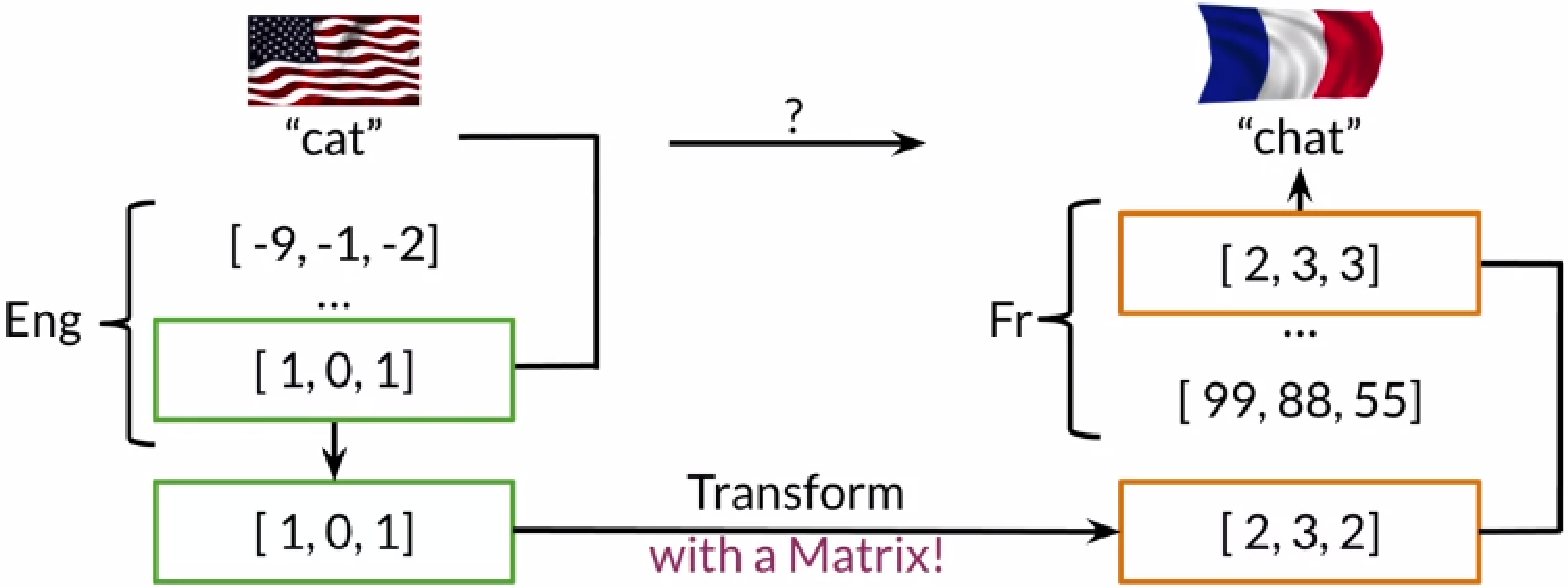
Overview of Translation



Overview of Translation



Overview of Translation



Transforming vectors

```
R = np.array([[2,0],
              [0,-2]])

x = np.array([[1,1]])

np.dot(x,R)
```

```
array([[2,-2]])
```

Transforming vectors

```
R = np.array([[2,0],
              [0,-2]])

x = np.array([[1,1]])

np.dot(x,R)
```

```
array([[2,-2]])
```

Try it yourself!

Align word vectors

$$\mathbf{XR} \approx \mathbf{Y}$$

$$\underset{\mathbf{X}}{\begin{pmatrix} \quad & \quad \\ \quad & \quad \end{pmatrix}} \qquad \underset{\mathbf{Y}}{\begin{pmatrix} \quad & \quad \\ \quad & \quad \end{pmatrix}}$$

Align word vectors

$$\mathbf{XR} \approx \mathbf{Y}$$

$$\underset{\mathbf{X}}{\begin{pmatrix} [\text{ “cat” vector }] \\ [\text{ ... vector }] \\ \\ \end{pmatrix}} \qquad \underset{\mathbf{Y}}{\begin{pmatrix} [\text{ “chat” vecteur }] \\ [\text{ ... vecteur }] \\ \\ \end{pmatrix}}$$

Align word vectors

$$\mathbf{XR} \approx \mathbf{Y}$$

$$\underset{\mathbf{X}}{\begin{pmatrix} [\text{ “cat” vector }] \\ [\text{ ... vector }] \\ [\text{ “zebra” vector }] \end{pmatrix}} \qquad \underset{\mathbf{Y}}{\begin{pmatrix} [\text{ “chat” vecteur }] \\ [\text{ ... vecteur }] \\ [\text{ “zébresse” vecteur }] \end{pmatrix}}$$

Align word vectors

$$\mathbf{XR} \approx \mathbf{Y}$$

$$\underset{\mathbf{X}}{\begin{pmatrix} [\text{ “cat” vector }] \\ [\text{ ... vector }] \\ [\text{ “zebra” vector }] \end{pmatrix}} \qquad \underset{\mathbf{Y}}{\begin{pmatrix} [\text{ “chat” vecteur }] \\ [\text{ ... vecteur }] \\ [\text{ “zébresse” vecteur }] \end{pmatrix}}$$

subsets of the full vocabulary

Solving for R

$$Loss = \| \mathbf{XR} - \mathbf{Y} \|_F$$

Solving for R

initialize R

$$Loss = \| \mathbf{XR} - \mathbf{Y} \|_F$$

Solving for R

initialize R

in a loop:

$$Loss = \| \mathbf{XR} - \mathbf{Y} \|_F$$

Solving for R

initialize R

in a loop:

$$Loss = \| \mathbf{XR} - \mathbf{Y} \|_F$$

$$g = \frac{d}{dR} Loss$$

gradient

Solving for R

initialize R

in a loop:

$$Loss = \| \mathbf{XR} - \mathbf{Y} \|_F$$

$$g = \frac{d}{dR} Loss$$ gradient

$$R = R - \alpha g$$ update

Frobenius norm

$$\parallel \mathbf{XR} - \mathbf{Y} \parallel_F$$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

$$\|\mathbf{A}_F\| = \sqrt{2^2 + 2^2 + 2^2 + 2^2}$$

Frobenius norm

$$\| \mathbf{XR} - \mathbf{Y} \|_F$$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

$$\|\mathbf{A}_F\| = \sqrt{2^2 + 2^2 + 2^2 + 2^2}$$

$$\|\mathbf{A}_F\| = 4$$

$$\|\mathbf{A}\|_F \equiv \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

Frobenius norm

```
A = np.array([[2,2],
              [2,2]])

A_squared = np.square(A)
A_squared
```

```
array([[4,4],
       [4,4]])
```

Try it yourself!

```
A_Frobenious = np.sqrt(np.sum(A_squared))
A_Frobenious
```

```
4.0
```

Frobenius norm squared

$$\|\mathbf{XR} - \mathbf{Y}\|_F^2$$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

$$\|\mathbf{A}\|_F^2 = \left(\sqrt{2^2 + 2^2 + 2^2 + 2^2}\right)^2$$

Frobenius norm squared

$$\|\mathbf{XR} - \mathbf{Y}\|_F^2$$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

$$\|\mathbf{A}\|_F^2 = \left(\sqrt{2^2 + 2^2 + 2^2 + 2^2}\right)^2$$

$$\|\mathbf{A}\|_F^2 = 16$$

Gradient

$$Loss = \|\mathbf{XR} - \mathbf{Y}\|_F^2$$

$$g = \frac{d}{dR} Loss$$

Gradient

$$Loss = \|\mathbf{XR} - \mathbf{Y}\|_F^2$$

$$g = \frac{d}{dR}Loss = \frac{2}{m}\left(\mathbf{X}^T(\mathbf{XR} - \mathbf{Y})\right)$$

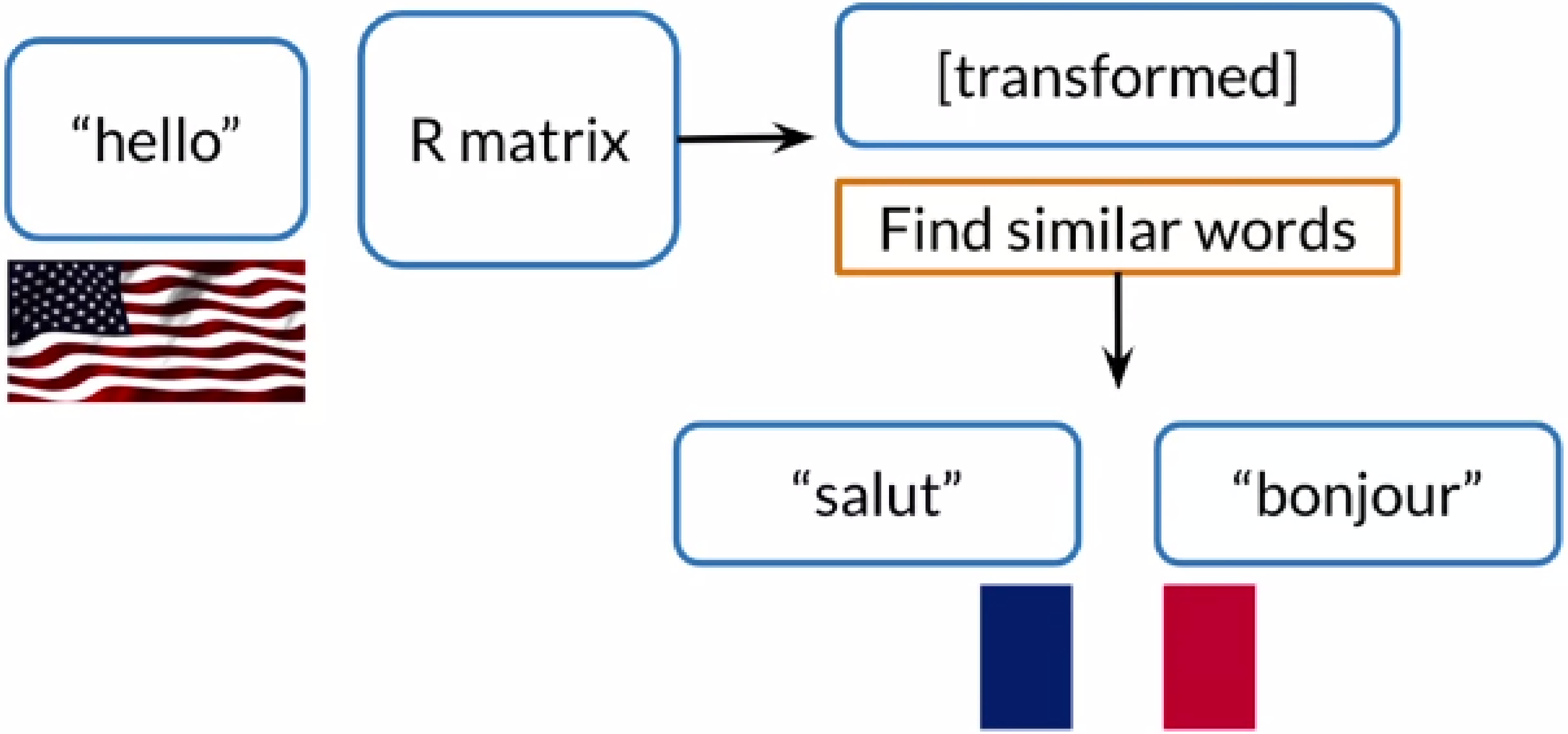
Gradient

$$Loss = \|\mathbf{XR} - \mathbf{Y}\|_F^2$$

$$g = \frac{d}{dR}Loss = \frac{2}{m}\left(\mathbf{X}^T(\mathbf{XR} - \mathbf{Y})\right)$$

Implement in the assignment!

Finding the translation



Nearest neighbours

You

San Francisco

Friend

Nearest neighbours

You

San Francisco

Friend	Location
	Shanghai
	Bangalore
	Los Angeles

Nearest neighbours

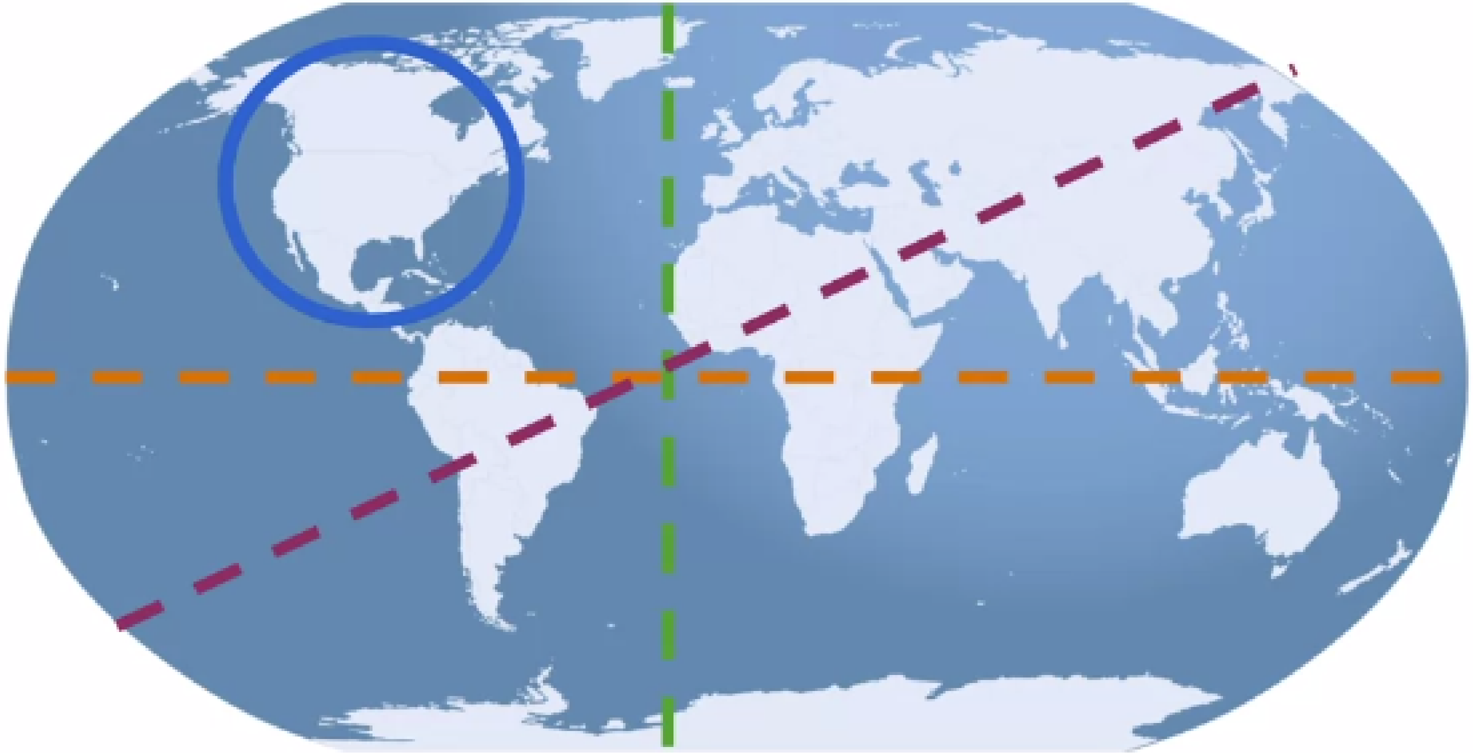
You

San Francisco

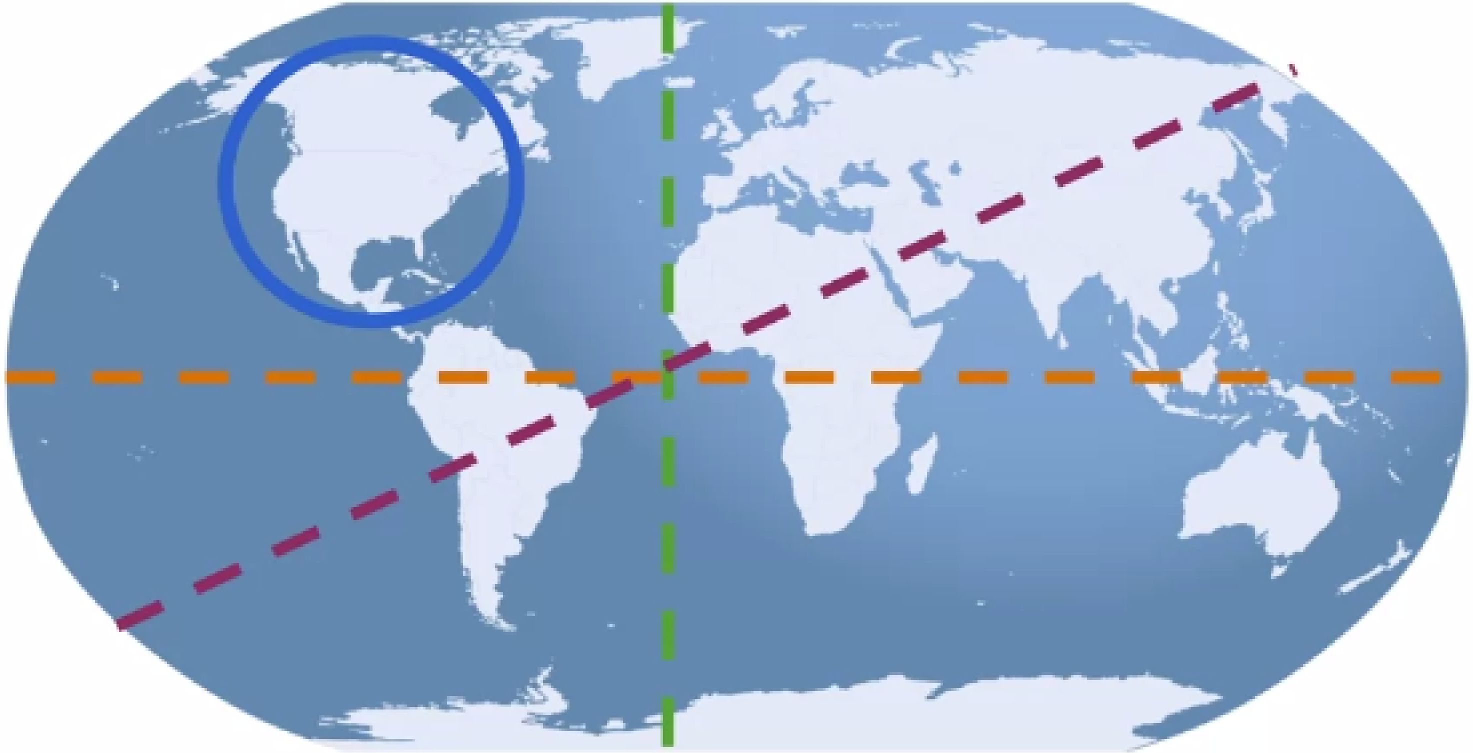
Friend	Location	Nearest
	Shanghai	2
	Bangalore	3
	Los Angeles	1

Nearest neighbors

Nearest neighbors



Nearest neighbors

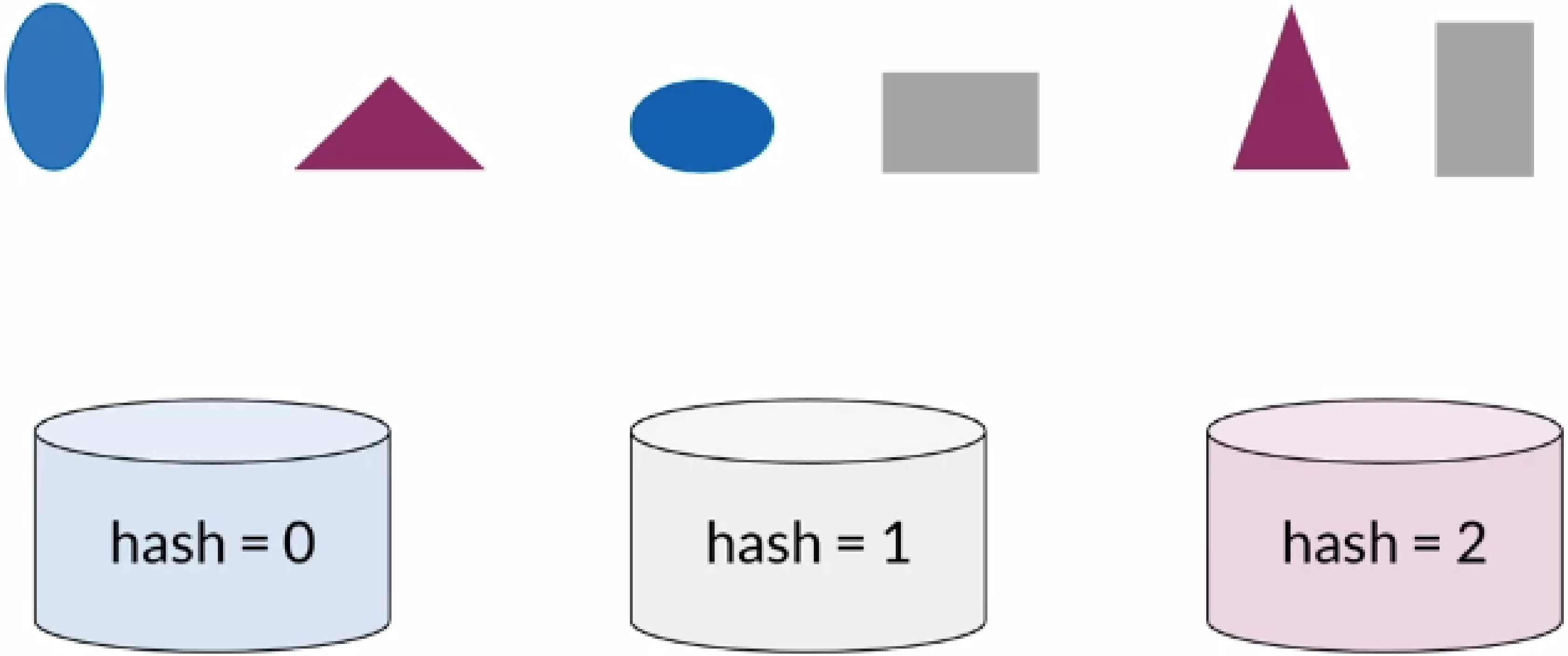


Hash
tables!

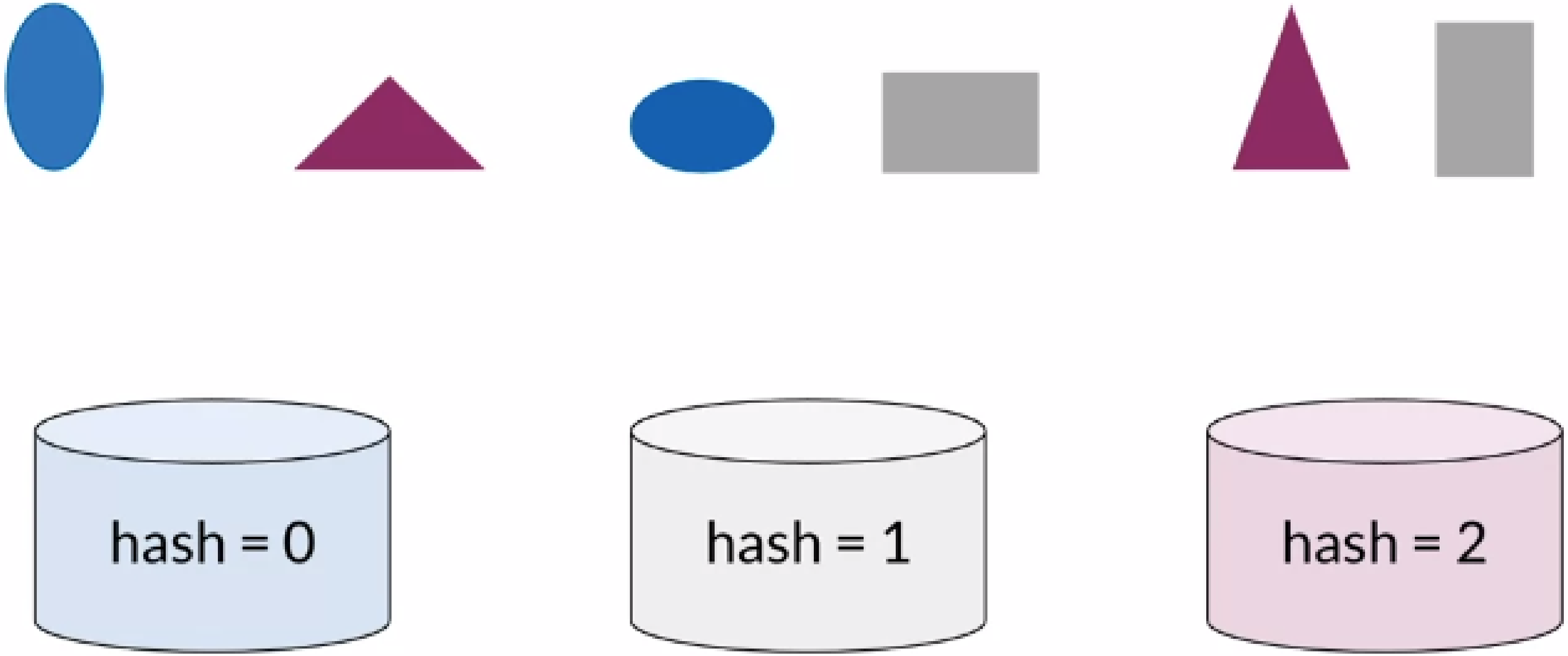
Summary

- K-nearest neighbors, for closest matches
- Hash tables

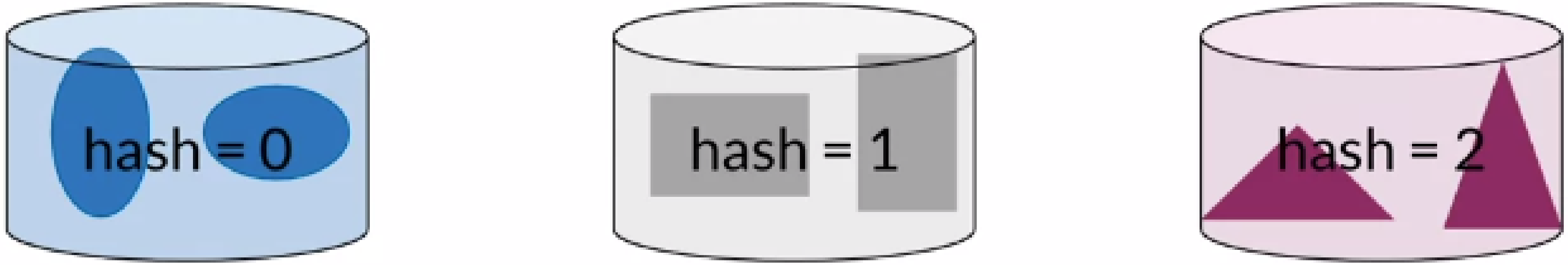
Hash tables



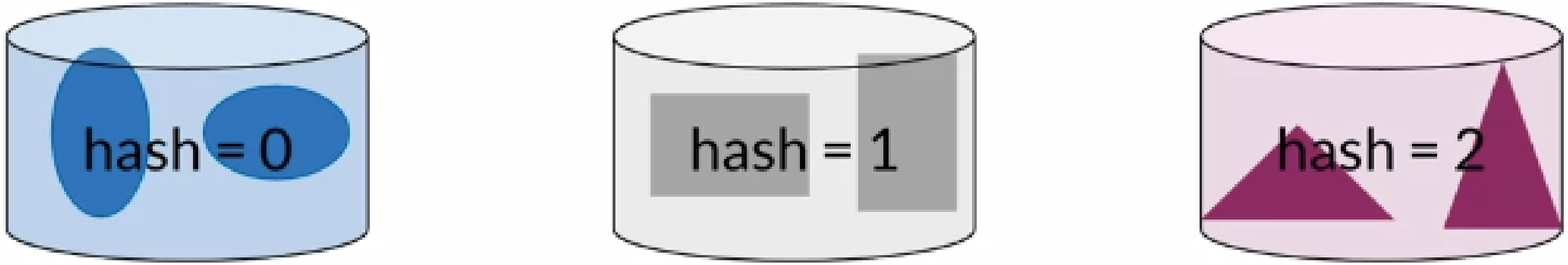
Hash tables



Hash tables



Hash tables

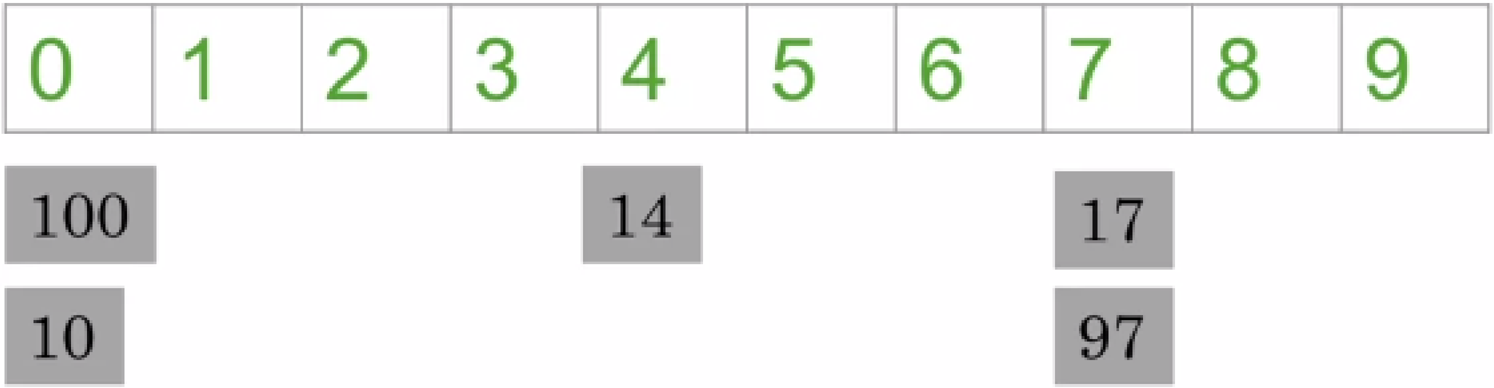


Hash function

100	14	17
10		97

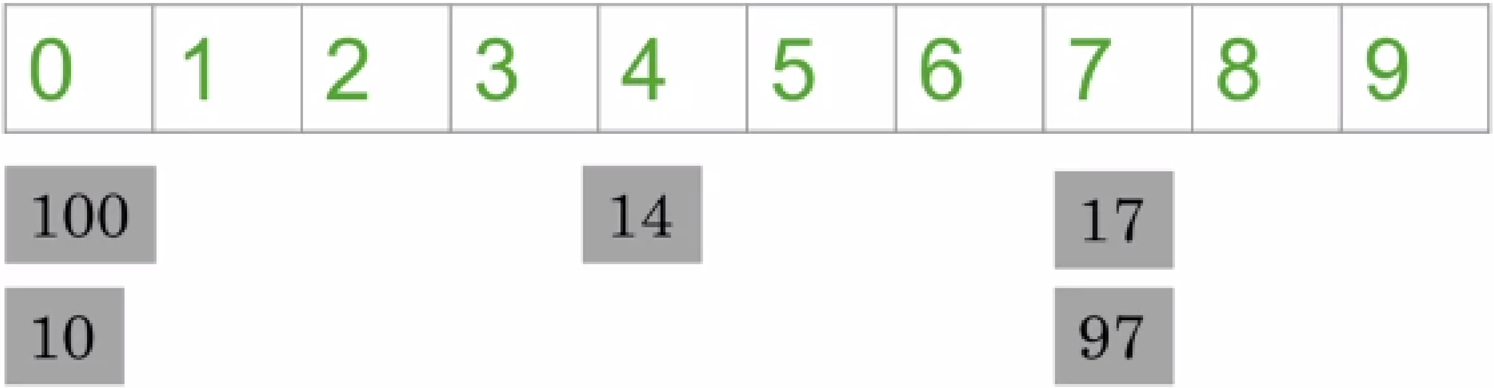
Hash function (vector) ⟶ Hash value

Hash function



Hash function (vector) ⟶ Hash value

Hash function



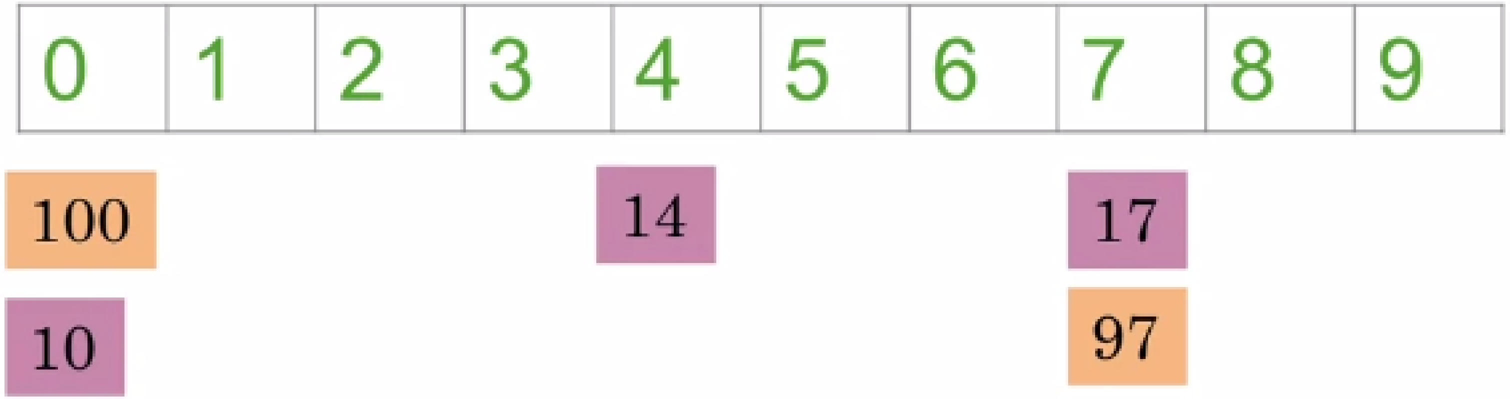
Hash function (vector) ⟶ Hash value

Hash value = vector % number of buckets

Create a basic hash table

```
def basic_hash_table(value_l,n_buckets):

    def hash_function(value_l,n_buckets):
        return int(value) % n_buckets

    hash_table = {i:[] for i in range(n_buckets)}
    for value in value_l:
        hash_value = hash_function(value,n_buckets)
        hash_table[hash_value].append(value)

    return hash_table
```

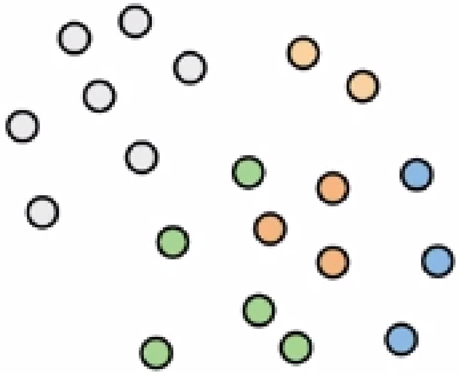
Hash function



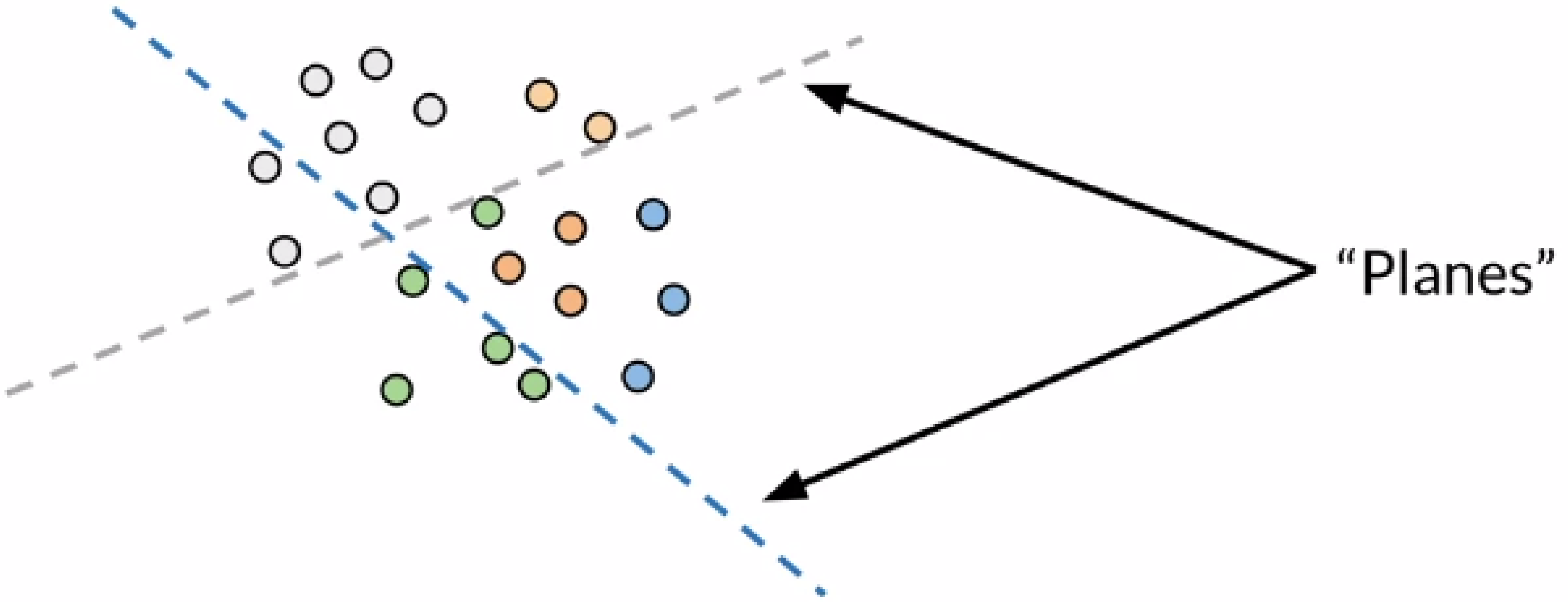
Hash function by location?

Locality sensitive hashing, next!

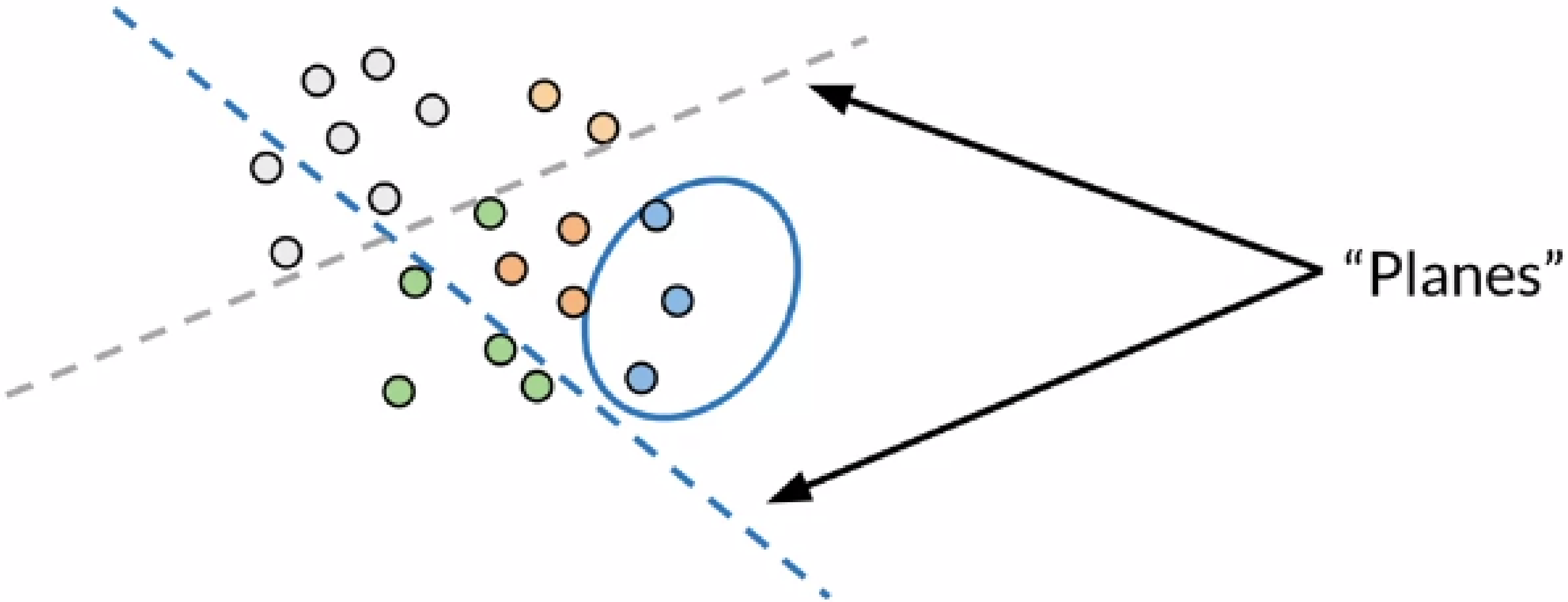
Locality Sensitive Hashing



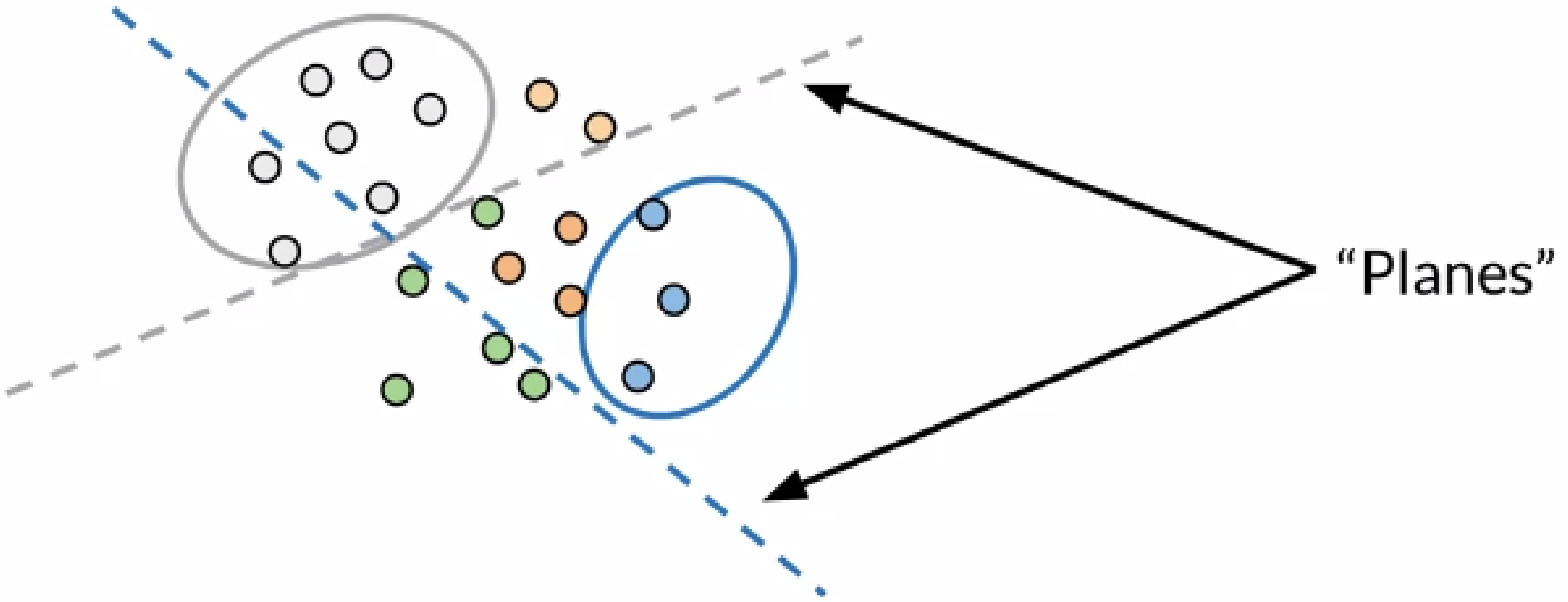
Locality Sensitive Hashing



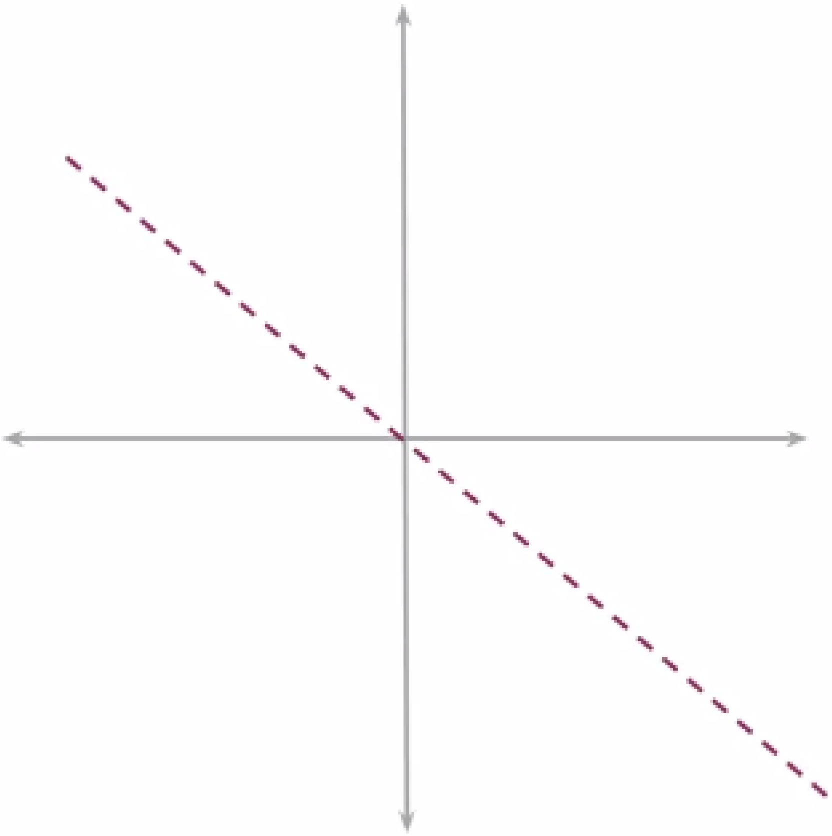
Locality Sensitive Hashing



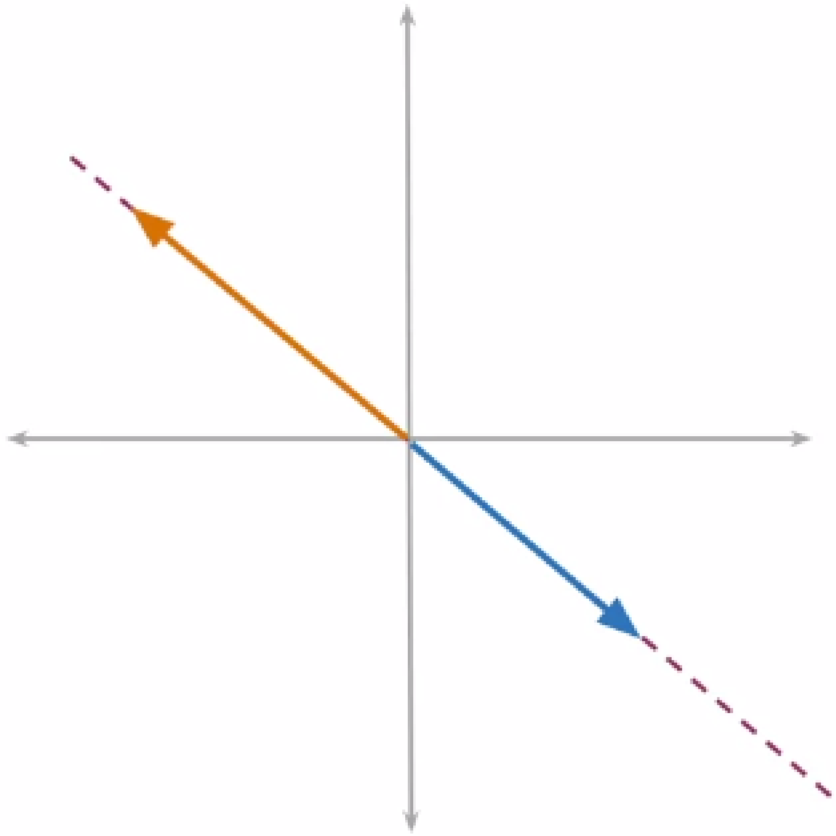
Locality Sensitive Hashing



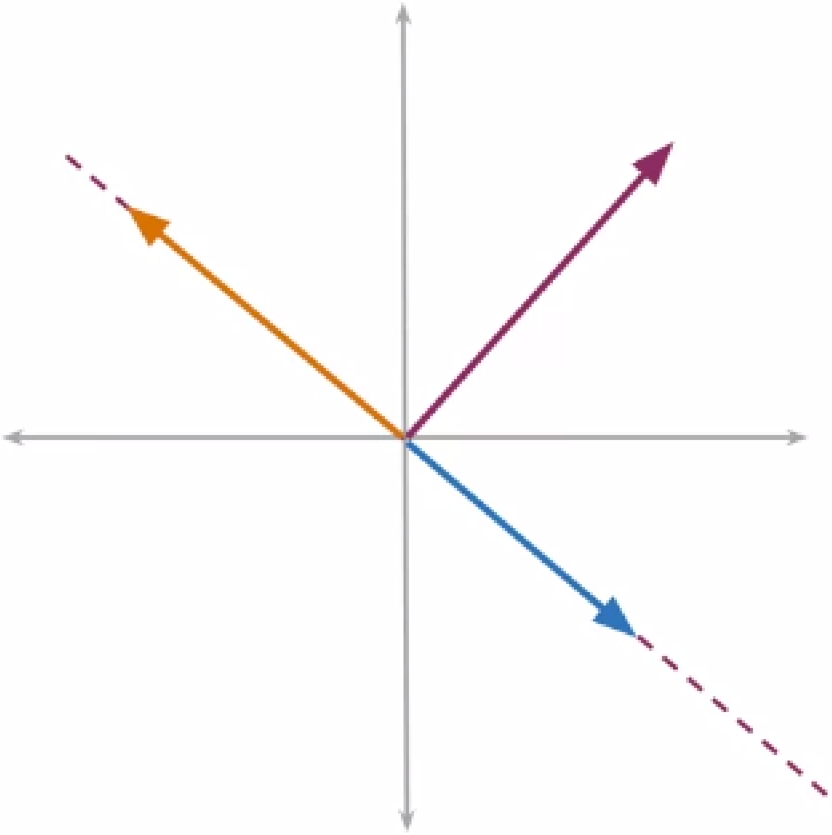
Planes



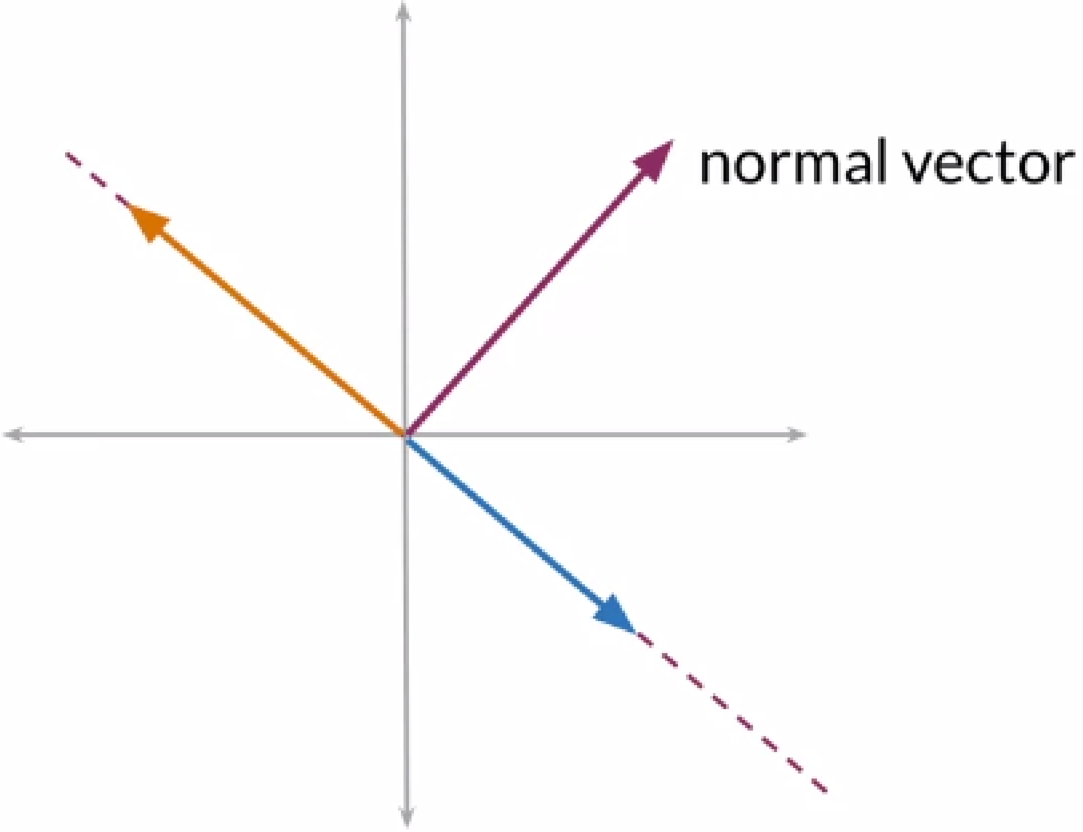
Planes



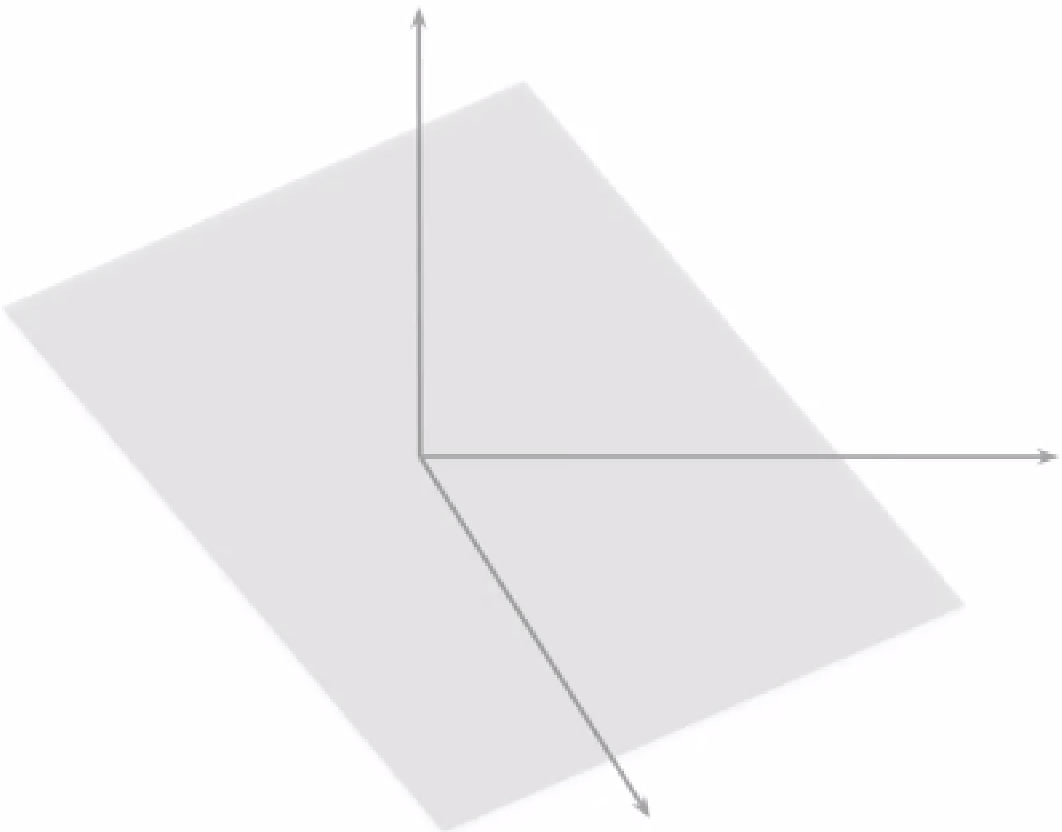
Planes



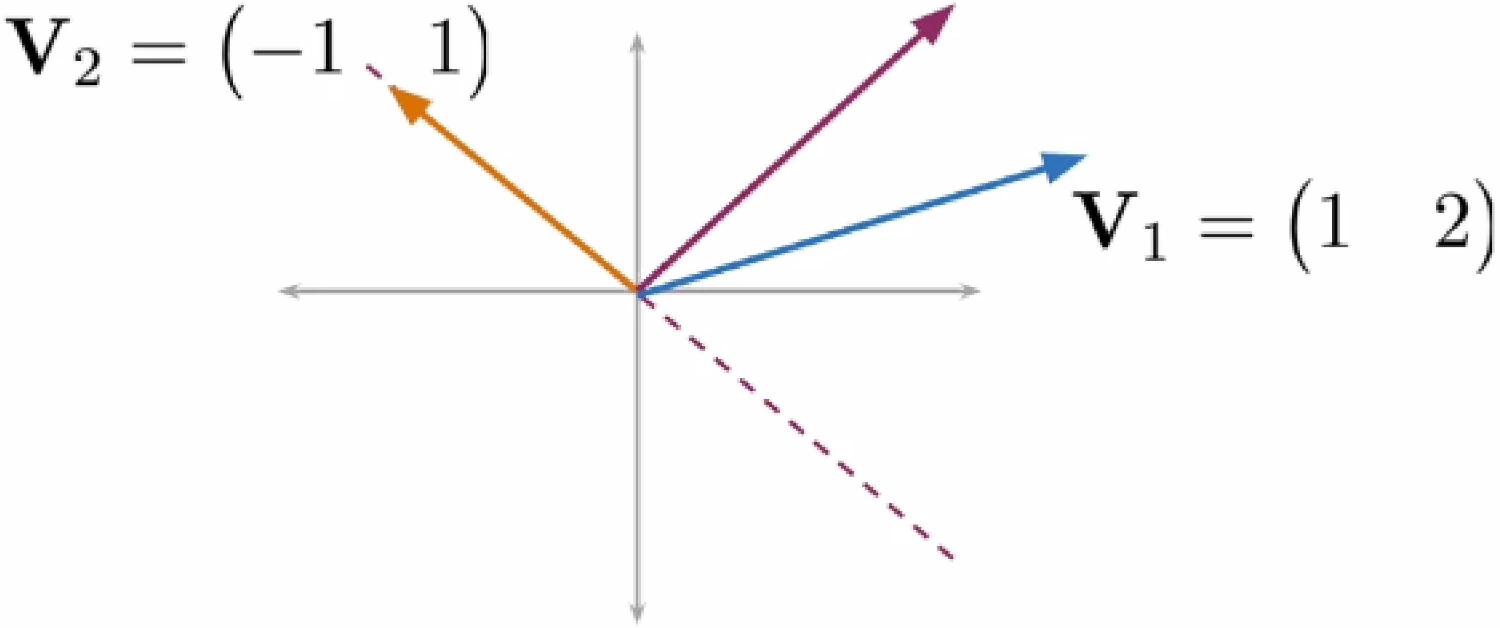
Planes



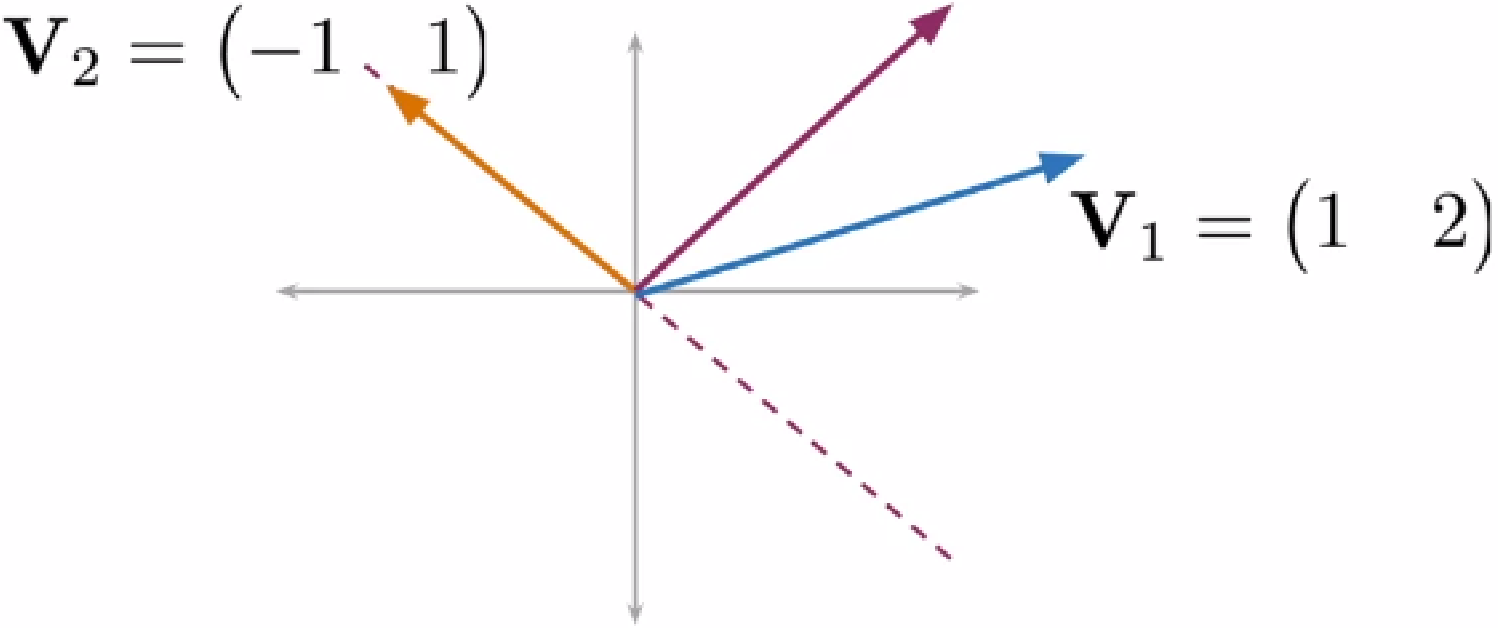
Planes



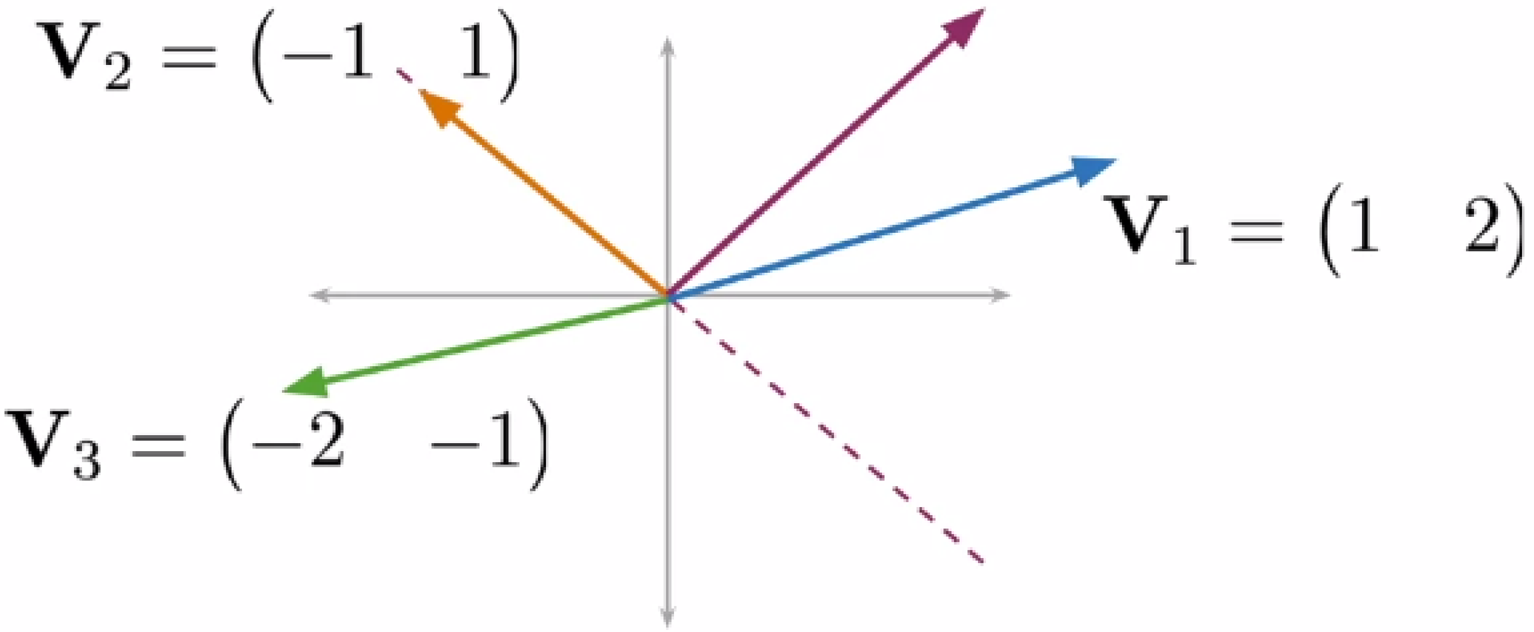
Which side of the plane?



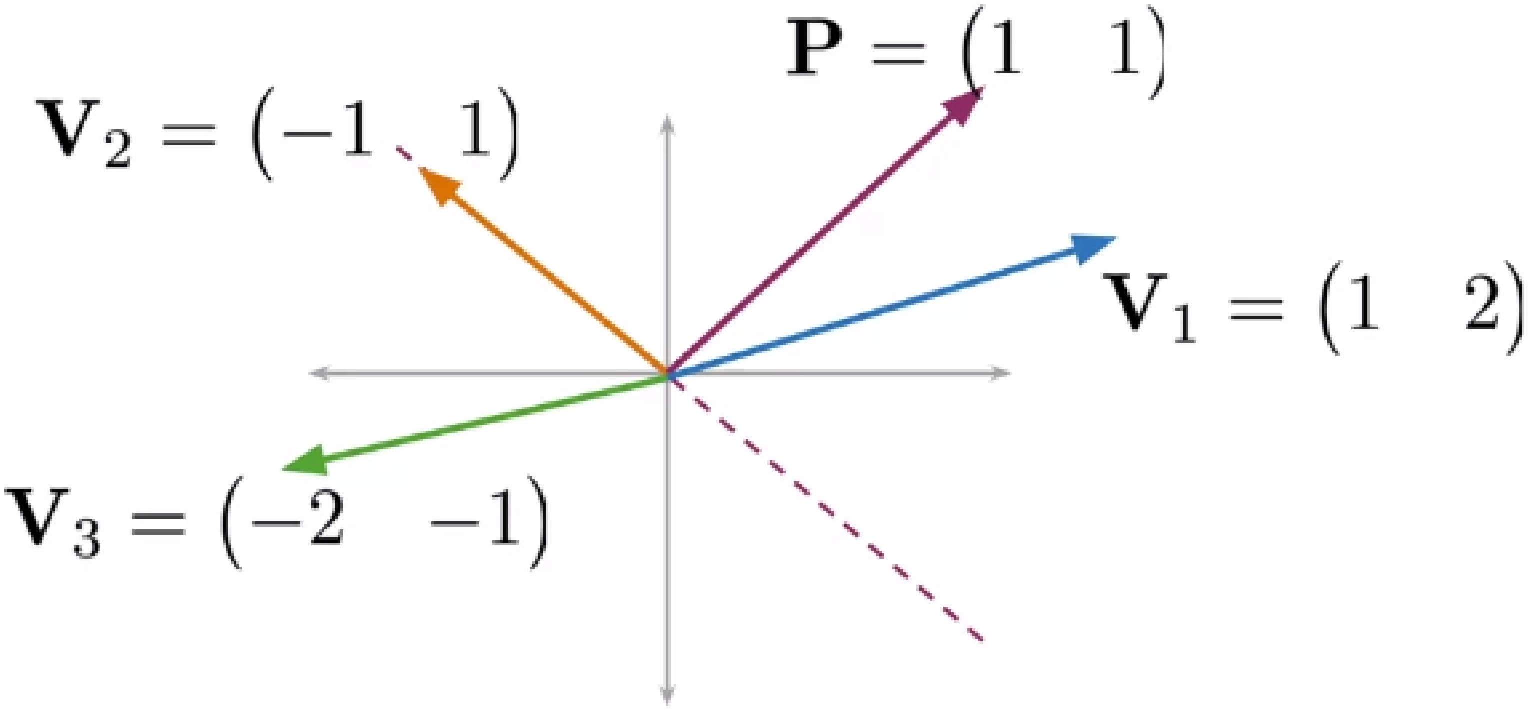
Which side of the plane?



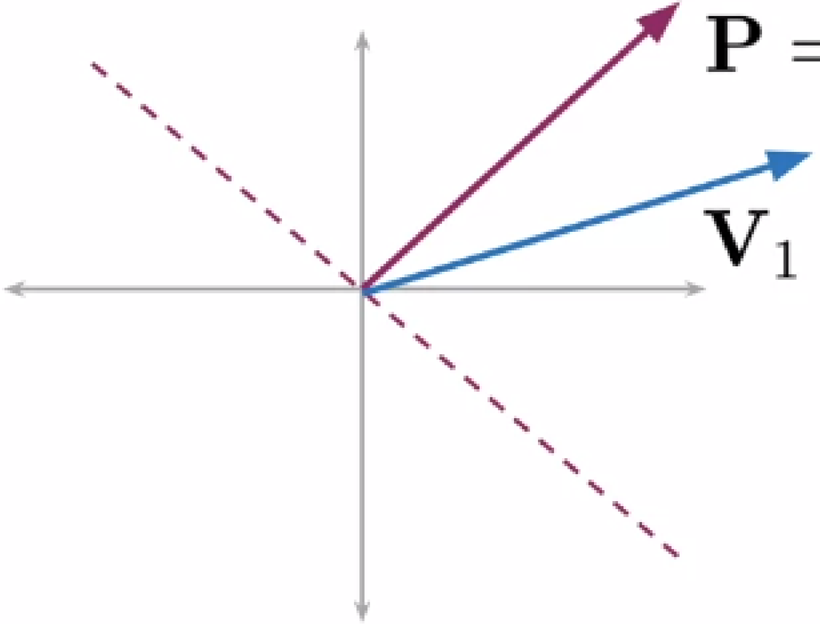
Which side of the plane?



Which side of the plane?



Which side of the plane?



$$\mathbf{P} = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

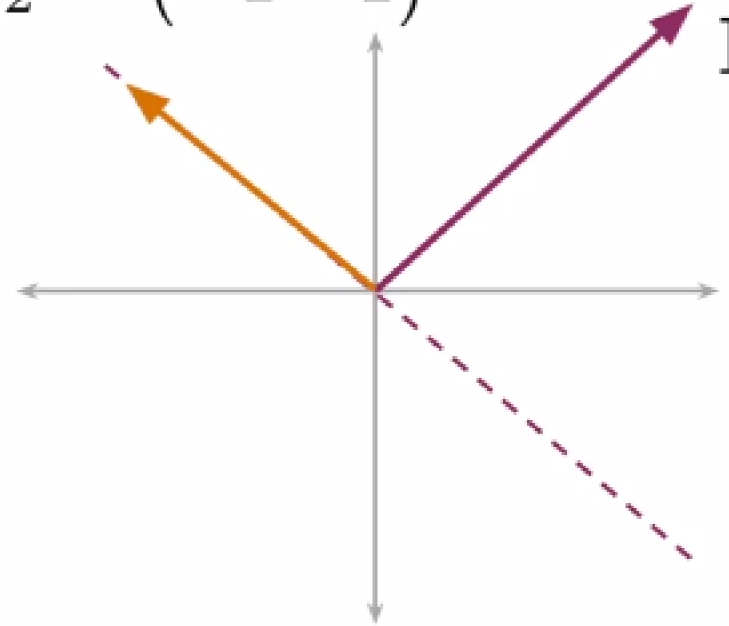
$$\mathbf{V}_1 = \begin{pmatrix} 1 & 2 \end{pmatrix}$$

$$\mathbf{P}\mathbf{V}_1^T = 3$$

Which side of the plane?

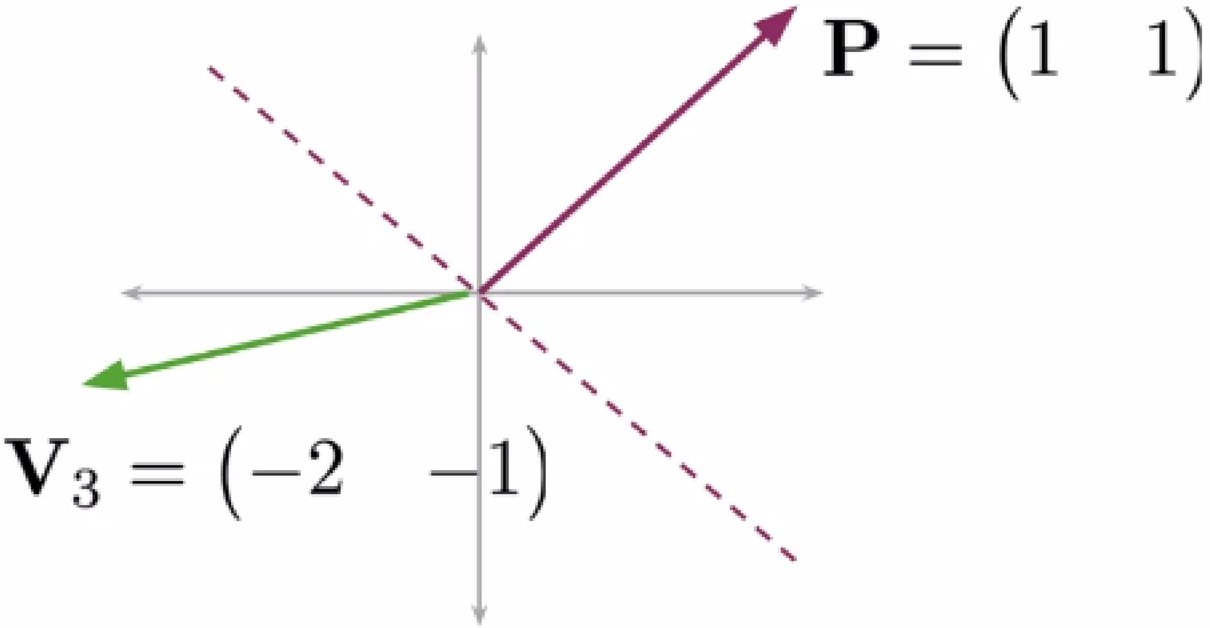
$\mathbf{V}_2 = \begin{pmatrix} -1 & 1 \end{pmatrix}$

$\mathbf{P} = \begin{pmatrix} 1 & 1 \end{pmatrix}$



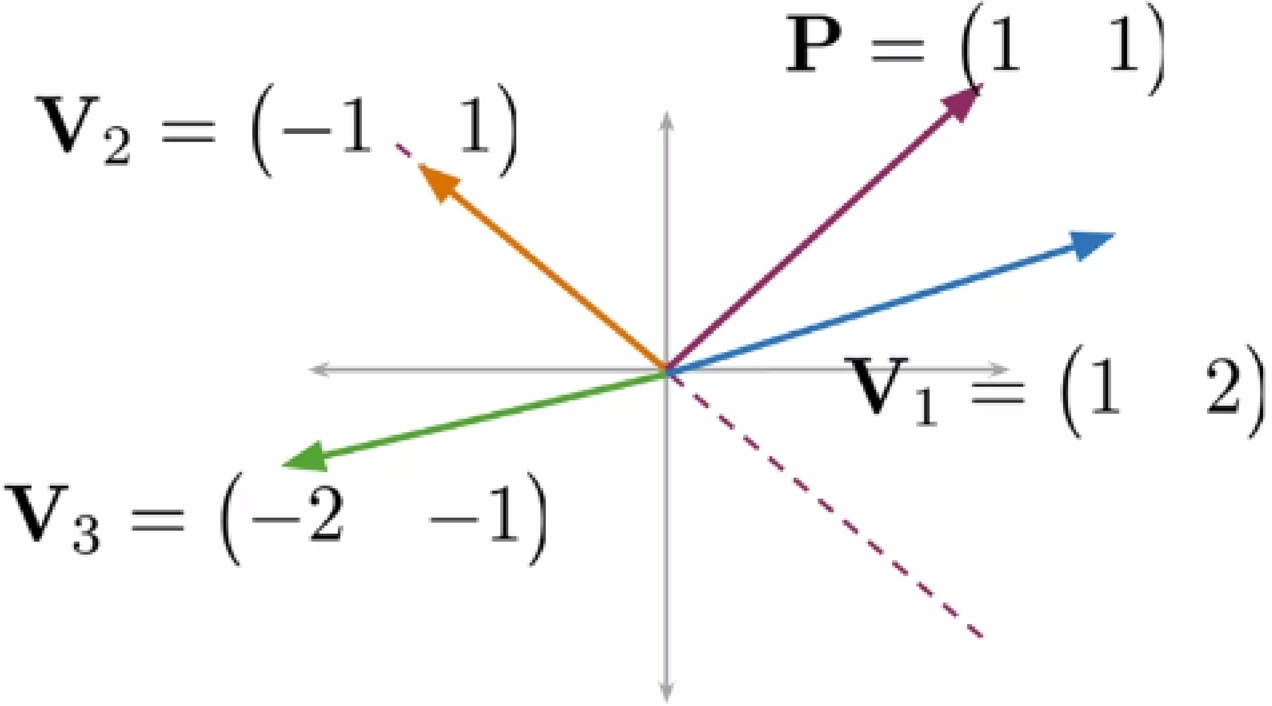
$$\mathbf{P}\mathbf{V}_2^T = 0$$

Which side of the plane?



$$\mathbf{P}\mathbf{V}_3^T = -3$$

Which side of the plane?

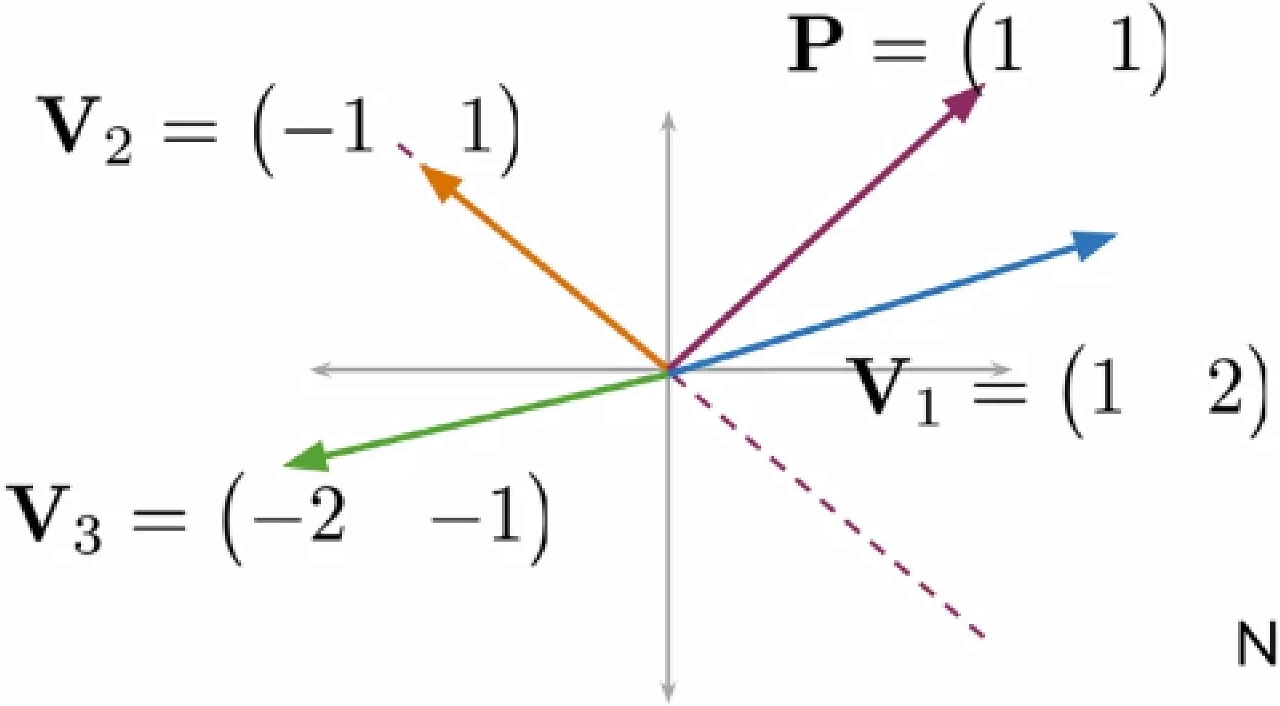


$$\mathbf{P}\mathbf{V}_1^T = 3$$

$$\mathbf{P}\mathbf{V}_2^T = 0$$

$$\mathbf{P}\mathbf{V}_3^T = -3$$

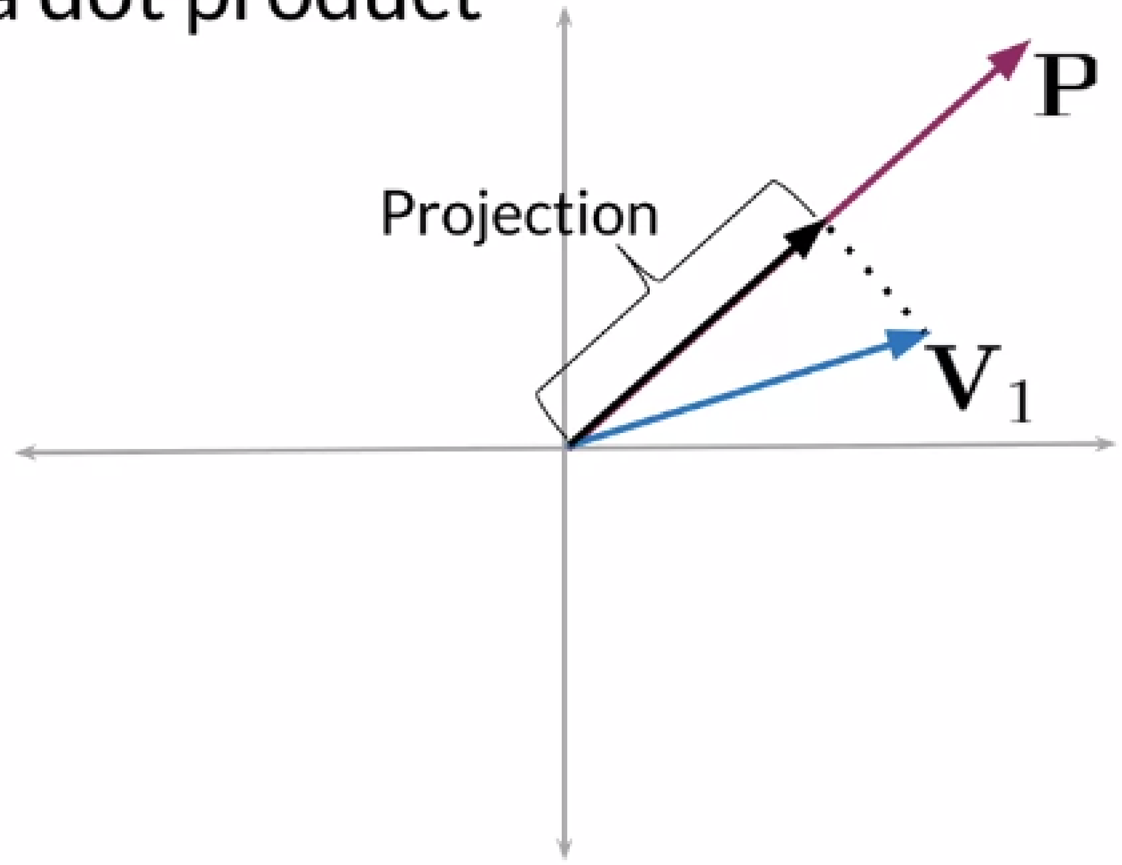
Which side of the plane?



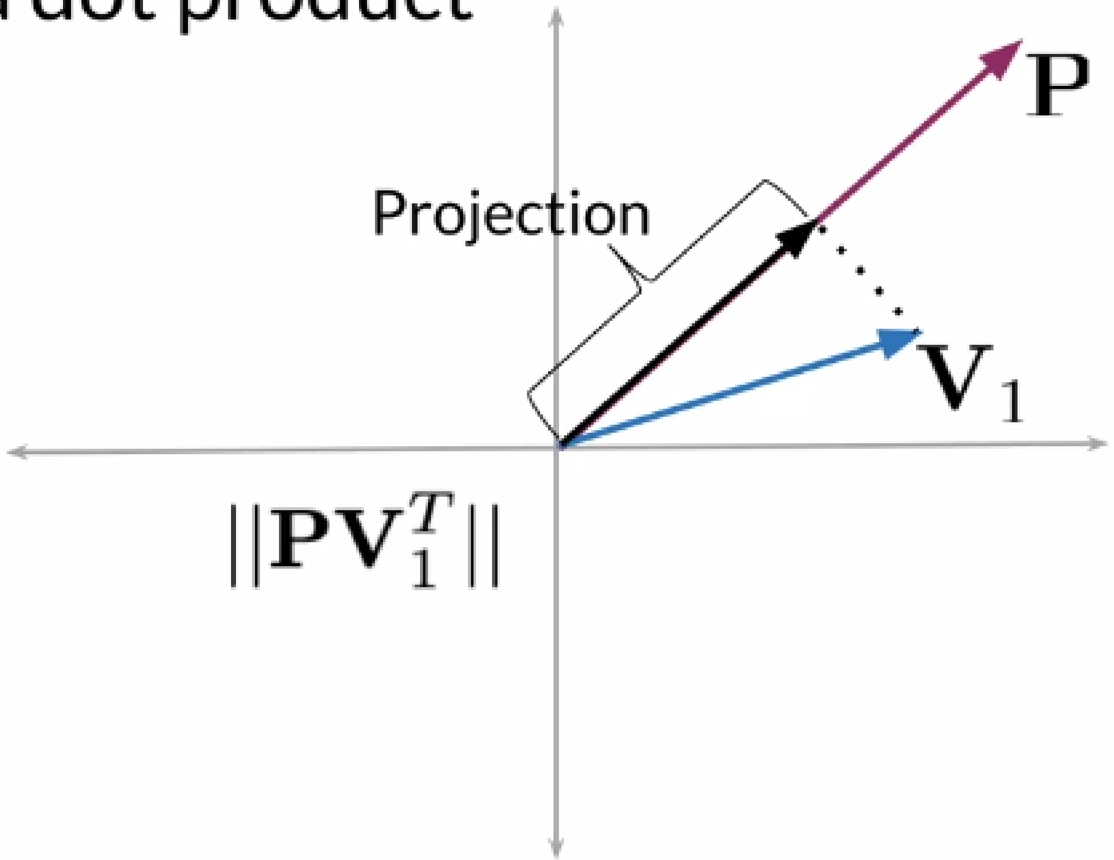
$$\mathbf{P}\mathbf{V}_1^T = 3$$
$$\mathbf{P}\mathbf{V}_2^T = 0$$
$$\mathbf{P}\mathbf{V}_3^T = -3$$

Notice the signs?

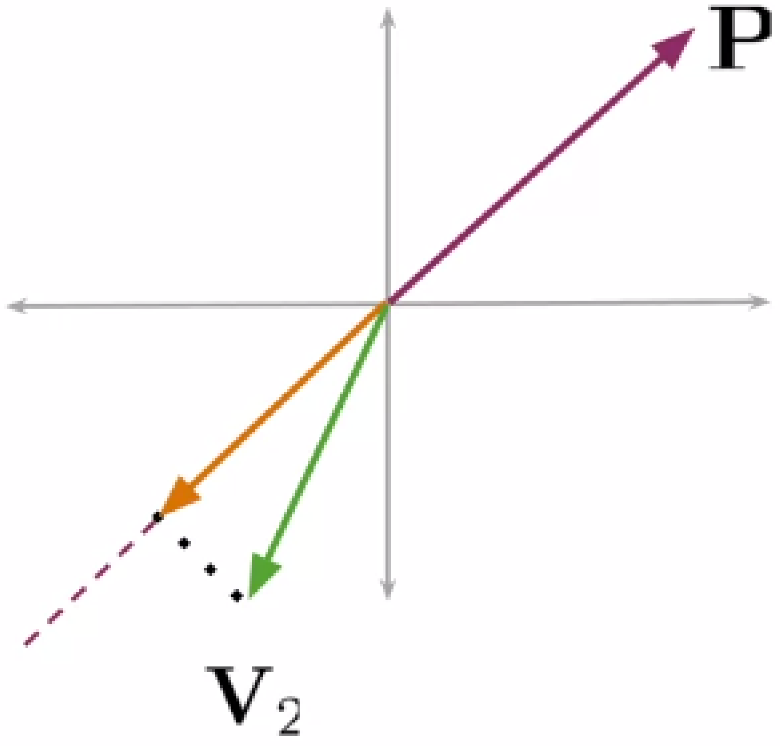
Visualizing a dot product



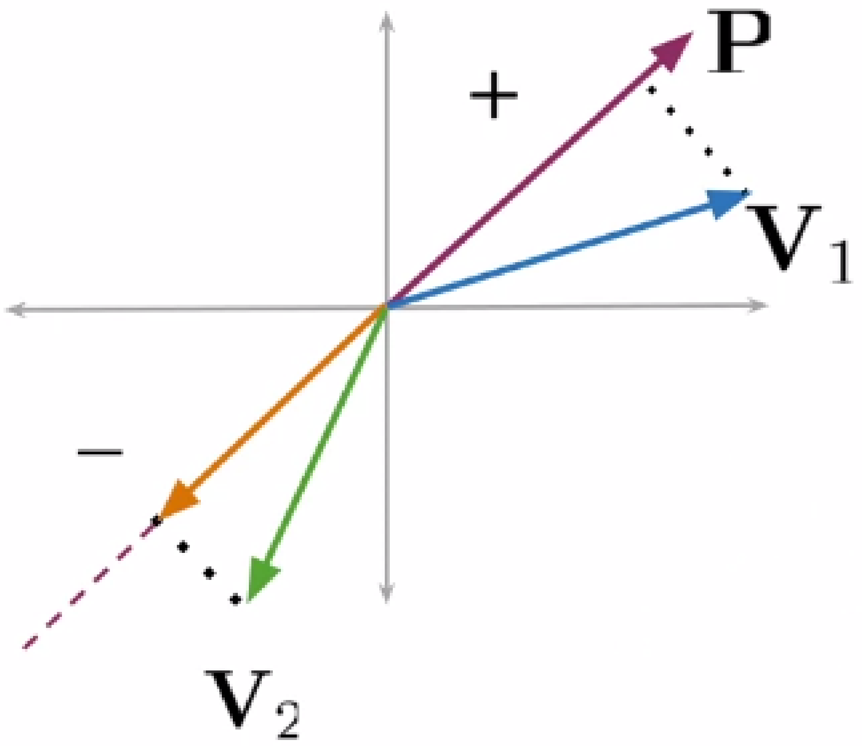
Visualizing a dot product



Visualizing a dot product



Visualizing a dot product



Sign indicates direction

Which side of the plane?

```
def side_of_plane(P,v):
    dotproduct = np.dot(P,v.T)
    sign_of_dot_product = np.sign(dotproduct)
    sign_of_dot_product_scalar= np.asscalar(sign_of_dot_product)
    return sign_of_dot_product_scalar
```

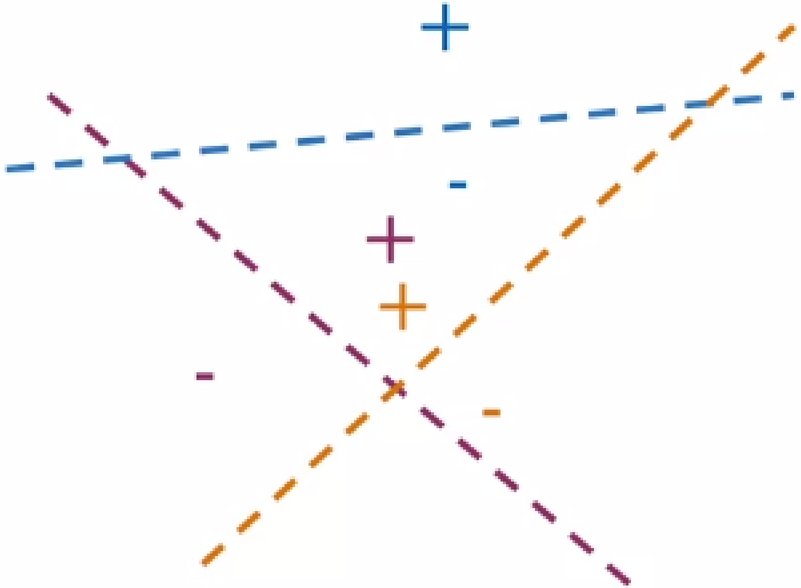
deeplearning.ai

Multiple Planes

Outline

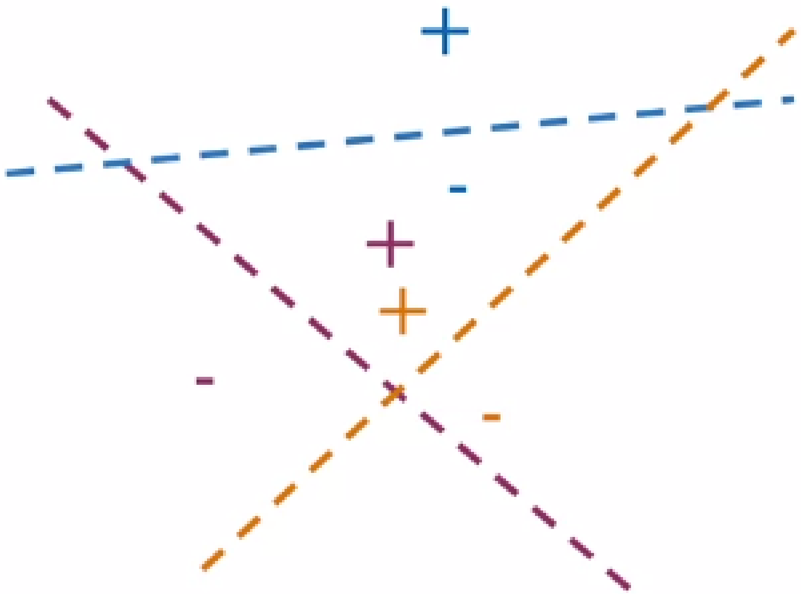
- Multiple planes $\longrightarrow$ Dot products $\longrightarrow$ Hash values

Multiple planes, single hash value?



$$\boxed{\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1}$$

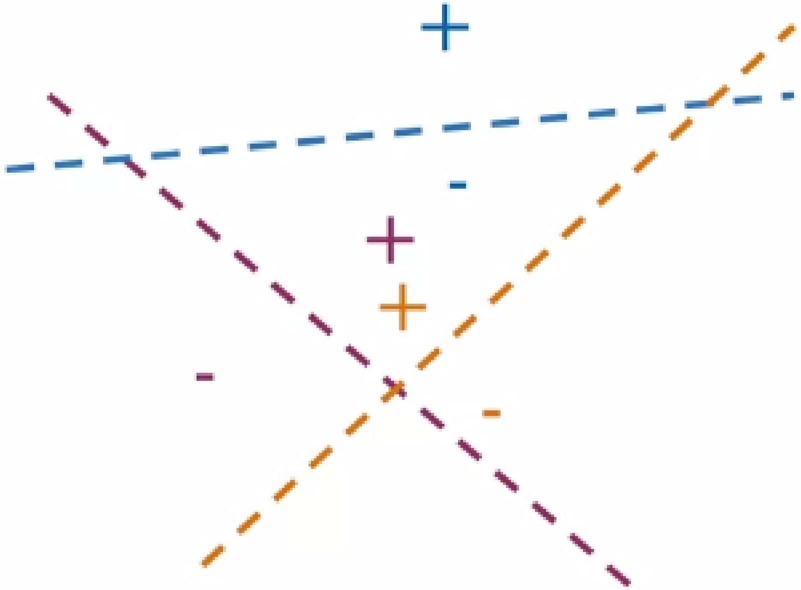
Multiple planes, single hash value?



$$\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1$$

$$\mathbf{P}_2\mathbf{v}^T = 5, sign_2 = +1, h_2 = 1$$

Multiple planes, single hash value?

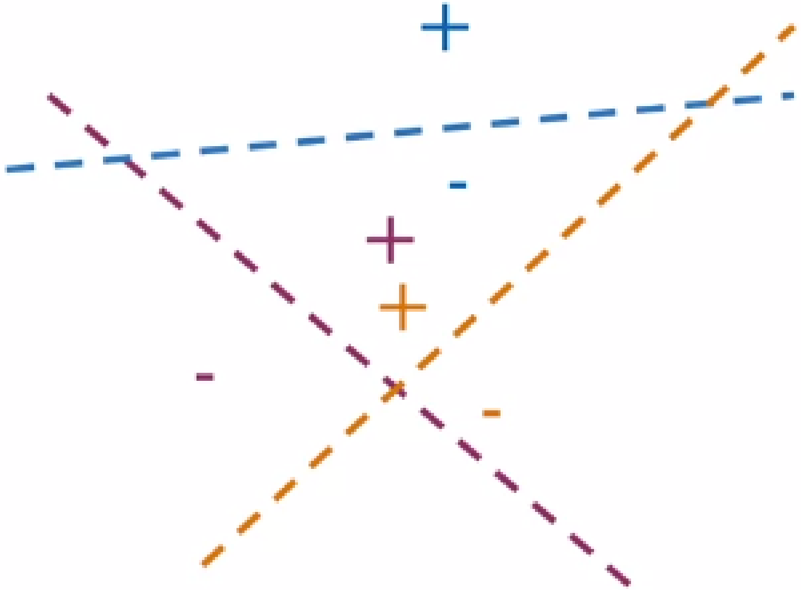


$$\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1$$

$$\mathbf{P}_2\mathbf{v}^T = 5, sign_2 = +1, h_2 = 1$$

$$\mathbf{P}_3\mathbf{v}^T = -2, sign_3 = -1, h_3 = 0$$

Multiple planes, single hash value?



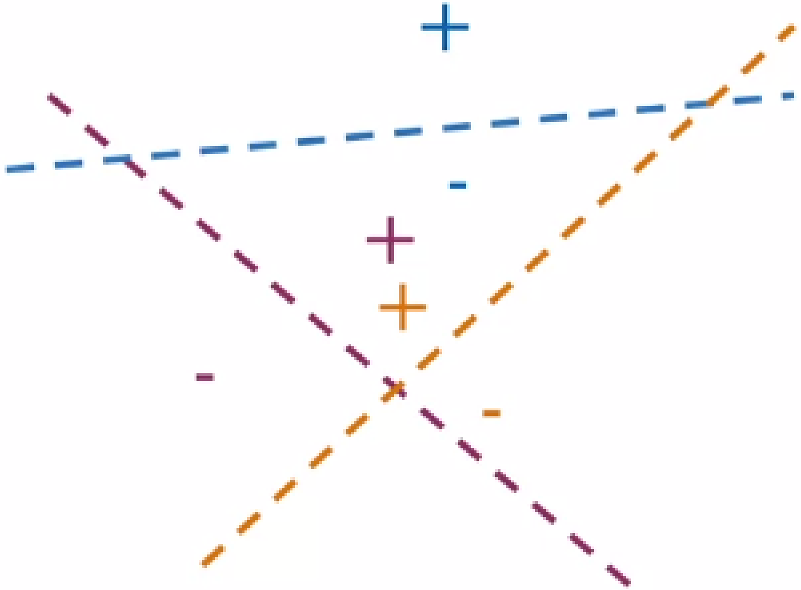
$$\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1$$

$$\mathbf{P}_2\mathbf{v}^T = 5, sign_2 = +1, h_2 = 1$$

$$\mathbf{P}_3\mathbf{v}^T = -2, sign_3 = -1, h_3 = 0$$

$$hash = 2^0 \times h_1 + 2^1 \times h_2 + 2^2 \times h_3$$

Multiple planes, single hash value?



$$\boxed{\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1}$$

$$\boxed{\mathbf{P}_2\mathbf{v}^T = 5, sign_2 = +1, h_2 = 1}$$

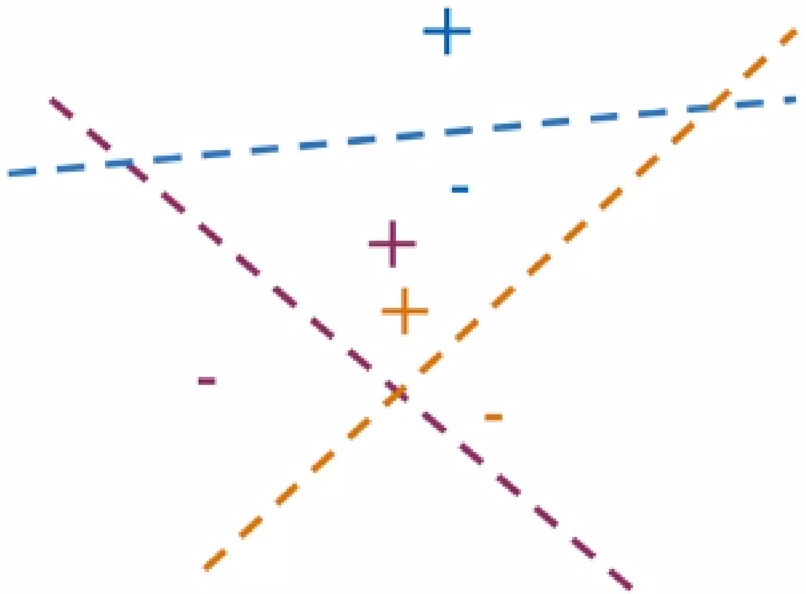
$$\boxed{\mathbf{P}_3\mathbf{v}^T = -2, sign_3 = -1, h_3 = 0}$$

$$hash = 2^0 \times h_1 + 2^1 \times h_2 + 2^2 \times h_3$$

$$= 1 \times 1 + 2 \times 1 + 4 \times 0$$

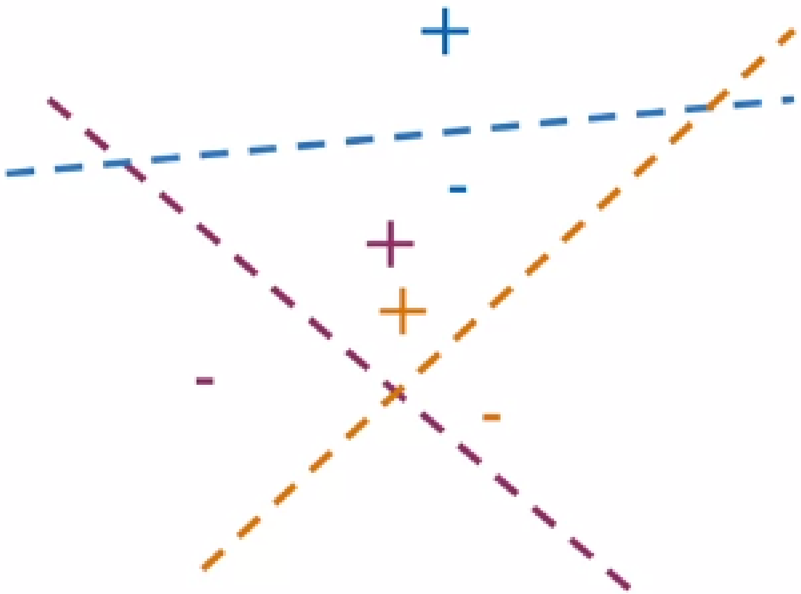
$$= 3$$

Multiple planes, single hash value!



$$sign_i \geq 0, \rightarrow h_i = 1$$

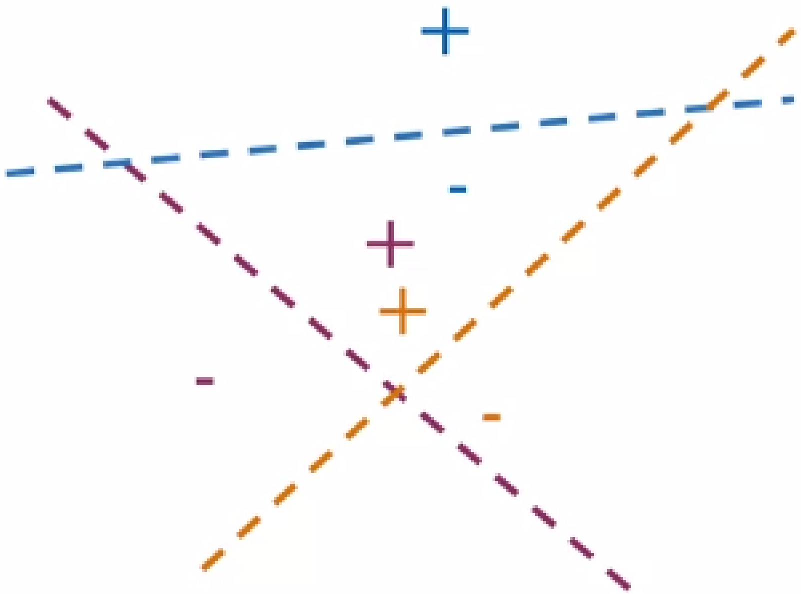
Multiple planes, single hash value!



$$sign_i \geq 0, \rightarrow h_i = 1$$

$$sign_i < 0, \rightarrow h_i = 0$$

Multiple planes, single hash value!



$$sign_i \geq 0, \rightarrow h_i = 1$$

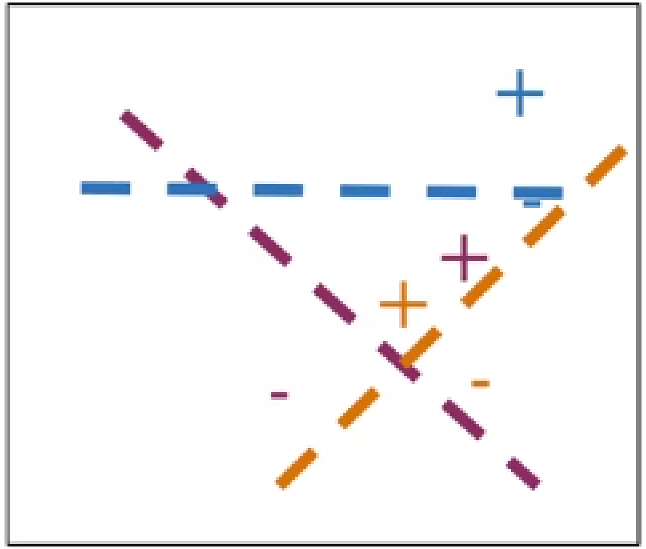
$$sign_i < 0, \rightarrow h_i = 0$$

$$\text{hash} = \sum_{i}^{H} 2^i \times h_i$$

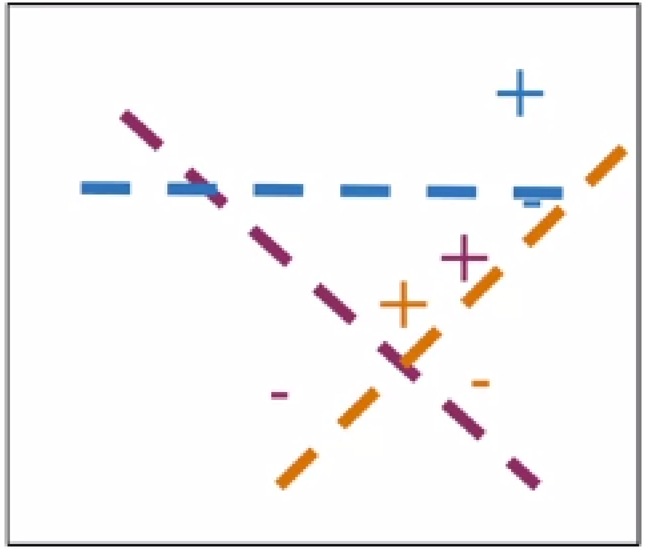
Multiple planes, single hash value!!

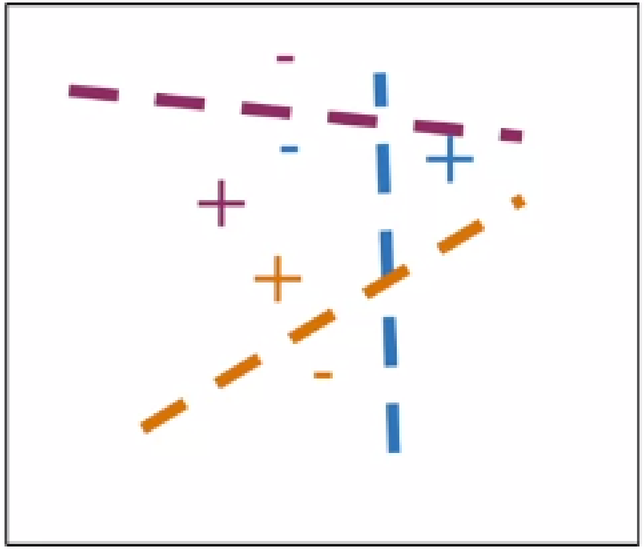
```
def hash_multiple_plane(P_l,v):

    hash_value = 0

    for i, P in enumerate(P_l):
        sign = side_of_plane(P,v)
        hash_i = 1 if sign >=0 else 0
        hash_value += 2**i * hash_i

    return hash_value
```

Random planes

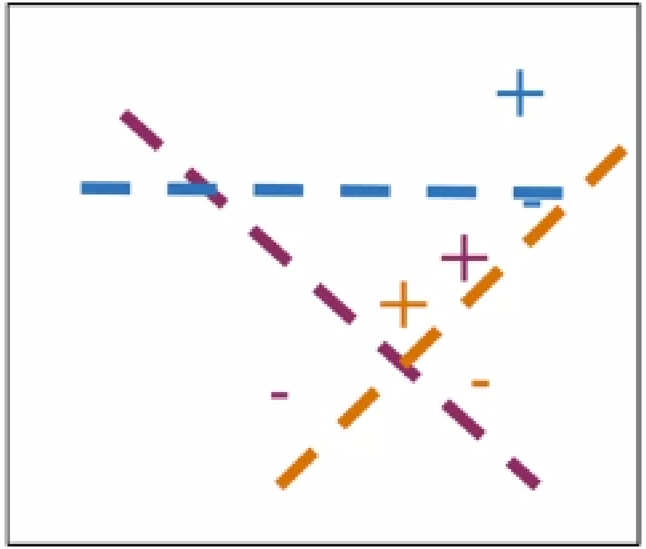


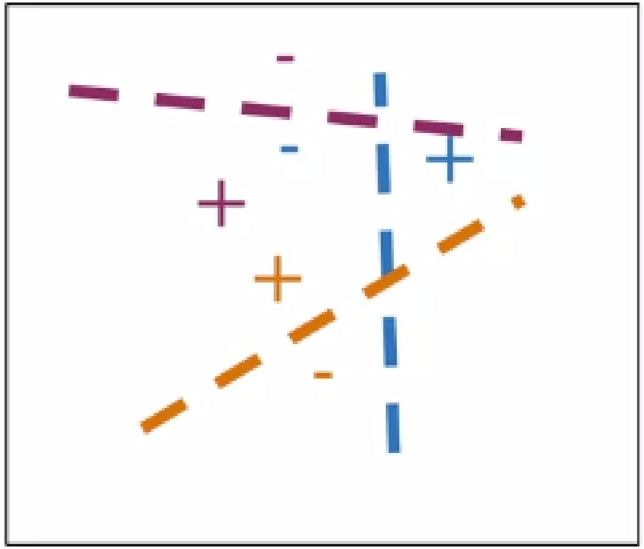
Random planes



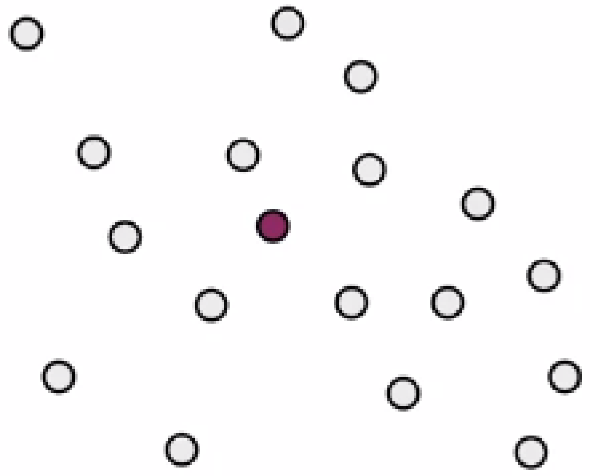


Random planes

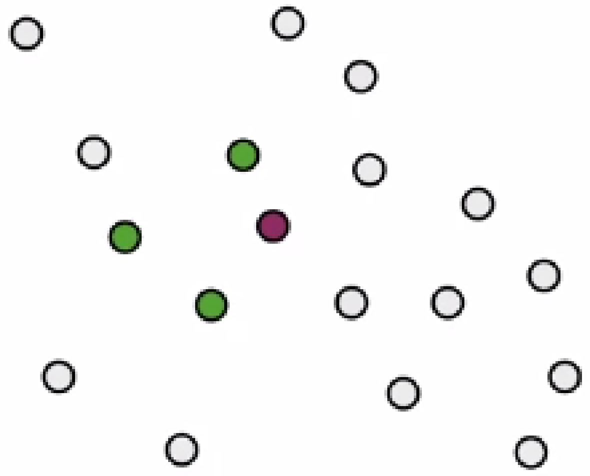




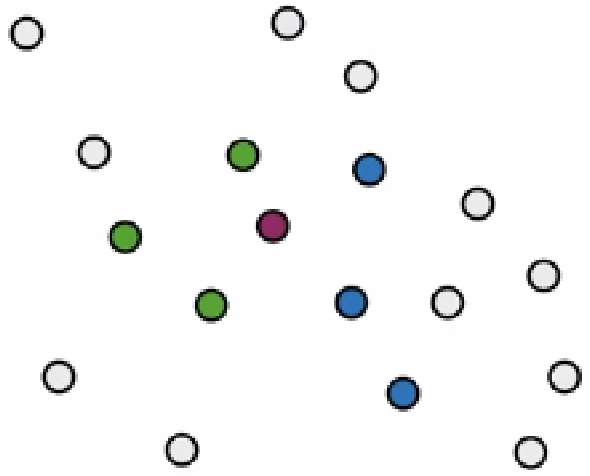
Multiple sets of random planes



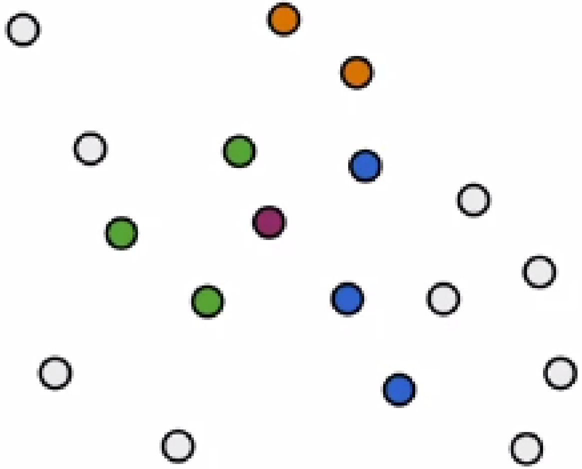
Multiple sets of random planes



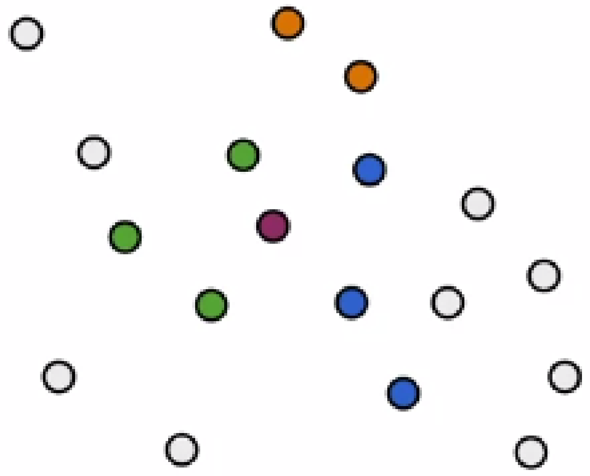
Multiple sets of random planes



Multiple sets of random planes



Multiple sets of random planes



Approximate nearest
(friendly) neighbors

Make one set of random planes

```
num_dimensions = 2 #300 in assignment
num_planes = 3 #10 in assignment

random_planes_matrix = np.random.normal(
                       size=(num_planes,
                             num_dimensions))
```

```
array([[ 1.76405235  0.40015721]
       [ 0.97873798  2.2408932 ]
       [ 1.86755799 -0.97727788]])
```

Make one set of random planes

```
num_dimensions = 2 #300 in assignment
num_planes = 3 #10 in assignment

random_planes_matrix = np.random.normal(
                       size=(num_planes,
                             num_dimensions))
```

```
array([[ 1.76405235  0.40015721]
       [ 0.97873798  2.2408932 ]
       [ 1.86755799 -0.97727788]])
```

```
v = np.array([[2,2]])
```

Make one set of random planes

```
num_dimensions = 2 #300 in assignment
num_planes = 3 #10 in assignment

random_planes_matrix = np.random.normal(
                       size=(num_planes,
                             num_dimensions))
```

```
array([[ 1.76405235  0.40015721]
       [ 0.97873798  2.2408932 ]
       [ 1.86755799 -0.97727788]])
```

```
v = np.array([[2,2]])
```

```
def side_of_plane_matrix(P,v):
    dotproduct = np.dot(P,v.T)
    sign_of_dot_product = np.sign(dotproduct)
    return sign_of_dot_product

num_planes_matrix = side_of_plane_matrix(
                    random_planes_matrix,v)
```

```
array([[1.]
       [1.]
       [1.]])
```

Make one set of random planes

```
num_dimensions = 2 #300 in assignment
num_planes = 3 #10 in assignment

random_planes_matrix = np.random.normal(
                       size=(num_planes,
                             num_dimensions))
```

```
array([[ 1.76405235  0.40015721]
       [ 0.97873798  2.2408932 ]
       [ 1.86755799 -0.97727788]])
```

```
v = np.array([[2,2]])
```

```
def side_of_plane_matrix(P,v):
     dotproduct = np.dot(P,v.T)
     sign_of_dot_product = np.sign(dotproduct)
     return sign_of_dot_product

num_planes_matrix = side_of_plane_matrix(
                    random_planes_matrix,v)
```

```
array([[1.]
       [1.]
       [1.])
```

See notebook for calculating the hash value!

Document representation

I love learning!

Document representation

I love learning! [?, ?, ?]

Document representation

I love learning!	[?, ?, ?]
I	[1, 0, 1]
love	[-1, 0, 1]
learning	[1, 0, 1]

Document representation

I love learning!	[?, ?, ?]
I	[1, 0, 1]
	+
love	[-1, 0, 1]
	+
learning	[1, 0, 1]
	=
I love learning!	[1, 0, 3]

Document vectors

```
word_embedding = {“I”: np.array([1,0,1]),
                  “love”: np.array([-1,0,1]),
                  “learning”: np.array([1,0,1])}

words_in_document = [‘I’, ‘love’, ‘learning’]

document_embedding = np.array([0,0,0])

for word in words_in_document:
    document_embedding += word_embedding.get(word,0)

print(document_embedding)
```

```
array([1 0 3])
```