Computational Tools for Big Data

Feature Hashing and Locality Sensitive Hashing

Bag of words

1. John likes to watch movies
2. Mary likes movies too
3. John also likes football

John	likes	to	watch	movies	Mary	too	also	football
1	1	1	1	1	0	0	0	0
0	1	0	0	1	1	1	0	0
1	1	0	0	0	0	0	1	1

Bag of words

Problems with bag-of-words:
- Term/document matrix can take up a large amount of space
- Its size grows with the training data
- If we fix the features before-hand, what if we see a new word?

4.John also likes *basketball*

John	likes	to	watch	movies	Mary	too	also	football
1	1	1	1	1	0	0	0	0
0	1	0	0	1	1	1	0	0
1	1	0	0	0	0	0	1	1

Feature hashing

Choose a fixed amount of features/buckets and pick a hash-function
Hash each word to a bucket index and add 1 there

1:	A B C D E F G
2:	H I J K L M N
3:	O P Q R S T
4:	U V W X Y Z

1. John likes to watch movies
2. Mary likes movies too
3. John also likes football

1	2	3	4
0	3	1	1
0	3	1	0
2	2	0	0

Feature hashing

4.John also likes *basketball*

1: A B C D E F G

2: H I J K L M N

3: O P Q R S T

4: U V W X Y Z

1.John likes to watch movies
2.Mary likes movies too
3.John also likes football

1	2	3	4
0	3	1	1
0	3	1	0
2	2	0	0
2	2	0	0

Feature hashing

The biggest problem is that features collide.

In practise, take a better hash function than the one shown here
Use more buckets (maybe 10^6)

Experience shows that feature hashing performs much better than you would expect.

Locality sensitive hashing

Normally a hash function is meant to randomly distribute things and avoid collisions.

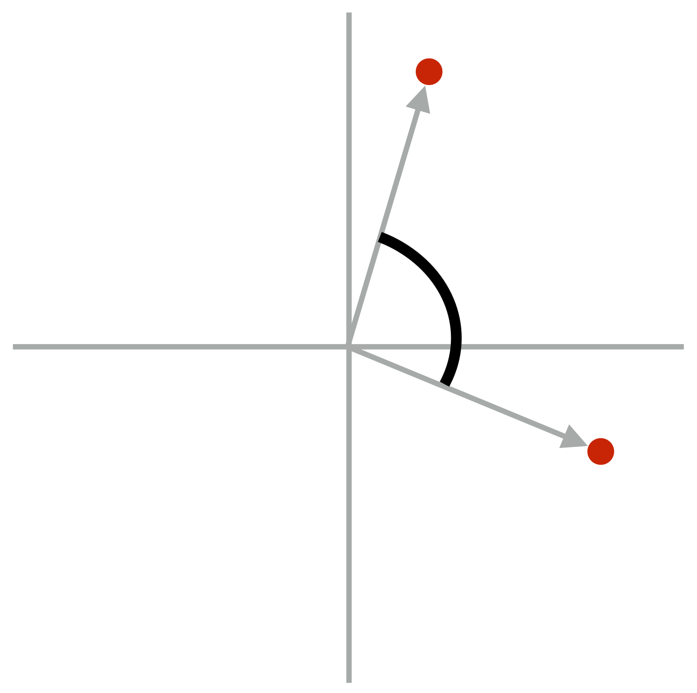
With a locality sensitive hash (LSH) function we want similar things to collide and end up in the same bucket.

You need to define what you mean by “close” before you come up with a suitable LSH-function.

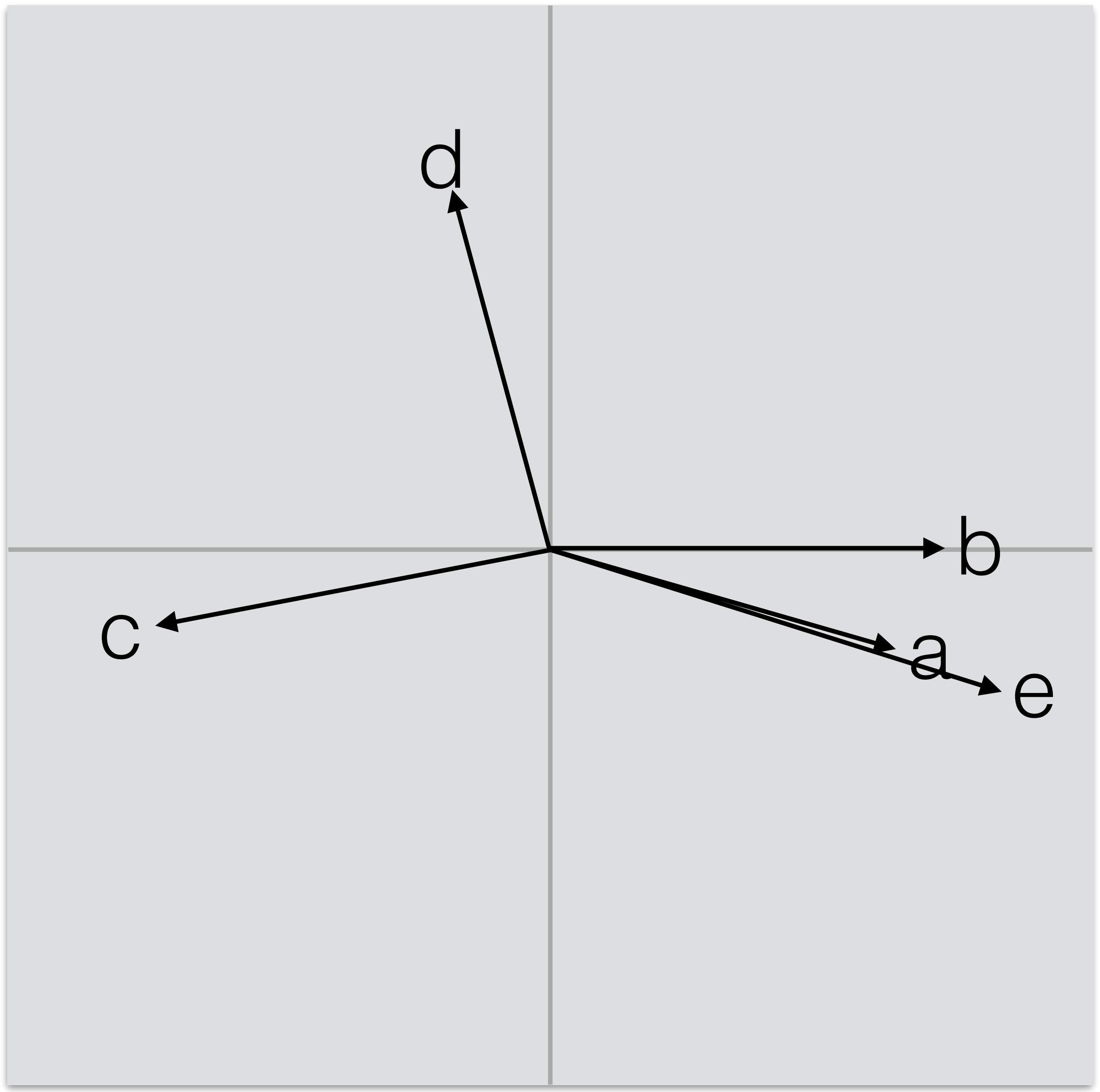
Random projections

A locality sensitive hash approach for approximating cosine-distances between points.

Points (vectors) which have similar cosine-distance will likely end up in same buckets.



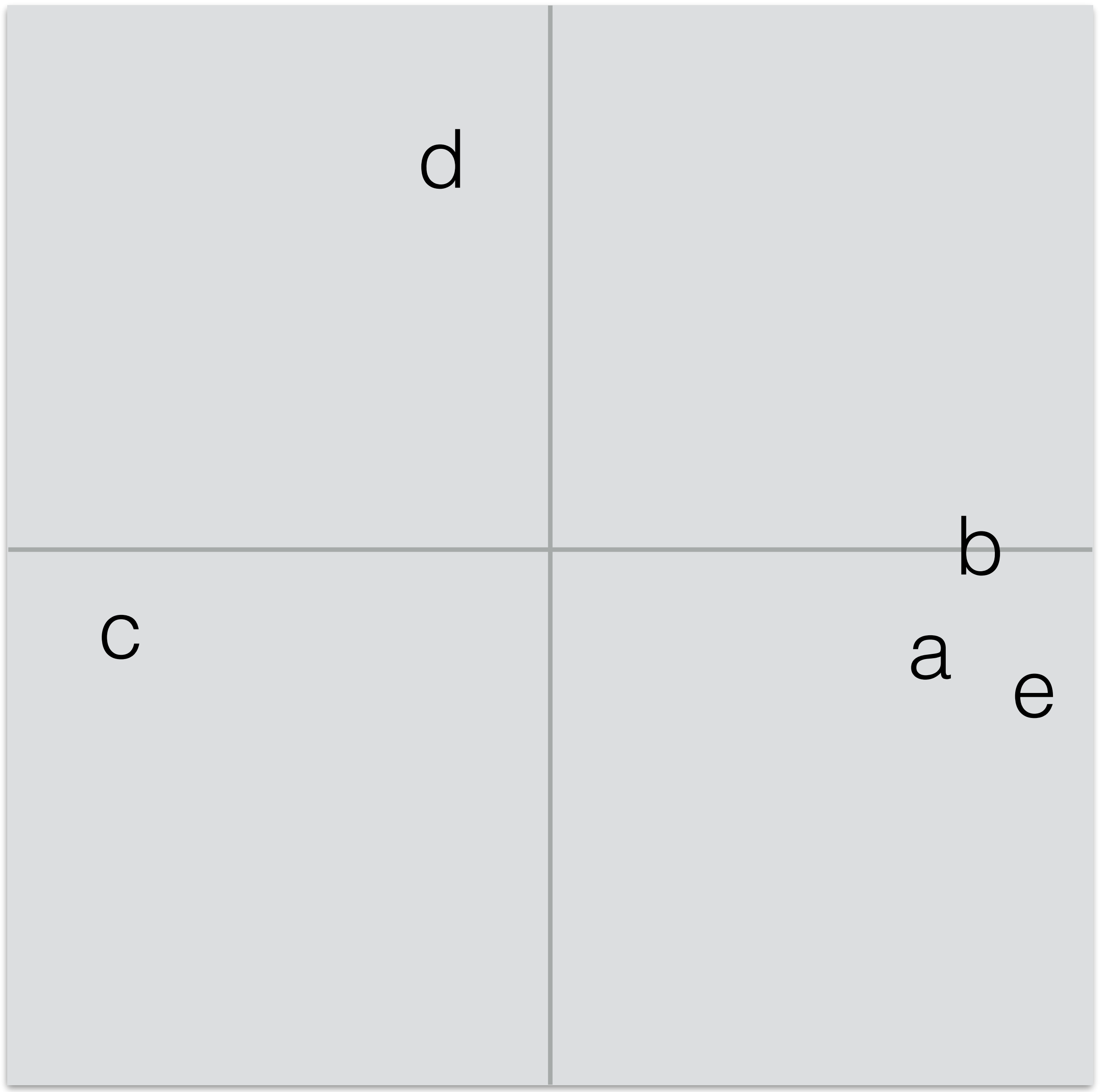
Random projections



Example in 2 dimensions

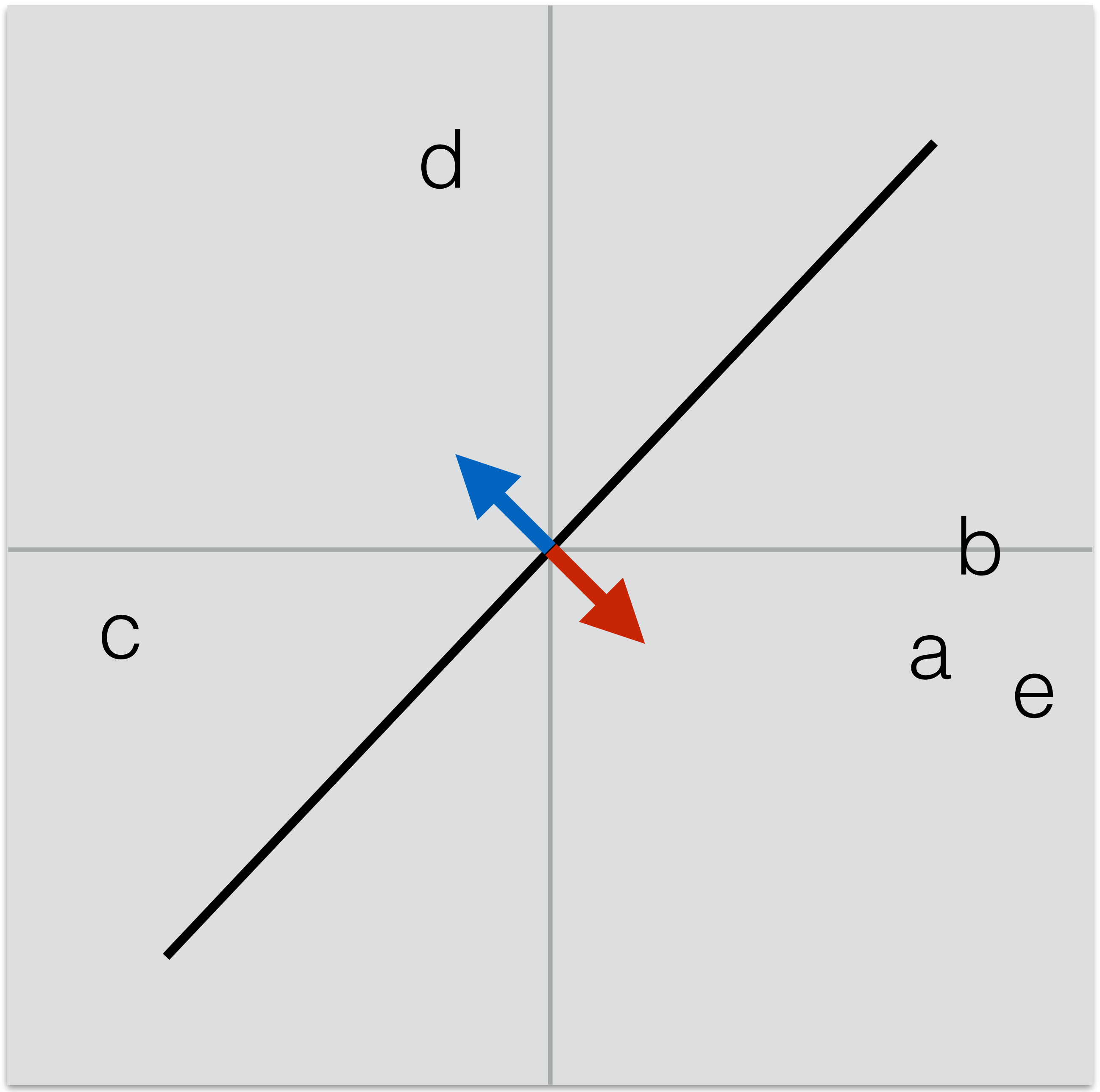
a and e are very similar
a,b,e are somewhat similar

Random projections



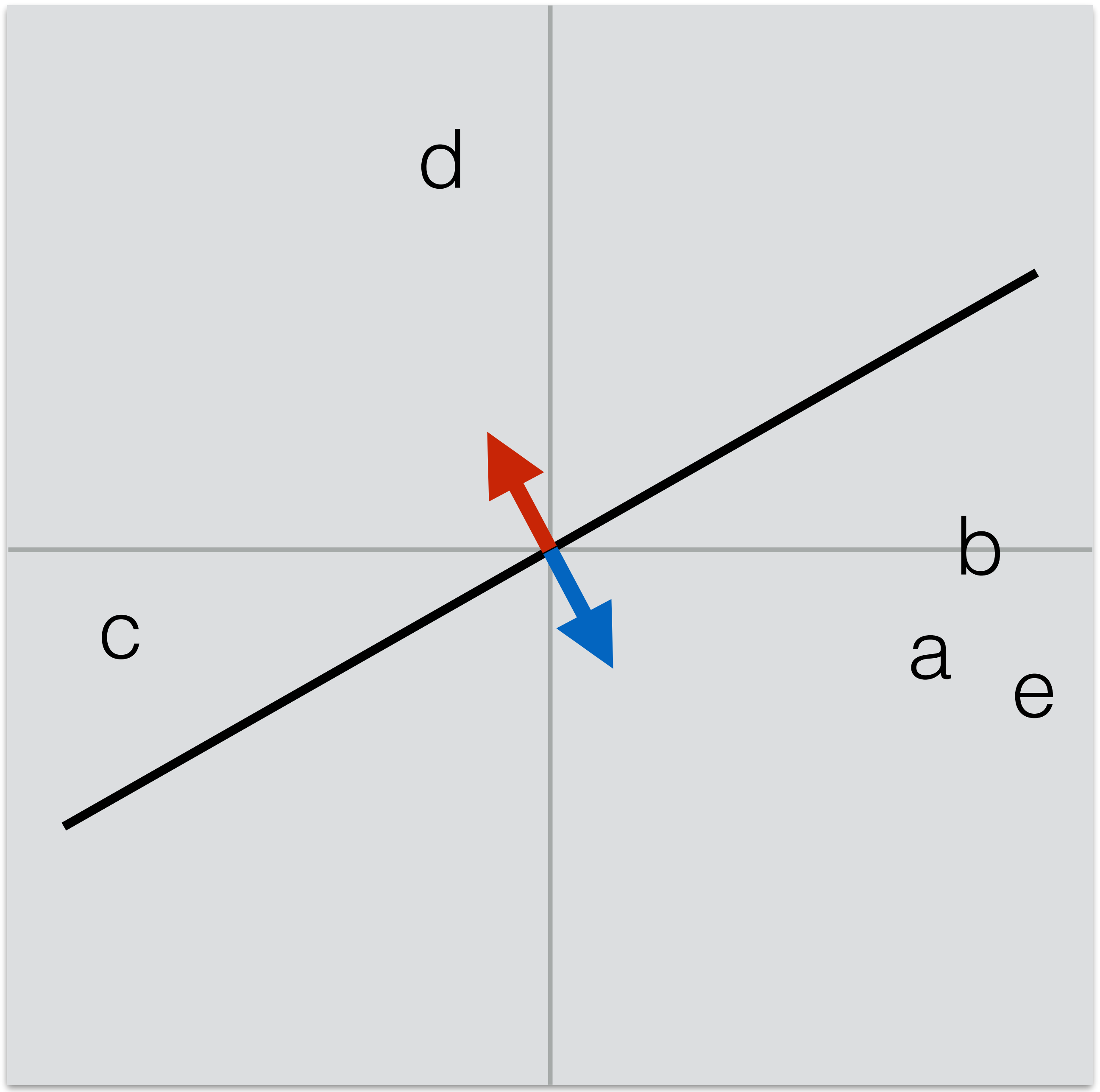
a									
b									
c									
d									
e									

Random projections



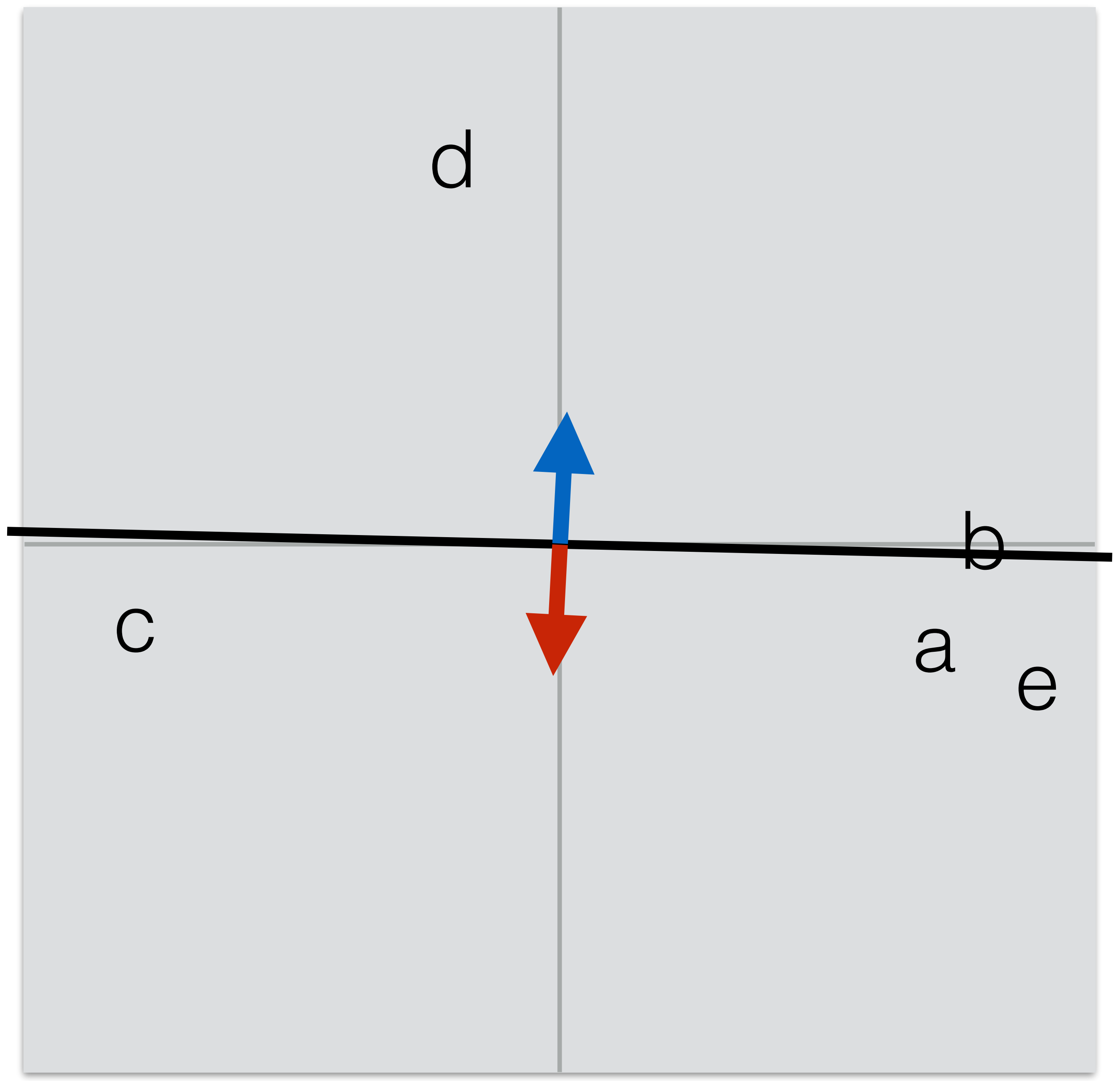
a	🟥								
b	🟥								
c	🟦								
d	🟦								
e	🟥								

Random projections



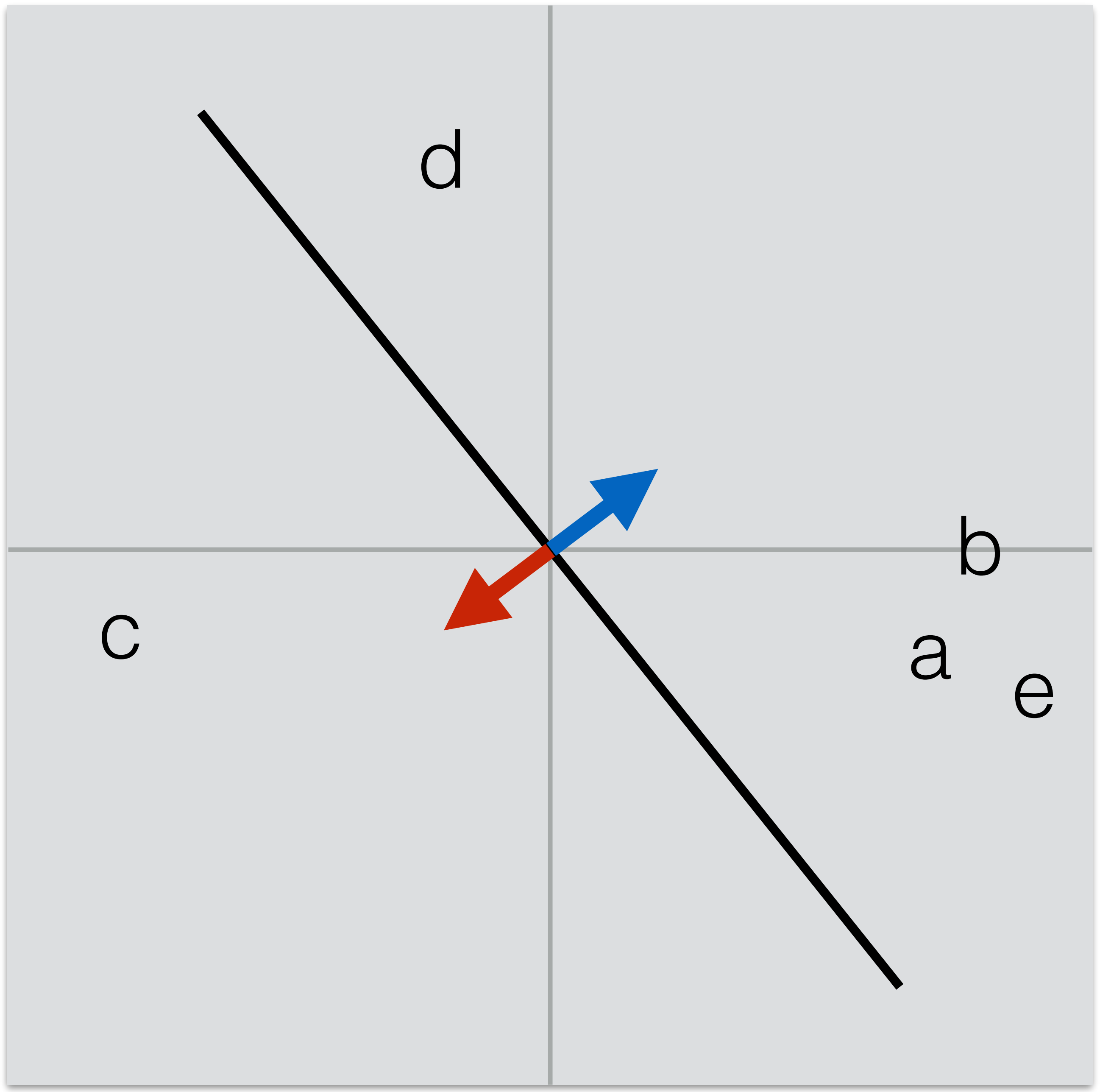
a	🟥	🟦							
b	🟥	🟦							
c	🟦	🟥							
d	🟦	🟥							
e	🟥	🟦							

Random projections



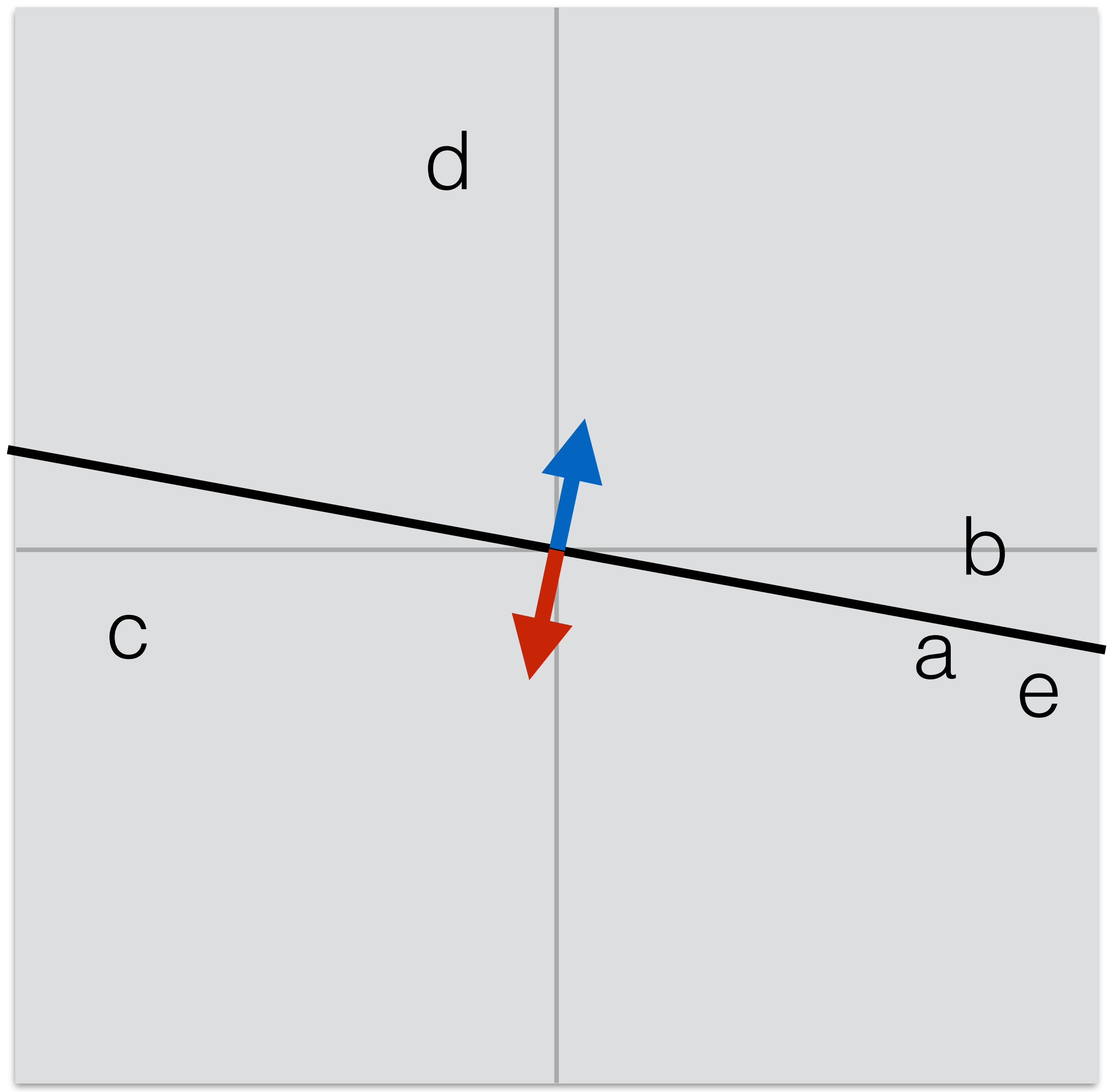
a	🟥	🟦	🟥						
b	🟥	🟦	🟦						
c	🟦	🟥	🟥						
d	🟦	🟥	🟦						
e	🟥	🟦	🟥						

Random projections



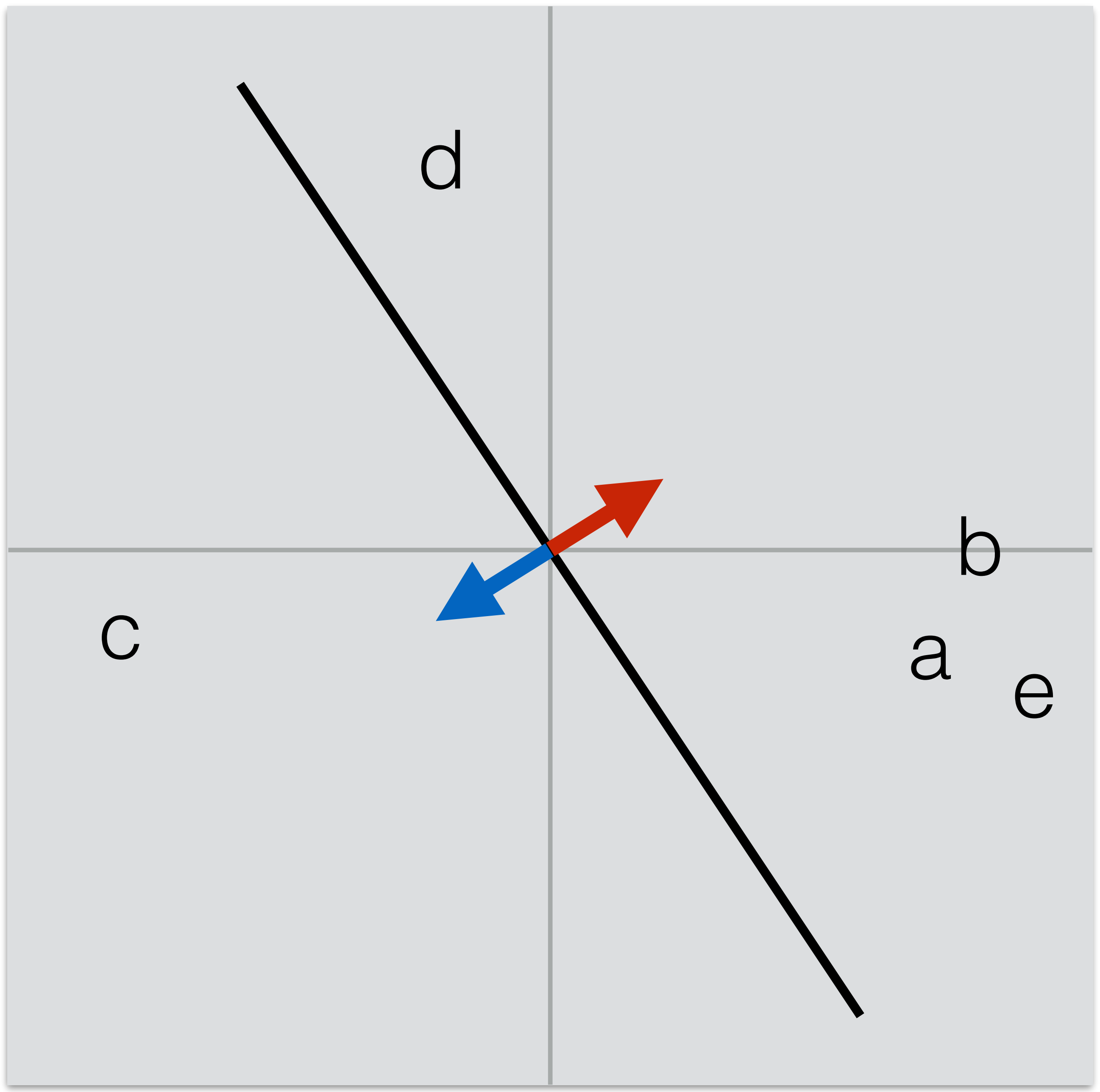
a	🟥	🟦	🟥	🟦					
b	🟥	🟦	🟦	🟦					
c	🟦	🟥	🟥	🟥					
d	🟦	🟥	🟦	🟦					
e	🟥	🟦	🟥	🟦					

Random projections



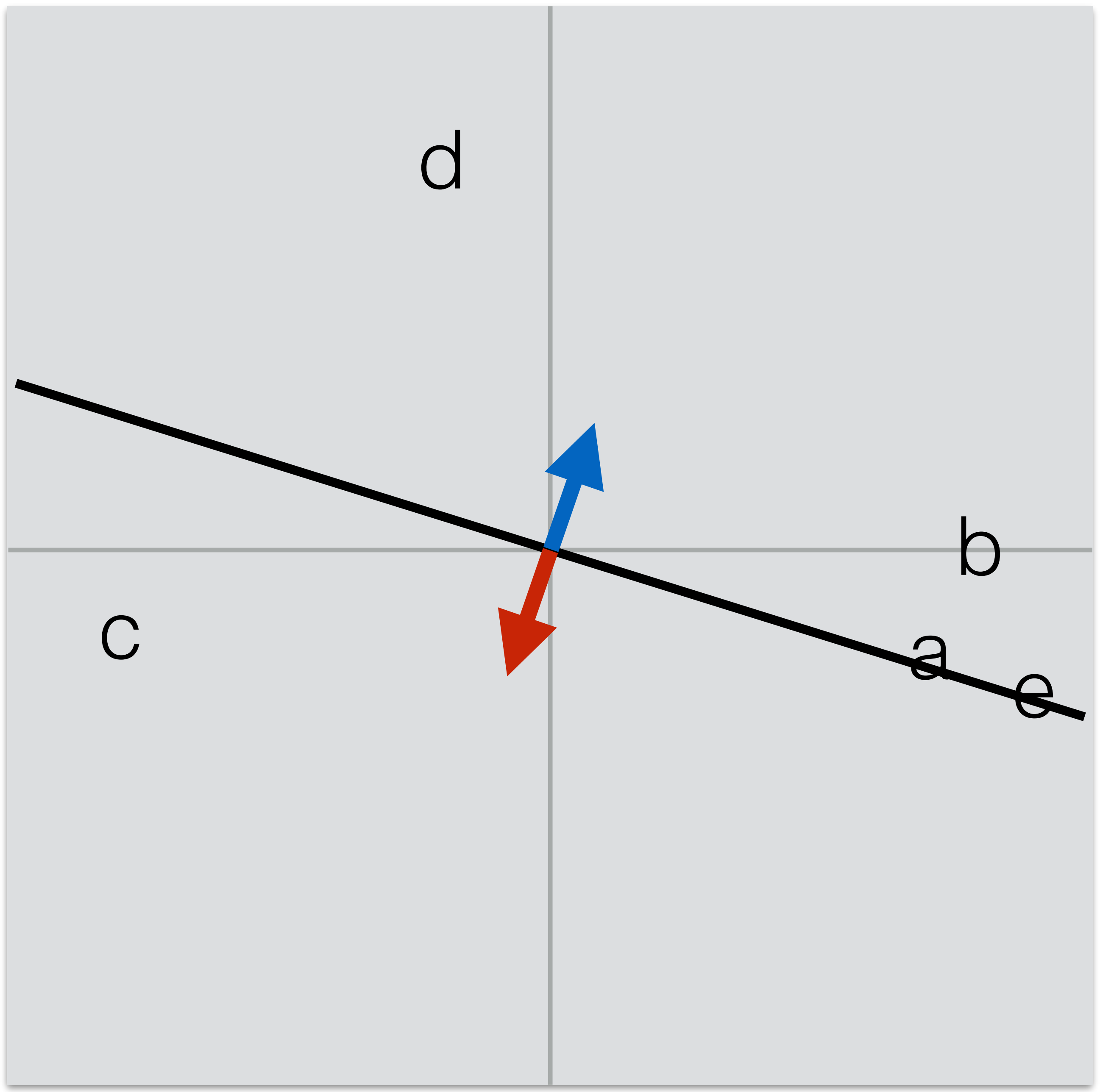
a	🟥	🟦	🟥	🟦	🟥				
b	🟥	🟦	🟦	🟦	🟦				
c	🟦	🟥	🟥	🟥	🟥				
d	🟦	🟥	🟦	🟦	🟦				
e	🟥	🟦	🟥	🟦	🟥				

Random projections



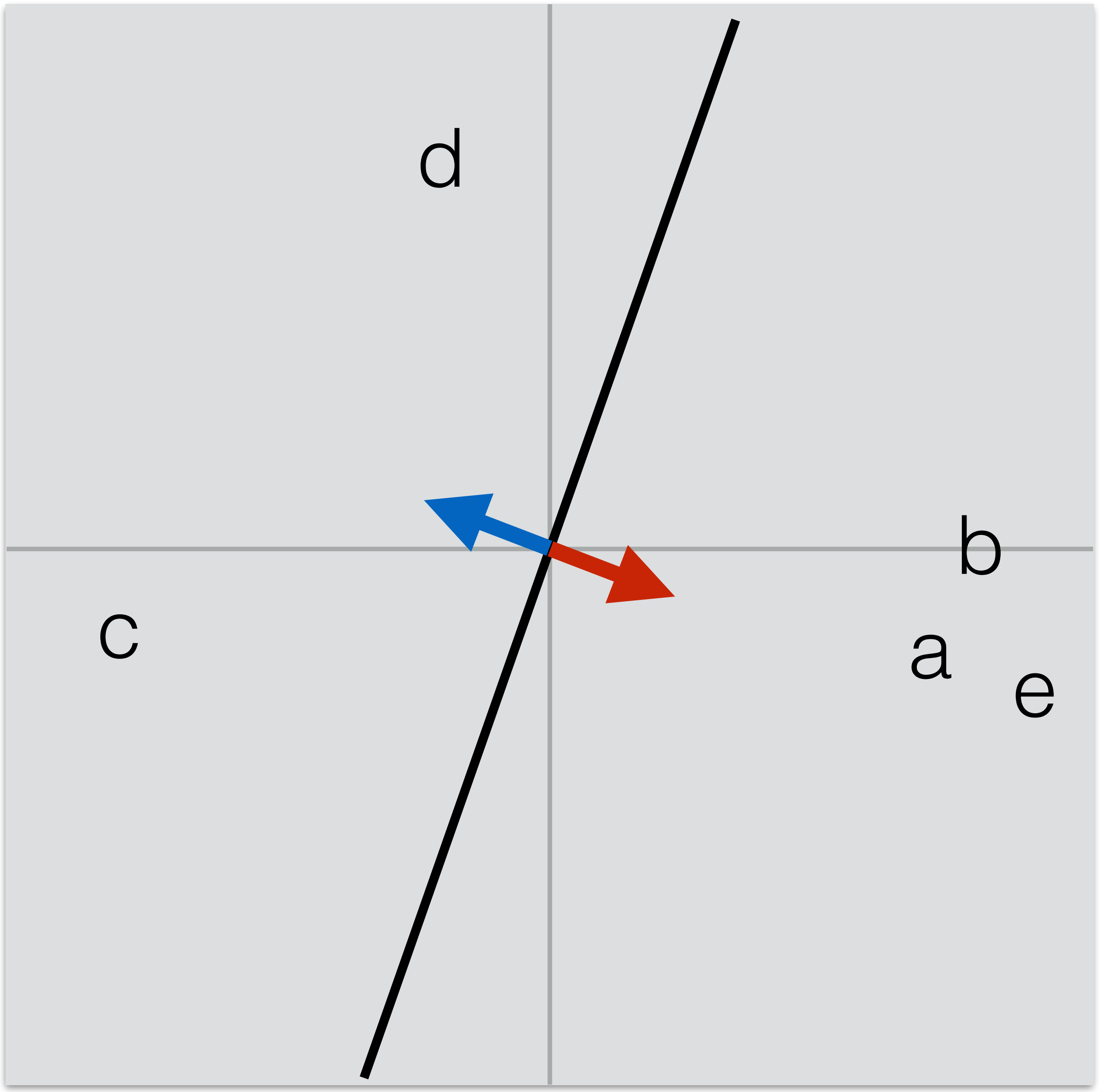
a	🟥	🟦	🟥	🟦	🟥	🟥			
b	🟥	🟦	🟦	🟦	🟦	🟥			
c	🟦	🟥	🟥	🟥	🟥	🟦			
d	🟦	🟥	🟦	🟦	🟦	🟥			
e	🟥	🟦	🟥	🟦	🟥	🟥			

Random projections



a	🟥	🟦	🟥	🟦	🟥	🟥	🟦		
b	🟥	🟦	🟦	🟦	🟦	🟥	🟦		
c	🟦	🟥	🟥	🟥	🟥	🟦	🟥		
d	🟦	🟥	🟦	🟦	🟦	🟥	🟦		
e	🟥	🟦	🟥	🟦	🟥	🟥	🟦		

Random projections



a	🟥	🟦	🟥	🟦	🟥	🟥	🟦	🟦	🟥
b	🟥	🟦	🟦	🟦	🟦	🟥	🟦	🟦	🟥
c	🟦	🟥	🟥	🟥	🟥	🟦	🟥	🟥	🟦
d	🟦	🟥	🟦	🟦	🟦	🟥	🟦	🟥	🟦
e	🟥	🟦	🟥	🟦	🟥	🟥	🟦	🟦	🟥

Points a and e end in the same bucket.

ImageHash

We want to make an LSH the hashes similar images together.

ImageHash

1. Grayscale the images
2. Resize to fx. 9 x 8 pixels
3. Compare adjacent values
4. Convert to hash

ImageHash

1. **Grayscale the images**
2. Resize to fx. 9 x 8 pixels
3. Compare adjacent values
4. Convert to hash

ImageHash

1. Grayscale the images
2. **Resize to fx. 9 x 8 pixels**
3. Compare adjacent values
4. Convert to hash

ImageHash

1. Grayscale the images
2. Resize to fx. 9 x 8 pixels
3. **Compare adjacent values (x > y)**
4. **Convert to hash**

```
[254, 254, 255, 253, 248, 254, 255, 254, 255]
[254, 255, 253, 248, 254, 255, 254, 255, 255]
[253, 248, 254, 255, 254, 255, 255, 255, 222]
[248, 254, 255, 254, 255, 255, 255, 222, 184]
[254, 255, 254, 255, 255, 255, 222, 184, 177]
[255, 254, 255, 255, 255, 222, 184, 177, 184]
[254, 255, 255, 255, 222, 184, 177, 184, 225]
[255, 255, 255, 222, 184, 177, 184, 225, 255]
```

```
[False, False, True, True, False, False, True, False]
[False, True, True, False, False, True, False, False]
[True, True, False, False, True, False, False, False]
[True, False, False, True, False, False, False, True]
[False, False, True, False, False, False, True, True]
[False, True, False, False, False, True, True, False]
[True, False, False, False, True, True, False, False]
[False, False, False, True, True, False, False, True]
```

ImageHash

1. Grayscale the images
2. Resize to fx. 9 x 8 pixels
3. Compare adjacent values
4. **Convert to hash**

```
[False, False, True, True, False, False, True, False]
[False, True, True, False, False, True, False, False]
[True, True, False, False, True, False, False, False]
[True, False, False, True, False, False, False, True]
[False, False, True, False, False, False, True, True]
[False, True, False, False, False, True, True, False]
[True, False, False, False, True, True, False, False]
[False, False, False, True, True, False, False, True]
```

```
4c8e3366c275650f
```