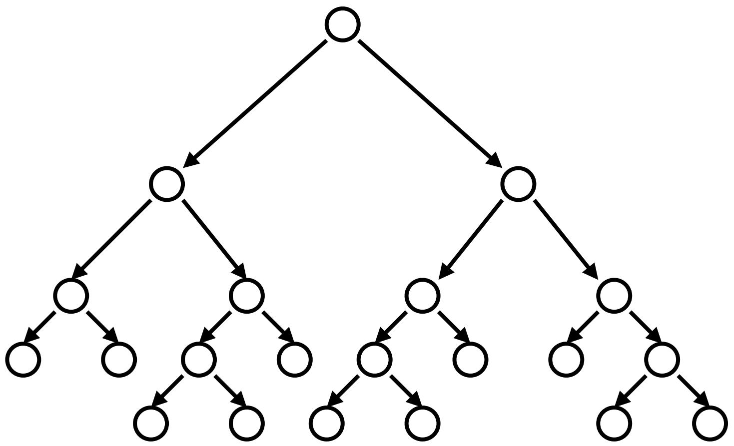
Locality sensitive hashing for approximate NN search

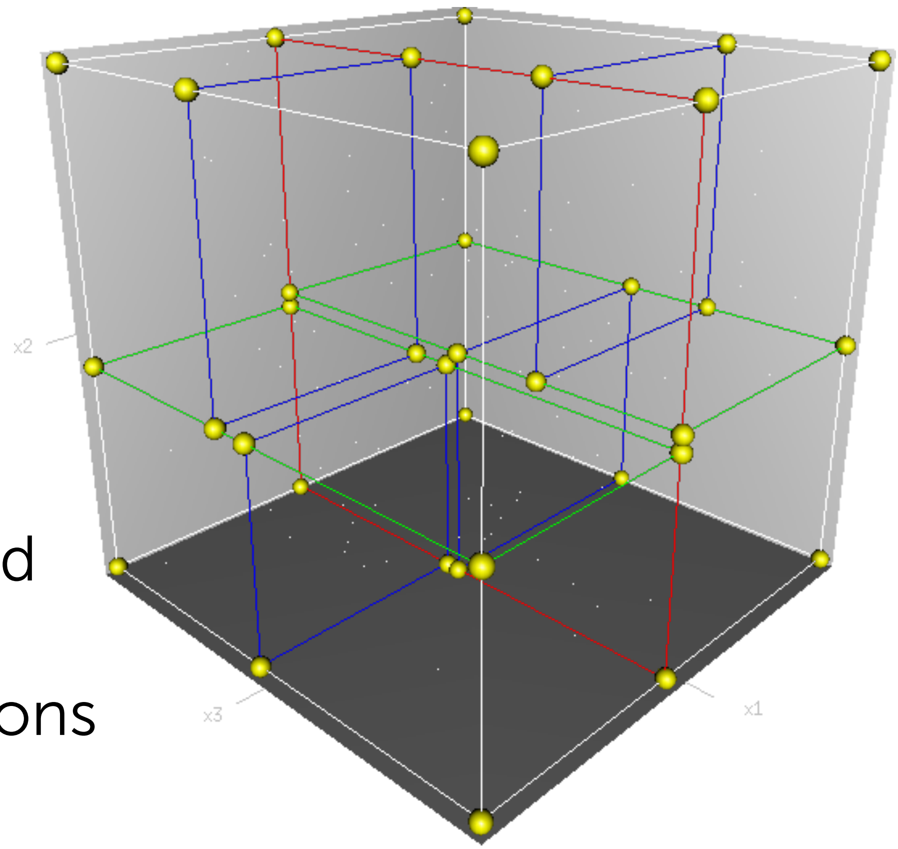
Motivating alternative approaches to approximate NN search

- KD-trees are cool, but…
  - Non-trivial to implement efficiently
  - Problems with high-dimensional data



KD-trees in high dimensions

- Unlikely to have any data points close to query point
- Once "nearby" point is found, the search radius is likely to intersect many hypercubes in at least one dim
- Not many nodes can be pruned
- Can show under some conditions that you visit at least 2^d nodes



Moving away from exact NN search

- Approximate neighbor finding…
  - Don't find exact neighbor, but that's okay for many applications

Out of millions of articles, do we need the closest article or just one that's pretty similar?

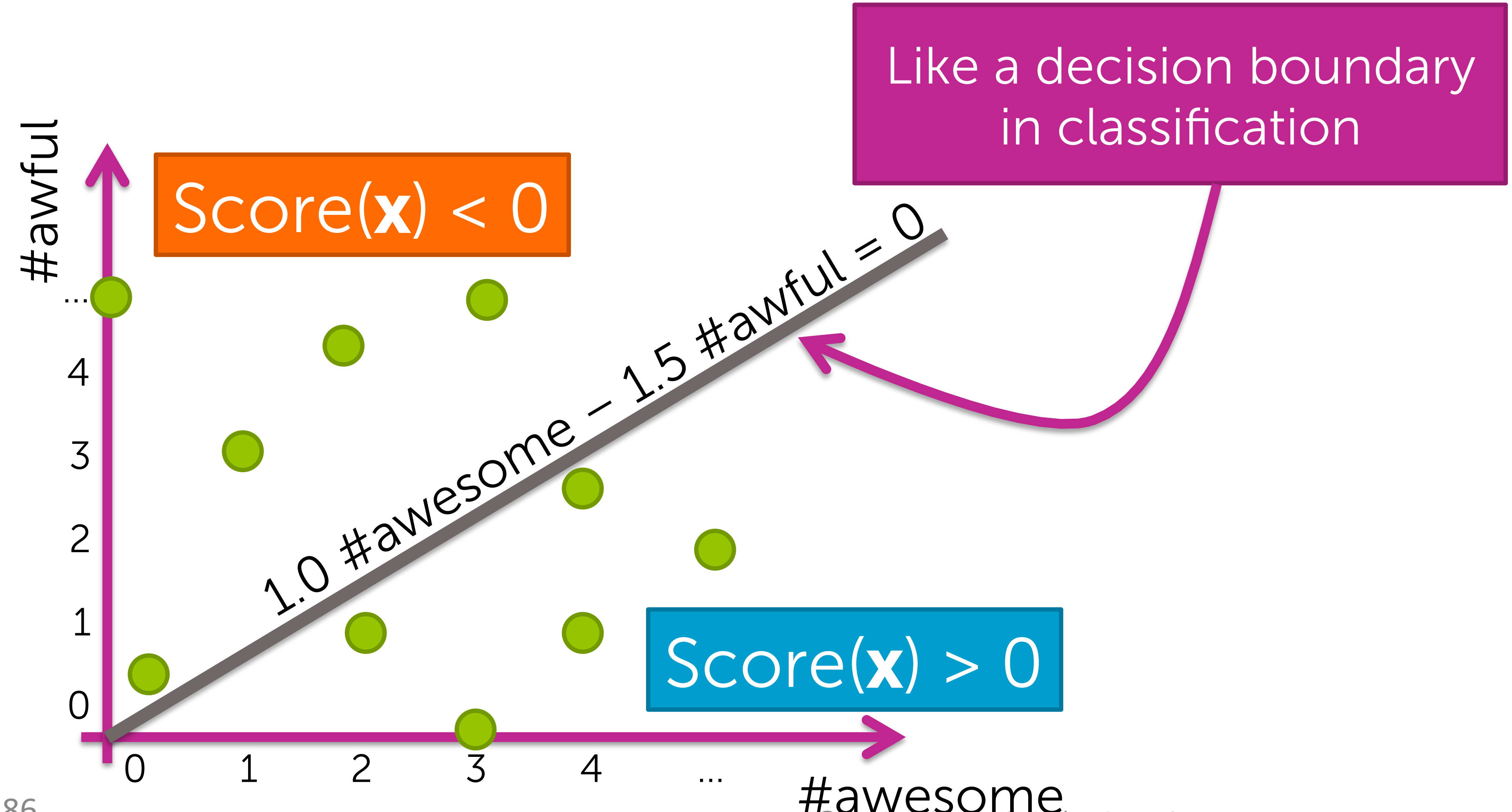
Do we even fully trust our measure of similarity???

- Focus on methods that provide good probabilistic guarantees on approximation

LSH as an alternative to KD-trees

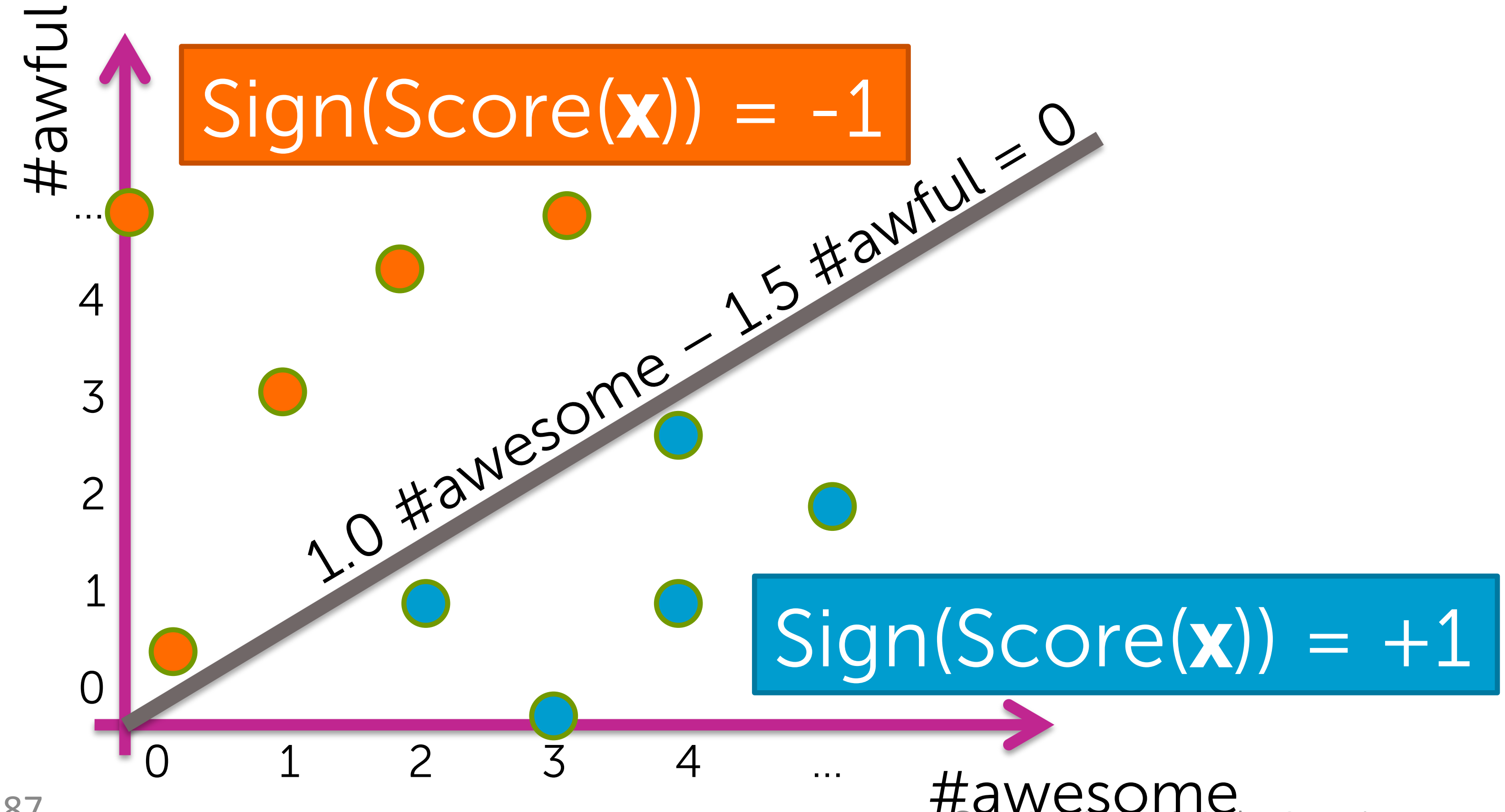
Simple "binning" of data into 2 bins

Score(**x**) = 1.0 #awesome – 1.5 #awful



Simple "binning" of data into 2 bins

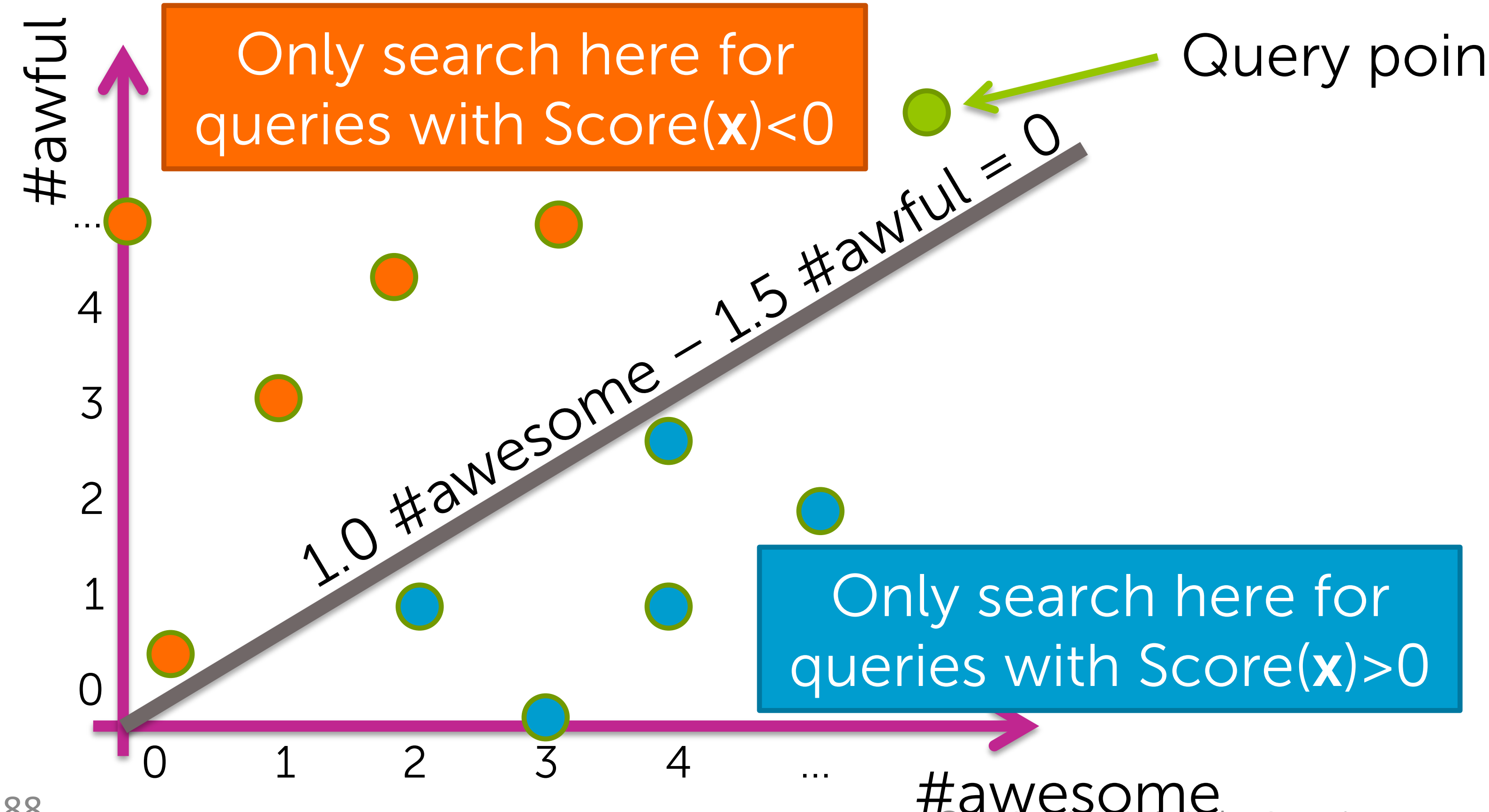
2D Data	Sign(Score)
$\mathbf{x}_1 = [0, 5]$	-1
$\mathbf{x}_2 = [1, 3]$	-1
$\mathbf{x}_3 = [3, 0]$	1
…	…



Using bins for NN search

2D Data	Sign(Score)	Bin index
$\mathbf{x}_1 = [0, 5]$	-1	0
$\mathbf{x}_2 = [1, 3]$	-1	0
$\mathbf{x}_3 = [3, 0]$	1	1
…	…	…

candidate neighbors if Score($\mathbf{x}$)<0



Using score for NN search

2D Data	Sign(Score)	Bin index
$\mathbf{x}_1$ = [0, 5]	-1	0
$\mathbf{x}_2$ = [1, 3]	-1	0
$\mathbf{x}_3$ = [3, 0]	1	1
…	…	…

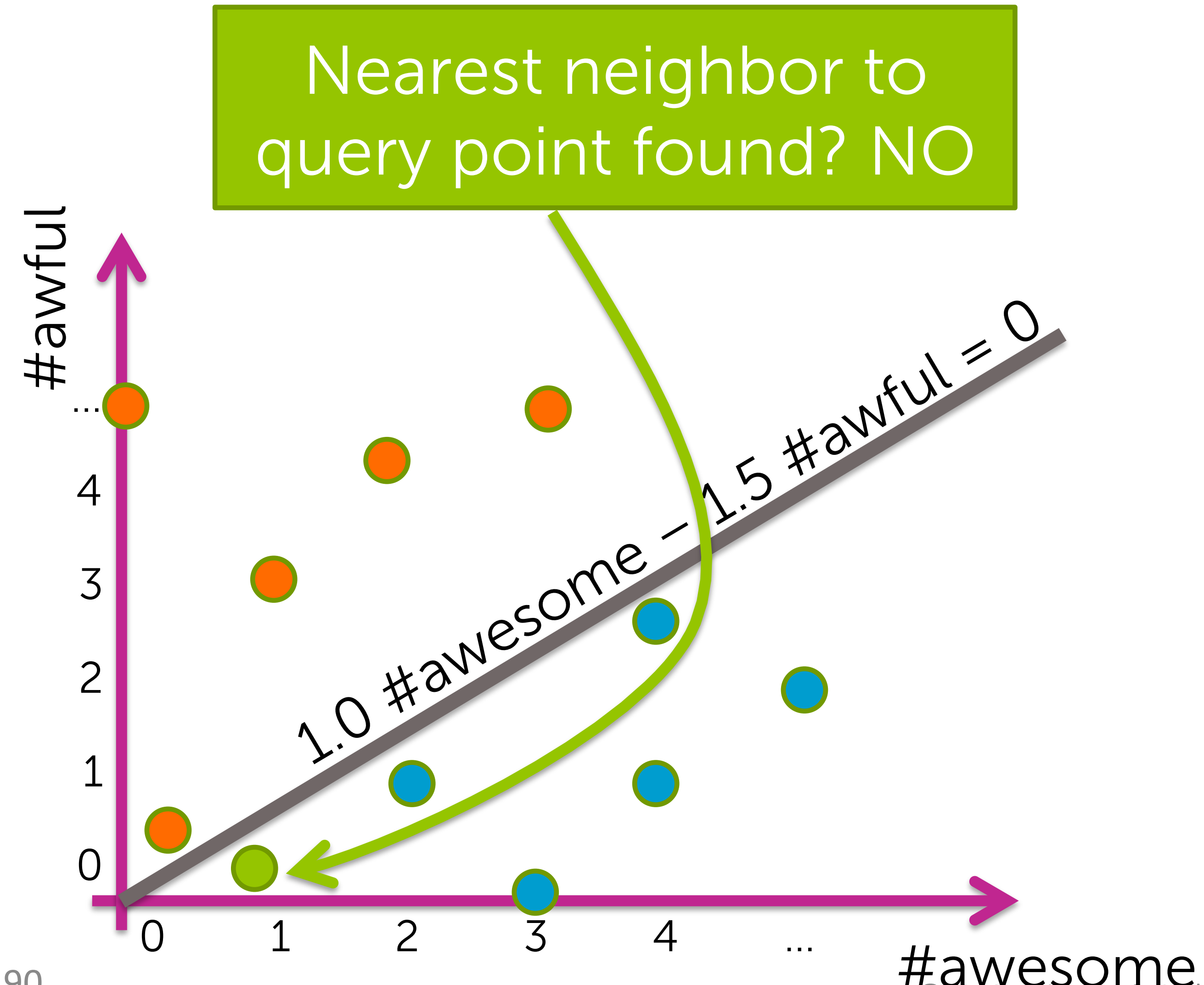
candidate neighbors if Score(x)<0

Bin	0	1
List containing indices of datapoints:	{1,2,4,7,…}	{3,5,6,8,…}

HASH TABLE

search for NN amongst this set

Provides approximate NN



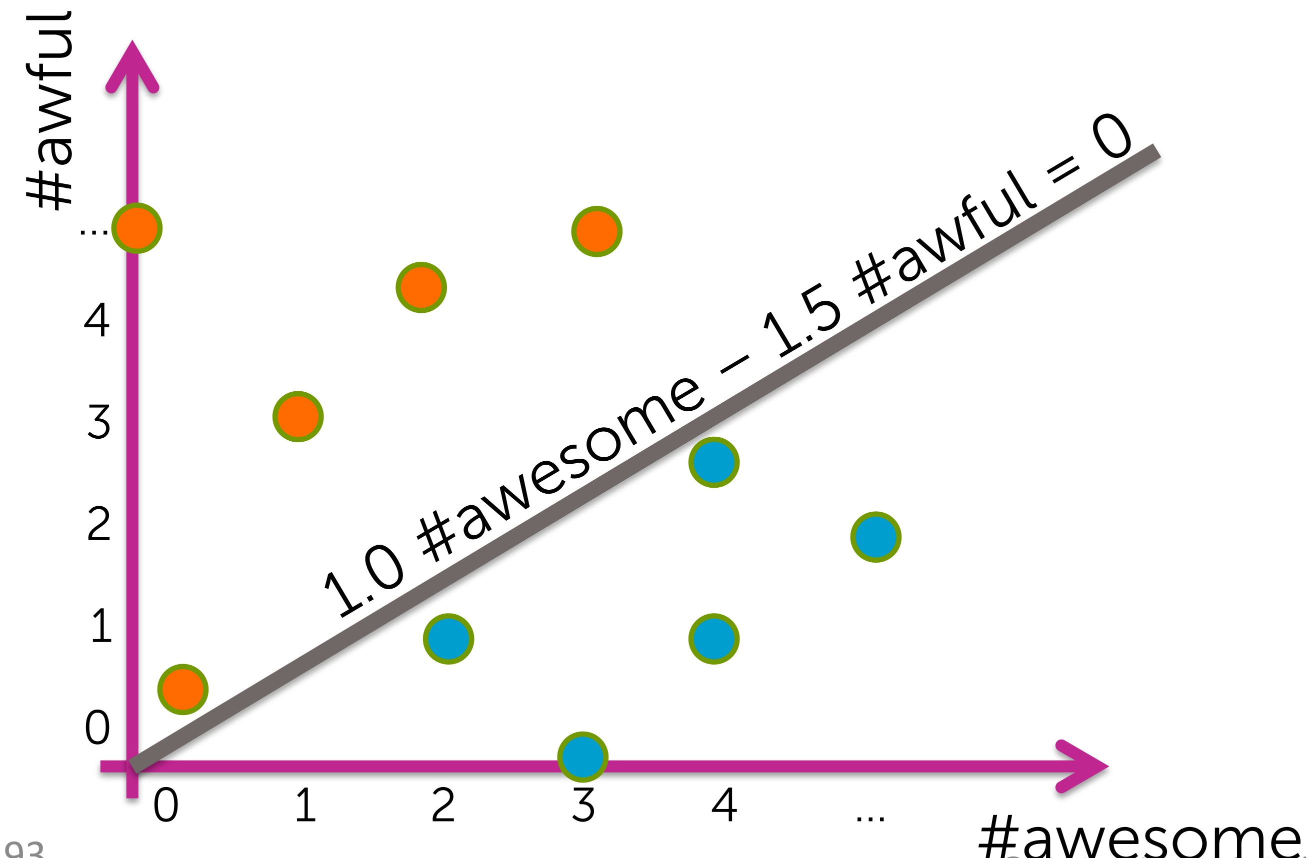
A practical implementation

Three potential issues with simple approach

1. Challenging to find good line
2. Poor quality solution:
   – Points close together get split into separate bins
3. Large computational cost:
   – Bins might contain many points, so still searching over large set for each NN query

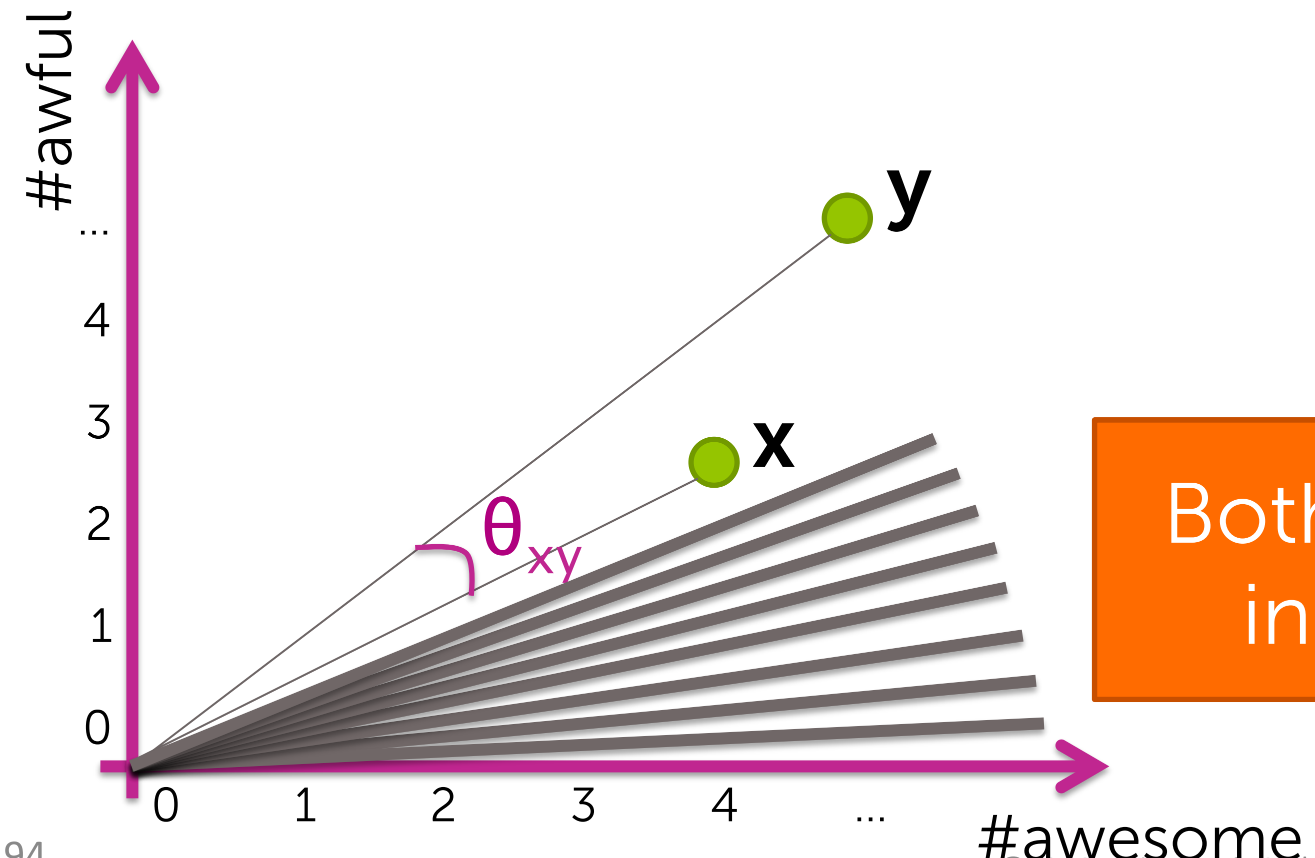
How to define the line?

Crazy idea:
Define line randomly!



How bad can a random line be?

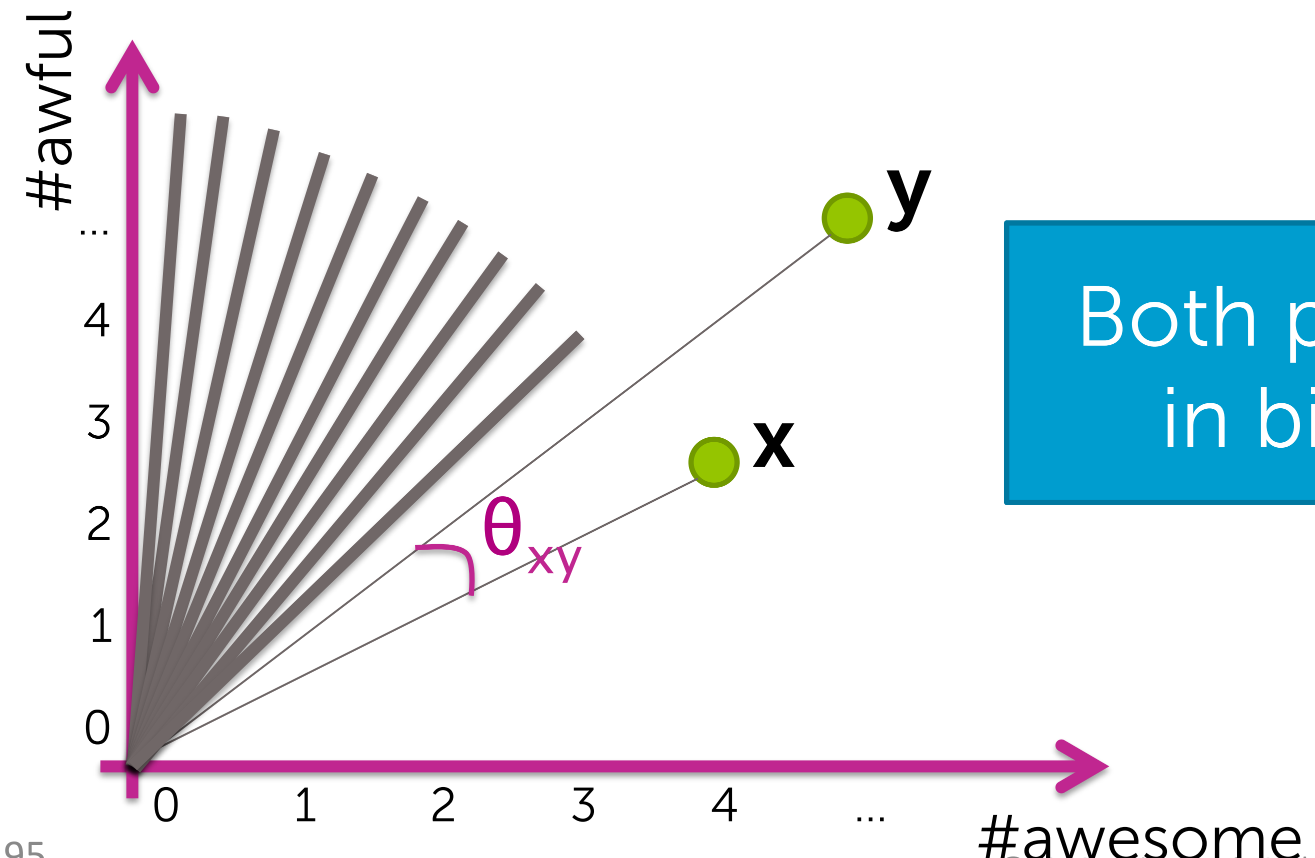
Goal: If **x**,**y** are close (according to cosine similarity), want binned values to be the same.



Both points in bin 0

How bad can a random line be?

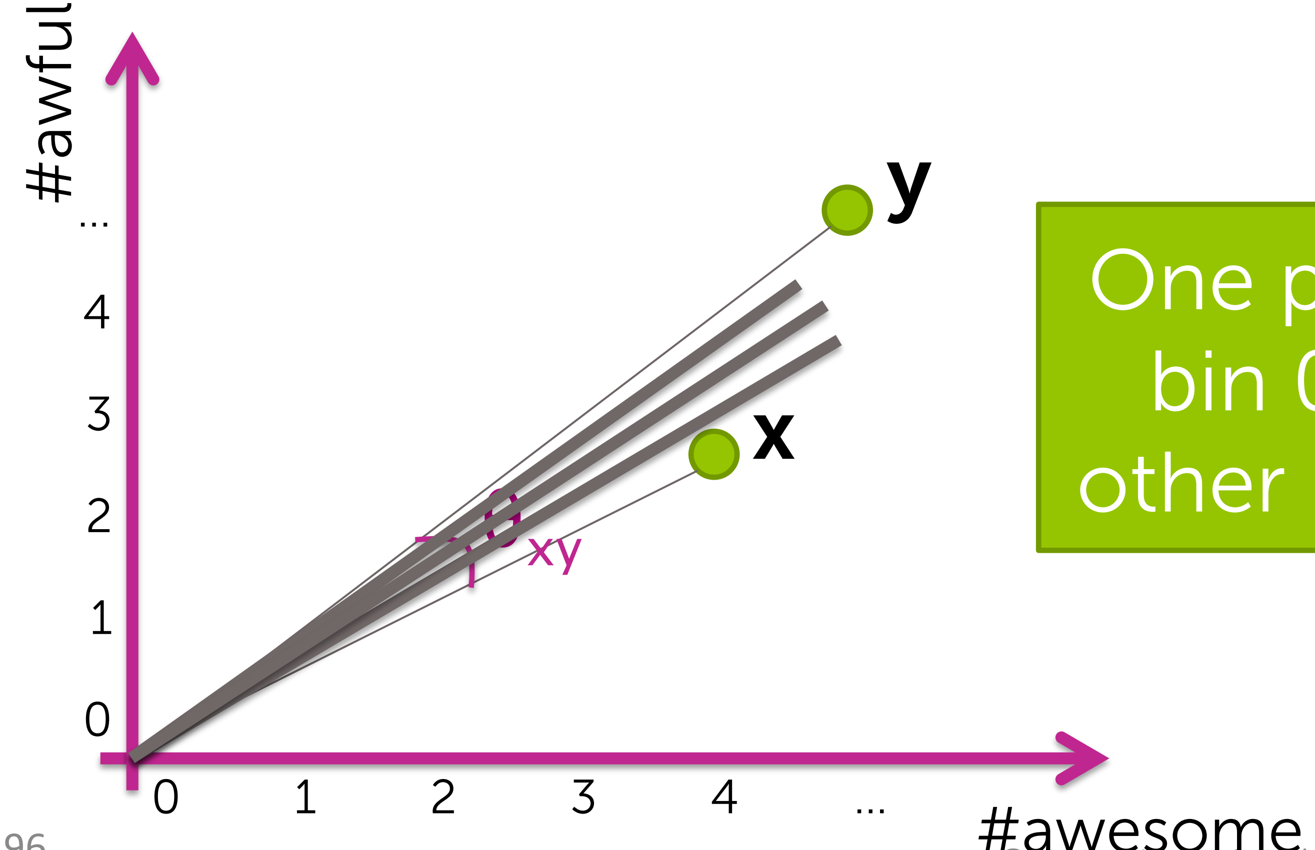
Goal: If **x**,**y** are close (according to cosine similarity), want binned values to be the same.



Both points in bin 1

How bad can a random line be?

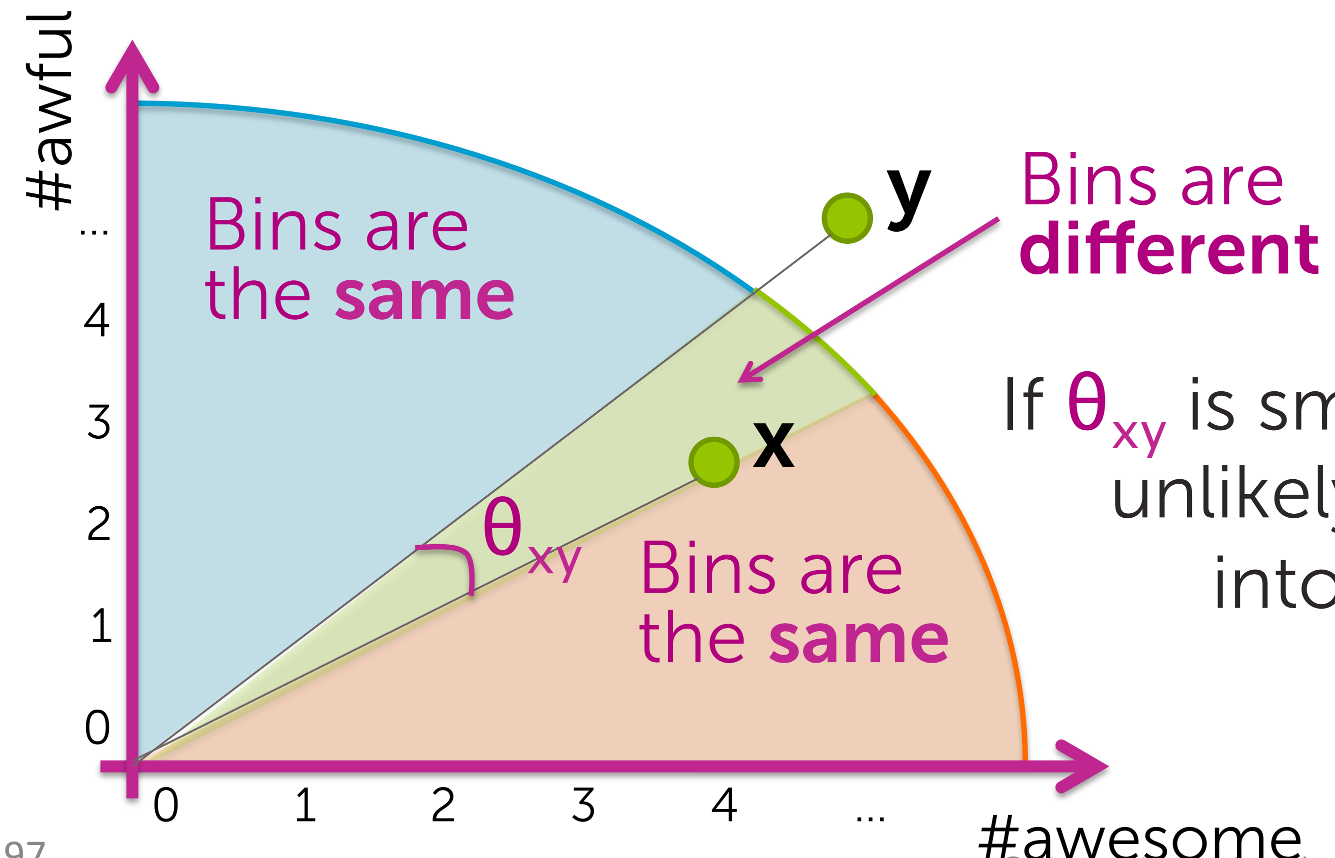
Goal: If **x**,**y** are close (according to cosine similarity), want binned values to be the same.



One point in bin 0 and other in bin 1

How bad can a random line be?

Goal: If **x**,**y** are close (according to cosine similarity), want binned values to be the same.



If θ_{xy} is small (**x**,**y** close), unlikely to be placed into separate bins

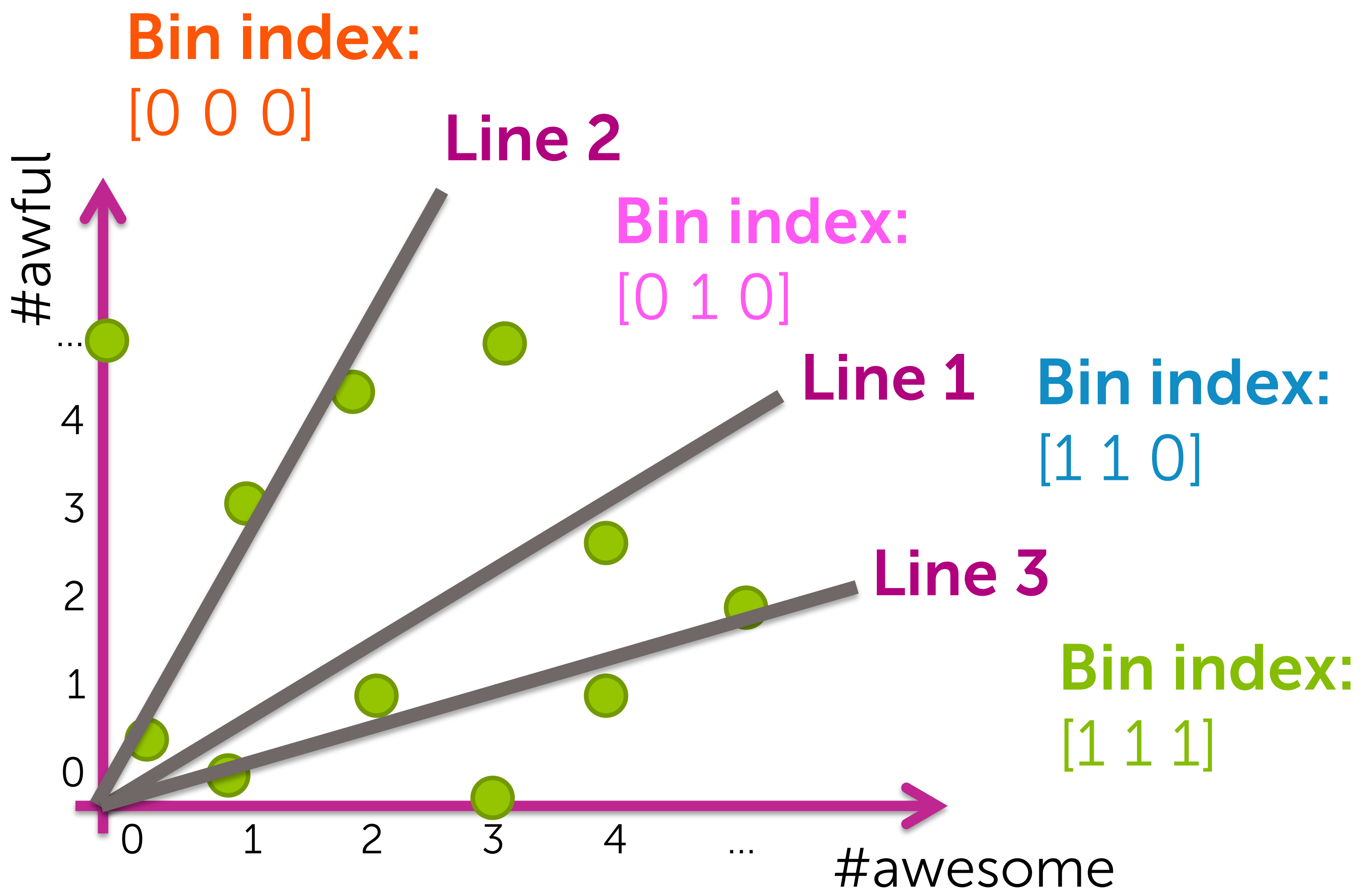
Three potential issues with simple approach

1. Challenging to find good line
2. Poor quality solution:
   – Points close together get split into separate bins
3. Large computational cost:
   – Bins might contain many points, so still searching over large set for each NN query

Bin	0	1
List containing indices of datapoints:	{1,2,4,7,...}	{3,5,6,8,...}

Improving efficiency:
Reducing # points examined per query

Reducing search cost through more bins



Using score for NN search

2D Data	Sign (Score$_1$)	Bin 1 index	Sign (Score$_2$)	Bin 2 index	Sign (Score$_3$)	Bin 3 index
$\mathbf{x}_1$ = [0, 5]	-1	0	-1	0	-1	0
$\mathbf{x}_2$ = [1, 3]	-1	0	-1	0	-1	0
$\mathbf{x}_3$ = [3, 0]	1	1	1	1	1	1
…	…	…	…	…	…	…

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
Data indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

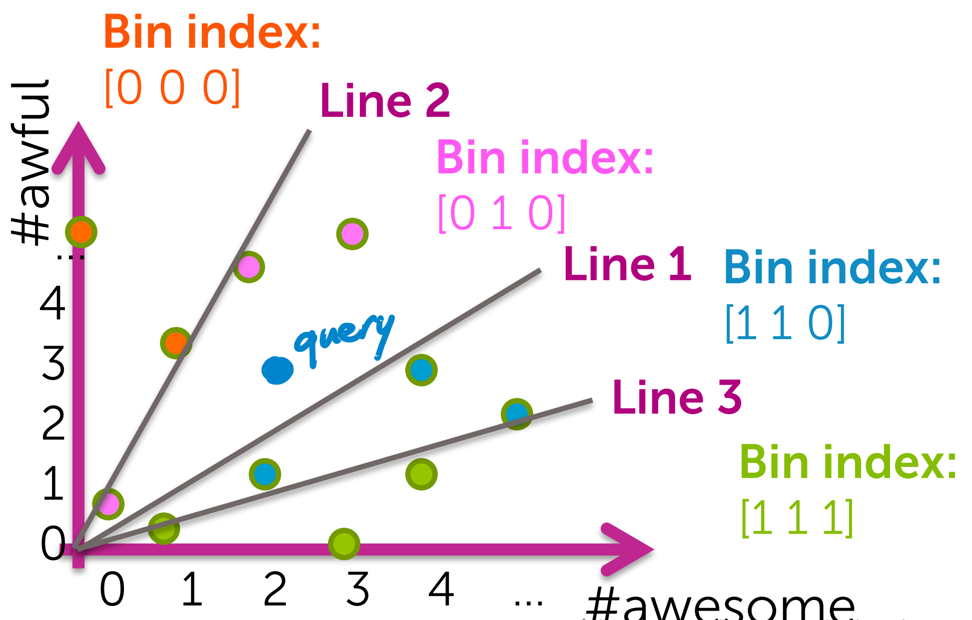
search for NN
amongst this set

Improving search quality by searching neighboring bins

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
Data indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

Query point here, but is NN?

Not necessarily

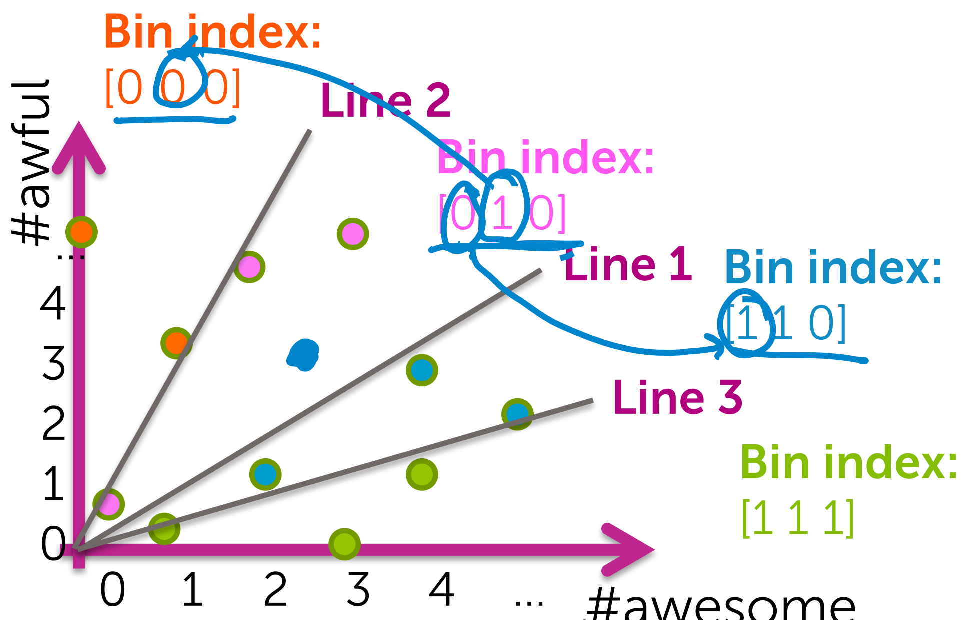


Even worse than before…Each line can split pts.
Sacrificing accuracy for speed

Improving search quality by searching neighboring bins

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
Data indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

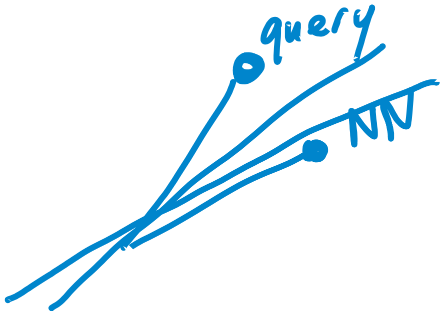
Next closest bins
(flip 1 bit)

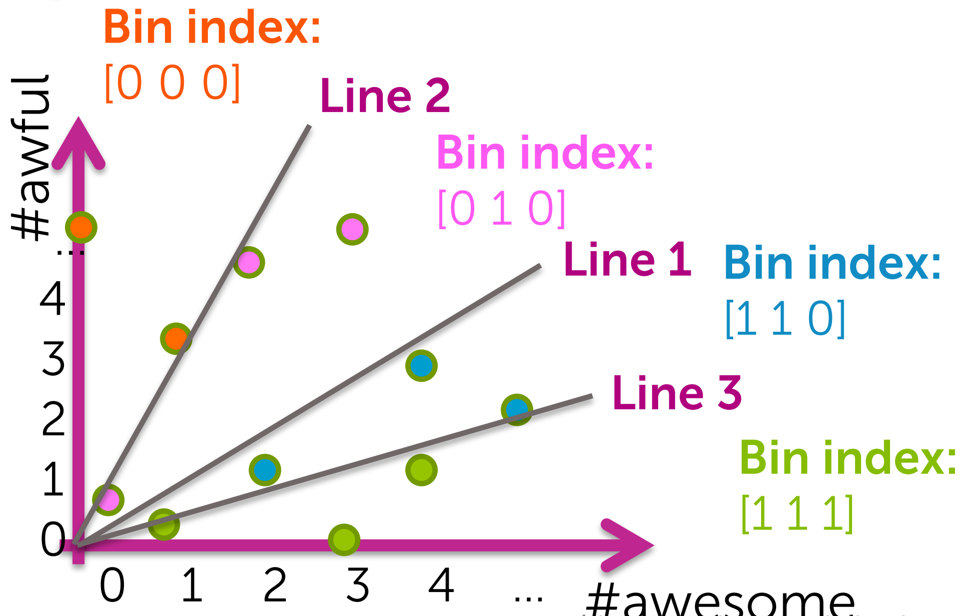


Improving search quality by searching neighboring bins

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
Data indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

Further bin (flip 2 bits)





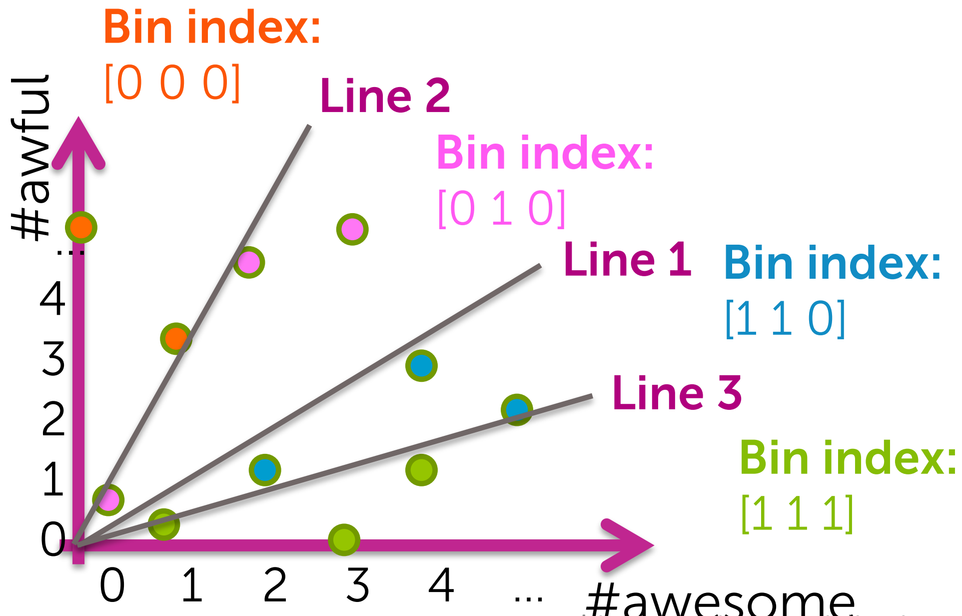
Improving search quality by searching neighboring bins

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
Data indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

Quality of retrieved NN can only improve with searching more bins

Algorithm:

Continue searching until computational budget is reached or quality of NN good enough



LSH recap

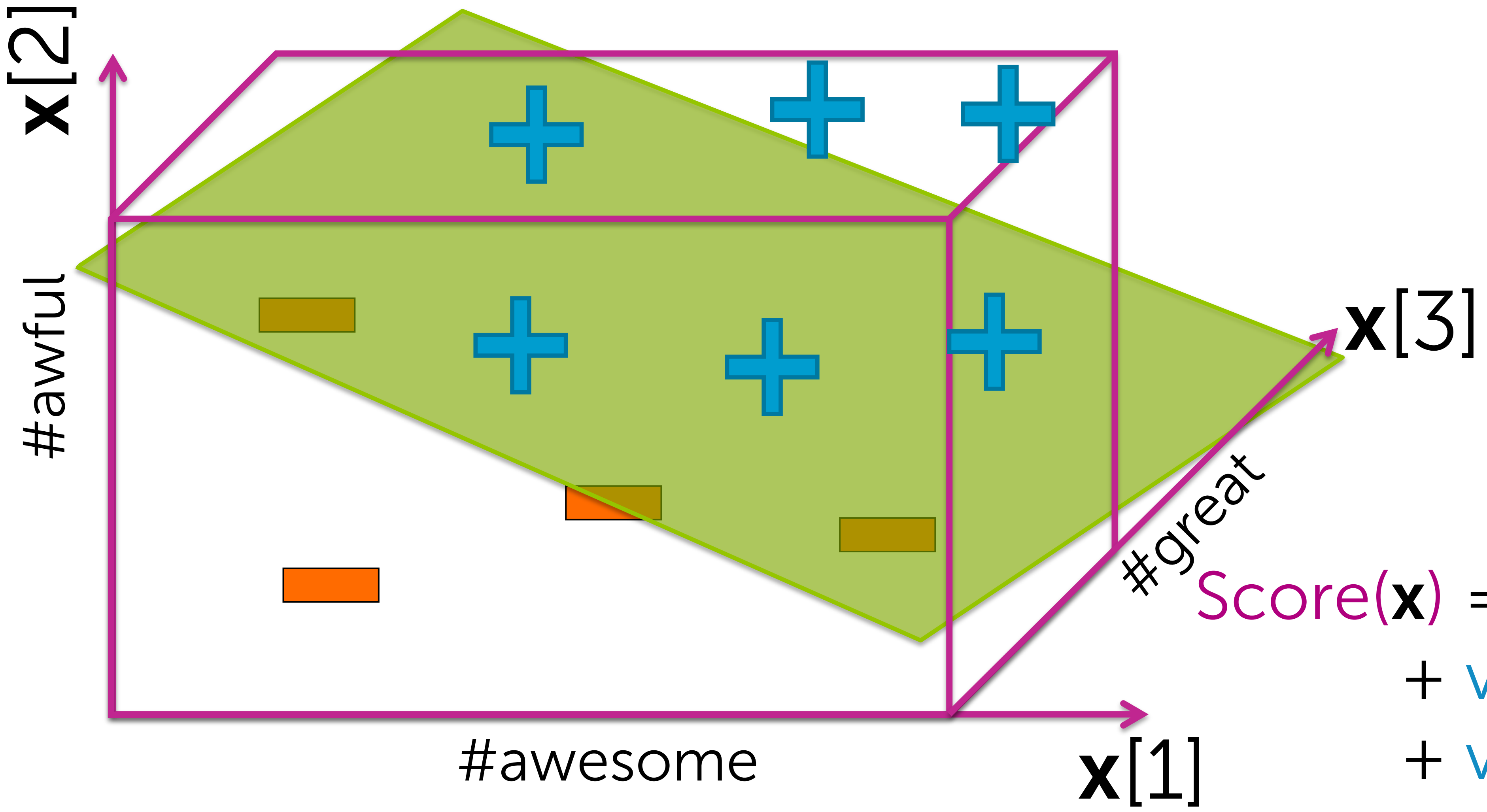
kd-tree competitor data structure

- Draw h random lines
- Compute "score" for each point under each line and translate to binary index
- Use h-bit binary vector per data point as bin index
- Create hash table

- For each query point **x**, search bin(**x**), then neighboring bins until time limit

Moving to higher dimensions d

Draw random *planes*



$$\text{Score}(\mathbf{x}) = v_1\,\#\text{awesome} + v_2\,\#\text{awful} + v_3\,\#\text{great}$$

Cost of binning points in d-dim

$$\text{Score}_i(\mathbf{x}) = v_1^{(i)}\#\text{awesome} + v_2^{(i)}\#\text{awful} + v_3^{(i)}\#\text{great}$$

ith hyperplane

Per data point, need d multiplies to determine bin index per plane

In high-dim, (and some applications) this is often a sparse mult.

One-time cost offset if many queries of fixed dataset

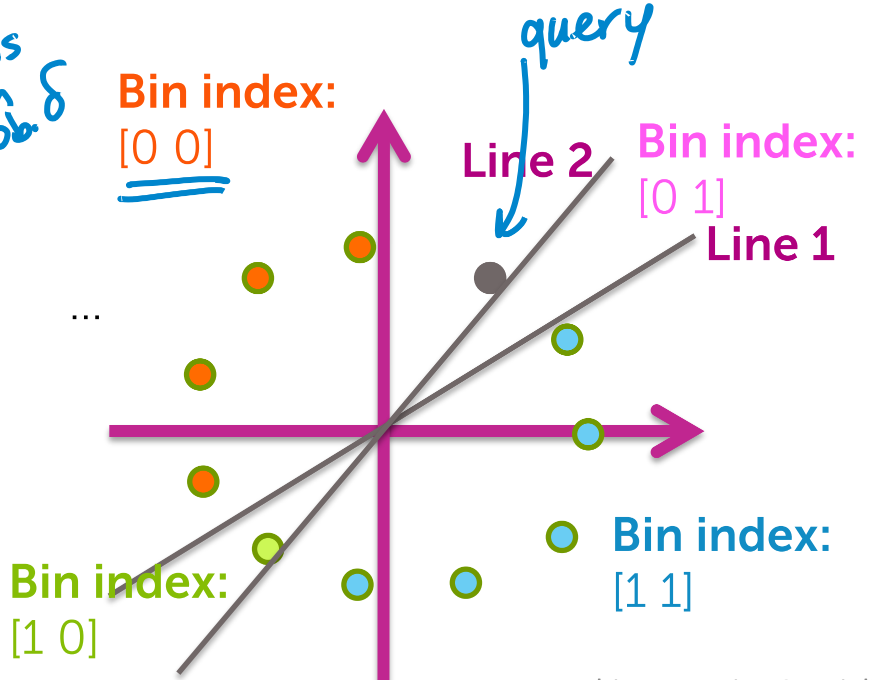
Using multiple tables for even greater efficiency in NN search

If I throw down 2 lines…

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

For simplicity, assume we **search bins 1 bit off** from query

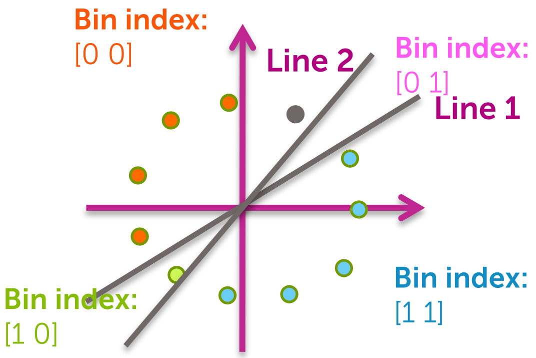
Let δ be the probability of a line falling between points θ apart



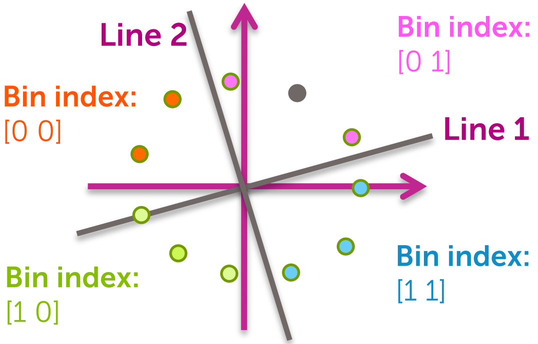
Search 3 bins and do not find NN with probability δ^2

What if I repeat the 2-line binning?

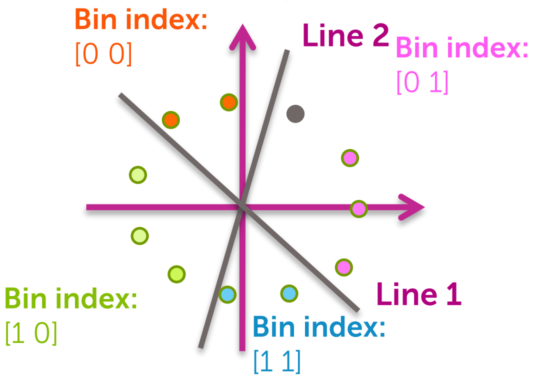
Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3



Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3



Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

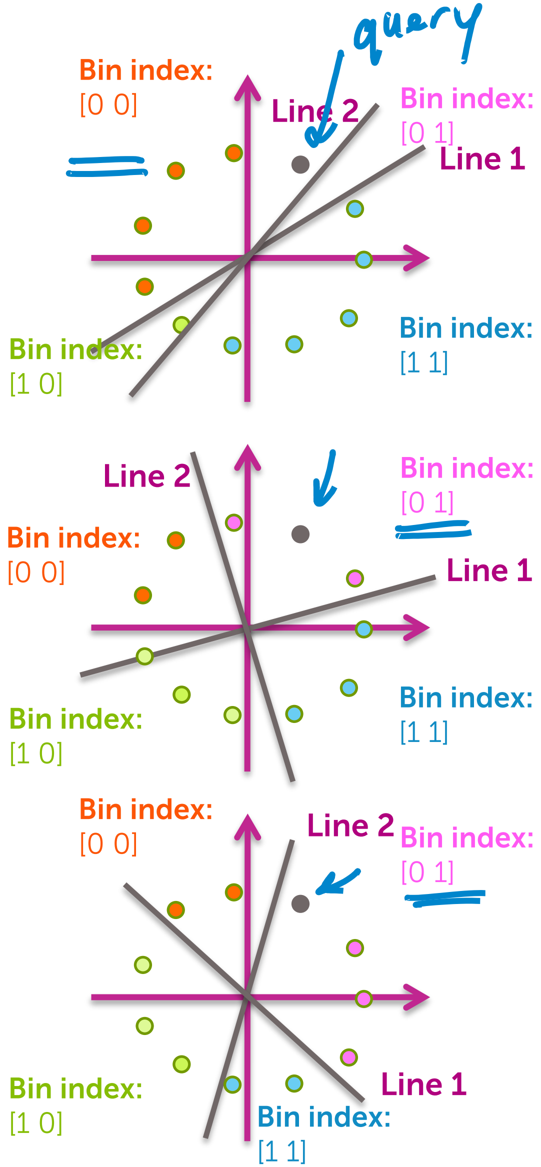


What if I repeat the 2-line binning?

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3



Now, search only query bin **per table**

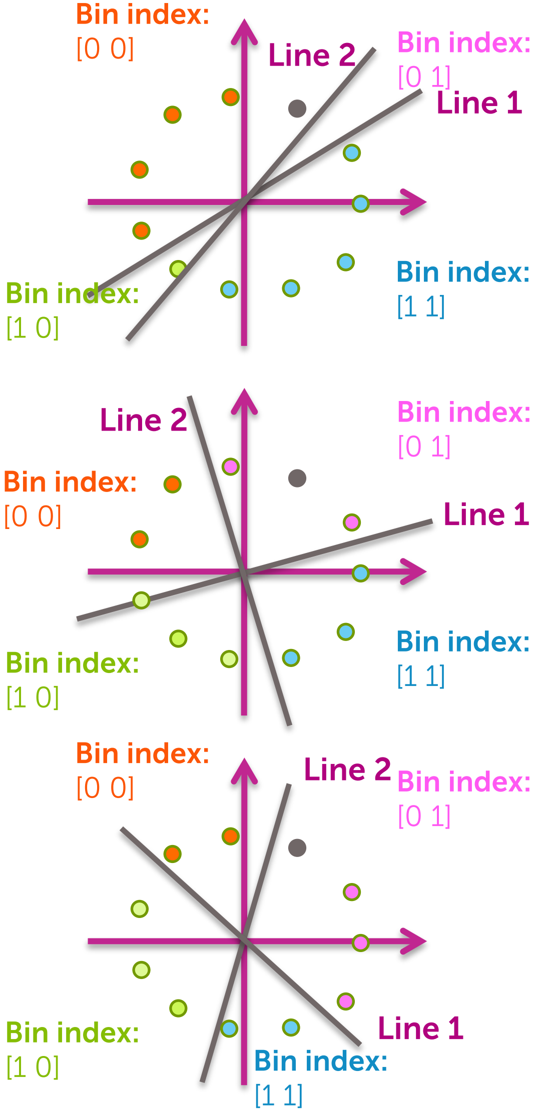
Still searching 3 bins, but what is chance of not finding NN?

What if I repeat the 2-line binning?

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

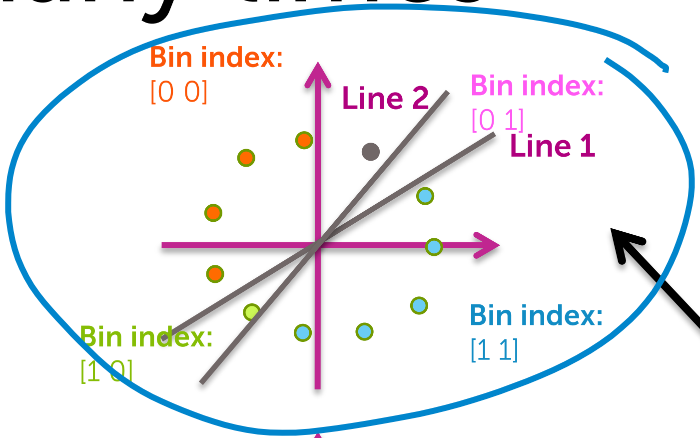
Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3



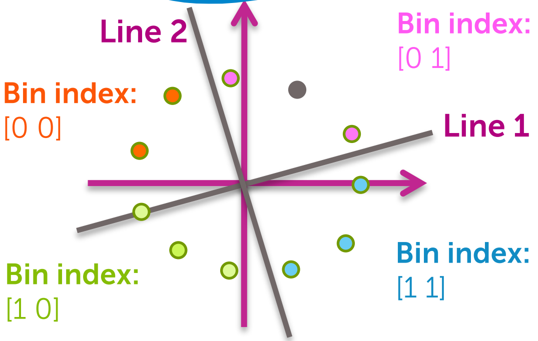
What is chance that query pt and NN are split in **all tables**?

Probability of splitting neighboring points many times

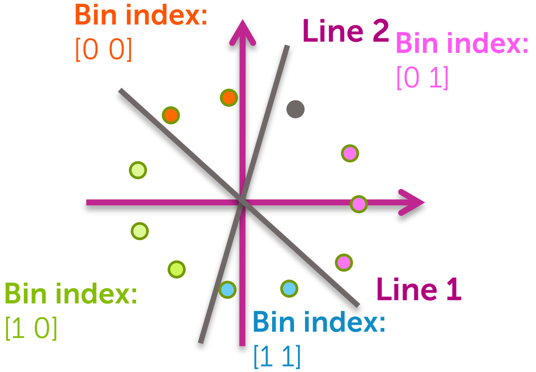
Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3



Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3



Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3



Probability NN is in different bin:

$$\text{Prob} = 1-\Pr(\text{same bin}) = 1-(1-\delta)^2 = 2\delta - \delta^2$$

$1-\delta$ = prob. that 1 line does not split query + NN

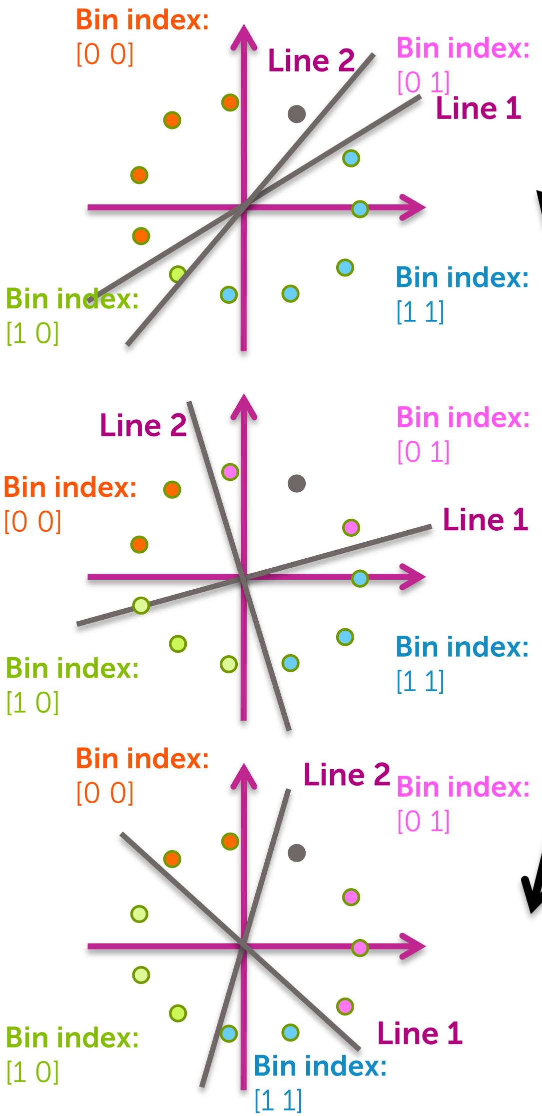
$(1-\delta)^2$ = prob. that 2 lines don't split pts

Probability of splitting neighboring points many times

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3



Probability NN is in different bin in **all 3 tables**:

Prob = $(2\delta - \delta^2)^3$

# of hash tables

Comparing approaches for 2-bit tables

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

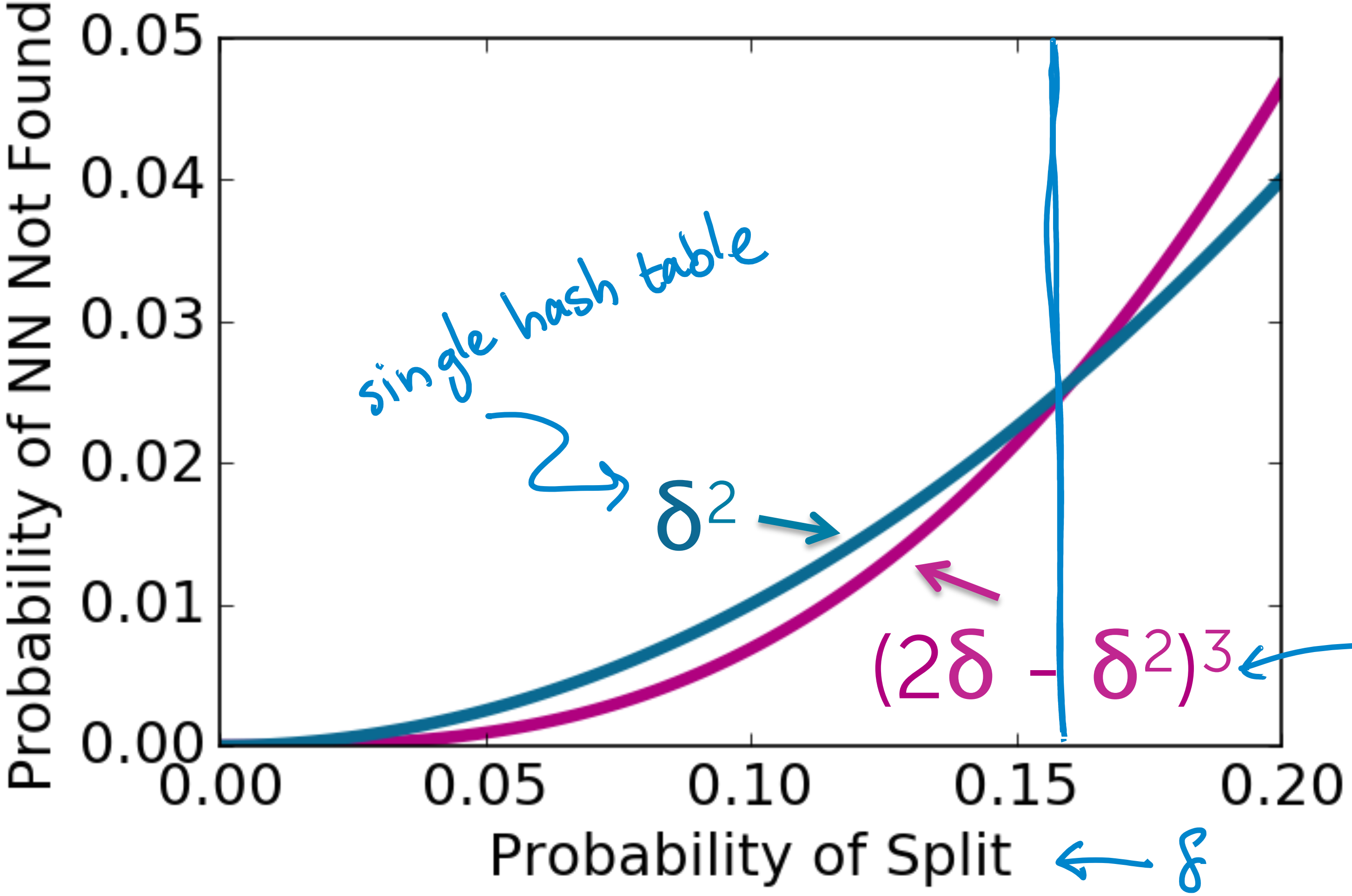
Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

# bins searched	prob. of no NN
3	δ^2
3	$(2\delta - \delta^2)^3$

Comparing probabilities



$\leftarrow \delta$

- throw down 2 lines
- searching 3 bins

multiple hash table

If I throw down h lines…

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Still assume we **search bins 1 bit off** from query

Prob. of being > 1 bit away

$$= 1\text{-}\Pr(\text{same bin})\text{-}\Pr(1 \text{ bin away})$$

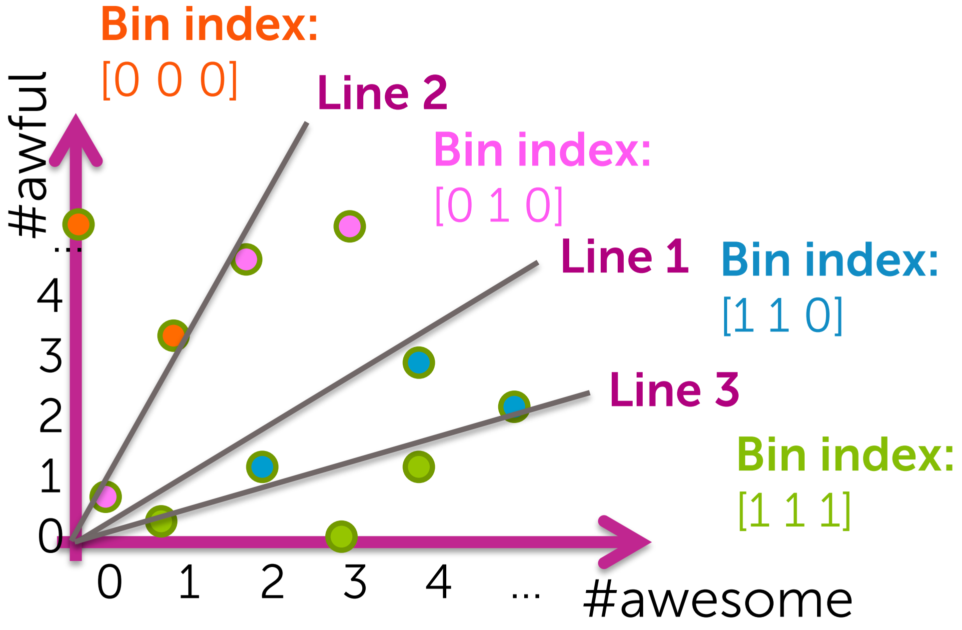
$$= 1\text{-}\Pr(\text{no split lines})\text{-}\Pr(1 \text{ split line})$$

$$= 1-\underbrace{(1-\delta)^h}-h\delta(1-\delta)^{h-1}$$

Prob. of 1 line splitting

prob. of h-1 lines not splitting

Prob of no split is 1-δ
+ we have h lines
⇒ prob. none of h lines split is $(1-\delta)^h$



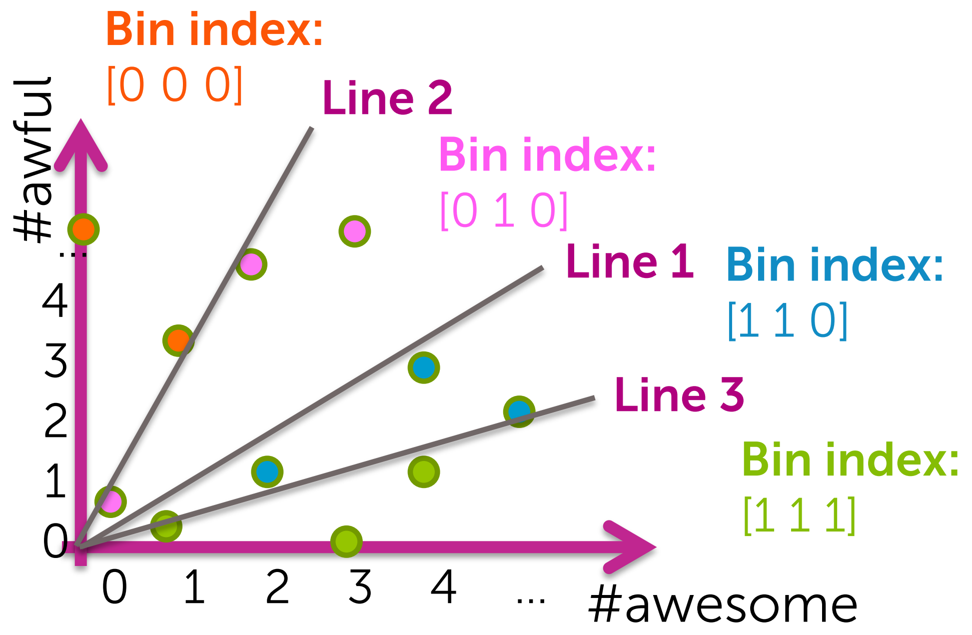
If I throw down h lines…

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Still assume we **search bins 1 bit off** from query

Prob. of being > 1 bit away

$$= 1\text{-Pr(same bin)-Pr(1 bin away)}$$
$$= 1\text{-Pr(no split lines)-Pr(1 split line)}$$
$$= 1-(1-\delta)^h - h\delta(1-\delta)^{h-1}$$



Search h+1 bins and do not find NN with probability $1-(1-\delta)^h - h\delta(1-\delta)^{h-1}$

Probability of splitting neighboring points many times

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Probability NN is in different bin in **all h+1 tables**

$$= (1\text{-}\Pr(\text{same bin}))^{h+1}$$
$$= (1\text{-}\Pr(\text{no split line}))^{h+1}$$
$$= (1-\underbrace{(1-\delta)^h}_{\text{prob. of no split from any of h lines thrown down}})^{h+1}$$

holds for all hash tables

Comparing approaches for h-bit tables

throw down h lines

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

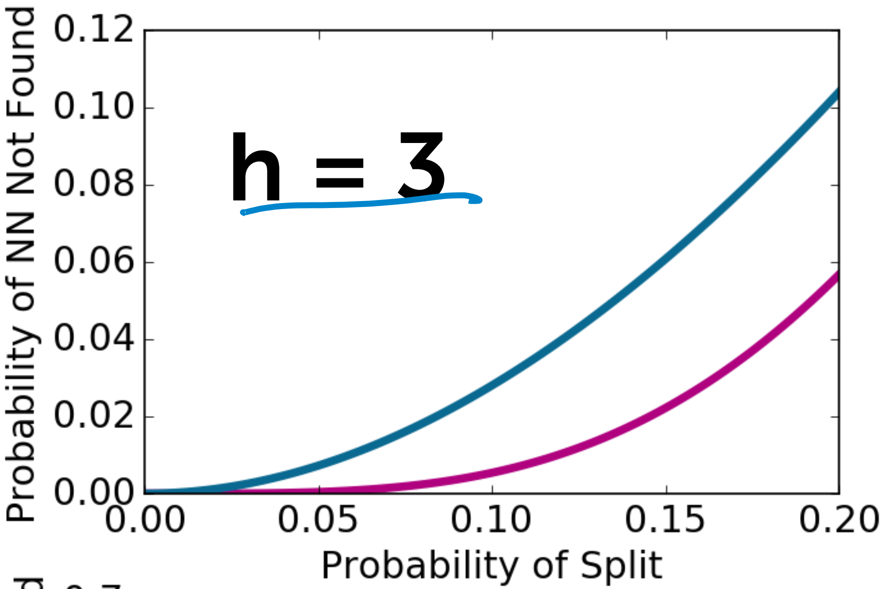
Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3

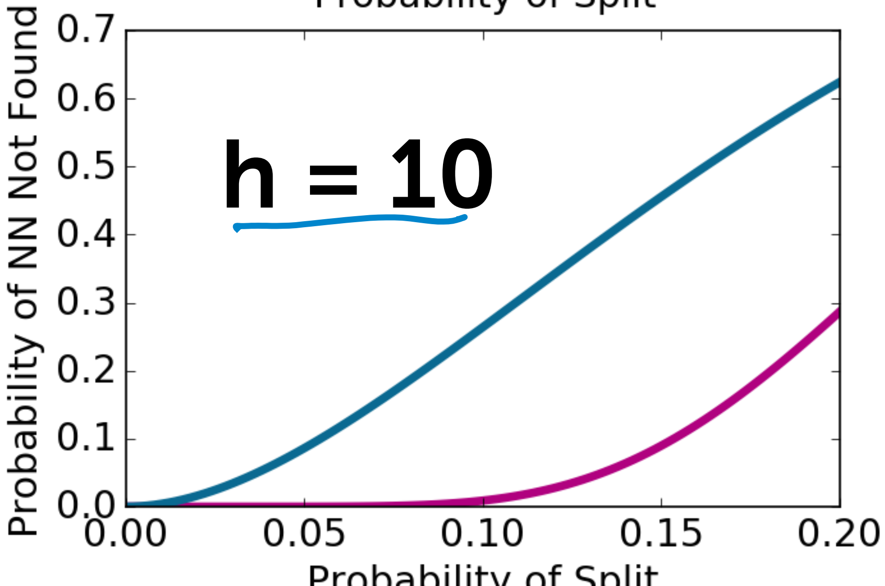
Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

# bins searched	prob. of no NN
$h+1$	$1-(1-\delta)^h-h\delta(1-\delta)^{h-1}$
$h+1$	$(1-(1-\delta)^h)^{h+1}$

Comparing probabilities





one hash table

$$1-(1-\delta)^h - h\delta(1-\delta)^{h-1}$$

$$(1-(1-\delta)^h)^{h+1}$$

multiple hash table

Fix #bits and increase depth

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Probability NN is in different bin in **all tables** falls off exponentially fast

\# hash tables

Prob

$$= (1-\Pr(\text{same bin}))^{m}$$

$$= (1-\Pr(\text{no split line}))^{m}$$

$$= (1-(1-\delta)^{h})^{m}$$

\# of lines (bits)

Fix #bits and increase depth

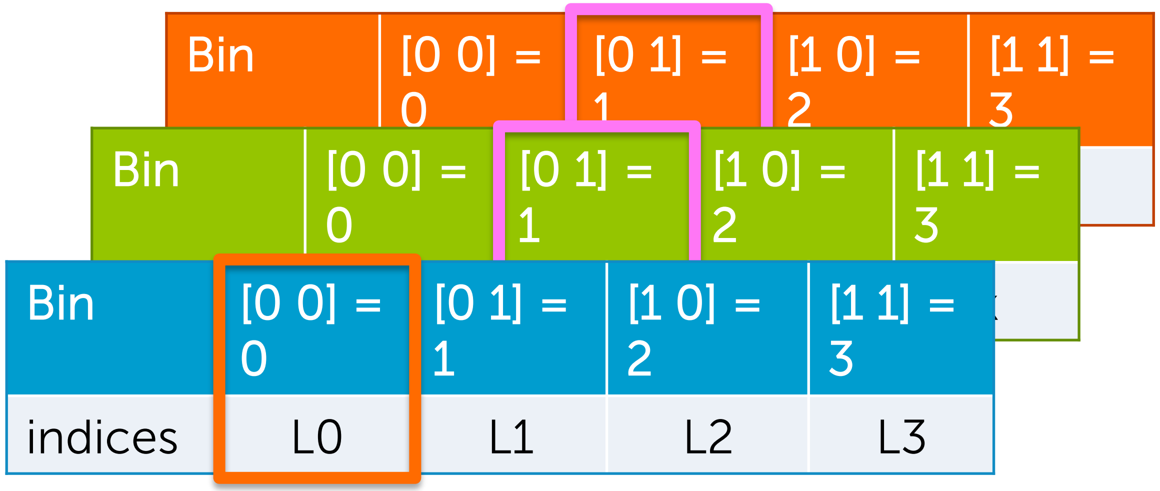
Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
indices	L0	L1	L2	L3	L4	L5	L6	L7

Typically higher probability of finding NN than searching m bins in 1 table

Summary of LSH approaches

Bin	[0 0] = 0	[0 1] = 1	[1 0] = 2	[1 1] = 3
indices	L0	L1	L2	L3

Cost of binning points is **lower**, but likely need to search **more bins** per query



Cost of binning points is **higher**, but likely need to search **fewer bins** per query

Summary for retrieval using nearest neighbors, KD-trees, and locality sensitive hashing