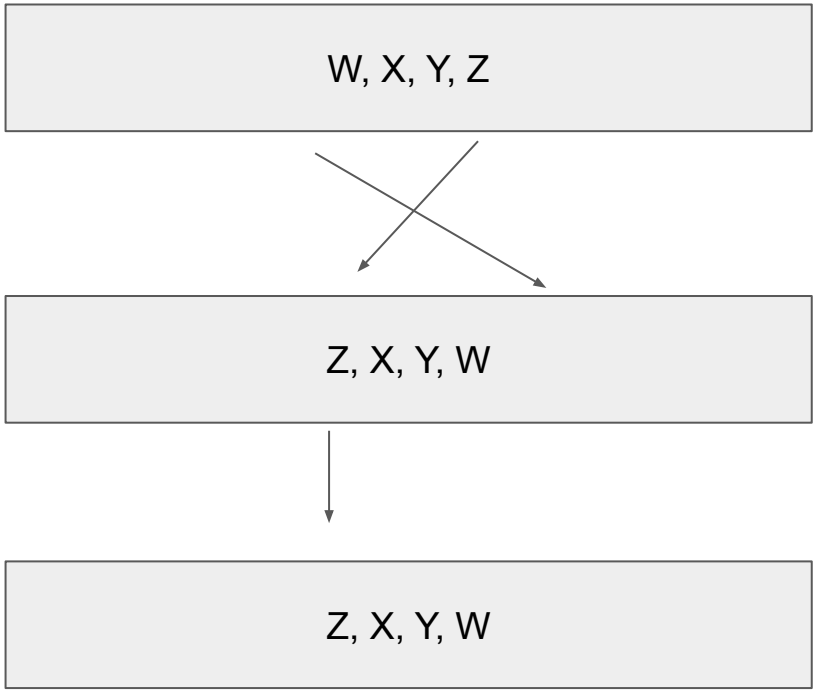
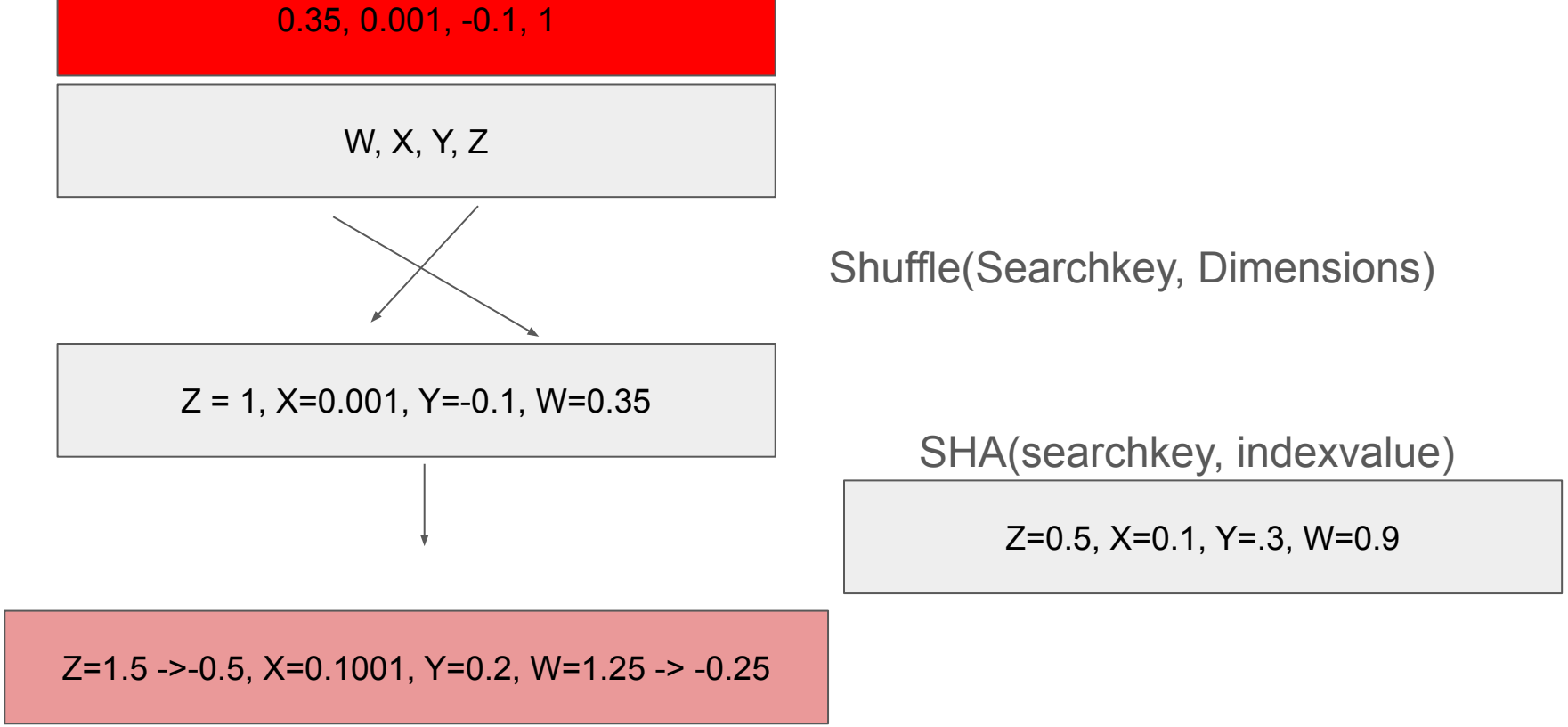
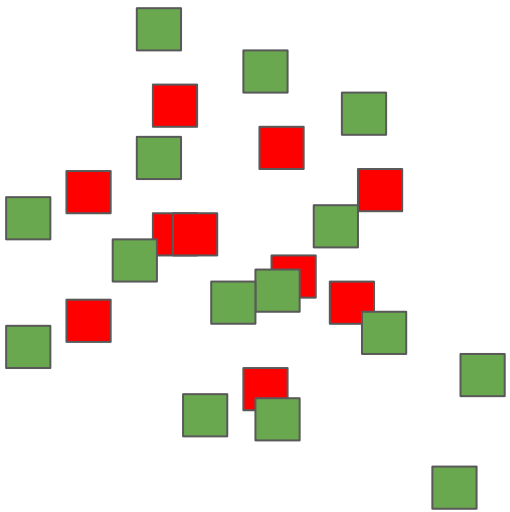
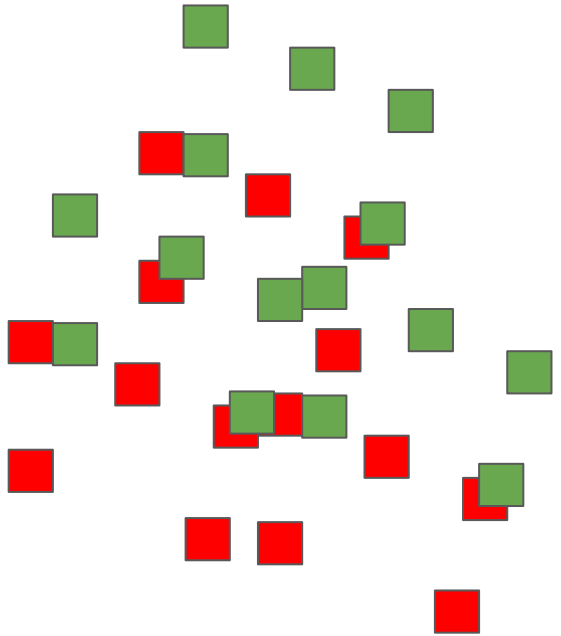


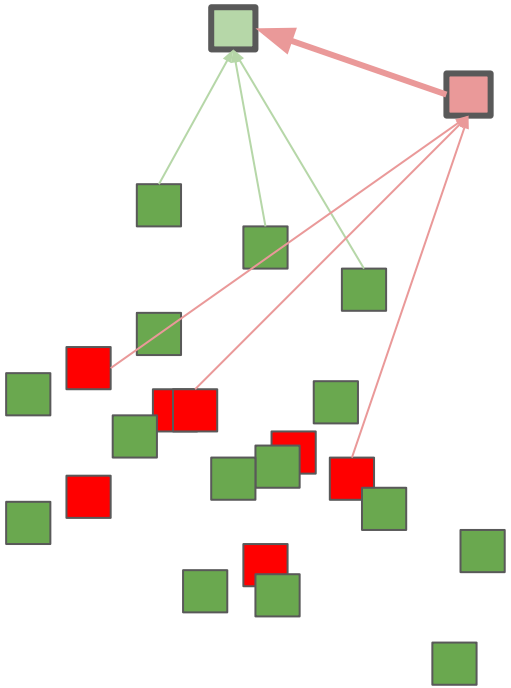
Shuffle(Searchkey, Dimensions)

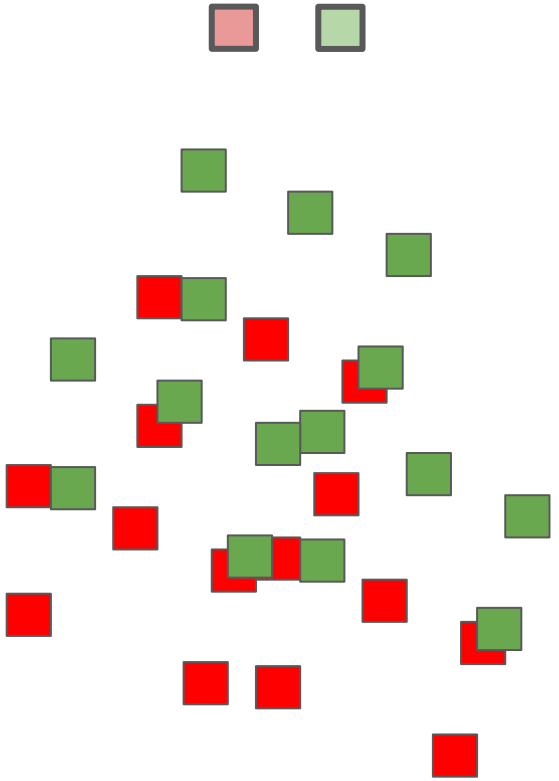
mod(Z + (rand(0,1))= New Z

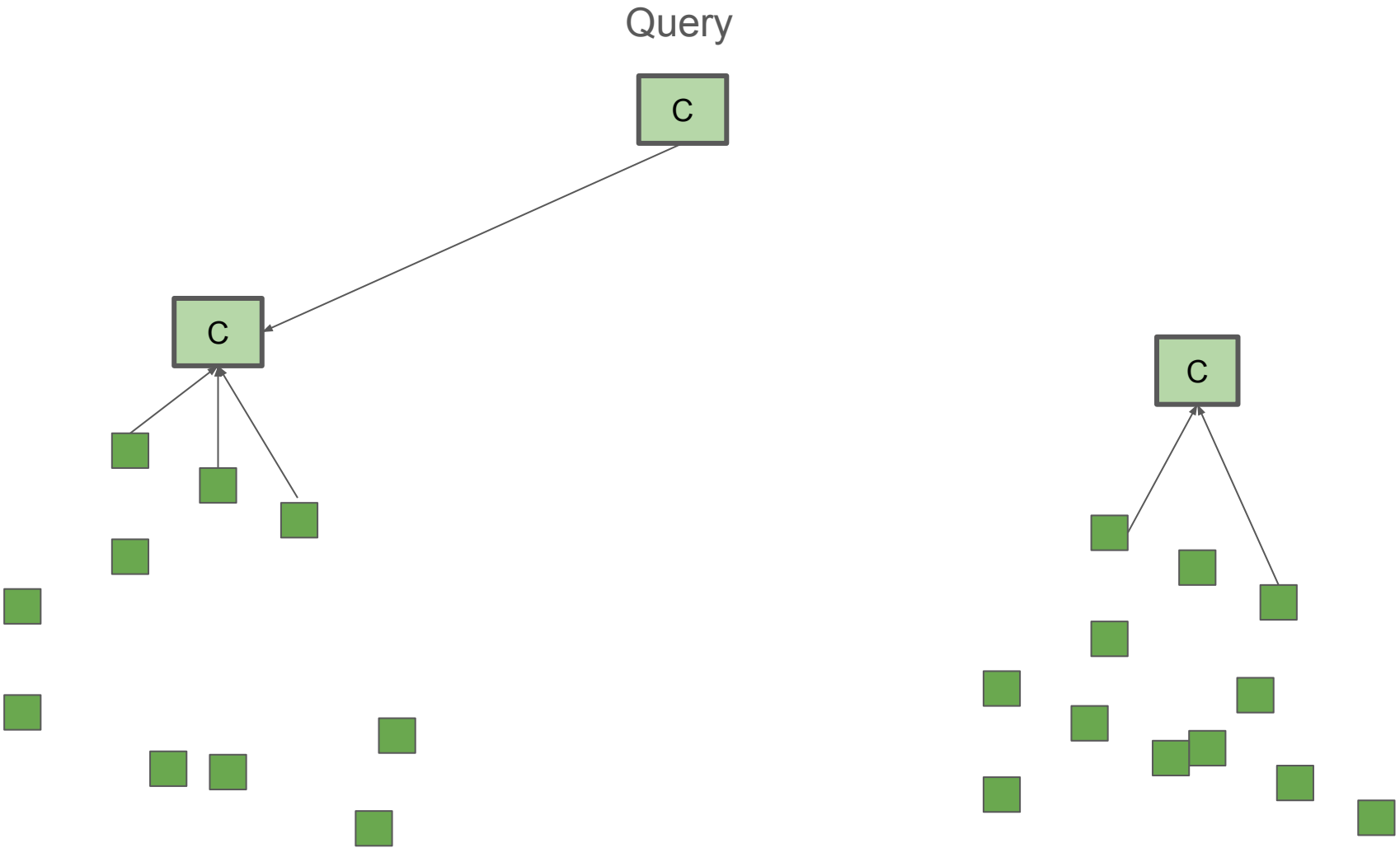

0.35, 0.001, -0.1, 1
W, X, Y, Z
Shuffle(Searchkey, Dimensions)
Z = 1, X=0.001, Y=-0.1, W=0.35
SHA(searchkey, indexvalue)
Z=0.5, X=0.1, Y=.3, W=0.9
Z=1.5 ->-0.5, X=0.1001, Y=0.2, W=1.25 -> -0.25

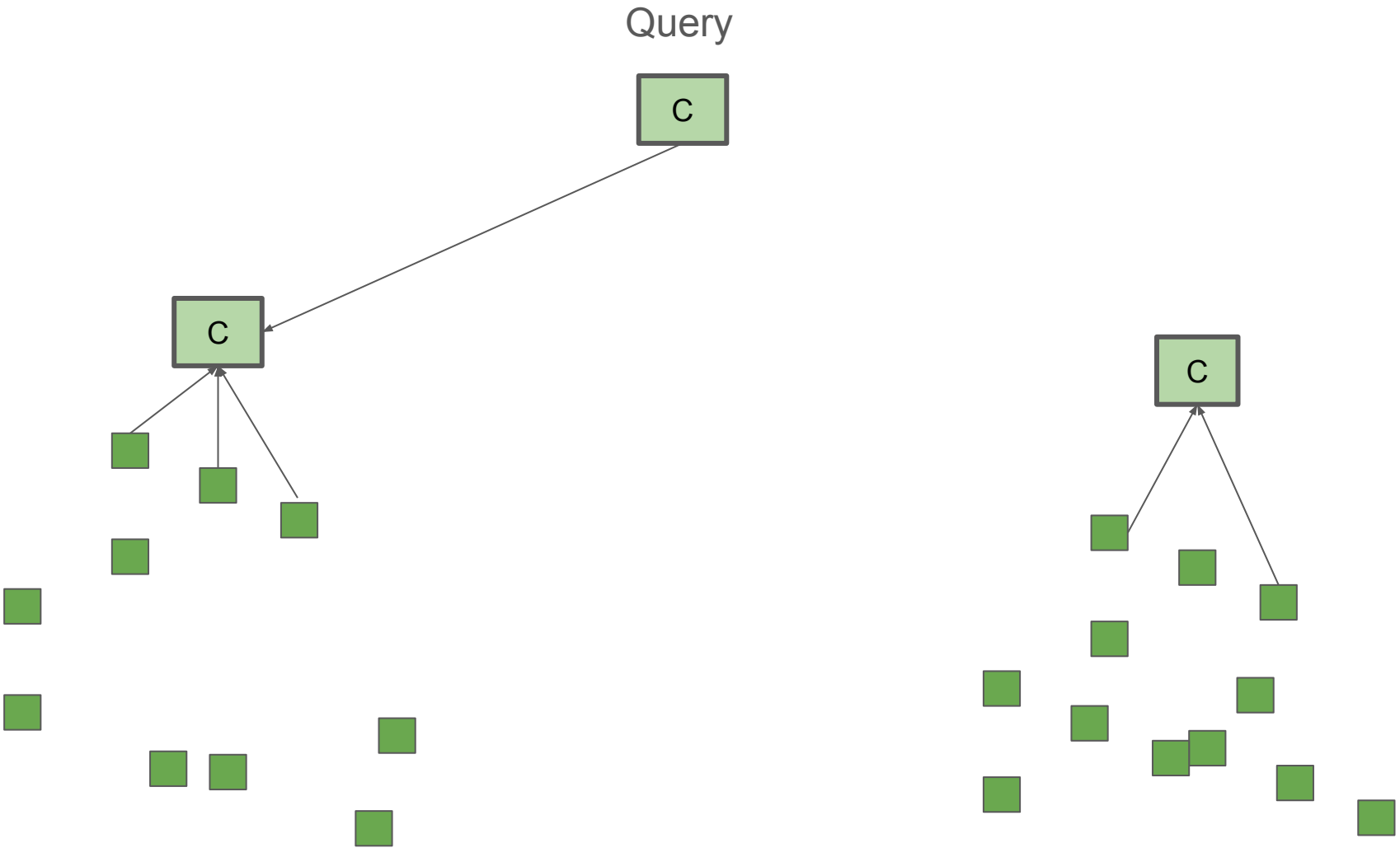




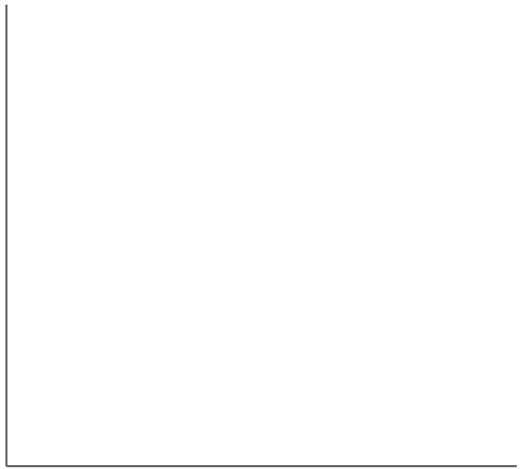





Query
C
C
C

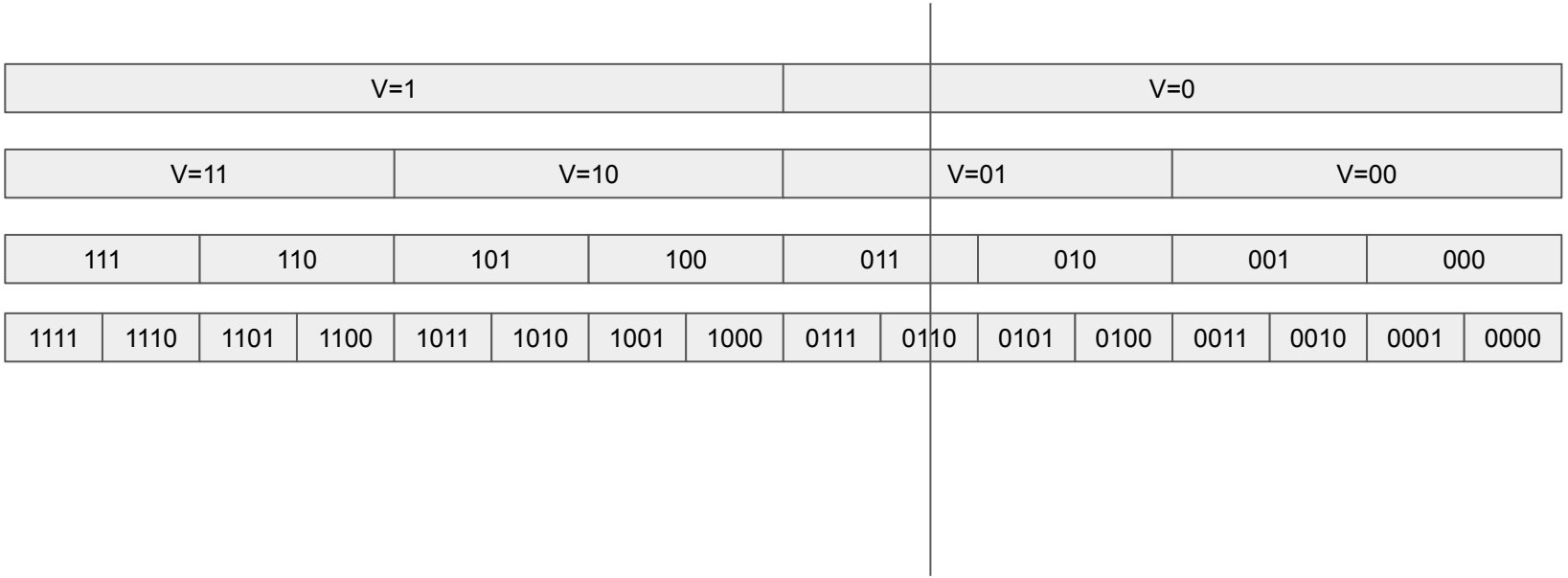

Query
C
C
C

<7



Scalar Binary Mapping

K=1: H(Search-KEY,VectorIndex,v)



hash(0110
Hash(011)
hash(01)
hash(0)

vectorpos=0 vectorpos=2

Vector = {0.80122302, 0.01140234, 0.6349501}

VectorSimHash = {Key, Vector, RangeMin=-1, RangeMax=1, BitResolution=8, kPerBit=1}

215 Scale it (0-255)

0110 1011 scalar=Binary INT

bitpos=7 bitpos=0

PositionHash = {Key, Position=1, scalar, k=1}

0110 1011 Scale=Binary INT —k→ Hash 1 = HMAC(key,{Position, INT})

1 Scale/2 —k→

1 Scale/2 —k→

0 Scale/2 —k→

1 Scale/2 —k→

`MSB_BITS=3, setting 3 bits`

0 Scale/2 —k→ Hash MSB = HMAC(key,{VectorPos, BitPosition, bitval{0,1}})

1 Scale/2 —k→ Hash MSB = HMAC(key,{VectorPos, BitPosition, bitval{0,1}})

1 Scale/2 —k→ Hash MSB = HMAC(key,{VectorPos, BitPosition, bitval{0,1}})

Native Output: 8-dimension Vector of Reals{0,1} = 32-bits * 8 = 256 bits

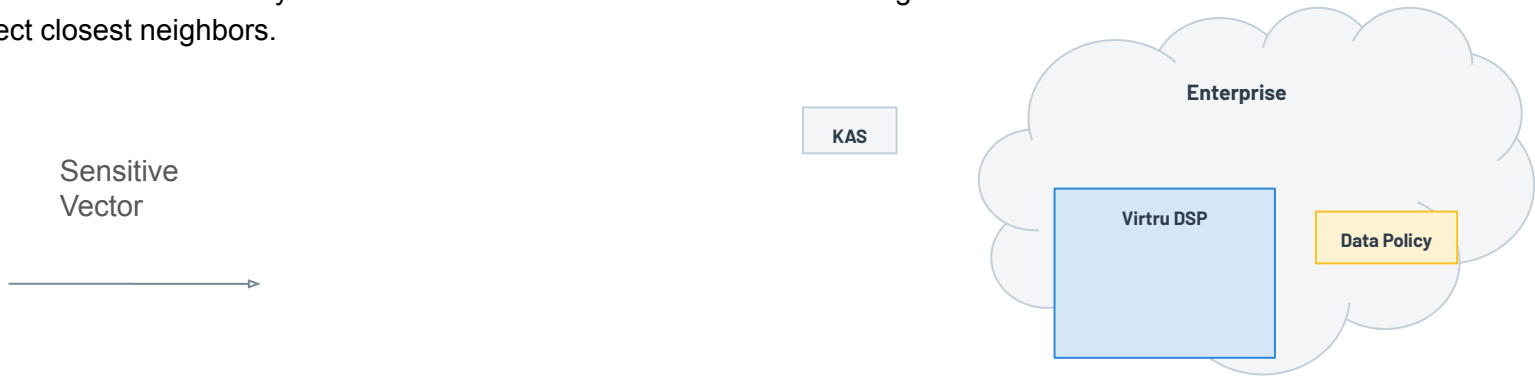
Binary Transform: 10 bits set per Dimension. (lsb=2,2,1,1,1,1,1,1)

Approx total bits set = 10 * 8 = 80.

1024 bits = 256 bytes

Explanation of Steps:

1. **xx**
2. **Step 2**: Generate synthetic vectors. This shows how to create random high-dimensional vectors, a common representation for AI-generated embeddings.
3. **Step 3**: Normalize the vectors, ensuring that each vector lies on the unit sphere, which is important for using cosine similarity.
4. **Step 4**: Create simhashes by converting each vector dimension to binary (1 for positive, 0 for negative). Simhashing is a common technique for compacting large vectors into smaller binary arrays.
5. **Step 5**: Compute Hamming distance, which is a fast way to approximate vector distances using binary arrays.
6. **Step 6**: Compute cosine similarity, which is often considered the "true" distance measure for high-dimensional embeddings.
7. **Step 7**: Compare the accuracy of the Hamming-based retrieval to the cosine-based retrieval, illustrating the trade-off between speed and accuracy.
8. **Step 8**: Visualize the accuracy distribution to demonstrate how often Hamming-based retrieval finds the correct closest neighbors.



ok perfect. lets now create a more real simhash.

Vector_Sim_Hash2(Vector=256, BitResolution=16, kPerBit=1}

Function to generate simhash from a vector (with detailed per dimension encodings)

Step 0 - initialize a large (8096-bit) binary vector called “large_sim_hash” with all zeros, for us to populate later.

step 1 - convert each dimension into a 16-bit unsigned int called “Vector_BITS” (all zeroes for -1.0 (minimum) up to 1111 for 1.0 (maximum)

Step 2 - for each dimension, create a set of 16 hashes. The first hash will be SHA(bit_position, BITS). Where BITS is the full 16 bits. Then SHIFT the BITS one to the RIGHT, losing the least significant bit. And then SHA again with those 15 bits. REpeat until you have produced 16 hashes.

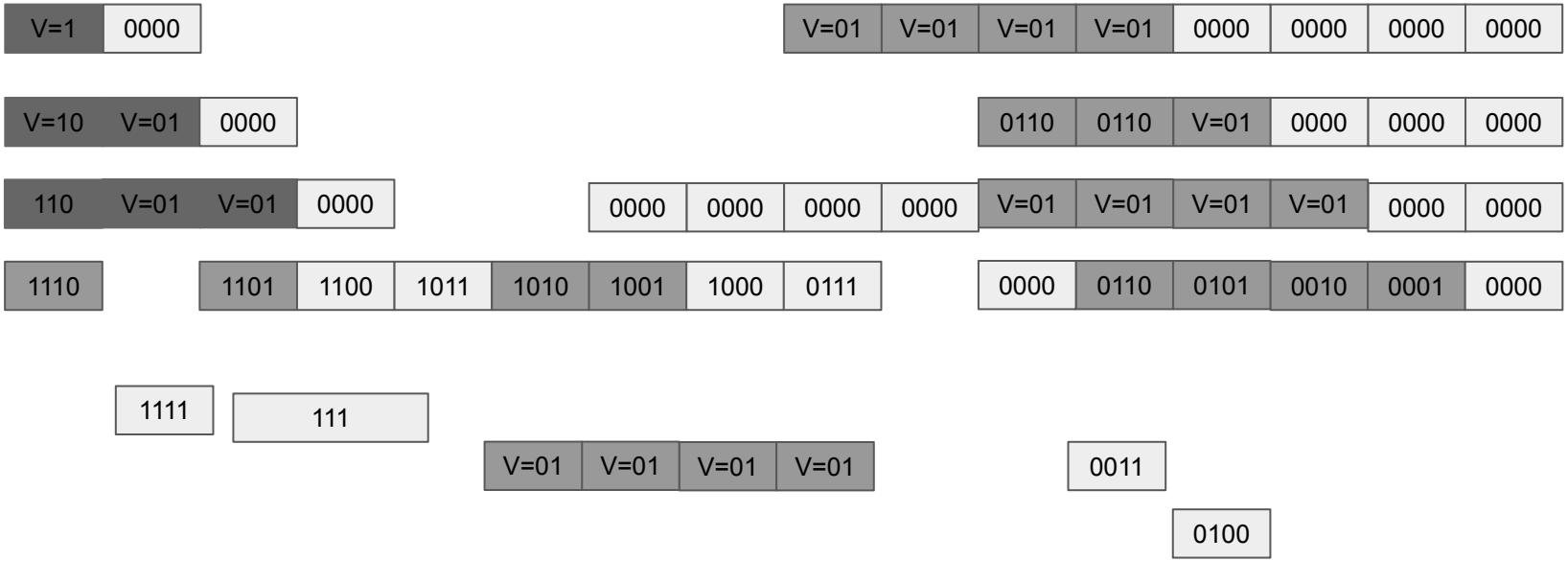
Step 3 - use those hashes to set ONE bit in large_sim_hash, perhaps by doing a 4096 mod on each hash.

Step 4 - repeat for each dimension until 256*16 = 2048 bits are set in the vector.

Step 5 - return large_sim_hash

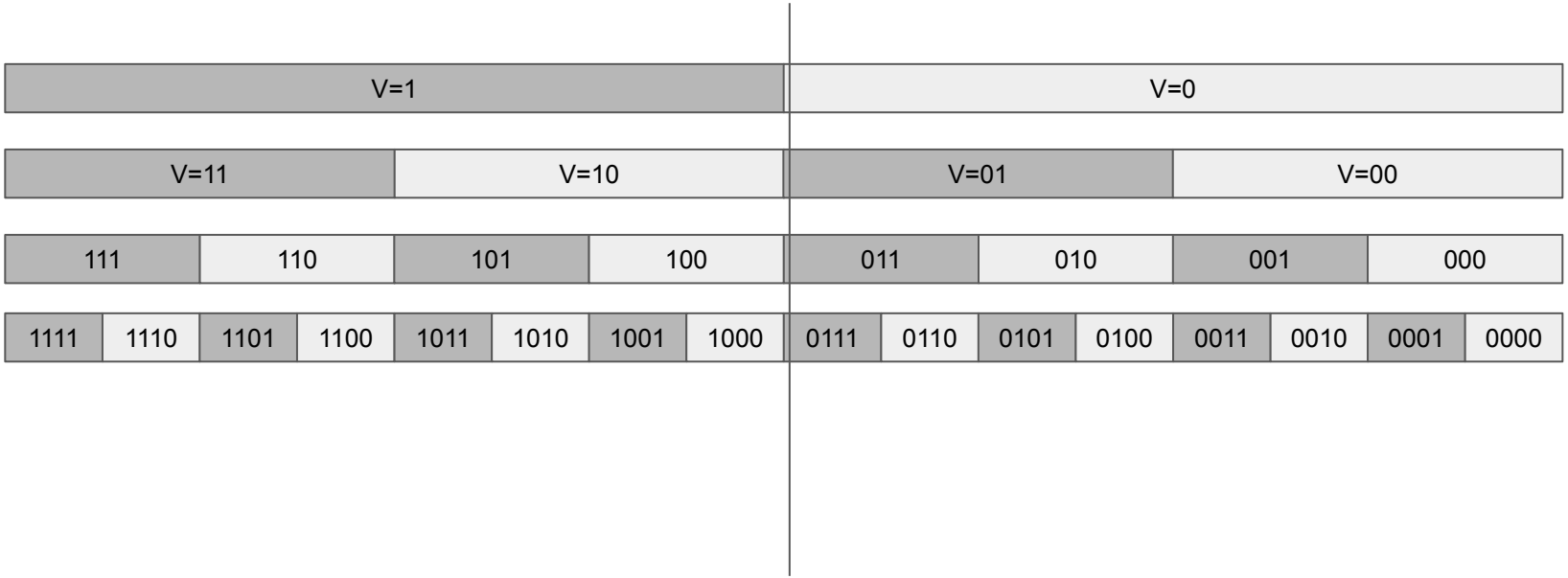
Scalar Binary Mapping

K=1: H(Search-KEY,VectorIndex,v)

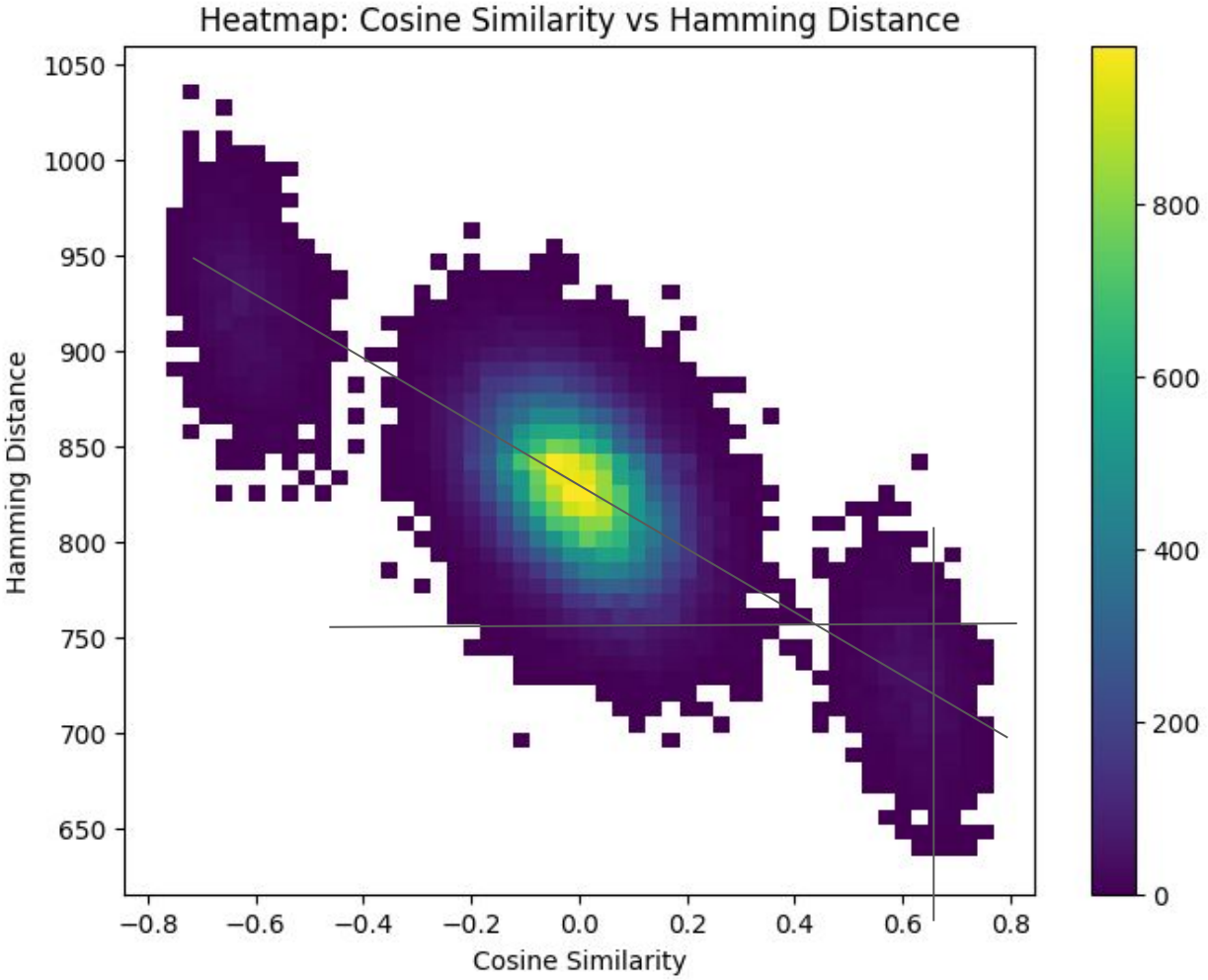


Scalar Binary Mapping

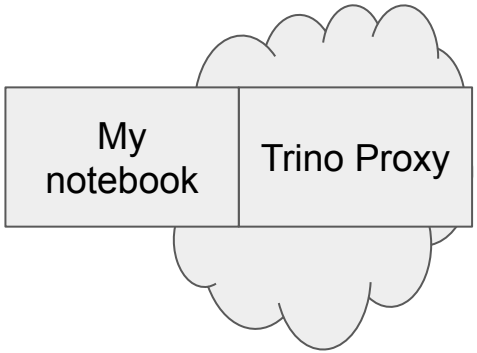
K=1: H(Search-KEY,VectorIndex,v)

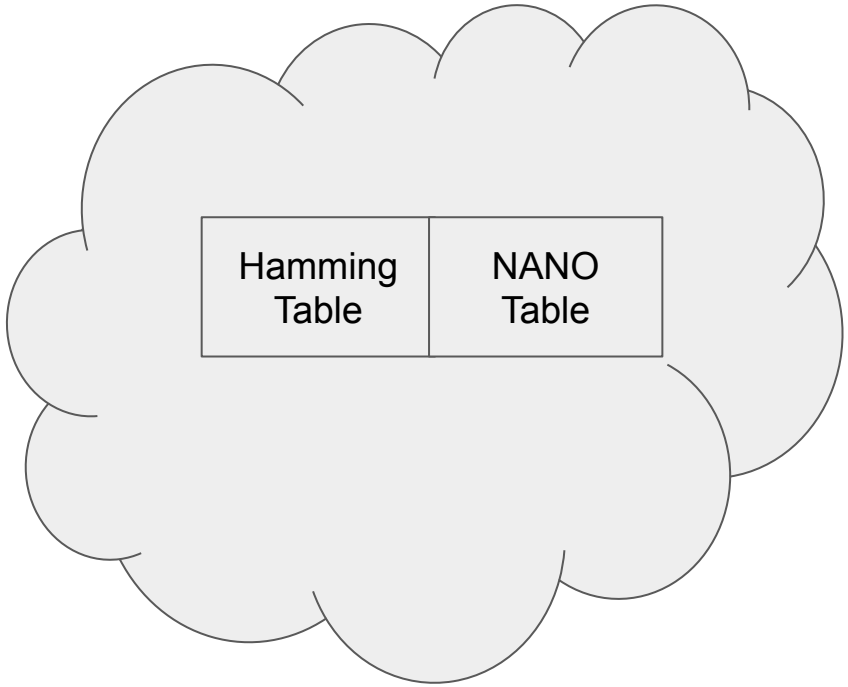


hash(0110
Hash(011)
hash(01)
hash(0)



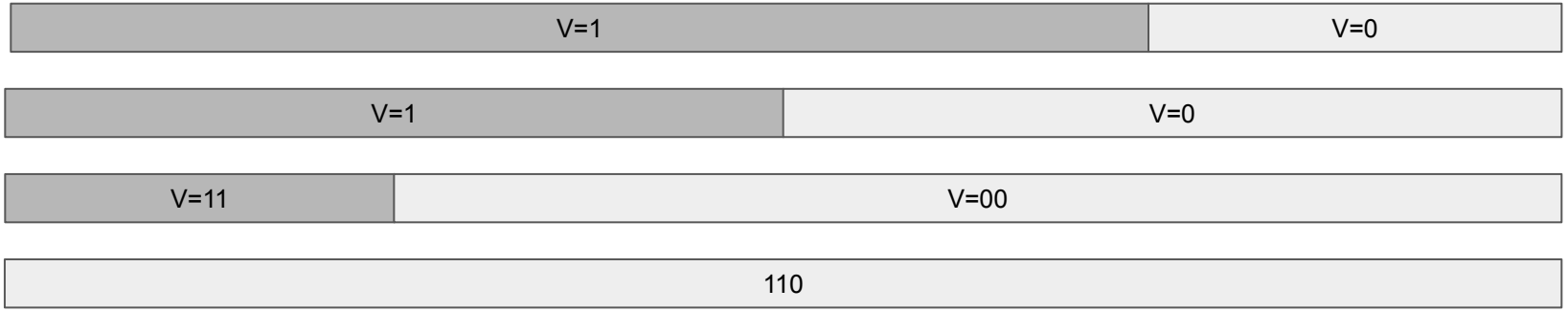
hi I have a hash algorithm that can preserve similarity of vectors. It uses an XOR, and I'd like to run benchmarks assuming i highly optimized packed binary array of

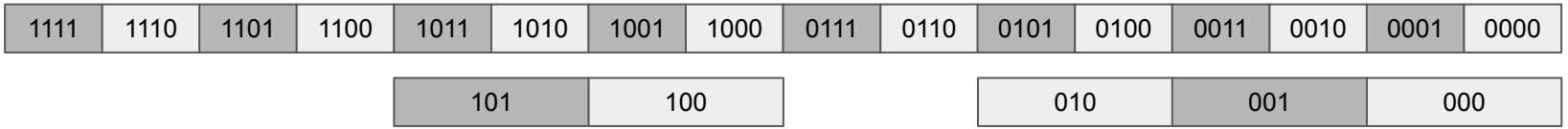

My
notebook
Trino Proxy


Hamming
Table
NANO
Table

Scalar Binary Mapping

K=1: H(Search-KEY,VectorIndex,v)



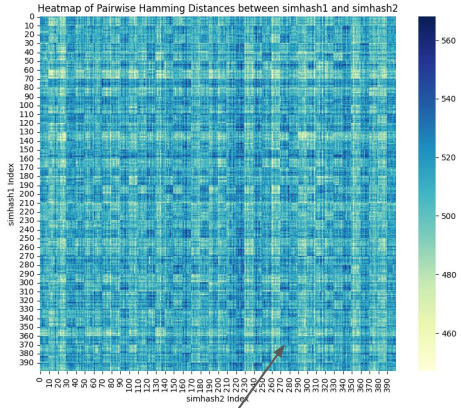


New Privacy-Enhanced RAG Vector Search (e.g. for LLM): Significant improvement in privacy state-of-the art with a **PUBLIC INDEX** threat model

Shares key similarities to Virtru's patented searchable encryption, adapted for RAG Vectors

- **Smaller -** 10.5x reduction in index space (1024-bit hash vector vs 384*32-bit reals) using improved "Sim Hash" approach
- **Faster** - **8.3x** faster in a Jupyter Notebook using default numpy (**390x** faster than numpy on A100 with custom CUDA kernel)
- **Tradeoff** - **Small** increase in false positive returns (similar to Bloom Filter, tunable), can be addressed w/ client-side filter.

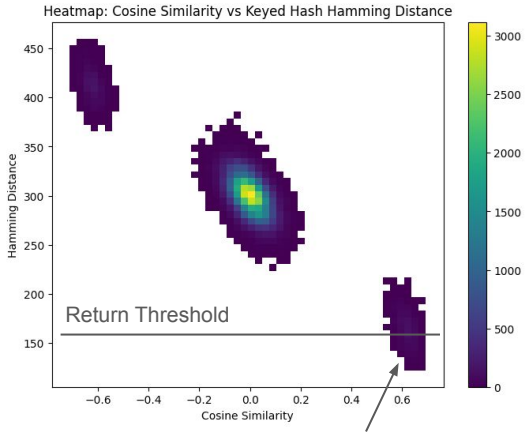
Results w/ INCORRECT KEY



Uncorrelated results

VS

Results w/ CORRECT KEY



Only High Quality Results

Relevant Results

```
query_embeddings(COLLECTION_BASELINE, QUERY_2)
query=What is new in postgres 16
```

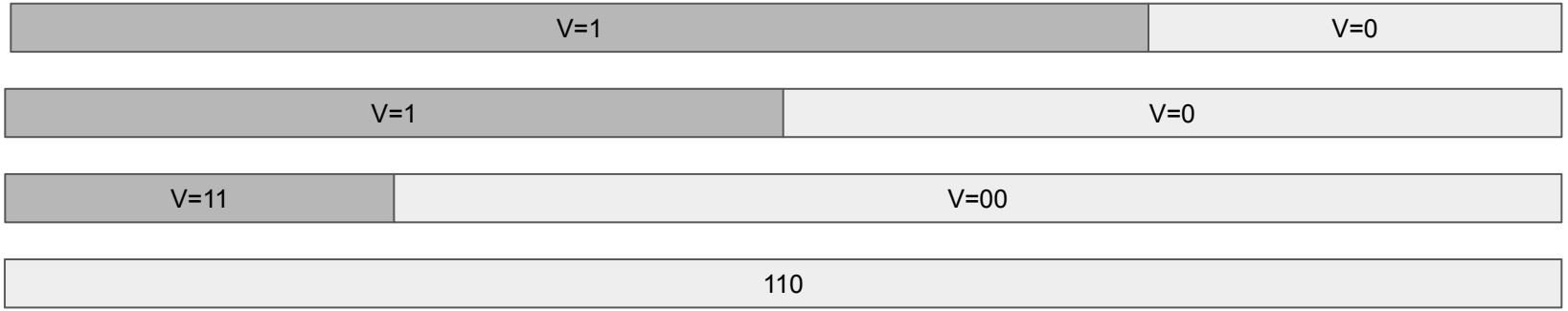
id	hamming_distance	text	simhash
1	174	In case you missed it, Postgres 16 came out la...	[1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, ...
46	195	We did a lot of things right by accident like ...	[0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, ...
71	202	to do for startups everything Julian had done ...	[1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, ...
78	206	In the summer of 2006, Robert and I started wo...	[1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, ...
96	208	[8] Most software you can launch as soon as it...	[0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, ...
...	...	...	...
92	260	Working on Bel was hard but satisfying. I work...	[1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, ...
64	260	It's not that unprestigious types of work are ...	[1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, ...
18	261	Toward the end of the summer I got a big surpr...	[1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, ...
24	263	is that it's the edge of an object. By subtly ...	[1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, ...
91	268	I had to ban myself from writing essays during...	[1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, ...

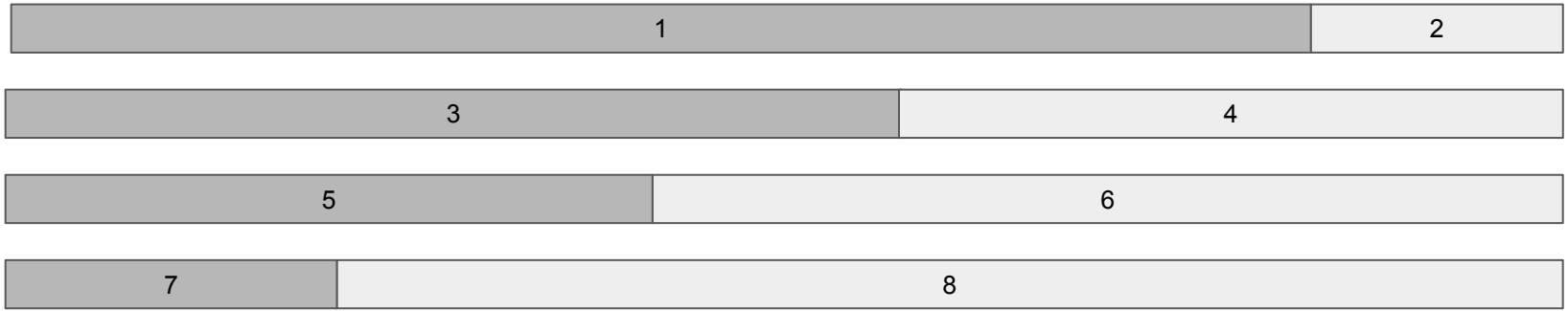
```
do_sim_hashing(QUERY_2)
```

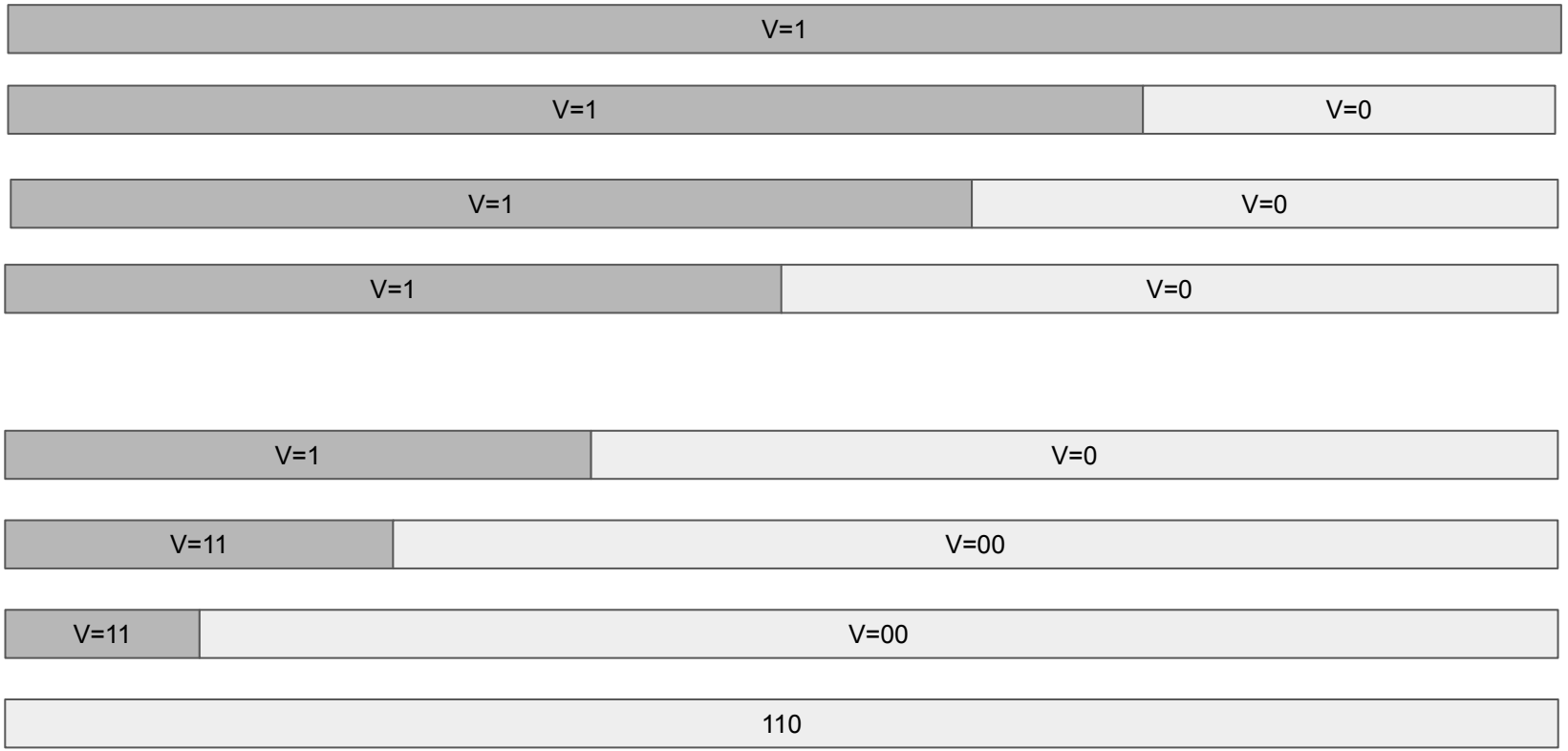
	id	hamming_distance	text	simhash
103	1	174	In case you missed it, Postgres 16 came out la...	[1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, ...
45	46	195	We did a lot of things right by accident like ...	[0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, ...
70	71	202	to do for startups everything Julian had done ...	[1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, ...
77	78	206	In the summer of 2006, Robert and I started wo...	[1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, ...
95	96	208	[8] Most software you can launch as soon as it...	[0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, ...
...	...	...	...	...
91	92	260	Working on Bel was hard but satisfying. I work...	[1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, ...
63	64	260	It's not that unprestigious types of work are ...	[1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, ...
17	18	261	Toward the end of the summer I got a big surpr...	[1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, ...
23	24	263	is that it's the edge of an object. By subtly ...	[1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, ...
90	91	268	I had to ban myself from writing essays during...	[1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, ...

Scalar Binary Mapping

K=1: H(Search-KEY,VectorIndex,v)

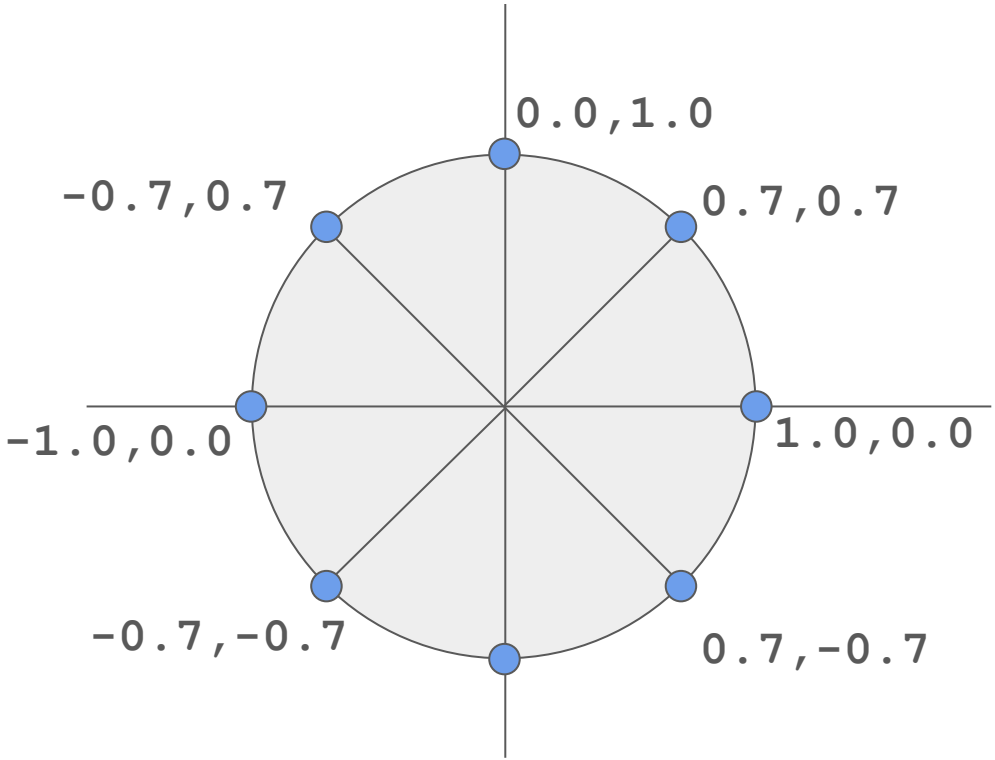





V=1
V=1
V=0
V=1
V=0
V=1
V=0
V=11
V=00
V=11
V=00
110

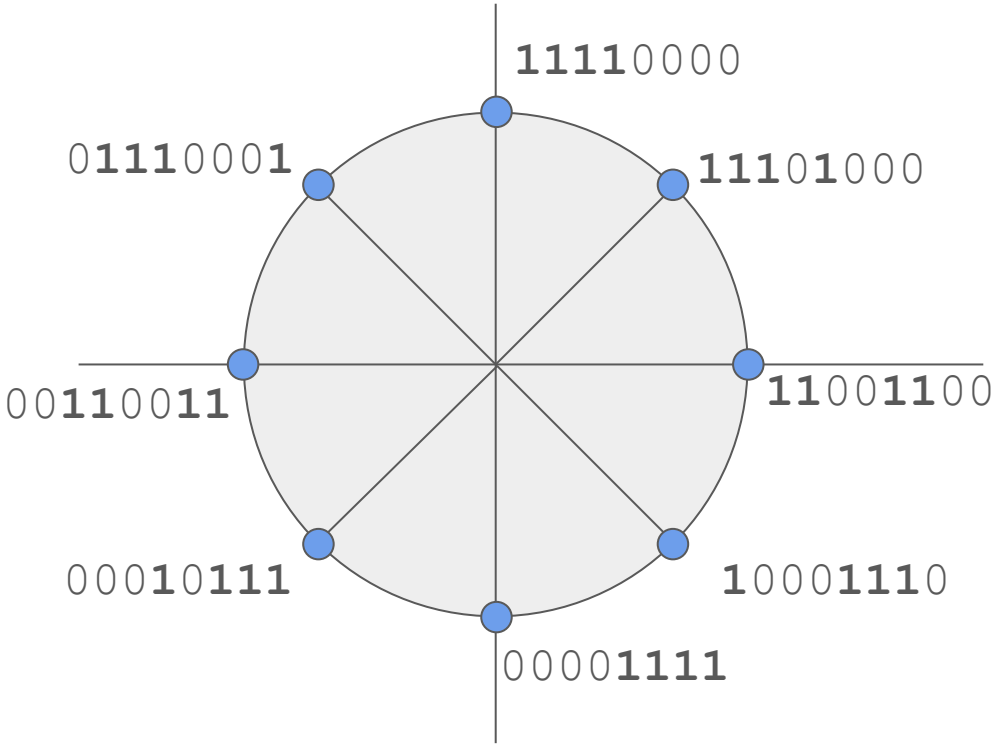
Native Coordinate System - Normalized X,Y Example (all distances from origin = 1)

Cosine similarity would be performed to evaluate how “similar” two items are.

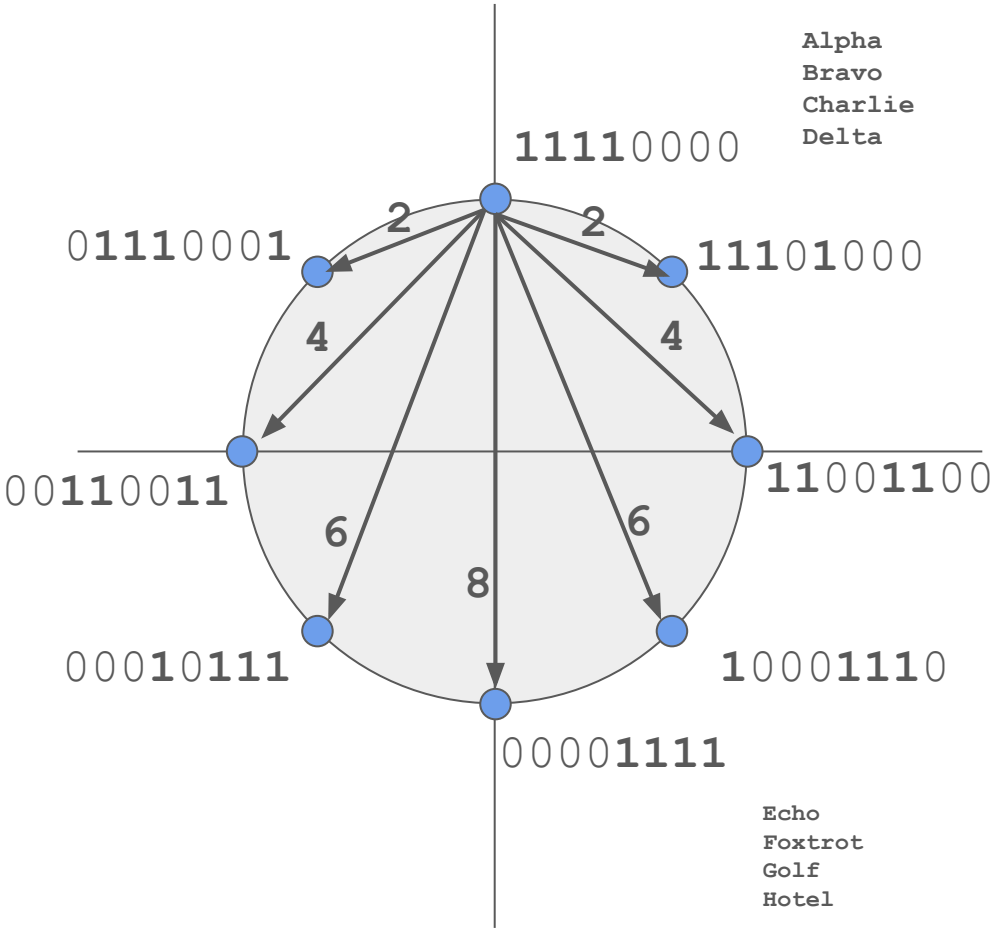


Binary Mapping for a vector or vector group

Hamming Distance used to compare “distance” between two items.



Hamming Distances - perfect symmetry, no singularities



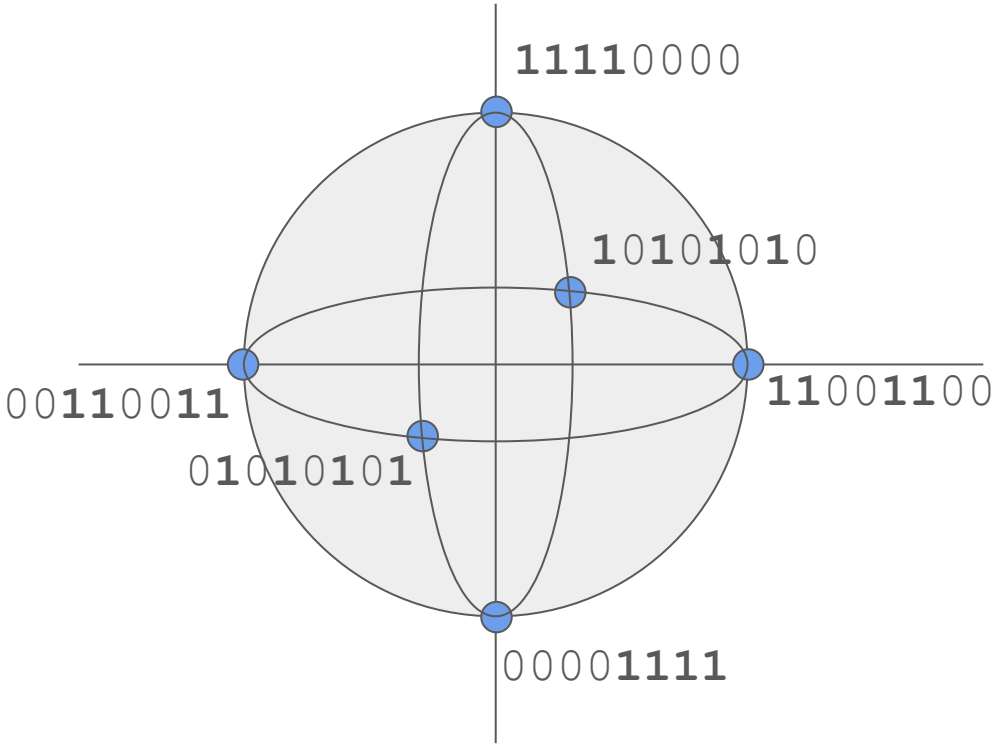
10000000 Alpha
01000000 Bravo

00100000 Charlie
00010000 Delta

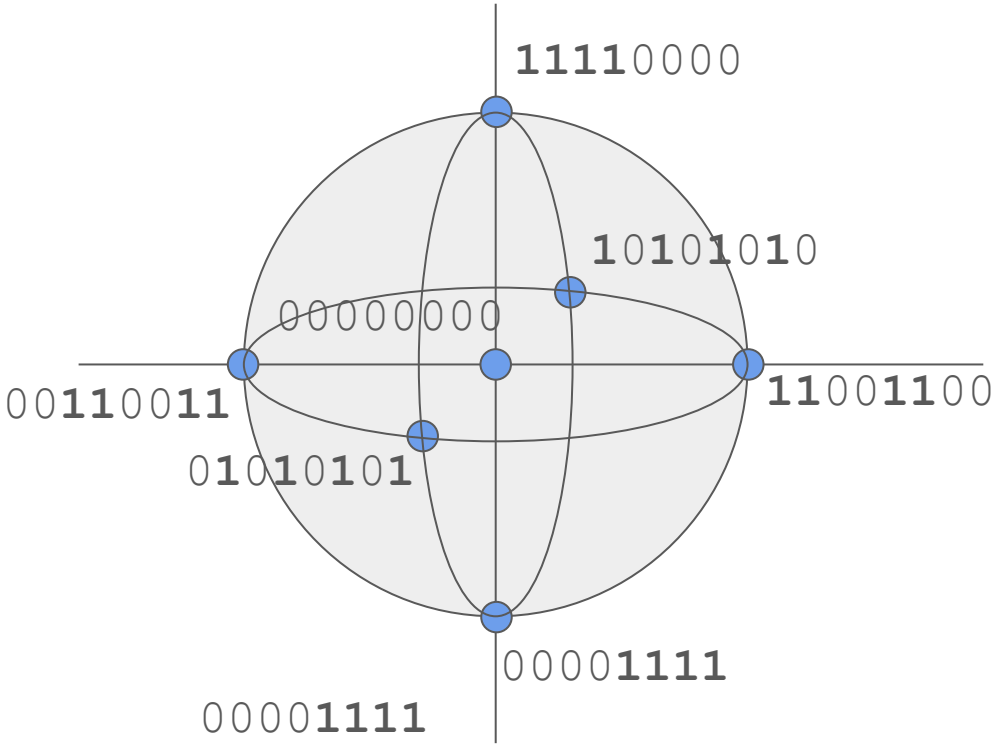
00001000 Echo
00000100 Foxtrot

00000010 Golf
00000001 Hotel

Sphere Surface mapping



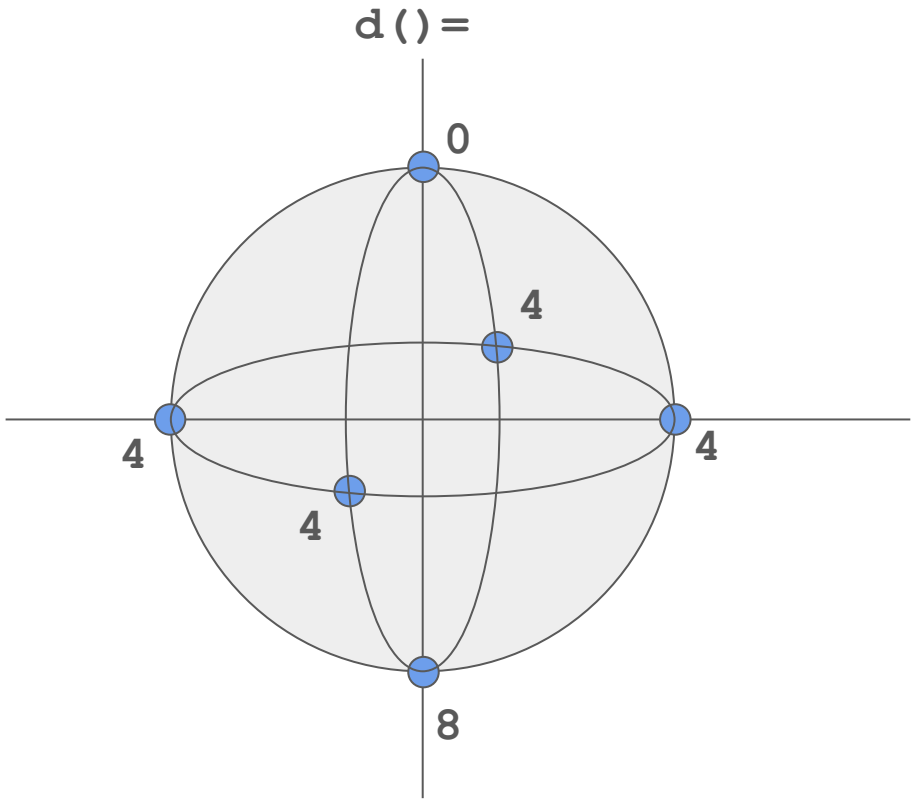
Sphere Volume mapping



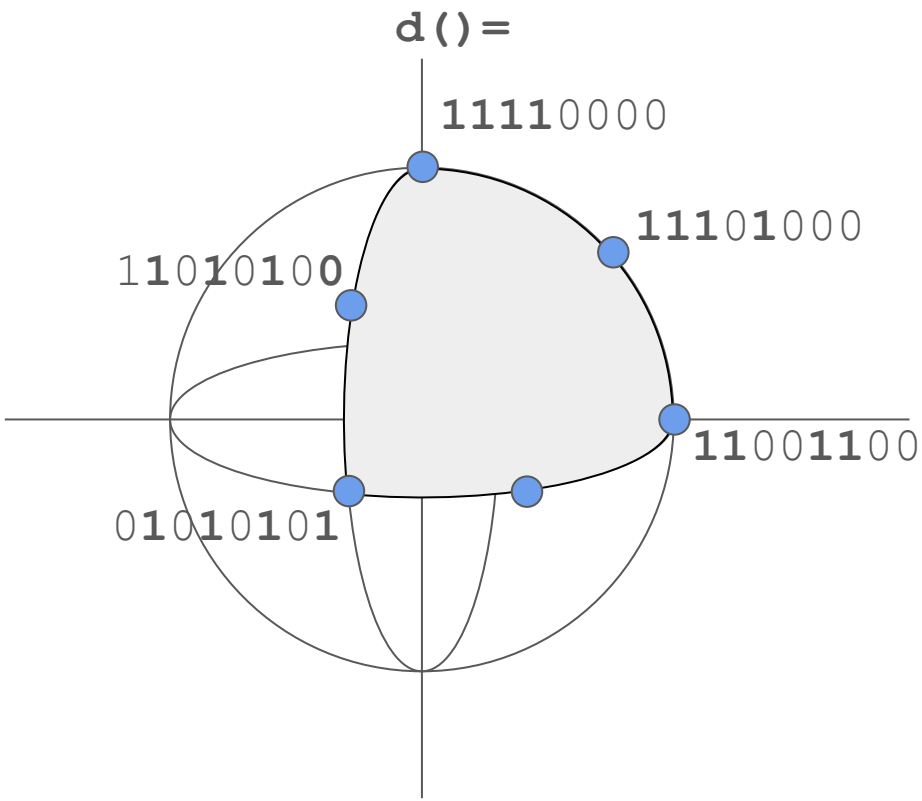
11110000

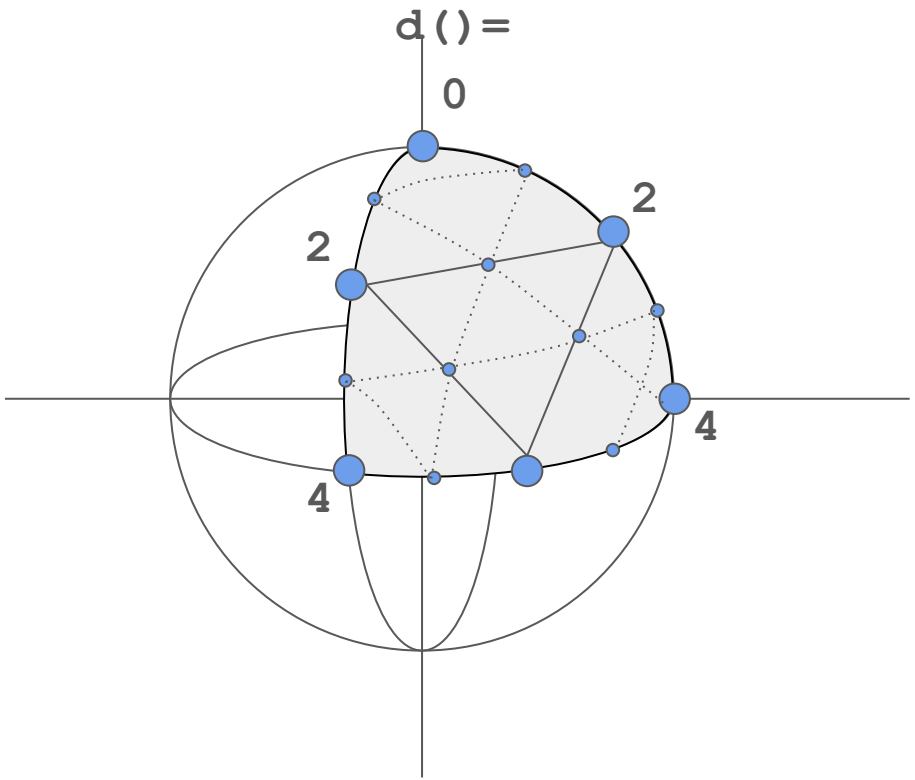
00001111

Sphere Surface mapping

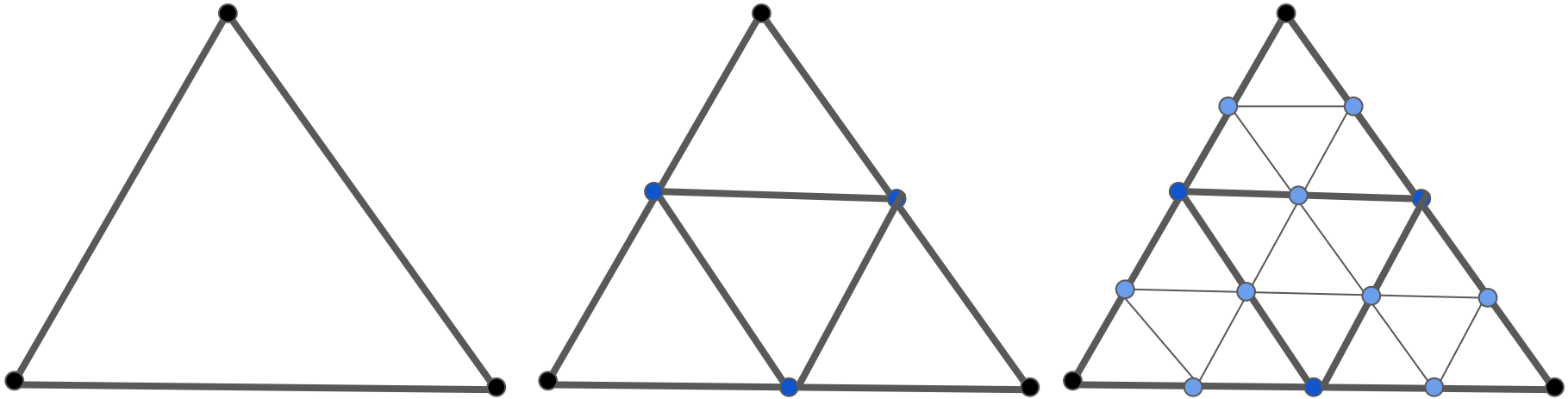


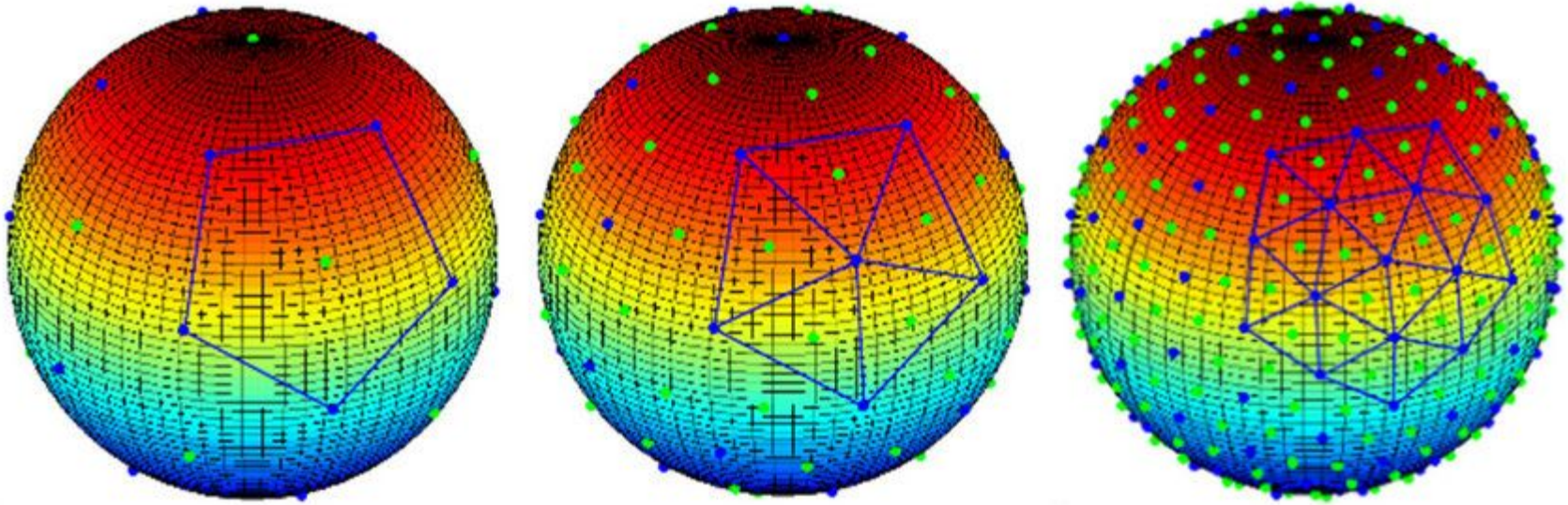
Sphere Surface mapping





Similar algorithm for Triangles, Tetrahedra, n-simplex
Space Filling Curve-ish – mapped to Hamming Distance optimized vectores of fixed POPCOUNT.





I think moving this approach as far up in the chain as possible may greatly benefit things.

Orthogonal Linear Discriminant Analysis (OLDA) or **Regularized Discriminant Analysis (RDA)** can sometimes help distribute weights more evenly by regularizing components to avoid over-reliance on a few dominant ones.

PCA NOTES

there are variations of PCA that address the issue of dominant components by balancing or equalizing the contribution of each dimension. Some of the main approaches include:

1. **Standardization and Whitening**: Standard PCA can be modified by **standardizing** (z-score normalization) or **whitening** the data. This gives each feature unit variance before applying PCA, helping prevent any single feature from dominating because of its large scale or variance. Whitening, in particular, decorrelates the components, ensuring they all have equal variance, though it can sometimes introduce noise.

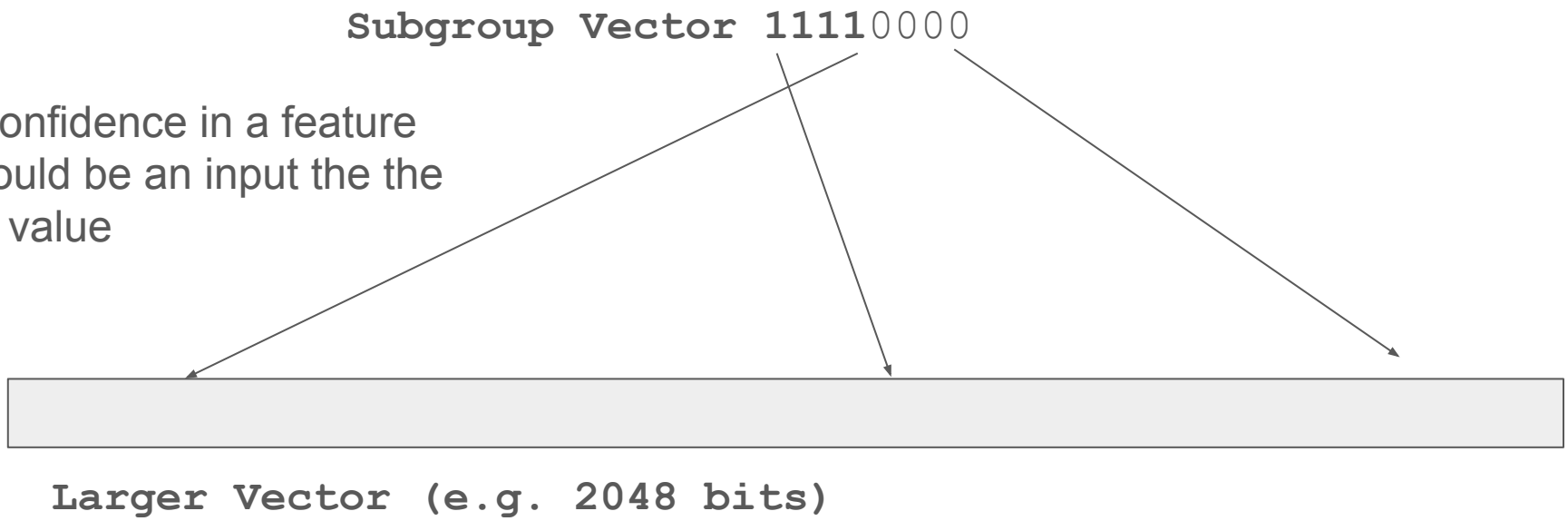
2. **Sparse PCA**: In **Sparse PCA**, constraints are added to encourage sparsity in the components, limiting the influence of individual features. Sparse PCA can be helpful if you want each component to be defined by only a subset of features, giving a more balanced view of the data.

3. **Kernel PCA**: **Kernel PCA** uses a kernel function to project data into a higher-dimensional space where PCA is then applied. Depending on the kernel, this can balance the contributions from various directions in the input space, helping mitigate the dominance of the first few components.

4. **Independent Component Analysis (ICA)**: Though not a version of PCA, **ICA** is an alternative technique that finds statistically independent components rather than maximally varying ones. ICA can be useful when you're more interested in separating mixed signals (e.g., for source separation) and can yield components that are more balanced in their explanatory power.

5. **Autoencoders (Nonlinear PCA)**: An autoencoder, a type of neural network, can be trained to find a lower-dimensional representation of data in a way that is not limited by maximizing variance. This representation can be more balanced, especially in nonlinear data structures, as it doesn't rely solely on variance for dimensionality reduction.

Select Function to map a Sub-Group vector to the aggregate



TO DO:

Implement the std (keyed) random hyperplane mode of creating bit vector.

Create ECC cluster version for multi-scale search

Insight: Space filling curves are for linear sorts. Space Filling Hash Maps are for something else but related. They’re both coordinate transforms, but one is constrained by a linear style adjacency, where adjacency in the hash space relates more to short or long hamming distances.