Web-Scale Multimedia: Optimizing LSH

Malcolm Slaney
Yury Liftshits
Junfeng He
Y! Research

Nearest Neighbor Search

Two Approaches

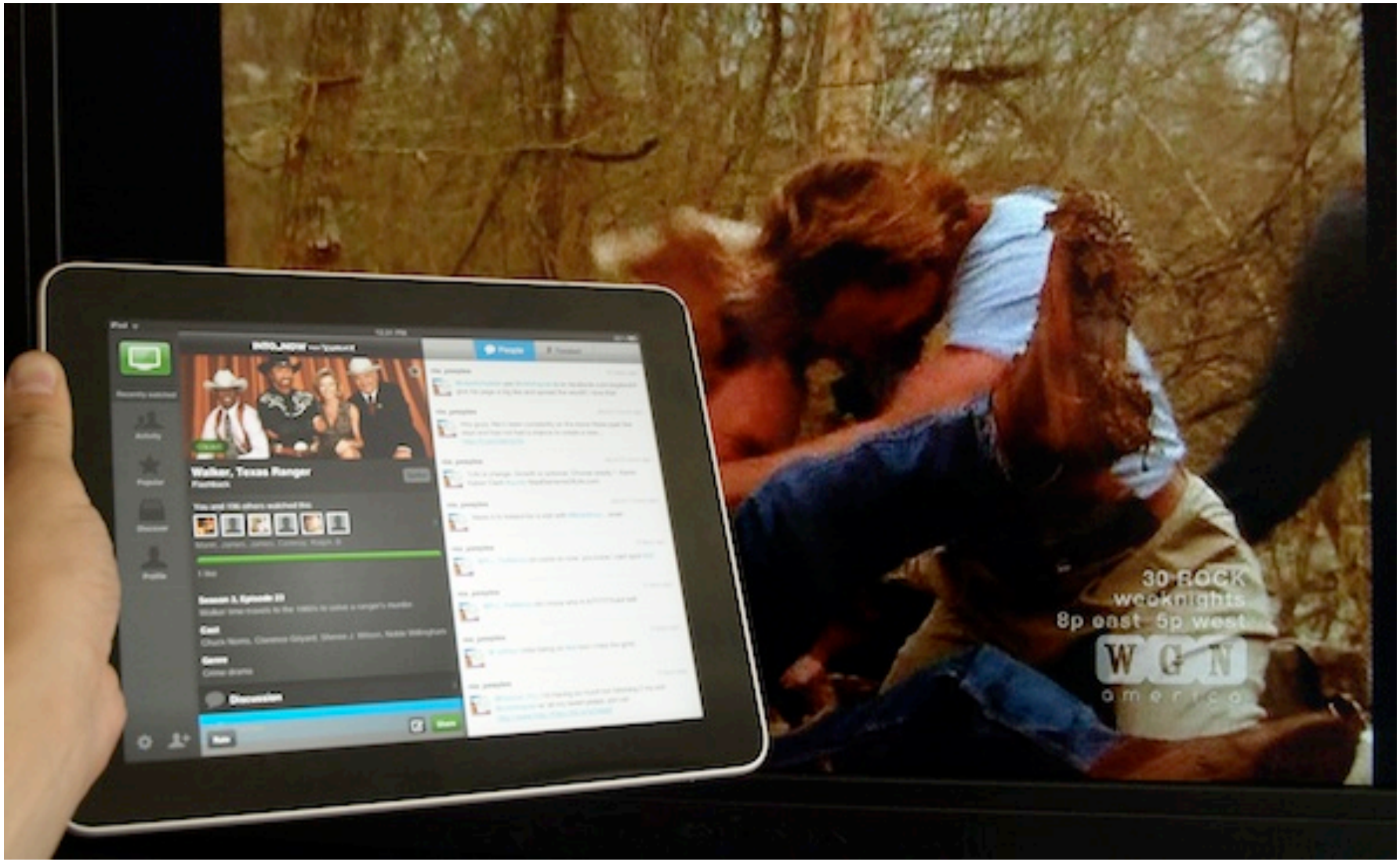
Probabilistic → **LSH**

Deterministic → **K-D Trees**

Why?

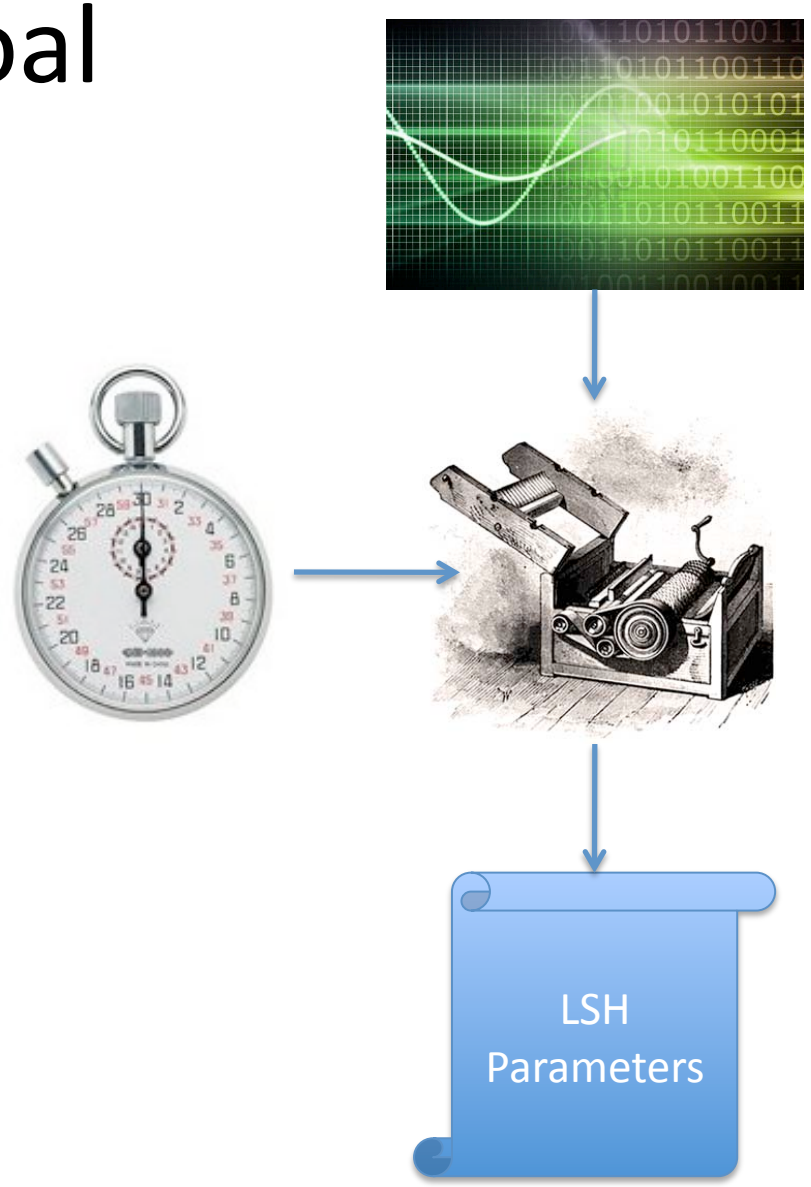
Potential LSH Uses

IntoNow and ConnectedTV



Goal

- Given large data set, optimize LSH parameters to minimize CPU time



The Hammer

- Locality Sensitive Hashing
  - Nearest neighbor
  - Identify
- Performance
  - Efficient index
- Probabilistic
  - Tunable error

Variations

- Smart Projections
  - Spectral Hashing (Weiss)
  - Forgiving Hashes (Baluja)
  - WTA (Yagnik)
- Other metrics….
  - Kernels
  - Closest plane

Talk Outline

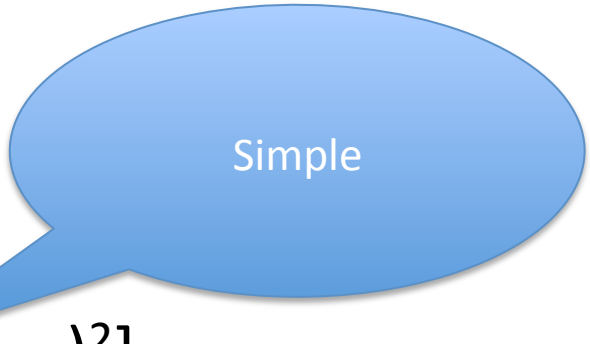
- Motivation
- LSH
- Parameter Optimization
- Extensions

Key Result

Speculation

Indexing Problem

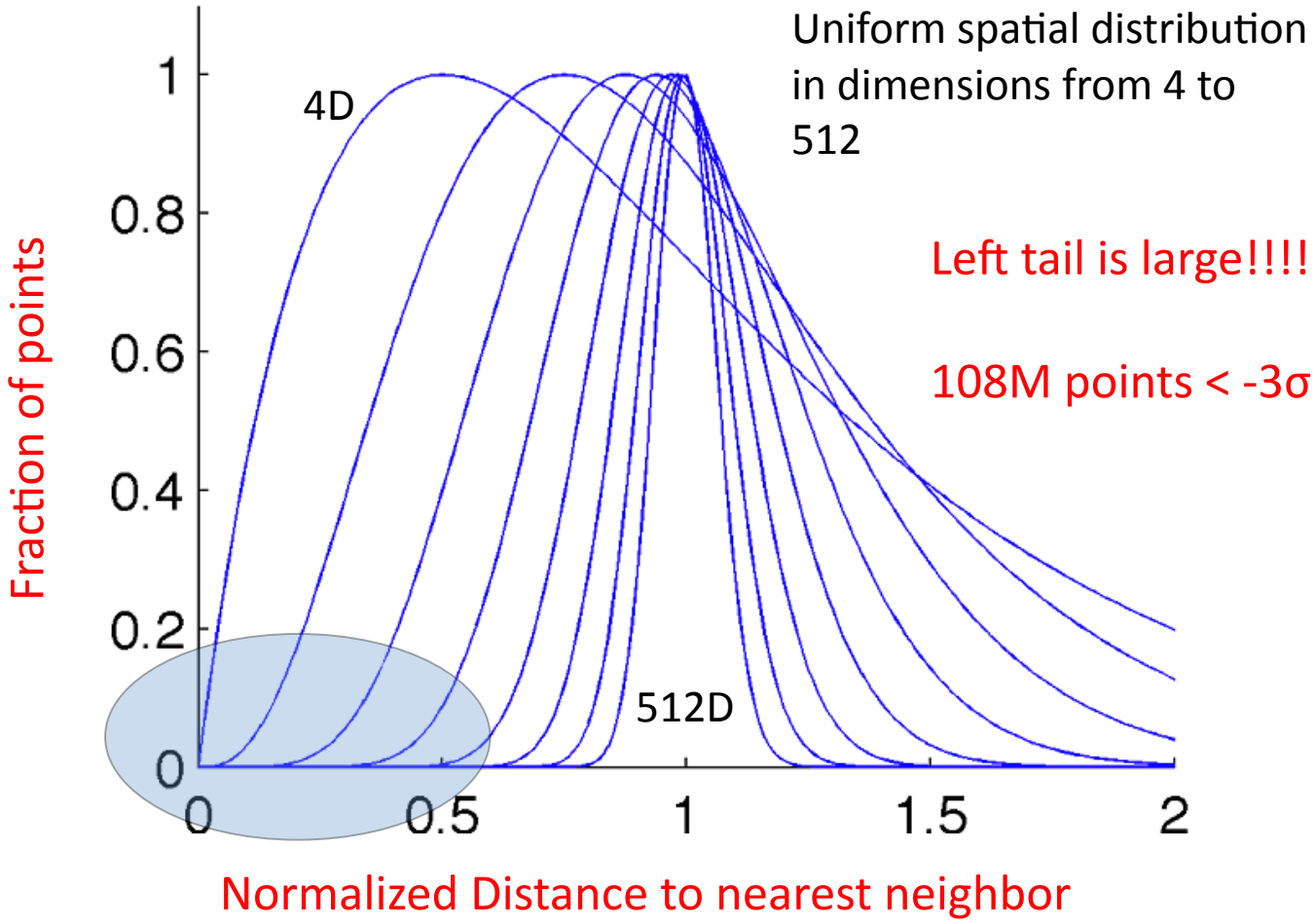
- Find nearest neighbors
  - Euclidean
  - Sqrt[$(x_1-q_1)^2+(x_2-q_2)^2+....+(x_{234}-q_{234})^2$]



- But (for audio)…
  - 1B entries
  - 234-D data
  - 234G Flops per query!

    >>>> 250ms

Curse of Dimensionality

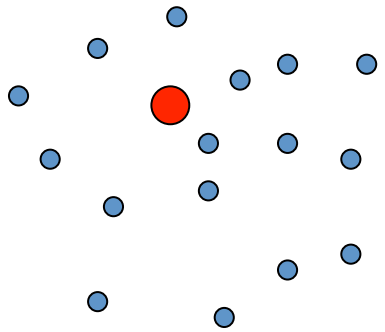


Hashing

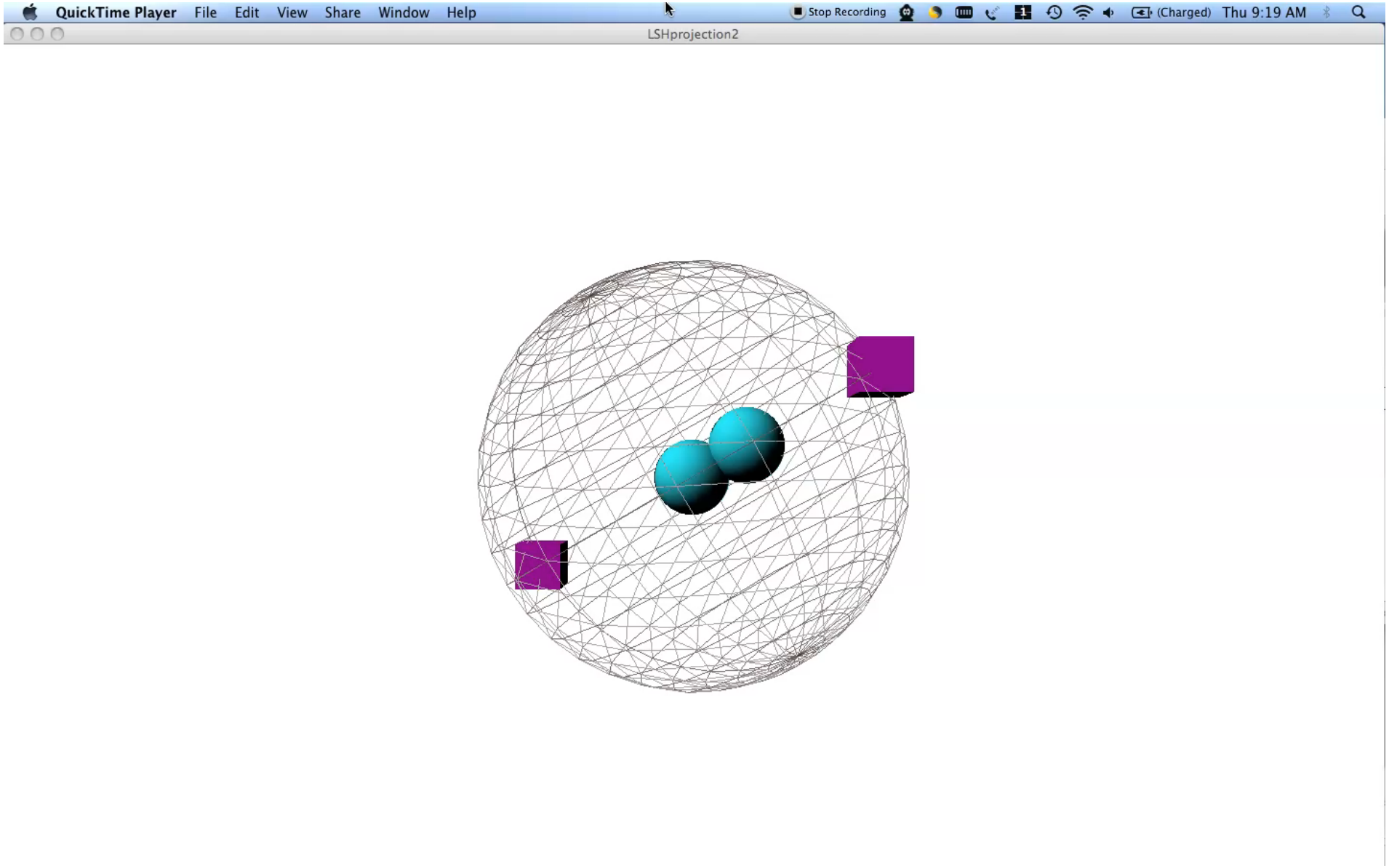
Conventional

Separate Neighbors

Locality Sensitive



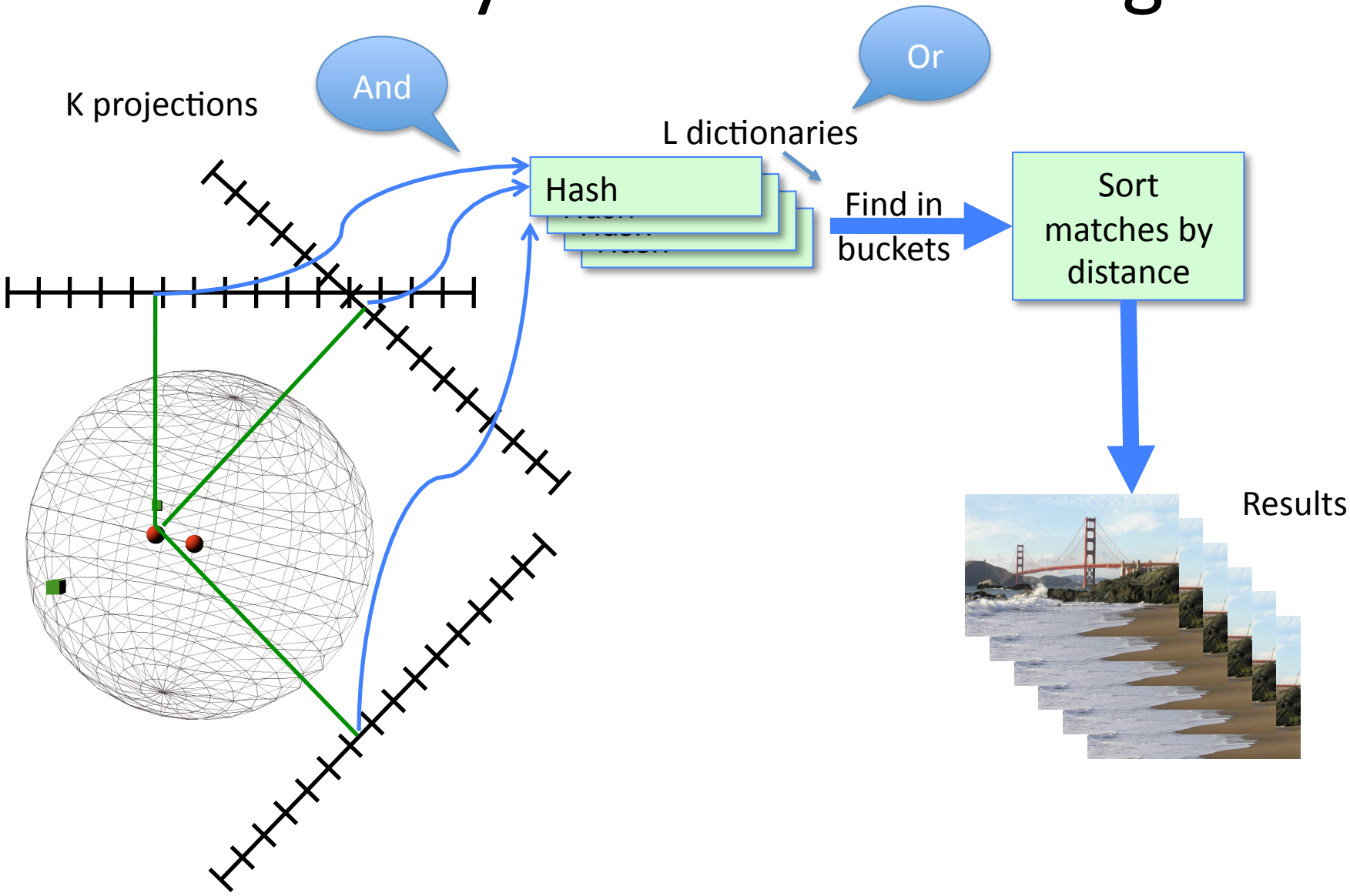
Find closest Neighbors



Key: Points that are close together stay close together


WELCOME
TO Fabulous
LAS VEGAS
NEVADA
What happens near Q stays near Q

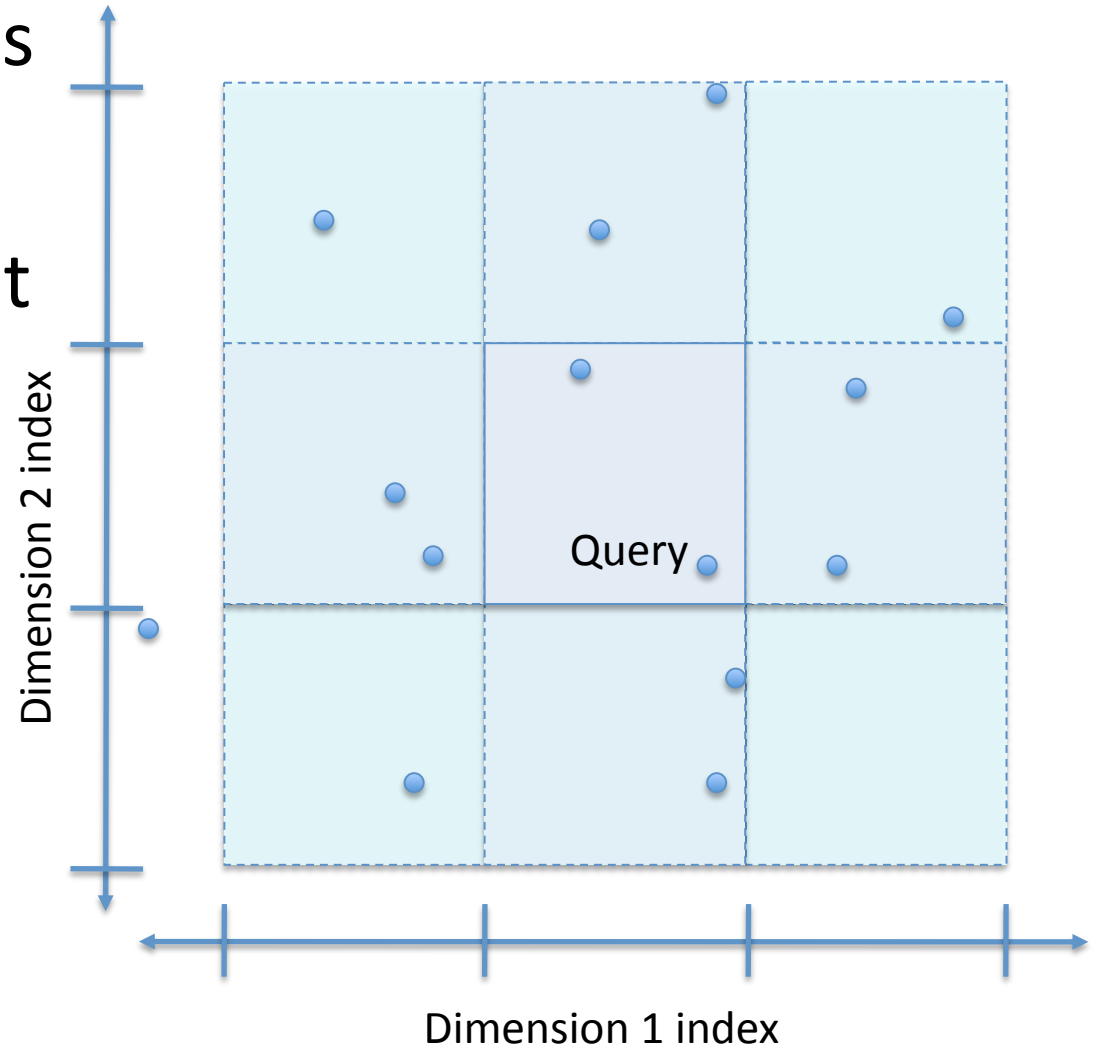
Locality-Sensitive Hashing



Multiprobe

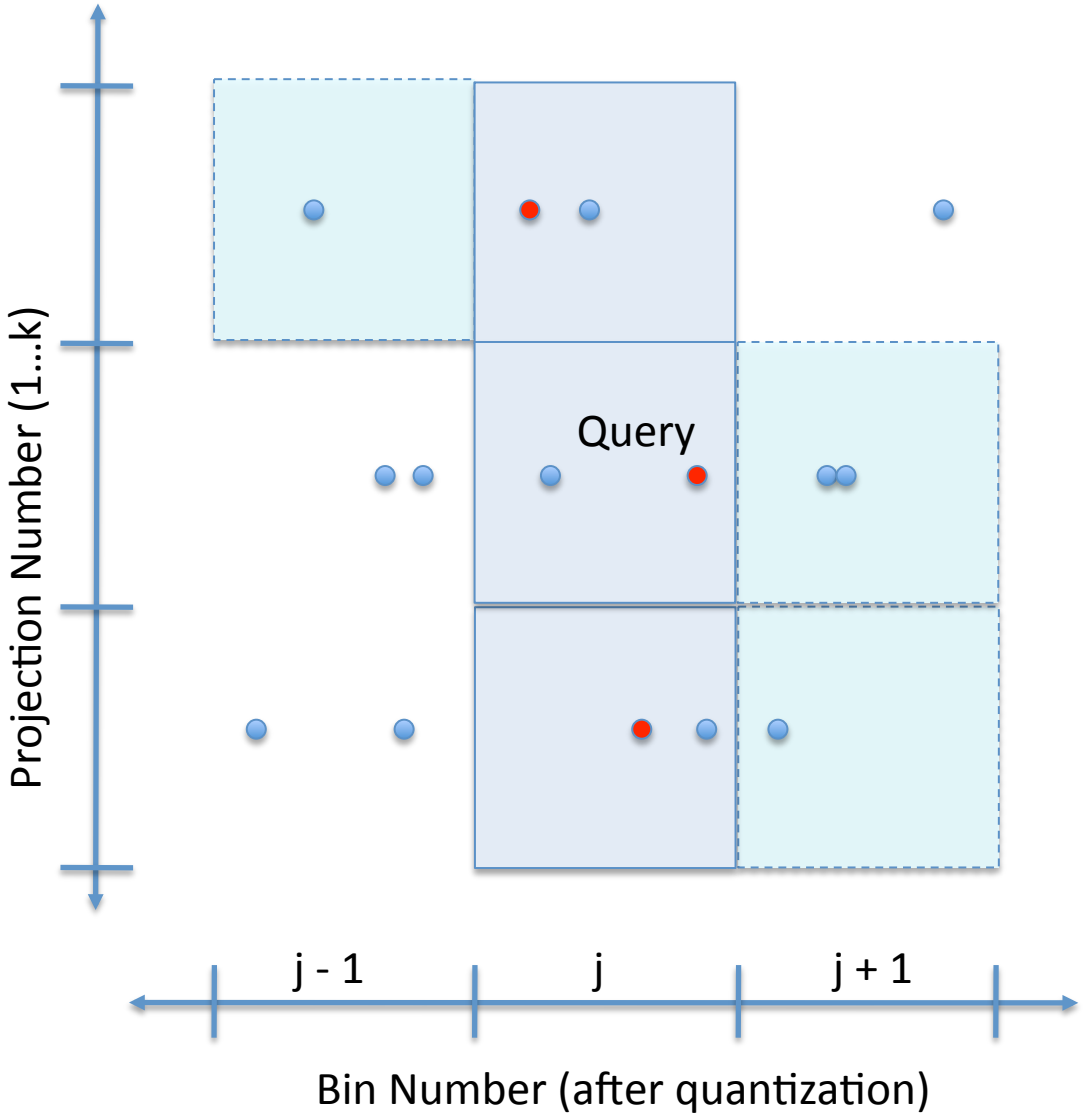
- Search nearby bins
- 2^k neighbors!!
- Only search subset

- Expands the quantization interval



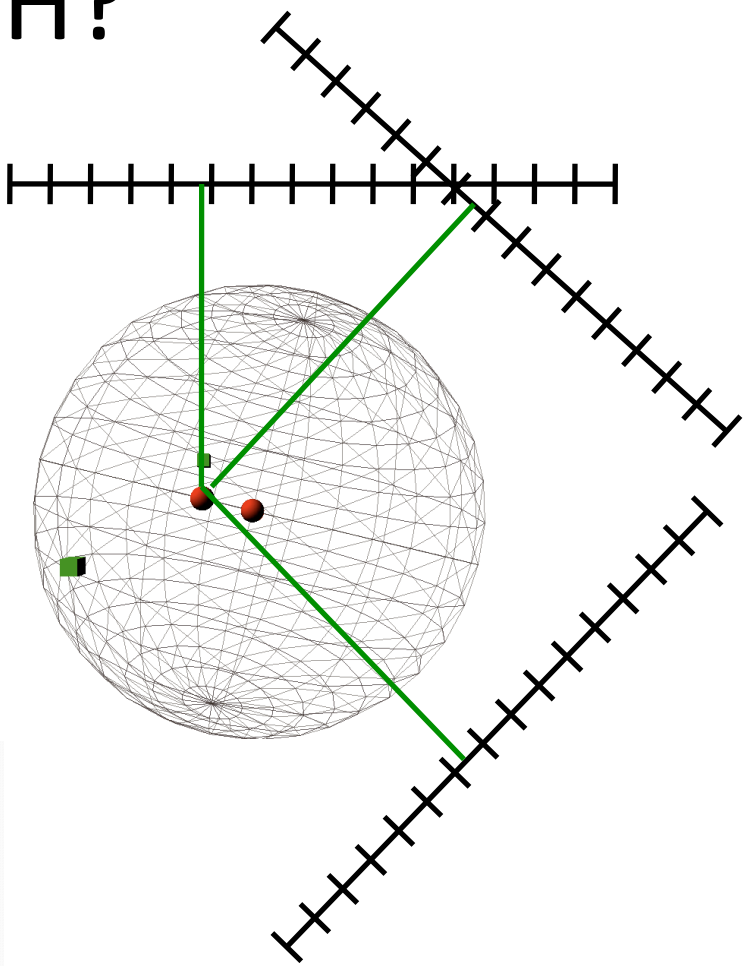
Unwrap Multiprobe

- Examine nearby bins
- Original
  - Order by distance
- Better
  - Choose r from k
  - C_k^r



Why LSH?

- One projection
  --> Many Buckets!

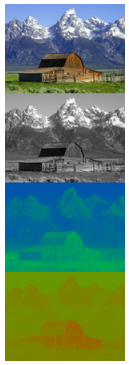


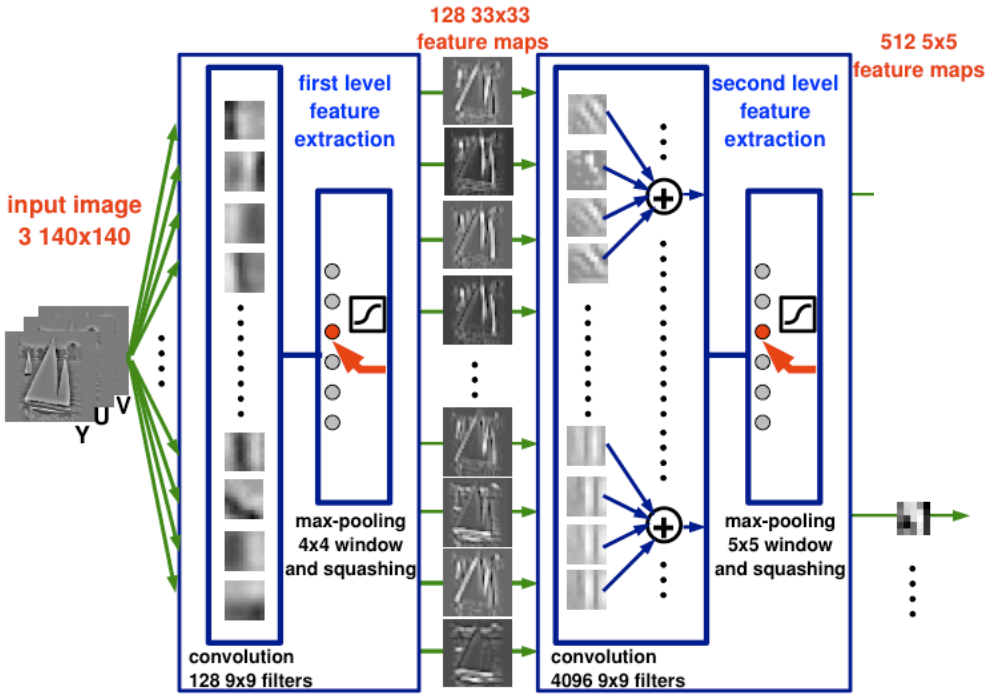
- Cool!
  Trade time for memory

- Penance

Example Features

- Image (DBN/CNN)
- Audio
- Synthetic
  - Random from 10-40D

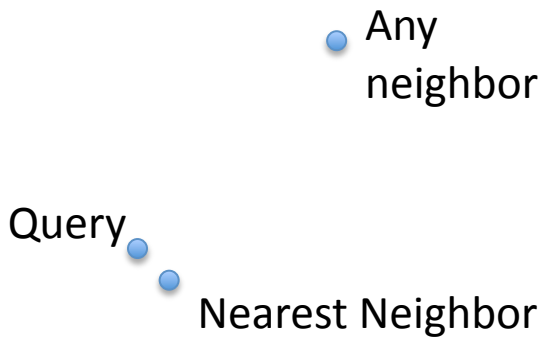




Designed by Marc'Aurelio Ranzato

Our LSH Optimization

- Depends only on the distance distribution
  - Independent of dimensionality
  - Real or intrinsic
- Parameters
  - U_{hash}: Cost to calculate projections
  - U_{check}: Cost to check distance



Expensive if data is on disk!!

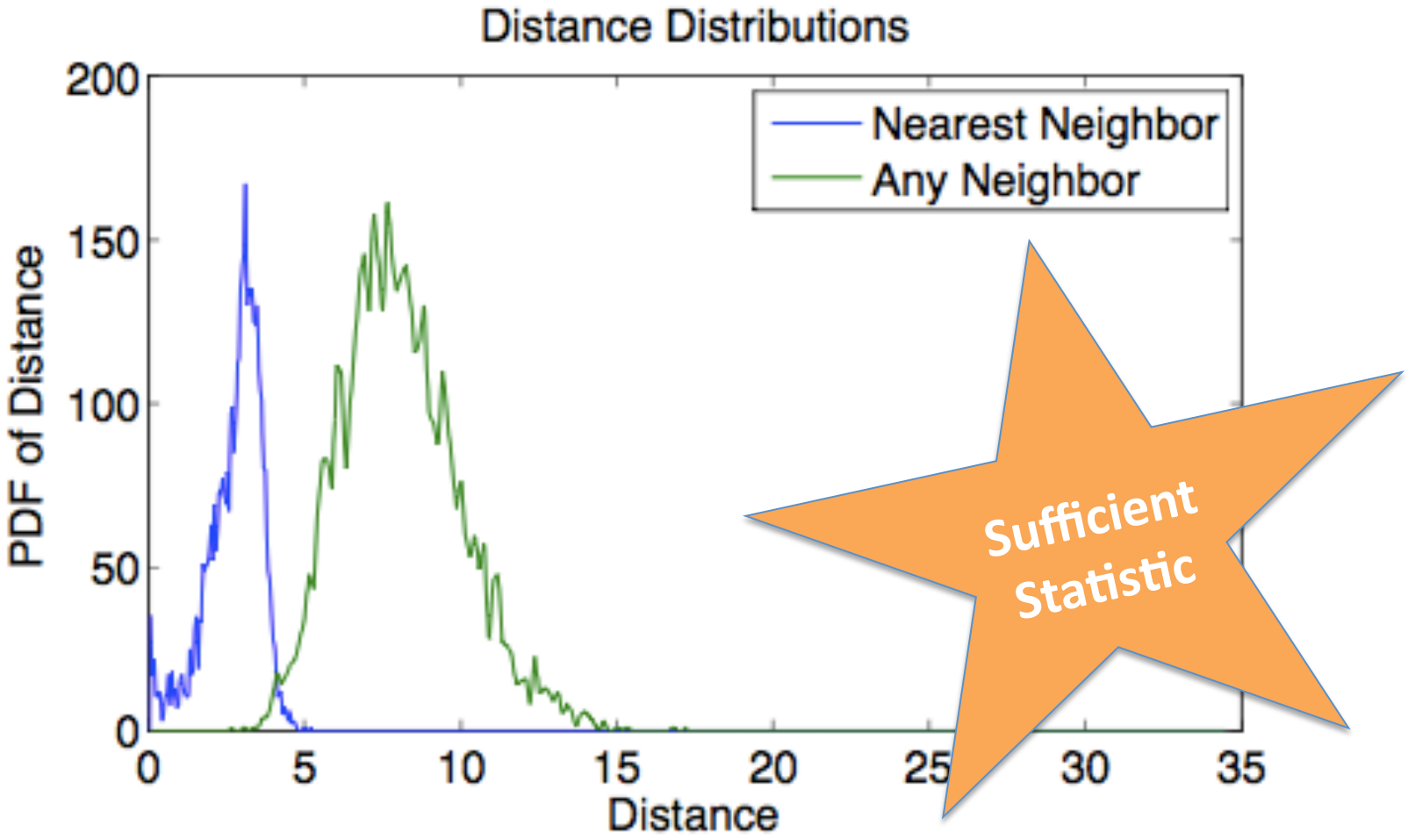
Inputs

- Distance profiles
  - D_{nn} and D_{any}
- Number of points
  - For false alarms

- Allowable error (δ)
  - Return true NN with probability of 1-δ
- Multiprobe radius (r)
  - Hamming distance
- CPU Costs
  - U_{hash}: Cost to calculate projections
  - U_{check}: Cost to check distance

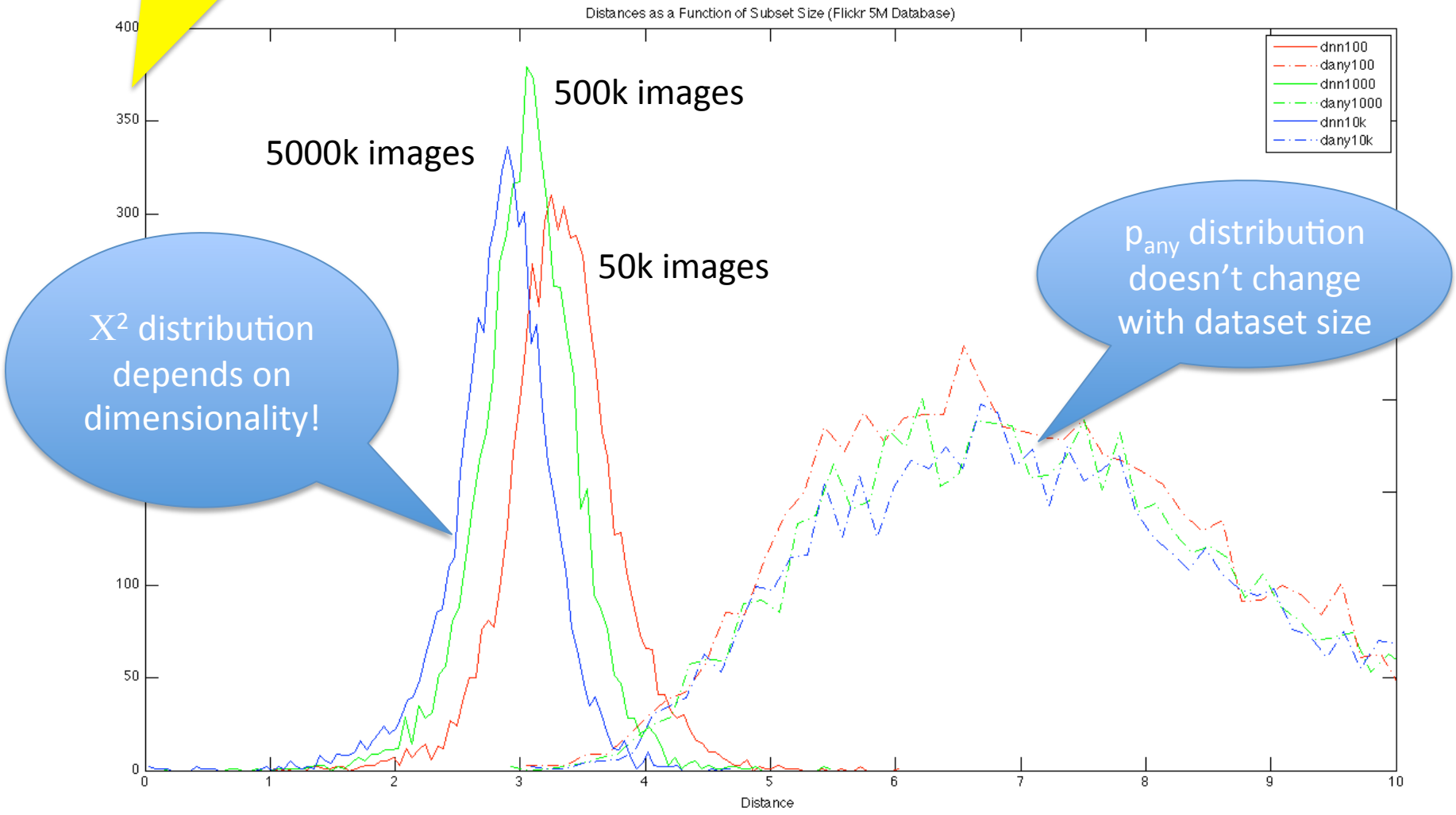
Distance Profile

- Empirical Estimate



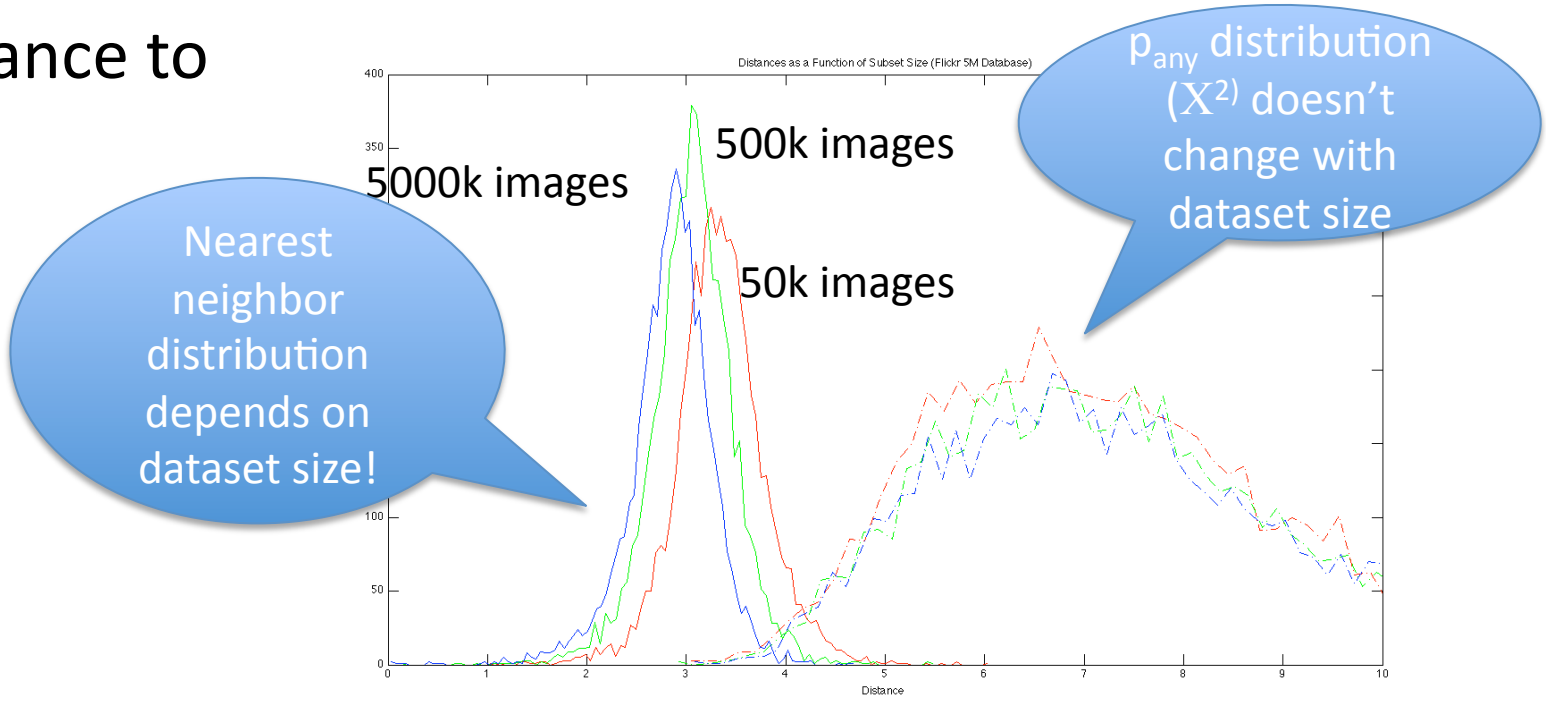
Digression

NN vs. Dataset Size



Notes

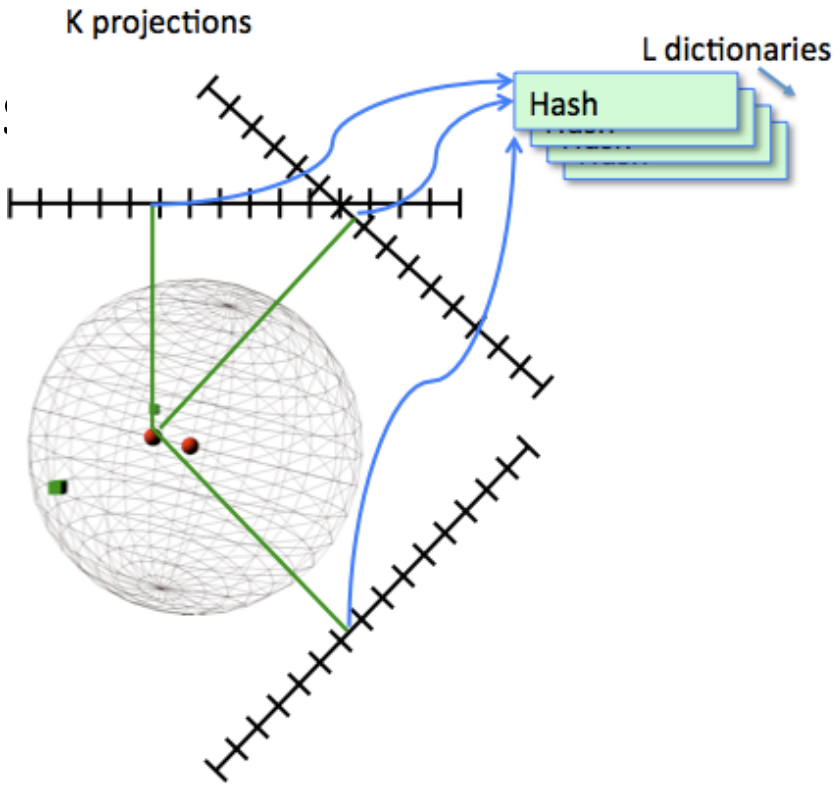
- Pure nearest neighbor
  - No noise
- Distance profile can be anything
  - Distance to 42nd



Outputs

- Parameters for minimum CPU time

- W – Bucket size
- K – Number of projection
- L – Number of tables



Cost Model

- Minimize CPU time
  - $T_s = L\, U_{hash} + |Candidates| U_{check}$
- Maintain performance
  - $P(error) < \delta$
- Memory
  - $O(L\, N)$

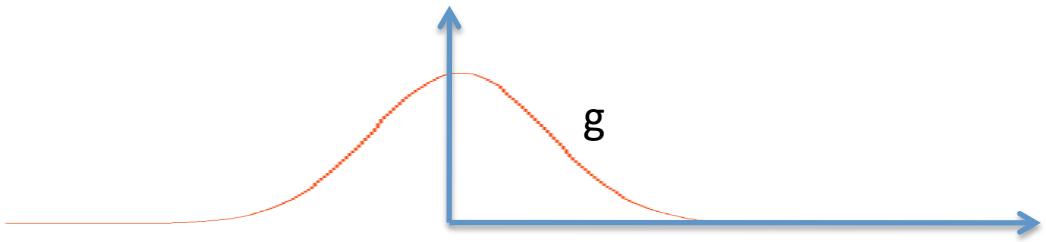
Projection Operations

- For query d form dot product

$$bucket_i = \left\lfloor \vec{d} \cdot \vec{g} / w + b \right\rfloor$$

Floor (quantization)

  - G is a vector of Gaussians N(0,1)
  - b is uniform random
- Any linear combination of Gaussians is a Gaussian



Calculate Projection Probabilities

- Form projection
  - d·g
  - d is the query vector

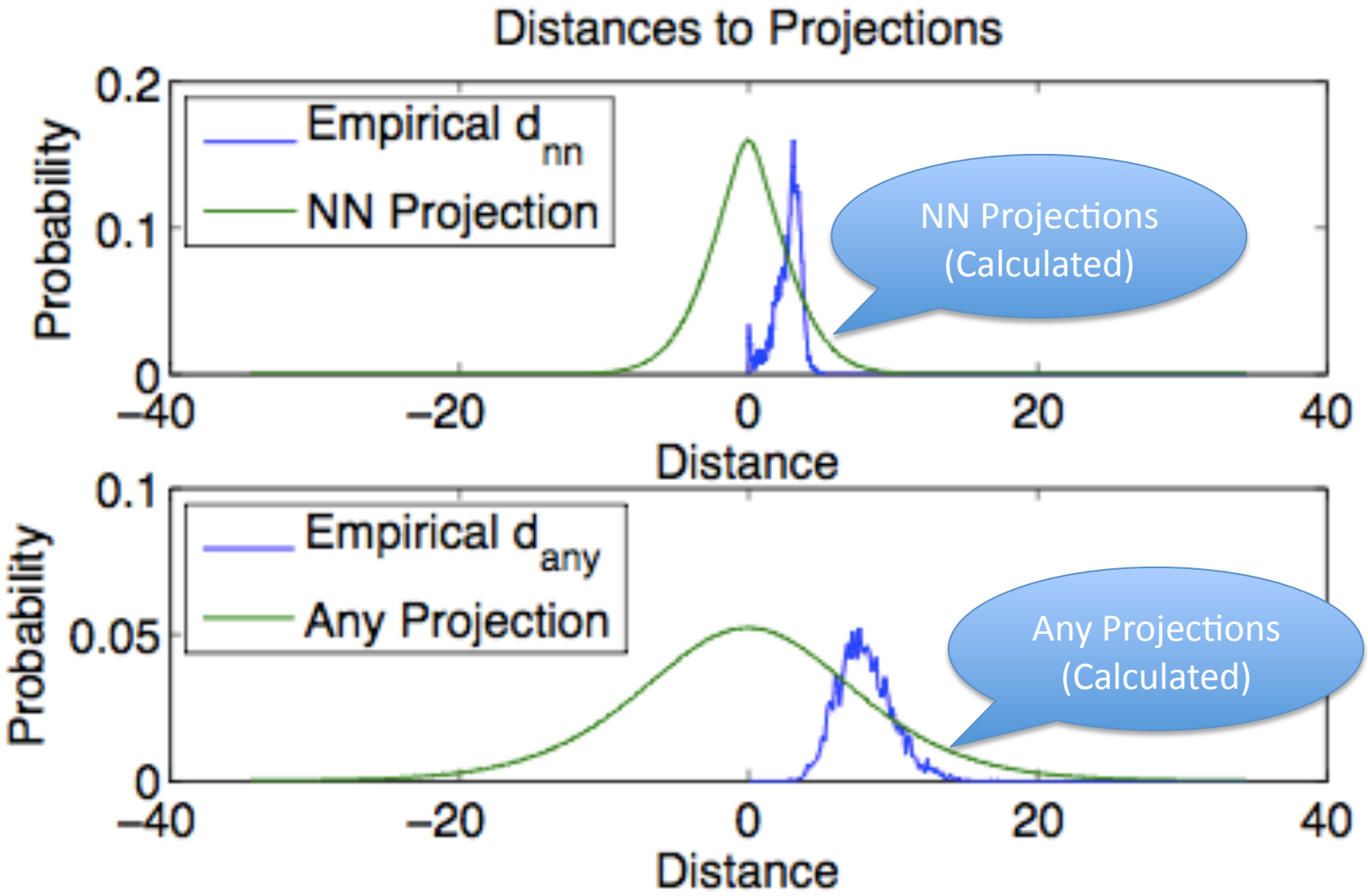
- PDF Multiplication

$$g_{nn} = d_{nn} \star N(0,1) \text{ and } g_{any} = d_{any} \star N(0,1)$$

$$g_{nn}(y) = \int N(0,1)(x) d_{nn}(\frac{y}{x}) \frac{1}{x} dx,$$

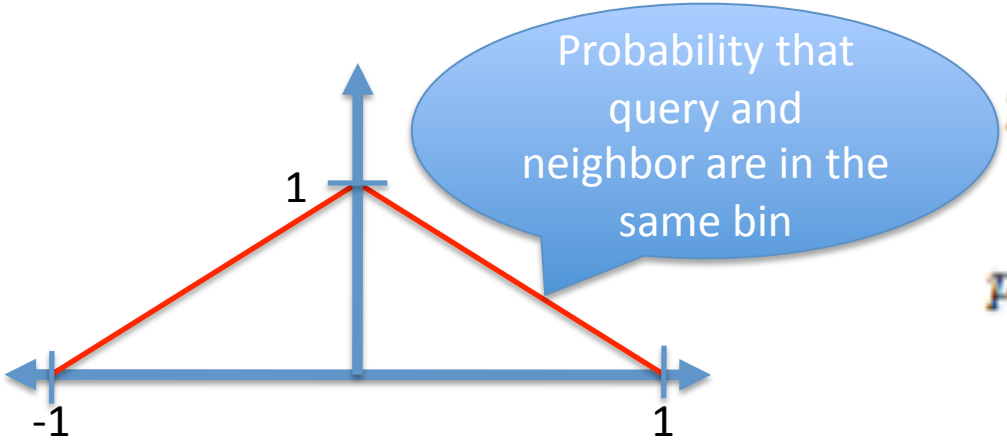
$$g_{any}(y) = \int N(0,1)(x) d_{any}(\frac{y}{x}) \frac{1}{x} dx.$$

Projection Histograms



Calculate Bucket Probabilities

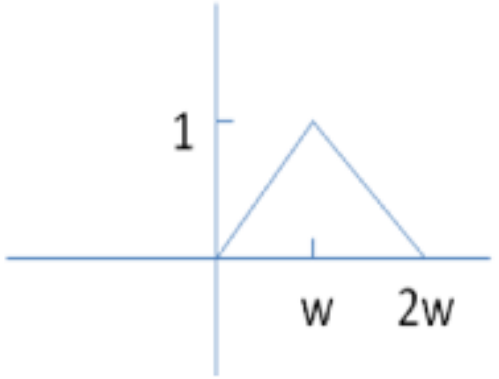
- Integrate over triangular window



$$p_{nn}(w) = \int g_{nn}(x)\Delta_x(x)dx$$

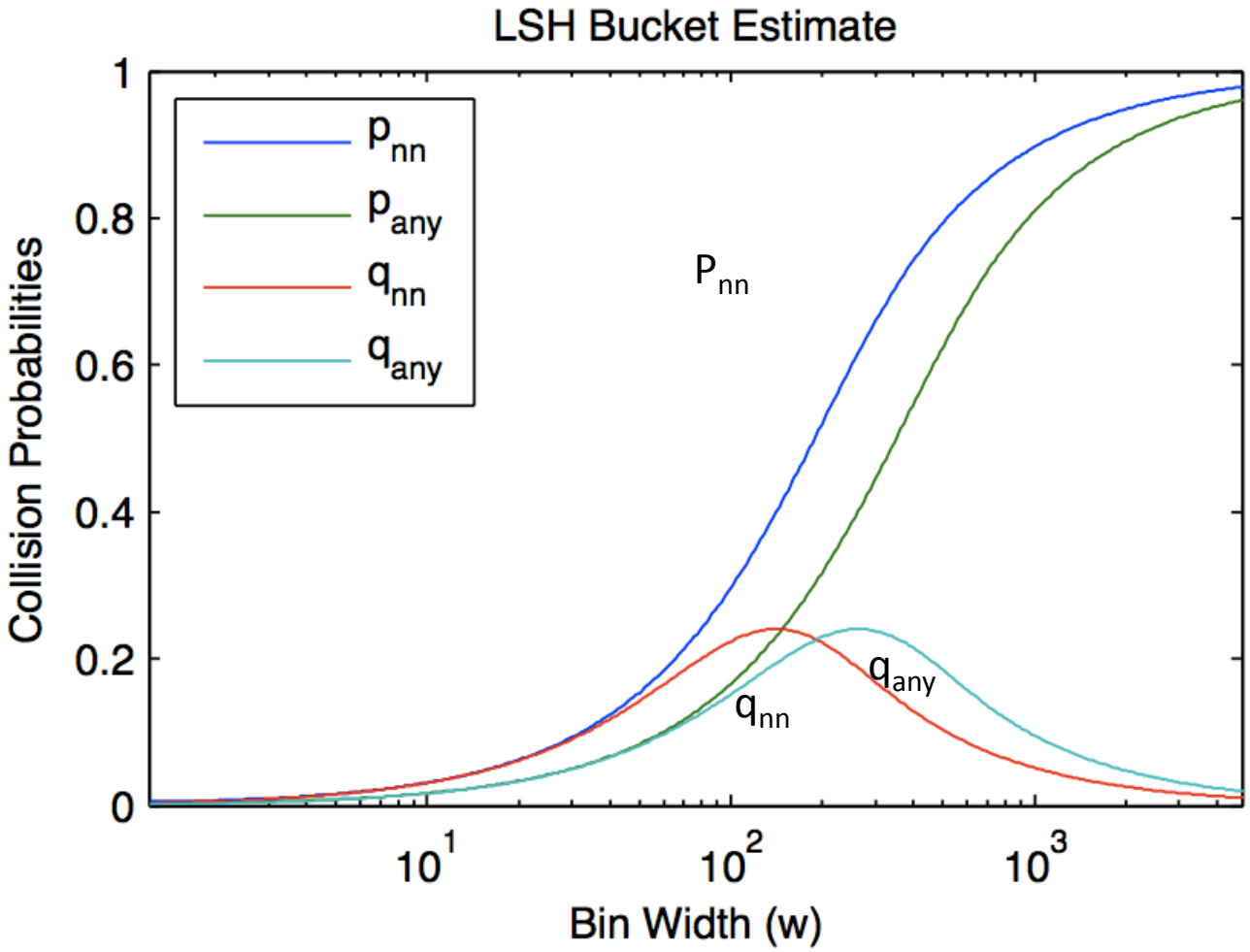
$$p_{any}(w) = \int g_{any}(x)\Delta_x(x)dx$$

- Similarly for adjacent bin

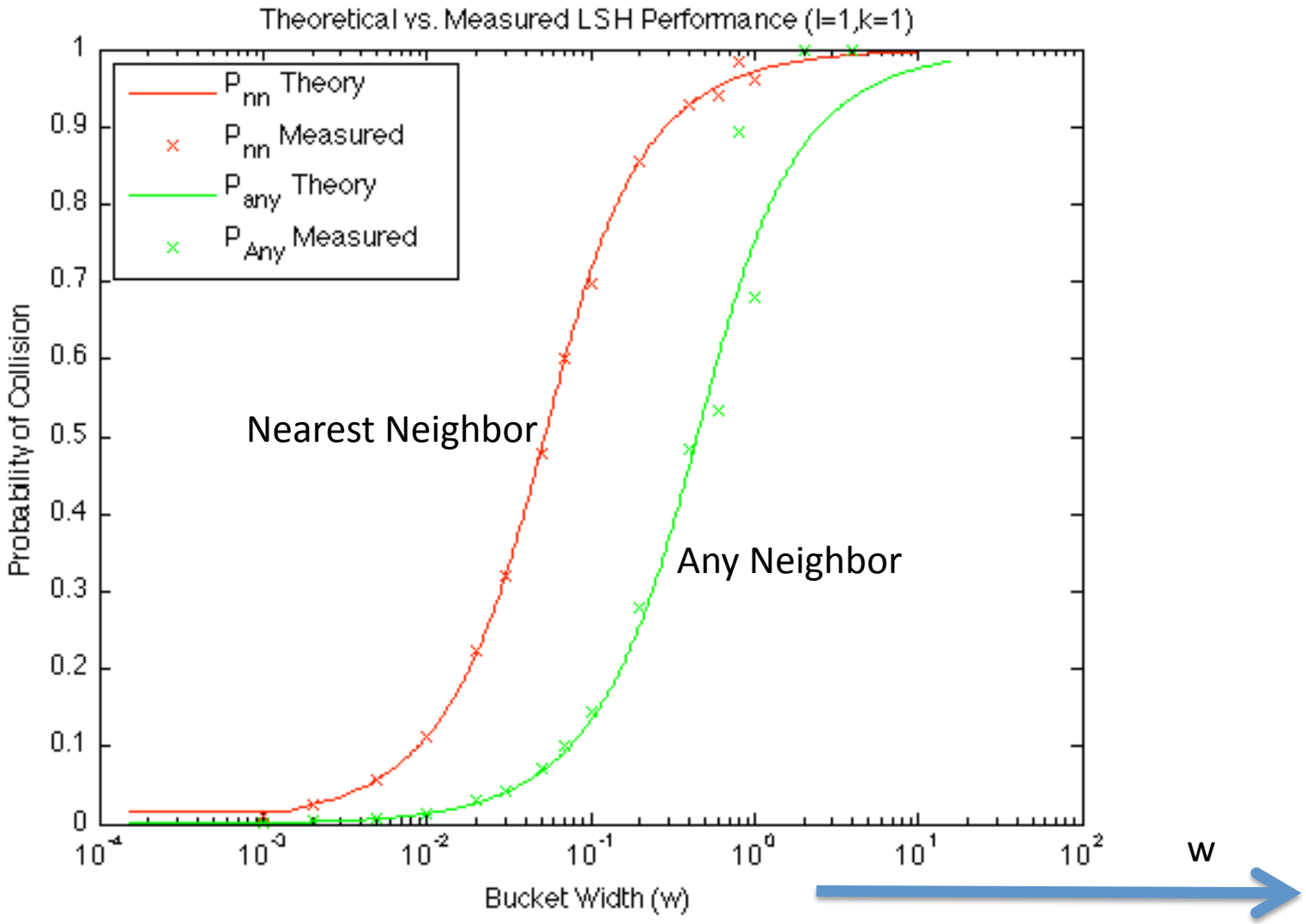


Bucket Collisions

- Same bucket
- Adjacent bucket



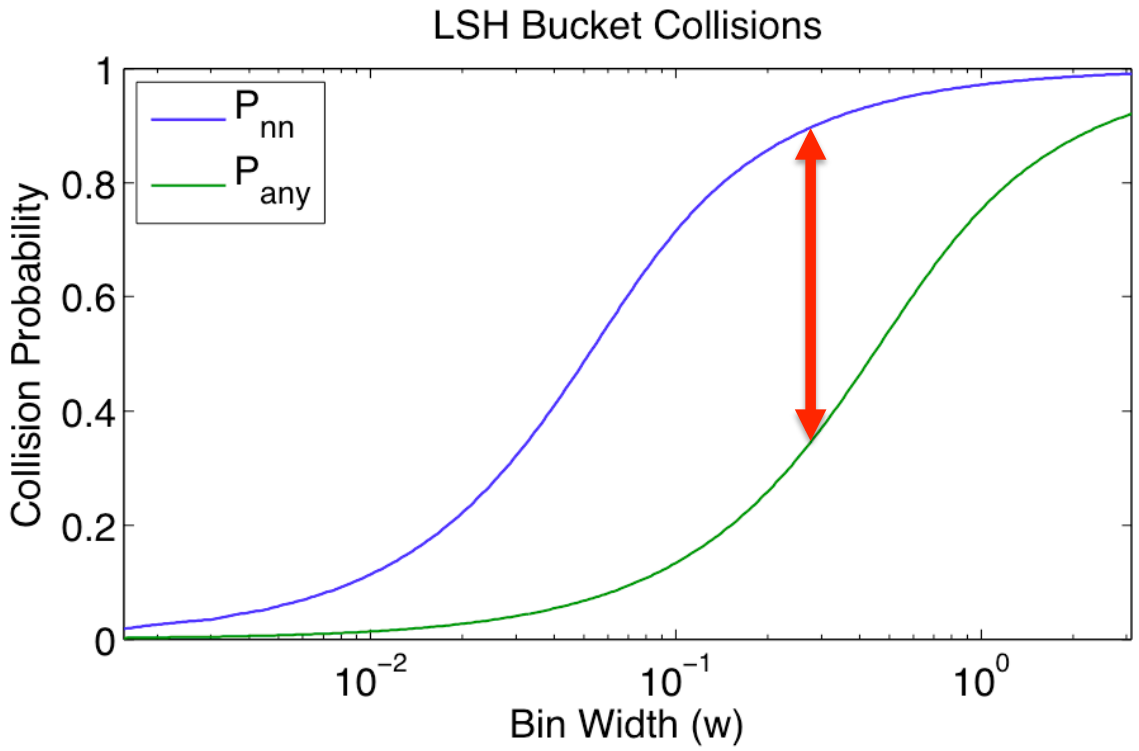
Collision probabilities



Optimization – Step 1

- Pick w (bin width)
- Maximize difference when quantizing

- Argmax over w $\log(P_{nn})/\log(P_{any})$



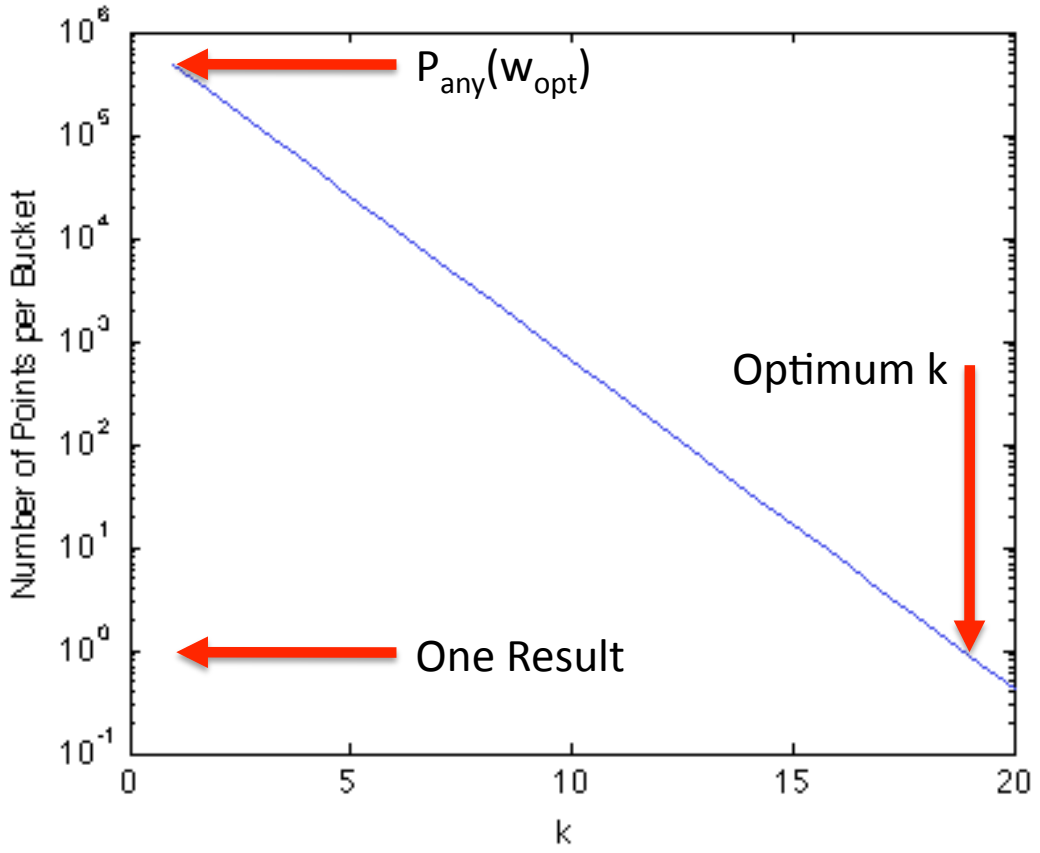
Optimization – Step 2

- Pick k (projections per bucket)

  Want $N*P_{any}(w_{opt})^k = 1$

- Get at least one hit per bucket

- One hit from **any** neighbor

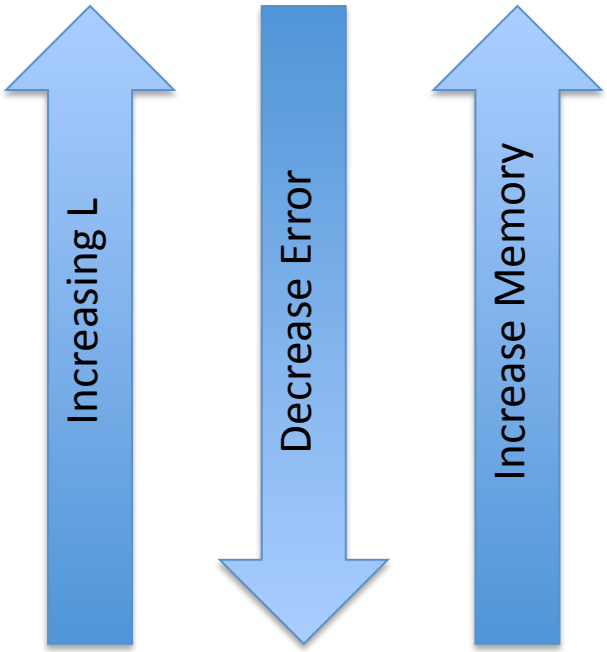


Optimization – Step 3

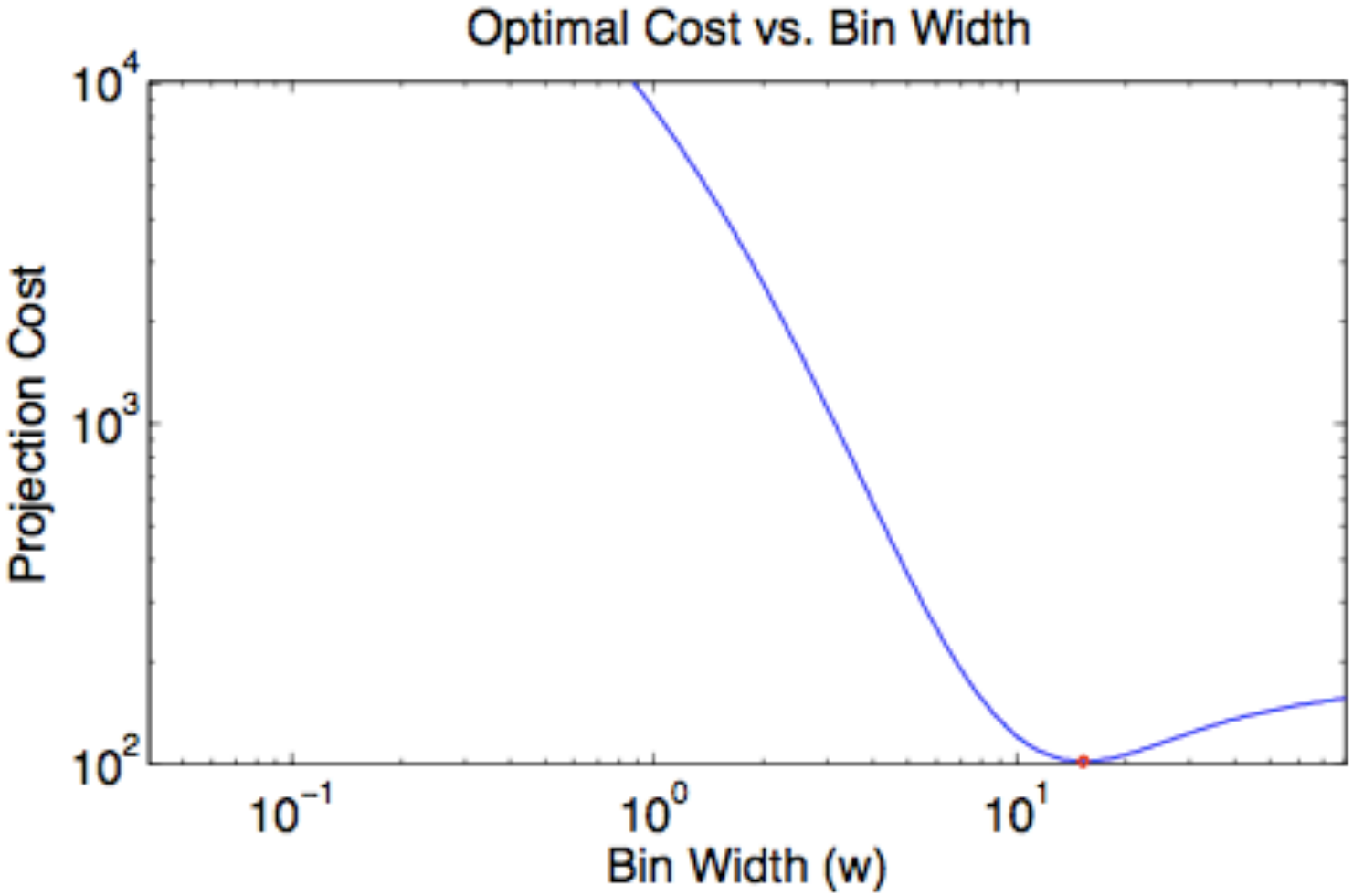
- Pick L (number of buckets)
- Probability of failure:

$$\delta = [1\text{-}P_{nn}(w)^k]^L$$

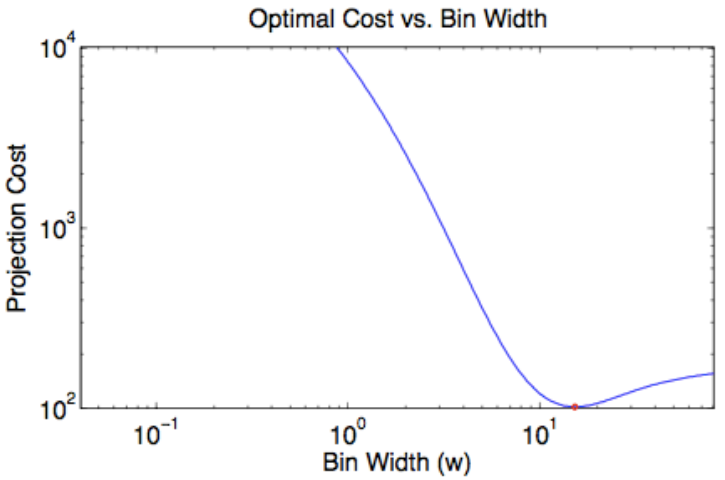
- Pick L so δ is as small as desired
- Memory footprint is N*L



Cost Function



Outcome



Factor of 2

Bin width goes to infinity!

- Do all this work
  ➔ nearly binary bins

Wow!!!

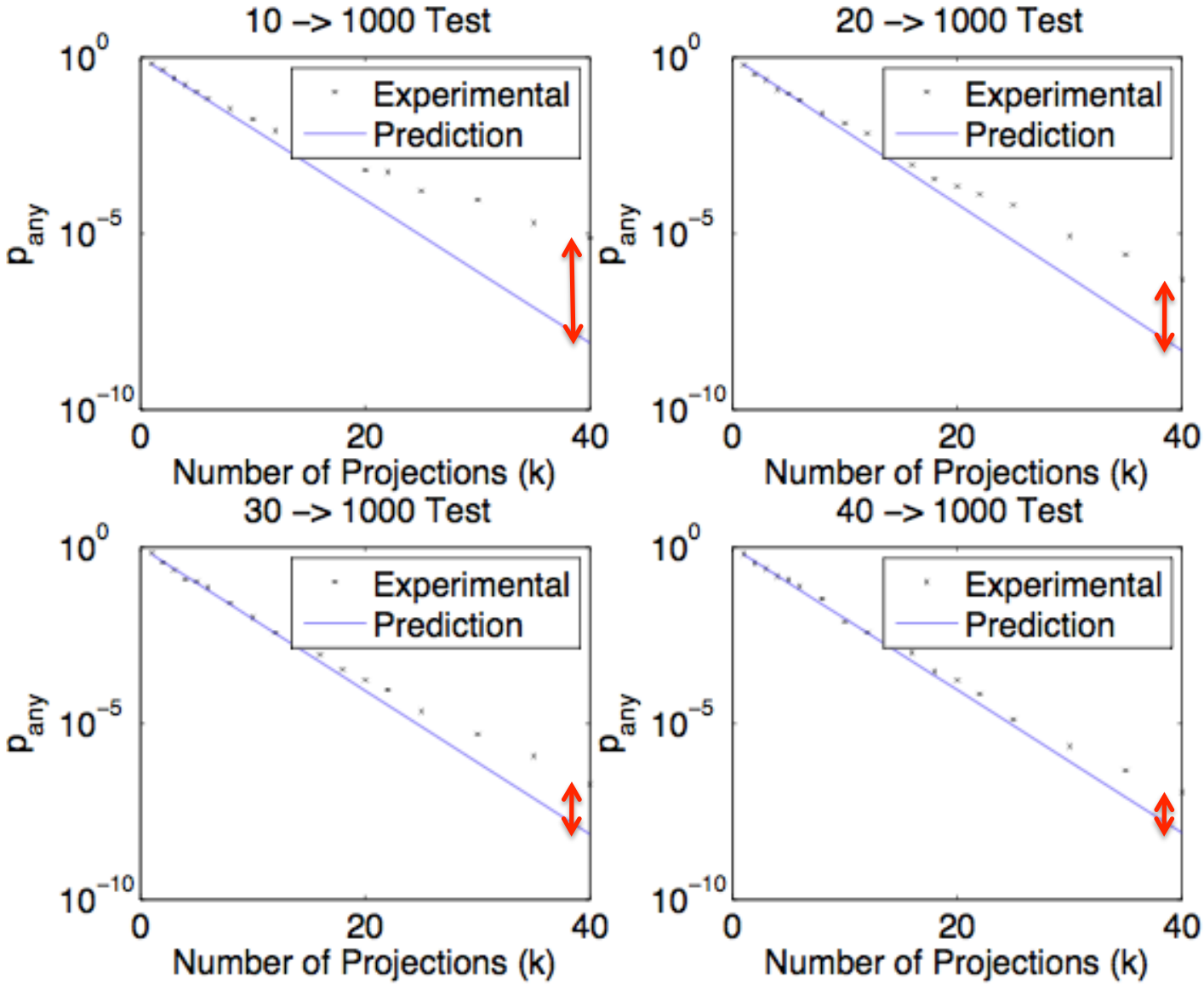
Experiments

- Checking the model
  - K
  - δ
  - Number of results
  - CPU Time
  - Multiprobe
- Extending the Model
  - Optimum P_{any}
  - Number of Candidates
  - Cost Changes
  - Multiprobe

Python and Matlab implementation on GitHub

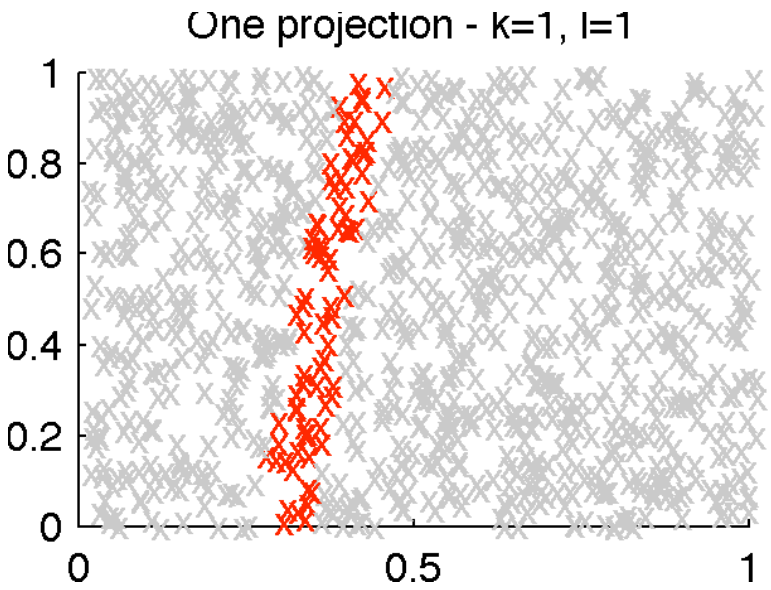
Extrapolate from calculated distance profiles

Performance vs. K

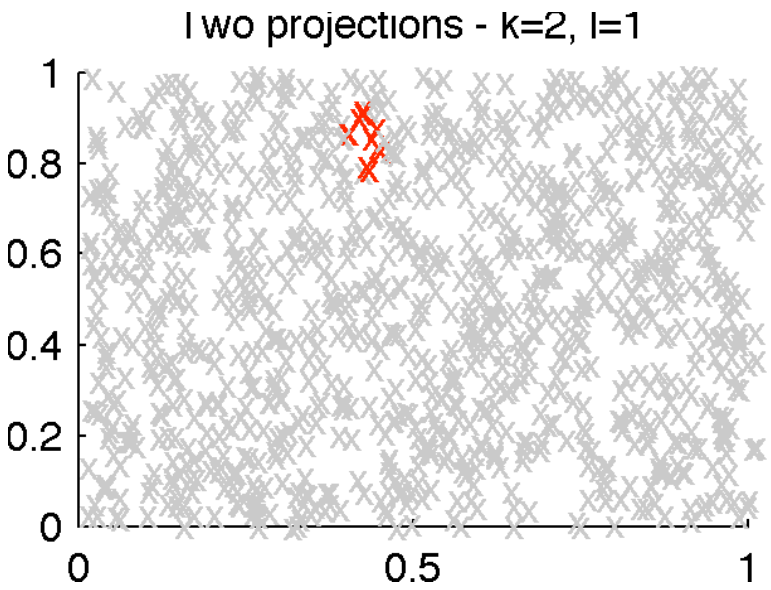


Independent Projections?

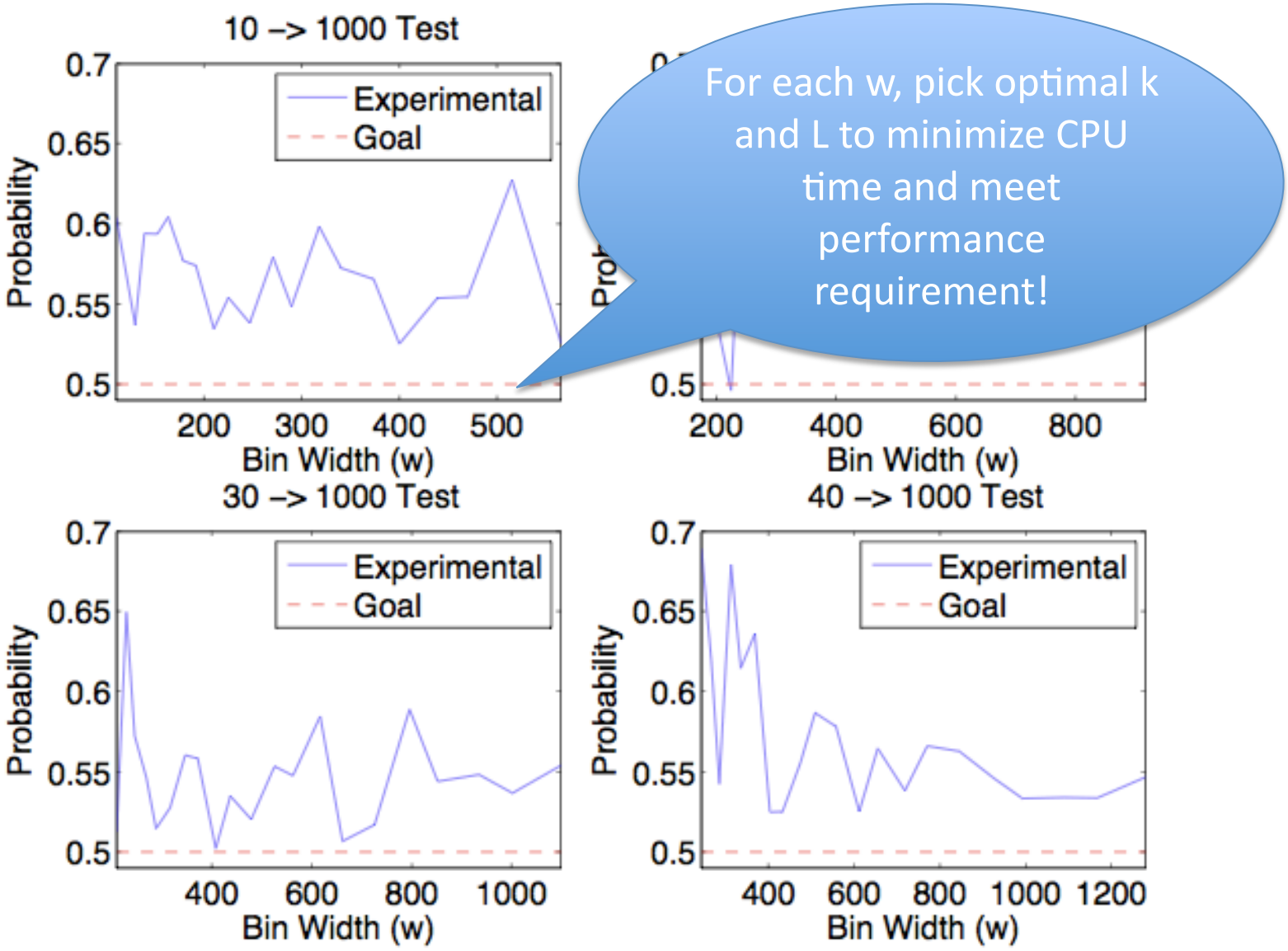
k=1



k=2

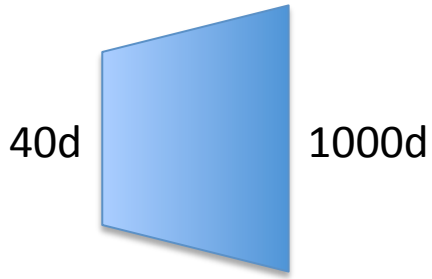


Error Performance

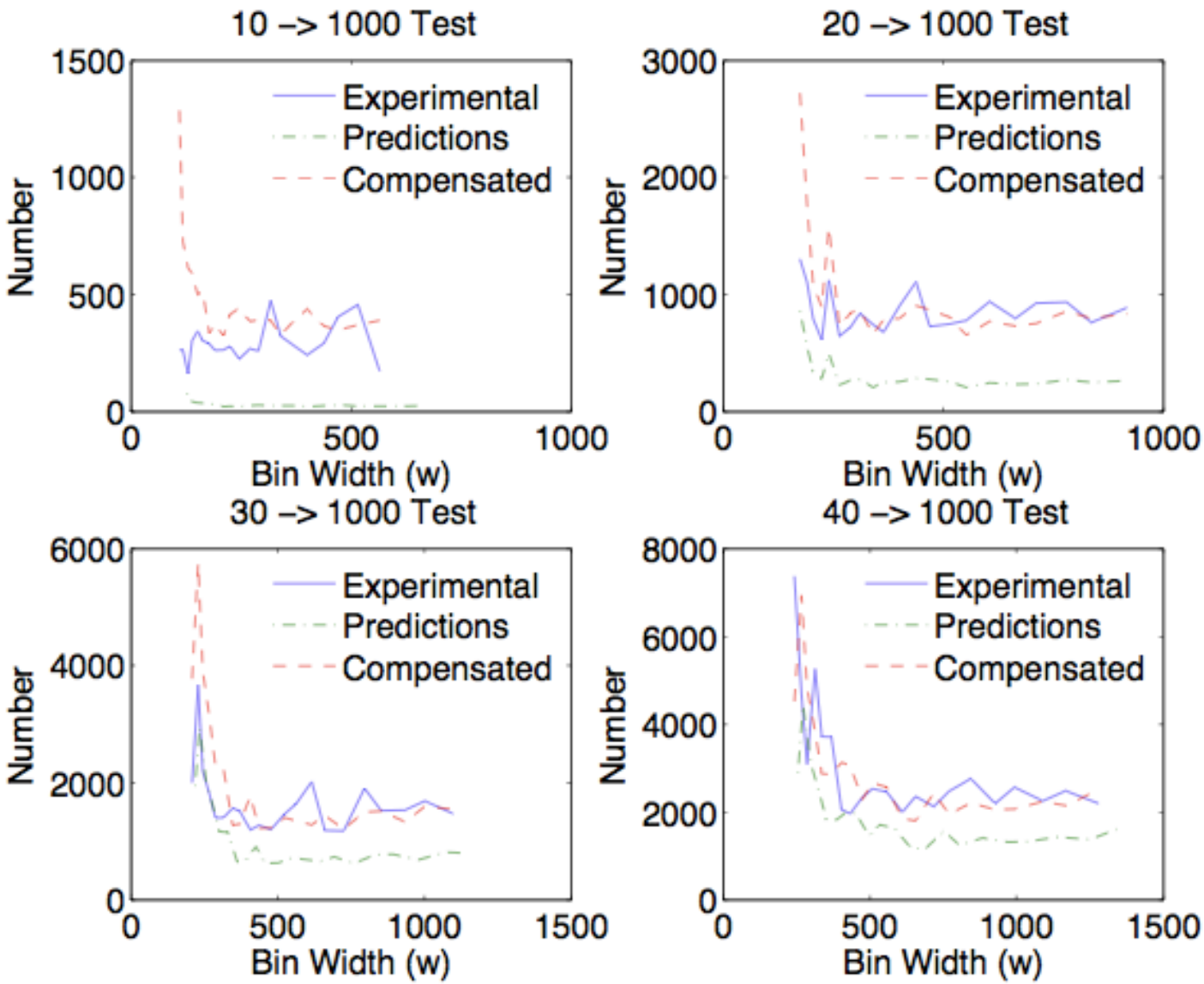


Synthetic Data

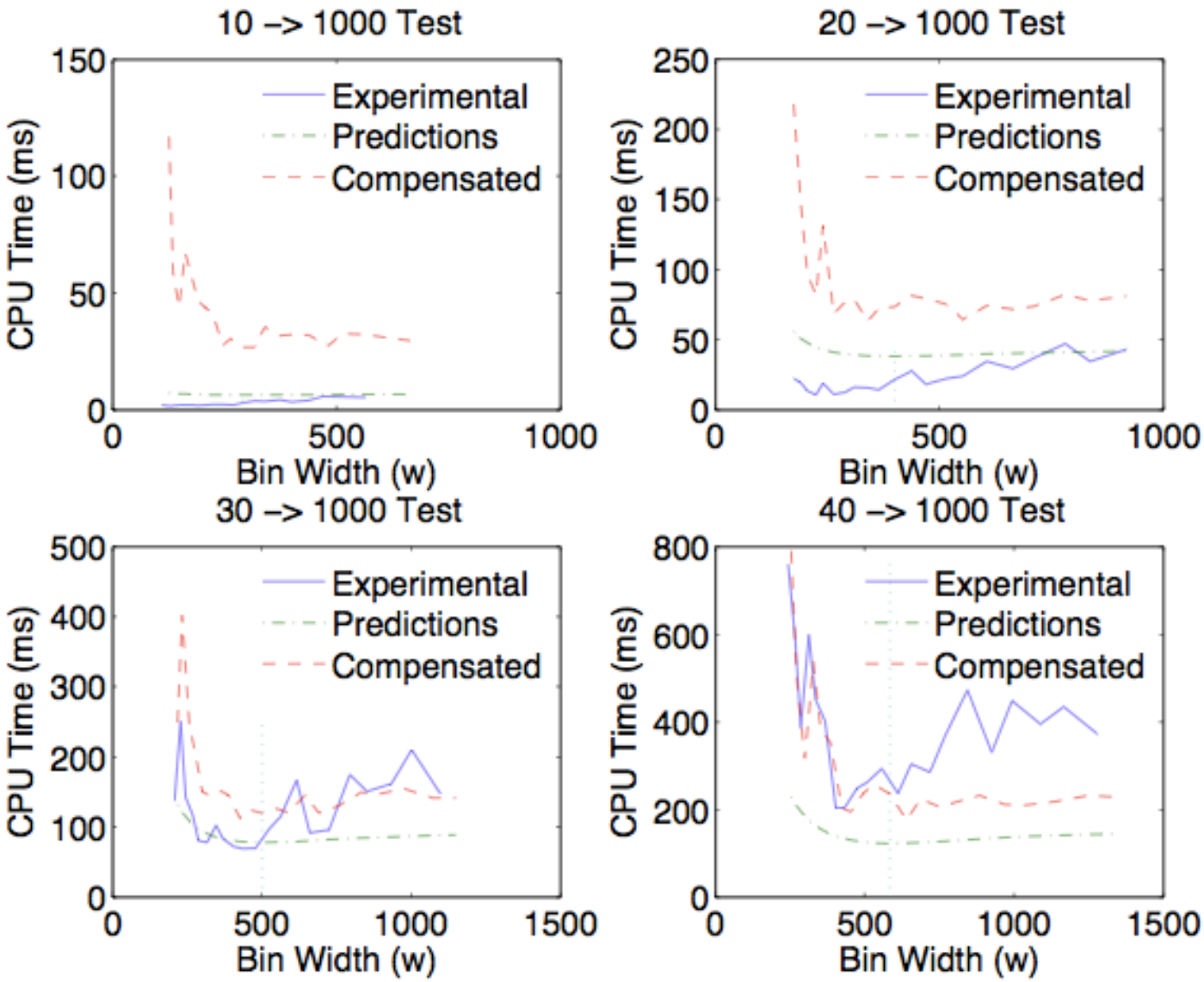
- 10-40 dimensional data
  - Uniform random
- Expand to 1000 dimensions
  - Fixed random matrix multiply
- Reasons
  - Different distance profiles
  - Controlled dimensionality
  - Better timing measures (NumPy)



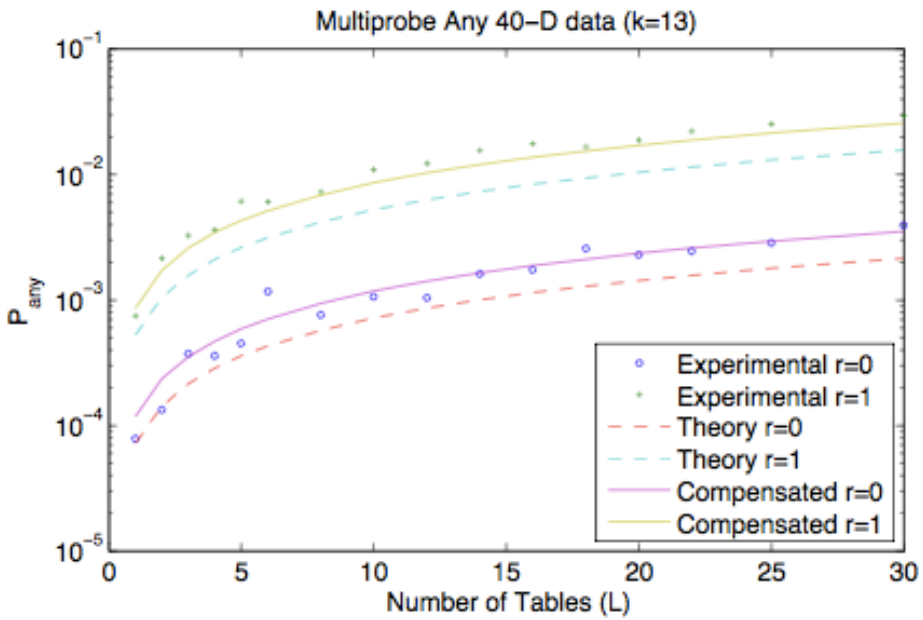
Number of Candidates

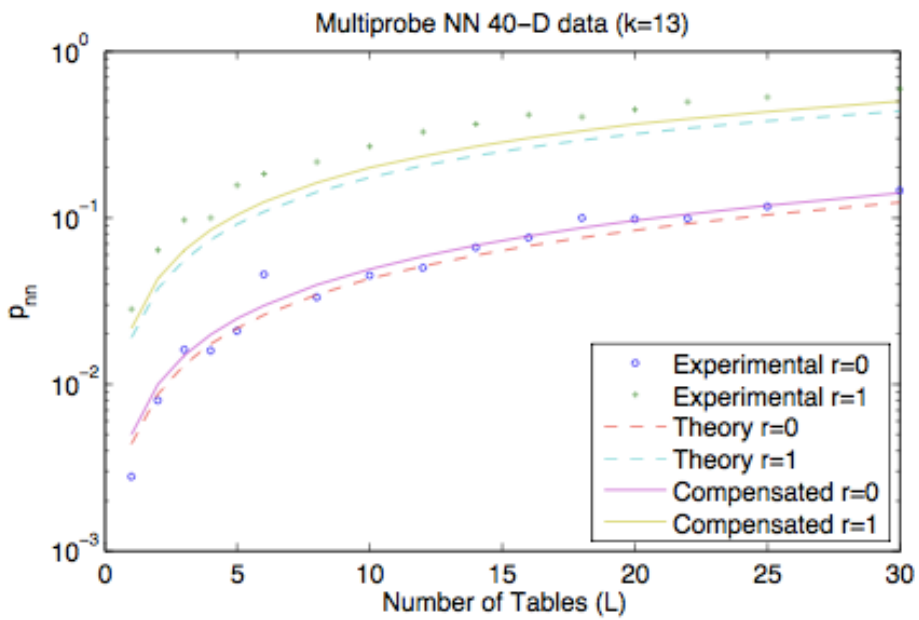


CPU Time


10 -> 1000 Test
20 -> 1000 Test
30 -> 1000 Test
40 -> 1000 Test
CPU Time (ms)
Bin Width (w)
Experimental
Predictions
Compensated
0
50
100
150
200
250
300
400
500
600
800
1000
1500

Multiprobe

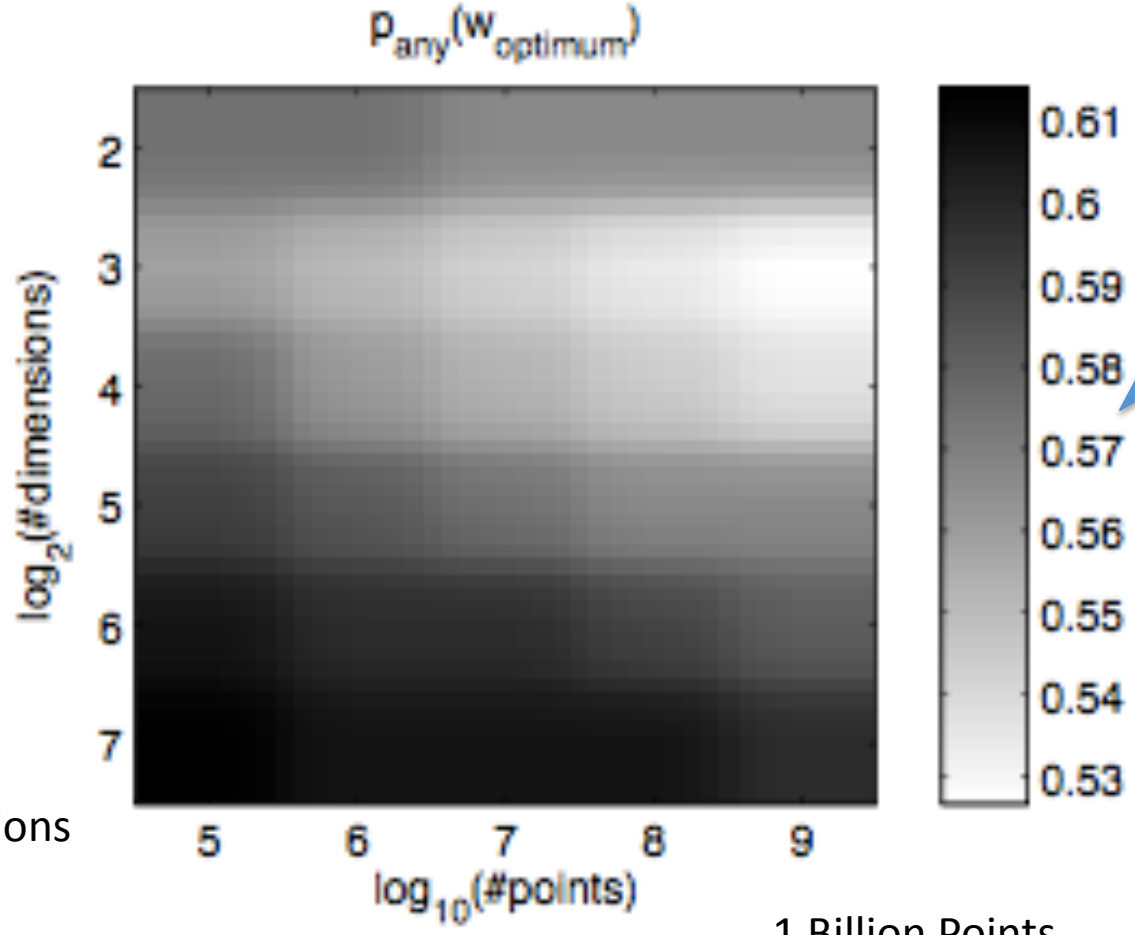

Multiprobe Any 40−D data (k=13)
P_{any}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
0
5
10
15
20
25
30
Number of Tables (L)
Experimental r=0
Experimental r=1
Theory r=0
Theory r=1
Compensated r=0
Compensated r=1


Multiprobe NN 40−D data (k=13)
p_{nn}
10^{0}
10^{-1}
10^{-2}
10^{-3}
0
5
10
15
20
25
30
Number of Tables (L)
Experimental r=0
Experimental r=1
Theory r=0
Theory r=1
Compensated r=0
Compensated r=1

Extensions

- Very large datasets and dimensionalities
- Calculate P_{any}
  - χ^2 gives difference between Gaussians
- Calculate P_{nn}
  - Based on rank statistics

Optimum P_{any}

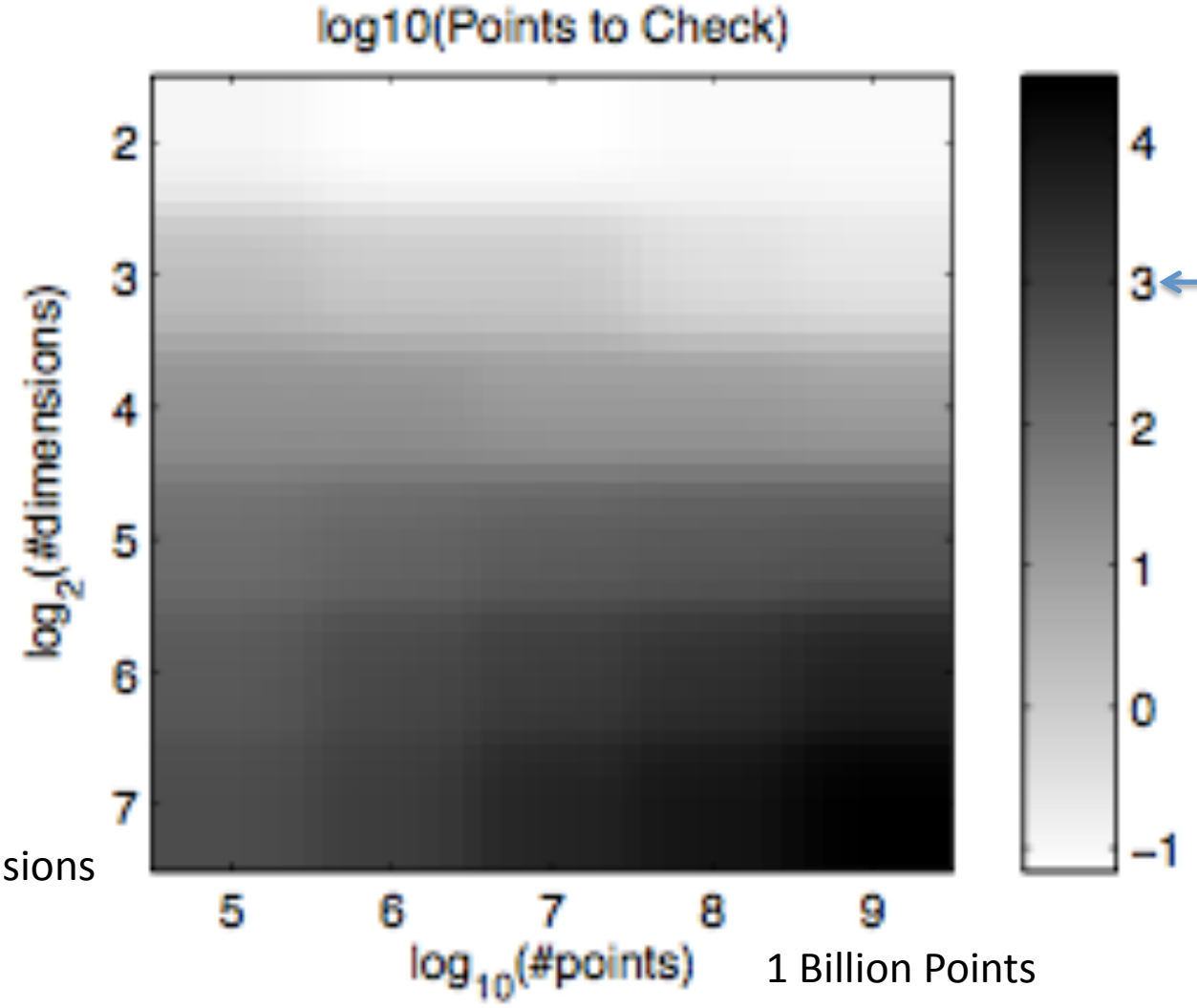


Wait! We do all this work and split a projection into 2 pieces !?!?!?!?

128 Dimensions

1 Billion Points

Number of Points to Check

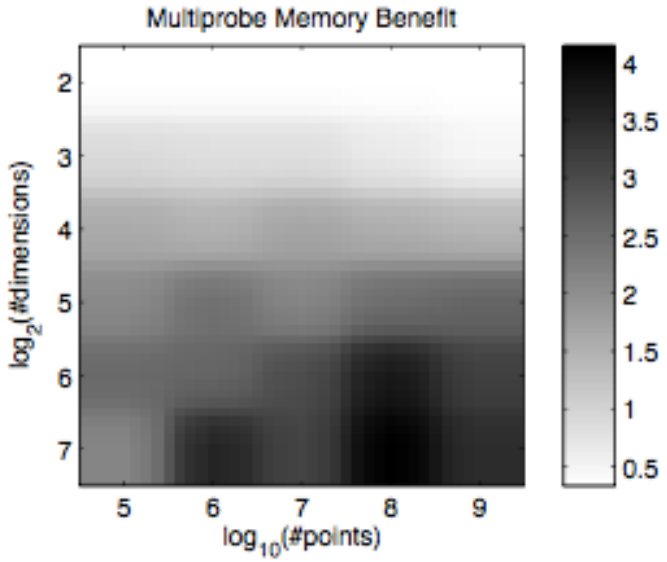


1000 Points

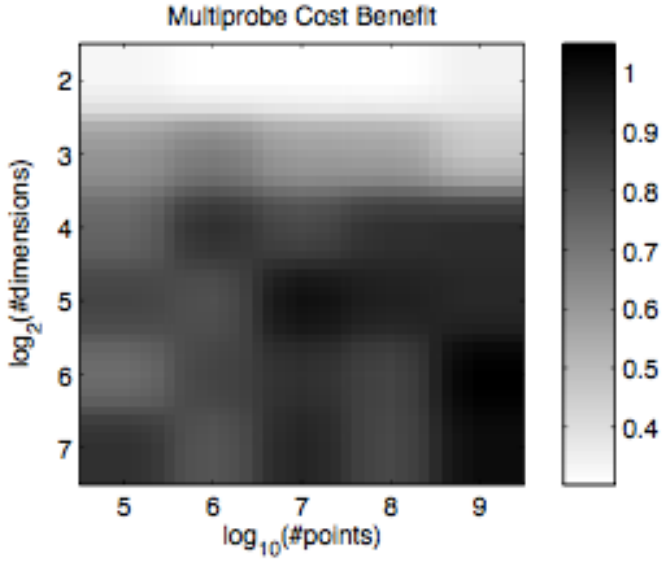
128 Dimensions

1 Billion Points

Multiprobe Benefit

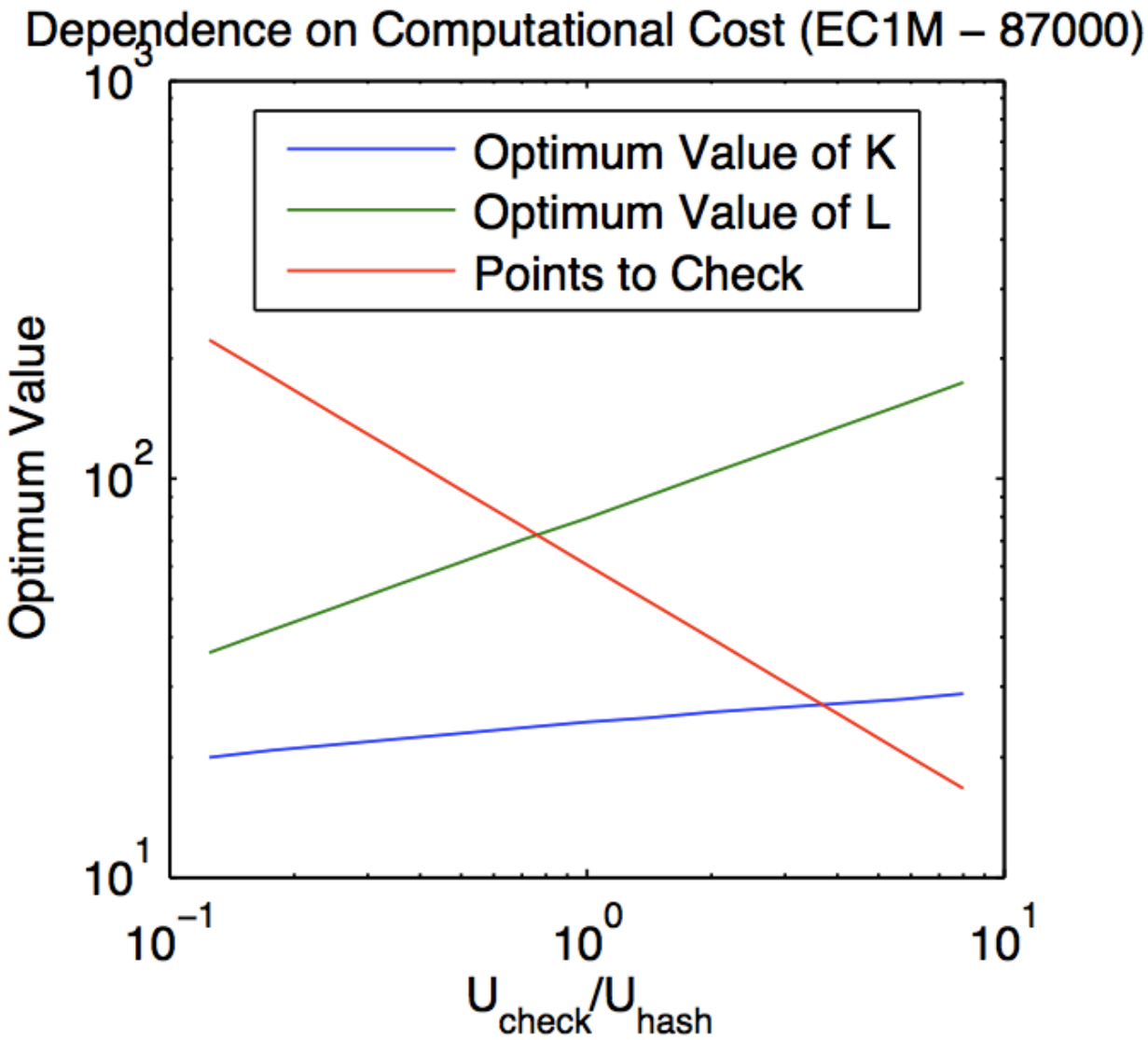


4x less memory



2x more CPU time

Cost Sensitivity



Connections

LSH

Binary

K-D Trees

Hamming Distance

- Image De-dupe
  - Spectral projection
  - Binary DCT
  - 64 or 128 bits

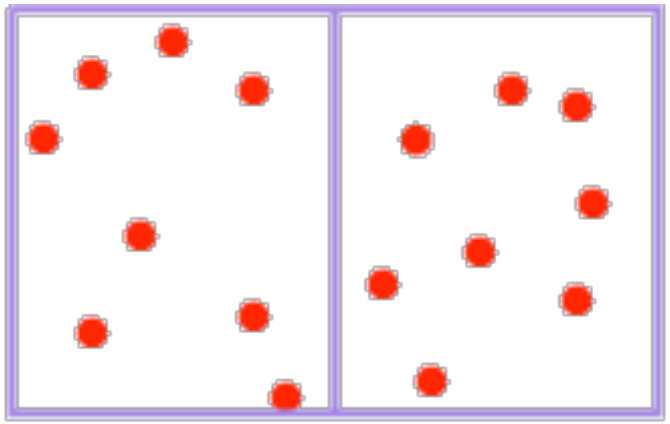
1010101011010**1**1010101011

– Tolerant to Errors

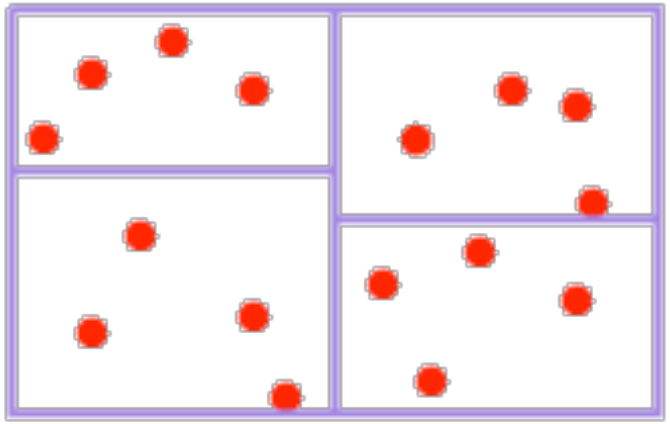
1010101011010**0**1010101011

K-D Trees

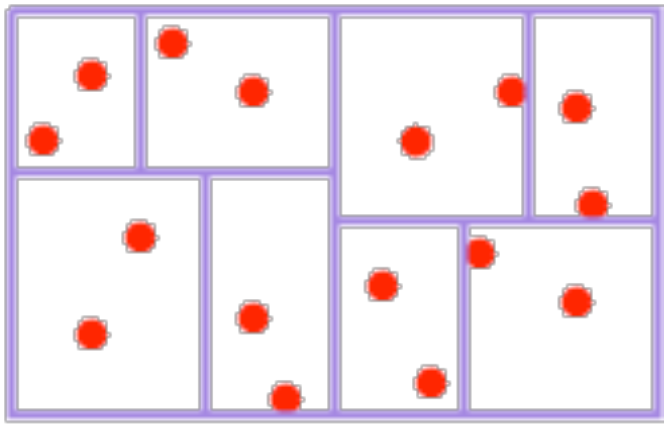
1.



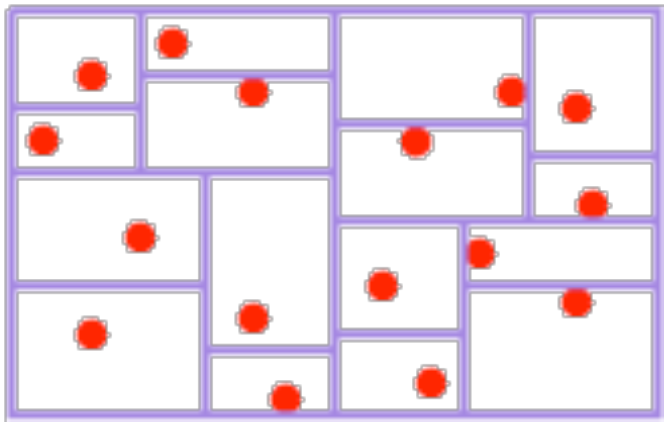
2.



3.

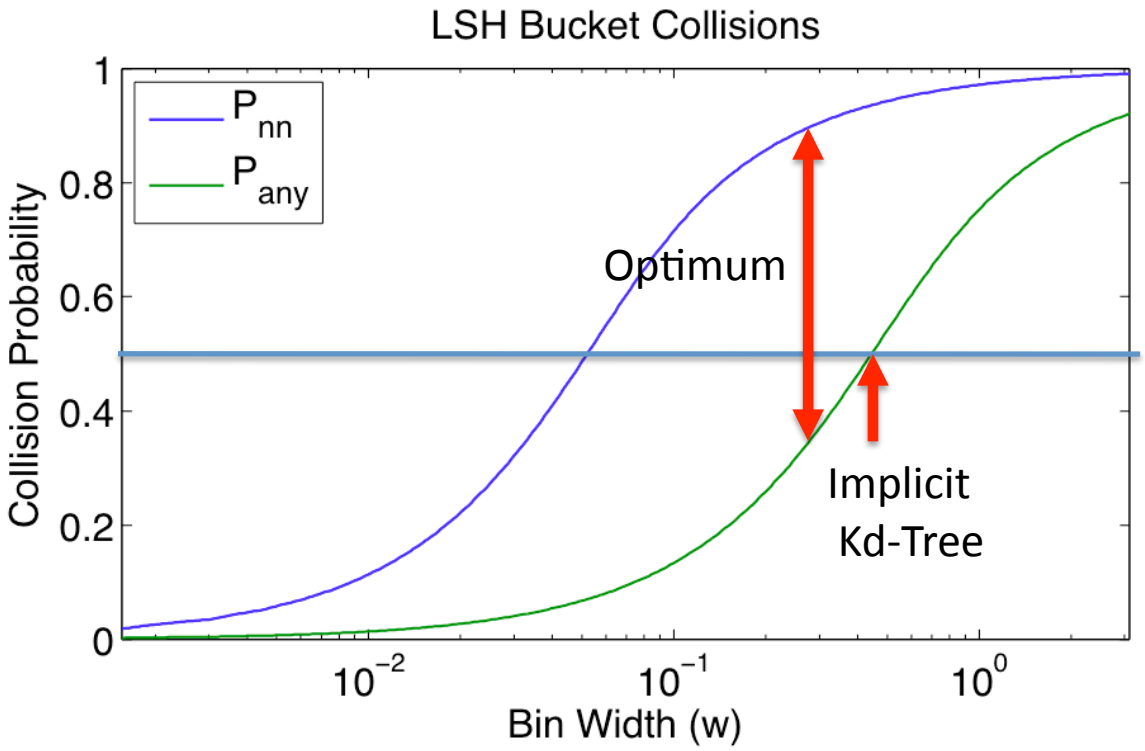


4.



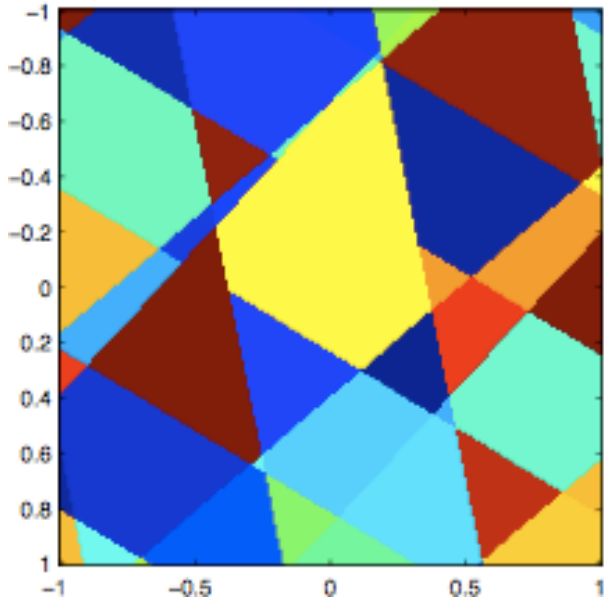
Kd-Tree Solution

- Kd-Tree assumes
  - Bin width gives 50% ratio
  - Not optimal
  - For our metric

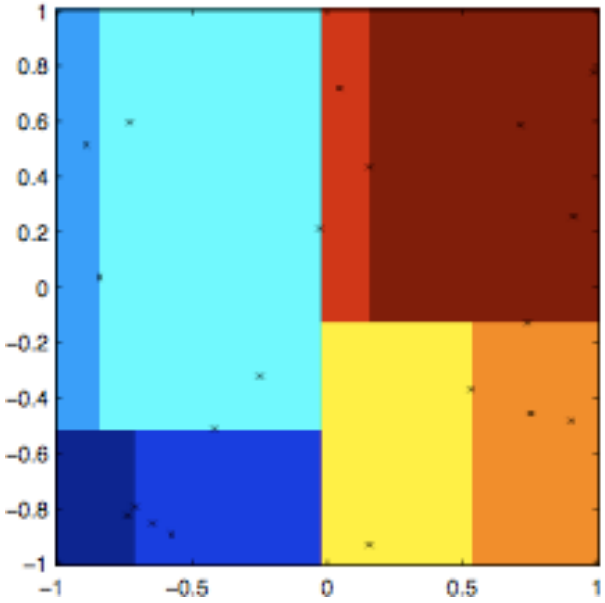


Variations

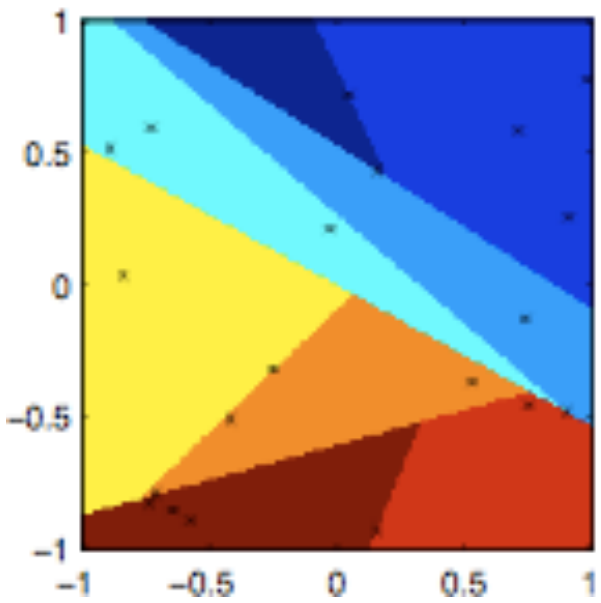
LSH (k=4)



K-d Tree (depth 4)

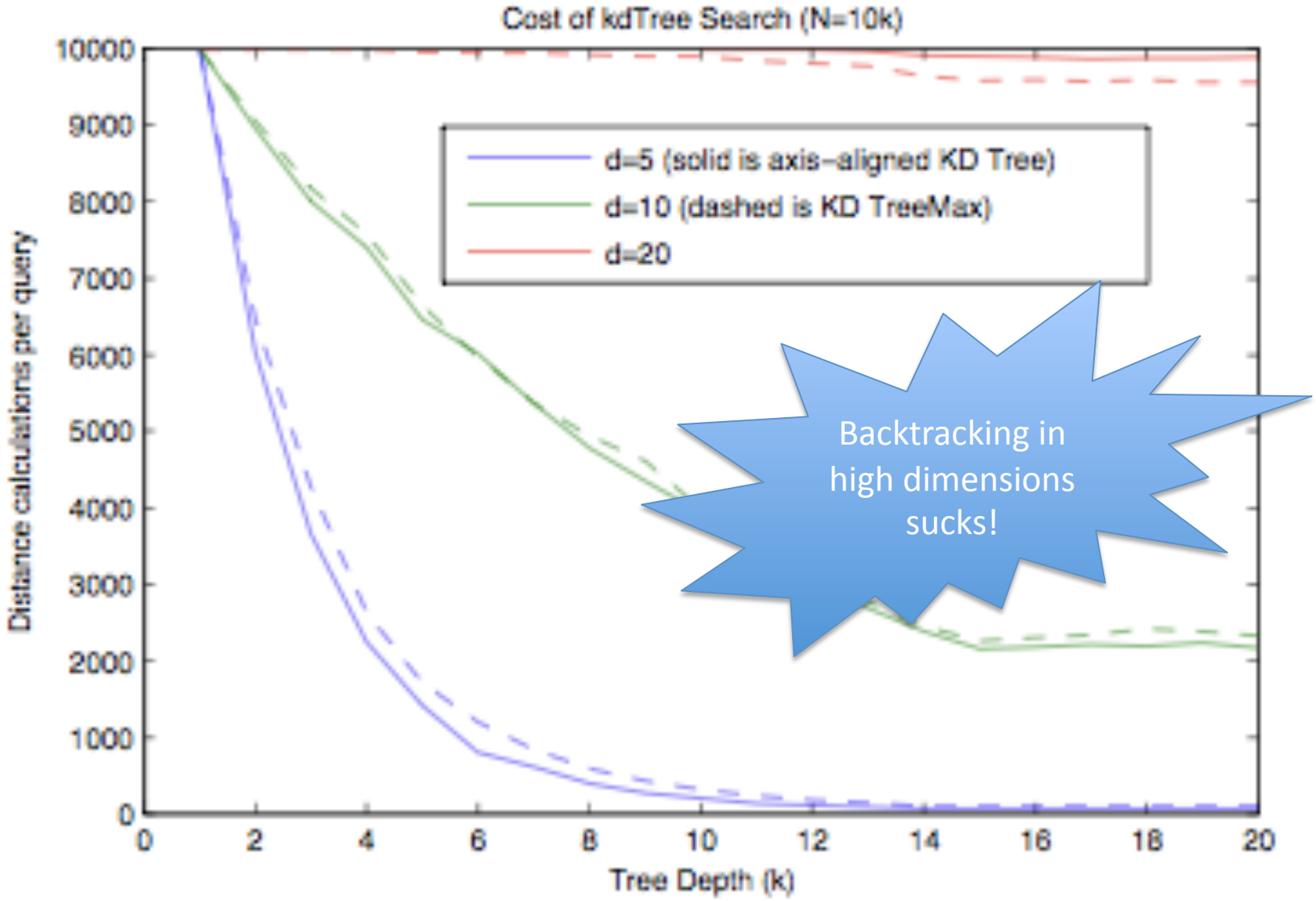


Random Projection Trees

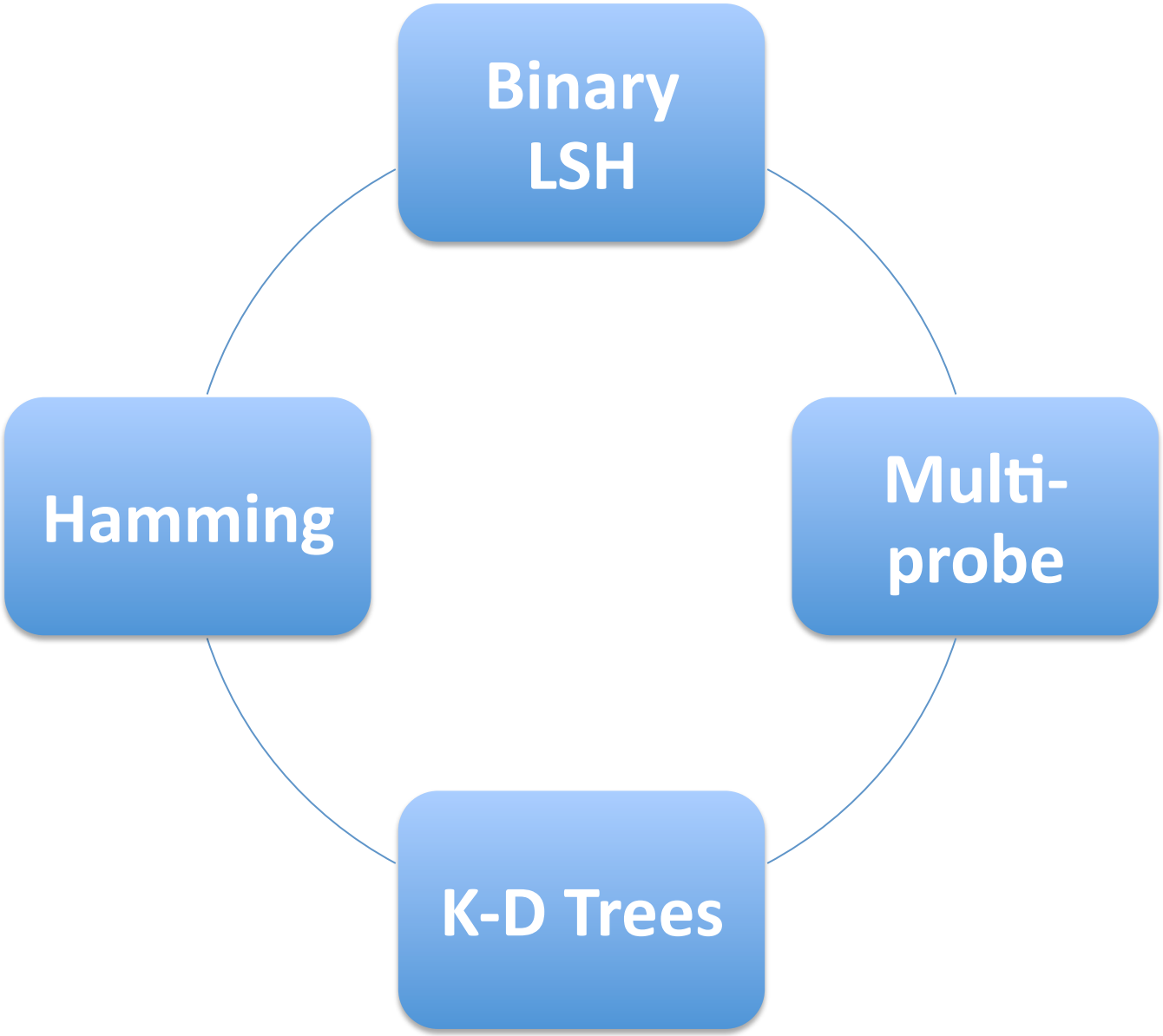


And don’t forget forests of trees

Backtracking Costs



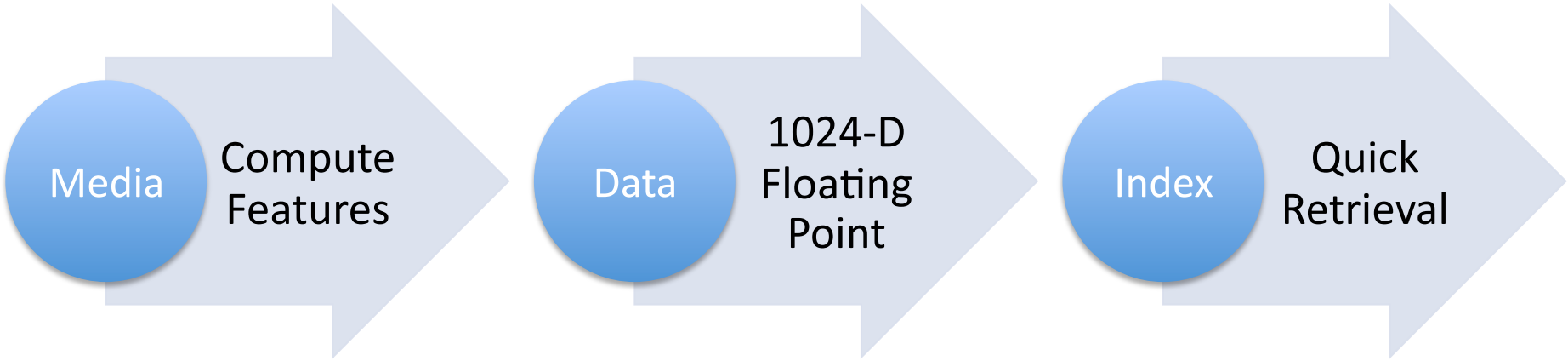
Connections



Wrap Up

- First optimization of LSH
  - Derived from distance profile
  - Independent of dimensionality
- Efficacy of Binary LSH
  - Implications for k-d trees
- Future
  - Extend our analysis to other variations

Thank You!



For more information: malcolm@ieee.org