The Inverted Multi-Index

Victor Lempitsky

Skolkovo Tech

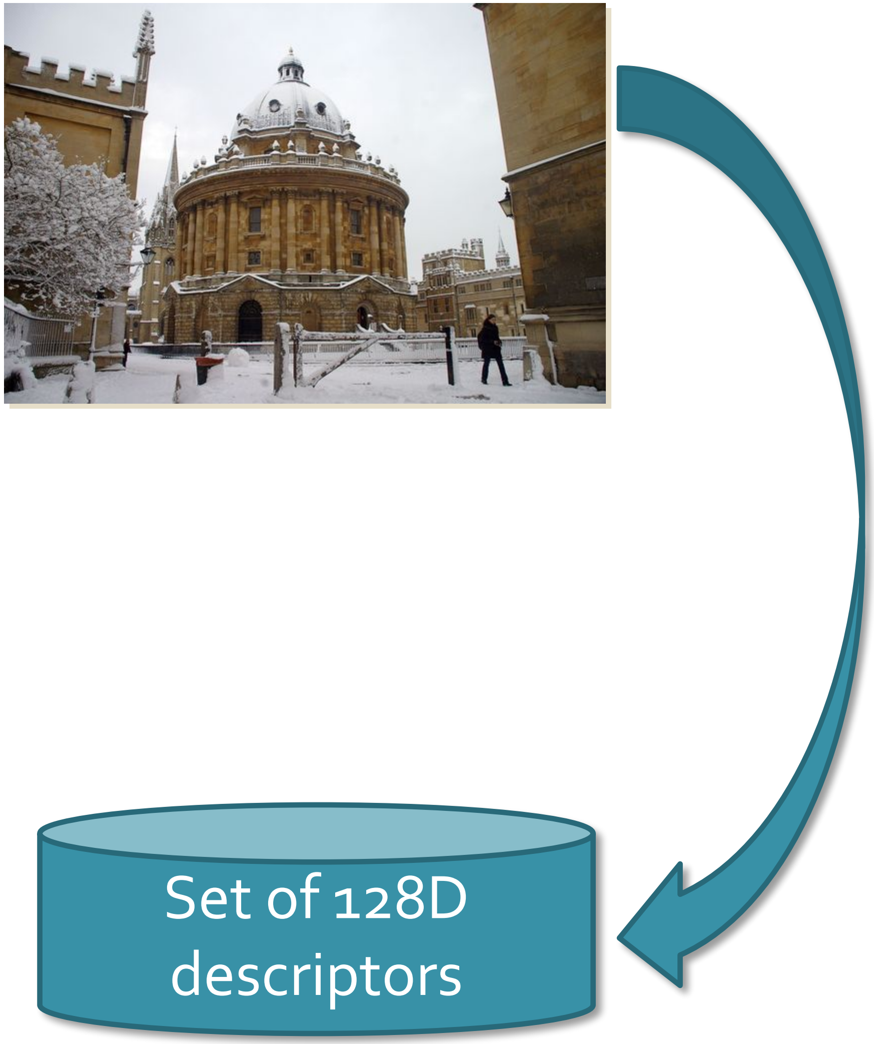
joint work with

Artem Babenko

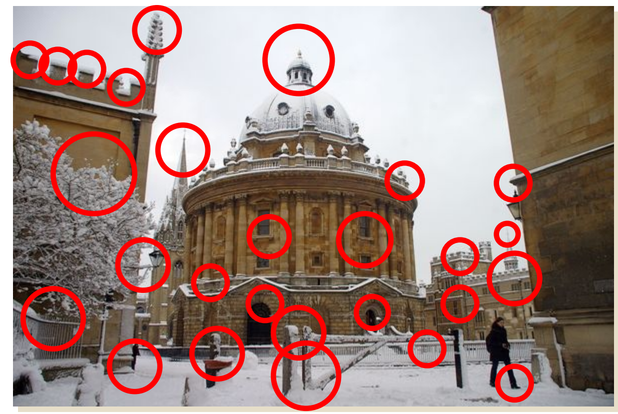
Yandex

VGG Oxford, 25 Oct 2012

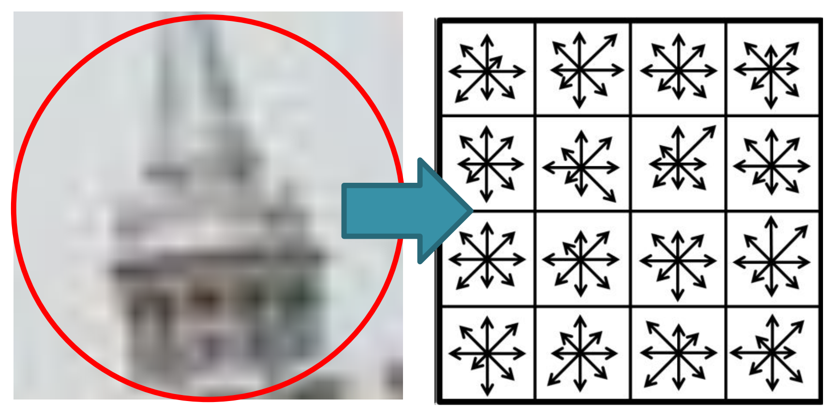
From images to descriptors



Interesting point detection:

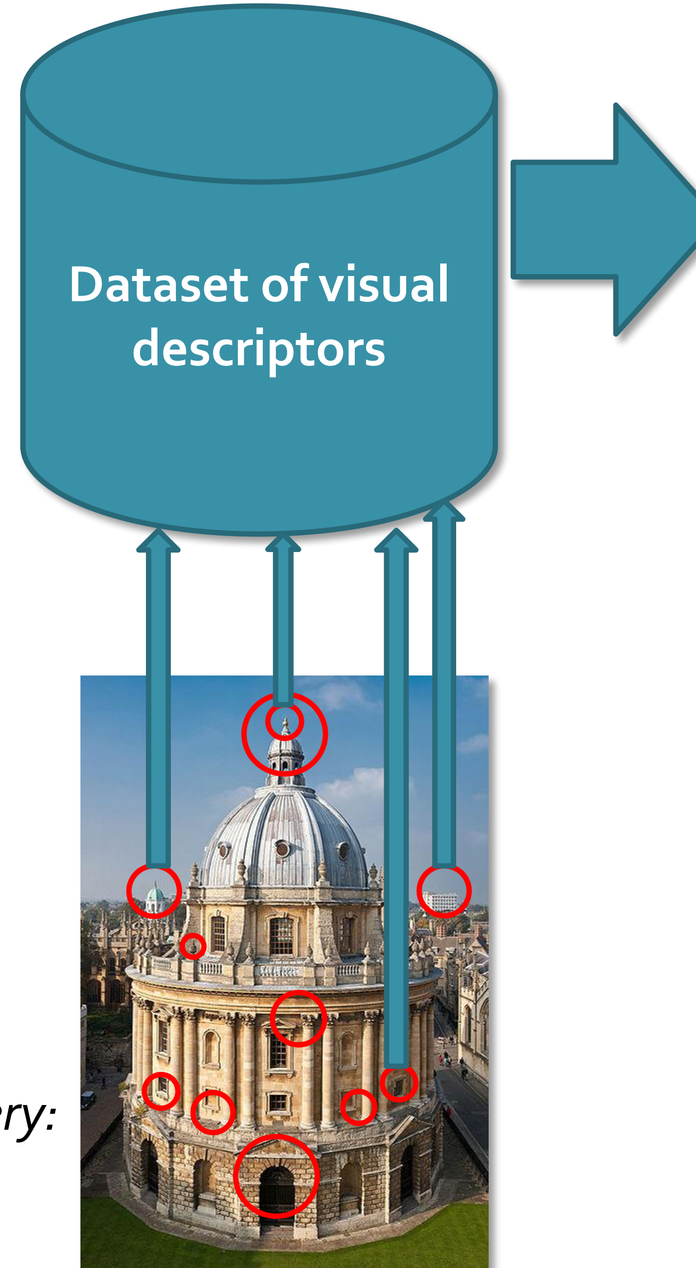


Interesting point description:

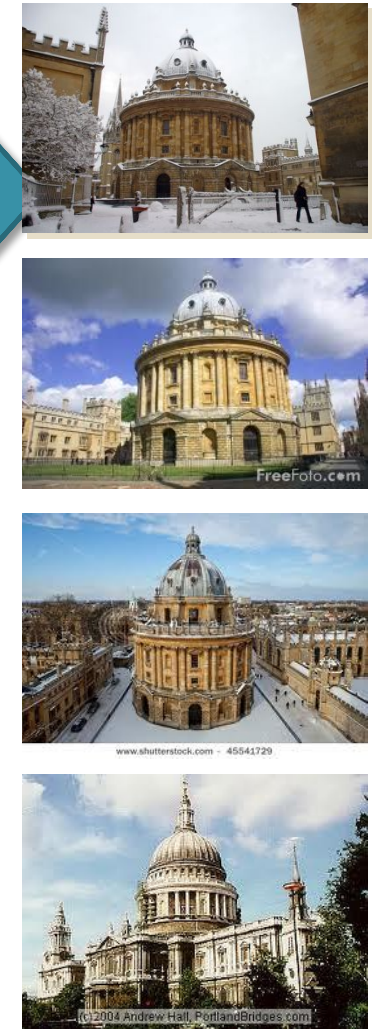


Query process

Image set:



Query:



Main operation:
Finding similar descriptors

Important extras:
+ geometric verification
+ query expansion

Demands

Initial setup:

Dataset size: few million images

Typical RAM size: few dozen gigabytes

Tolerable query time: few seconds

Dataset of visual descriptors

Each image has ~1000 descriptors

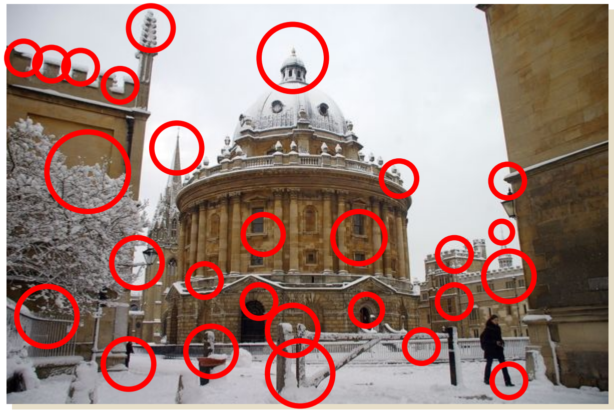
Search problem:

Dataset size: few billion features

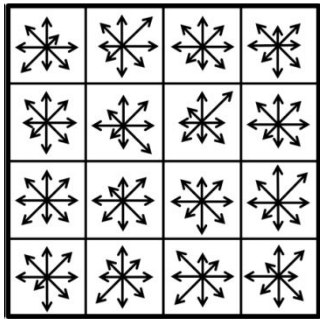
Feature footprint: ~ a dozen bytes

Tolerable time: few milliseconds per feature

nearest neighbor search problem we are tackling



Meeting the demands



Main observation: the vectors have a specific structure: *correlated dimensions, natural image statistics, etc…*

Technologies:

- Dimensionality reduction
- Vector quantization
- Inverted index
- Locality-sensitive hashing
- Product quantization
- Binary/Hamming encodings

Our contribution:

Inverted Multi-Index

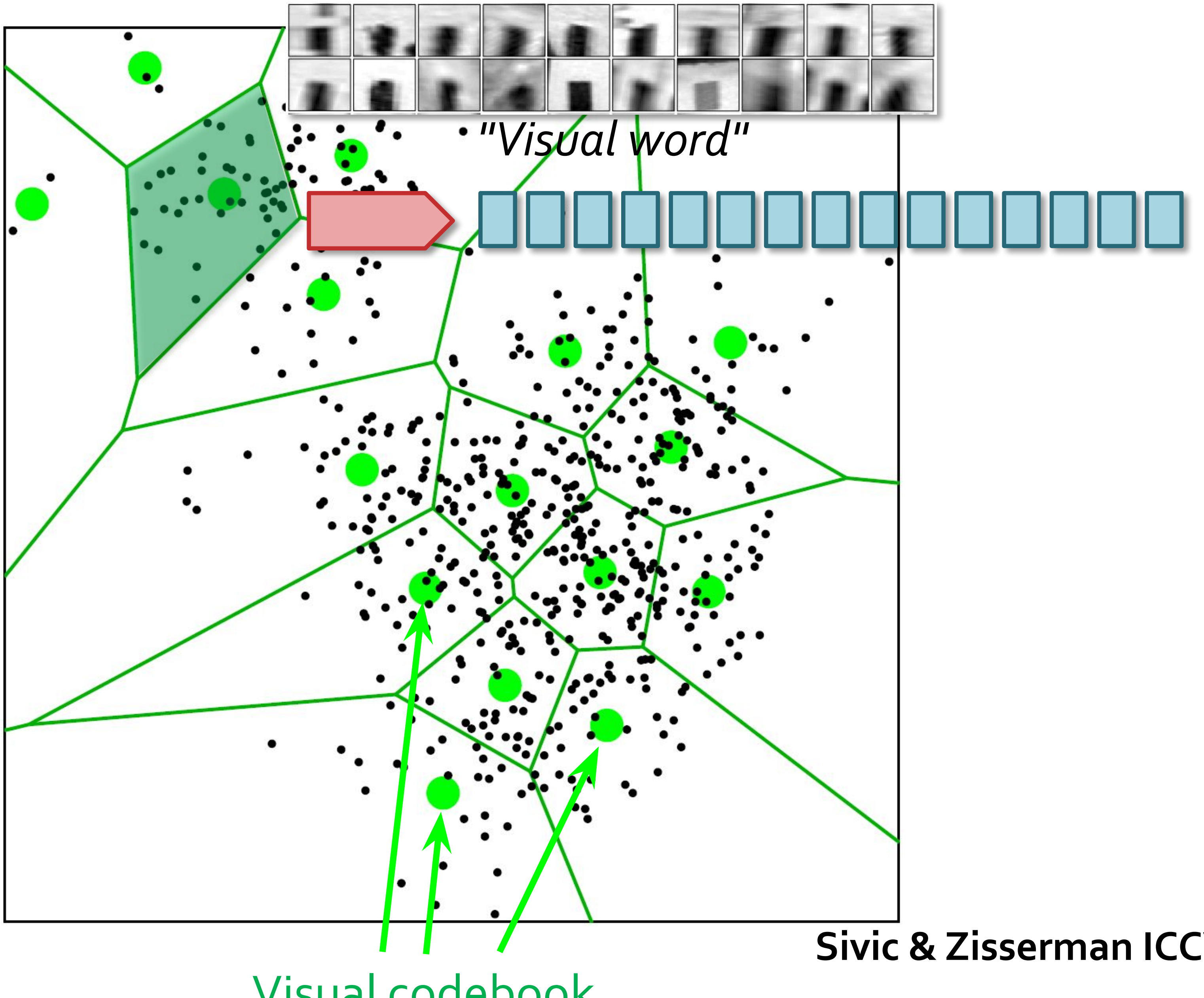
Best combinations (previous state-of-the-art):

- *Inverted index + Product Quantization [Jegou et al. TPAMI 2011]*
- *Inverted index + Binary encoding [Jegou et al. ECCV 2008]*

New state-of-the-art for BIGANN:

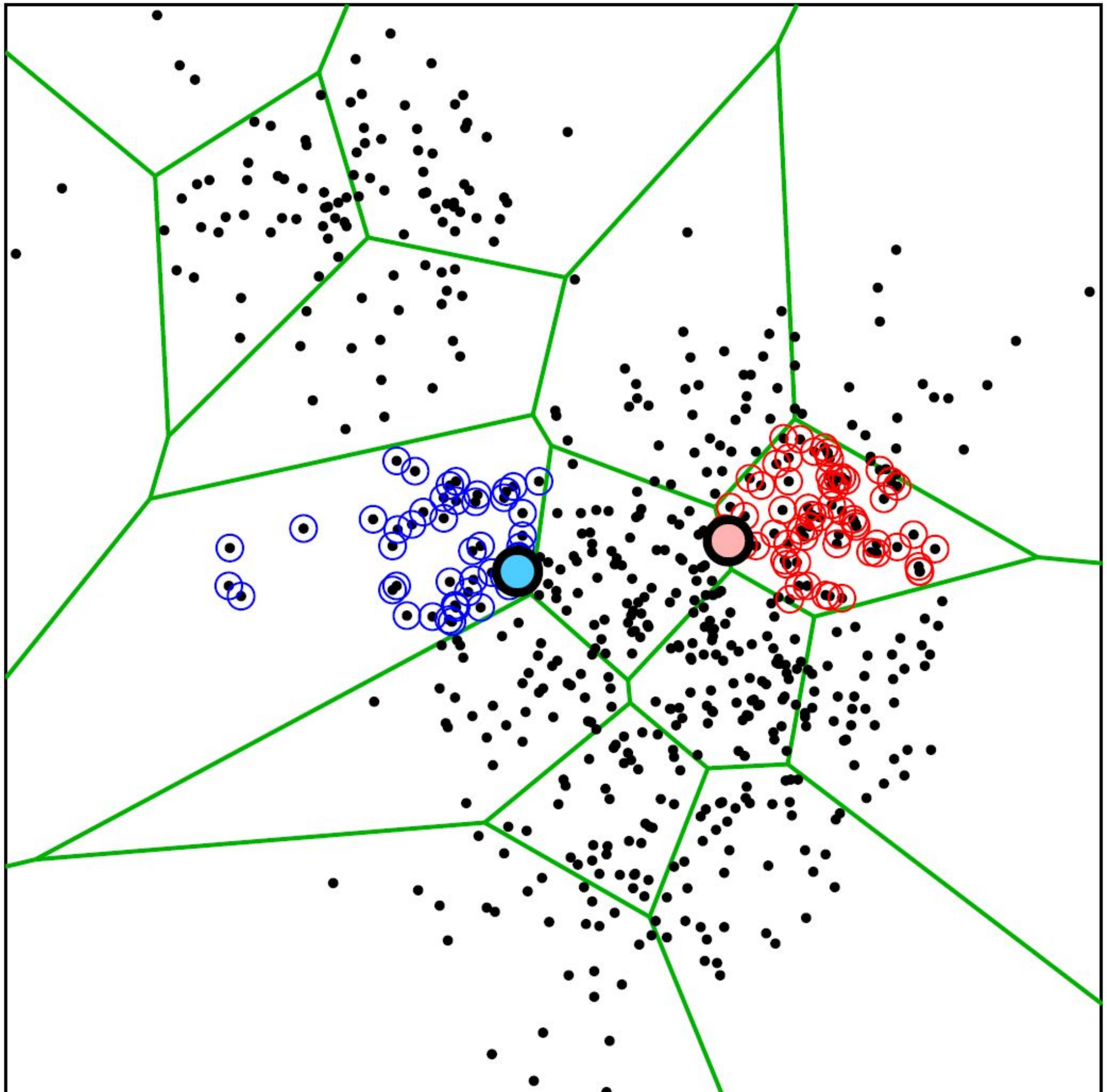
- *Inverted multi-index + Product Quantization [CVPR 2012]*

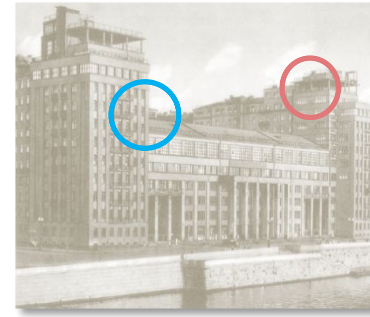
The inverted index



Sivic & Zisserman ICCV 2003

Querying the inverted index



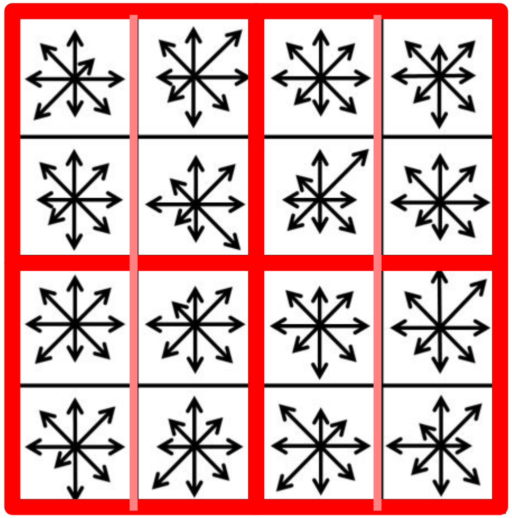
Query: 

- Have to consider several words for best accuracy
- Want to use as big codebook as possible

conflict

- Want to spend as little time as possible for matching to codebooks

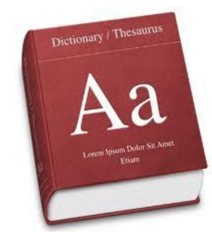
Product quantization



[Jegou, Douze, Schmid // TPAMI 2011]:

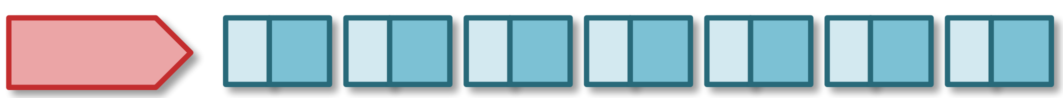
1. Split vector into correlated subvectors
2. use separate small codebook for each chunk

Quantization vs. Product quantization:



For a budget of 4 bytes per descriptor:

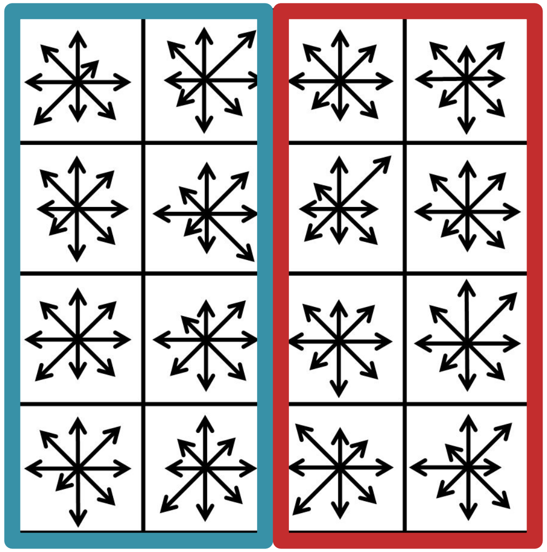
1. Can use a single codebook with 1 billion codewords
2. Can use 4 different codebooks with 256 codewords each

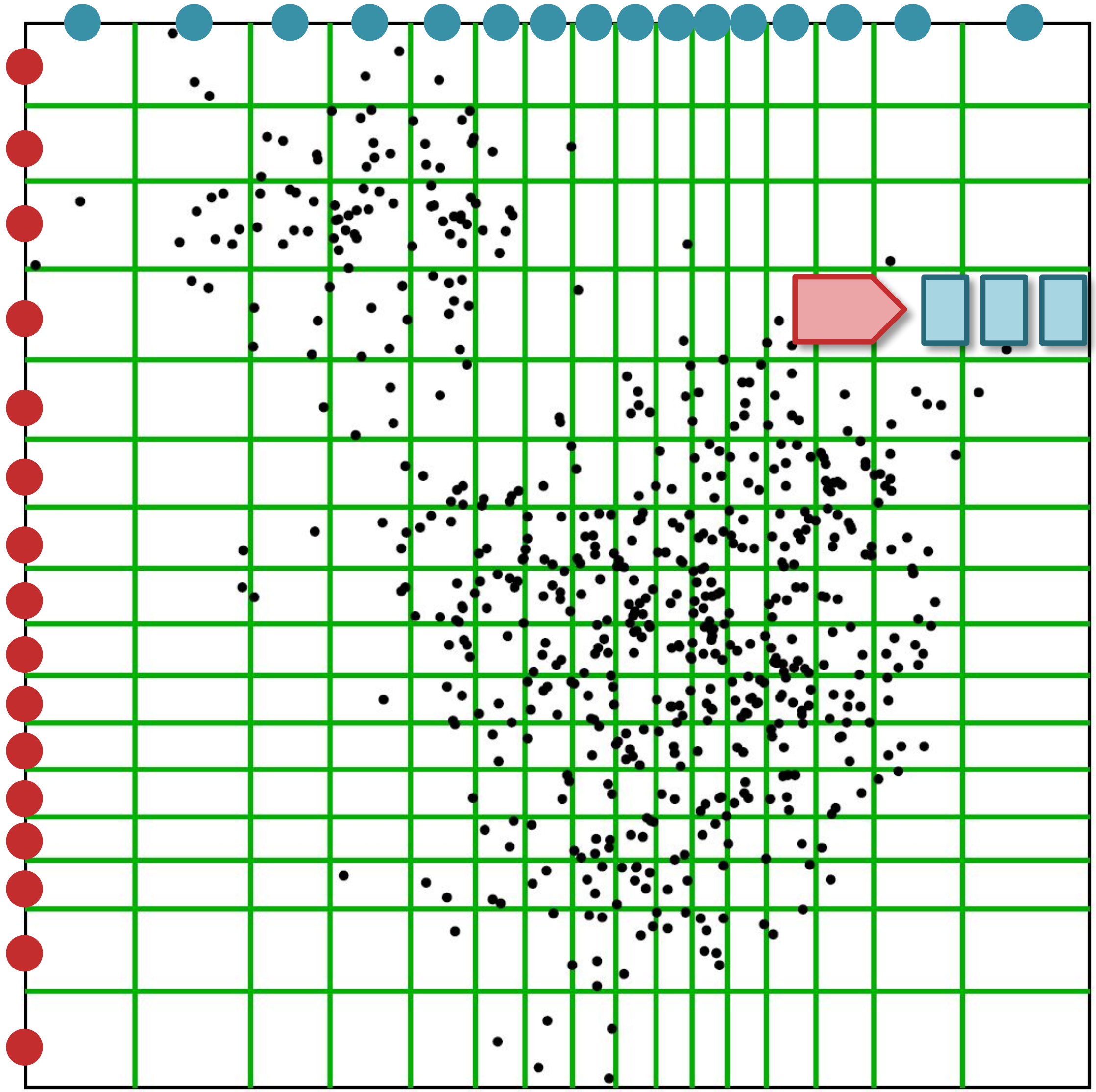


IVFADC+ variants (state-of-the-art for billion scale datasets) = inverted index for indexing + product quantization for reranking

The inverted multi-index

Our idea: use product quantization for indexing





Main advantage:
For the same K, much finer subdivision achieved

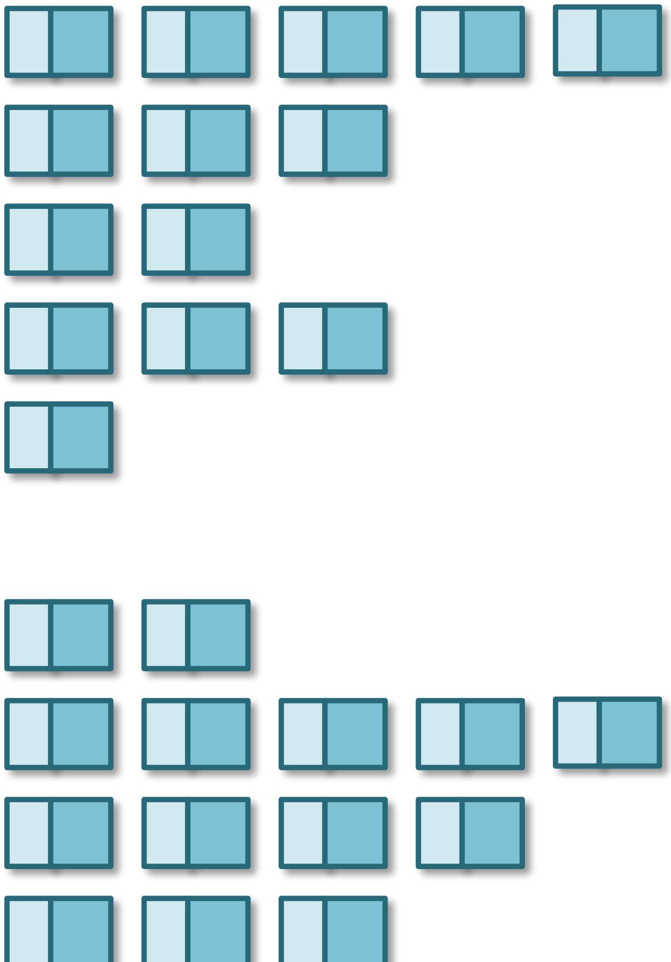
Main problem:
Very non-uniform entry size distribution

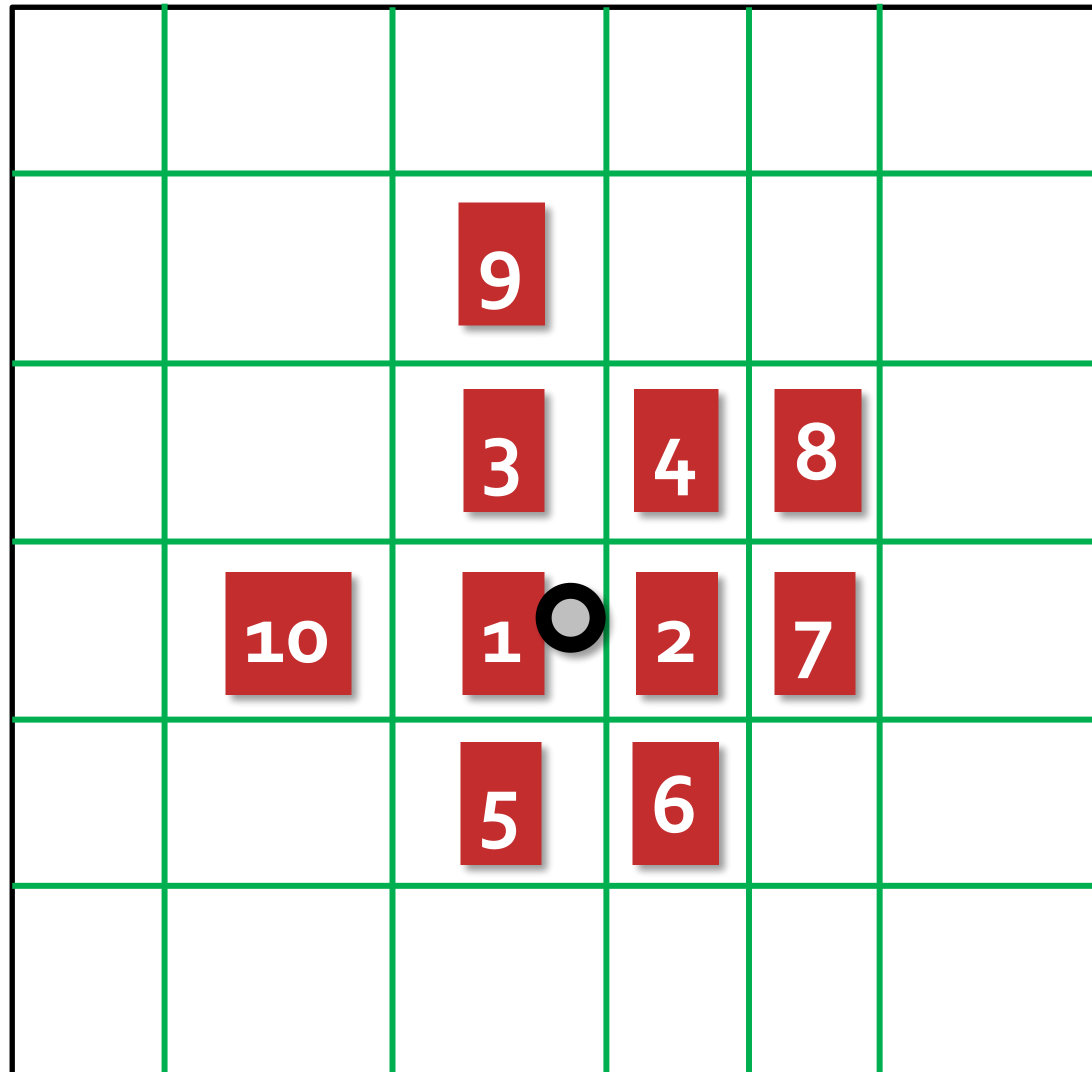
Querying the inverted multi-index

Input: query

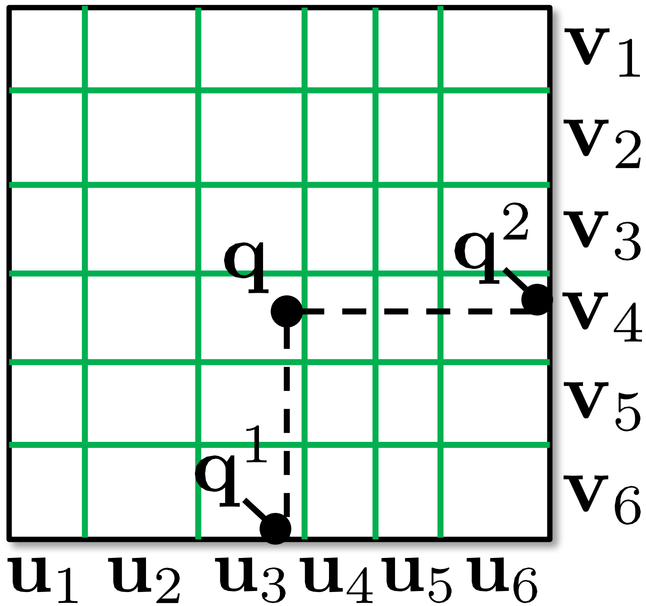
Output: stream of entries

Answer to the query:





Querying the inverted multi-index – Step 1



$\mathbf{q}^2$ vs. $\mathcal{V}$

j	$\mathbf{v}_{\beta(j)}$	s
1	$\mathbf{v}_4$	0.1
2	$\mathbf{v}_3$	2
3	$\mathbf{v}_5$	3
4	$\mathbf{v}_2$	6
5	$\mathbf{v}_6$	7
6	$\mathbf{v}_1$	11

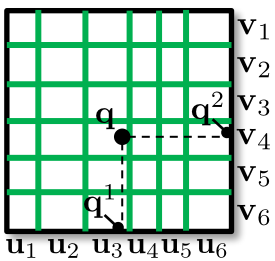
$\mathbf{q}^1$ vs. $\mathcal{U}$

i	$\mathbf{u}_{\alpha(i)}$	r
1	$\mathbf{u}_3$	0.5
2	$\mathbf{u}_4$	0.7
3	$\mathbf{u}_5$	4
4	$\mathbf{u}_2$	6
5	$\mathbf{u}_1$	8
6	$\mathbf{u}_6$	9

	inverted index	inverted multi-index
number of entries	K	K^2
operations to match to codebooks	2K+O(1)	2K+O(1)

Querying the inverted multi-index – Step 2

Step 2: the multi-sequence algorithm



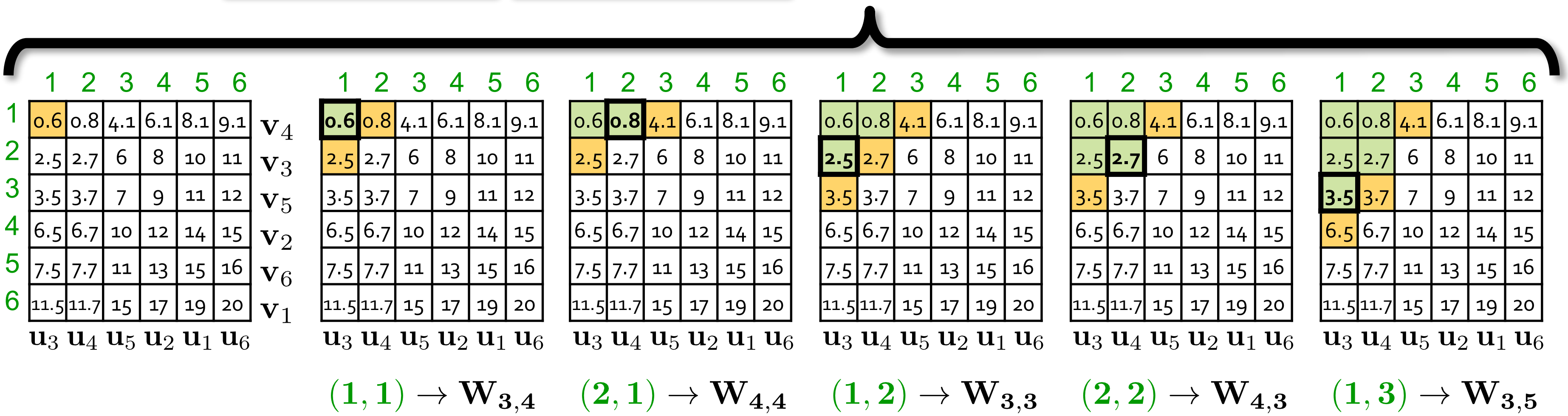
$\mathbf{q}^1$ vs. $\mathcal{U}$

i	$\mathbf{u}_{\alpha(i)}$	r
1	$\mathbf{u_3}$	0.5
2	$\mathbf{u_4}$	0.7
3	$\mathbf{u_5}$	4
4	$\mathbf{u_2}$	6
5	$\mathbf{u_1}$	8
6	$\mathbf{u_6}$	9

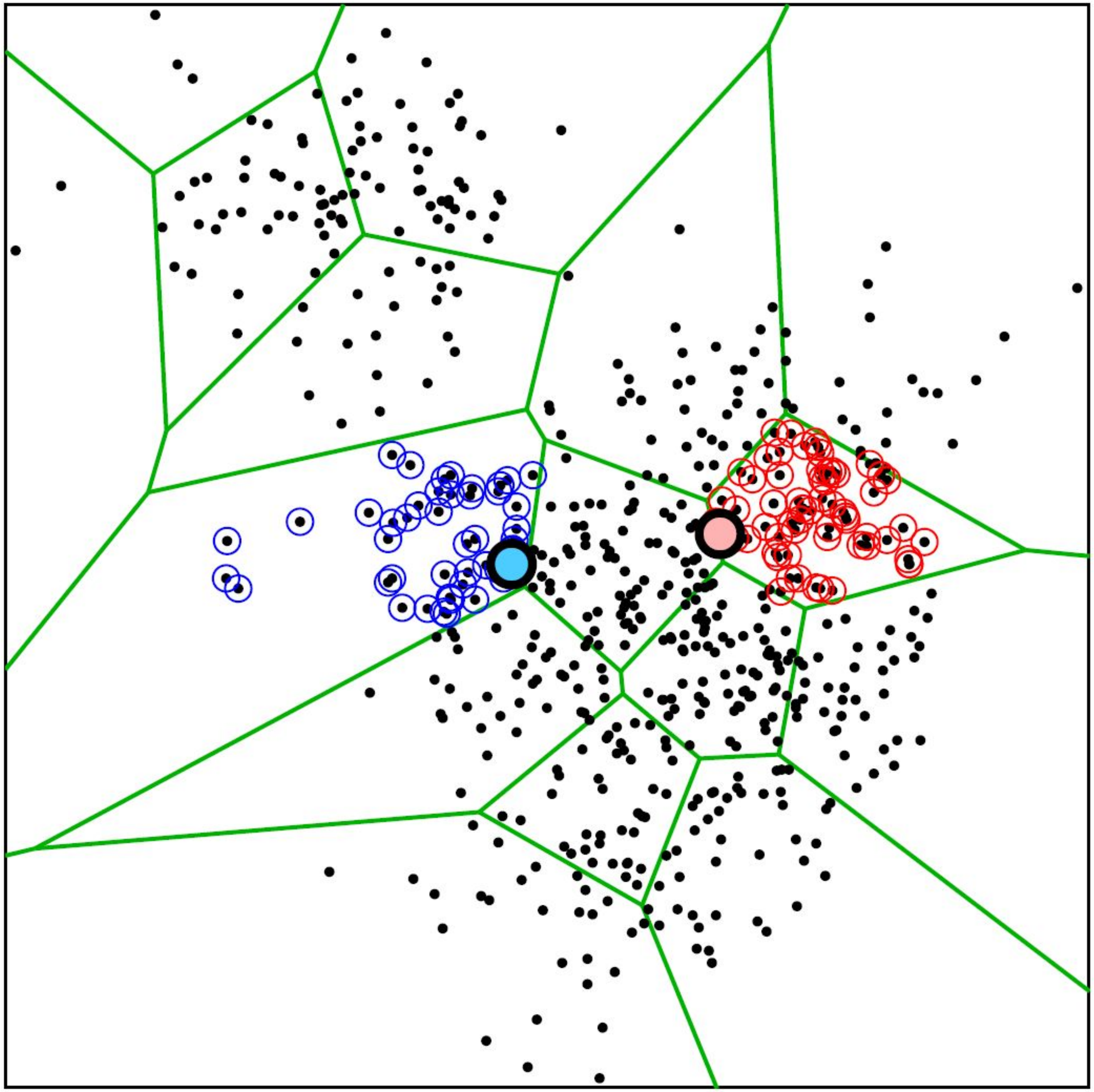
$\mathbf{q}^2$ vs. $\mathcal{V}$

j	$\mathbf{v}_{\beta(j)}$	s
1	$\mathbf{v_4}$	0.1
2	$\mathbf{v_3}$	2
3	$\mathbf{v_5}$	3
4	$\mathbf{v_2}$	6
5	$\mathbf{v_6}$	7
6	$\mathbf{v_1}$	11

$[\mathbf{u}_{\alpha(i)}\ \mathbf{v}_{\beta(j)}]$	(i,j)	$r(i)+s(j)$
$[\mathbf{u_3}\ \mathbf{v_4}]$	(1,1)	0.6 (0.5+0.1)
$[\mathbf{u_4}\ \mathbf{v_4}]$	(2,1)	0.8 (0.7+0.1)
$[\mathbf{u_3}\ \mathbf{v_3}]$	(1,2)	2.5 (0.5+2)
$[\mathbf{u_4}\ \mathbf{v_3}]$	(2,2)	2.7 (0.7+2)
$[\mathbf{u_3}\ \mathbf{v_5}]$	(1,3)	3.5 (0.5+3)
$[\mathbf{u_4}\ \mathbf{v_5}]$	(2,3)	3.7 (0.7+3)
$[\mathbf{u_5}\ \mathbf{v_4}]$	(3,1)	4.1 (4+0.1)
$[\mathbf{u_5}\ \mathbf{v_3}]$	(3,2)	6 (4+2)
$[\mathbf{u_3}\ \mathbf{v_2}]$	(1,4)	6.5 (0.5+6)
…		



Querying the inverted multi-index



Experimental protocol

Dataset:

1. 1 billion of SIFT vectors [Jegou et al.]
2. Hold-out set of 10000 queries, for which Euclidean nearest neighbors are known

Comparing index and multi-index:

Set a candidate set length T

For each query:

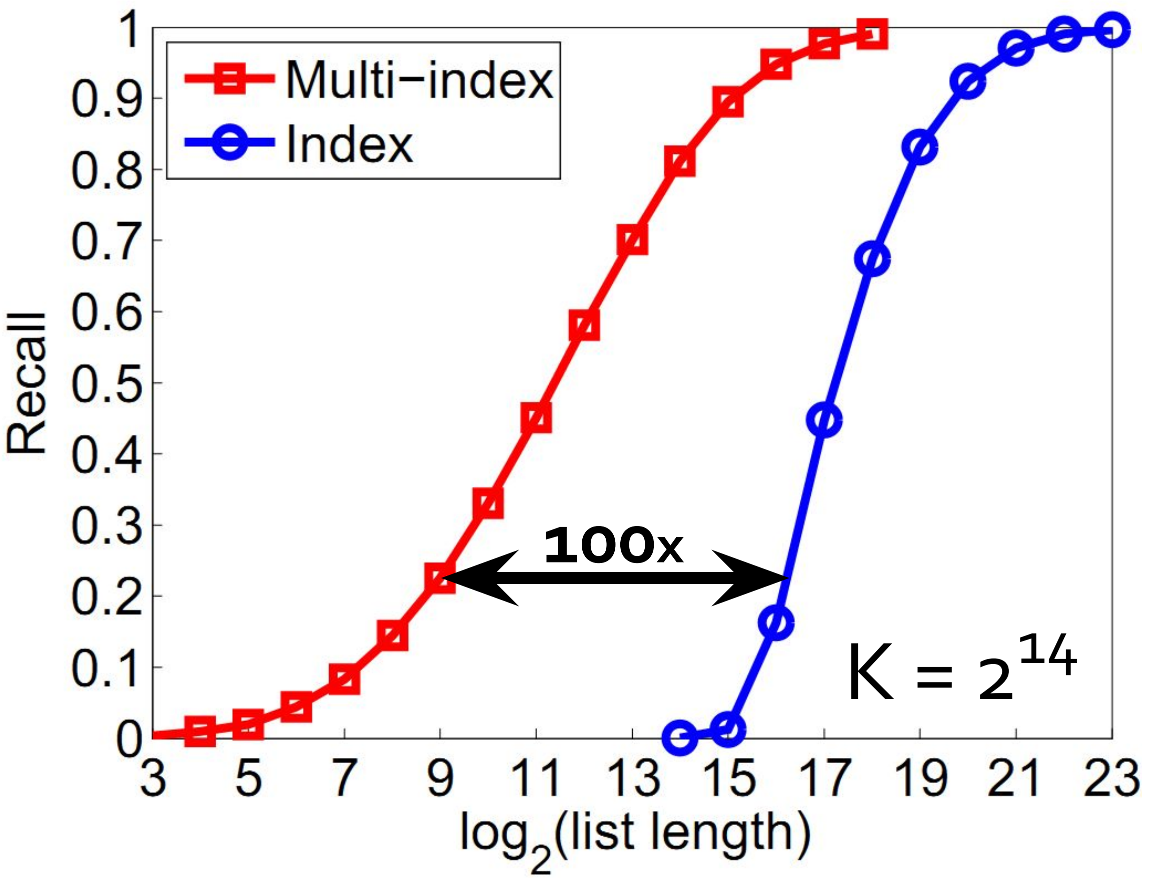
- Retrieve closest entries from index or multi-index and concatenate lists
- Stop when the next entry does not fit
  - For small T inverted index can return empty list
- Check whether the true neighbor is in the list

Report the share of queries where the neighbor was present (*recall@T*)

Performance comparison

Recall on the dataset of 1 billion of visual descriptors:

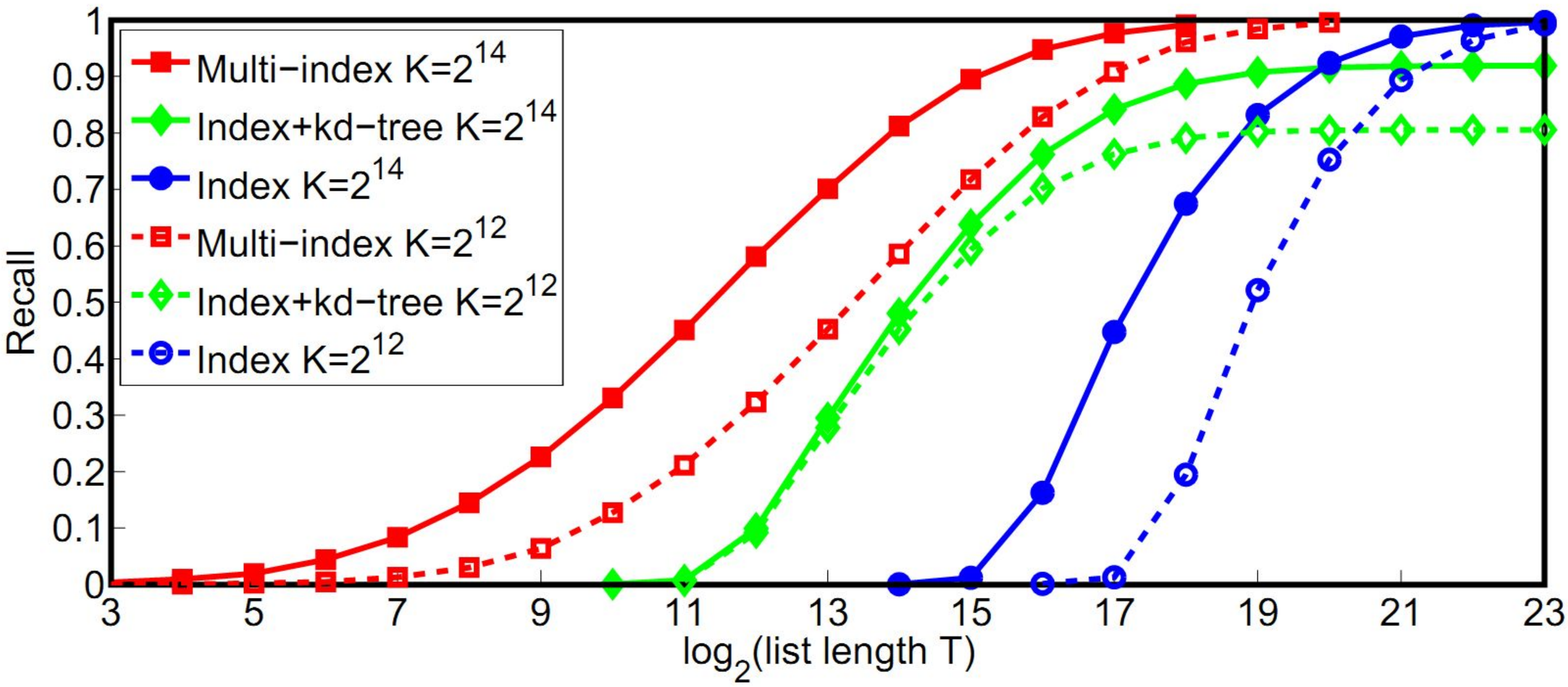
"How fast can we catch the nearest neighbor to the query?"



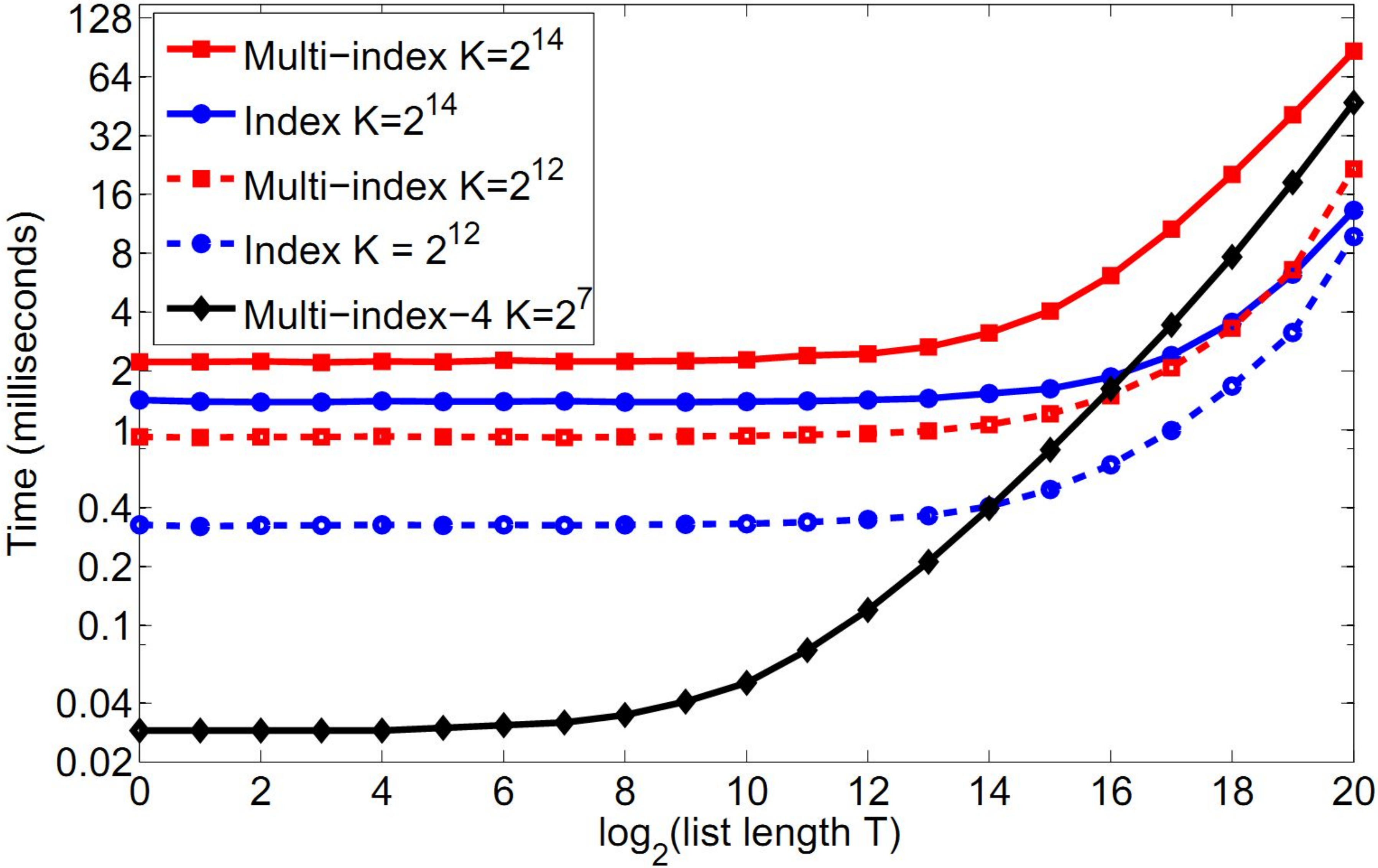
Time increase: 1.4 msec -> 2.2 msec on a single core (with BLAS instructions)

Performance comparison

Recall on the dataset of 1 billion 128D visual descriptors:

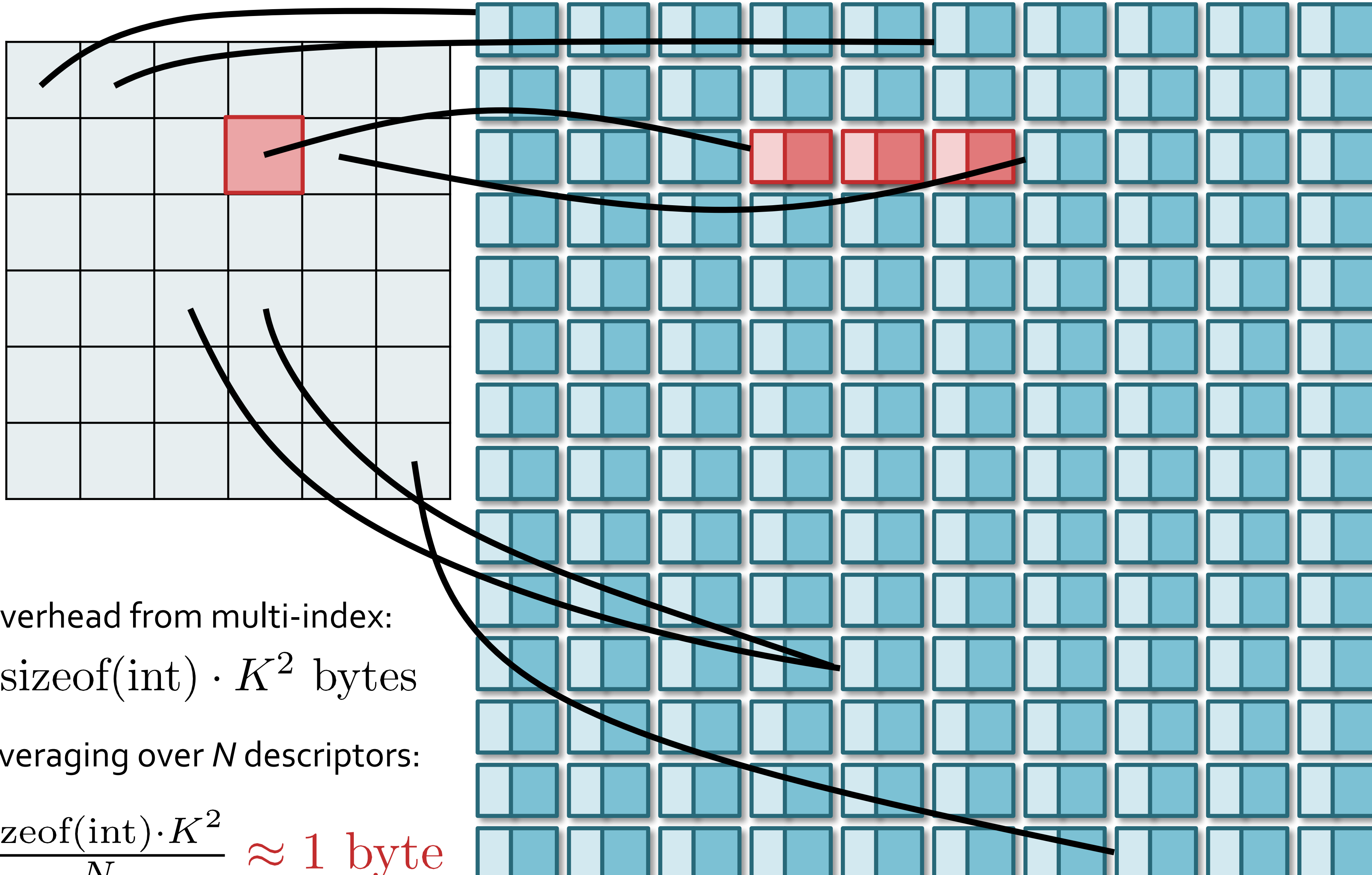


Time complexity



For same K index gets a slight advantage because of BLAS instructions

Memory organization



Overhead from multi-index:

$$\text{sizeof(int)} \cdot K^2 \text{ bytes}$$

Averaging over *N* descriptors:

$$\frac{\text{sizeof(int)} \cdot K^2}{N} \approx 1 \text{ byte}$$

$$K = 2^{14}, N = 1 \text{ billion}$$

Why two?

For larger number of parts:

- Memory overhead becomes larger

$$\text{sizeof(int)} \cdot K^2 \text{ bytes} \Rightarrow \text{sizeof(int)} \cdot K^4 \text{ bytes}$$

- Population densities become even more non-uniform (multi-sequence algorithm has to work harder to accumulate the candidates)

In our experiments, 4 parts with small K=128 may be competitive for some datasets and reasonably short candidate lists (*e.g. duplicate search*). Indexing is blazingly fast in these cases!

Multi-Index + Reranking

- "Multi-ADC": use *m* bytes to encode the original vector using product quantization
  faster (efficient caching possible for distance computation)

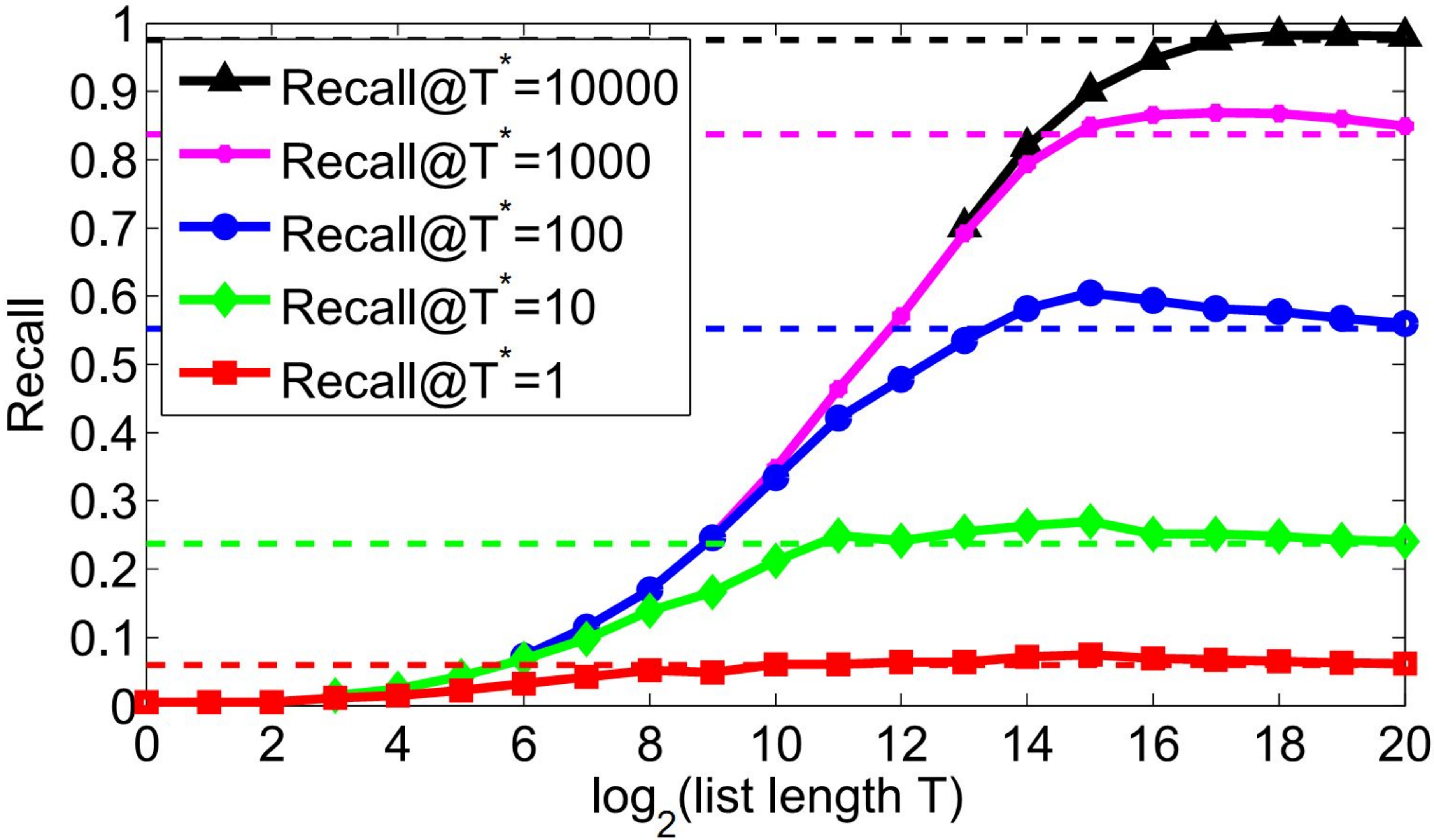
- "Multi-D-ADC": use *m* bytes to encode the remainder between the original vector and the centroid
  - *Same architecture as IVFADC of Jegou et al., but replaces index with multi-index*

  more accurate

Evaluation protocol:

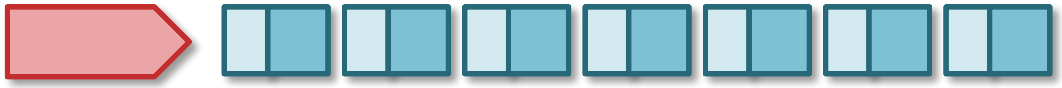
1. Query the inverted index for ***T*** candidates
2. Reconstruct the original points and *rerank* according to the distance to the query
3. Look whether the true nearest neighbor is within top ***T****

Multi-ADC vs. Exhaustive search



Multi-D-ADC vs State-of-the-art

Combining multi-index + reranking:

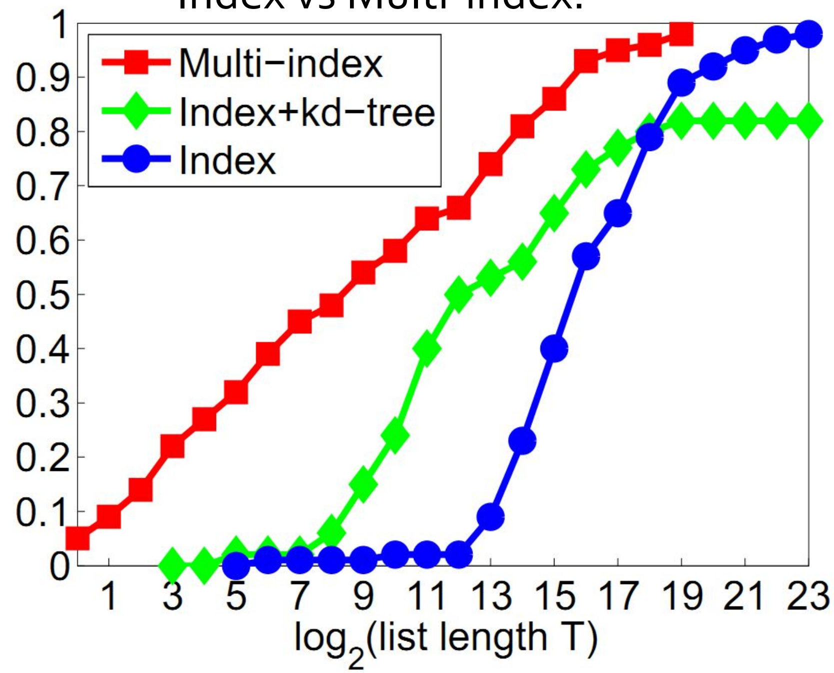


System	List len. T	R@1	R@10	R@100	Time
BIGANN, 1 billion SIFTs, 8 bytes per vector					
IVFADC	8 million	0.112	0.343	0.728	155
State-of-the-art [Jegou et al.]		*(0.088)*	*(0.372)*	*(0.733)*	*(74*)*
Multi-D-ADC	10000	0.158	0.472	0.706	6
Multi-D-ADC	30000	0.164	0.506	0.813	13
Multi-D-ADC	100000	0.165	0.517	0.860	37

Performance on 80 million GISTs

Same protocols as before, but on 80 million GISTs (384 dimensions) of Tiny Images [Torralba et al. PAMI'08]

Index vs Multi-index:

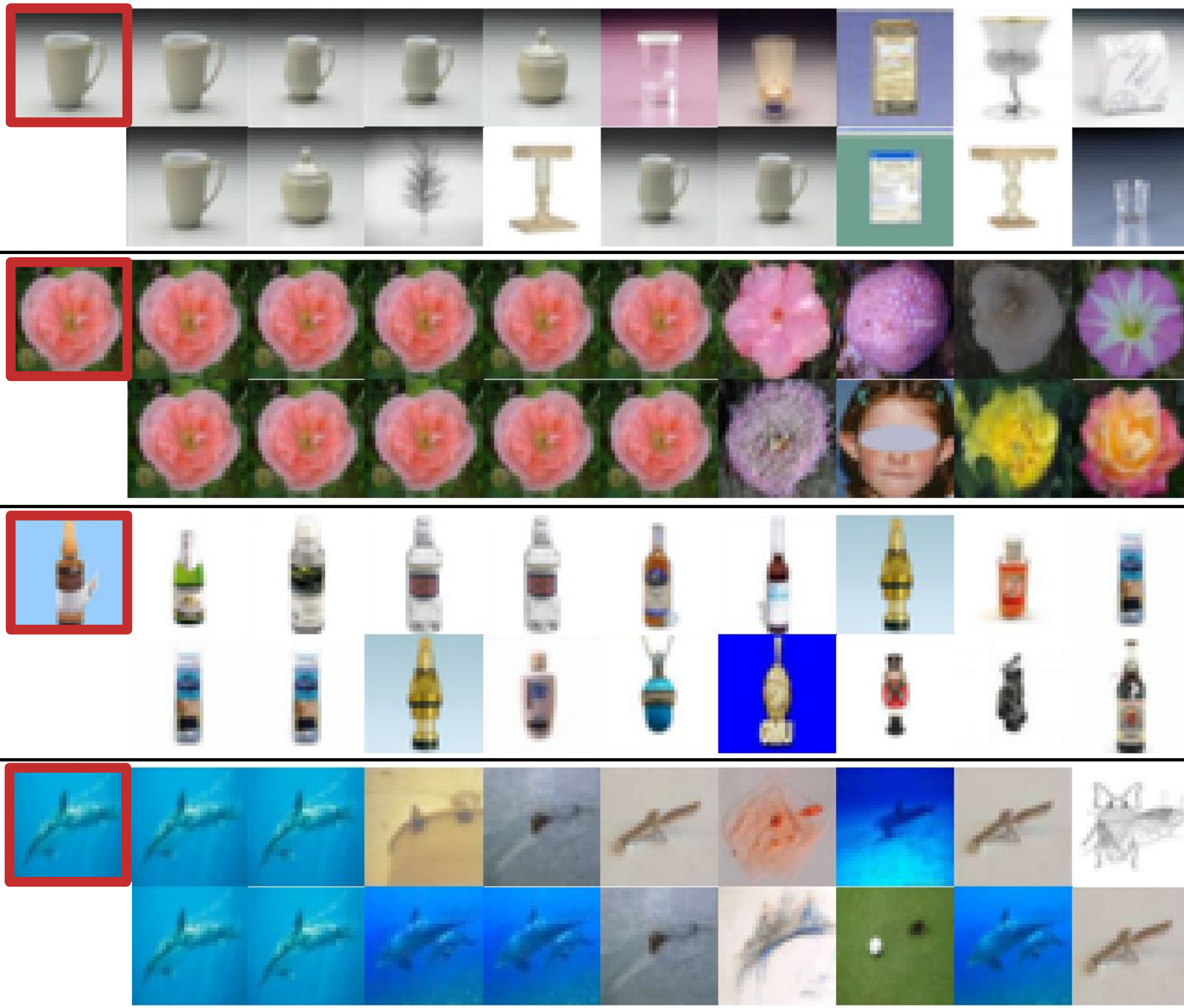


Multi-D-ADC performance:

Tiny Images, 80 million GISTs, 8 bytes per vector					
Multi-D-ADC	10000	0.06	0.40	0.59	19
Multi-D-ADC	30000	0.06	0.41	0.63	41
Multi-D-ADC	100000	0.06	0.41	0.66	119

Tiny Images, 80 million GISTs, 16 bytes per vector					
Multi-D-ADC	10000	0.06	0.49	0.64	19
Multi-D-ADC	30000	0.06	0.56	0.76	46
Multi-D-ADC	100000	0.06	0.56	0.85	139

Retrieval examples



Exact NN
Uncompressed GIST

Multi-D-ADC
16 bytes

Exact NN
Uncompressed GIST

Multi-D-ADC
16 bytes

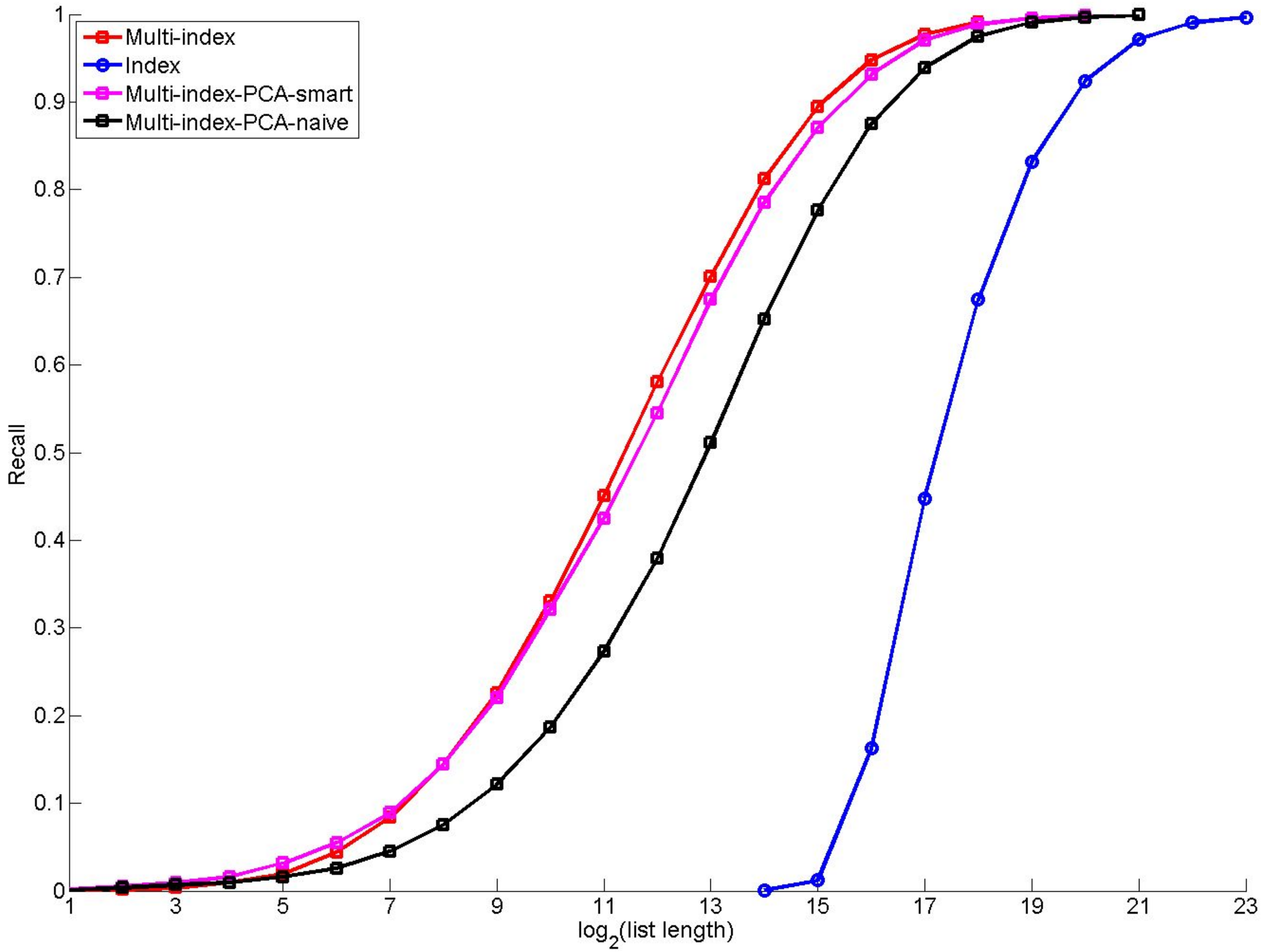
Exact NN
Uncompressed GIST

Multi-D-ADC
16 bytes

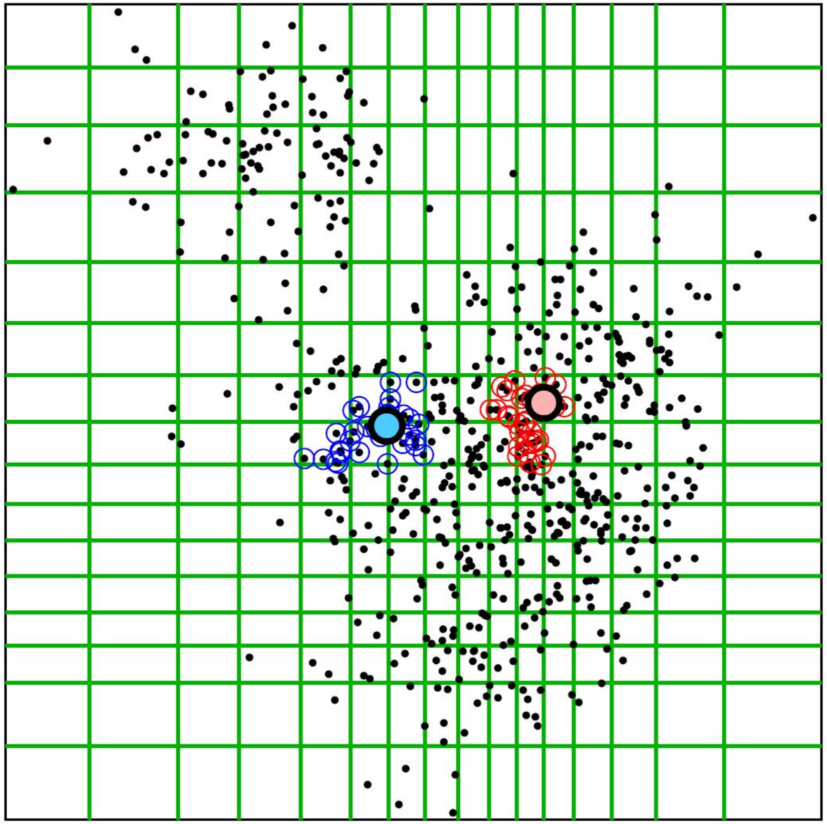
Exact NN
Uncompressed GIST

Multi-D-ADC
16 bytes

Multi-Index and PCA (128->32 dimensions)



Conclusions



- A new data structure for indexing the visual descriptors
- Significant accuracy boost over the inverted index at the cost of the small memory overhead
- Code available (will soon be online)

Other usage scenarios

(Mostly) straightforward extensions possible:

- Large-scale NN search' based approaches:
  - Holistic high dimensional image descriptors: GISTs, VLADs, Fisher vectors, classemes…
  - Pose descriptors
  - Other multimedia
- Additive norms and kernels: L1, Hamming, Mahalanobis, chi-square kernel, intersection kernel, etc.

Visual search

What is this painting?

The closest match:

Van Gogh, 1890
"Landscape with Carriage and Train in the Background"
Pushkin museum, Moscow
Learn more about it