Vocabulary tree

Vocabulary tree

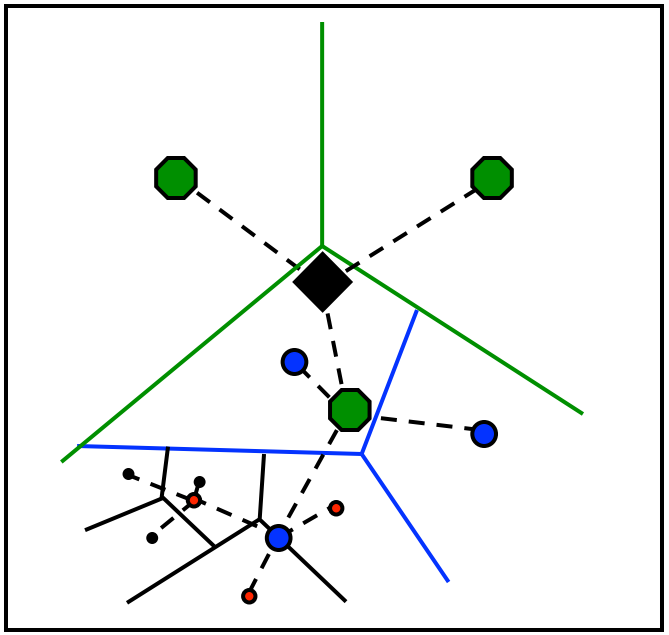
- Recognition can scale to very large databases using the Vocabulary Tree indexing approach [Nistér and Stewénius, CVPR 2006]. Vocabulary Tree performs instance object recognition. It does not perform recognition at category-level.

- Vocabulary Tree method follows three steps:
  1. organize local descriptors of images a in a tree using hierarchical k-means clustering. Inverted files are stored at each node with scores (offline).
  2. generate a score for a given query image based on Term Frequency – Inverse Document Frequency.
  3. find the images in the database that best match that score.

- Vocabulary tree supports very efficient retrieval. It only cares about the distance between a query feature and each node.

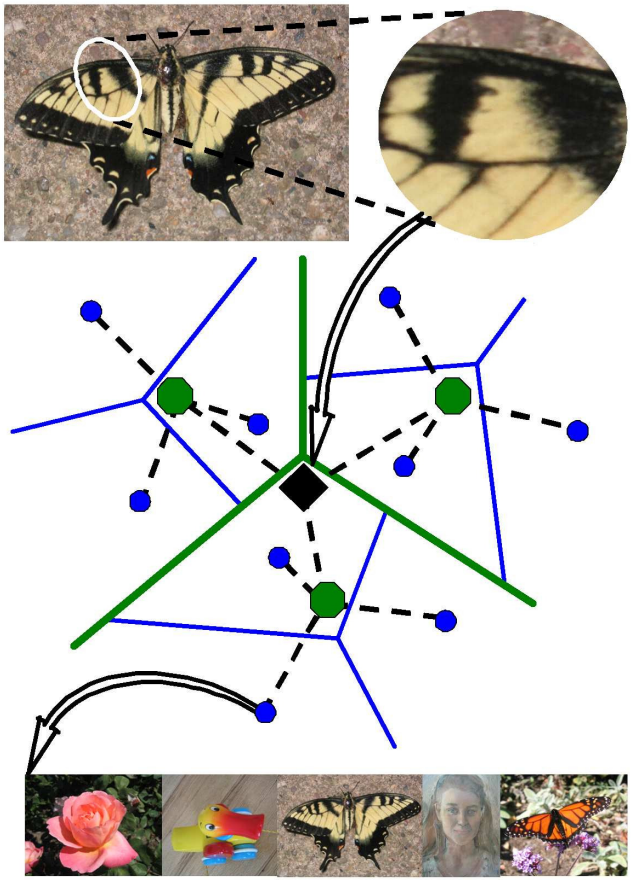
Building the Vocabulary Tree

- The vocabulary tree is a hierarchical set of cluster centers and their corresponding Voronoi regions:
  - For each image in the database, extract MSER regions and calculate a set of feature point descriptors (e.g. 128 SIFT).
  - Build the vocabulary tree using hierarchical k-means clustering:
    - run k-means recursively on each of the resulting quantization cells up to a max number of levels L (L=6 max suggested)
    - nodes are the centroids; leaves are the visual words.
    - k defines the branch-factor of the tree, which indicates how fast the tree branches (k=10 max suggested)

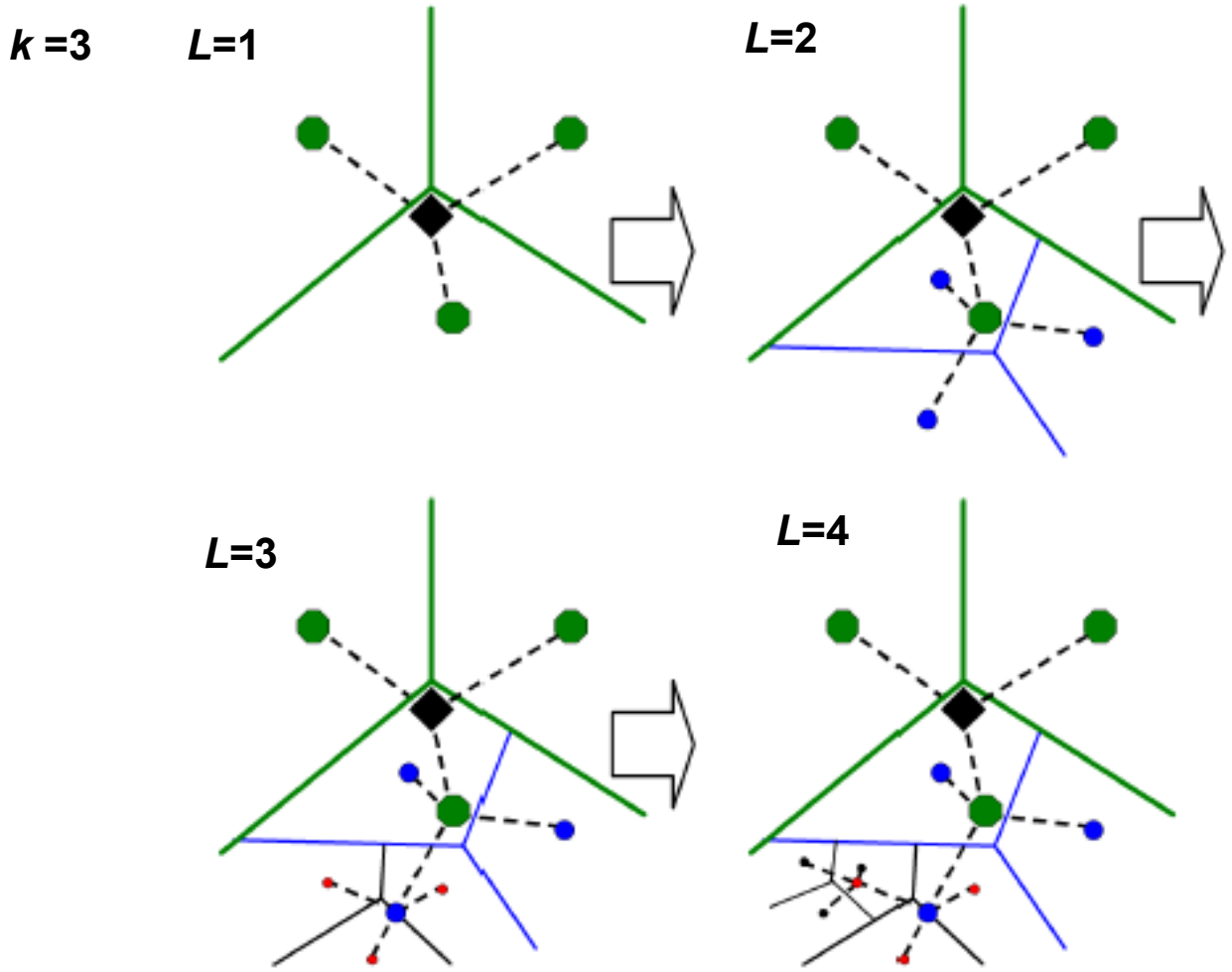


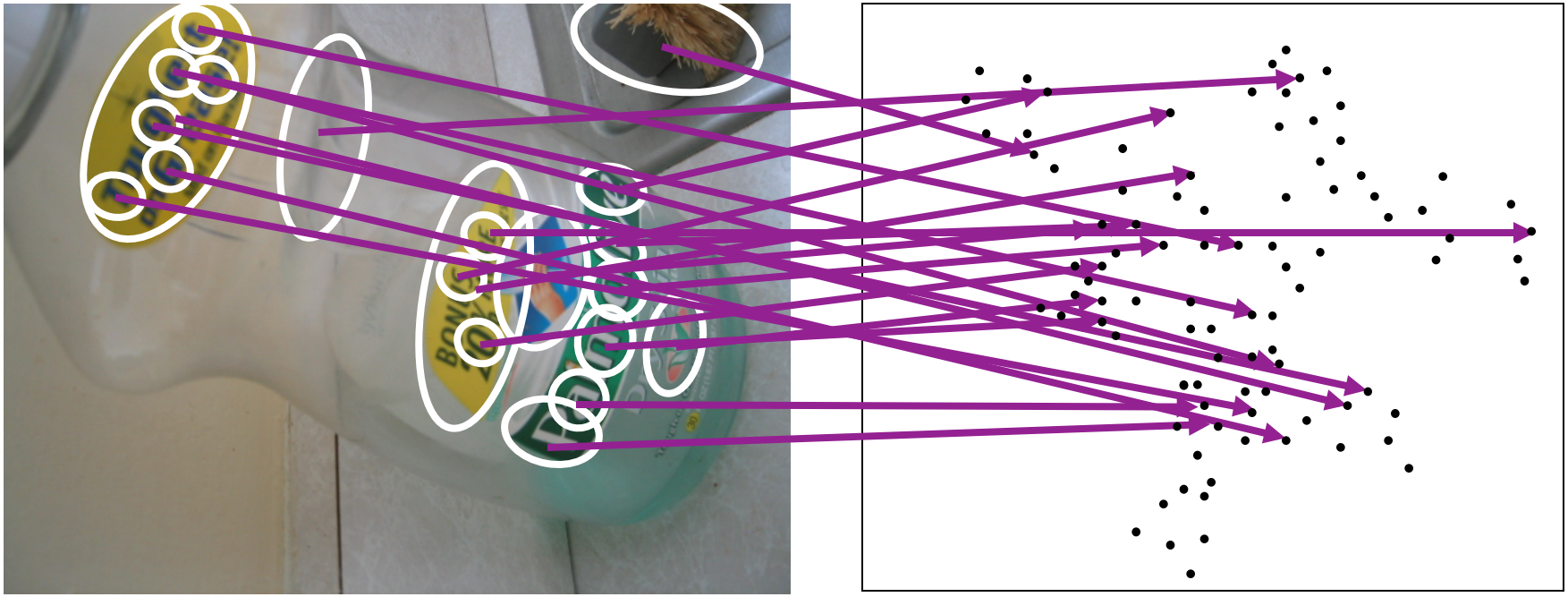
- A large number of elliptical regions are extracted from the image and warped to canonical positions. A descriptor vector is computed for each region. The descriptor vector is then hierarchically quantized by the vocabulary tree.

- With each node in the vocabulary tree there is an associated inverted file with references to the images containing an instance of that node.

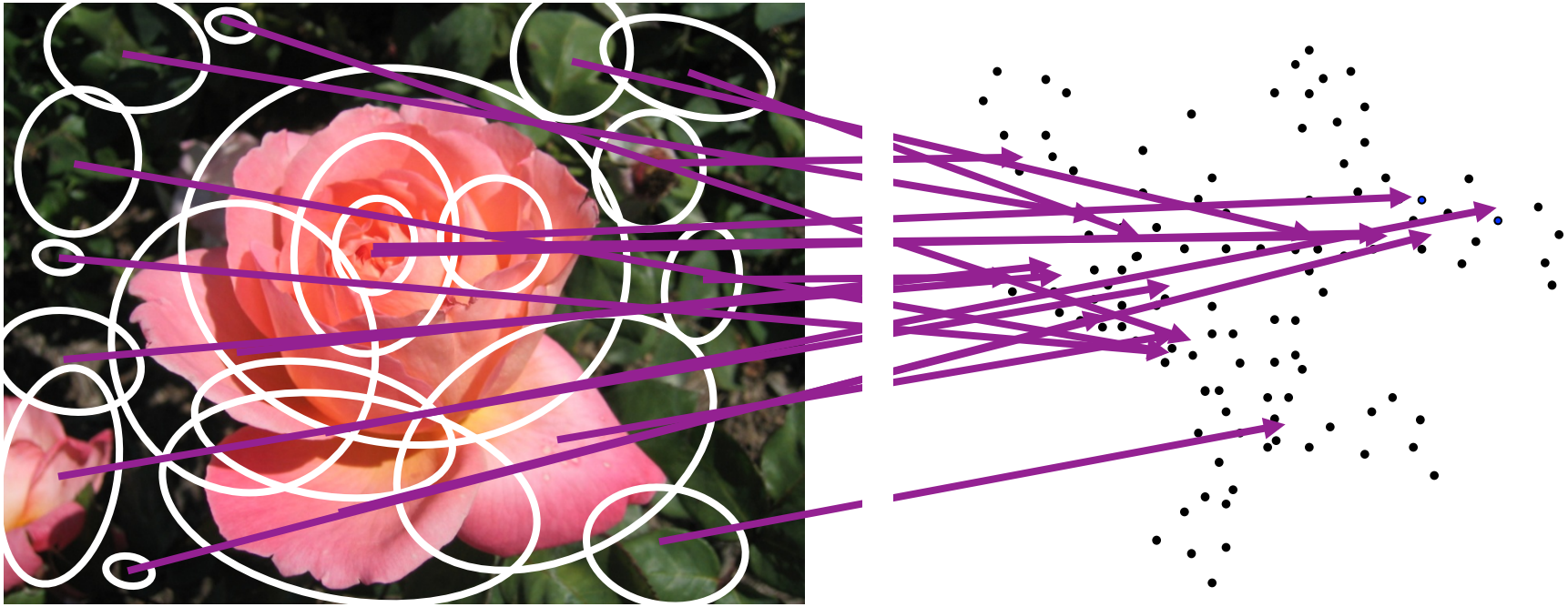


Hierarchical k-means clustering

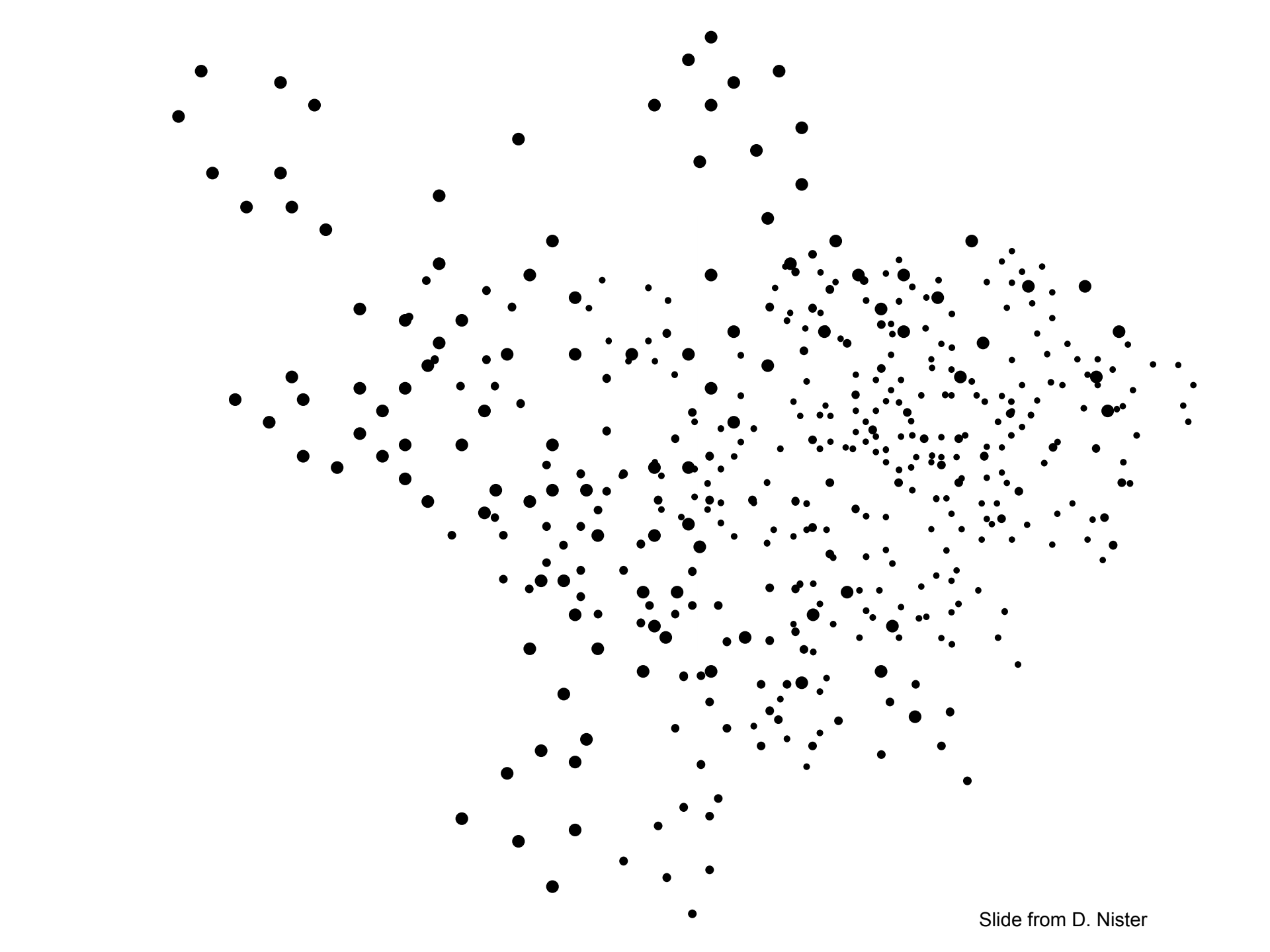




Slide from D. Nister

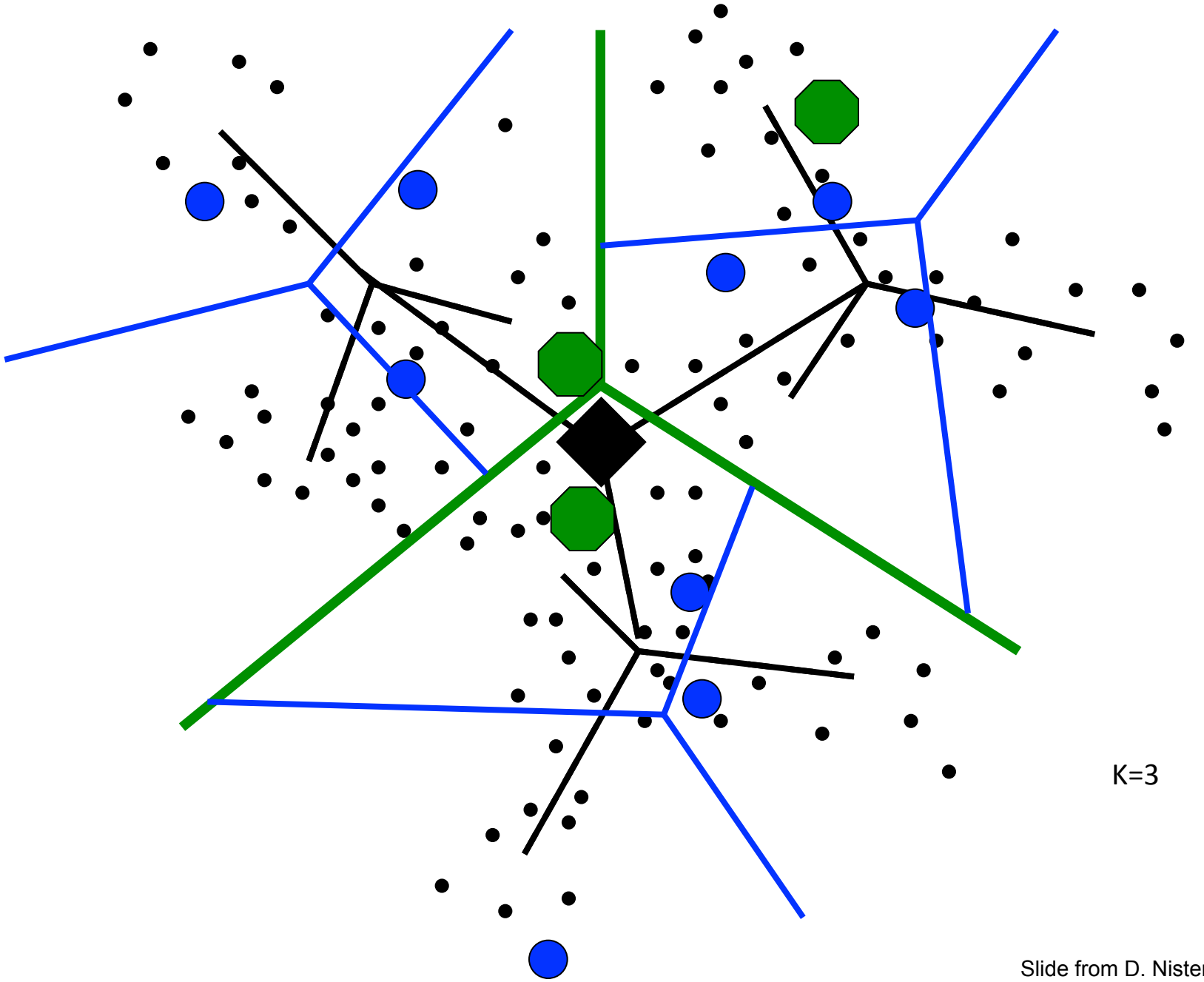


Slide from D. Nister

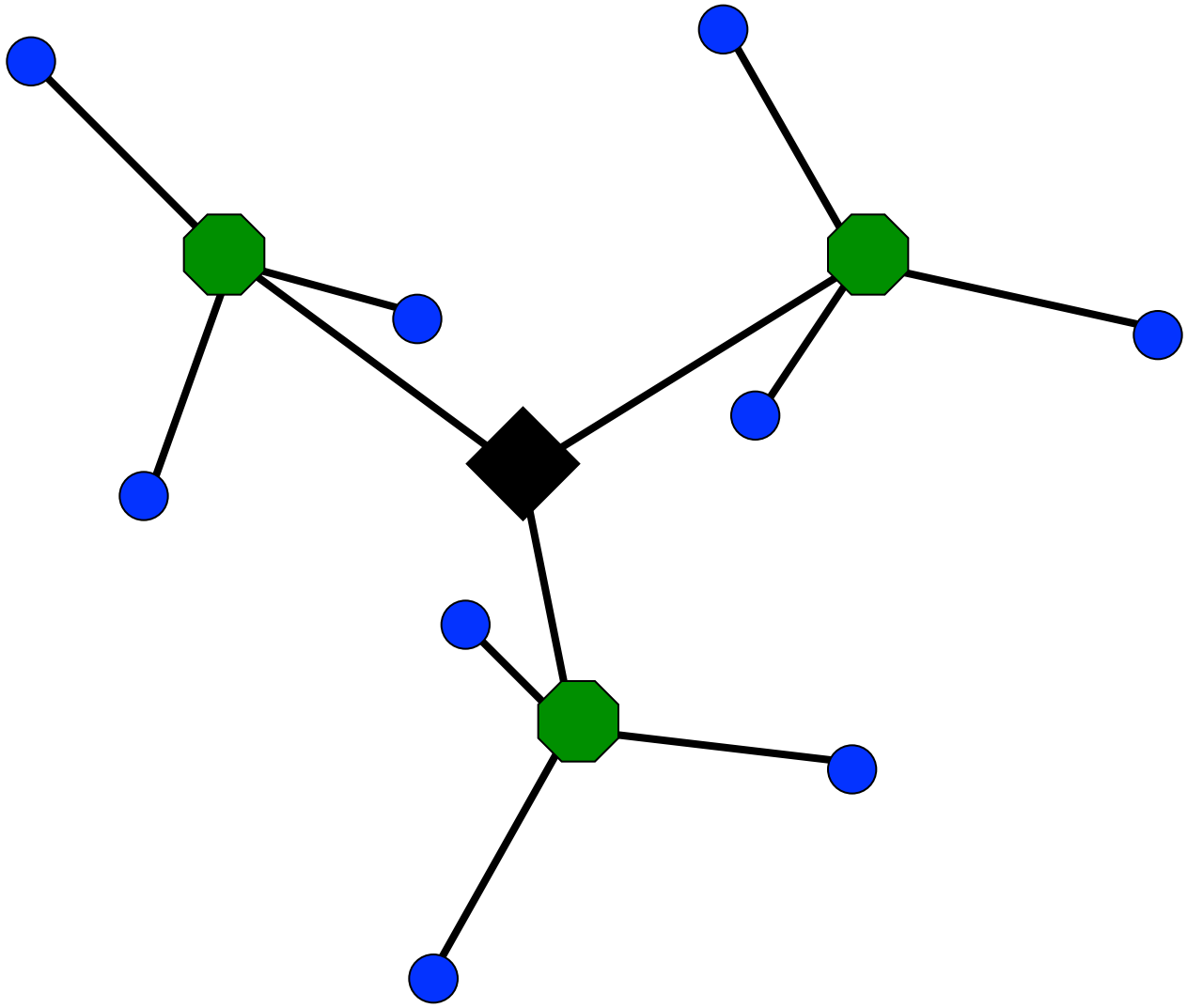


Slide from D. Nister

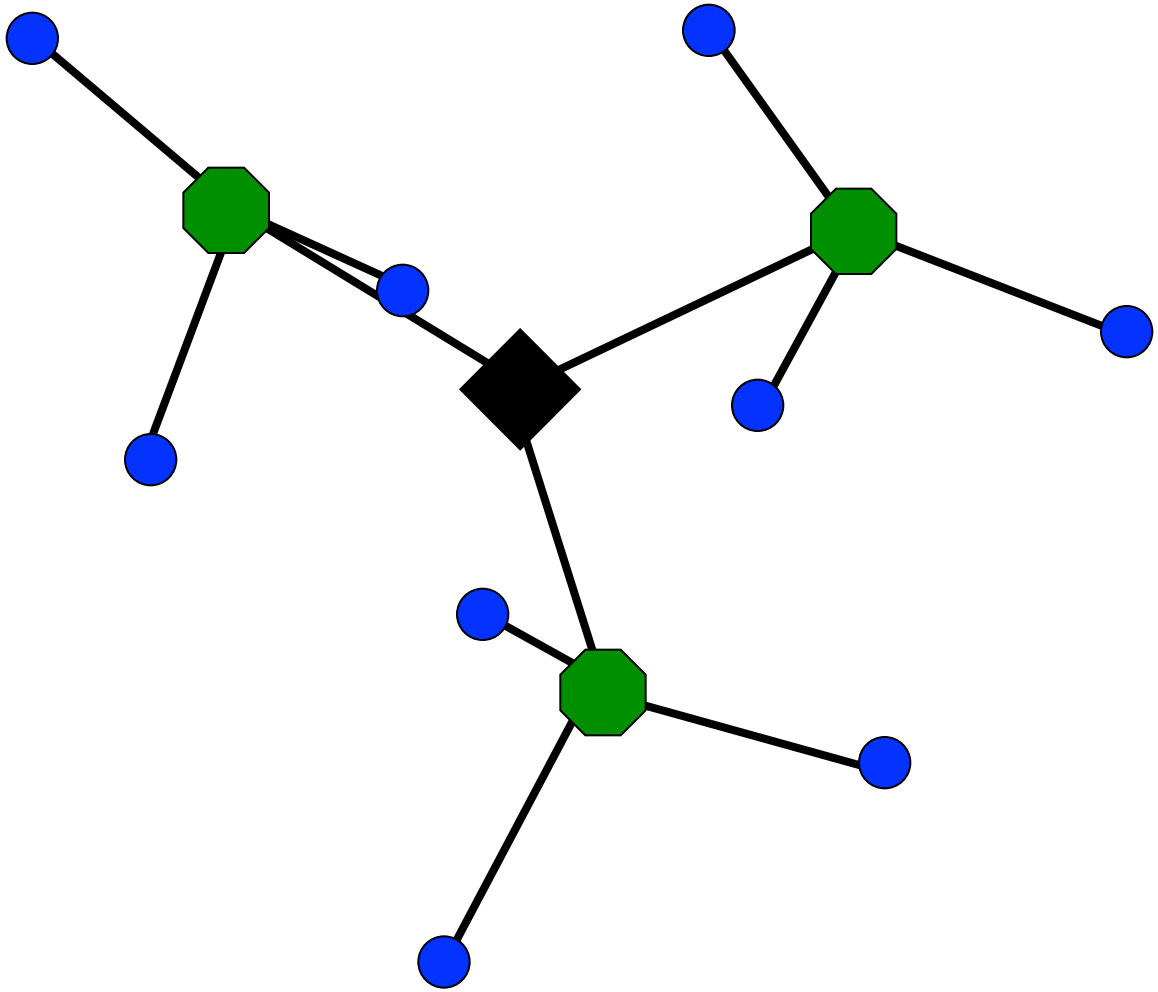
Perform hierarchical k-means clustering



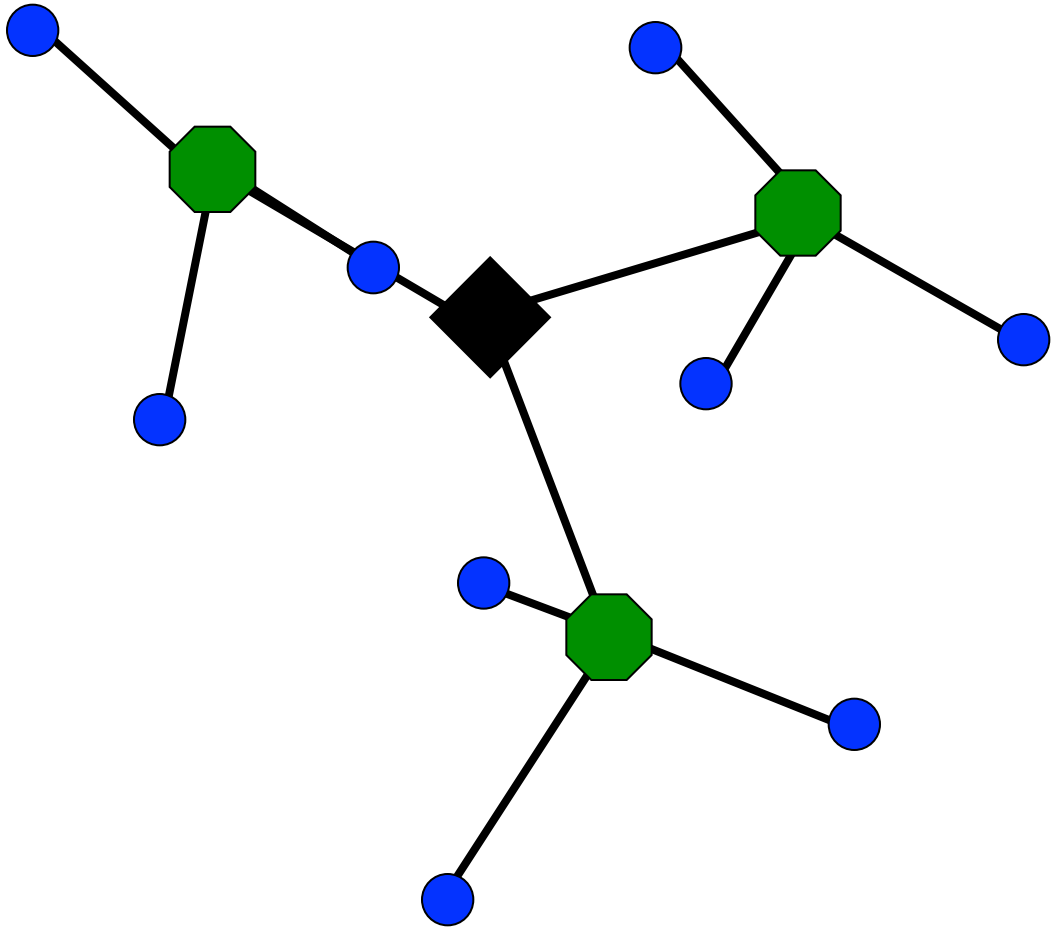
Slide from D. Nister



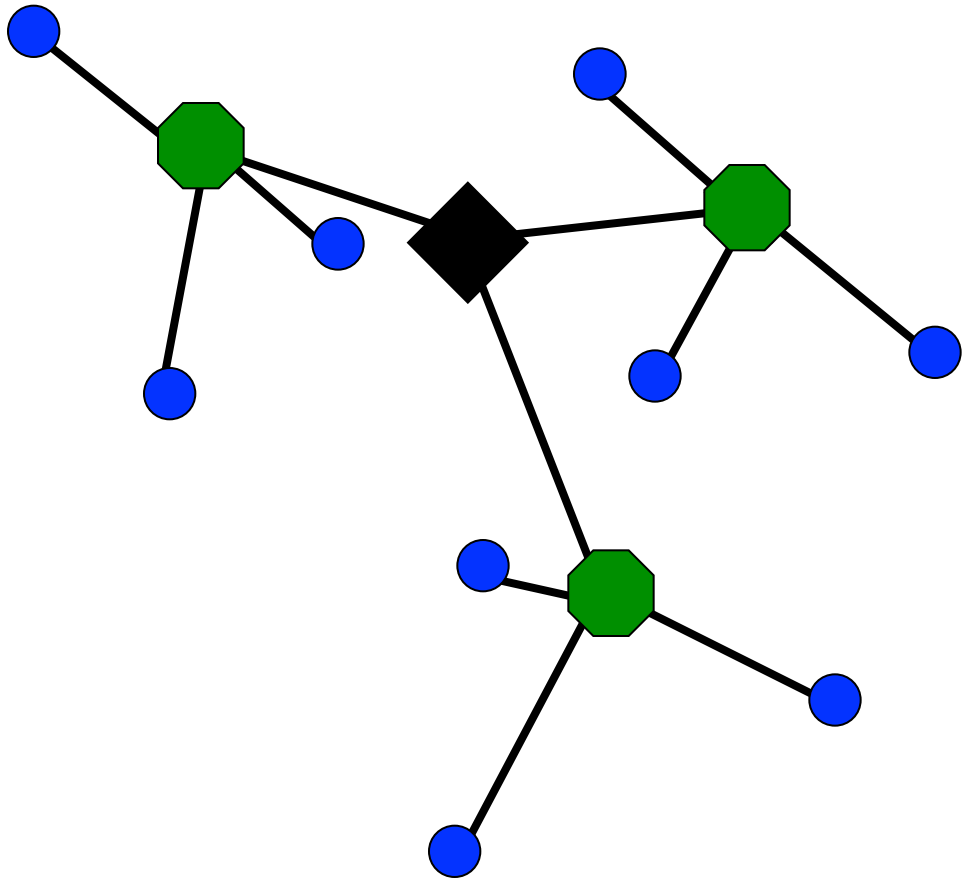
Slide from D. Nister



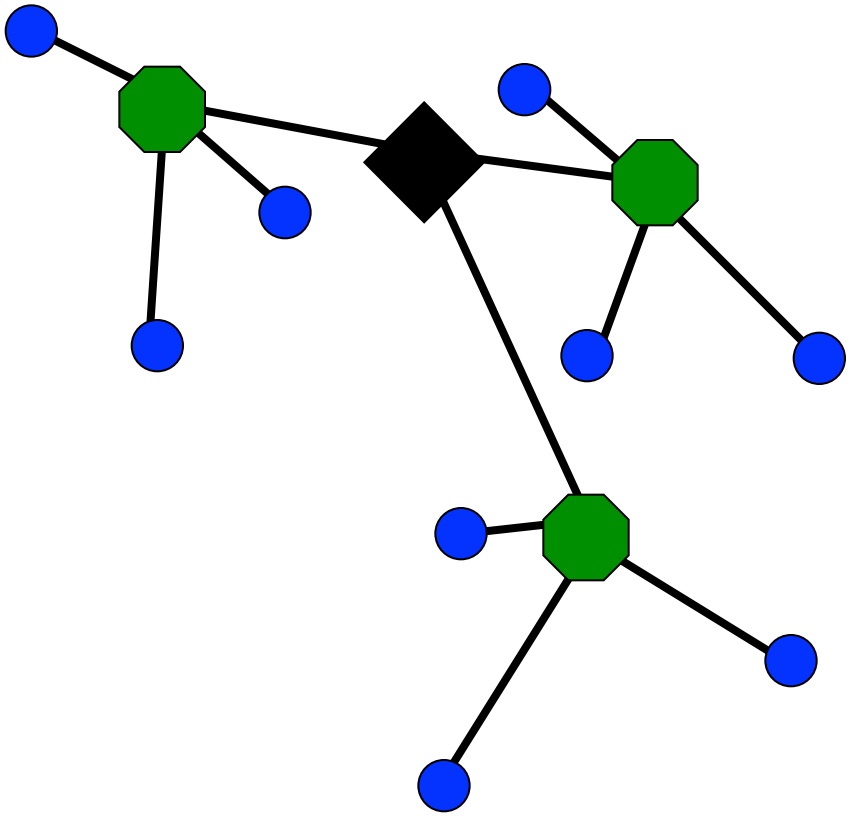
Slide from D. Nister



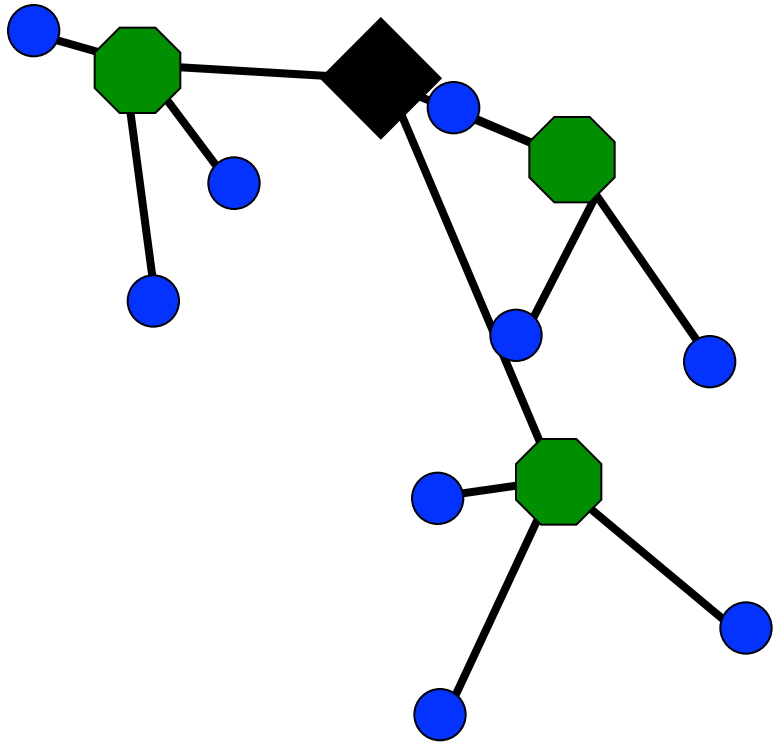
Slide from D. Nister



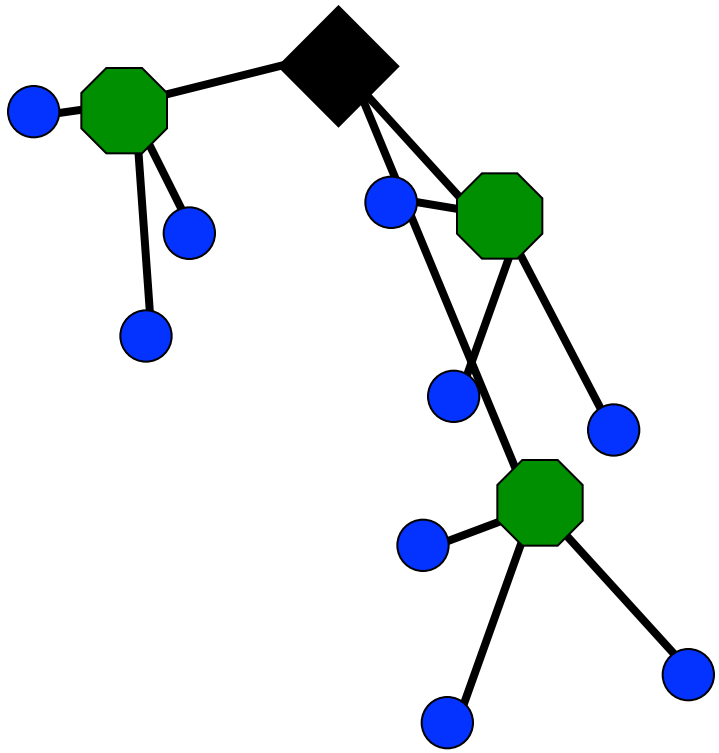
Slide from D. Nister



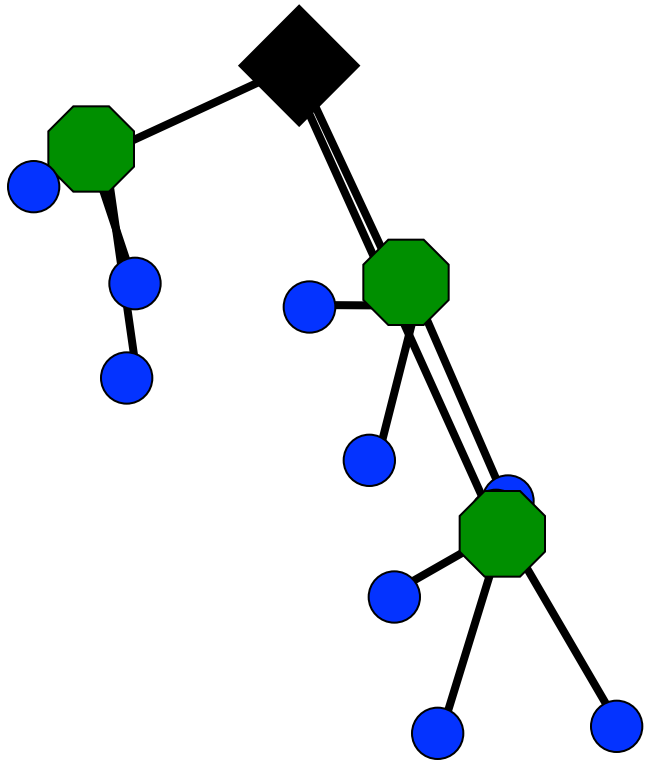
Slide from D. Nister



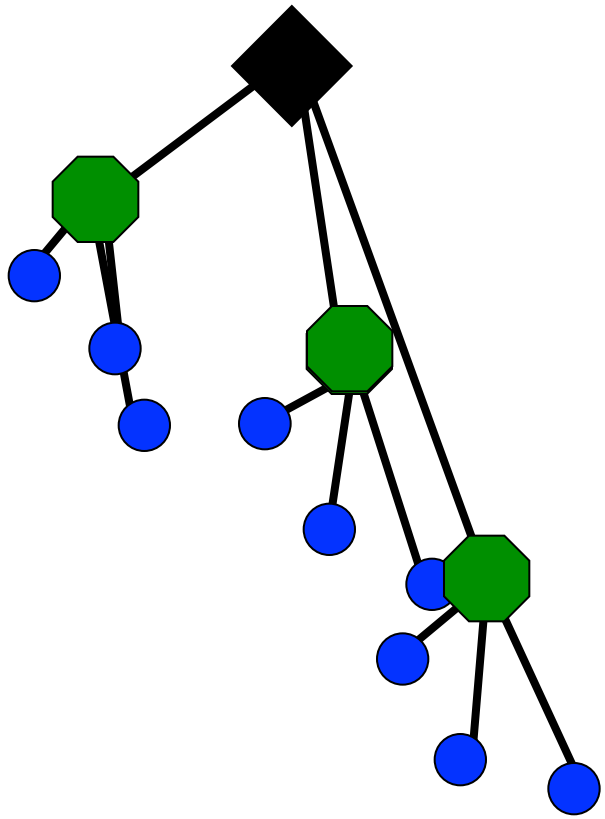
Slide from D. Nister



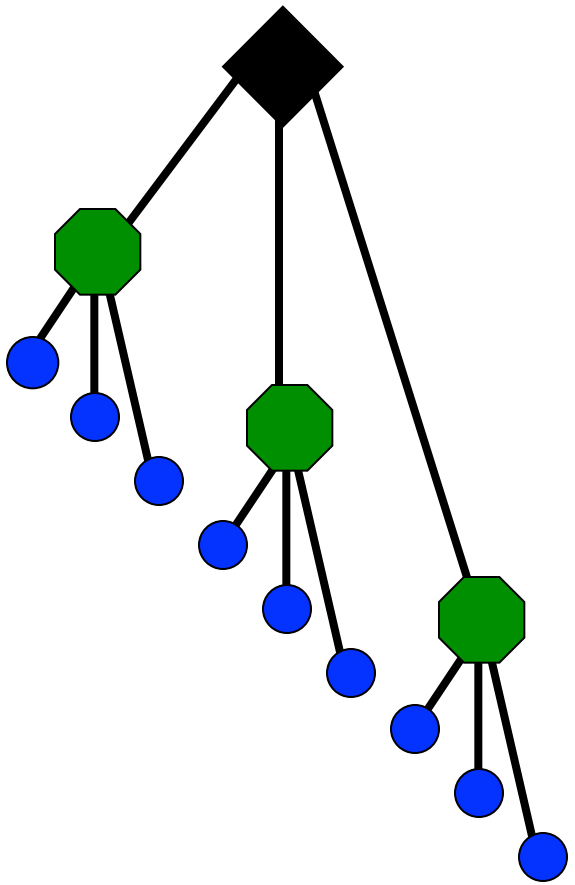
Slide from D. Nister



Slide from D. Nister



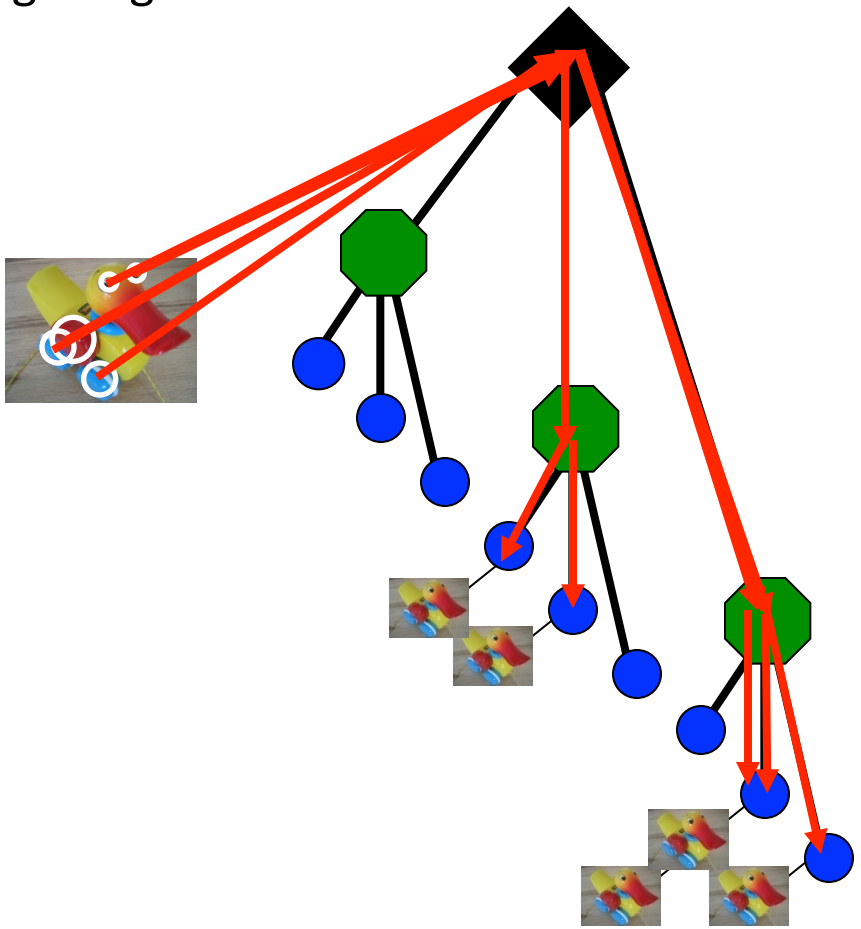
Slide from D. Nister



As the vocabulary tree is formed and is on-line, new images can be inserted in the database.

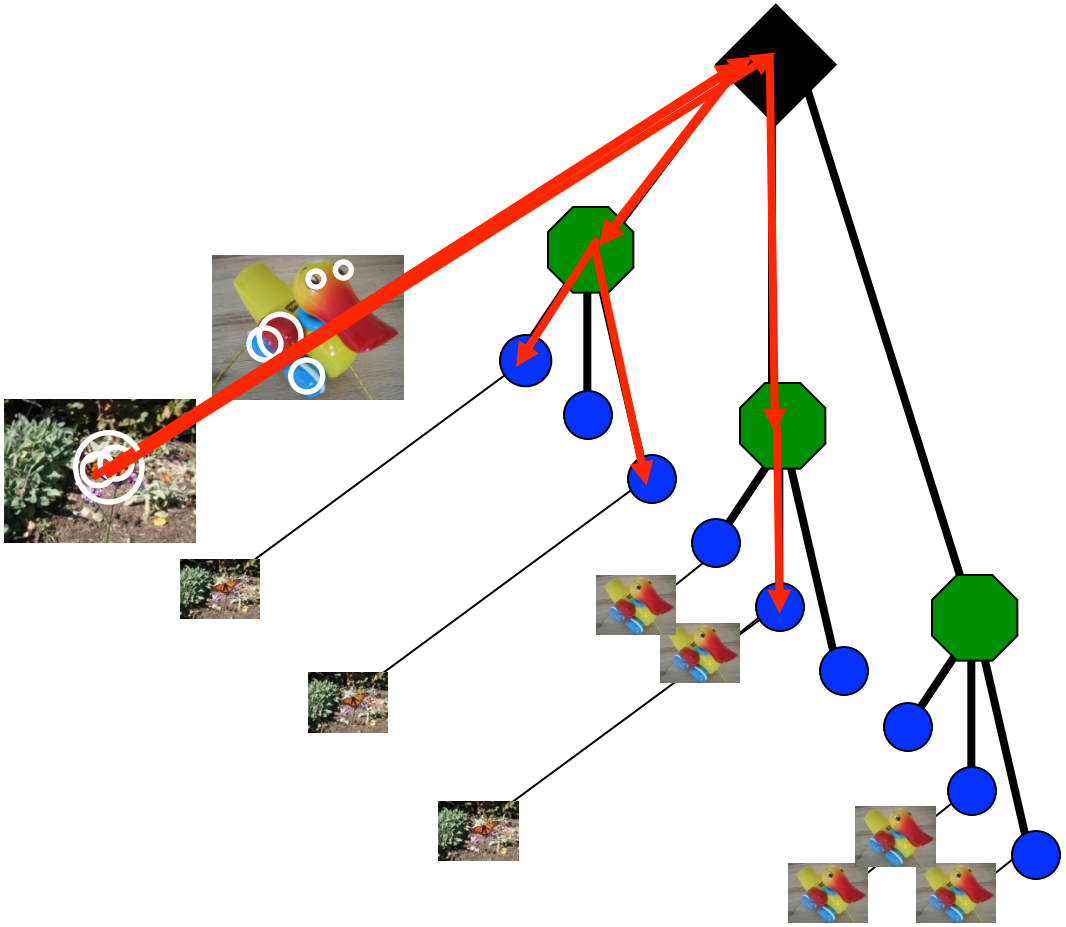
Slide from D. Nister

Adding images to the tree

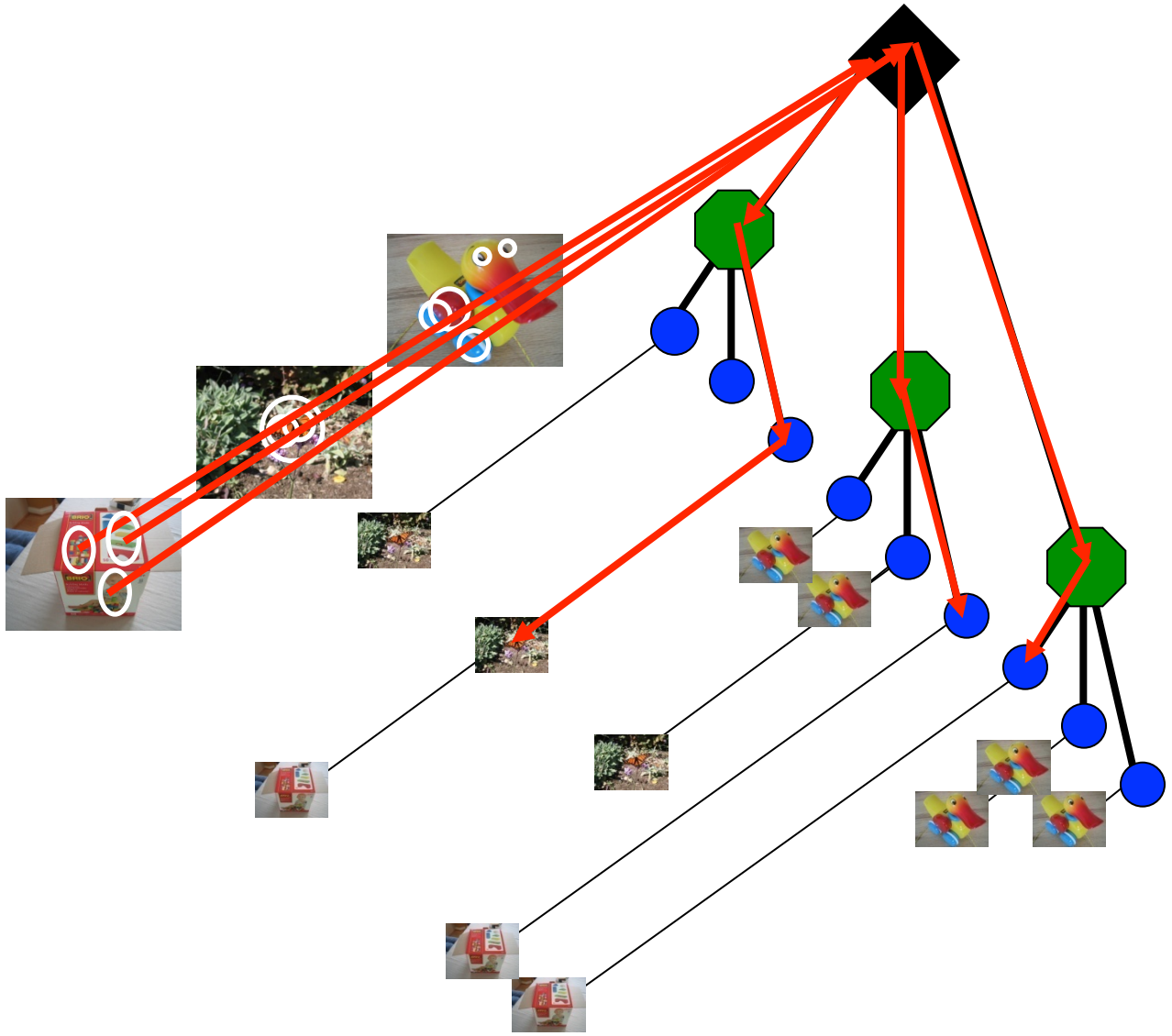


- Adding an image to the database requires the following steps:
  - Image feature descriptors are computed.
  - Each descriptor vector is dropped down from the root of the tree and quantized into a path down the tree

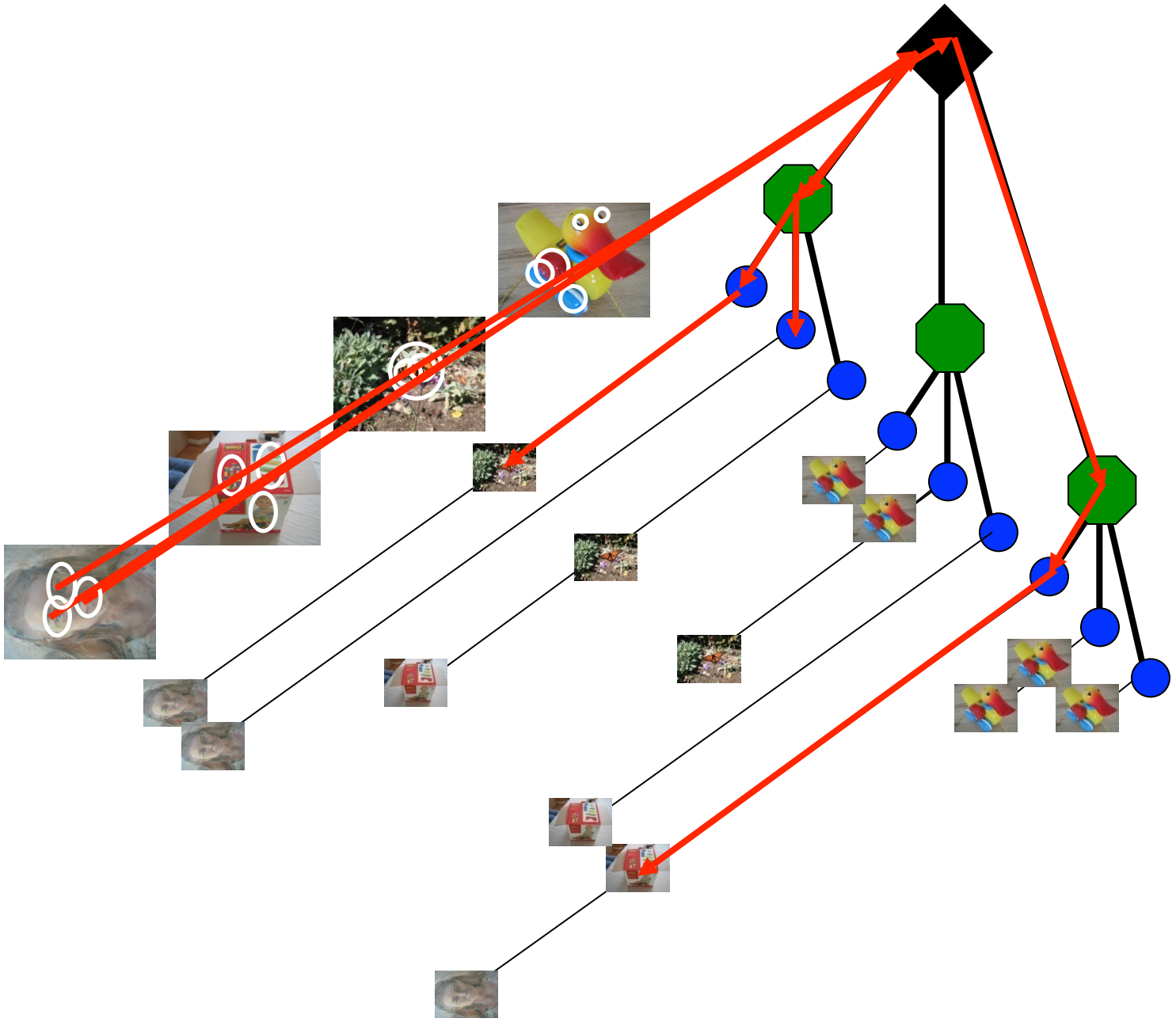
Slide from D. Nister



Slide from D. Nister



Slide from D. Nister



Slide from D. Nister

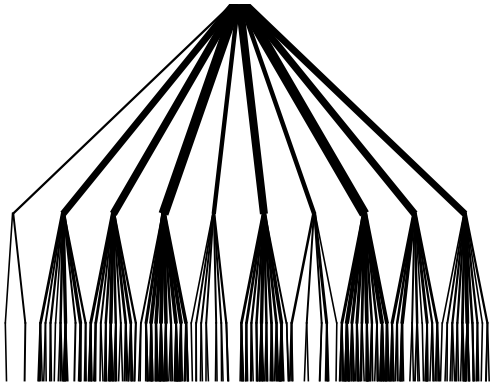
Querying with Vocabulary Tree

- In the online phase, each descriptor vector is propagated down the tree by at each level comparing the descriptor vector to the k candidate cluster centers (represented by k children in the tree) and choosing the closest one.

- k dot products are performed at each level, resulting in a total of kL dot products, which is very efficient if k is not too large. The path down the tree can be encoded by a single integer and is then available for use in scoring.

- The relevance of a database image to the query image based on how similar the paths down the vocabulary tree are for the descriptors from the database image and the query image. The scheme assigns weights to the tree nodes and defines relevance scores associated to images.

Paths of the tree for one image with 400 features

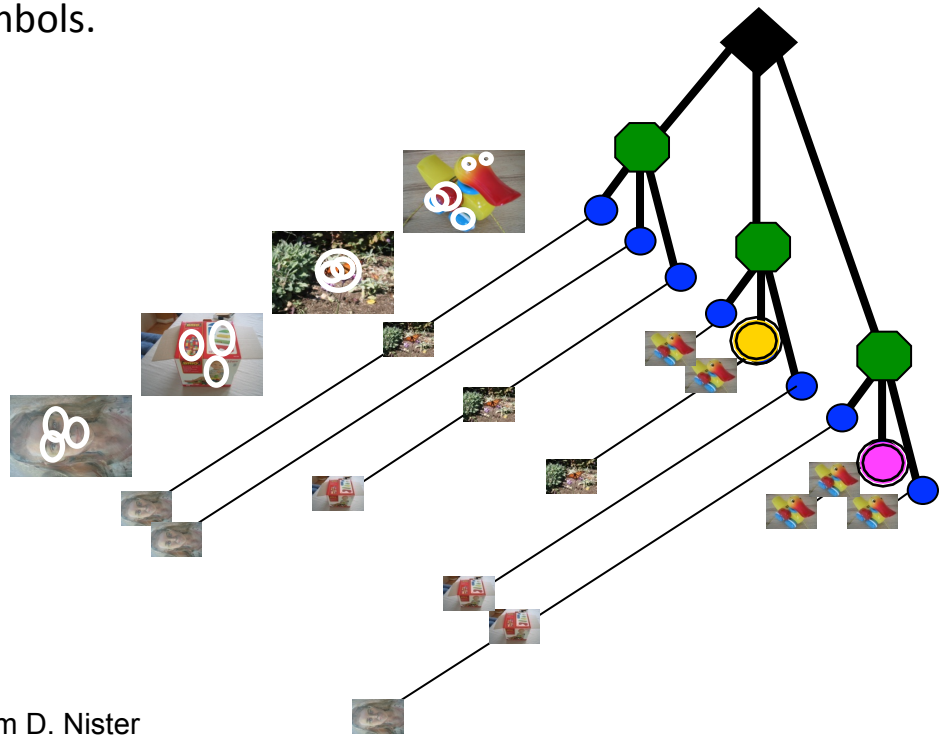


Scoring

- At each node i a weight w_i is assigned that can be defined according to one of different schemes:
  - a constant weighting scheme $w_i = k$
  - an *entropy weighting* scheme: $w_i = log\left(\frac{N}{N_i}\right)$ (*inverse document frequency*)

    where N is the number of database images and N_i is the number of images with at least one descriptor vector path through node i

- It is possible to use stop lists, where w_i is set to zero for the most frequent and/or infrequent symbols.



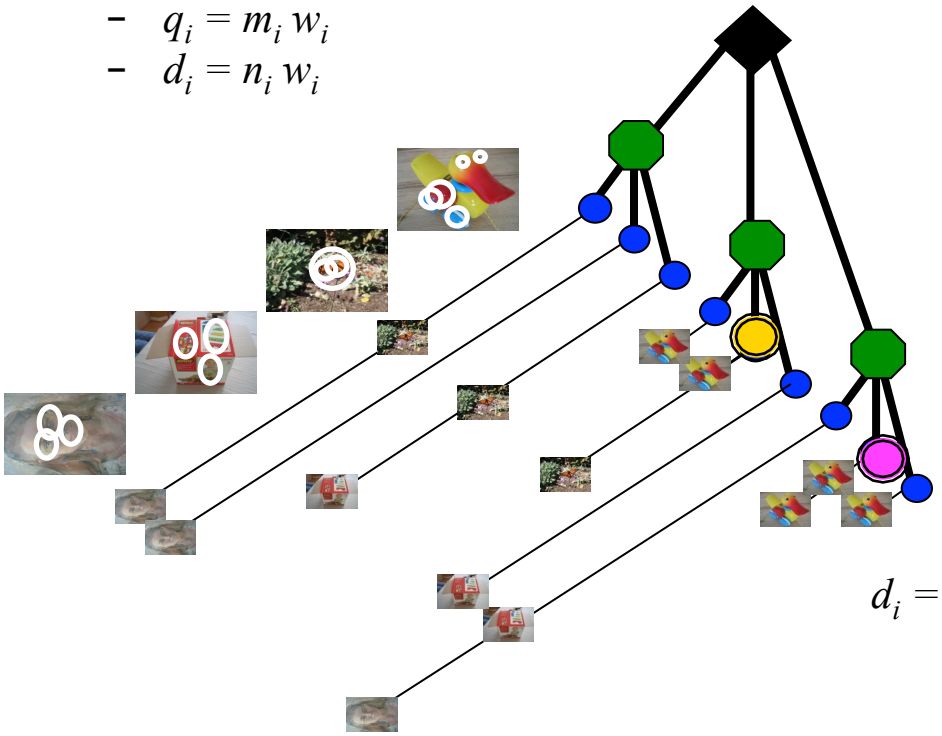
Node score

N=4 N_i=2 → w = log(2)

N=4 N_i=1 → w = log(4)

Image from D. Nister

- Query q_i and database vectors d_i are defined according to the assigned weights as :
  - $q_i = m_i w_i$
  - $d_i = n_i w_i$



where m_i is the number of the descriptor vectors of the query with a path along the node i and w_i its weight

n_i is the the number of the descriptor vectors of each database image with a path along the node i.

$d_i = n_i w_i$ → S= 2 log(2) S =2 log(4)

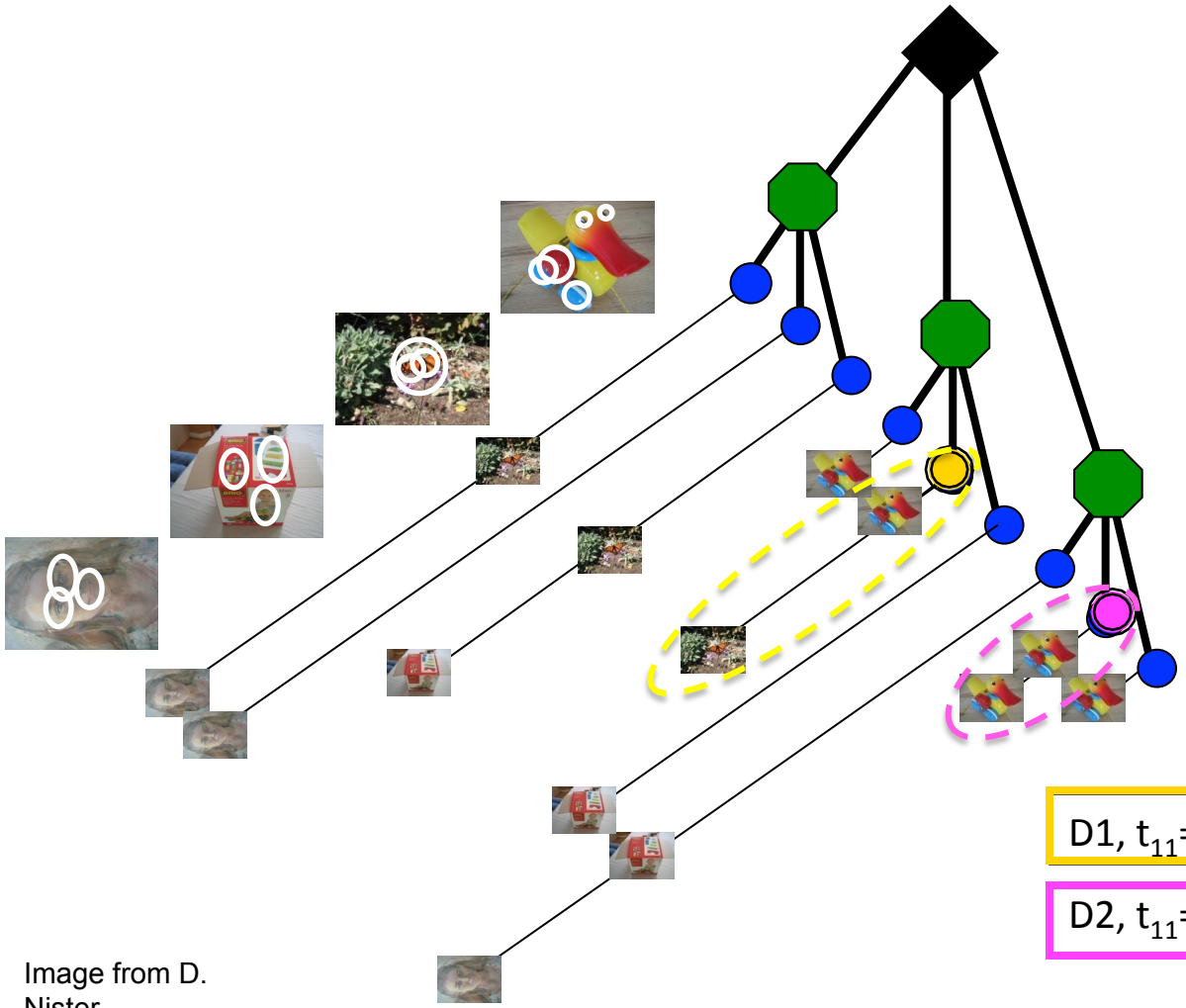
- Each database image is given a relevance score based on the L1 normalized difference between the query and the database vectors

$$s(q,d) = \| \frac{q}{\| q \|} - \frac{d}{\| d \|} \|$$

- Scores for the images in the database are accumulated. The winner is the image in the database with the most common information with the input image.

Inverted file index

- To implement scoring efficiently, an inverted file index is associated to each node of the vocabulary tree (the inverted file of inner nodes is the concatenation of it's children's inverted files).
- Inverted files at each node store the id-numbers of the images in which a particular node occurs and the *term frequency* of that image. Indexes back to the new image are then added to the relevant inverted files.



n_{id} = n. times visual word i appears in doc d

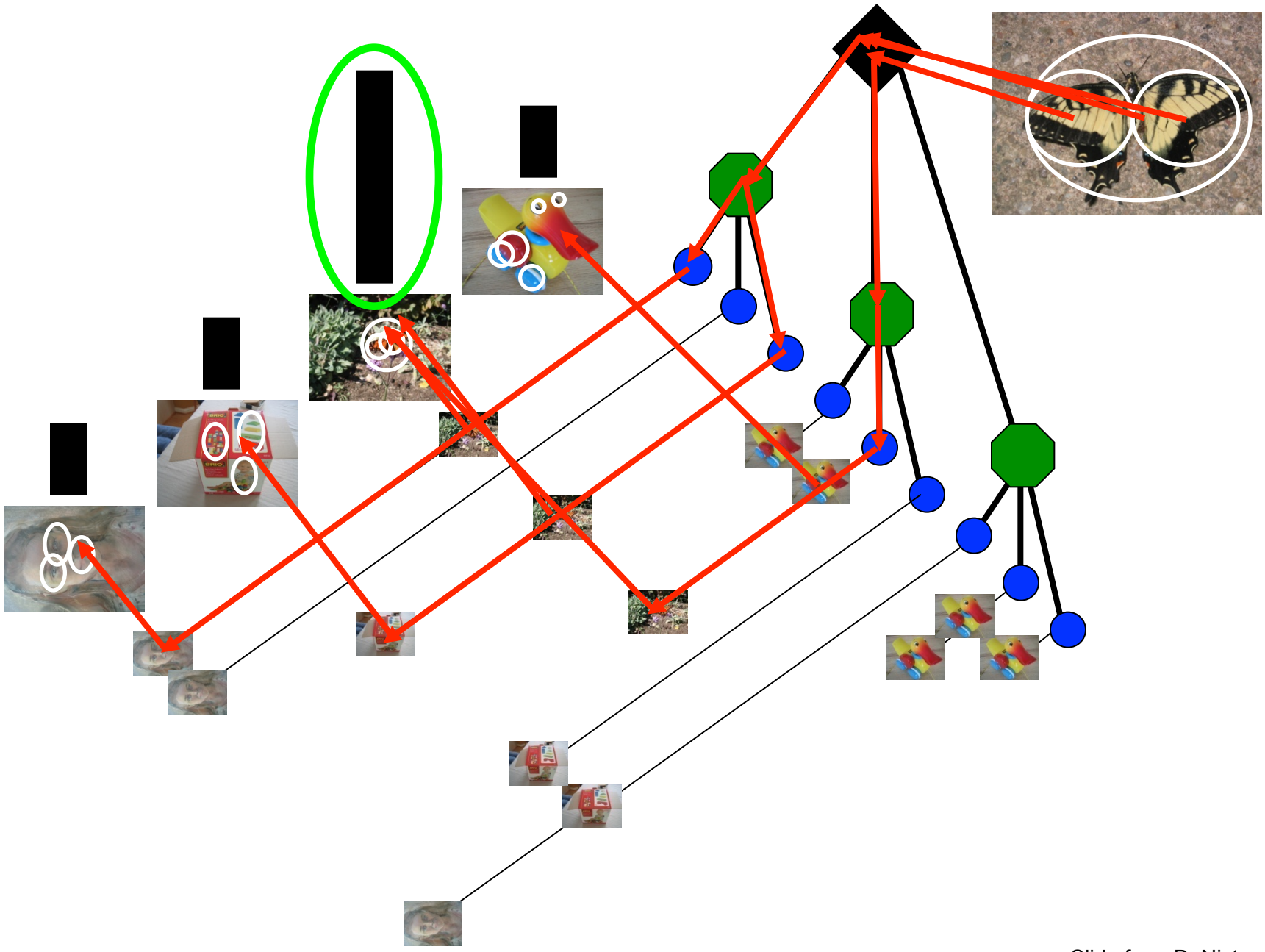
n_d = n. visual words in doc d

$$t_i = \frac{n_{id}}{n_d}$$

Inverted file index

D1, $t_{11}=1/n_1$, Img1	D1, $t_{21}=1/n_2$, Img2
D2, $t_{11}=2/n_1$, Img1	

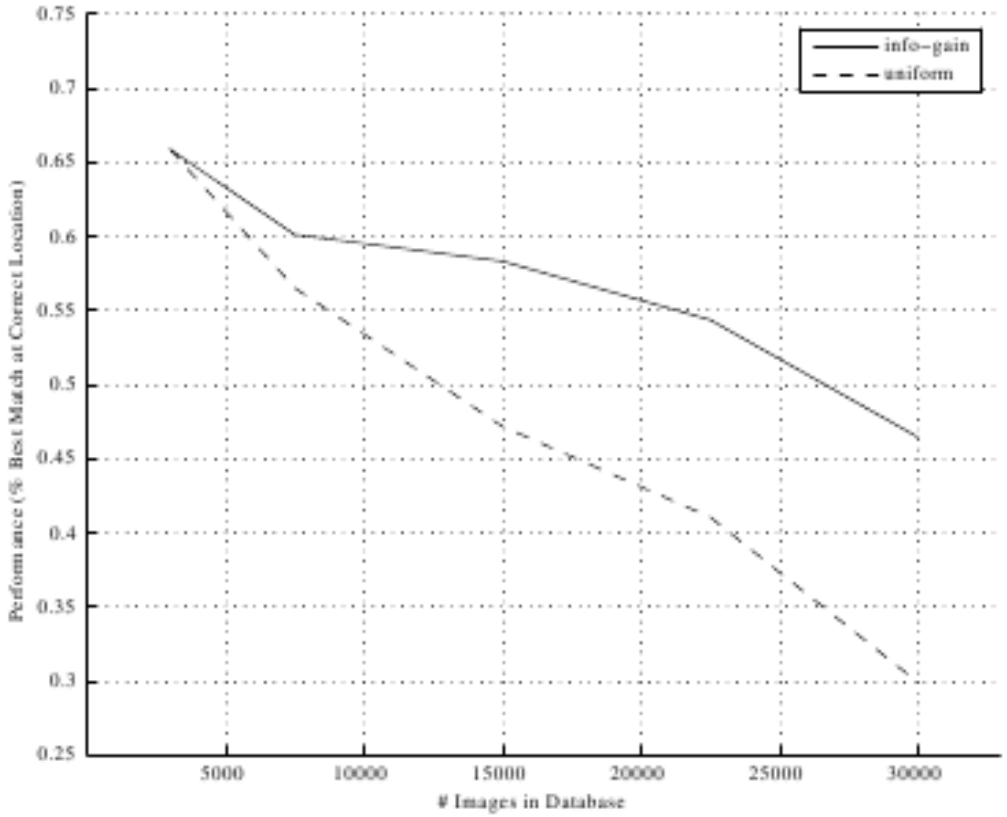
Image from D. Nister



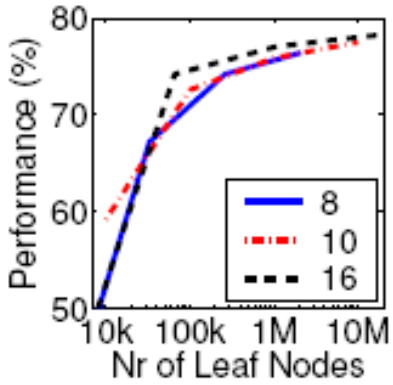
Slide from D. Nister

Performance considerations

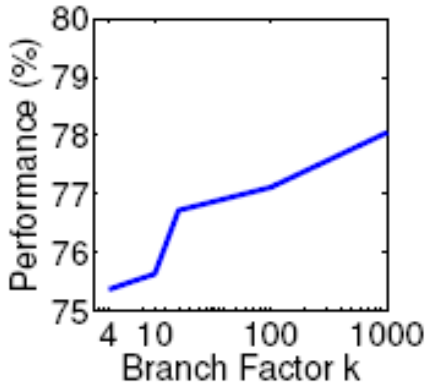
- Performance of vocabulary tree is largely dependent upon its structure. Most important factors to make the method effective are:
  - A large vocabulary tree (16M words against 10K of Video Google)
  - Using informative features vs. uniform (compute *information gain* of features and select the most informative to build the tree i.e. features found in all images of a location, features not in any image of another location)



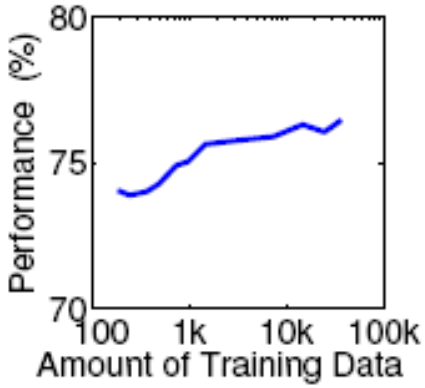
Performance figures on 6376 images



Performance increases significantly with the number of leaf nodes



Performance increases with the branch factor *k*



Performance increases when the amount of training data grows

From Tommasi