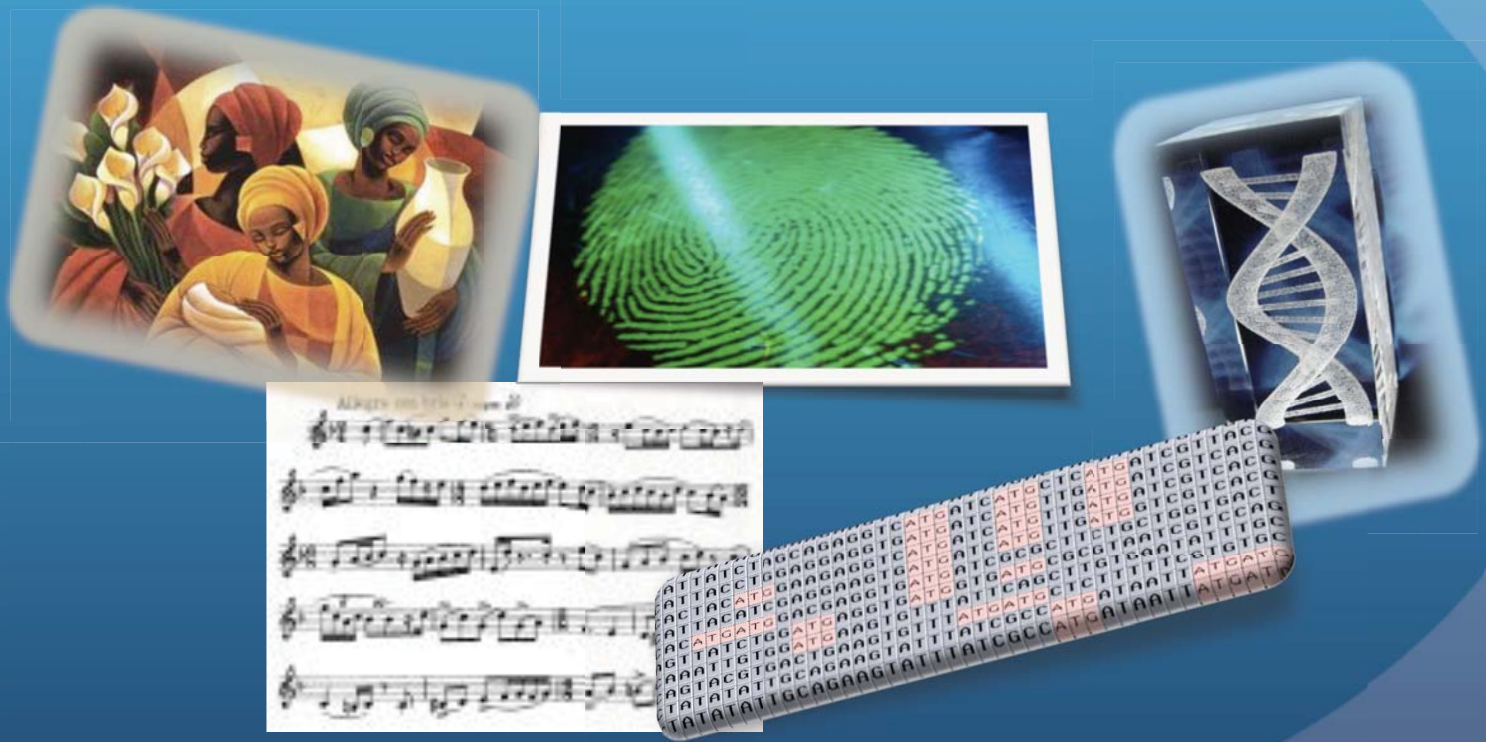
Dynamic Spatial Approximation Trees for Massive Data

Gonzalo Navarro – Universidad de Chile

Nora Reyes – Universidad Nacional de San Luis

Introduction and Motivation

- Similarity searching has applications in many fields, such as multimedia databases, text retrieval, etc.



Introduction and Motivation

- To answer similarity queries the dataset is preprocessed so as to build an index that reduces query time.
- Most of the existing indexes are static.
- There are also many interesting databases where:

Introduction and Motivation

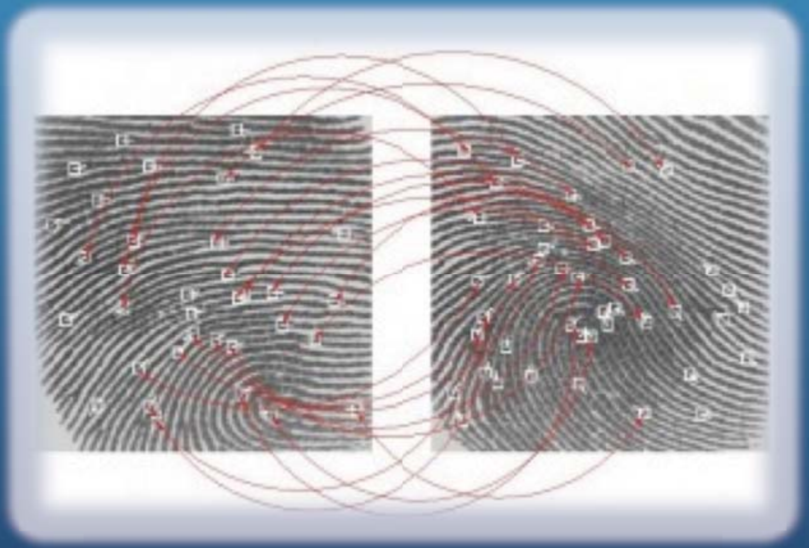
The objects are too large

Introduction and Motivation

Introduction and Motivation

- So, the objects or the index itself cannot fit in main memory.

- Similarity computation can be expensive,



But we cannot disregard disk costs

Introduction and Motivation

- From the few dynamic indexes, even fewer work well in secondary memory.
- We introduce a dynamic index aimed at secondary memory, based on *the Dynamic Spatial Approximation Tree* (dsa-tree).
- It gives an attractive tradeoff between *memory usage*, *construction time*, *and search performance*.

Introduction and Motivation

- Our secondary memory versions: *dsa*-tree* and *dsa+-tree* retain these good features, and perform well in secondary memory.

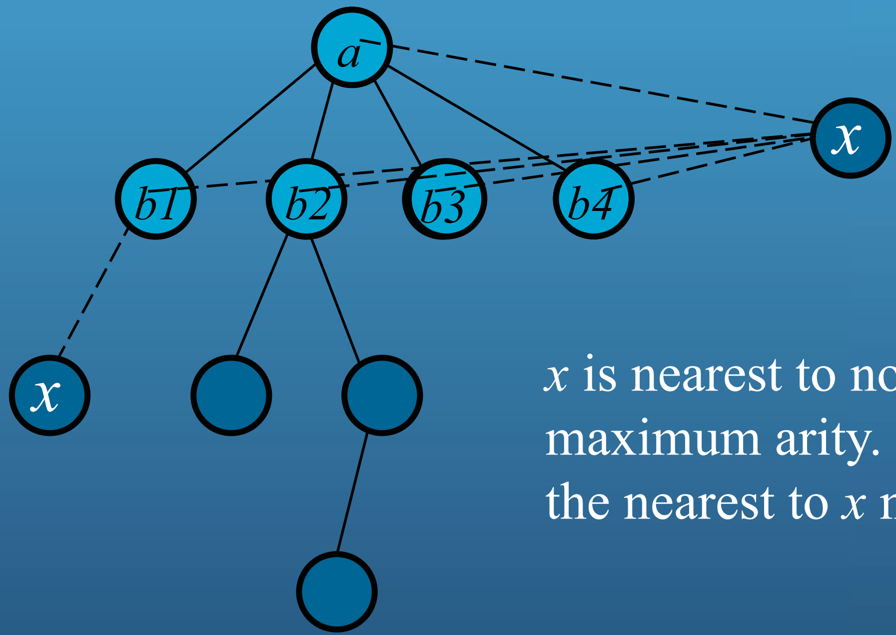
- Our structures achieve very good disk page utilization and are competitive.

Basic Concepts

- The distance is assumed to be expensive to compute, but also is the number of I/O operations.
- Given a dataset with *n* objects of total size *M* and disk page size *B*: queries need to perform at most *n* distance *evaluations* and *M/B* I/O operations.
- The goal is *to answer queries with as few distance evaluations and I/Os as possible*.

Dynamic Spatial Approximation Trees

- Insertions:



$A = 4$

x is nearest to node a, but a has maximum arity. So, it chooses the nearest to x neighbor of a.

Secondary Memory

- Our implementation maintains exactly the same structure and carries out the same distance evaluations of the main-memory version.
- For any *a*, the set N(*a*) will be packed together in a disk page.
- MaxArity must be such that a disk page can store at least two *N*() lists of maximum length.

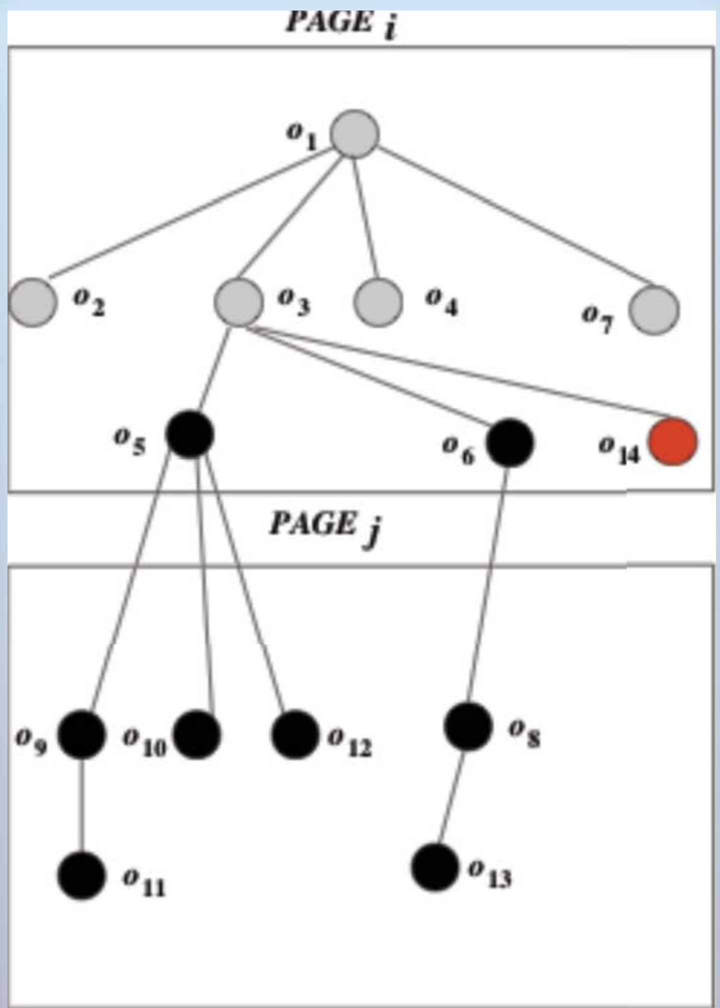
Secondary Memory

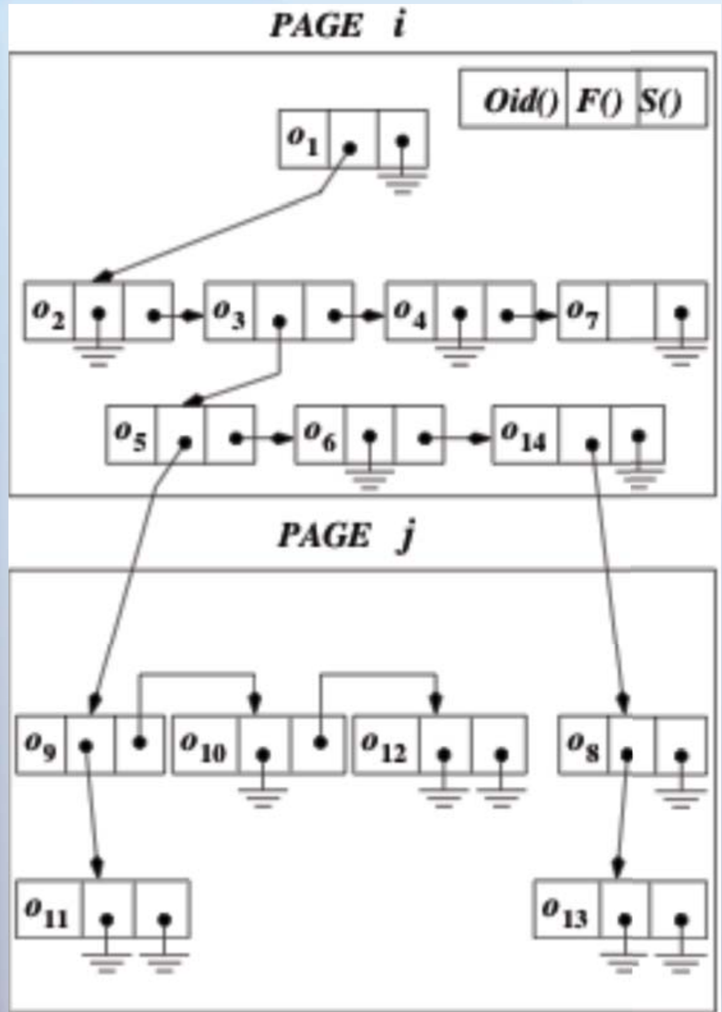
- To avoid disk underutilization, we will allow several nodes to share a single disk page.
- We define insertion policies that maintain a partitioning of the tree into disk pages.
- We will guarantee a minimum average disk page utilization of 50%, and achieve much more in practice.

Secondary Memory

- We represent the children or neighbors of a node as a linked list.
- This allows making most changes to $N(a)$ without accessing a, which might be in another disk page.
- Far pointers have two parts: the page and the node offset inside that page.

Secondary Memory



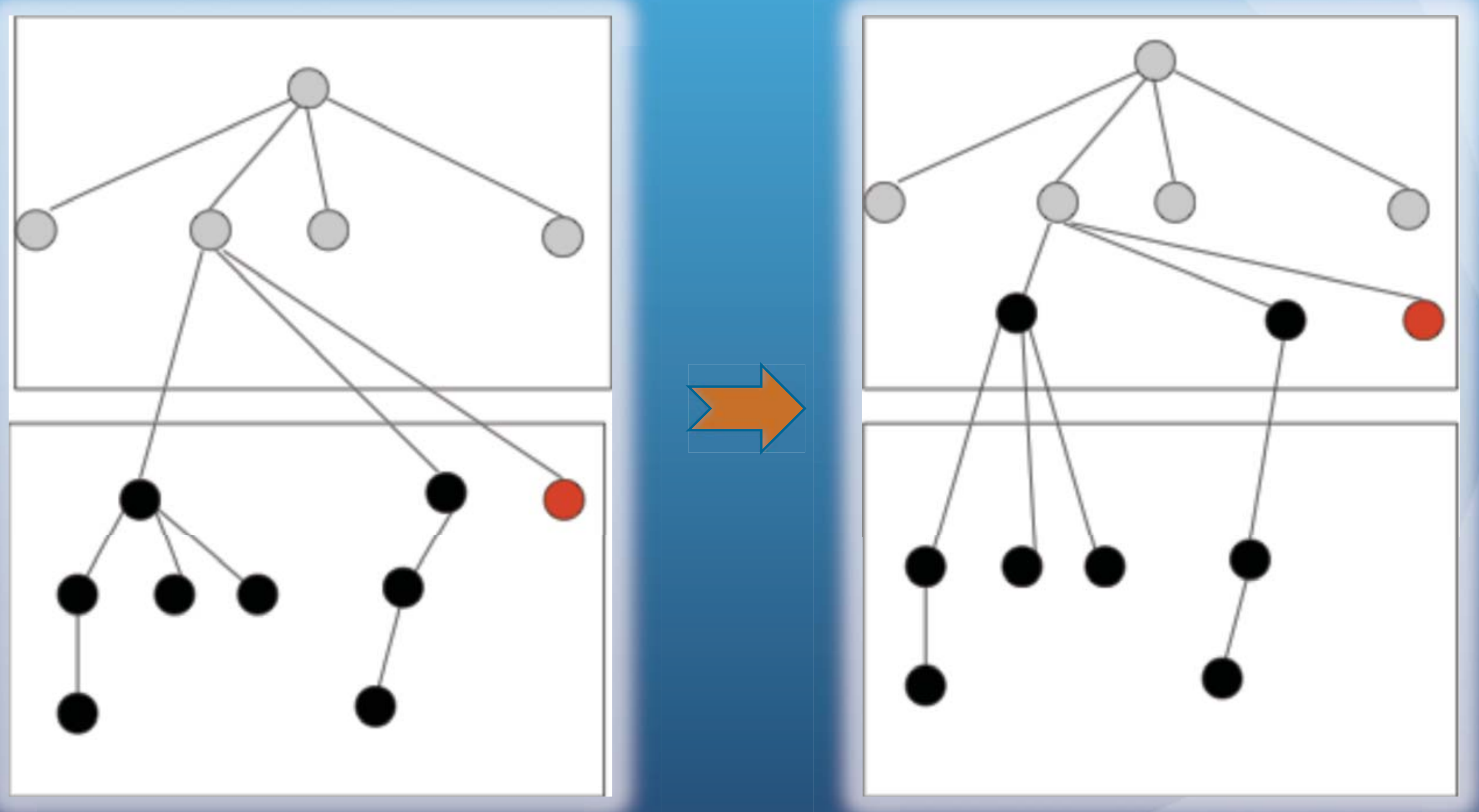


Secondary Memory

- When the insertion of x in $N(a)$ produces a page overflow, we try out the following strategies, in order:
    - Move N(a) to parent
    - Vertical split
    - Horizontal split

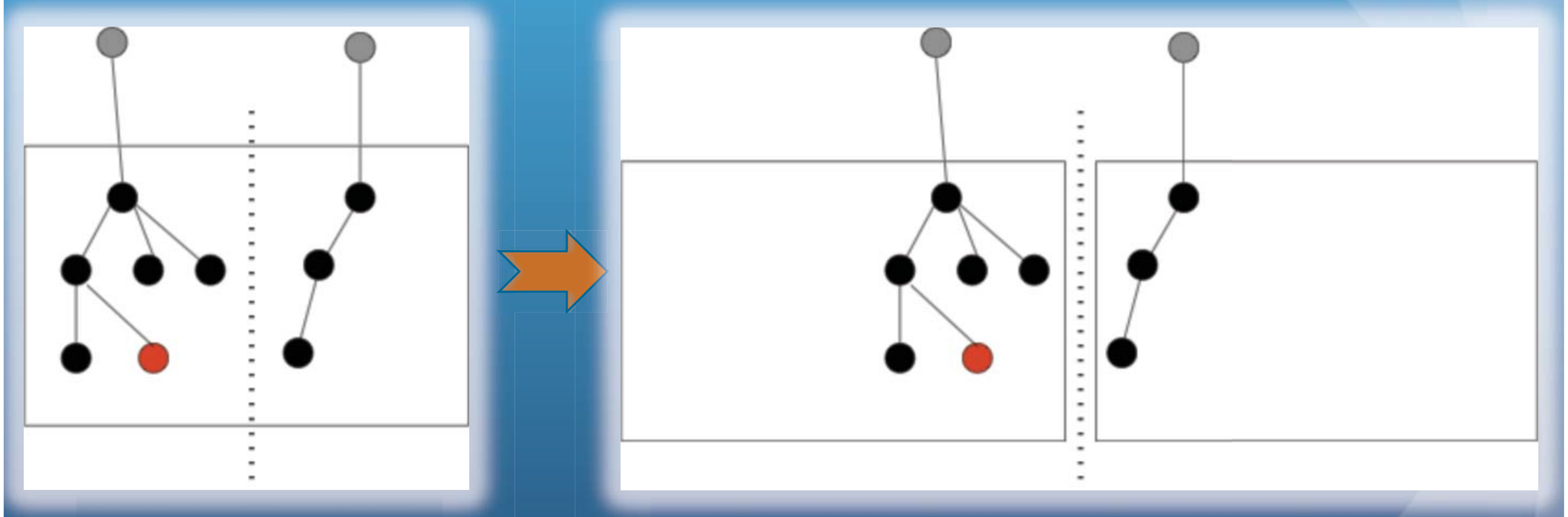
Secondary Memory

- 1st (move to parent)



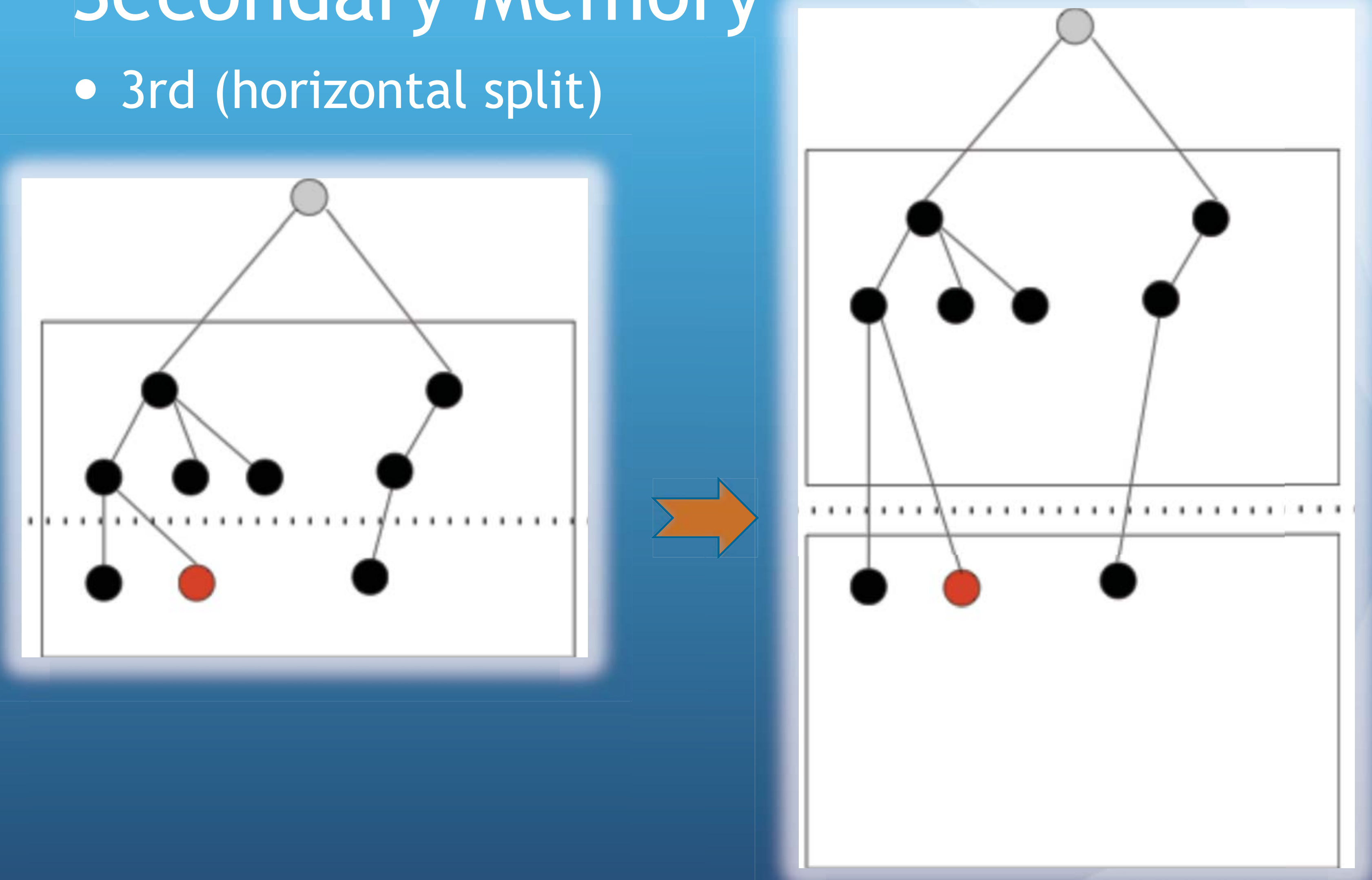
Secondary Memory

- 2nd (vertical split)



Secondary Memory

- 3rd (horizontal split)



Secondary Memory

- The previous operations do not yet ensure that disk pages are at least half-full.
- To enforce the desired fill ratio, we will not allow indiscriminate creation of new pages.
- We will point all the time to one disk page, which will be the only one allowed to be less than half-full.

A Heuristic Variant

- The dsa*-tree we have described ensures 50% fill ratio, but this has a price in terms of compactness.

- dsa+-tree, which tries to achieve better locality at the price of not ensuring 50% fill ratio.

- In the dsa+-tree each subtree root at a page maintains a far pointer to its parent, and knows its global tree level.

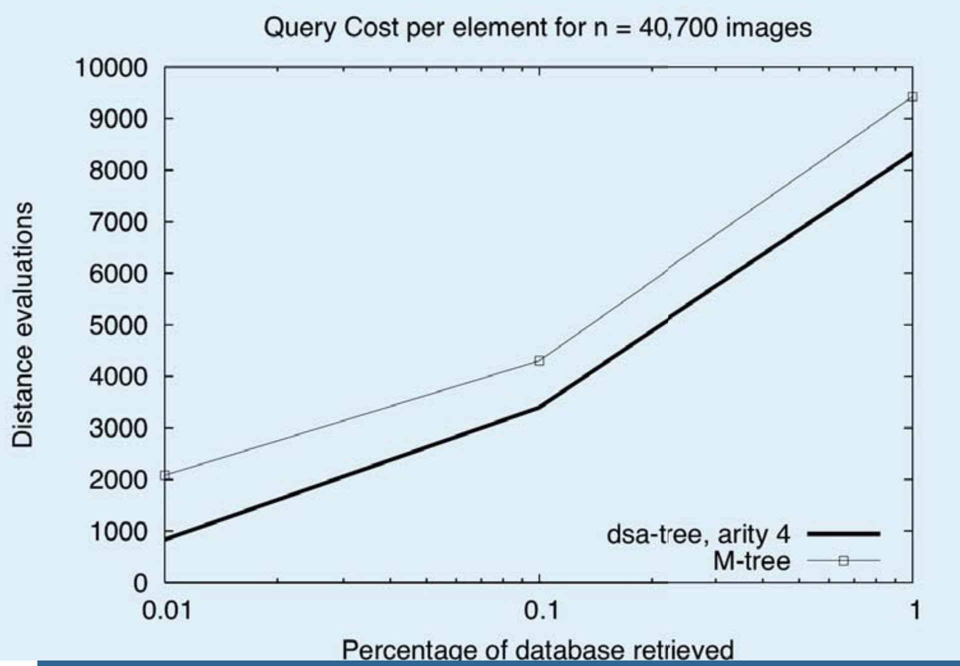
Experimental Results

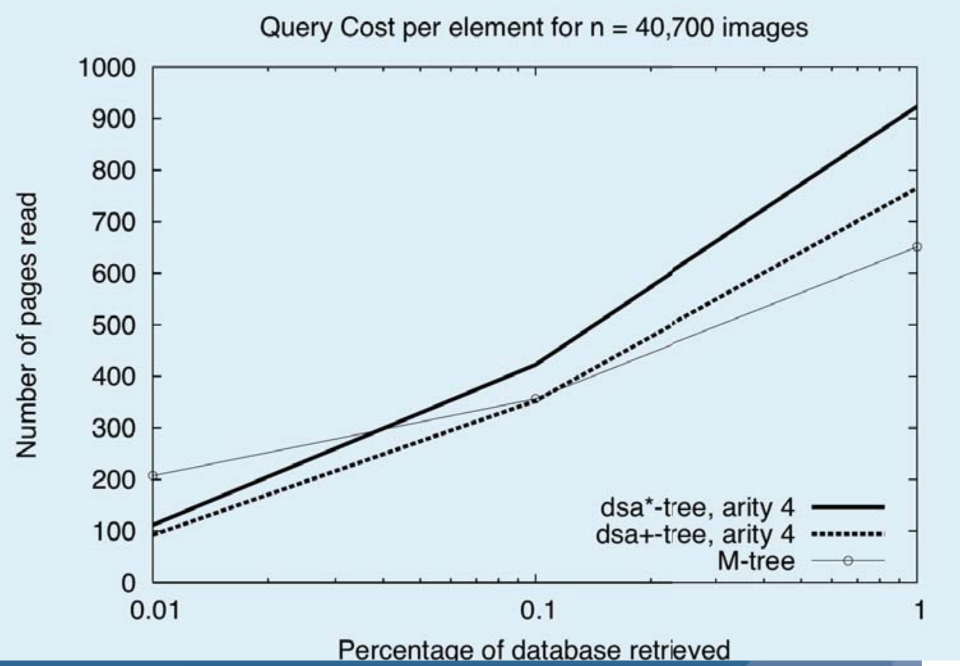
- We have selected four widely different metric spaces, all from the SISAP Metric Library.
- The disk page size is 4KB.
- All our results are averaged over 10 index constructions using different permutations of the datasets.

Experimental Results

Dataset	Fill Ratio		Total Pages Used		
	dsa*-tree	dsa+-tree	dsa*-tree	dsa+-tree	M-tree
Words	83%	66%	904	1,536	1,608
Documents	84%	68%	12	22	31
Images	80%	67%	1,271	1,726	1,973
Histograms	75%	67%	18,781	21,136	31,791

Experimental Results





Conclusions

- We have presented two variants of the dsa-tree for secondary-memory, which retain the original tree structure.
- We have shown that the resulting structure achieves good space utilization and is competitive in distance computations and I/Os.
- The focus is on disk page layout policies that achieve competitive performance in terms of I/Os.

Thanks for your attention!

Contact Information

Gonzalo Navarro gnavarro@dcc.uchile.cl

Nora Reyes nreyes@unsl.edu.ar