SAARLAND UNIVERSITY
COMPUTER SCIENCE
Information Systems Group

Database Systems

WS 08/09

Prof. Dr. Jens Dittrich

Chair of Information Systems Group
http://infosys.cs.uni-saarland.de

Topics (2/6)

- indexing
  - one- and multidimensional
  - tree-structured
  - partition-based indexing
  - bulk-loading
  - main-memory indexing (continued)
  - hash-indexes
  - differential indexing
  - read-optimized indexing
  - write-optimized indexing
  - data warehouse indexing
  - text indexing: inverted files
  - (flash-indexing)

Other Main Memory Indexes.

Arrays

- everything in main memory => no DB-buffer => why not allocate everything as one big chunk of data => array
- a[1...n]
- usually some fixed type (int, long, etc.)
- or complex object type (Object, pointers or refs to objects)
- binary search simple:
  - just sort once
  - then binary search in log n time
- obvious problem:
  - inserts: how to make space for new data
  - solutions: same as for B^+-trees
  - only if that does not work: reorganize

Problem with Binary Search on Arrays

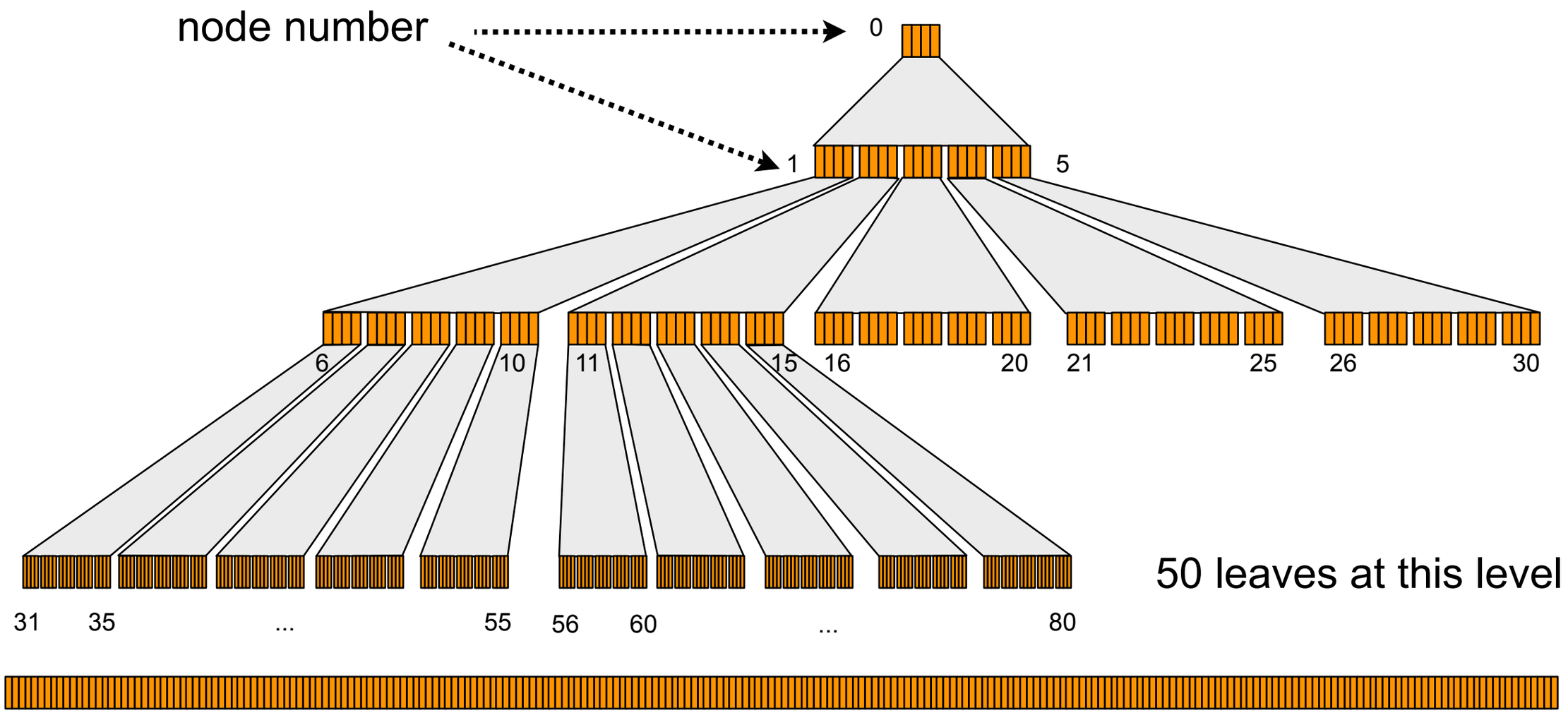
- many searches in the array result in a cache miss
- analogy: binary search on ordered list of data pages on external memory (see previous slides)
- worst case: number of cache misses is of the order of the number of key comparisons
- how to be better?

CSS trees

- main idea:
  - create additional directory structure **on top** of existing array
  - based on an (m+1)-ary tree
  - m is chosen such that a node fits exactly into a cache line
- result
  - $\log_{m+1} n$ cache misses for a query
  - instead of $\log_2 n$ cache misses as for array bin search
- two variants: full and level CSS-Trees
- we will look at full CSS-trees
- Literature: Jun Rao, Ken Ross: Cache Conscious Indexing for Decision-Support in Main Memory. VLDB 199.

Full CSS-Trees

m=4



15 leaves at this level

50 leaves at this level

65*4 elements in array

- each node or leaf contains m keys => m+1 fan-out
- no pointers stored whatsoever
- all nodes are sequentially stored starting from node 0 to node 80 in an **additional directory array**

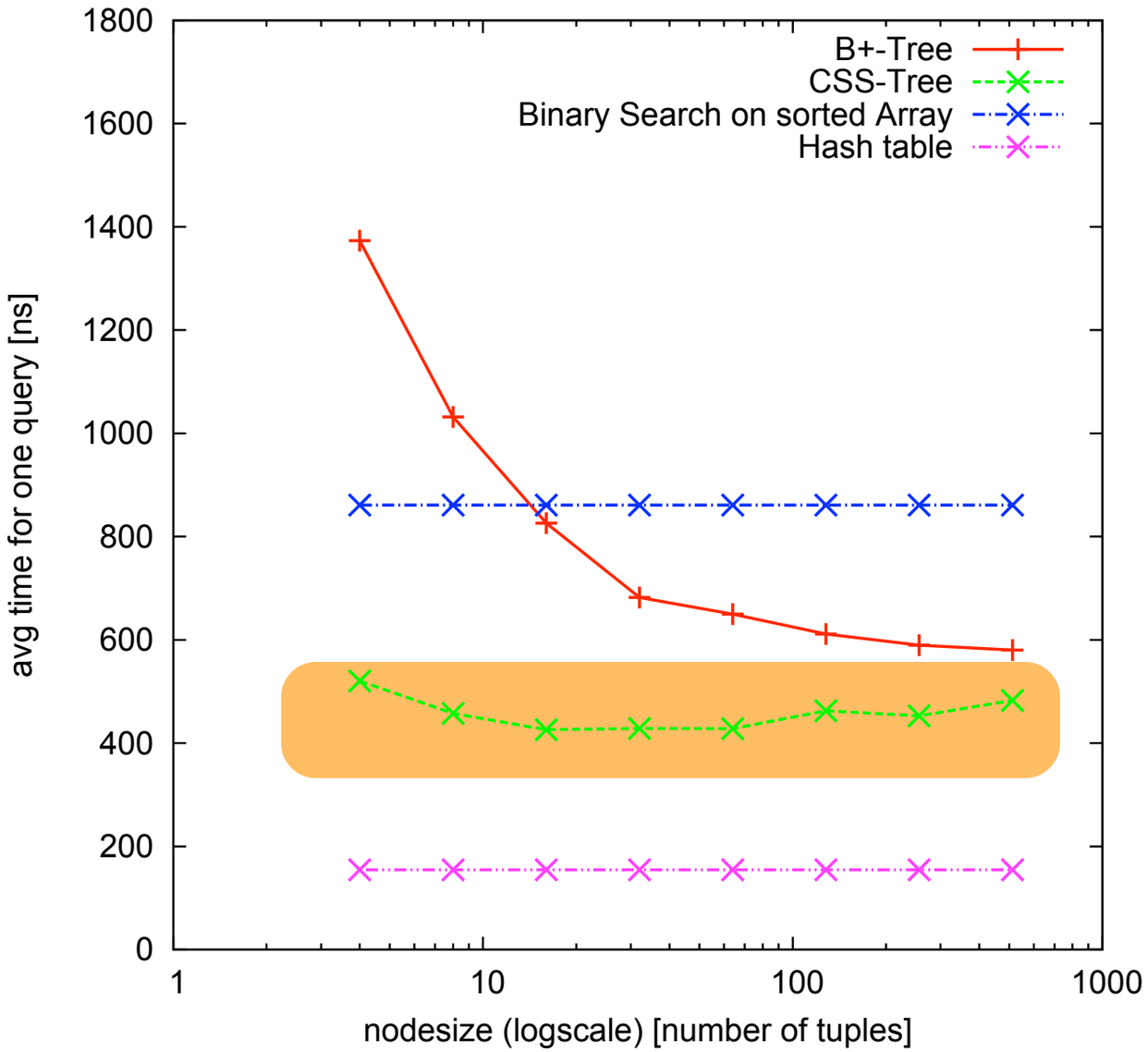
Properties

- expected $\log_{m+1} n$ cache misses
- instead of $\log_2 n$ cache misses as for array bin search
- if b is current node number, children are numbered from b(m+1)+1 to b(m+1)+(m+1)
- tree traversal is computed by computing node numbers directly
- some similarities to a heap storage
- not perfectly balanced
- different path may differ by one step

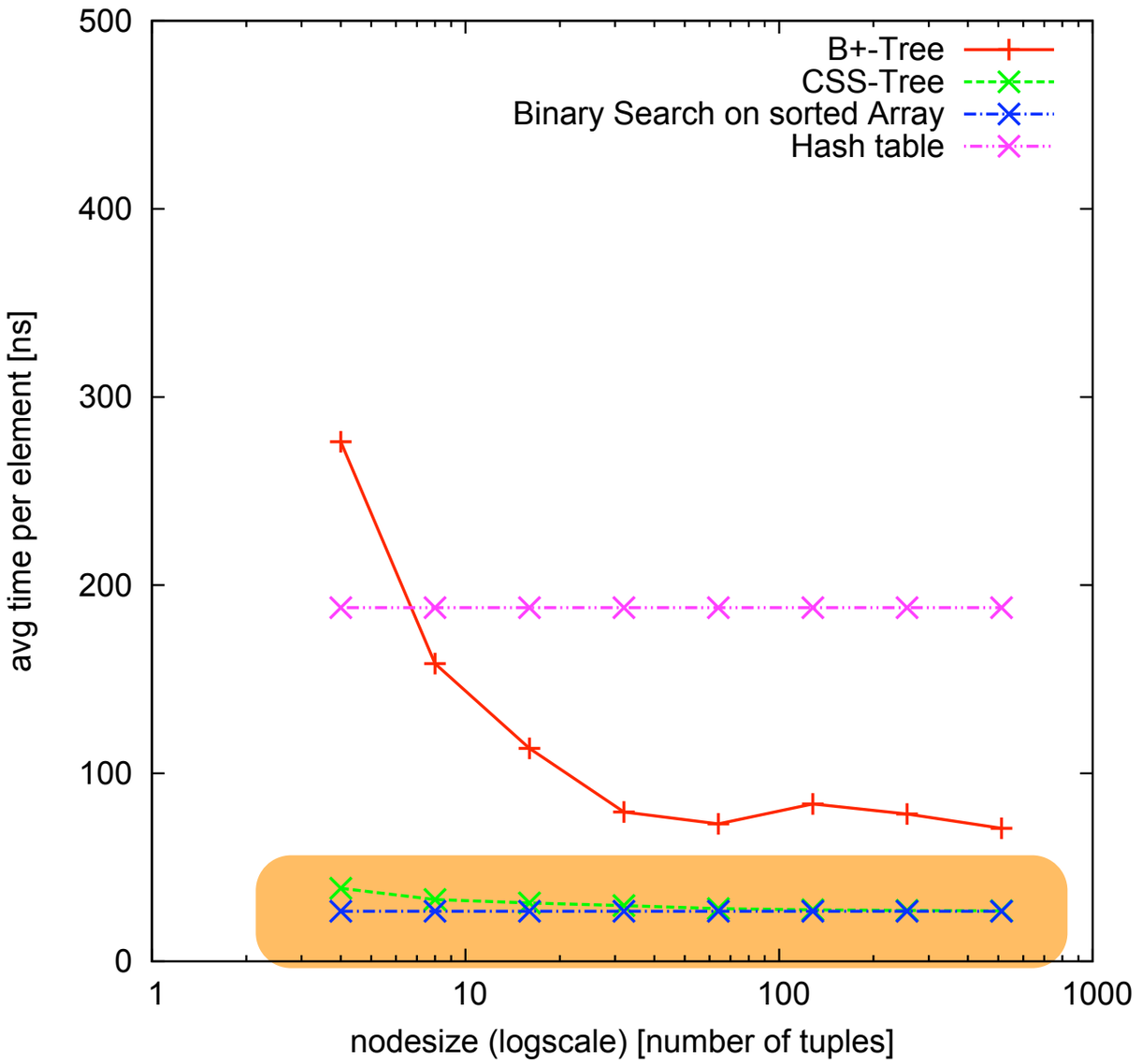
Creation of Full CSS-Trees

- establish mapping between leaf nodes and elements in the sorted array
- start with last node and proceed bottom-up
- for each node entry: fill it with value of the largest key in its immediate left subtree
- finding the largest key can be done by following the link in the rightmost branch until we reach the leaf nodes
- note: this algorithm is similar to linear bottom-up heap creation

Experimental Results



query performance



index creation performance

- index size = 8 million entries
- results from one of my MSc students (S. Hildenbrand, 09/2008)
- very good trade-off for CSS-tree

Next Topic: Hash-Indexes.