HCTrie: A Structure for Indexing Hundreds of Dimensions for Use in File Systems Search

Yasuhiro Ohara
Storage Systems Research Center
University of California Santa Cruz
yasu@soe.ucsc.edu

iTRIS
Center for Information Technology Research in the Interest of Society
ssrc
Storage Systems Research Center
Baskin Engineering
UC Santa Cruz

Summary of HCTrie

- A novel multi-dimensional search structure.
- It supports hundreds of dimensions. (Currently numerical attributes only.)
- It avoids intersection (JOIN) operations.
- It is simple, preserves locality of data, and may be good for range searches.
  - Trie of hypercubes using B+-Tree, or
  - Quadtree with B+-Tree descendant pointer arrays.
- It outperformed MySQL on range search speed on 5 million random dataset.

Assumptions

- Multidimensional metadata search is beneficial for file search.
- Metadata are numeric.
- Range search is required. (data locality)
- There are hundreds of dimensions (attributes).
- Low cardinality on some attributes (and they are frequently used by search).
  - For example, types, such as ethnicity and gender for personnel database.
- Sometimes even range search is issued on those low-cardinality attributes.
  - For example, Mongoloid (Chinese through American Indian).

Problems

- Combination of one-dimensional indexing
  - RDBMS falls in this: search results in B-Tree indices and combine, which leads to the Intersection or JOIN operation.
  - Intersection operations are slow: O(n) for each pair of attributes specified in the query.

- Existing multidimensional indexing structures
  - Are either inefficient in range search,
    - Such as K-D Tree.
  - Or do not scale against the number of dimensions.
    - Such as BSP-Tree, Grid-file, R-Tree, and Quadtree.
    - Basically for descendant pointer array, of which the size is $O(2^d)$.

Approach / Idea

- Multidimensional indexing structure with scalability to the number of dimensions.
- Divide the search space in each dimension in each level: such as in Prefix-tree or Trie for one dimensional, and Quadtree for multidimensional.
- Just use B-Tree for the descendant pointer arrays to maintain the scalability to the number of dimensions.

Example

Data item

Attributes (numerical [0-255])

	d_1	d_2	d_3	d_4	d_5
=	3	128	88	200	249

Example

Data item

Attributes (numerical [0-255])

	d_1	d_2	d_3	d_4	d_5
=	3	128	88	200	249
Bit-0(128)	0	1	0	1	1
Bit-1(64)	0	0	1	1	1
Bit-2(32)	0	0	0	0	1
Bit-3(16)	0	0	1	0	1
Bit-4(8)	0	0	1	1	1
Bit-5(4)	0	0	0	0	0
Bit-6(2)	1	0	0	0	0
Bit-7(1)	1	0	0	0	1

Example

Data item

Attributes (numerical [0-255])

	d_1	d_2	d_3	d_4	d_5
=	3	128	88	200	249
Bit-0(128)	0	1	0	1	1
Bit-1(64)	0	0	1	1	1
Bit-2(32)	0	0	0	0	1
Bit-3(16)	0	0	1	0	1
Bit-4(8)	0	0	1	1	1
Bit-5(4)	0	0	0	0	0
Bit-6(2)	1	0	0	0	0
Bit-7(1)	1	0	0	0	1

Example

Data item

Attributes (numerical [0-255])

	d_1	d_2	d_3	d_4	d_5
=	3	128	88	200	249
Bit-0(128)	0	1	0	1	1
Bit-1(64)	0	0	1	1	1
Bit-2(32)	0	0	0	0	1
Bit-3(16)	0	0	1	0	1
Bit-4(8)	0	0	1	1	1
Bit-5(4)	0	0	0	0	0
Bit-6(2)	1	0	0	0	0
Bit-7(1)	1	0	0	0	1

Example

Data item

Attributes (numerical [0-255])

	d_1	d_2	d_3	d_4	d_5			
=	3	128	88	200	249			
Bit-0(128)	0	1	0	1	1	->	11	L_0
Bit-1(64)	0	0	1	1	1	->	7	L_1
Bit-2(32)	0	0	0	0	1	->	1	L_2
Bit-3(16)	0	0	1	0	1	->	5	L_3
Bit-4(8)	0	0	1	1	1	->	7	L_4
Bit-5(4)	0	0	0	0	0	->	0	L_5
Bit-6(2)	1	0	0	0	0	->	16	L_6
Bit-7(1)	1	0	0	0	1	->	17	L_7

Example

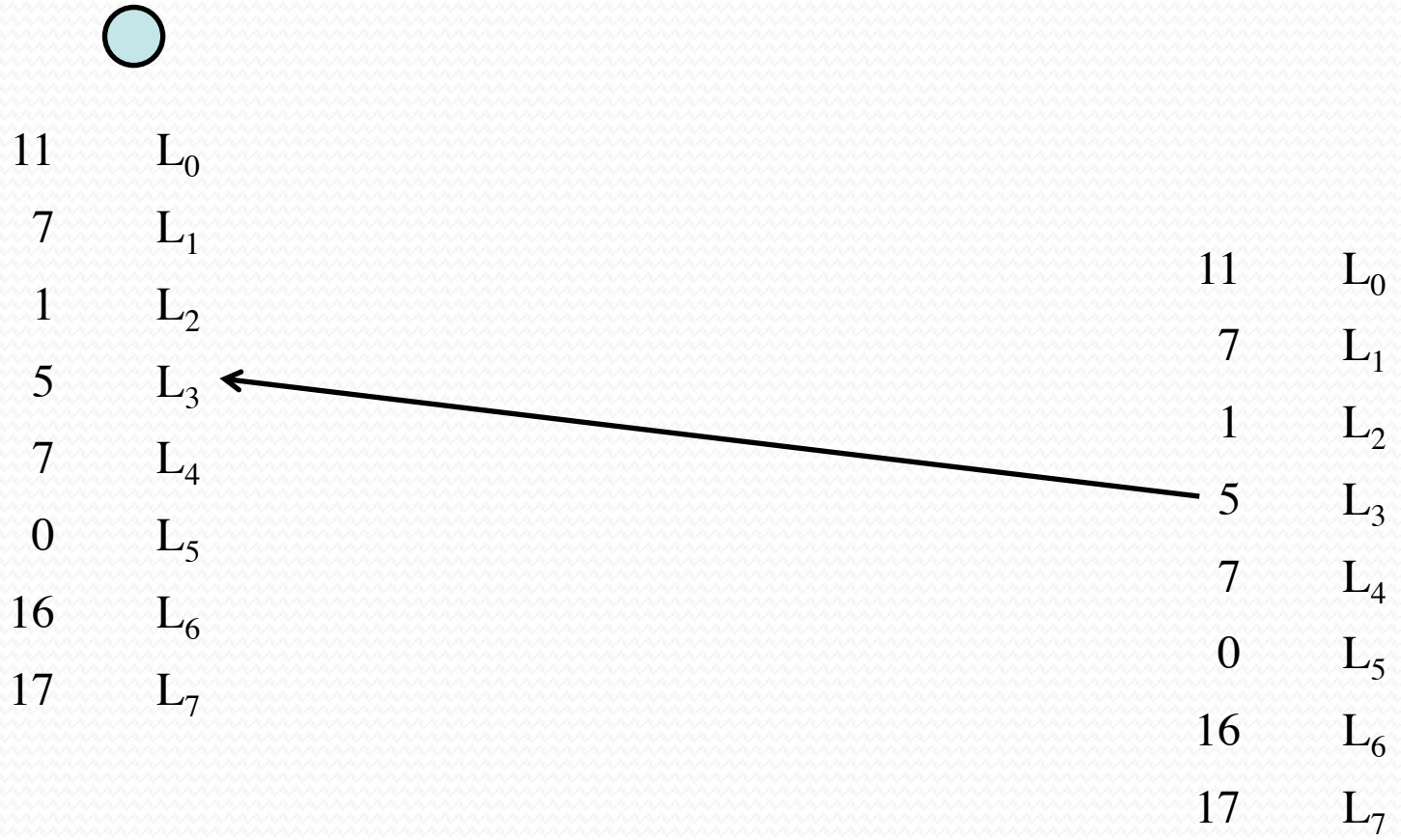
Data item

Attributes (numerical [0-255])

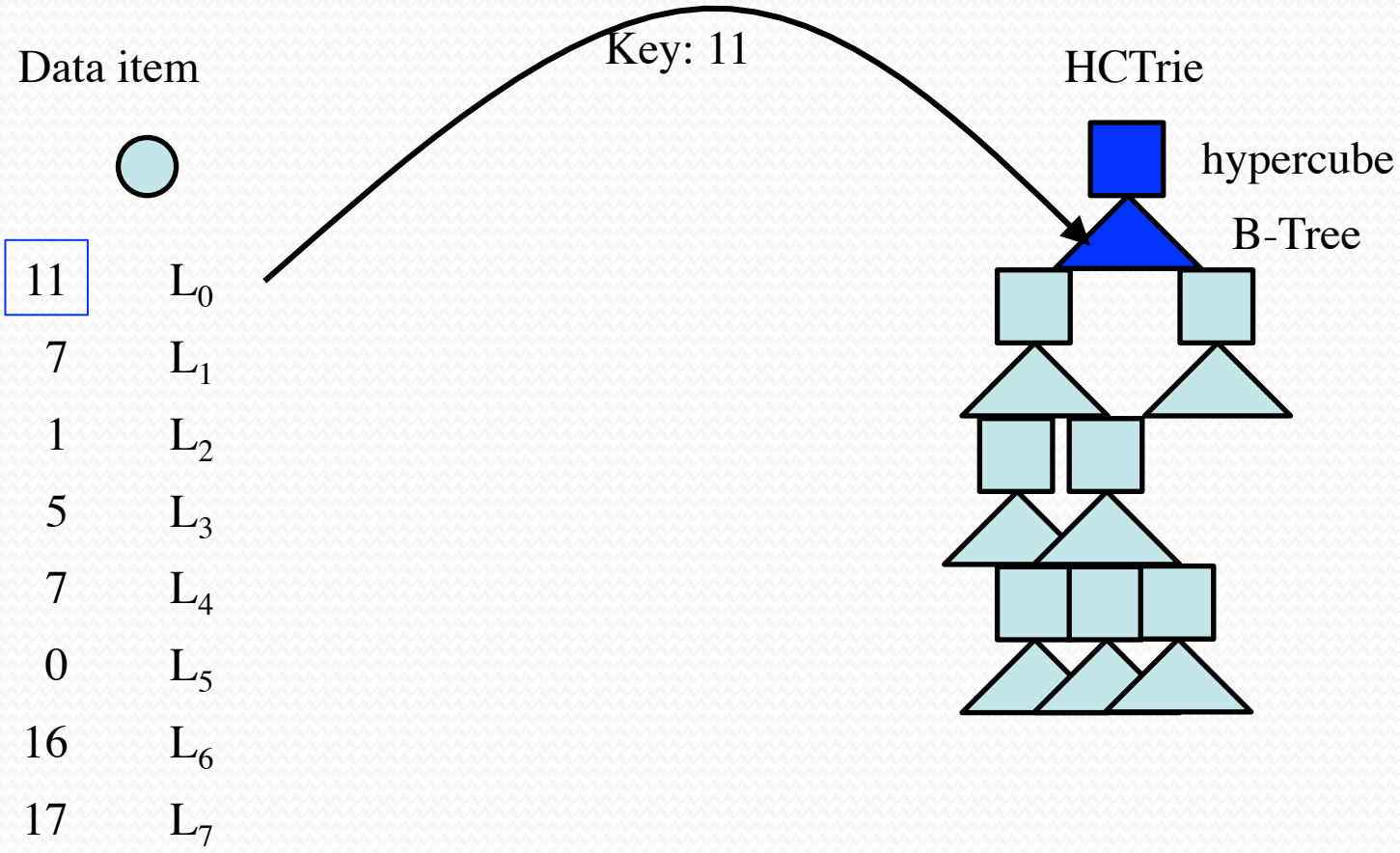
	d_1	d_2	d_3	d_4	d_5			
=	3	128	88	200	249			
Bit-0(128)	0	1	0	1	1	->	11	L_0
Bit-1(64)	0	0	1	1	1	->	7	L_1
Bit-2(32)	0	0	0	0	1	->	1	L_2
Bit-3(16)	0	0	1	0	1	->	5	L_3
Bit-4(8)	0	0	1	1	1	->	7	L_4
Bit-5(4)	0	0	0	0	0	->	0	L_5
Bit-6(2)	1	0	0	0	0	->	16	L_6
Bit-7(1)	1	0	0	0	1	->	17	L_7

Example

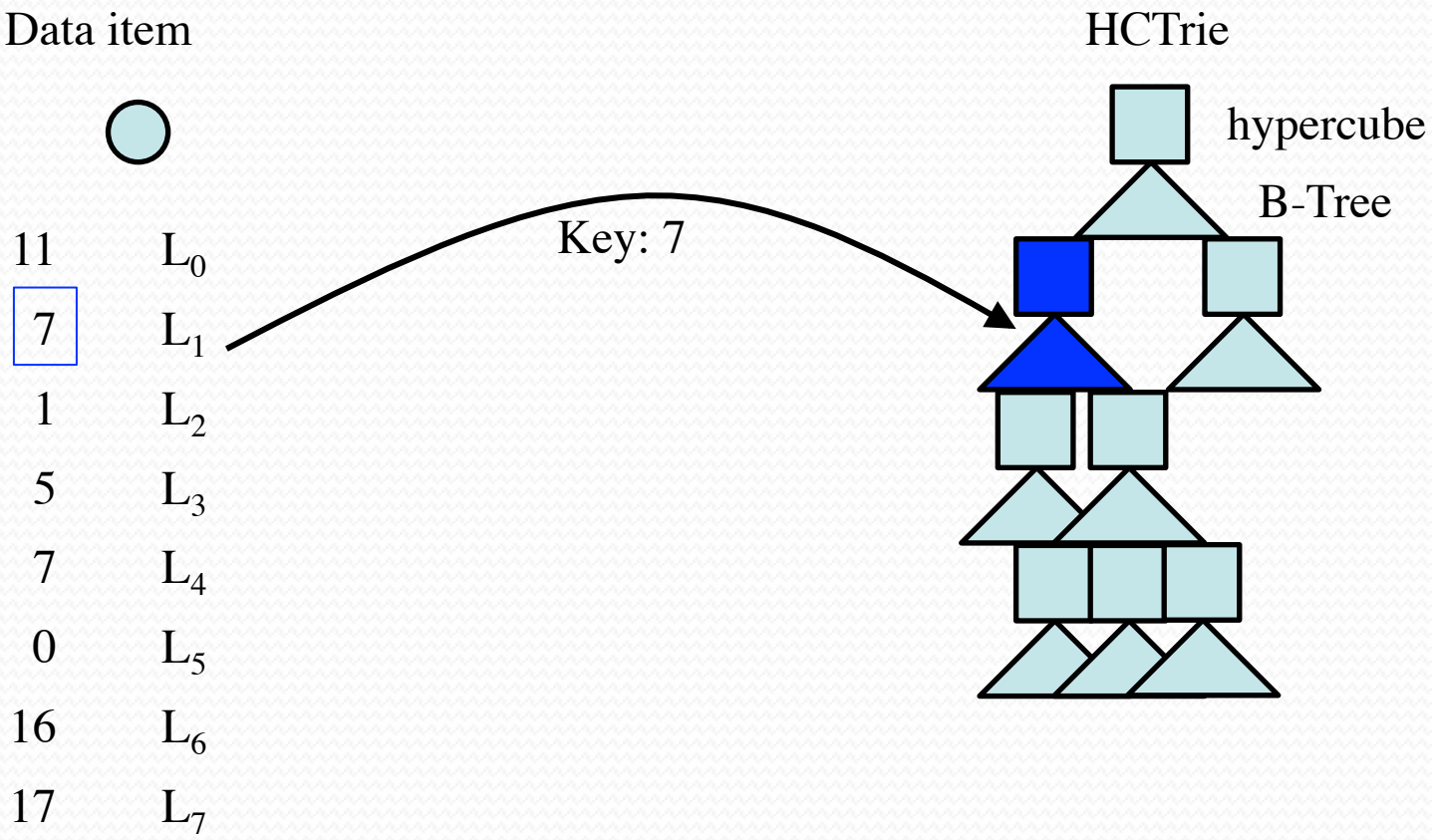
Data item



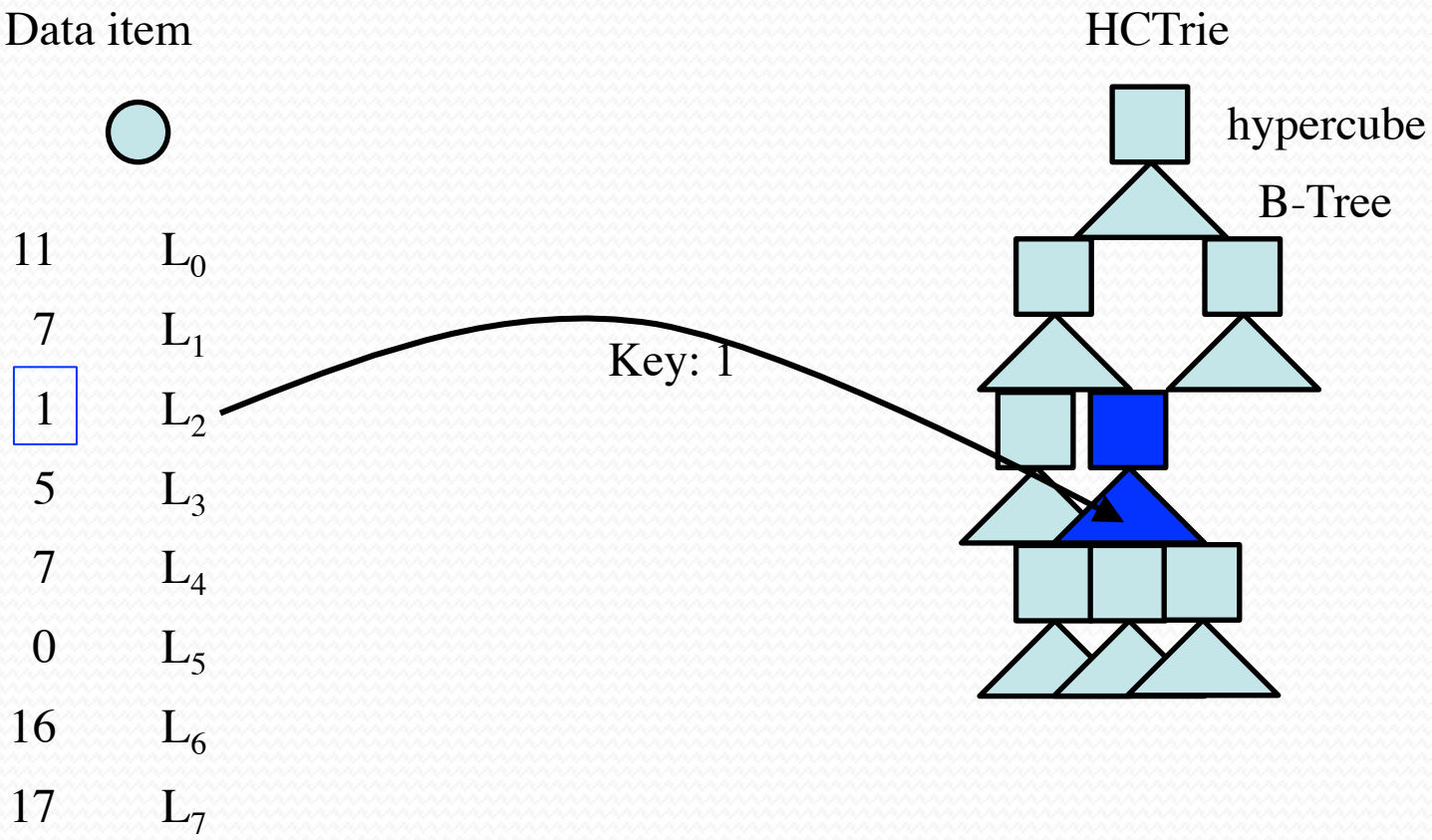
Example


Data item
Key: 11
HCTrie
hypercube
B-Tree
11 L_0
7 L_1
1 L_2
5 L_3
7 L_4
0 L_5
16 L_6
17 L_7

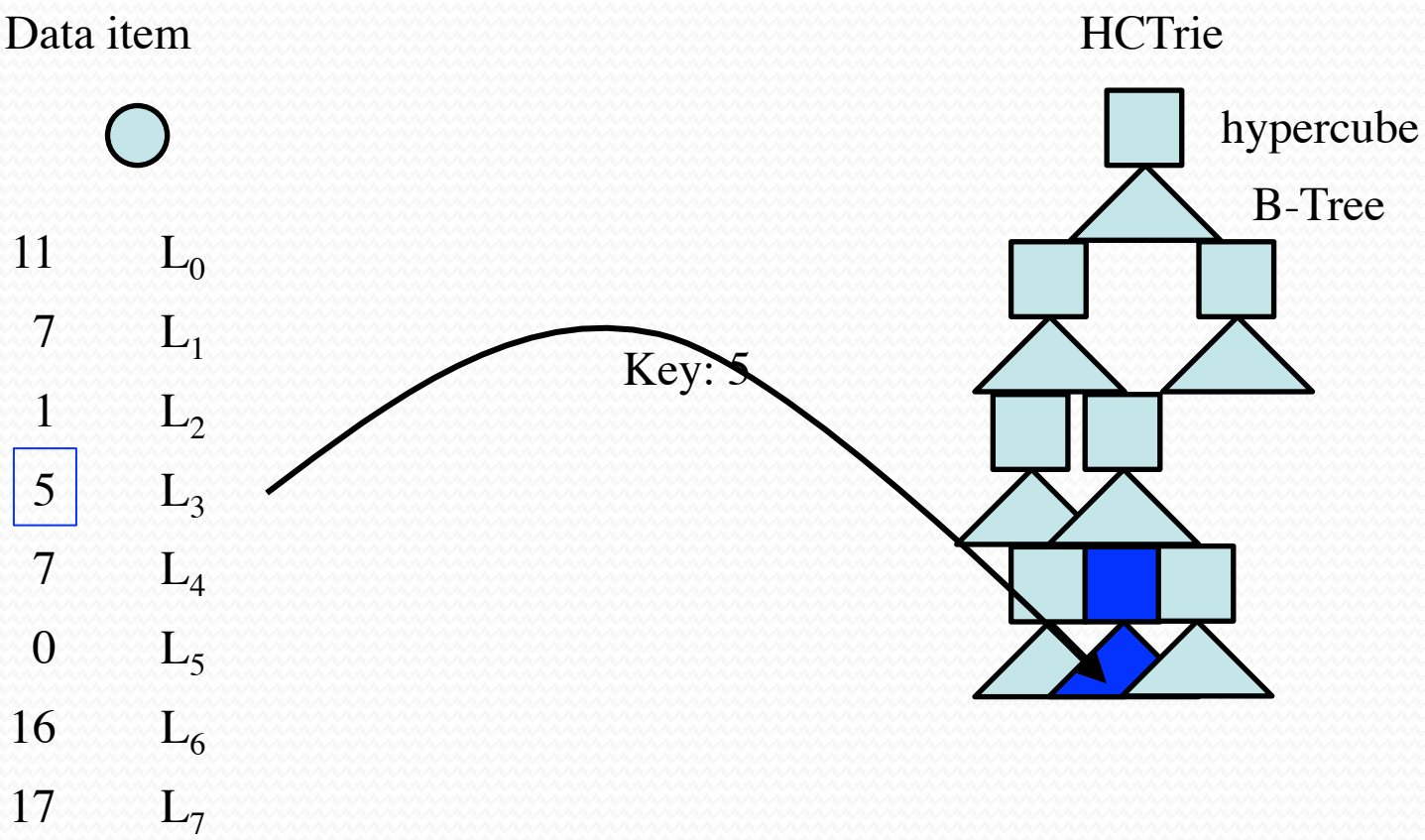
Example


Data item
HCTrie
hypercube
B-Tree
Key: 7
11 L0
7 L1
1 L2
5 L3
7 L4
0 L5
16 L6
17 L7

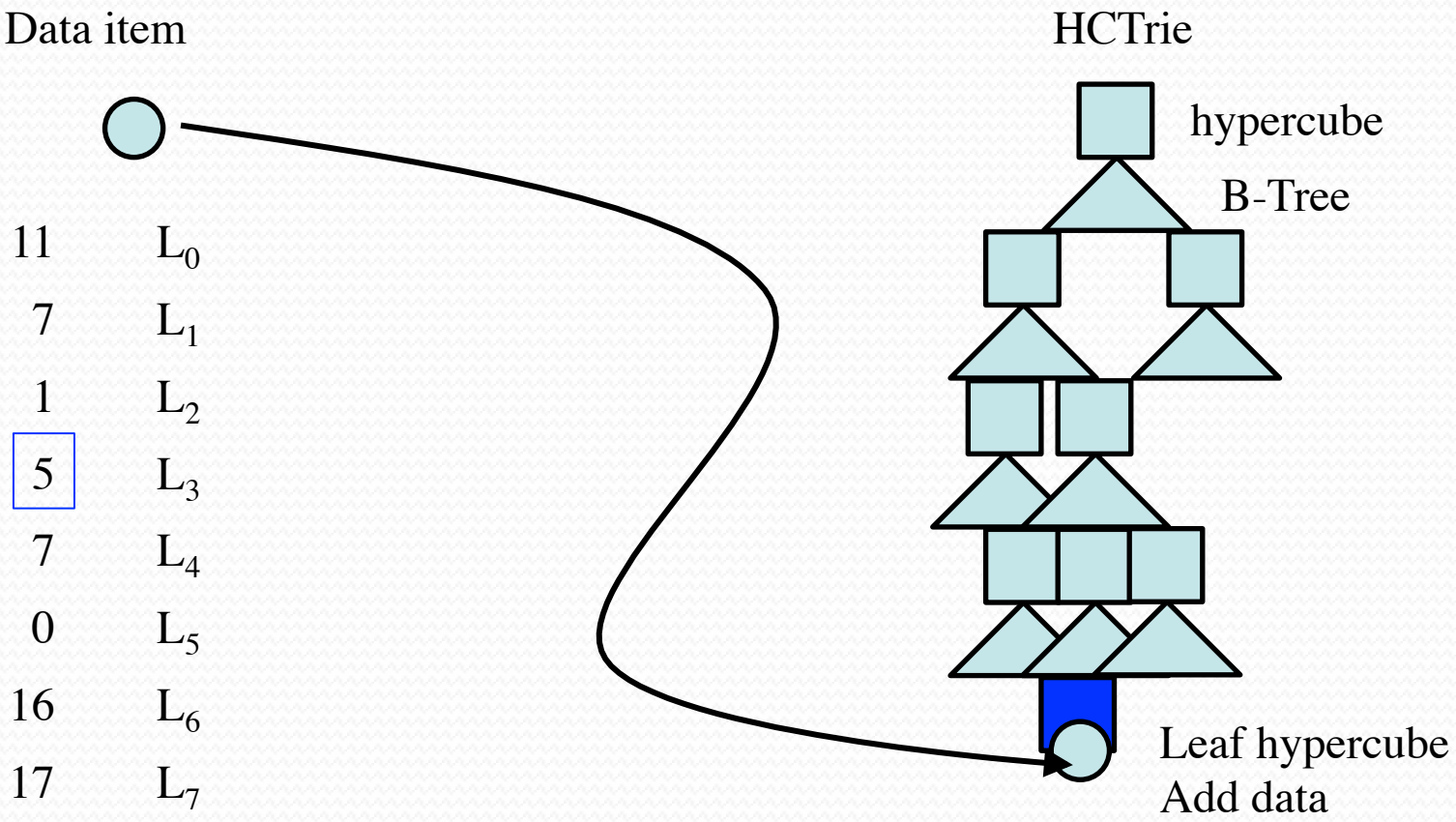
Example


Data item
HCTrie
hypercube
B-Tree
11 L_0
7 L_1
1 L_2
5 L_3
7 L_4
0 L_5
16 L_6
17 L_7
Key: 1

Example

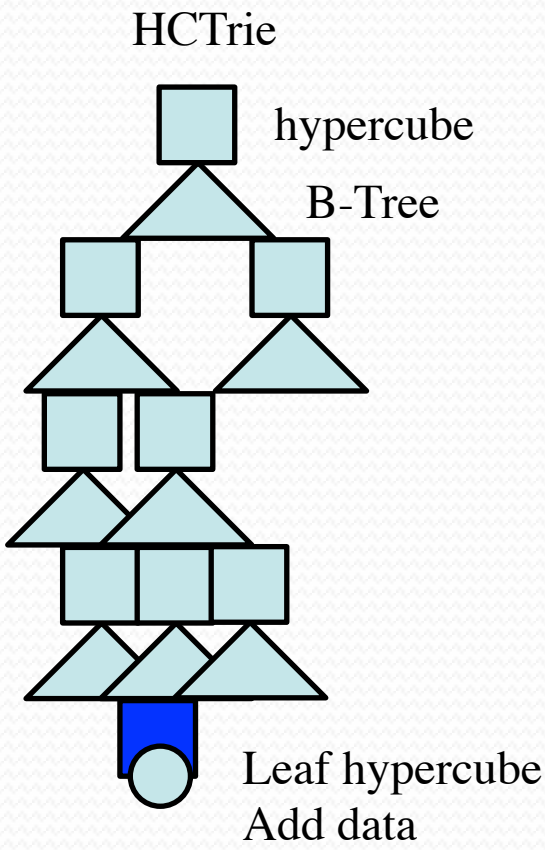

Data item
HCTrie
hypercube
B-Tree
11 L_0
7 L_1
1 L_2
5 L_3
7 L_4
0 L_5
16 L_6
17 L_7
Key: 5

Example


Data item
HCTrie
hypercube
B-Tree
11 L_0
7 L_1
1 L_2
5 L_3
7 L_4
0 L_5
16 L_6
17 L_7
Leaf hypercube
Add data

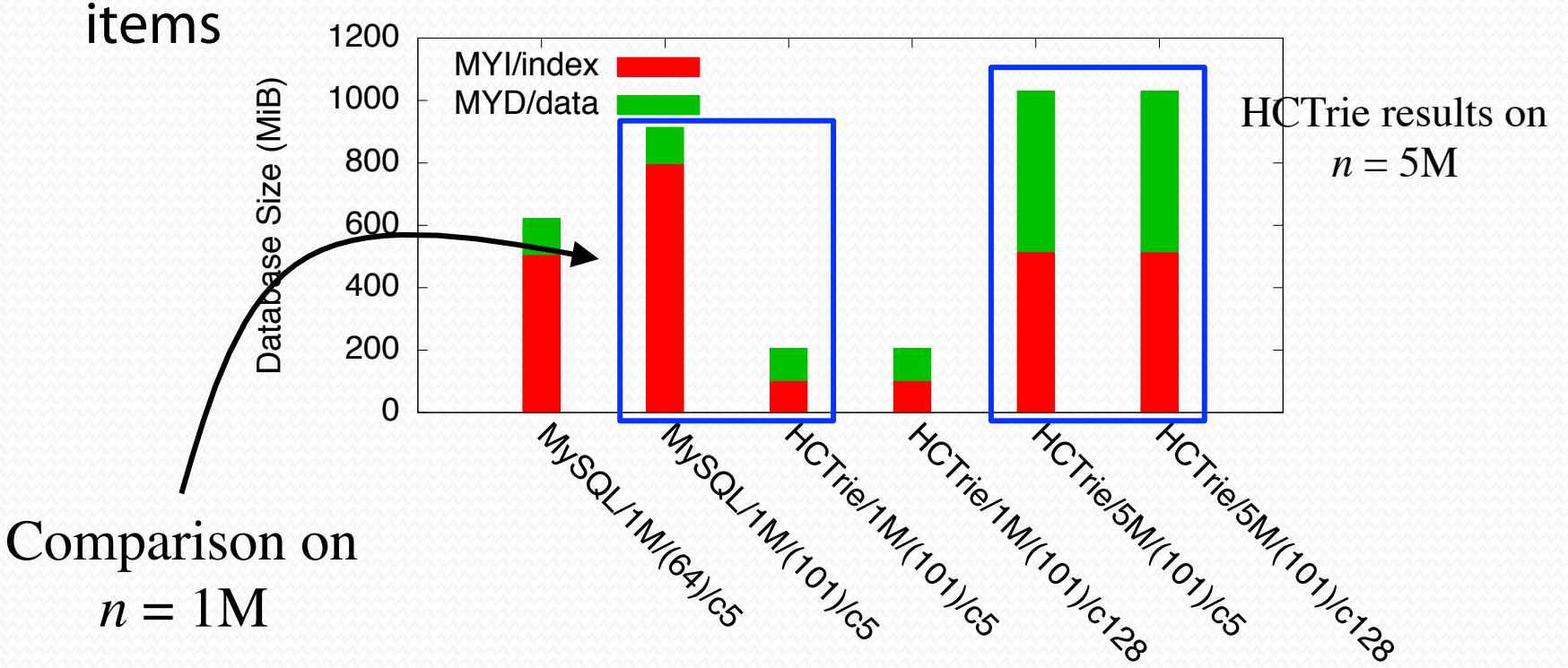
Feasibility

- The width of the descendant pointer array: B-Tree
  - Possibly 2^d (e.g., 2^{100}, is infeasible), but
  - Because nonempty hypercube is ignored, at most *n* hypercube is held.
- The height of the HCTrie
  - Equivalently, the number of levels in HCTrie
  - Possibly *w* (i.e., the bit length of keys, in the worst case)
  - Trie stops when the data is significantly discriminated from the others.
- The range search
  - The data items may be skipped in the unit of the hypercube.



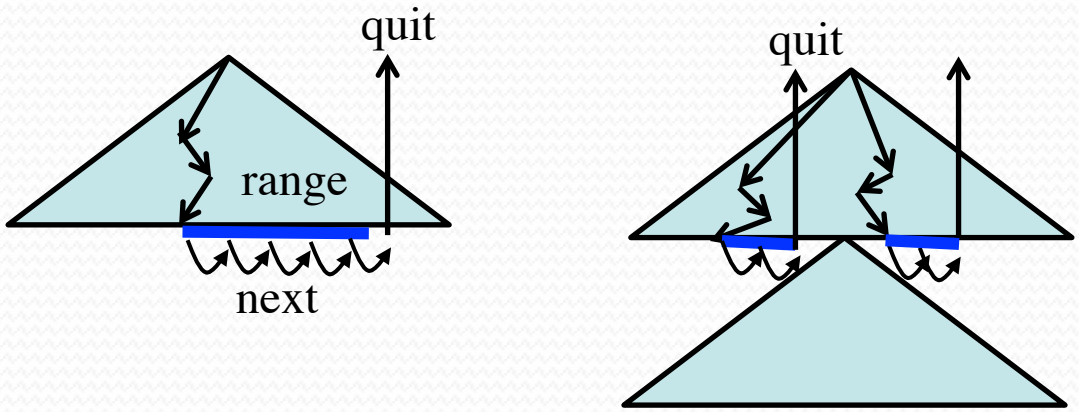
The index size

- The complexity for the index size: *O(Bdn)*
  - The worst case is when a B-Tree page is consumed for each branch, and *Bd* = B-Tree page size and *n* = the number of items



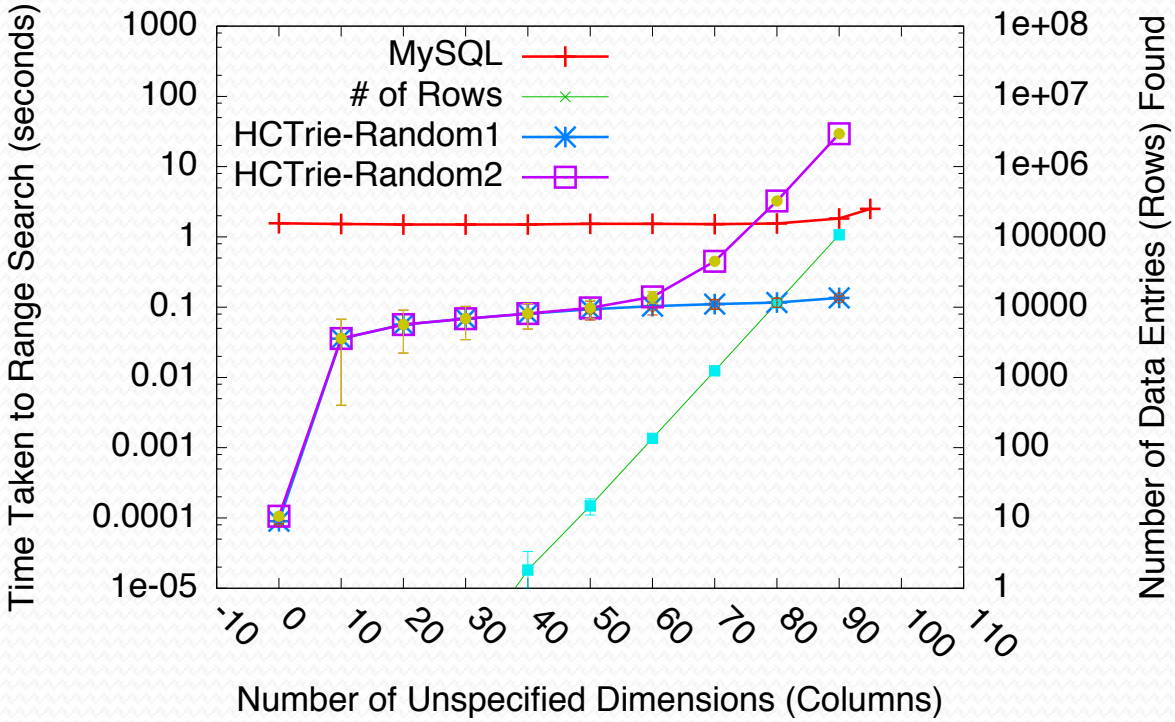
How it runs for range search

- Search down the B-Tree for the key where range starts.
- Go to next (B+-Tree)
- Quit upon reaching beyond the query range.
- Candidates can (hopefully) be skipped by the unit of hypercubes at any levels.



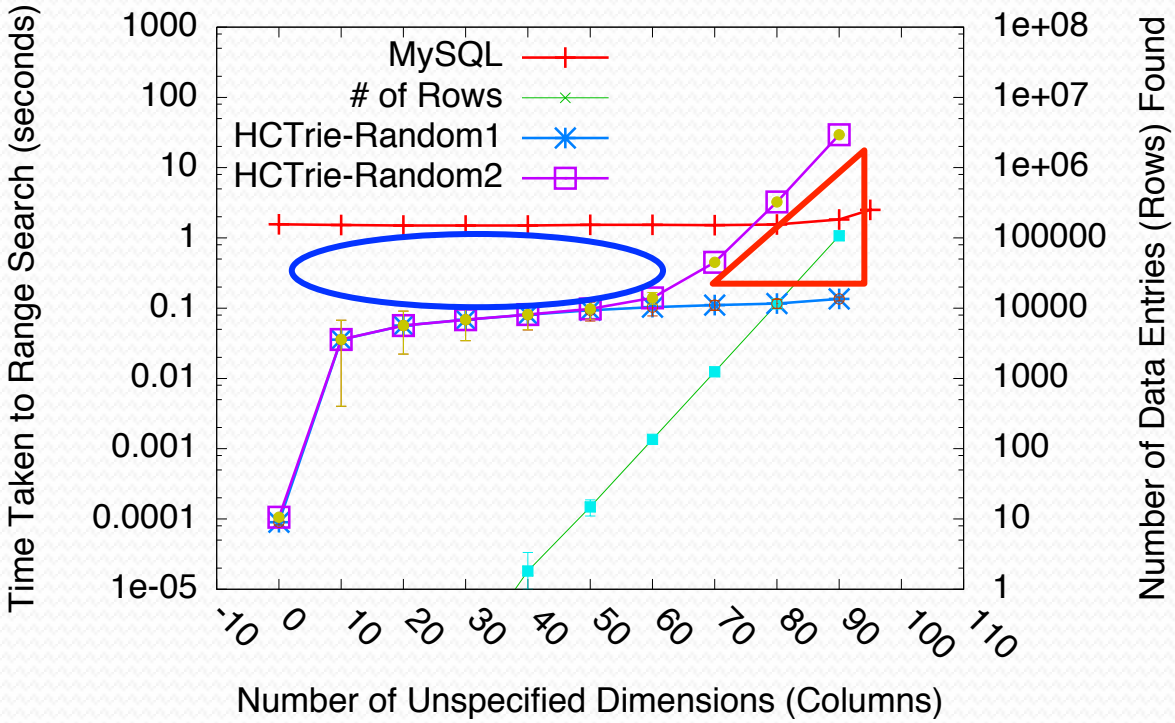
The range search speed

- Random synthetic data with 100 dimensions. n = 1M.
- Low cardinality attributes: [0-4] (cardinality = 5).
- For each attribute, a range may be specified in the range search query. [0-3] if specified, or unspecified ([0-4]).
- For HCTrie, unpepcified dimensions are randomly selected. For MySQL, the first x dimensions are unspecified.



The range search speed

- Random synthetic data with 100 dimensions. n = 1M.
- Low cardinality attributes: [0-4] (cardinality = 5).
- For each attribute, a range may be specified in the range search query. [0-3] if specified, or unspecified ([0-4]).
- For HCTrie, unpepcified dimensions are randomly selected. For MySQL, the first x dimensions are unspecified.



Conclusion

- A novel multidimensional indexing structure is proposed. It works on hundreds of dimensions and avoid intersection operations to speed up the range search.

THANKS

Supported by

U.S. DEPARTMENT OF ENERGY | Office of Science