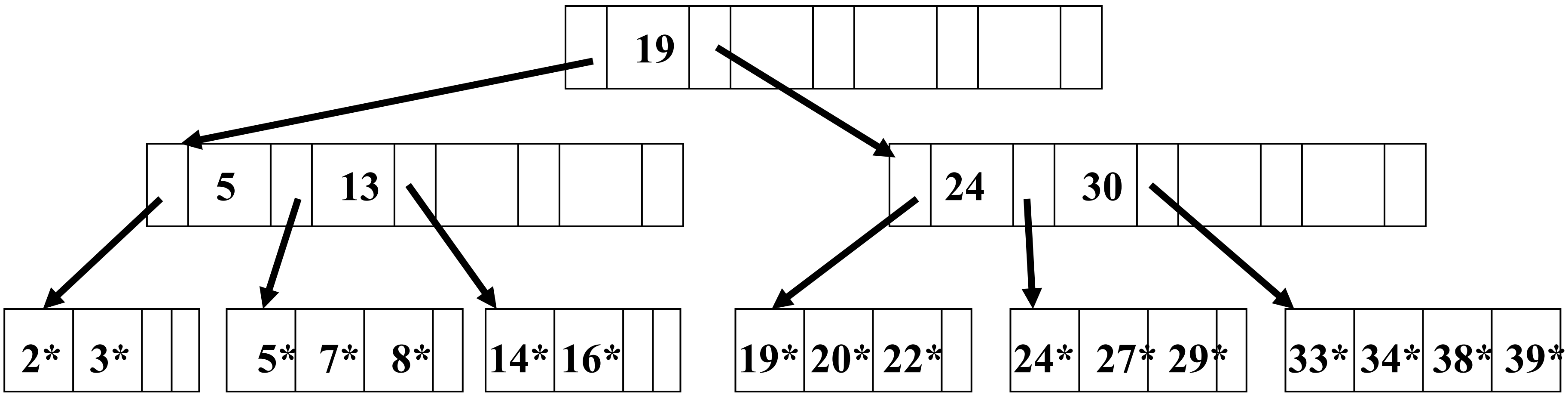
B+ trees

Basics 3

B+ trees as indexes into database tables

Douglas H. Fisher



1) At depth 2, a B+ tree of order 2 has a MAXIMUM of 5*5*4 = 100 items across all leaves (and a minimum of 2*3*2 = 12 items at leaves);

2) At depth M, a B+ tree of order 2 has a MAXIMUM of $5^M * 4$ items across all leaves

. . .

3) At depth M, a B+ tree of order 50 has a MAXIMUM of $(2*50+1)^M * (2*50) = 101^M*100 > 100^{M+1}$ items at each leaf.

Two questions

What is the MAXIMUM number of items that a B+ tree of order 50 and depth 2 can have at its leaves?

What is the MINIMUM number of links that would have to be followed to find an item at the leaf of a B+ tree of order 50 and that contained 1,006,201 items across all leaves?

What is the MAXIMUM number of items that a B+ tree of order 50 and depth 2 can have at its leaves? 1,020,100

The maximum number of child links of each internal node is 2*50+1 or 101. The maximum number of items at each leaf is 100.

- **101 child links (max) at the root (depth 0)**
- **101^2 (= 10,201) child links (max) across all depth 1 nodes**
- **101^2 * 100 (= 1,020,100) items (max) across all depth 2 leaves**

What is the MINIMUM number of pointers that would have to be followed to find an item at the leaf of a B+ tree of order 50 and that contained 1,006,201 items across all leaves? 2 pointers at minimum

A depth 1 B+ tree of order 50 could hold at most 101*100 = 10,100 (not enough), but a depth 2 tree has the capacity (see above)

B+ trees and binary search trees

1) The depth of a B+ tree of order 50 that holds N items at leaves is $O(\log_{100} N)$. The depth also corresponds to the number of pointers that would have to be followed to find an item.

2) In contrast, the depth of a good ol' suitably balanced binary search tree is $O(\log_2 N)$. Again, depth corresponds to the number of pointers that would have to be followed.

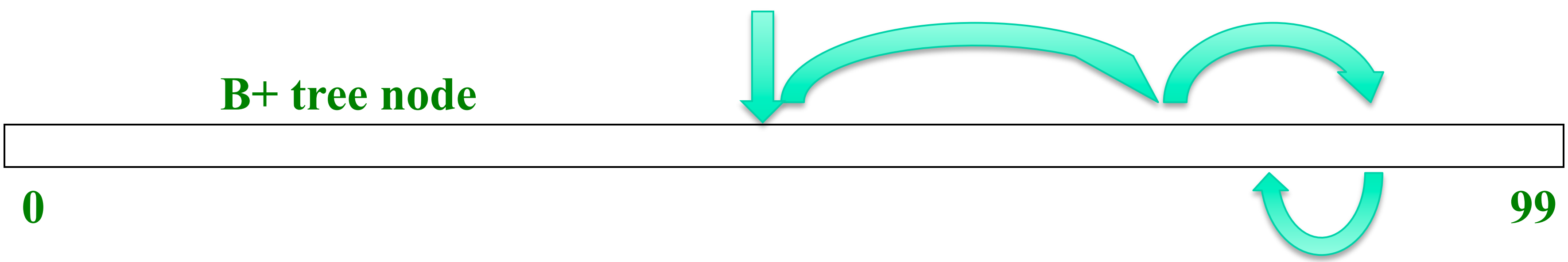
3) If N is 1,000,000, then 2-3 pointers must be followed in B+ tree of order 50 (with no less than 50 and up to 100 keys in each node)

4) For a balanced binary search tree, there would be about 20 pointers followed in N is 1,000,000

5) Looks like a big win for the B+ tree, but not so fast!

… but not so fast!

For EACH node of the B+ tree, we must find where the search is to be directed. We could use sequential search for B+ trees of small orders, but for anything but the smallest orders, binary search would be better (and the average cost of binary search is almost the worst case of about $\log_2 n$)



So the # "steps" using a B+ tree of order 50 (100 entries)

$= $ number of nodes times cost per node

$\cong \log_{100} n * \log_2 100$ (generalizes to any order)

$= \log_2 n$

$\cong$ # "steps" using a balanced binary search tree

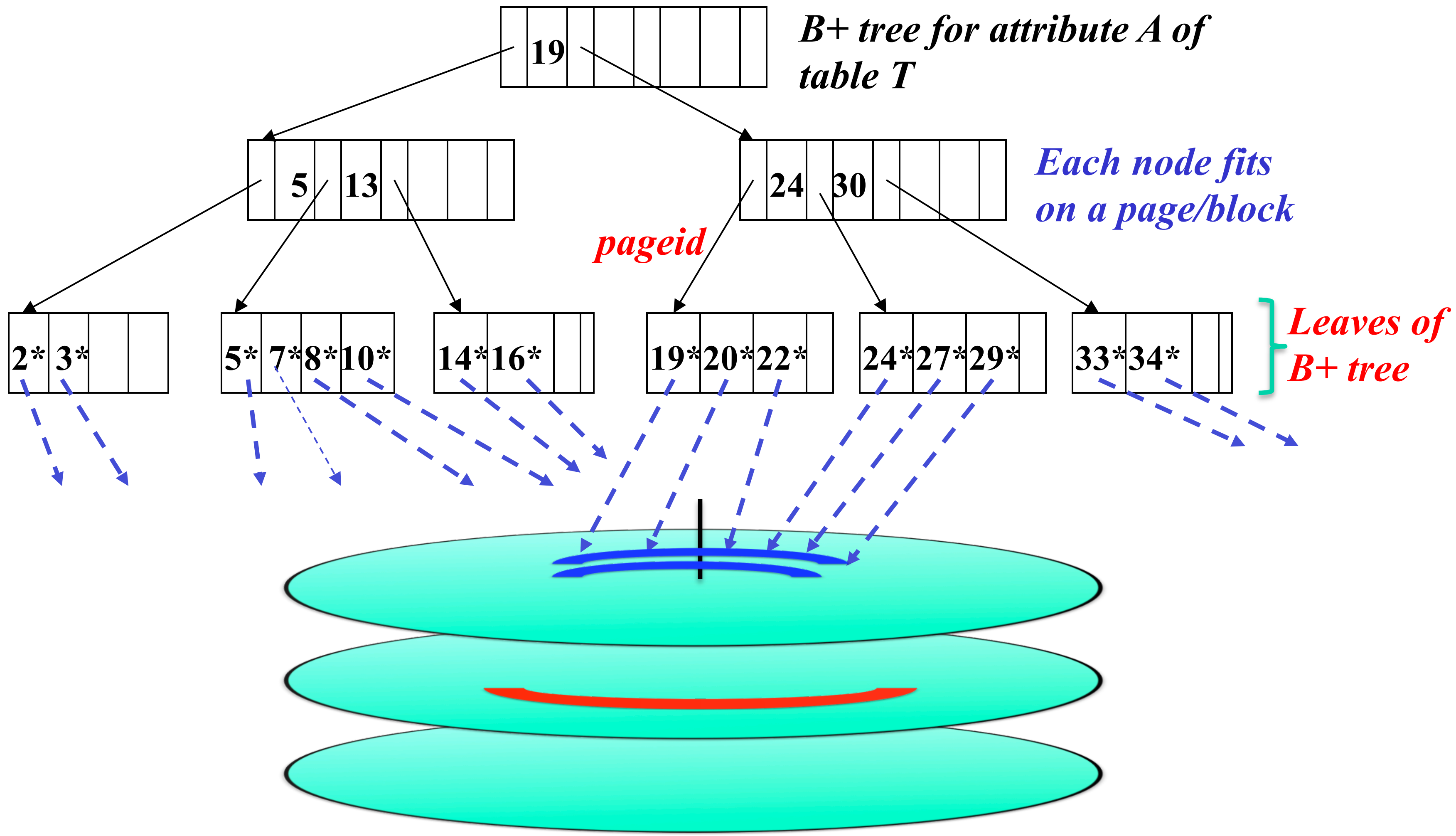
1) The asymptotic cost of using a B+ tree and a binary search tree are the same,

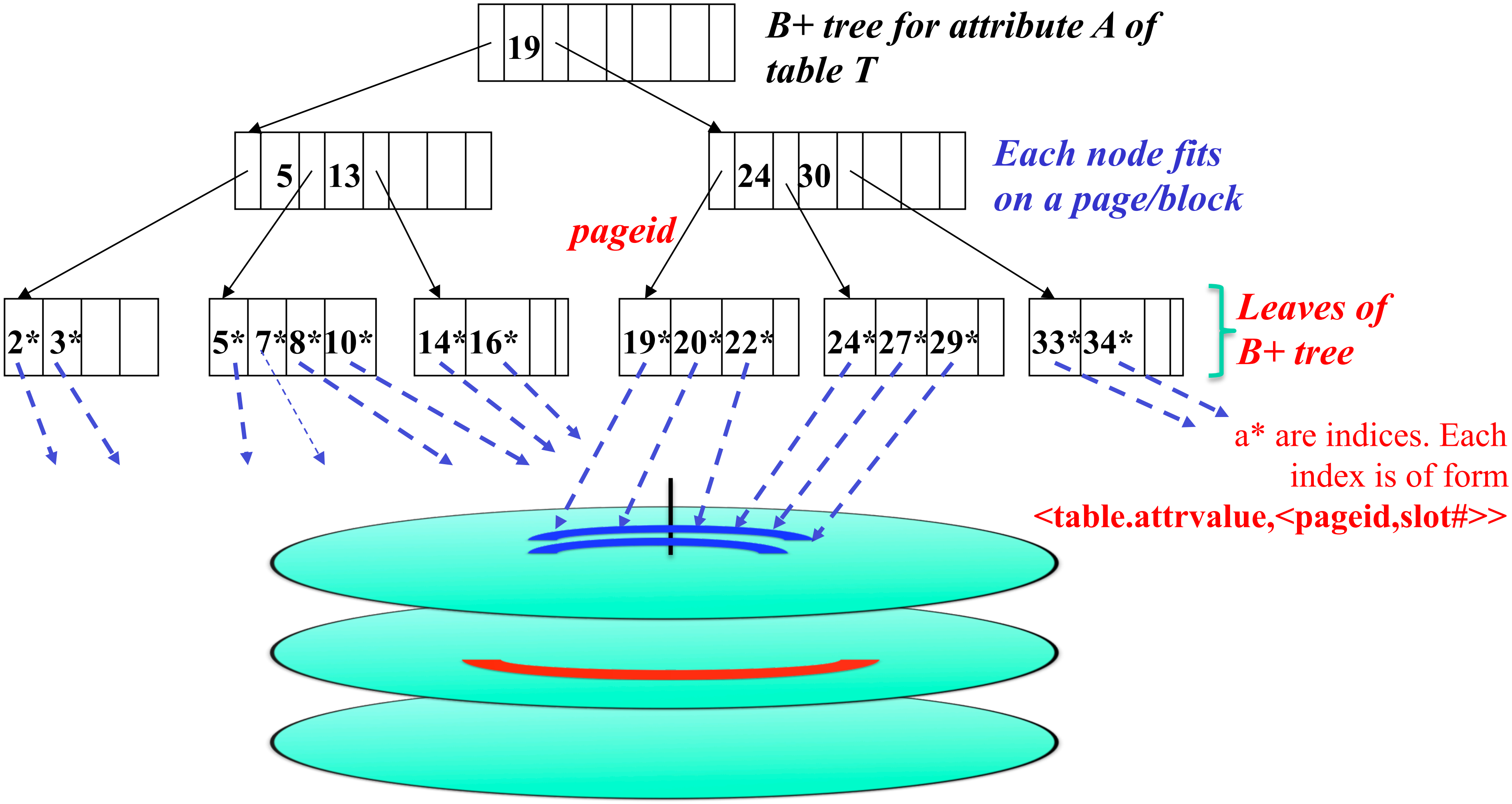
2) but in accessing database tables, following pointers is VERY expensive,

3) because database tables (and the B+ trees that index them) are stored on disk,

4) where each access takes time proportional to disk rotational delay * seek time * read(write) time

5) so we want to minimize the # of pointers followed


19
B+ tree for attribute A of table T
5
13
24
30
Each node fits on a page/block
pageid
2*
3*
5*
7*
8*
10*
14*
16*
19*
20*
22*
24*
27*
29*
33*
34*
Leaves of B+ tree



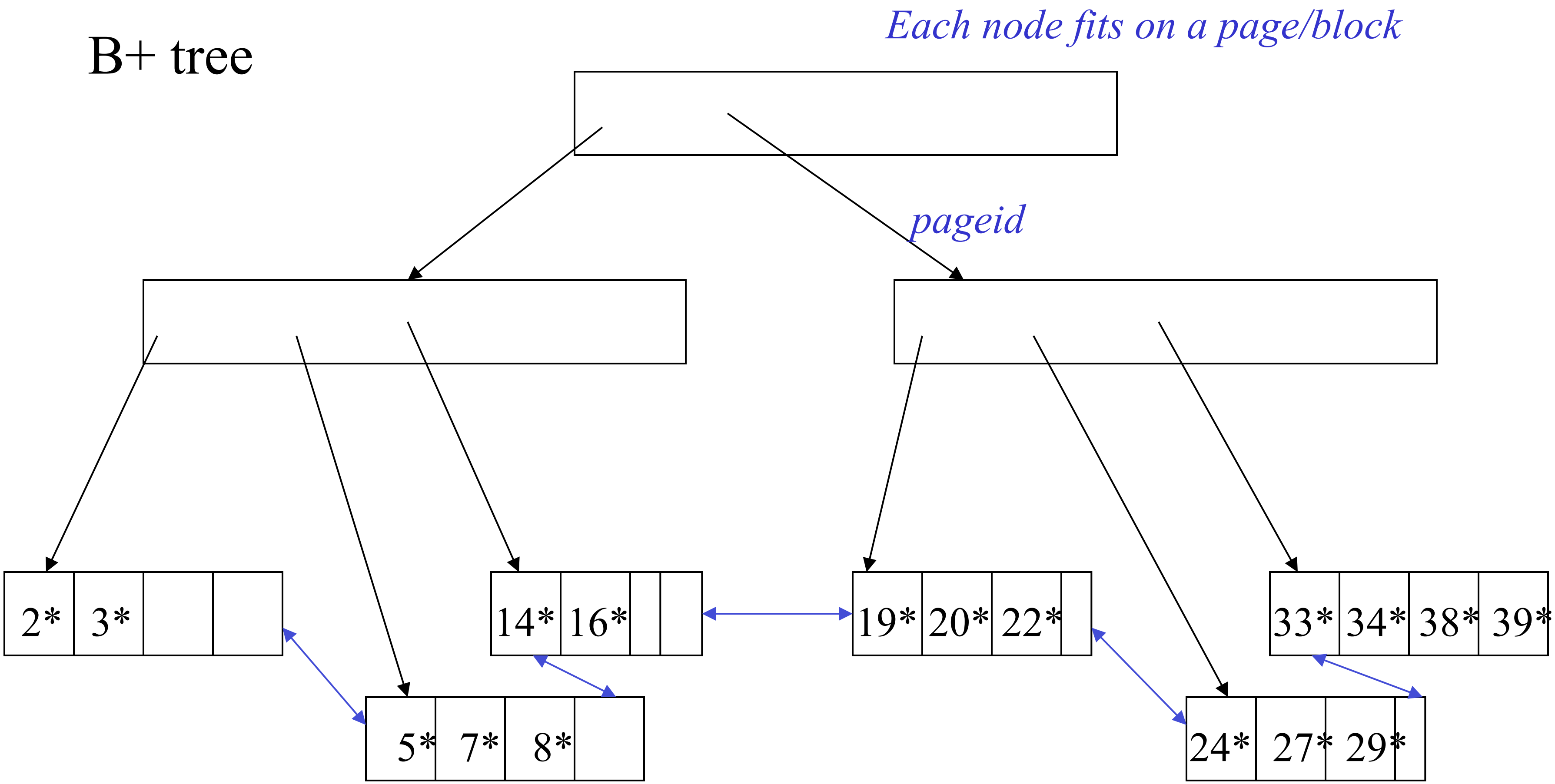
As an example, suppose that a block (or page) holds 2^{12} bytes

Suppose each tuple of table T requires 2^4 bytes

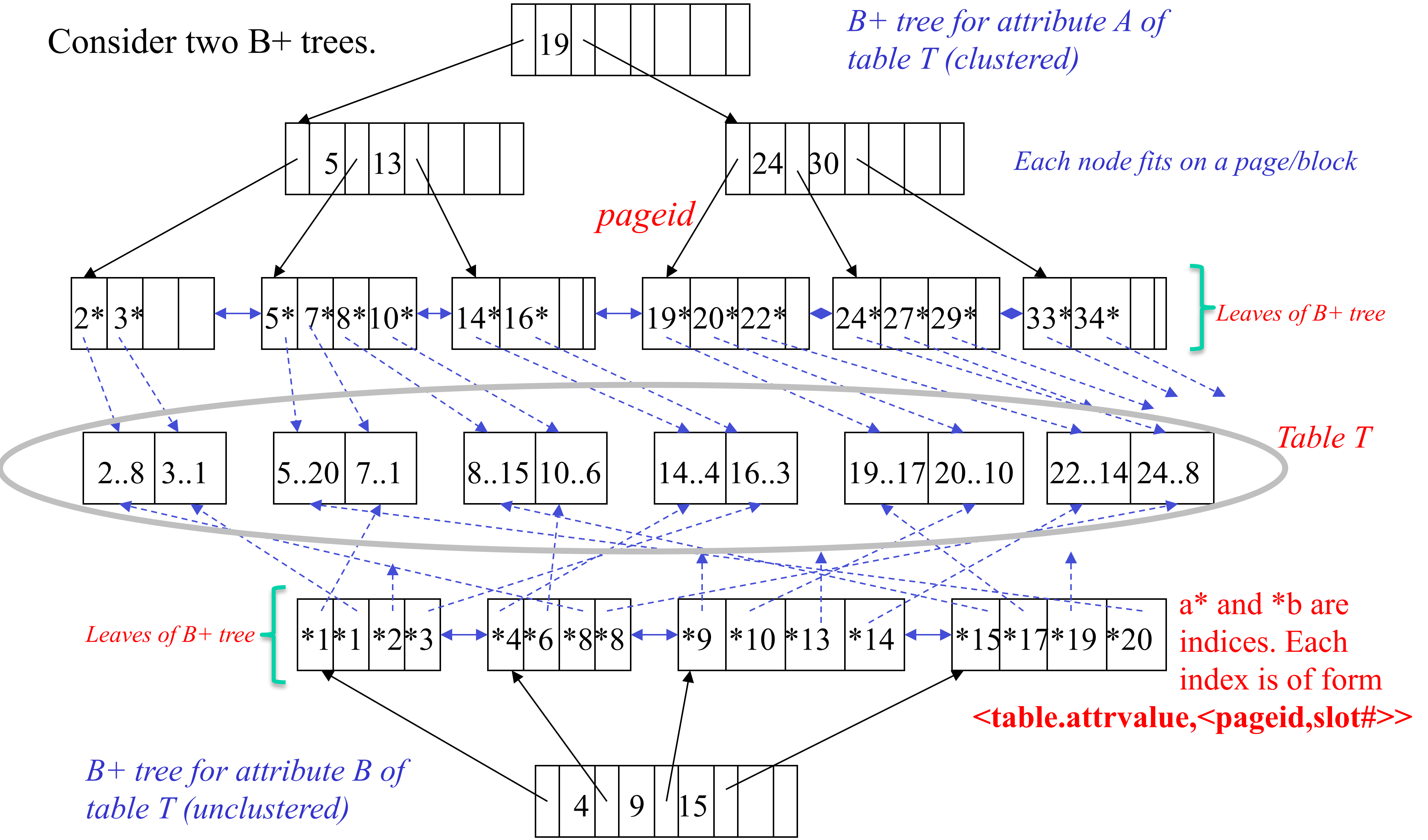
Suppose each index for table T requires 2^3 bytes

B+ tree order 2^8 would hold up to 2^9 indexes

B+ tree


Each node fits on a page/block
pageid
2* 3*
5* 7* 8*
14* 16*
19* 20* 22*
24* 27* 29*
33* 34* 38* 39*

Consider two B+ trees.



a* and *b are indices. Each index is of form **<table.attrvalue,<pageid,slot#>>**

SELECT T.C FROM T WHERE T.A > 14 AND T.B <= 10

Exploiting T.A clustered B+ tree index will result in fewer pages being read from disk.