BOSTON UNIVERSITY

CS 561: Data Systems Architectures

Introduction to Indexing:

Trees, Tries, Hashing, Bitmap Indexes, Database Cracking

Zichen Zhu

https://bu-disc.github.io/CS561/

Recap: Key-Value Stores

<key, value>

put(key, value)
stores value and associates with key

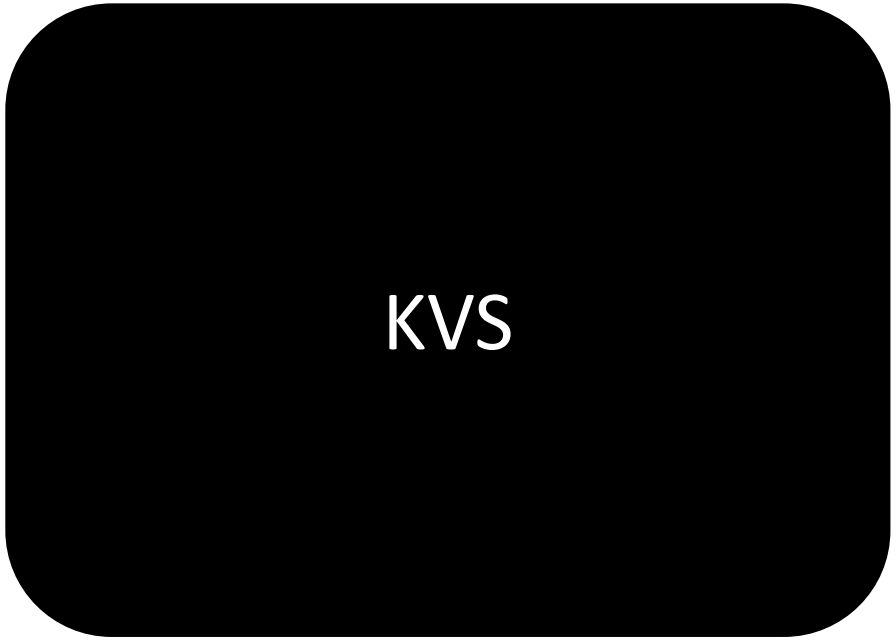
get(key)
returns the associated value

delete(key)
deletes the value associated with the key

get_range (key_start,key_end)
get_set(key1, key2, …)

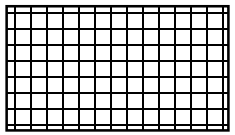
how to organize keys/values?

depends on the workload!

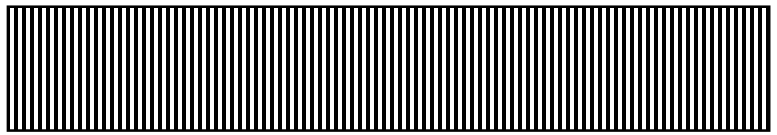


Recap: Key-Value Stores

inserts and point queries?



hash table



log

inserts, point queries, and range queries?

log-structured merge tree

key-value stores vs. indexes

What is an index?

Auxiliary structure to quickly find rows based on arbitrary attribute

Special form of <key, value>

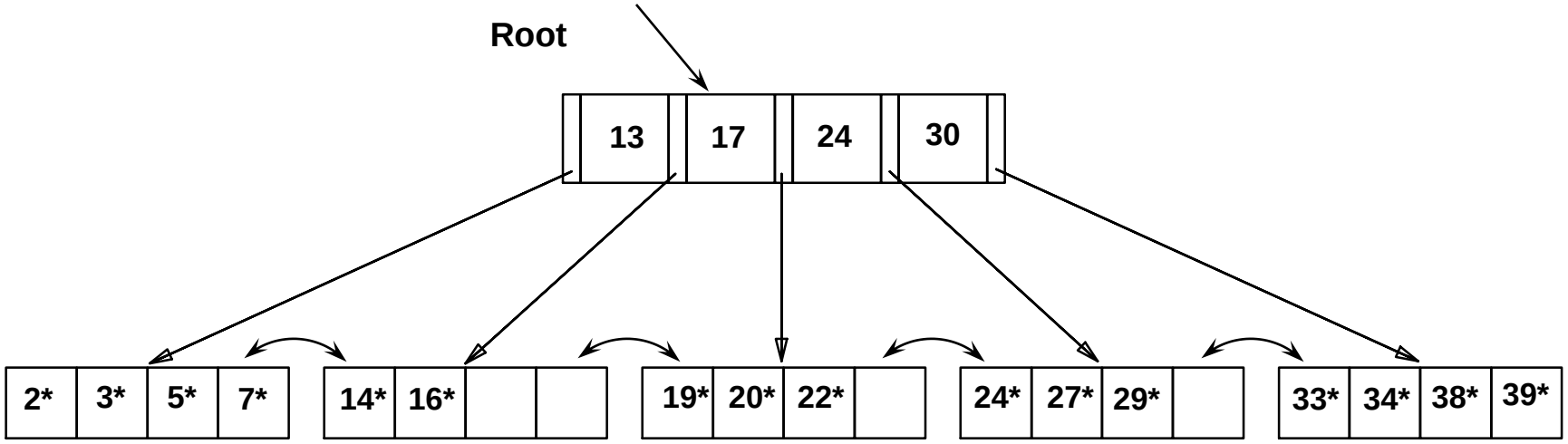
What are the possible ***index designs***?

	Data Organization	Point Queries	Short Range Queries	Long Range Queries	Comments
B+ Trees	Range	✓	✓	✓	Partition ***k-ways*** recursively
LSM Trees	Insertion & Sorted	✓	✗	✓	Optimizes ***insertion***
Radix Trees	Radix	✓	✓	✓	Partition using the ***key radix*** representation
Hash Indexes	Hash	✓	–	✗	Partition by ***hashing the key***
Bitmap Indexes	None	✓	–	✗	Succinctly represent ***all rows with a key***
Scan Accelerators	None	✗	–	✓	Metadata to ***skip accesses***

B+ Trees

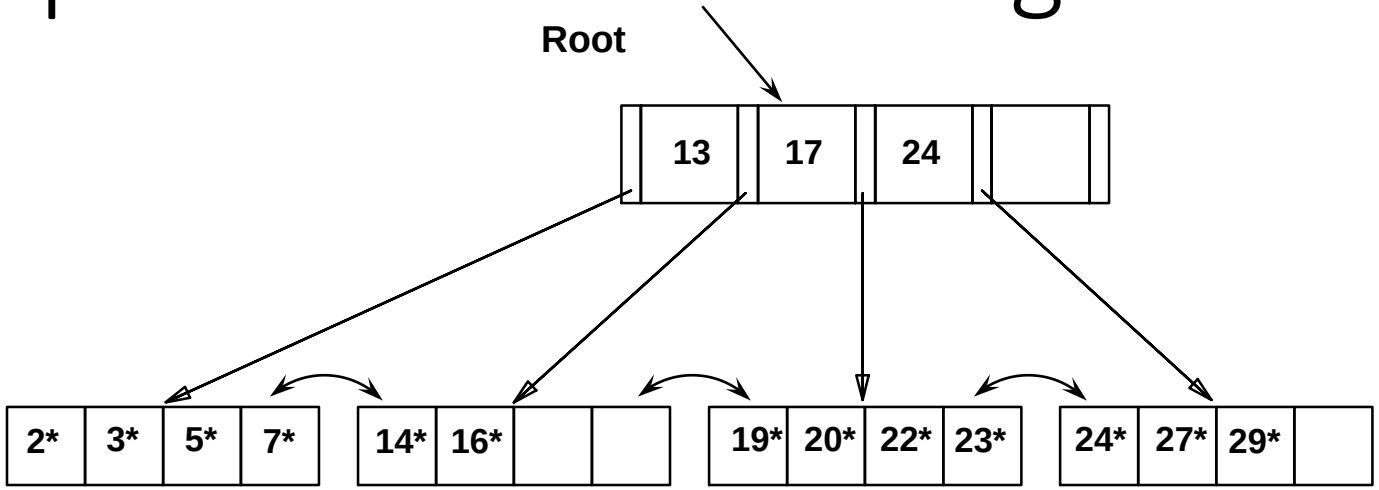
Search begins at root, and key comparisons direct it to a leaf.

Search for 5*, 15*, all data entries >= 24* ...

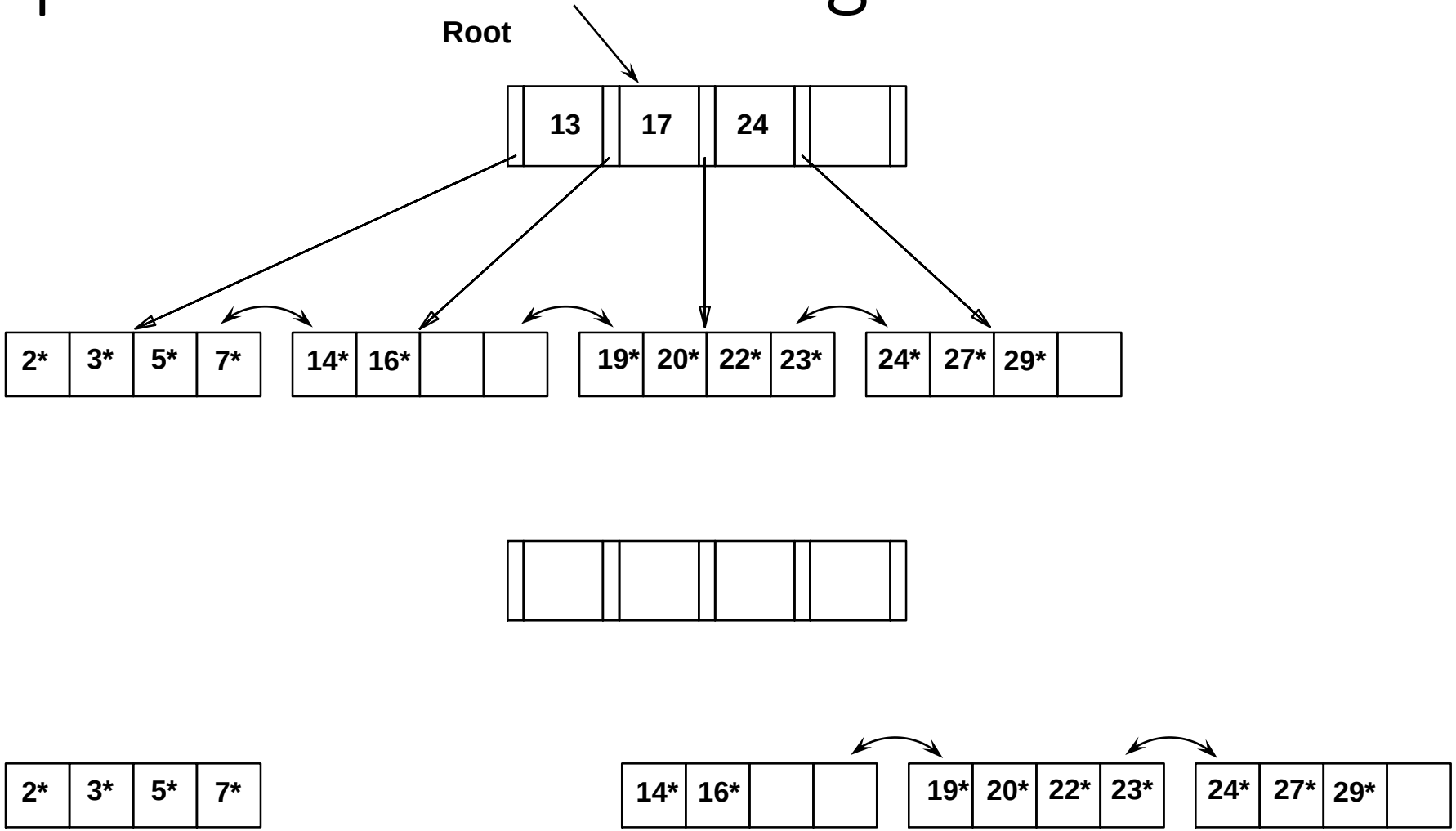


Based on the search for 15, we know it is not in the tree!*

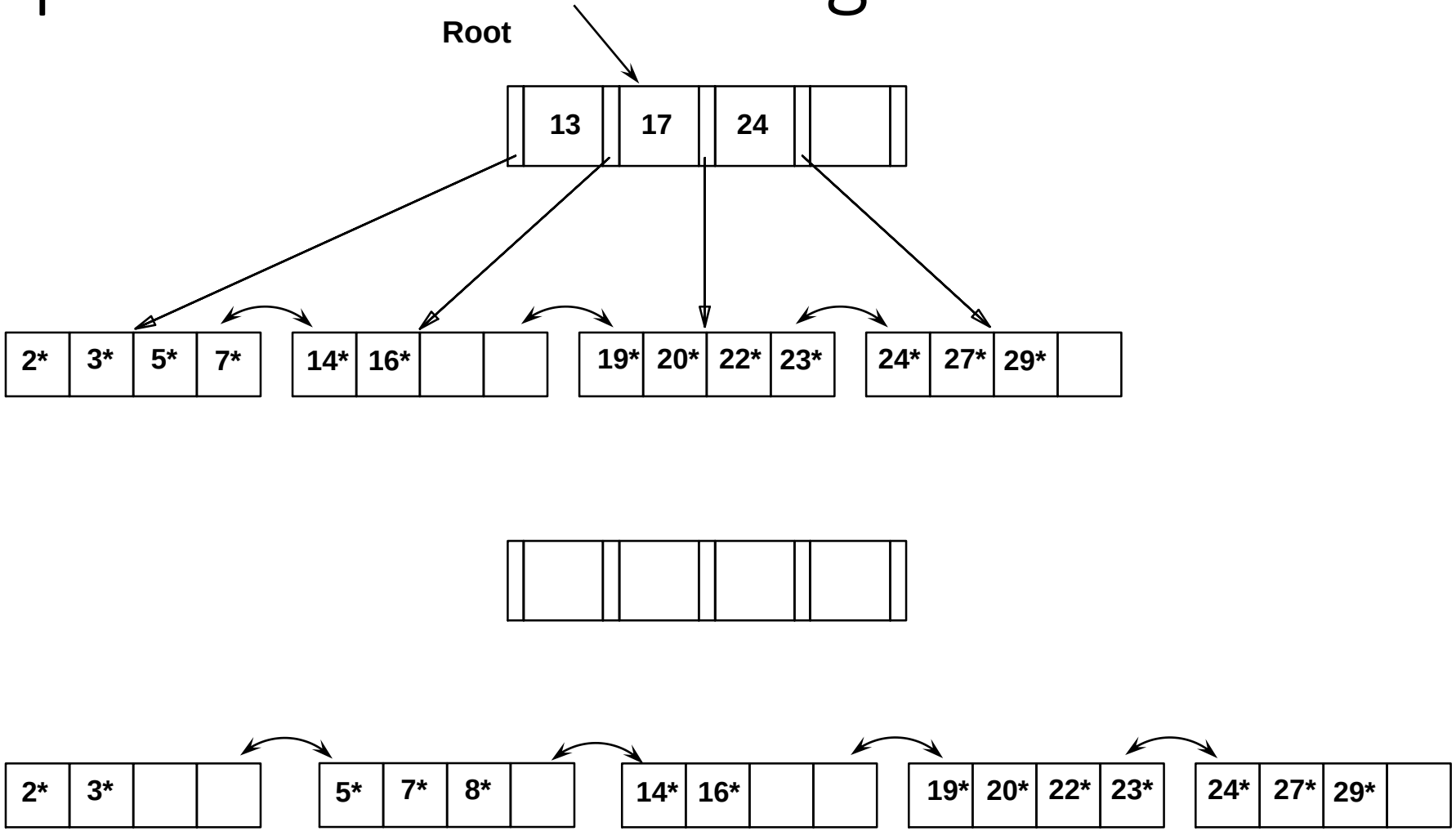
Example B+ Tree - Inserting 8*



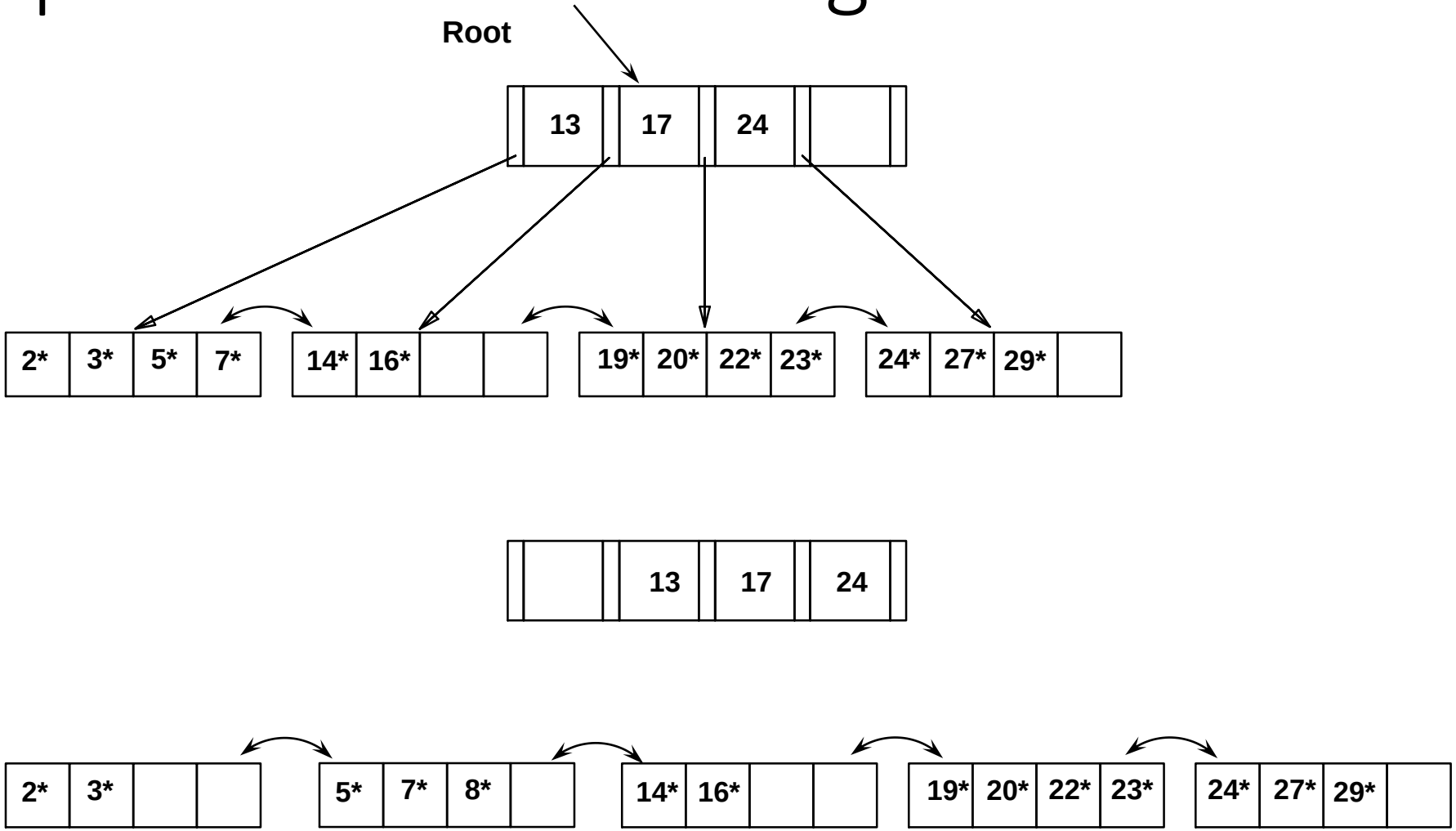
Example B+ Tree - Inserting 8*



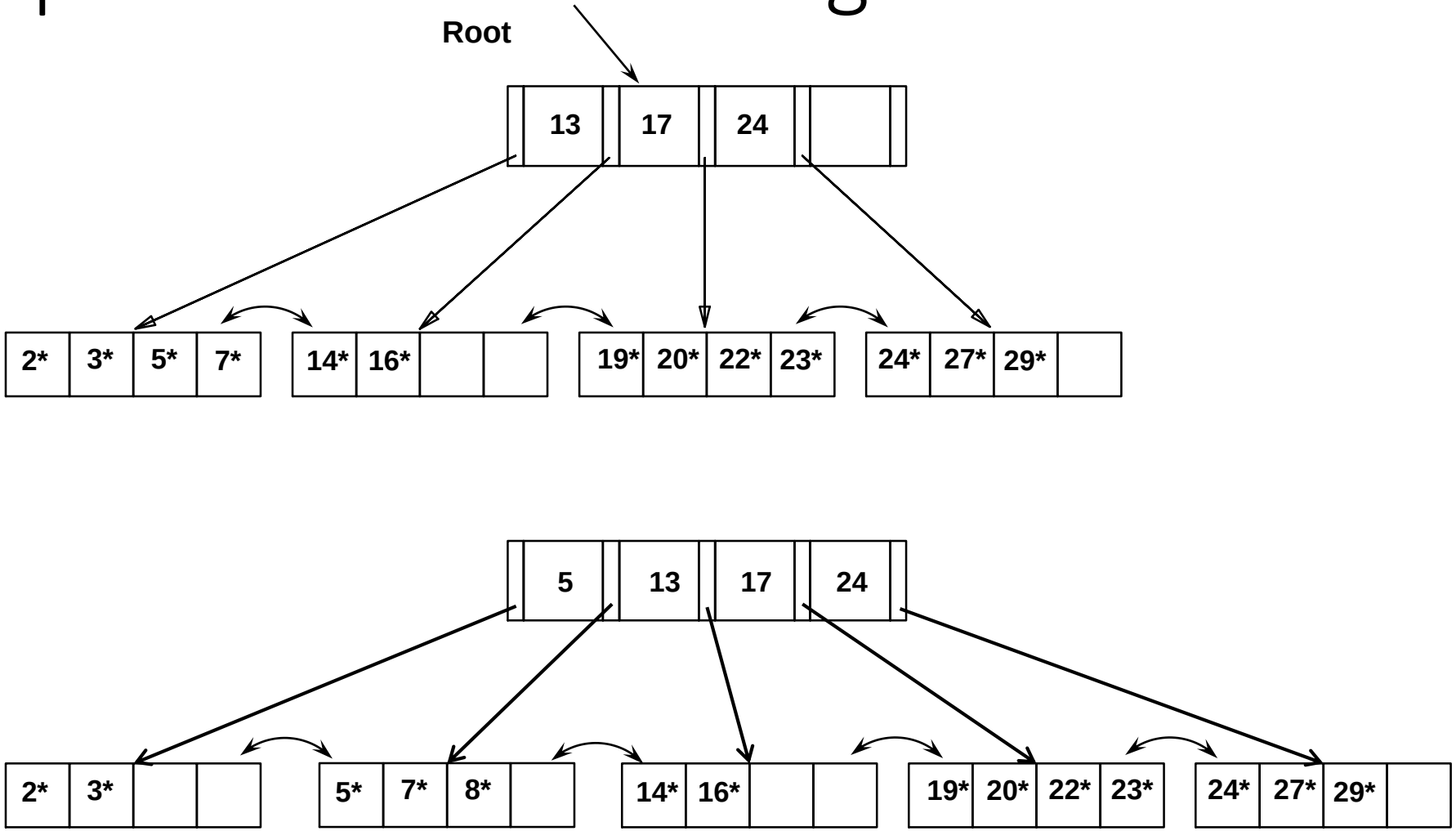
Example B+ Tree - Inserting 8*



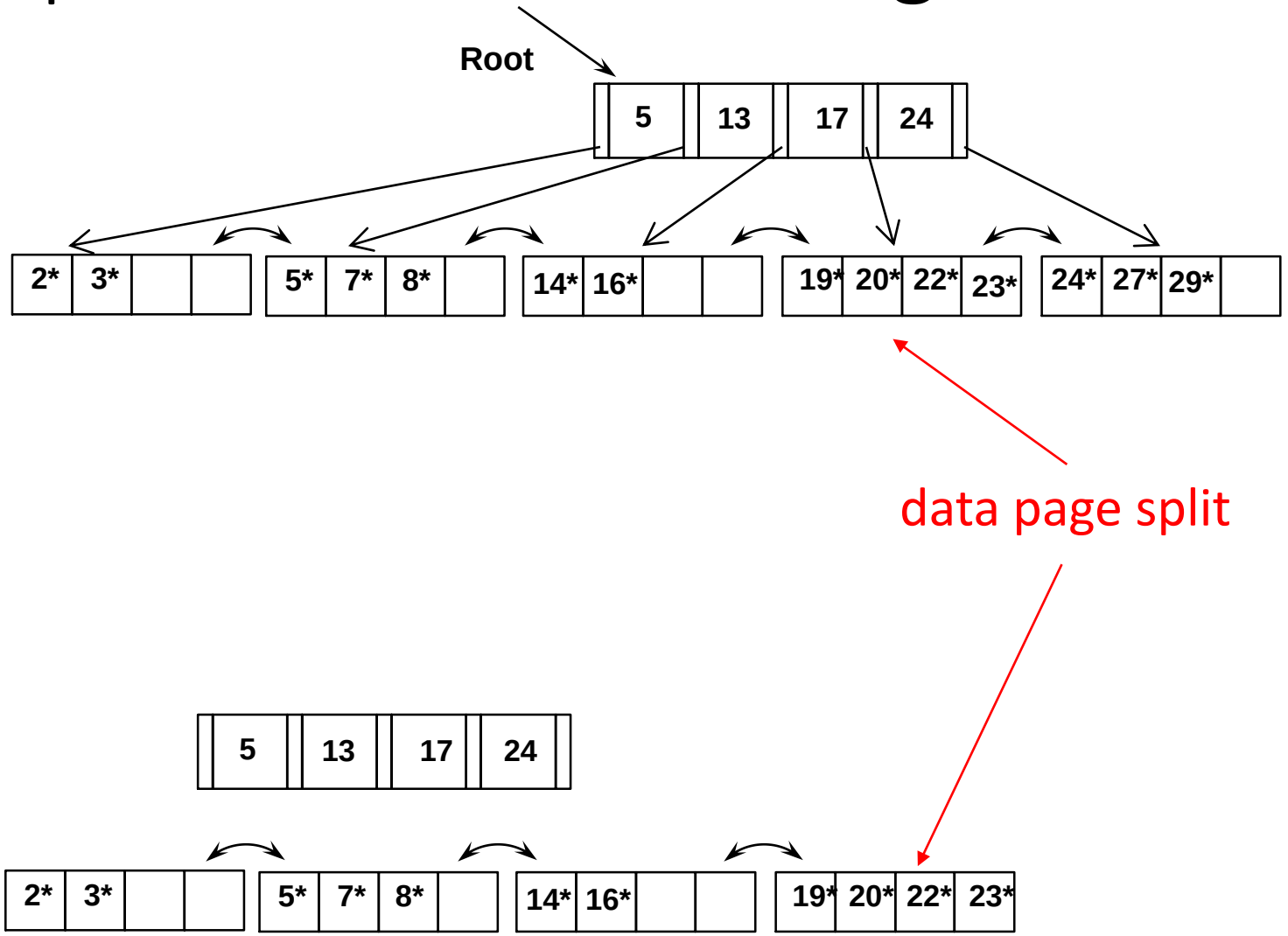
Example B+ Tree - Inserting 8*



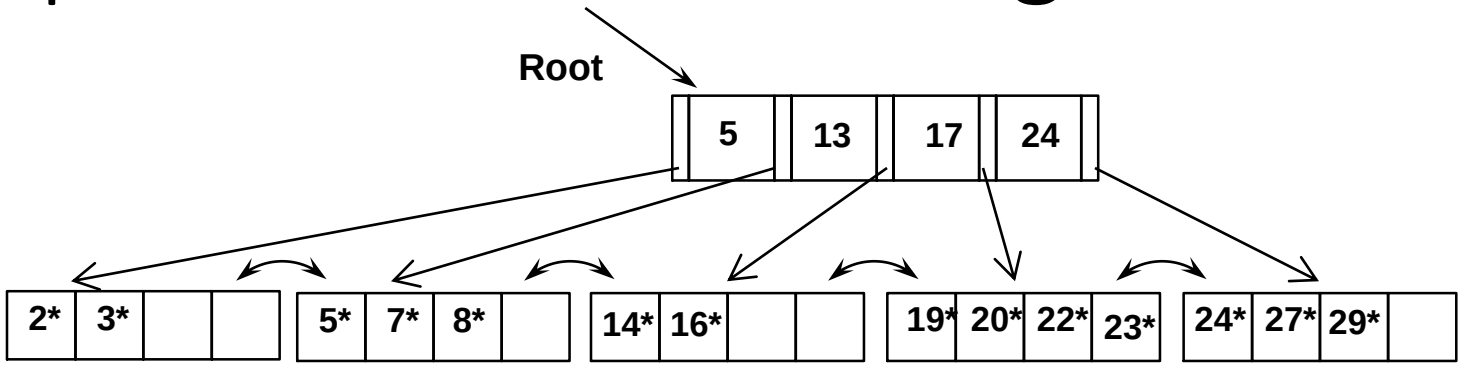
Example B+ Tree - Inserting 8*

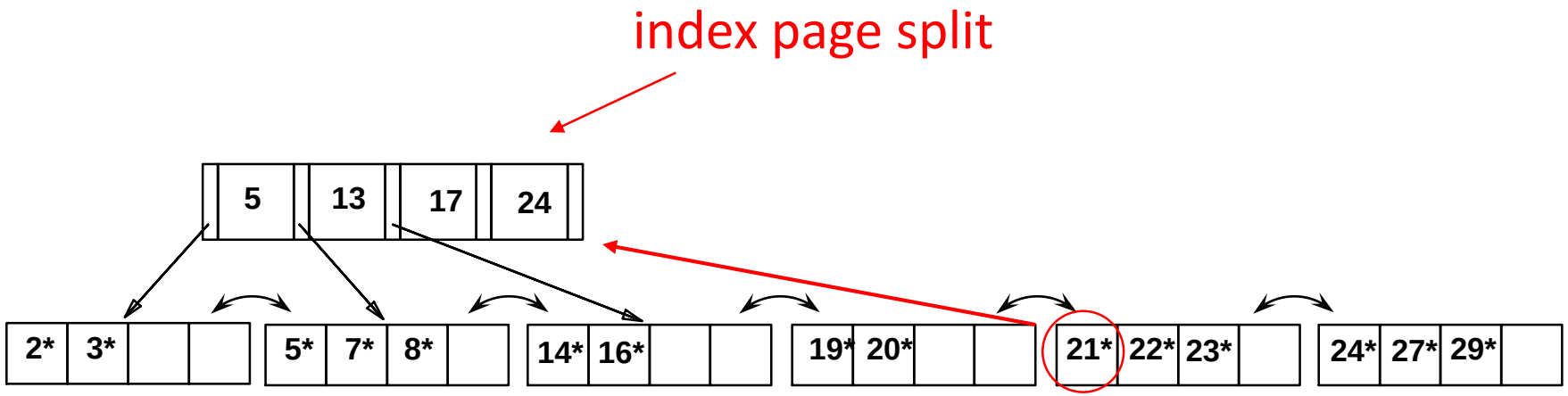


Example B+ Tree - Inserting 21*

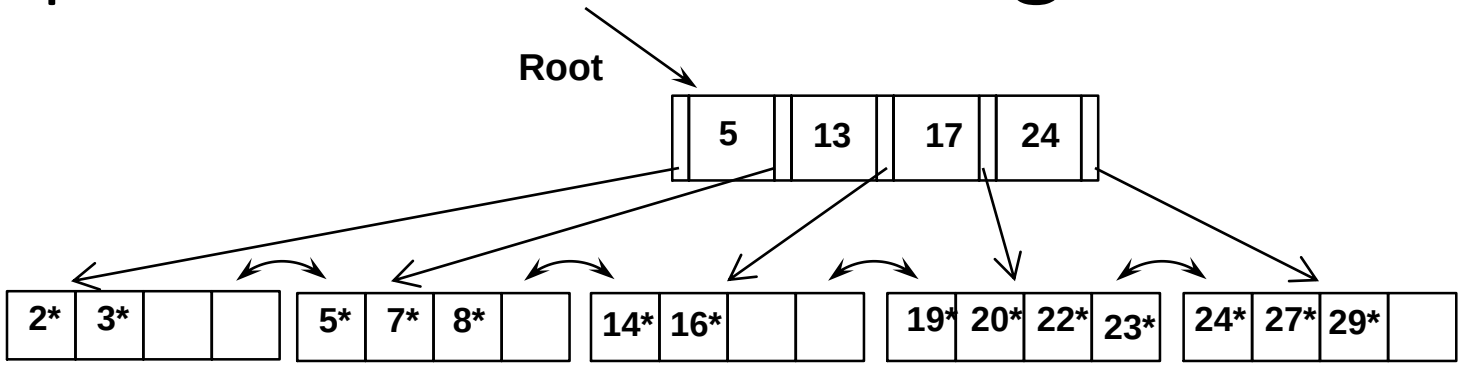


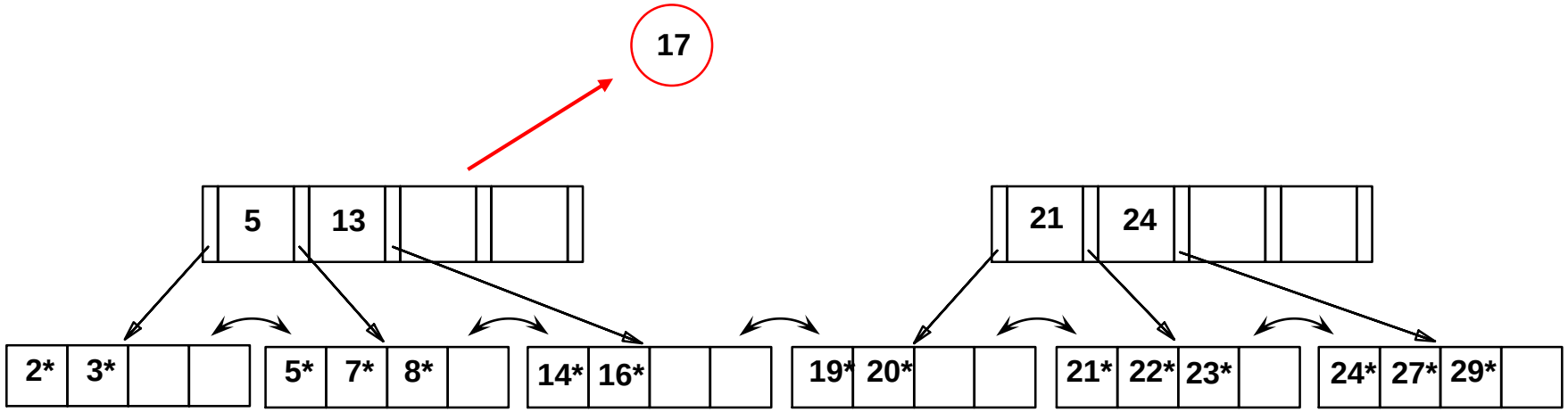
Example B+ Tree - Inserting 21*



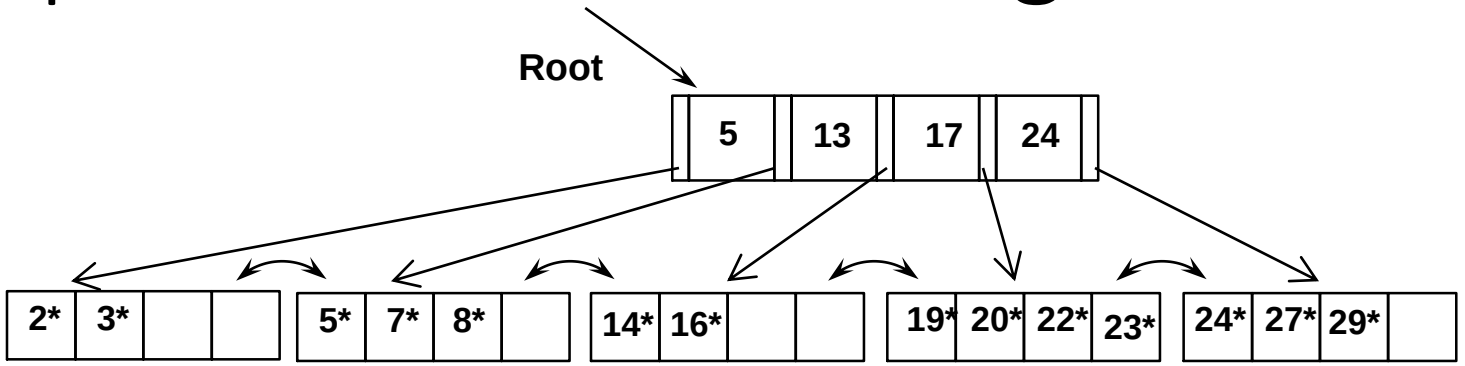


Example B+ Tree - Inserting 21*





Example B+ Tree - Inserting 21*



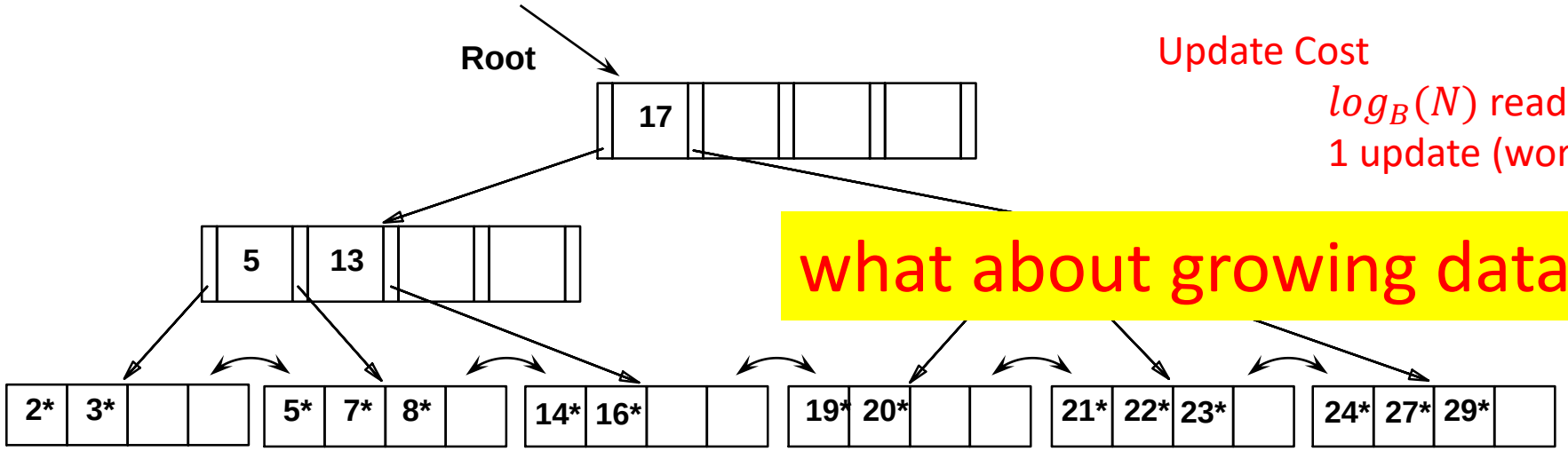
Read Cost: $log_B(N)$

Update Cost

$log_B(N)$ reads

1 update (worse case $log_B(N)$)

what about growing dataset size?

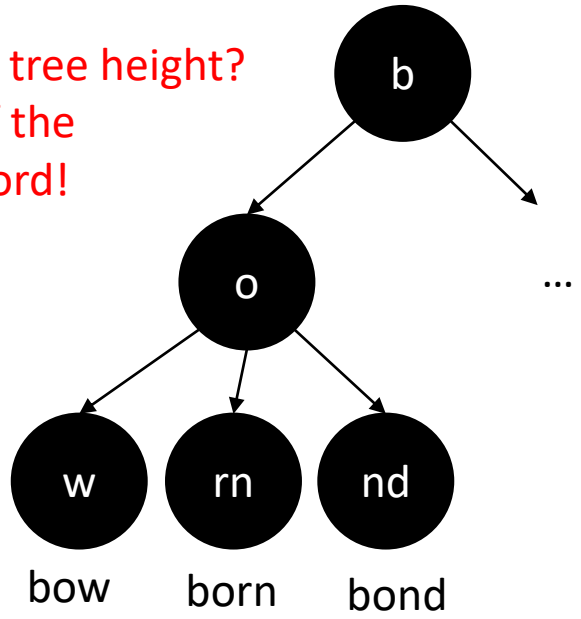


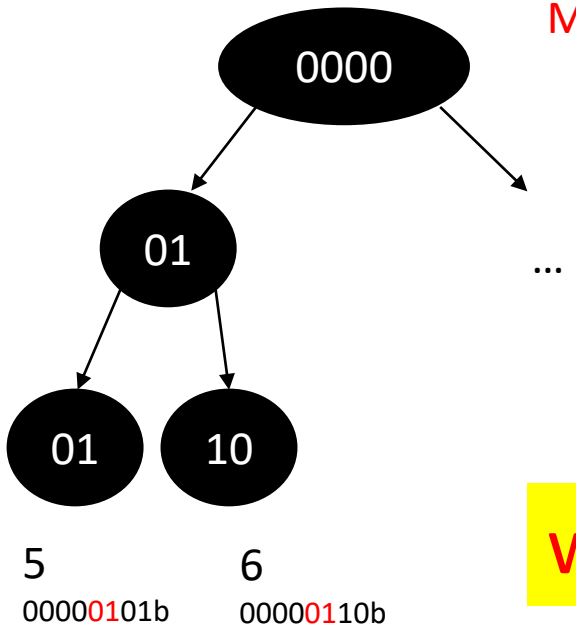
Radix Trees (special case of tries and prefix B-Trees)

Idea: use common prefixes for internal nodes to reduce size/height!

Binary representation of any domain can be used

Maximum tree height?
the size of the
longest word!





Maximum tree height?

8, that is, $log_2(\max\_domain\_value)$
fixed worst case!

what about data skew?

Bitmap Indexes

Column A	A=10	A=20	A=30
30	0	0	1
20	0	1	0
30	0	0	1
10	1	0	0
20	0	1	0
10	1	0	0
30	0	0	1
20	0	1	0

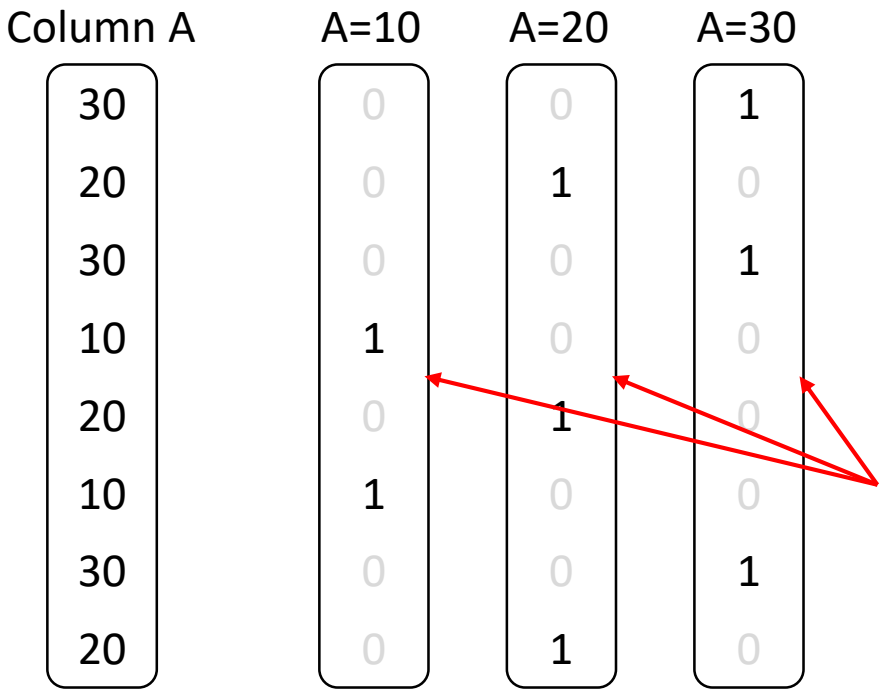
Speed & Size

- ✅ Compact representation of query result
- ✅ Query result is readily available

Bitvectors

- ✅ Can leverage fast Boolean operators
- ✅ Bitwise AND/OR/NOT faster than looping over meta data

Bitmap Indexes



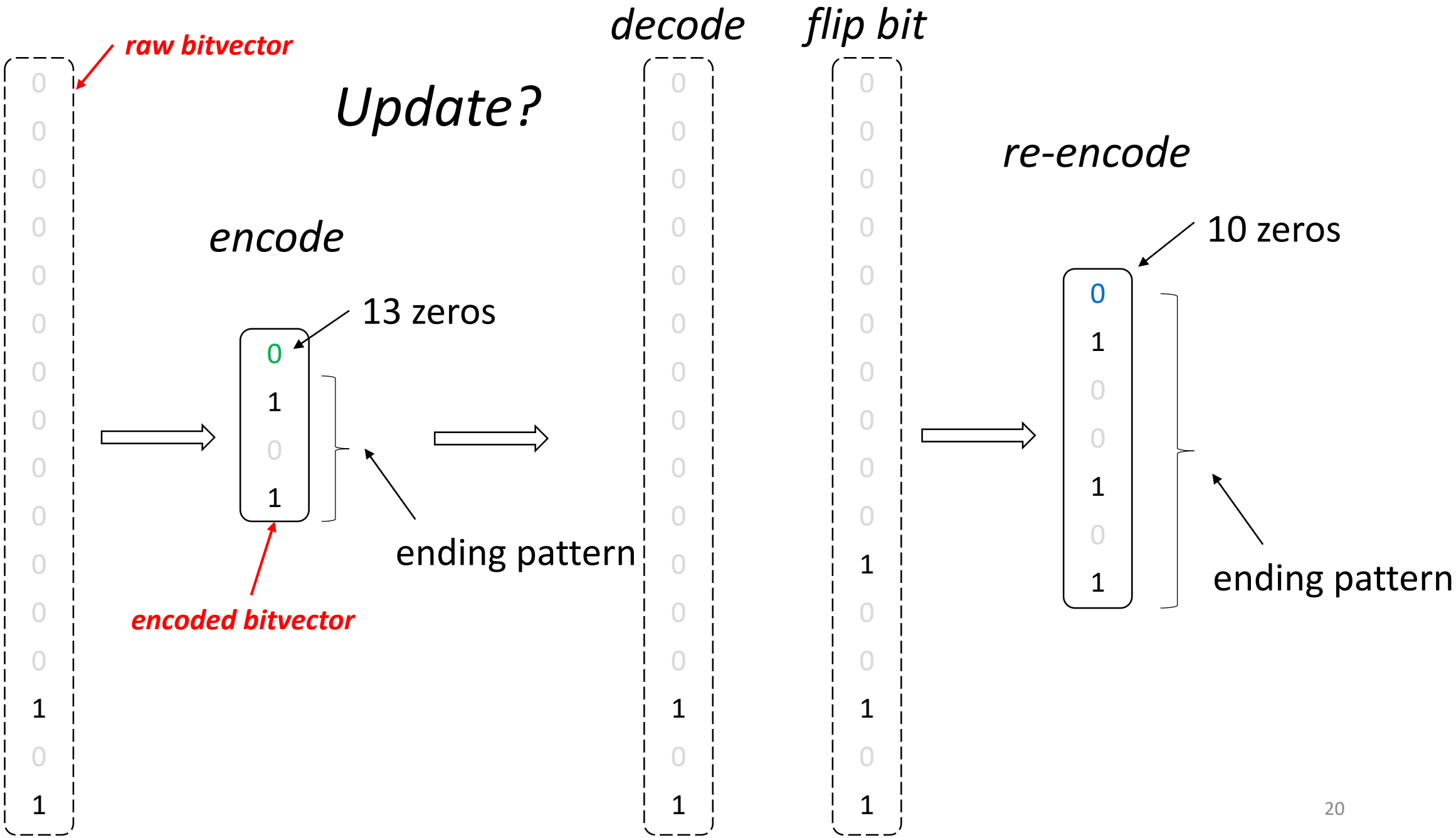
Index Size

Space-inefficient for domains with large cardinality

Addressed by bitvector encoding/compression

core idea: *run-length encoding* in prior work

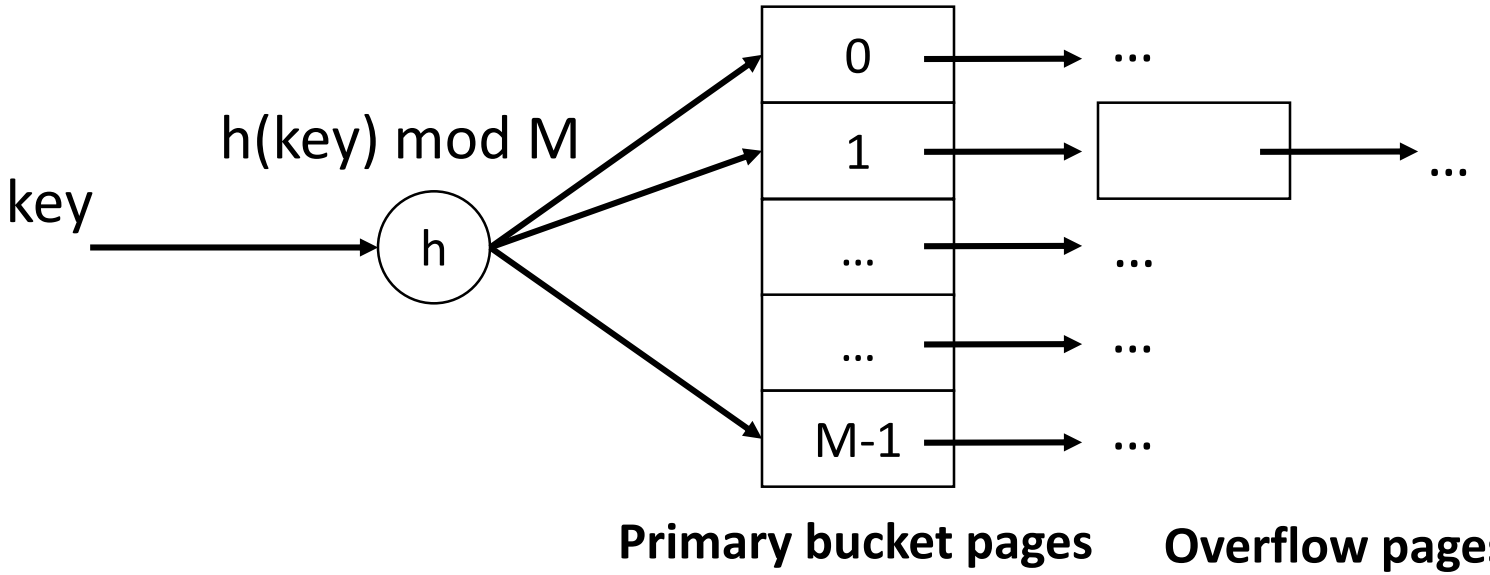
what about updates?


raw bitvector
Update?
decode
flip bit
encode
13 zeros
ending pattern
encoded bitvector
re-encode
10 zeros
ending pattern

Hash Indexes (static hashing)

#primary bucket pages fixed, allocated sequentially, never de-allocated; overflow pages if needed

h(k) mod M = bucket to insert data entry with key *k* (M: #buckets)



what if I have skew in the data set (or a bad hash function)?

Scan Accelerators

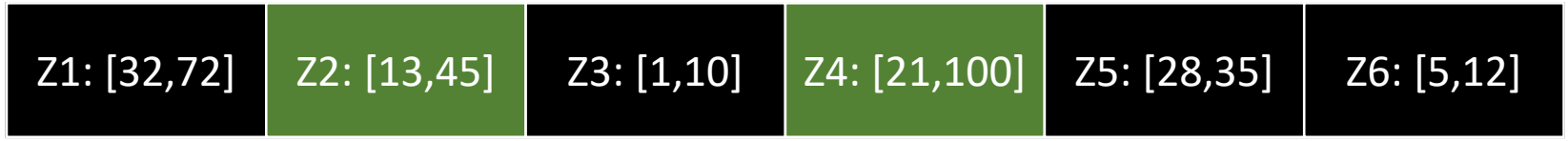
Zonemaps

Search for 25

Z1: [32,72]	Z2: [13,45]	Z3: [1,10]	Z4: [21,100]	Z5: [28,35]	Z6: [5,12]

Scan Accelerators

Zonemaps



Search for 25

Search for [5,11]

Scan Accelerators

Zonemaps

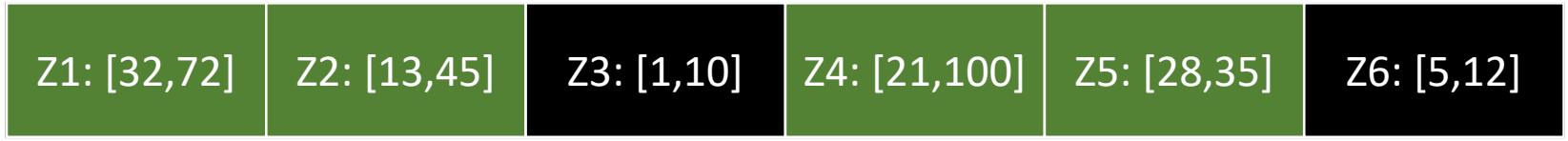
Search for 25

Search for [5,11]

Search for [31,46]

Scan Accelerators

Zonemaps



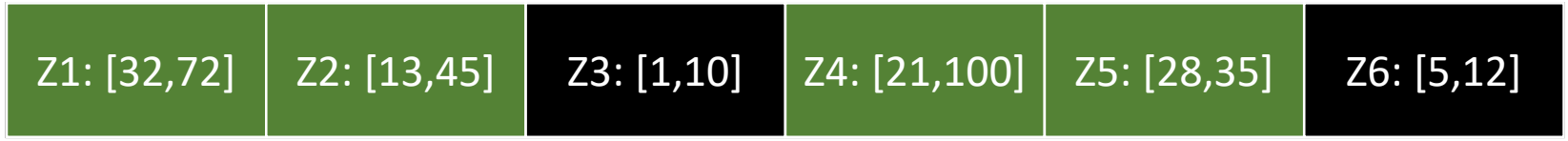
Search for 25

Search for [5,11]

Search for [31,46]

Scan Accelerators

Zonemaps



Search for 25

Search for [5,11]

Search for [31,46]

if data were sorted:

Search for 25

Search for [5,11]

Search for [31,46]

Scan Accelerators

Zonemaps

Z1: [32,72]	Z2: [13,45]	Z3: [1,10]	Z4: [21,100]	Z5: [28,35]	Z6: [5,12]

Search for 25

Search for [5,11]

Search for [31,46]

if data were sorted:

Z1: [1,15]	Z2: [16,30]	Z3: [31,50]	Z4: [50,67]	Z5: [68,85]	Z6: [85,100]

Search for 25

Search for [5,11]

Search for [31,46]

what if data is perfectly uniformly distributed?

Z1: [1,99]	Z2: [2,95]	Z3: [1,100]	Z4: [2,100]	Z5: [3,97]	Z6: [2,99]

What are the possible ***index designs***?

	Data Organization	Point Queries	Short Range Queries	Long Range Queries	Data Skew	Updates	Affected by Physical Order
B+ Trees	Range	✓	✓	✓	✓	✓	–
LSM Trees	Insertion & Sorted	✓	✗	✓	✓	✓	–
Radix Trees	Radix	✓	✓	✓	✗	–	–
Hash Indexes	Hash	✓	–	✗	✗	✓	–
Bitmap Indexes	None	✓	–	✗	–	✗	***no***
Scan Accelerators	None	✗	–	✓	✓	–	***yes***

Adaptive Data Organization: Database Cracking

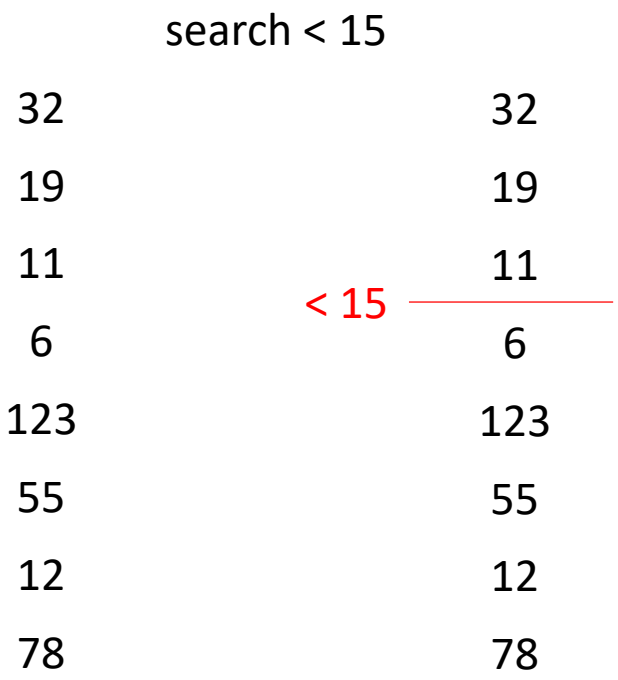
idea: there is an ***ideal*** data organization

what is it (for a column of integers)?

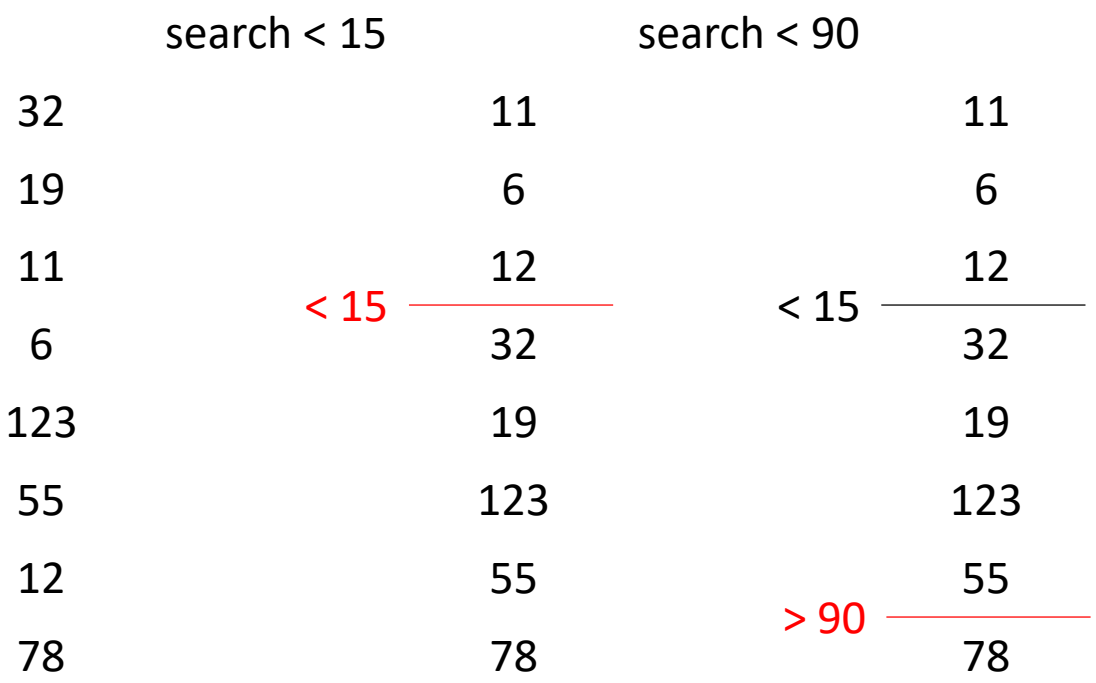
sorted!

we can reach it ***eventually*** if we use the ***workload as a hint***

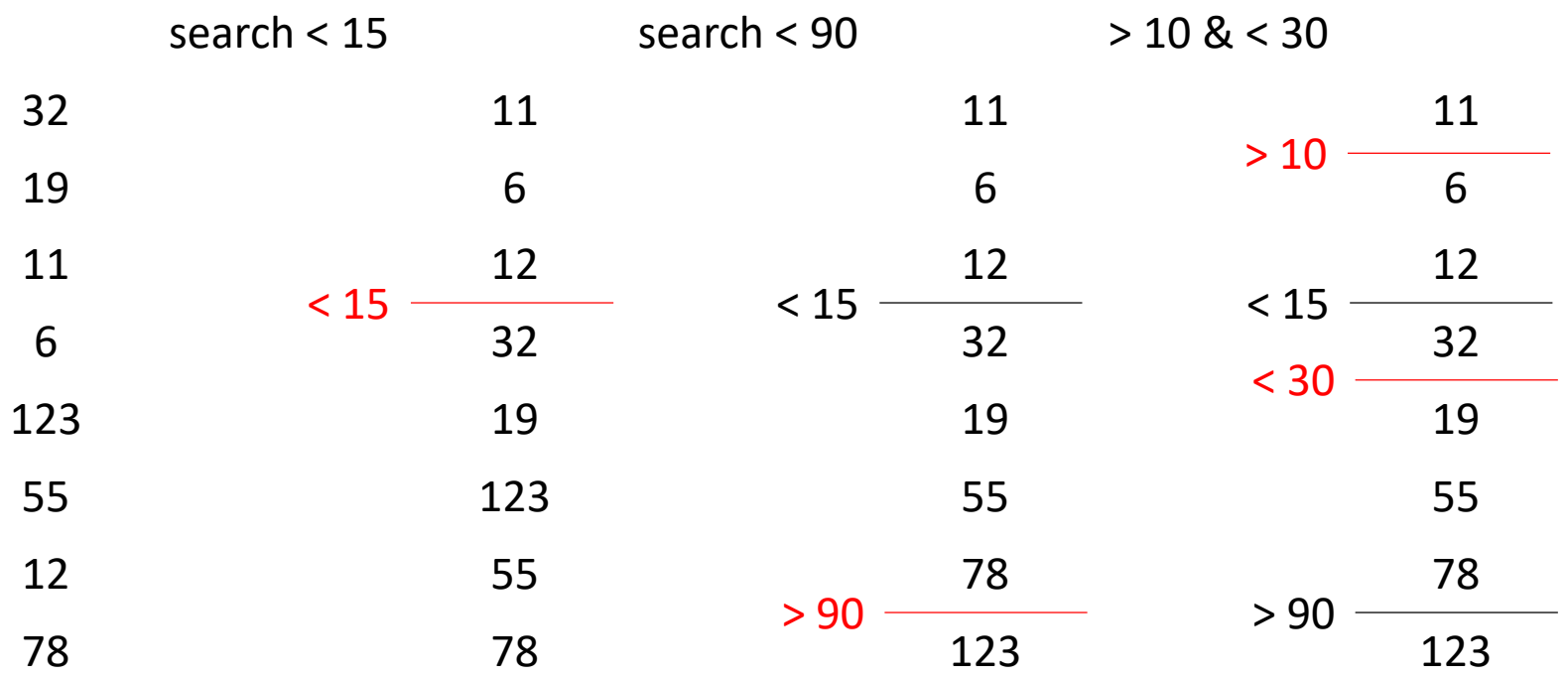
Adaptive Data Organization: Database Cracking



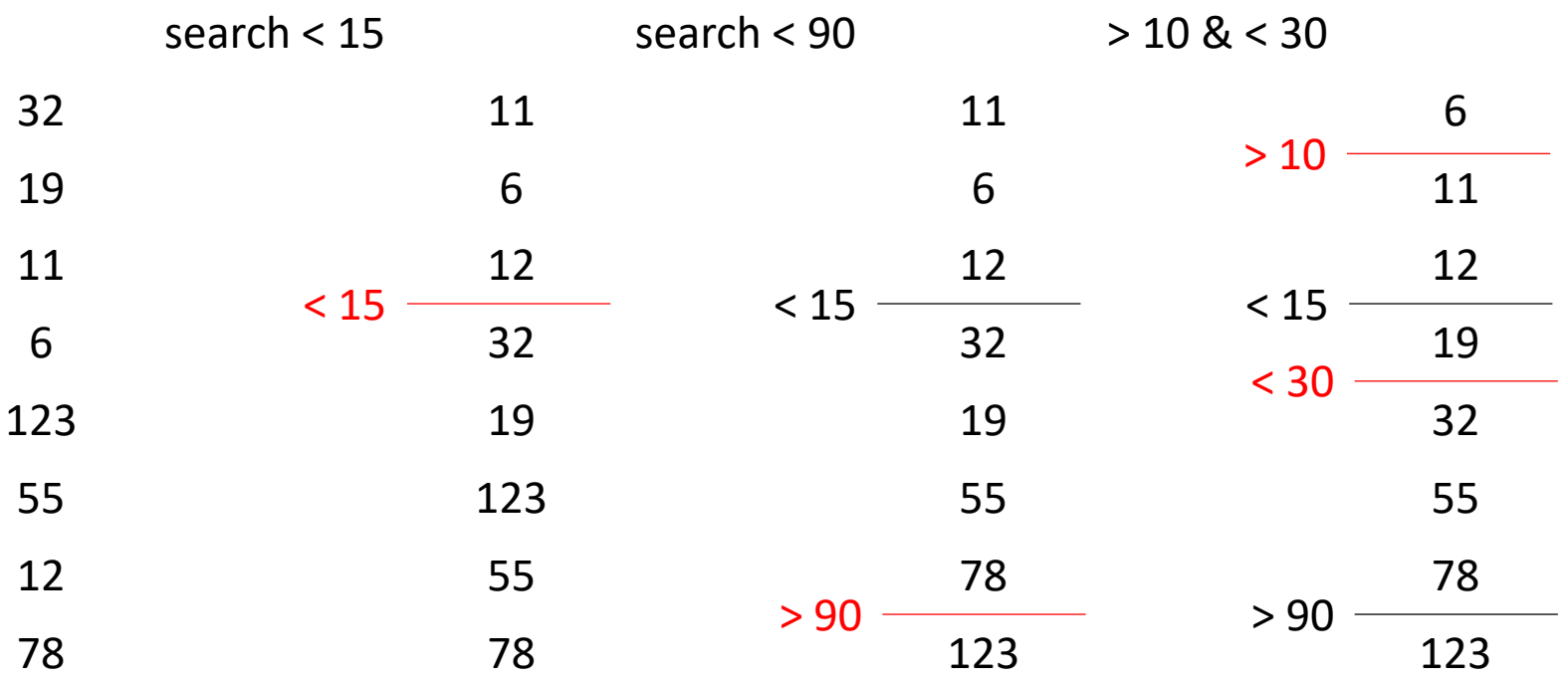
Adaptive Data Organization: Database Cracking



Adaptive Data Organization: Database Cracking



Adaptive Data Organization: Database Cracking



what about updates/inserts?

LSM-tree Project Implementation

What to plan for the implementation (1/3)

Durable Database (open/close without losing state)

Components:

- Memory buffer (array, hashtable, B+ tree)
- Files (sorted levels/tiers)
- Fence pointers (**Zonemaps**)
- **Bloom filters**

What to plan for the implementation (2/3)

Durable Database (open/close without losing state)

Components:

- Memory buffer (search, read, write, unpin)
- Priority data structure
- Eviction policy

What to plan for the implementation (3/3)

API + basic testing and benchmarking available at:

LSM Implementation:
https://github.com/BU-DiSC/cs561_templatedb
with a Reference Bloom filter implementation

Bufferpool Implementation:
https://github.com/BU-DiSC/cs561_templatebufferpool

BOSTON UNIVERSITY

CS 561: Data Systems Architectures

Introduction to Indexing:

Trees, Tries, Hashing, Bitmap Indexes, Database Cracking

Zichen Zhu

https://bu-disc.github.io/CS561/