Database Development and Design (CPT201)

Lecture 3b:
B+ Tree Index

Dr. Wei Wang

Department of Computing

Learning Outcomes

- B+-Tree Index
  - Queries
  - update

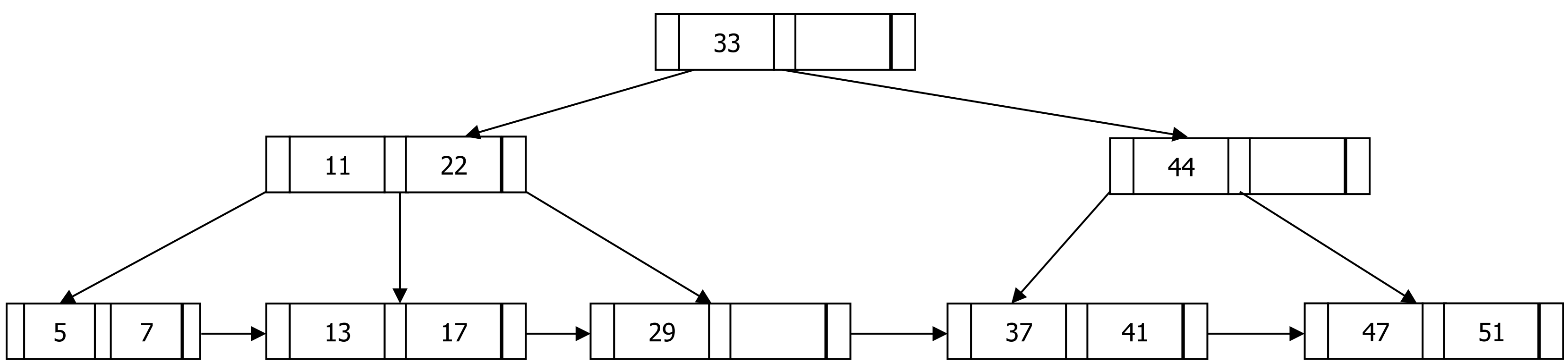
B^+-Tree Index

- B^+-Tree is "short" and "Fat"
  - Disk-based: usually one node per block; large fan-out
  - Balanced (more or less): good performance guarantee.
- In a B^+-Tree,
  - n (or sometimes N) is the number of pointers in a node; pointers: P1, P2, …Pn
  - Search keys: K1 < K2 < K3 < . . . < Kn–1
  - All paths (from root to leaf) have same length
  - Root must have at least two children
  - In each non-leaf node (inner node), more than 'half' ($\geq\lceil n/2 \rceil$) pointers must be used
  - Each leaf node must contain at least $\lceil (n-1)/2 \rceil$ keys

P_1	K_1	P_2	$\ldots$	P_{n-1}	K_{n-1}	P_n

Example

- An Example B+-Tree with n = 3
  - All paths have same length.
  - Root has (at least) two children
  - In each non-leaf node (inter node), more than half ($\geq\lceil 3/2 \rceil = 2$) pointers are used
  - Each leaf node contains at least $\lceil (3-1)/2 \rceil = 1$ key



Queries on B^+-Trees

- Find record with search-key value V.
  - 1. C=root
  - 2. While C is not a leaf node
    - 2.1. Let i be least value such that V ≤ Ki.
    - 2.2. If no such exists, set C = last non-null pointer in C
    - 2.3. Else { if (V= Ki) Set C = Pi +1 else set C = Pi}
  - 3. Let i be least value such that Ki = V
  - 4. If there is such a value i, follow pointer Pi to the desired record.
  - 5. Else no record with search-key value V exists.

Observations about B$^+$-trees

- Since the inter-node connections are done by pointers, "logically" close blocks need not be "physically" close.
- The non-leaf levels of the B$^+$-tree form a hierarchy of sparse indices.
- If there are K search-key values in the file
  - The B$^+$-tree height is no more than $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$.
  - Level below root has at least 2* $\lceil n/2 \rceil$ values
  - Next level has at least 2* $\lceil n/2 \rceil$ * $\lceil n/2 \rceil$ values
  - .. etc.

Observations about B^+-trees cont'd

- Searching can be conducted efficiently.
  - a node is generally the same size as a disk block, typically 4 kilobytes
  - n is typically around 100 (40 bytes per index entry).
  - with 1 million search key values and n = 100
  - at most $\log_{50}(1{,}000{,}000) = 4$ nodes are accessed in a lookup.
- Insertion and deletion to the main file can be handled efficiently, as the index can be restructured in logarithmic time.

Updates on B^+-Trees: Insertion

- 1. Find the leaf node in which the search-key value would appear
- 2. If the search-key value is already present in the leaf node
  - 2.1. Add record to the file
  - 2.2. If necessary add a pointer to the bucket.
- 3. If the search-key value is not present, then
  - 3.1. add the record to the main file (and create a bucket if necessary)
  - 3.2. If there is room in the leaf node, insert (key-value, pointer) pair in the leaf node
  - 3.3. Otherwise, split the node (along with the new (key-value, pointer) entry) as discussed in the next slide.

Updates on B^{+}-Trees: Insertion cont'd

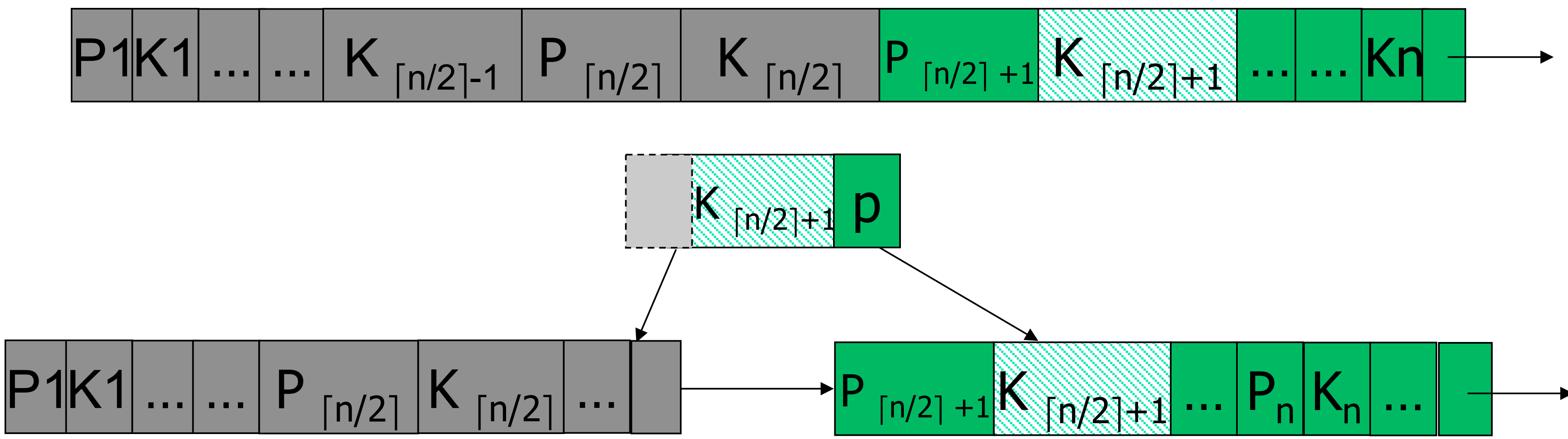
- Splitting a leaf node:
  - take the (search-key value, pointer) pairs and the one being inserted) in an in-memory area M in sorted order. Assume there are n search key values in total.
  - Place the first $\lceil n/2 \rceil$ in the original node, and the rest in a new node.
  - let the new node be p, and let k be the least key value in p. Insert (k,p) in the parent of the node being split.
  - If the parent is full, split it and propagate the split further up.
- Splitting of nodes proceeds upwards till a node that is not full is found.
  - In the worst case the root node may be split, increasing the height of the tree by 1.

Updates on B$^+$-Trees: Insertion cont'd

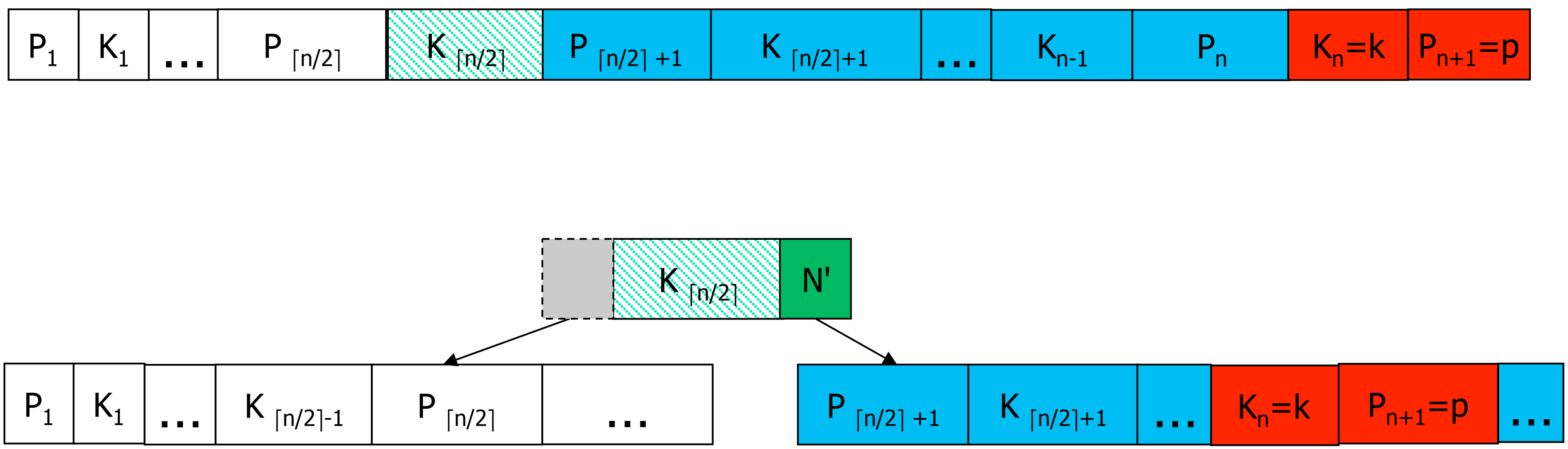
- Splitting a non-leaf node: when inserting (k,p) into an already full internal node N
  - Copy N to an in-memory area M with space for n+1 pointers and n keys
  - Insert (k,p) into M in sorted order
  - Copy $P_1, K_1, \ldots, K_{\lceil n/2 \rceil - 1}, P_{\lceil n/2 \rceil}$ from M back into node N
  - Copy $P_{\lceil n/2 \rceil + 1}, K_{\lceil n/2 \rceil + 1}, \ldots, K_n, P_{n+1}$ from M into newly allocated node N'
  - Insert ($K_{\lceil n/2 \rceil}$,N') into parent N

Xi'an Jiaotong-Liverpool University 西交利物浦大學

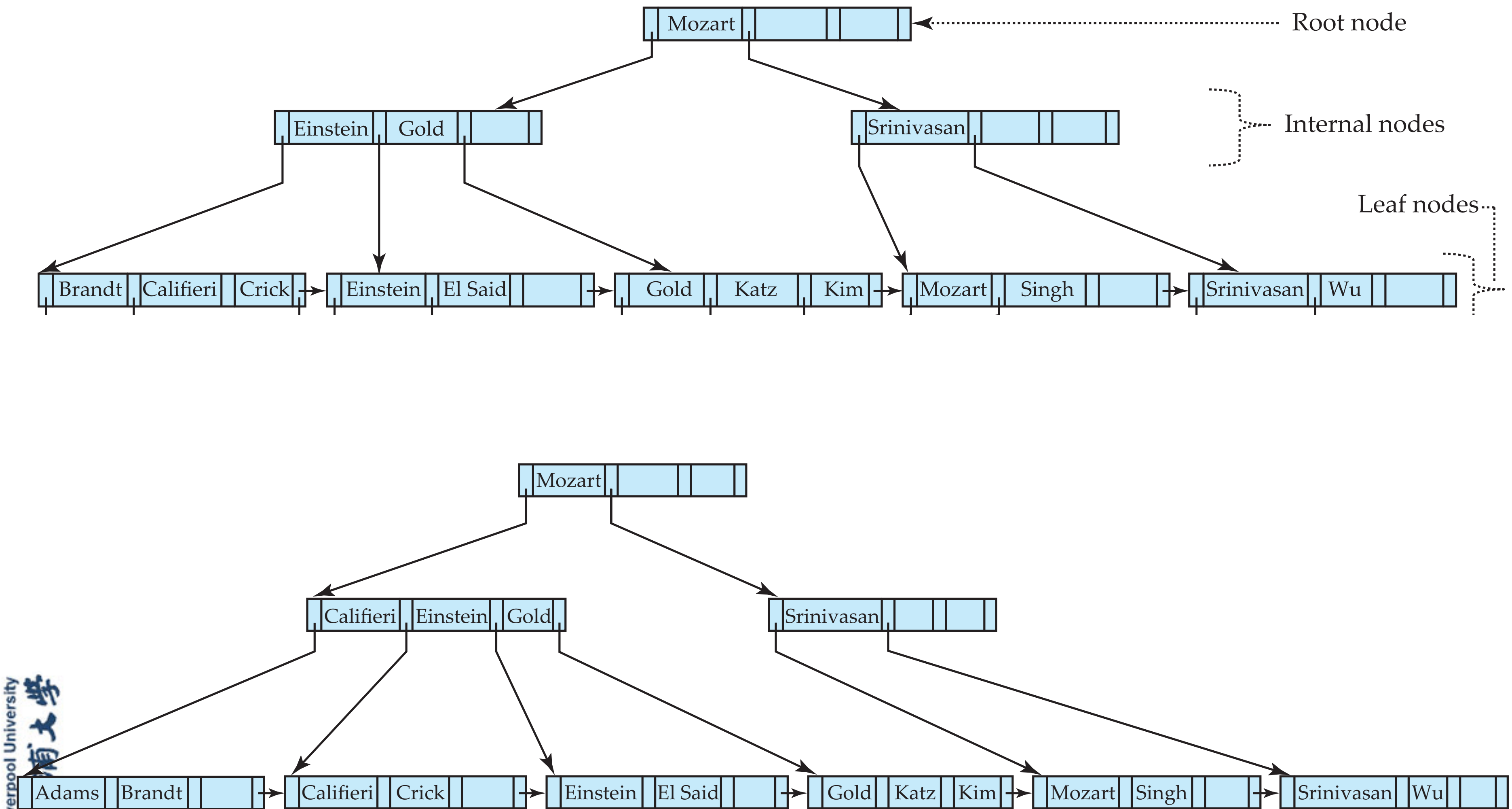
Splitting a Leaf Node



Splitting a Non-leaf Node

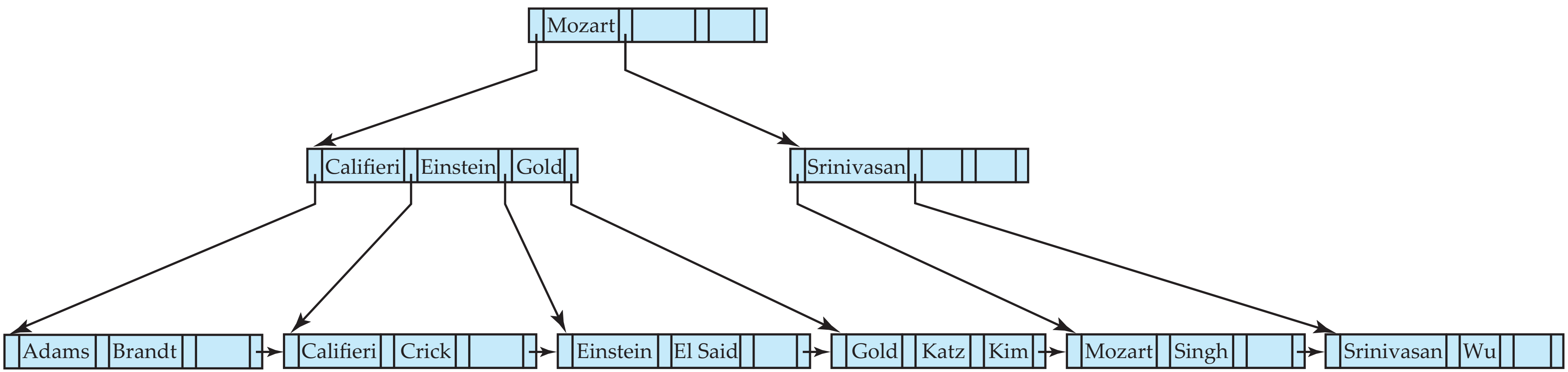


Insertion Example



B⁺-Tree before and after insertion of “Adams”

Insertion Example cont'd



Question:

What will happen after insertion of "Lamport"?

Read pseudocode in textbook!

Exercise

- Construct a B+ tree for the following set of key values for n=3.
  - (2, 3, 5, 7, 11, 13, 17)

Updates on B^+-Trees: Deletion

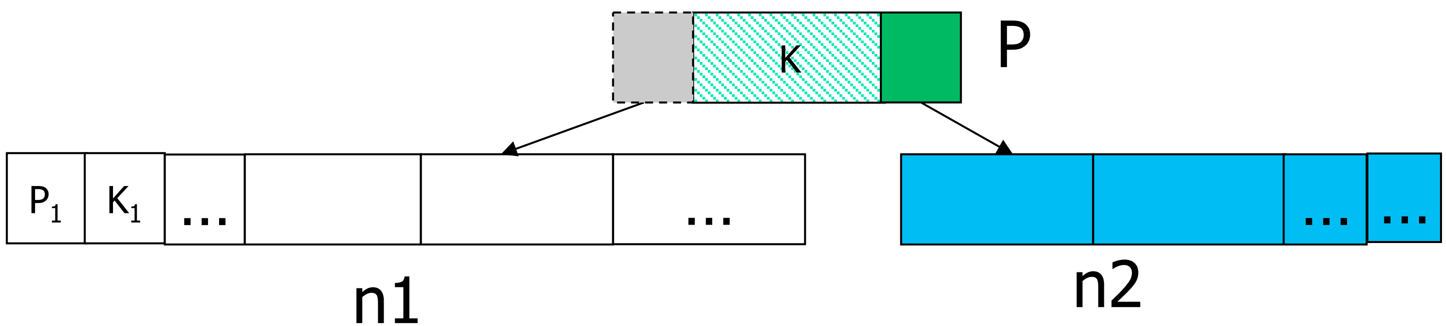
- Find the record to be deleted, and remove it from the main file and from the bucket (if present)
- Remove (search-key value, pointer) from the leaf node if there is no bucket or if the bucket has become empty
- If the node has too few entries due to the removal, and the entries in the node and a sibling fit into a single node, then merge siblings:
  - Insert all the search-key values in the two nodes into a single node, and delete the other node.
  - If it is a non-leaf node, copy the value from the parent (between the two nodes) into the merged node
  - Delete the the value from the parent (between the two nodes). (Change may propagate to upper levels.)

Updates on B+-Trees: Deletion cont'd

- Otherwise, if the node has too few entries due to the removal, but the entries in the node and a sibling do not fit into a single node, then redistribute pointers:
  - Redistribute the pointers between the node and a sibling such that both have more than the minimum number of entries; update the corresponding search-key value in the parent of the node.
  - If leaf node: take a proper value from sibling (value removed from sibling) and insert it to the underfull node; update the value in parent.
  - If non-leaf node: insert the value at (and remove from) parent to the underfull node, remove the value from sibling and update the parent.
  - Read pseudocode in textbook!
- The node deletions may cascade upwards till a node which has $\lceil n/2 \rceil$ or more pointers is found.
- If the root node has only one pointer after deletion, it is deleted and the sole child becomes the root.

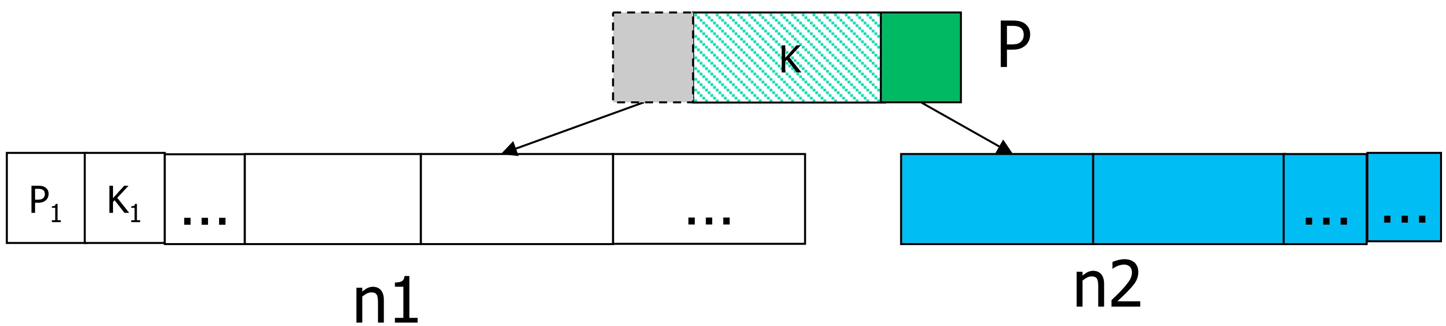
Merge Siblings – at Leaf Node

- Merge siblings n1 and n2
- Delete K (and the appropriate pointer) from parent P



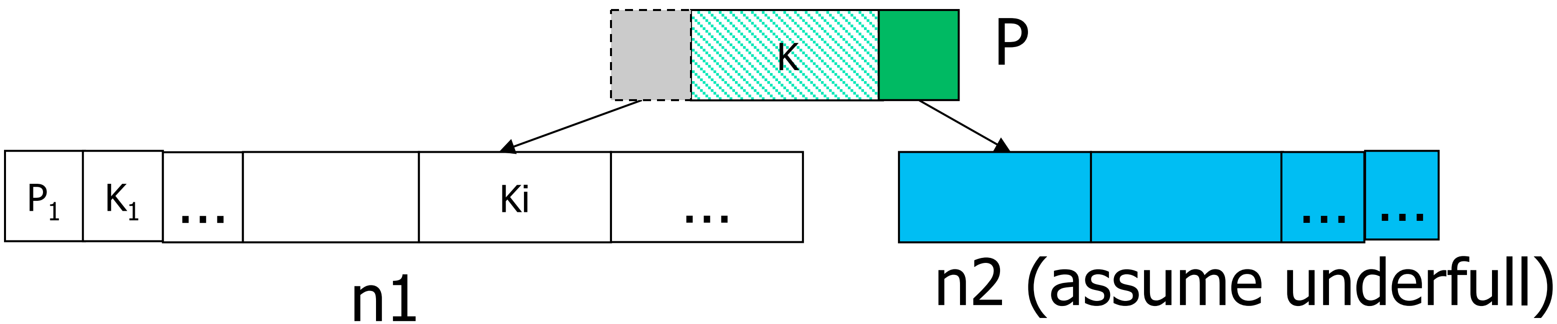
Merge Siblings – at non-Leaf Node

- Merge siblings n1 and n2 and K
- Delete K (and the appropriate pointer) from parent P



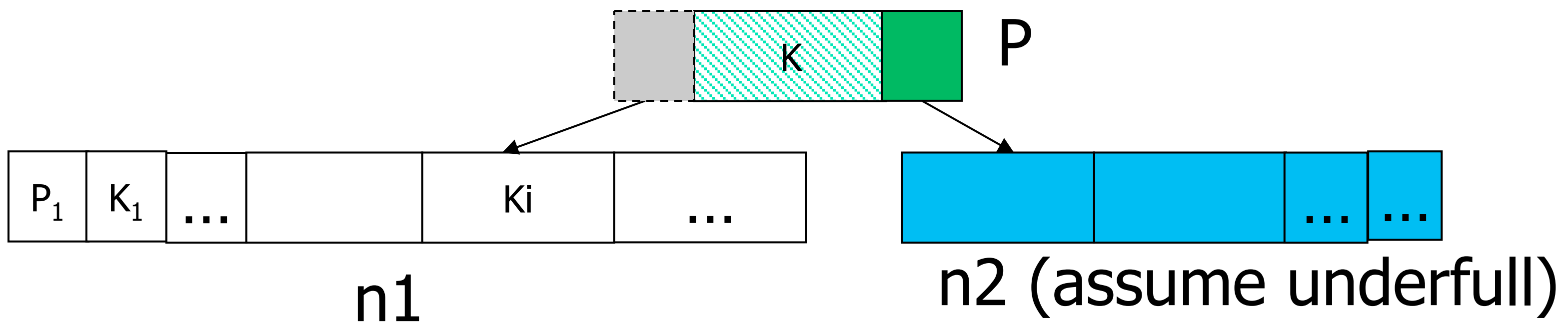
Redistribute Pointers – at Leaf Node

- Copy Ki from n1 and add it to n2
- Delete Ki from n1
- Replace the old value K in parent P with Ki

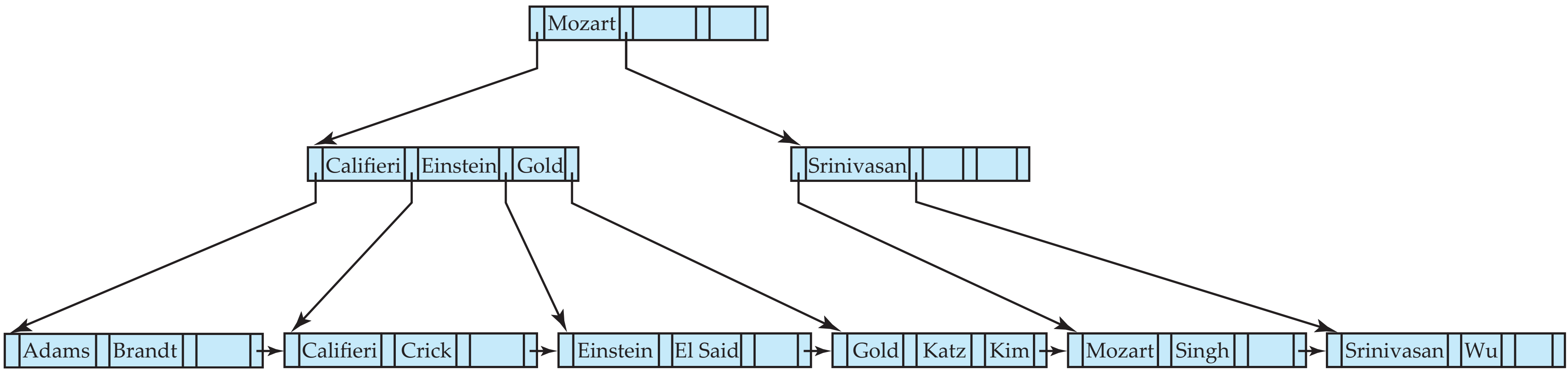


Redistribute Pointers – at non-Leaf Node

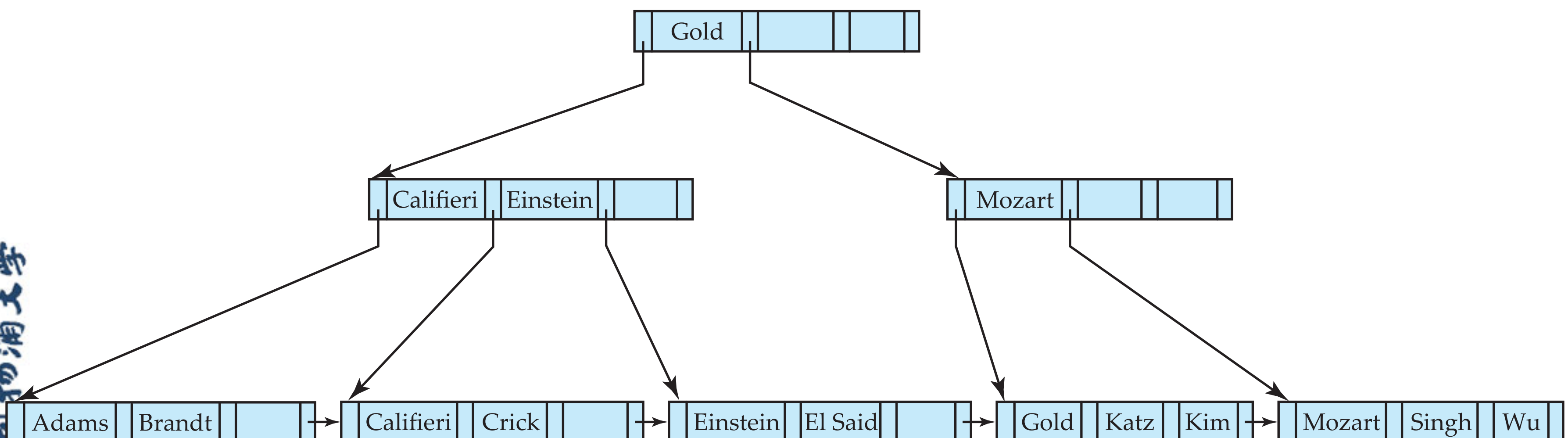
- Copy K from parent P and add it to n2
- Replace the old value K in parent P with Ki from n1
- Delete Ki from n1



Deletion Example

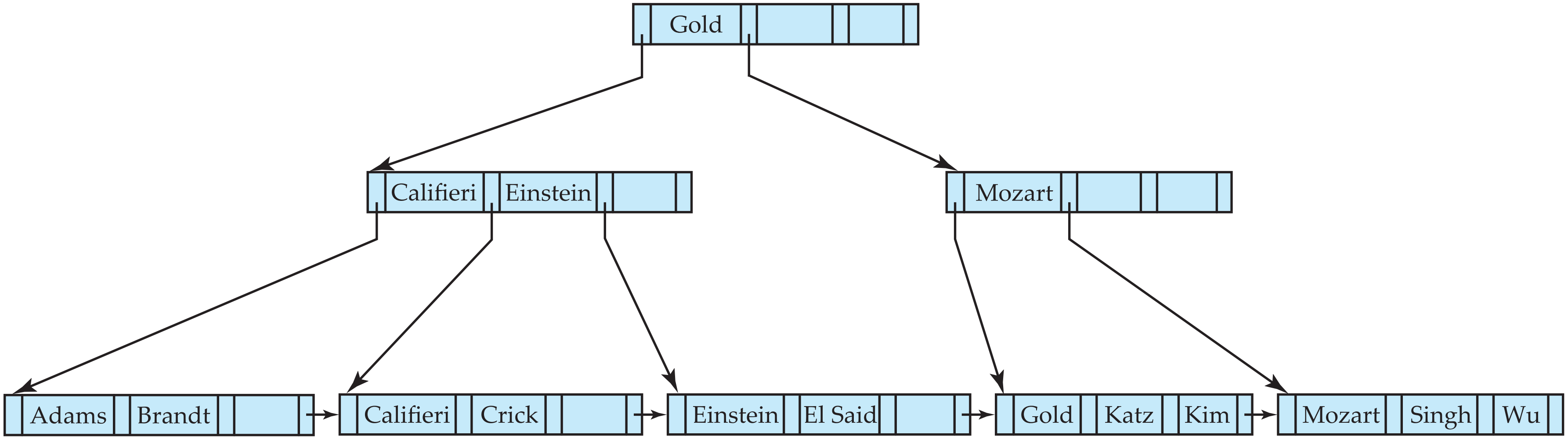


Before and after deleting “Srinivasan”

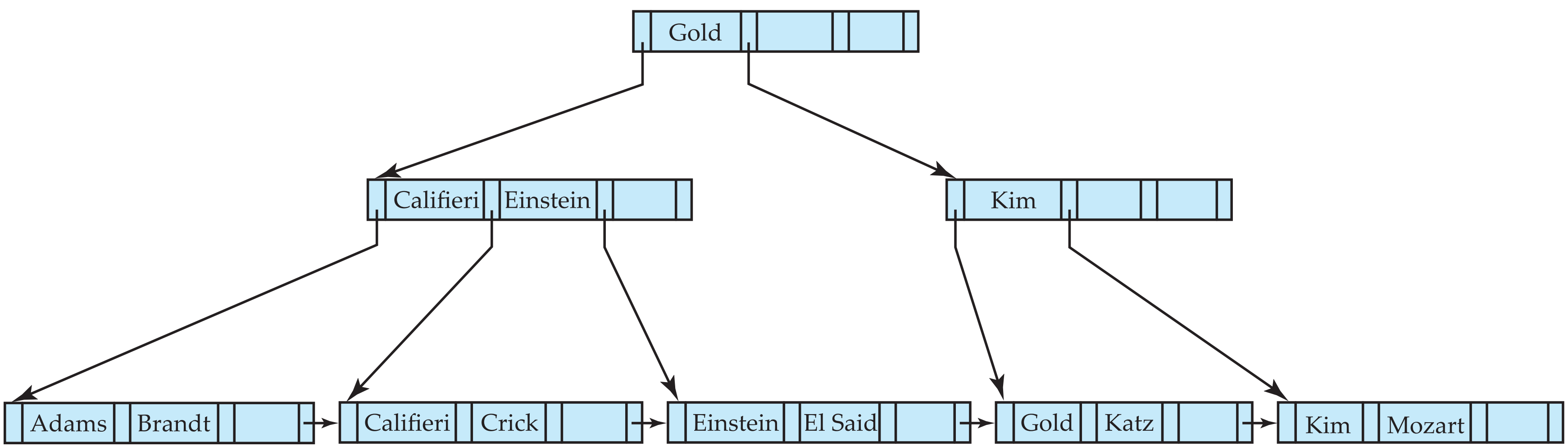


- Deleting “Srinivasan” causes merging of under-full leaves

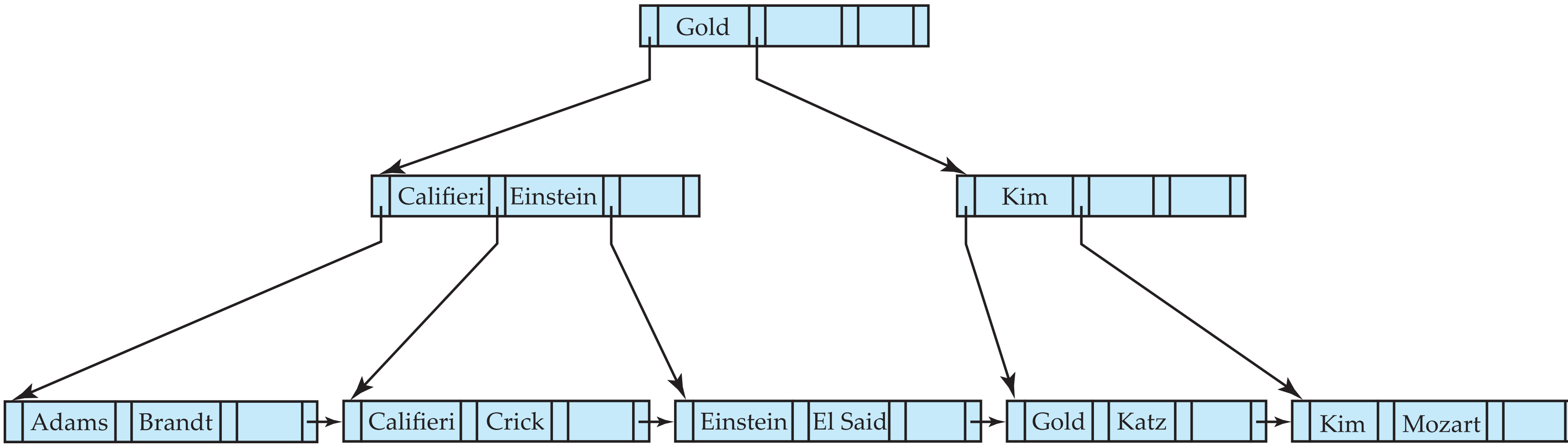
Deletion Example cont'd



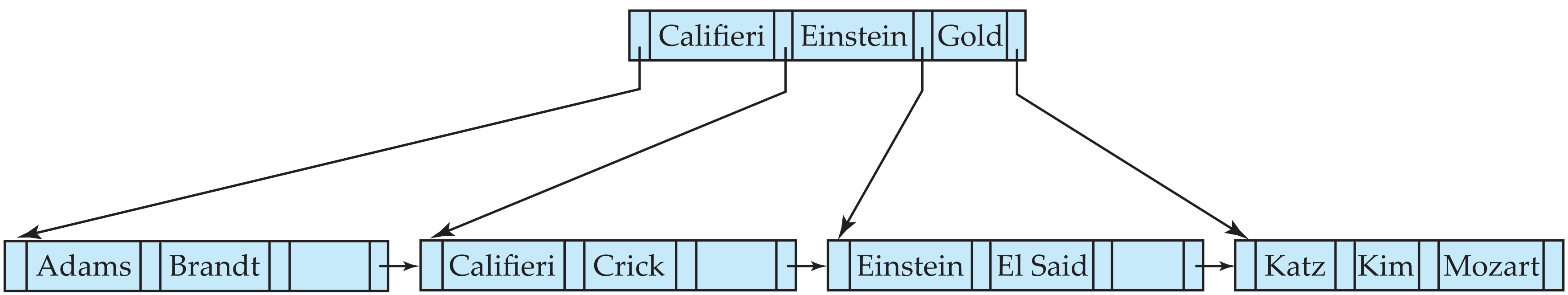
Before and after deleting "Singh and Wu"



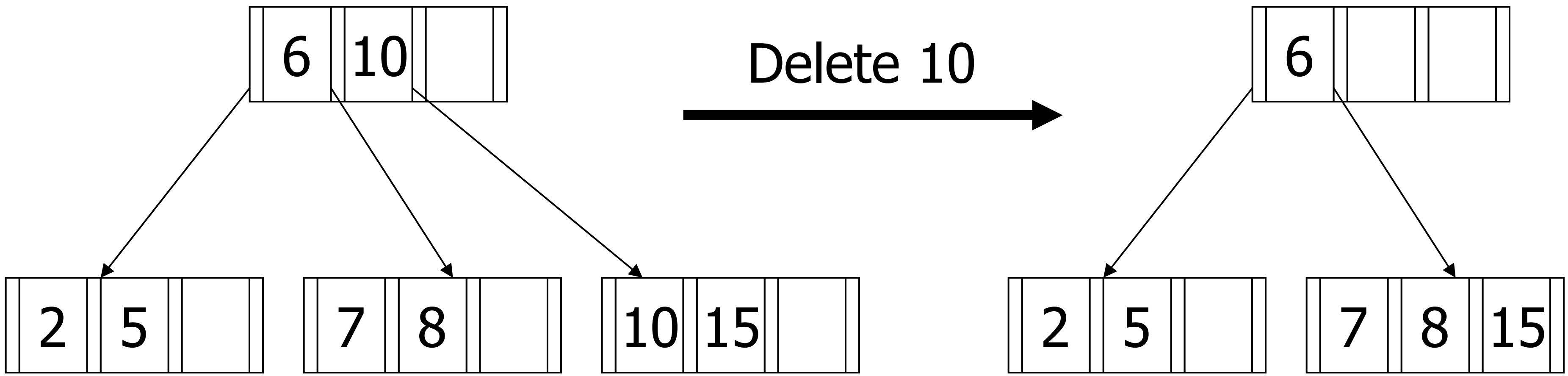
Deletion Example cont’d



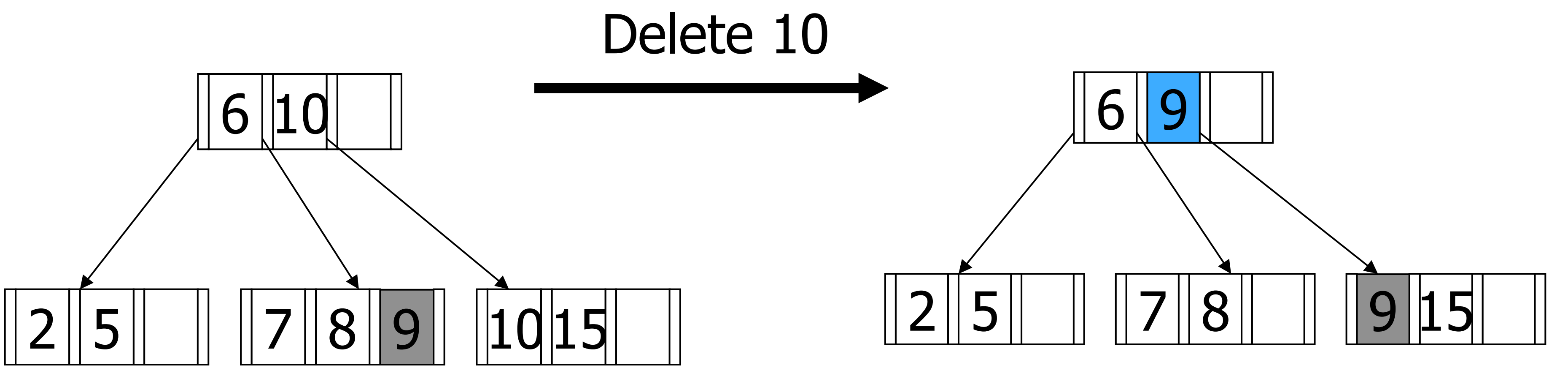
Before and after deleting “Gold”



More Example



Another Example



20/9/14
26

End of Lecture

- Summary
  - B+-Tree Index Files
    - lookup
    - Insertion
    - Deletion
- Reading
  - Database System Concepts, 6th edition, chapter 11.1, 11.2, 11.3
  - Database System Concepts, 7th edition, chapter 14.1, 14.2, 14.3