[DBP 9-1] 27th Apr. 2023

Database Programming

(Database Indexes – Hash, B-Tree & B+Tree)

Spring, 2023

Jaewook Byun

Ph.D., Assistant Professor, Department of Software, Sejong University

jwbyun@sejong.ac.kr

https://sites.google.com/view/jack-dfpl/home

https://www.youtube.com/channel/UC988e-Y8nto0LXVae0aqaOQ

Schedule

Week	Title		
1	Course Overview and Environment Setting	Environment Setting and Revisiting Java	-
2	Introduction to Database		CH1
3	Relational Model	Relational Algebra	CH2
4	Database Language: SQL (DDL, DML, DQL, DCL)		CH3-5
5	Database Language: SQL (DDL, DML, DQL, DCL)		CH3-5
6	Database Language: SQL (DDL, DML, DQL, DCL)		CH3-5
7	Database Language: SQL (DDL, DML, DQL, DCL)		CH3-5
8	Midterm Examination		
9	Physical Database Design: Database Indexes		CH12
10	Physical Database Design: Database Indexes		CH12
11	Conceptual Database Design – E-R Data Model		CH6
12	Logical Database Design 1 – Schema Mapping		CH6
13	Logical Database Design 2 – Normalization		CH7
14	Query Processing and Optimization, or View (TBD)		CH13-14, CH3-5
15	Final Examination		

Subject to change

Calendar

- 수업일수는 요일별 15주 이상이며 수업 결손이 발생하지 않도록 진행

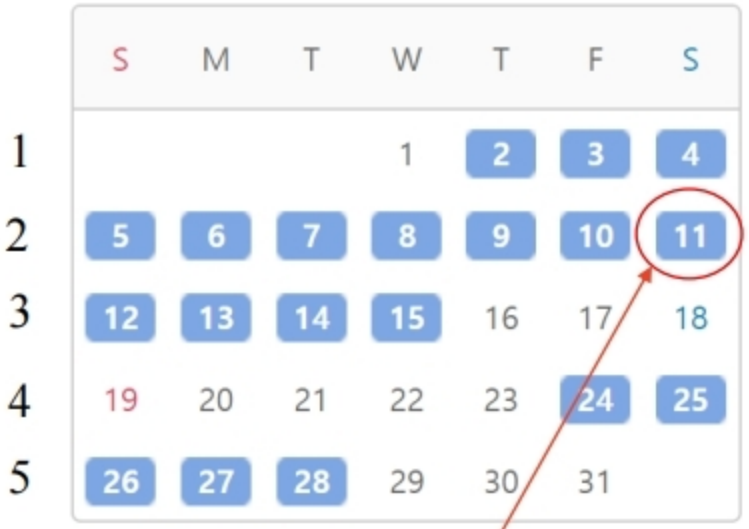
요 일 별 수업일수	월	화	수	목	금
	16일	15일	16일	16일	15일

나. 수업주차 : 한 주차는 목요일부터 수요일까지 임

수업주차	기간	수업주차	기간
1주차	03.02.(목) ~ 03.08.(수)	9주차	04.27.(목) ~ 05.03.(수)
2주차	03.09.(목) ~ 03.15.(수)	10주차	05.04.(목) ~ 05.10.(수)
3주차	03.16.(목) ~ 03.22.(수)	11주차	05.11.(목) ~ 05.17.(수)
4주차	03.23.(목) ~ 03.29.(수)	12주차	05.18.(목) ~ 05.24.(수)
5주차	03.30.(목) ~ 04.05.(수)	13주차	05.25.(목) ~ 05.31.(수)
6주차	04.06.(목) ~ 04.12.(수)	14주차	06.01.(목) ~ 06.07.(수)
7주차	04.13.(목) ~ 04.19.(수)	15주차	06.08.(목) ~ 06.14.(수)
8주차 (중간고사)	04.20.(목) ~ 04.26.(수)	16주차 (기말고사)	06.15.(목) ~ 06.21.(수)

Calendar

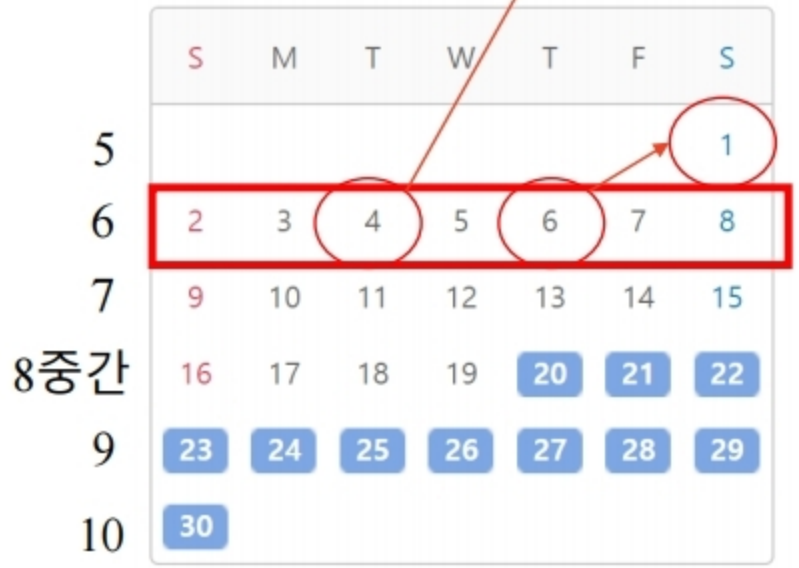
2023년 3월. March



	S	M	T	W	T	F	S
1				1	2	3	4
2	5	6	7	8	9	10	11
3	12	13	14	15	16	17	18
4	19	20	21	22	23	24	25
5	26	27	28	29	30	31	

- 2(목) 1학기 개강
- 3(금) - 8(수) 수강신청 과목 확인 및 변경
- 6(월) - 15(수) 교직신청
- 24(금) - 28(화) 수강신청과목 철회

2023년 4월. April



	S	M	T	W	T	F	S
5							1
6	2	3	4	5	6	7	8
7	9	10	11	12	13	14	15
8중간	16	17	18	19	20	21	22
9	23	24	25	26	27	28	29
10	30						

- 20(목) - 26(수) 1학기 중간고사
- 27(목) - 5.1(월) 1학기 중간고사 성적 입력

IEEE ICDE 2023, Anaheim, California, US

Calendar

2023년 5월. May

	S	M	T	W	T	F	S
10		1	2	3	4	5	6
11	7	8	9	10	11	12	13
12	14	15	16	17	18	19	20
13	21	22	23	24	25	26	27
14	28	29	30	31			

- 2(화) - 7(일) 1학기 중간고사 성적 열람 및 정정
- 4(목) - 30(화) 복수·부전공, 연계융합전공 신청
- 5(금) 창립 83주년 기념휴일 (창립일 : 1940. 5. 20)
- 29(월) - 31(수) 하계 계절학기 수강신청

Calendar

2023년 6월. June

	S	M	T	W	T	F	S
14					1	2	3
	4	5	6	7	8	9	10
15기말	11	12	13	14	15	16	17
	18	19	20	21	22	23	24
	25	26	27	28	29	30	

▪ 9(금) - 26(월)	1학기 강의평가
▪ 15(목) - 21(수)	1학기 기말고사 및 수업결손 보충
▪ 22(목) - 26(월)	1학기 기말고사 성적 입력
▪ 22(목)	하계방학 시작 및 계절학기 개강
▪ 27(화) - 7.3(월)	1학기 기말고사 성적 열람 및 정정

Calendar

2023년 7월. July

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

- 4(화) - 5(수)
- 24(월) - 30(일)

1학기 기말고사 성적마감

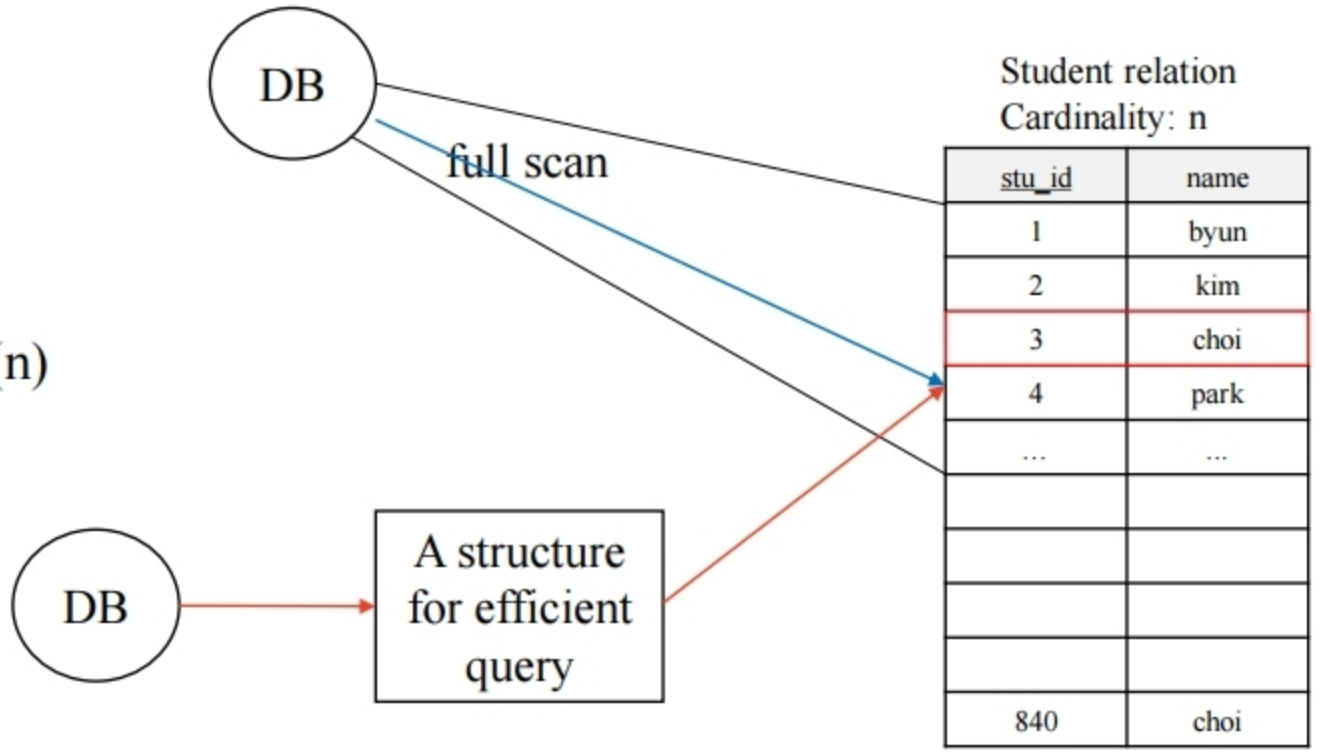
2학기 복학, 휴학 신청

Table of contents

Introduction

- Motivation
  - Get the record (row) that stu_id is 4
    - Naïve approach
      - Worst-case time complexity: O(n)

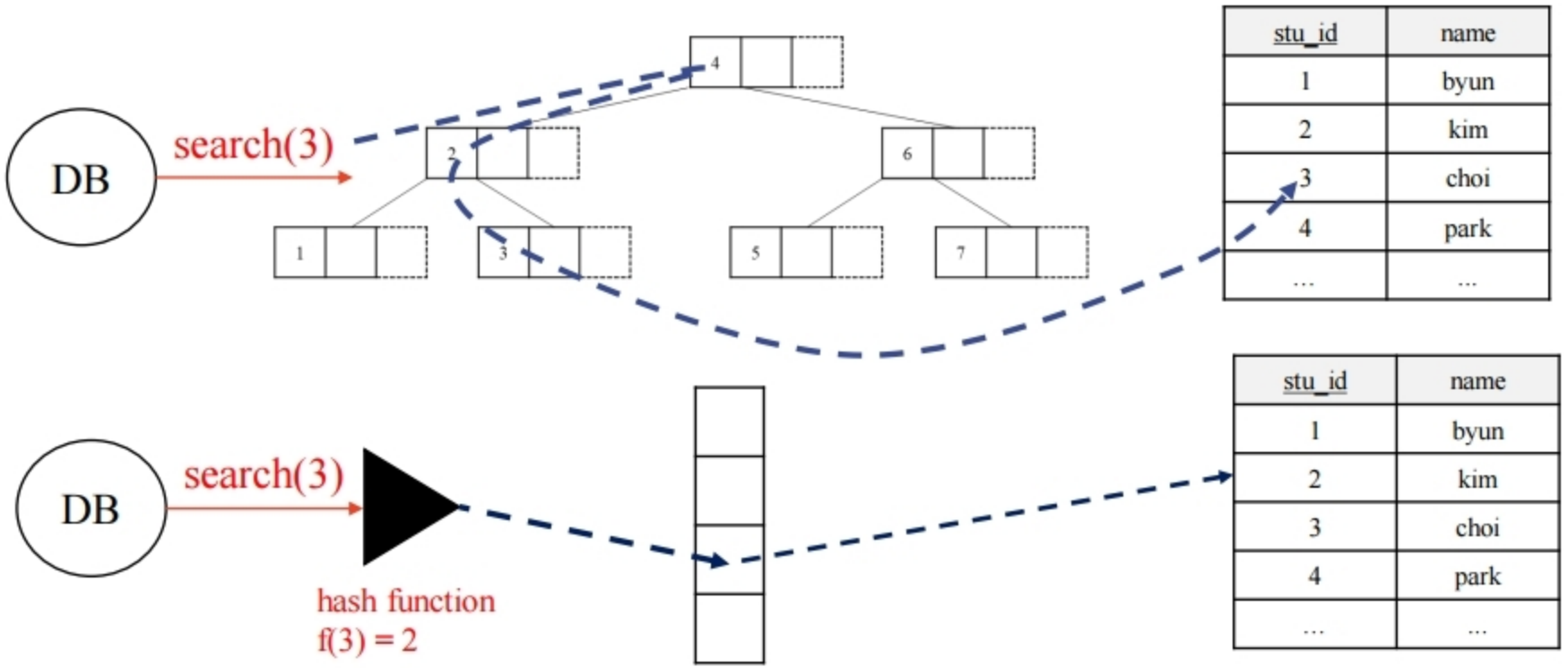
SELECT * FROM student WHERE name = ‘choi’;



  - Smarter approach
    - better than O(n)

Introduction

- Index
  - An efficient way to access records in a file
  - A pair of key and its pointer to the record
- Type
  - Hash index: using a hashing function (e.g., Hash)
  - Ordered index: consider the order of keys (e.g., B-Tree)

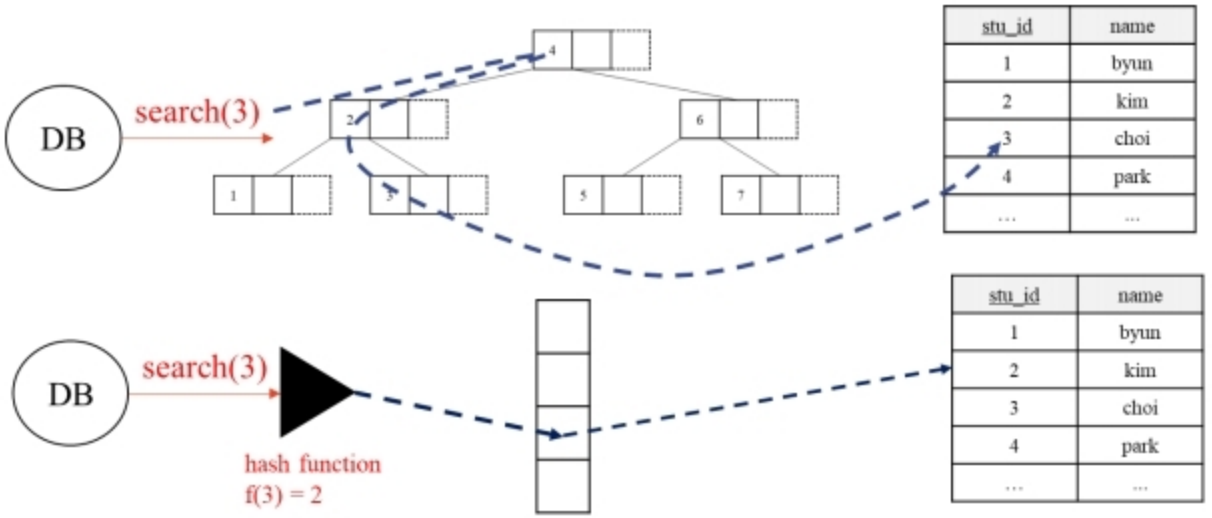


Introduction

- Criteria to select the type of index
  - Access type (in terms of advantages)
    - e.g., for ceiling(key): the least value greater than or equal to x
      - B-Tree → $O(\log N)$
      - Hash → O(n)
    - e.g., for find(key)
      - B-Tree → $O(\log N)$
      - Hash → O(1)
  - Insertion time (in terms of disadvantages)
  - Deletion time (in terms of disadvantages)
  - Space overhead (in terms of disadvantages)

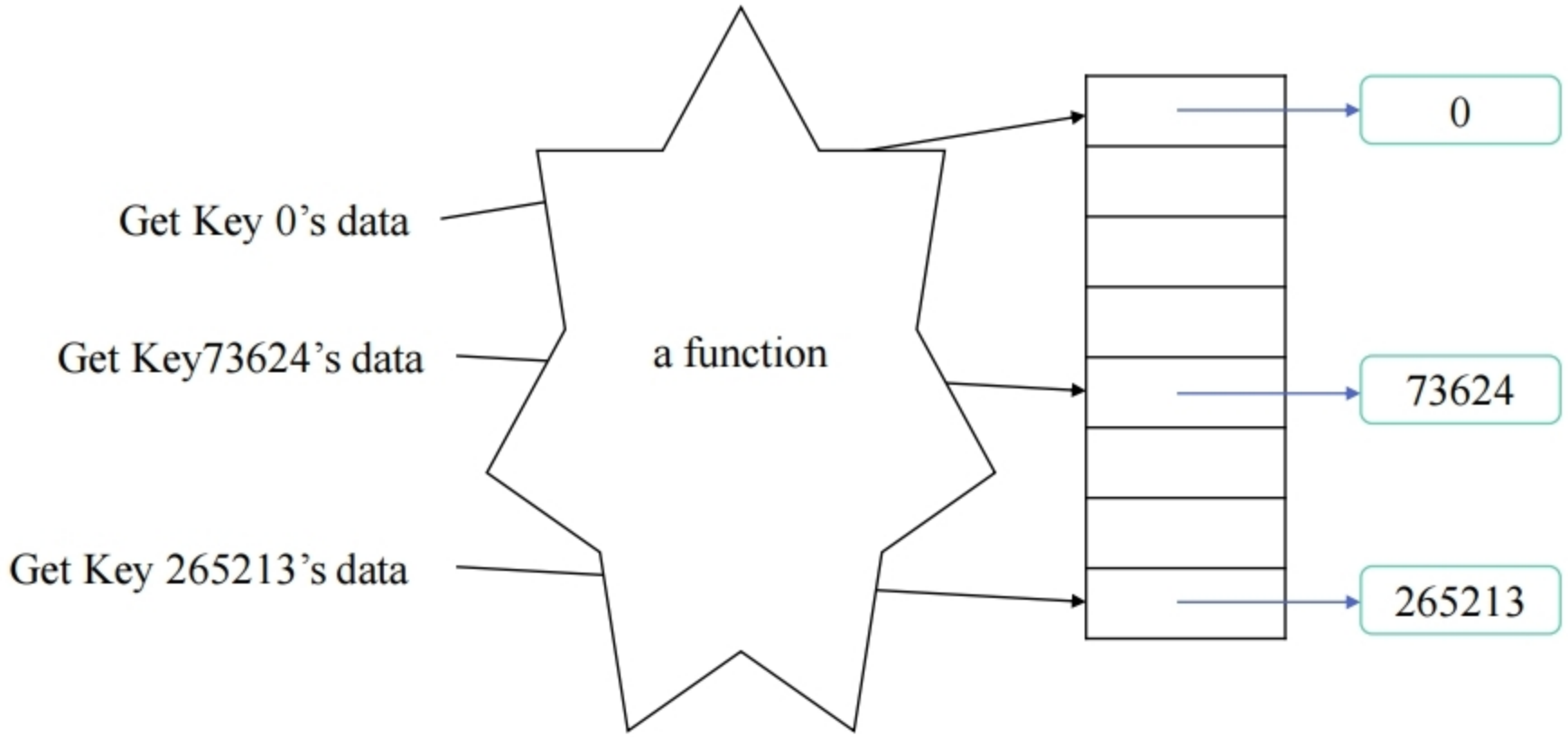
	B-Tree		Hash	
	Average	Worst Case	Average	Worst Case
Search	$O(\log N)$	$O(\log N)$	O(1)	O(n)
Insert	$O(\log N)$	$O(\log N)$	O(1)	O(n)
Delete	$O(\log N)$	$O(\log N)$	O(1)	O(n)
Space	$O(N)$	$O(N)$	O(n)	O(n)

[Wikipedia]



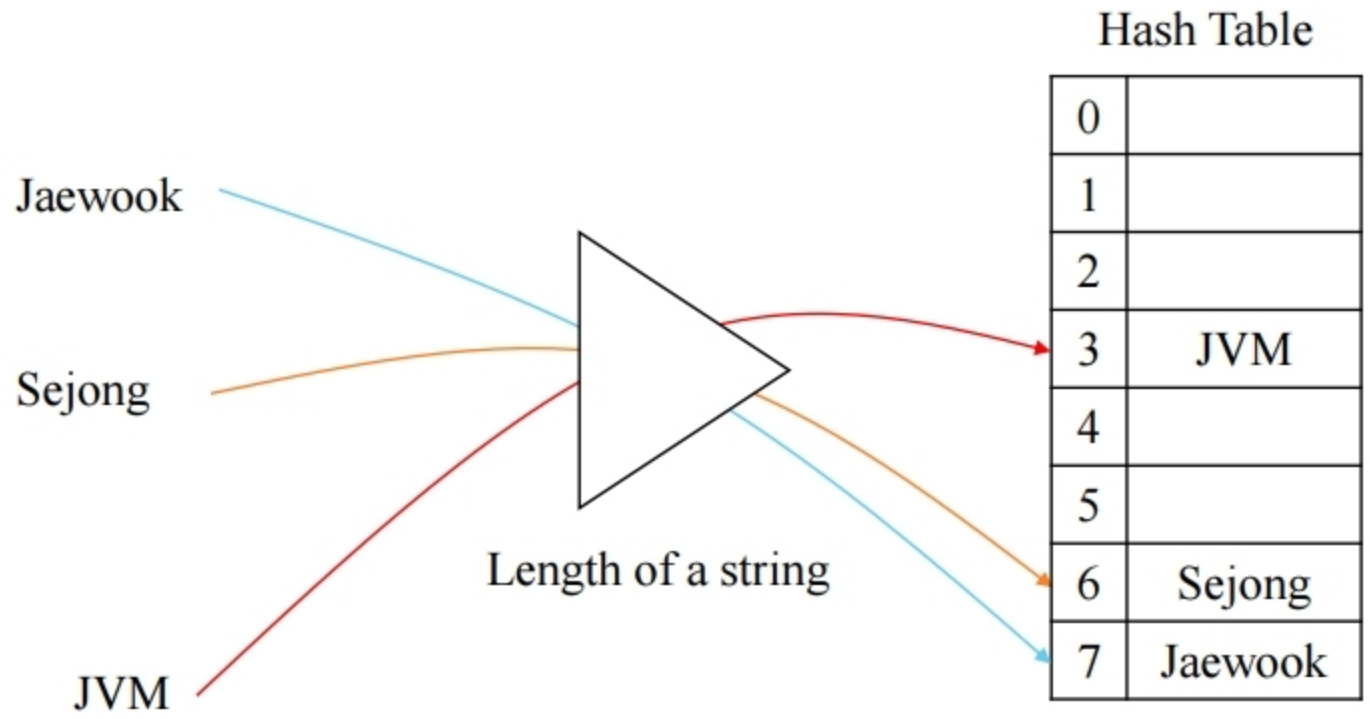
Hash

- Motivation
  - Find the relevant data by using a function to map between a key to an index of a data structure



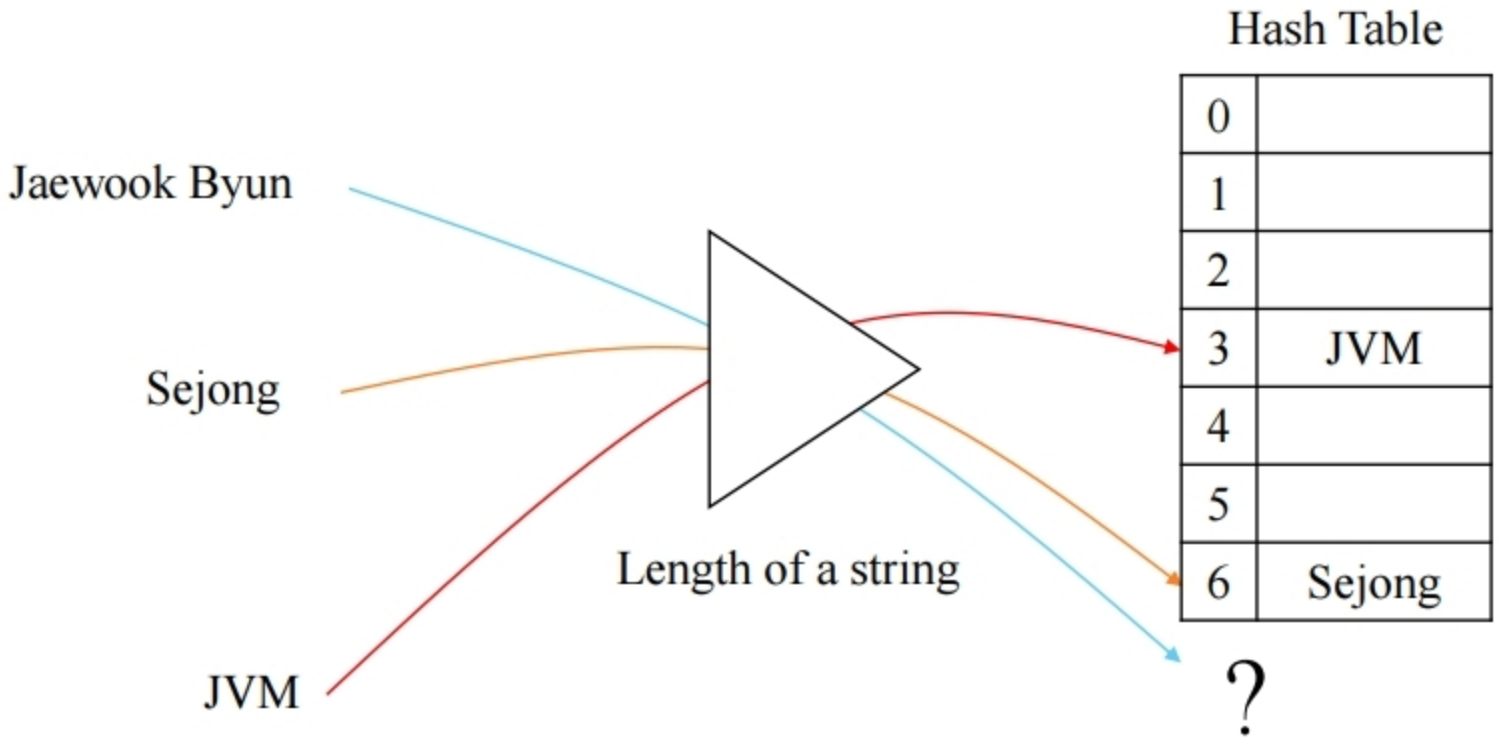
Hash

- Hash Table
  - Data is stored in a data structure called Hash Table
  - Hash is a function to map a key to an index to its information
    - e.g., Length of a string



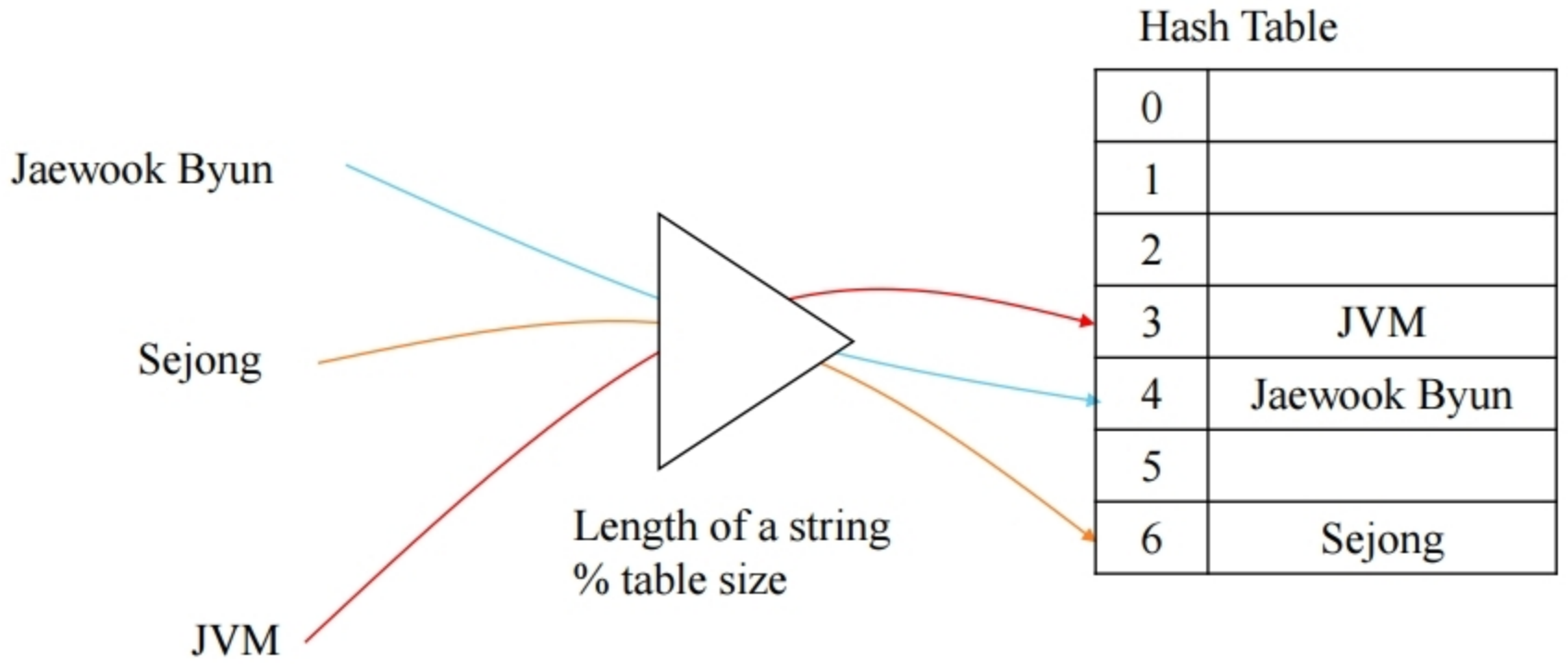
Hash

- Hash Table
  - Consideration 1
    - what if the result of the function yields index out-of-bound



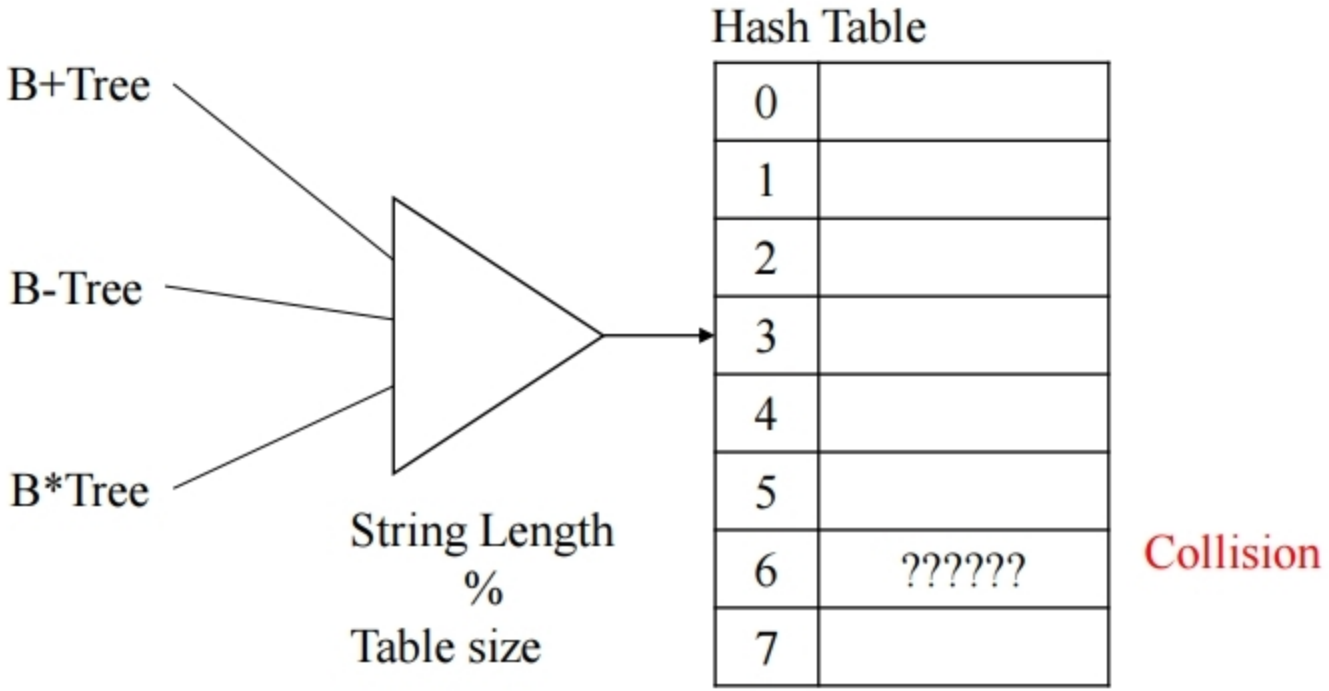
Hash

- Hash Table
  - Consideration 1
    - what if the result of the function yields index out-of-bound
    - → e.g., modular



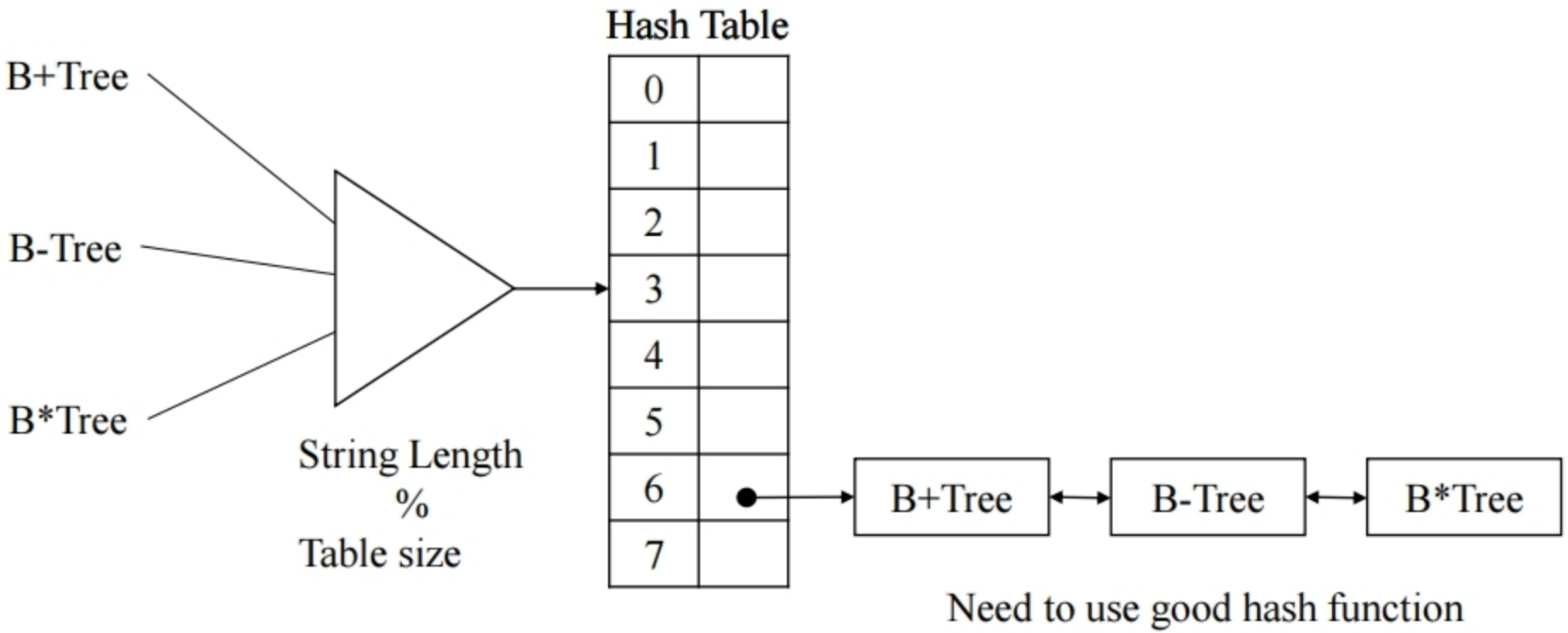
Hash

- Hash Table
  - Consideration 2
    - what if the result of the function conflicts for different keys



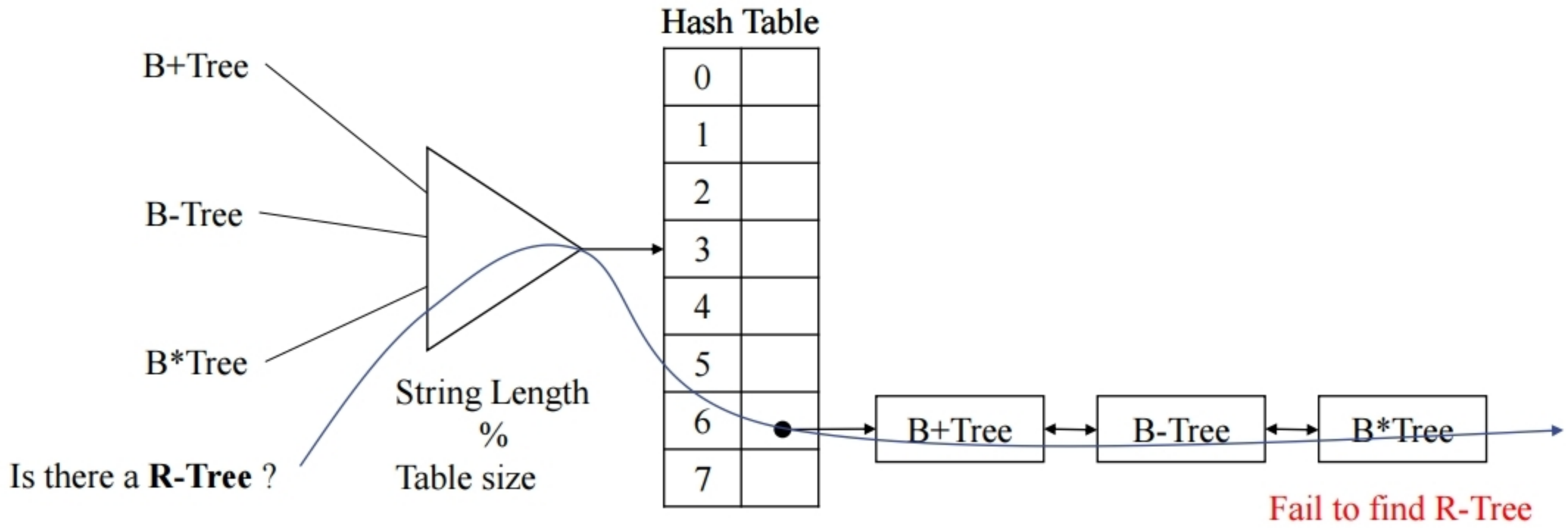
Hash

- Hash Table
  - Consideration 2
    - what if the result of the function conflicts for different keys
    - → Close Addressing vs. Open Addressing



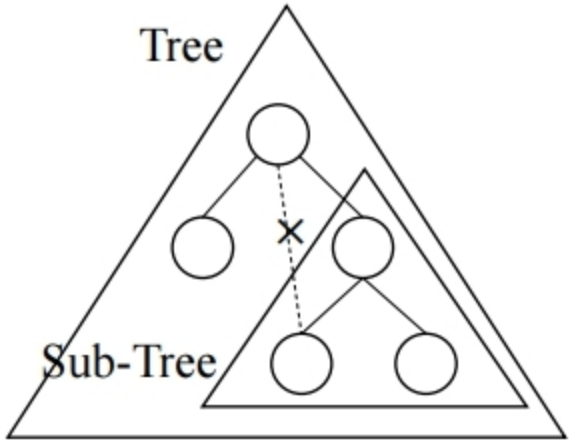
Hash

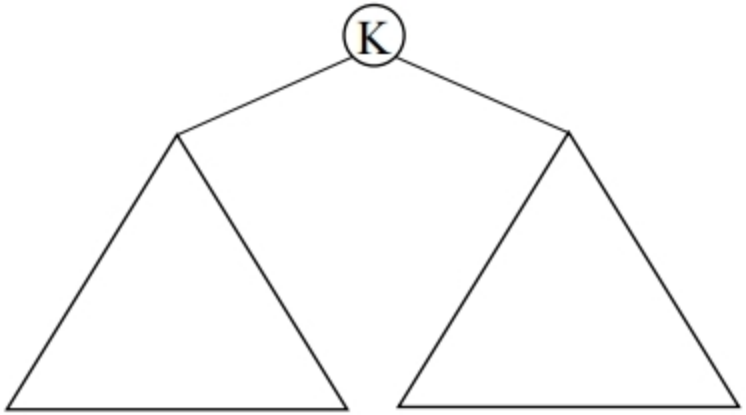
- Hash Table
  - Consideration 2
    - what if the result of the function conflicts for different keys
    - → Close Addressing vs. Open Addressing



Binary Search Tree

- Tree
  - A root as a parent could have children
  - A root has no parent
  - Each child as a root node of a sub-tree could have children
  - Each child has only one parent
- **M**-way tree
  - A tree that the maximum number of children is M
- Binary Search Tree (BST)
  - 2-way tree: a node could have left child node and right child node
  - B-Tree: generalization of BST in which a node can have more than one key & more than 2 children
  - Each node is an entry <key, a pointer to the record>
  - For each node n <k,p>,
    - all the keys in left sub-tree is less than k
    - all the keys in right sub-tree is larger than k

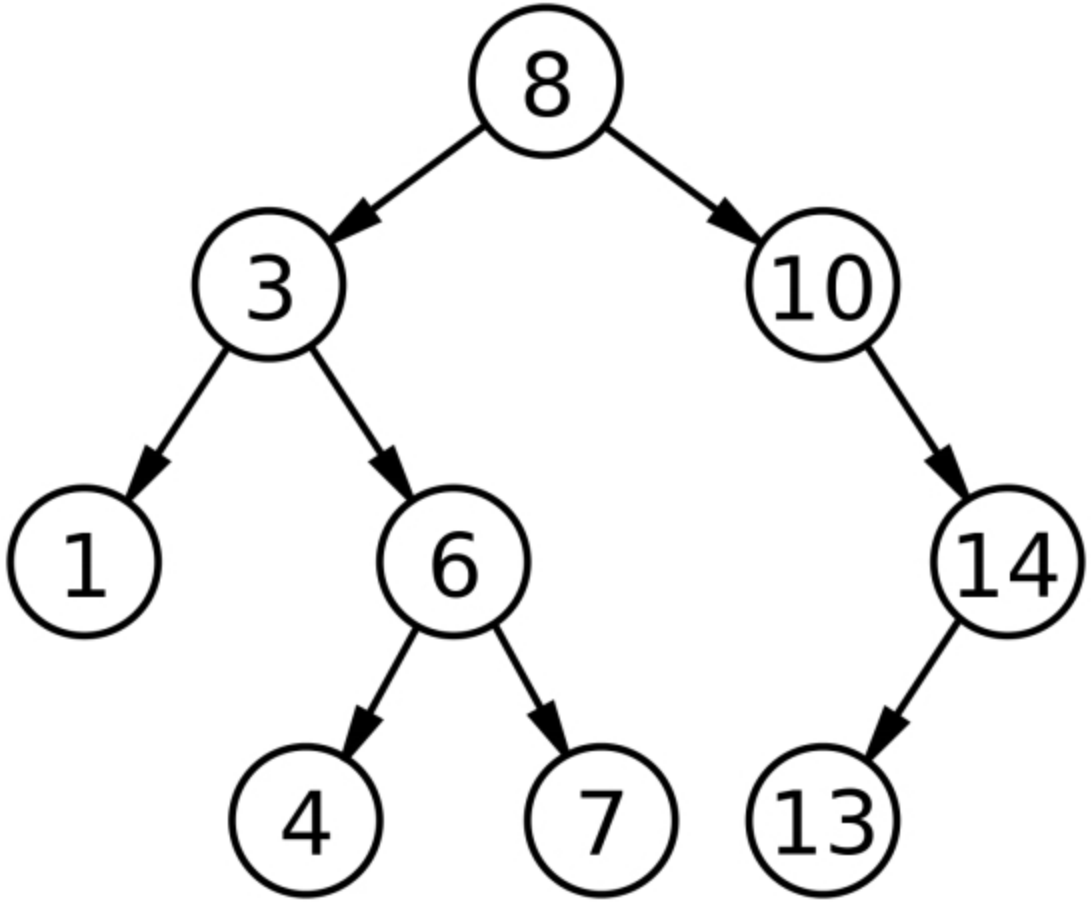




Binary Search Tree

- Binary Search Tree (BST)
  - Search
  - Insert
  - Delete
  - Space

	BST
Search	$O(h)$
Insert	$O(h)$
Delete	$O(h)$
Space	$O(n)$

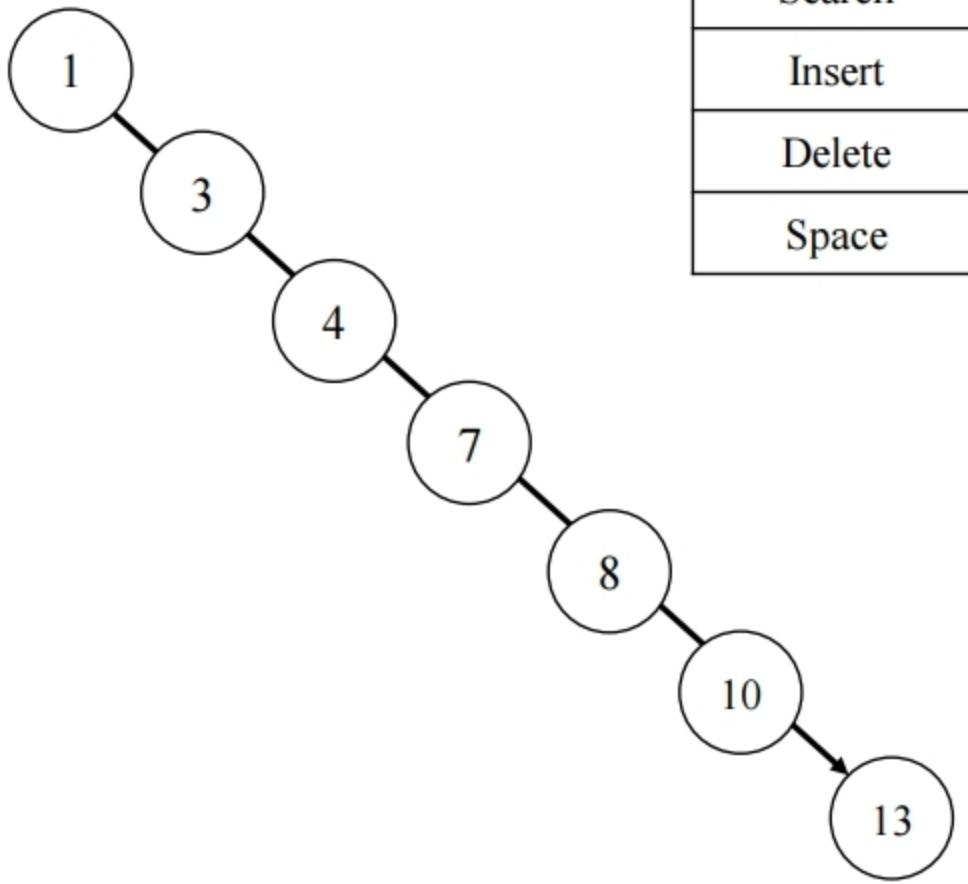


Binary Search Tree

- Binary Search Tree (BST)
  - Search
  - Insert
  - Delete
  - Space

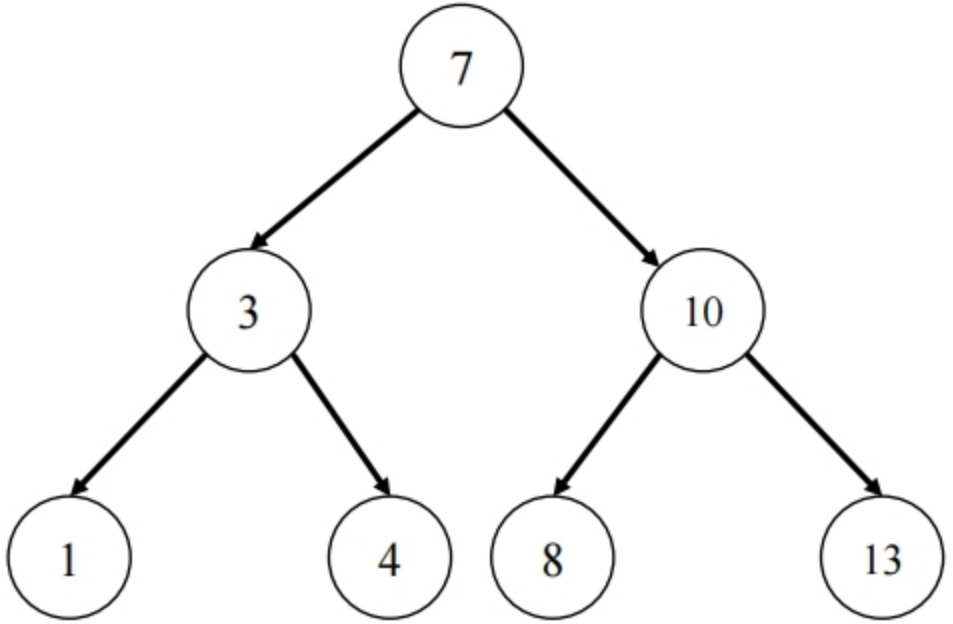
- In a worse case, $h = n$
  - thus,
    - Search: $O(n)$
    - Insert: $O(n)$
    - Delete: $O(n)$
    - Space: $O(n)$

	BST
Search	$O(h)$
Insert	$O(h)$
Delete	$O(h)$
Space	$O(n)$



Binary Search Tree

- In a good case (full binary tree or complete binary tree), $h = \log n$
  - thus,
    - Search: $O(\log n)$
    - Insert:$O(\log n)$
    - Delete:$O(\log n)$
    - Space: $O(n)$

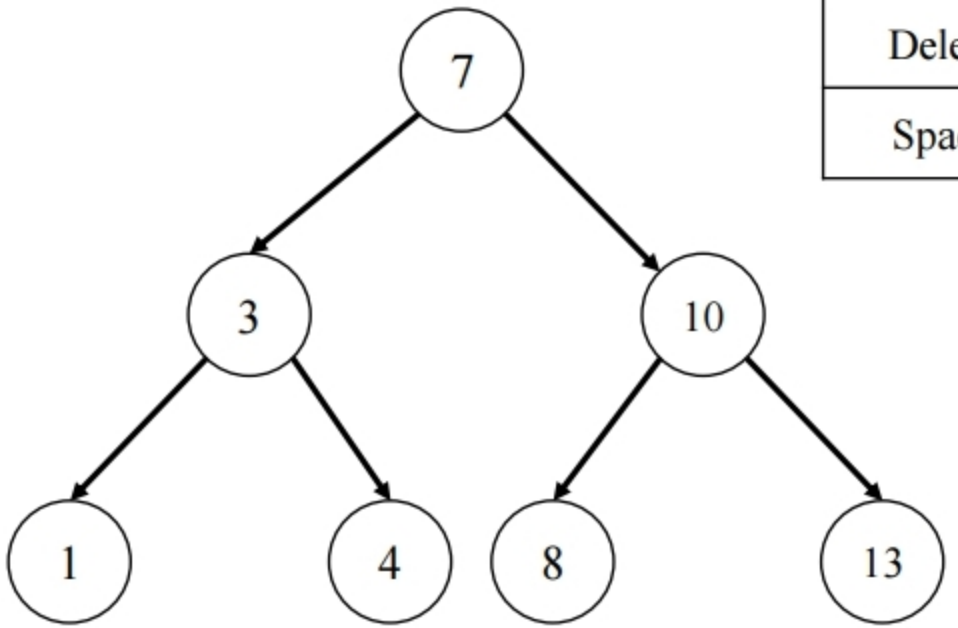


	BST
Search	$O(h)$
Insert	$O(h)$
Delete	$O(h)$
Space	$O(n)$

Binary Search Tree

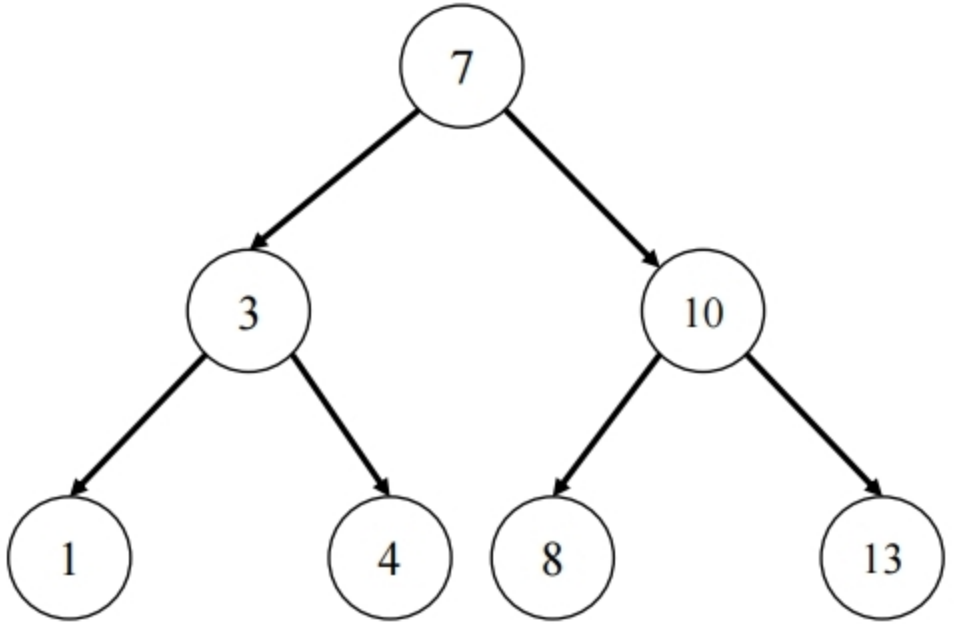
- The maximum number of nodes for a height h
  (when a height is the maximum number of edges from a leaf node to the root node)
  - $n =$

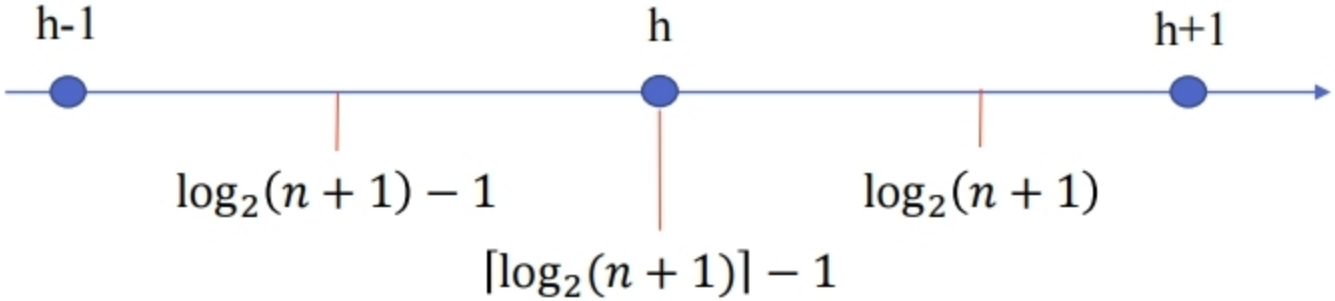
	BST
Search	$O(h)$
Insert	$O(h)$
Delete	$O(h)$
Space	$O(n)$



Binary Search Tree

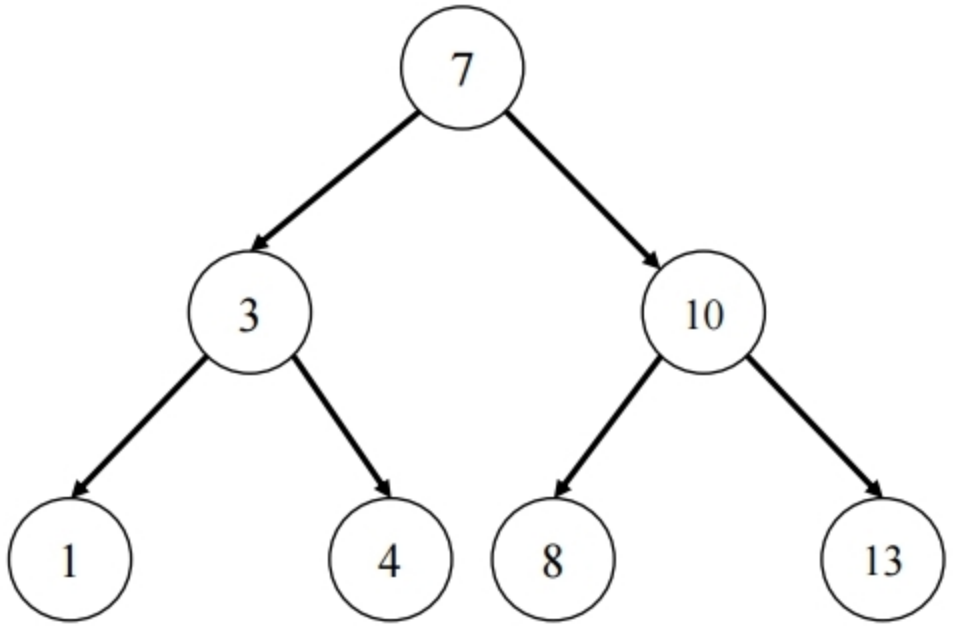
- The minimum height of binary search tree
  - $2^h - 1 < n \leq 2^{h+1} - 1$
  1. $2^h - 1 < n$
  2. $n \leq 2^{h+1} - 1$
  - $\log_2(n+1) - 1 \leq h < \log_2(n+1)$
    -





Binary Search Tree

- Balanced Tree
  - AVL
  - Red-black Tree
  - **B-Tree**
  - **B+-Tree**



	BST
Search	$O(h)$
Insert	$O(h)$
Delete	$O(h)$
Space	$O(n)$

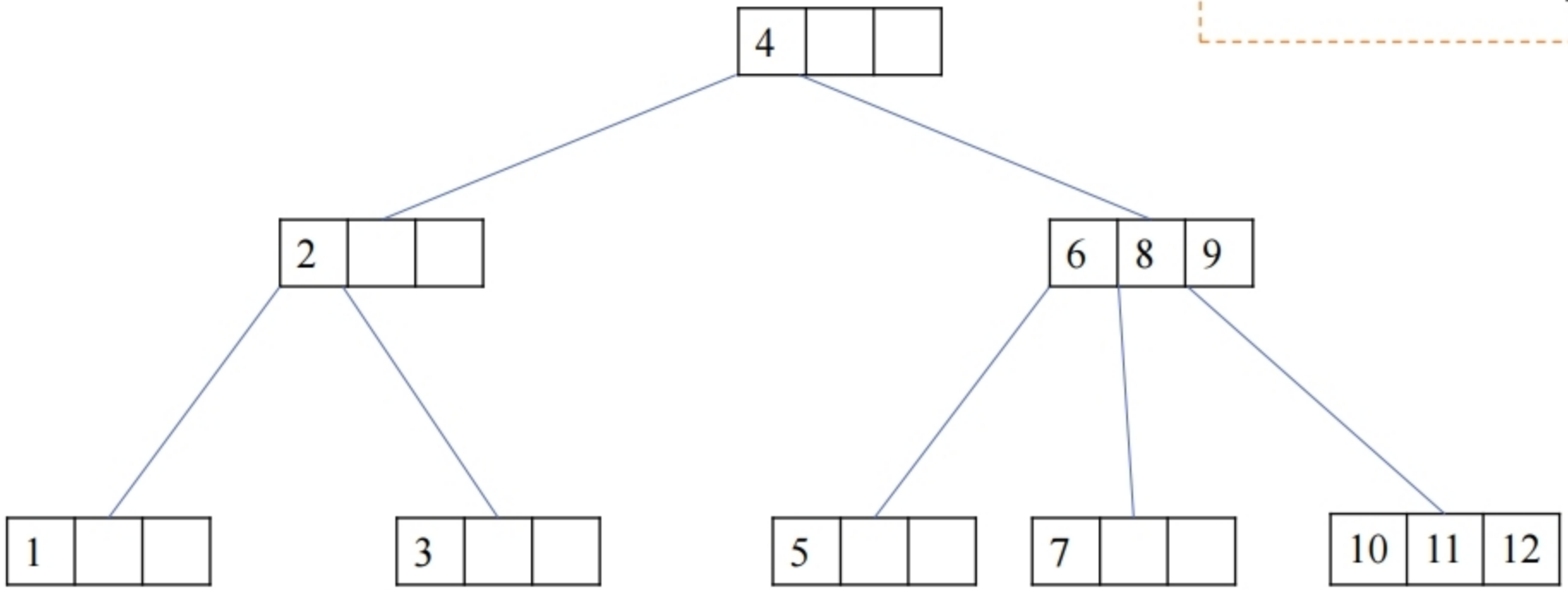
B-Tree

- balanced m-way tree
- Generalization of BST in which a node can have more than one key & more than 2 children
- Maintains sorted data
- All leaf node must be at same level
- B-tree of order m has following properties:
  - every node can have max m children
  - min children: leaf → 0
    (non-leaf) root → 2
    internal nodes → $\left\lceil \frac{m}{2} \right\rceil$
  - every node has max (m-1) keys
  - min keys: root node → 1
    all other nodes → $\left\lceil \frac{m}{2} \right\rceil - 1$

B-Tree

- 4-way tree

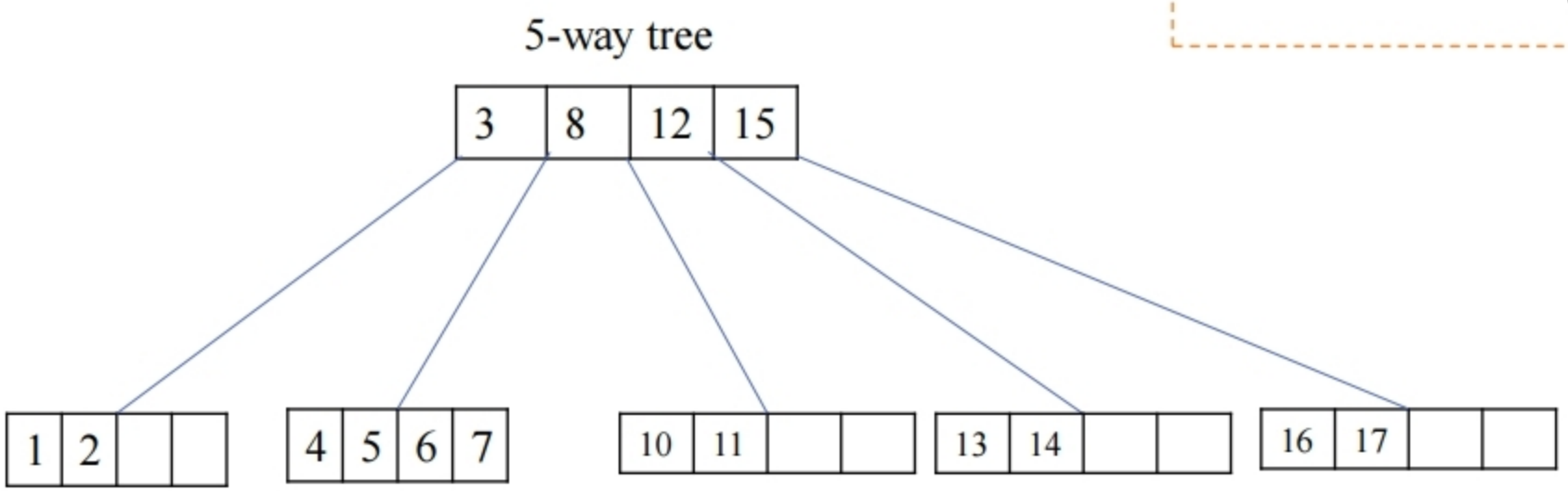
- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root
    $\left\lceil \frac{m}{2} \right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil \frac{m}{2} \right\rceil - 1$ for the others



B-Tree

- search(Btree tree, Key *k*)
  - Like BST

- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root
    $\lceil \frac{m}{2} \rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\lceil \frac{m}{2} \rceil - 1$ for the others



B-Tree

- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root
    $\left\lceil \frac{m}{2} \right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil \frac{m}{2} \right\rceil - 1$ for the others

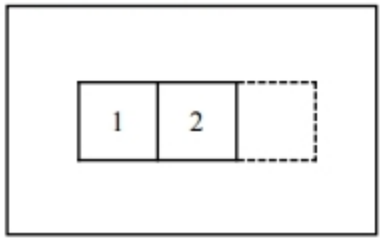
- insert(Btree tree, Key *k*)
  1. Search a leaf node, **T**, that *k* to be inserted
  2. If |T| <= m-1, |T|=the number of keys in **T,**
     (i.e., do not violate the max keys prop.), then DONE
  3. else
     - the leaf node splits into three parts (0,1,…,m-1)
       - Left: 0 to $\left\lfloor \frac{m-1}{2} \right\rfloor$-1 Middle: $\left\lfloor \frac{m-1}{2} \right\rfloor$
       - Right: $\left\lfloor \frac{m-1}{2} \right\rfloor + 1$ to m-1
     - Middle goes to parent node
     - Left, Right become a left child and a right child of Middle, respectively
     - **T** = A node contains Middle
     - Go to 2

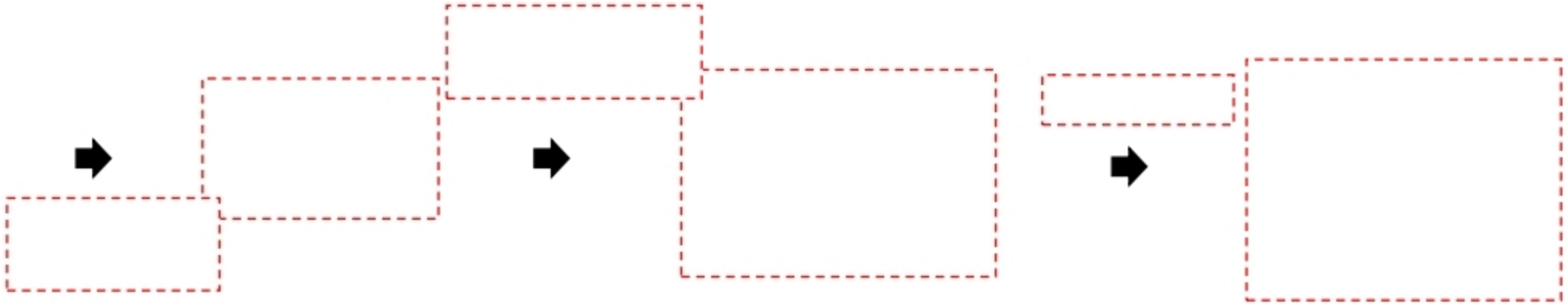
B-Tree

- insert(Btree tree, Key *k*)
  1. Search a leaf node, **T**, that *k* to be inserted
  2. If |T| <= m-1, |T|=the number of keys in **T**,
     (i.e., do not violate the max keys prop.), then DONE
  3. else
     - the leaf node splits into three parts (0,1,…,m-1)
       - Left: 0 to $\left\lfloor \frac{m-1}{2} \right\rfloor$-1 Middle: $\left\lfloor \frac{m-1}{2} \right\rfloor$
       - Right: $\left\lfloor \frac{m-1}{2} \right\rfloor + 1$ to m-1
     - Middle goes to parent node
     - Left, Right become a left child and a right child of Middle, respectively
     - **T** = A node contains Middle
     - Go to 2

- Example (3-way)
  - insert(tree, 3)

Btree tree



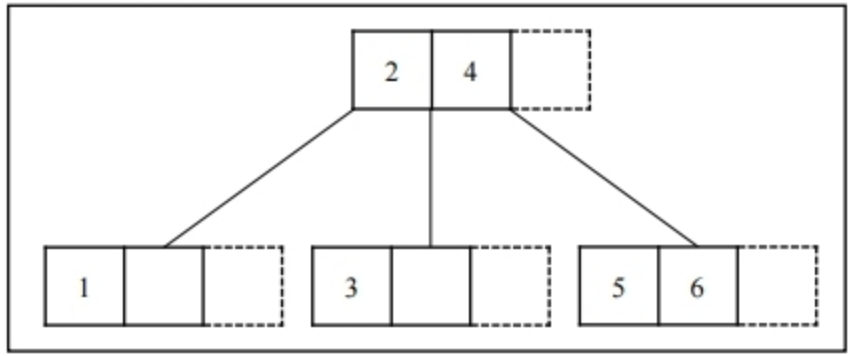


- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root
    $\left\lceil \frac{m}{2} \right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil \frac{m}{2} \right\rceil - 1$ for the others

B-Tree

- insert(Btree tree, Key *k*)
- Example (3-way)
  - insert(tree, 7)

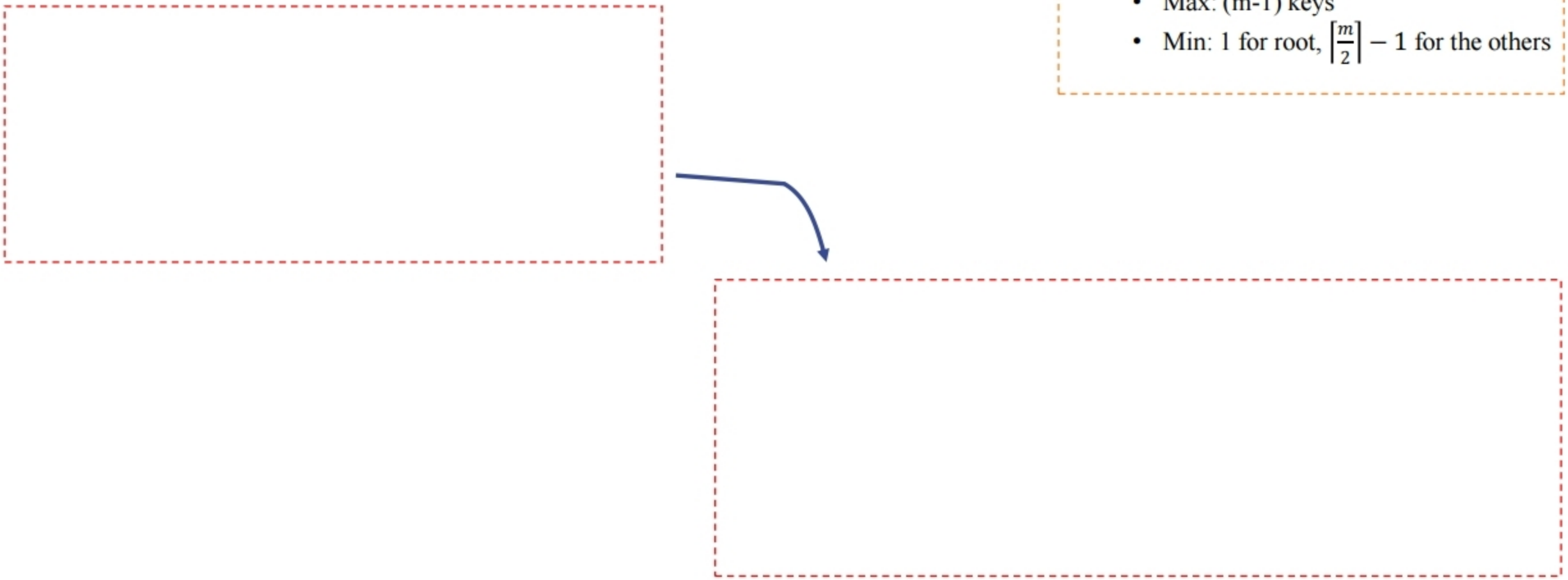
Btree tree



B-Tree

- insert(Btree tree, Key *k*)
- Example (3-way)
  - insert(tree, 7)

- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root
    $\left\lceil \frac{m}{2} \right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil \frac{m}{2} \right\rceil - 1$ for the others

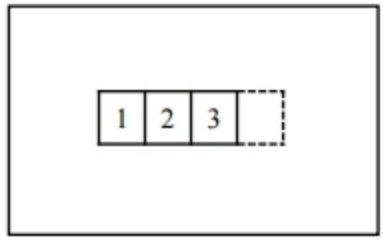


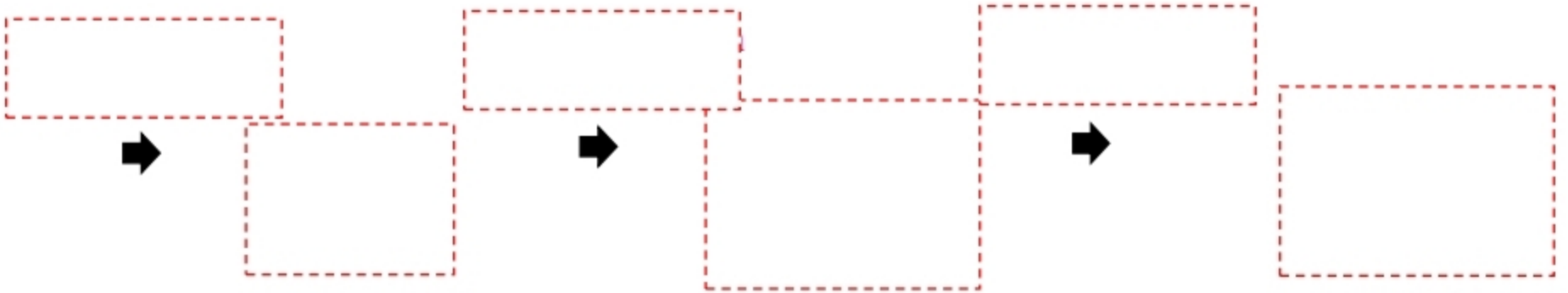
B-Tree

- insert(Btree tree, Key *k*)
  1. Search a leaf node, **T**, that *k* to be inserted
  2. If |T| <= m-1, |T|=the number of keys in **T,**
     (i.e., do not violate the max keys prop.), then DONE
  3. else
     - the leaf node splits into three parts (0,1,…,m-1)
       - Left: 0 to $\left\lfloor \frac{m-1}{2} \right\rfloor$-1 Middle: $\left\lfloor \frac{m-1}{2} \right\rfloor$
       - Right: $\left\lfloor \frac{m-1}{2} \right\rfloor + 1$ to m-1
     - Middle goes to parent node
     - Left, Right become a left child and a right child of Middle, respectively
     - **T** = A node contains Middle
     - Go to 2
- Example (4-way)
  - insert(tree, 4)

- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root
    $\left\lceil \frac{m}{2} \right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil \frac{m}{2} \right\rceil - 1$ for the others

Btree tree

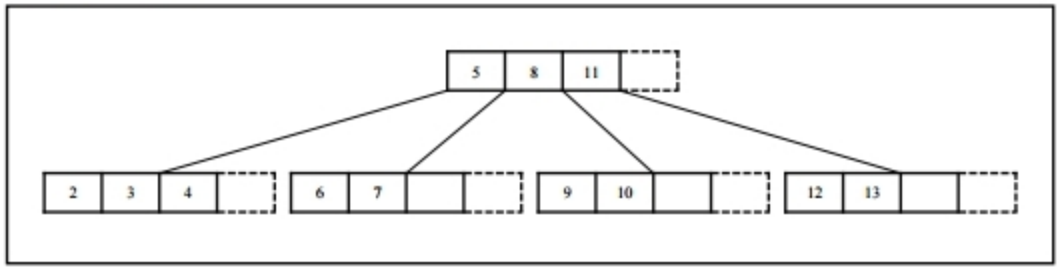




B-Tree

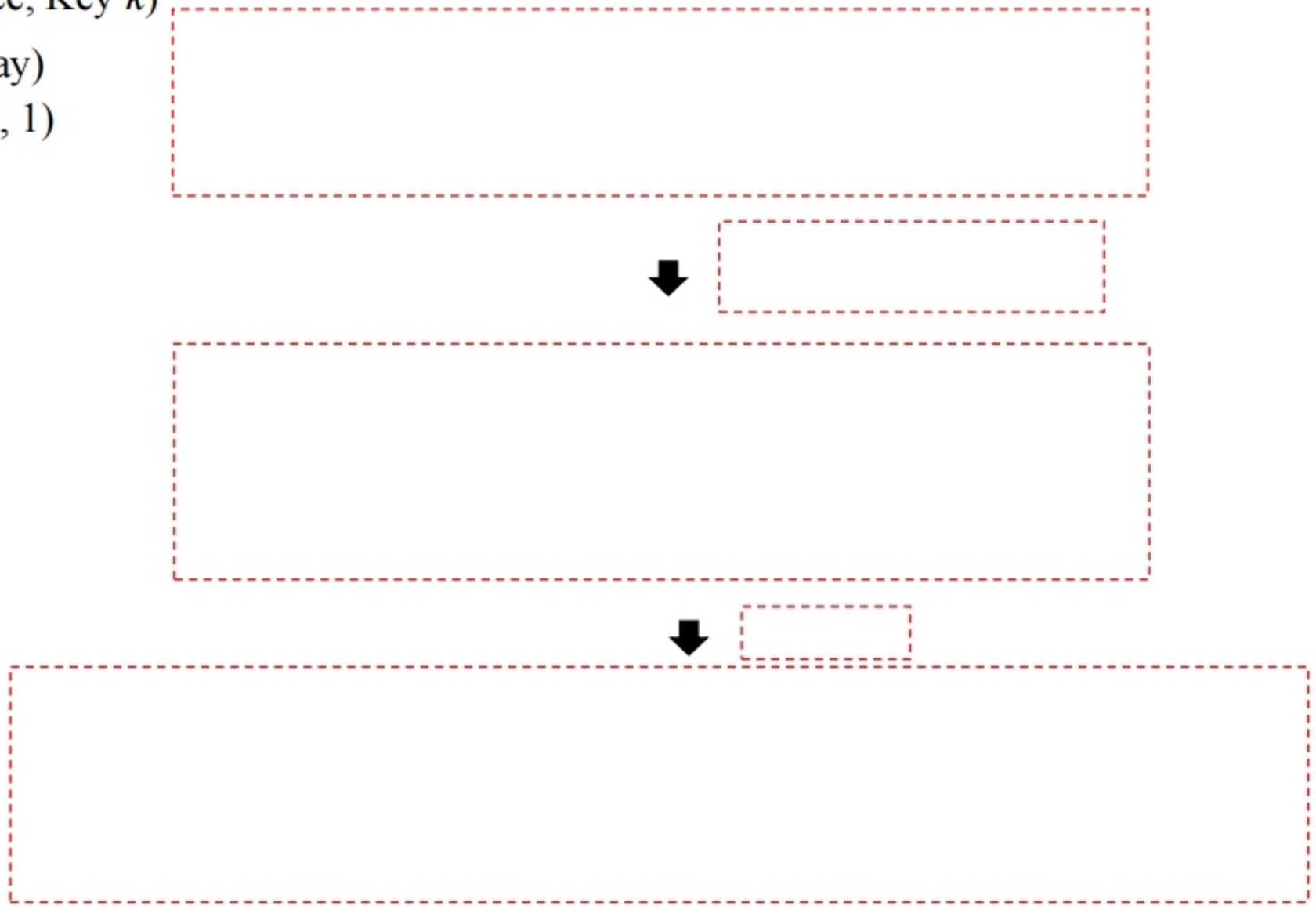
- insert(Btree tree, Key *k*)
- Example (4-way)
  - insert(tree, 1)

Btree tree



B-Tree

- insert(Btree tree, Key *k*)
- Example (4-way)
  - insert(tree, 1)



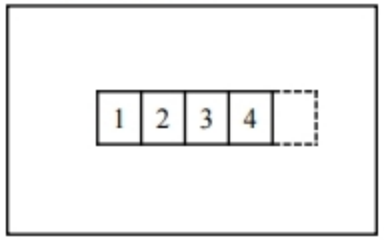
B-Tree

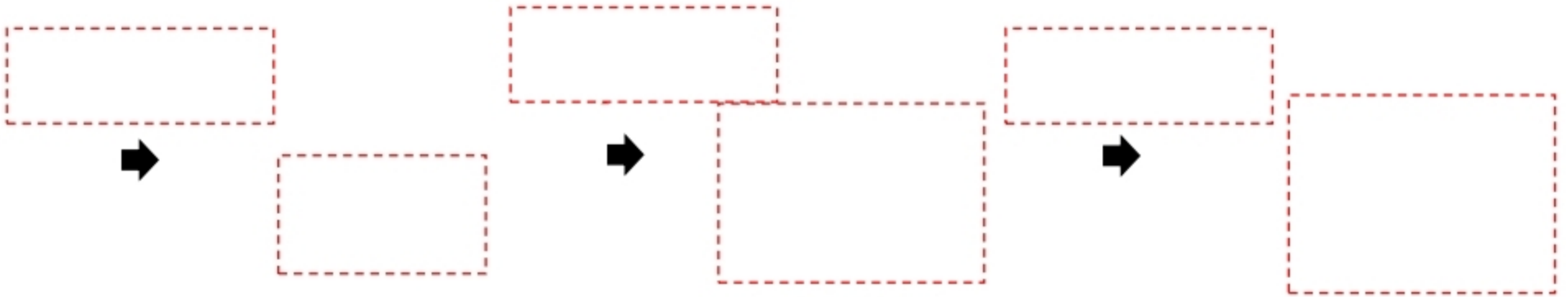
- insert(Btree tree, Key *k*)
  1. Search a leaf node, **T**, that *k* to be inserted
  2. If |T| <= m-1, |T|=the number of keys in **T,**
     (i.e., do not violate the max keys prop.), then DONE
  3. else
     - the leaf node splits into three parts (0,1,…,m-1)
       - Left: 0 to $\left\lfloor \frac{m-1}{2} \right\rfloor$-1 Middle: $\left\lfloor \frac{m-1}{2} \right\rfloor$
       - Right: $\left\lfloor \frac{m-1}{2} \right\rfloor + 1$ to m-1
     - Middle goes to parent node
     - Left, Right become a left child and a right child of Middle, respectively
     - **T** = A node contains Middle
     - Go to 2

- Example (5-way)
  - insert(tree, 5)

- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root
    $\left\lceil \frac{m}{2} \right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil \frac{m}{2} \right\rceil - 1$ for the others

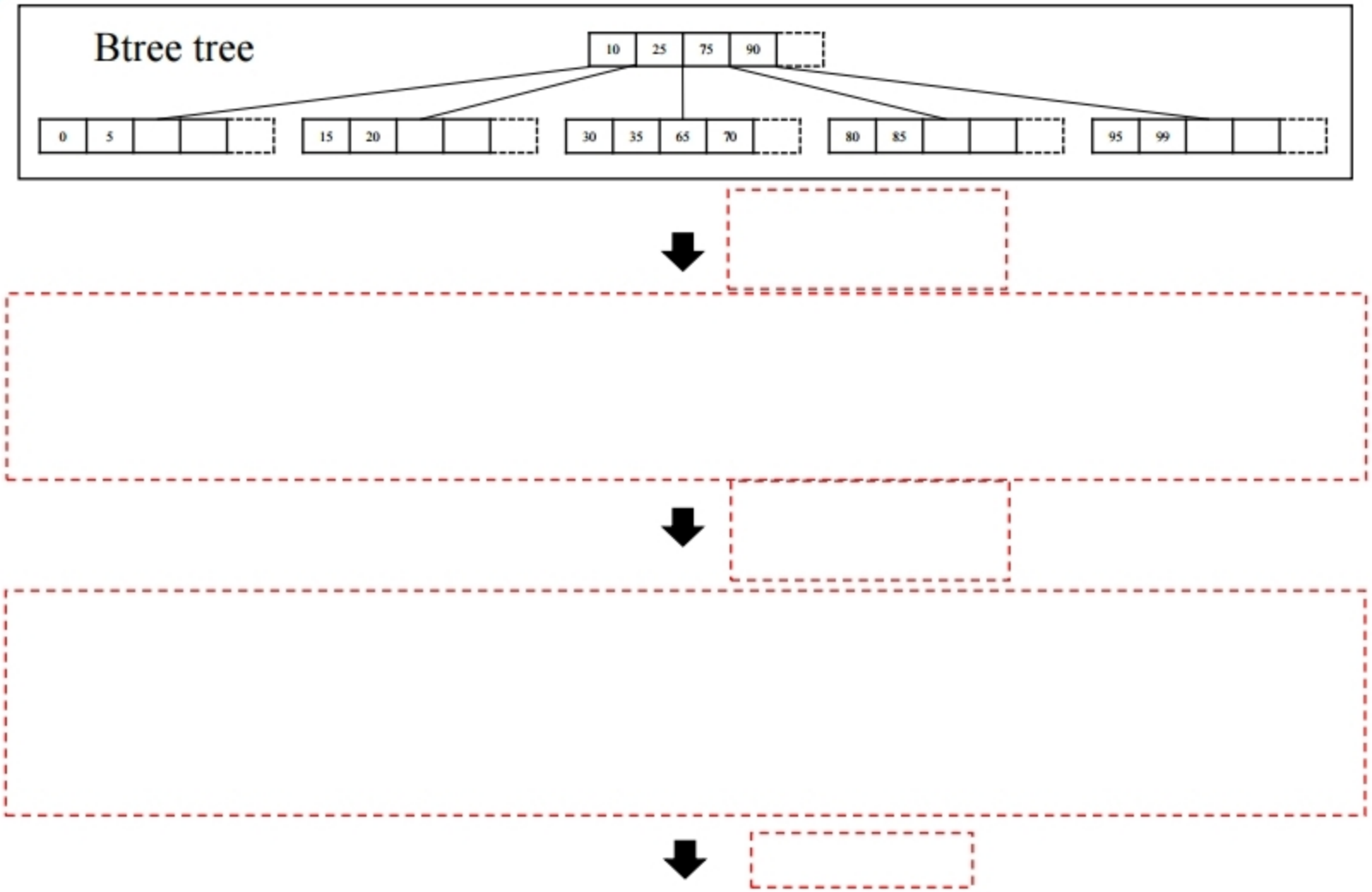
Btree tree





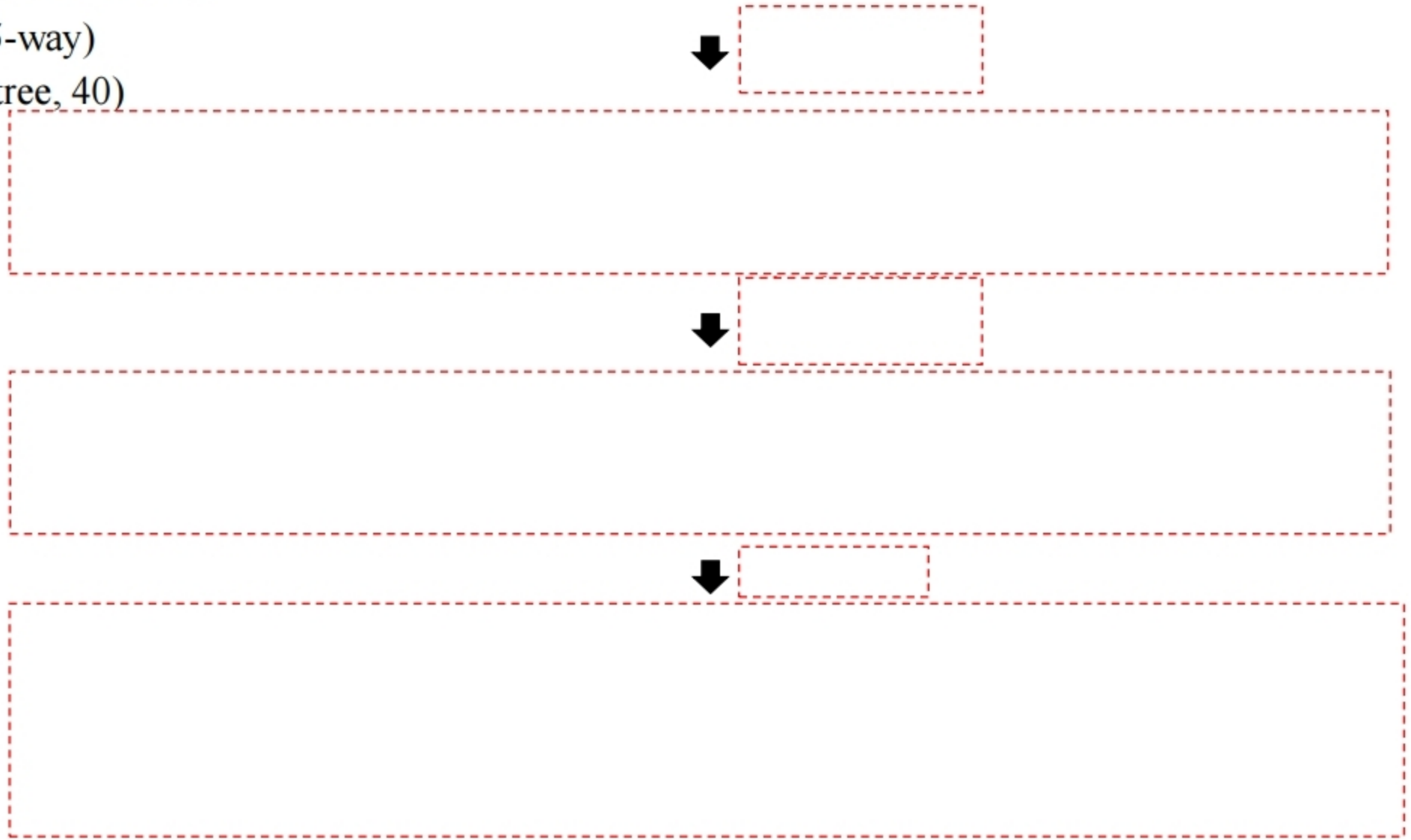
B-Tree

- insert(Btree tree, Key *k*)
- Example (5-way)
  - insert(tree, 40)



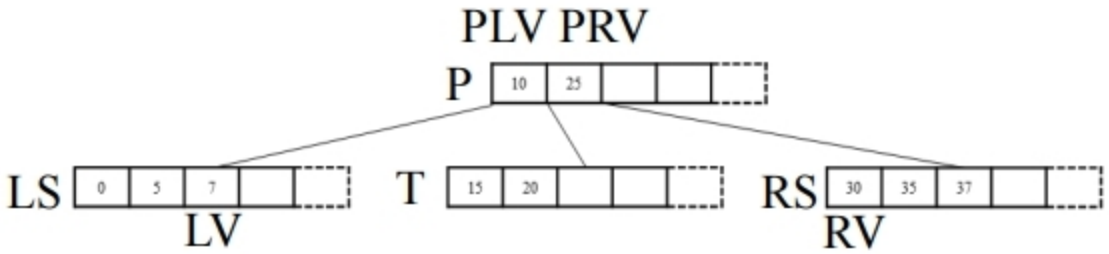
B-Tree

- insert(Btree tree, Key *k*)
- Example (5-way)
  - insert(tree, 40)



B-Tree

- delete(Btree tree, Key *k*)
  - Search a node, **T**, that *k* is included
  - IF (T is a leaf node)
    - Remove *k* from **T**
    - Prepare the variables



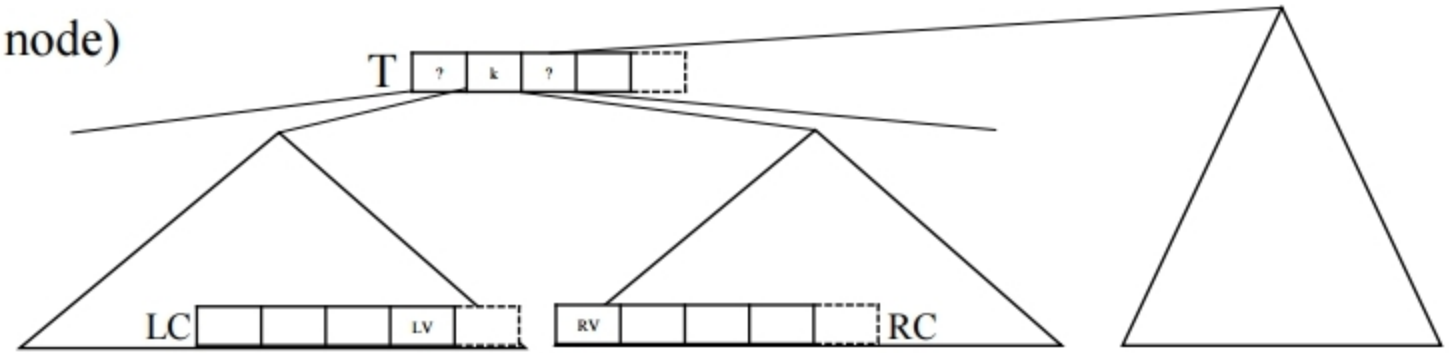
  - IF T violates the min keys prop ($|T| < \left\lceil \frac{m}{2} \right\rceil - 1$)
    - IF $|L| >= \left\lceil \frac{m}{2} \right\rceil$, **Reorganize**: reorganize LV, PLV, T
    - ELSE IF $|R| >= \left\lceil \frac{m}{2} \right\rceil$, **Reorganize**: reorganize RV, PRV, T
    - ELSE (i.e., P violates the min keys prop ($|P| < \left\lceil \frac{m}{2} \right\rceil - 1$))
      - **Merge:** merge LS, PLV, T and be a left child of PRV **or**
      - **Merge:** merge T, PRV, RS and be a right child of PLV
      - T = P and GO TO the arrow
  - ELSE (T is an internal node)
    - …

- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root $\left\lceil \frac{m}{2} \right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil \frac{m}{2} \right\rceil - 1$ for the others

6 way: >=2
5 way : >=2
4 way: >=1

B-Tree

- delete(Btree tree, Key *k*)
  - Search a node, **T**, that *k* is included
  - IF (T is a leaf node)
    - …
  - ELSE (T is an internal node)



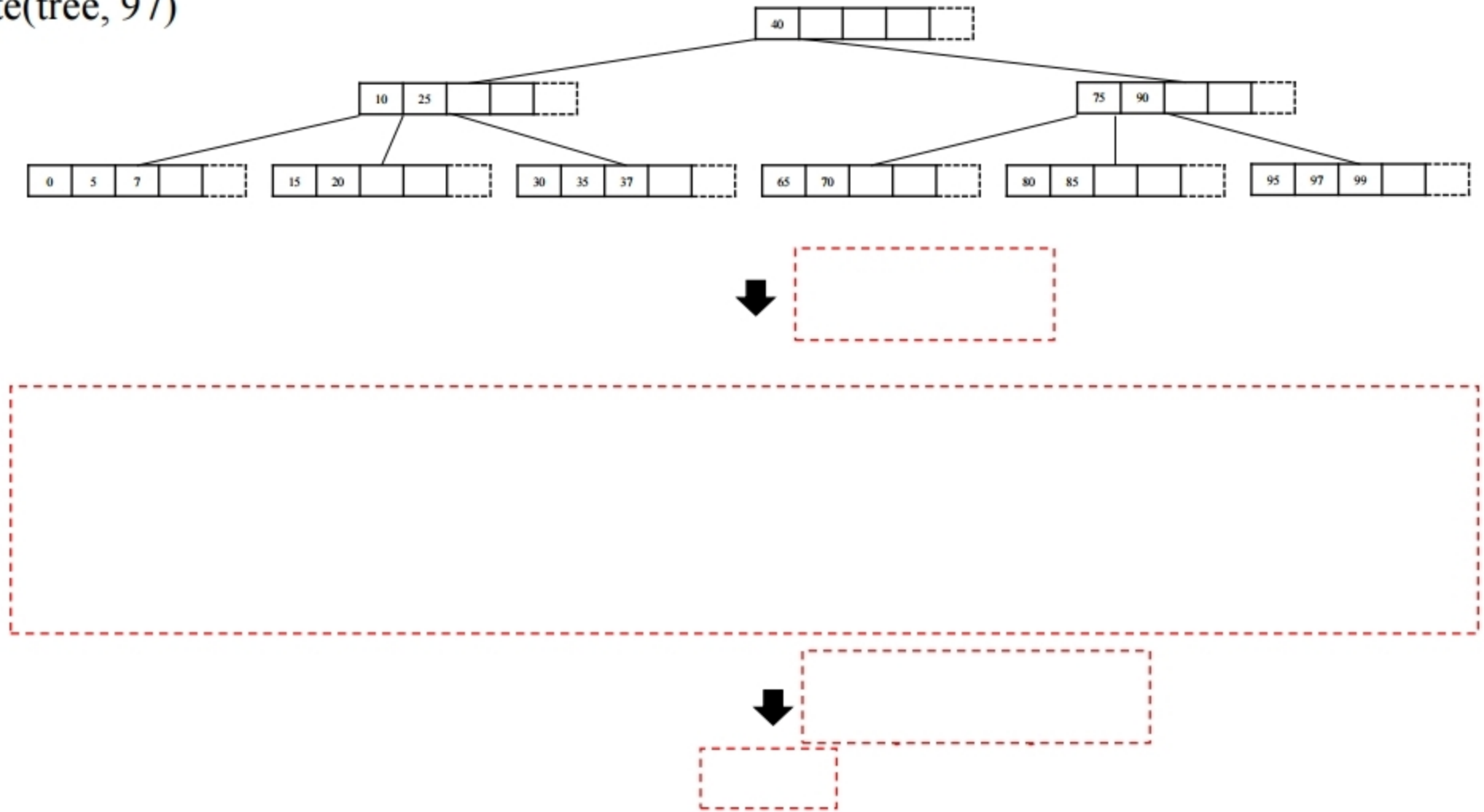
    - Remove *k* from **T**
      - IF |LC| >= $\left\lceil \frac{m}{2} \right\rceil$, **Borrow**: Locate LV to *k*
      - ELSE IF |RC| >= $\left\lceil \frac{m}{2} \right\rceil$, **Borrow**: Locate RV to *k*
      - ELSE (i.e., the min keys prop (|T| < $\left\lceil \frac{m}{2} \right\rceil - 1$))
        - Borrow: Locate LV to *k*
        - T = LC
        - GO TO the arrow

OR

Borrow: Locate RV to *k*
T = RC
GO TO the arrow

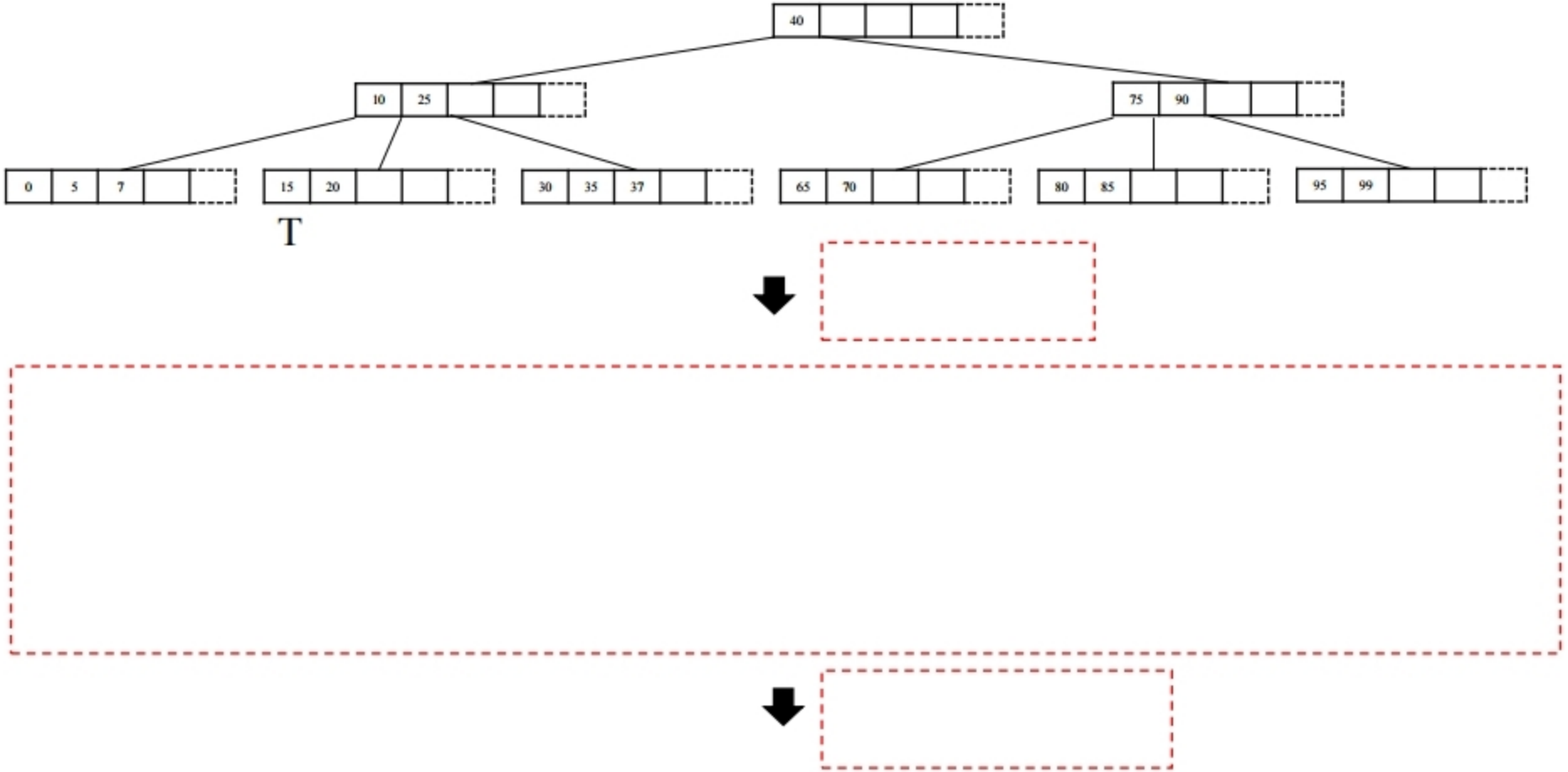
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 1) delete(tree, 97)



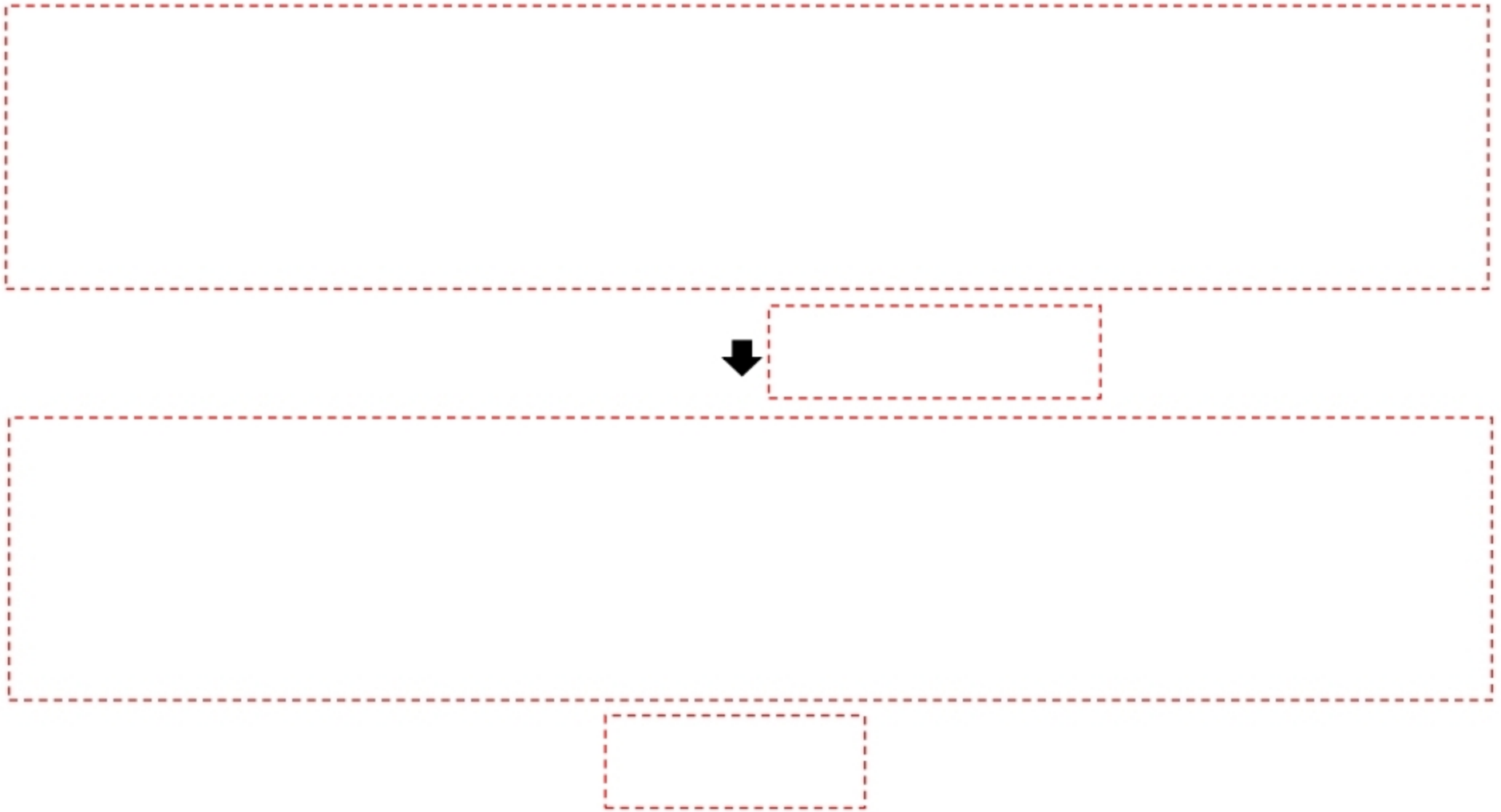
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 2) delete(tree, 20)



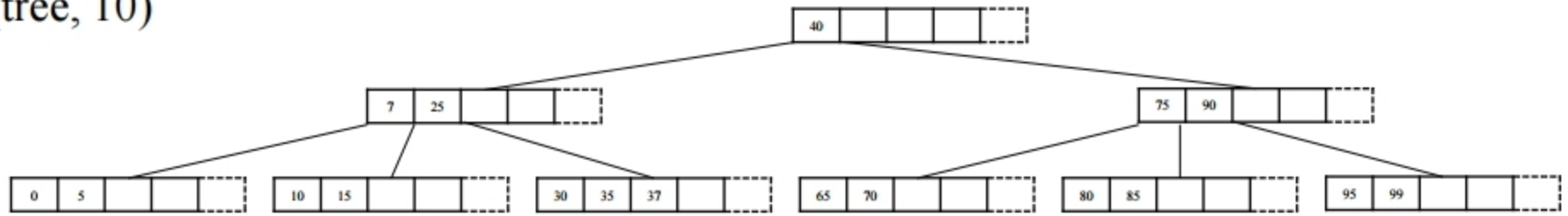
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 2) delete(tree, 20)



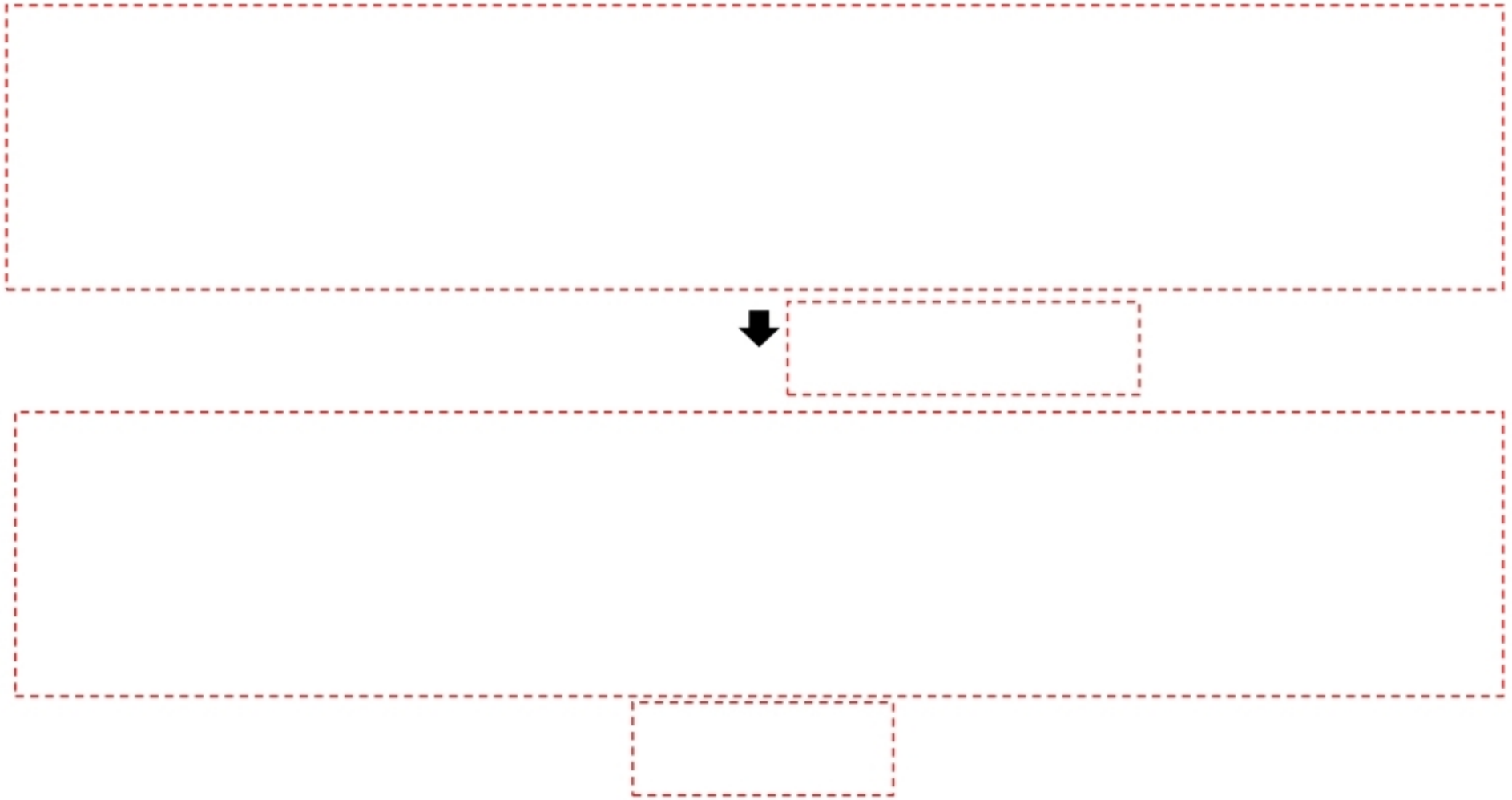
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 3) delete(tree, 10)



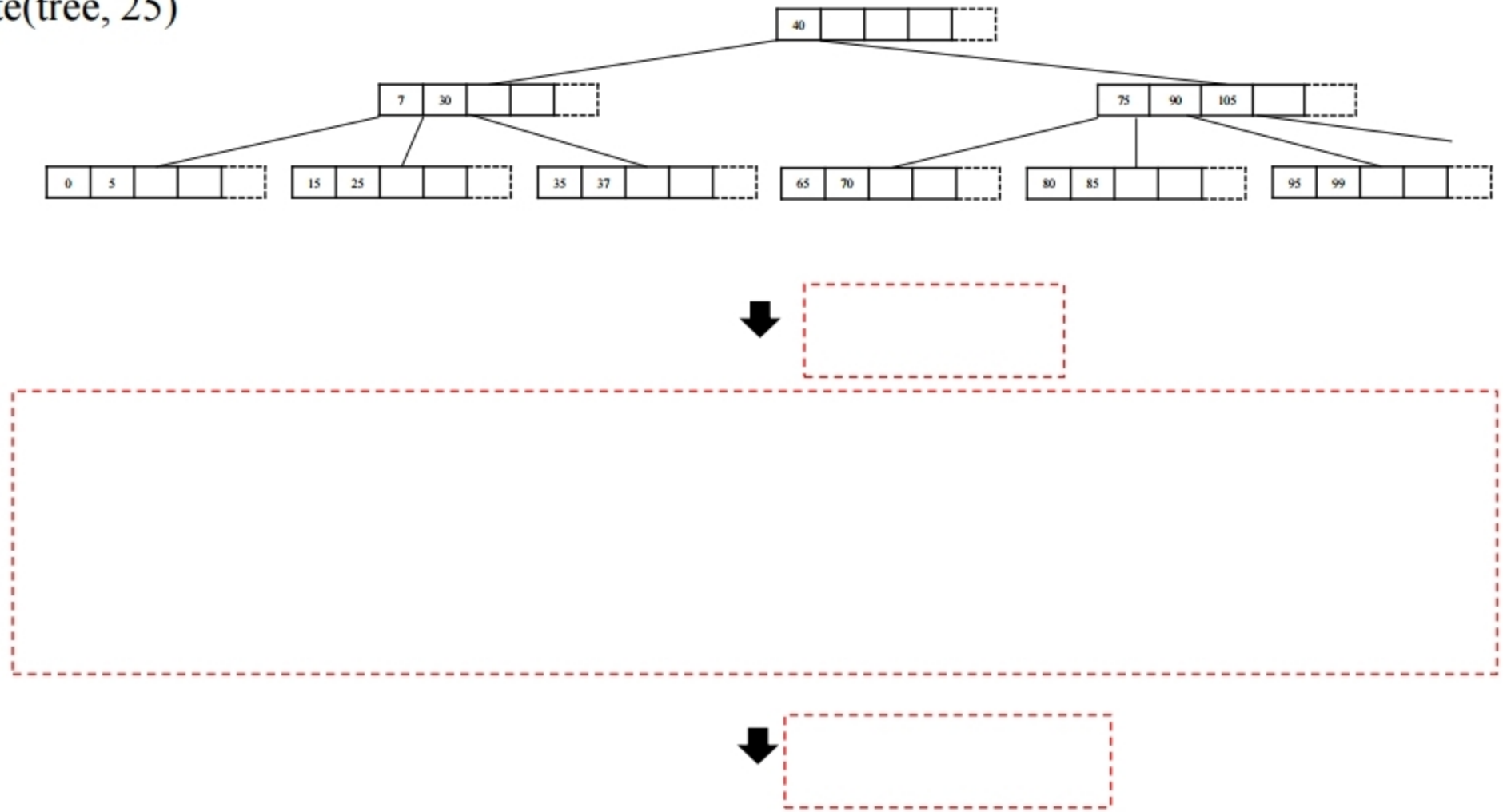
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 3) delete(tree, 10)



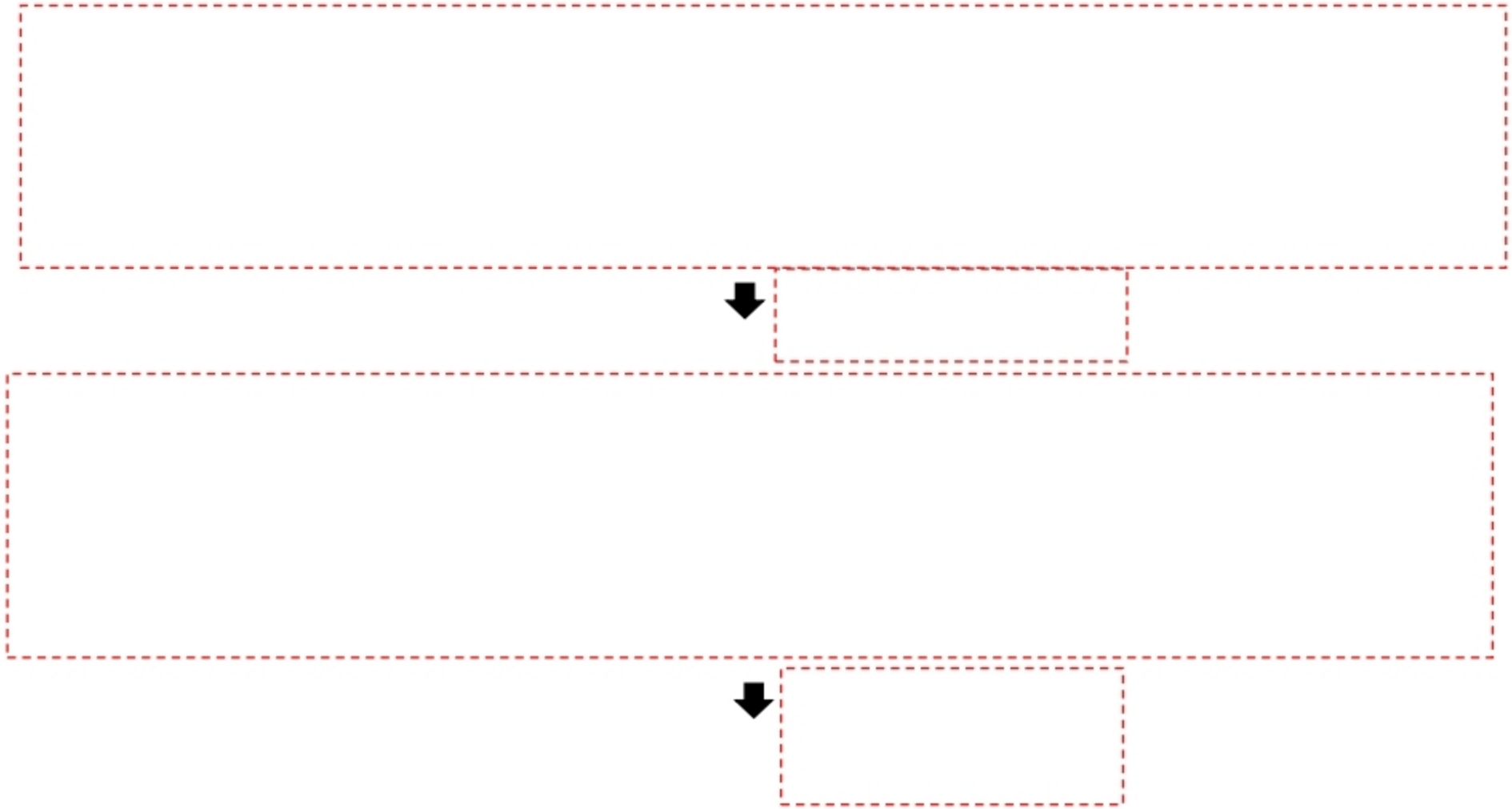
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 4) delete(tree, 25)



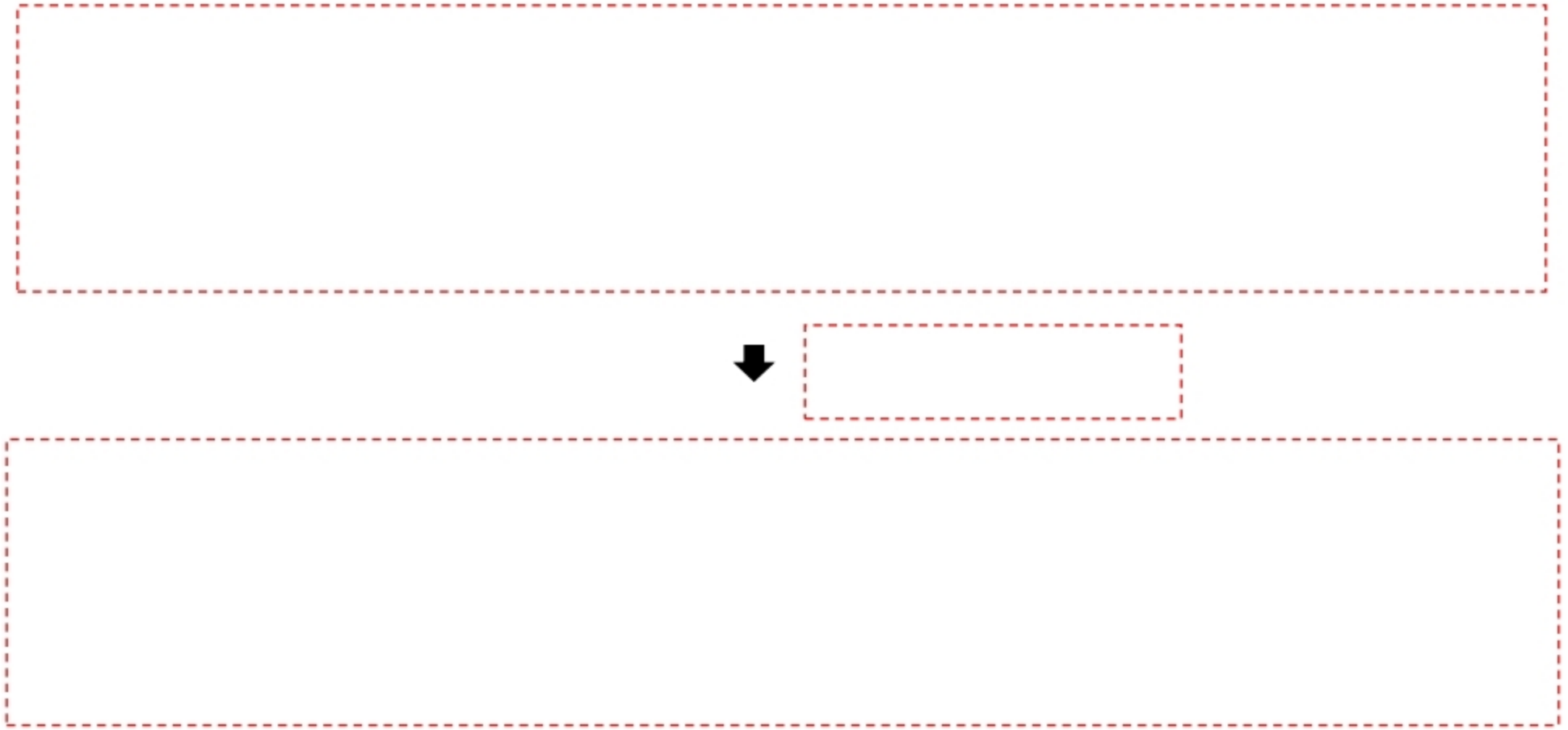
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 4) delete(tree, 25)



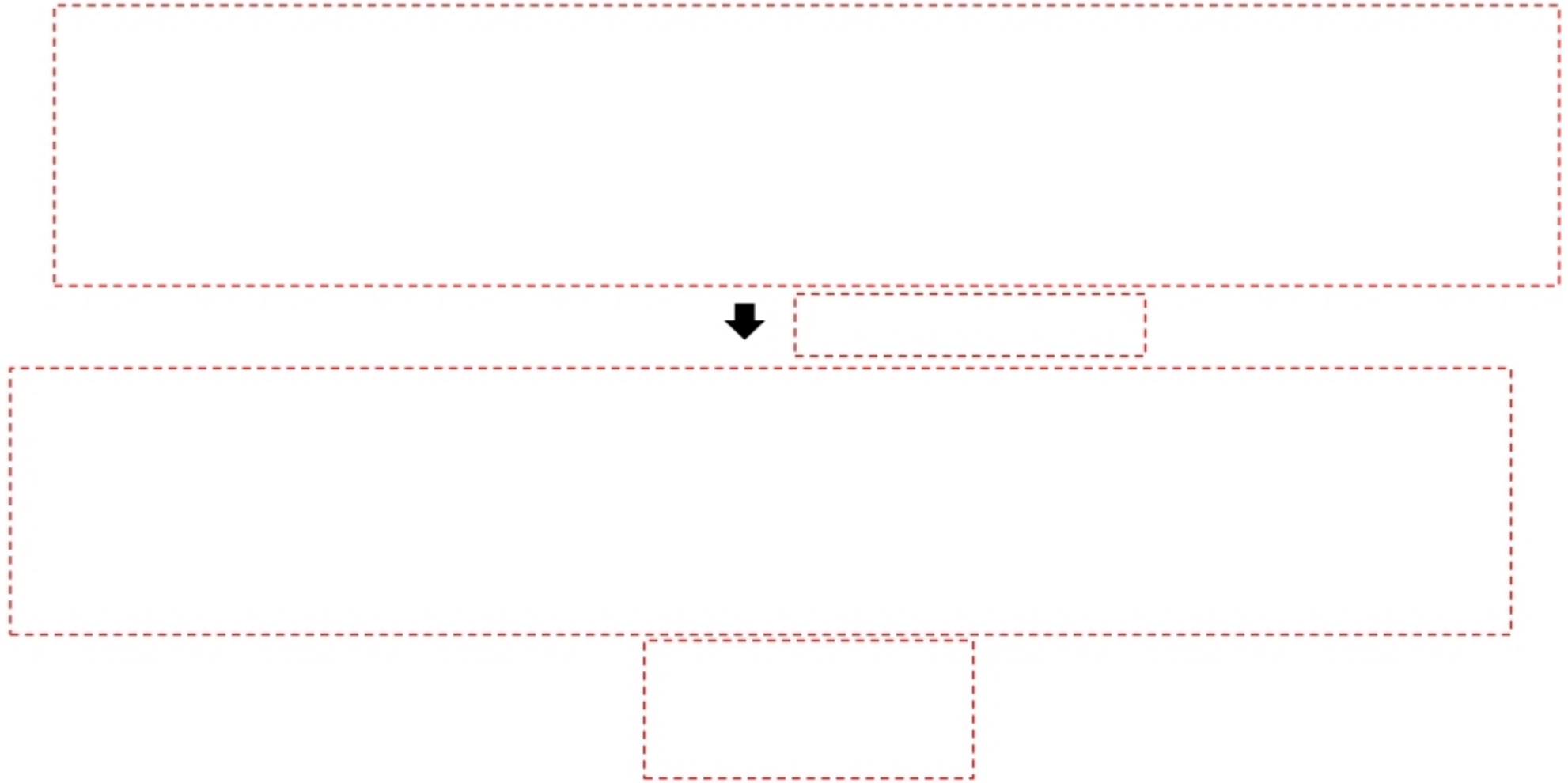
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 4) delete(tree, 25)



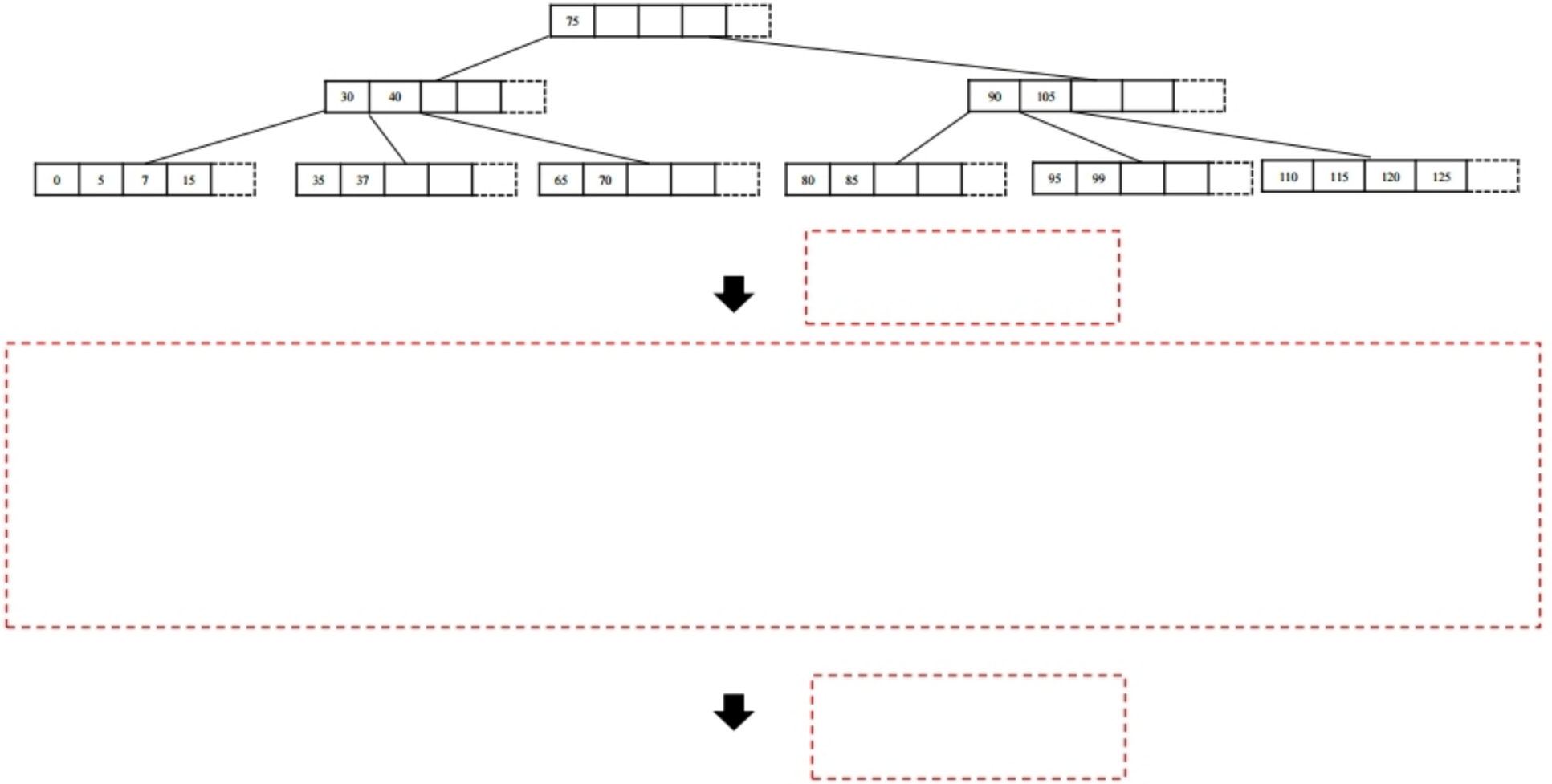
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 4) delete(tree, 25)



B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 5) delete(tree, 65)

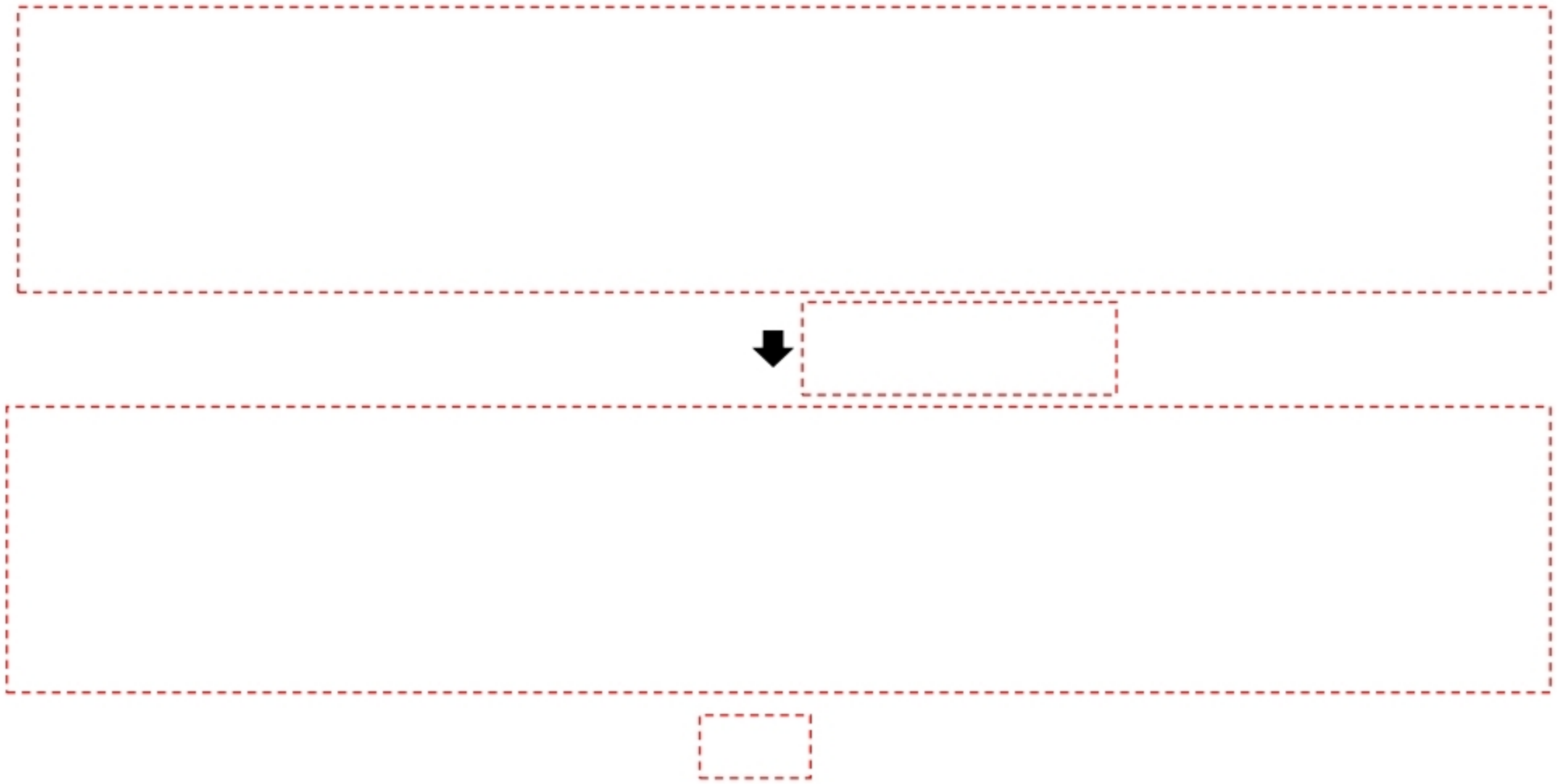


B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 5) delete(tree, 65)

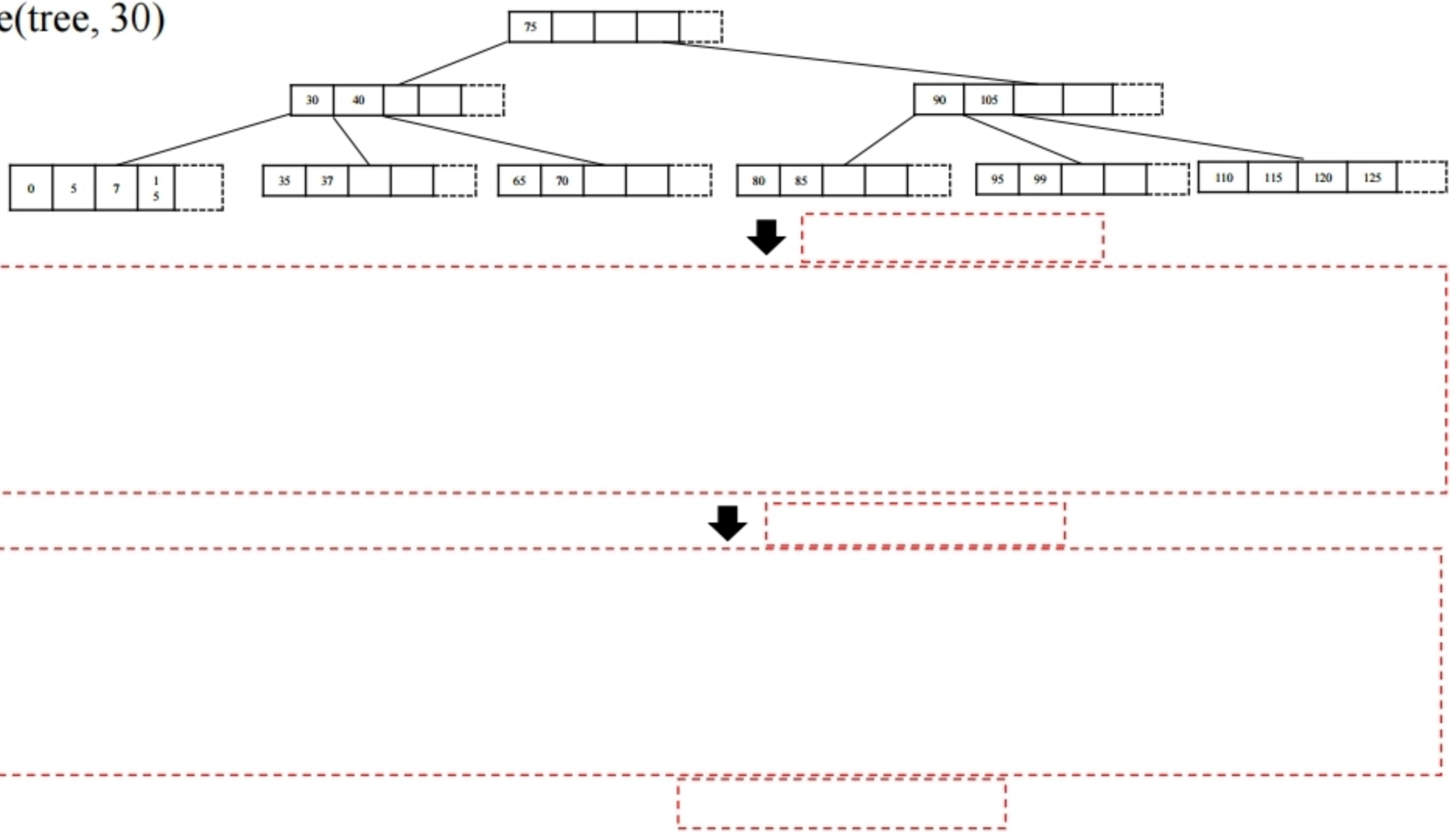
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 5) delete(tree, 65)



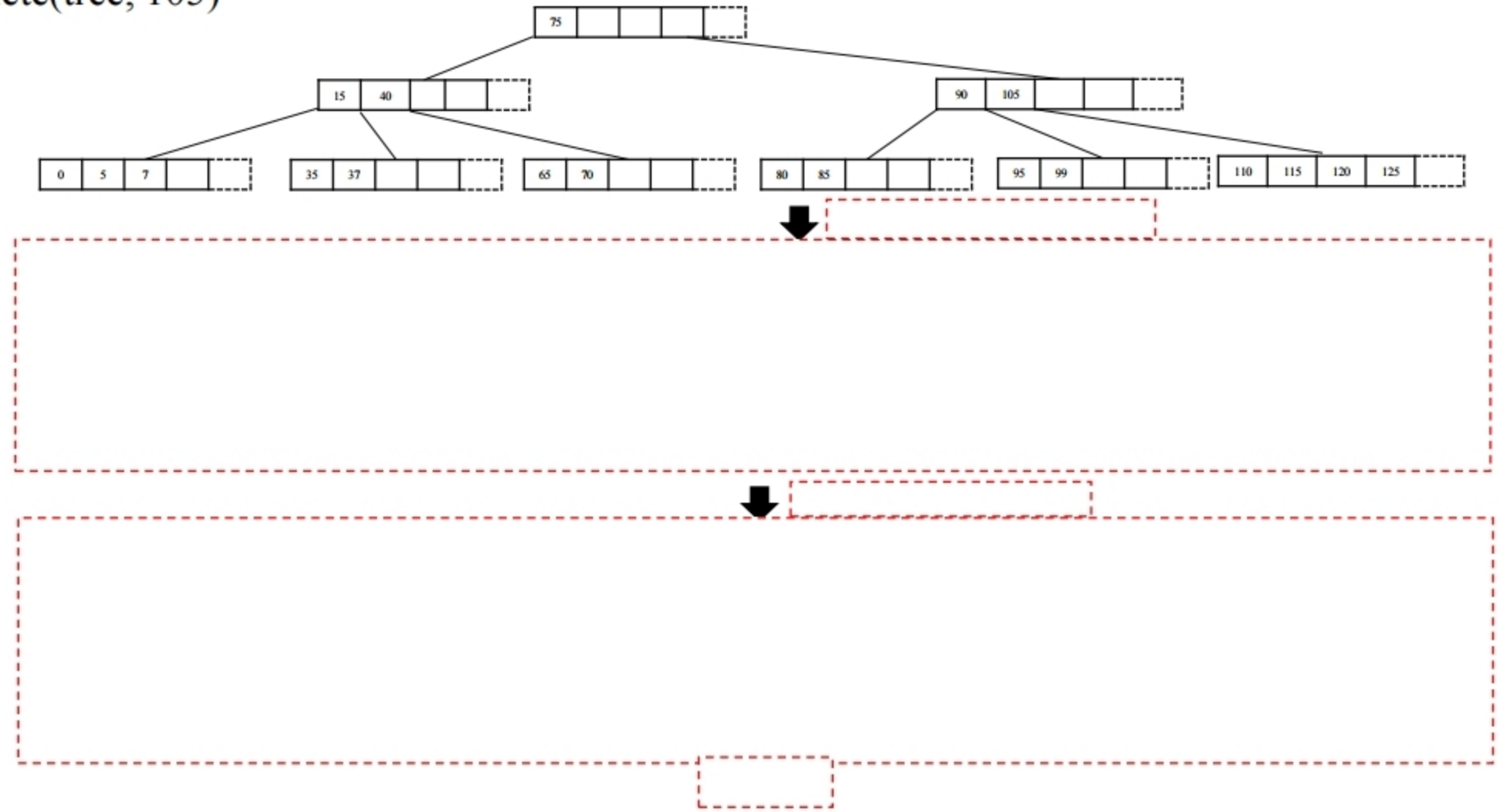
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 6) delete(tree, 30)



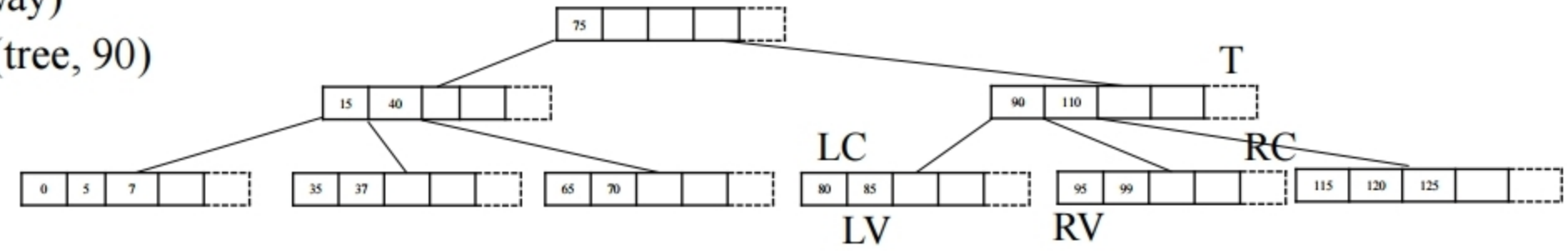
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 7) delete(tree, 105)



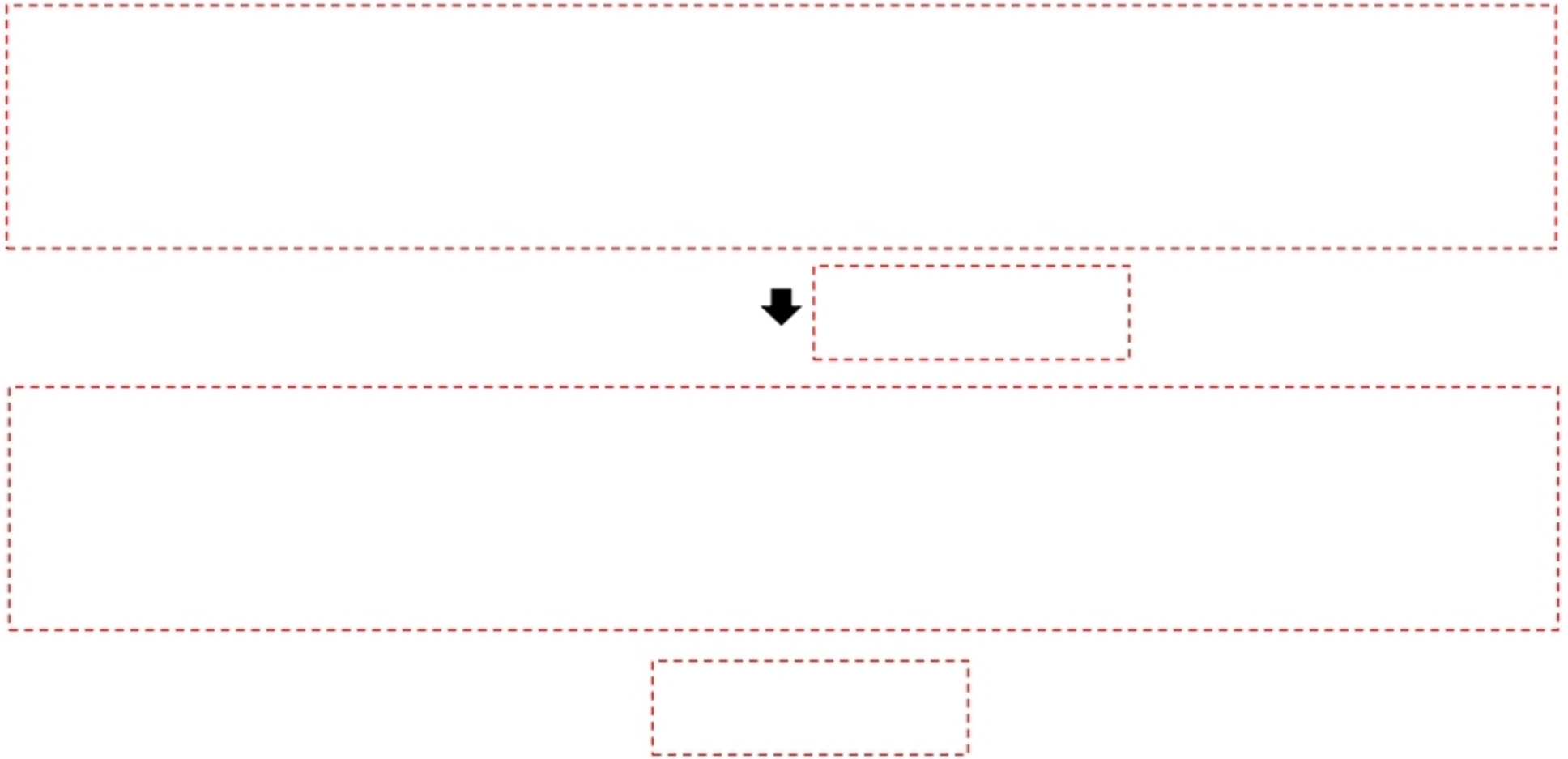
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 8) delete(tree, 90)



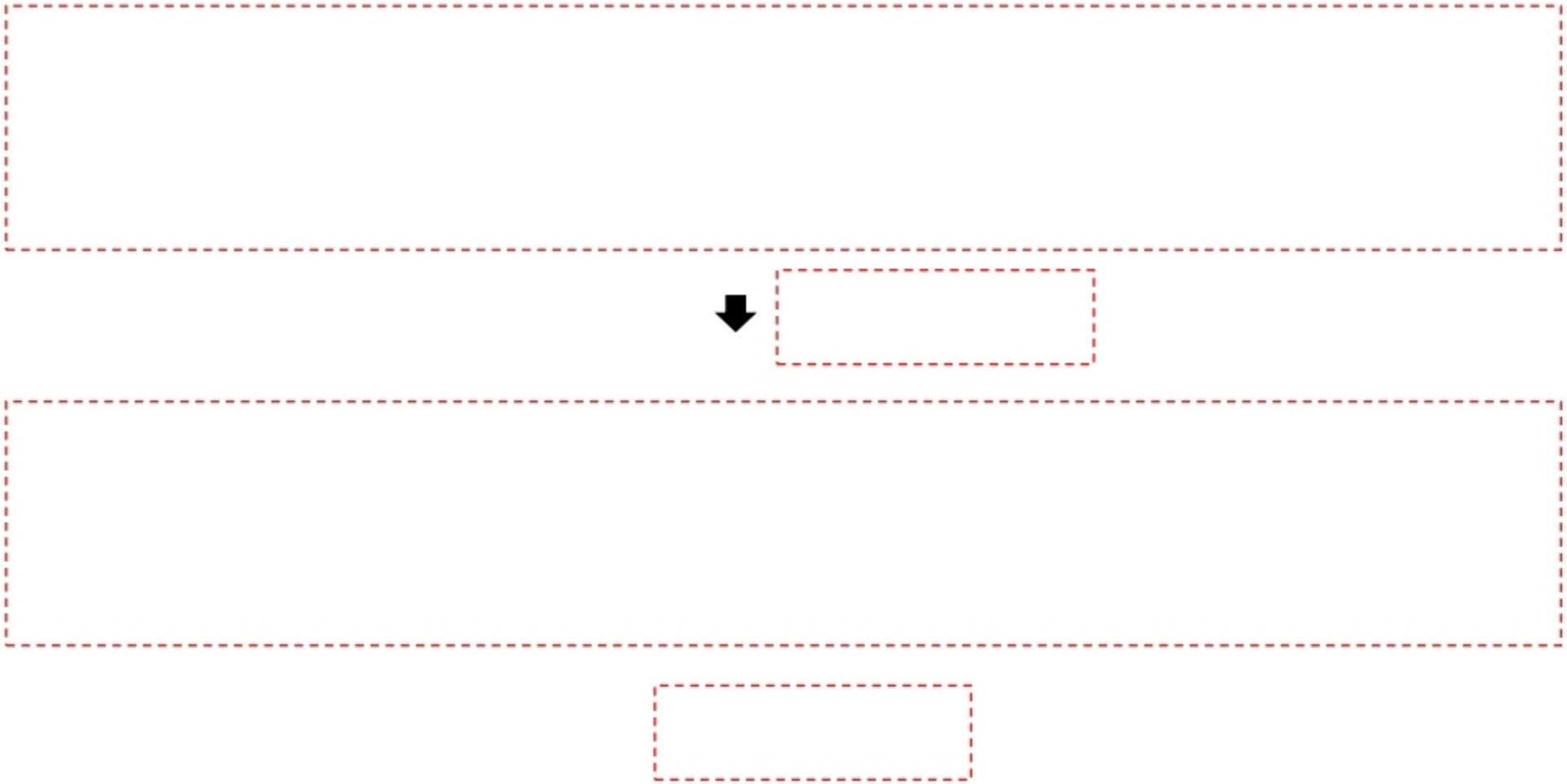
B-Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 8) delete(tree, 90)



B-Tree

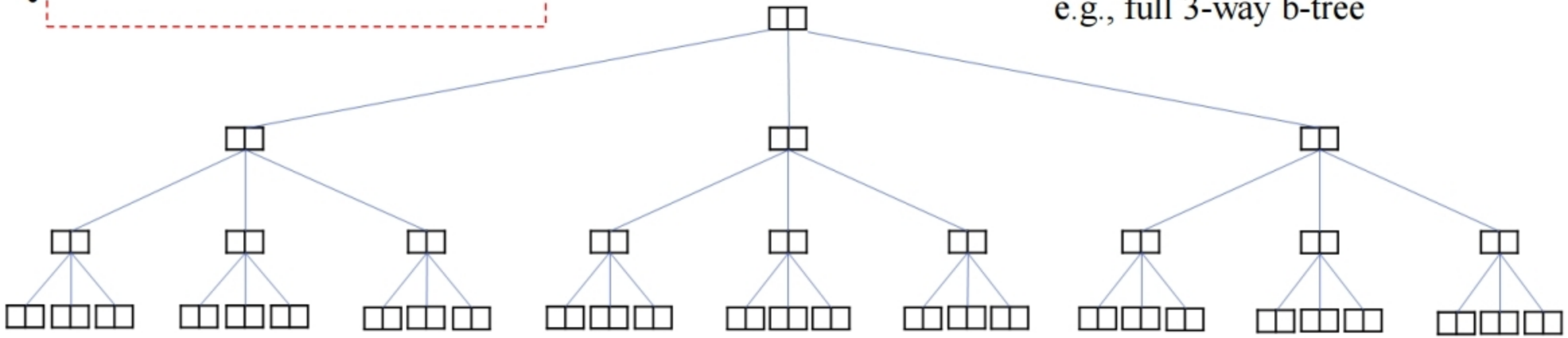
- delete(Btree tree, Key *k*)
- Example (5-way)
  - 8) delete(tree, 90)



B- Tree

- The maximum number of keys (i.e., **n**) for a height h, way m
  (when a height is the maximum number of edges from a leaf node to the root node)
  - ***n*** =

e.g., full 3-way b-tree

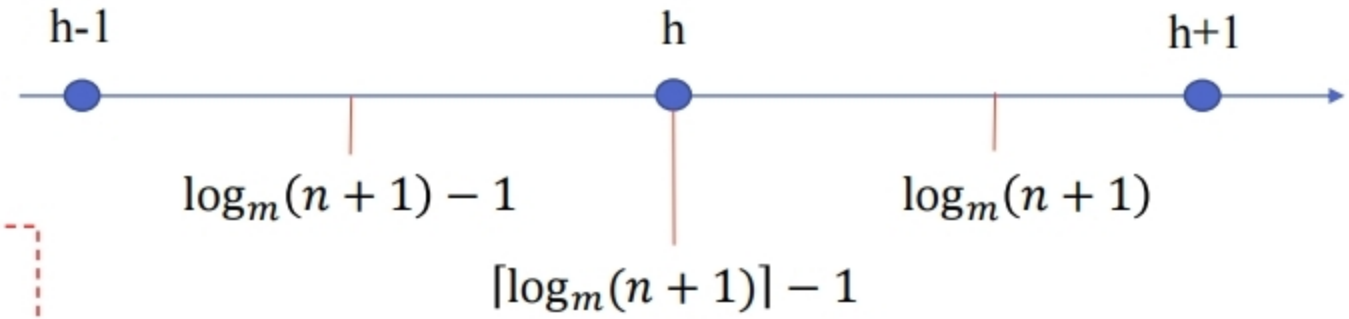


B- Tree

- The minimum height of the B-Tree
  - $m^h - 1 < n \leq m^{h+1} - 1$

1. $m^h - 1 < n$

2. $n \leq m^{h+1} - 1$



  - $\log_m(n+1) - 1 \leq h < \log_m(n+1)$

B- Tree

- The minimum number of keys (i.e., **n**) for a height h, way m
  (when a height is the maximum number of edges from a leaf node to the root node)
  - $n =$

e.g., 5-way b-tree, height=2

$$min\_key = \left\lceil \frac{m}{2} \right\rceil - 1 = 2$$

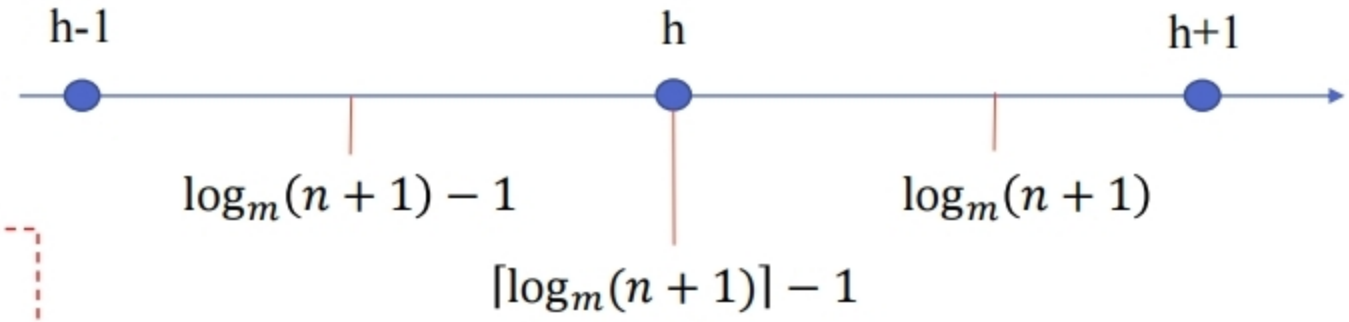
$$b = \left\lceil \frac{m}{2} \right\rceil$$

B- Tree

- The maximum height of the B-Tree
  - $b^h - 1 < n \leq b^{h+1} - 1$

1. $b^h - 1 < n$

2. $n \leq b^{h+1} - 1$



  - $\log_b(n+1) - 1 \leq h < \log_b(n+1)$

B- Tree

- The maximum/minimum height of the B-Tree
  - Maximum
    - $h_{max} = \lceil \log_b(n+1) \rceil - 1$
  - Minimum
    - $h_{min} = \lceil \log_m(n+1) \rceil - 1$

- The height of the B-Tree
  - $\lceil \log_m(n+1) \rceil - 1 \leq h \leq \lceil \log_b(n+1) \rceil - 1$
  - $\lceil \log_m(n+1) \rceil - 1 \leq h \leq \left\lceil \log_{\lceil \frac{m}{2} \rceil}(n+1) \right\rceil - 1$
  - $h = O(\log_m n) = O\left(\frac{\log_2(n)}{\log_2(m)}\right) = O(\log n)$

B+ Tree

- balanced m-way tree
- Generalization of BST in which a node can have more than one key & more than 2 children
- A node maintains sorted keys
- All leaf node must be at same level
- **Data is only in leaf nodes**
- **Leaf nodes formulate a linked list**
- B+tree of order m has following properties:
  - every node can have max m children
  - min children: leaf → 0
    (non-leaf) root → 2
    internal nodes → $\left\lceil \frac{m}{2} \right\rceil$
  - every node has max (m-1) keys
  - min keys: root node → 1
    all other nodes → $\left\lceil \frac{m}{2} \right\rceil - 1$

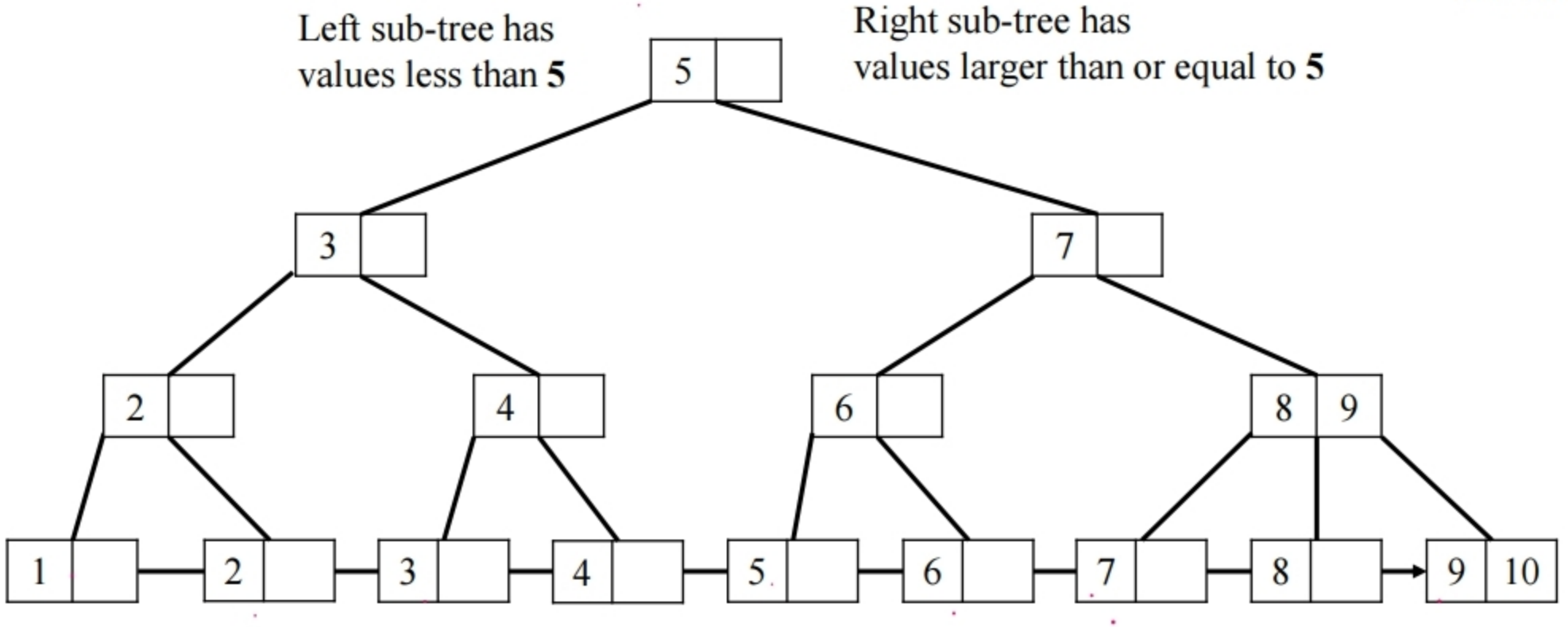
6 way: 3
5 way : 3
4 way: 2

6 way: 2
5 way : 2
4 way: 1

B+ Tree

- Example: 3-way tree

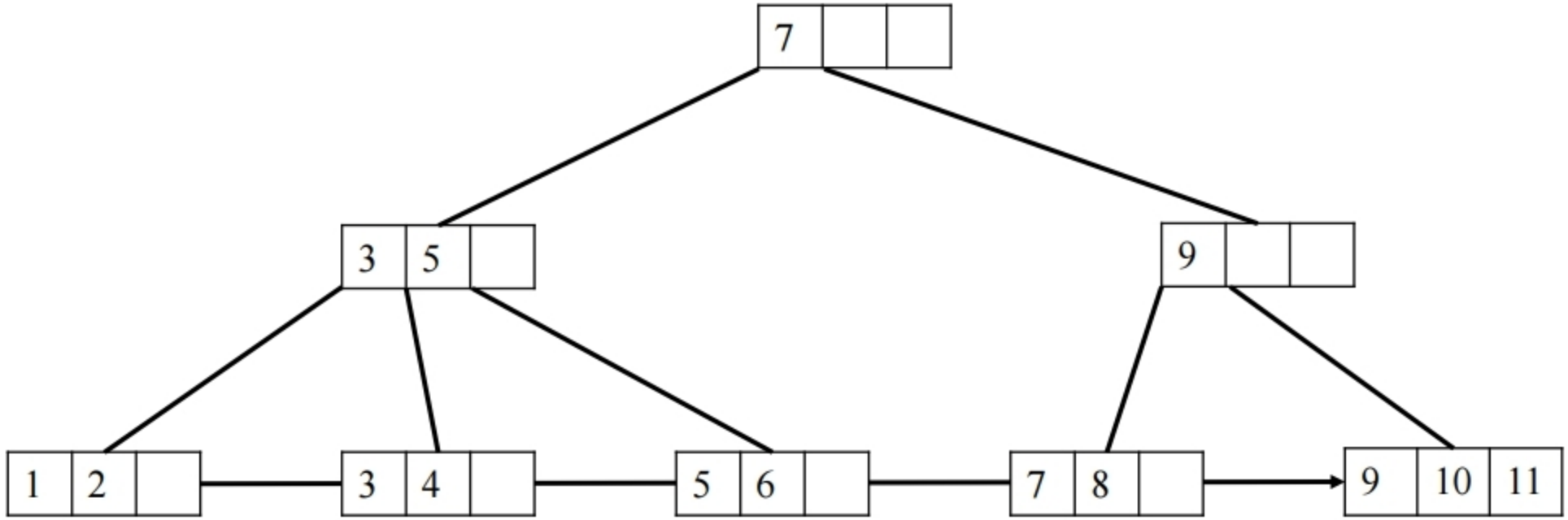
- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root $\left\lceil \frac{m}{2} \right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil \frac{m}{2} \right\rceil - 1$ for the others



B+ Tree

- Example: 4-way tree

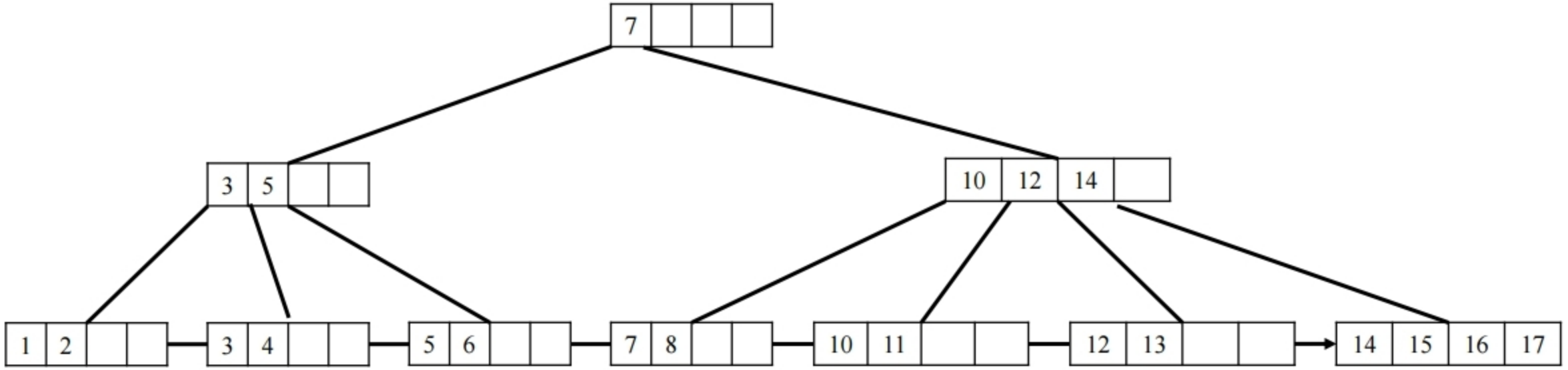
- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root
    $\lceil \frac{m}{2} \rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\lceil \frac{m}{2} \rceil - 1$ for the others



B+ Tree

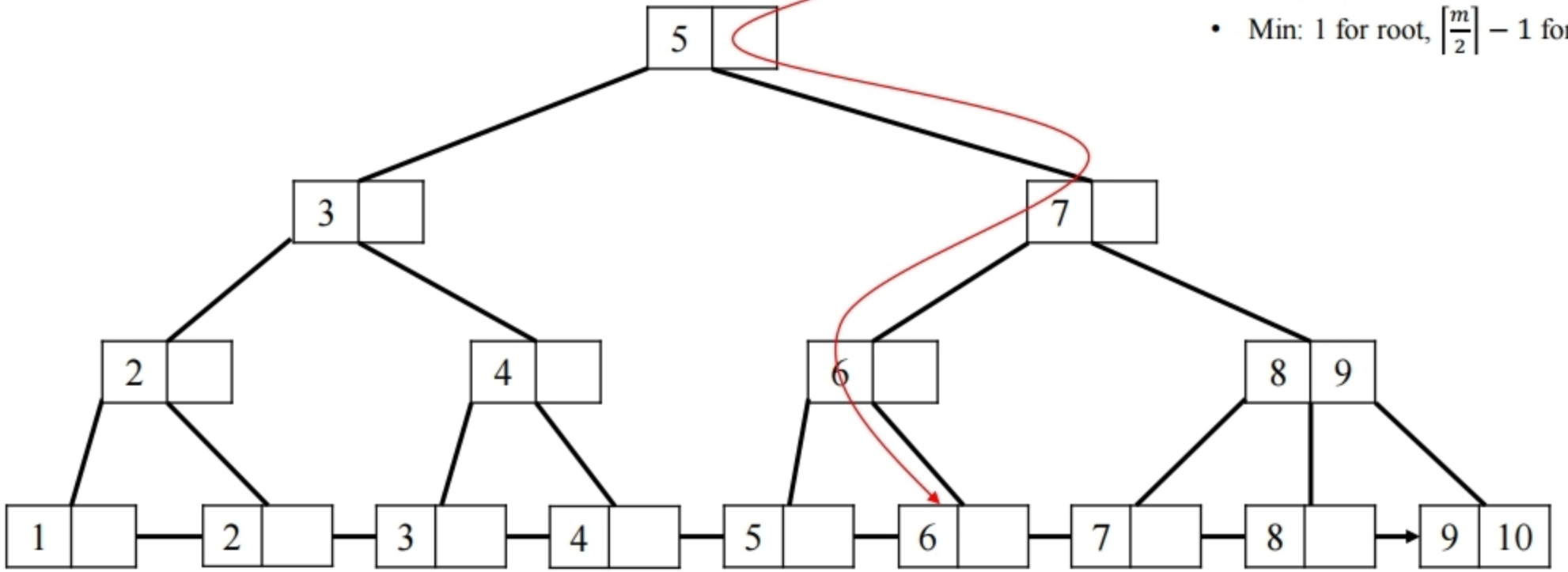
- Example: 5-way tree

- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root
    $\lceil \frac{m}{2} \rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\lceil \frac{m}{2} \rceil - 1$ for the others



B+ Tree

- search(B+tree tree, Key *k*)
  - Like BST
    - search(tree, 6)
      - should go to a leaf node having its data

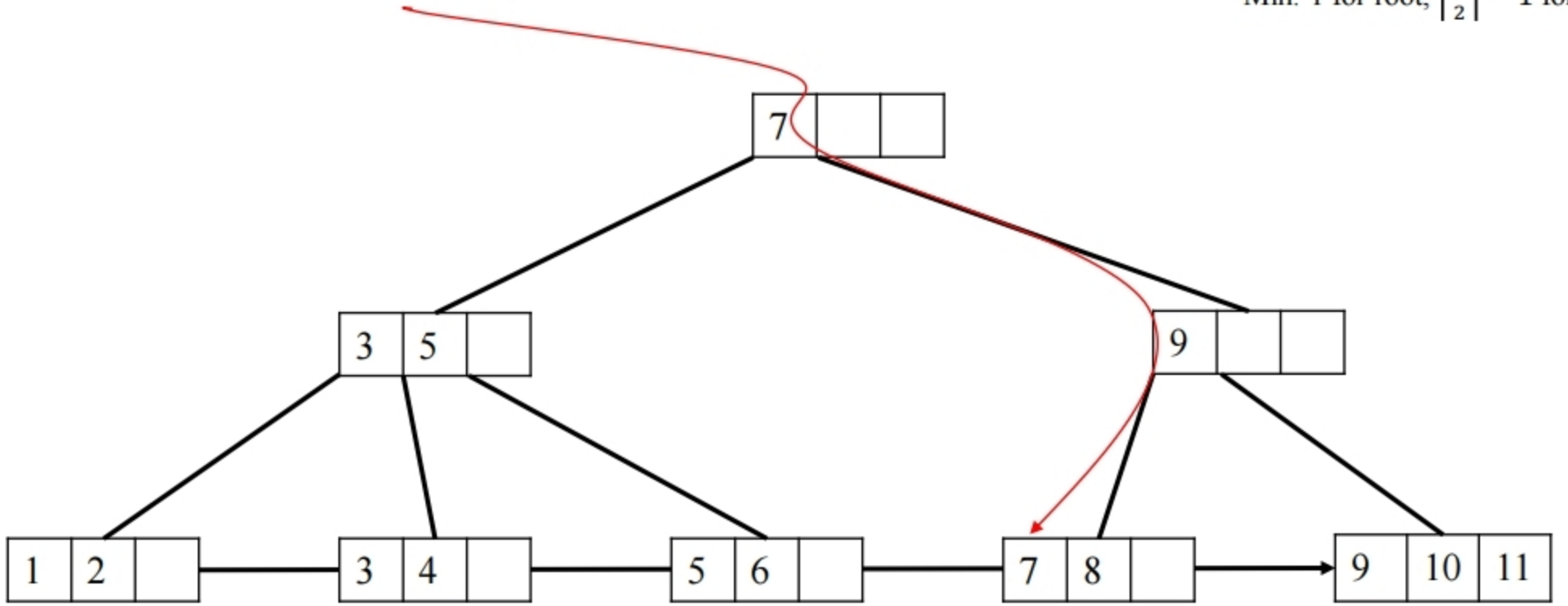


- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root $\lceil \frac{m}{2} \rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\lceil \frac{m}{2} \rceil - 1$ for the others

B+ Tree

- search(B+tree tree, Key *k*)
  - Like BST
    - search(tree, 7)
      - should go to a leaf node having its data

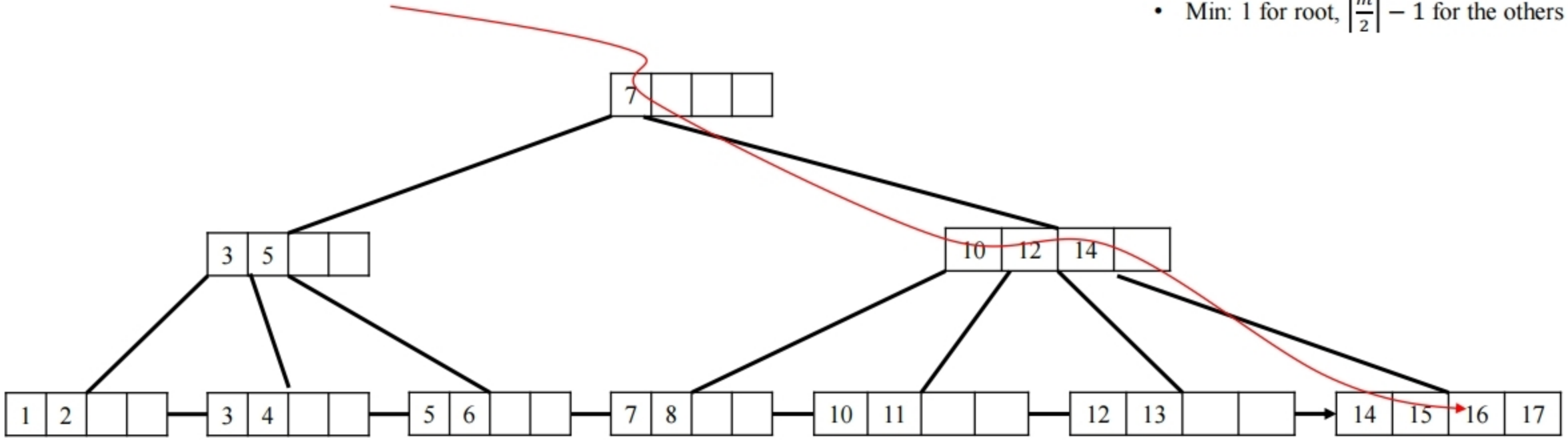
- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root $\lceil \frac{m}{2} \rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\lceil \frac{m}{2} \rceil - 1$ for the others



B+ Tree

- search(B+tree tree, Key *k*)
  - Like BST
    - search(tree, 16)
      - should go to a leaf node having its data

- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root $\left\lceil \frac{m}{2} \right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil \frac{m}{2} \right\rceil - 1$ for the others



B+ Tree

- insert(B+tree tree, Key *k*)
  1. Search a leaf node, **T**, that *k* to be inserted
  2. If |T| <= m-1, |T|=the number of keys in **T,** (i.e., do not violate the max keys prop.), then DONE
  3. else
     - the leaf node splits into three parts (0,1,…,m-1)
       - Left: 0 to $\left\lceil \frac{m-1}{2} \right\rceil - 1$ Middle: $\left\lceil \frac{m-1}{2} \right\rceil$
       - Right: $\left\lceil \frac{m-1}{2} \right\rceil$ to m-1
     - Middle goes to parent node
     - Left, Right become a left child and a right child of Middle, respectively
     - **T** = A node contains Middle
     - Go to Btree's 2

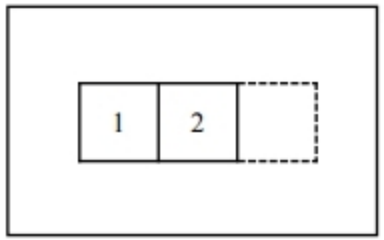
- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root $\left\lceil \frac{m}{2} \right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil \frac{m}{2} \right\rceil - 1$ for the others+

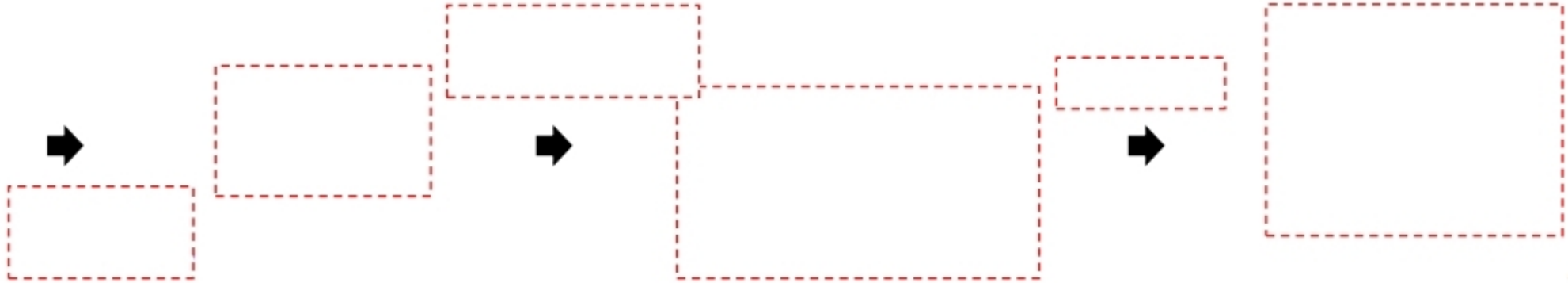
B+ Tree

- insert(B+tree tree, Key *k*)
  1. Search a leaf node, **T**, that *k* to be inserted
  2. If |T| <= m-1, |T|=the number of keys in **T,** (i.e., do not violate the max keys prop.), then DONE
  3. else
     - the leaf node splits into three parts (0,1,…,m-1)
       - Left: 0 to $\left\lceil\frac{m-1}{2}\right\rceil - 1$ Middle: $\left\lceil\frac{m-1}{2}\right\rceil$
       - Right: $\left\lceil\frac{m-1}{2}\right\rceil$to m-1
     - Middle goes to parent node
     - Left, Right become a left child and a right child of Middle, respectively
     - **T** = A node contains Middle
     - Go to Btree's 2

- Example (3-way)
  - insert(tree, 3)

B+tree tree

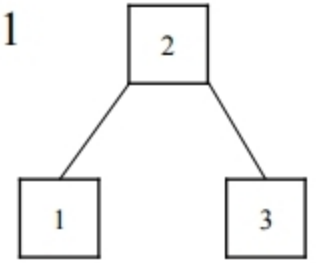




- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root $\left\lceil\frac{m}{2}\right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil\frac{m}{2}\right\rceil - 1$ for the others+

Left: 0 to $\left\lceil\frac{m-1}{2}\right\rceil$-1 Middle: $\left\lceil\frac{m-1}{2}\right\rceil$

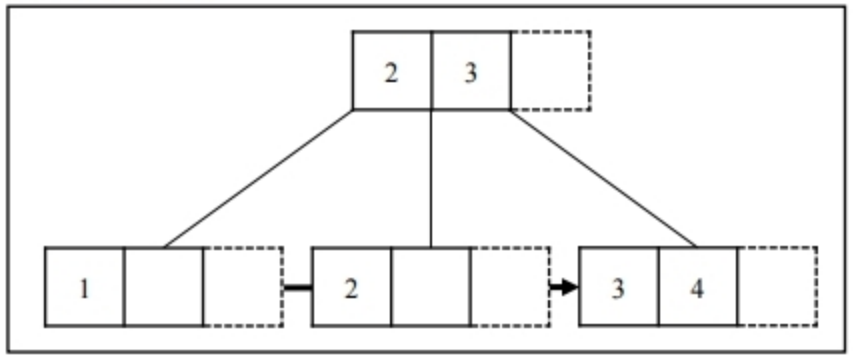
Right: $\left\lceil\frac{m-1}{2}\right\rceil + 1$ to m-1



B+ Tree

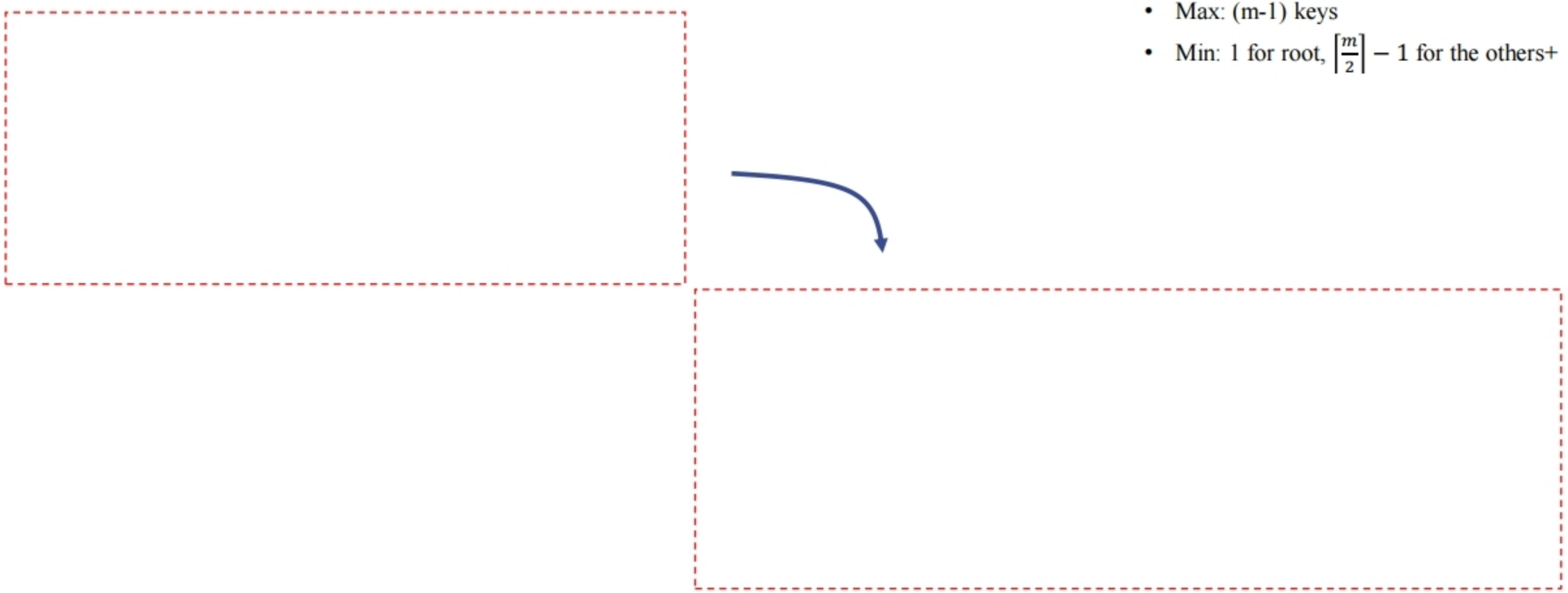
- insert(B+tree tree, Key *k*)
- Example (3-way)
  - insert(tree, 5)

B+tree tree



B+ Tree

- insert(B+tree tree, Key *k*)
- Example (3-way)
  - insert(tree, 5)



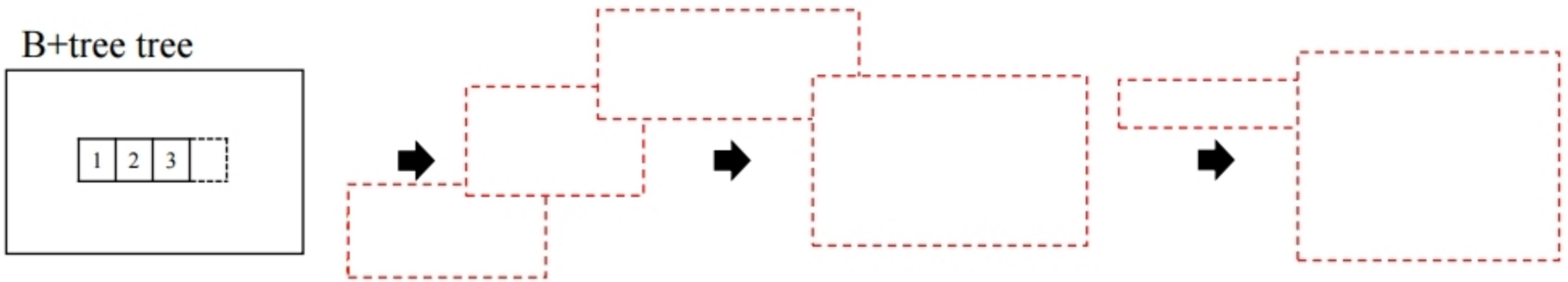
- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root
    $\lceil \frac{m}{2} \rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\lceil \frac{m}{2} \rceil - 1$ for the others+

B+ Tree

- insert(B+tree tree, Key *k*)
  1. Search a leaf node, **T**, that *k* to be inserted
  2. If |T| <= m-1, |T|=the number of keys in **T,** (i.e., do not violate the max keys prop.), then DONE
  3. else
     - the leaf node splits into three parts (0,1,…,m-1)
       - Left: 0 to $\left\lceil \frac{m-1}{2} \right\rceil - 1$ Middle: $\left\lceil \frac{m-1}{2} \right\rceil$
       - Right: $\left\lceil \frac{m-1}{2} \right\rceil$ to m-1
     - Middle goes to parent node
     - Left, Right become a left child and a right child of Middle, respectively
     - **T** = A node contains Middle
     - Go to Btree's 2

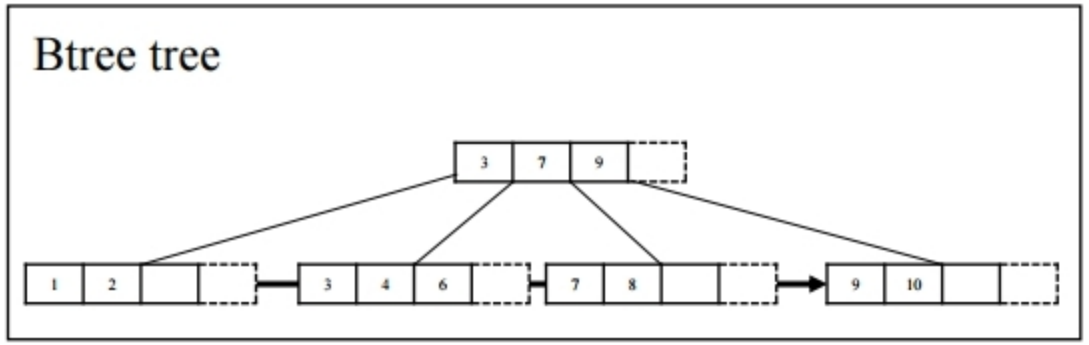
- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root $\left\lceil \frac{m}{2} \right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil \frac{m}{2} \right\rceil - 1$ for the others+

- Example (4-way)
  - insert(tree, 4)



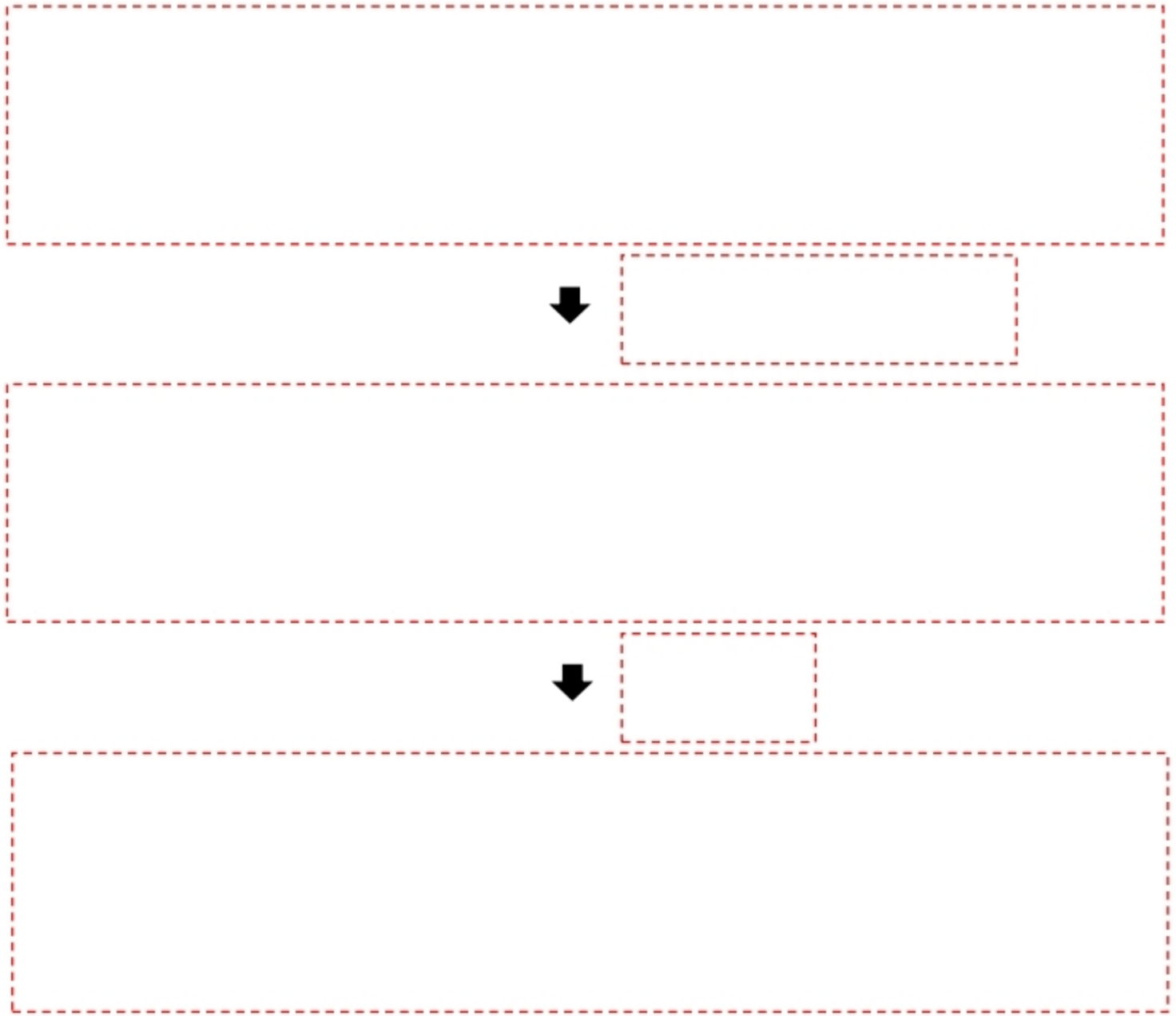
B+ Tree

- insert(B+tree tree, Key *k*)
- Example (4-way)
  - insert(tree, 5)



B+ Tree

- insert(B+tree tree, Key *k*)
- Example (4-way)
  - insert(tree, 5)

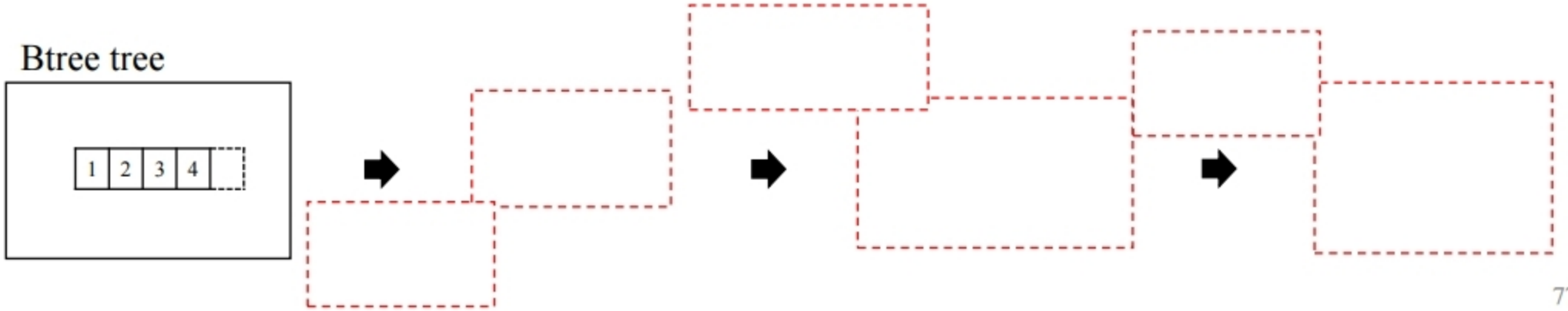


B+ Tree

- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root $\left\lceil \frac{m}{2} \right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil \frac{m}{2} \right\rceil - 1$ for the others

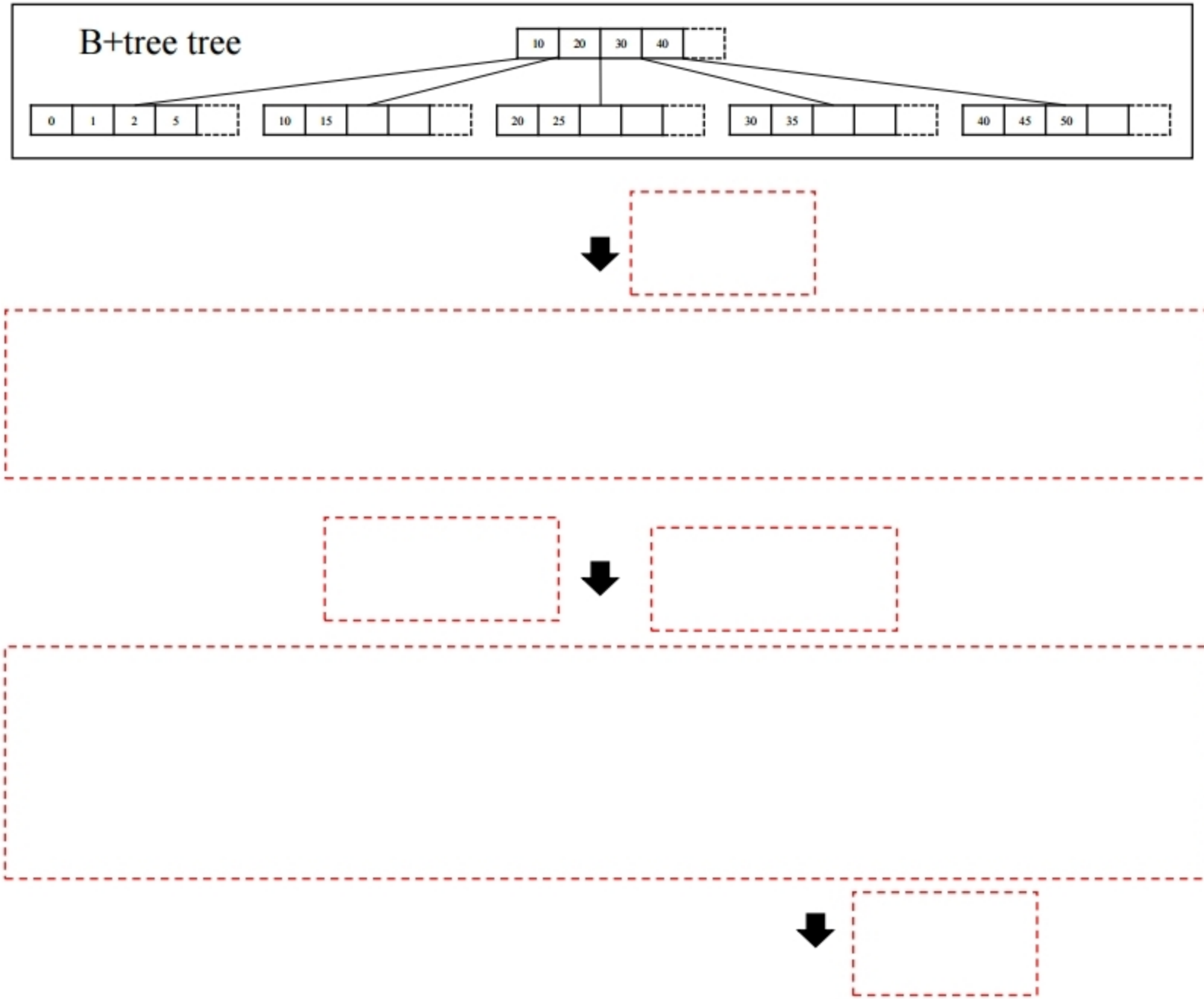
- insert(B+tree tree, Key *k*)
  1. Search a leaf node, **T**, that *k* to be inserted
  2. If |T| <= m-1, |T|=the number of keys in **T,** (i.e., do not violate the max keys prop.), then DONE
  3. else
     - the leaf node splits into three parts (0,1,…,m-1)
       - Left: 0 to $\left\lceil \frac{m-1}{2} \right\rceil - 1$ Middle: $\left\lceil \frac{m-1}{2} \right\rceil$
       - Right: $\left\lceil \frac{m-1}{2} \right\rceil$ to m-1
     - Middle goes to parent node
     - Left, Right become a left child and a right child of Middle, respectively
     - **T** = A node contains Middle
     - Go to Btree's 2

- Example (5-way)
  - insert(tree, 5)



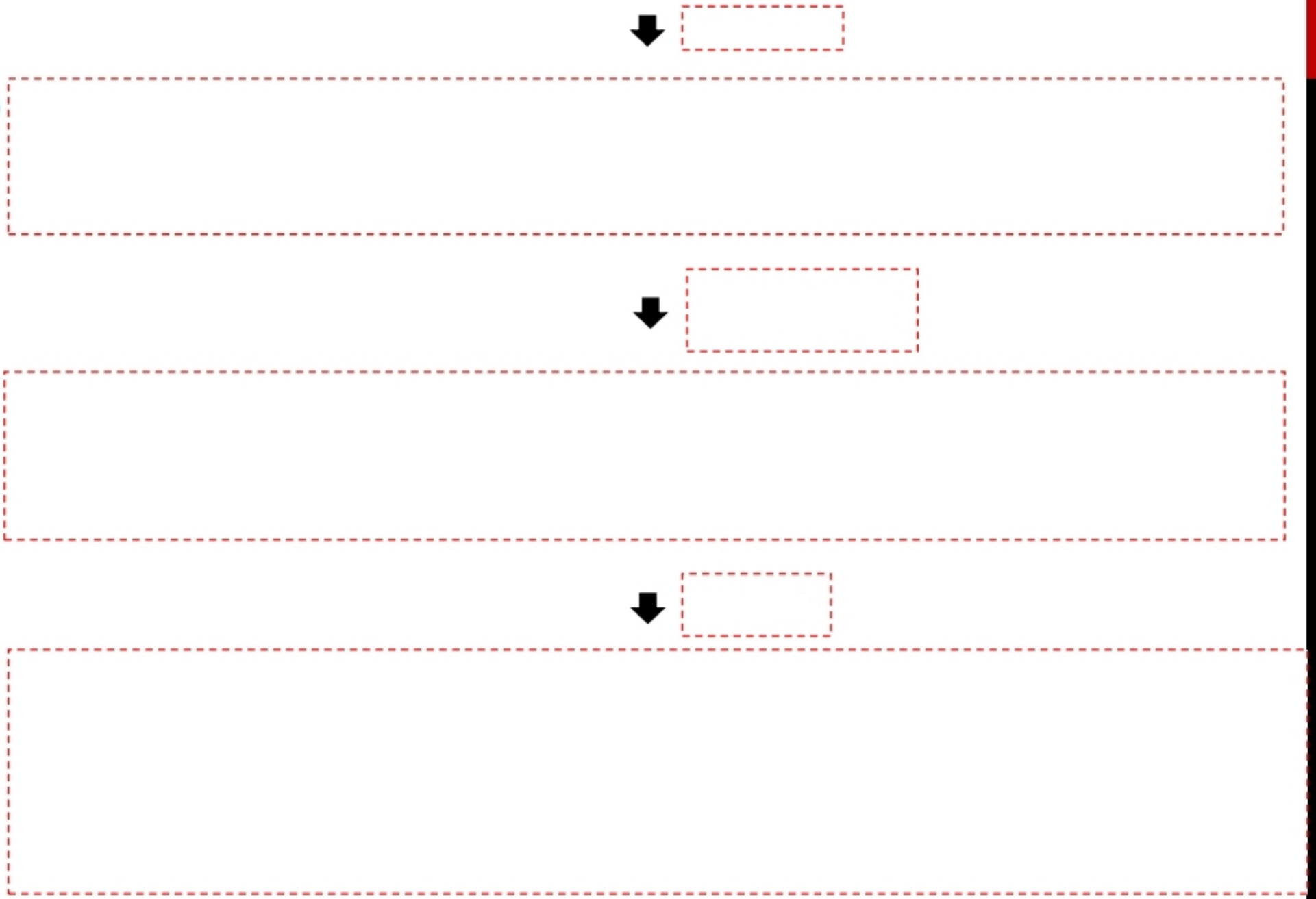
B+ Tree

- insert(B+tree tree, Key *k*)
- Example (5-way)
  - insert(tree, 3)



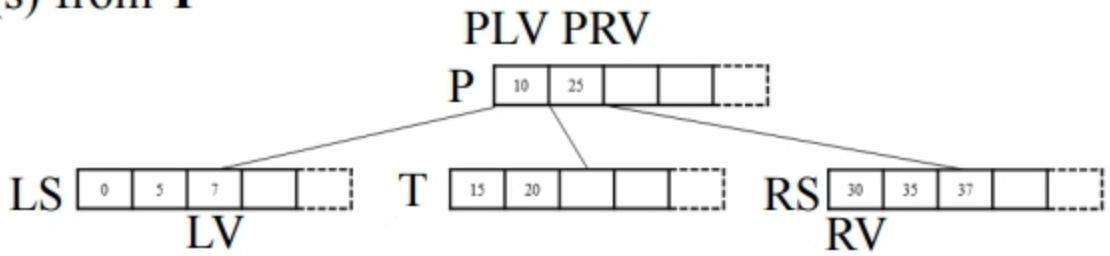
B+ Tree

- insert(B+tree tree, Key *k*)
- Example (5-way)
  - insert(tree, 3)



B+ Tree

- delete(B+tree tree, Key *k*)
  - Search a node, **T**, that *k* is included
  - IF (T is a leaf node)
    - Remove *k* and update parent key(s) from **T**
    - Prepare the variables



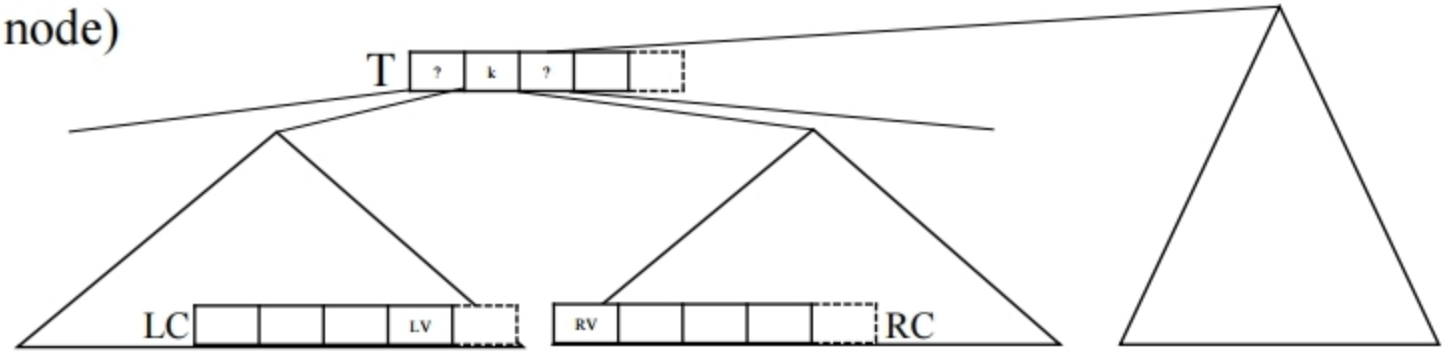
6 way: >=2
5 way : >=2
4 way: >=1

  - IF T violates the min keys prop ($|T| < \left\lceil \frac{m}{2} \right\rceil - 1$)
    - IF $|L| >= \left\lceil \frac{m}{2} \right\rceil$, **Reorganize**: reorganize LV, PLV, T
    - ELSE IF $|R| >= \left\lceil \frac{m}{2} \right\rceil$, **Reorganize**: reorganize RV, PRV, T
    - ELSE (i.e., P violates the min keys prop ($|P| < \left\lceil \frac{m}{2} \right\rceil - 1$))
      - **Merge:** merge LS, PLV, T and be a left child of PRV **or**
      - **Merge:** merge T, PRV, RS and be a right child of PLV
      - T = P and GO TO the arrow
  - ELSE (T is an internal node)
    - …

- Sorted keys, Leaf nodes in same level
- Children
  - Max: m children
  - Min: 0 for leaf, 2 for (non-leaf) root $\left\lceil \frac{m}{2} \right\rceil$ for internal nodes
- Keys
  - Max: (m-1) keys
  - Min: 1 for root, $\left\lceil \frac{m}{2} \right\rceil - 1$ for the others

B+ Tree

- delete(B+tree tree, Key *k*)
  - Search a node, **T**, that *k* is included
  - IF (T is a leaf node)
    - …
  - ELSE (T is an internal node)



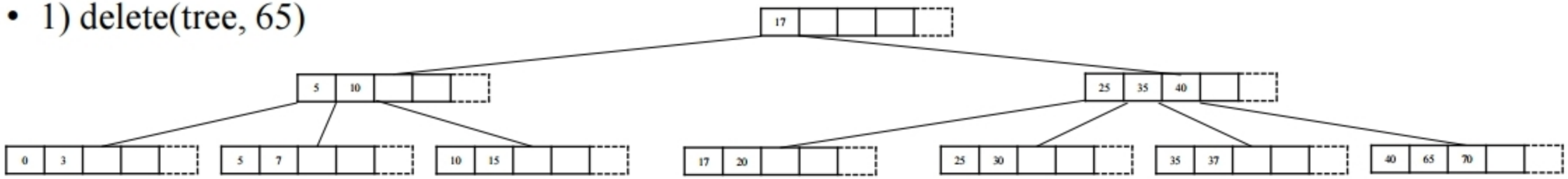
    - Remove *k* from **T**
      - IF |LC| >= $\left\lceil \frac{m}{2} \right\rceil$, **Borrow**: Locate LV to *k*
      - ELSE IF |RC| >= $\left\lceil \frac{m}{2} \right\rceil$, **Borrow**: Locate RV to *k*
      - ELSE (i.e., the min keys prop (|T| < $\left\lceil \frac{m}{2} \right\rceil - 1$))
        - Borrow: Locate LV to *k*
        - T = LC
        - GO TO the arrow

OR

Borrow: Locate RV to *k*
T = RC
GO TO the arrow

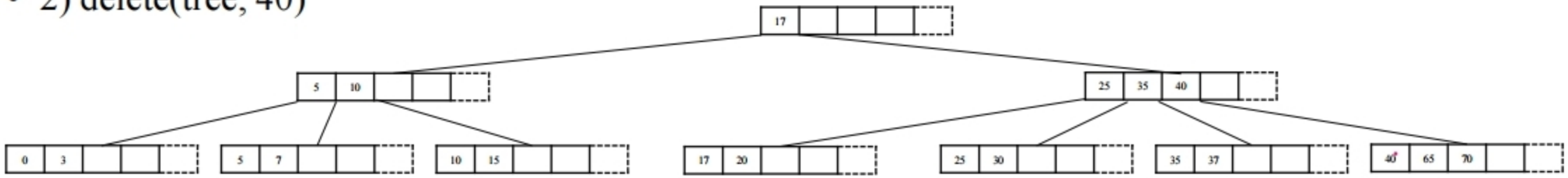
B+ Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 1) delete(tree, 65)



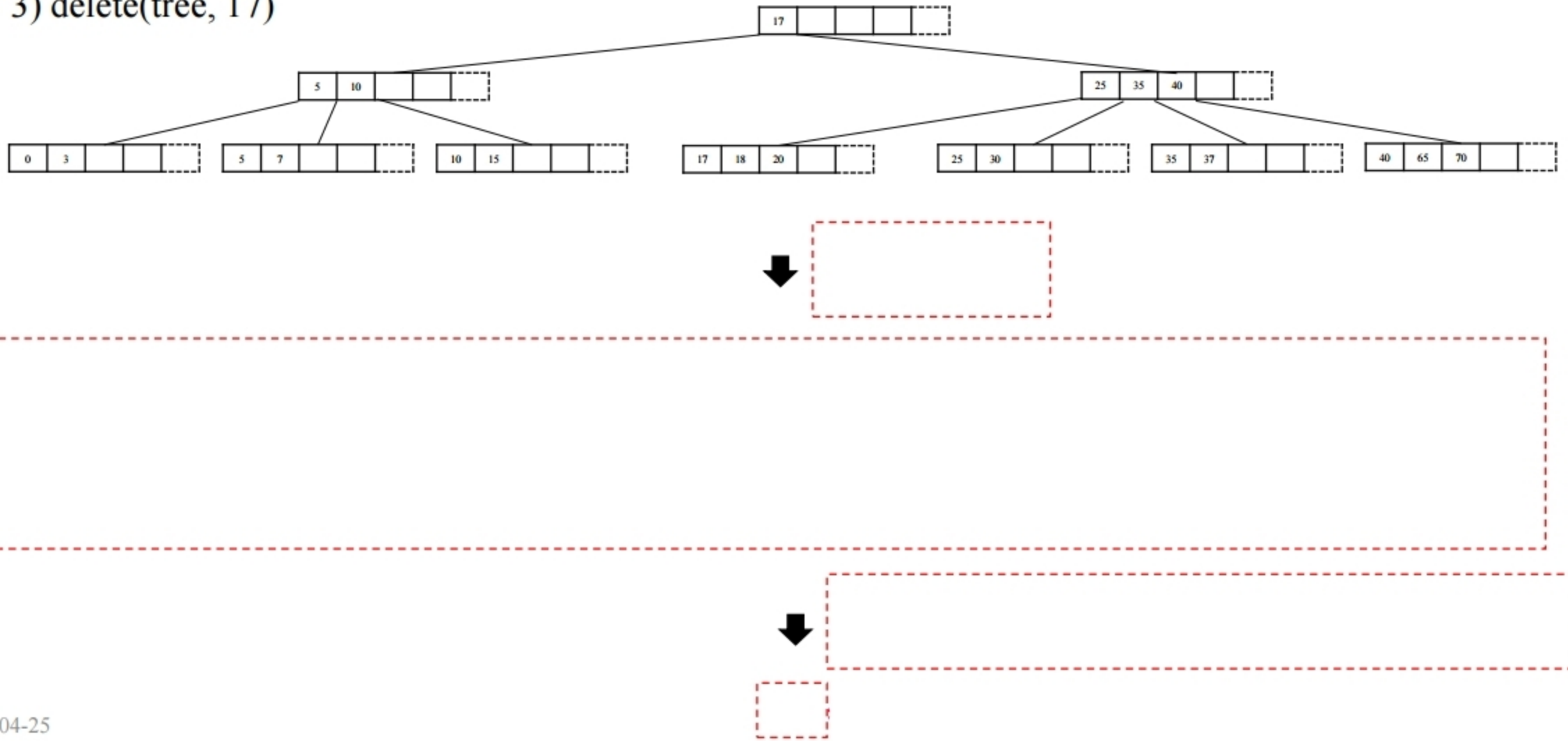
B+ Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 2) delete(tree, 40)



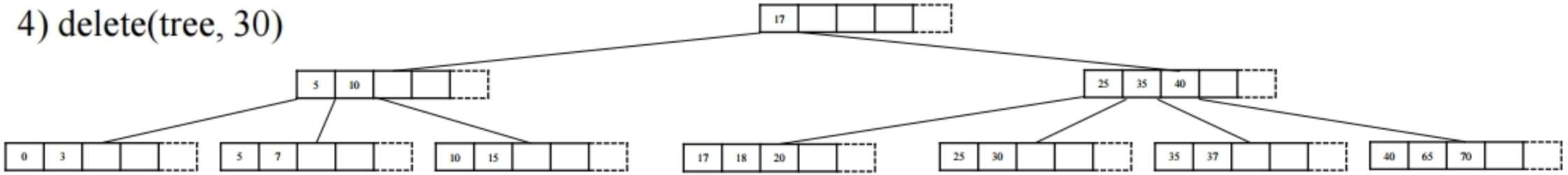
B+ Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 3) delete(tree, 17)



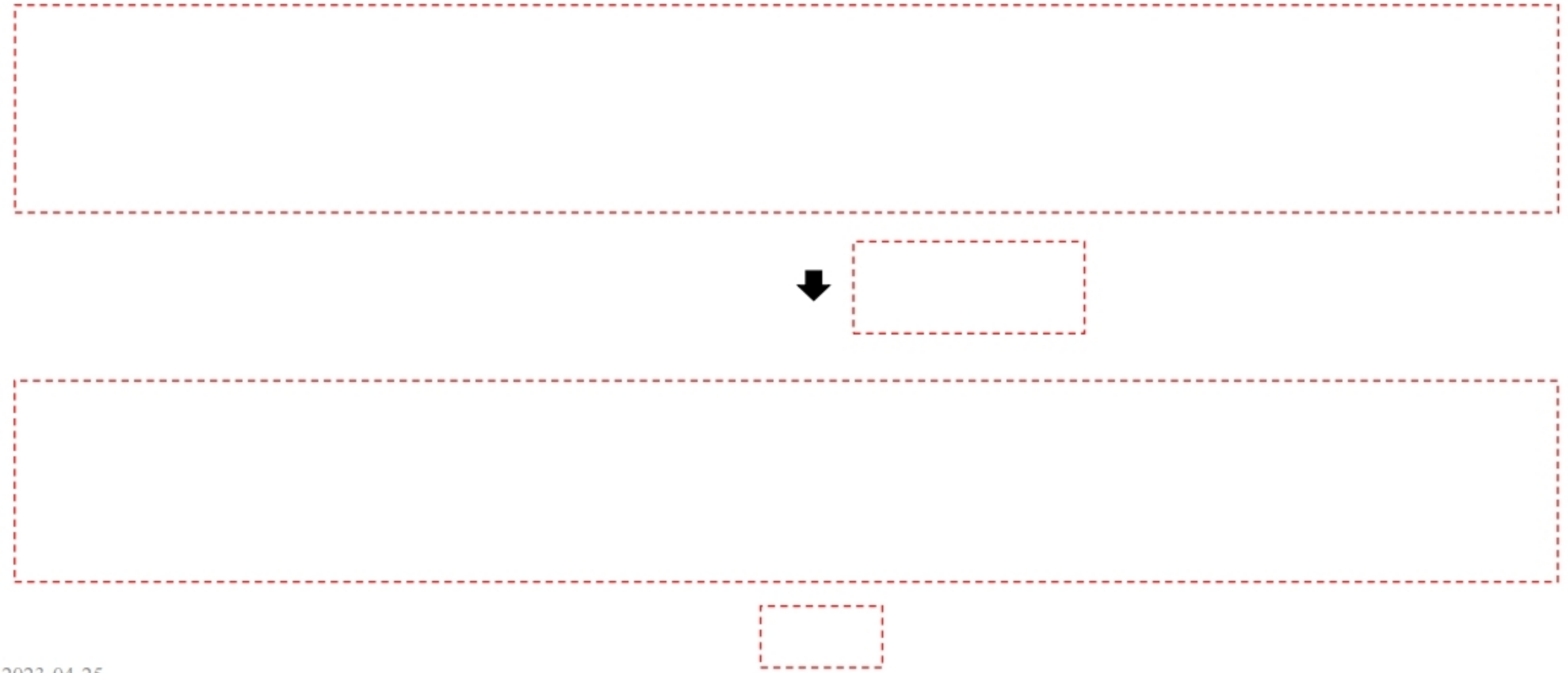
B+ Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 4) delete(tree, 30)



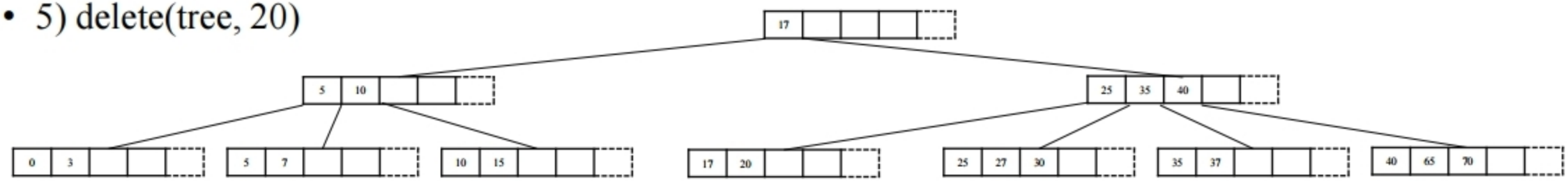
B+ Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 4) delete(tree, 30)



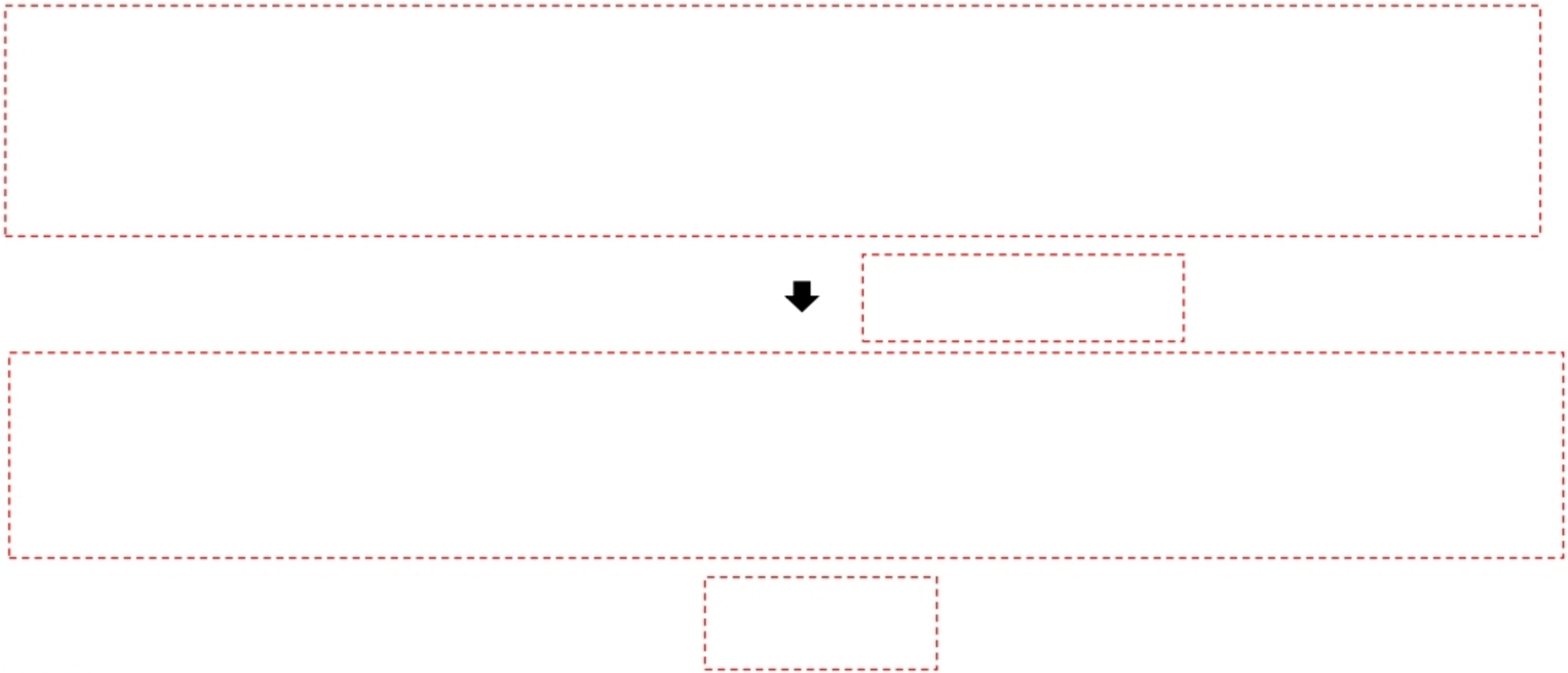
B+ Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 5) delete(tree, 20)



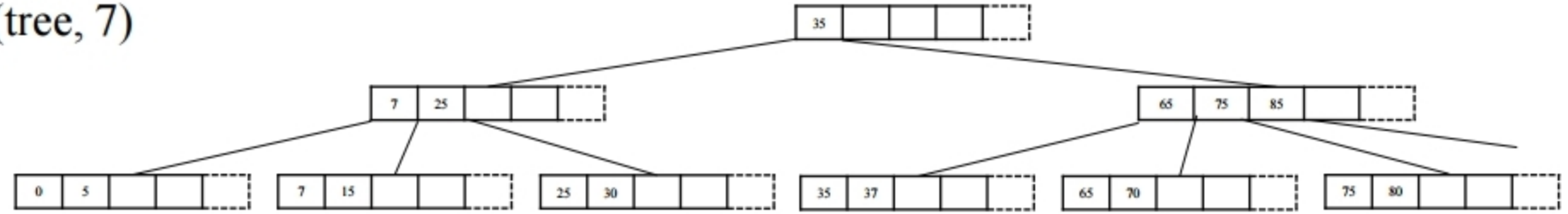
B+ Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 5) delete(tree, 20)



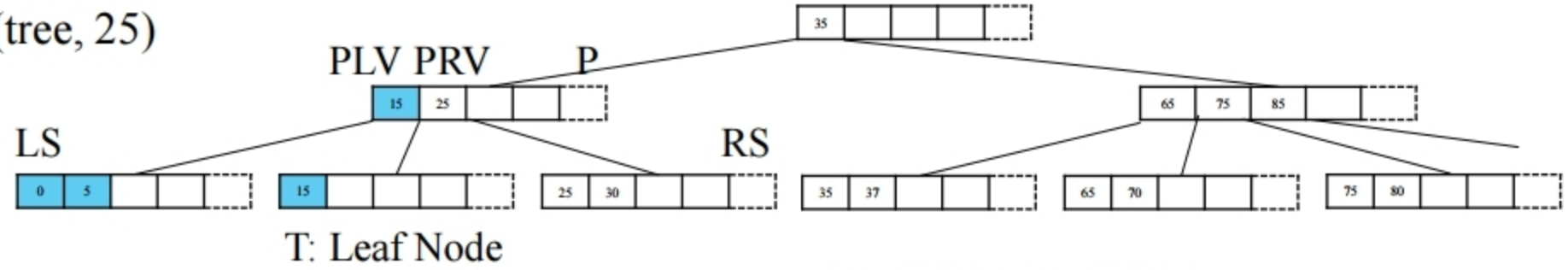
B+ Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 4) delete(tree, 7)



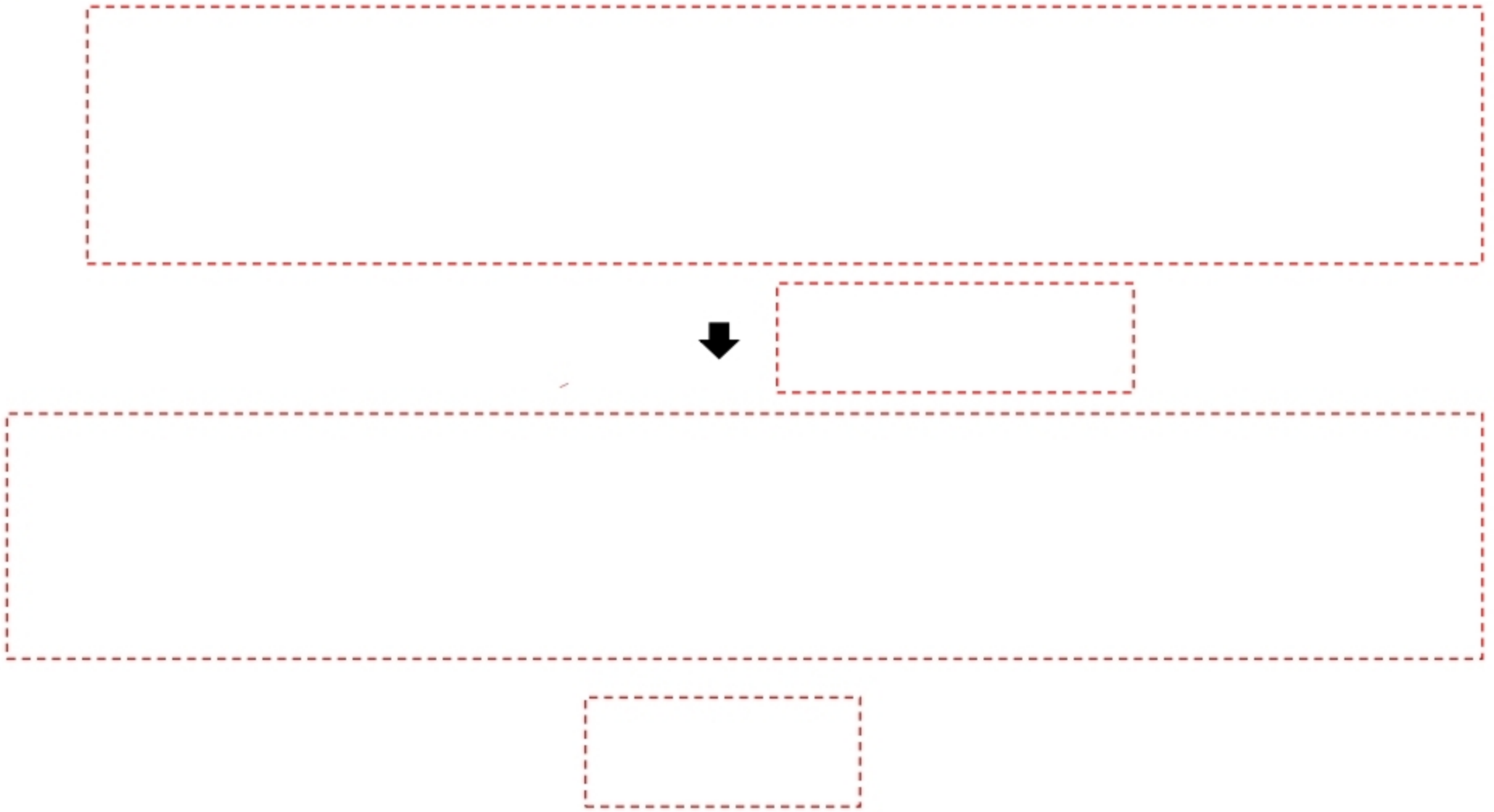
B+ Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 4) delete(tree, 25)



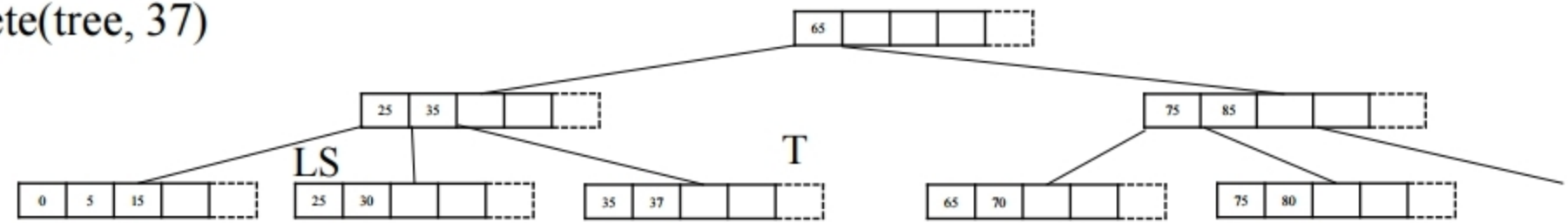
B+ Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 4) delete(tree, 25)



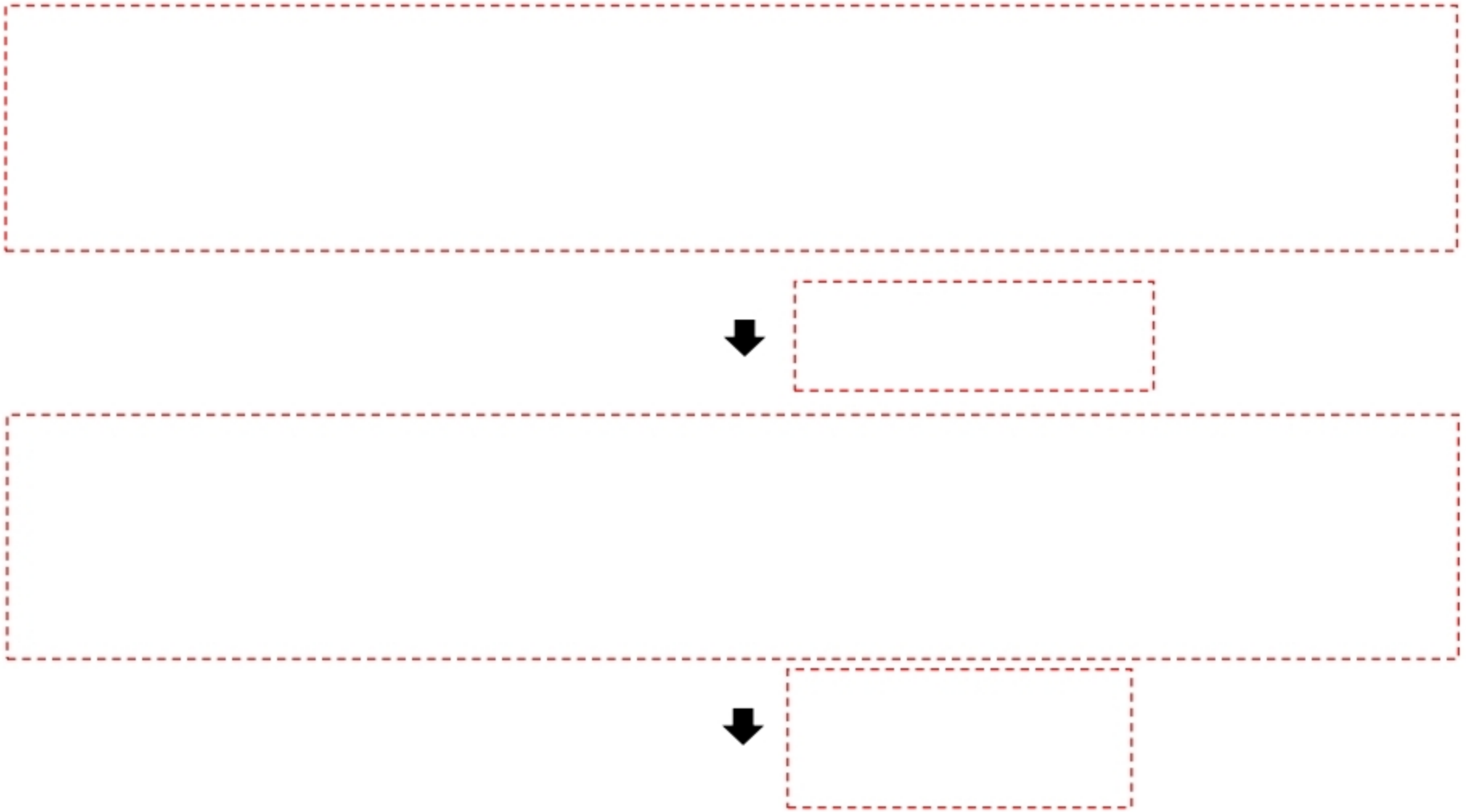
B+ Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 5) delete(tree, 37)



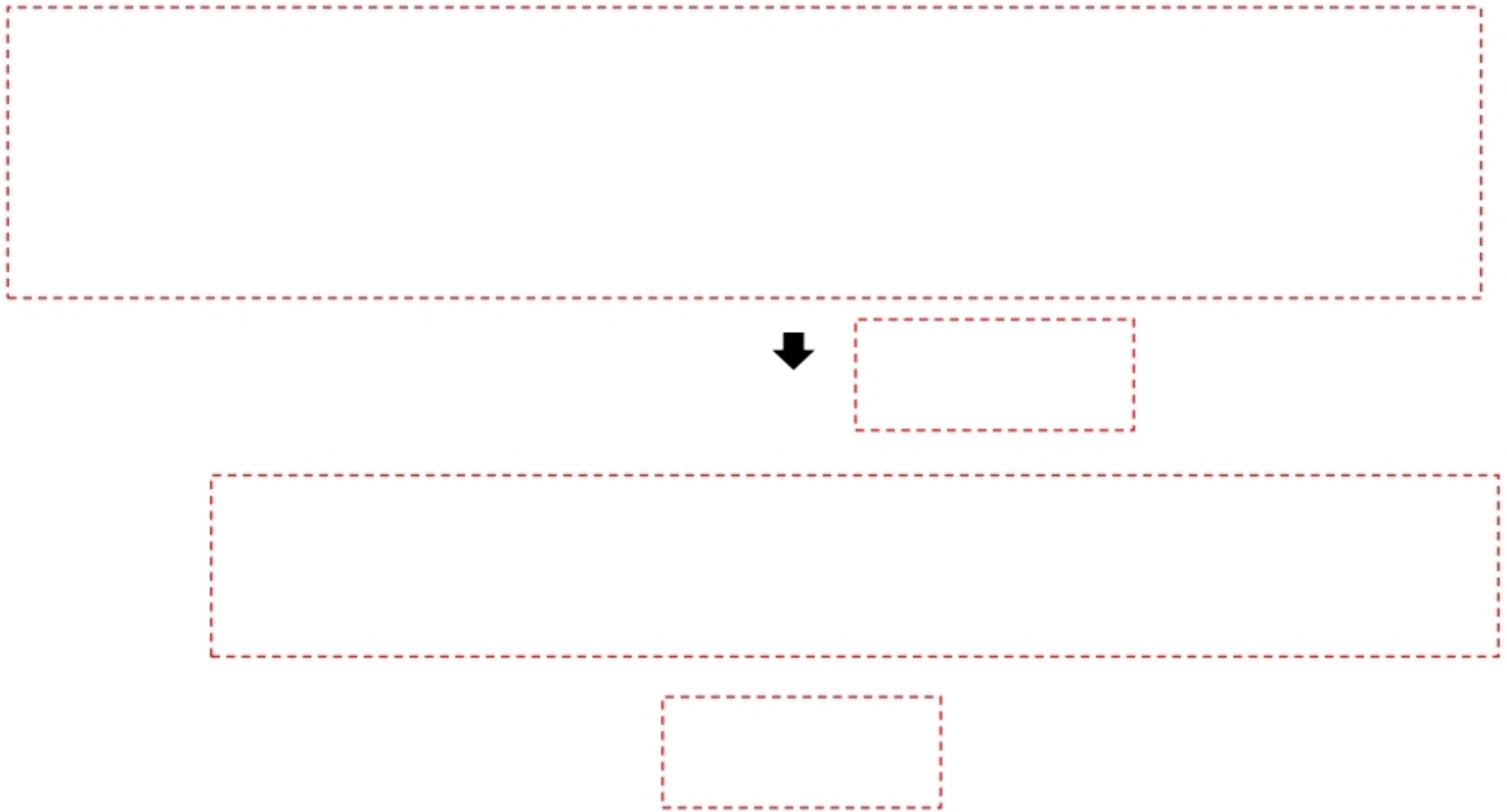
B+ Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 5) delete(tree, 37)



B+ Tree

- delete(Btree tree, Key *k*)
- Example (5-way)
  - 5) delete(tree, 37)



B-Tree vs. B+Tree

- B-Tree
  - Non-redundant keys
  - Spending less space
  - Search time depends on key
  - Deletion is more difficult than B+Tree
- B+Tree
  - All the data is in leaf nodes
  - Redundant keys
  - Search time requires traversals to leaf nodes in same level
  - Deletion is simpler than B-Tree
  - Database developer usually prefers to the simplicity of B+Tree

Create indexes in MariaDB

- CREATE INDEX
  - mapped to an ALTER TABLE with indexes

account2 **relation**

account_number	balance
A-101	500
A-102	400
A-201	900
A-215	700
A-217	750
A-222	700
A-305	350

branch2 **relation**

branch_name	branch_city	assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

account_branch **relation**

account_number	branch_name
A-101	Downtown
A-102	Perryridge
A-201	Brighton
A-215	Mianus
A-217	Brighton
A-222	Redwood
A-305	Round Hill

Syntax

```
CREATE [OR REPLACE] [UNIQUE|FULLTEXT|SPATIAL] INDEX
  [IF NOT EXISTS] index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [WAIT n | NOWAIT]
    [index_option]
    [algorithm_option | lock_option] ...

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH | RTREE}

index_option:
    [ KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | CLUSTERING={YES| NO} ]
  [ IGNORED | NOT IGNORED ]

algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY|NOCOPY|INSTANT}

lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

Create indexes in MariaDB

- CREATE INDEX
  - mapped to an ALTER TABLE with indexes

account2 relation

account_number	balance
A-101	500
A-102	400
A-201	900
A-215	700
A-217	750
A-222	700
A-305	350

CREATE INDEX idx_account_number ON account2 (account_number);
CREATE INDEX idx_balance ON account2 (balance);

Syntax

+2023-1 DB

```
CREATE [OR REPLACE] [UNIQUE|FULLTEXT|SPATIAL] INDEX
  [IF NOT EXISTS] index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [WAIT n | NOWAIT]
    [index_option]
    [algorithm_option | lock_option] ...

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH | RTREE}

index_option:
    [ KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | CLUSTERING={YES| NO} ]
  [ IGNORED | NOT IGNORED ]

algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY|NOCOPY|INSTANT}

lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

Create indexes in MariaDB

- SHOW INDEX
  - https://mariadb.com/kb/en/show-index/

account2 relation

account_number	balance
A-101	500
A-102	400
A-201	900
A-215	700
A-217	750
A-222	700
A-305	350

Syntax

```
SHOW {INDEX | INDEXES | KEYS}
 FROM tbl_name [FROM db_name]
 [WHERE expr]
```

SHOW INDEXES FROM account2;

```
MariaDB [db]> SHOW INDEXES FROM account2;
+----------+------------+--------------------+--------------+----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| Table    | Non_unique | Key_name           | Seq_in_index | Column_name    | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Ignored |
+----------+------------+--------------------+--------------+----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
| account2 |          1 | idx_account_number |            1 | account_number | A         |           7 |     NULL | NULL   | YES  | BTREE      |         |               | NO      |
| account2 |          1 | idx_balance        |            1 | balance        | A         |           7 |     NULL | NULL   | YES  | BTREE      |         |               | NO      |
+----------+------------+--------------------+--------------+----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+
2 rows in set (0.011 sec)
```

Create indexes in MariaDB

- CREATE INDEX
  - multiple columns (Composite Index)

```
CREATE OR REPLACE TABLE idx_test (n INTEGER, a VARCHAR(50));
INSERT INTO idx_test VALUES (1,'a'),(1,'c'),(2,'b'),(3,'b'),(3,'d'),(1,'b'),(2,'a'),(3,'a'),(3,'c');
```

```
SELECT * FROM idx_test WHERE  n = 1 AND ( a > 'a' AND a < 'd');
```

idx_test

n	a
1	a
1	c
2	b
3	b
3	d
1	b
2	a
3	a
3	c

CREATE INDEX idx_n_a ON idx_test(n,a);

n (1)	a (2)
1	a
1	b
1	c
2	a
2	b
3	a
3	b
3	c
3	d

CREATE INDEX idx_a_n ON idx_test(a,n);

a (1)	n (2)
a	1
a	2
a	3
b	1
b	2
b	3
c	1
c	3
d	3

Create indexes in MariaDB

- DROP INDEX
  - https://mariadb.com/kb/en/drop-index/

Syntax

```
DROP INDEX [IF EXISTS] index_name ON tbl_name
    [WAIT n |NOWAIT]
```

- CREATE OR REPLACE TABLE account2 (account_number VARCHAR(50), balance INTEGER);
- INSERT INTO account2 VALUES ('A-101',500),('A-102', 400), ('A-201',900), ('A-215', 700), ('A-217', 750), ('A-222', 700), ('A-305', 350);
- DESCRIBE account2;
- CREATE INDEX idx_a ON account2 (account_number);

```
MariaDB [db]> SHOW INDEXES FROM account2;
+----------+------------+----------+--------------+----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table    | Non_unique | Key_name | Seq_in_index | Column_name    | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+----------+------------+----------+--------------+----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| account2 |          1 | idx_a    |            1 | account_number | A         |           7 |     NULL | NULL   | YES  | BTREE      |         |               |
+----------+------------+----------+--------------+----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
1 row in set (0.006 sec)
```

- DROP INDEX idx_a ON account2;

```
MariaDB [db]> SHOW INDEXES FROM account2;
Empty set (0.007 sec)
```

Create indexes in MariaDB

- CREATE INDEX
  - Storage engine index type
    - BTREE: B+Tree implementation

https://blog.jcole.us/2013/01/10/btree-index-structures-in-innodb/

    - HASH
    - RTREE: a tree data structure used for spatial access method
  - Storage engines and their supported index types

Storage Engine	Permitted Indexes
Aria	**BTREE**, RTREE
MyISAM	**BTREE**, RTREE
InnoDB	**BTREE**
MEMORY/HEAP	HASH, **BTREE**

CREATE TABLE account2 …. ENGINE=InnoDB;
CREATE INDEX idx_a ON account2 (account_number) USING BTREE;

Index_type
BTREE

Syntax

```
CREATE [OR REPLACE] [UNIQUE|FULLTEXT|SPATIAL] INDEX
  [IF NOT EXISTS] index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [WAIT n | NOWAIT]
    [index_option]
    [algorithm_option | lock_option] ...

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH | RTREE}

index_option:
    [ KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | CLUSTERING={YES| NO} ]
  [ IGNORED | NOT IGNORED ]

algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY|NOCOPY|INSTANT}

lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

Create indexes in MariaDB

- CREATE INDEX
  - Index type
    - Plain indexes
      - Do not necessarily need to be unique
      - Practiced so far…
      - CREATE INDEX *plain_index* ON *tbl_name* (*col*,…)
    - Unique indexes
      - Must be unique
      - CREATE UNIQUE INDEX *plain_index* ON *tbl_name* (*col*,…)
    - Primary key
      - Must be unique and not-null
      - With PRIMARY KEY
      - CREATE TABLE tbl_name (pk, col1,…, PRIMARY KEY(pk));
    - Full-text indexes
      - Support full-text indexing and searching
      - CREATE FULLTEXT INDEX *idx_name* ON *tbl_name* (*col*);
    - Spatial indexes
      - Support spatial access method
      - CREATE SPATIAL INDEX *idx_name* ON *tbl_name* (*col*);

Syntax

```
CREATE [OR REPLACE] [UNIQUE|FULLTEXT|SPATIAL] INDEX
  [IF NOT EXISTS] index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [WAIT n | NOWAIT]
    [index_option]
    [algorithm_option | lock_option] ...

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH | RTREE}

index_option:
    [ KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | CLUSTERING={YES| NO} ]
  [ IGNORED | NOT IGNORED ]

algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY|NOCOPY|INSTANT}

lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

Create indexes in MariaDB

- CREATE INDEX
  - Index type
    - Plain indexes
    - Unique indexes
      - Must be unique
      - CREATE UNIQUE INDEX *plain_index* ON *tbl_name* (*col*,...)

INSERT INTO uidx VALUES (1,1),(1,2),(2,1),(2,2);

A	B
1	1
1	2
2	1
2	2

CREATE UNIQUE INDEX u1 ON uidx (a);
→ Duplicate entry ‘1’ for key ‘u1’
CREATE UNIQUE INDEX u1 ON uidx (b);
→ Duplicate entry ‘1’ for key ‘u1’
CREATE UNIQUE INDEX u1 ON uidx (a,b);
→ SUCCESS
INSERT INTO uidx VALUES (1,1);
→ Duplicate entry '1-1' for key 'u1'

```
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| uidx  |          0 | u1       |            1 | a           | A         |           3 |     NULL | NULL   | YES  | BTREE      |         |               |
| uidx  |          0 | u1       |            2 | b           | A         |           3 |     NULL | NULL   | YES  | BTREE      |         |               |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
```

Create indexes in MariaDB

- CREATE INDEX
  - Index type
    - Primary key
      - Must be unique and not-null
      - With PRIMARY KEY
      - CREATE TABLE tbl_name (pk, col1,…, PRIMARY KEY(pk));

CREATE OR REPLACE TABLE account3 (account_number VARCHAR(50), balance INTEGER, PRIMARY KEY (account_number));
INSERT INTO account3 VALUES ('A-101',500),('A-102', 400), ('A-201',900), ('A-215', 700), ('A-217', 750), ('A-222', 700), ('A-305', 350);
DESCRIBE account2;
SHOW INDEXES FROM account2;

```
MariaDB [db]> SHOW INDEXES FROM account2;
+----------+------------+----------+--------------+----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table    | Non_unique | Key_name | Seq_in_index | Column_name    | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+----------+------------+----------+--------------+----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| account2 |          0 | PRIMARY  |            1 | account_number | A         |           7 |     NULL | NULL   |      | BTREE      |         |               |
+----------+------------+----------+--------------+----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
1 row in set (0.001 sec)
```

Create indexes in MariaDB

- CREATE INDEX
  - Index type
    - Foreign key

account2 **relation**

account_number	balance
A-101	500
A-102	400
A-201	900
A-215	700
A-217	750
A-222	700
A-305	350

branch2 **relation**

branch_name	branch_city	assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

account_branch **relation**

account_number	branch_name
A-101	Downtown
A-102	Perryridge
A-201	Brighton
A-215	Mianus
A-217	Brighton
A-222	Redwood
A-305	Round Hill

```
+----------------+------------+----------------+--------------+----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table          | Non_unique | Key_name       | Seq_in_index | Column_name    | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+----------------+------------+----------------+--------------+----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| account_branch |          1 | account_number |            1 | account_number | A         |           7 |     NULL | NULL   | YES  | BTREE      |         |               |
| account_branch |          1 | branch_name    |            1 | branch_name    | A         |           7 |     NULL | NULL   | YES  | BTREE      |         |               |
+----------------+------------+----------------+--------------+----------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
```

Create indexes in MariaDB

- Building the best INDEX for a given SELECT
  - https://mariadb.com/kb/en/building-the-best-index-for-a-given-select/
  - A cookbook on how to build the best INDEX
  - Four patterns
    - EQ: Equality filtering with constant values
      - Case
        - col = value
        - col is NULL
      - Not a case
        - DATE(dt) = …
        - LOWER(s) = …
        - CAST(s…) = …
        - Not a column of current table
    - RQ: Range filtering
      - Case
        - Col >= value, Col > value, Col < value, Col <= value
        - Col BETWEEN value1 AND value2
        - Col LIKE 'brah%'
      - Not a case
        - Col <> value, Col != value
        - Col LIKE '%blah'
    - G: GROUP BY
      - All the columns for GROUP BY should be added
    - O: ORDER BY
      - All the columns for ORDER BY should be added
      - unless it is not a composite index of a mixture of DESC, ASC or not with RQ or expression group by

Define some method
EQ(Q): Equality filtering on column Q
RQ(Q): Range filtering on column Q
G(Q,…): GROUP BY column Q,…
O(Q,…): ORDER BY column Q,…

INDEX(A,B,…) : recommended index

Create indexes in MariaDB

- Building the best INDEX for a given SELECT
  - https://mariadb.com/kb/en/building-the-best-index-for-a-given-select/
  - A cookbook on how to build the best INDEX

- Rules

1.	EQ(Q)	→	INDEX(Q)
2.	EQ(Q1) & EQ(Q2)	→	INDEX(Q1,Q2) or INDEX(Q2,Q1)
3.	SELECT * FROM t1 WHERE EQ(t1.Q1) & EQ(t2.Q2)	→	INDEX(Q1)
4.	RQ(Q2) & EQ(Q1)	→	INDEX(Q1,Q2)
5.	RQ(Q1)	→	INDEX(Q1)
6.	RQ(Q1) & RQ(Q2)	→	INDEX(Q1) or INDEX(Q2)
7.	RQ(Q1) O(Q1)	→	INDEX(Q1)
8.	EQ(Q1) & EQ(Q2) G(Q3)	→	INDEX(Q1,Q2,Q3) or INDEX (Q2,Q1,Q3)
9.	RQ(Q1) G(Q2)	→	INDEX(Q1)
10.	G(Q1,Q2)	→	INDEX(Q1,Q2)
11.	EQ(Q1) G(Q2, (Q3+Q4))	→	INDEX(Q1)
12.	EQ(Q1) G(Q2) O(Q3)	→	INDEX(Q1,Q2)
13.	EQ(Q1) G(Q2) O(Q2)	→	INDEX(Q1, Q2)
14.	EQ(Q1) O(Q2 ASC, Q3 DESC)	→	INDEX(Q1)

Create indexes in MariaDB

- Building the best INDEX for a given SELECT
  - https://mariadb.com/kb/en/building-the-best-index-for-a-given-select/
  - A cookbook on how to build the best INDEX

- Considerations
  1. Flags and low cardinality
     - Is almost never useful (e.g., '1' occurs more than 20% of a table)
  2. Covering indexes
     - An index that contains all the columns in the SELECT (projection)
     - E.g.,
       - SELECT x FROM t WHERE y = 5; → INDEX(y,x)
       - SELECT x,z FROM t WHERE y = 5 AND q = 7; → INDEX(y,q,x,z)
       - SELECT x FROM t WHERE y > 5 AND q > 7; → INDEX(y,q,x) or INDEX(q,y,x)
     - Note
       - Not wise with lots of columns (5, Rule of Thumb)
       - See the page

Create indexes in MariaDB

- Building the best INDEX for a given SELECT
  - https://mariadb.com/kb/en/building-the-best-index-for-a-given-select/
  - A cookbook on how to build the best INDEX

- Considerations
  3. WHERE a = 1 OR a =2 → WHERE a IN (1,2)
  4. Dates
     - Not good
       - date_col LIKE '2016-01%' because start converting date_col to string
     - Better
       - date_col >= '2016-01-01'
       - date_col < '2016-01-01' + INTERVAL 3 MONTH

Create indexes in MariaDB

- Considerations
  5. USE EXPLAIN
     - https://mariadb.com/kb/en/explain/

```
MariaDB [db]> EXPLAIN SELECT * FROM account2 WHERE account_number = 'A-101';
+------+-------------+----------+-------+---------------+---------+---------+-------+------+-------+
| id   | select_type | table    | type  | possible_keys | key     | key_len | ref   | rows | Extra |
+------+-------------+----------+-------+---------------+---------+---------+-------+------+-------+
|    1 | SIMPLE      | account2 | const | PRIMARY,a     | PRIMARY | 52      | const | 1    |       |
+------+-------------+----------+-------+---------------+---------+---------+-------+------+-------+
1 row in set (0.009 sec)
```

Column name	Description
Id	Sequence number that shows in which order tables are joined
Select_type	What kind of SELECT the table comes from
Table	Alias name of table. Materialized temporary tables for sub queries are named <subquery#>
Type	How rows are found from the table (join type)
Possible_keys	Keys in table that could be used to find rows in the table
Key	The name of the key that is used to retrieve rows. NULL is no key was used
Key_len	How many bytes of the key that was used (shows if we are using only parts of the multi-column key).
Ref	The reference that is used as the key value
Rows	An estimate of how many rows we will find in the table for each key lookup.
Extra	Extra information about this join.

Wrap-up

- Introduction
- Hash (brief recap)
- Binary Tree (brief recap)
- B-Tree
- B+Tree
- Create indexes in MariaDB