Mod 5 - Lecture 11,12:

B Tree

- Deepak Poonia
(IISc Bangalore; *GATE AIR 53 & 67*)

Content of this Lecture:

1. B Tree COMPLETE Details

DBMS Complete Course Link:

https://www.goclasses.in/courses/Database-Management-Systems

Instructor:

Deepak Poonia

IISc Bangalore

GATE CSE AIR 53; AIR 67; AIR 206; AIR 256

DBMS Complete Course Link:

https://www.goclasses.in/courses/Database-Management-Systems

GO CLASSES

GATE 2024
COMPLETE COURSE CS-IT
(1 YEAR + 3 MONTHS)

GATE 2024 COMPLETE COURSE CS - IT

(1 YEAR + 3 MONTHS)

GO CLASSES

GATE 2023
COMPLETE COURSE CS-IT
(6 MONTHS)

GATE 2023 COMPLETE COURSE CS-IT

(6 MONTHS)

GO CLASSES

GATE 2025
COMPLETE COURSE CS-IT
(2 YEARS)

GATE 2025 COMPLETE COURSE CS - IT

(2 YEARS + 3 MONTHS)

2023 Discrete Mathematics

★ ★ ★ ★ ★ 5.0 (62 ratings)

Deepak Poonia (MTech IISc Bangal...

Free

2023 C Programming

★ ★ ★ ★ ★ 5.0 (59 ratings)

Sachin Mittal (MTech IISc Bangalor...

Free

2023 Discrete Mathematics

Learn, Understand, Discuss. "GO" for the Best.

★★★★★ 5.0 (62 ratings)

5997 Learners Enrolled

Language: English

Instructors: Deepak Poonia (MTech IISc Bangalore, GATE CSE AIR 53; 67)

FREE

On “GATE-Overflow ” Website

Join **GO+ GO Classes Combined Test Series** for BEST quality tests, matching GATE CSE Level:

Visit www.gateoverflow.in website to join Test Series.

1. **Quality Questions:** No Ambiguity in Questions, All Well-framed questions.
2. Correct, **Detailed Explanation**, Covering Variations of questions.
3. **Video Solutions.**

NOTE :

Complete Discrete Mathematics & Complete C-Programming

Courses, by GO Classes, are **FREE** for ALL learners.

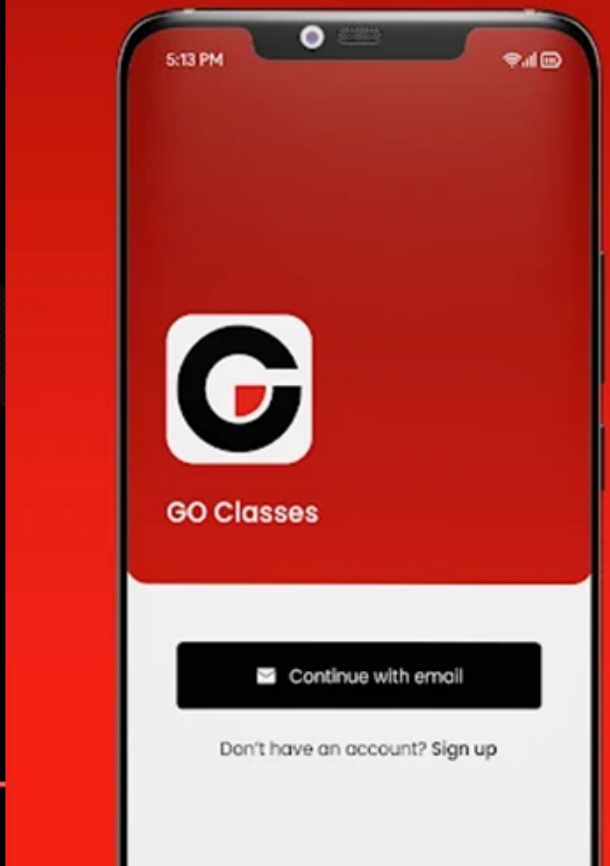
Visit here to watch : https://www.goclasses.in/s/store/

SignUp/Login on Goclasses website for free and start learning.

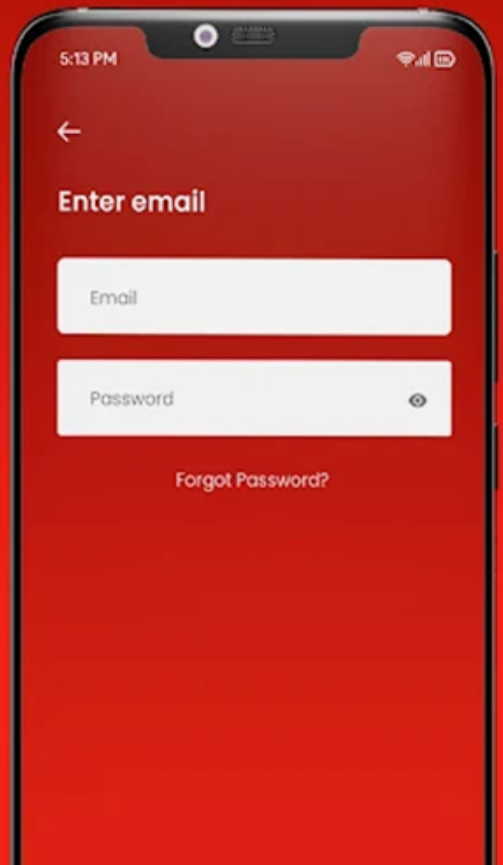
Download the GO Classes Android App:

https://play.google.com/store/apps/details?id=com.goclasses.courses

Search "GO Classes" on Play Store.

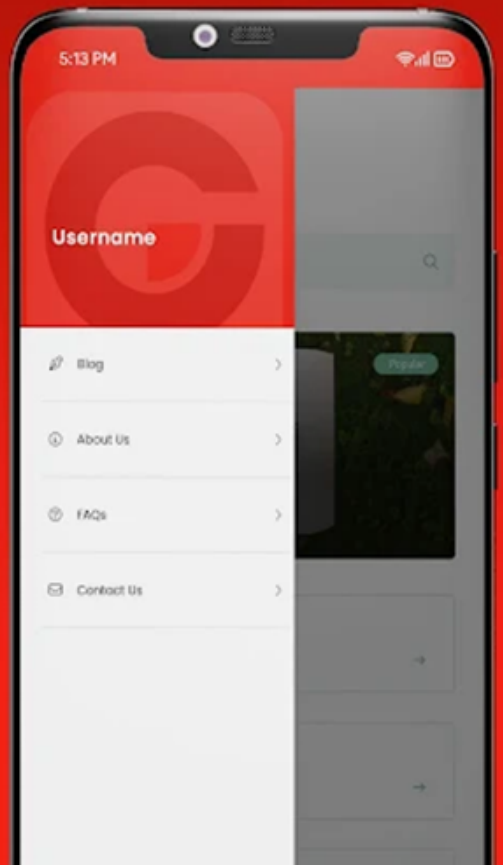


Welcome to
GO Classes
Learn at your own pace



Learn from
Top educators
Anytime, anywhere!

Easy to use
Navigation Panel
with all essentials



We are on Telegram. Contact Us for any help.

Join GO Classes Resources, Notes, Content, information Telegram Channel:

Public Username: GOCLASSES_CSE

Join GO Classes **Doubt Discussion** Telegram Group :

Username: GATECSE_Goclasses

(Any doubt related to Goclasses Courses can also be asked here.)

Join GATEOverflow Doubt Discussion Telegram Group :

Username: gateoverflow_cse

Recap:

Indexes So Far...

Index Structures that we have studied so far:

Primary Index
Clustering Index
Secondary Index
Multilevel Indexes
ISAM Files

⇒ Static Indexes

Modifying these indices is Costly... So, Useful when data in file is static i.e. rarely changes

So, for Dynamic file, these are Not good option.

Index Sequential Access Method (ISAM) Files

ISAM files –

Ordered files with a multilevel primary/clustering index

Most suitable for files that are relatively static

If the files are dynamic, we need to go for dynamic multi-level index structures based on B^+- trees

B-Tree
B+ Tree

Dynamic multilevel Tree Based Indexes

- Modification is less Costly.
- Modification friendly.

Next Topic:

Indexing

B, B+ Tree Indexes

Idea

Tree Structures that we have study for Main Memory:

BST
AVL Tree
Heap
2-3-4 Tree

These are in-memory Data Structures... NOT suitable for Disk Data Structures... WHY?

The in-memory Data Structures are NOT suitable for Disk Data Structures... WHY?

Ask yourself:

What should be a NODE in the Tree??

The in-memory Data Structures are NOT suitable for Disk Data Structures... WHY?

Ask yourself:

What should be a NODE in the Tree??

In main memory, a Node was a Structure, having few Bytes only, So, suitable for Memory as we access main memory in Words...

The in-memory Data Structures are NOT suitable for Disk Data Structures... WHY?

Ask yourself:

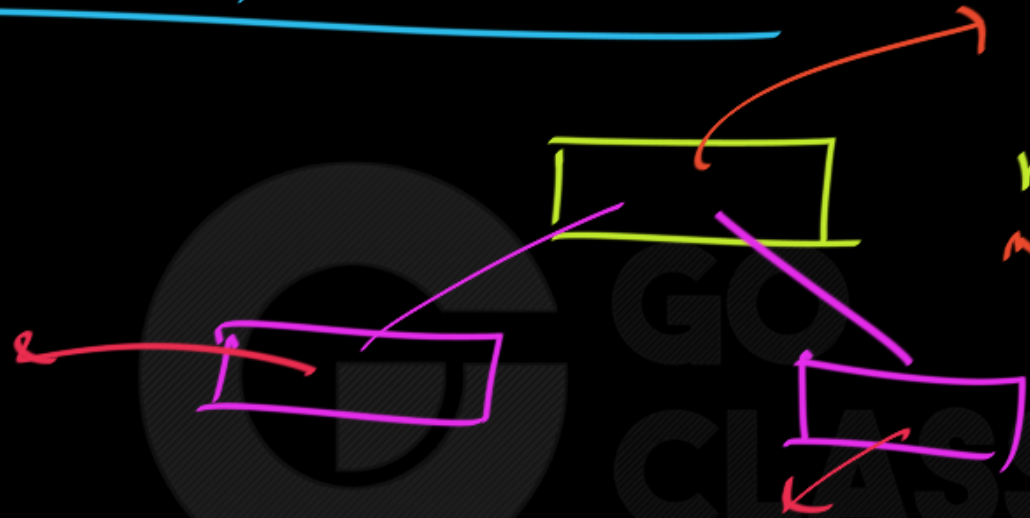
What should be a NODE in the Tree??

In main memory, a Node was a Structure, having few Bytes only, So, suitable for Memory as we access main memory in Words...

BUT Disk Access happens Block by Block & Block size is VERY Large than the main memory word size...

So, If we use in-memory data structures for Disk, A LOT of space per block will be Wasted...

AVL Tree for Disk:



Q:

Assume 1,000,000 records on disk.

Balanced

Binary Search Tree OR AVL Tree etc will take how many Disk Block Access??

Index

Q:

Assume 1,000,000 records on disk.

Balanced Binary Search Tree OR AVL Tree etc will take how many Disk Block Access??

Records in Data file = 1,000,000

if AVL Tree index (Dense index)

20 level

#Disk Access for a Record = Index 20 Block + 1 for Data Block

What should be a NODE in the Data Structure for Disk??

Disk Access happens Block by Block... So, We will want to take Block as a Node for Disk Data Structures...

& Block size is VERY Large (typically 4KB etc)...

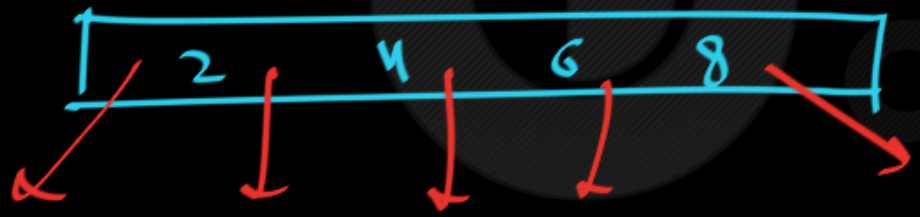
So, a LOT of Keys can be put in a single Block(Node)...

That's the core idea of Disk Data Structures.

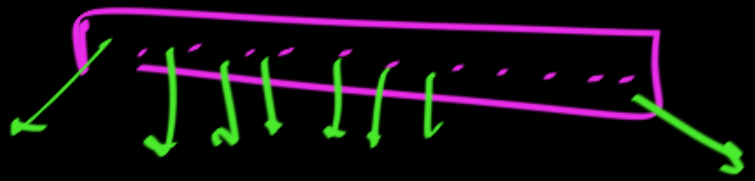
B - Tree
B+ Tree

→ put a lot of index entries in a simple node (Block)

→ Disk Search Tree Data Structure



Block



Next Topic:

Tree Terminology:

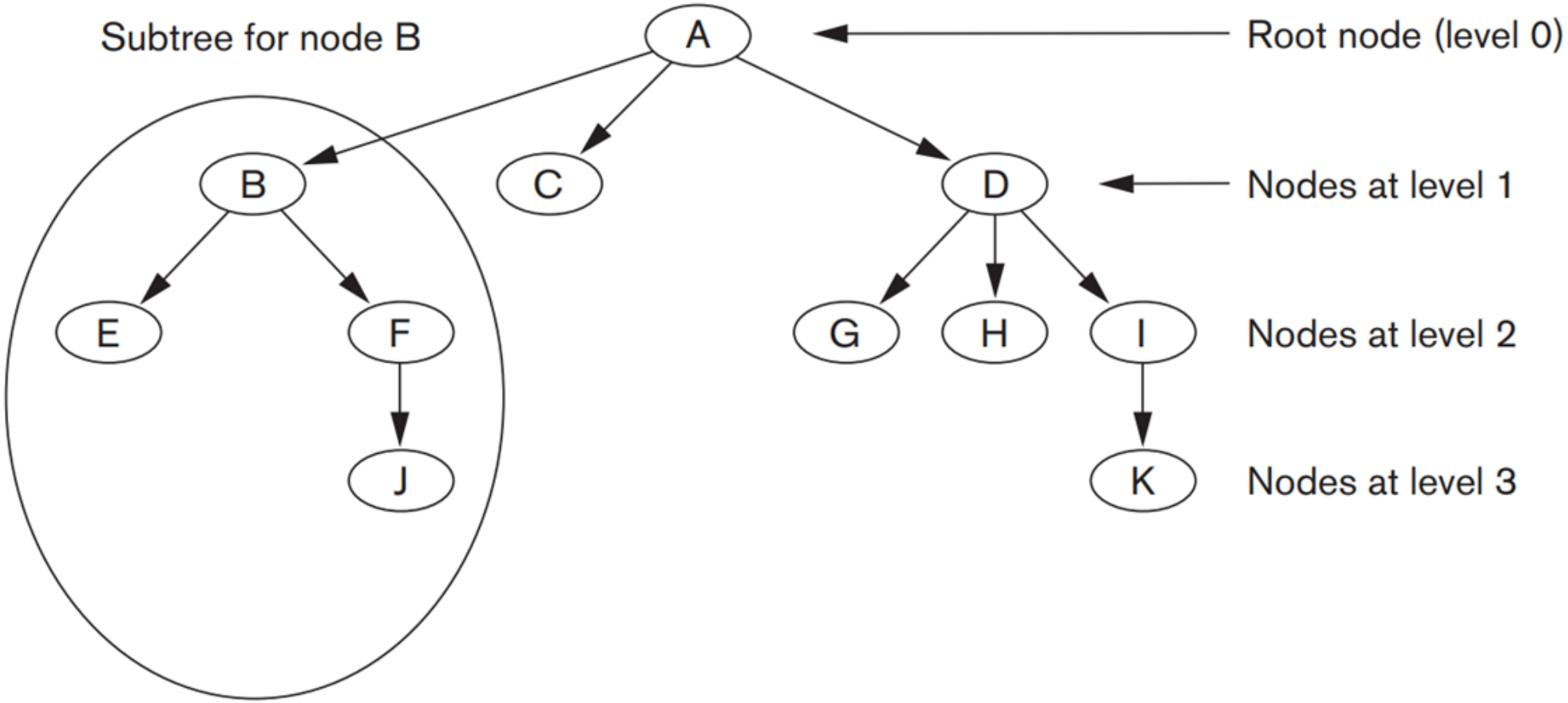
Recap

17.3 Dynamic Multilevel Indexes Using B-Trees and B+-Trees

B-trees and B+-trees are special cases of the well-known search data structure known as a **tree**. We briefly introduce the terminology used in discussing tree data structures. A **tree** is formed of **nodes**. Each node in the tree, except for a special node called the **root**, has one **parent** node and zero or more **child** nodes. The root node has no parent. A node that does not have any child nodes is called a **leaf** node; a nonleaf node is called an **internal** node. The **level** of a node is always one more than the level of its parent, with the level of the root node being *zero*.[7] A **subtree** of a node consists of that node and all its **descendant** nodes—its child nodes, the child nodes of its child nodes, and so on. A precise recursive definition of a subtree is that it consists of a node *n* and the subtrees of all the child nodes of *n*. Figure 17.7 illustrates a tree data structure. In this figure the root node is A, and its child nodes are B, C, and D. Nodes E, J, C, G, H, and K are leaf nodes. Since the leaf nodes are at different levels of the tree, this tree is called **unbalanced**.

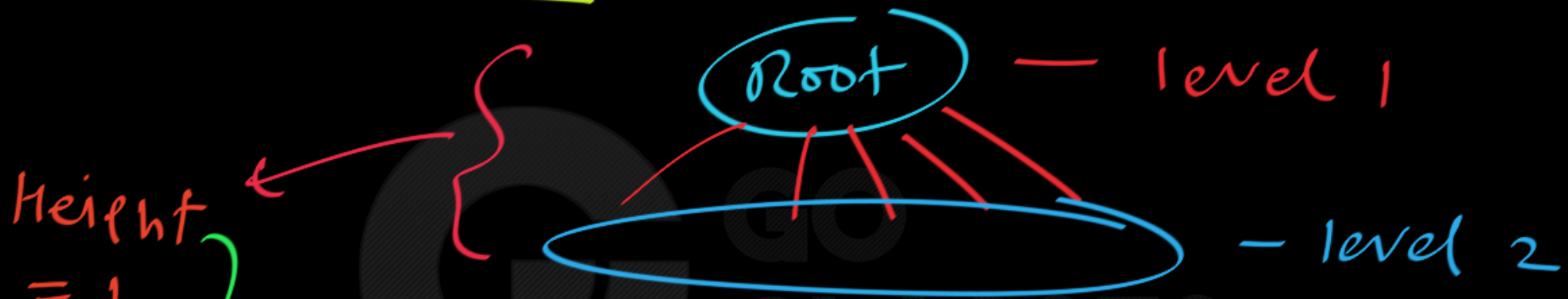
Figure 17.7

A tree data structure that shows an unbalanced tree.



(Nodes E, J, C, G, H, and K are leaf nodes of the tree)

Our Convention:



Height = 1

in terms of Edges from Root to leaf

Next Topic:

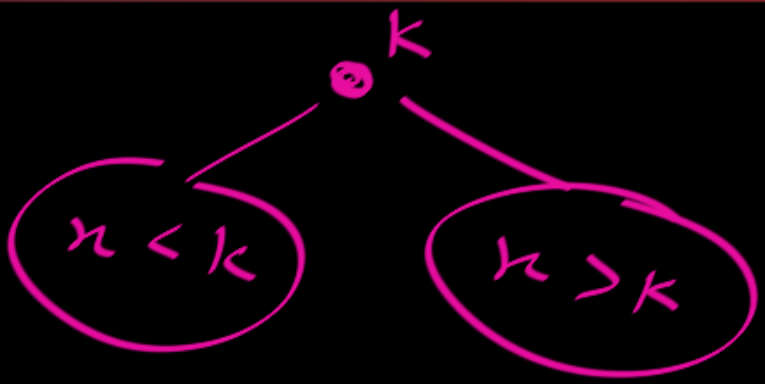
Search Tree Obvious Rules

Search Trees :

In-memory : BST, AVL, ~~Heap~~

Disk : B Tree

B+ "



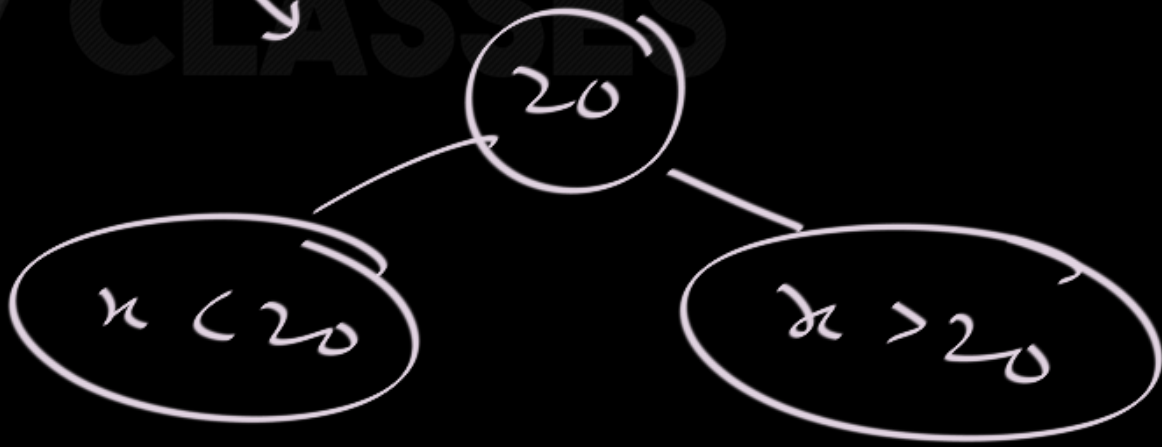
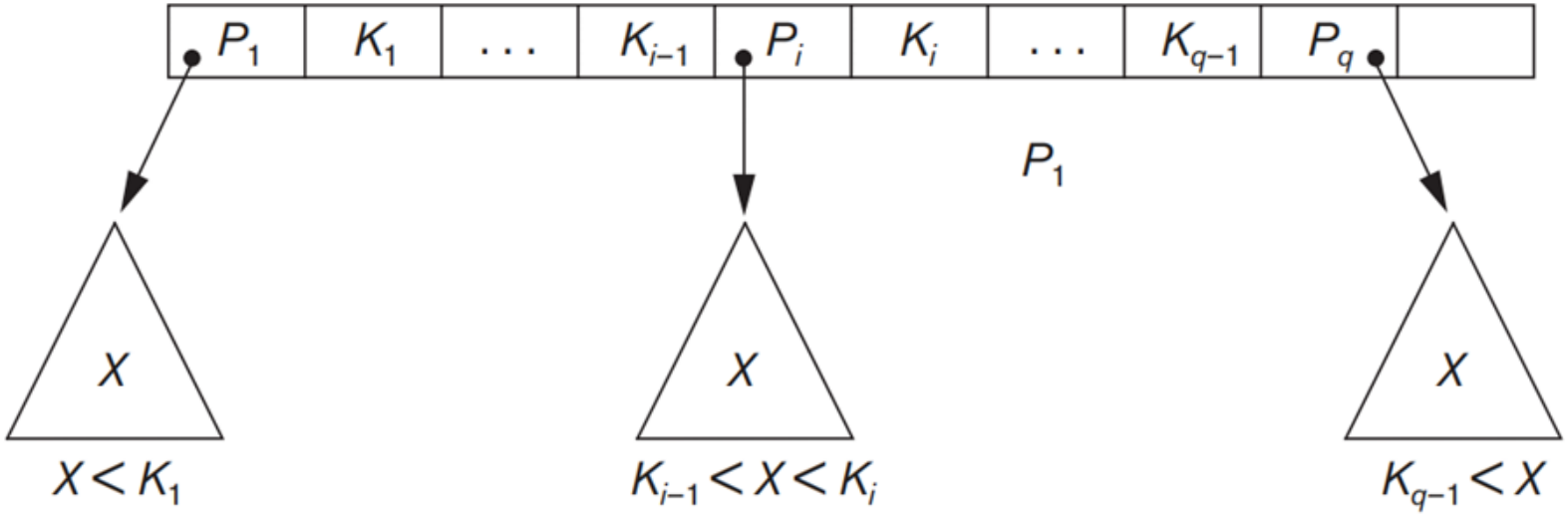
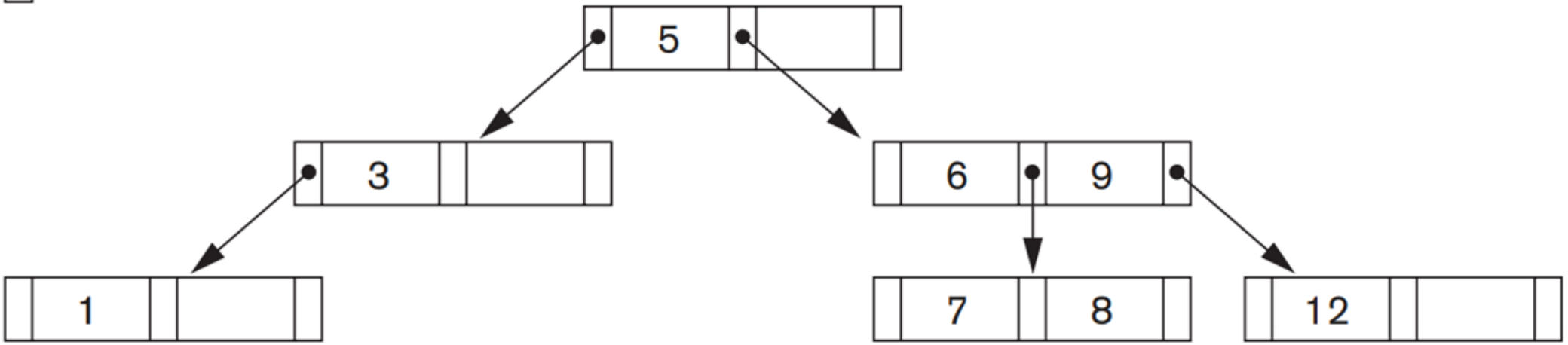


Figure 17.8
A node in a search tree with pointers to subtrees below it.



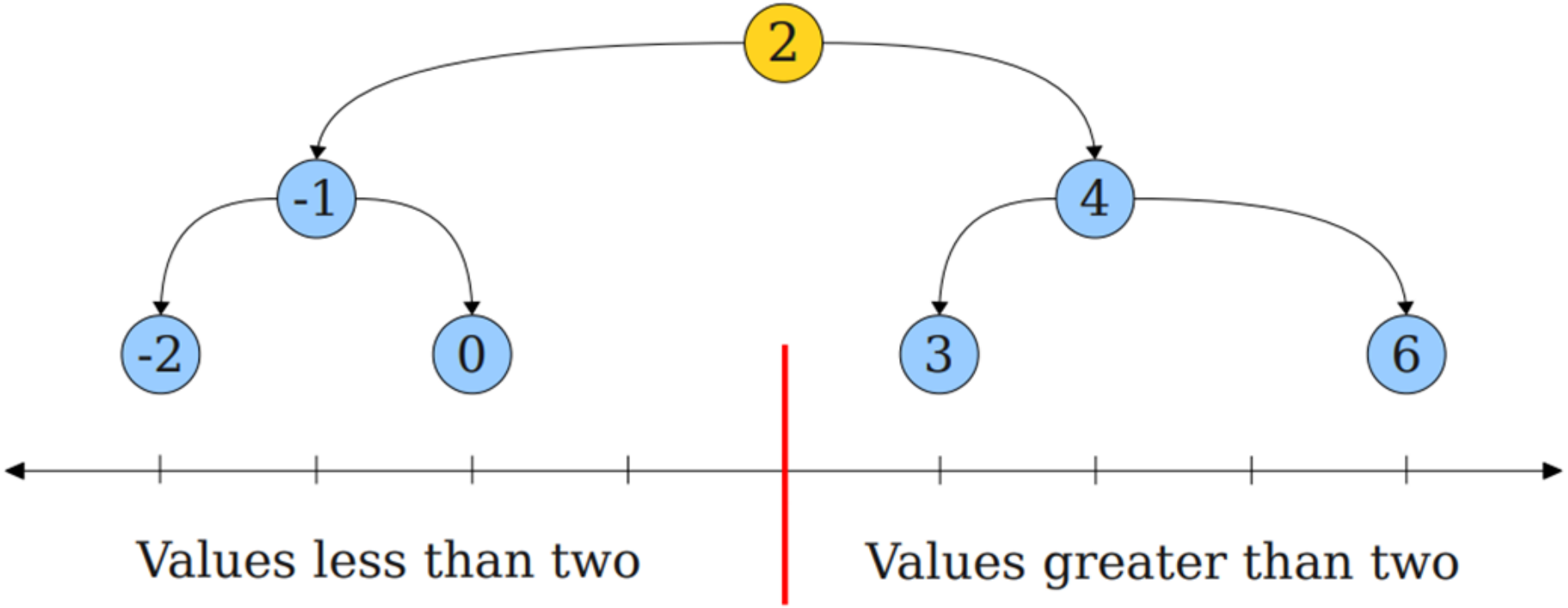

5
3
6
9
1
7
8
12

Next Topic:

B Tree

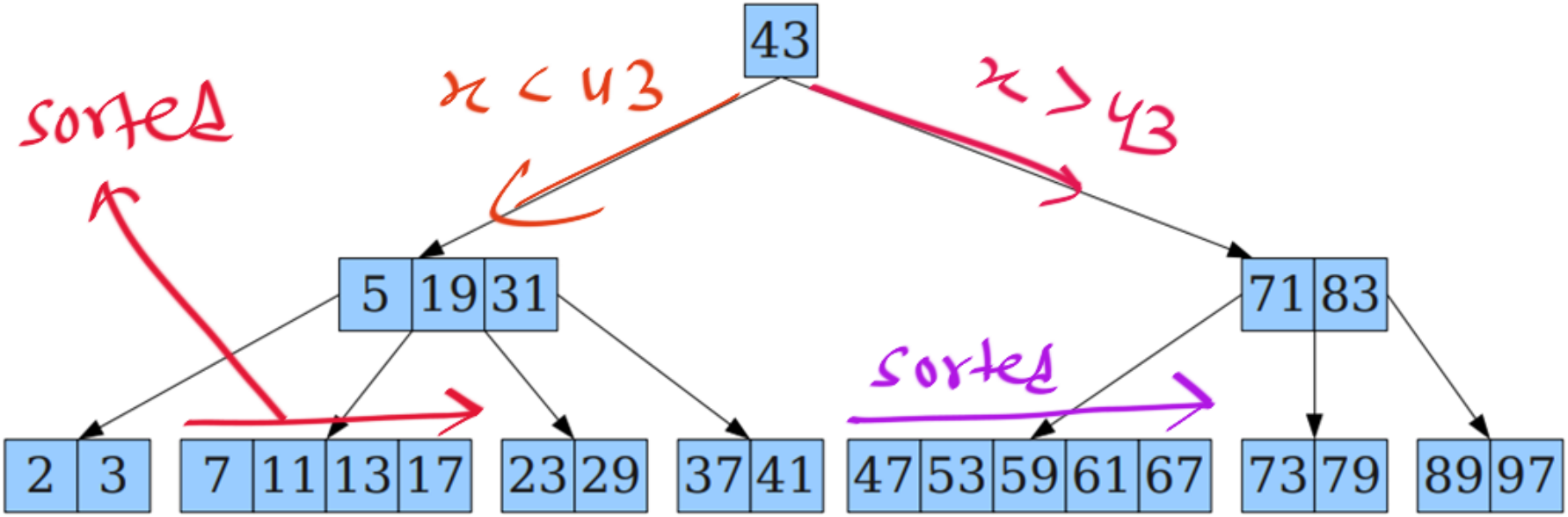
Generalizing BSTs

- In a binary search tree, each node stores a single key.
- That key splits the “key space” into two pieces, and each subtree stores the keys in those halves.



Generalizing BSTs

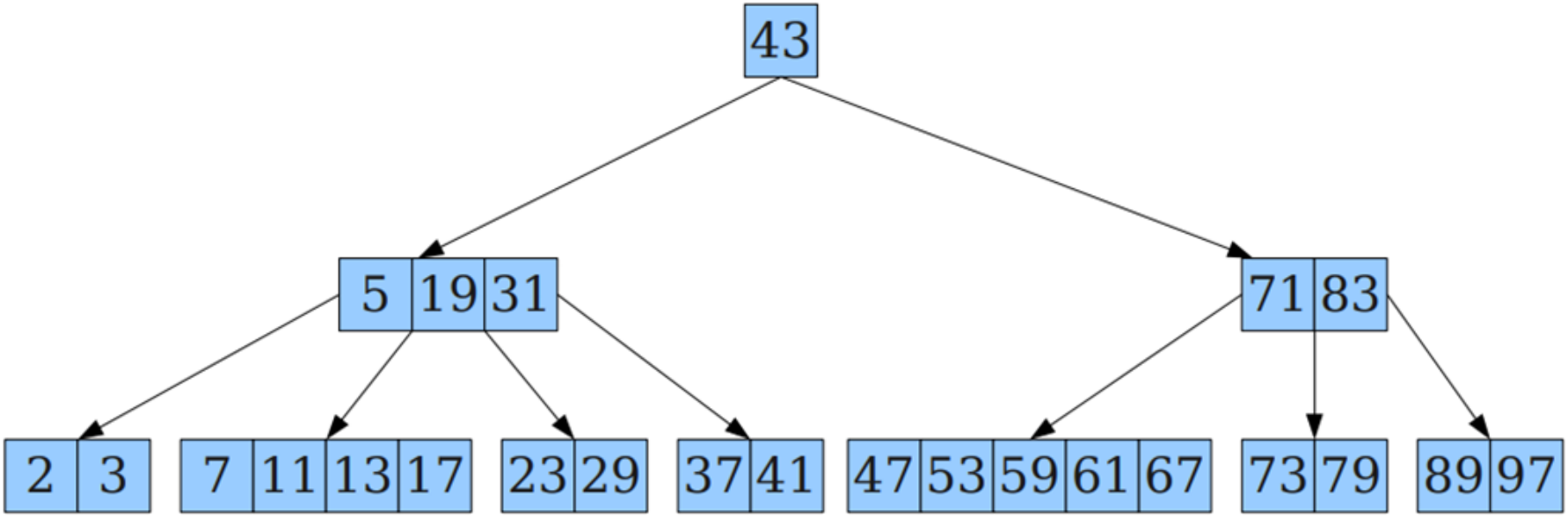
- In a **multiway search tree**, each node stores an arbitrary number of keys in sorted order.



- In a node with k keys splits the "key space" into $k + 1$ pieces, and each subtree stores the keys in those pieces.

Generalizing BSTs

- In a **multiway search tree**, each node stores an arbitrary number of keys in sorted order.



- In a node with k keys splits the "key space" into $k + 1$ pieces, and each subtree stores the keys in those pieces.

What do we need to make sure while designing any Disk Data Structure (Since a Node a Block)??

1. Minimizing the number of levels in the tree is one goal. (To reduce I/O Access Cost)
2. Block Space Utilized as much as possible (the nodes to be as full as possible)

What do we need to make sure while designing any Disk Data Structure (Since a Node a Block)??

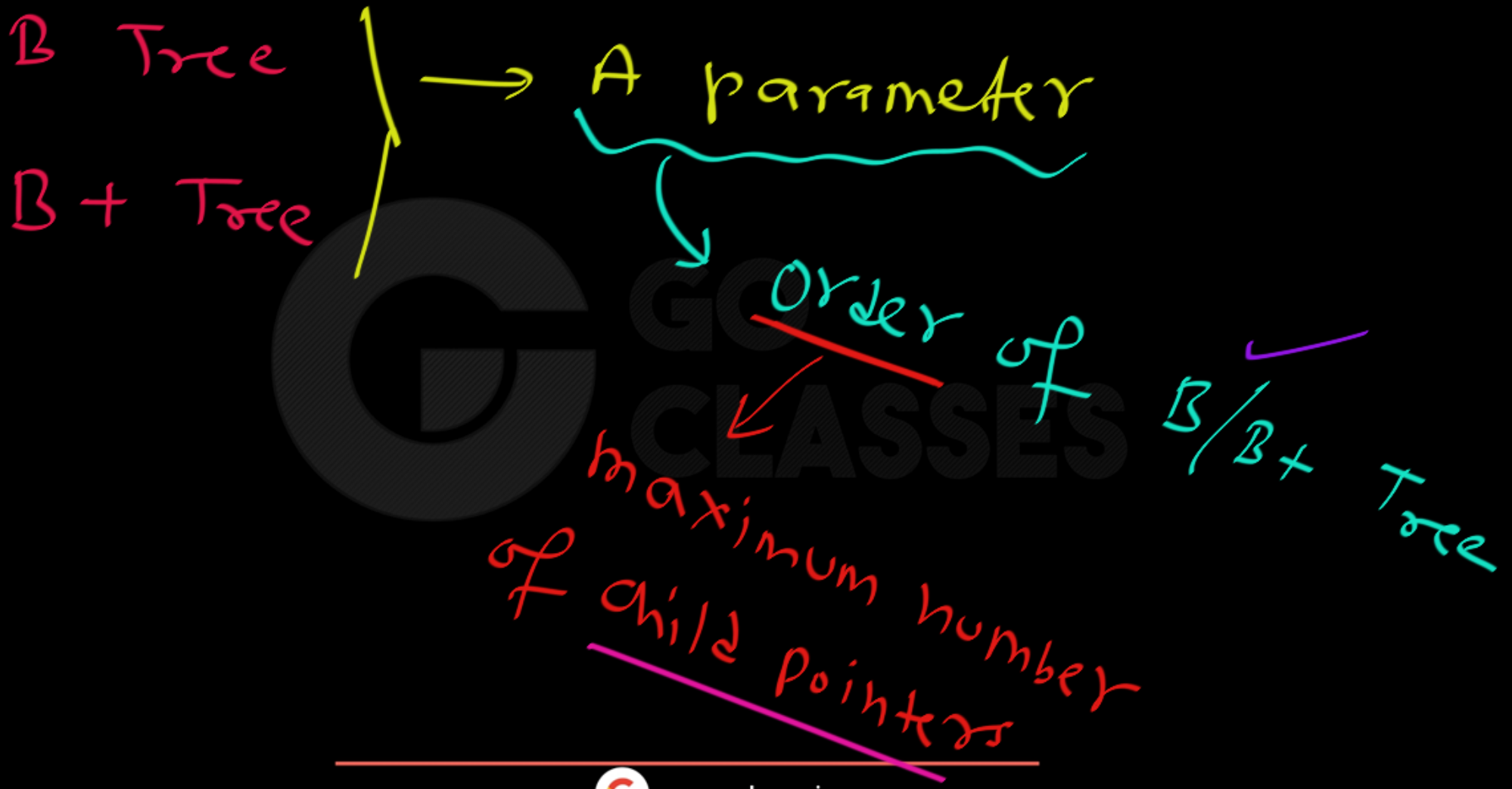
1. Minimizing the number of levels in the tree is one goal. (To reduce I/O Access Cost)
2. Block Space Utilized as much as possible (the nodes to be as full as possible)

The B-tree addresses both of these problems by specifying additional constraints on the search tree.

B Tree
B+ Tree
→ Easy Concepts

Let's Start B Tree

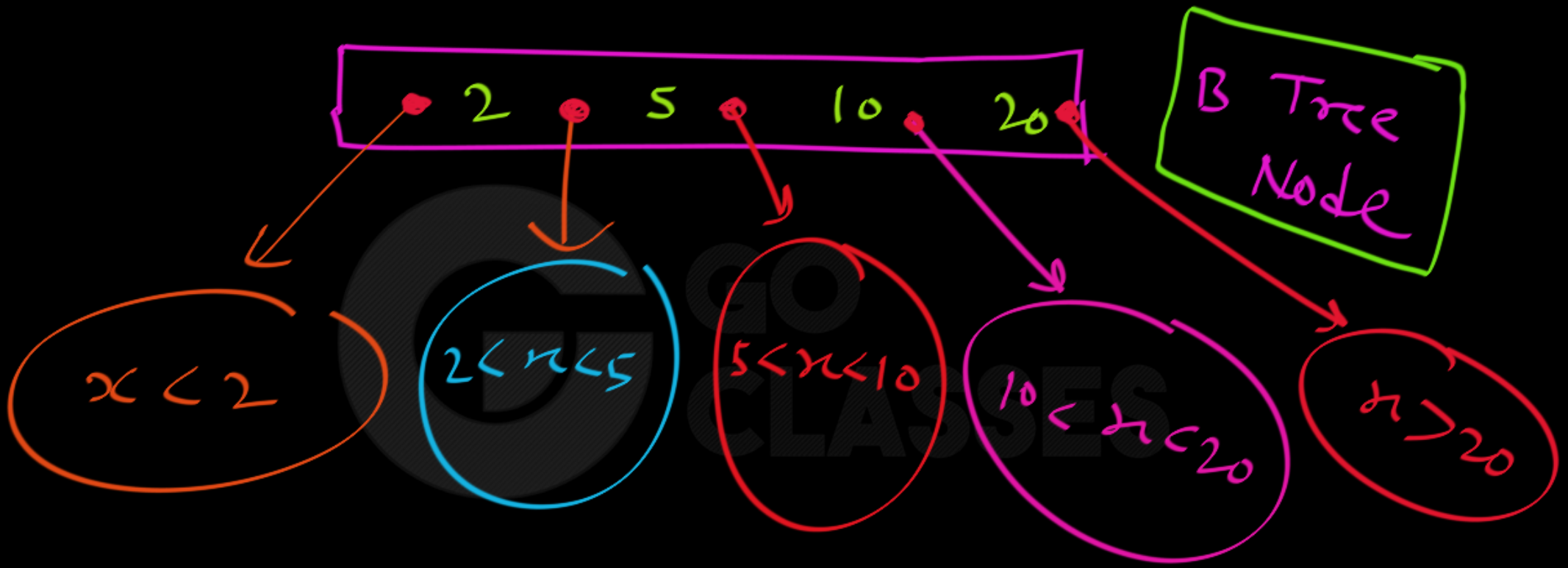
- Rules (Definition)
- Insertion
- Questions, Variations


B Tree
B+ Tree
A parameter
Order of B/B+ Tree
maximum number of child pointers

NOTE:

There is a parameter p associated with each B-tree index, Called ORDER of B Tree.

Order of B Tree = Maximum number of Tree(Block or Node) Pointers per Node.

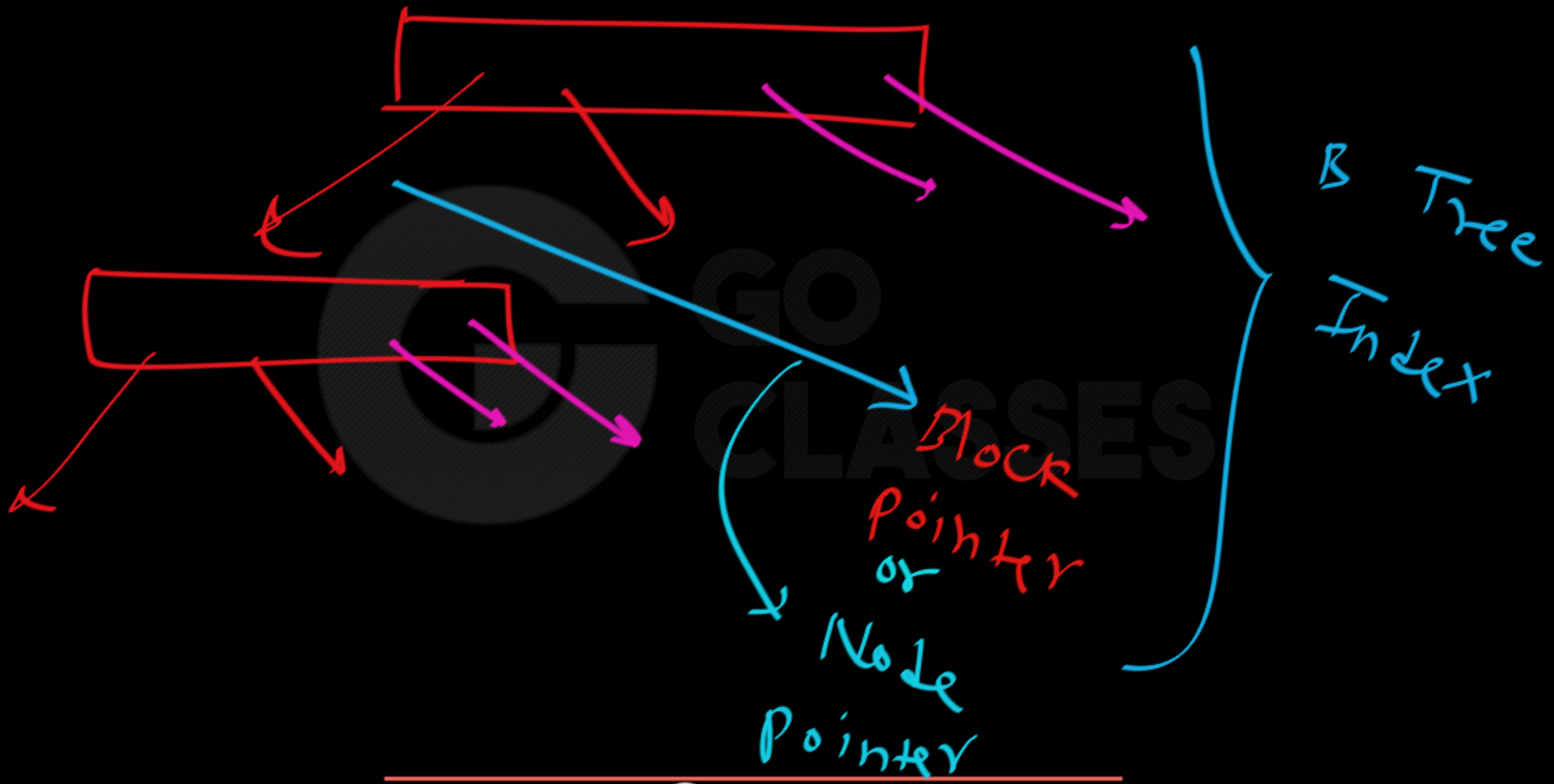

2
5
10
20
B Tree Node
x < 2
2 < x < 5
5 < x < 10
10 < x < 20
x > 20
Order : 5

Pointer :

- Block Pointer — BP
- Record Pointer — RP = BP + offset

Block ≡ Node

Block Pointer ≡ Node Pointer
≡ Tree Pointer
≡ Child pointer

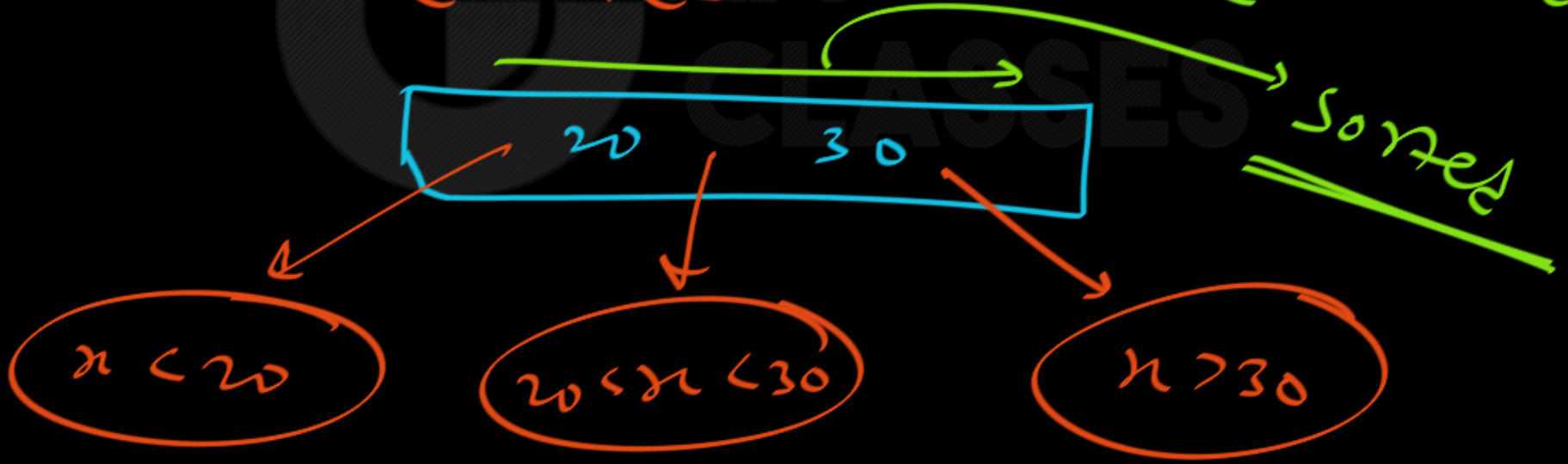

B Tree
Index
Block
Pointer
or
Node
Pointer

Definition(Rules) of B Tree of Order p

B Tree of order p

→ max. no. of Block Pointers in each node

(1) Search Tree Rules (obvious)



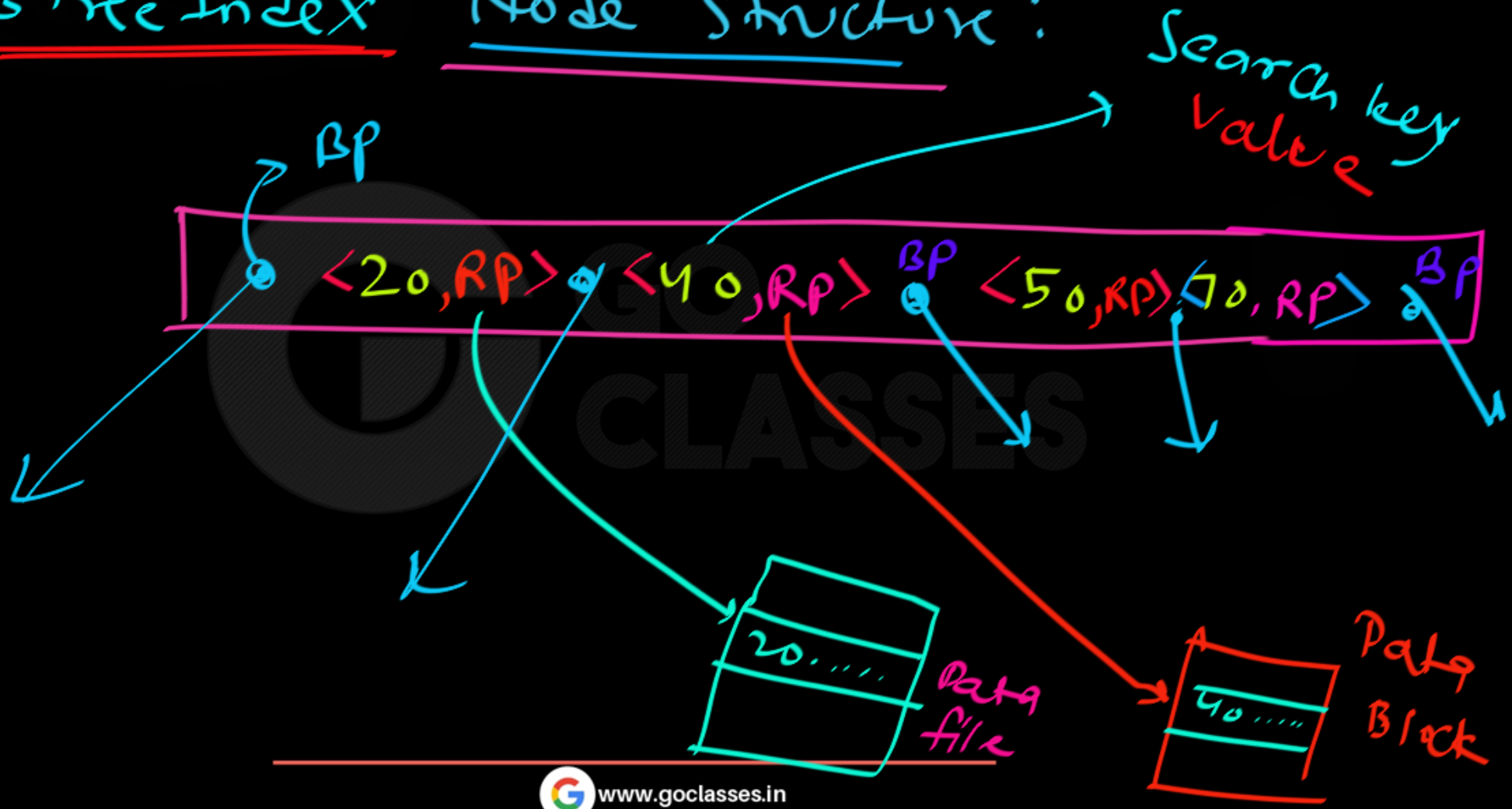
(2) All Leaves must be on Same Level.

(3) Space utilization Rule: (≥50% utilization)

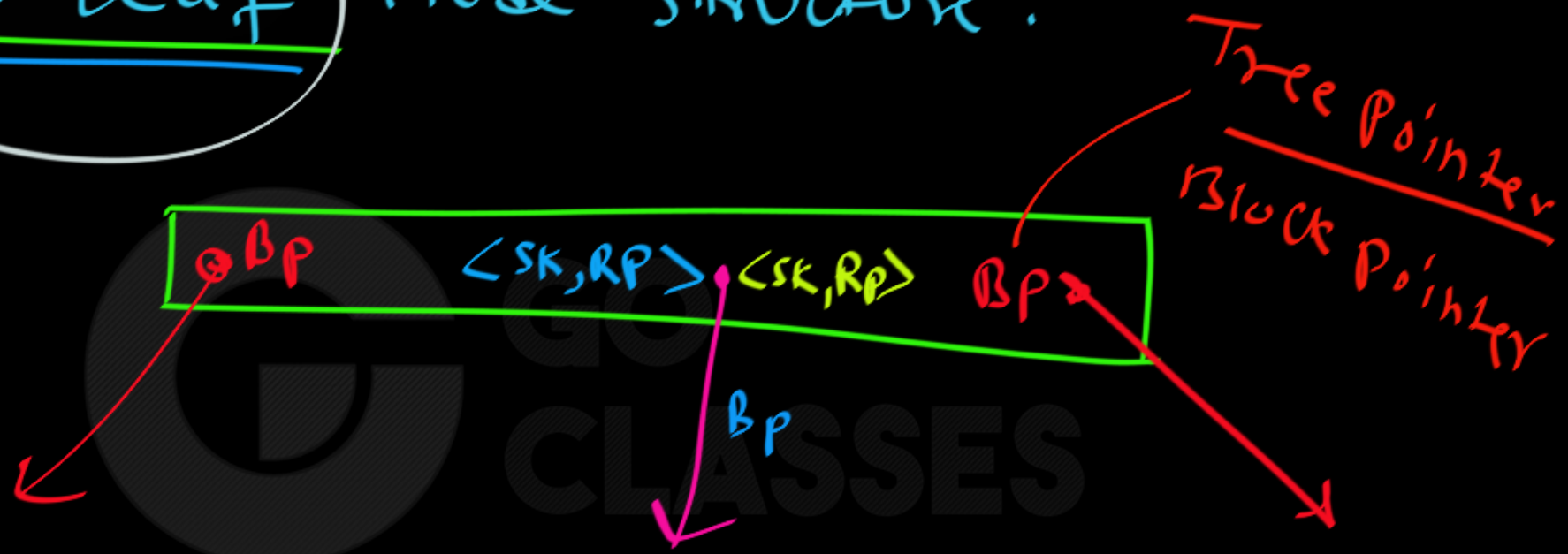
Root node: 2 BP to P BP

Non-Root node: $\lceil \frac{P}{2} \rceil$ to P BP

(4) B Tree Index Node Structure :

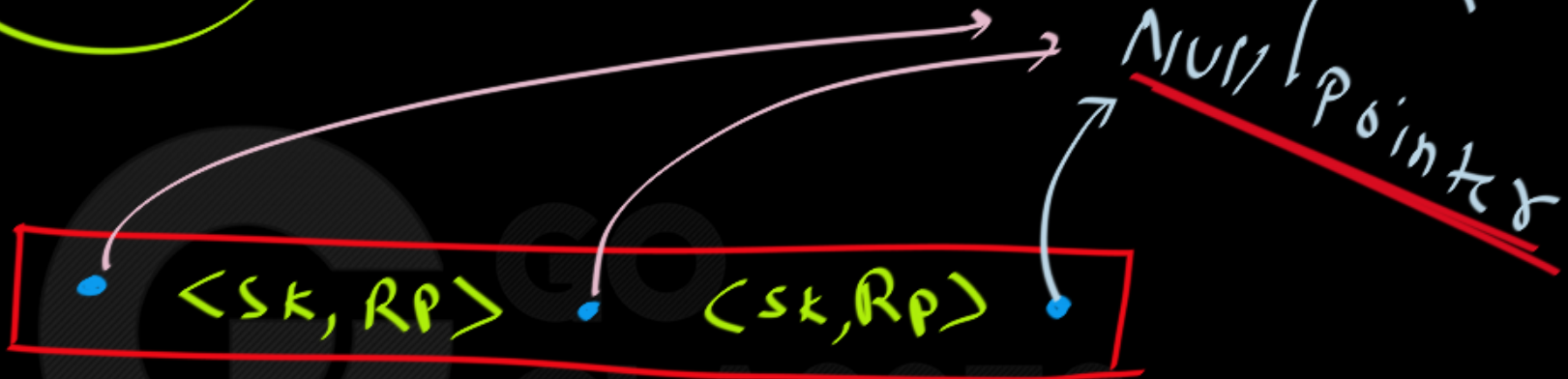


4 Non-Leaf Node structure :



4

Leaf Node Structure :



So; ALL Nodes in B Tree Have Same Structure

Conclusion :

B Tree of order p :

- max. no. of keys per node : p-1
- max. number of pointers in any B Tree Node

Conclusion :

B Tree of order p :

→ max. number of Node Pointers in any B Tree Node

If a node has x BP then

keys in that node :

x - 1

Conclusion :

B Tree of order p :

Root node : 2 BP to p BP

Other nodes : $\lceil \frac{p}{2} \rceil$ BP to p BP

→ minimum 50% utilization of order must happen for every node (except Root)

Conclusion :

B Tree of order p :

- Root node : 2 - 1 key to p - 1 key
- Other nodes : $\lceil \frac{p}{2} \rceil$ - 1 key to p - 1 keys

→ minimum 50% utilization of order must happen for every node (except Root)

Conclusion:

B Tree of order 7:

- maximum no. of Tree pointers
- Root: 1 key to 6 keys
- Other nodes: 3 key to 6 keys
- $\geq 50\%$ utilization of order
  - min $\left\lceil \frac{7}{2} \right\rceil = 4$ BP

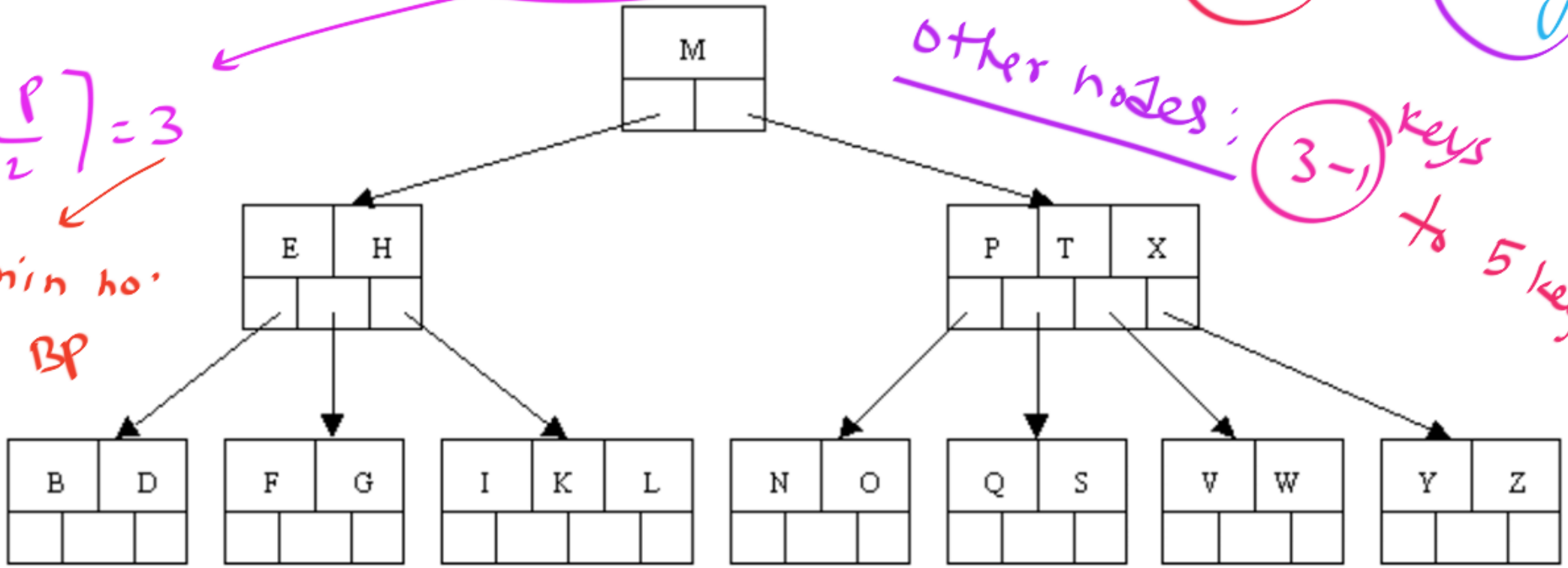
B-Tree of order 6;

Root : 1 key to 5 key

$\lceil \frac{p}{2} \rceil = 3$

min no. of BP

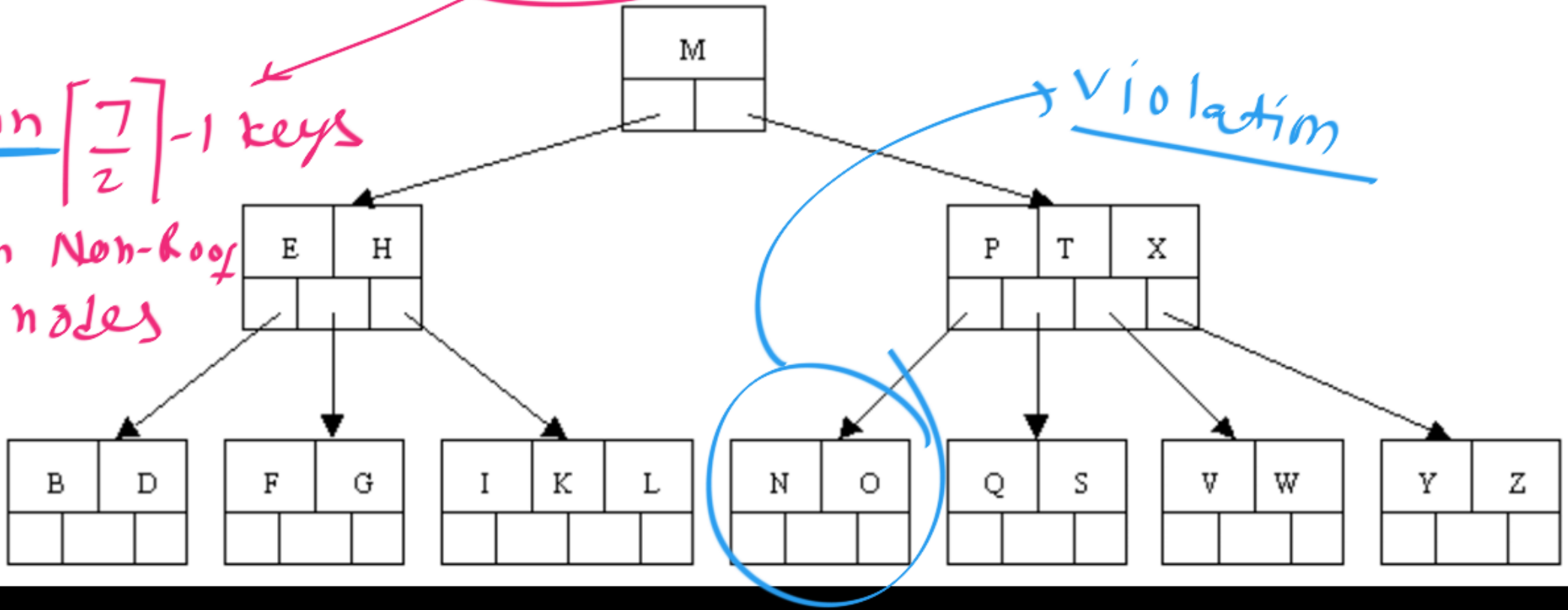
Other nodes : 3-1 keys to 5 keys



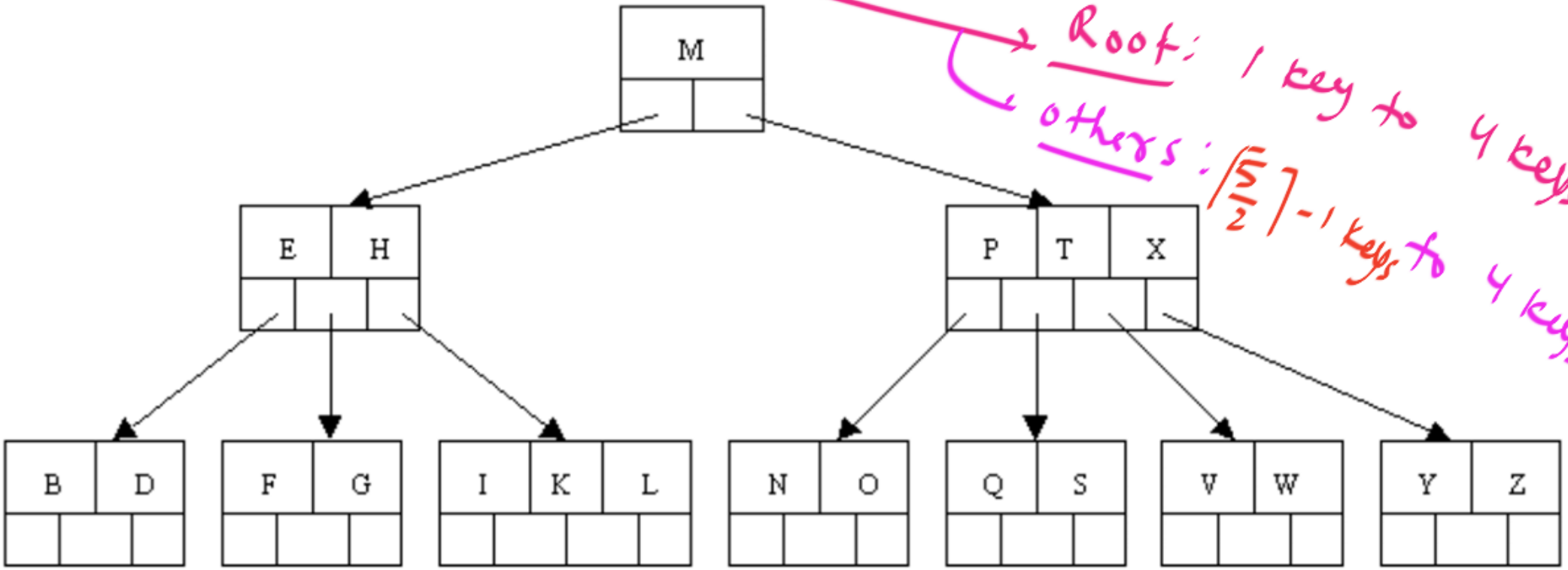
B-Tree of order 7 : ✗

min $\lceil \frac{7}{2} \rceil - 1$ keys in Non-root nodes

violation



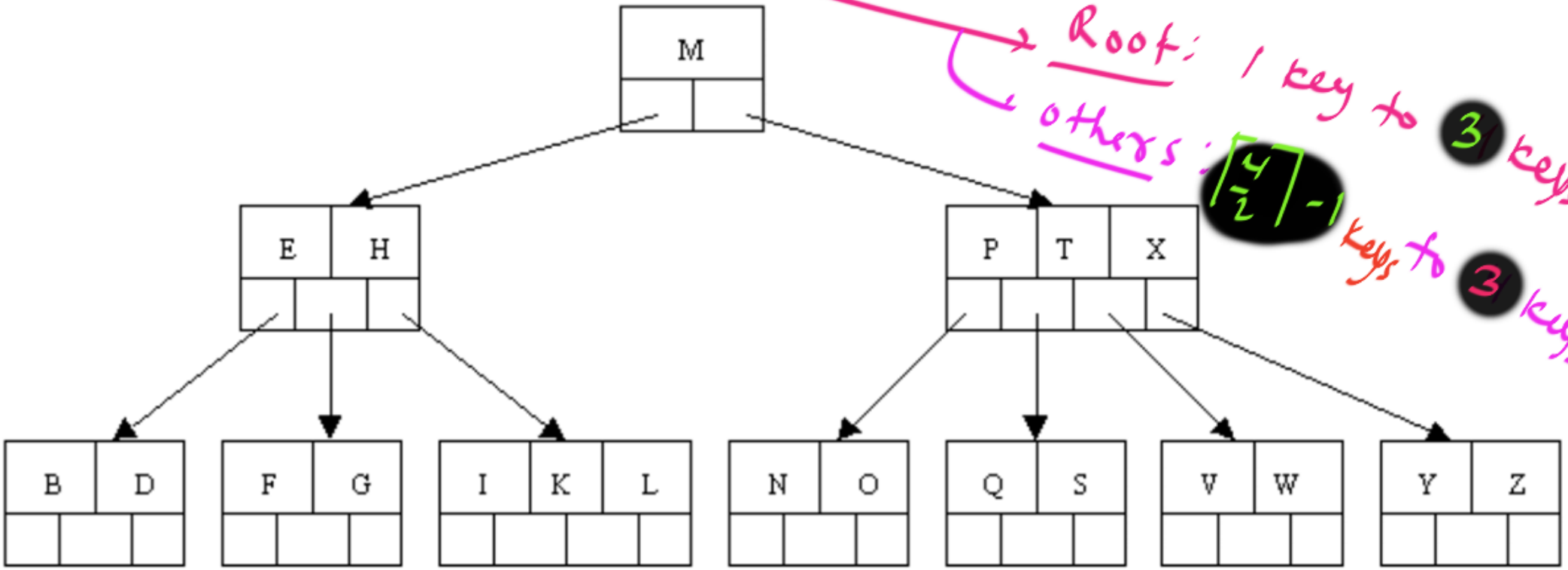
B-Tree of order 5 :



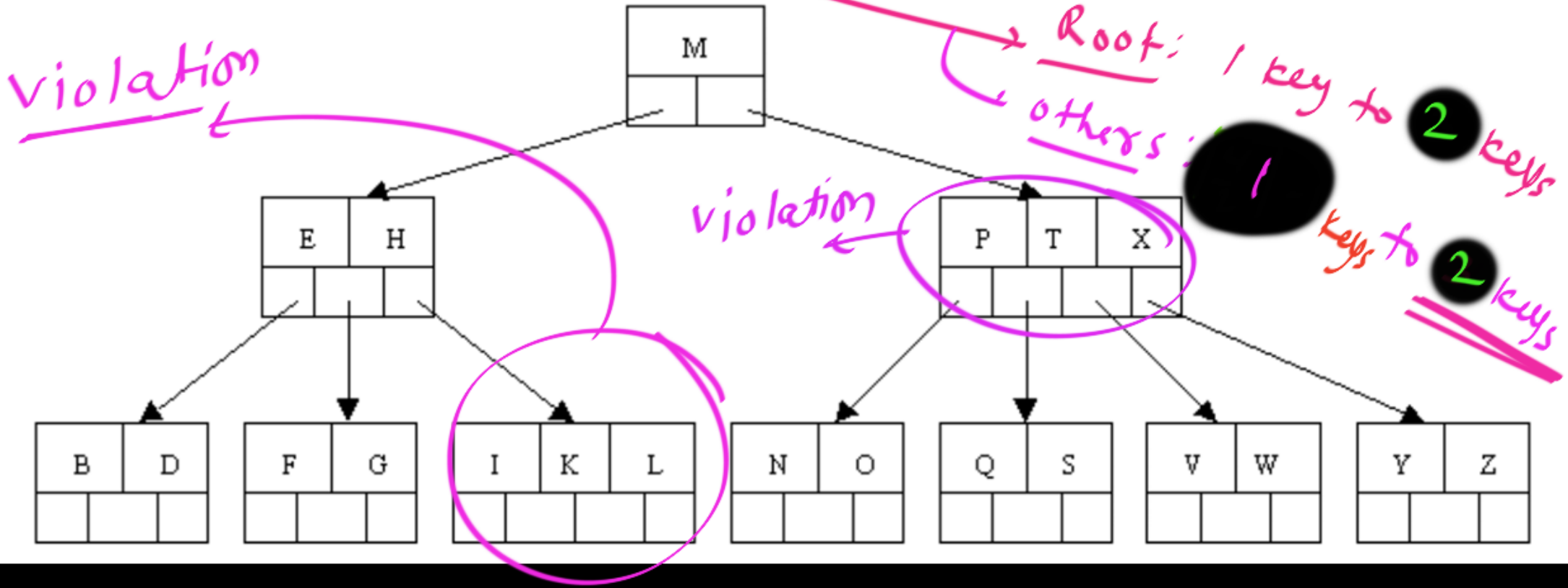
Root: 1 key to 4 keys

others: $\lceil \frac{5}{2} \rceil - 1$ keys to 4 keys

B-Tree of order 4 :

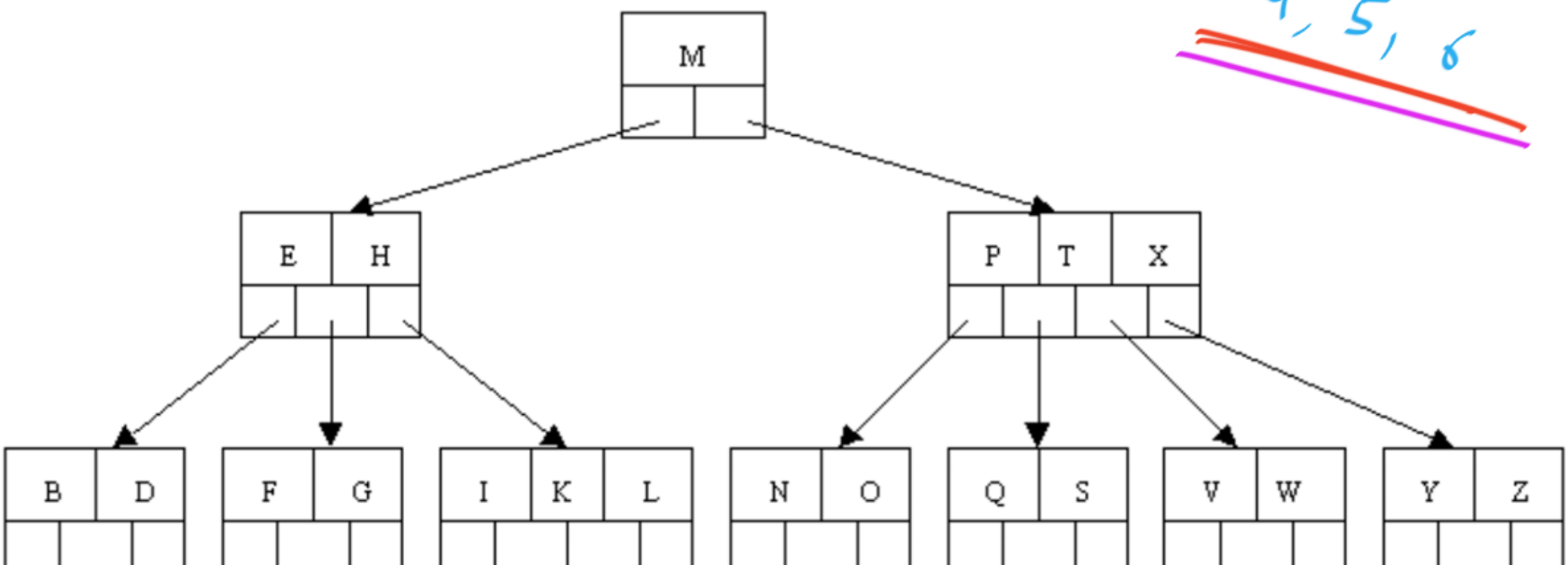


B-Tree of order 3 :

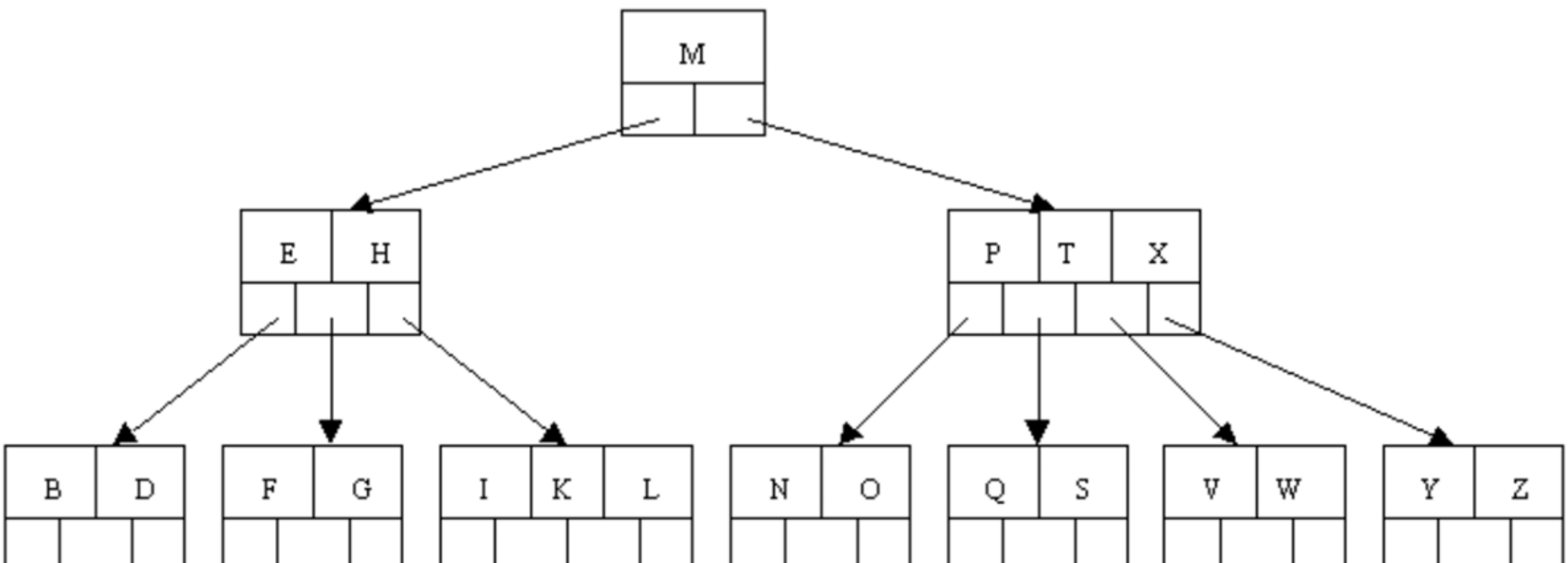


Q: for the following B Tree : Possible orders

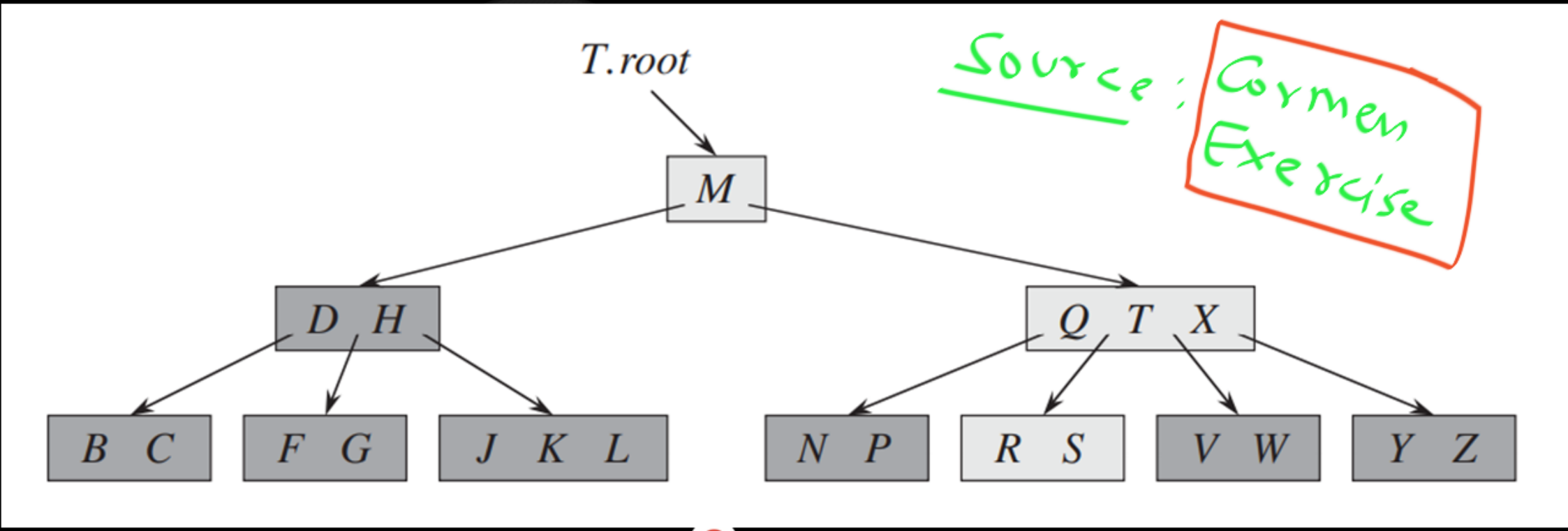
= 4, 5, 6



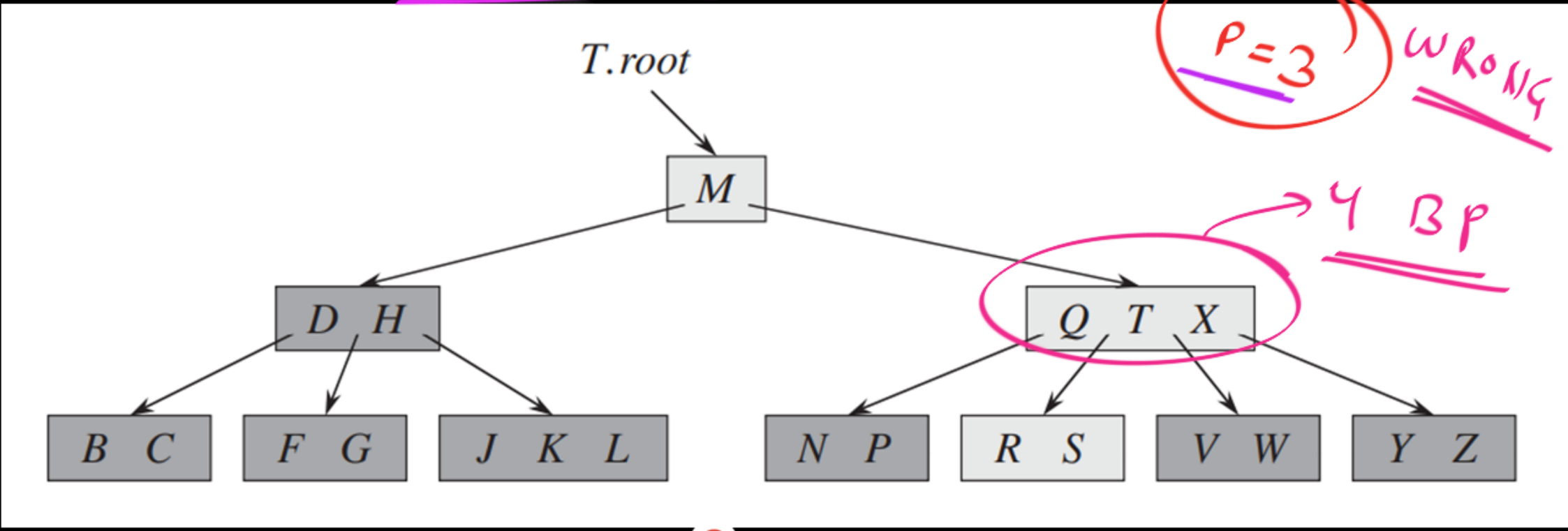
The following is an example of a B-tree of order 5. This means that (other that the root node) all internal nodes have at least ceil(5 / 2) = ceil(2.5) = 3 children (and hence at least 2 keys). Of course, the maximum number of children that a node can have is 5 (so that 4 is the maximum number of keys). According to condition 4, each leaf node must contain at least 2 keys. In practice B-trees usually have orders a lot bigger than 5.



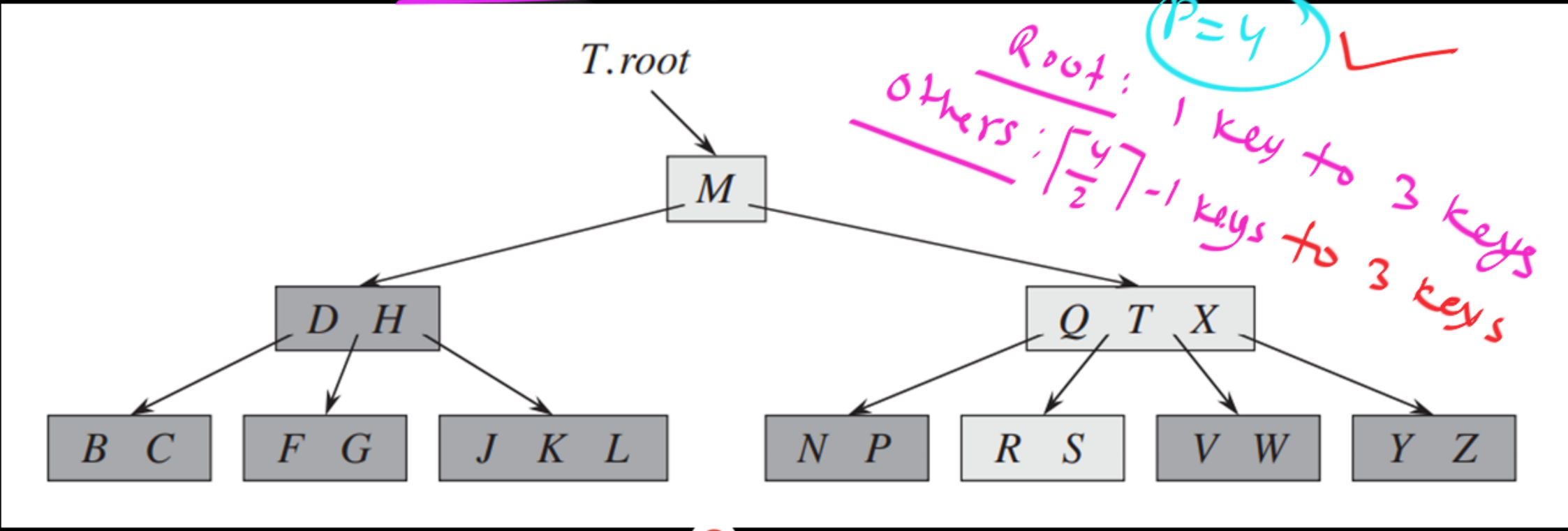
Q: Order p of B Tree = Maximum number of Tree(Block or Node) Pointers per Node. For what values of p is the tree of below Figure a legal B-tree?



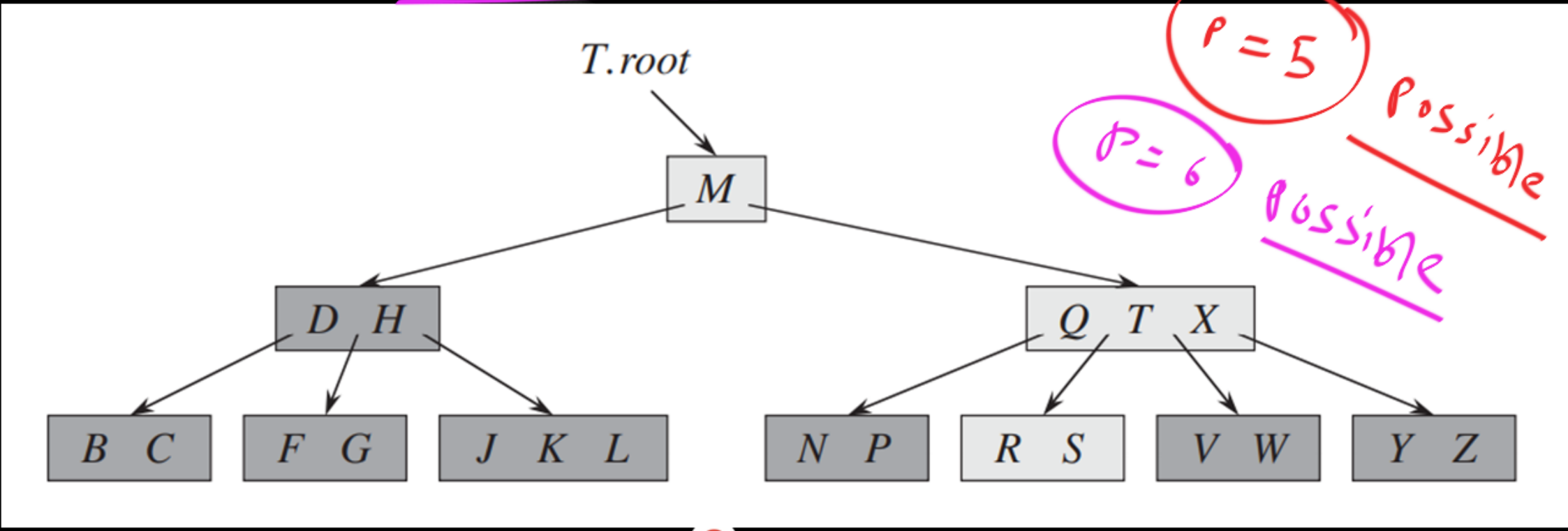
Q: Order p of B Tree = Maximum number of Tree(Block or Node) Pointers per Node. For what values of p is the tree of below Figure a legal B-tree?



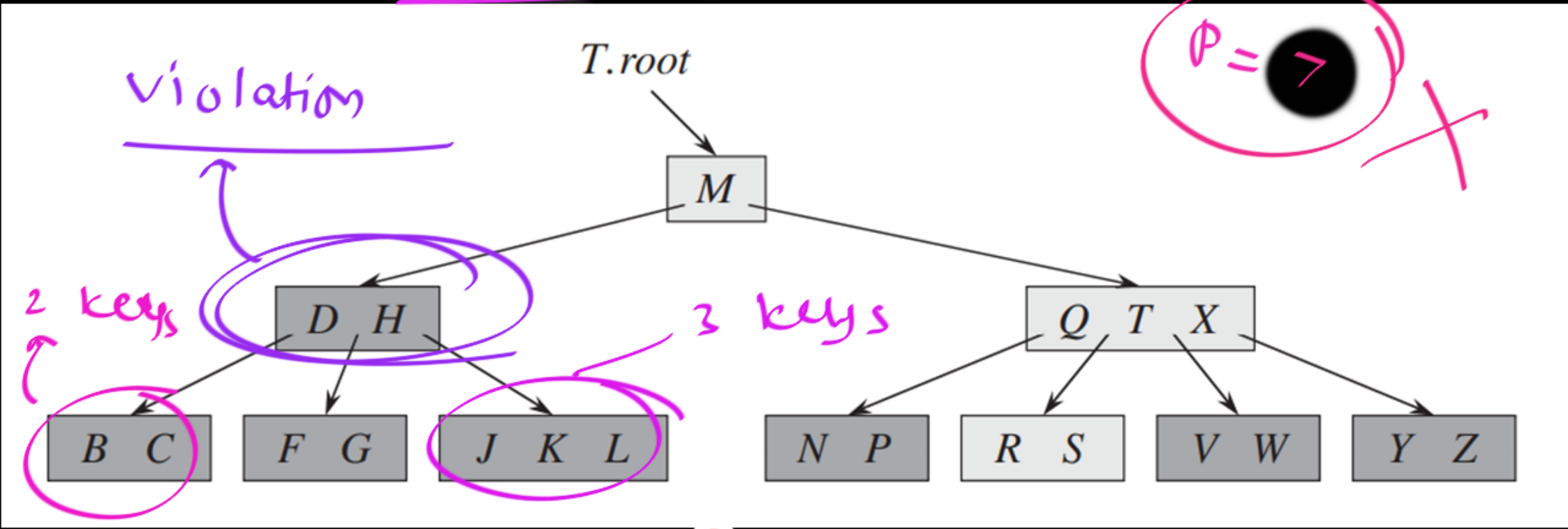
Q: Order p of B Tree = Maximum number of Tree(Block or Node) Pointers per Node. For what values of p is the tree of below Figure a legal B-tree?



Q: Order p of B Tree = Maximum number of Tree(Block or Node) Pointers per Node. For what values of p is the tree of below Figure a legal B-tree?



Q: Order p of B Tree = Maximum number of Tree(Block or Node) Pointers per Node. For what values of p is the tree of below Figure a legal B-tree?



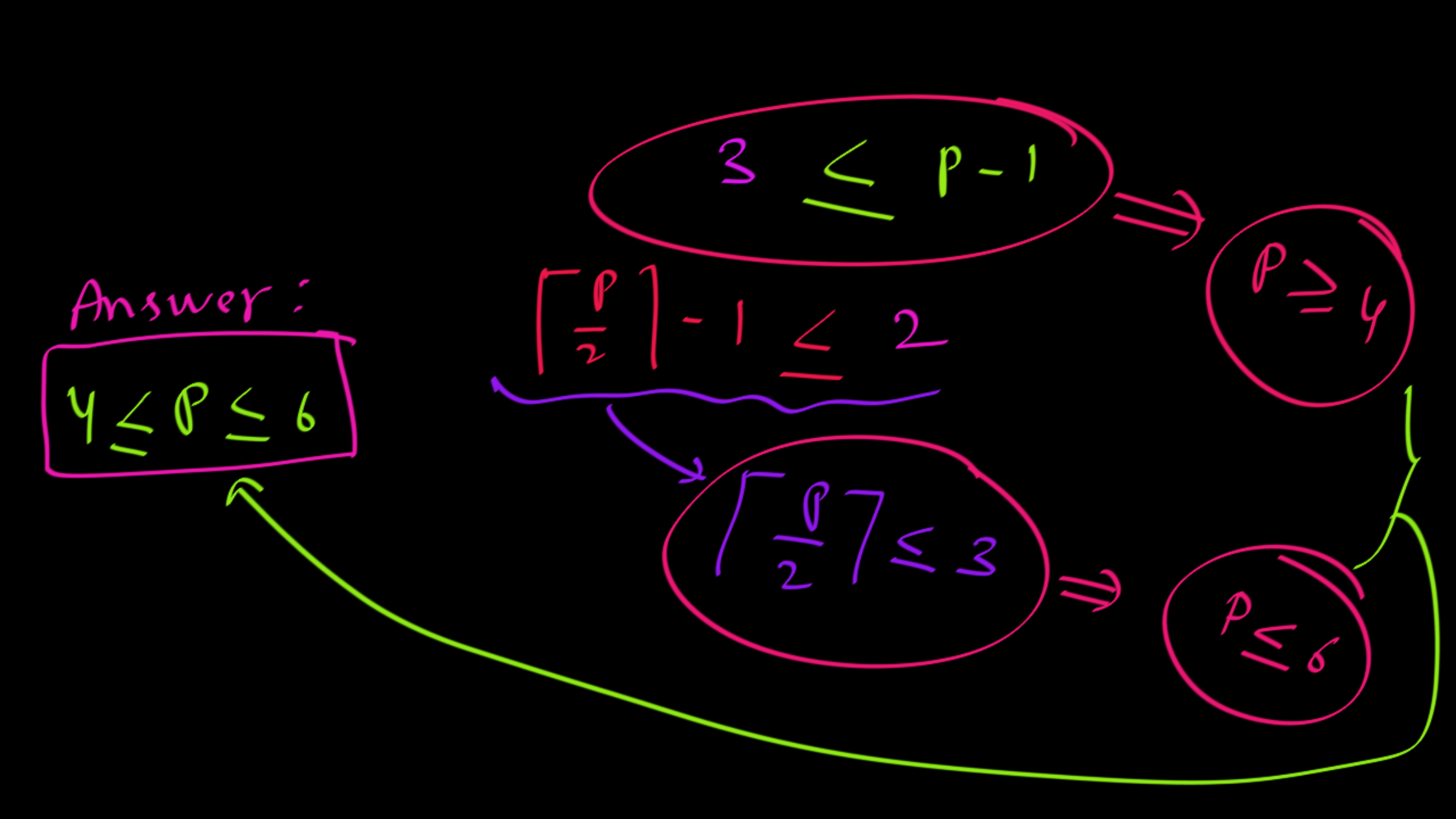
method: Order p

Root: 1 key —— p-1 keys

max keys in Root

$1 \leq p-1 \Rightarrow p \geq 2$

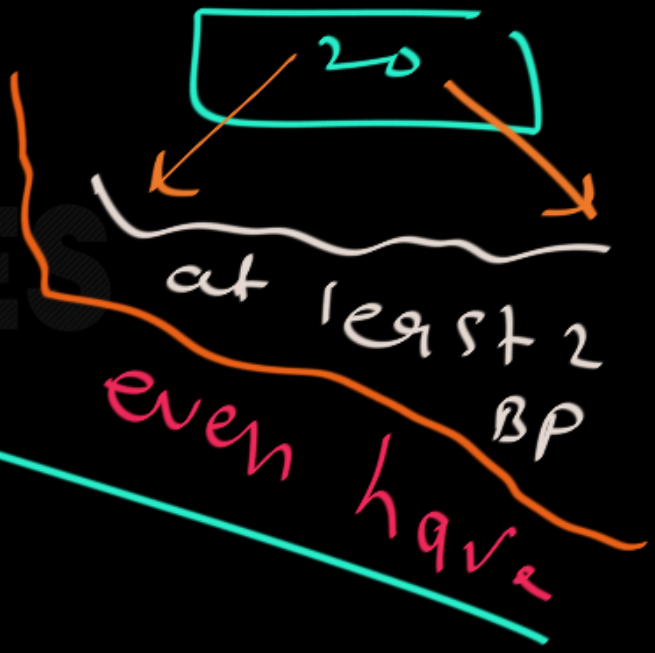
Other nodes: $\lceil \frac{p}{2} \rceil - 1$ keys $\leq$ #keys $\leq p-1$ keys


Answer:
$4 \leq p \leq 6$
$\lceil \frac{p}{2} \rceil - 1 \leq 2$
$3 \leq p - 1$
$p \geq 4$
$\lceil \frac{p}{2} \rceil \leq 3$
$p \leq 6$

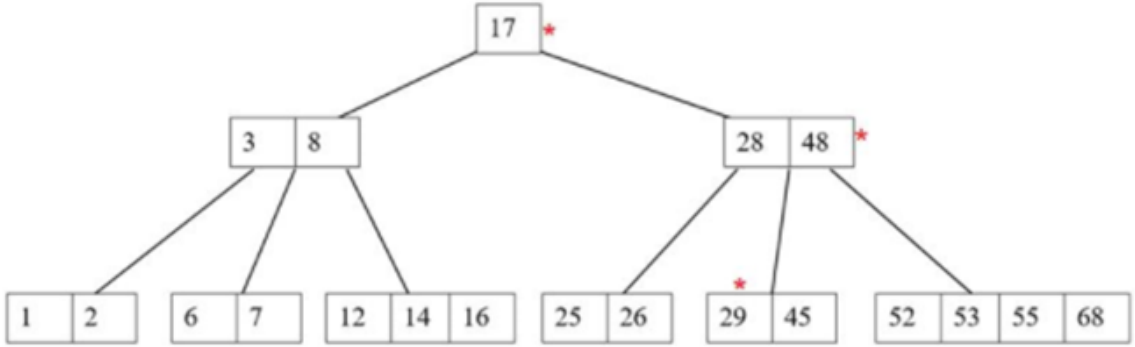
Q: Can order of B Tree be (1)?

↳ NEVER possible

↳ You can't even have Search Tree.

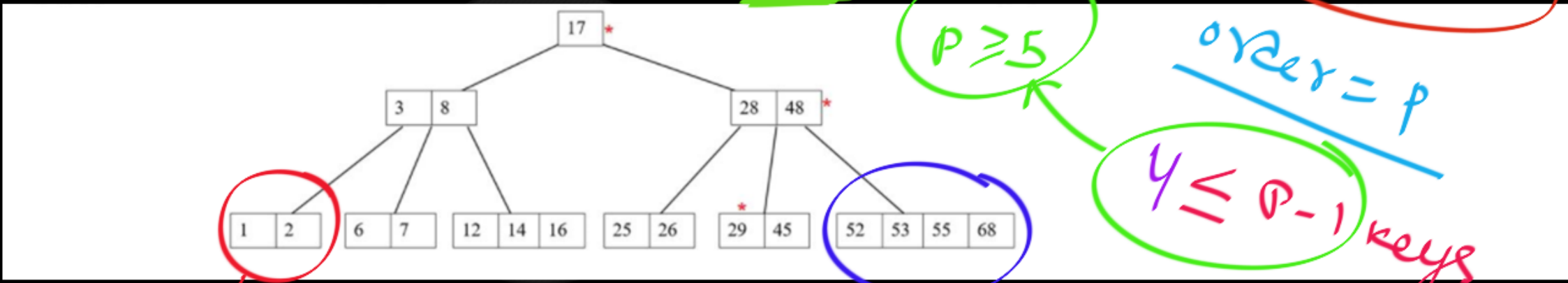


Q: Order p of B Tree = Maximum number of Tree(Block or Node) Pointers per Node. For what values of p is the tree of below Figure a legal B-tree?



Q: Order p of B Tree = Maximum number of Tree(Block or Node) Pointers per Node. For what values of p is the tree of below Figure a legal B-tree? ⇒ 5, 6

$5 \le p \le 6$



$p \ge 5$

order = p

$y \le p-1$ keys

2 keys

4 keys

$\lceil \frac{p}{2} \rceil - 1 \le 2$

$p \le 6$

Definition(Rules) of B Tree of Order p

A B-tree of order p:

1. Each internal node in the B-tree (Figure 17.10(a)) is of the form

$$<P_1, <K_1, Pr_1>, P_2, <K_2, Pr_2>, \ldots, <K_{q-1}, Pr_{q-1}>, P_q>$$

where $q \leq p$. Each P_i is a **tree pointer**—a pointer to another node in the B-tree. Each Pr_i is a **data pointer**[10]—a pointer to the record whose search key field value is equal to K_i (or to the data file block containing that record).

2. Within each node, $K_1 < K_2 < \ldots < K_{q-1}$.

3. For all search key field values X in the subtree pointed at by P_i (the ith subtree, see Figure 17.10(a)), we have:

$$K_{i-1} < X < K_i \text{ for } 1 < i < q; X < K_i \text{ for } i = 1; \text{ and } K_{i-1} < X \text{ for } i = q$$

[10] A data pointer is either a block address or a record address; the latter is essentially a block address and a record offset within the block.

A B-tree of order p:

1. Each internal node in the B-tree (Figure 17.10(a)) is of the form

$$<P_1, <K_1, Pr_1>, P_2, <K_2, Pr_2>, \ldots, <K_{q-1}, Pr_{q-1}>, P_q>$$

where $q \leq p$. Each P_i is a **tree pointer**—a pointer to another node in the B-tree. Each Pr_i is a **data pointer**[10]—a pointer to the record whose search key field value is equal to K_i (or to the data file block containing that record).

2. Within each node, $K_1 < K_2 < \ldots < K_{q-1}$. → sorted Node

3. For all search key field values X in the subtree pointed at by P_i (the ith subtree, see Figure 17.10(a)), we have:

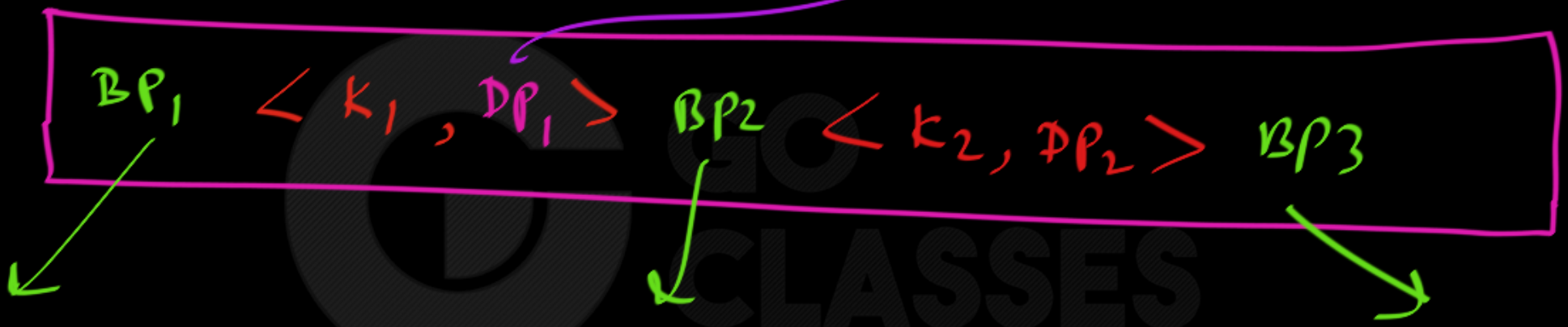
$$K_{i-1} < X < K_i \text{ for } 1 < i < q; \; X < K_i \text{ for } i = 1; \text{ and } K_{i-1} < X \text{ for } i = q$$

[10] A data pointer is either a block address or a record address; the latter is essentially a block address and a record offset within the block.

4. Each node has at most p tree pointers.
5. Each node, except the root node, has at least $\lceil (p/2) \rceil$ tree pointers. The root node has at least two tree pointers unless it is the only node in the tree.
6. A node with q tree pointers, $q \leq p$, has $q - 1$ search key field values (and hence has $q - 1$ data pointers).
7. All leaf nodes are at the same level. Leaf nodes have the same structure as internal nodes except that all of their *tree pointers* P_i are NULL.

B Tree Node structure :

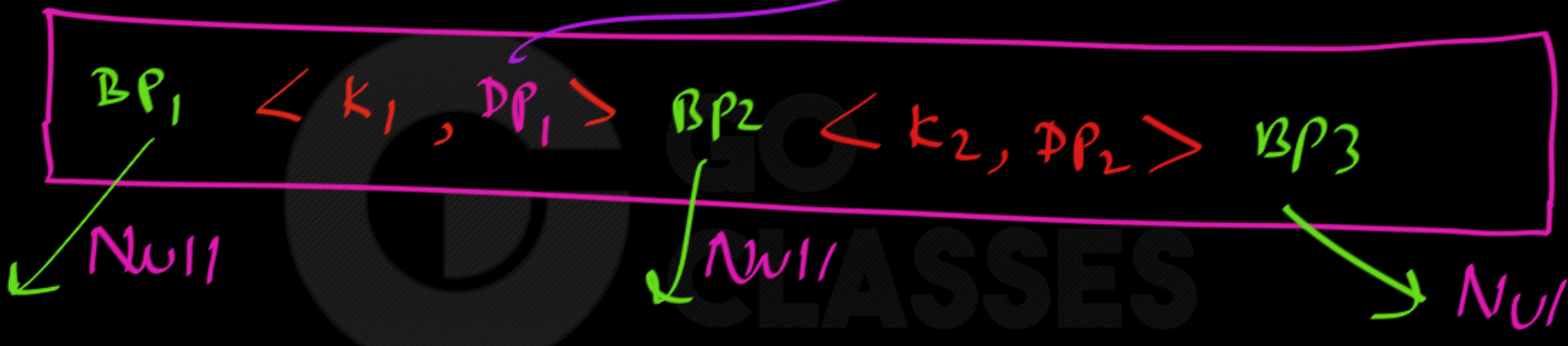
by default take Record pointer



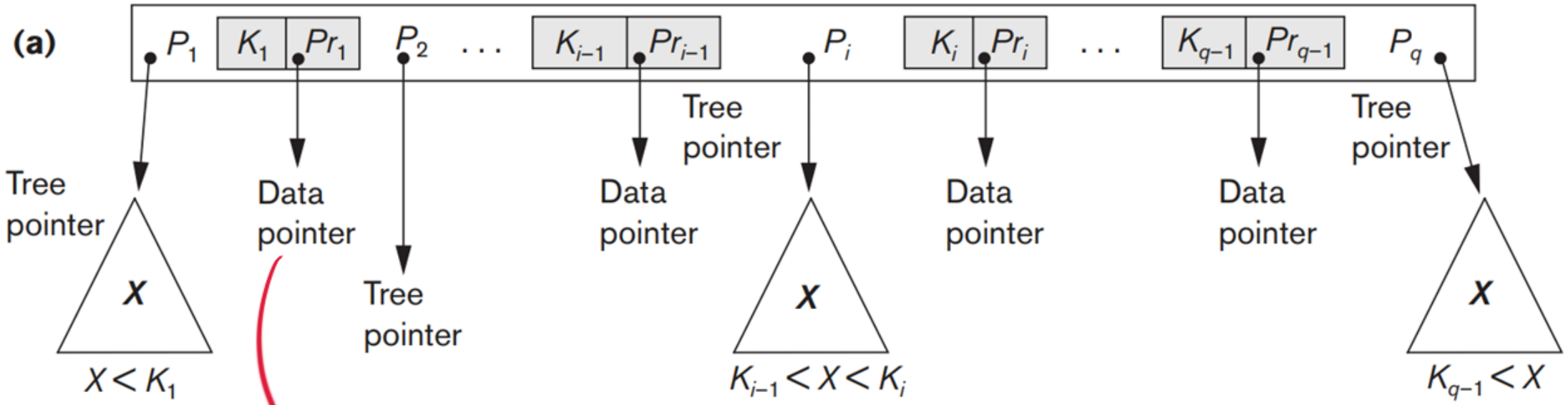
Data pointer OR pointer { Block Pointer OR Record Pointer }

Leaf Node structure :

by default take Record pointer



Data pointer OR pointer { Block Pointer OR Record Pointer }



by Default Record pointer

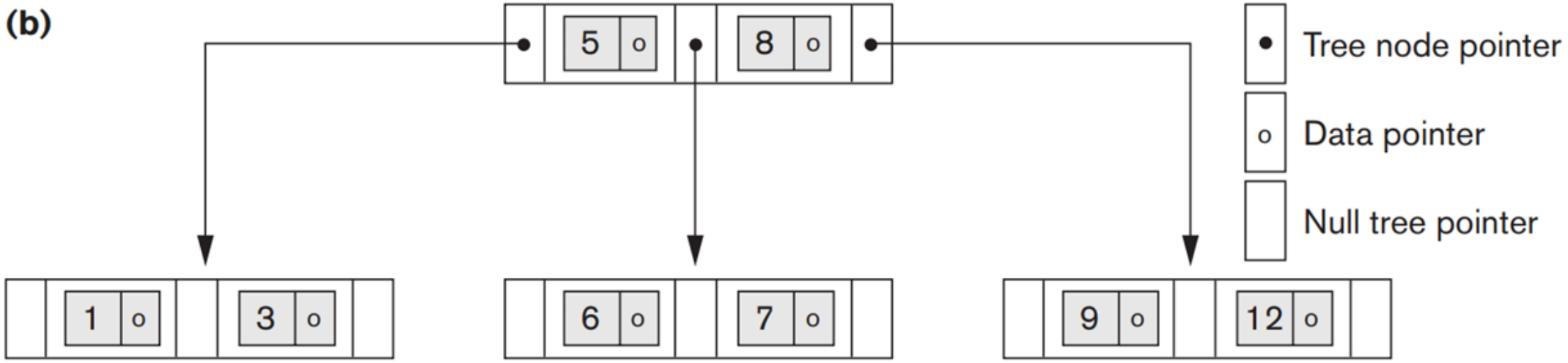


Figure 17.10

B-tree structures. (a) A node in a B-tree with $q - 1$ search values. (b) A B-tree of order $p = 3$. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.

Q 14.10: (Ullman DBMS Book 2nd Ed)

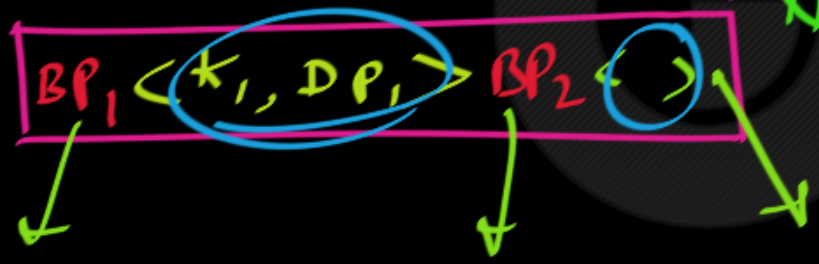
Suppose our blocks are 4096 bytes. Also let keys be integers of 4 bytes and let pointers be 8 bytes. If there is no header information kept on the blocks, then find the Order of B Tree(where order is defined as maximum number of tree pointers per B tree node)

Order of B Tree:

→ maximum no. of Tree pointer per node

→ P

Structure of EVERY node:



In one B-Tree node

{ max BP : P

max (Key, DP) Pairs = P-1 }

$$P(8) + (P-1)(4+8) \leq 4096$$

max we can do

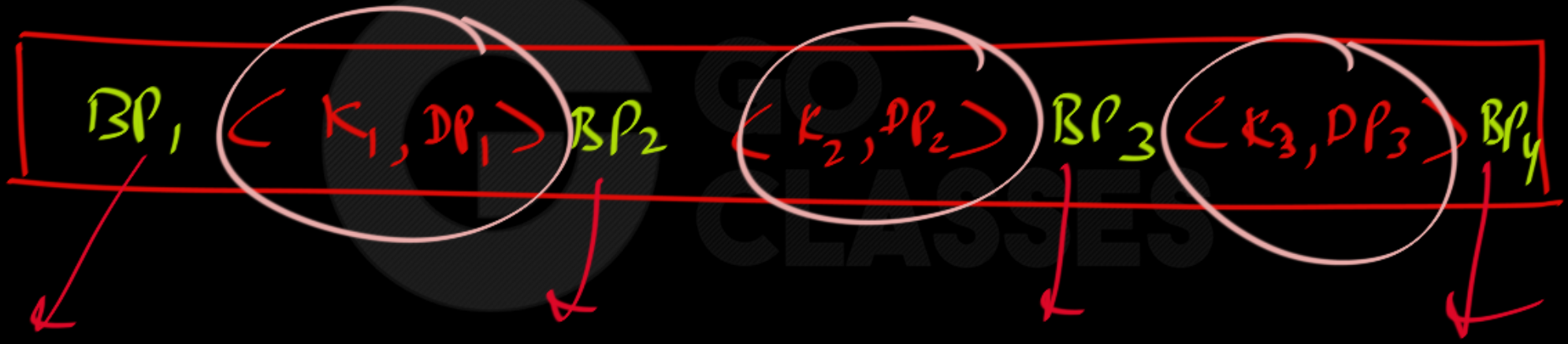
$$8P + 12P - 12 \leq 4096$$

$$P \leq \frac{4108}{20} = 205.4$$

$$P \leq 205.4 \Rightarrow \max P = 205$$

B Tree:

Structure of EVERY Node:

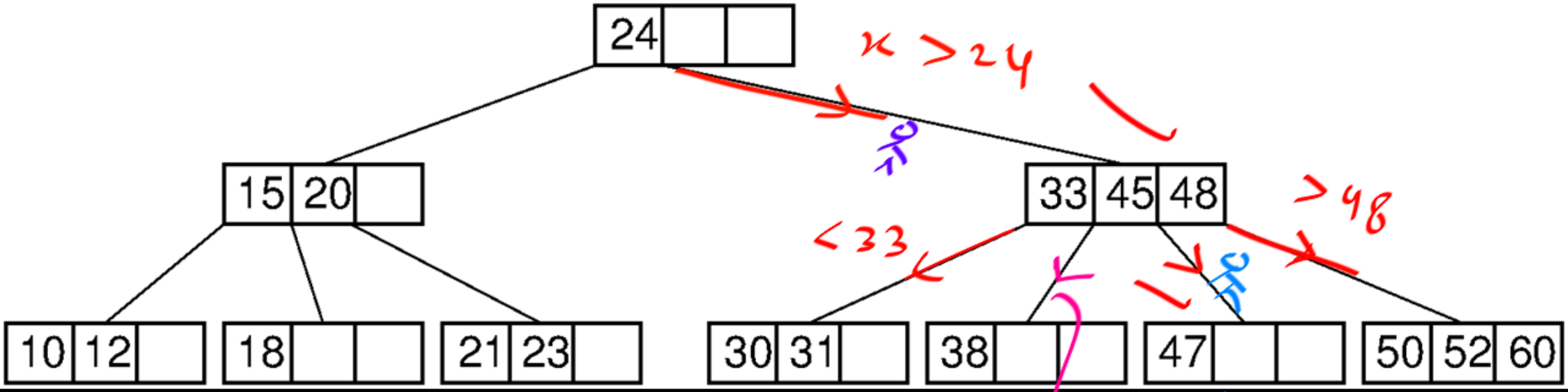


Data pointer < Record Pointer (by default) OR Block pointer

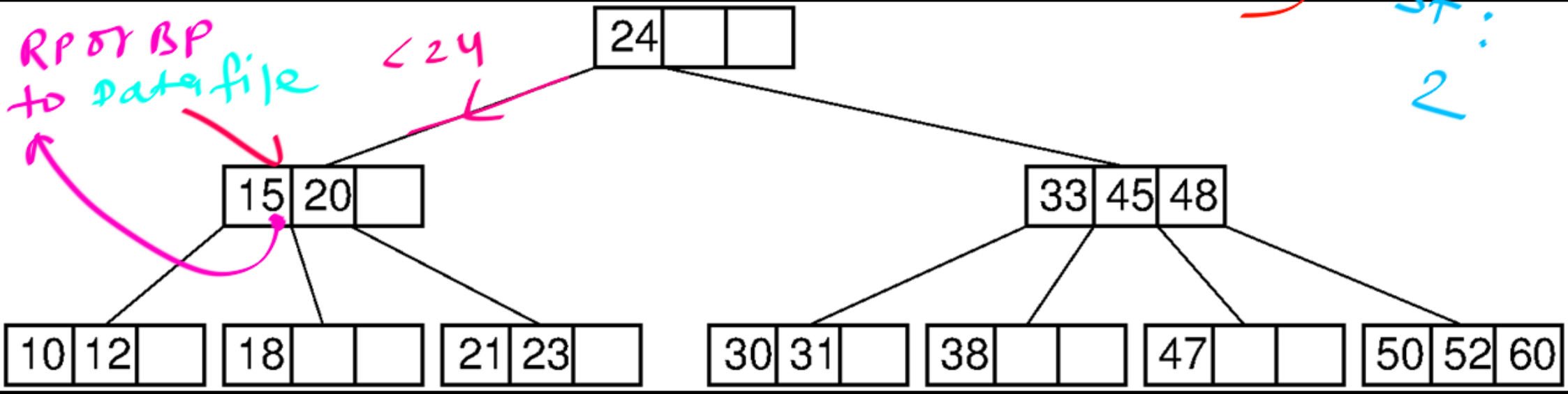
Next Topic:

B Tree Searching obvious Algorithm

Search 46 : → # Disk Block Access = 3

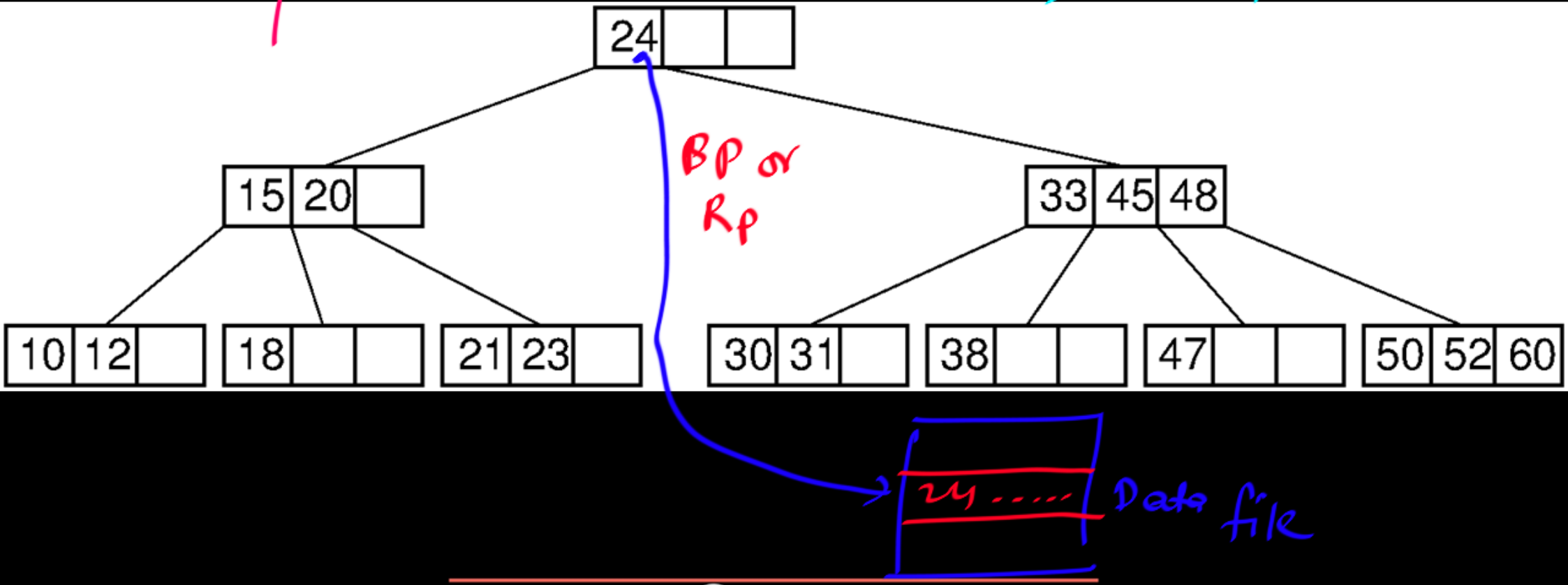


Search B Tree Index for search key = 15 } I/O cost : 2

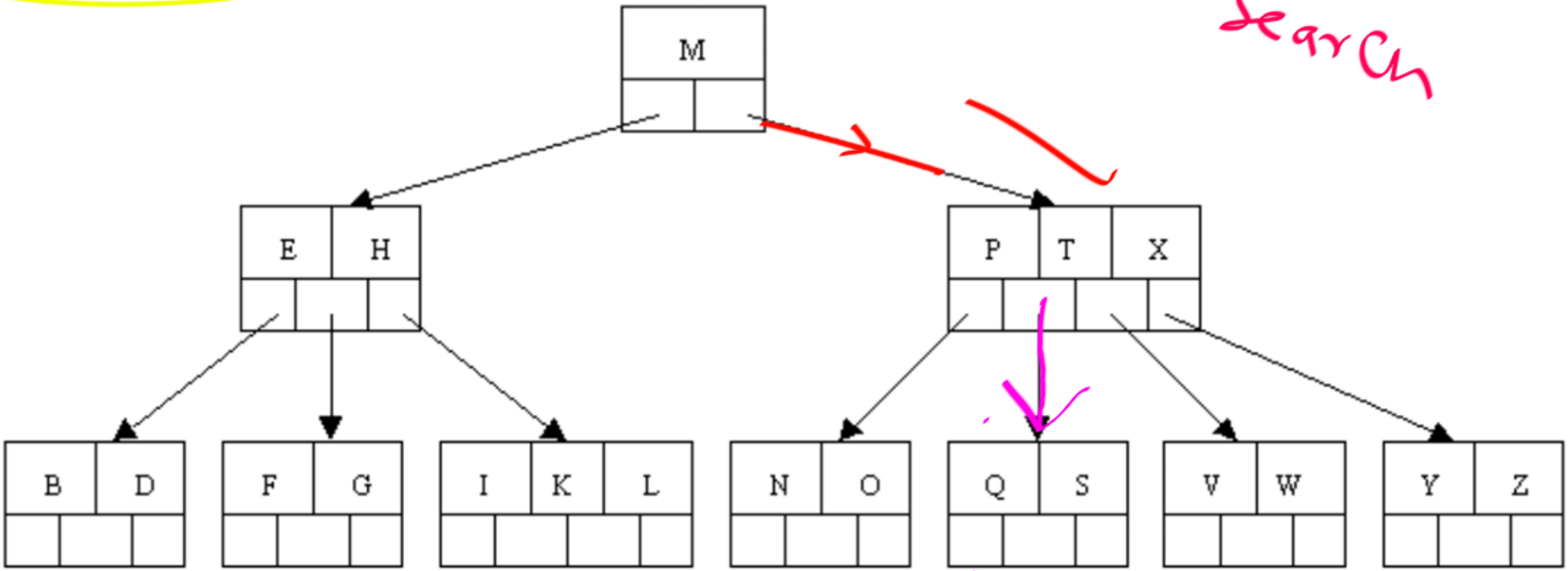


I/O cost for Data Record having Search key value = 15 ⟹ 2 + 1

I/o Cost to find Record (Data Record) having search key value 24 ⟹ 1+1



Search R ; → Unsuccessful search



B Tree:

Every Unsuccesful search goes till the Leaf level.

Successful search Can Stop as soon as we find search key velue in any node of B Tree.

Searching in a B-Tree

- Doing a search in a B-tree involves
  - searching the root node for the key, and
  - if it's not found, recursively exploring the correct child.
- Using binary search within a given node, can find the key or the correct child in time O(log *number-of-keys*).
- Repeat this process O(*tree-height*) times.
- Time complexity is

$$O(\log \textit{number-of-keys} \cdot \textit{tree-height})$$

Time Complexity of Search:

Worst Case: $\Rightarrow O($ Height $*$ 

go til Leaf level

within node

$O(p-1) = O(p)$

Search within Node :

→ can do Binary search

→ p-1 keys

Search time : $O(\log_2 p)$

Reason :
Binary search

Time Complexity of Search:

- Worst Case: $\Rightarrow O($Height $*$  $)$
  - go til Leaf level
  - within node
  - $O(\log_2 p)$

B Tree ; ⟹ n keys

$O(\log_{p/2} n)$

Worst Case Height : (maximum Height)

Time Comp of search in B Tree having n keys :

$O\left(\log_{p/2} n * \log_2 p/2 \right)$

max Height

Every node has O(p/2) keys

within Node binary search

Note:

$$\log_x y = \frac{\log_z y}{\log_z x}$$

$$O\left(\left(\log_{p/2} n\right) * \log_2 p/2\right) = O\left(\log_2 n\right)$$

$$\log_{p/2} n \rightarrow \frac{\log_2 n}{\log_2 p/2}$$

Recap:

Q: a B Tree with n keys & order p:

Search time Comp ?

(a) $\log_p n$

(b) $\log_2 n$

(c) $\log_{p/2} n$

Recap:

Q: a B Tree with n keys

& order p:

Search time Comp ?

(a) $\log_p n$

(b) $\log_2 n$

(c) $\log_{p/2} n$

Recap:

Q: a B Tree with n keys & order p:

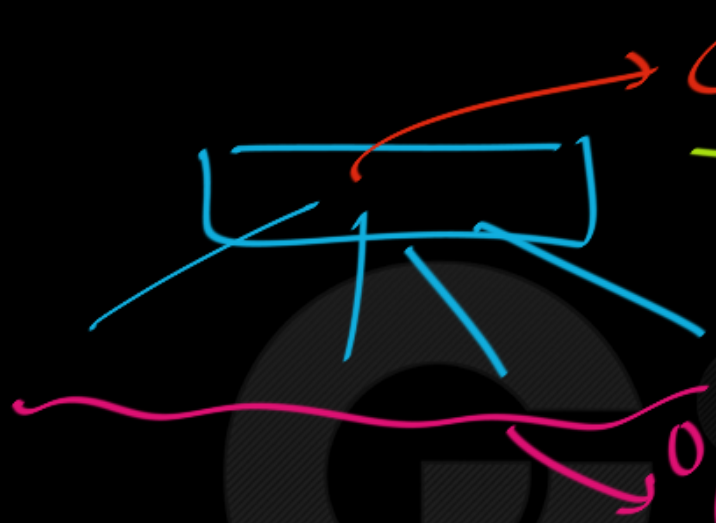
Search time Comp ?

Worst Case: max Height

n keys

To get max Height: Put minimum no. of keys in every node

Put $O(P/2)$ keys per Node $= O(P/2)$



$O(p/2)$ keys

$O(p/2)$ Tree Pointers

$O(p/2)$ Tree pointers

n keys

$n \rightarrow \frac{n}{p/2} \rightarrow \frac{n}{(p/2)^2} \rightarrow \cdots$

Height = $O(\log_{p/2} n)$

n keys

→ Max Height = $O(\log_{p/2} n)$

→ Per Node #keys = $O(p/2)$

Within a Node : $(\log_2 p/2)$

B Tree with n keys ; order p :

Search Tc : $O\left(\log_{p/2} n * \log_2 p/2\right)$

$= O(\log_2 n)$

n keys :

Search time Comp :

- AVL Tree : $O(\log_2 n)$ ⇒ more no. of levels
- B Tree : $O(\log_2 n)$ → no. of levels less BUT within node search ← Takes more time

Next Topic:

B Tree Insertion

The algorithm for insertion takes an entry, finds the leaf node where it belongs, and inserts it there.

Insertion in B Tree ALWAYS happens at leaf Node.

B Tree of order p:

Root: 1 key to p-1 keys

All other Nodes: $\lceil \frac{p}{2} \rceil - 1$ keys to p-1 keys

B Tree of order 5

Root: 1 key to 4 keys

All other Nodes: 2 keys to 4 keys

B Tree of order 5 :

Insertion Case 1 :

→ Always at leaf

leaf is NOT full

then insert in sorted way.

leaf Node

NOT full

insert 7

B Tree of order 5 :

Insertion 2 : Leaf Node is full : Split

Leaf Node

2	6	7	8

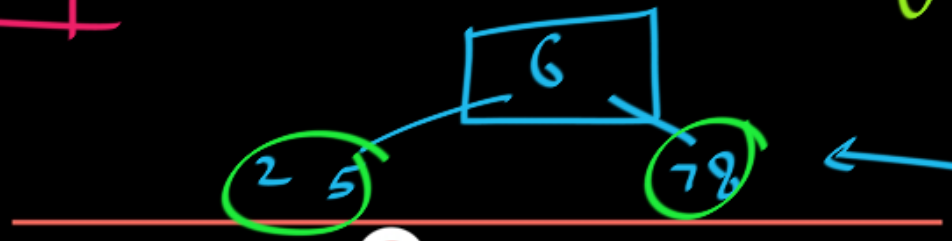
↳ Already full

insert 5 →

Ideally : violation

2	5	6	7	8

↳ Split



Insertion in B-Trees

- **OVERFLOW CONDITION:**

  A root-node or a non-root node of a B-tree of order m overflows if, after a key insertion, it contains m keys. (i.e. the node, before insertion was full and had m-1 keys)

- **Insertion algorithm:**

If a node overflows, split it into two, propagate the "middle" key to the parent of the node. If the parent overflows the process propagates upward. If the node has no parent, create a new root node.

- Note: Insertion of a key always starts at a leaf node.

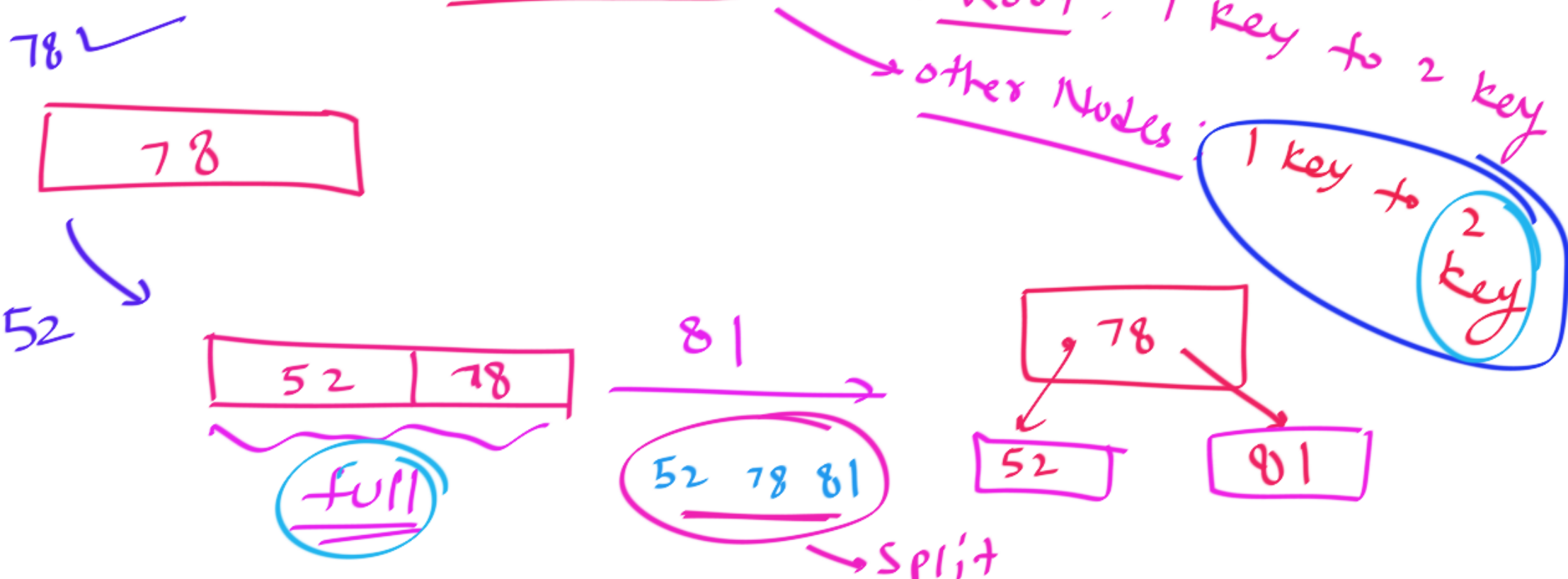
Insertion in B-Trees

- **Insertion in a B-tree of odd order**

- Example: Insert the keys 78, 52, 81, 40, 33, 90, 85, 20, and 38 in this order in an initially empty B-tree of order 3

Insertion in B-Trees

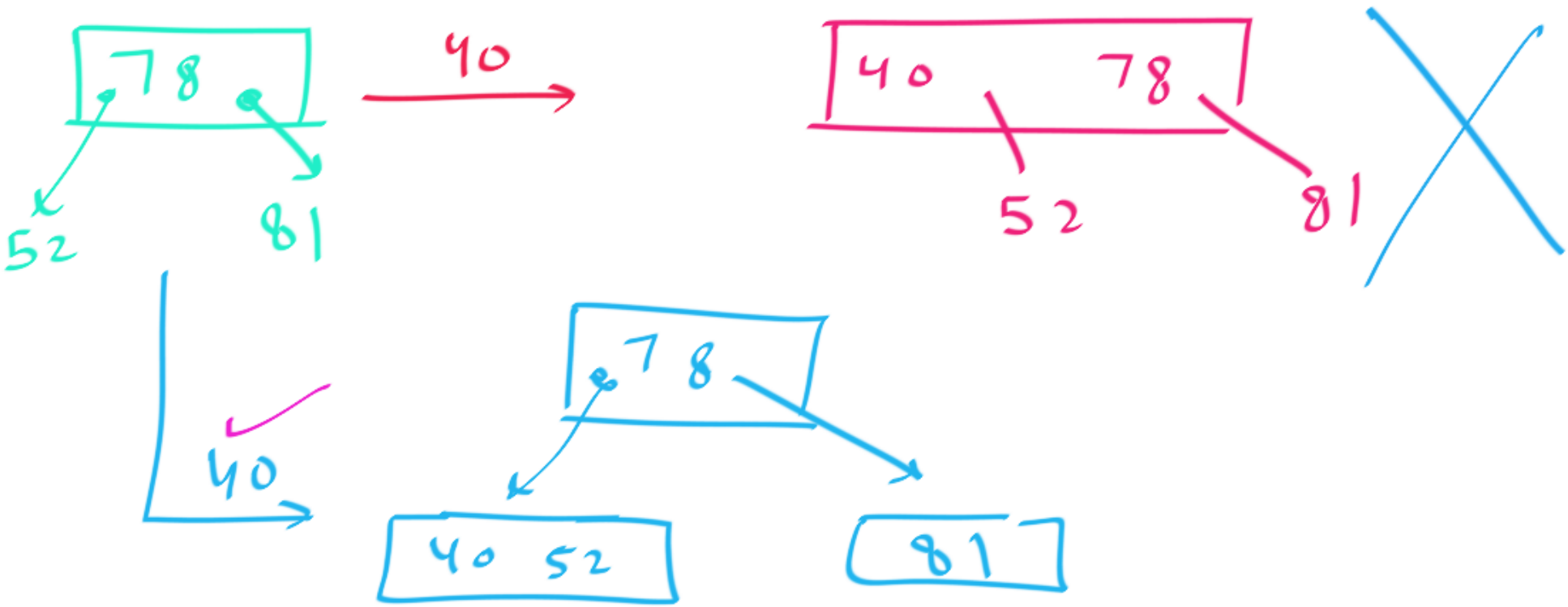
- **Insertion in a B-tree of odd order**
- Example: Insert the keys 78, 52, 81, 40, 33, 90, 85, 20, and 38 in this order in an initially empty B-tree of order 3



Insertion in B-Trees

- **Insertion in a B-tree of odd order**

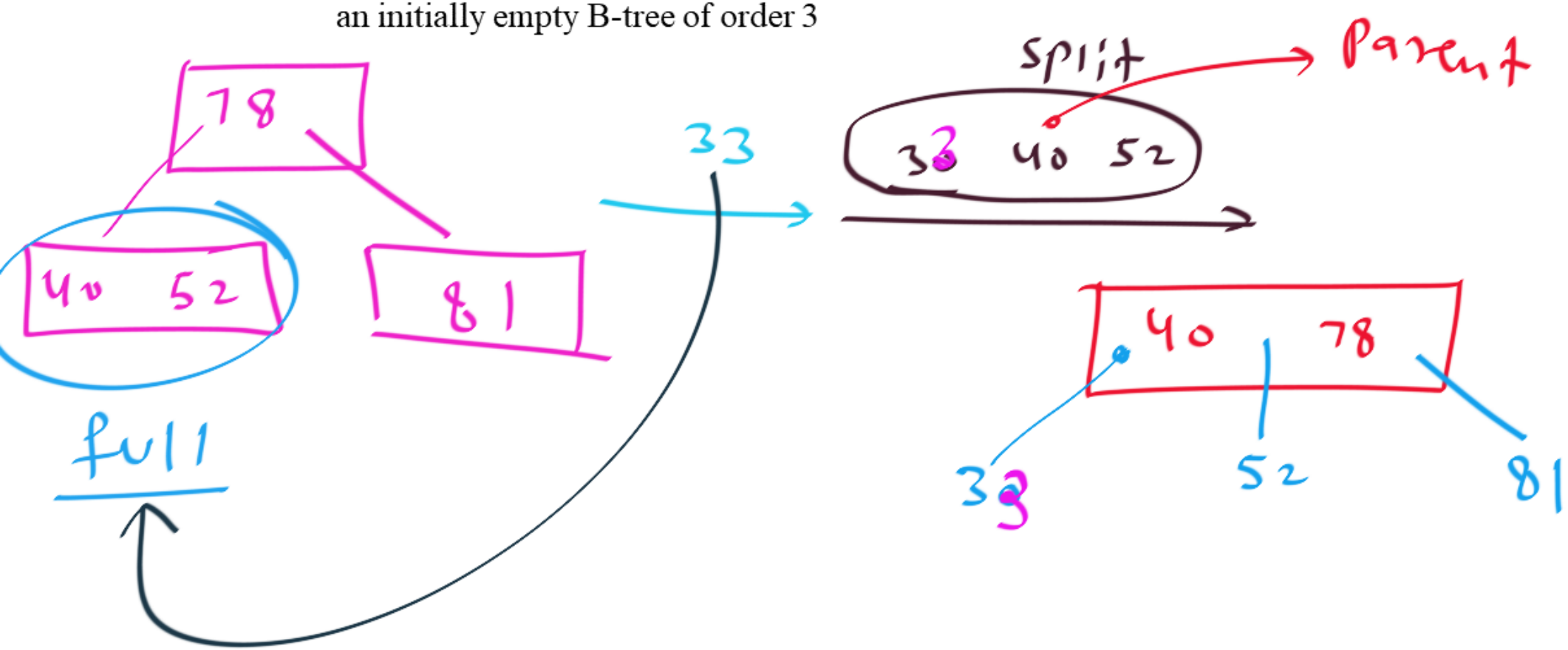
- Example: Insert the keys 78, 52, 81, 40, 33, 90, 85, 20, and 38 in this order in an initially empty B-tree of order 3



Insertion in B-Trees

- **Insertion in a B-tree of odd order**

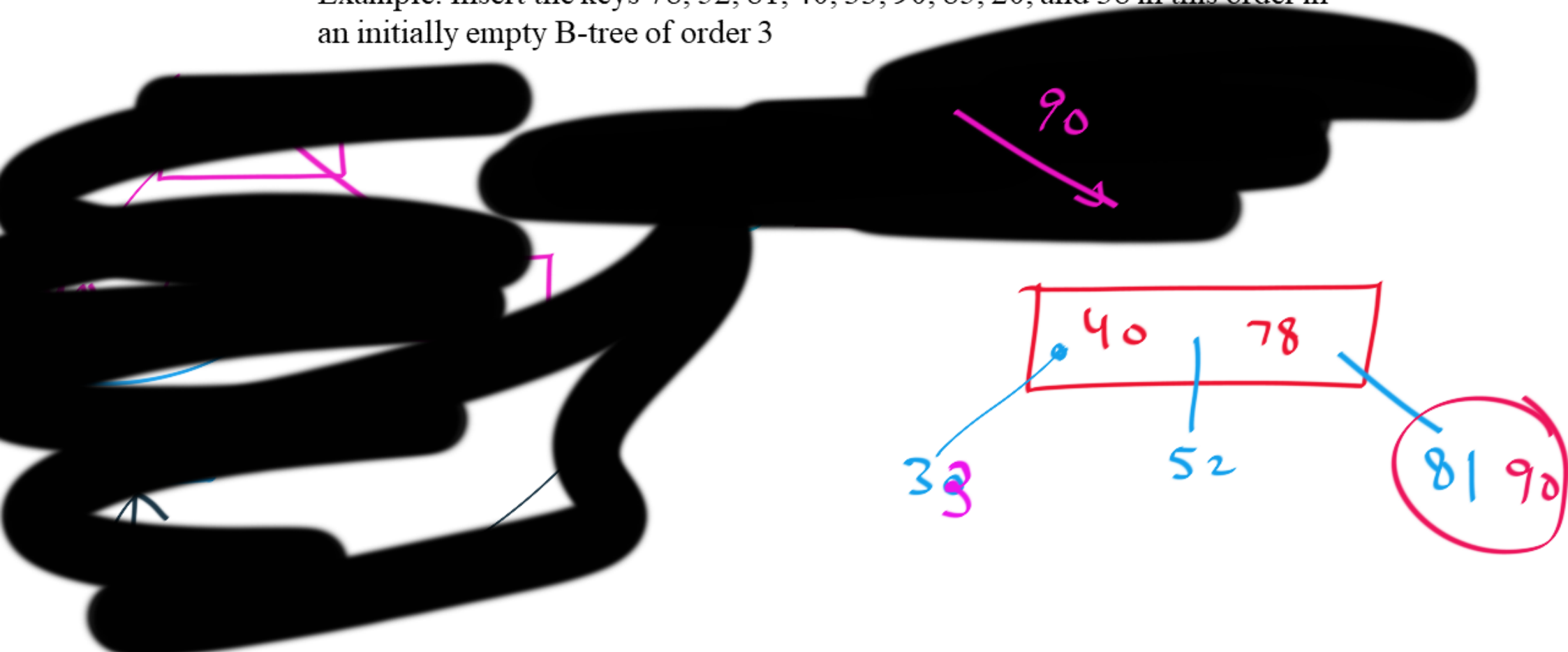
- Example: Insert the keys 78, 52, 81, 40, 33, 90, 85, 20, and 38 in this order in an initially empty B-tree of order 3



Insertion in B-Trees

- **Insertion in a B-tree of odd order**

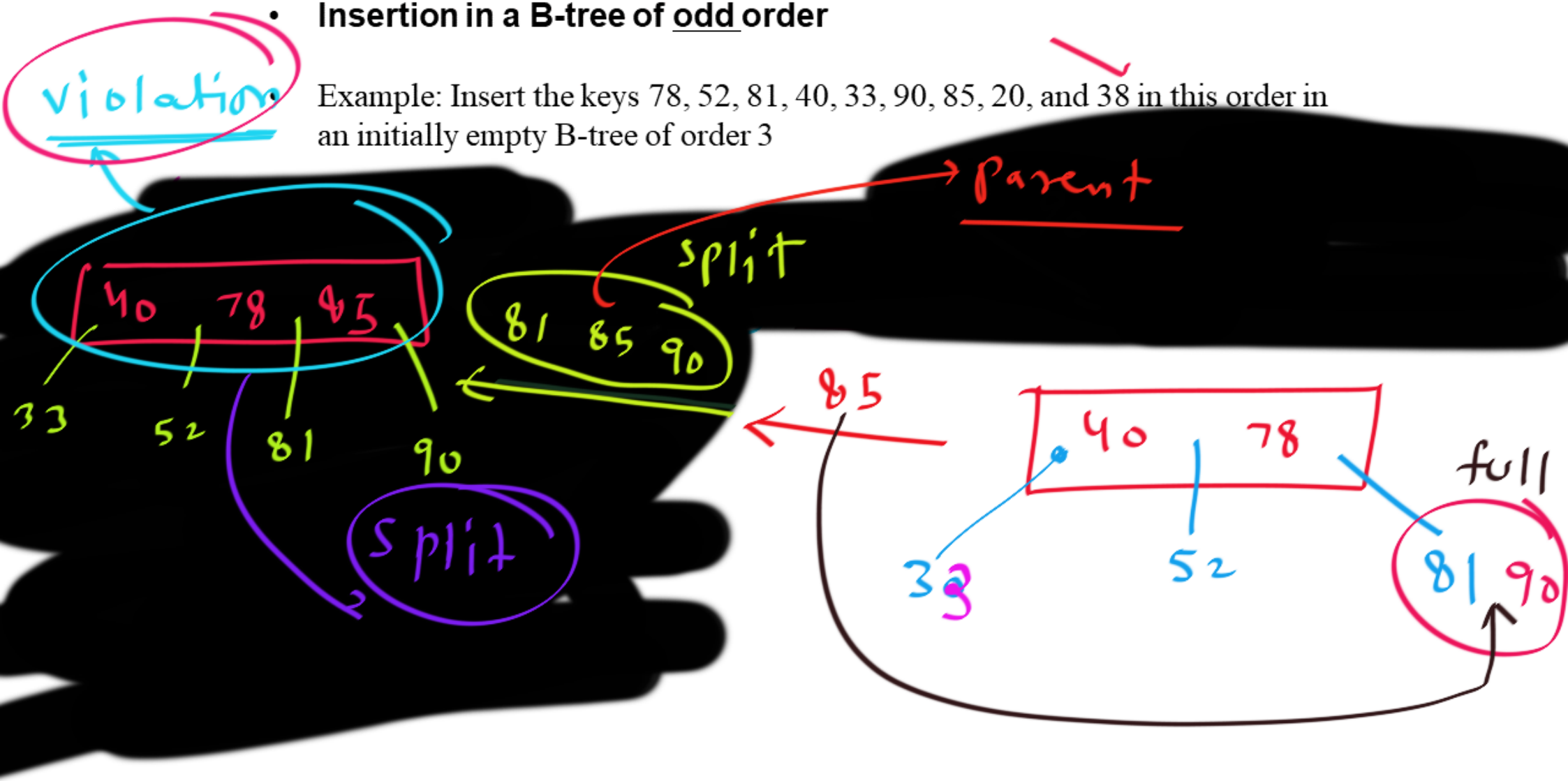
- Example: Insert the keys 78, 52, 81, 40, 33, 90, 85, 20, and 38 in this order in an initially empty B-tree of order 3

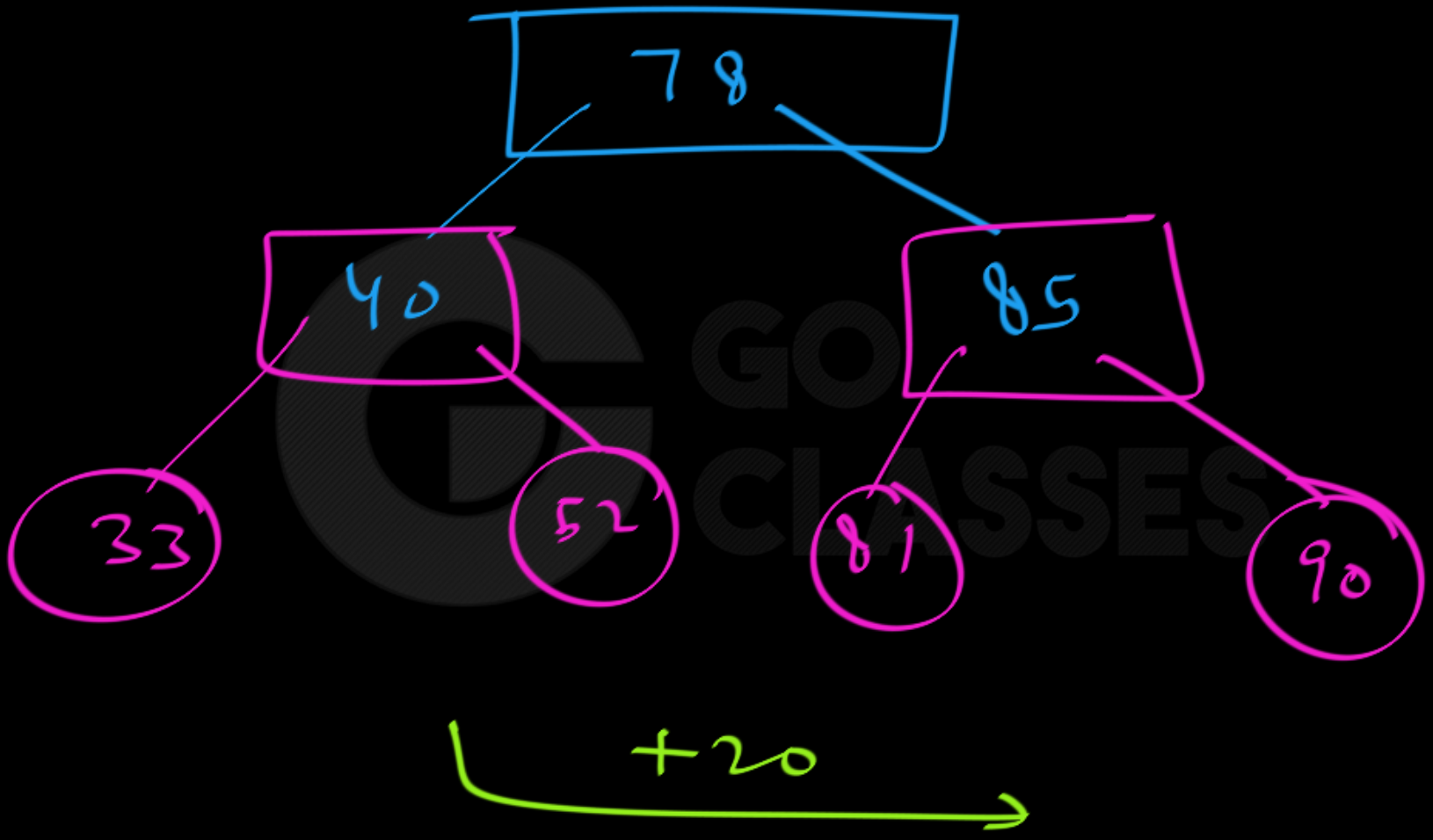


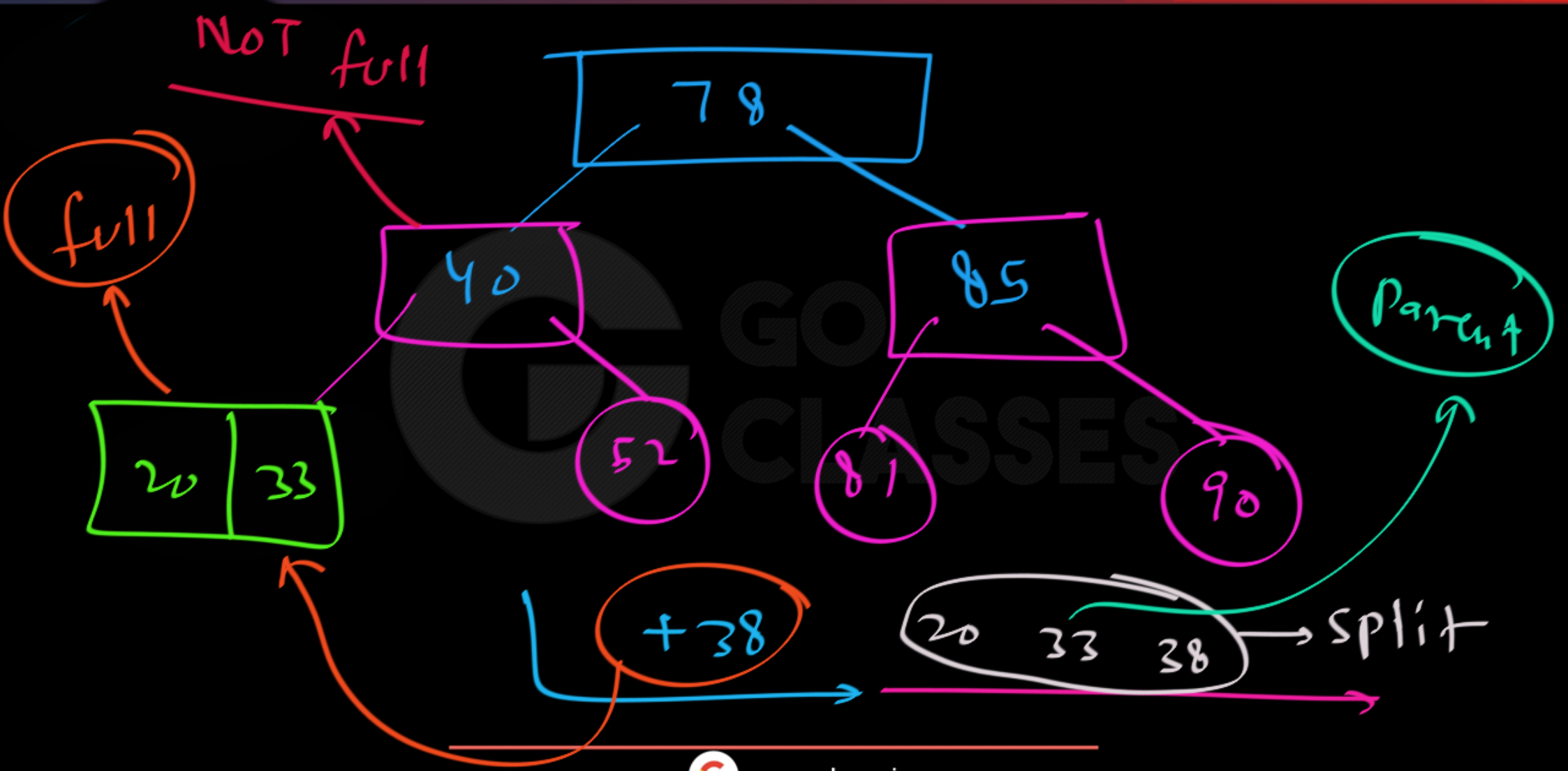
Insertion in B-Trees

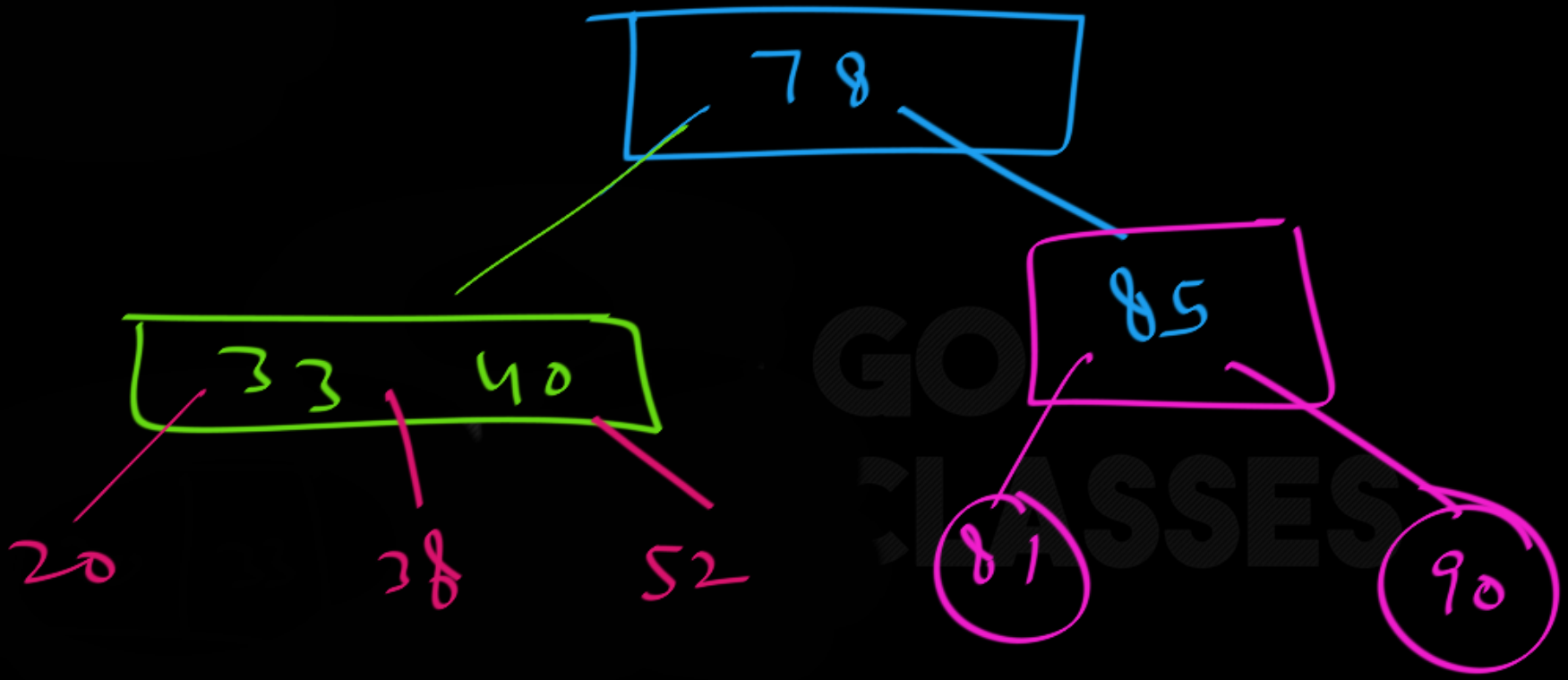
- **Insertion in a B-tree of odd order**

Example: Insert the keys 78, 52, 81, 40, 33, 90, 85, 20, and 38 in this order in an initially empty B-tree of order 3




78
40
85
33
52
81
90
+20

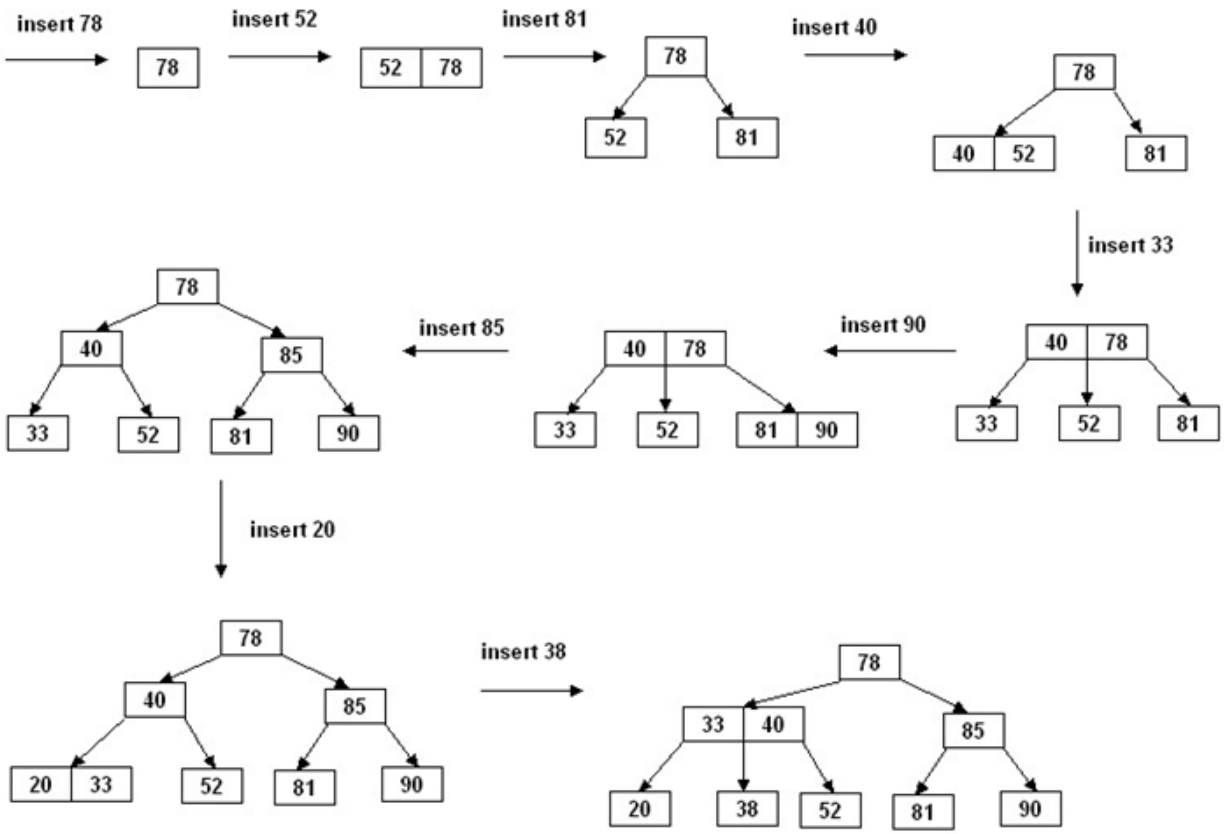

NOT full
78
full
40
85
Parent
20
33
52
81
90
+38
20
33
38
split


78
33
40
85
20
38
52
81
90

Insertion in B-Trees

- **Insertion in a B-tree of odd order**

- Example: Insert the keys 78, 52, 81, 40, 33, 90, 85, 20, and 38 in this order in an initially empty B-tree of order 3



Insertion in B Tree : → Always start at Leaf

(1) Leaf Not full : → Just put it in sorted order.

(2) Leaf full But Parent NOT full :

Split leaf & put middle element in parent.

Insertion in B Tree : → Always start at Leaf

③ Leaf full & Parent Also full :

Split leaf & put middle element in parent.

Split Parent Also

Q:

Creating a B tree of order 5.

A G F B K D H M J E S I R X C L N T U P

Q:

Creating a B tree of order 5.

min #keys = 2

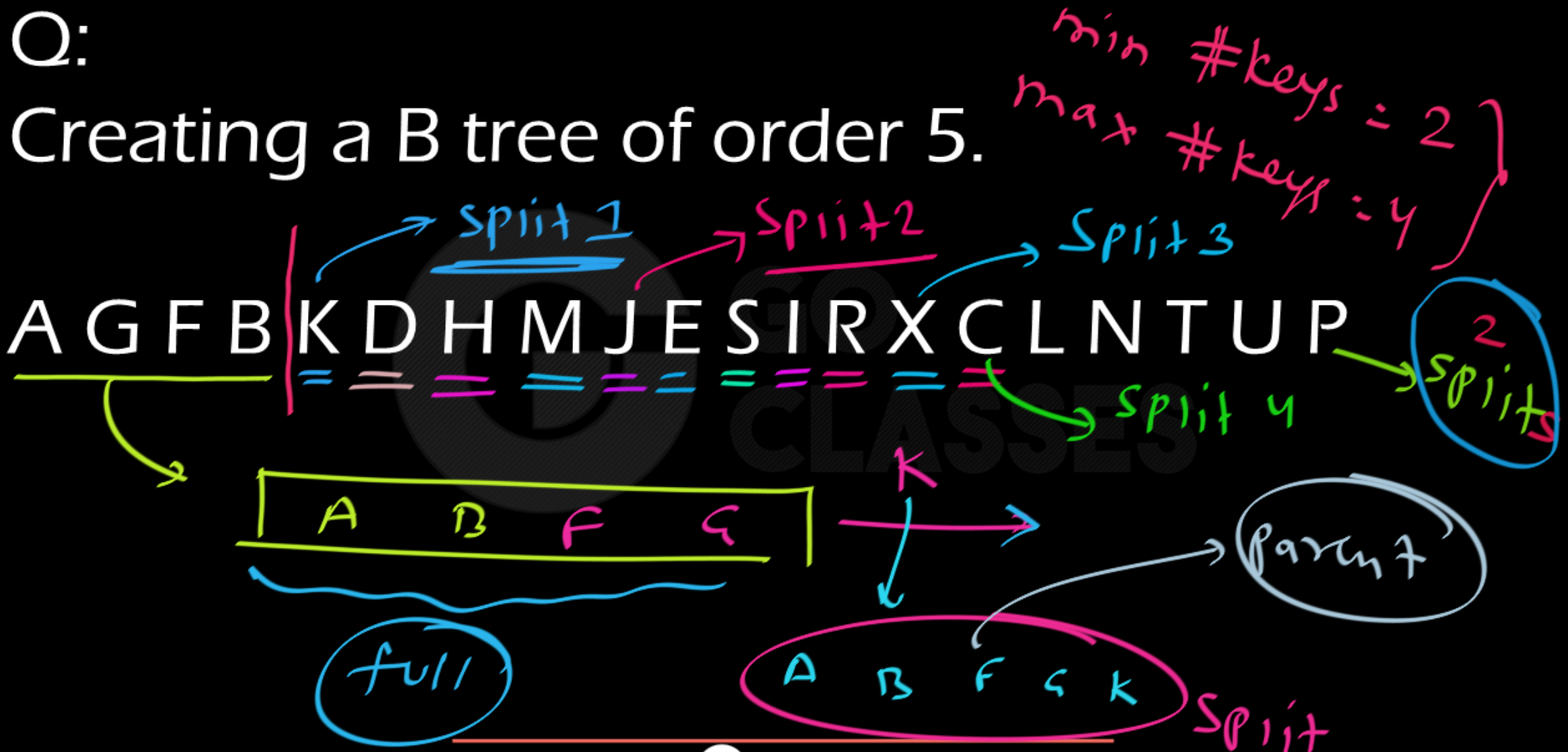
max #keys = 4

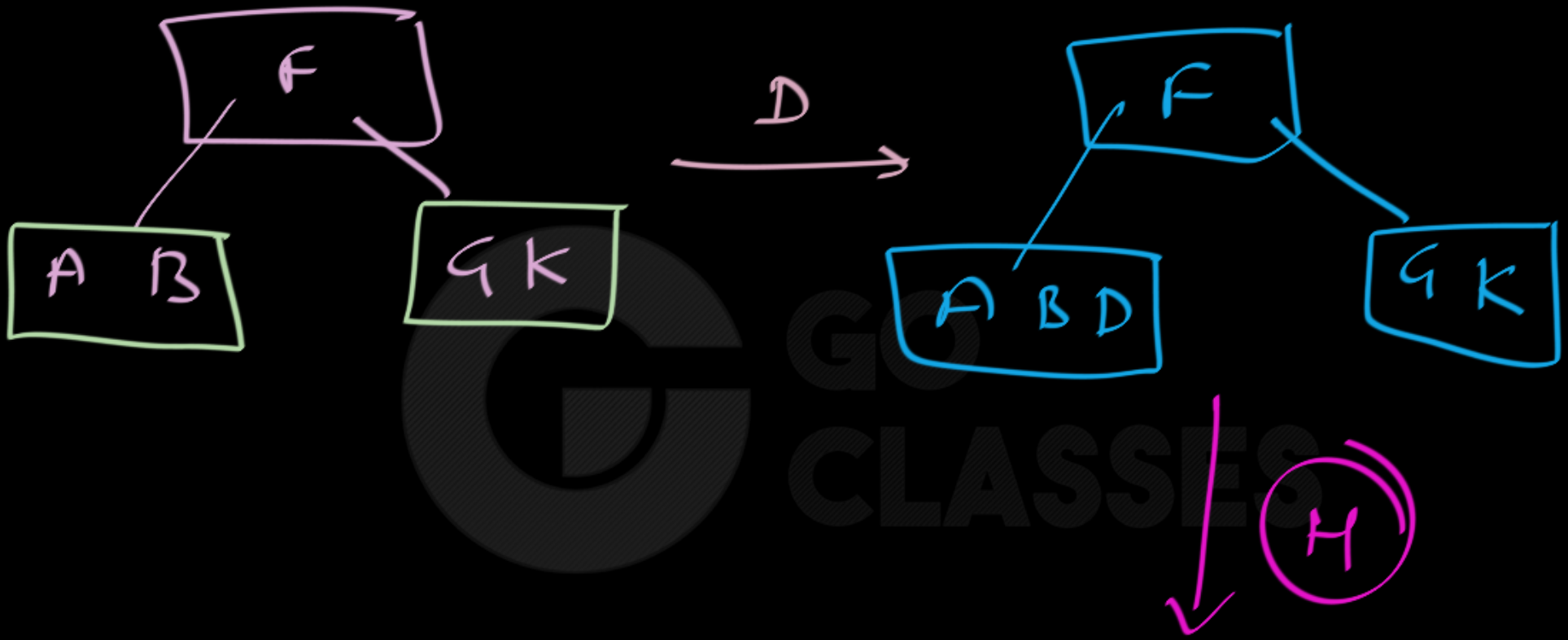
A G F B K D H M J E S I R X C L N T U P

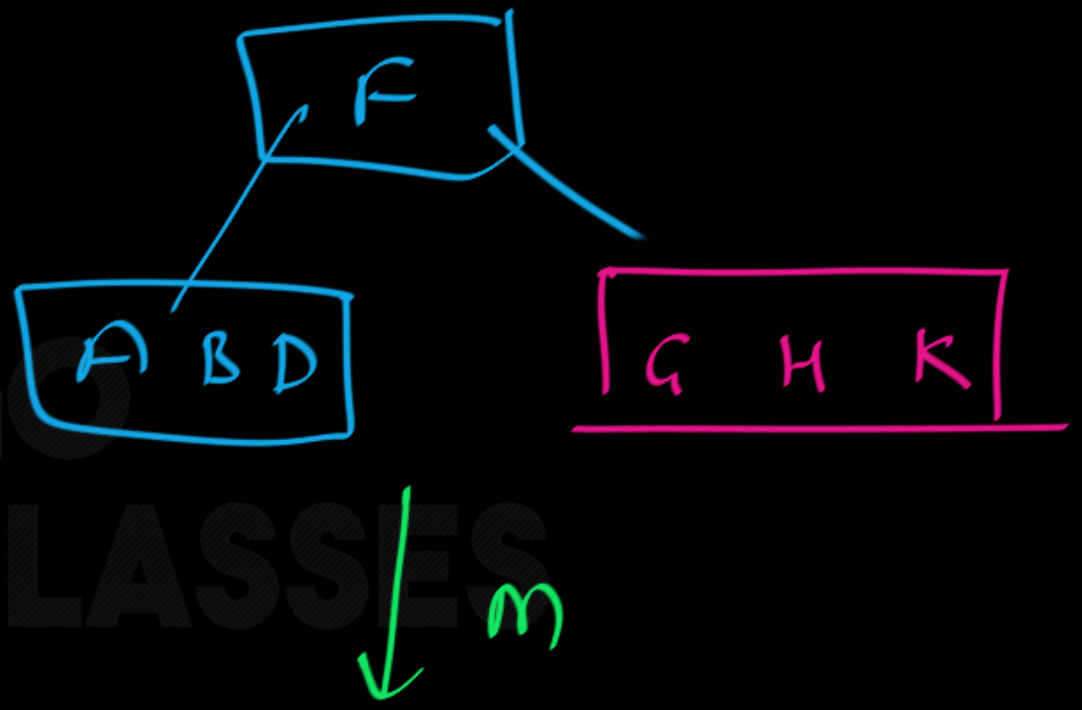
Leaf full when #keys = 4

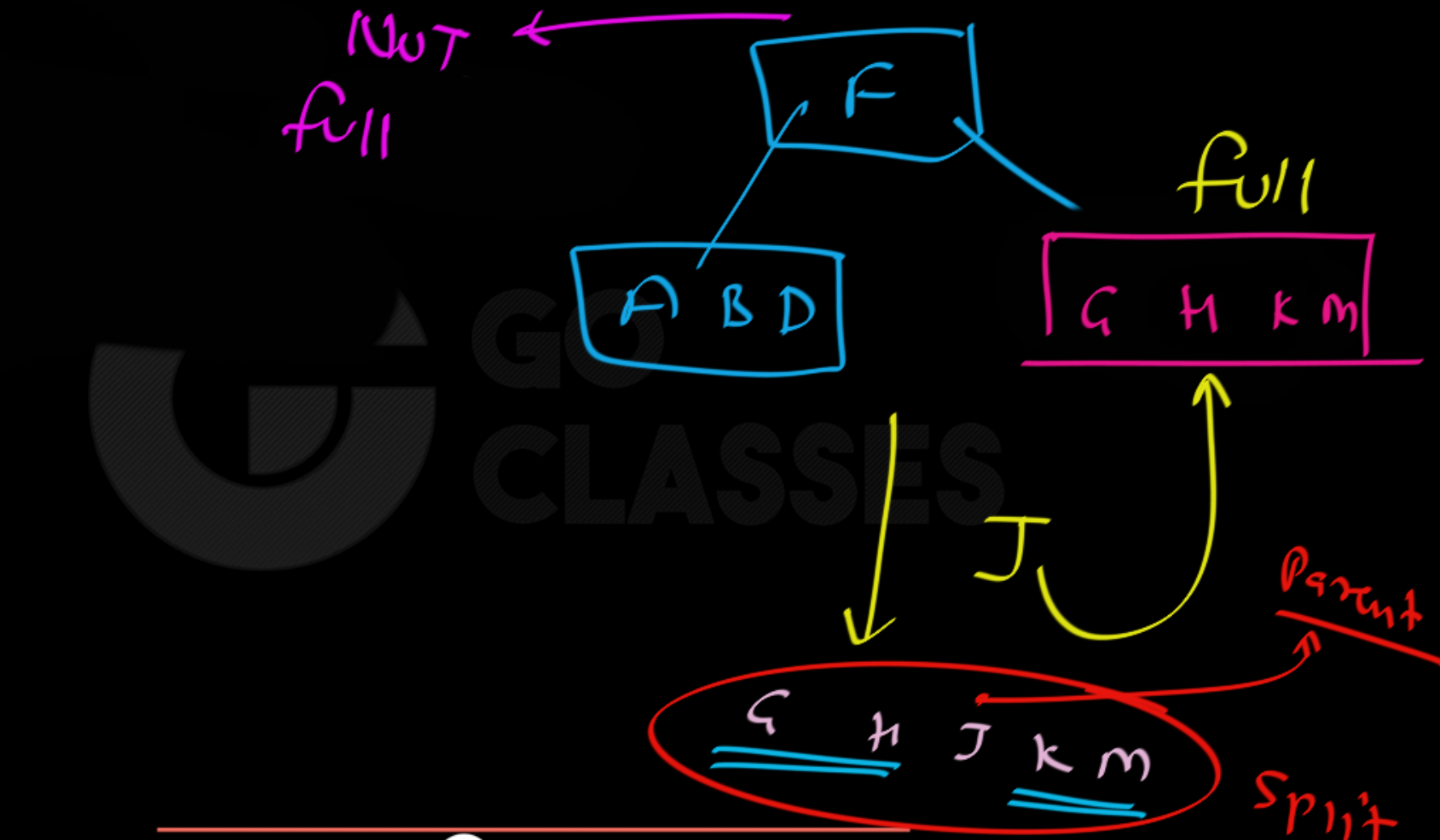
Q:

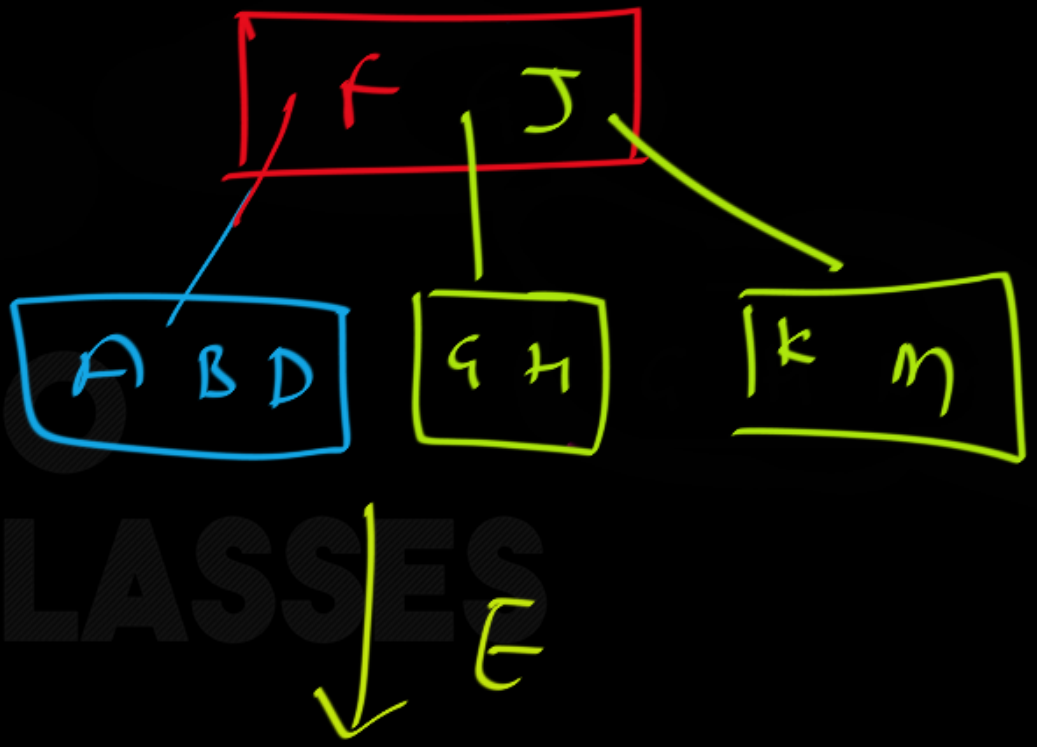
Creating a B tree of order 5.

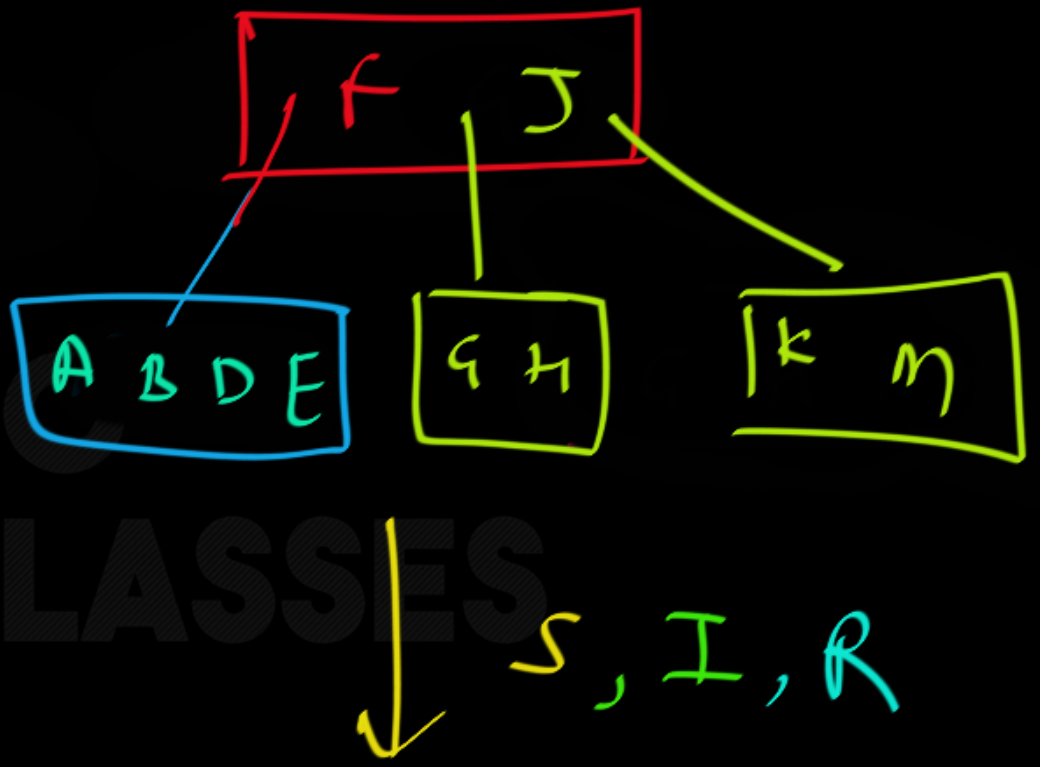


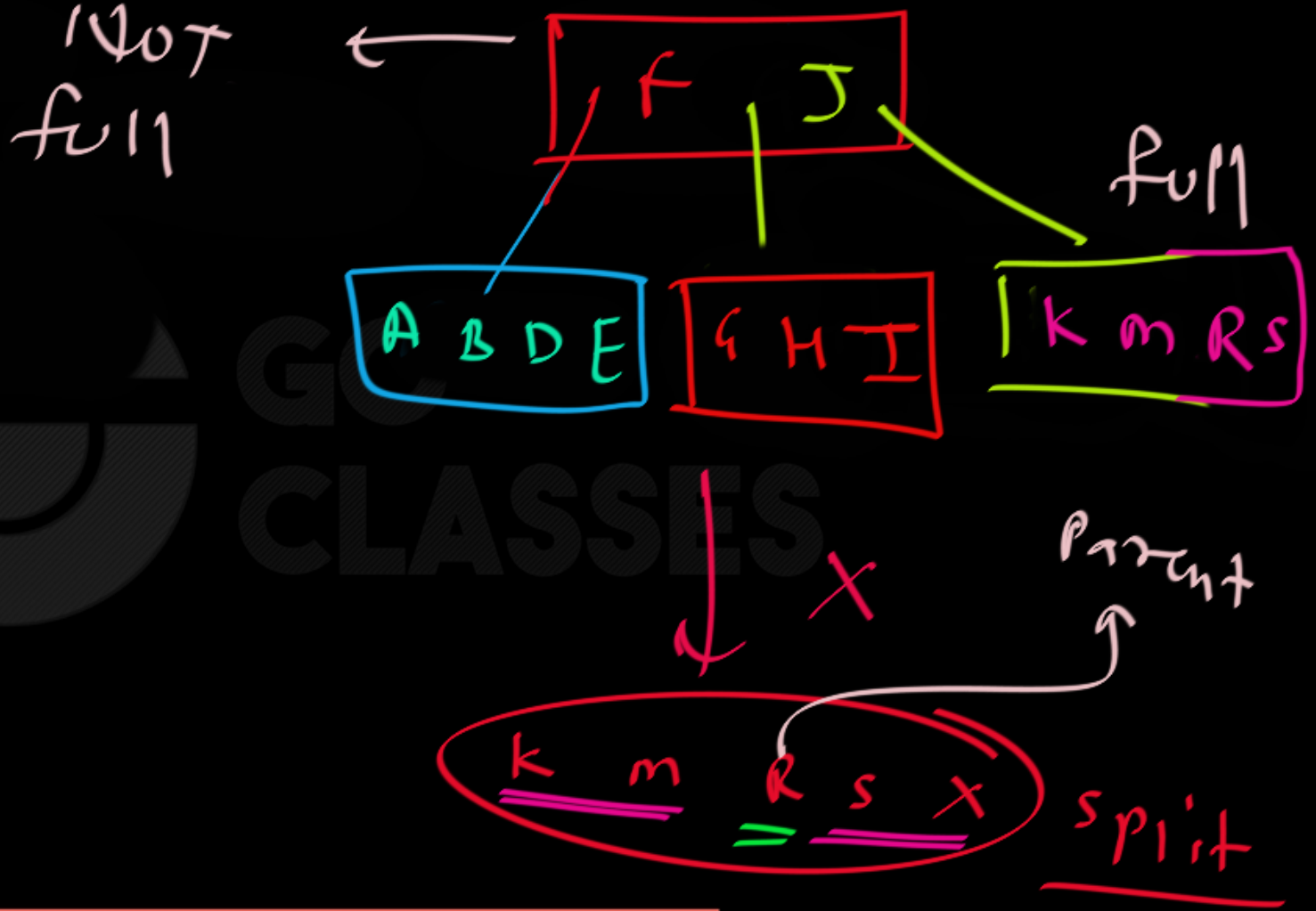

F
A B
G K
D
F
A B D
G K
H

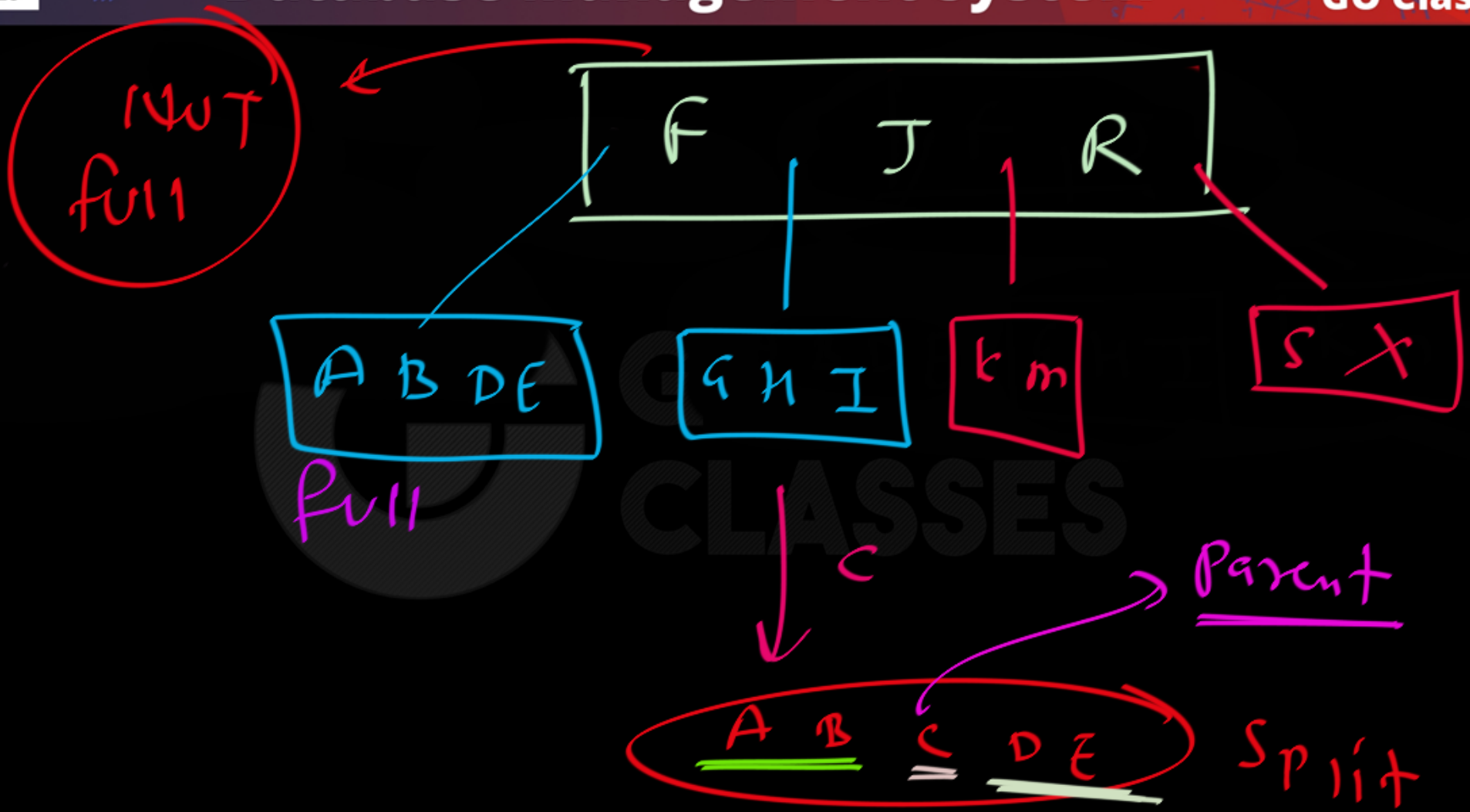

F
A B D
G H K
m

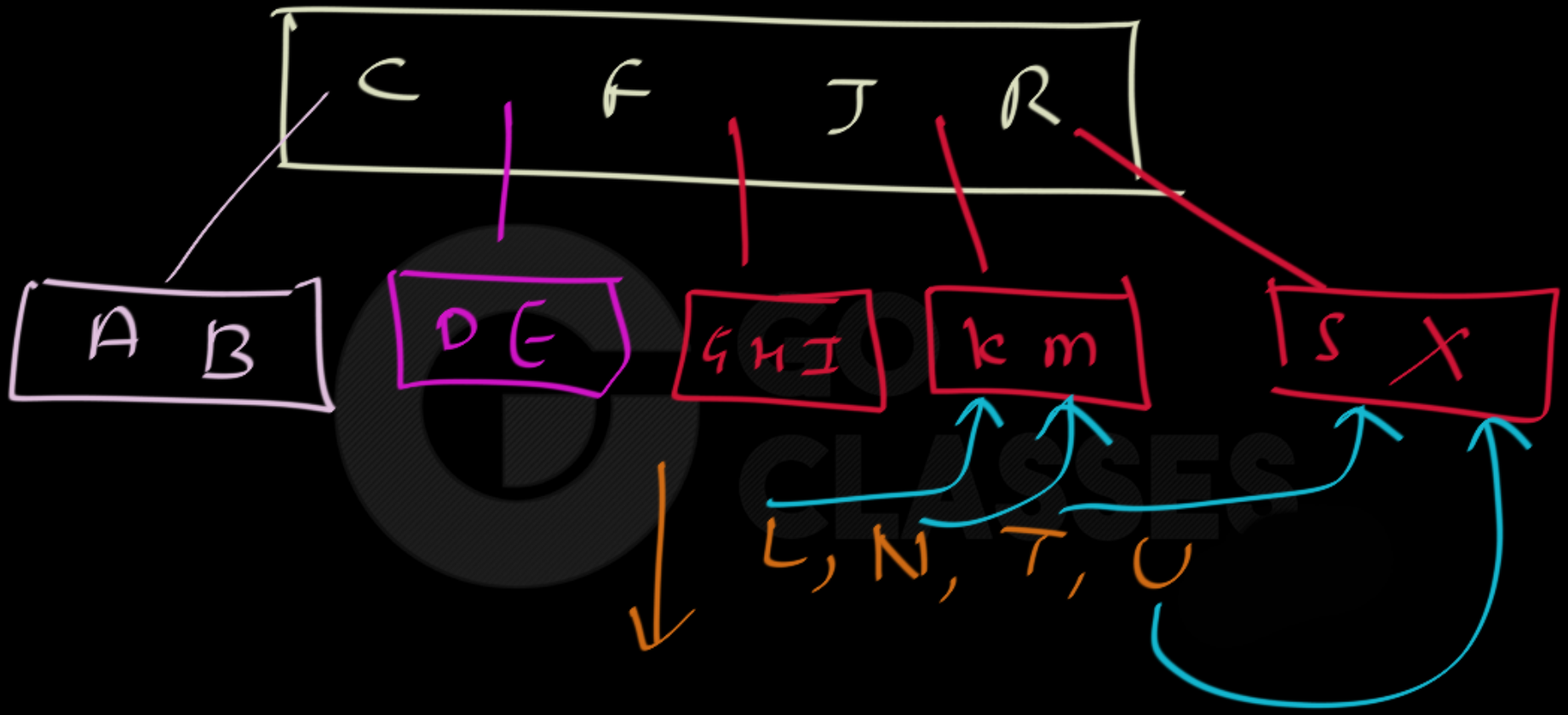

NOT
full
F
A B D
full
G H K M
J
Partial
G H J K M
Split

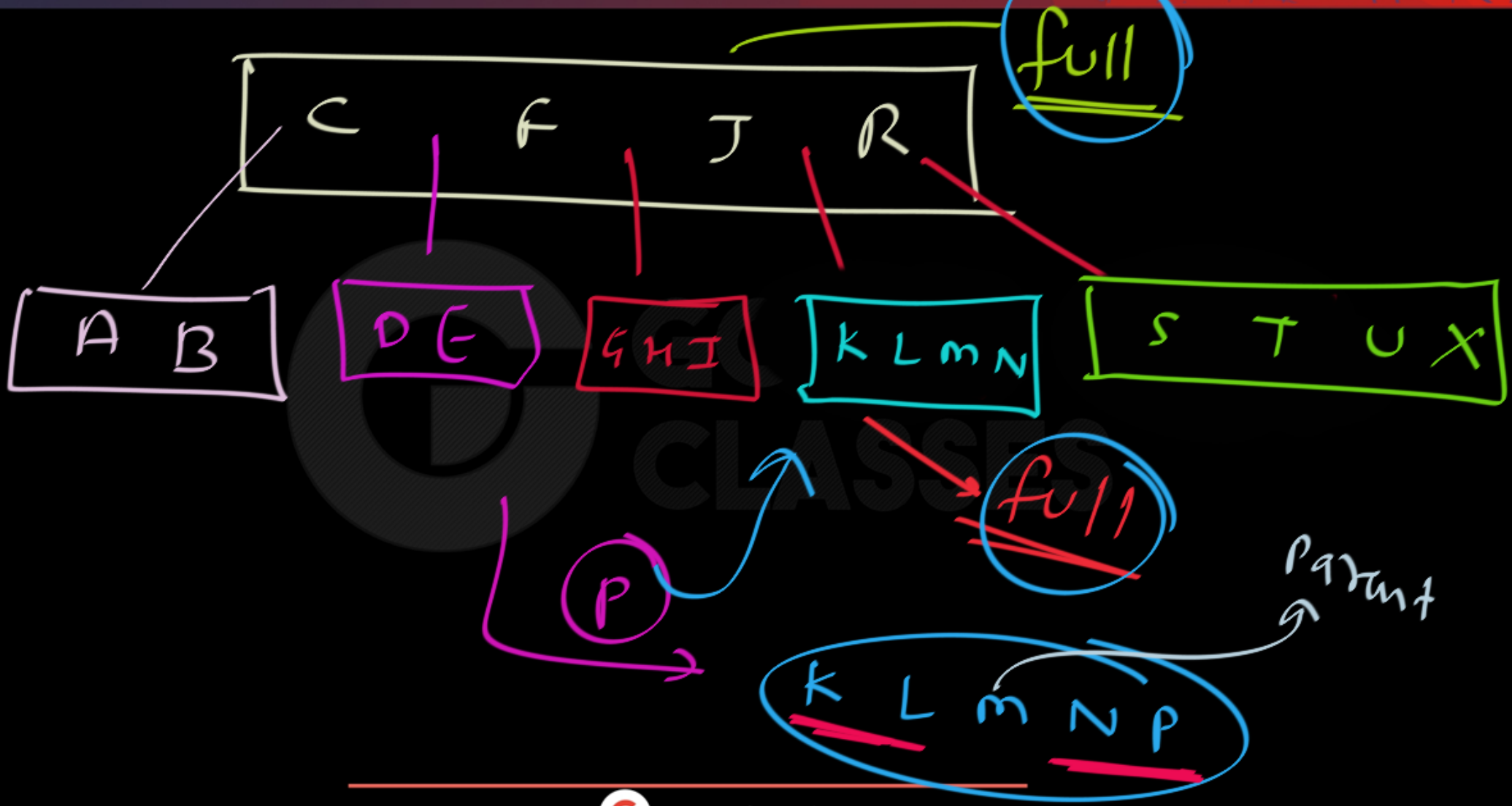

F J
A B D
G H
K M
E

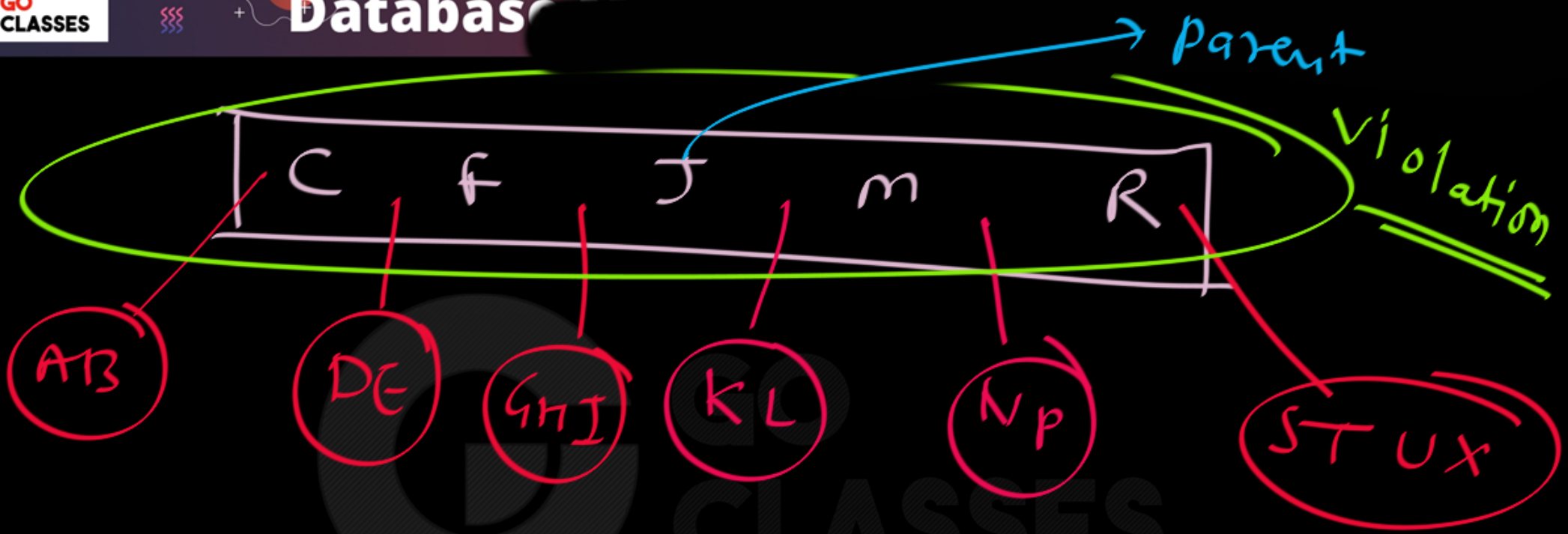

F J
A B D E
G H
K M
S, I, R

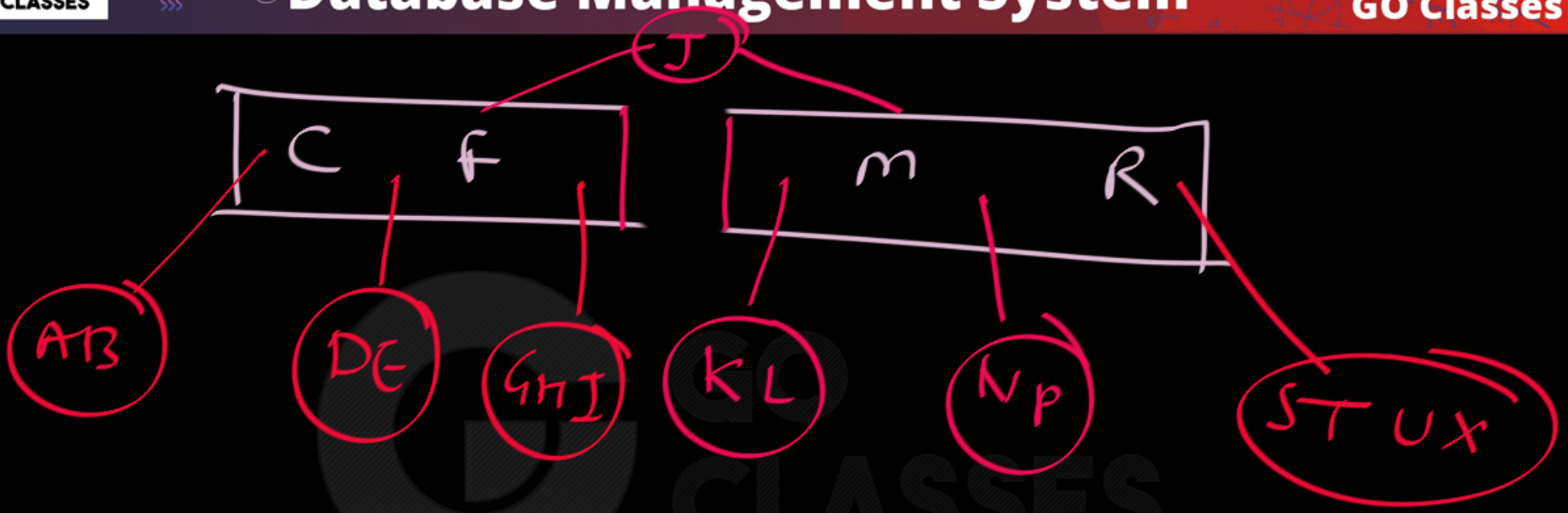

Not full
F J
full
A B D E
G H I
k m R s
X
Parent
k m R s X
split


INVT
full
F
J
R
A B D E
G H I
k m
S X
full
C
Parent
A B C D E
Split


C
F
J
R
A B
D E
G H I
k m
S X
L, N, T, U


C F J R
full
A B
D E
G H I
K L M N
S T U X
full
P
K L M N P
Parent

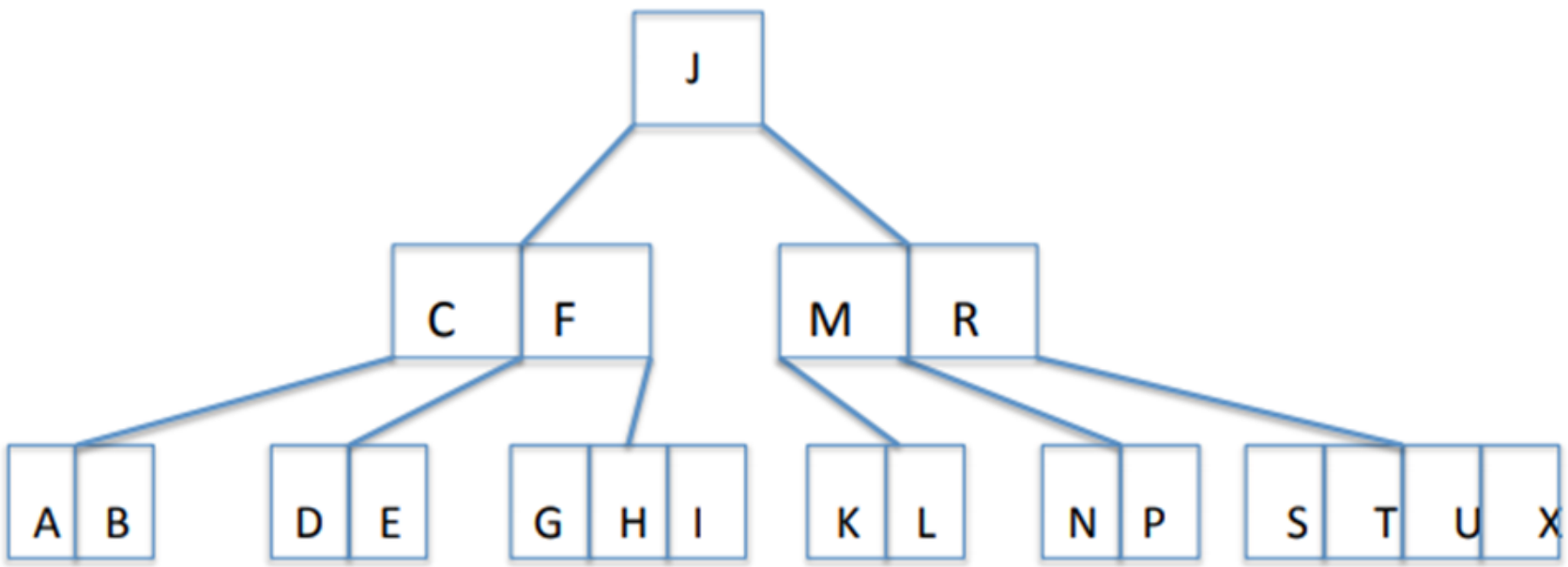

Parent
Violation
C
F
J
m
R
AB
DE
GHI
KL
NP
STUX



#Splits = 6

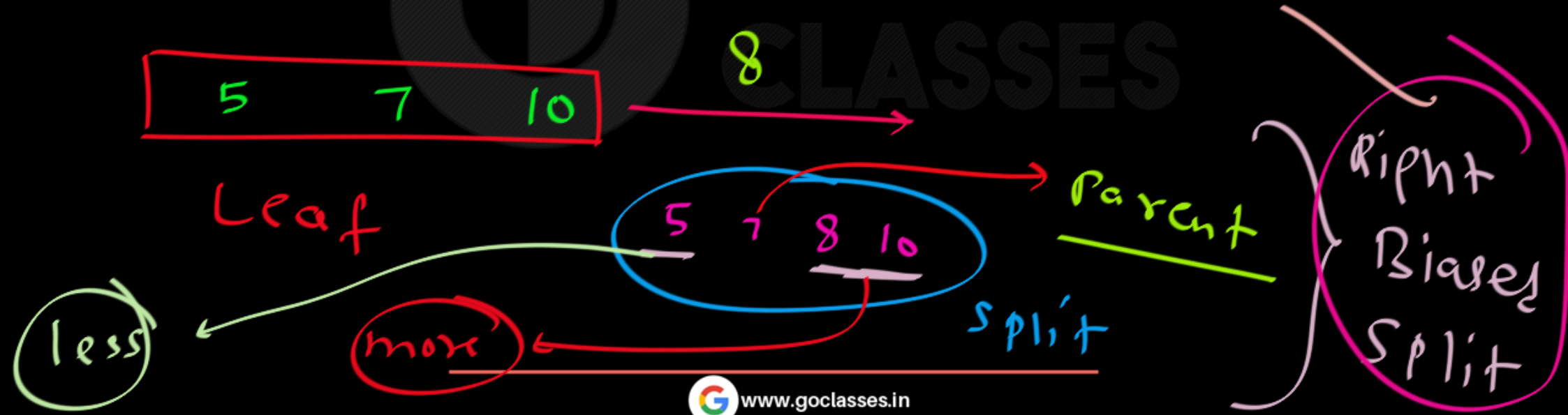
Creating a B-tree of order 5

A G F B K D H M J E S I R X C L N T U P



B Tree Order is Even :

Order 4 → #key = 1 to 3



B Tree Order is Even :

Order 4 : → #key = 1 to 3

Note :

Always Stick to Left OR Right Biased BUT only one of them

Consistently

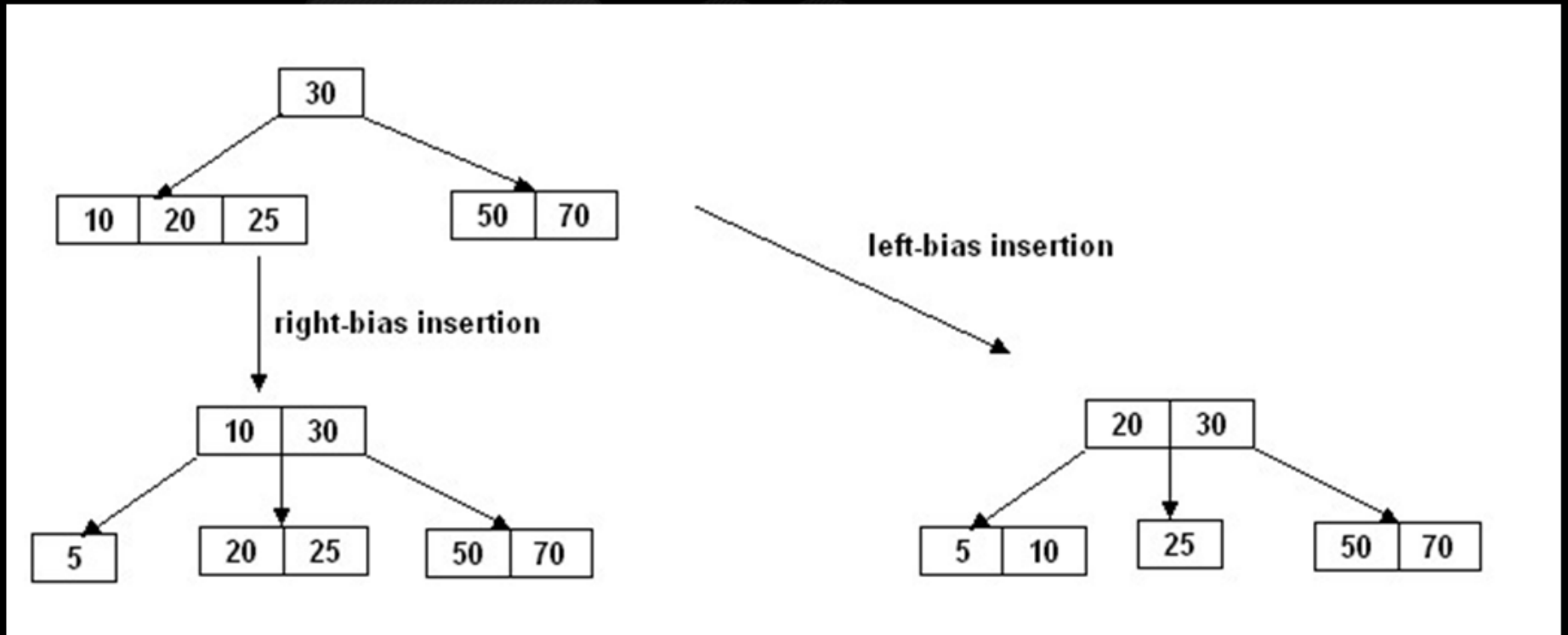
Do NOT mix them

- **Insertion in a B-tree of <u>even</u> order**

At each node the insertion can be done in two different ways:

- **right-bias:** The node is split such that its right subtree has more keys than the left subtree.
- **left-bias:** The node is split such that its left subtree has more keys than the right subtree.

- **Example:** Insert the key **5** in the following B-tree of order **4**:



Q: HW

1 to 3 keys

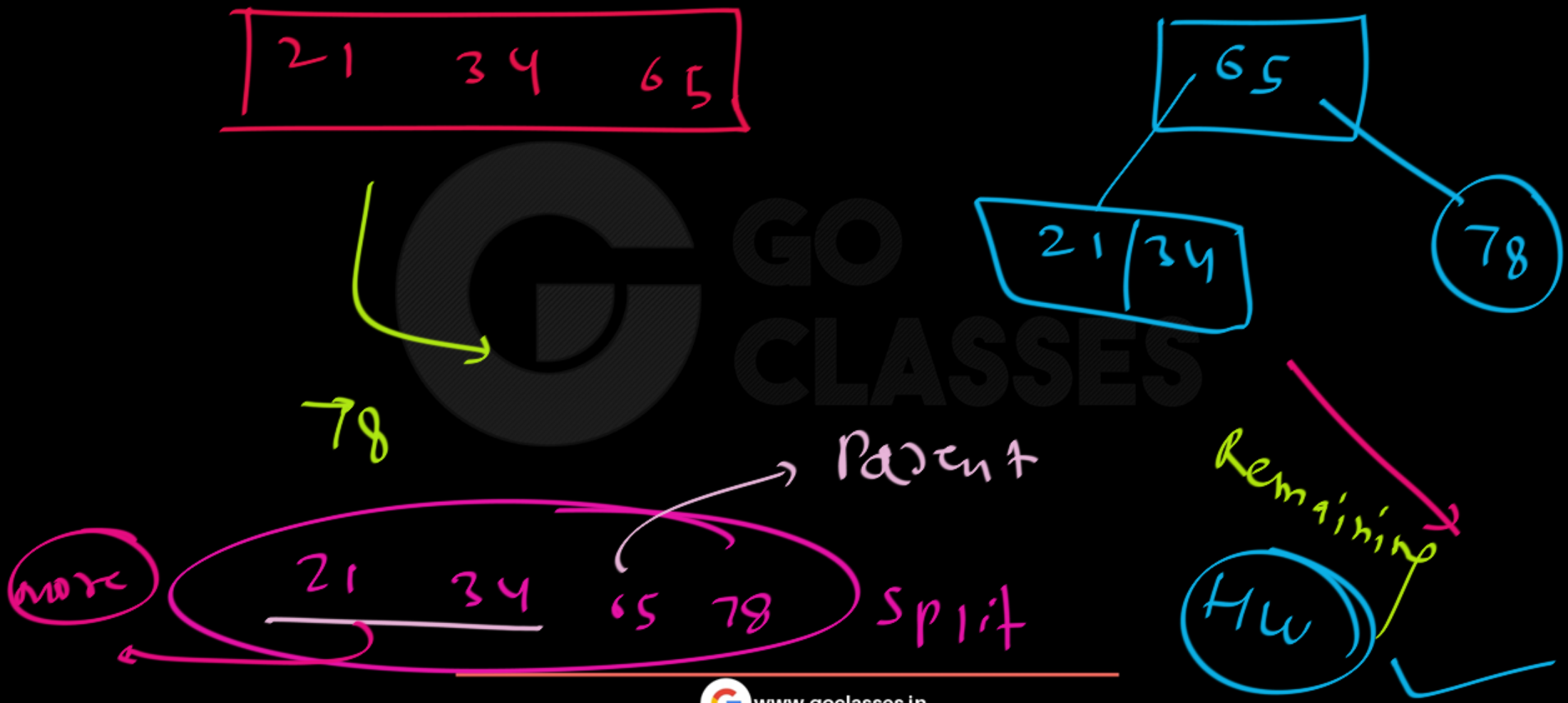
Define B Tree. Create a B Tree of order 4 using the keys:

21, 34, 65, 78, 90, 09, 23, 11, 55, 76, 22, 33

and make the tree a "left-biased".

1st Split

2nd Split


21 34 65
65
21 34
78
78
Parent
more
21 34 65 78
split
Remaining
HW

B - Tree : → GATE

Insertion & frequent

B+ Tree → finding over & are frequent

Homework:

Define B Tree. Create a B Tree of order 4 using the keys:
21, 34, 65, 78, 90, 09, 23, 11, 55, 76, 22, 33
and make the tree a "Right-biased".

Q: Data Structures: A Pseudocode Approach with C, 2nd Ed
Richard F. Gilberg & Behrouz A. Forouzan

Does Order in which keys are inserted in B Tree Matter??

Can we get Different B trees, of Different heights for the same set of keys, But using different order of insertion of keys in B tree?

Q:

Does Order in which keys are inserted in B Tree Matter??

Can we get Different B trees, of Different heights for the same set of keys, But using different order of insertion of keys in B tree?

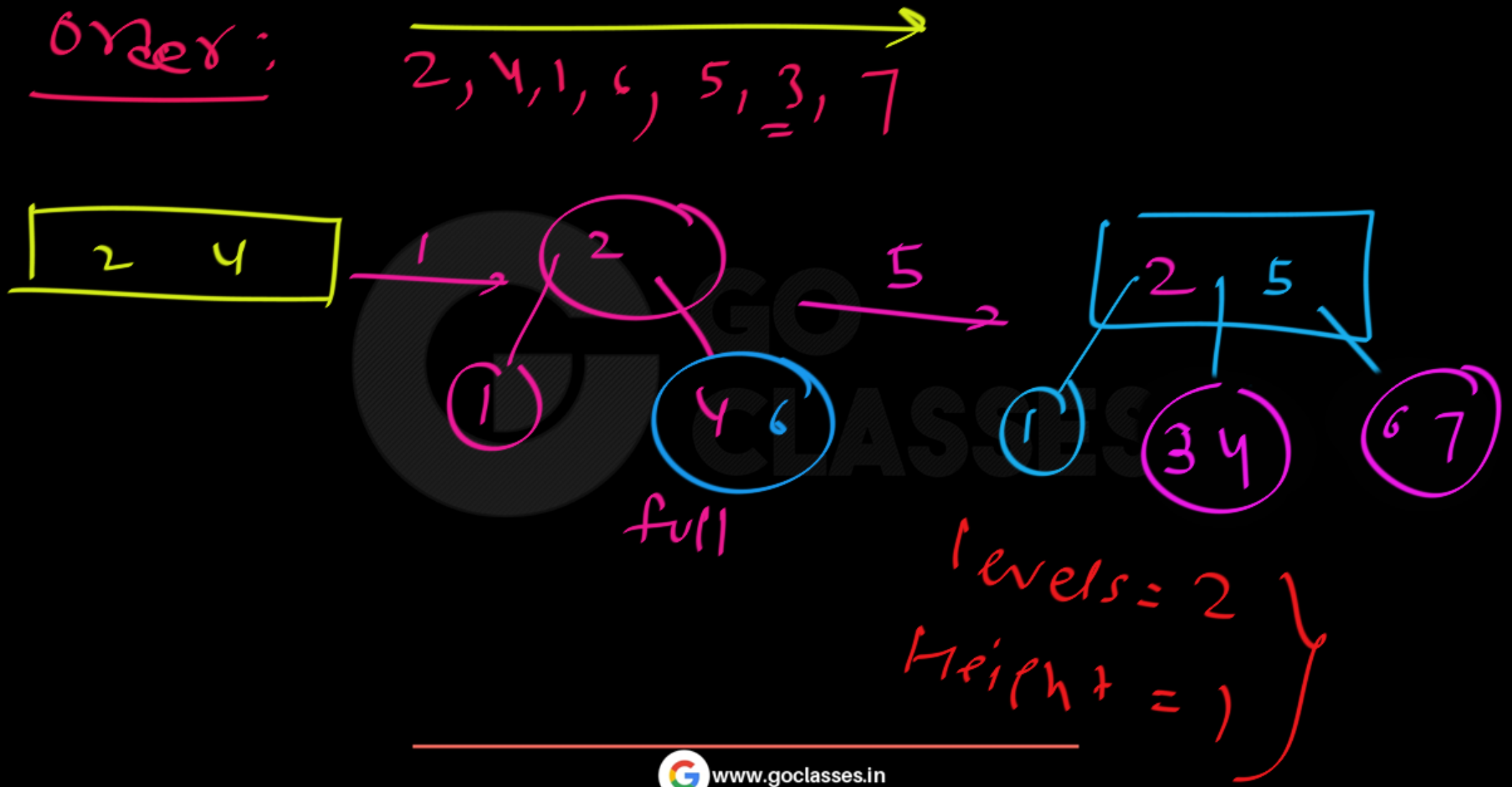
Ans: YES. Consider a B Tree of Order 3, and the following set of keys: {1,2,3,4,5,6,7}

Insertion Order 1: 1,2,3,4,5,6,7

Insertion Order 2: 2,4,1,6,5,3,7

B Tree: order 3 ⟹ # keys = 1 to 2

order: 1, 2, 3, 4, 5, 6, 7 ⟹ levels = 3, Height = 2


Order:
2, 4, 1, 6, 5, 3, 7
2 4
1
2
5
2 5
1
4 6
1
3 4
6 7
full
levels = 2
Height = 1

Q: Data Structures: A Pseudocode Approach with C, 2nd Ed
Richard F. Gilberg & Behrouz A. Forouzan

6. Draw two different B-trees of order 3 that can store seven entries.

Done

Q: Homework

Data Structures: A Pseudocode Approach with C, 2nd Ed Richard F. Gilberg & Behrouz A. Forouzan

5. Draw two B-trees of order 3 created by inserting data arriving in sequence from the two sets shown below. Compare the two B-trees to determine whether the order of data creates different B-trees.

89 78 8 19 20 33 56 44

44 56 33 20 19 8 78 89

Q: Data Structures: A Pseudocode Approach with C, 2nd Ed Richard F. Gilberg & Behrouz A. Forouzan

10. Using the B-tree of order 3 shown in Figure 10-24, add 50, 78, 101, and 232.

#levels = 4

50 → No split

78 — 1 split

101 — No split

232 — 3 splits

#splits = 1 + 3 = 4

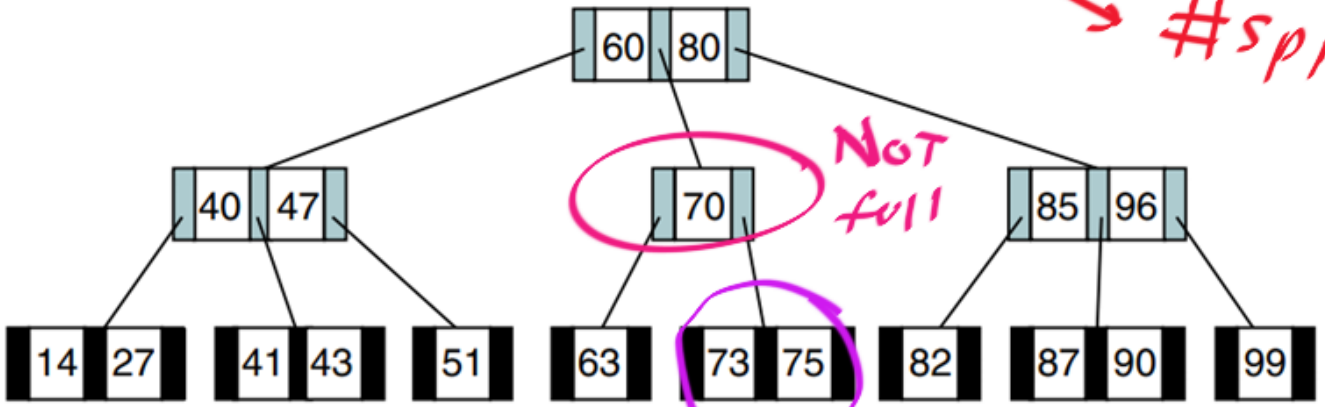


FIGURE 10-24 B-tree for Exercises 10

Q: Data Structures: A Pseudocode Approach with C, 2nd Ed Richard F. Gilberg & Behrouz A. Forouzan

10. Using the B-tree of order 3 shown in Figure 10-24, add [redacted] 101, and 232.

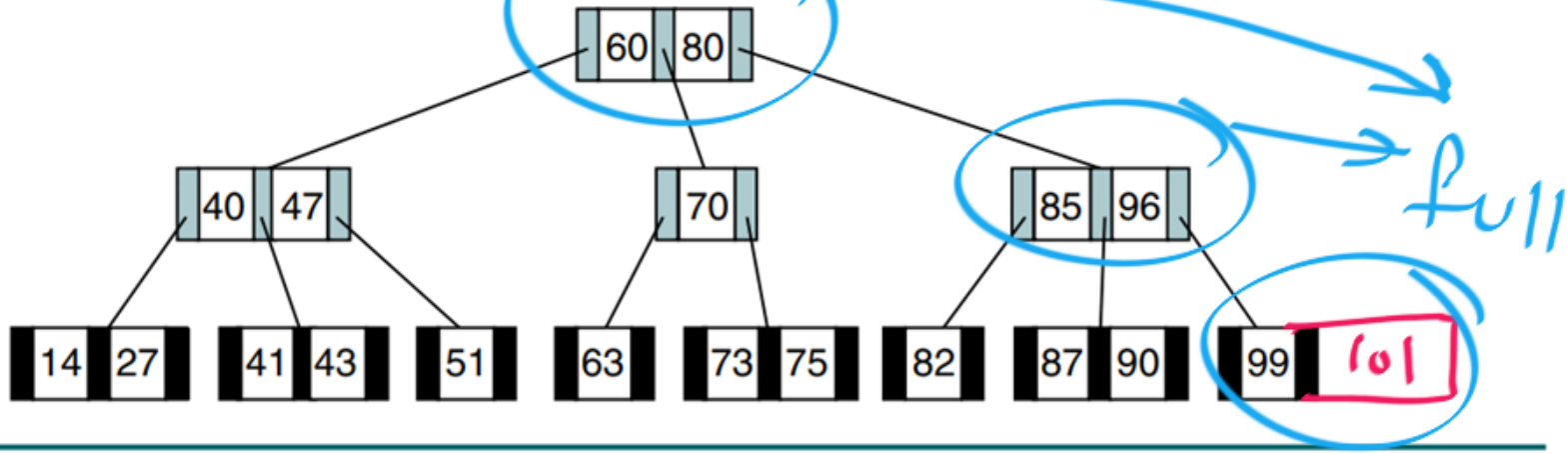


FIGURE 10-24 B-tree for Exercises 10

Q: Data Structures: A Pseudocode Approach with C, 2nd Ed Richard F. Gilberg & Behrouz A. Forouzan

10. Using the B-tree of order 3 shown in Figure 10-24, add 50, 78, 101, and 232.

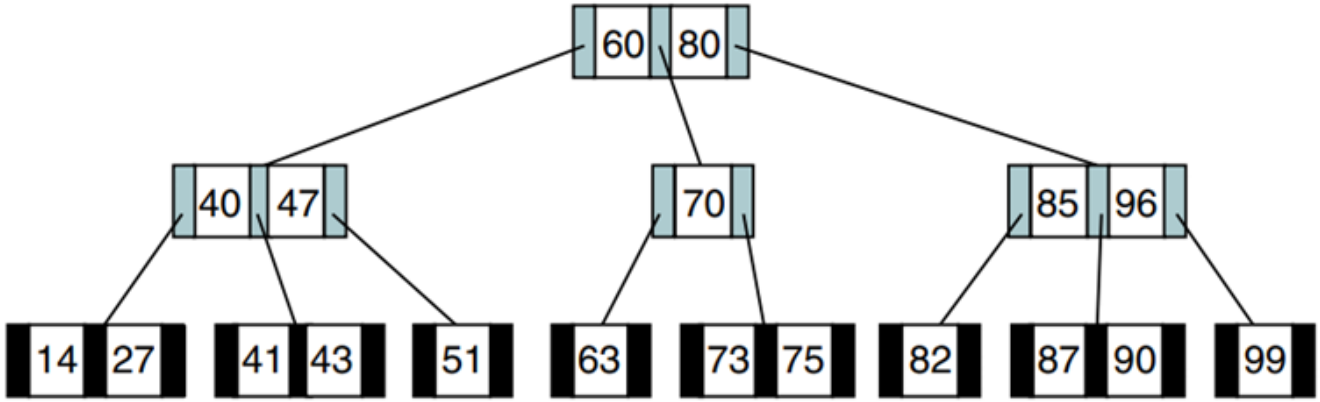


FIGURE 10-24 B-tree for Exercises 10

Insertion in B Tree:

Can Cause Split from Leaf to Root node.

Q: Homework

Data Structures: A Pseudocode Approach with C, 2nd Ed Richard F. Gilberg & Behrouz A. Forouzan

3. Draw the B-tree of order 3 created by inserting the following data arriving in sequence:

```
92 24 6 7 11 8 22 4 5 16 19 20 78
```

4. Draw the B-tree of order 4 created by inserting the following data arriving in sequence:

```
92 24 6 7 11 8 22 4 5 16 19 20 78
```

A B-tree starts with a single root node (which is also a leaf node) at level 0 (zero). Once the root node is full with $p - 1$ search key values and we attempt to insert another entry in the tree, the root node splits into two nodes at level 1. Only the middle value is kept in the root node, and the rest of the values are split evenly between the other two nodes. When a nonroot node is full and a new entry is inserted into it, that node is split into two nodes at the same level, and the middle entry is moved to the parent node along with two pointers to the new split nodes. If the parent node is full, it is also split. Splitting can propagate all the way to the root node, creating a new level if the root is split.

Inserting a New Item

The insertion algorithm proceeds as follows: When inserting an item, first do a search for it in the B-tree. If the item is not already in the B-tree, this unsuccessful search will end at a leaf. If there is room in this leaf, just insert the new item here. Note that this may require that some existing keys be moved one to the right to make room for the new item. If instead this leaf node is full so that there is no room to add the new item, then the node must be "split" with about half of the keys going into a new node to the right of this one. The median (middle) key is moved up into the parent node. (Of course, if that node has no room, then it may have to be split as well.) Note that when adding to an internal node, not only might we have to move some keys one position to the right, but the associated pointers have to be moved right as well. If the root node is ever split, the median key moves up into a new root node, thus causing the tree to increase in height by one.

Next Topic:

B Tree Questions

Q 1. True/False??

Assume a B Tree has Single Node i.e. Only the root node.

Is this Alone root node a Leaf node OR internal node??

Q 1. True/False??

Assume a B Tree has Single Node i.e. Only the root node.

→ All BP are Null.

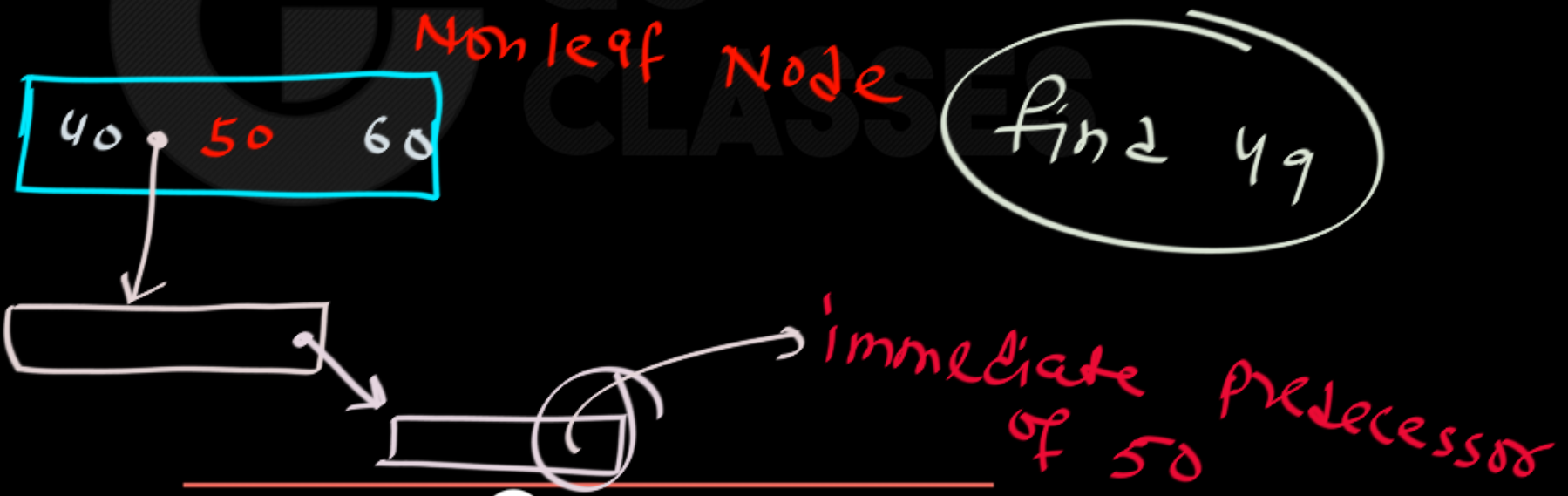
Is this Alone root node a Leaf node OR internal node?? ⇒ Leaf Node

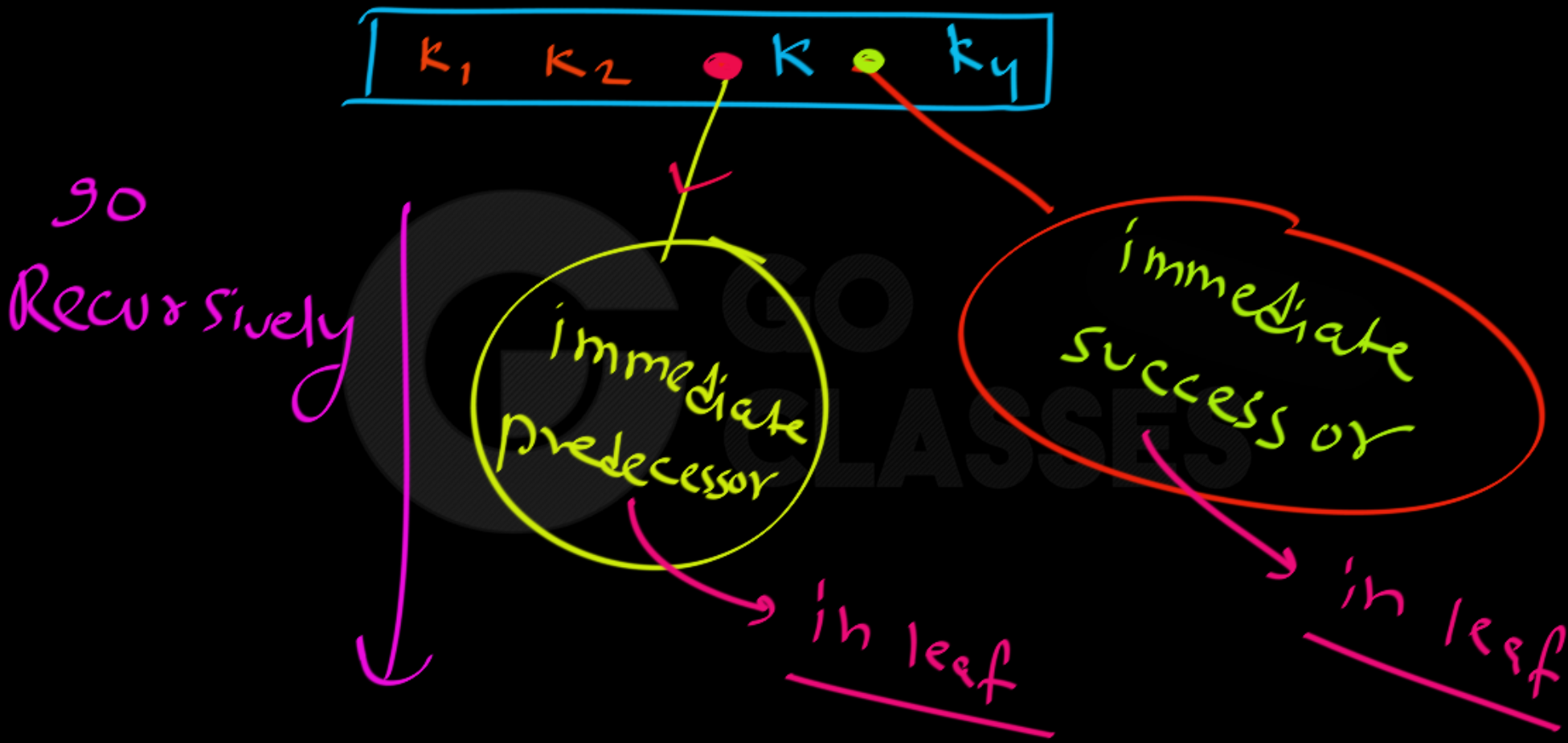
Q 2. True/False??

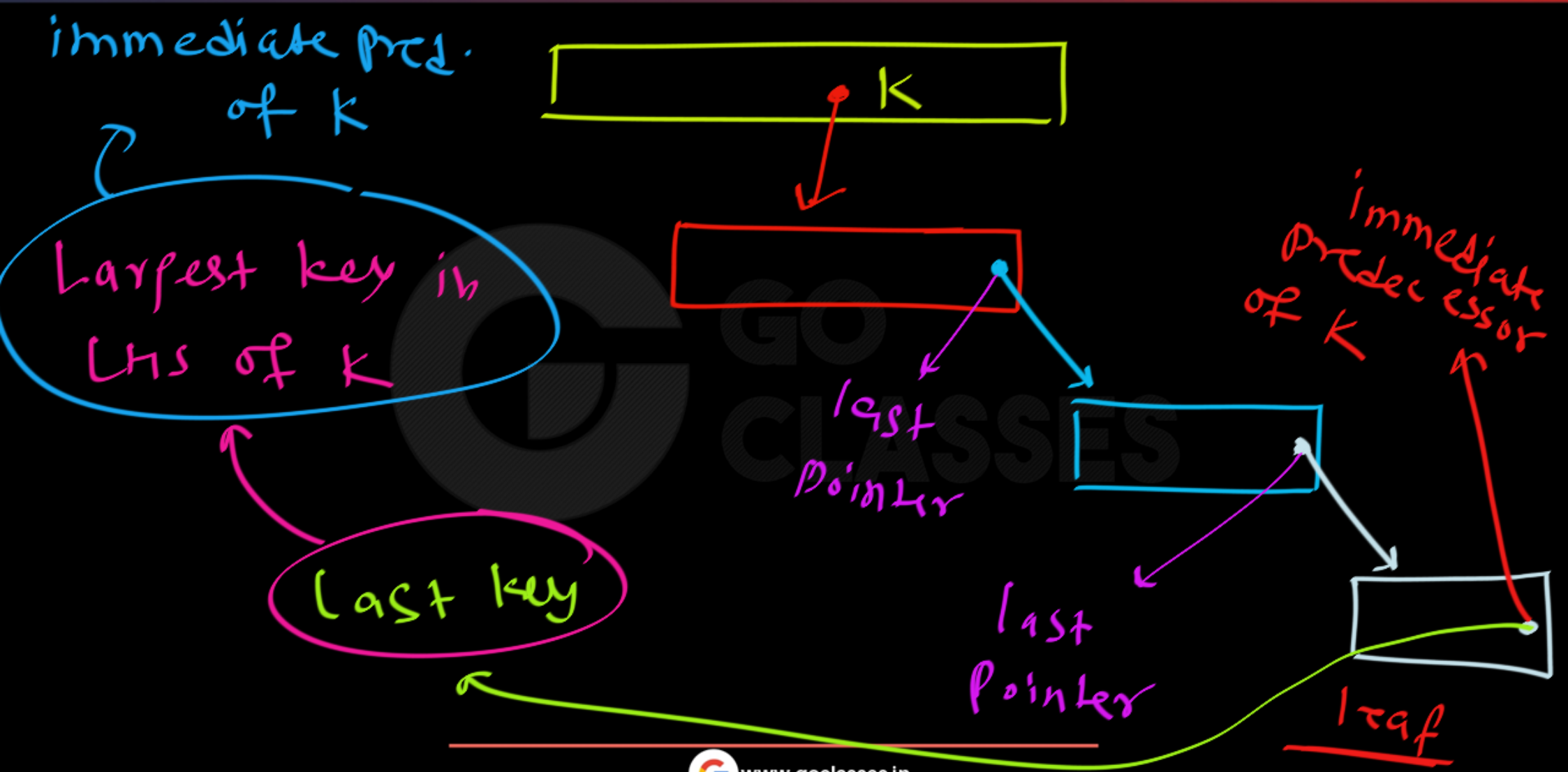
Immediate predecessor of every key of non leaf is always in a leaf.

Q 2. True/False??

Immediate predecessor of every key of non leaf is always in a leaf. ⇒ True

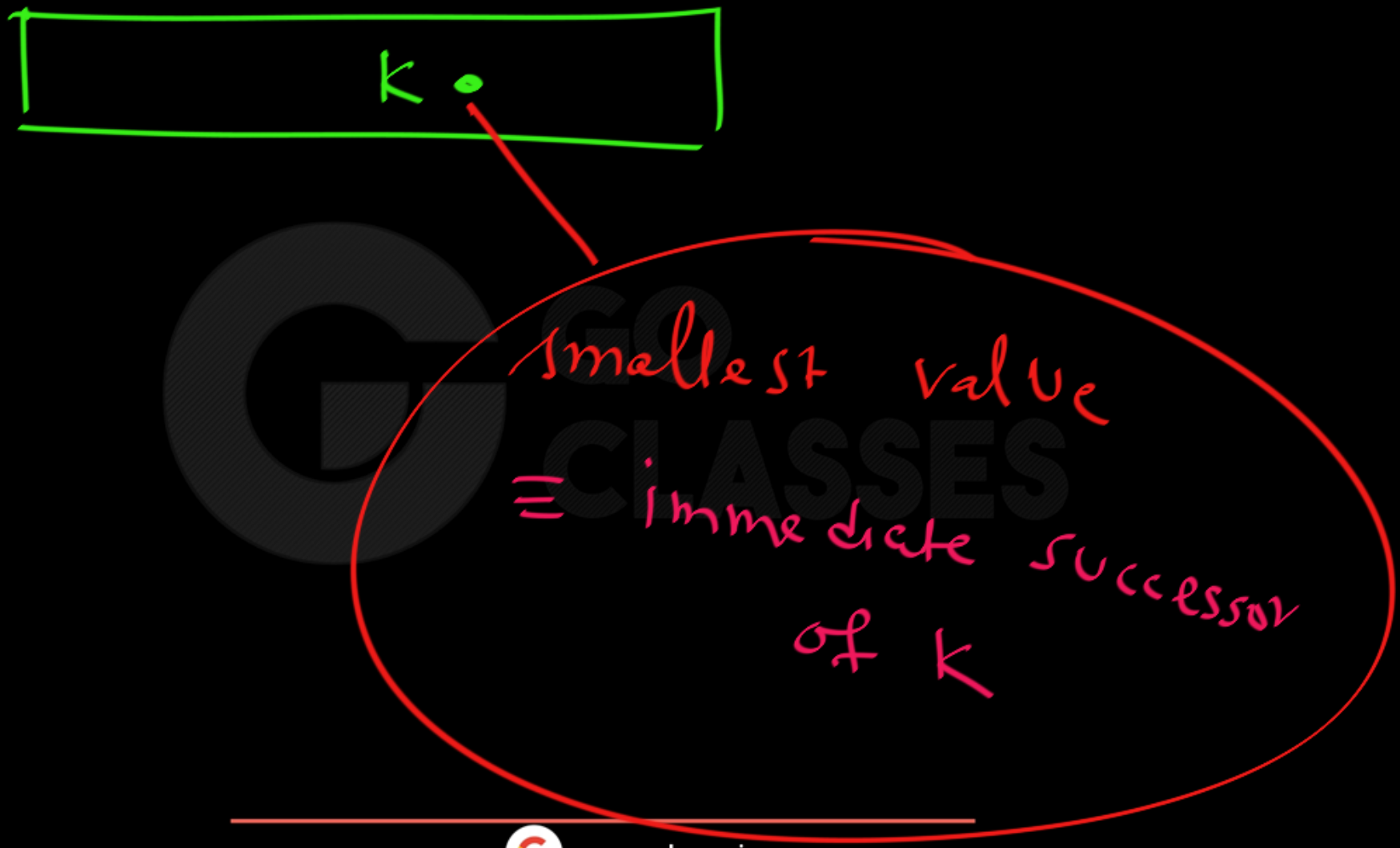


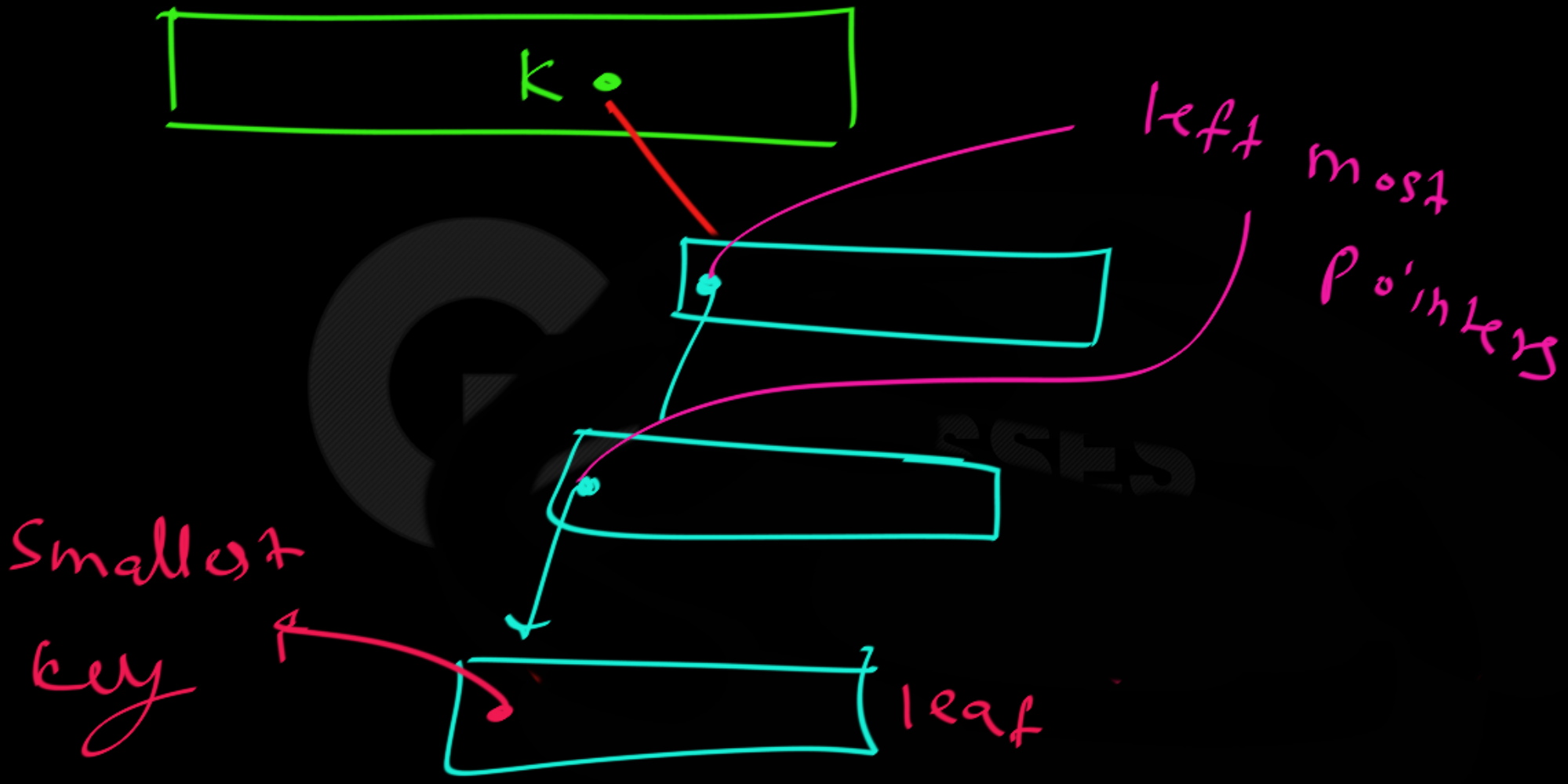

K_1 K_2 K K_y
go Recursively
immediate predecessor
immediate successor
in leaf
in leaf


immediate Pred. of K
K
Largest key in LHS of K
last Pointer
immediate Predecessor of K
last key
last Pointer
leaf

Q 3. True/False??

Immediate successor of every key of non leaf is always in a leaf. True


k
smallest value
≡ immediate successor
of k


K
left most Pointer
Smallest key
leaf

Q 4. True/False??

Immediate successor of every key of leaf is always in a non-leaf.

false

Q 4. True/False??

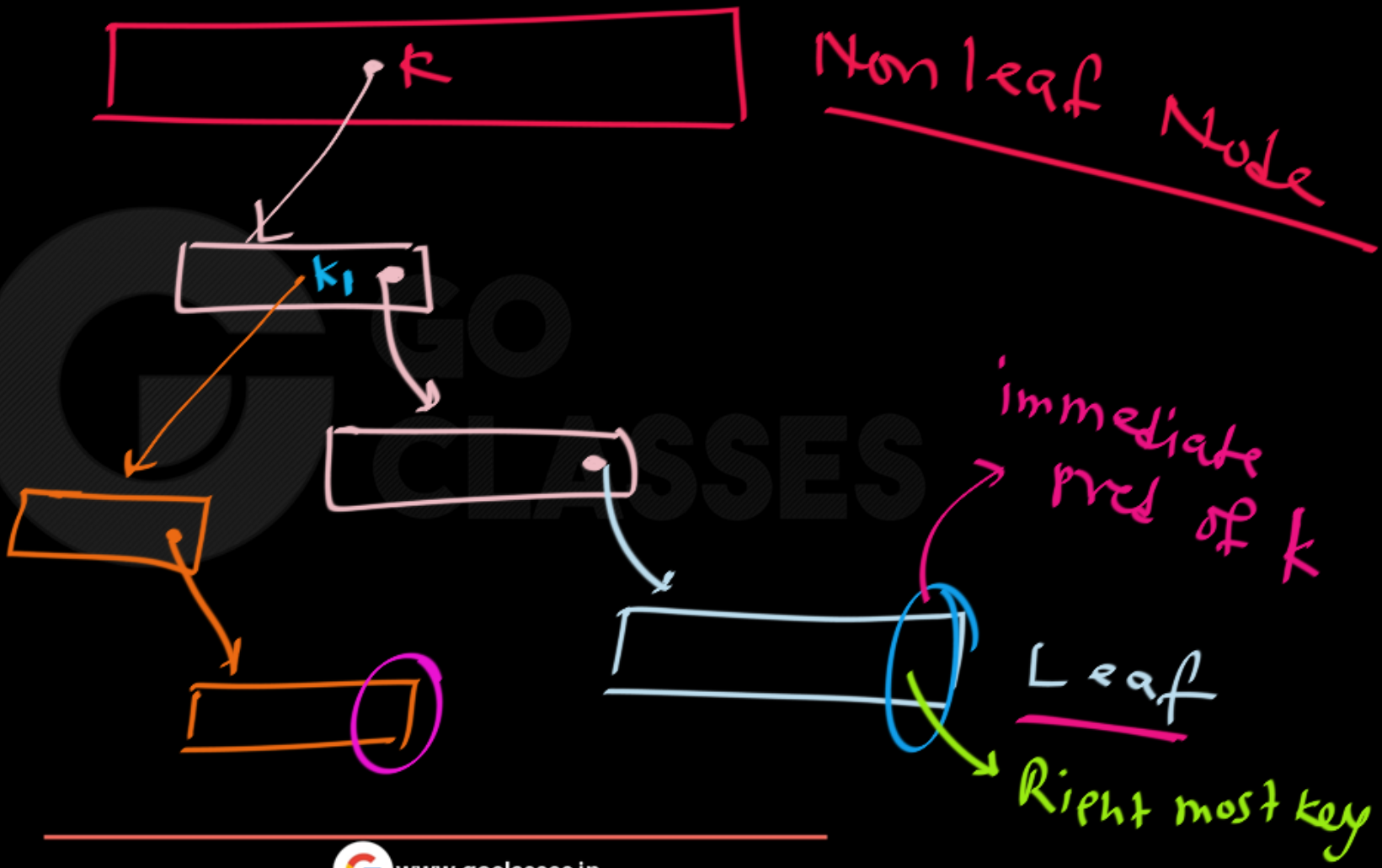
Immediate successor of every key in leaf is always in a non-leaf.

Q 5.

Find the Relationship between Number of Leaf Nodes & Number of Keys in Non-leaf nodes??

Number of leaf Nodes = 1 + Number of keys in Non-leaf Nodes

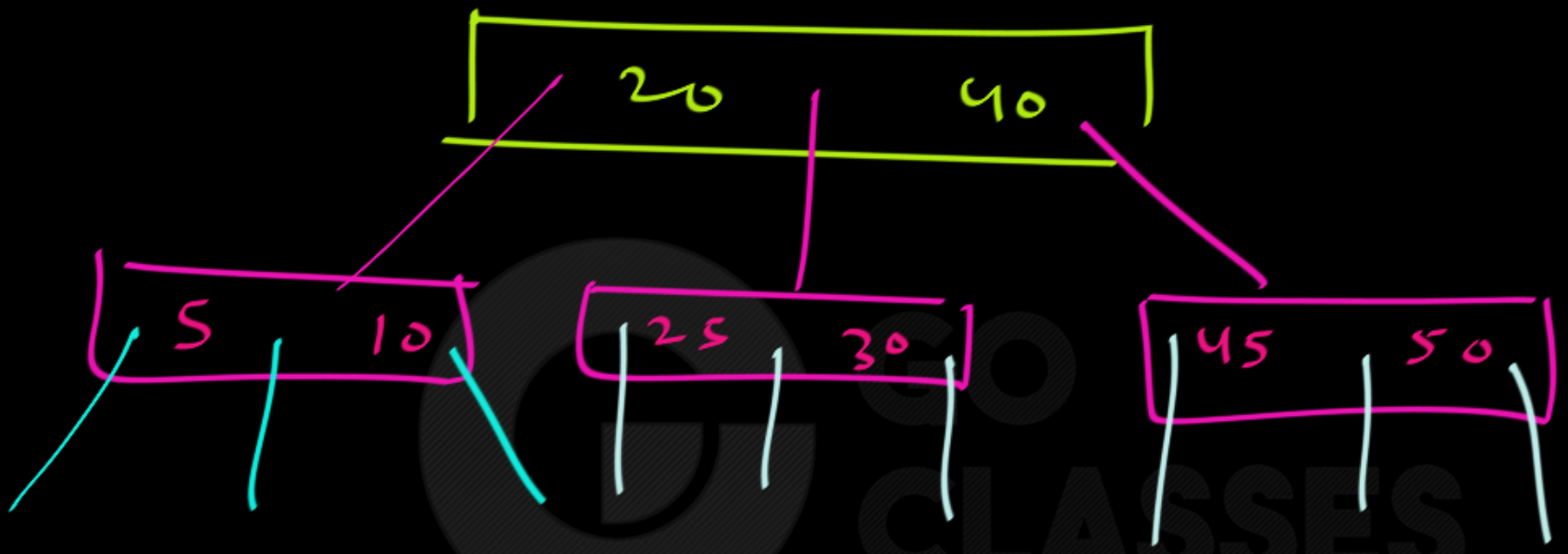
proof by finding Immediate Predessor;



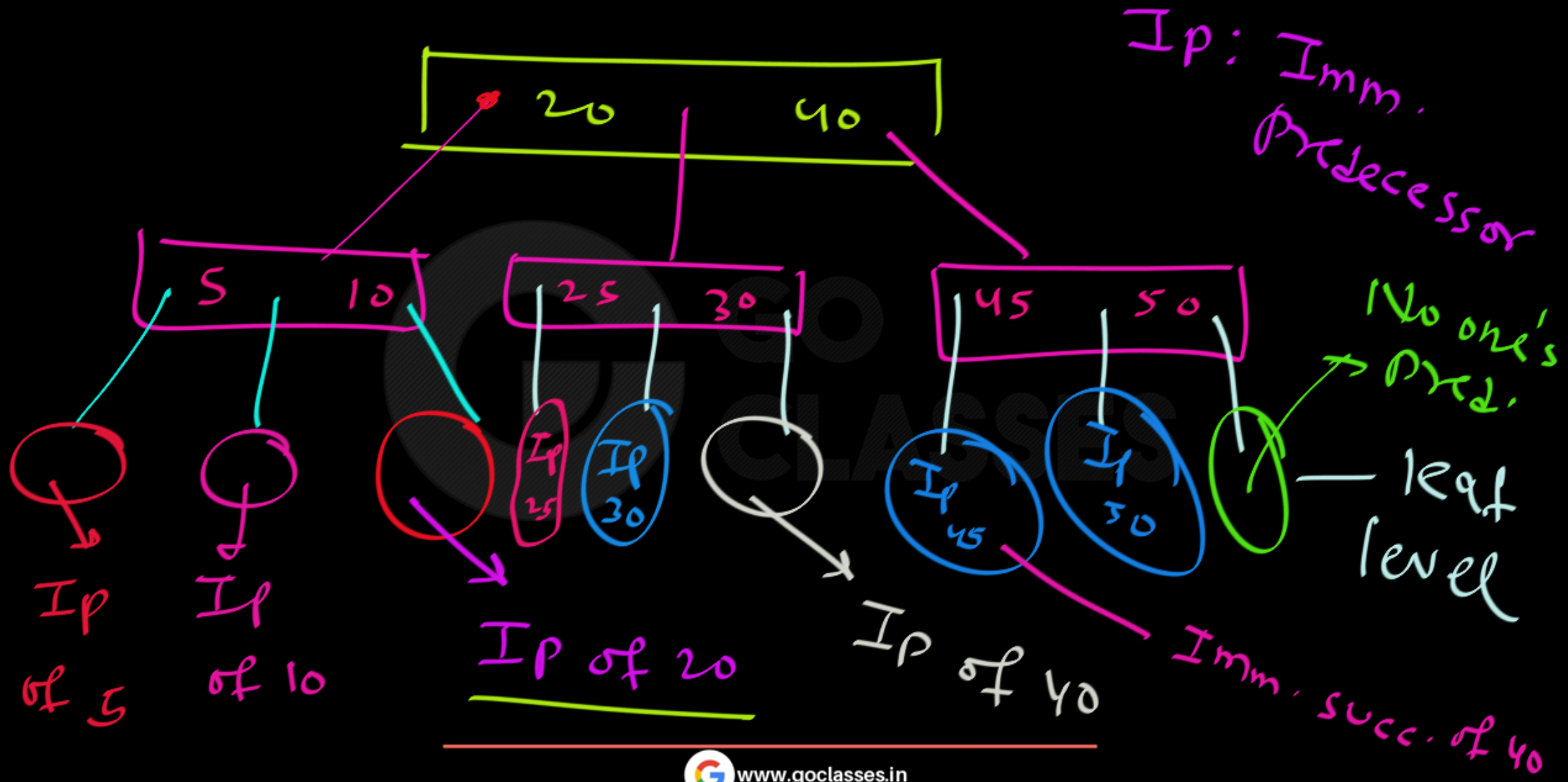
for every Key of Non-Leaf Node

we have Unique Immediate Predecessor Leaf Node.

#Leaf Nodes = #Keys in Non-leaf Nodes + 1 → Right most leaf Node



— leaf level


Ip: Imm. Predecessor
20
40
5
10
25
30
45
50
No one's Pred.
Ip of 5
Ip of 10
Ip of 20
Ip of 40
Imm. succ. of 40
leaf level

Leaf Nodes = 1 + # Keys in Non-leaf Nodes

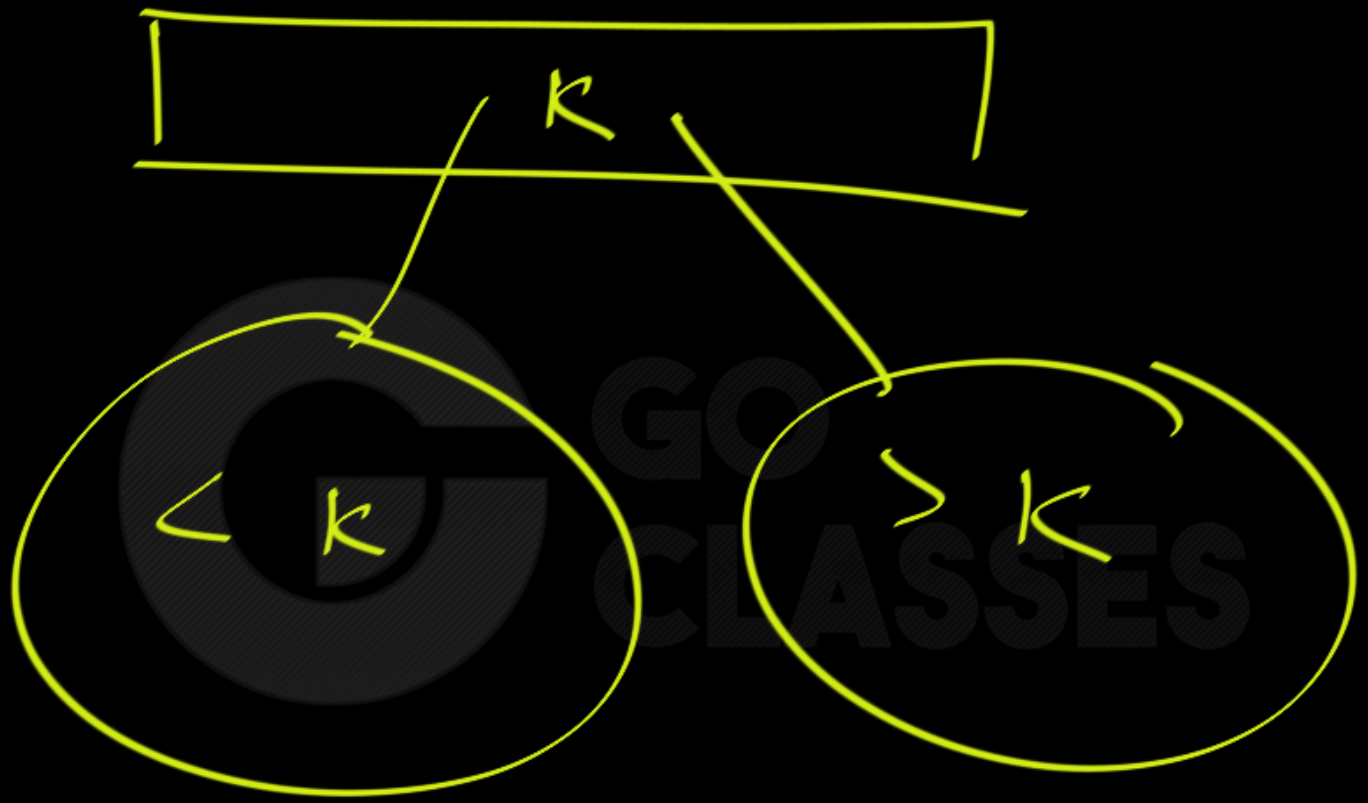
Leaf Nodes = 1 + # Non-leaf Nodes

False

Q 6. True/False?

If B tree is created for set of distinct keys,
Then NO key repeats in two nodes of B tree.

True


K
< K
> K

Conclusion/Results:

1. Immediate successor of every key in non leaf is always in a leaf.
2. Immediate predecessor of every key in non leaf is always in a leaf.

3. For a B Tree of at least two levels:

Number of Leaf Nodes = 1 + Number of Keys in Non-leaf nodes

Next Topic:

B Tree

Find the Order Questions

Default Definition of ORDER of a B Tree Node:

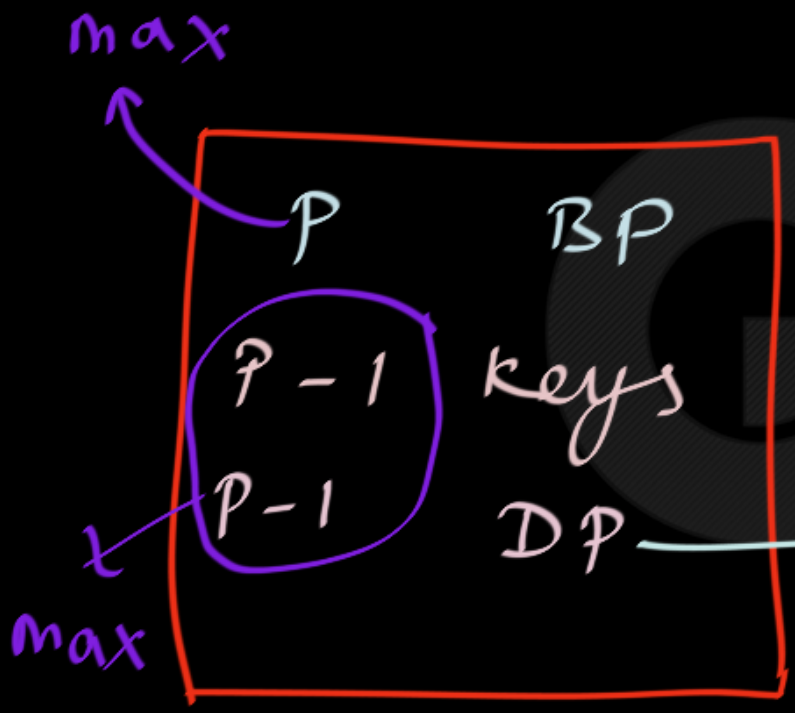
(Definition by Donald Knuth, Used in Navathe, Korth, Ullman, NPTEL etc Every DBMS Book)

Order of B Tree is p:

Each B-tree node can have at most p tree/Block/Node pointers, p − 1 data(Record or Block) pointers, and p − 1 search key field values.

Find the order;

P: max no. of BP in a Node (Block)



$P(BP) + (P-1)(key + DP) \leq$ Block Size

RP → by default
OR
BP

Disk Block (BTree Node)

EXAMPLE 4: Suppose the search field is V = 9 bytes long, the disk block size is B = 512 bytes, a record (data) pointer is P r = 7 bytes, and a block pointer is P = 6 bytes. Each B-tree node can have at most p tree pointers, p - 1 data pointers, and p - 1 search key field values. These must fit into a single disk block if each B-tree node is to correspond to a disk block.

find p ?

EXAMPLE 4: Suppose the search field is V = 9 bytes long, the disk block size is B = 512 bytes, a record (data) pointer is P r = 7 bytes, and a block pointer is P = 6 bytes. Each B-tree node can have at most p tree pointers, p - 1 data pointers, and p - 1 search key field values. These must fit into a single disk block if each B-tree node is to correspond to a disk block.

find p ? ↳ order of B Tree

$P(6) + (P-1)(9+7) \leq 512\,B$

$22P - 16 \leq 512 \Rightarrow P \leq \frac{528}{22}$

$P = \left\lfloor \frac{528}{22} \right\rfloor = 24$

Tree pointer ≡ Node " ≡ child " ≡ Block "

EXAMPLE 4: Suppose the search field is V = 9 bytes long, the disk block size is B = 512 bytes, a record (data) pointer is P r = 7 bytes, and a block pointer is P = 6 bytes. Each B-tree node can have at most p tree pointers, p - 1 data pointers, and p - 1 search key field values. These must fit into a single disk block if each B-tree node is to

correspond to a disk block. Hence, we must have:

$(p * P) + ((p - 1) * (P r + V)) \leqq B$

$(p * 6) + ((p - 1) * (7 + 9)) \leqq 512$

$(22 * p) \leqq 528$

[redacted]

When Life Gives You Lemons....

Make Lemonade

If Some other Definition of ORDER of a B Tree Node is given then somehow get back to OUR default definition of Order.

Stick to your powers...

Convert the situation to your advantage...

If Some other Definition of ORDER of a B Tree Node is given then somehow get back to OUR default definition of Order:

Eg: Order of B Tree is p:

Each B-tree node can have at most p keys.

find p?

$$P(K+DP)+(P+1)BP \leq \text{Block size}$$

If Some other Definition of ORDER of a B Tree Node is given then somehow get back to OUR default definition of Order:

Eg: Order of B Tree is p:

Each B-tree node, except root can have at least p, and at most 2p tree pointers. If tree is non-empty, root can have minimum 2 pointers.

find p ?

At most 2p Tree pointers per Node:

$$2p(BP) + (2p-1)(K+DP) \leq \text{Block Size}$$

IDEA:

$$(\max \#Bp)(BP) + (t-1)(K+DP) \leq \text{Block Size}$$

t

Q:

P: order

The order p of B tree node is defined as:

Each B-tree node can have at most p keys.

Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the B tree node?

find p !

B Tree: Order P : max no. of keys per Node

$$P(K + DP) + (P+1)(BP) \leq \text{Block Size}$$

$$P(9+7) + (P+1)(6) \leq 1024$$

$$22P + 6 \leq 1024$$

$$P \leq \frac{1018}{22} = 46.\ldots$$

$P = 46$

Note:

Definition of Order of B Tree can be given anything.

If nothing is Given then our Definition of Order is by Default. → max # Bp Per Node

Whatever the Definition of Order p :

IDEA ;

$$(\max \#Bp)(Bp) + (y-1)(k+Dp) \leq \text{Block Size}$$

y

In any B Tree Node :

- q keys
- q DP
- q+1 BP

In any B Tree Node :

- q BP
- q-1 keys
- q-1 Dp

Q:

The order of a leaf node in a B tree is the maximum number of (value, data record pointer) pairs it can hold. Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node?

p

find p ?

for leaf Node;

$P(9+7) \leq 1024$ ✗

P : max no. of (keys, Rp) pairs in leaf node

for leaf Node;

Leaf Node of B Tree
↳ Space of BP is Still there; All BP are NULL

$P(9+7)+(P+1)(6) \leq 1024$

$P = 46$

In B Tree, ALL Nodes have Same Structure.

Q:

The order p of B tree node is defined as:

Each B-tree node, except root can have at least p and at most 2p keys. Root can have at least 1, at most 2p keys.

Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the B tree node?

find p?

$$2p(9+7) + (2p+1)(6) \leq 1024$$

$$32p + 12p + 6 \leq 1024$$

$$44p + 6 \leq 1024$$

$$p \leq \frac{1018}{44} = 23....$$

$$p = 23$$

Q: The order p of B tree node is defined as:
Each B-tree node, except root can have at least p and at most 2p block pointers. Root can have at least 2, at most 2p tree pointers. (Definition by Cormen)

Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the B tree node?

$$2p(6) + (2p-1)(9+7) \leq 1024$$

$$12p + 32p - 16 \leq 1024$$

$$p \leq \frac{1040}{44} = 23....$$

$$p = 23$$

max # keys Per Node = $(2p-1) = 45$

Default Definition of ORDER of a B Tree Node:

(Definition by Donald Knuth, Used in Navathe, Korth, Ullman, NPTEL etc Every DBMS Book)

Order of B Tree is p:

Each B-tree node can have at most p tree/Block/Node pointers, p − 1 data(Record or Block) pointers, and p − 1 search key field values.

https://gateoverflow.in/3723/gate-it-2004-question-79

3.1.24 B Tree: GATE IT 2004 | Question: 79 top https://gateoverflow.in/3723

Consider a table T in a relational database with a key field K. A B-tree of order p is used as an access structure on K, where p denotes the maximum number of tree pointers in a B-tree index node. Assume that K is 10 $bytes$ long; disk block size is 512 $bytes$; each data pointer P_D is 8 bytes long and each block pointer P_B is 5 bytes long. In order for each B-tree node to fit in a single disk block, the maximum value of p is

A. 20
B. 22
C. 23
D. 32

https://gateoverflow.in/3723/gate-it-2004-question-79

Index

3.1.24 B Tree: GATE IT 2004 | Question: 79 top https://gateoverflow.in/3723

Consider a table T in a relational database with a key field K. A B-tree of order p is used as an access structure on K, where p denotes the maximum number of tree pointers in a B-tree index node. Assume that K is 10 *bytes* long; disk block size is 512 *bytes*; each data pointer P_D is 8 bytes long and each block pointer P_B is 5 bytes long. In order for each B-tree node to fit in a single disk block, the maximum value of p is

A. 20
B. 22
C. 23
D. 32

$$p(5) + (p-1)(10+8) \leq 512$$

$$p \leq \frac{530}{23} = 23, \ldots \Rightarrow p = 23$$

A Beautiful Variation Question on B Tree

Before the variation, Solve this:

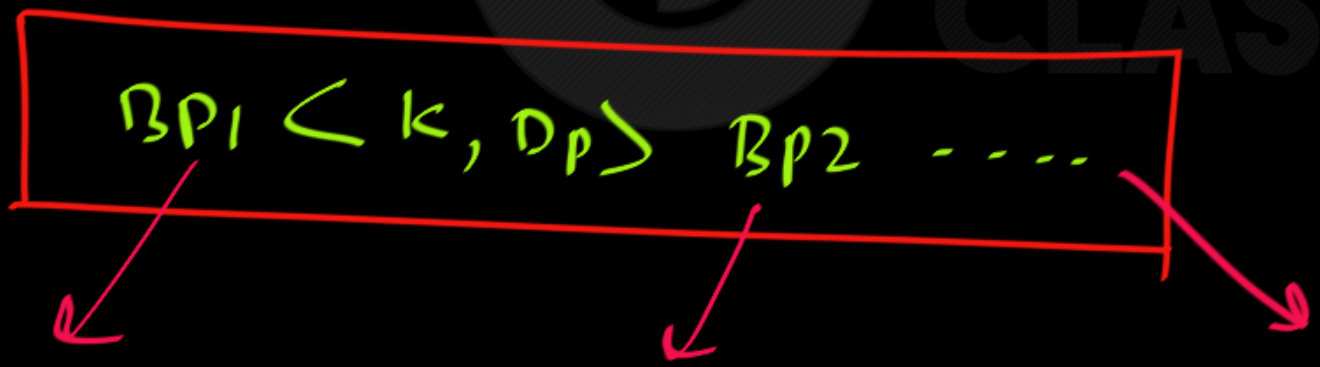
Q: Assume order of a leaf node of B tree is the maximum number of keys that can be sored in the leaf node.

Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node of B tree?

Already Done

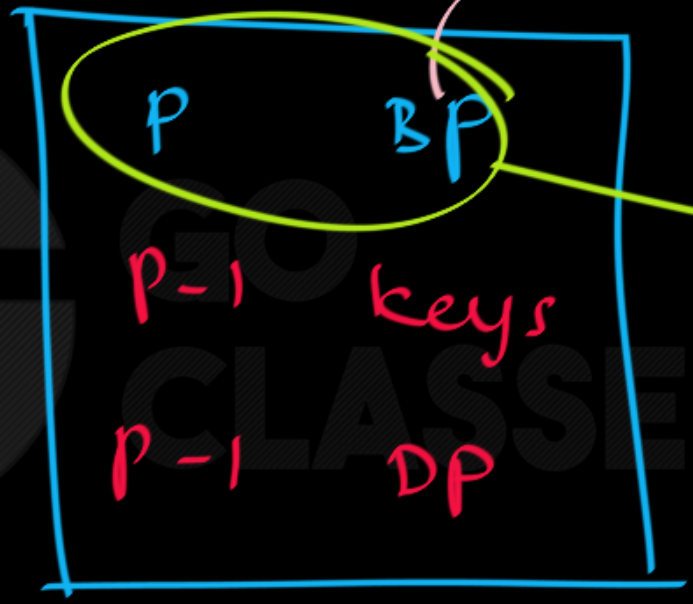
B Tree :

Leaf Node Structure ≡ Internal Node Structure ≡ Root Node Structure



B Tree :

Leaf Node :



Null pointers

Wasted Space in Leaf Node of B Tree

NOTE that in B tree, ALL the nodes have the SAME structure...

In the leaf nodes, ALL tree pointers are NULL
(But tree pointers are present in the leaf node, even though they all are NULL).

NOTE that in B tree, ALL the nodes, except root node, have the SAME structure...

In the leaf nodes, ALL tree pointers are NULL
(But tree pointers are present in the leaf node, even though they all are NULL).

Seems like a WASTE of SPACE in the Lead Node...Right??

Q: Cormen Algorithms Book (CLRS Book)

18.2-5

Since leaf nodes require no pointers to children, they could conceivably use a different (larger) t value than internal nodes for the same disk page size.

In this VARIANT of B tree, How to find the Order of Leaf Node...??

Q: Cormen Algorithms Book (CLRS Book)

18.2-5

Since leaf nodes require no pointers to children, they could conceivably use a different (larger) t value than internal nodes for the same disk page size.

[redacted]

In this VARIANT of B tree, How to find the Order of Leaf Node...??

max # keys per leaf Node $\geq$ max # keys in Internal nodes

The Variation:

Q: Since leaf nodes require no pointers to children, they could conceivably store a different (larger) number of keys than internal nodes for the same disk page size.

Considering this Variation of B tree, Assume order of a leaf node of B tree is the maximum number of keys that can be sored in the leaf node.

Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node of B tree?

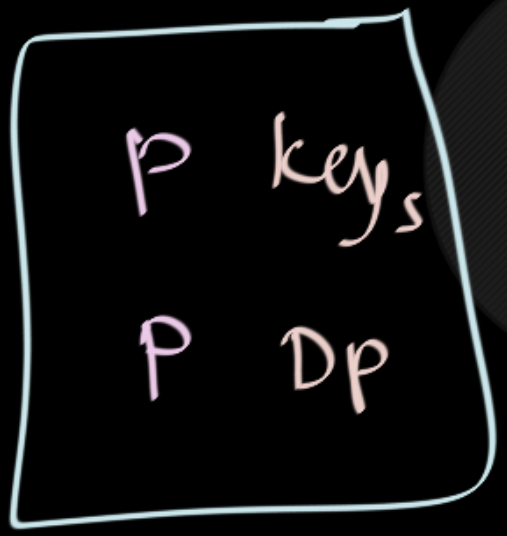
The Variation:

Q: Since leaf nodes require no pointers to children, they could conceivably store a different (larger) number of keys than internal nodes for the same disk page size.

Considering this Variation of B tree, Assume order of a node of B tree is the maximum number of keys that can be sored in the node.

Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node of B tree?

Leaf Node Structure in prev. Variant of B Tree :



Leaf Node

$p(9+7) \leq 1024$

$p \leq \frac{1024}{16} = 64$

order of leaf Node

Internal Node Structure in prev. Variant of B Tree :

order of 'internal node' = y

$$y(9+7)+(y+1)(6) \leq 1024$$

$$y \leq \frac{1018}{22} = 46\frac{}{}$$

y

Internal Node

order of 'internal node'

The Variation:

Q: Since leaf nodes require no pointers to children, they could conceivably store a different (larger) number of keys than internal nodes for the same disk page size.

Considering this Variation of B tree, Assume order of a node of B tree is the maximum number of keys that can be sored in the node.

The block size is 2K bytes, pointer is 12 bytes long, 56 Bytes Block header, the value field is 8 bytes long, what is the order of the leaf node, internal node of B tree?

Order of a leaf Node : P

$$P(8 + 12) \leq (2048 - 56)$$

$P = 99$ → max. no. of keys in leaf Node

Order of a internal Node : y

$$y(8+12)+(y+1)(12) \leq (2048-56)$$

$$y = 61$$

→ max no. of keys in Non-leaf Node

NOTE:

The previous Variant of B tree was a Variation...

In B Tree, ALL NODES Have SAME Structure...

So, in leaf node, we Consider Tree pointers for B tree (though all of them are NULL in leaf node)

Next Topic:

B Tree

Questions about B Tree Structure

Order of B Tree is p:

Each B-tree node can have at most p tree(Block/Node) pointers, p − 1 data(Record or Block) pointers, and p − 1 search key field values.

Problem 2: *B-tree* (20 points)

Consider a DBMS that has the following characteristics:

- 2KB fixed-size blocks
- 12-byte pointers
- 56-byte block headers

We want to build an index on a search key that is 8 bytes long. Calculate the maximum number of records we can index with

a 3-level B tree

Not secure | infolab.stanford.edu/~nsample/cs245/handouts/hw2sol/sol2.html

Problem 2: ***B-tree*** **(20 points)**

max #keys per Block : y

Consider a DBMS that has the following characteristics:

$y(8+12)+(y+1)(12) \leq 2048-56$

- 2KB fixed-size blocks
- 12-byte pointers
- 56-byte block headers ✓

min #keys $= \lceil \frac{p}{2} \rceil - 1$

$= 30$ ✓

$y = 61$; order $= 62 = p$

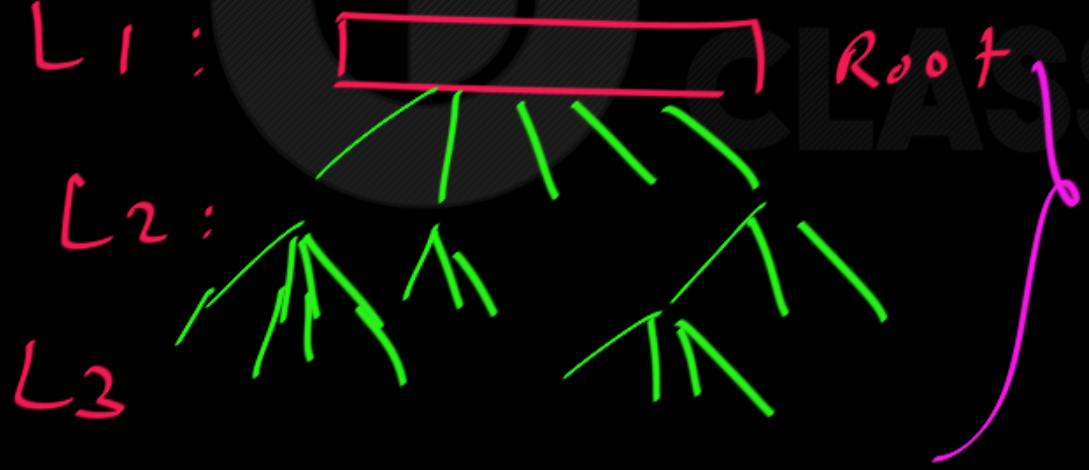
We want to build an index on a search key that is 8 bytes long. Calculate the maximum number of records we can index with

a 3-level B tree

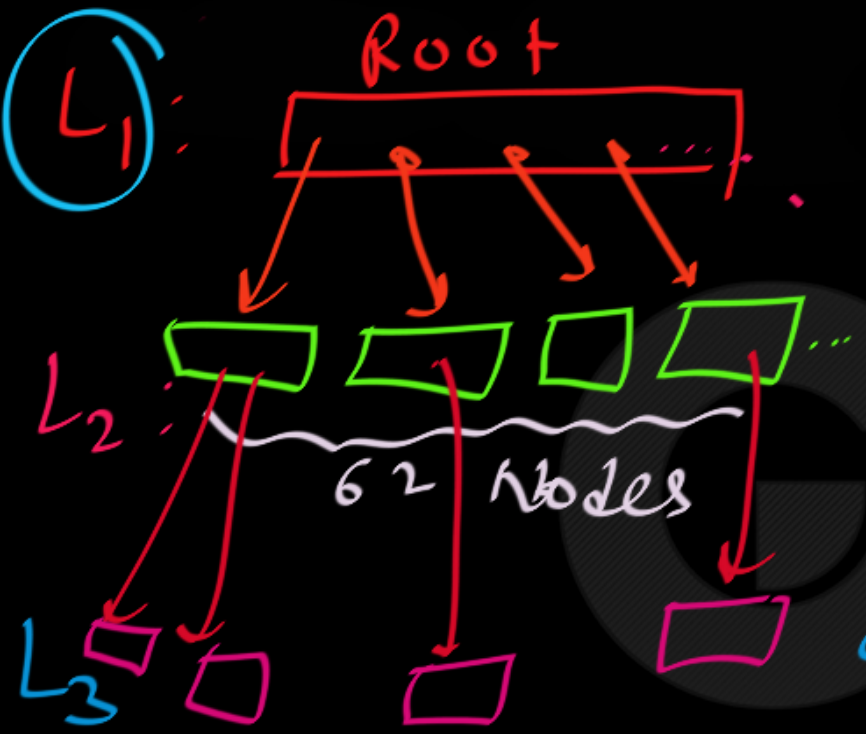
max. number of keys we can store in B-Tree of 3-levels.

Root : 1 key to 61 keys

Non Root Nodes : 30 keys to 61 keys



fill Every Node as much as Possible



	#Nodes	#keys	#BP
L_1	1	61	62
L_2	62	62×61	62×62
L_3	62×62	$62 \times 62 \times 61$	$62 \times 62 \times 62$

All Null Pointers

max no. of keys in 3-levels:

$$61 + (62 \times 61) + (62 \times 62 \times 61)$$

$$= 238327 \text{ keys}$$

Q: Order of B Tree : 62

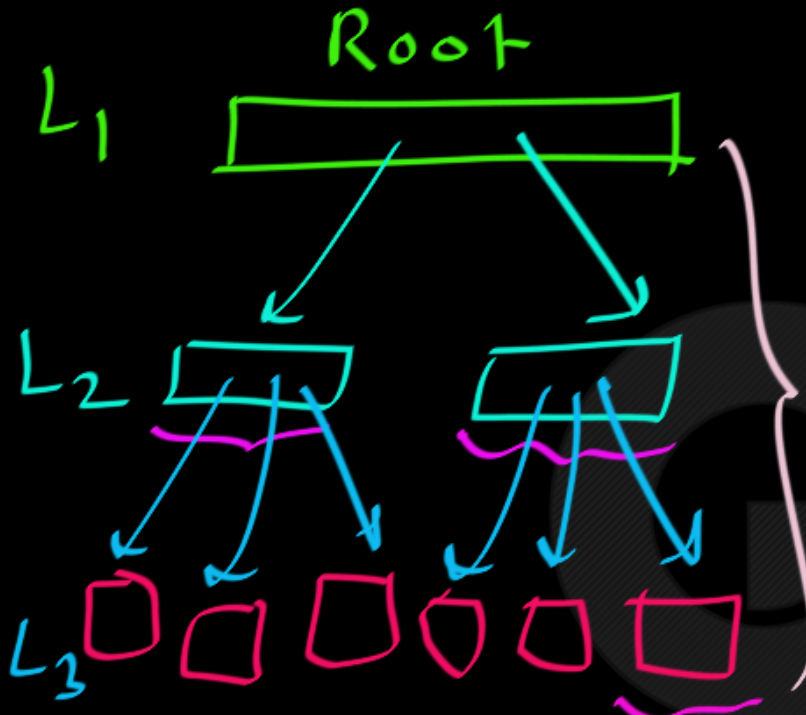
3-levels; minimum # keys we can store?

Q: Order of B Tree : 62

3-levels; minimum # keys we can store?

IDEA: Fill Every Node as minimum as possible.

1 for Root
30 for Non Root



	#Nodes	#keys	#BP
L_1	1	1	2
L_2	2	2×30	2×31
L_3	2×31	$2 \times 31 \times 30$	$2 \times 31 \times 31$

All Null

Min. #keys in 3 levels =

$1 + 60 + 60 \times 31 = 1921$

The Variation:

Q: Since leaf nodes require no pointers to children, they could conceivably store a different (larger) number of keys than internal nodes for the same disk page size.

Considering this Variation of B tree, What will be the answer??

Consider a DBMS that has the following characteristics:

- 2KB fixed-size blocks
- 12-byte pointers
- 56-byte block headers

max no. of BP per internal node = 62

" " of keys per leaf Node = 99

We want to build an index on a search key that is 8 bytes long. Calculate the maximum number of records we can index with

max " " key per internal node = 61

b) a 3-level B tree

Given: min # keys per leaf Node = 50

max #keys in Leaf Node : y

$$y(8+12) \leq 2048 - 56$$

$$y = 99$$

Variant:

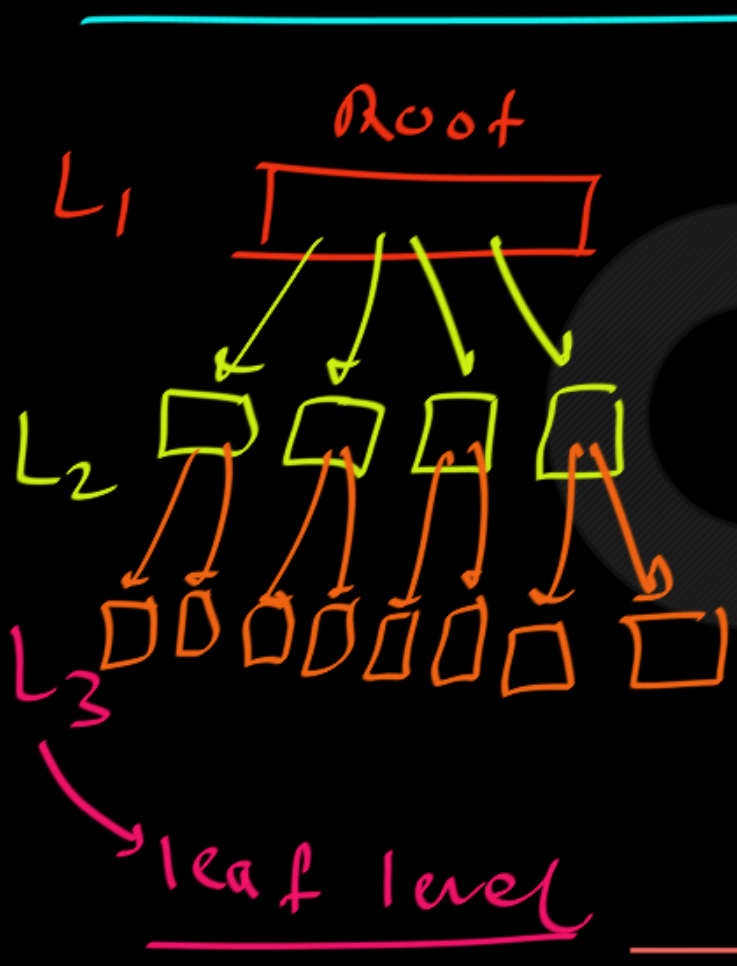
Root : 1 key to 61 keys

Non-Root, Non-leaf Node : 30 keys to 61 keys

Leaf Node : $\lceil \frac{99}{2} \rceil = 50$ to 99 keys

min 50% Space Utilization of 99 keys

B Tree Variant :



	max #Nodes	max #keys	max #BP
L_1	1	61	62
L_2	62	$+ 62 \times 61$	62×62
L_3	62×62	$+ 62 \times 62 \times 99$	—

Not secure | infolab.stanford.edu/~nsample/cs245/handouts/hw2sol/sol2.html

b)

Let each inner node of a B-tree contain at most n index pointers, n-1 keys, and n-1 record pointers. 8 * (n-1) + 12 * (2n-1) + 56 <= 2048. Therefore, n <= 62. The first level of a B-tree can hold at most 61 record pointers. The second level can hold at most 62 * 61 record pointers. The leaf level can hold at most 62 * 62 * 61 record pointers. Therefore, the maximum number of records that can be indexed is 61 + 62 * 61 + 62 * 62 * 61 = 238327. — for B Tree

NOTE: You can also assume that each leaf node of a B-tree can hold at most 99 record pointers. Then, the answer is 384399.

for variation of B Tree

Height of B Tree:

#levels − 1

#Edges from Root to any leaf.

Order of B Tree is p:

Each B-tree node can have at most p tree(Block/Node) pointers, p − 1 data(Record or Block) pointers, and p − 1 search key field values.

Question Type:

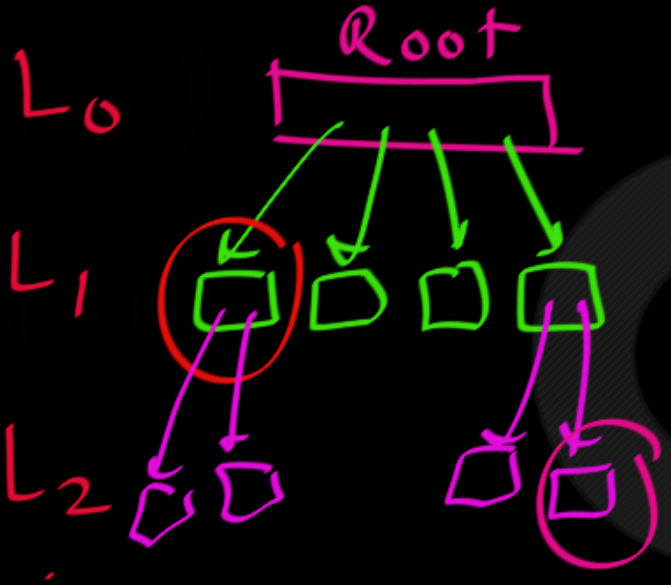
For Given height, maximum number of keys, nodes etc that can be stored.

B Tree: order p;

Root:	2 BP to P BP	1 keys to P-1 keys
Non Root:	$\lceil \frac{P}{2} \rceil$ BP to P BP	$\lceil \frac{P}{2} \rceil - 1$ keys to P-1 keys

$\lceil \frac{P}{2} \rceil = t$

Q1: Height h ; max # keys in B Tree?



	# nodes	# BP	# keys
L_0	1	p	p-1
L_1	p	p x p	p x (p-1)
L_2	p^2	p^2 x p	p^2 x (p-1)
⋮	⋮	⋮	⋮
L_h	p^h	p^h x p	p^h x (p-1)

Q1: Height h ; max # keys in B Tree?

$$= (p-1) + (p-1)p + (p-1)p^2 + \cdots + (p-1)p^h$$

$$= (p-1)\left[1 + p + p^2 + \cdots + p^h\right]$$

$$= (p-1)\left[\frac{p^{h+1} - 1}{p-1}\right] = p^{h+1} - 1$$

Q2: Height h ; max #Nodes in B Tree?

$$= 1 + p + p^2 + \cdots + p^h$$

$$= \frac{(p^{h+1} - 1)}{p - 1}$$

Q: Data Structures: A Pseudocode Approach with C, 2nd Ed Richard F. Gilberg & Behrouz A. Forouzan

2. Calculate the maximum number of data entries in a:

   a. B-tree of order 5 with a height of 3
   b. B-tree of order 5 with a height of 5
   c. B-tree of order 5 with a height of h

Q: Data Structures: A Pseudocode Approach with C, 2nd Ed
Richard F. Gilberg & Behrouz A. Forouzan

2. Calculate the maximum number of data entries in a:

a. B-tree of order 5 with a height of 3 ⟹ HW without formula
b. B-tree of order 5 with a height of 5
c. B-tree of order 5 with a height of h

Order p ; Height h ; max # keys = $p^{h+1} - 1$

(a) $5^{3+1} - 1$

(b) $5^{5+1} - 1$

(c) $5^{h+1} - 1$

Order of B Tree is p:

Each B-tree node can have at most p tree(Block/Node) pointers, p − 1 data(Record or Block) pointers, and p − 1 search key field values.

Question Type:

For Given height, minimum number of keys that can be stored.

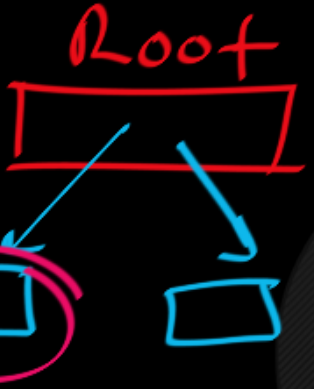
B-Tree: order p;

Root : 2 BP to p BP | 1 key to p-1 keys

Non Root : $\lceil \frac{p}{2} \rceil$ BP to p BP | $\lceil \frac{p}{2} \rceil - 1$ keys to p-1 keys

$= t$

Height h; Order p; min #keys ?



		#nodes	#Bp	#keys
L_0		1	2	1
L_1		2	$2 \times t$	$2 \times (t-1)$
L_2		$2t$	$2t \times t$	$2t \times (t-1)$
L_3		$2t^2$	$2t^2 \times t$	$2t^2 \times (t-1)$
L_h		$2t^{h-1}$	$2t^{h-1} \times t$	$2t^{h-1} \times (t-1)$

height h; order p: $t = \lceil \frac{p}{2} \rceil$

min # keys =

$$1 + 2(t-1) + 2t(t-1) + \cdots + 2t^{h-1}(t-1)$$

$$= 1 + 2(t-1)\left[1 + t + \cdots + t^{h-1}\right]$$

$$= 1 + 2(t-1)\frac{t^h - 1}{t-1} = 1 + 2(t^h - 1)$$

height h; order p:

$$t = \left\lceil \frac{p}{2} \right\rceil$$

min #nodes =

$$1 + 2 + 2t + \cdots + 2t^{h-1}$$

$$= 1 + 2\left[1 + t + \cdots + t^{h-1}\right]$$

$$= 1 + 2\left(\frac{t^h - 1}{t - 1}\right)$$

Q: Data Structures: A Pseudocode Approach with C, 2nd Ed
Richard F. Gilberg & Behrouz A. Forouzan

2. Calculate the minimum number of data entries in a:

a. B-tree of order 5 with a height of 3 ⇒ HW without formula
b. B-tree of order 5 with a height of 5
c. B-tree of order 5 with a height of h

$t = \lceil \frac{p}{2} \rceil = 3$; for height h :

min. #keys $= 1 + 2(t^h - 1)$

(a) $1 + 2(3^3 - 1)$

(b) $1 + 2(3^5 - 1)$

(c) $1 + 2(3^h - 1)$

Order of B Tree is p:

Each B-tree node can have at most p tree(Block/Node) pointers, p − 1 data(Record or Block) pointers, and p − 1 search key field values.

Question Type:

Given Number of search keys, maximum or minimum height that B tree can have.

Given: #keys

→ max height possible ?

→ min " " ?

Reverse Question

Given : # keys = n ; order : p ; $t = \lceil p/2 \rceil$

max height ≡ min no. of keys per node

→ Height = h

$$1 + 2\left(t^h - 1\right) = n$$

$$t^h = \left(\frac{n-1}{2} + 1\right)$$

$$h = \log_t\left(\frac{n+1}{2}\right)$$

Given: # keys ; order p

max Height :

$$h = \left\lfloor \log_t \left(\frac{n-1}{2} + 1 \right) \right\rfloor$$

→ floor value

$= 48.3$

max Height

max Height

$\begin{cases} 48 \checkmark \\ \text{or} \\ 49 \times \end{cases}$

Given : # keys = n ; order : p

min height $\equiv$ **max** no. of keys per node

h

$p^{h+1} - 1 = n$ $\Rightarrow$ $h = \lceil \log_p (n+1) \rceil - 1$

Given : #keys = n ; order : p

min height : $\Rightarrow h = \left(\log_p (n+1) \right) - 1$

$= 23.7$

24 ✓

23 ✗

min height

Best case and worst case heights

Let h be the height of the classic B-tree

Let $n \geq 0$ be the number of entries in the tree. Let m be the maximum number of children a node can have. Each node can have at most $m-1$ keys.

It can be shown (by induction for example) that a B-tree of height h with all its nodes completely filled has $n = m^{h+1}-1$ entries. Hence, the best case height (i.e. the minimum height) of a B-tree is:

$$h_{\min} = \lceil \log_m (n+1) \rceil - 1$$

Let d be the minimum number of children an internal (non-root) node must have. For an ordinary B-tree, $d = \lceil m/2 \rceil$.

Comer (1979) and Cormen et al. (2001) give the worst case height (the maximum height) of a B-tree as[20]

$$h_{\max} = \left\lfloor \log_d \frac{n+1}{2} \right\rfloor .$$

Q:

Consider a B-tree with Order P, where Order P is defined as usual (maximum number of child pointers a node can have).
The maximum levels of index required to store 400 distinct keys in order P= 5 B-tree index _____. ?

Q: with formula: $\max$ height = (min no. of keys per node) h

$t = \lceil P/2 \rceil = 3$

Consider a B-tree with Order P, usual (maximum number of c. have).

The maximum levels of index requir keys in order P= 5 B-tree index ___

$1 + 2(t^h - 1) = n = 400$

$h = \log_t\left(\frac{400+1}{2}\right)$

#levels = height + 1 $= \log_3\left(\frac{401}{2}\right) = \log_3(200.\ldots)$

$\log_3 (3^{4\ldots})$

$= 4\ldots$

$\Rightarrow$ max height

$3^4 = 81$

$3^5 = 81 \times 3$

$= 243$

Somewhere b/w

$\#\text{levels} = 5$ ✓

$h = 4$

Q: Conceptually : ⇒ BEST WAY

Consider a B-tree with Order P, where Order P is defined as usual (maximum number of child pointers a node can have).

The maximum levels of index required to store 400 distinct keys in order P= 5 B-tree index _____. ?

#keys = 4000 ; P = 5 ;

Root : 1 key to 4 keys

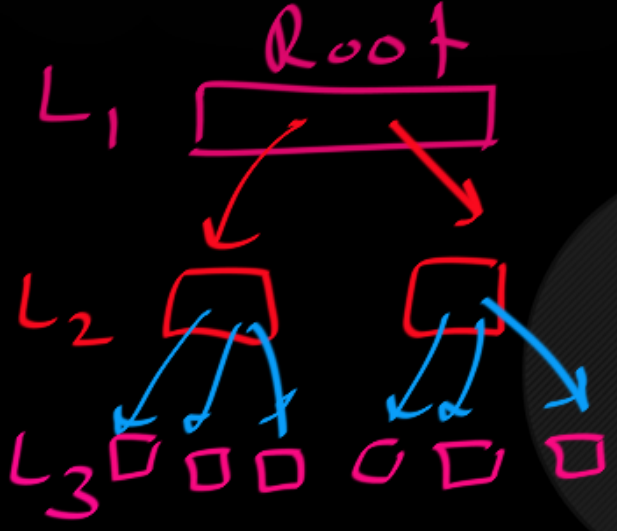
Non Root : 2 keys to 4 keys

maximum no. of levels :

fill every node as less as possible.

minimum keys possible

$\lceil \frac{P}{2} \rceil = 3$



L_4

L_5

L_6

	min #Nodes	min #BP	min #keys
L_1	1	2	1
L_2	2	2 × 3	2 × 2 = 4
L_3	6	6 × 3	6 × 2 = 12
L_4	18	18 × 3	18 × 2 = 36
L_5	54	54 × 3	54 × 2 = 108
L_6	162	162 × 3	162 × 2 = 324

till Level 5 : # keys : 108 + 36 + 17 = 161 (min)

till level 6 : # keys : 161 + 324 = 485 (min)

keys we have : 400 ⇒ { 5 levels OR 6 levels ✗

If we have level 6 :

minimum 485 keys we need to have.

Variation in Prev. Q:

keys = 400 ; Order p = 5 ;

find min. Height

fill every node as much as possible

# Nodes	# BP	# keys
1	5	4
5	5 × 5	5 × 4 = 20
25	25 × 5	25 × 4 = 100
125	125 × 5	125 × 4 = 500

till level 3: #keys = 124 max

till level 4: #keys = 624 max

#keys we have: 400

If we have 3 levels:
max 124 keys we can put;

$\Rightarrow$ level 3 ✗
or
level 4 ✓

Height = 3

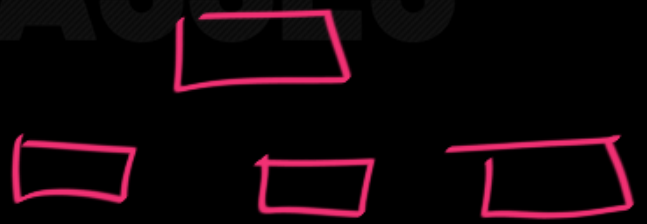
Navathe:

in Data file

Example 5. Suppose that the search field is a nonordering key field, and we construct a B-tree on this field with $p = 23$. Assume that each node of the B-tree is 69% full. (at least 69% full)

Find the number of keys stored in 4 Levels (Height 3)?

B-Tree:



Navathe: p: max # Node pointers in Data file

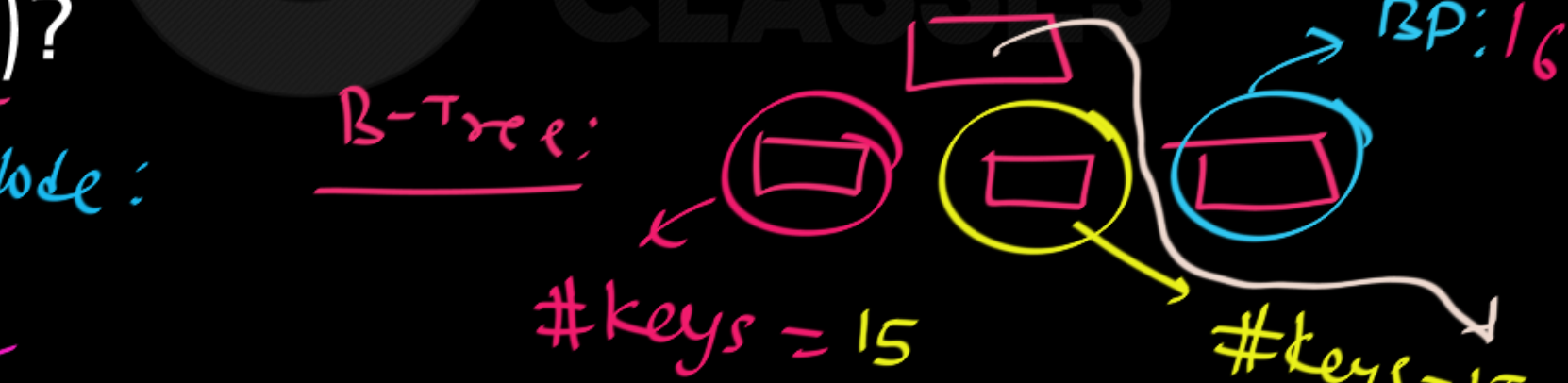
Example 5. Suppose that the search field is a nonordering key field, and we construct a B-tree on this field with $p = 23$. Assume that each node of the B-tree is 69% full.

69% of order

Find the number of keys stored in 4 Levels (Height 3)?

Bp Per Node :
23 X 0.69
= 16

B-Tree:

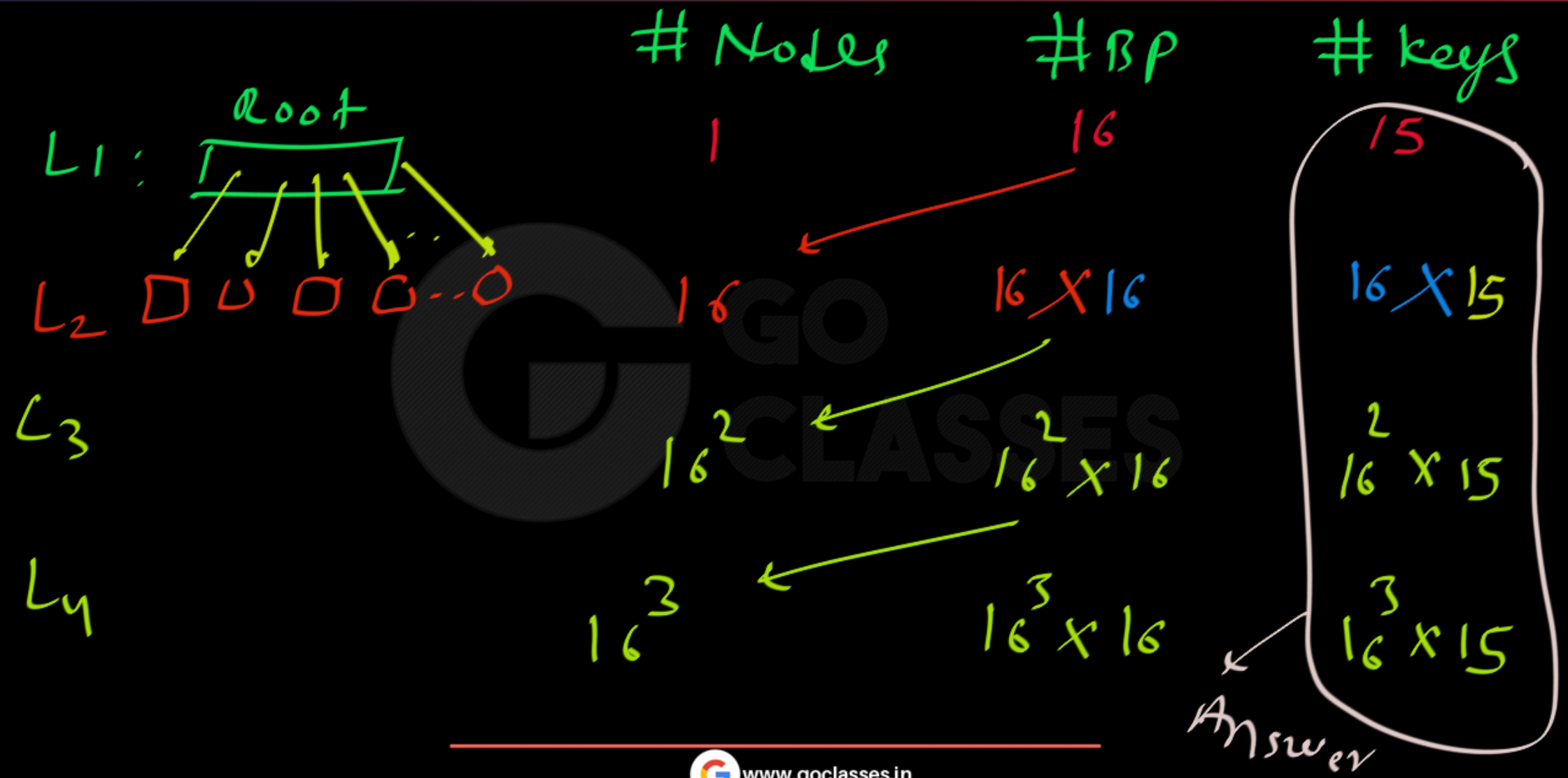


max: 9 TV

50% of it: 5 TV ✓

B-Tree:

→ Every Node: 15 keys; 16 BP


Nodes
BP
keys
Root
L1 :
L2
L3
L4
1
16
15
16
16 X 16
16 X 15
16^2
16^2 X 16
16^2 X 15
16^3
16^3 X 16
16^3 X 15
Answer

Example 5. Suppose that the search field is a nonordering key field, and we construct a B-tree on this field with $p = 23$. Assume that each node of the B-tree is 69% full. Each node, on the average, will have $p * 0.69 = 23 * 0.69$ or approximately

16 pointers and, hence, 15 search key field values. The **average fan-out** $fo = 16$. We can start at the root and see how many values and pointers can exist, on the average, at each subsequent level:

Root:	1 node	15 key entries	16 pointers
Level 1:	16 nodes	240 key entries	256 pointers
Level 2:	256 nodes	3,840 key entries	4,096 pointers
Level 3:	4,096 nodes	61,440 key entries	

At each level, we calculated the number of key entries by multiplying the total number of pointers at the previous level by 15, the average number of entries in each

Which of the following items is **FALSE** about B-trees?

- A binary search can be used to search within a node in a B-tree.
- A B-tree is a multi-way tree with many data items per node.
- A B-tree is a binary tree.
- A B-tree can be used for external storage.

Which of the following items is **FALSE** about B-trees?

- A binary search can be used to search within a node in a B-tree. True
- A B-tree is a multi-way tree with many data items per node. True
- A B-tree is a binary tree. — False
- A B-tree can be used for external storage. True

B+ Tree

Next...