FILE ORGANISATION Vs INDEXING.

File Organisation : → Determines how records are stored on disk

→ Provides primary acess mode
Eg: Unordered heaps, ordered files.

Indexing : → Provides secondary access methods
Eg: primary Indexing, B + trees.

* Not all Indices and file organisation can be used together.

↪ Eg: You cannot use a primary indexing on an unordered file organisation since the records necessarily have to be ordered by primary indexing field.

UNORDERED HEAPS (As per Project)

→ Data used in this project is Z^+ (non -ve Int)

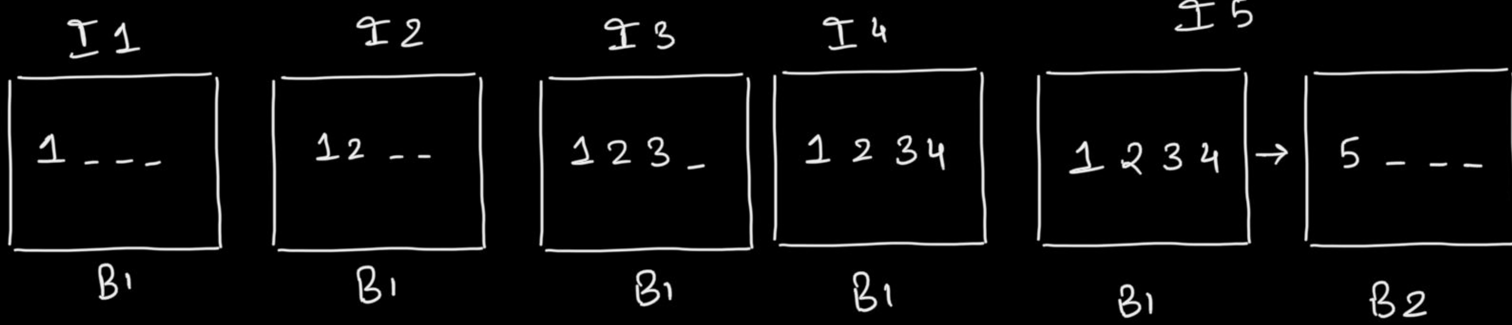
→ These records are stored in Linked List of Blocks → file organisation for the project.

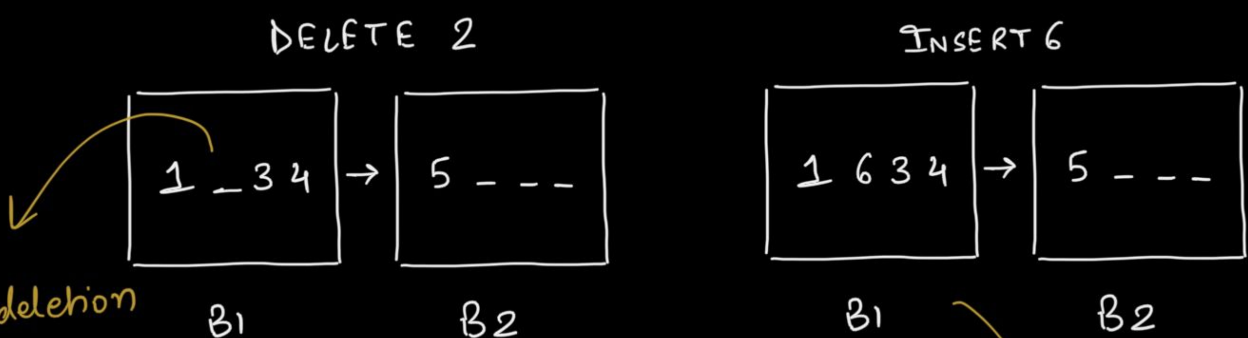
Classes : Blocks, Unordered Heaps.

* A block can store at most BLOCK_SIZE (=4) elements

→ We insert only unique records (keys) in a B+ Tree or the unordered heap.

Example on Unordered Heap





Upon deletion that particular record psn is replaced by a DELETE MARKER (-1 in our case)

There are multiple Deletion strategies (mentioned in the text book)

The strategy we use is using Delete marker without File reorganisation.

Inserting again will search for the first delete marker and replace it with the new record

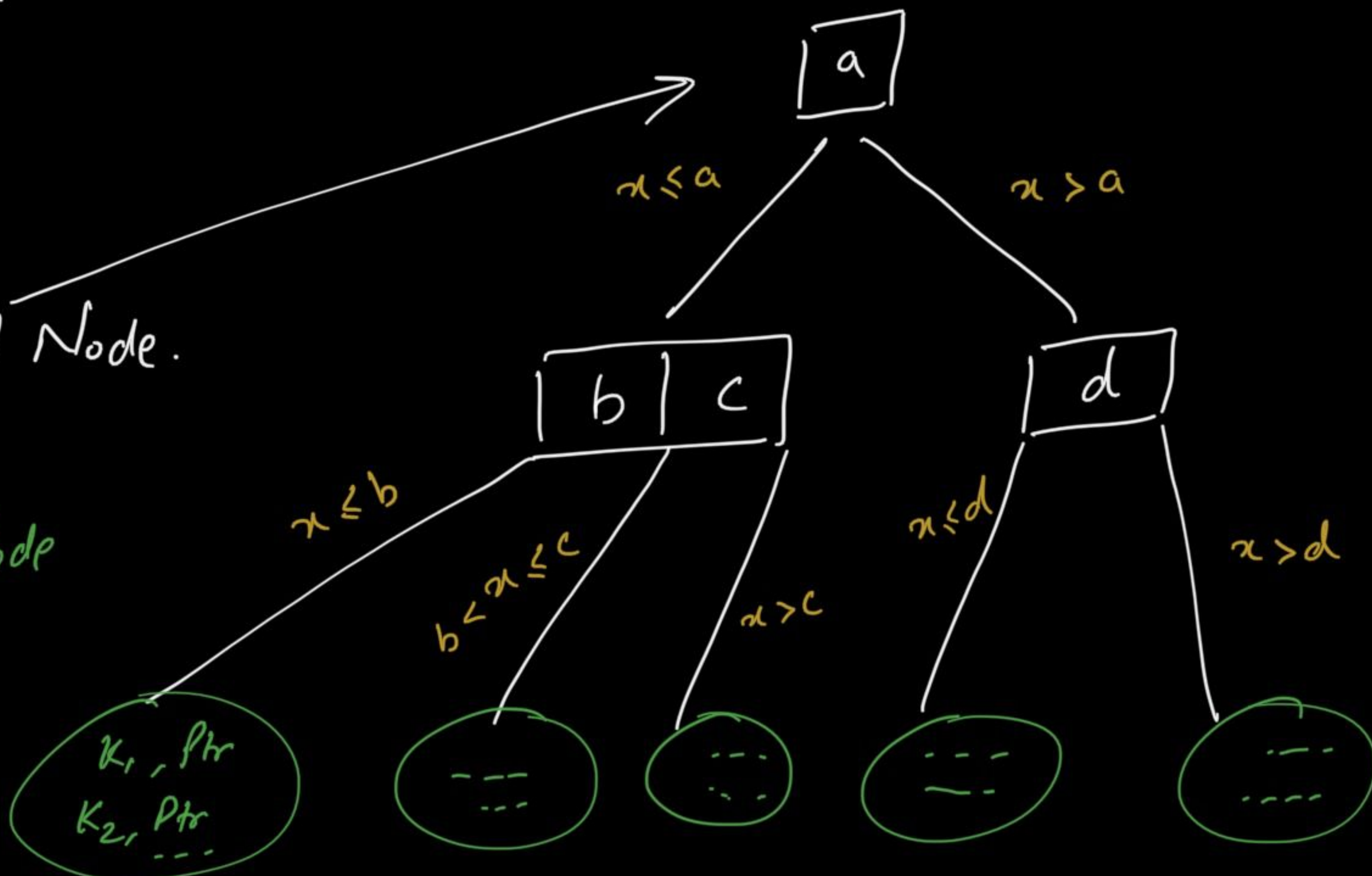
The above part (Unordered heaps) is completelely implemented and nothing is to be done for the same.

B + Trees

The tree structure Consists of Nodes

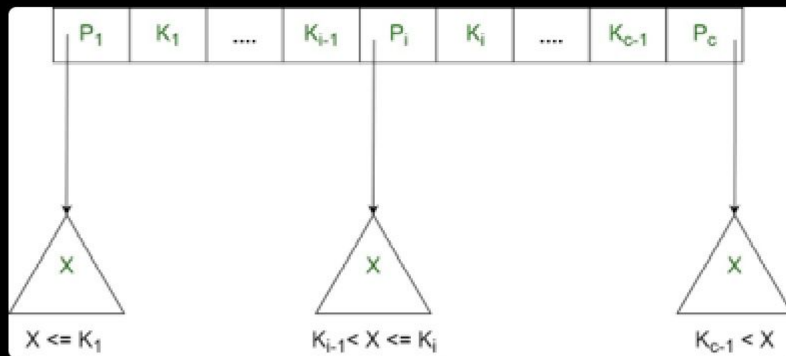
Internal Node.

Leaf Node



PROPERTIES of B+ Trees.

INTERNAL NODES



$\langle P_1, K_1, P_2, K_2, P_3 \ldots \rangle$

1) $K_1 < K_2 \ldots\ldots < K_{c-1}$

2) Every Element in

P_i

is $\le K_i$

and $> K_{i-1}$

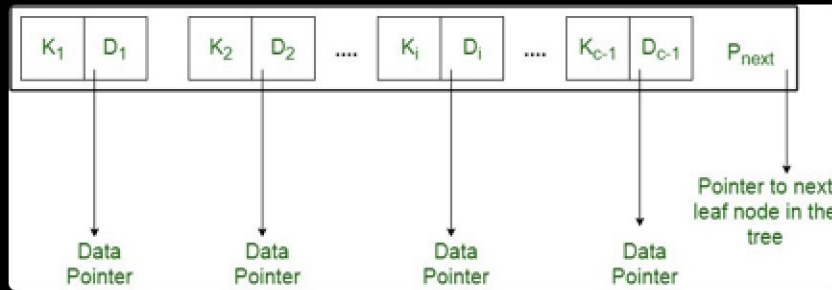
3) Every Internal Node (Except Root Node) has

atleast $\left\lceil \frac{FANOUT}{2} \right\rceil$ child nodes.

and **atmost** FANOUT child nodes.

LEAF NODES

→ $\langle K_1, P_1 \rangle, \langle K_2, P_2 \rangle$ ---- ---- $\langle K_{c-1}, P_{c-1} \rangle$



→ $K_1 < K_2$ ---- $< K_{c-1}$

→ D_1 → Data Pointer to record 1 (Block ptr or $\underbrace{\text{record ptr}}$)

{ Class = Record Ptr
↳ where data for this key exist on the disk }

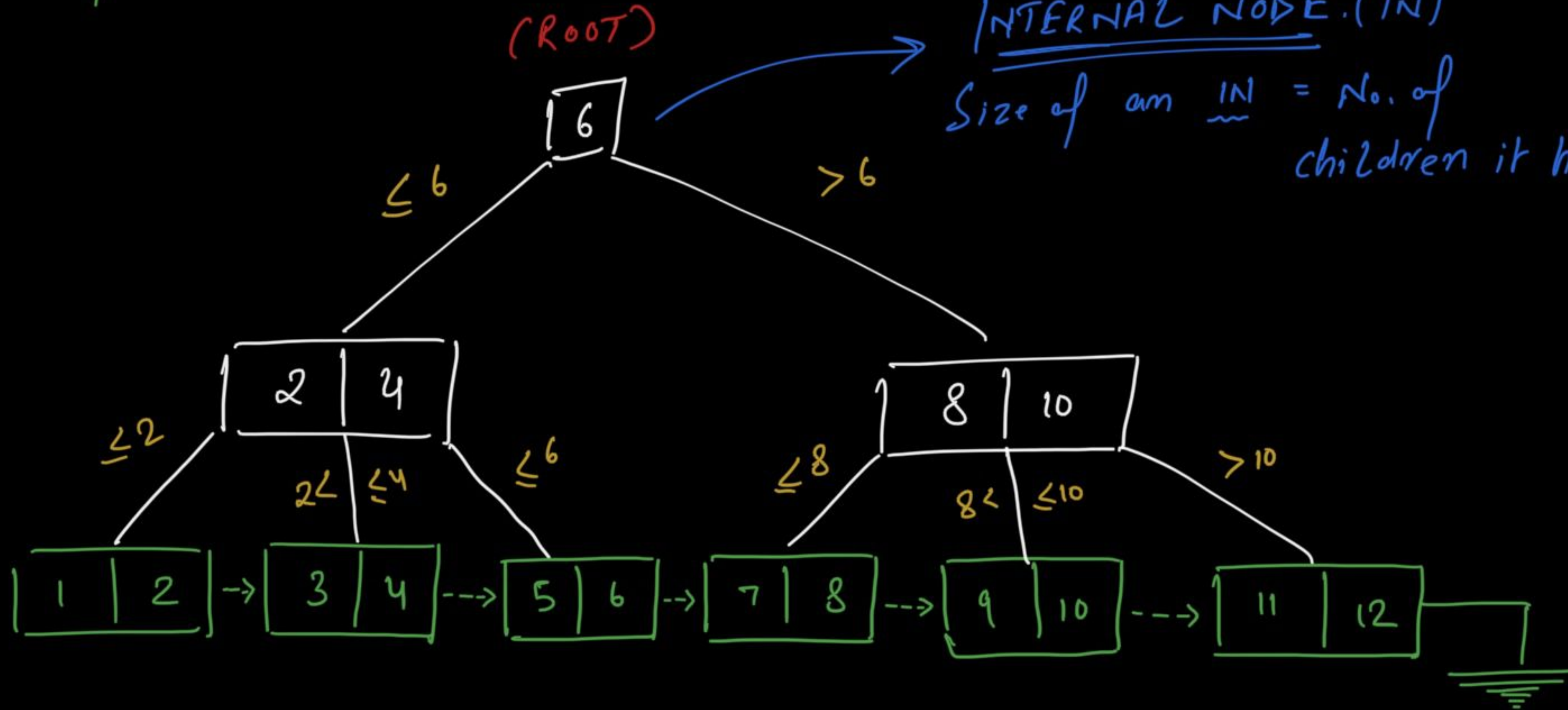
→ Every leaf Node (Except root node) must have atleast $\left\lceil \frac{\text{FANOUT}}{2} \right\rceil$ $\langle$ Key, Ptr $\rangle$

and atmost FANOUT $\langle$ Key, ptr $\rangle$ Pairs.

* Record Ptr in our case is the block ptr of that particular key in the unordered heap along with that key's pos^n in that block.

You can technically have different FANOUTS for leafNodes and Internal nodes, but to keep implementation and understanding easier for this project we'll have them equal.

Example of B+ Tree (FANOUT = 3)



INTERNAL NODE (IN)

Size of an IN = No. of children it has

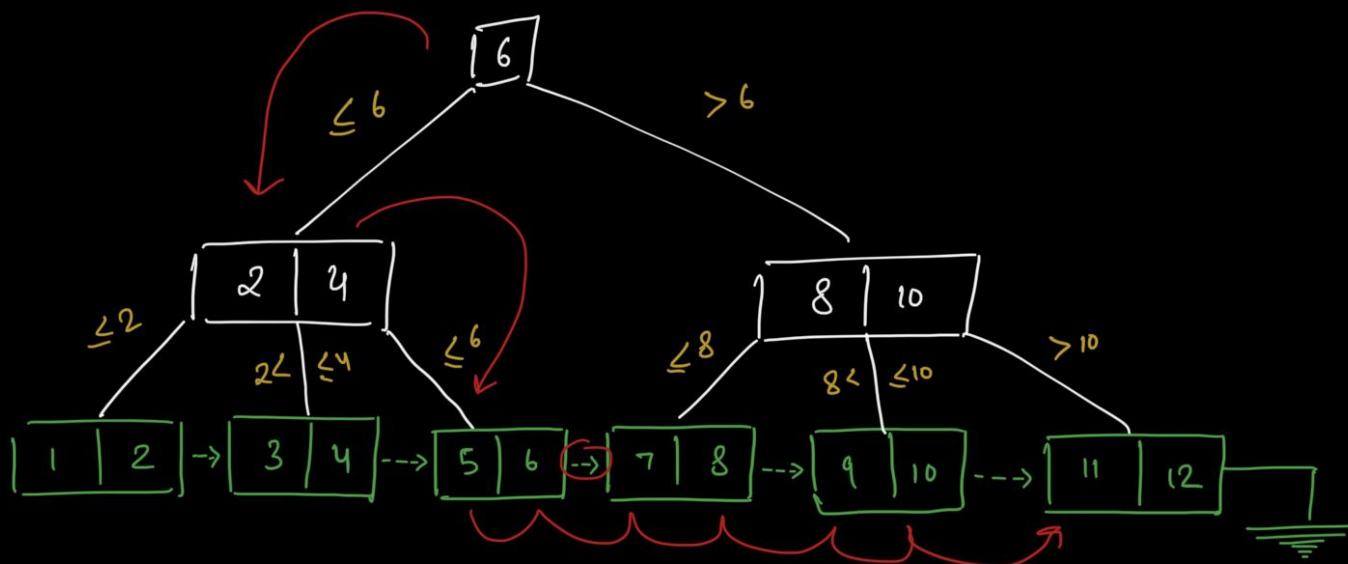
Q: SERCH : (RANGE MIN_KEY MAX_KEY)

Q: INSERTION :

Q: DELETION :

RANGE

RANGE Min_Key Max_Key : Returns all the keys b/w Min and Max keys passed. Including min, max

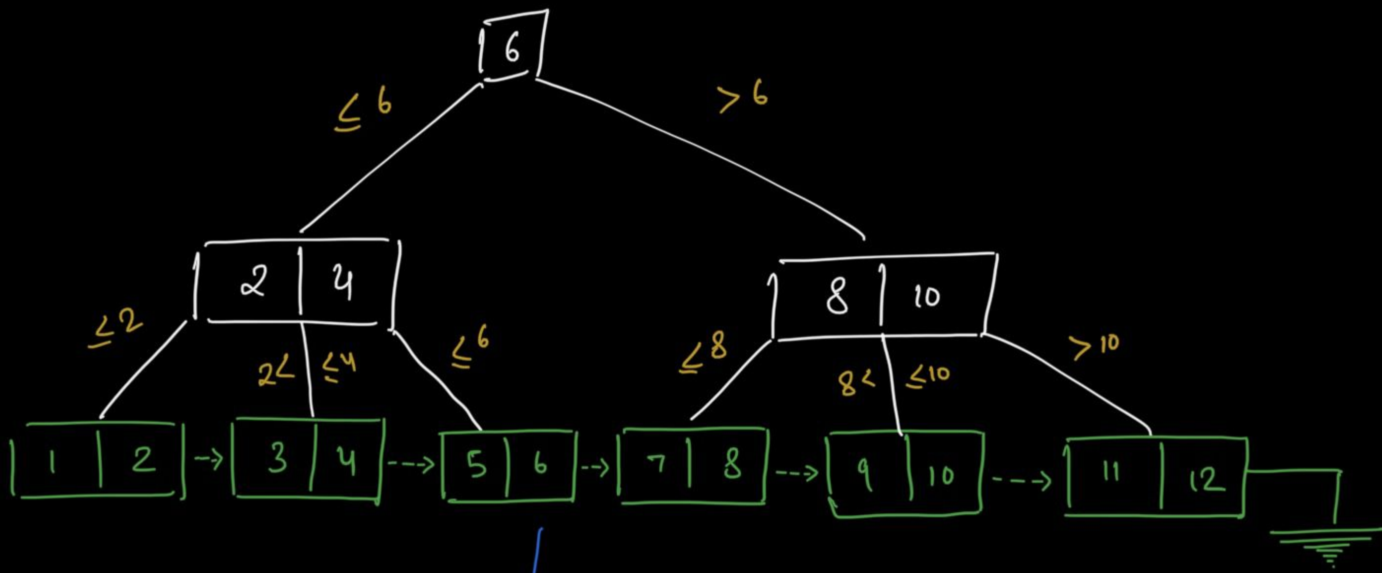


How range func works is it drops down to the posⁿ at which min_key is present and traverses along the linked list untill max_key is reached counting the no. of elements b/w them.

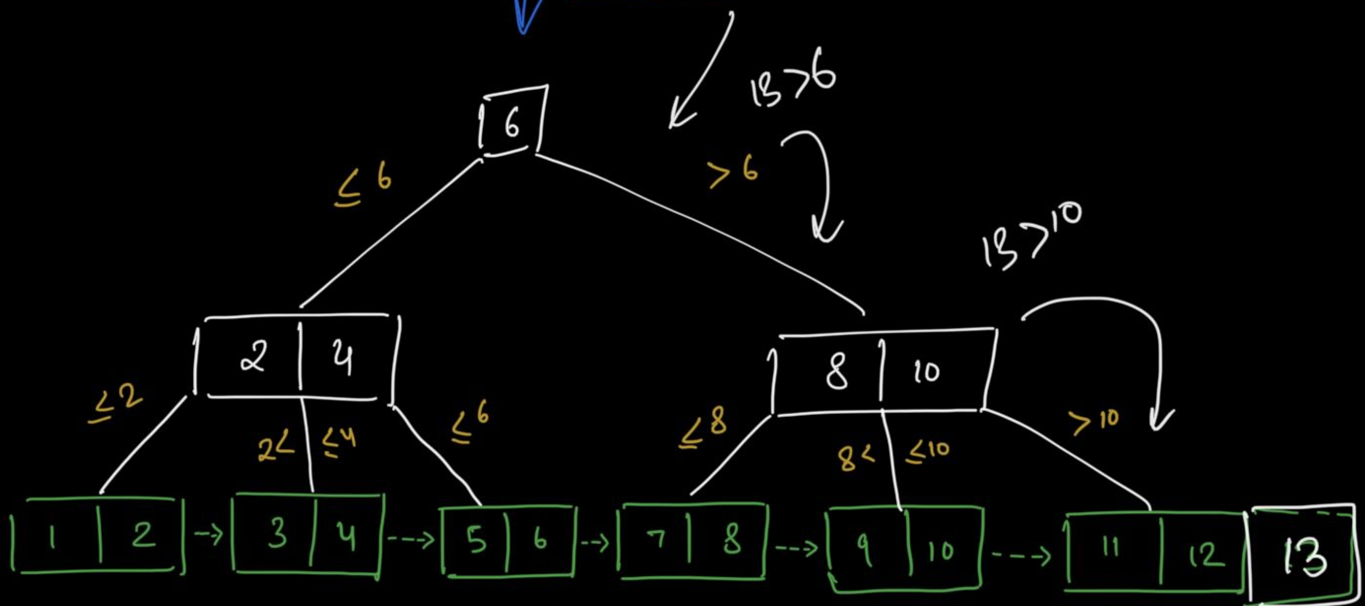
* RANGE funcⁿ can be used to SEARCH for a particular element. To see if a particular element exist in the B+ Tree

RANGE Key Key → Min Key == Max Key.

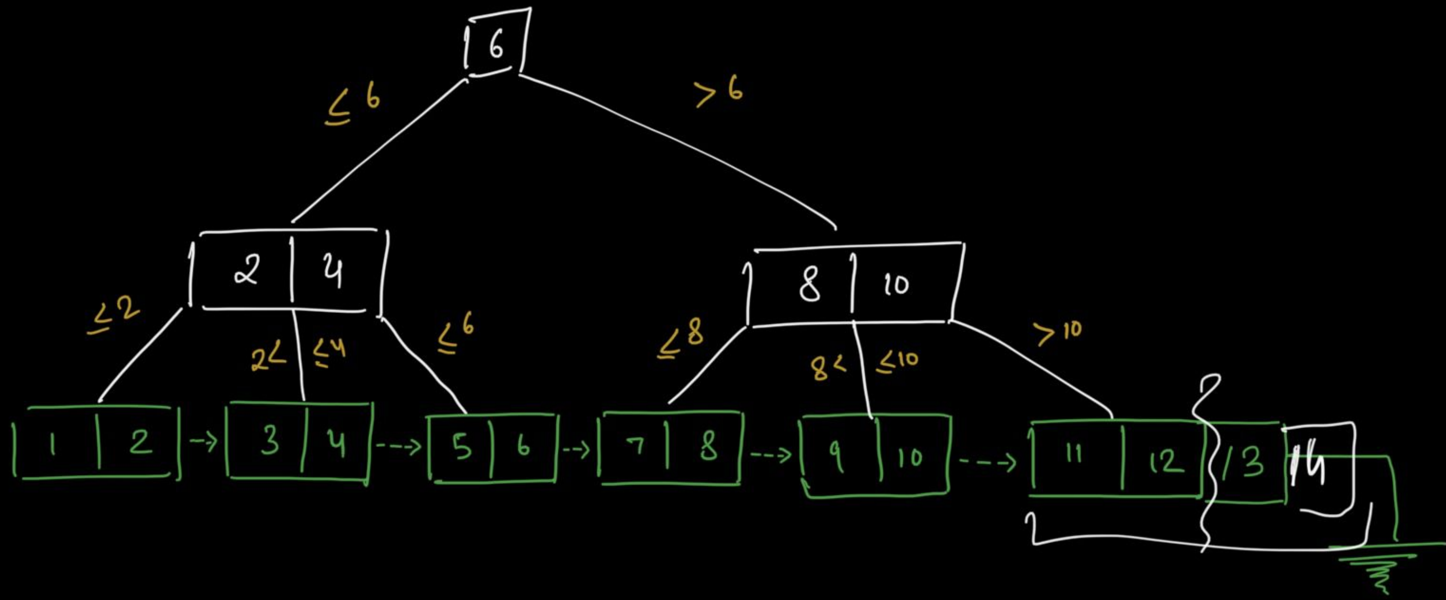
INSERTION



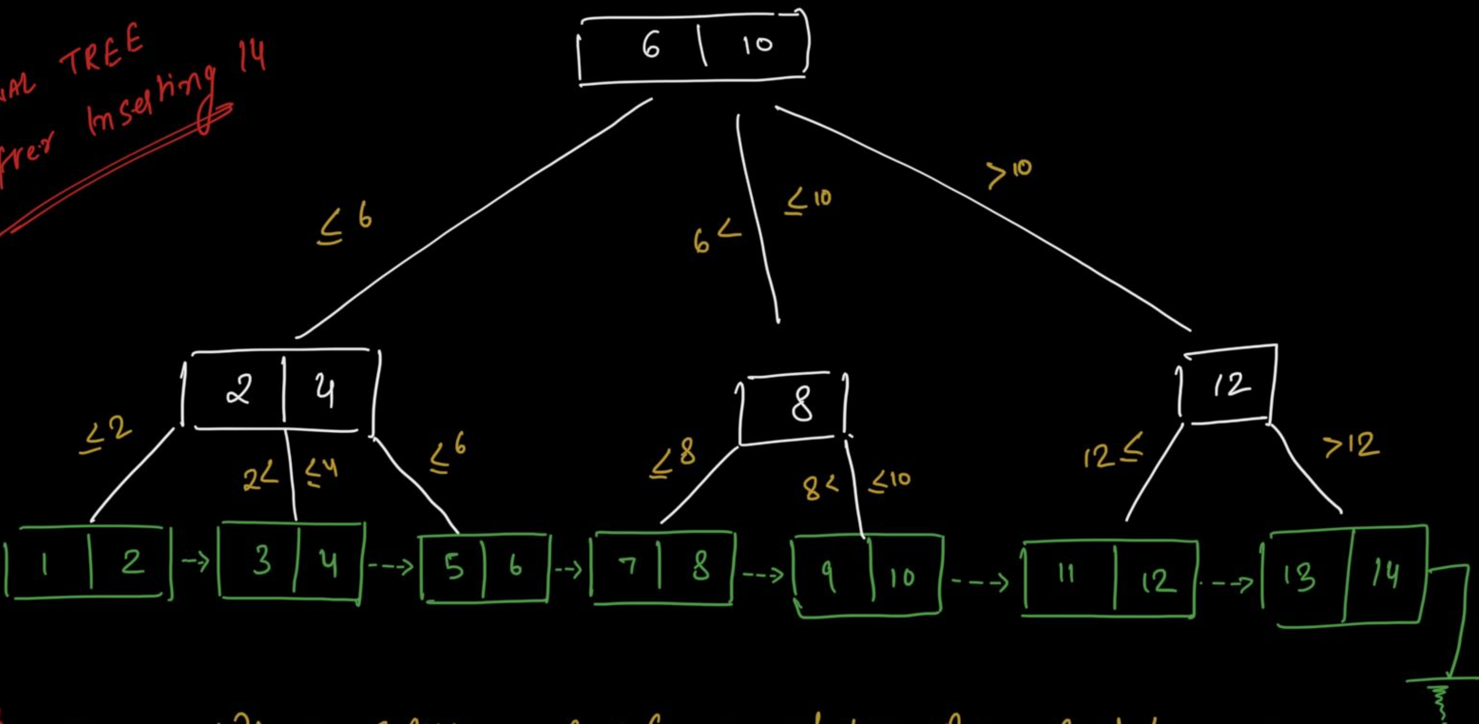
INSERT 13



INSERT 14



Final TREE
After Inserting 14



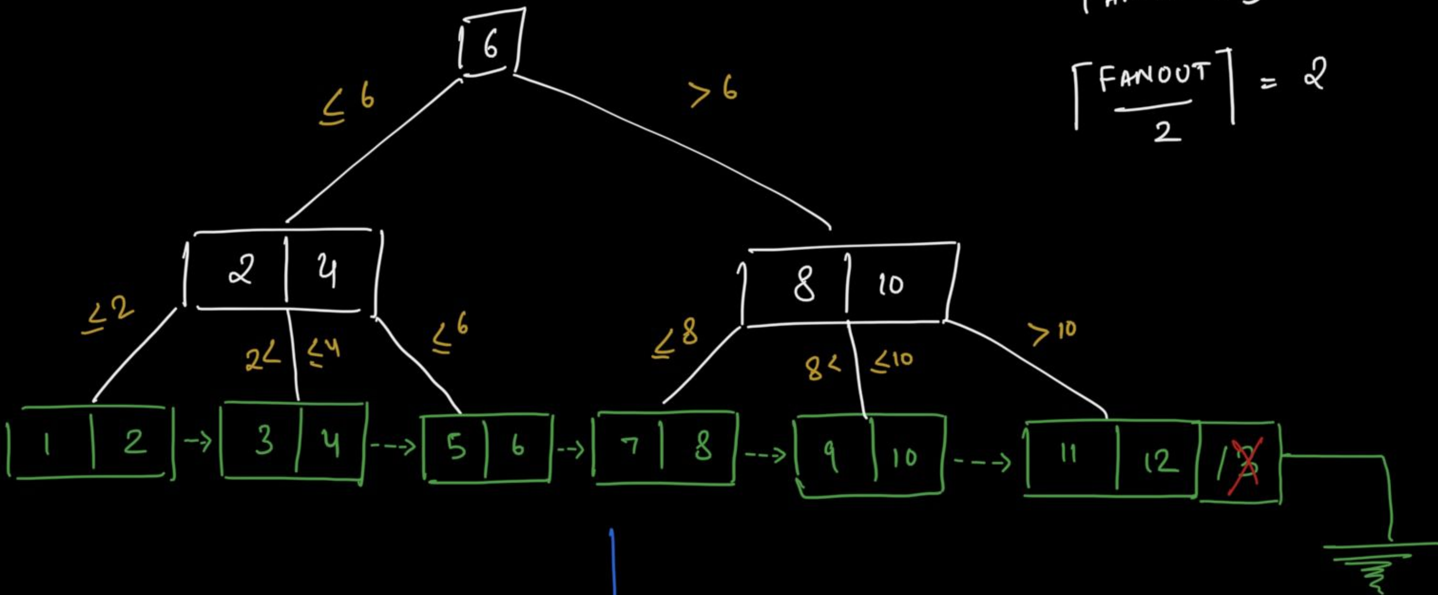
NOTE: When Splitting Overflown Nodes, Orignal Node gets $\lceil \frac{FANOUT}{2} \rceil$ children and the rest goes to the new node.

→ While Inserting return the new split key to the parent node and keep repeating the process of adding new key and splitting untill reach a steady state.

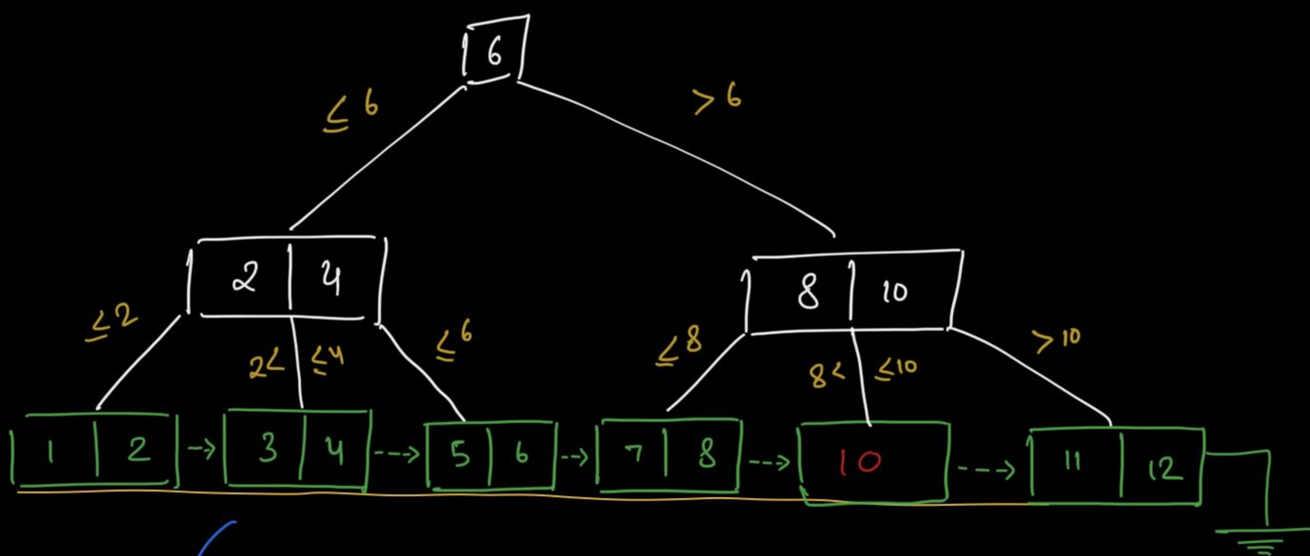
DELETION

$\text{FANOUT} = 3$

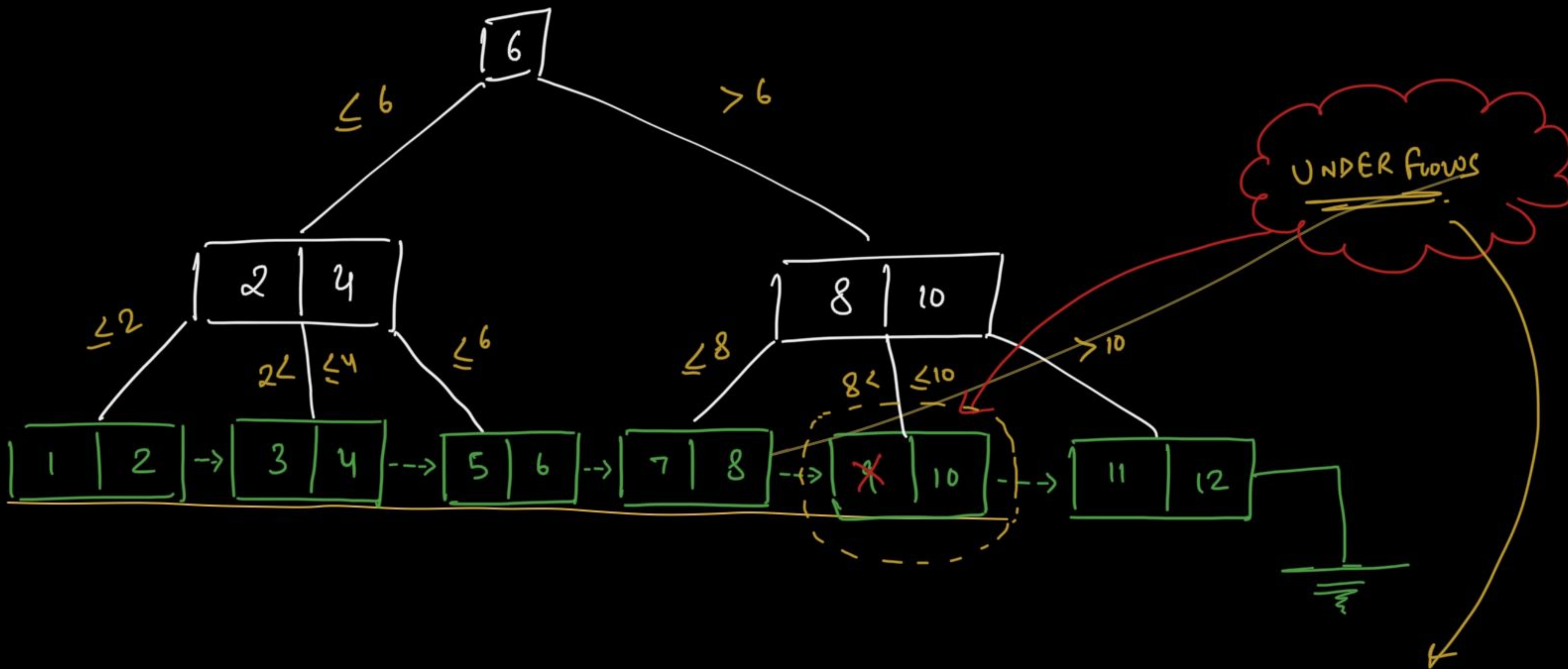
$\left\lceil \frac{\text{FANOUT}}{2} \right\rceil = 2$



DELETE 13



What do you think will happen if we perform DELETE 9 on this tree??



Since after the deletion this leaf will have only 1 record which is lesser than the minimum requirement of leaf node.

There are majorly 2 ways of Handelin Underflows

1) Redistribution with a sibling:

$$\text{If}\Big(\underbrace{\text{this}}_{\text{Underflown Node.}}.\text{Size} + \text{Sibling.Size} \geq 2 * \underbrace{\left\lceil \frac{\text{FANOUT}}{2} \right\rceil}_{\text{Minimum condition for one Node.}}\Big)$$

$$\text{this} \xleftarrow[\text{Takes children from sibling.}]{} \text{sibling}$$

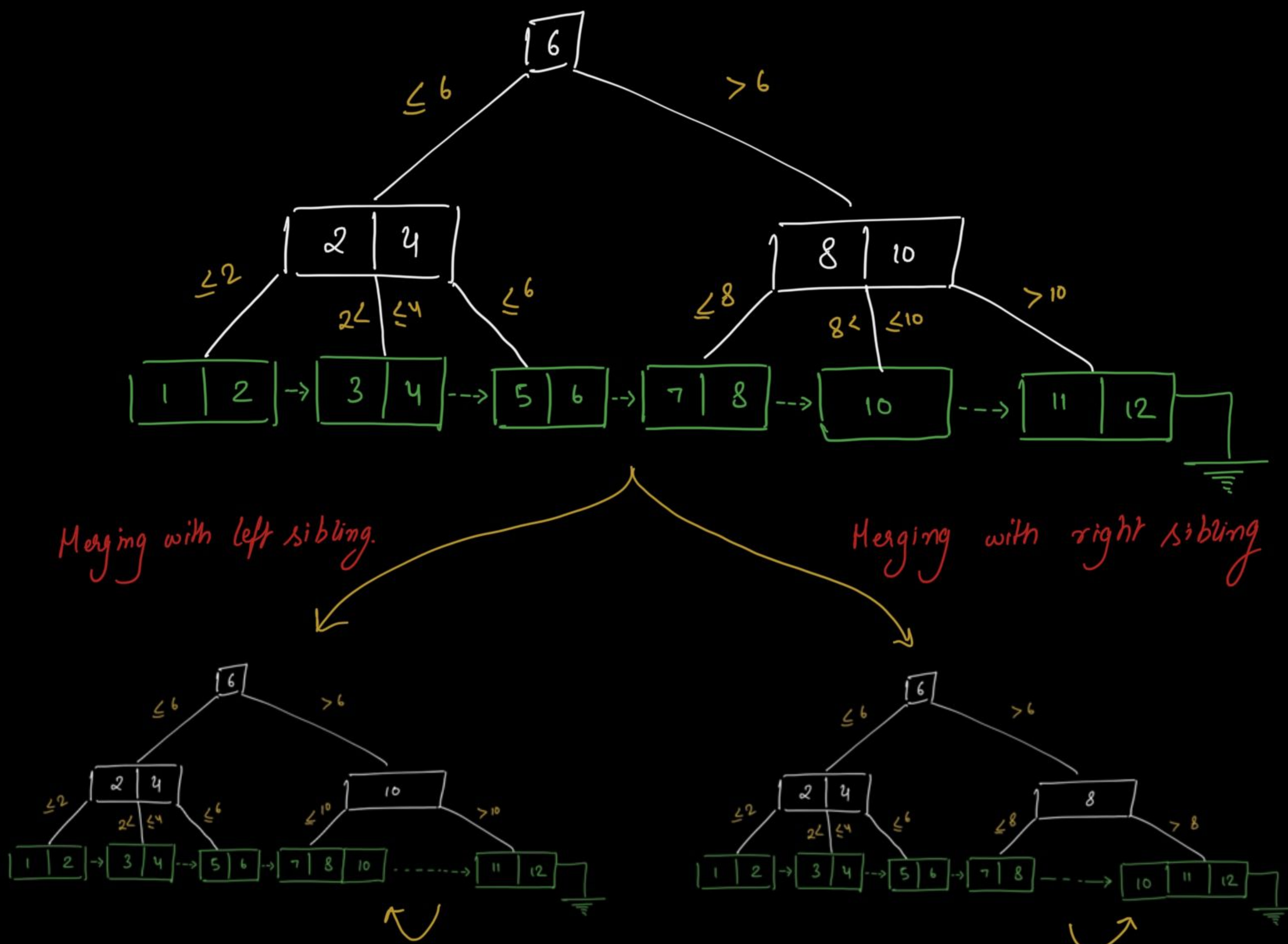
2) Merge with a sibling

$$\text{If}\ (\text{this.Size} + \text{sibling.size} \leq \underbrace{\text{FANOUT}}_{\text{Maximum condition for a node.}})$$

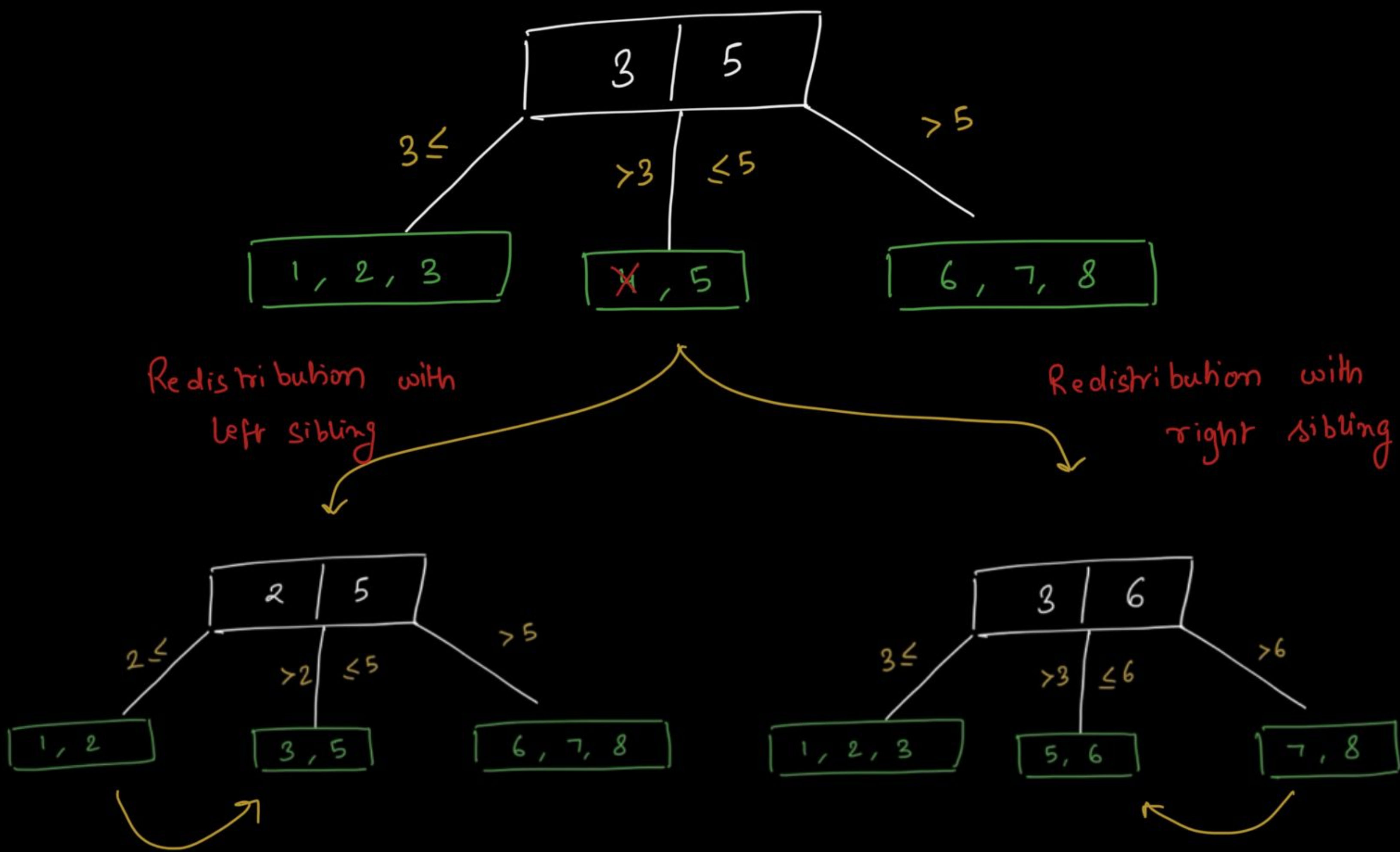
$$\text{new node} \longleftarrow \text{this} \underbrace{+}_{\text{merge}} \text{Sibling}$$

* It is not always possible to do both redistribution and Merging. Usually you can only do one of either.

Coming Back to our Example



Redistribution Example.



* NOTE: When redistributing with sibling, the underflown node gets exactly $\lceil \frac{\text{FANOUT}}{2} \rceil$ keys after redistribution.

PREFERENCE ORDER FOR DELETION

1) Redistribute with Left sibling
2) Merge with left sibling
3) Redistribute with Right sibling
4) Merge with Right sibling.

One of the 4 is necessarily possible,

Whenever you insert something to memory, insertion on unordered heap occurs at first, which returns a Record Ptr.

This Record Ptr along with the key is then used for insertion in the B + Tree.

B + Tree

↓

Class : Tree Node.

Internal Node

Leaf Node.

(Inherited Classes)