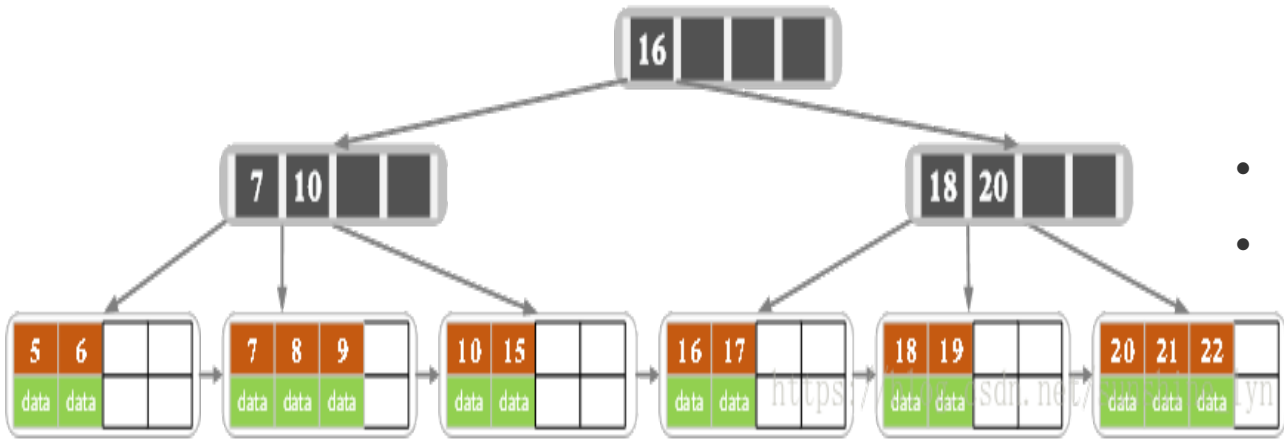
索引上的并发控制问题
(B+树为例)

B+树的数据结构



- 从根节点到叶节点的所有路径都具有相同的长度
- 所有数据信息都存储在叶节点上，非叶节点仅作为叶节点的索引存在
- 每个树节点最多拥有M个键值对
- 每个树节点（除了根节点）拥有至少M/2个键值对

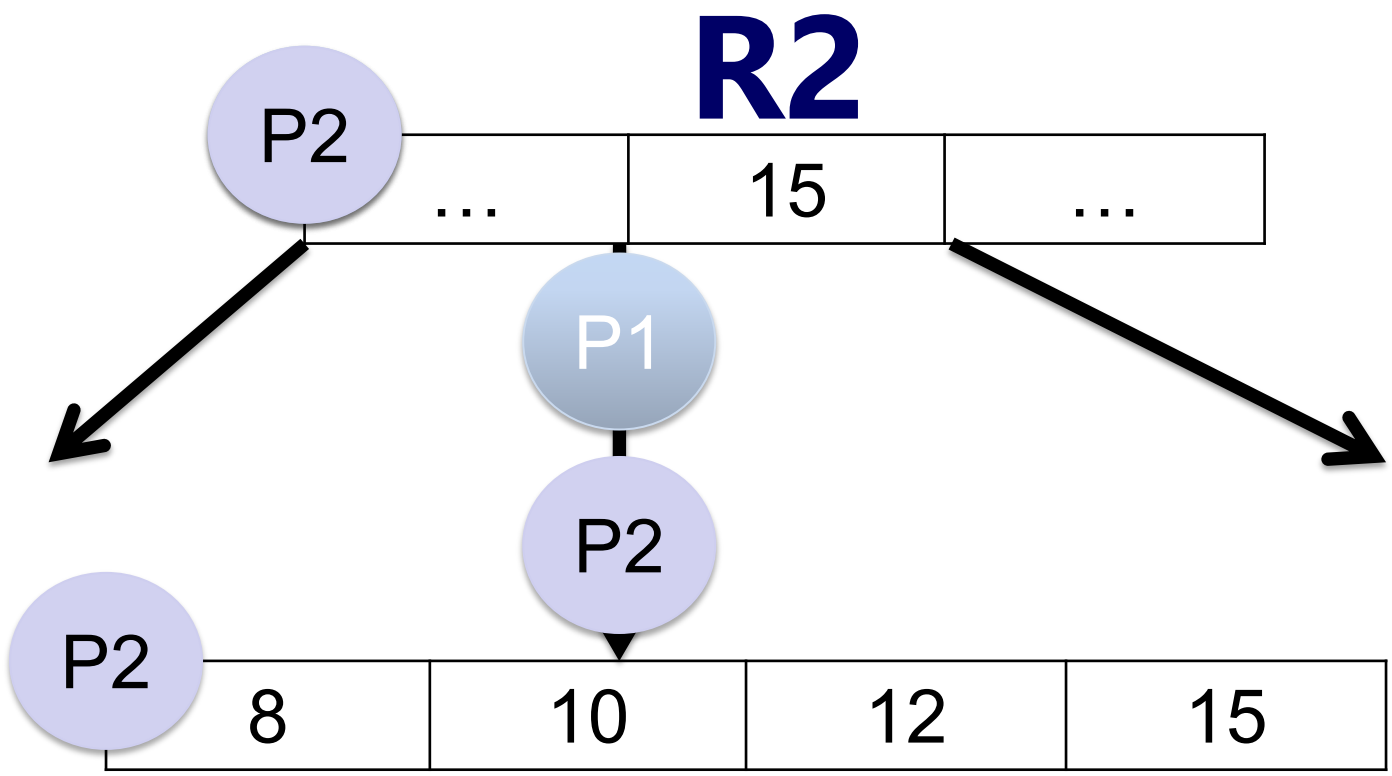
- B+树需支持以下操作：
- 单键值操作：Search/Insert/Update/Delete（以Search/Insert操作为例)
- 范围操作：Range Search

B+树并发控制的要求

- 正确的读操作：
  - R.1 不会读到一个处于中间状态的键值对：读操作访问中的键值对正在被另一个写操作修改
  - R.2 不会找不到一个存在的键值对：读操作正在访问某个树节点，这个树节点上的键值对同时被另一个写操作（分裂/合并操作）移动到另一个树节点，导致读操作没有找到目标键值对
- 正确的写操作：
  - W.1 两个写操作不会同时修改同一个键值对

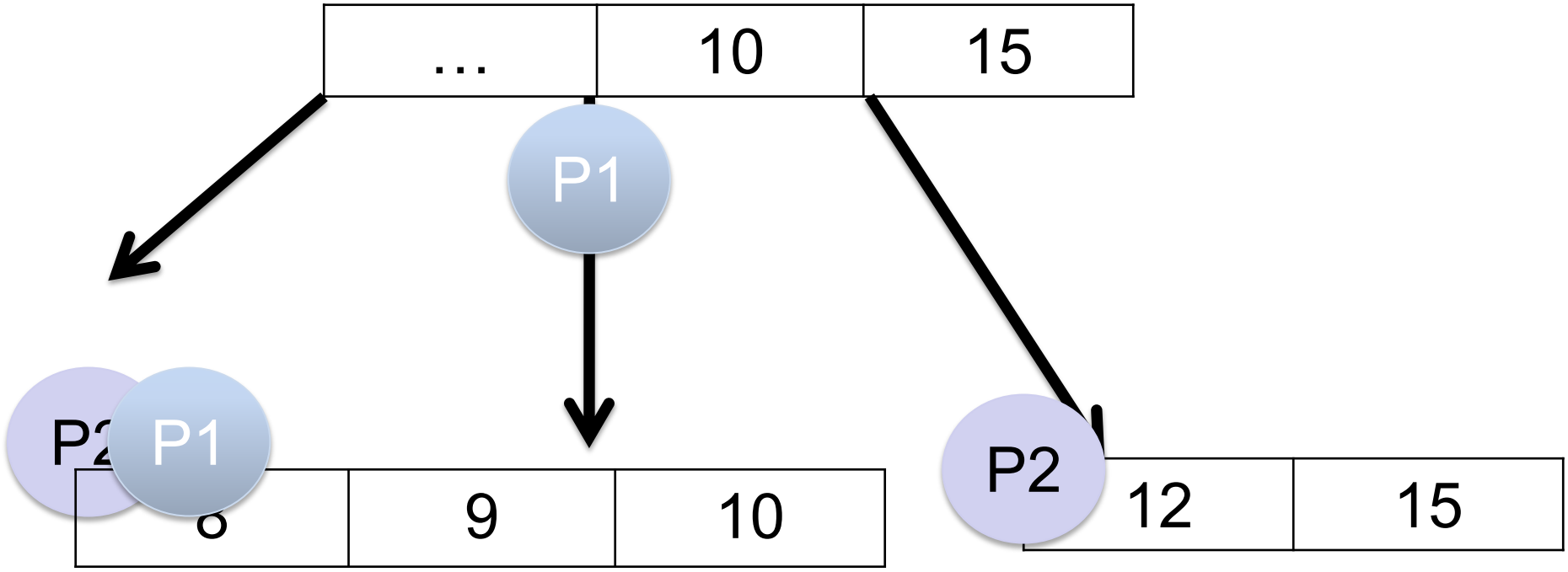
R2的并发控制需求

- R1, W1 是常见的写写冲突与读写冲突，如果是单版本存储，几乎任何时候都存在
- R2是B+树特殊的并发控制需求
  - 场景： 考虑一个读操作刚拿到一个叶子节点的指针，打算访问这个节点后半部分的数据，此时一个写操作发生使得节点分裂，那么后半部分数据将分裂到另一个节点上，因此读操作的指针无法获得目标数据。
  - 简单的对数据加锁无法保证R2的需求。



- P1 searches for 15
- P2 inserts 9

After the Insertion



- P1 searches for 15
- P2 inserts 9

P1 Finds no 15!

How could we fix this?

LATCH CRABBING

Acquire and release latches on B+Tree nodes when traversing the data structure.

A thread can release latch on a parent node if its child node considered **safe**.

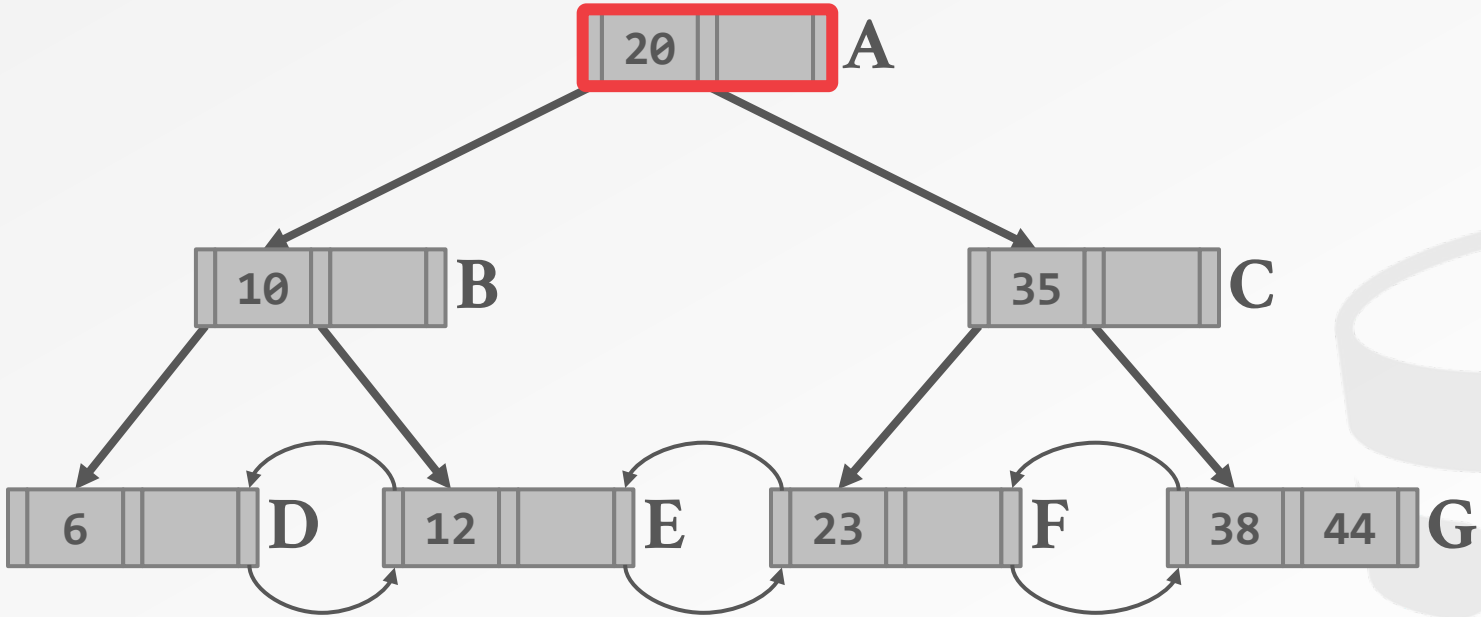
→ Any node that won't split or merge when updated.
→ Not full (on insertion)
→ More than half-full (on deletion)

LATCH CRABBING

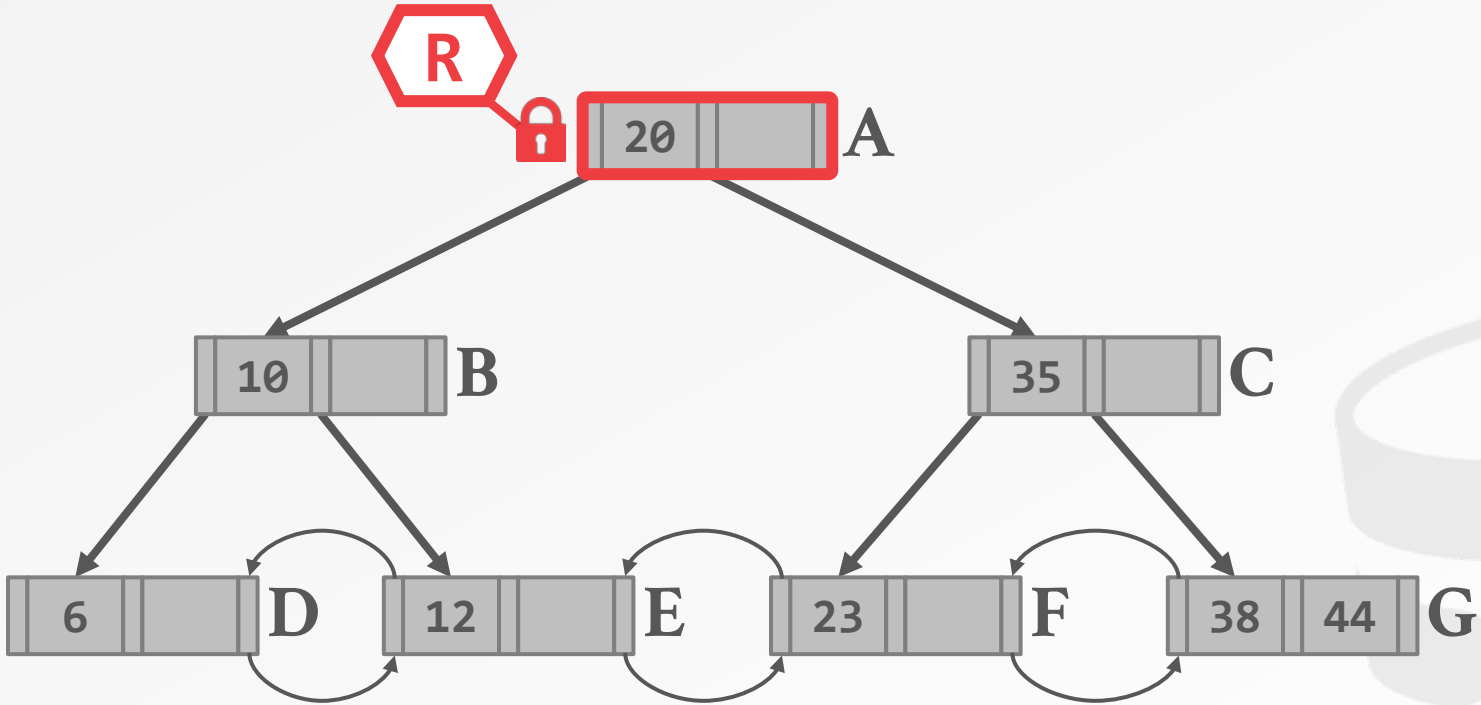
Search: Start at root and go down; repeatedly,
→ Acquire read (R) latch on child
→ Then unlock parent if the child is safe.

Insert/Delete: Start at root and go down, obtaining write (W) latches as needed.
Once child is locked, check if it is safe:
→ If child is safe, release all locks on ancestors.

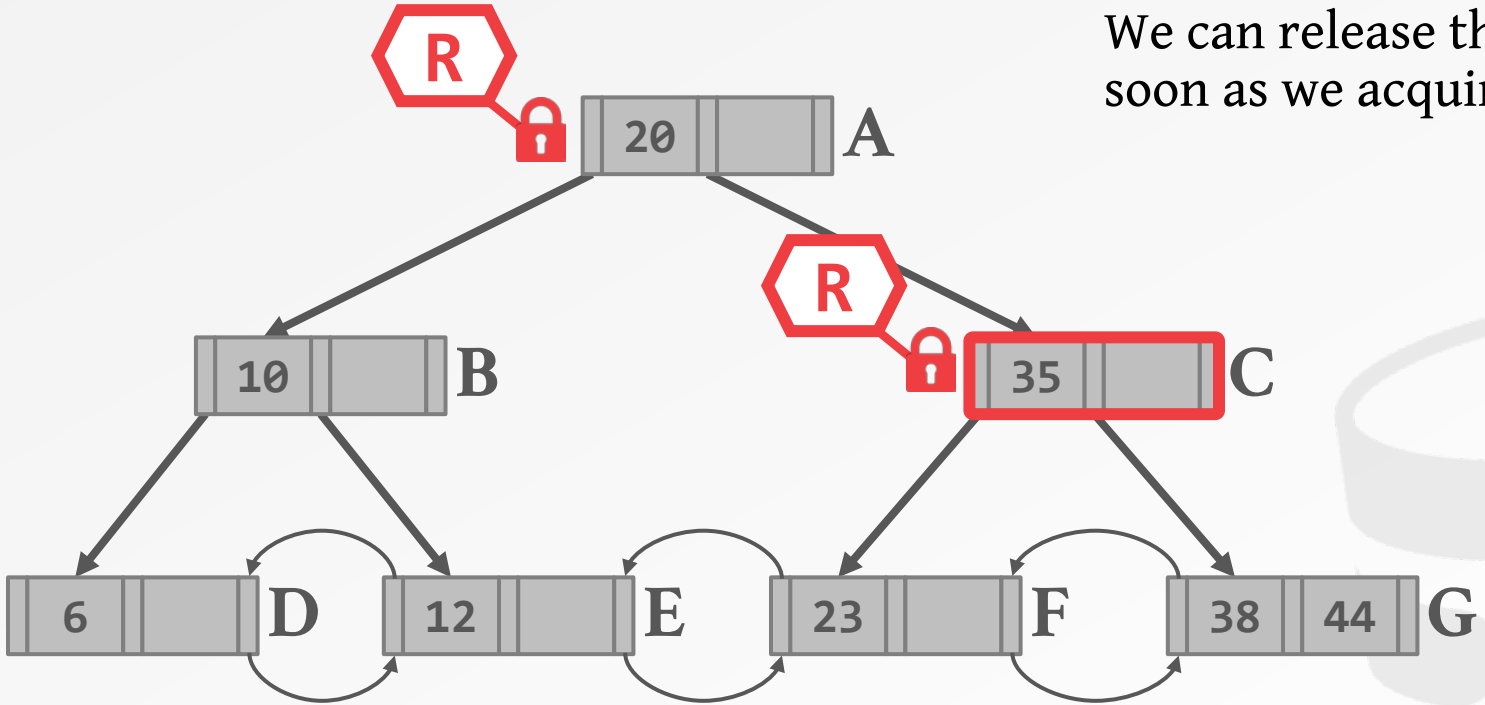
EXAMPLE #1: SEARCH 23



EXAMPLE #1: SEARCH 23



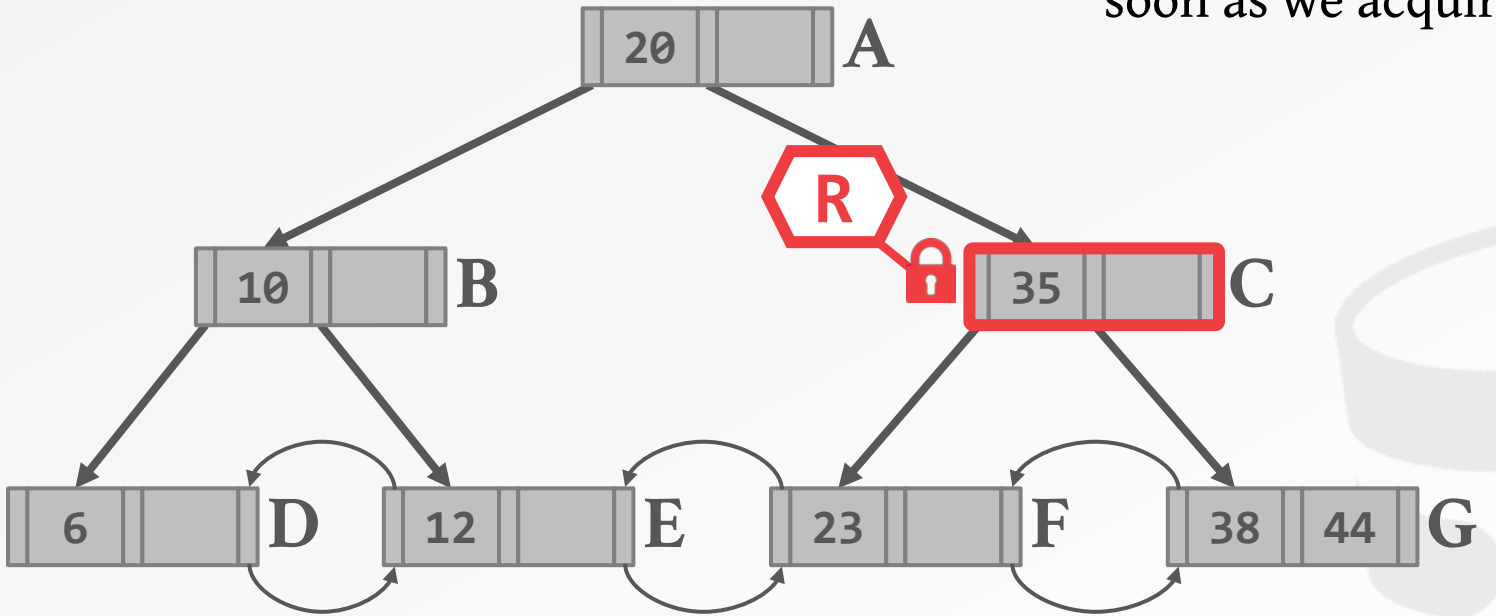
EXAMPLE #1: SEARCH 23



We can release the latch on **A** as soon as we acquire the latch for **C**.

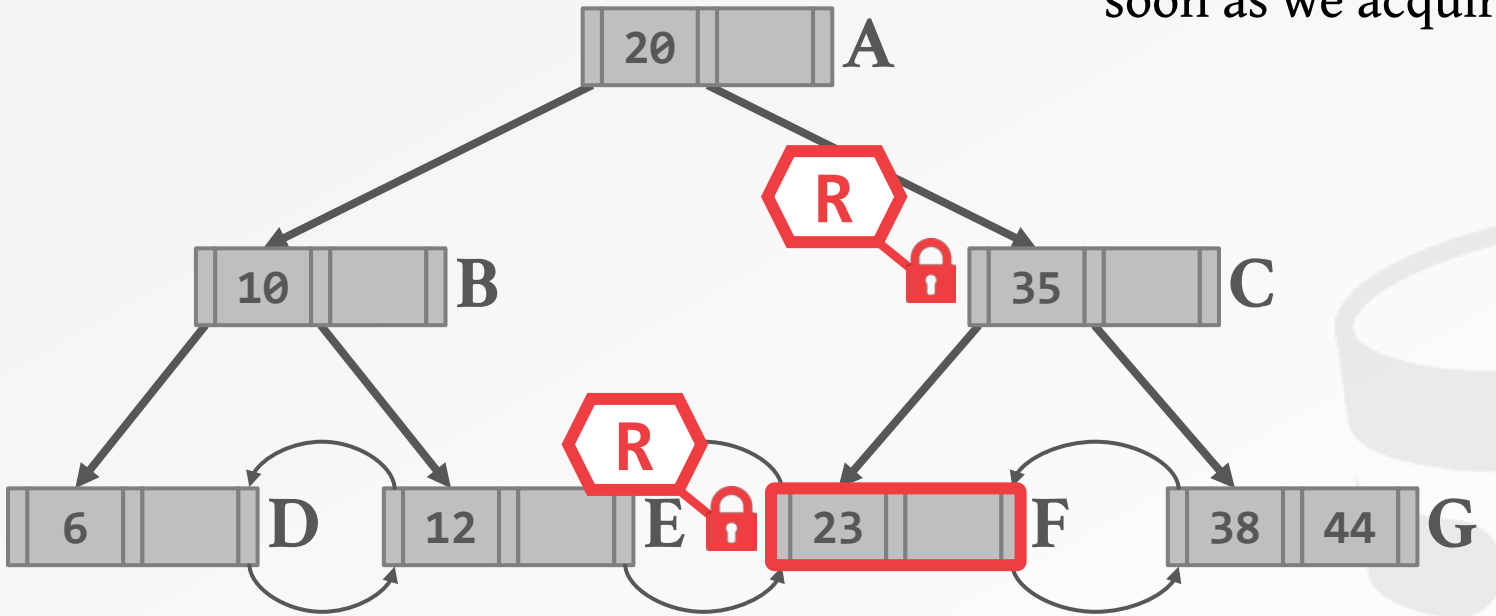
EXAMPLE #1: SEARCH 23

We can release the latch on **A** as soon as we acquire the latch for **C**.



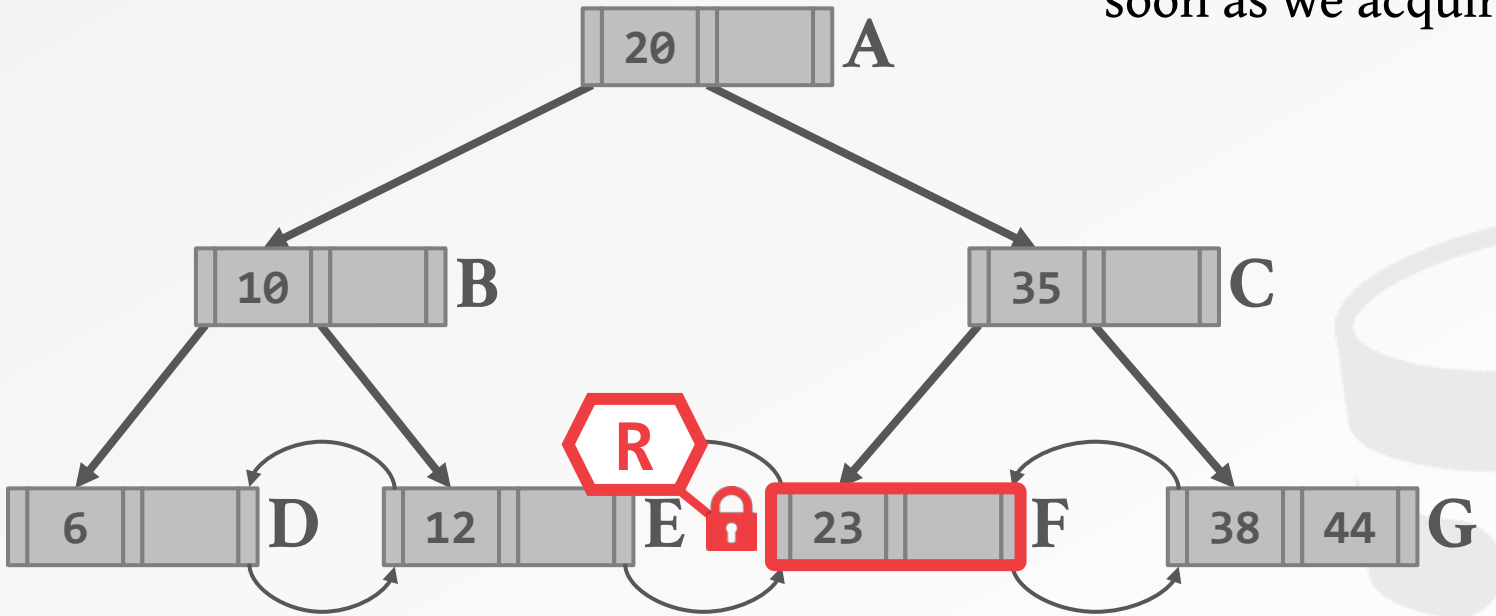
EXAMPLE #1: SEARCH 23

We can release the latch on **A** as soon as we acquire the latch for **C**.

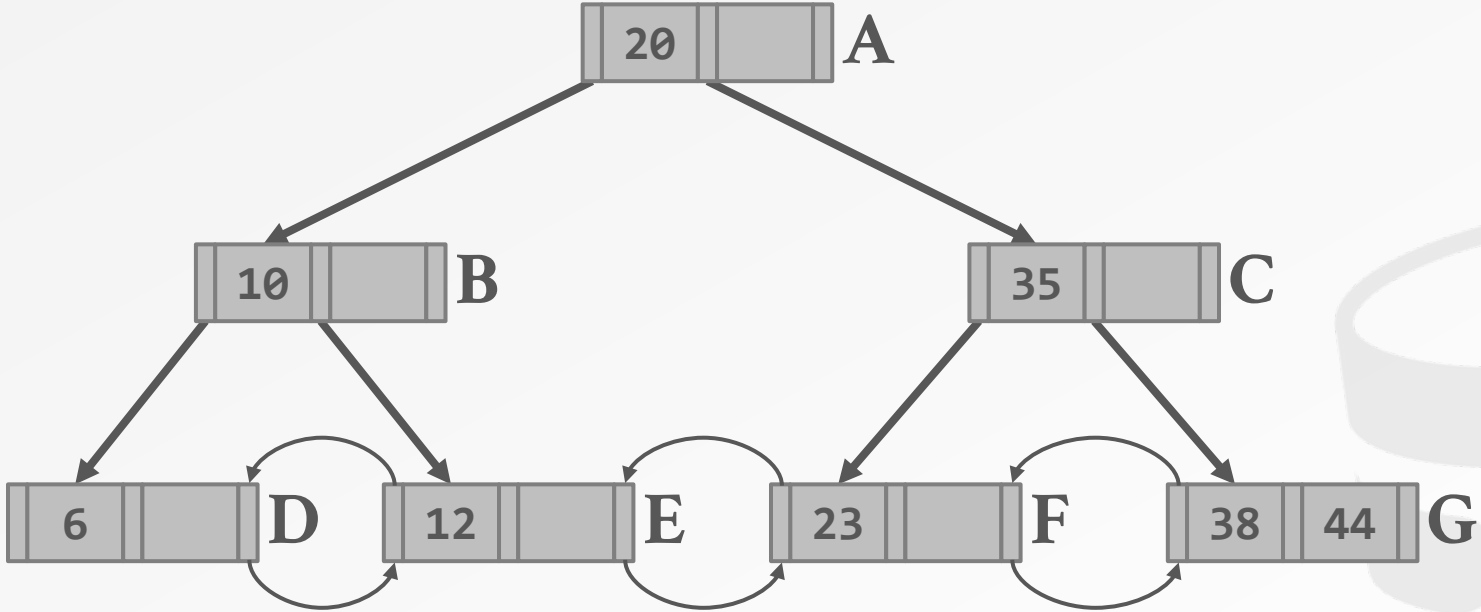


EXAMPLE #1: SEARCH 23

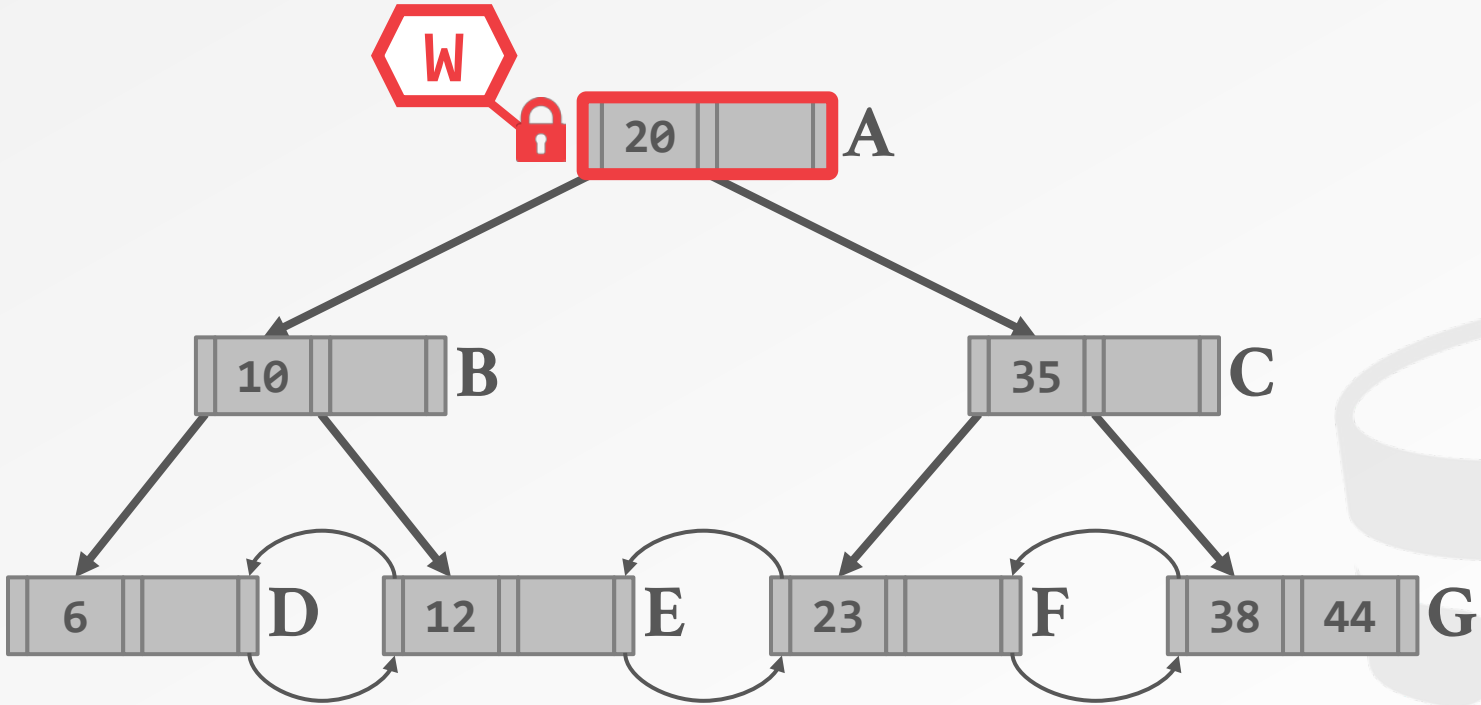
We can release the latch on **A** as soon as we acquire the latch for **C**.



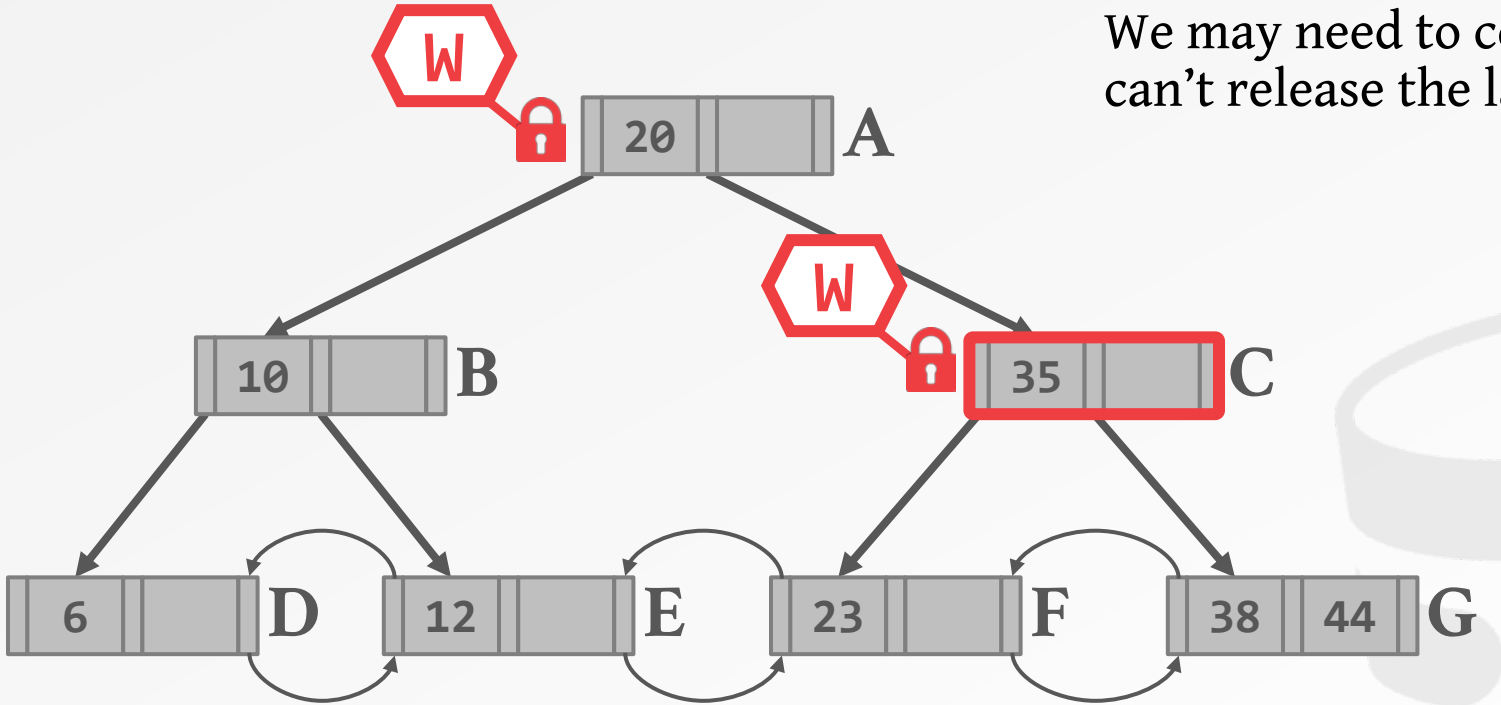
EXAMPLE #2: DELETE 44



EXAMPLE #2: DELETE 44

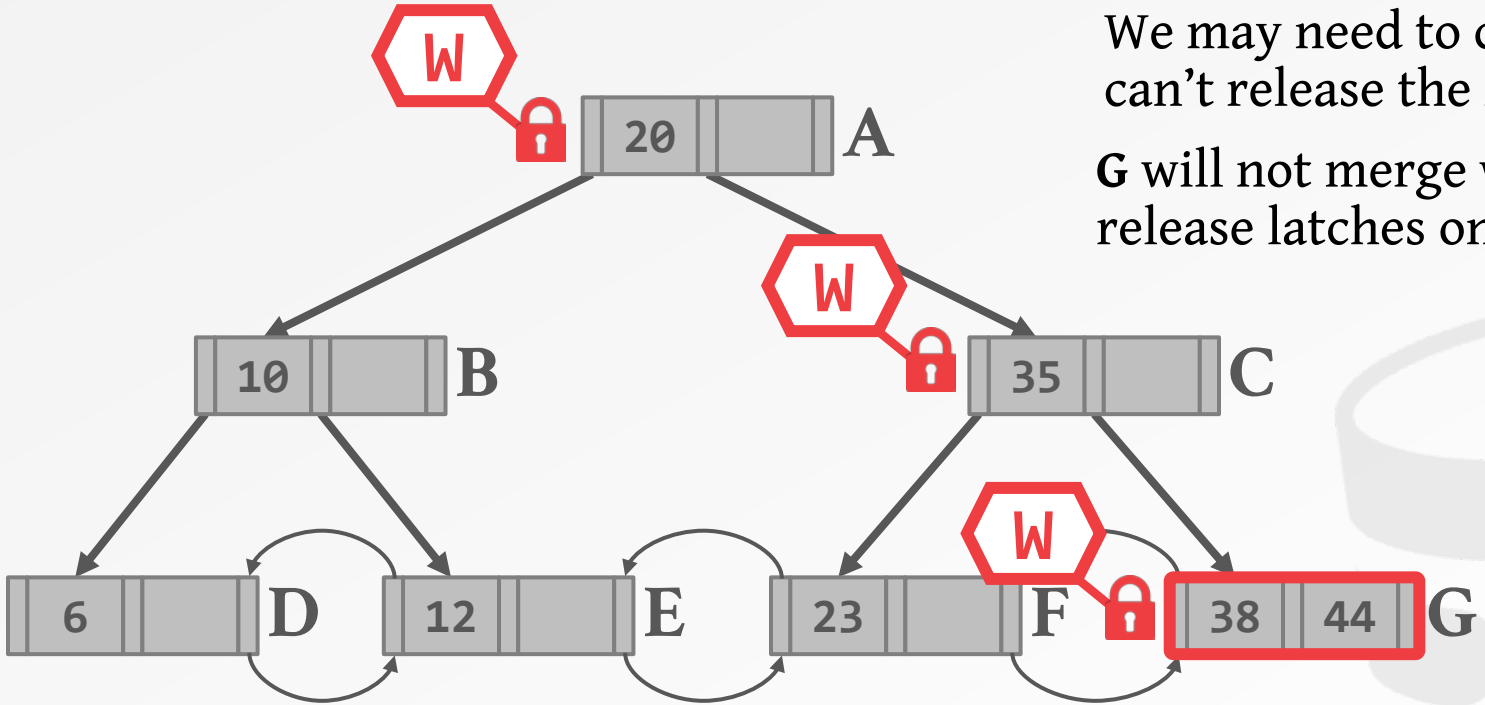


EXAMPLE #2: DELETE 44



We may need to coalesce **C**, so we can't release the latch on **A**.

EXAMPLE #2: DELETE 44



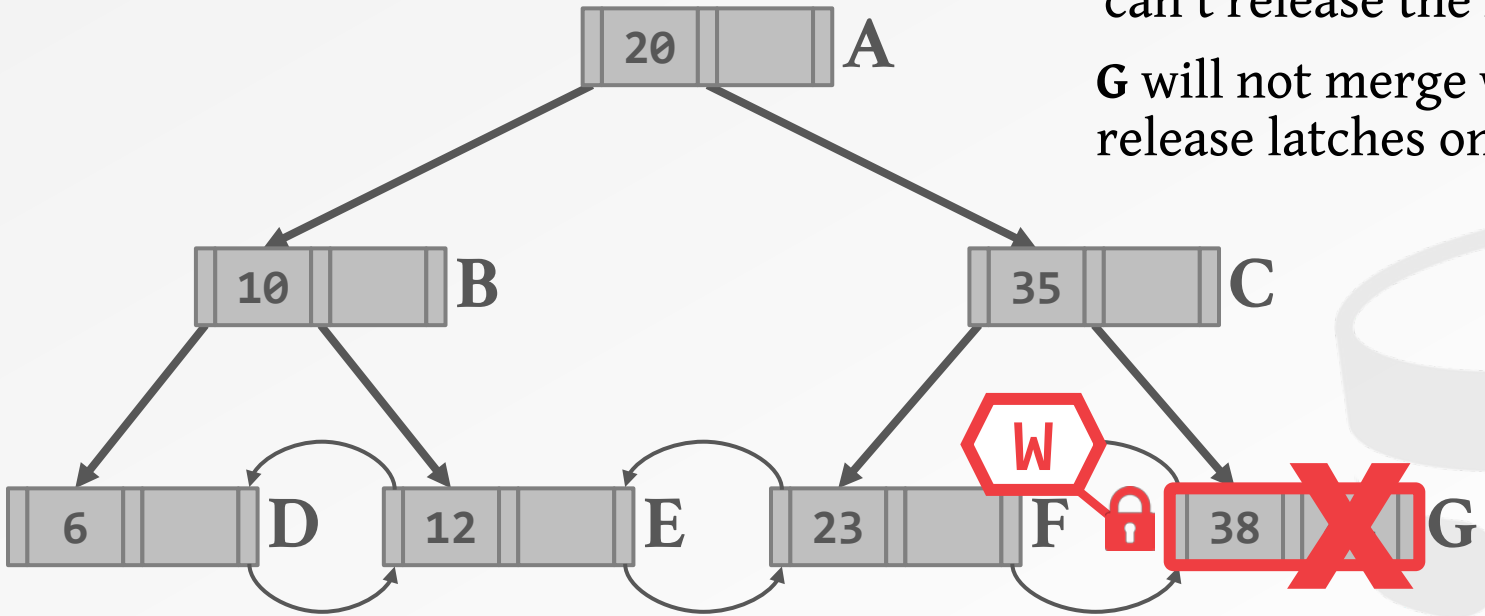
We may need to coalesce **C**, so we can't release the latch on **A**.

G will not merge with **F**, so we can release latches on **A** and **C**.

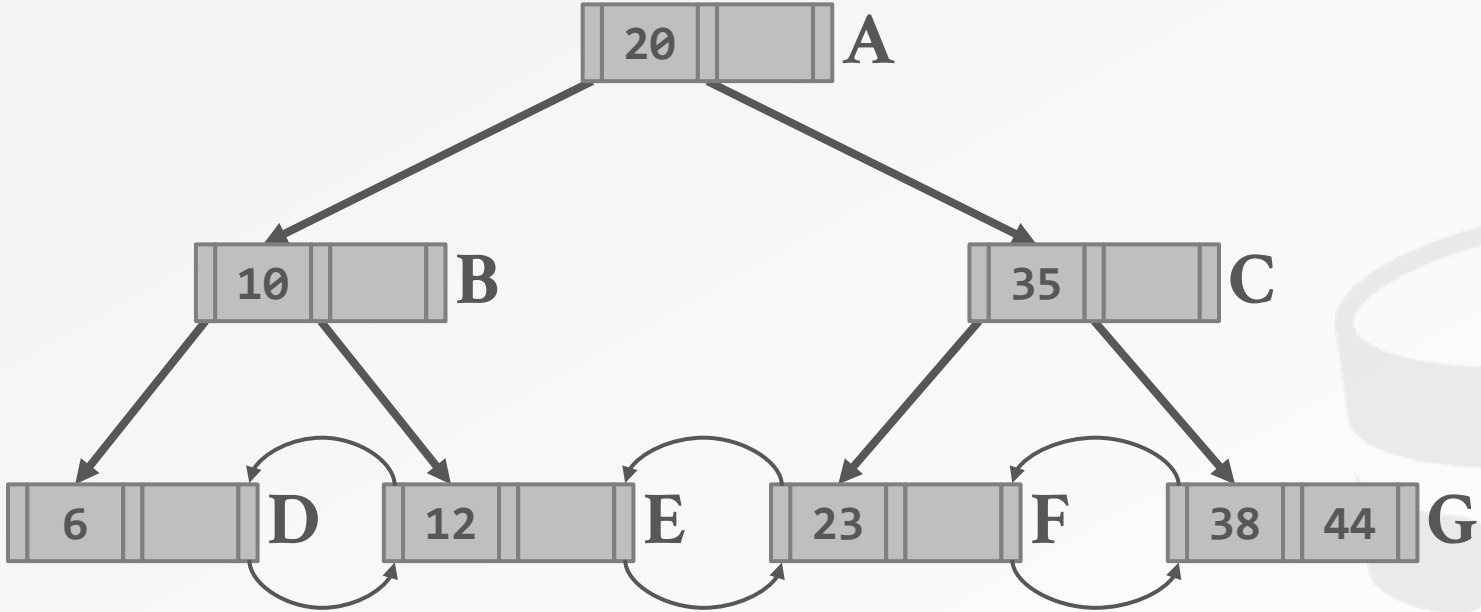
EXAMPLE #2: DELETE 44

We may need to coalesce **C**, so we can't release the latch on **A**.

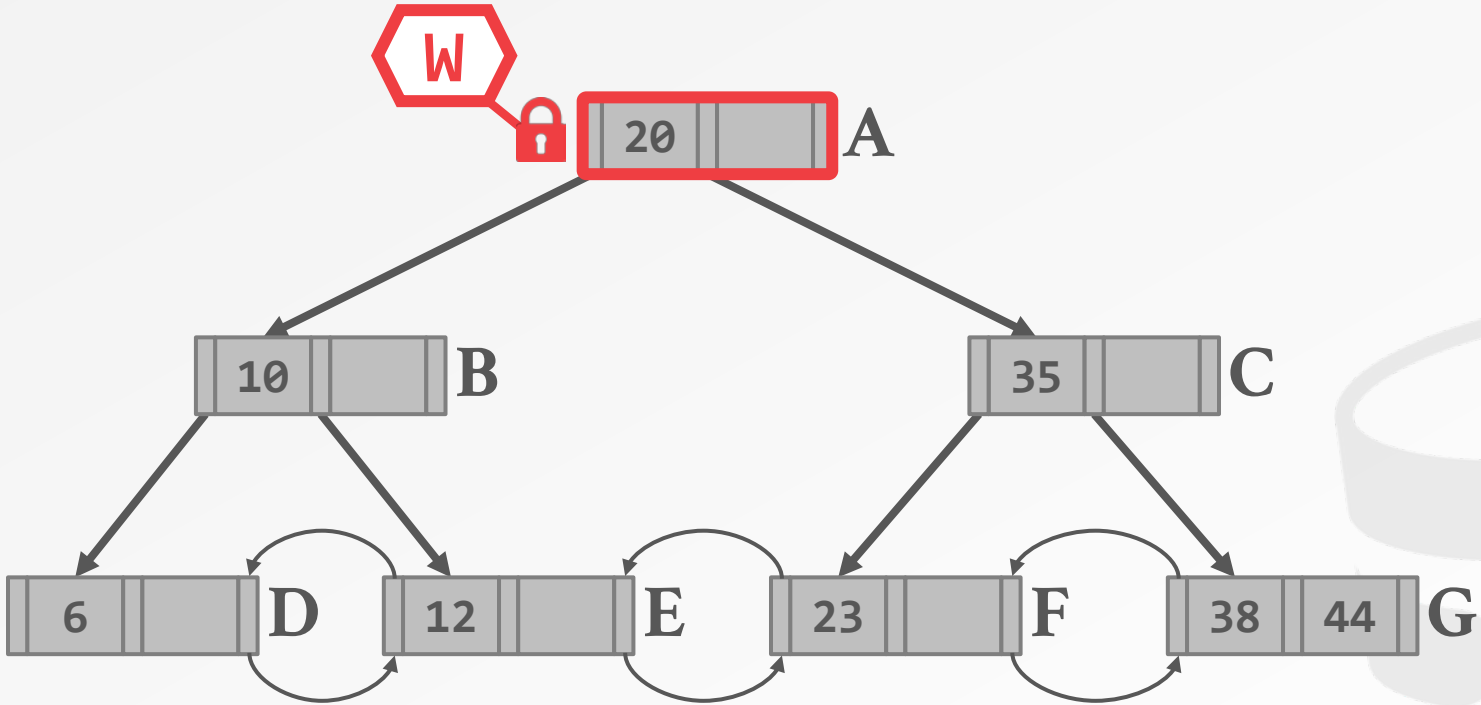
G will not merge with **F**, so we can release latches on **A** and **C**.



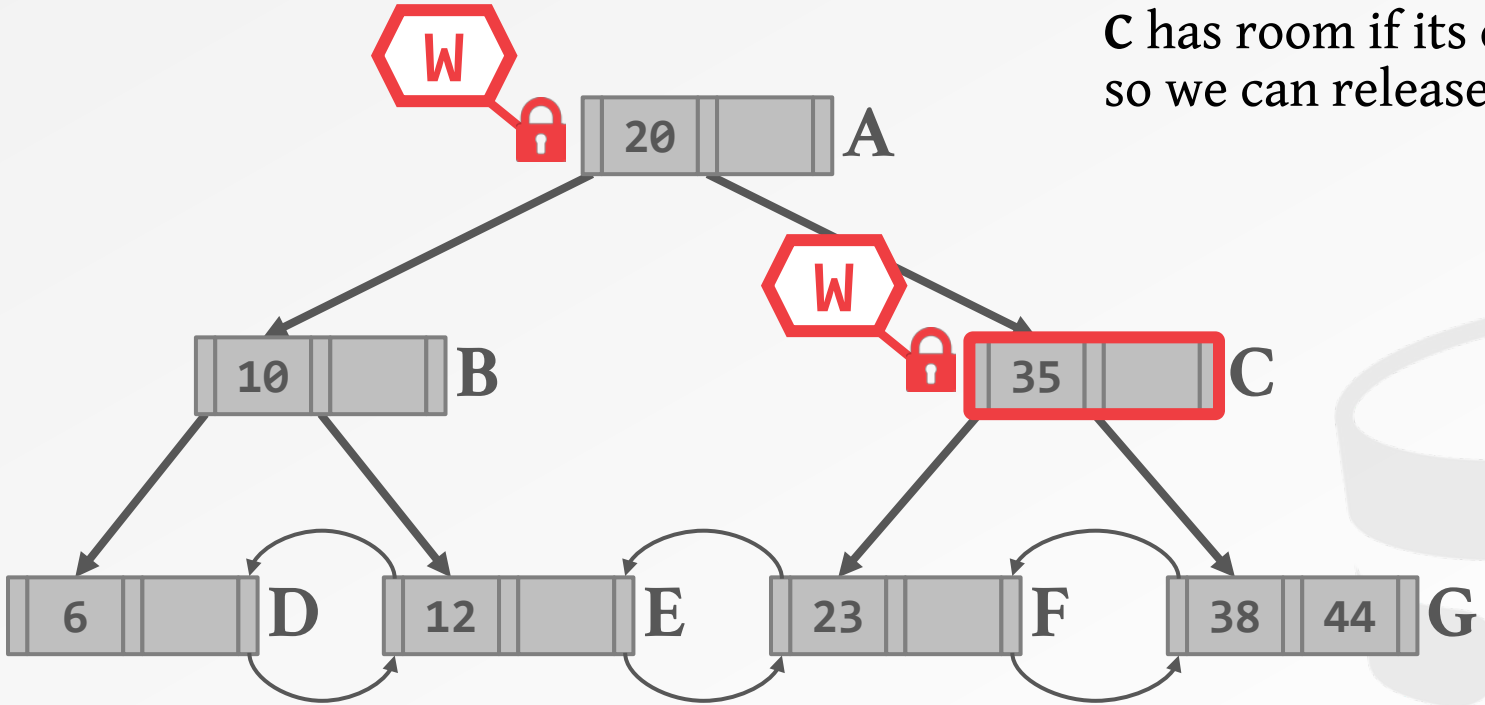
EXAMPLE #3: INSERT 40



EXAMPLE #3: INSERT 40



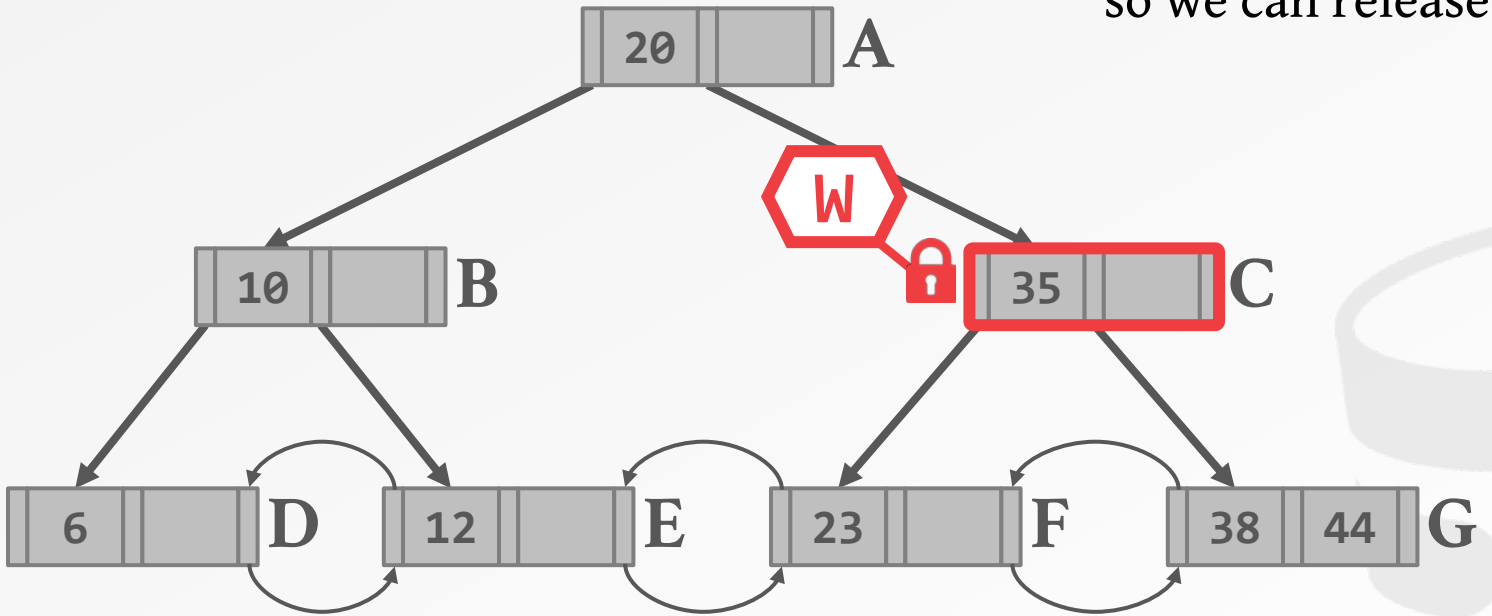
EXAMPLE #3: INSERT 40



C has room if its child has to split, so we can release the latch on **A**.

EXAMPLE #3: INSERT 40

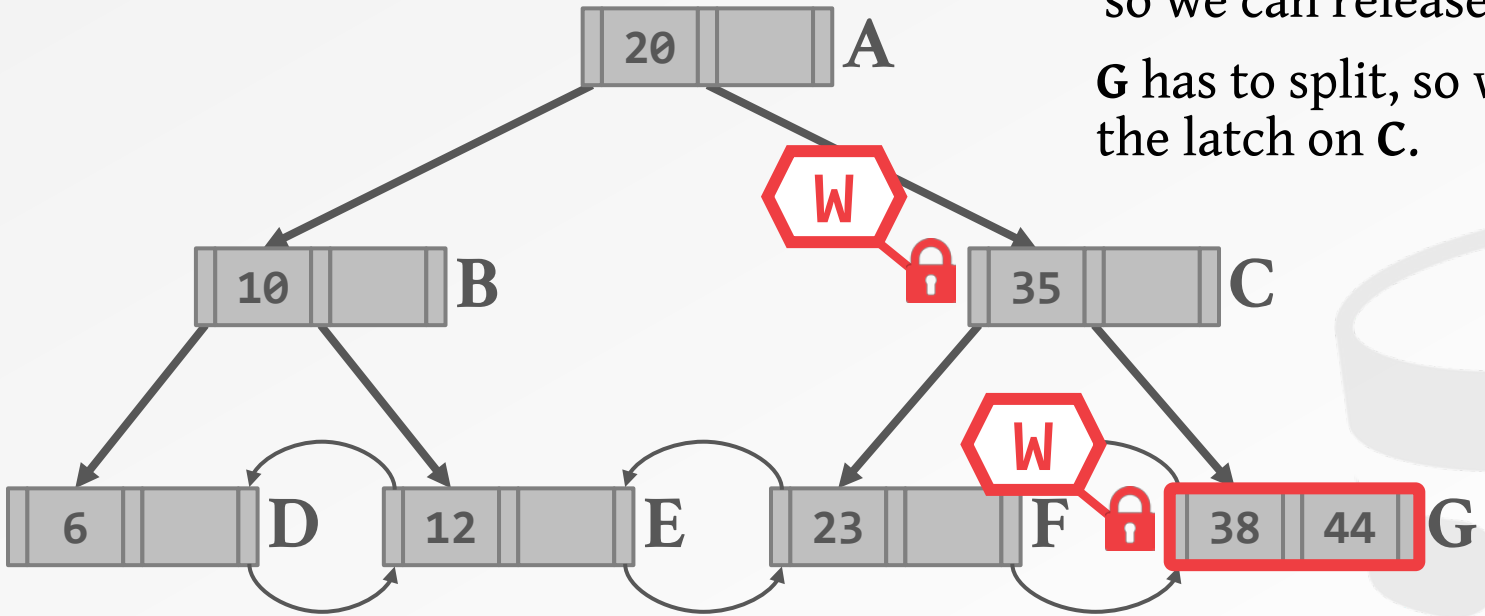
C has room if its child has to split, so we can release the latch on **A**.



EXAMPLE #3: INSERT 40

C has room if its child has to split, so we can release the latch on **A**.

G has to split, so we can't release the latch on **C**.



EXAMPLE #3: INSERT 40

C has room if its child has to split, so we can release the latch on **A**.

G has to split, so we can't release the latch on **C**.