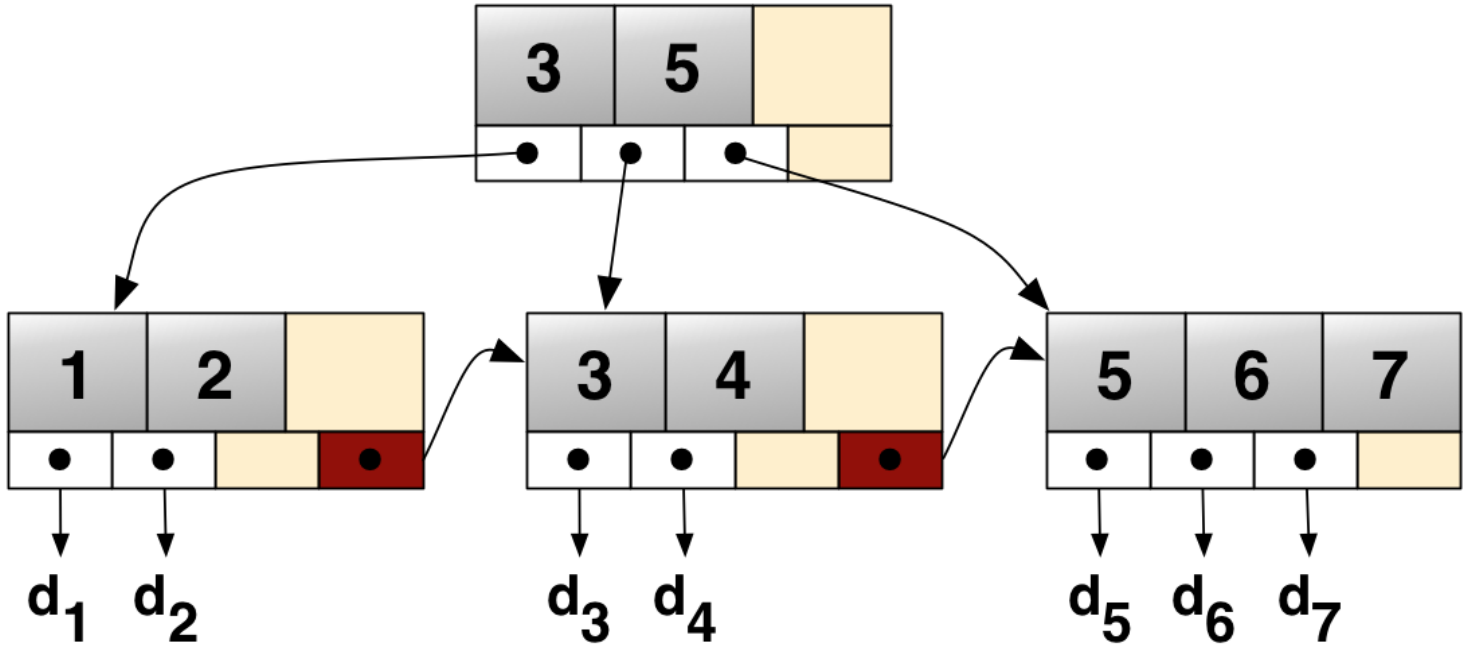
Copy on Write (COW) and B (Read B+) Trees

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Sandip Chakraborty
sandipc@cse.iitkgp.ac.in

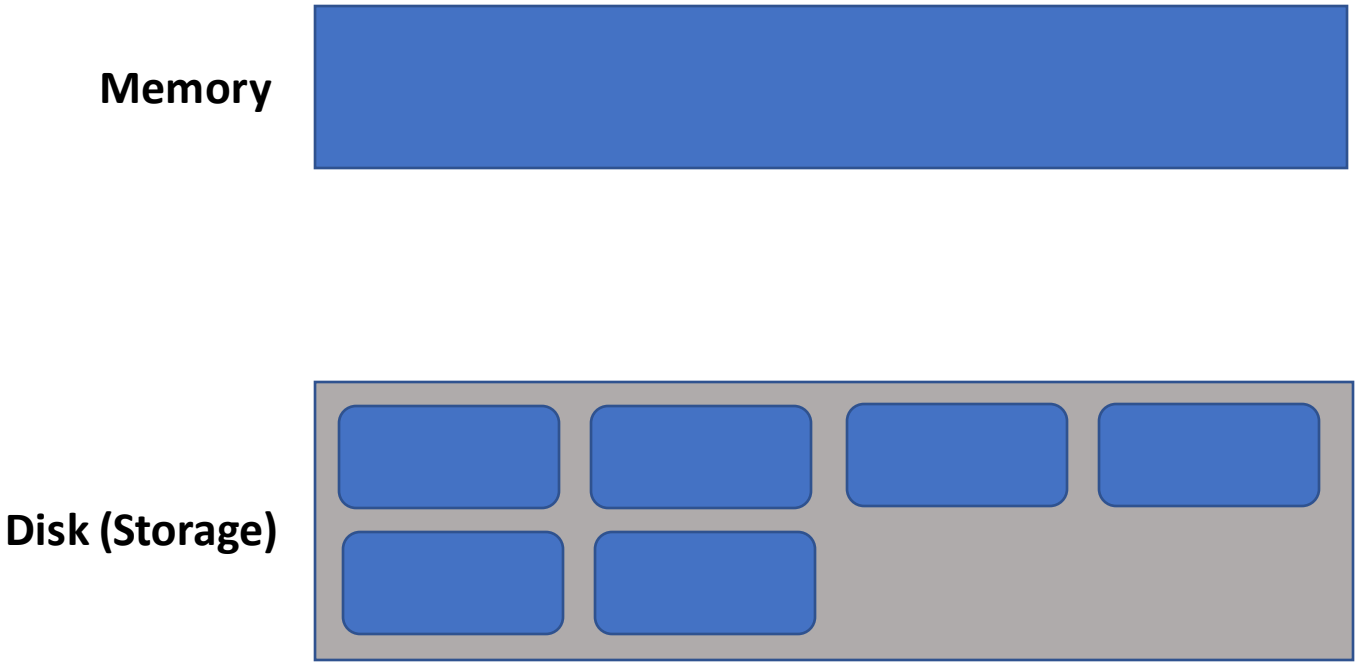


B Trees in File Systems

- B Trees are widely used for file system representation (WAFL, ZFS, BTRFS)
  - Logarithmic time key search, insert and remove
  - Well represents sparse files
- The File System as a large tree made up of fixed size pages
- **Shadowing:** Technique to support atomic updates over persistent data structures
  - Implement snapshots, crash recovery, write-batching, RAID

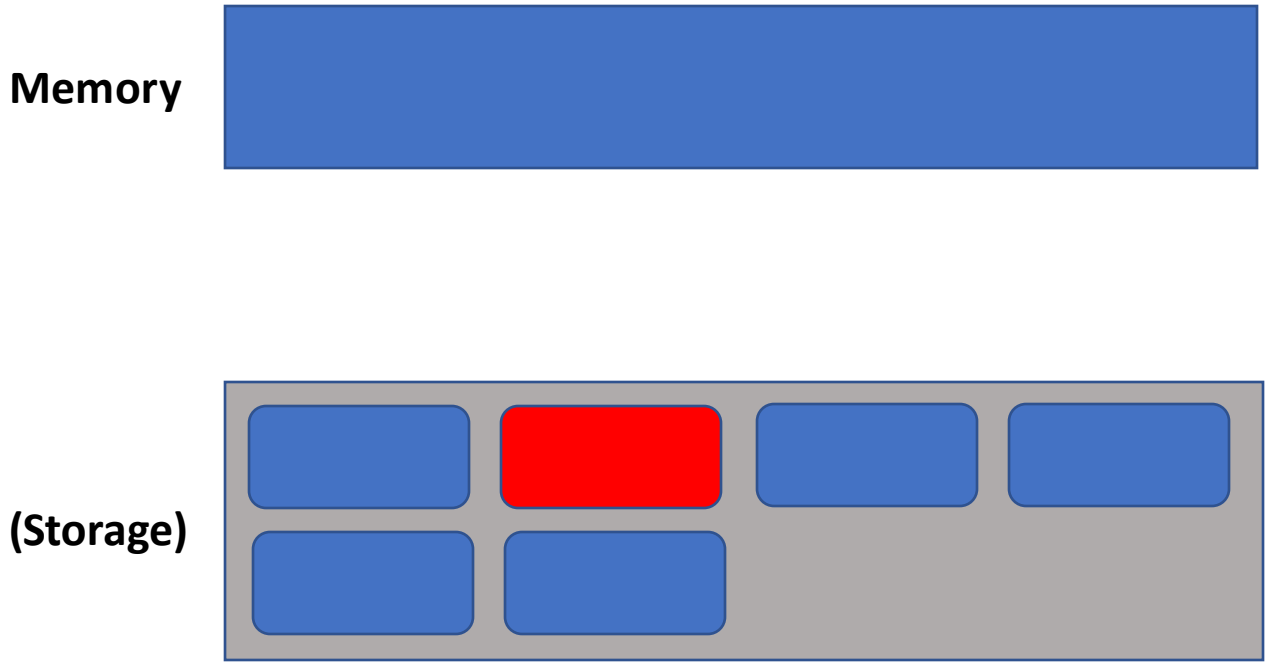
Shadowing

- To update an on-disk page (the page is in the disk, not available in the memory)
  - Read the entire page in the memory
  - Modify the page
  - Write in an alternate location



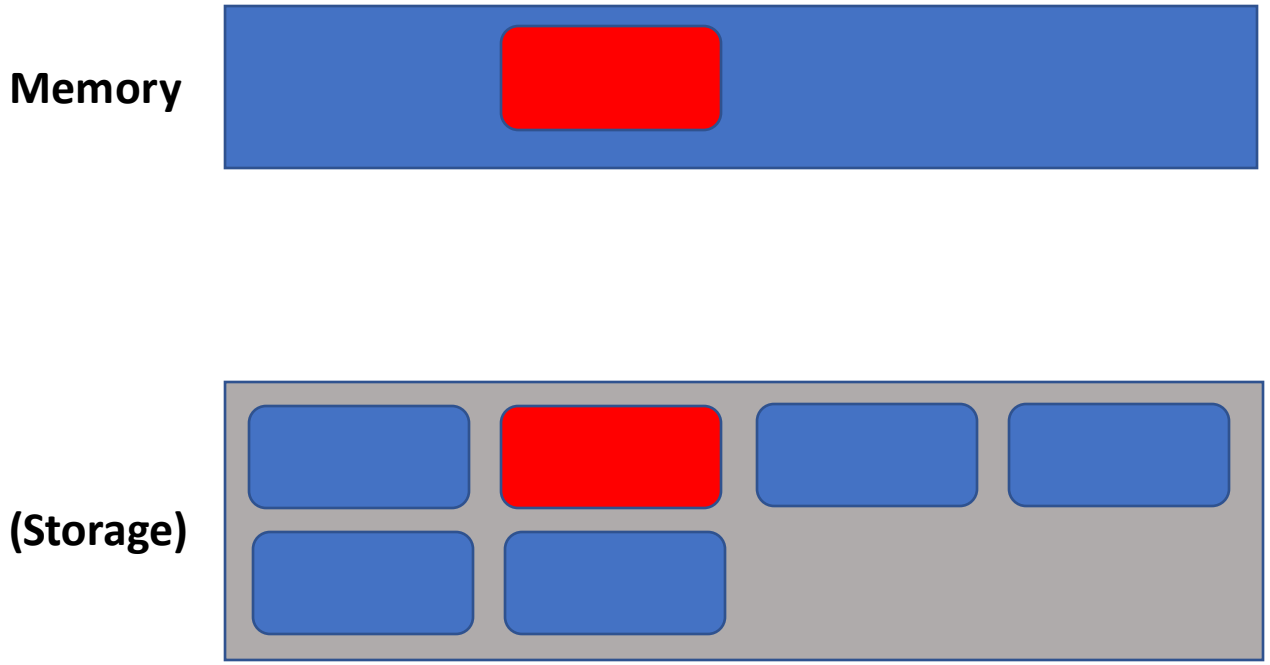
Shadowing

- To update an on-disk page (the page is in the disk, not available in the memory)
  - Read the entire page in the memory
  - Modify the page
  - Write in an alternate location



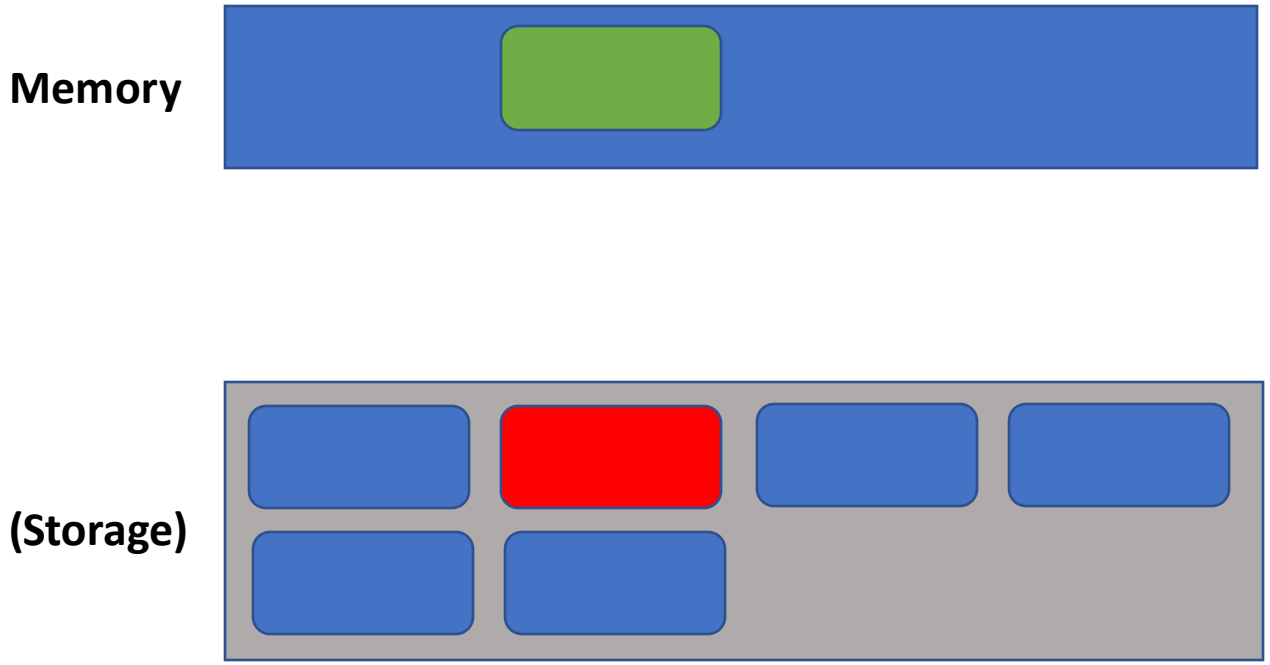
Shadowing

- To update an on-disk page (the page is in the disk, not available in the memory)
  - Read the entire page in the memory
  - Modify the page
  - Write in an alternate location



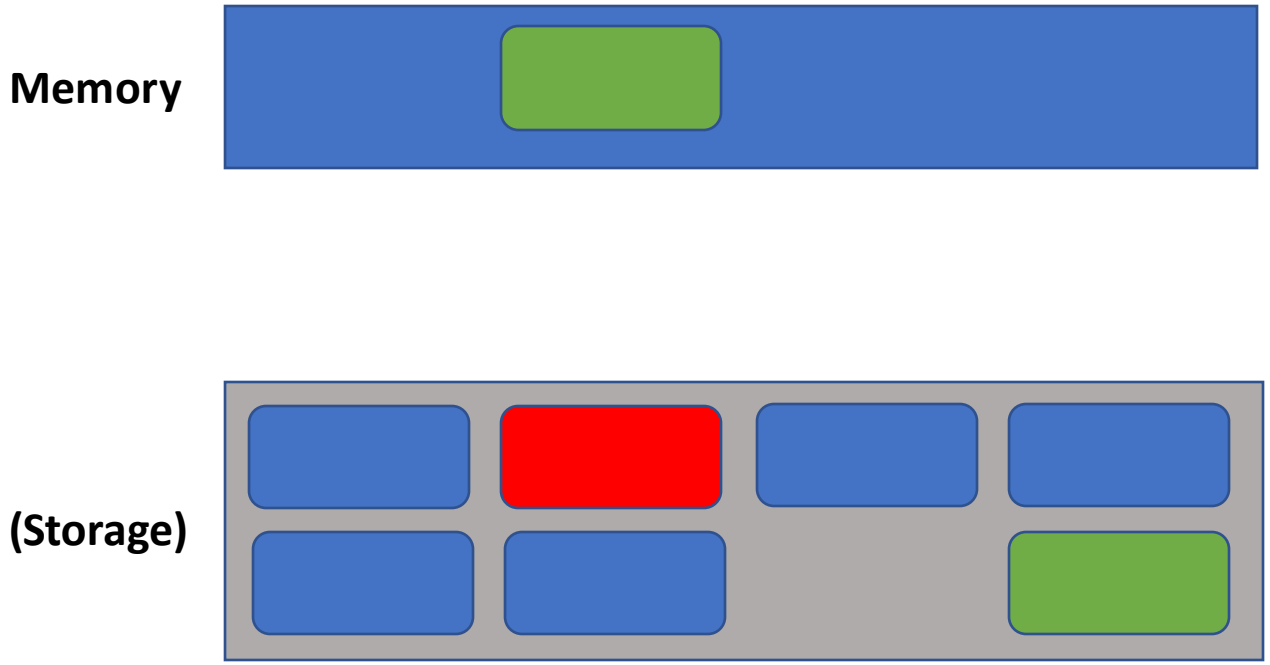
Shadowing

- To update an on-disk page (the page is in the disk, not available in the memory)
  - Read the entire page in the memory
  - Modify the page
  - Write in an alternate location



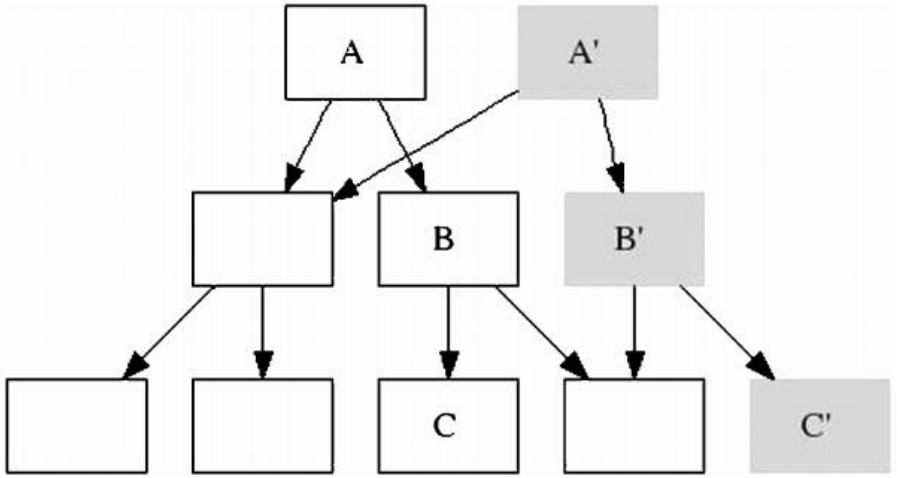
Shadowing

- To update an on-disk page (the page is in the disk, not available in the memory)
  - Read the entire page in the memory
  - Modify the page
  - Write in an alternate location



Shadowing

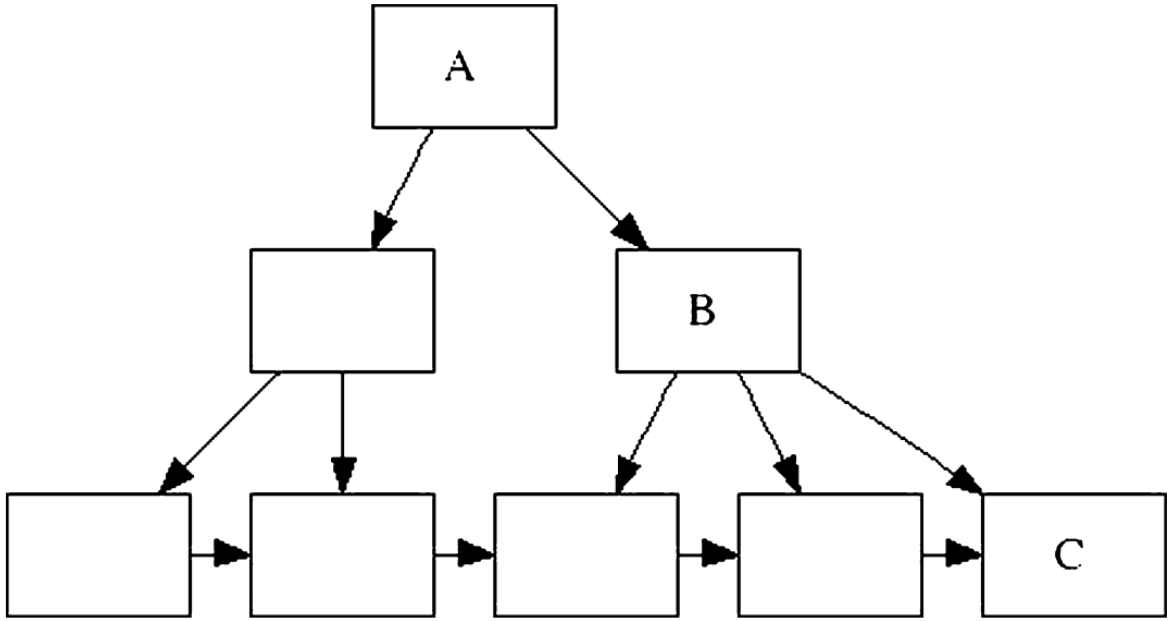
- To update an on-disk page (the page is in the disk, not available in the memory)
  - Read the entire page in the memory
  - Modify the page
  - Write in an alternate location



- When a page is shadowed, its location on the disk changes
  - Update (and shadow) the immediate ancestor of the page with the new address
  - Propagates up to the file system root

Shadowing over B Trees

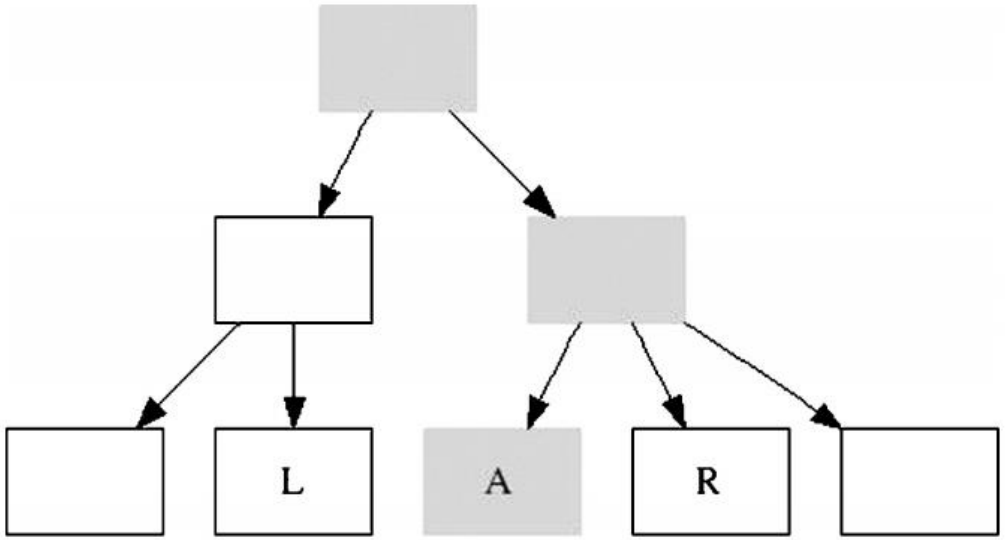
- **Leaf Chaining**
  - Used in B-trees for tree rebalancing and range lookup
  - COW needs entire B Tree to be modified



Shadowing over B Trees

- **Concurrency**
  - Only leaf changes in a regular B Tree updates
  - COW needs locks up to the root

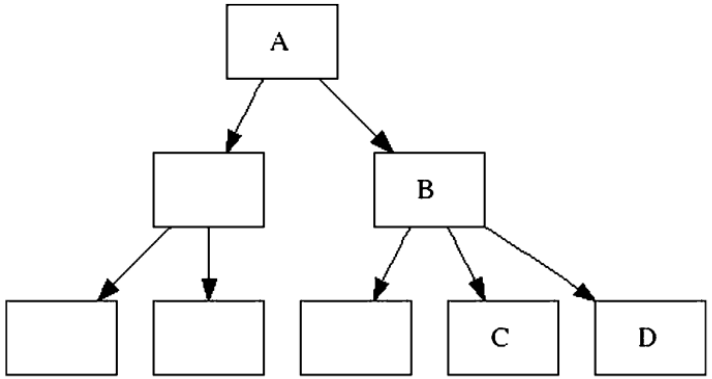
- **Modifying a Single Path**
  - Regular B Trees shuffle keys between neighbors for rebalancing after a "remove key"
  - COW makes this expensive



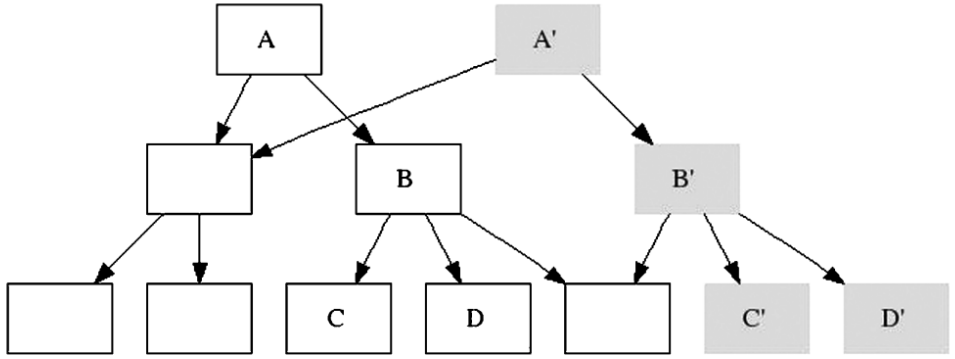
Checkpoint and Recoverability

- Shadowing file systems ensure recoverability by taking periodic checkpoints, and logging commands in between

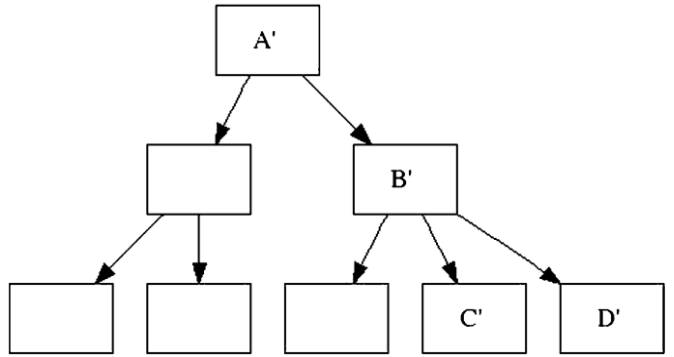
- Checkpoint includes the entire file system tree
  - Once a checkpoint is taken on the disk, the previous checkpoint can be removed
  - In case of a crash, go back to the last saved checkpoint and replay the log



(a) initial file system tree



(b) modifications



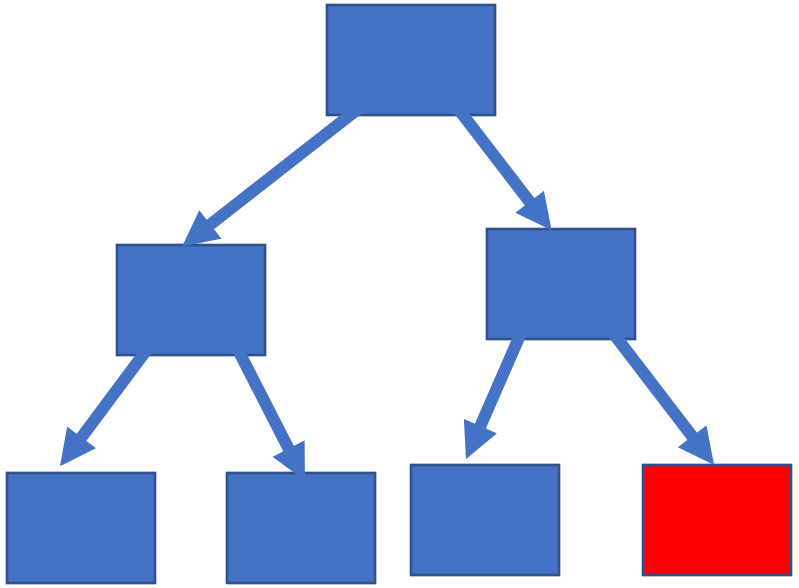
(c) new checkpoint

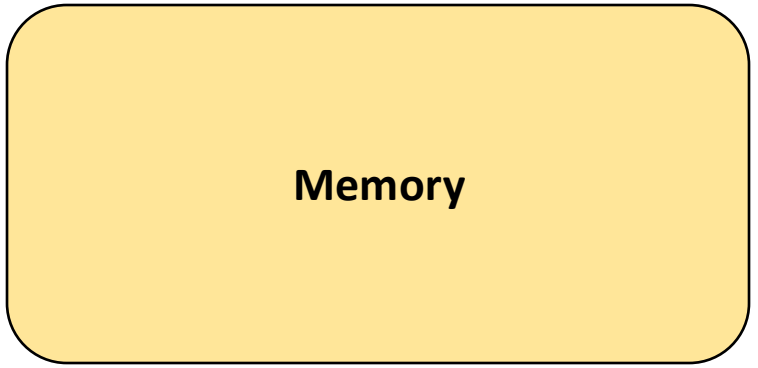
COW with Checkpoints

- Checkpoint makes COW efficient – modifications can be batched and written sequentially in the disk
- Command logging combines multiple operations on the file systems on a single log entry
- When a page belonging to a checkpoint is first shadowed, a cached copy of it is created and held in memory
  - All modifications to the page can be done on the cached shadow copy
  - The dirty pages can be held in the memory until the next checkpoint (assuming that there is sufficient memory)
  - For swap out of pages, write the pages to the shadow location of the disk

Shadowing and Batch Update

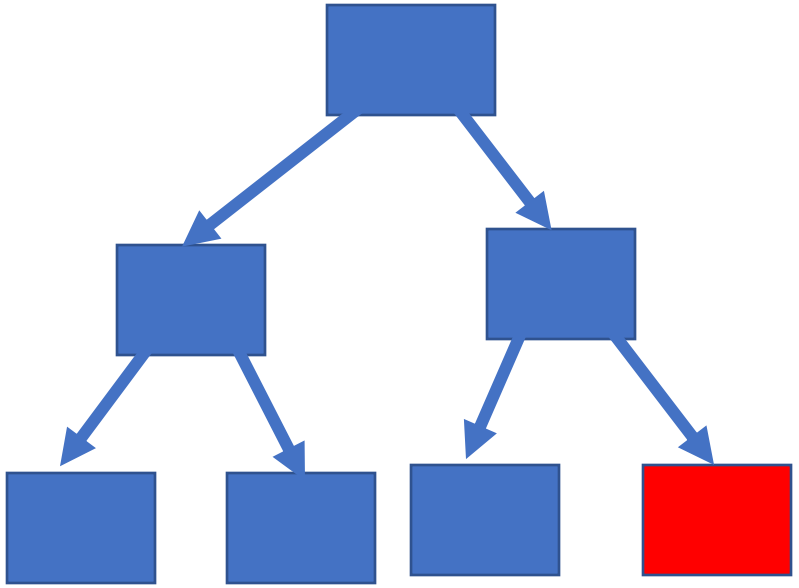
- A page is loaded in the memory and all the updates are performed on it

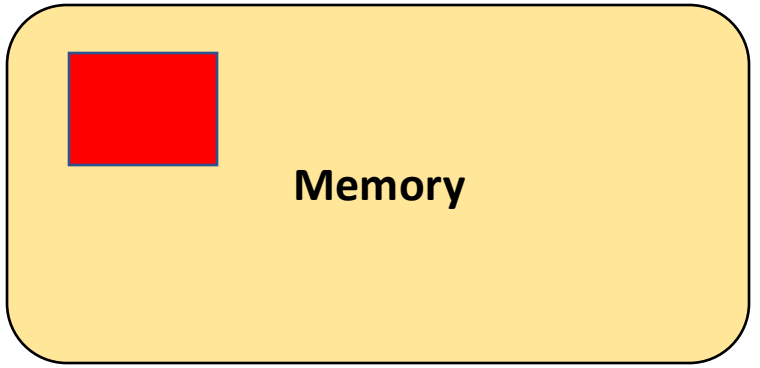




Shadowing and Batch Update

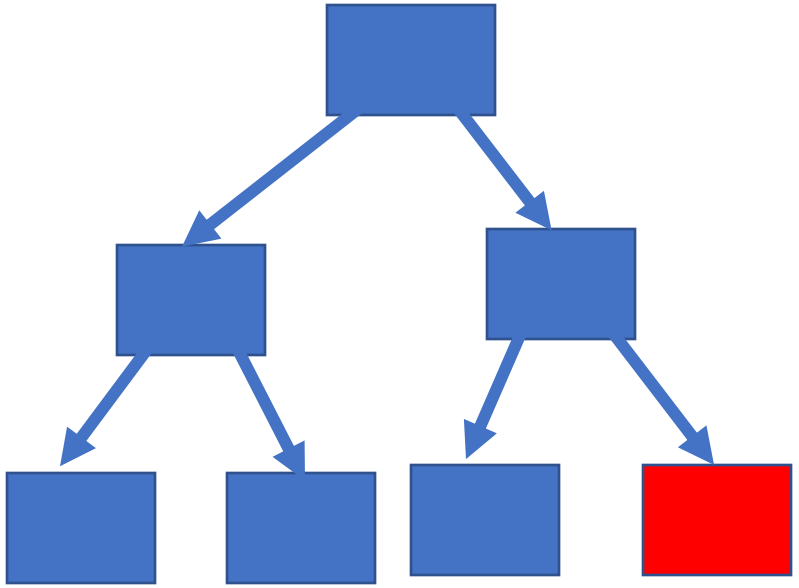
- A page is loaded in the memory and all the updates are performed on it

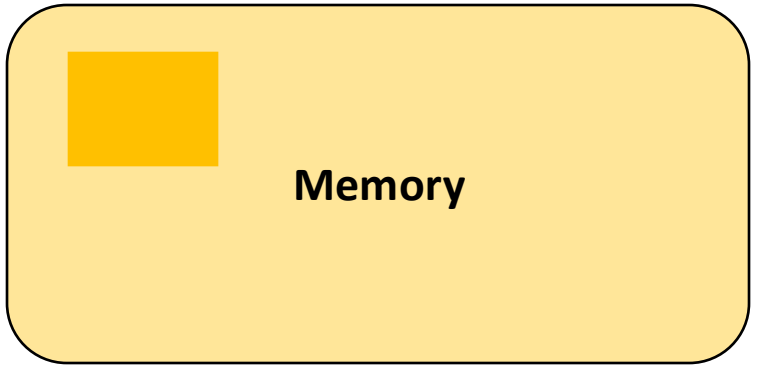




Shadowing and Batch Update

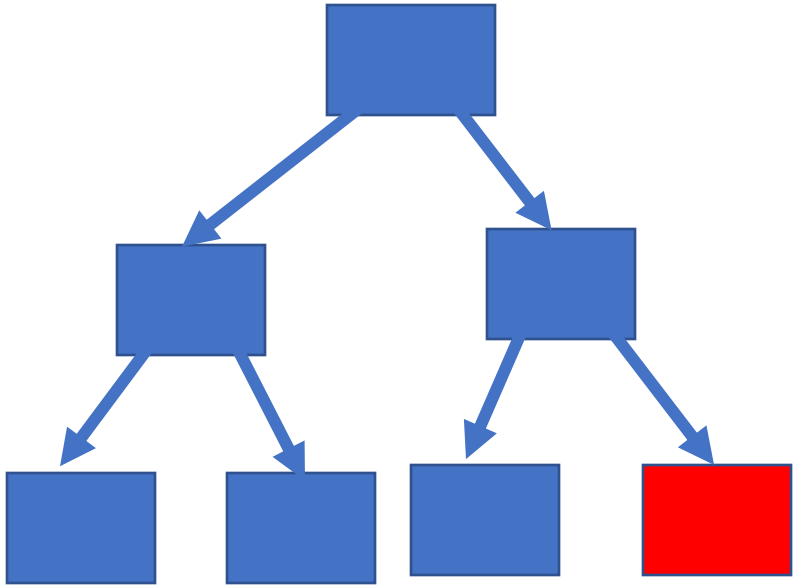
- A page is loaded in the memory and all the updates are performed on it

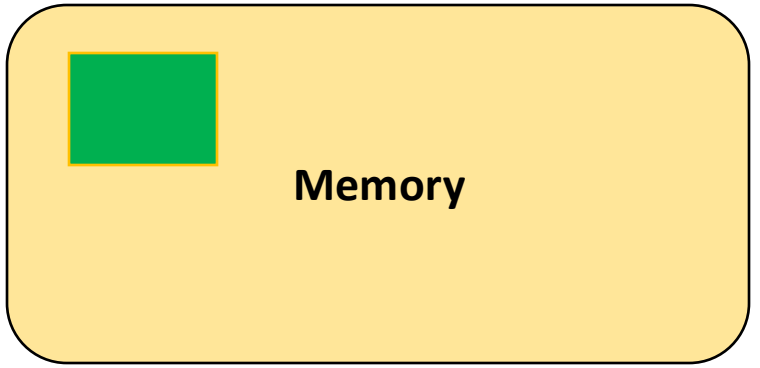




Shadowing and Batch Update

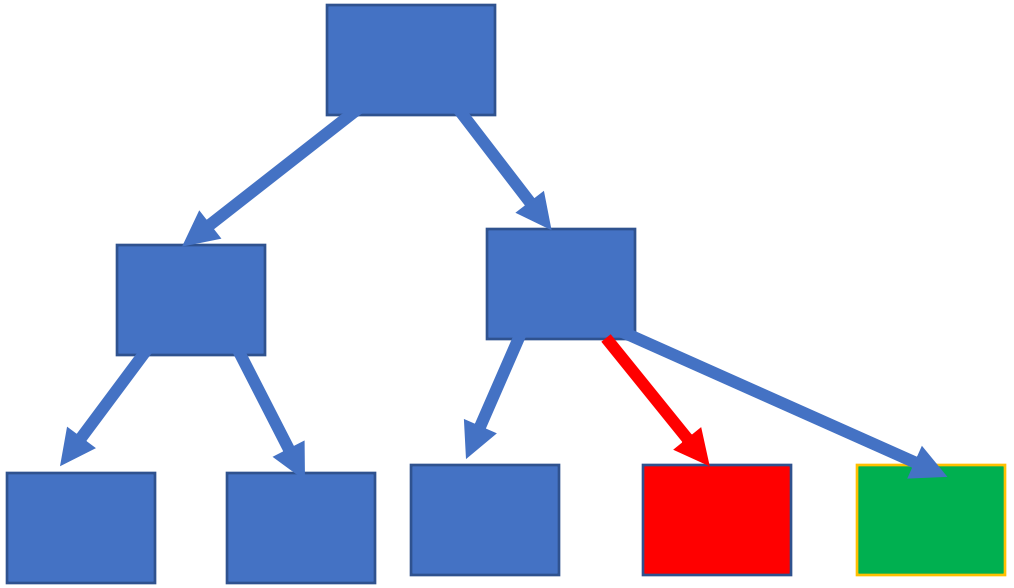
- A page is loaded in the memory and all the updates are performed on it

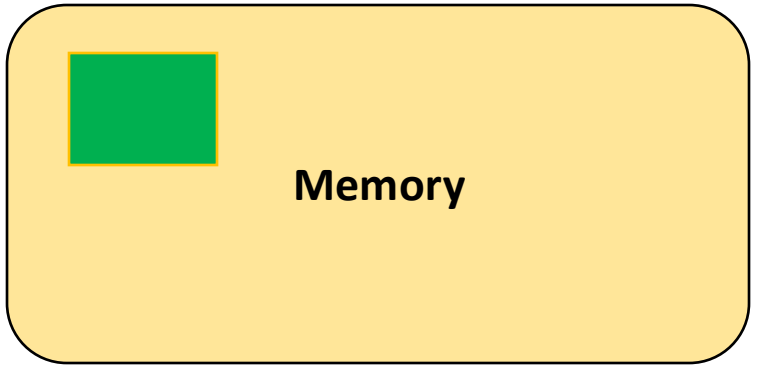




Shadowing and Batch Update

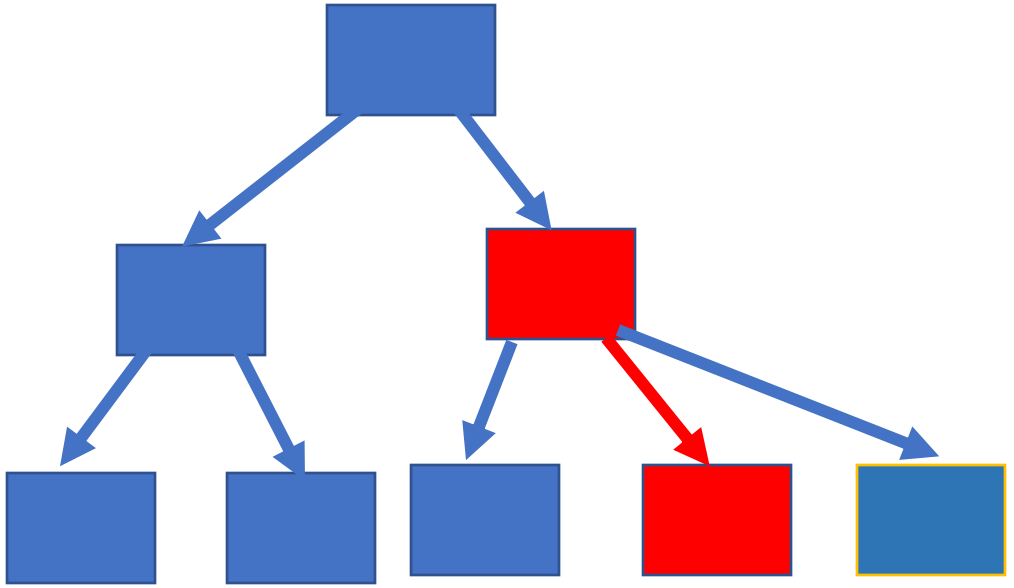
- A page is loaded in the memory and all the updates are performed on it
- During the checkpoint, load its parent page for the update of the address.

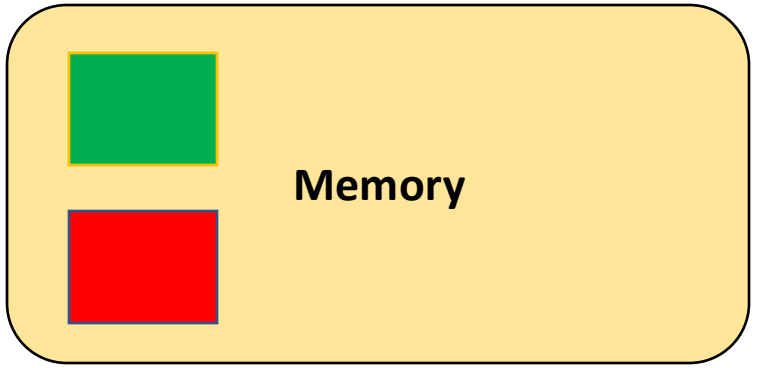




Shadowing and Batch Update

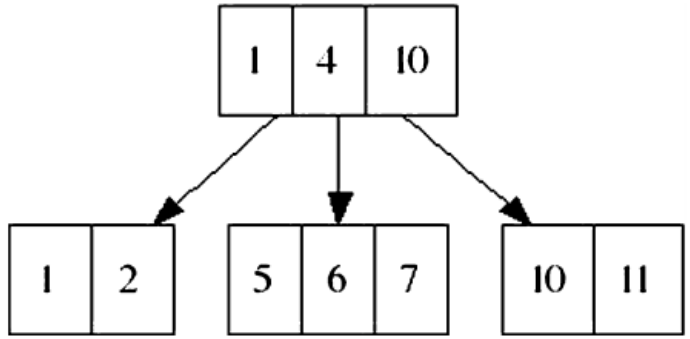
- A page is loaded in the memory and all the updates are performed on it
- During the checkpoint, load its parent page for the update of the address.





B (Typically B+) Tree Insertion

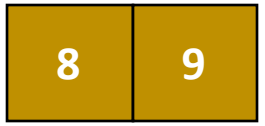
- Normally between *b* and *2b-1* entries per node
- During the insertion, if the entries to a node becomes more than *2b-1,* then the node is split into two
- Assume *b=2*

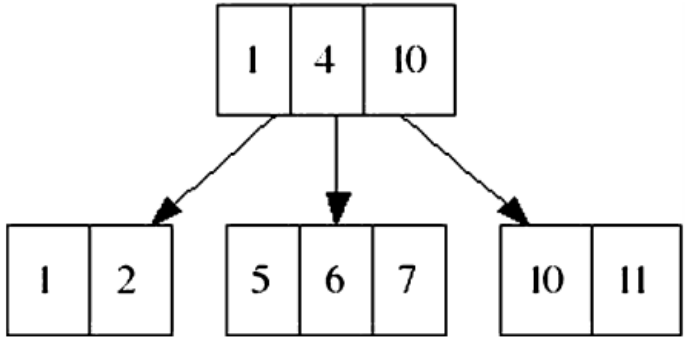


8

B (Typically B+) Tree Insertion

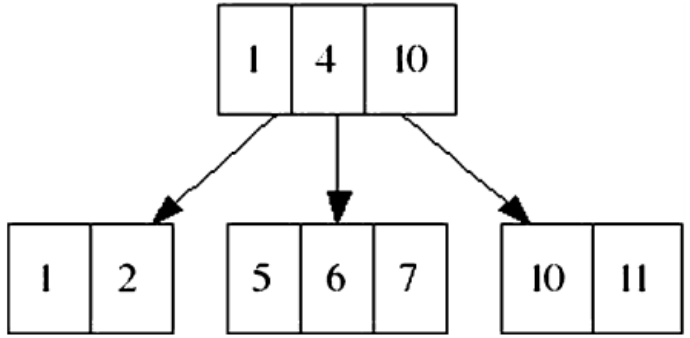
- Normally between *b* and *2b-1* entries per node
- During the insertion, if the entries to a node becomes more than *2b-1,* then the node is split into two
- Assume *b=2*





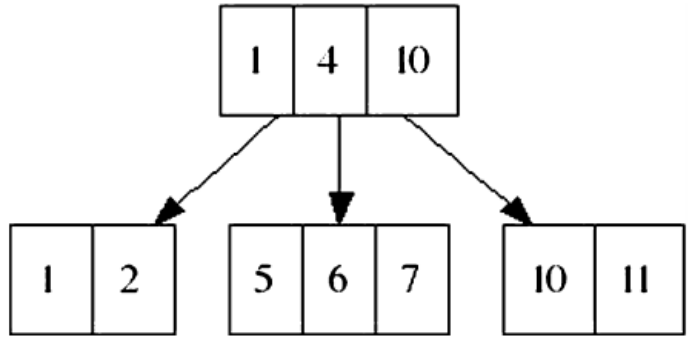
B (Typically B+) Tree Insertion

- Normally between *b* and *2b-1* entries per node
- During the insertion, if the entries to a node becomes more than *2b-1,* then the node is split into two
- Assume *b=2*



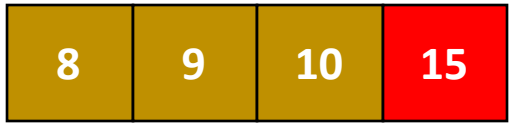
B (Typically B+) Tree Insertion

- Normally between *b* and *2b-1* entries per node



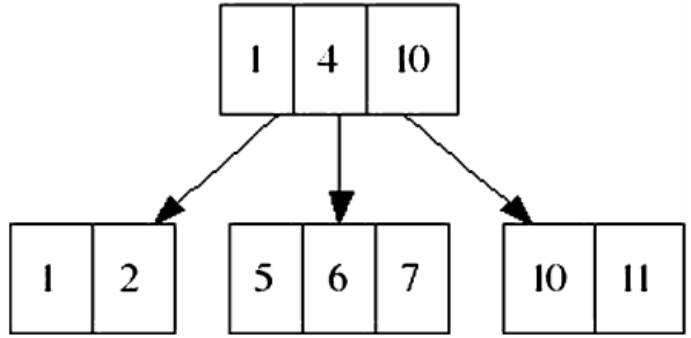
- During the insertion, if the entries to a node becomes more than *2b-1,* then the node is split into two

- Assume *b=2*



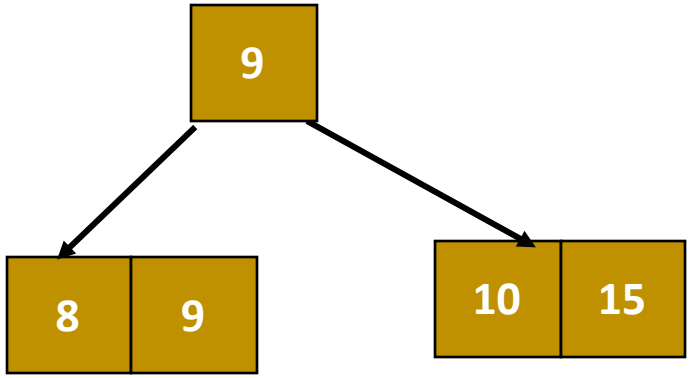
B Tree Insertion

- Normally between *b* and *2b-1* entries per node



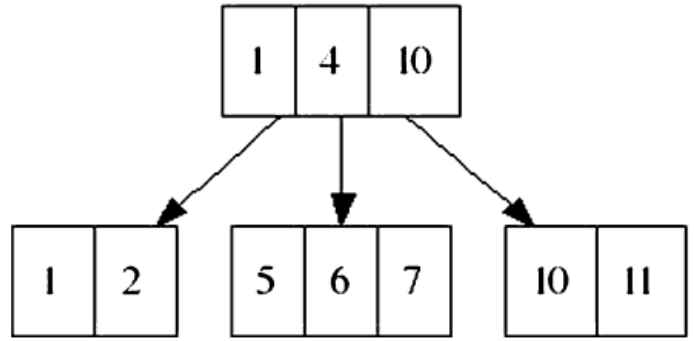
- During the insertion, if the entries to a node becomes more than *2b-1,* then the node is split into two

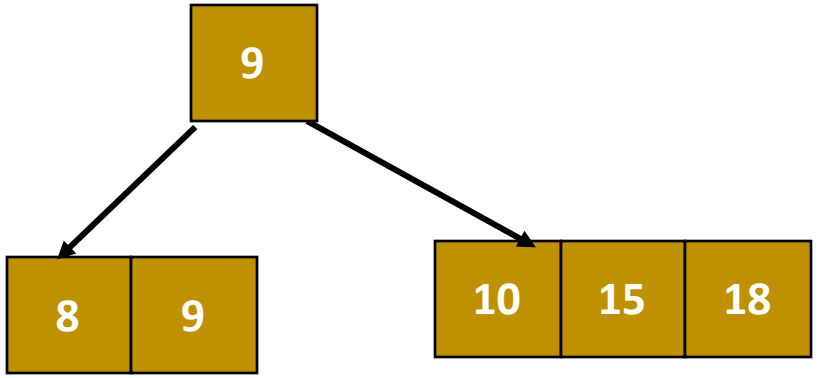
- Assume *b=2*



B Tree Insertion

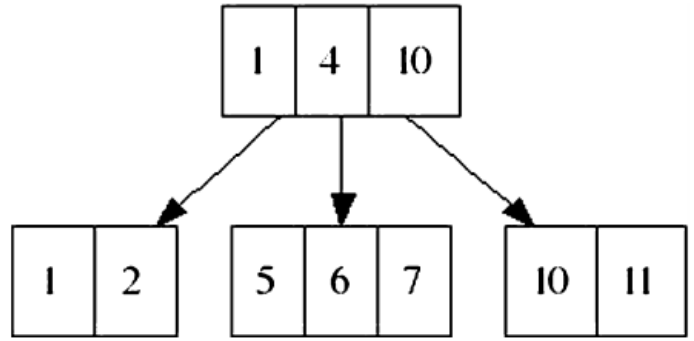
- Normally between *b* and *2b-1* entries per node
- During the insertion, if the entries to a node becomes more than *2b-1,* then the node is split into two
- Assume *b=2*





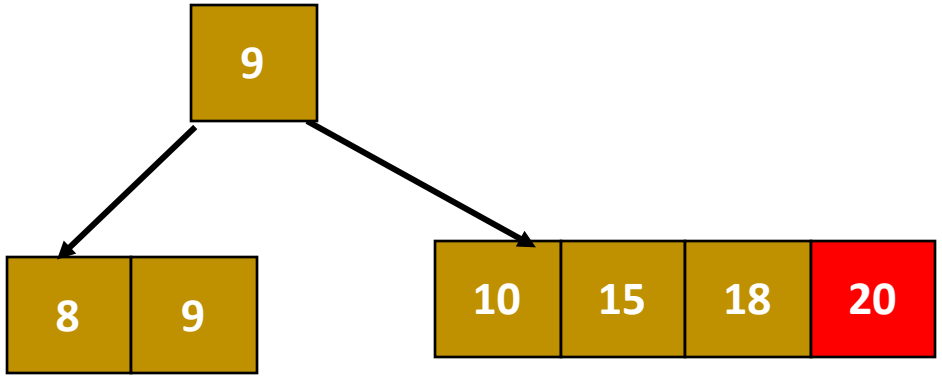
B Tree Insertion

- Normally between *b* and *2b-1* entries per node



- During the insertion, if the entries to a node becomes more than *2b-1,* then the node is split into two

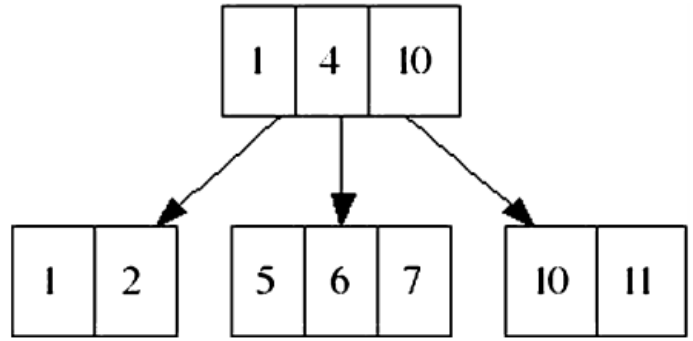
- Assume *b=2*



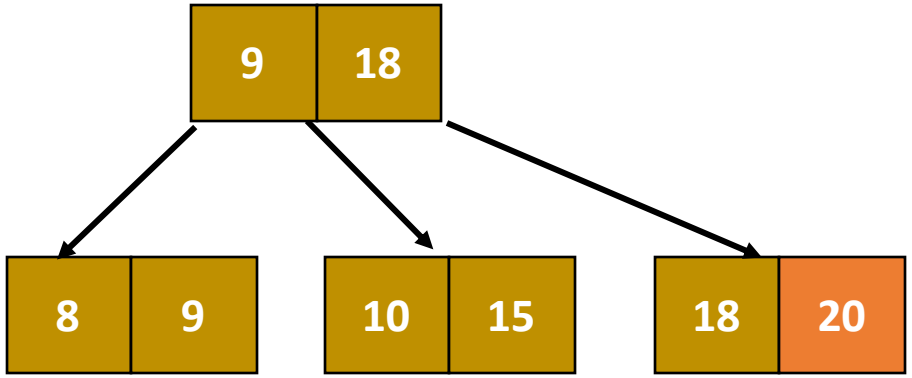
B Tree Insertion

- Normally between *b* and *2b-1* entries per node

- During the insertion, if the entries to a node becomes more than *2b-1,* then the node is split into two

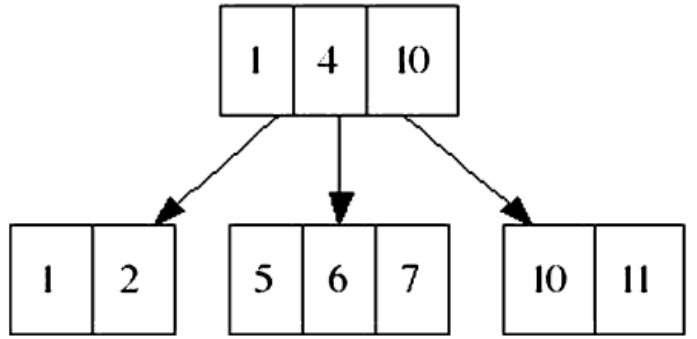


- Assume *b=2*



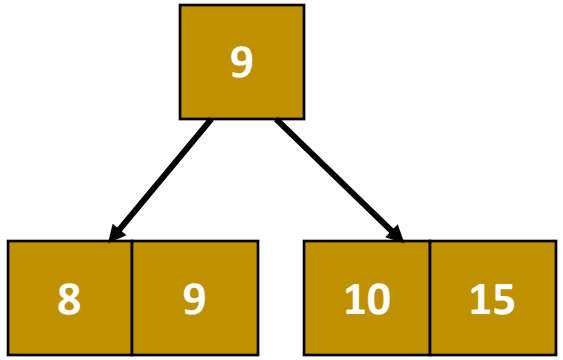
B Tree Insertion

- Normally between *b* and *2b-1* entries per node



- During the insertion, if the entries to a node becomes more than *2b-1,* then the node is split into two

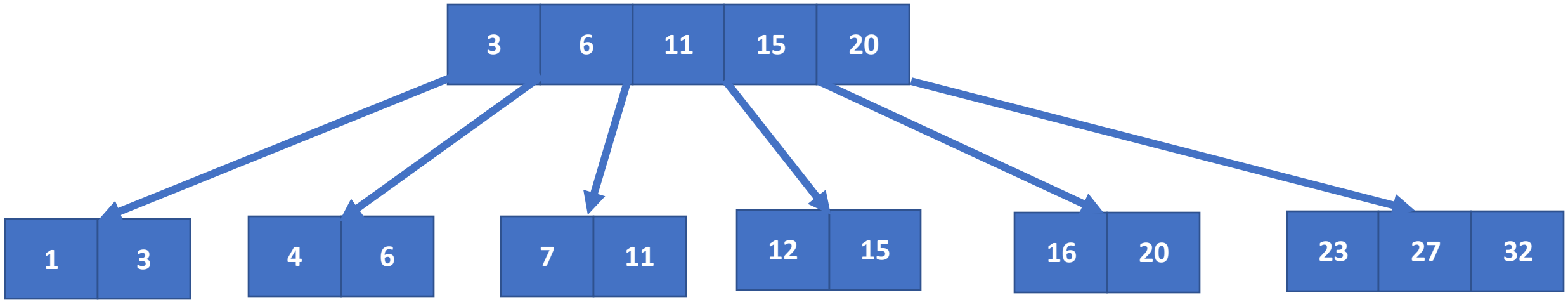
- **Problems with COW**
  - The updates may get back-propagated up to the root
  - All the intermediate pages need to be read to the memory (affect the batch update)
  - Further, the intermediate nodes might be locked due to another concurrent shadowing – the update gets delayed until the previous lock is released



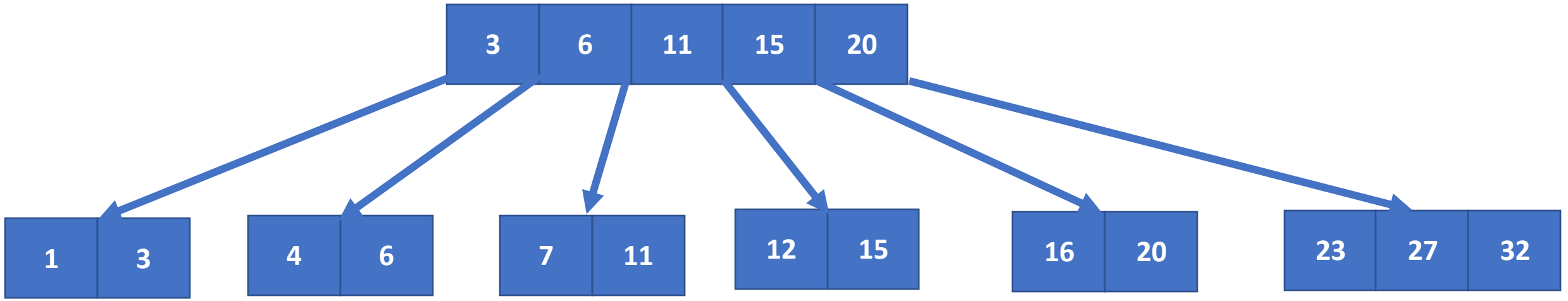
B Tree Operations (COW Friendly)

- Use a proactive approach for tree rebalancing
  - Take a top-down approach instead of a bottom-up approach
- Relax the constraints
  - A node may contain between b and $2b+1$ *keys* for $b \geq 2$
- During the insert-key operation
  - When a node with $2b+1$ entries is encountered, it is split
- During the remove-key operation
  - When a node with b entries is encountered, it is merged

A B-Tree with *b* and *2b+1* keys (*b=2*)

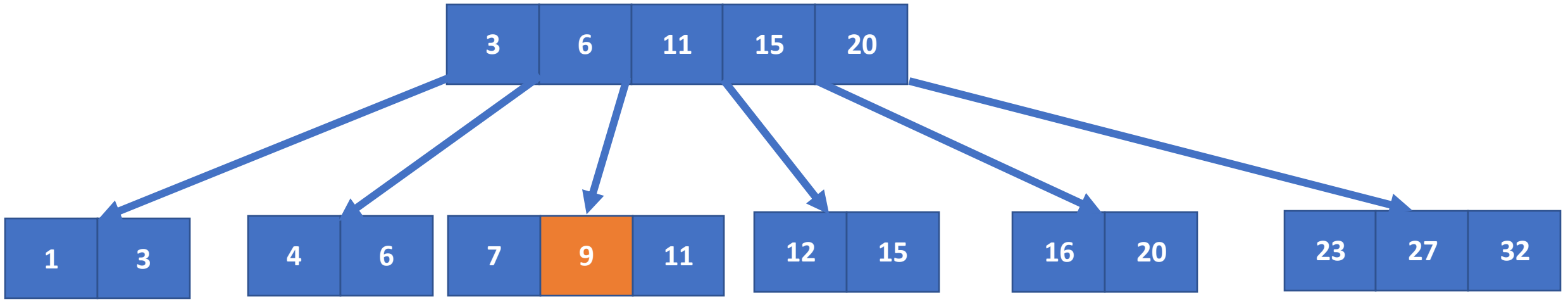


A B-Tree with *b* and *2b+1* keys (*b*=2)

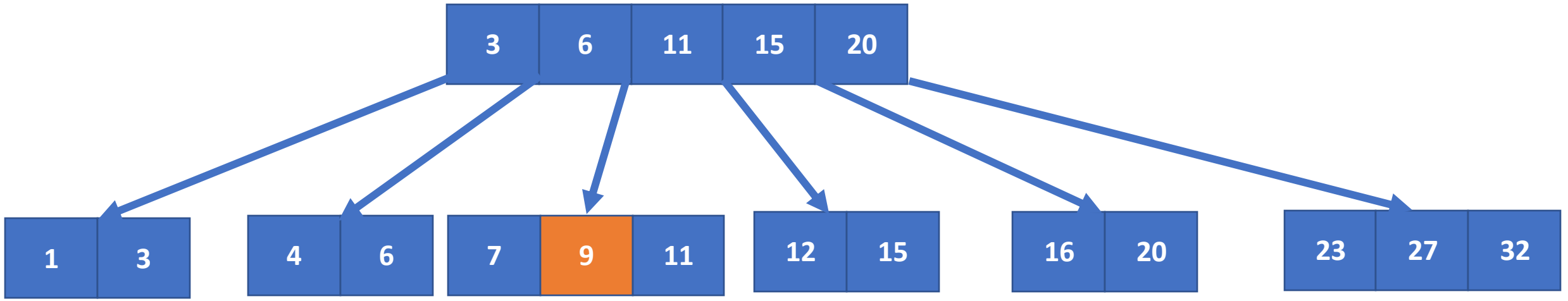


Insert 9 to this tree – what will be the normal operation?

Insert 9 (with normal operations)

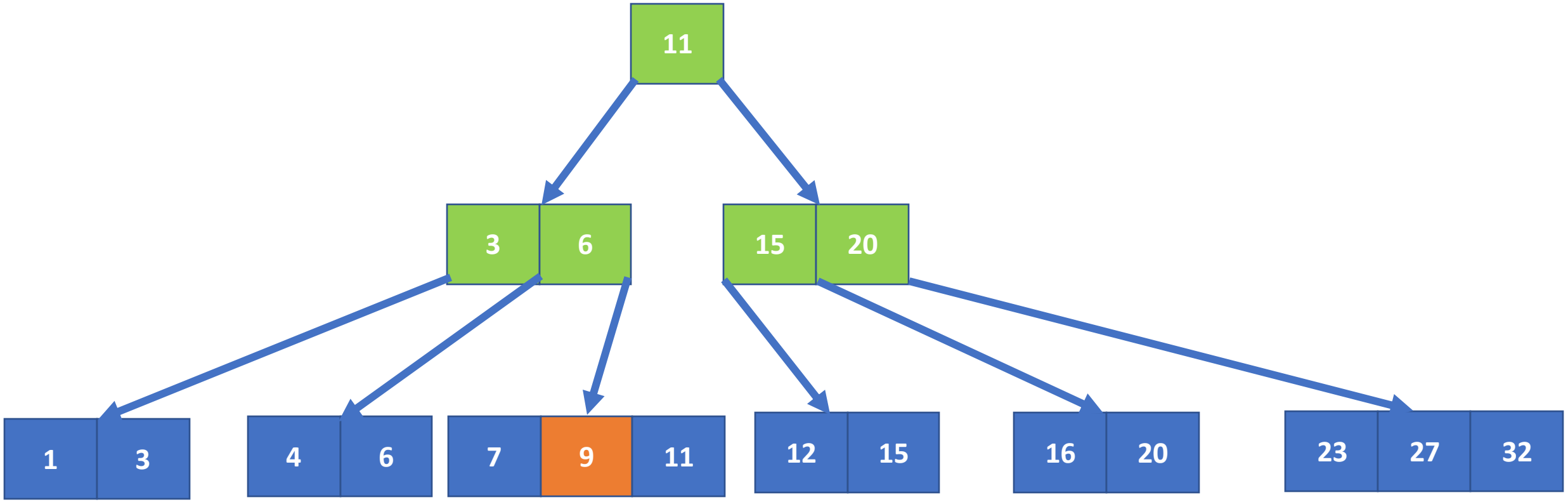


Insert 9 (with normal operations)



What will be the tree structure with proactive split?

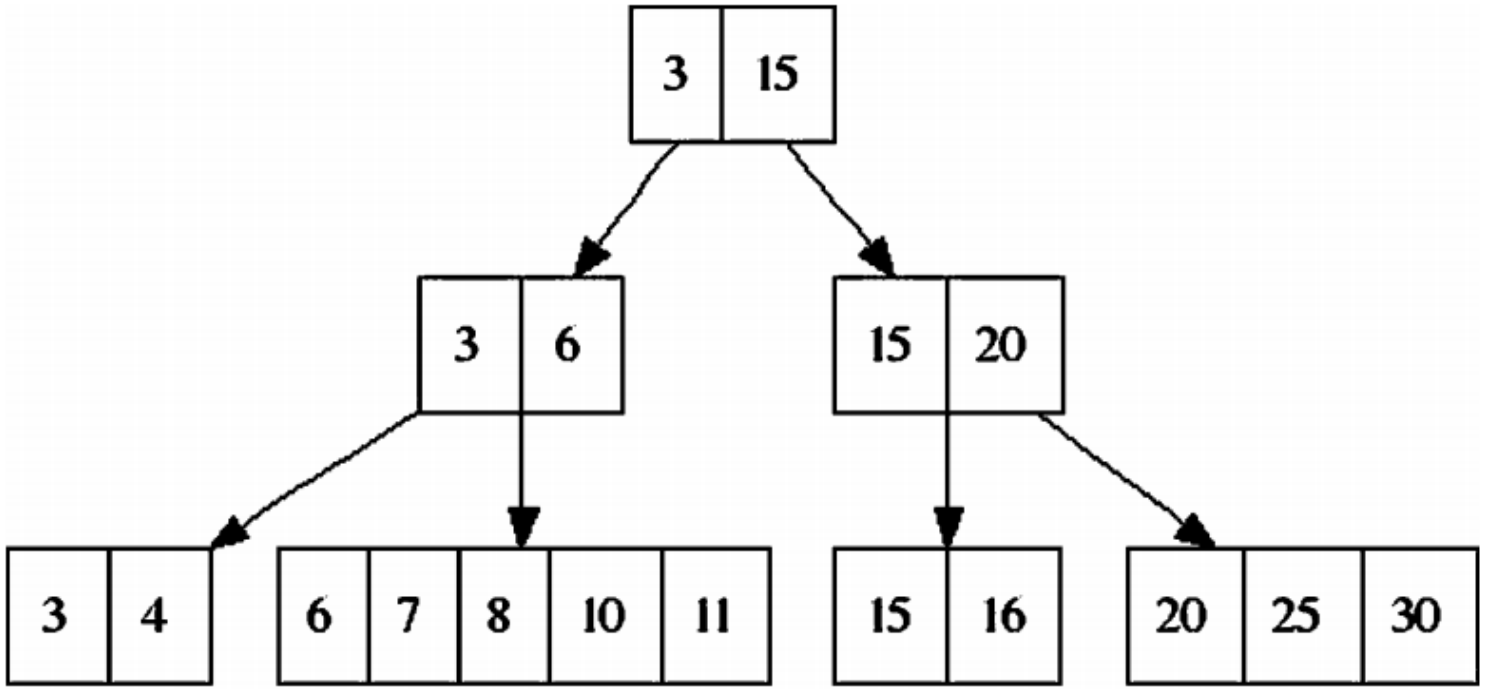
Insert 9 (with Proactive Split)



Remove-key

- Proactive merge is used during the remove-key operations

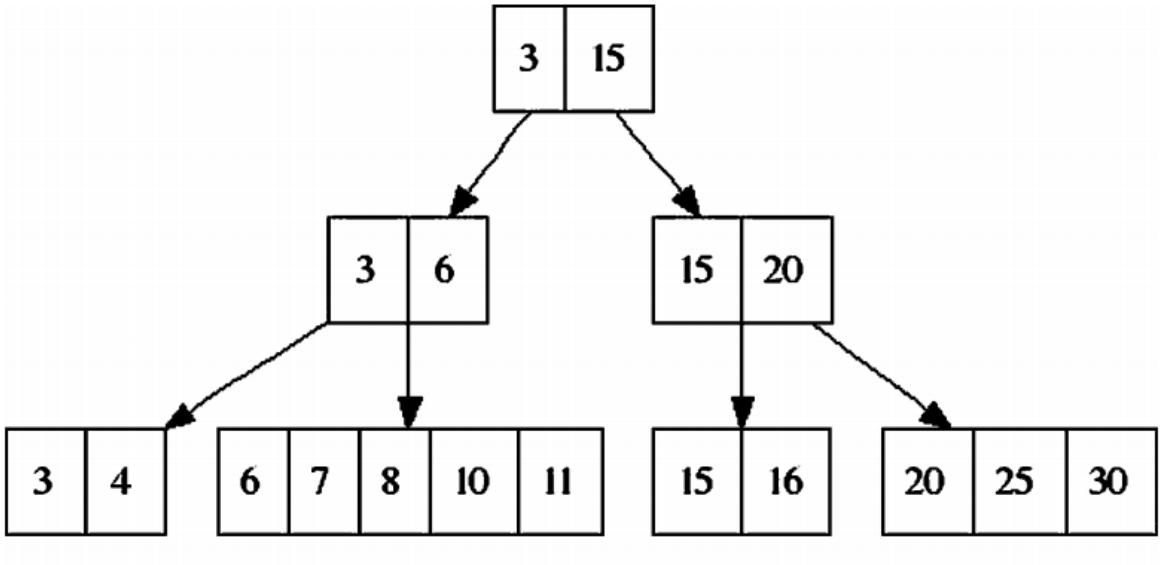
Remove 10 fom the tree

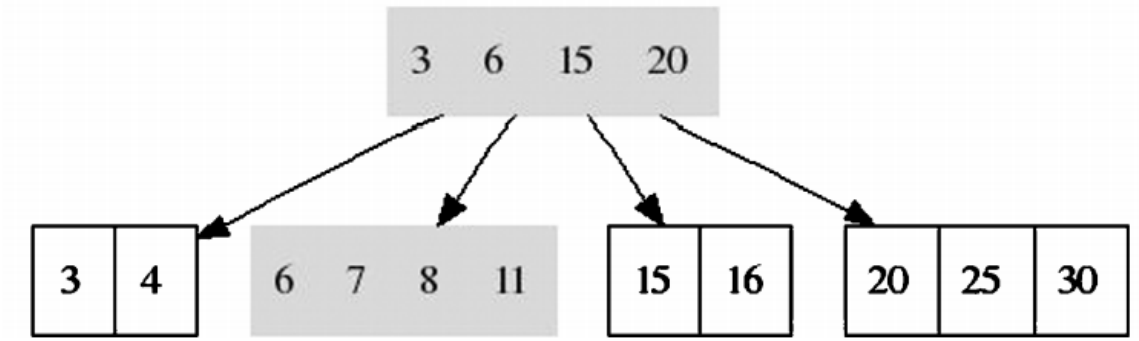


Remove-key

- Proactive merge is used during the remove-key operations

- Grey nodes are shadowed





Cloning a File System

- Create another instance of the file system preserving its structure

- Let Tp be a B tree and Tq is the clone of Tp
  - **Space Efficiency:** Tp and Tq should share the common pages as much as possible
  - **Speed:** Constructing Tq from Tp should take minimum time
  - **Number of clones:** It should be possible to clone Tp as much times as needed
  - **Clones as First-class Citizens:** It should be possible to clone Tq

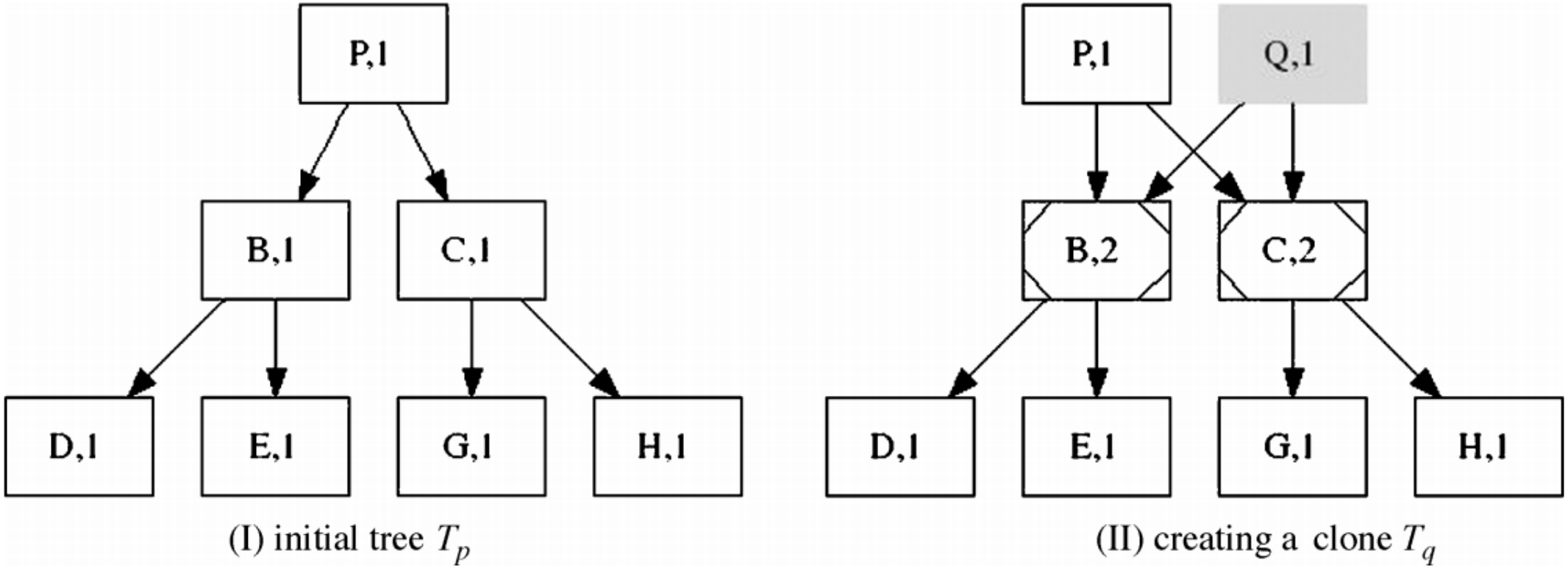
Cloning using Shadowing

- Use a free space map that maintains a reference count for each block
  - Records how many times a page is pointed to
  - Zero reference count -> the block is free

- Instead of copying a tree, the ref-counts of all its nodes are incremented by one
  - Indicates that the nodes belong to two trees instead of one; the nodes are shared pages

- **Lazy updates of reference counts –** Instead of updating the reference counts for all the nodes in the tree, just update the immediate children's; rest follow

Cloning using Shadowing

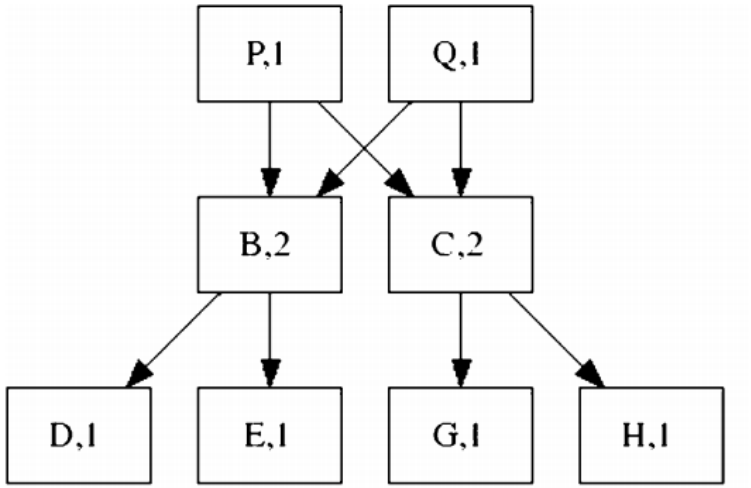


Insert-key and Remove-key on a Cloned Tree

- Before modifying a page, it is "marked-dirty"
  - The runtime system knows that the page is about to be modified
  - Gives it a chance to shadow the page if necessary

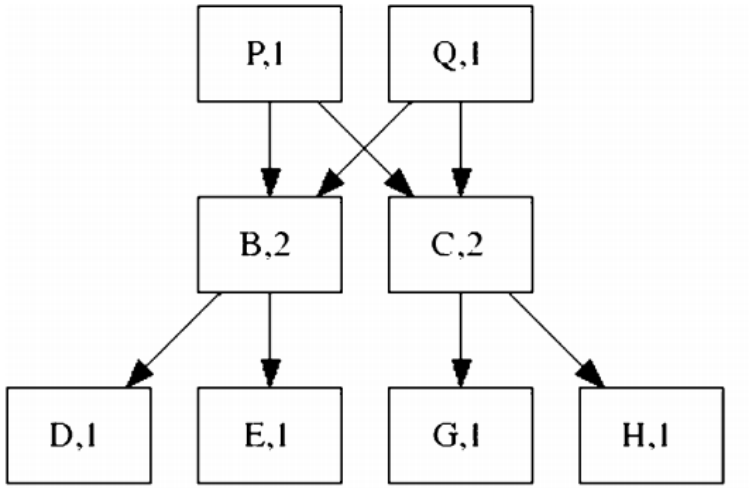
- When marking dirty a clean page N
  - If the reference count is 1, nothing special is needed; same as the tree without cloning
  - If the reference count is more than 1, and page N is relocated from address L1 to address L2
    - The reference count of L1 is decremented
    - The reference count of L2 is made 1
    - The reference count of L's children is incremented by 1

Insert-key and Remove-key on a Cloned Tree

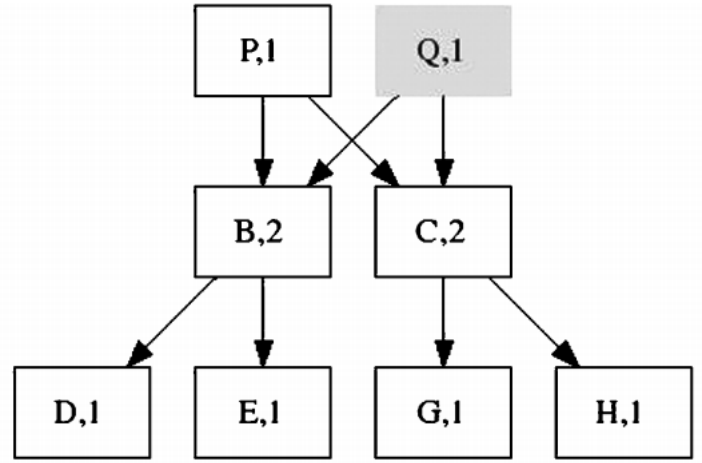


(I) initial trees T_p and t_q

Insert-key and Remove-key on a Cloned Tree

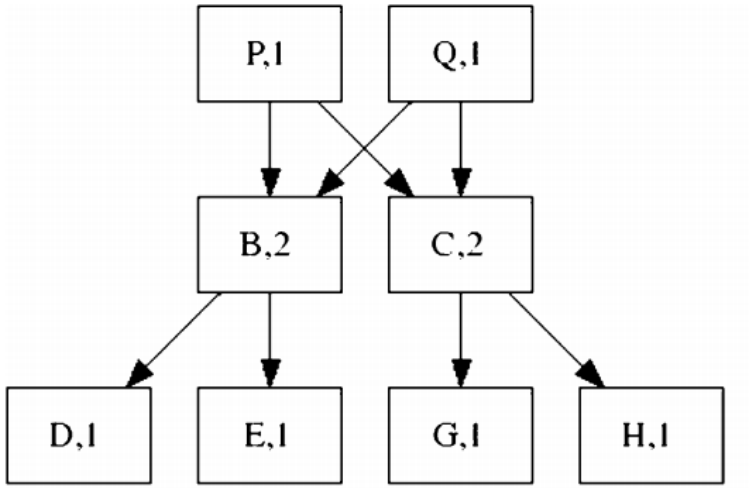


(I) initial trees T_p and t_q

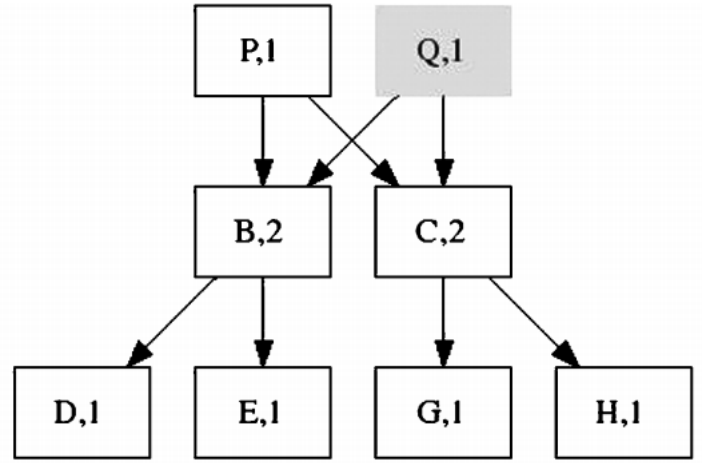


(II) shadow Q

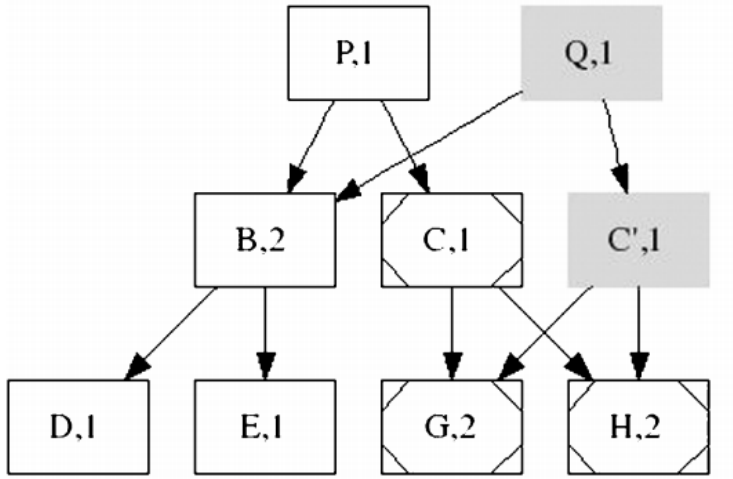
Insert-key and Remove-key on a Cloned Tree



(I) initial trees T_p and t_q

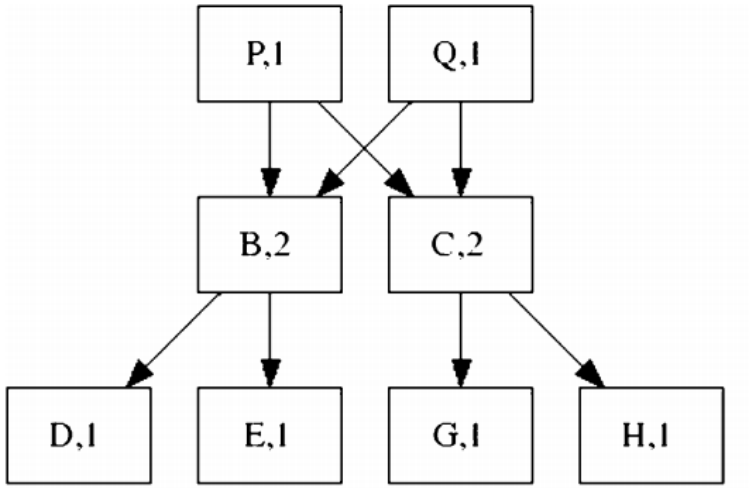


(II) shadow Q

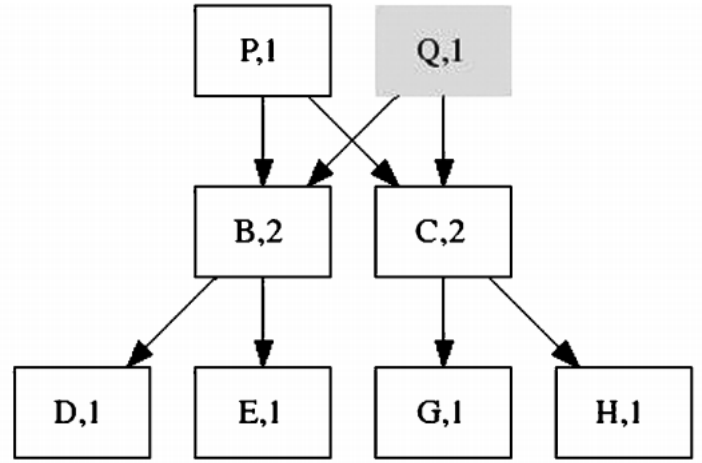


(III) shadow C

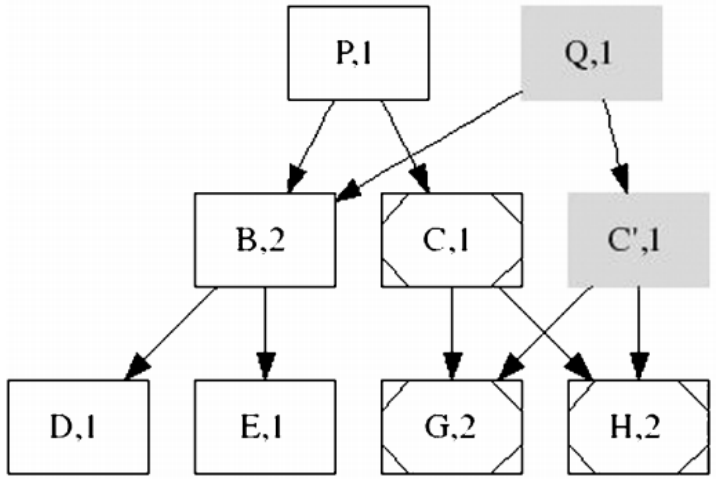
Insert-key and Remove-key on a Cloned Tree



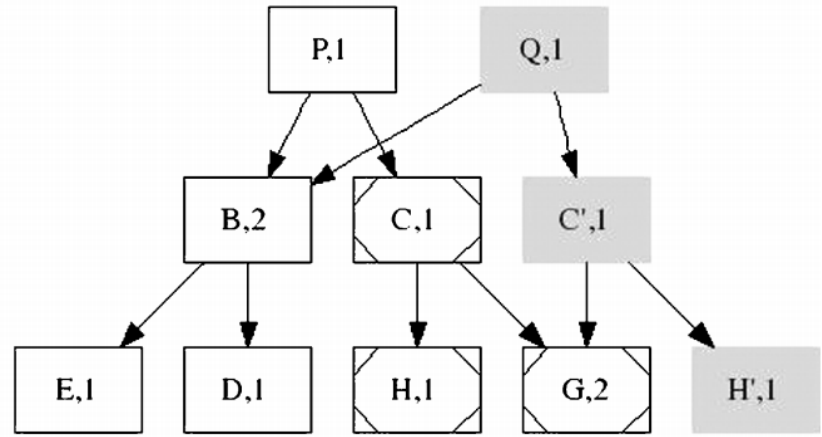
(I) initial trees T_p and t_q



(II) shadow Q



(III) shadow C



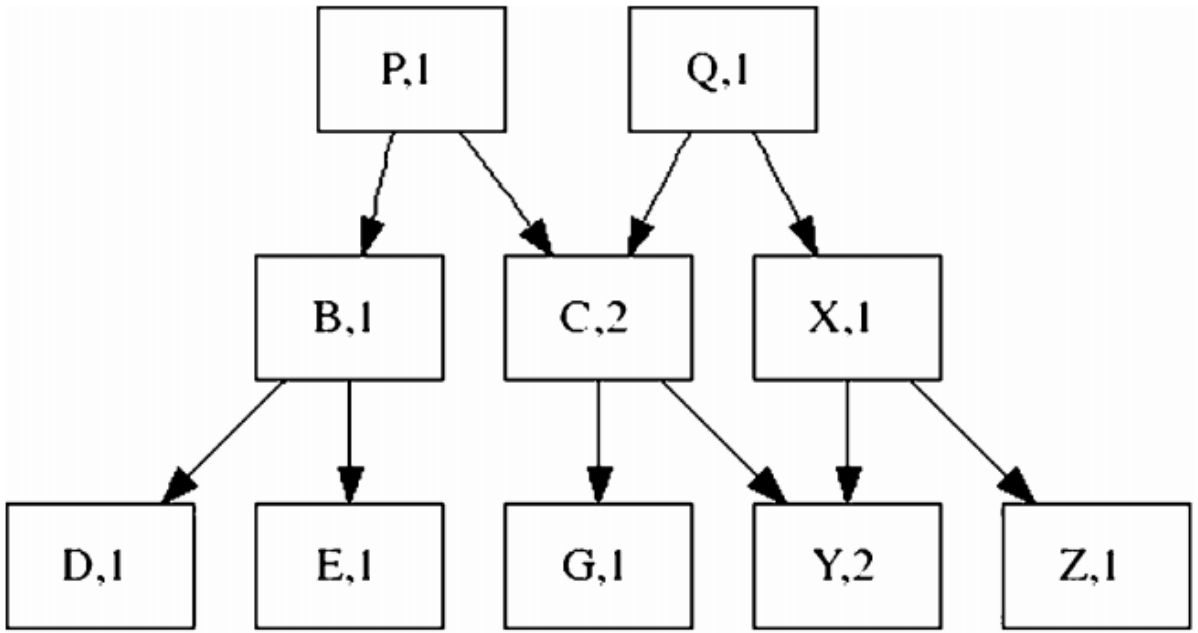
(IV) shadow H

Delete a Cloned Tree

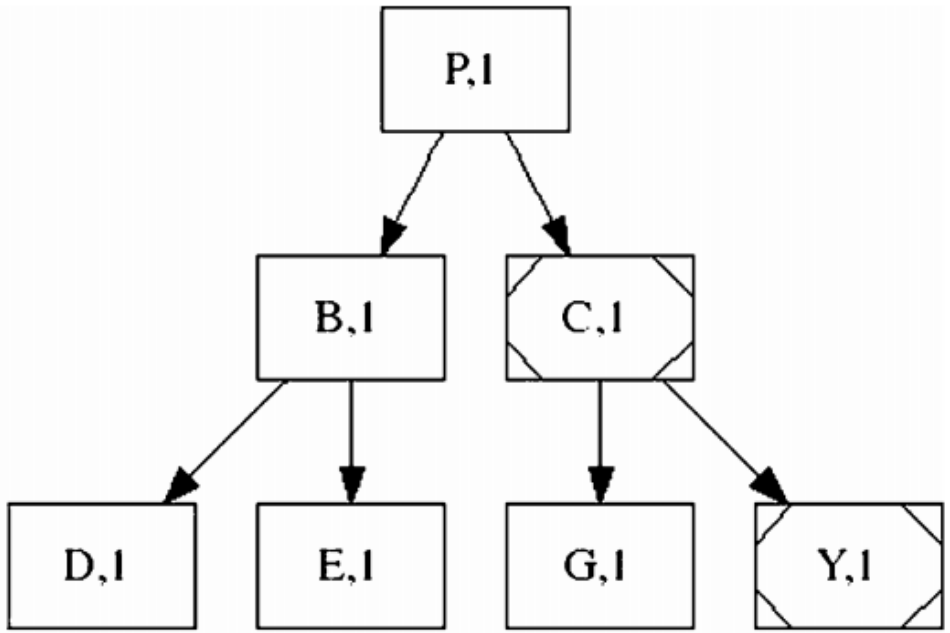
- All nodes are deallocated based on a post-order traversal of the tree
  - Reference counts need to be updated

- Tree Tp is being deleted; during the downward part of the post-order traversal, node N is reached
  - If the ref-count of N > 1, decrement the ref-count and stop downward traversal
  - If the ref-count of N = 1, then it belongs only to Tp. Continue the downward traversal, and on the way back, deallocate N

Delete a Cloned Tree

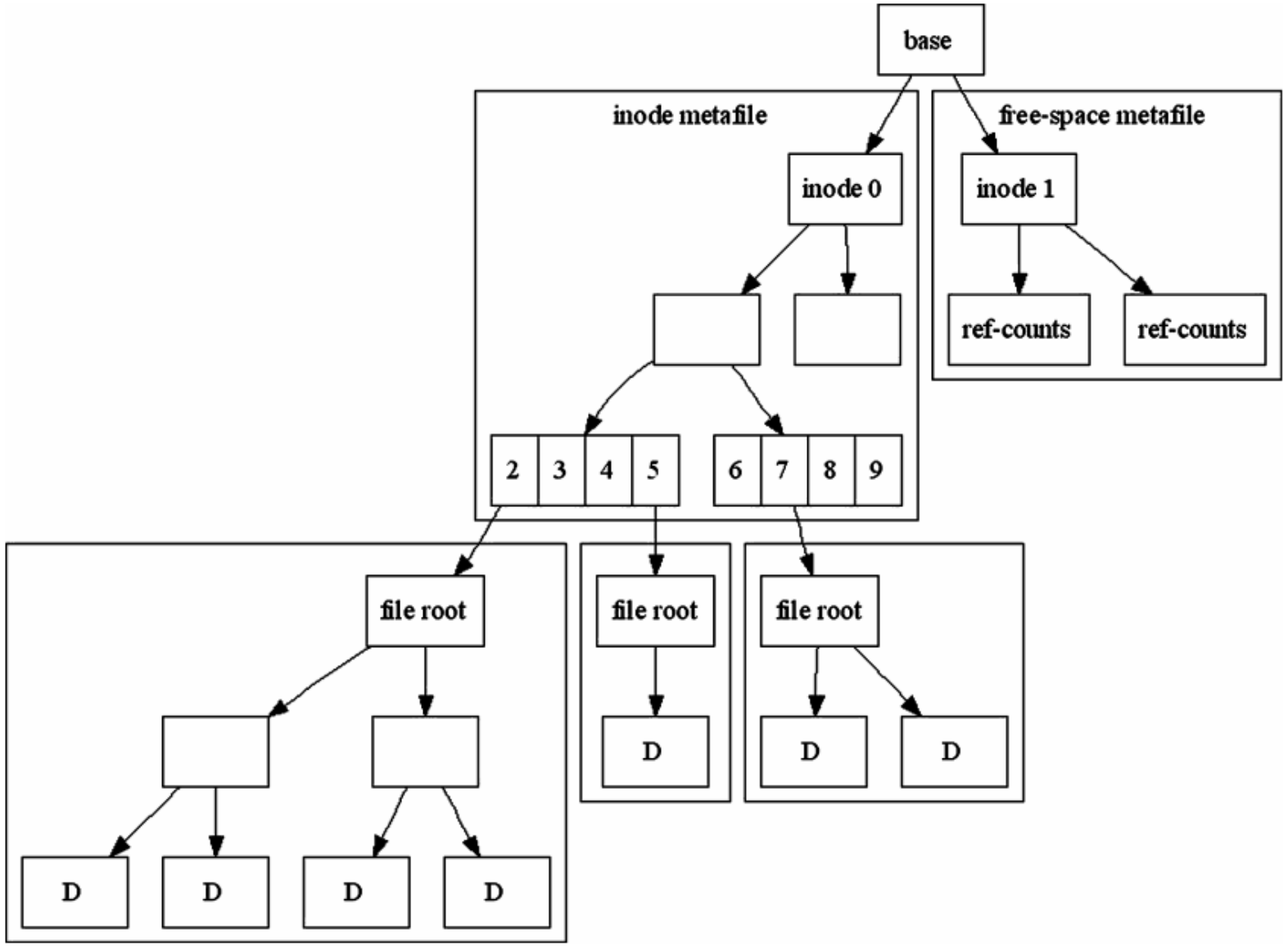


(I) initial tree T_p and T_q

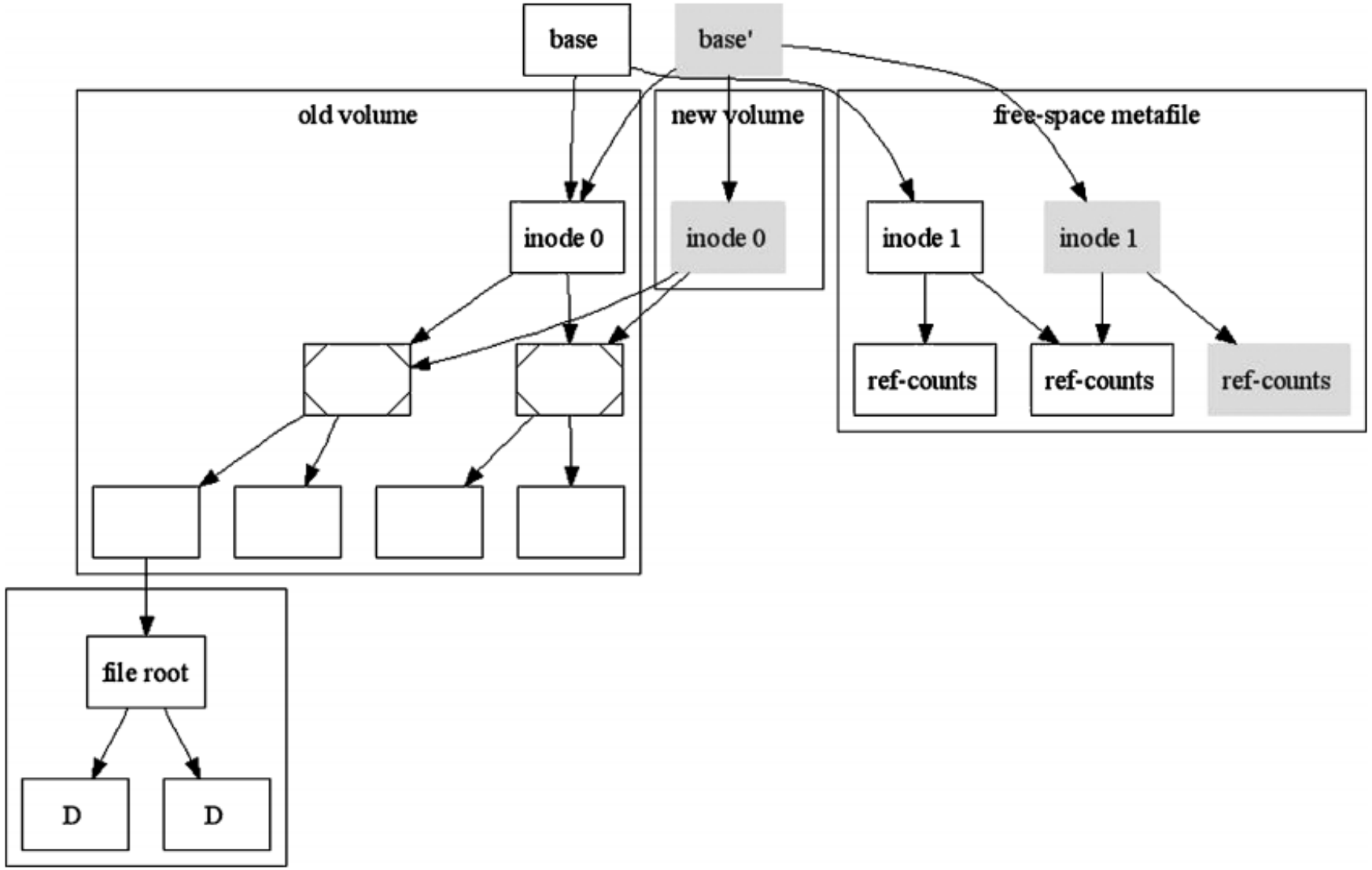


(II) After deleting T_q

A Basic B Tree based File System

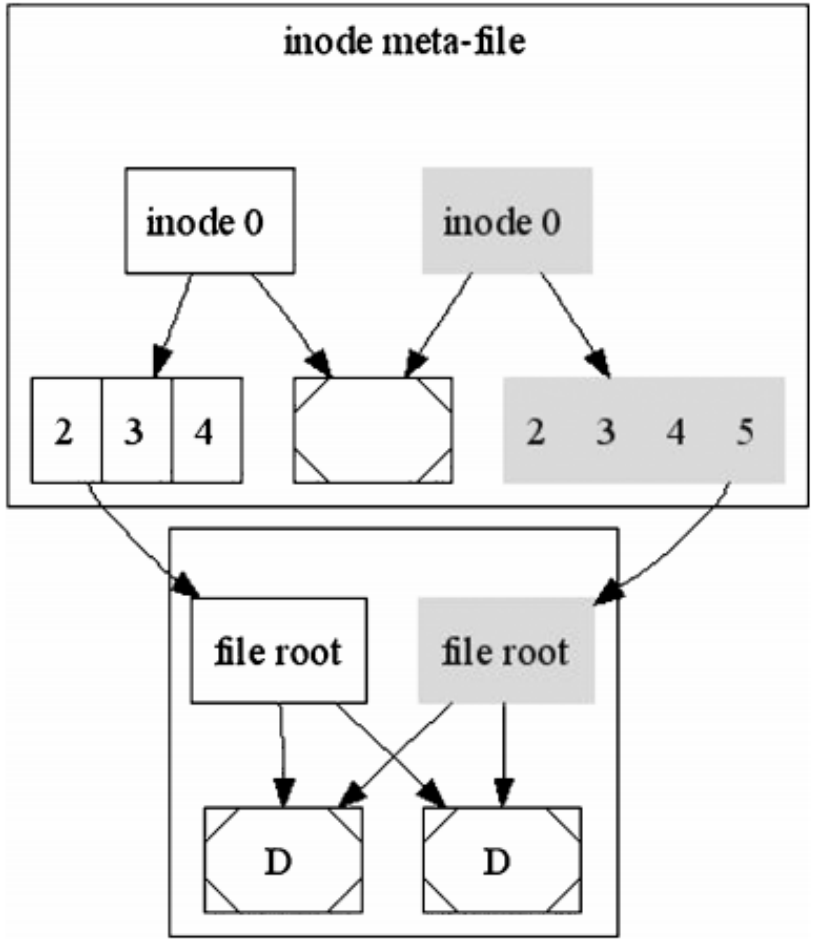


Cloning a Volume



Cloning a File

- Create a new inode with a new file name, clone the B tree holding the file data
  - Inode 2 points the original file
  - Create a new inode (inode 5)
    - Note that the root (inode 0) needs to be shadowed as well
  - Create a new file root; inode 5 points to the new file root



Reference

Rodeh, Ohad. "B-trees, shadowing, and clones." *ACM Transactions on Storage (TOS)* 3.4 (2008): 1-27.

Happy Learning