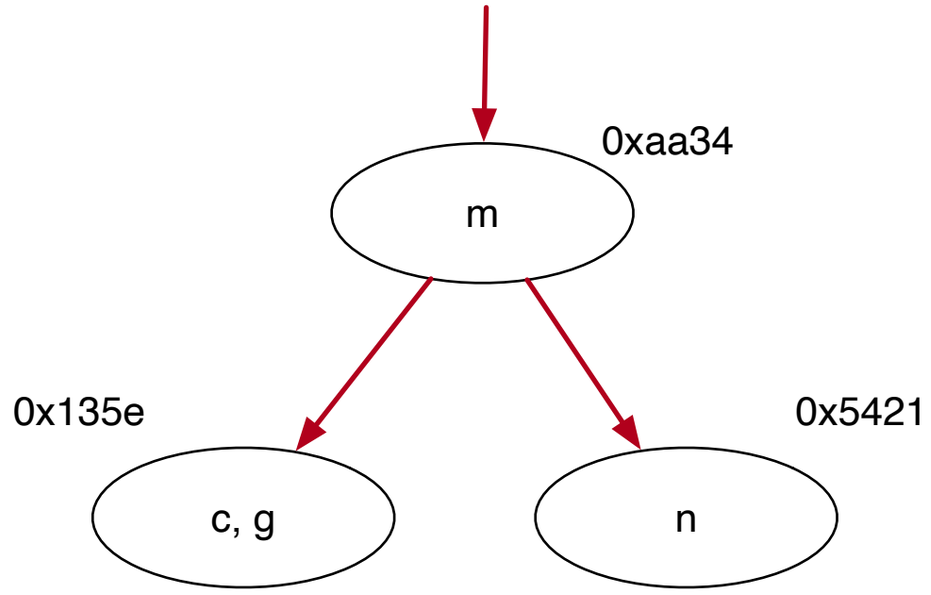
Concurrent Data Structures

Thomas Schwarz, SJ

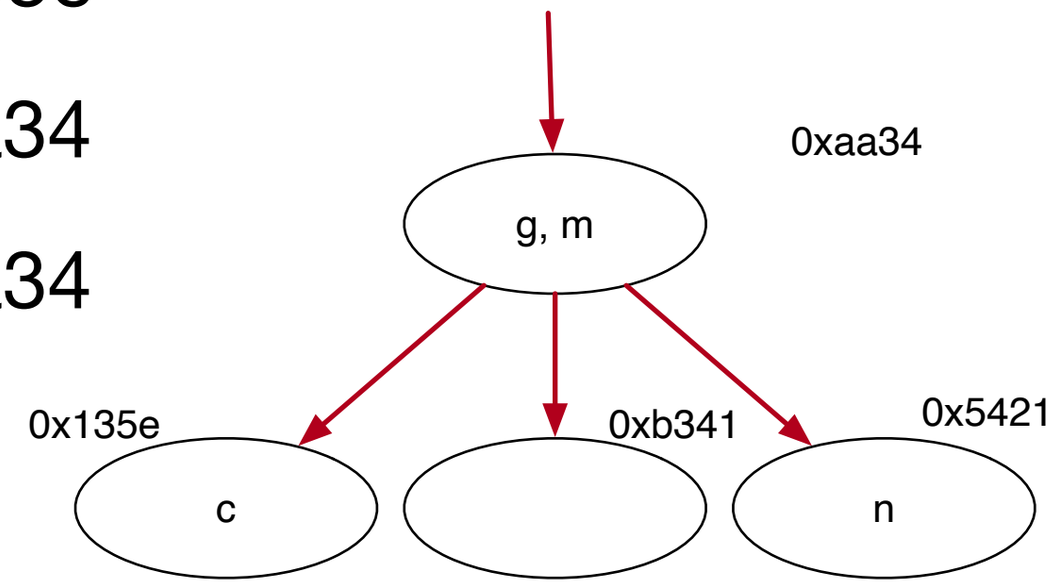
B-tree Locking

- Two threads working on a B-tree can interfere with each other if at least one changes the tree structure
  - Thread 1: Look-up c
  - Thread 2: insert k



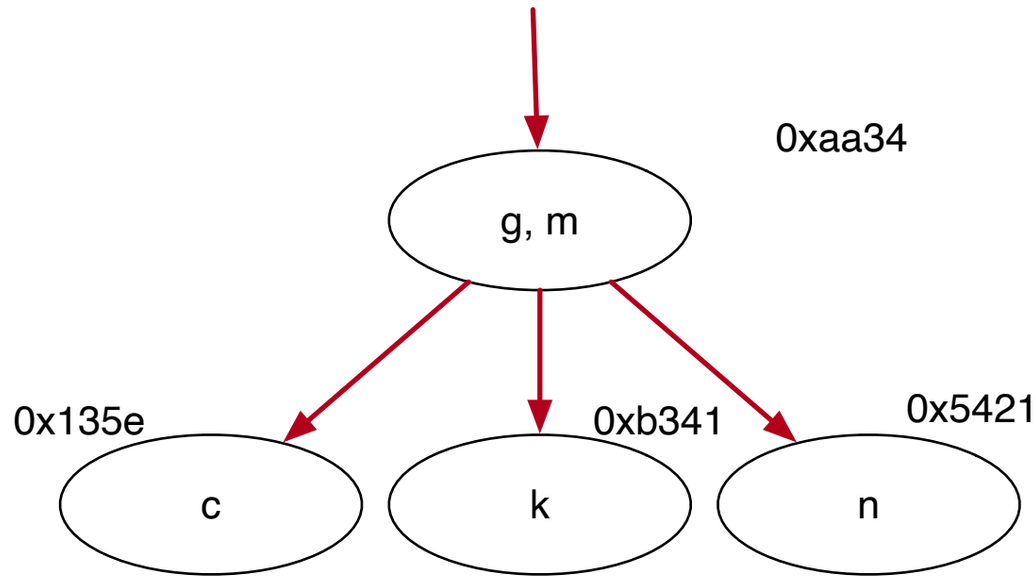
B-tree Locking

- Thread 1 accesses Node at 0xaa34
- Thread 1 accesses Node at 0x135e
- Thread 2 accesses Node at 0xaa34
- Thread 2 accesses Node at 0xaa34
- Thread 2 creates a new Node 0xb341
- Thread 2 inserts g into Node 0xaa34
- Thread 2 removes g from Node 0x135e



B-tree Locking

- Thread 1 cannot find g in Node 0x135e and returns
- Thread 2 inserts *k* into Node 0xb341
- Thread 2 returns



B-tree Locking

- B-tree locking for disks and solid state:
  - Only complete blocks can be written
- B-tree locking for main memory
  - Can lock parts of a node

B-tree Locking

- Complete locking:
  - All readers and updaters locks the root
    - No concurrency possible
  - Top-down locking
    - Pure reader: Starting with root, lock node, identify child, acquire child node, release parent node
    - Updater: Starting with root: lock node, identify child, acquire child node, until done. Then release all locks.

B-tree Locking

- Improving top-down locking
  - Safe node:
    - A node that the updater is never going to change
    - Updater can unlock all nodes above the safe node

B-tree Locking

- When is a node safe?
  - Insert operation:
    - A node can change by a rotation
    - A node can change by splitting
  - In both cases:
    - Parent of a node with overflow potential can change

B-tree Locking

- When is a node safe?
  - Deletion
    - A node can change by rotation
    - A node can change by merging
    - In both cases, parent of a node with potential underflow can change

B-Tree Locking Protocols

- Use different types of locks

Mode	S	IX	SIX	X
S	✓	✓	✓	
IX	✓	✓		
SIX	✓			
X				

- An X-lock is exclusive
- An S-lock prevents other threads to get a higher class lock

B-Tree Locking Protocols

- Lock-coupling for top-down algorithms
  - Request a lock on an index page while holding a lock on the page's parent
  - Check that new page is safe
  - Release lock on parent

B-Tree Locking Protocols

- Sama 76:
  - All operations get an X (exclusive lock) on root and lock-couple their way to a leaf

B-Tree Locking Protocols

- Bayer-Schkolnick 1977
  - Uses a B+ tree: all data is in the leaves
  - Searches get an S-lock on the root and lock-couple to the leaf using S-locks
- B-X algorithm:
  - Updates get an X-lock on the root and lock-couple to the leaf
  - Excludes readers from parts of the tree not in the scope of the update

B-Tree Locking Protocols

- B-SIX algorithm
  - Updates lock-couple with SIX locks
    - This allows readers to get S-locks on these nodes
    - Prevents updaters to interfere with each other as SIX locks are incompatible with each other
  - Updater when reaching a leaf have to convert the SIX locks to X locks
    - This separates readers and the updater
      - Updater cannot get an X-lock with an S-lock by another thread
      - Readers cannot get an S-lock with an X-lock by another thread

B-Tree Locking Protocols

- B-X and B-SIX still make updaters compete a higher nodes
- B-OPT: updaters take an IX lock on root and then IX-lock-couple to leaf
    - Take X-lock on leaf
    - If leaf is safe, operations succeed
    - Otherwise:
        - Release X-lock on leaf
        - Default to B-SIX

B-Tree Locking Protocols

- Top-down algorithms
  - Updaters perform preparatory splits and merges
  - If inserter encounters a full node during descent:
    - Splits the node
  - If deleter encounters a node with one entry, merge with sibling.
- Now leaf level insertion / deletion know that the parent is always safe

B-Tree Locking Protocols

- TD-X
  - Updater gets an X lock on the root
  - Lock-couples to an X lock to the leaf
  - Before releasing the lock on the parent:
    - make split / merge
- Readers use S-lock coupling

B-Tree Locking Protocols

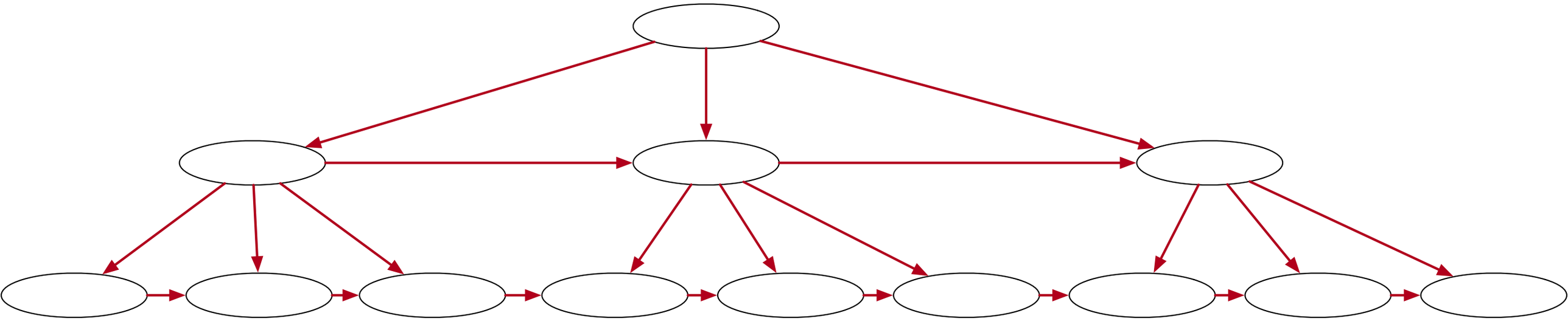
- TD-SIX
  - Updater gets SIX locks
  - Converts to X locks only if a split or merge is actually necessary

B-Tree Locking Protocols

- TD-OPT
  - Updaters make an optimistic first pass
  - Use S-locks lock-coupling from root to leaf
  - If the leaf is unsafe, update to X-locks

B-tree Locking

- B-link trees: (Yao-Lehman)
  - At all levels:
    - Add a link to the right neighbor (can be null)



B-tree Locking

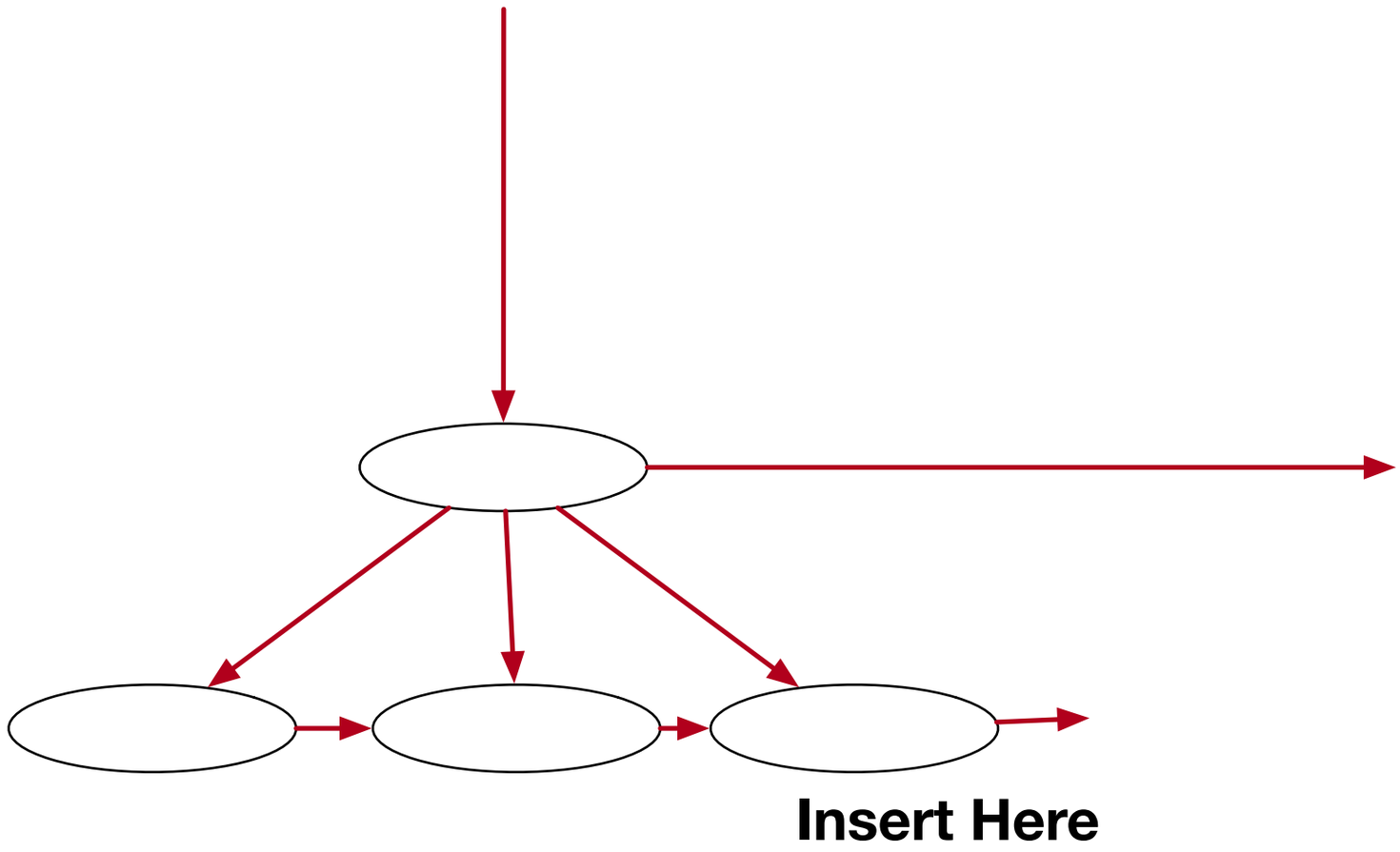
- B-link trees:
  - Add a high key to each node: The first key of the neighbor to the right
  - New invariant:
    - We can find a record by following the link from the father OR from the left neighbor

B-tree Locking

- B-link tree
  - When a node splits:
    - Link is introduced when nodes are split
    - The new link makes the pair of nodes functionally behave as one node
    - Even before other links are updated

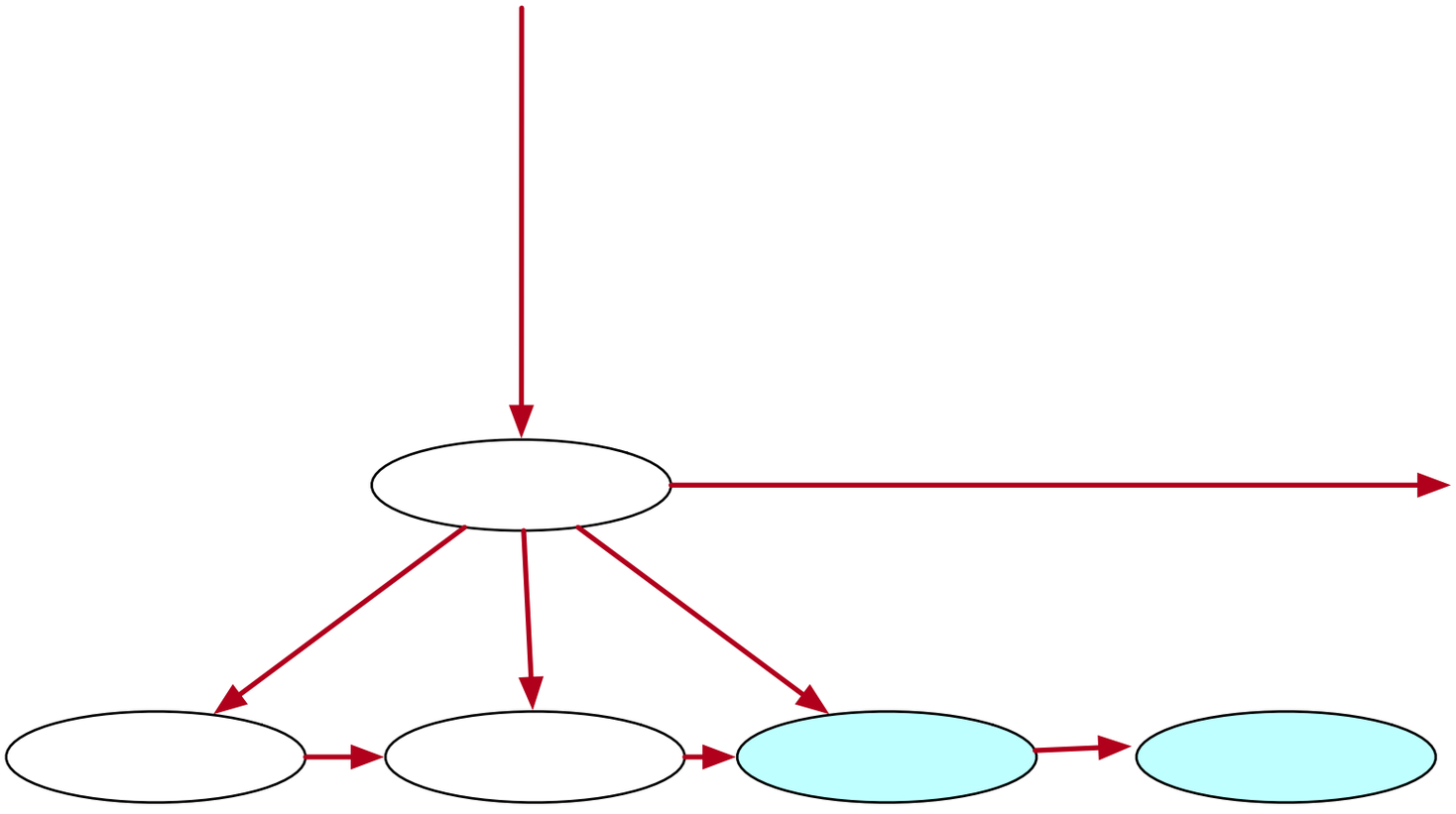
B-tree Locking

- B-link tree



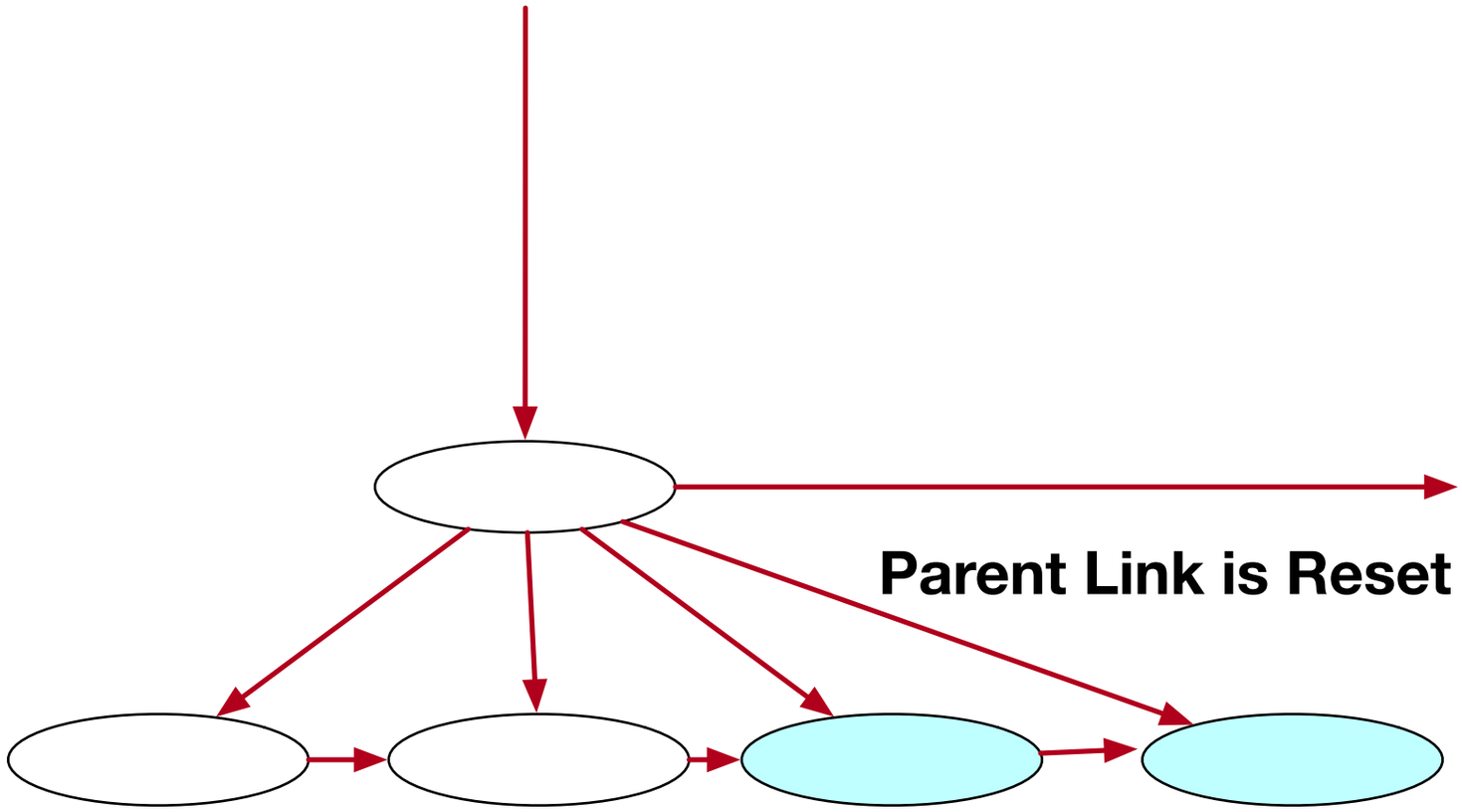
B-tree Locking

- B-link tree



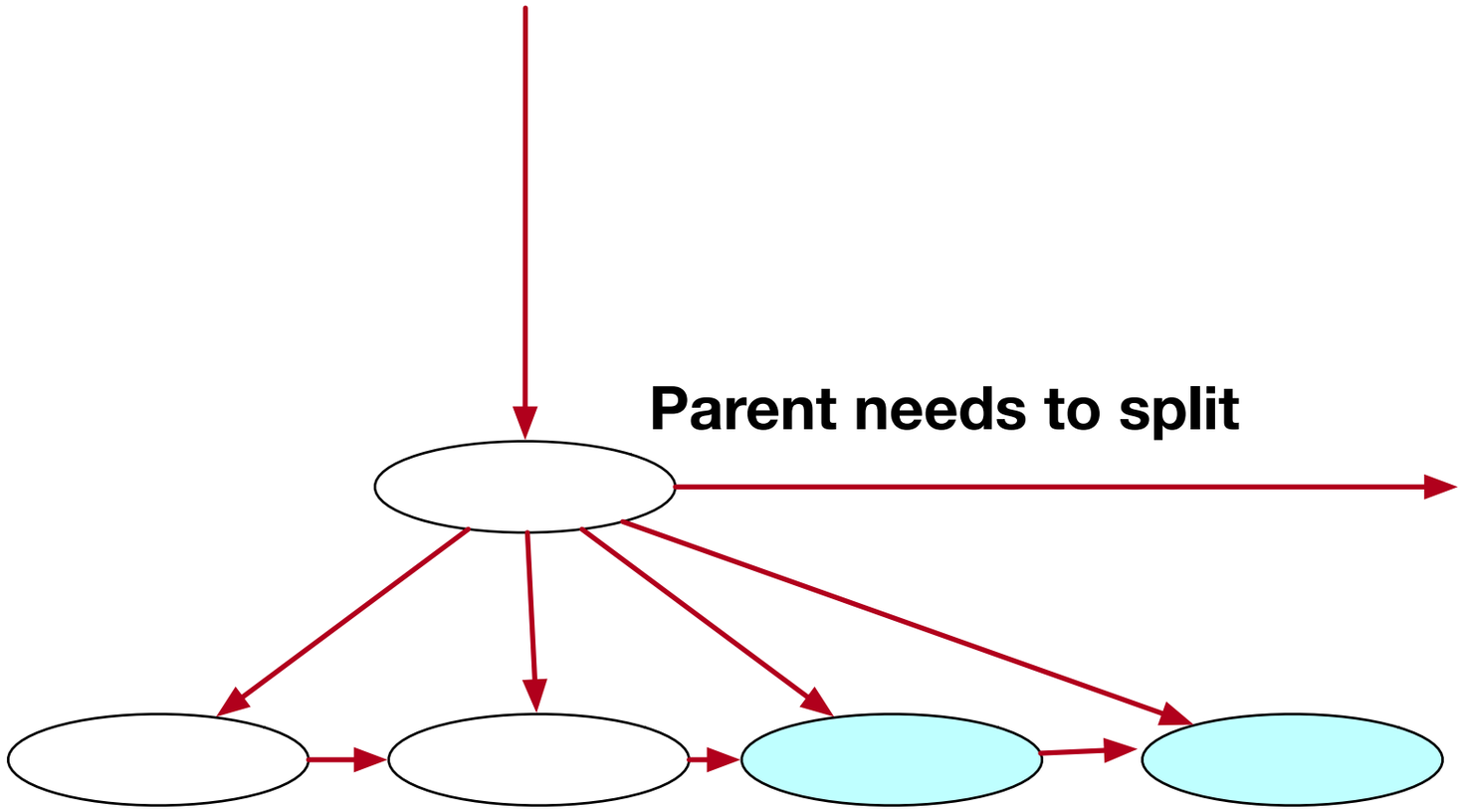
B-tree Locking

- B-link tree



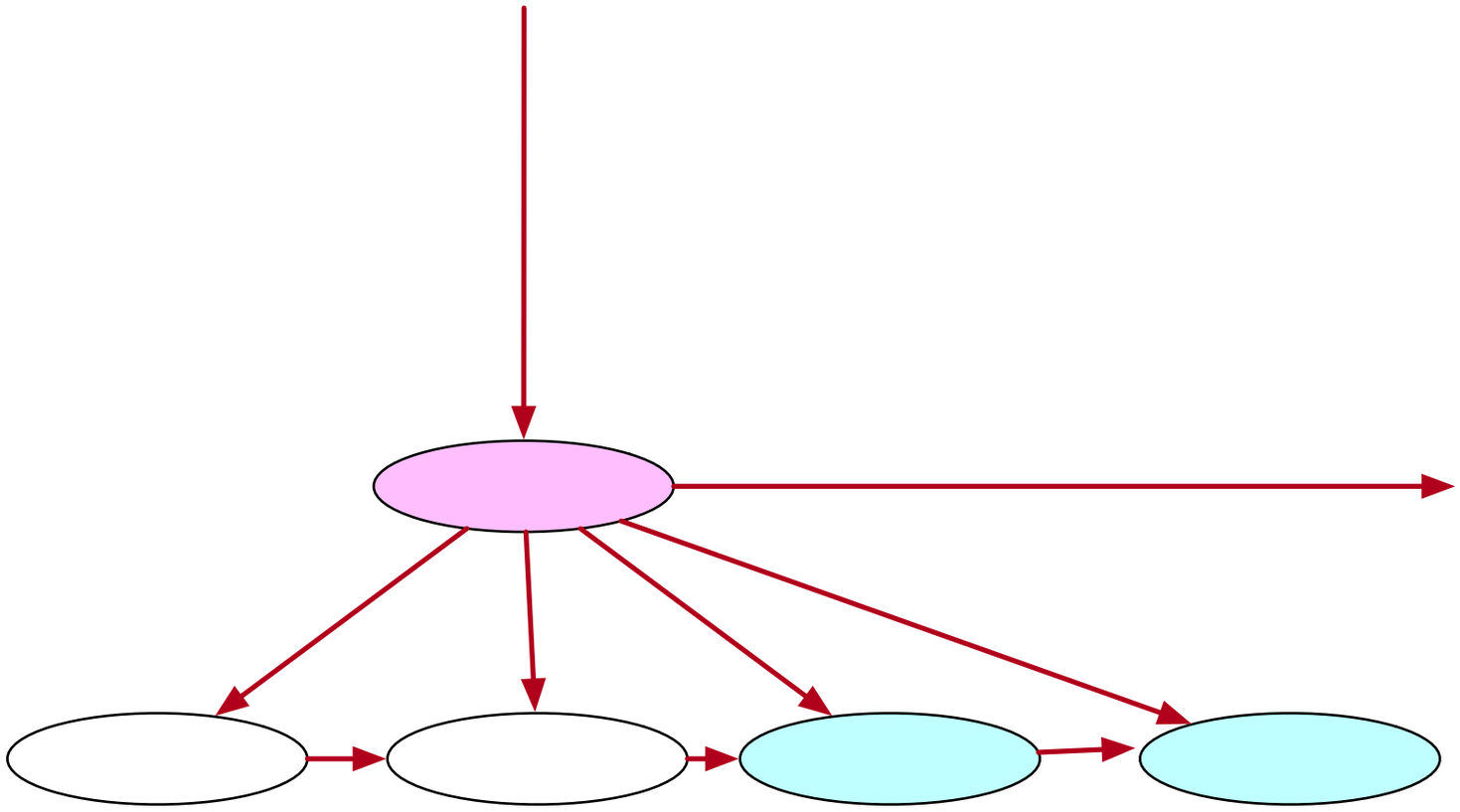
B-tree Locking

- B-link tree



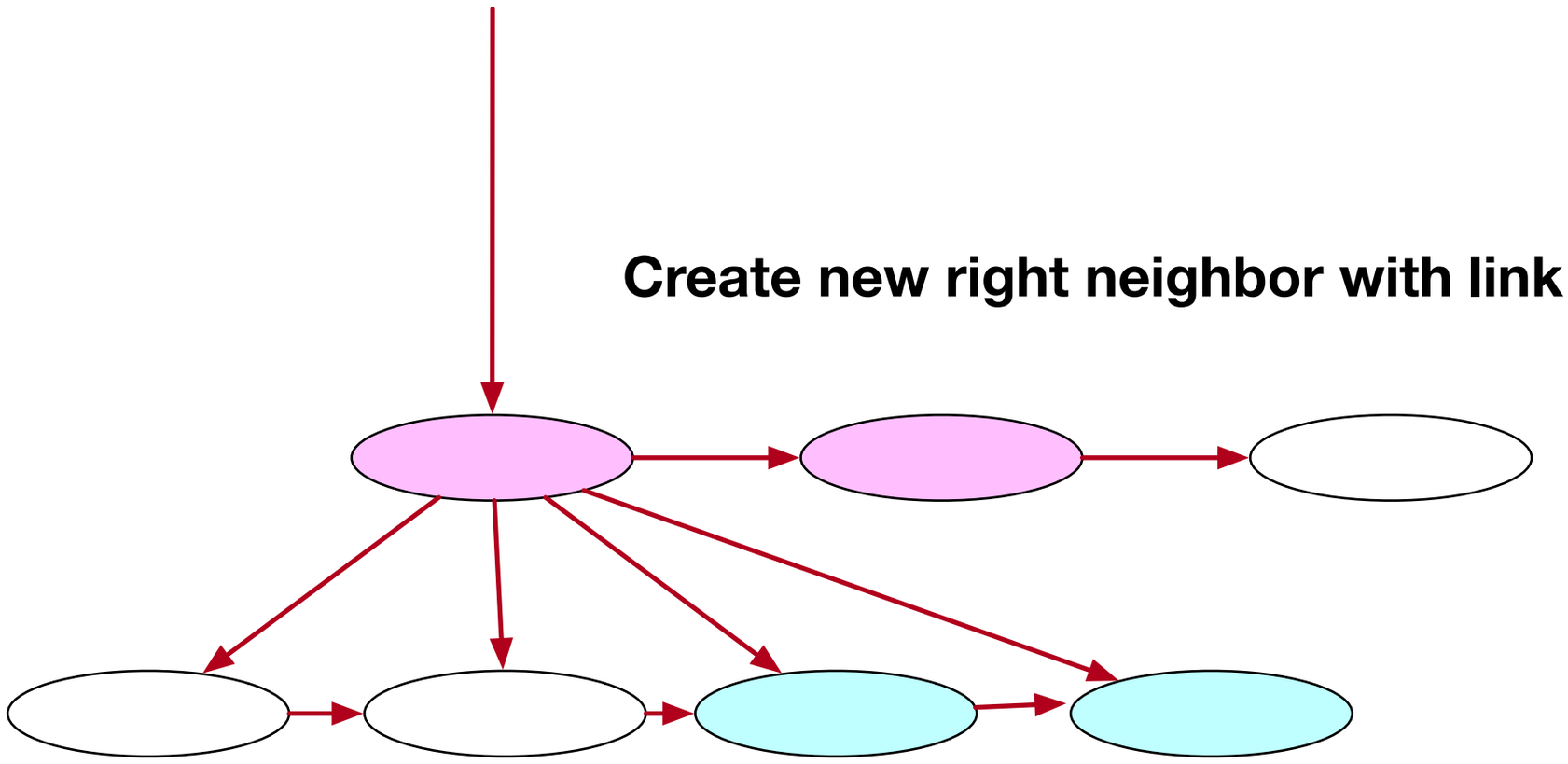
B-tree Locking

- B-link tree



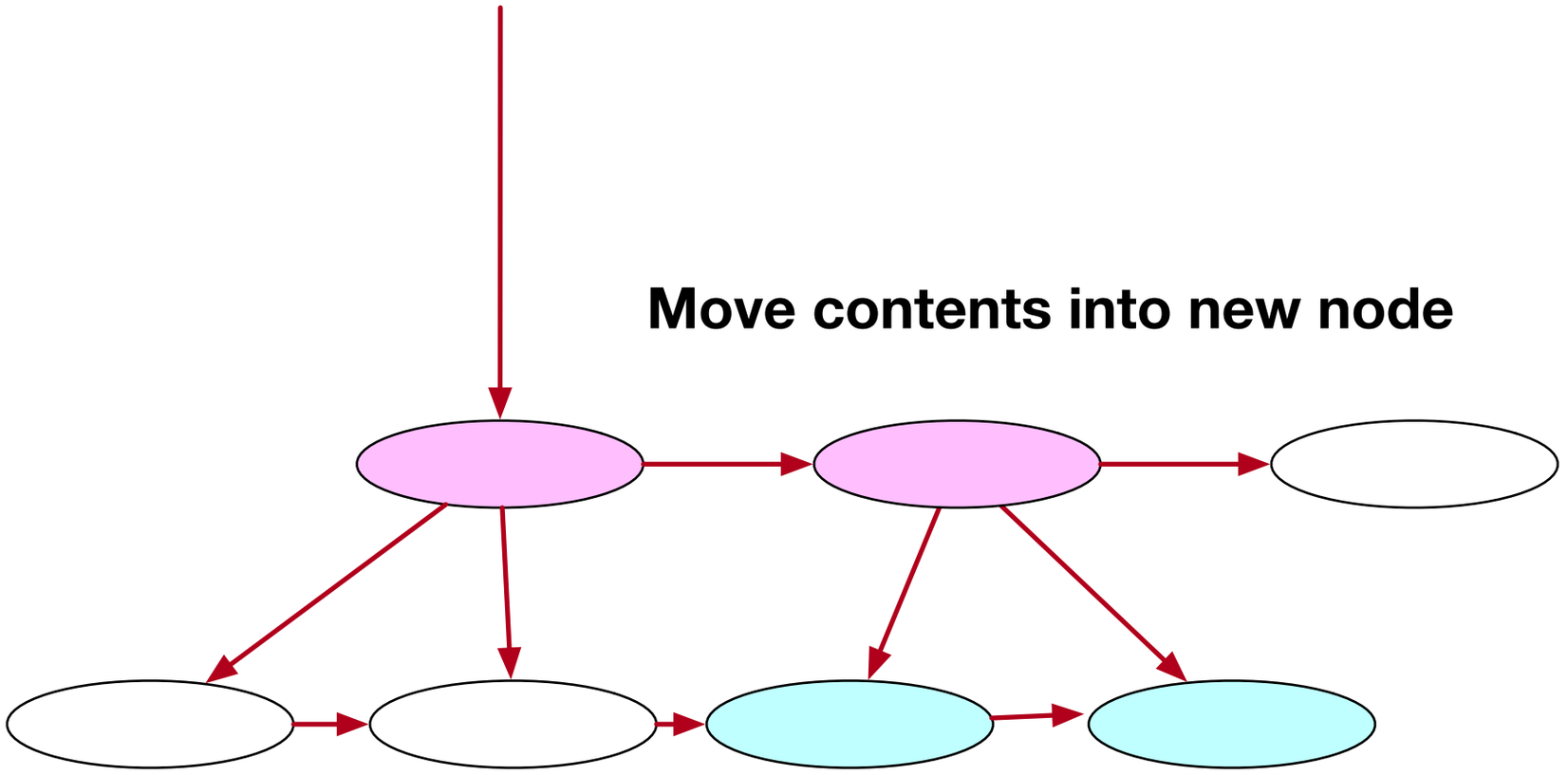
B-tree Locking

- B-link tree



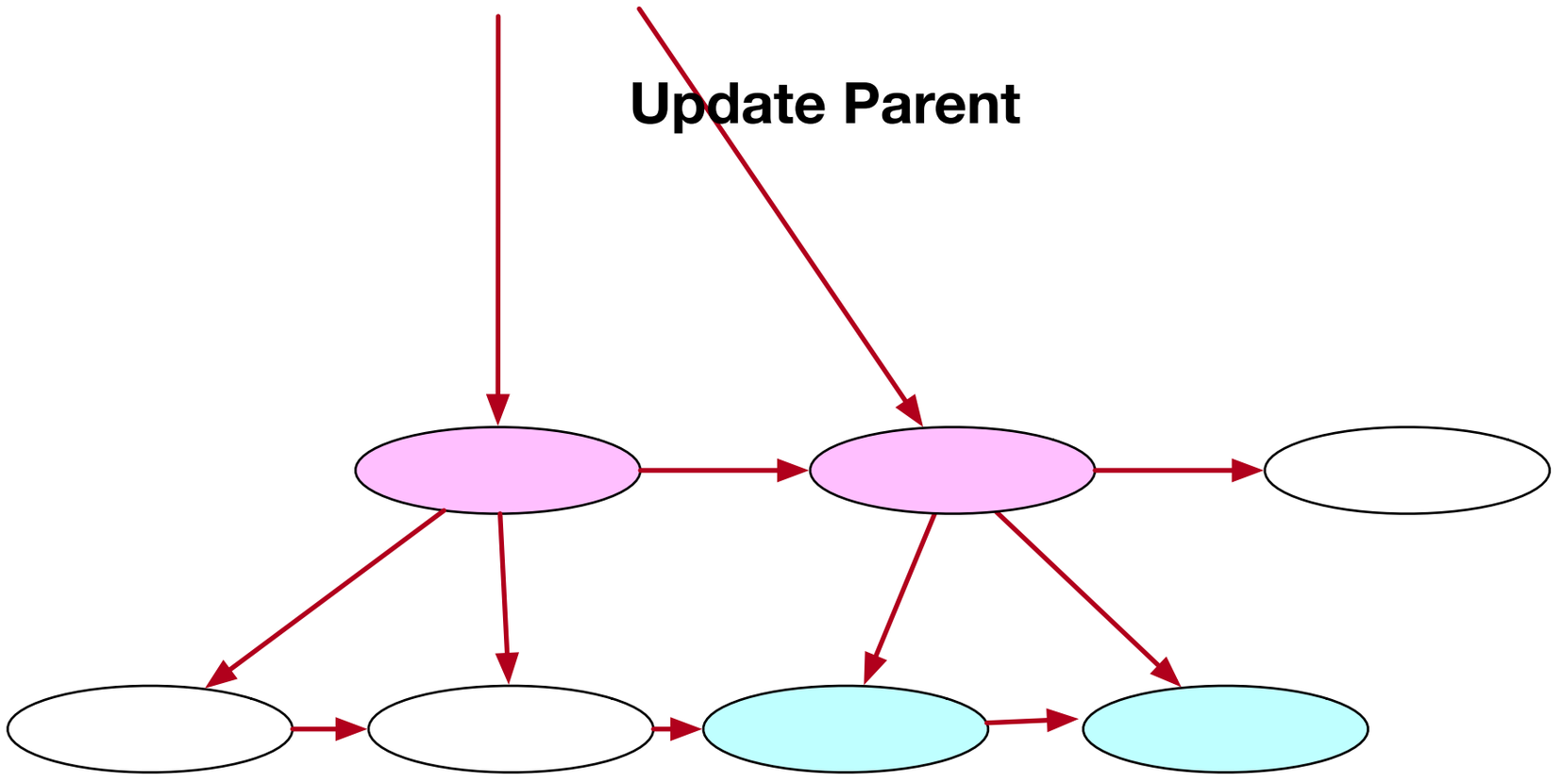
B-tree Locking

- B-link tree



B-tree Locking

- B-link tree



B-tree Locking

- B-link tree
  - Claim:
    - As long as all of these updates are atomic:
      - A reader can always find the right path
- Idea of Proof:
  - Look at changes in reachability after each operation

B-tree Locking

- B-link tree
  - No dead-locks
    - Inserters can only acquire locks on nodes in order
      - Node1 < Node2 if
        - Node1 has less distance from root than Node2
        - OR
        - Node2 can be reached from Node1 through links

B-tree Locking

- B-link tree
  - Livelock
    - Livelocks are possible:
      - A bunch of inserters insert new links whenever a given reader accesses a node

B-tree Locking

- B-link tree
  - Deletion:
    - Just allow nodes to have less than *k* children
    - Do not restructure

B-Tree Locking Protocols

- Lehman Yao algorithm
  - Reader descends the tree using S locks
    - Each page searched is either the child or the right sibling of the current page
      - Parent lock is released before the lock on the next page is acquired
  - Updaters behave the same way
    - Once reaching leaf, update their lock to an X-lock
    - Use link chasing with X-lock to find the right node
- This still has problems, look at the linked paper.

B-tree for Flash

- Problem:
  - Any change to a node results in a page write
    - Some writes propagate

B-tree for Flash

- Wu, Chang, Kuo: BFTL
  - Add a log page to each node
    - Log contains changes
    - Log is contained in RAM and mirrored to Flash
    - Needs an Node Translation Table (NTT) to find log entries for a given node

B-tree for Flash

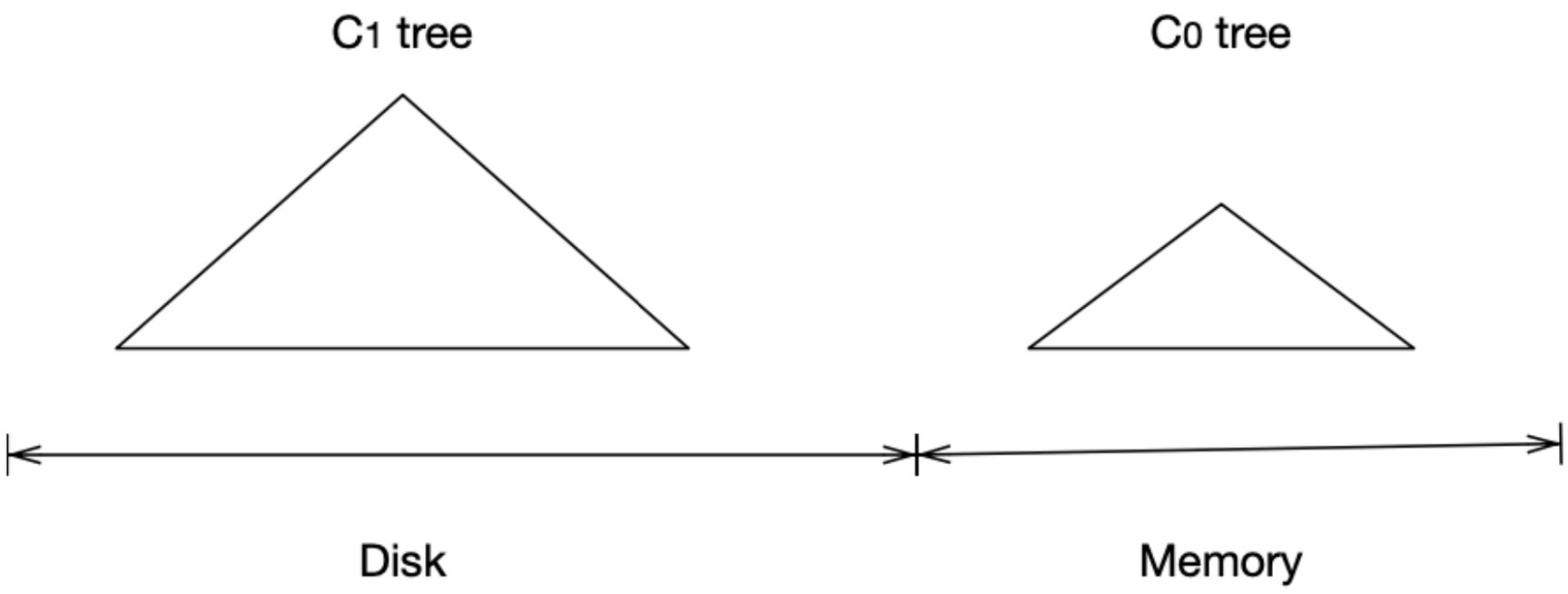
- FlashDB
  - Uses also logs for each tree node
  - Distinguishes between mainly read nodes (kept in Flash) and often-written nodes (kept in RAM)

B-tree for Flash

- In Page Logging B+ tree (IPL)
  - Colocate nodes and their logs in the same erase unit

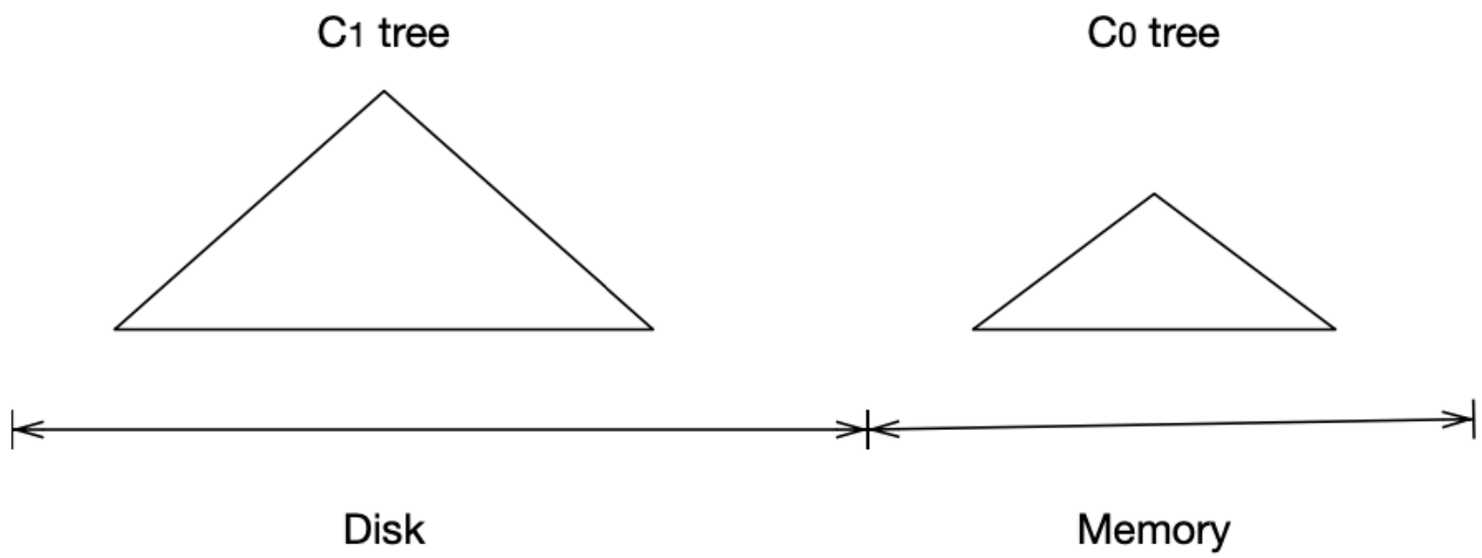
Log Structured Merge Trees

- Create two (or more trees)
  - Insert new records into the C0 tree
    - Keep C0 in memory
  - Log updates in storage



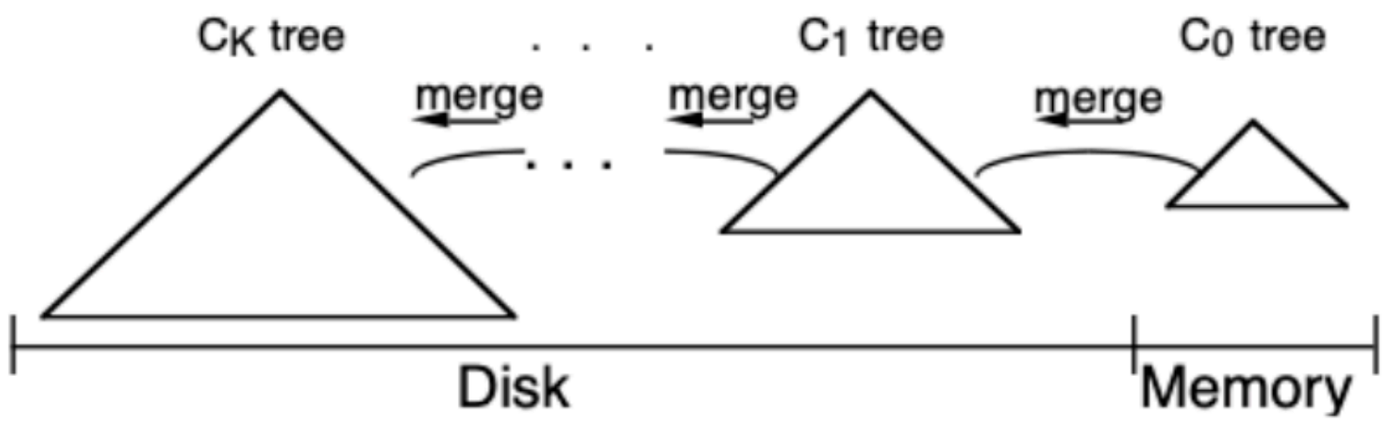
Log Structured Merge Trees

- Deletes:
  - Create tombstone records
- Seeks:
  - Seek both trees



Log Structured Merge Trees

- Whenever C0 becomes to big:
  - Create a new C0
  - Merge old C0 and C1
  - Both are now static



Log Structured Merge Trees

- To allow updates to values:
  - Use a version number