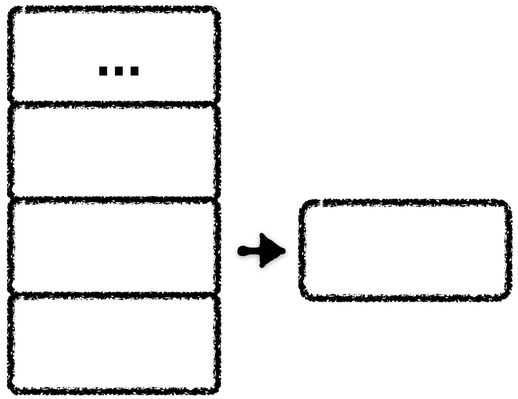
Tutorial on Concurrency & Consistency

CSC443H1 Database System Technology

Niv Dayan

Question 1 - Chained Hash Table Concurrency

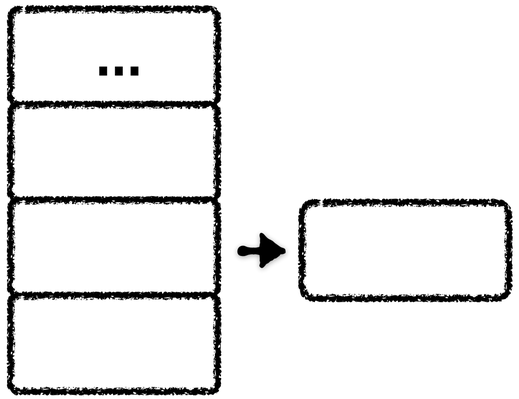
Consider a chained hash table in storage subject to gets, inserts and deletes. Each node stores B fixed-sized entries, and overflows are handled using chaining. Deletes traverse a chain, removing the entry when they find it. Insertions traverse the chain, inserting an entry into a “hole” or add another node if there are no holes. Whenever a node runs out of entries, it is deleted.



e.g., bucket with 2 nodes

Question 1 - Chained Hash Table Concurrency

Consider a chained hash table in storage subject to gets, inserts and deletes. Each node stores B fixed-sized entries, and overflows are handled using chaining. Deletes traverse a chain, removing the entry when they find it. Insertions traverse the chain, inserting an entry into a “hole” or add another node if there are no holes. Whenever a node runs out of entries, it is deleted.



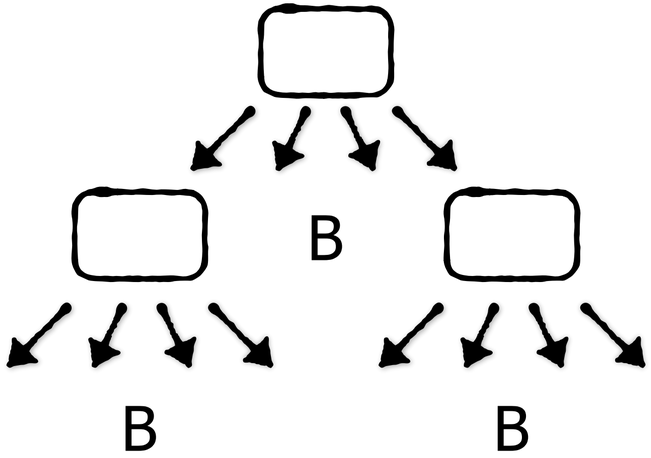
e.g., bucket with 2 nodes

(1) Is a chained hash table as susceptible as B-trees to locking contention? Why or why not? (2) Give an example where exclusively locking a whole bucket at a time for inserts/deletes can lead to contention? (3) How could you reduce this contention?

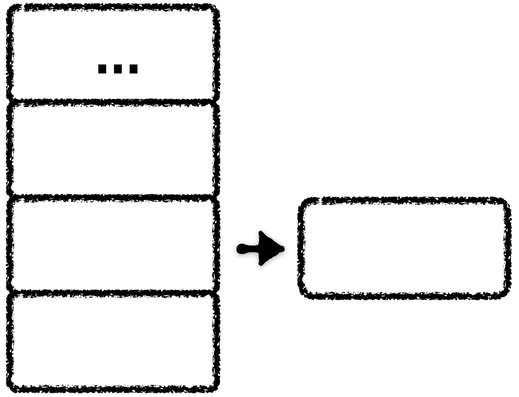
Question 1 - Chained Hash Table Concurrency

(1) Is a chained hash table as susceptible as B-trees to locking contention? Why or why not?

B-trees exhibit high contention at root and internal nodes since all accesses go through them.



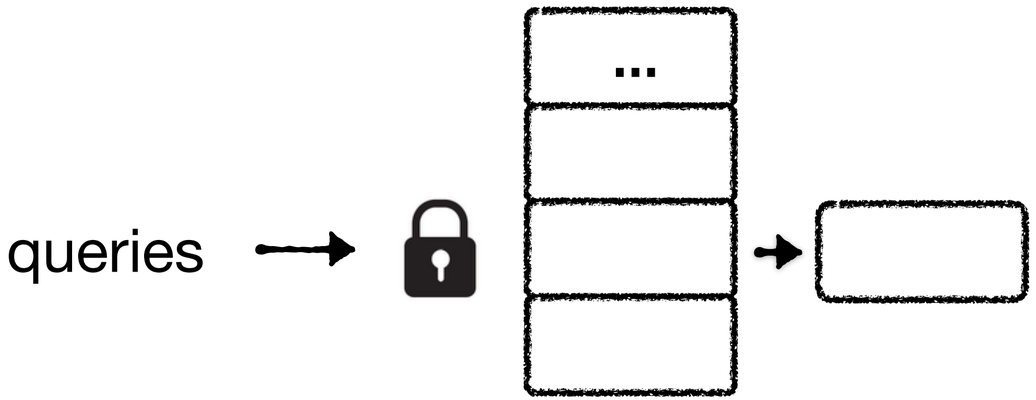
Hash tables have uniform contention across buckets, so there is less blocking



Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/deletes can lead to contention?

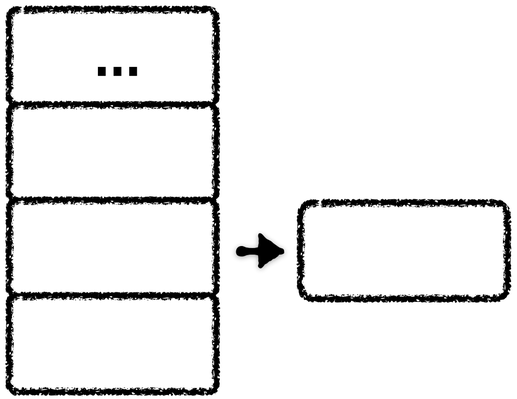
e.g., we have many insertions and deletions of the same key, so we exclusively lock the bucket over and over. All queries to other entries in the bucket are blocked.



Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/deletes can lead to contention?

(3) How could you reduce such bottlenecks?

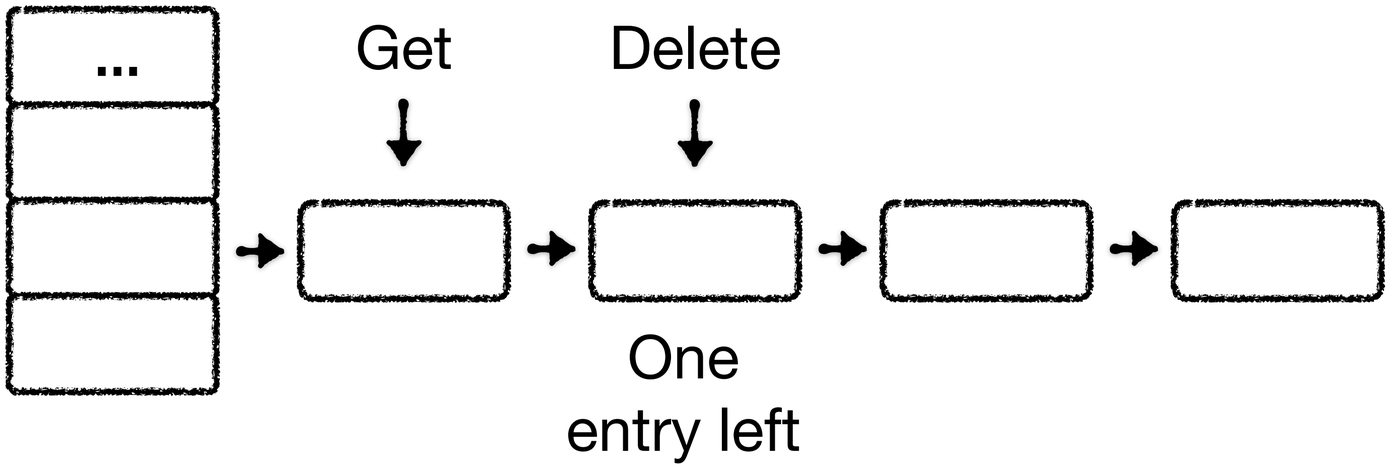


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/ deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(A) To prevent a node being deleted right before we read it, employ lock coupling with shared locks for inserts, deletes and queries for traversal of a bucket.

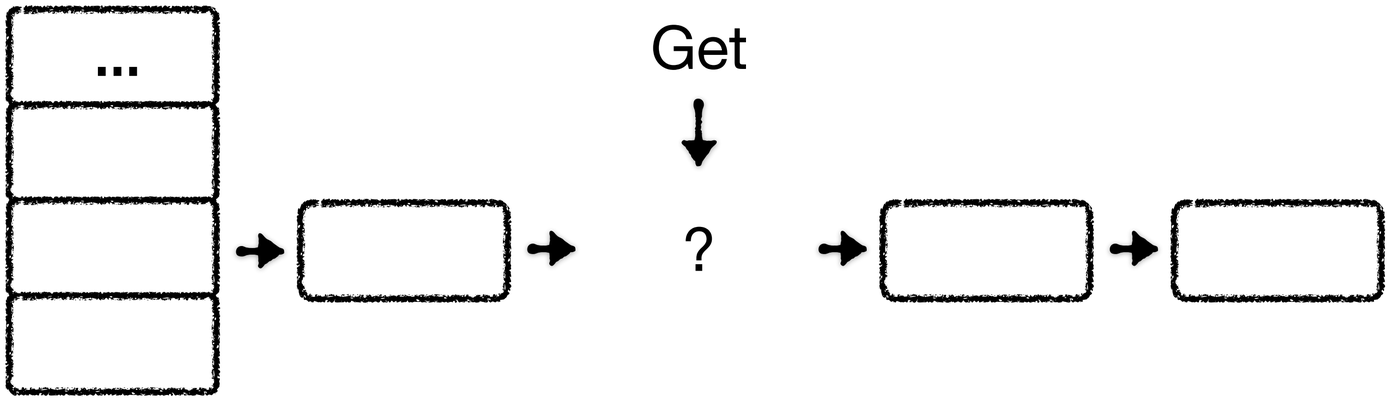


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/ deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(A) To prevent a node being deleted right before we read it, employ lock coupling with shared locks for inserts, deletes and queries for traversal of a bucket.

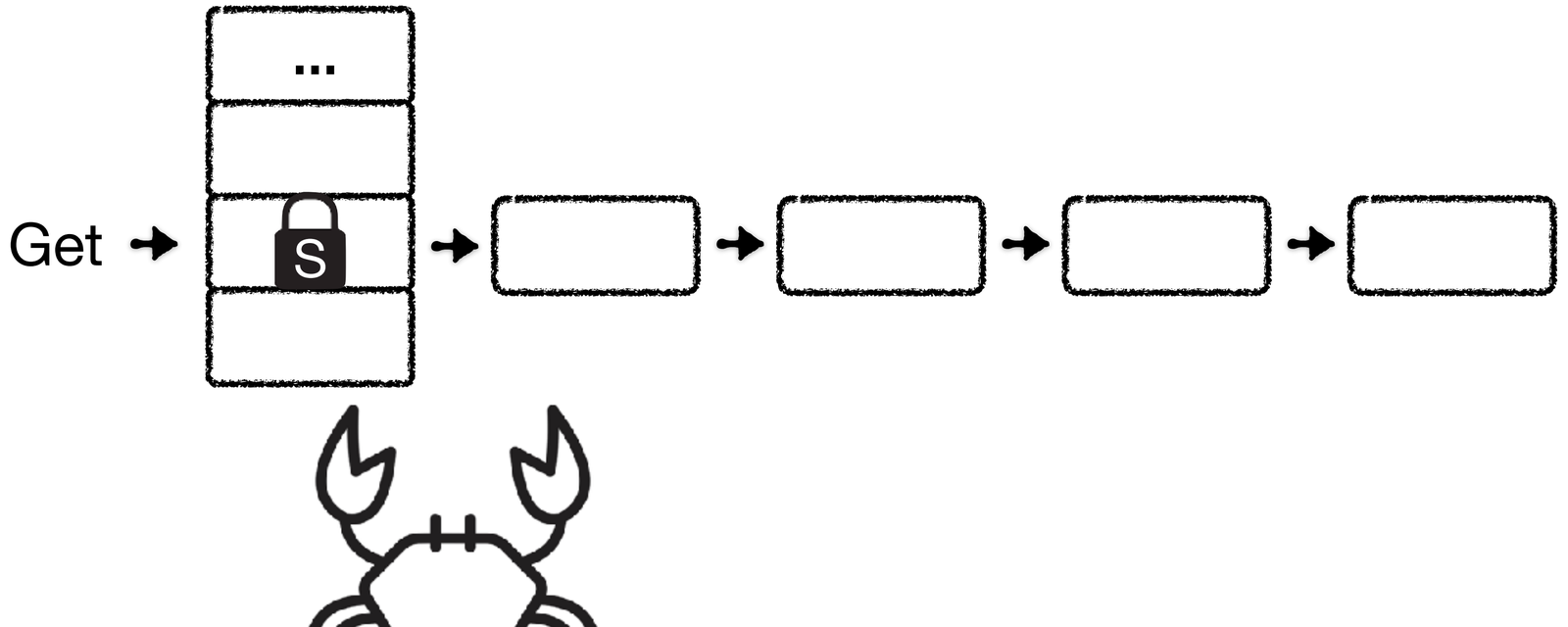


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/ deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(A) To prevent a node being deleted right before we read it, employ lock coupling with shared locks for inserts, deletes and queries for traversal of a bucket.

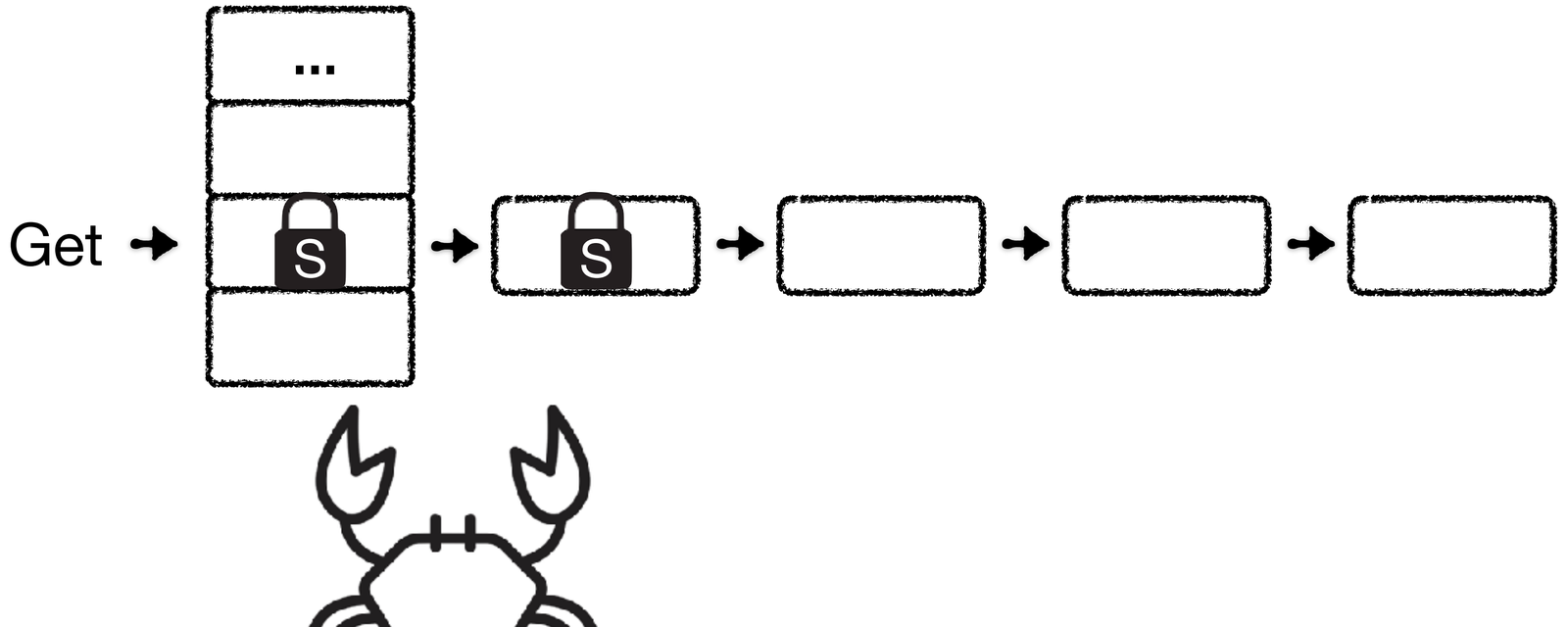


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/ deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(A) To prevent a node being deleted right before we read it, employ lock coupling with shared locks for inserts, deletes and queries for traversal of a bucket.

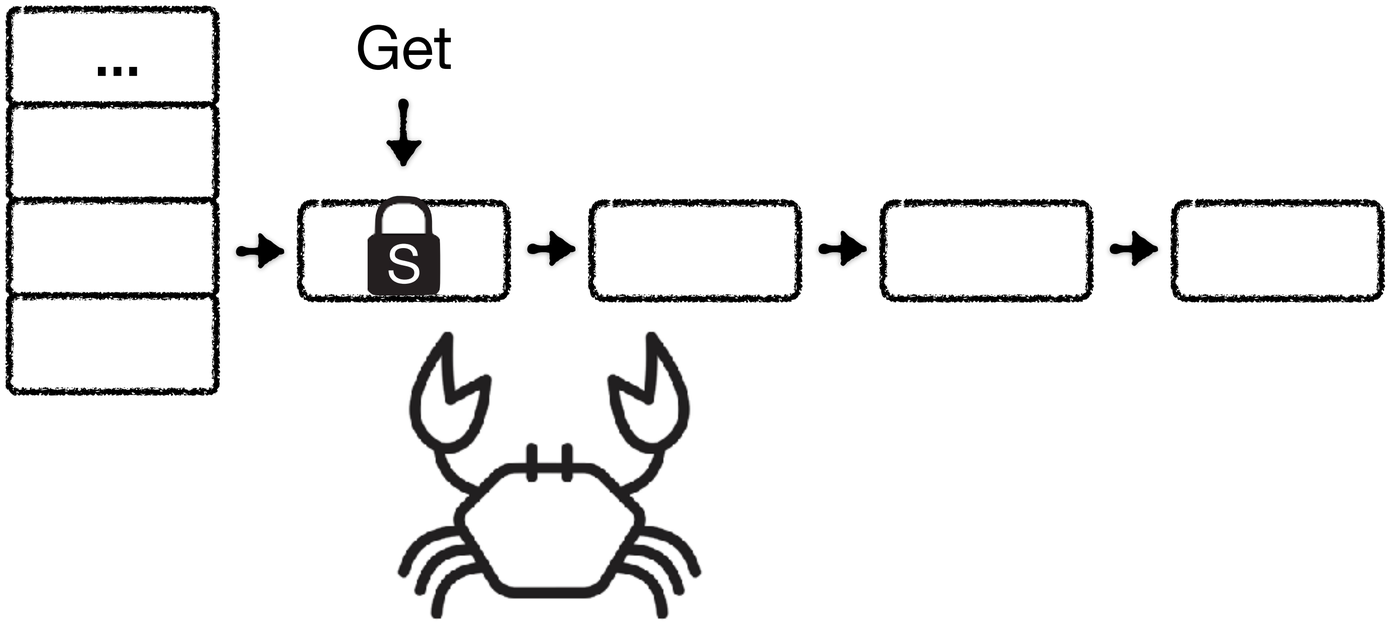


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(A) To prevent a node being deleted right before we read it, employ lock coupling with shared locks for inserts, deletes and queries for traversal of a bucket.

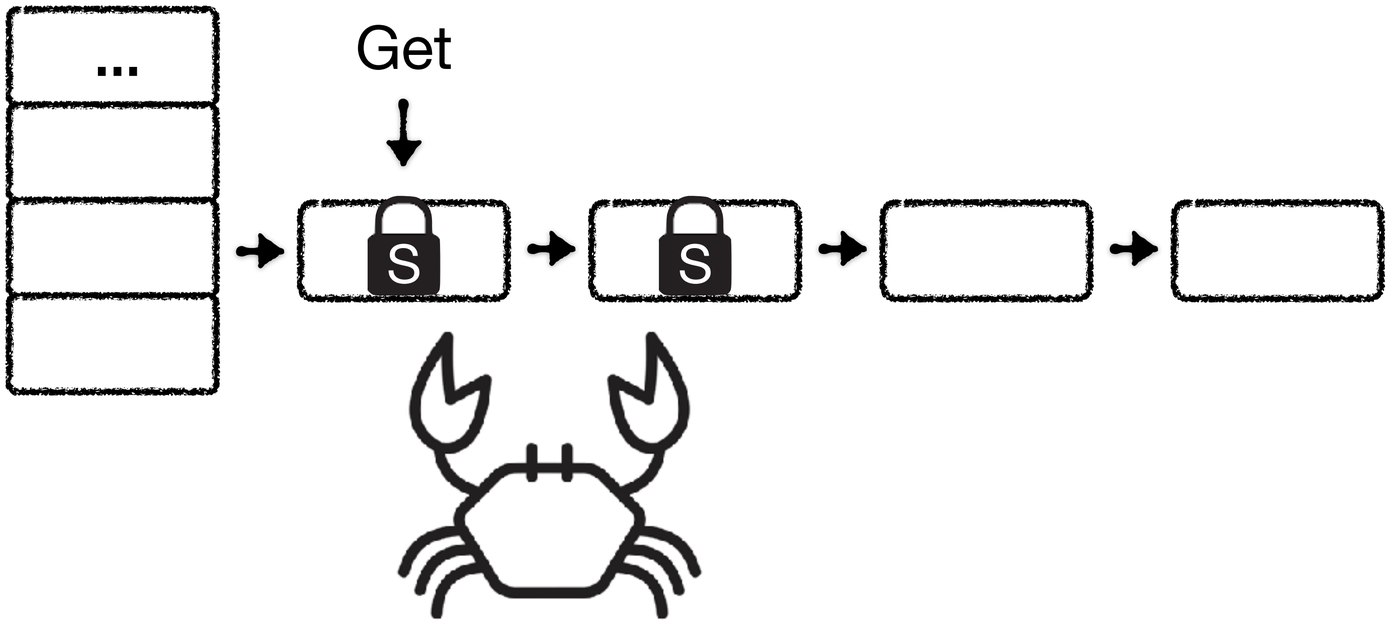


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/ deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(A) To prevent a node being deleted right before we read it, employ lock coupling with shared locks for inserts, deletes and queries for traversal of a bucket.

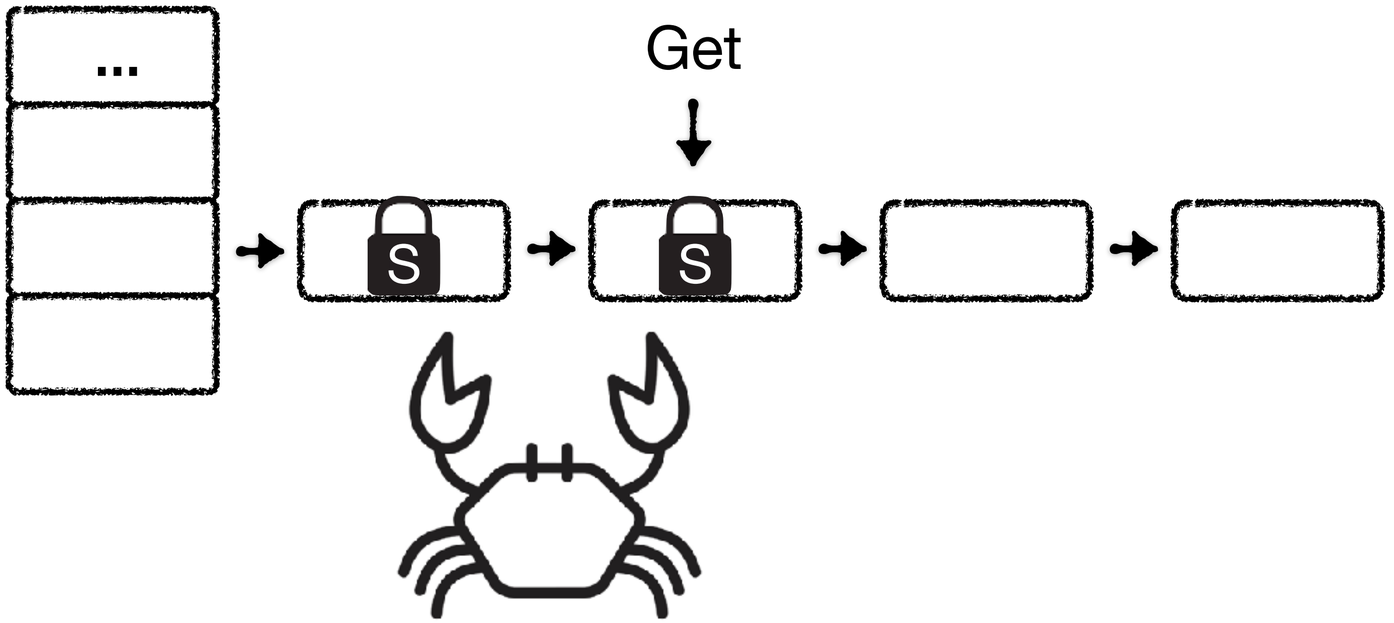


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/ deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(A) To prevent a node being deleted right before we read it, employ lock coupling with shared locks for inserts, deletes and queries for traversal of a bucket.

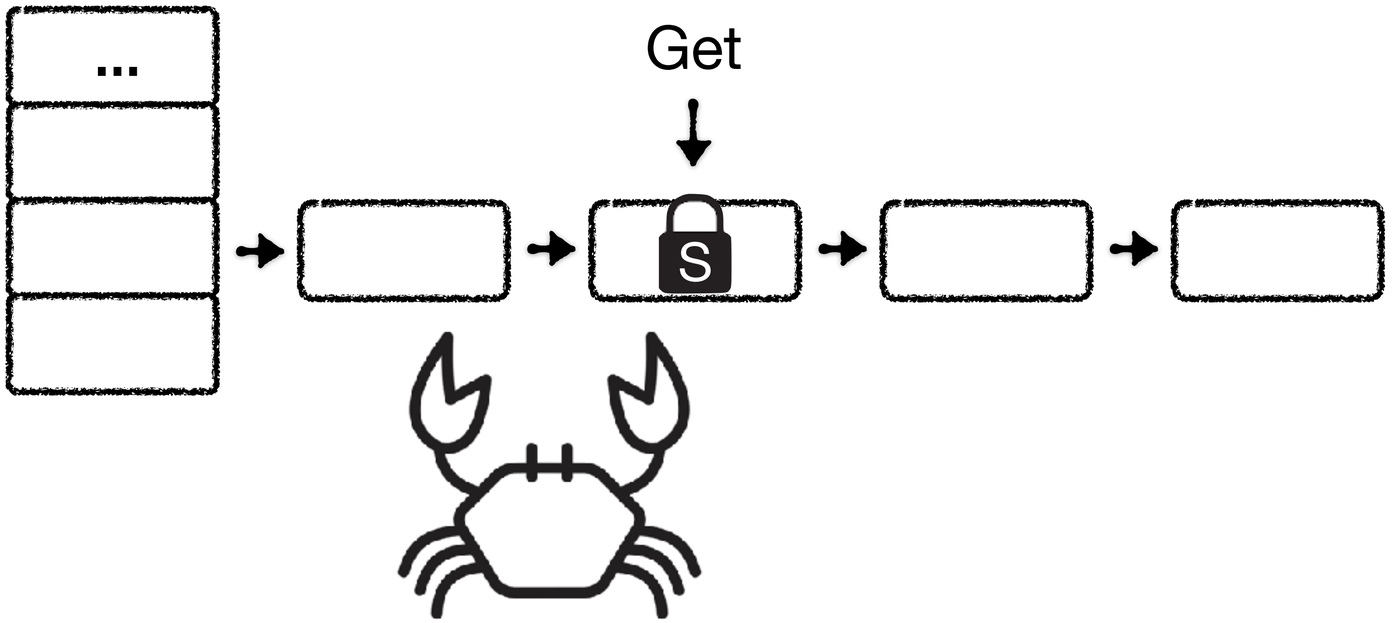


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(A) To prevent a node being deleted right before we read it, employ lock coupling with shared locks for inserts, deletes and queries for traversal of a bucket.

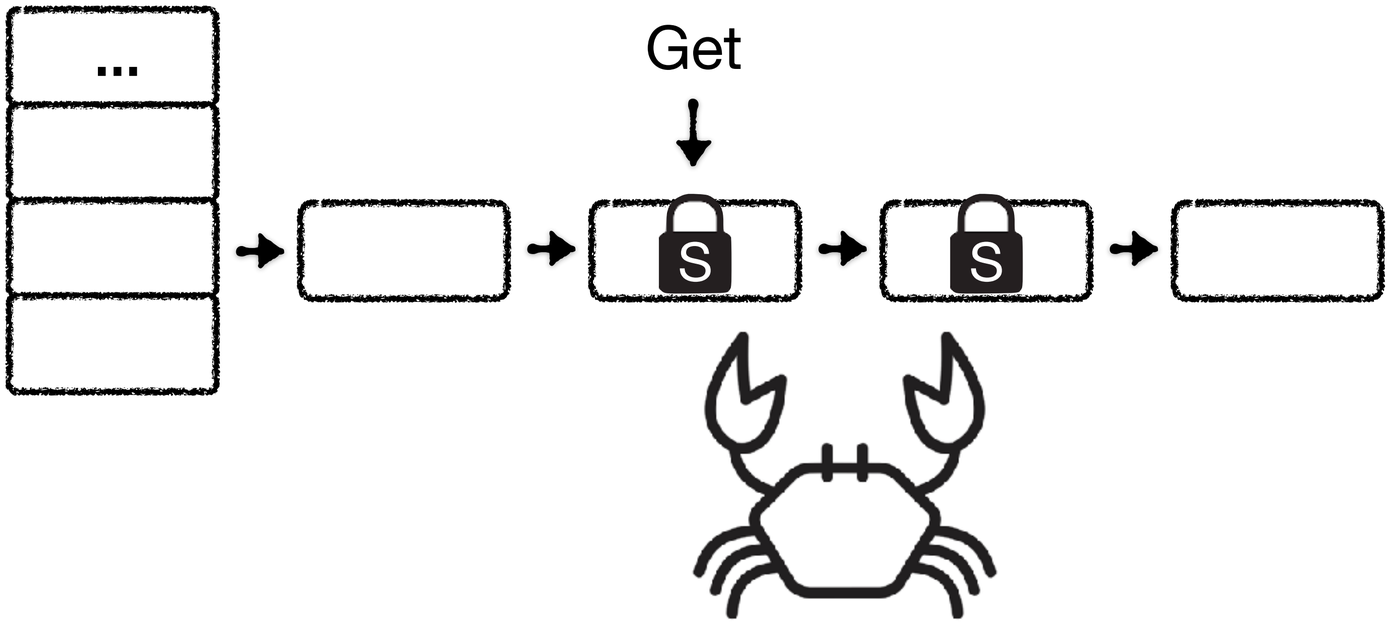


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/ deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(A) To prevent a node being deleted right before we read it, employ lock coupling with shared locks for inserts, deletes and queries for traversal of a bucket.

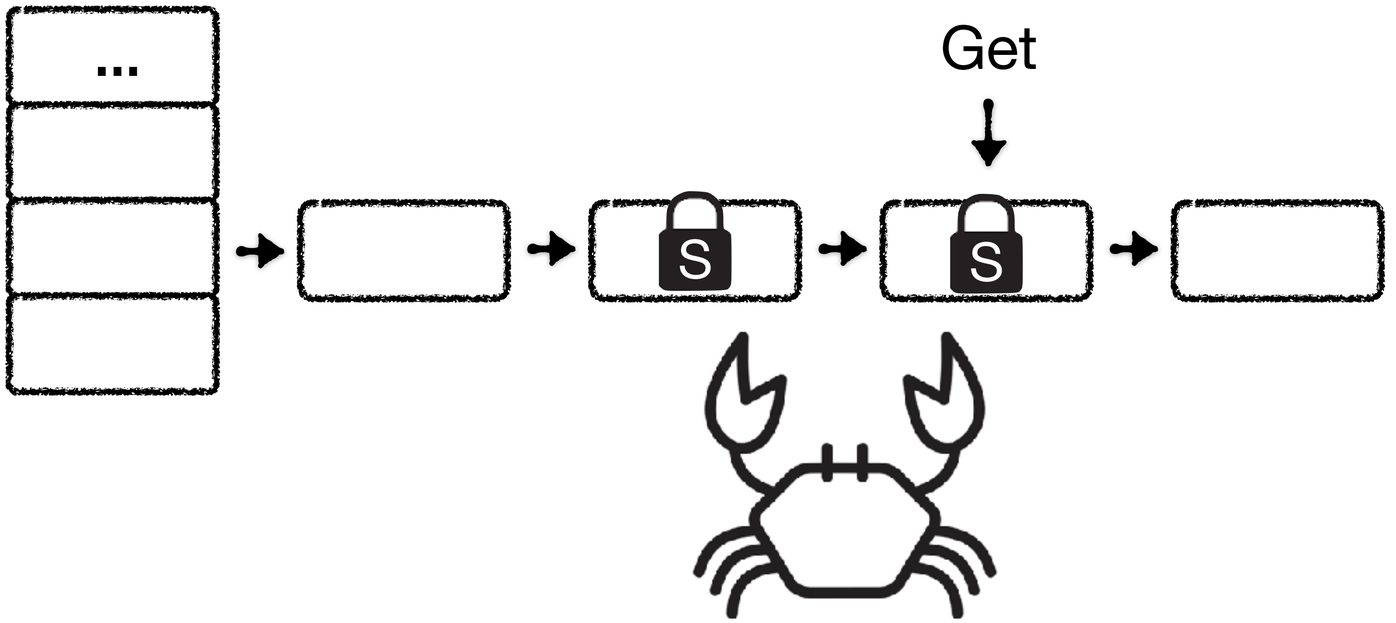


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/ deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(A) To prevent a node being deleted right before we read it, employ lock coupling with shared locks for inserts, deletes and queries for traversal of a bucket.

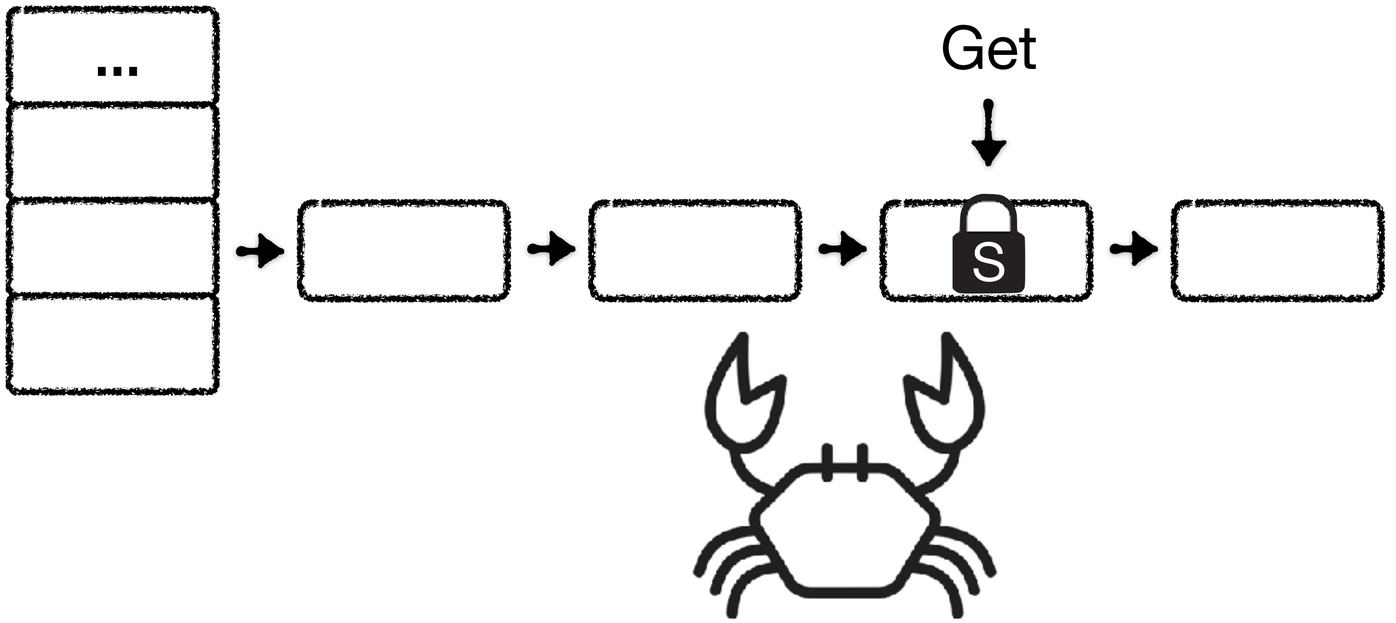


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/ deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(A) To prevent a node being deleted right before we read it, employ lock coupling with shared locks for inserts, deletes and queries for traversal of a bucket.

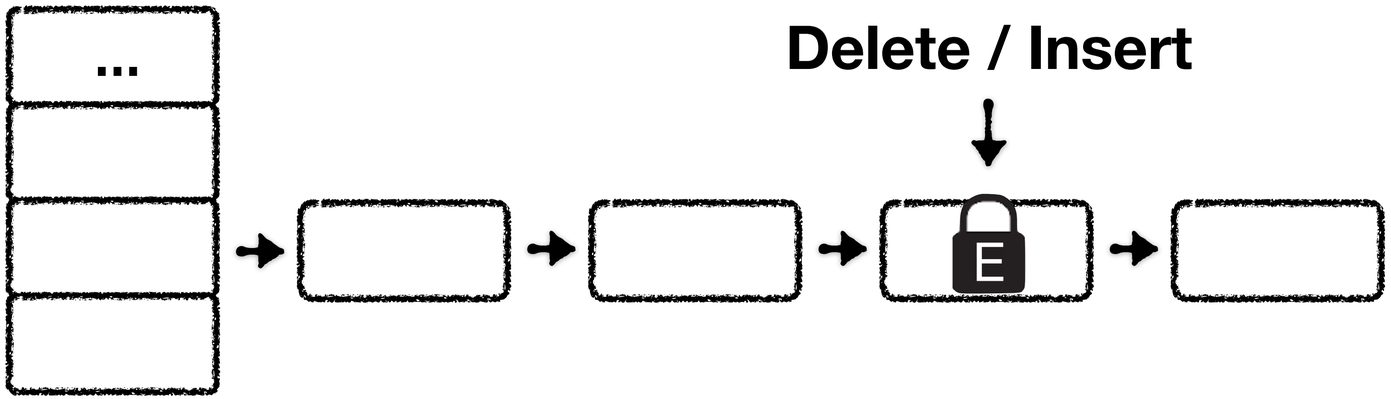


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(B) For delete/insert/update, upgrade the lock to exclusive

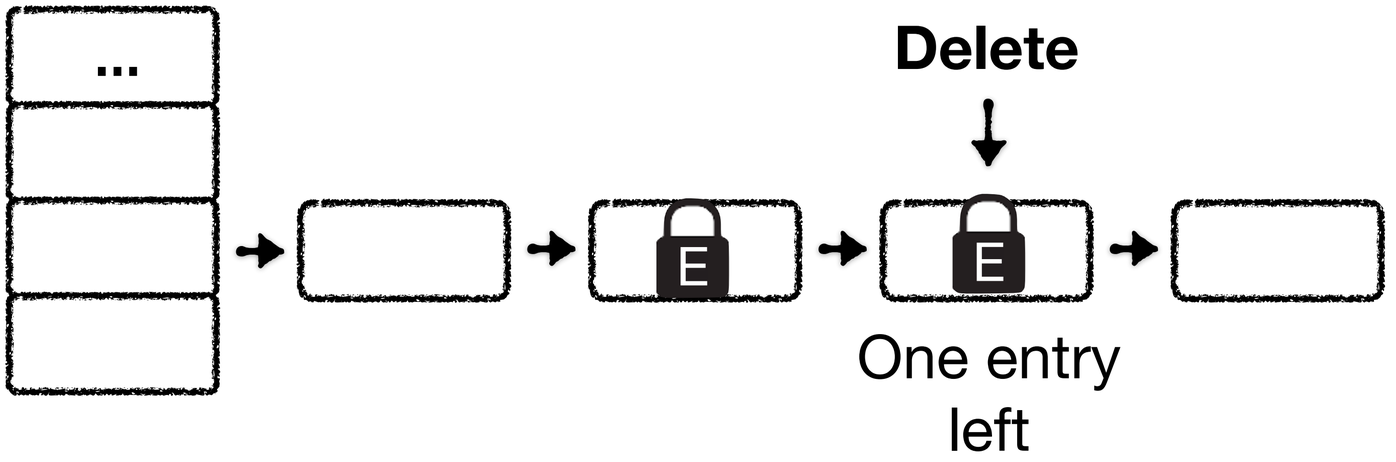


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/ deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(C) If we are deleting and next node contains only target key, keep holding lock on preceding node and upgrade to exclusive

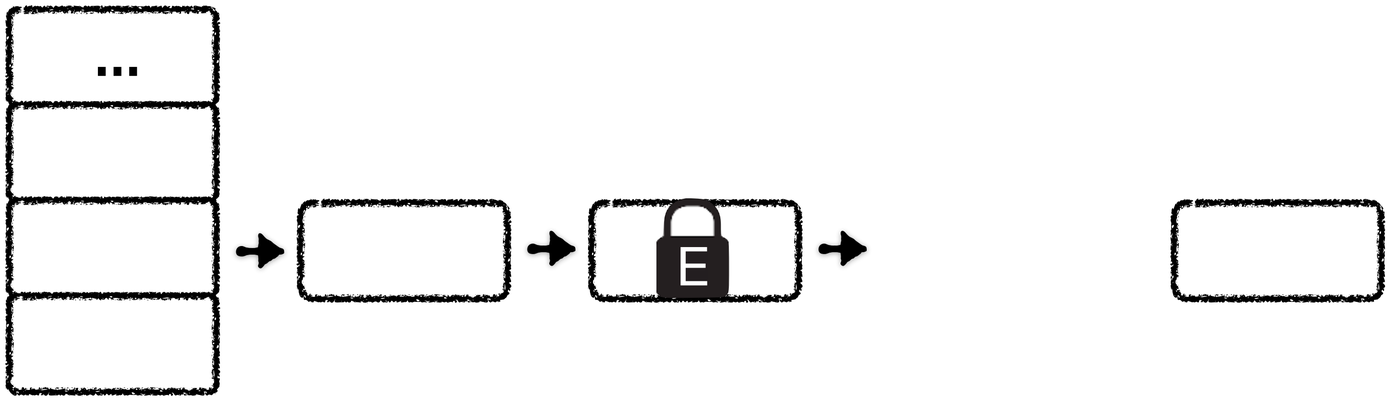


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(C) If we are deleting and next node contains only target key, keep holding lock on preceding node and upgrade to exclusive

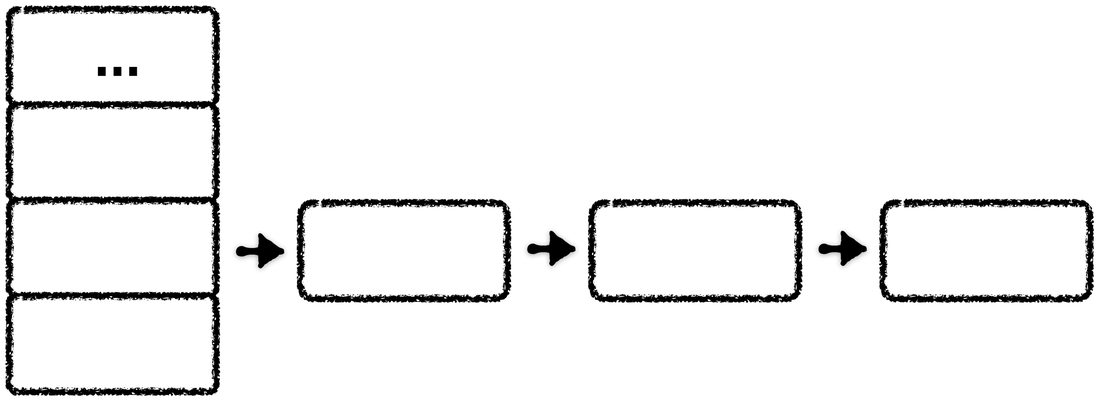


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(C) If we are deleting and next node contains only target key, keep holding lock on preceding node and upgrade to exclusive

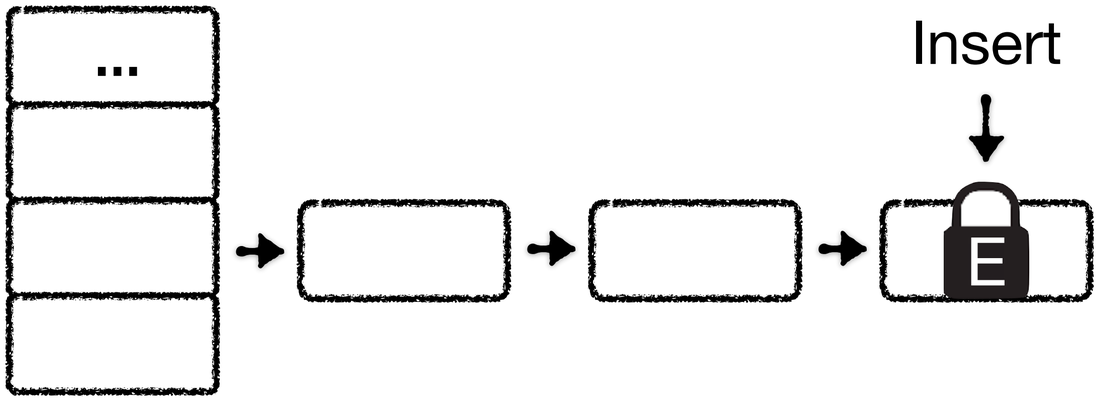


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(D) If we are inserting a new node at the end, hold exclusive lock on current last node until we extend chain

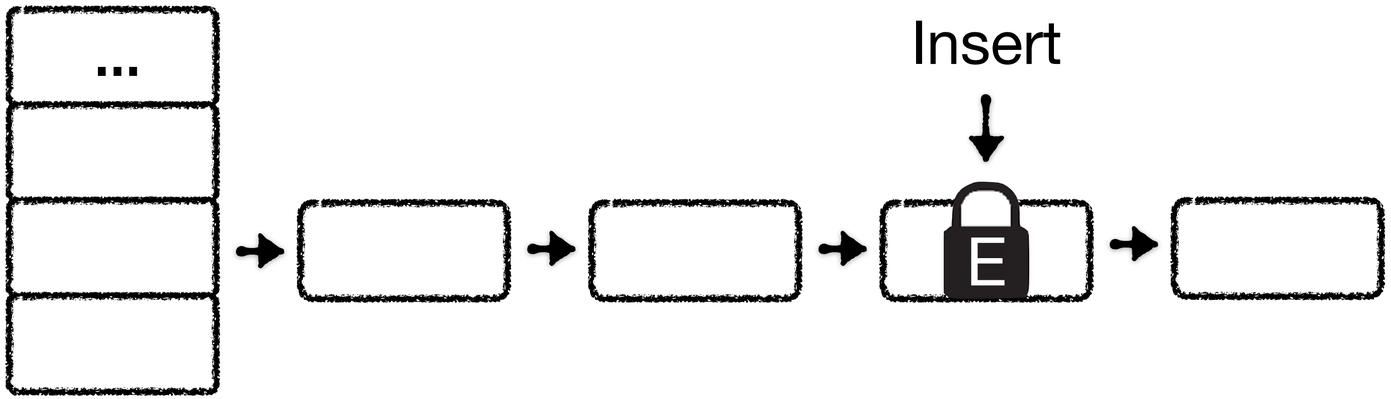


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(D) If we are inserting a new node at the end, hold exclusive lock on current last node until we extend chain

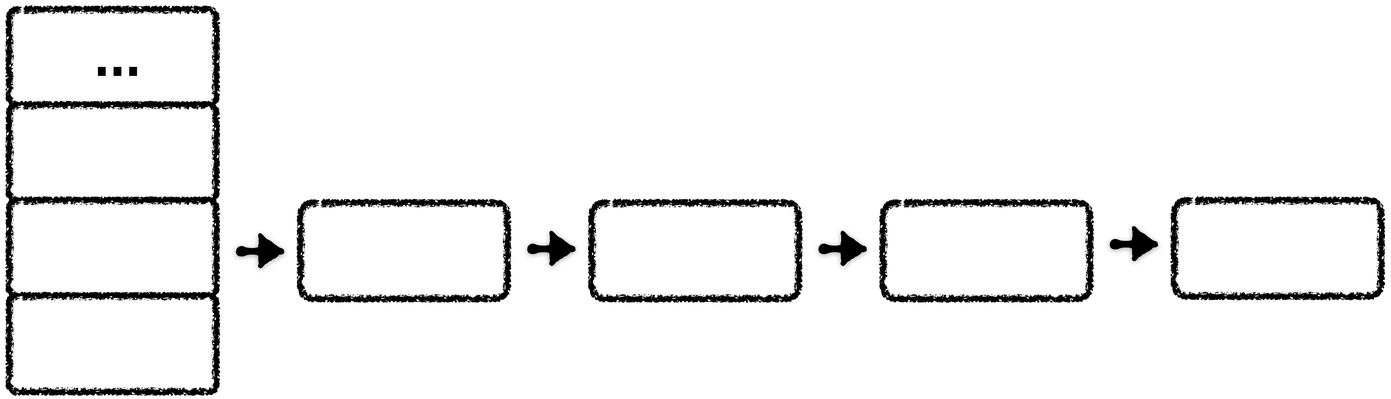


Question 1 - Chained Hash Table Concurrency

(2) Give an example where exclusively locking a whole bucket at a time for inserts/deletes can lead to contention?

(3) How could you reduce such bottlenecks?

(D) If we are inserting a new node at the end, hold exclusive lock on current last node until we extend chain

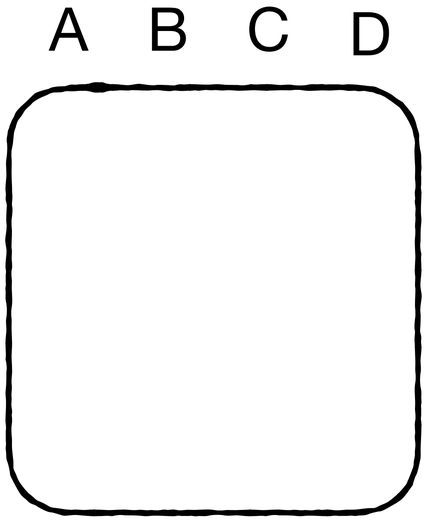


Question 2 - Predicate Locking

The locking mechanism we have seen in class lock physical objects: rows, pages or tables. Instead, it is also possible to lock predicates: “name = “bob” or “salary > 1000”.

(1) How would you implement a predicate lock manager?

(2) Compare the costs qualitatively to a physical lock manager.

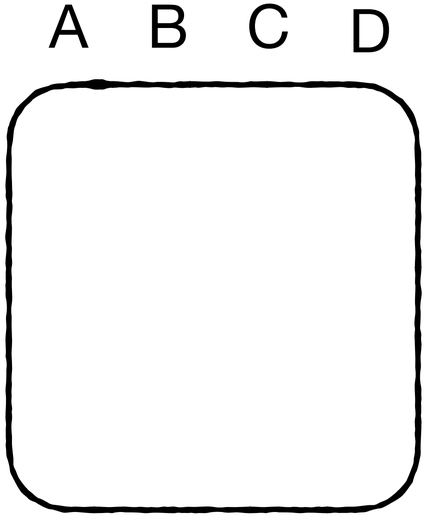


Example query:

Select * where (A = “a” and B=“b”) or C=“c” or D > “d”

Question 2 - Predicate Locking

(1) How would you implement a predicate lock manager?



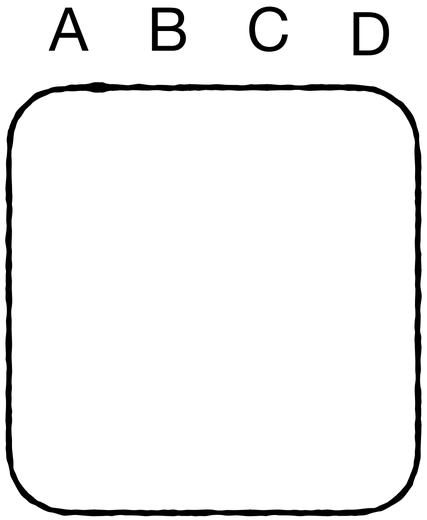
It seems excessive lock predicate individually: **Lock(A=“a”)** **Lock(B=“b”)**

Select * where (**A = “a” and B=“b”**) or C=“c” or D > “d”

Question 2 - Predicate Locking

(1) How would you implement a predicate lock manager?

Seems better to lock conjunction: Lock(A="a" and B="b")



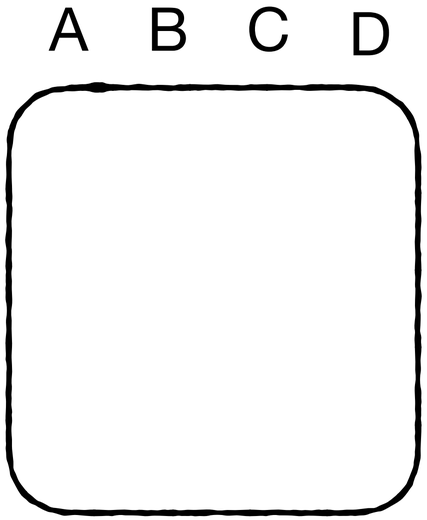
Select * where (**A = "a" and B="b"**) or C="c" or D > "d"

Question 2 - Predicate Locking

(1) How would you implement a predicate lock manager?

Can lock each conjunctions

Lock(A=“a” and B=“b”)
Lock(C=“c”)
Lock(D > ”d”)



Select * where (**A = “a” and B=“b”**) or **C=“c”** or **D > “d”**

Question 2 - Predicate Locking

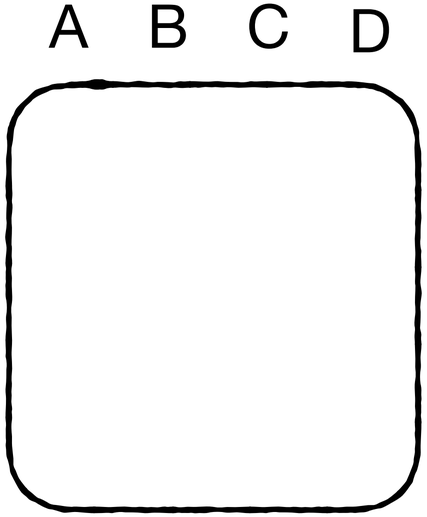
(1) How would you implement a predicate lock manager?

Can lock each conjunctions

Lock(A="a" and B="b")

Lock(C="c")

Lock(D > "d")



Select * where (A = "a" and B="b") or C="c" or **D > "d"**

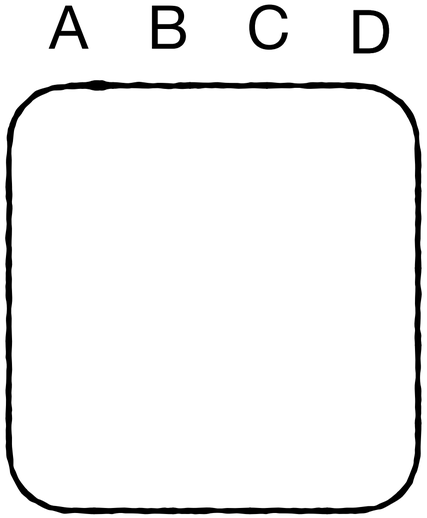
Range predicates require a tree for locking different ranges, so each lock is costlier to acquire

Question 2 - Predicate Locking

(1) How would you implement a predicate lock manager?

Can lock each conjunctions

Lock(A="a" and B="b")
Lock(C="c")
Lock(D > "d")

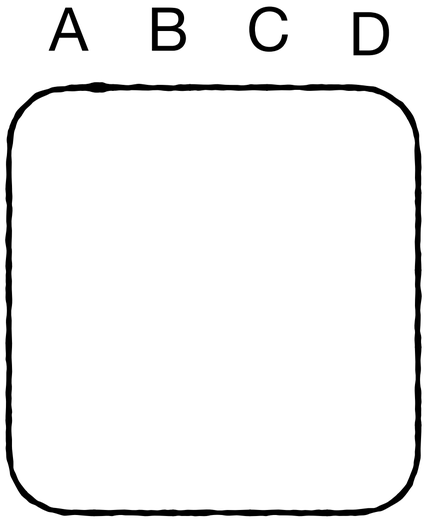


Select * where (**A = "a" and B="b"**) or C="c" or D > "d"

Consider sorting by column name to prevent having to lock each permutation (e.g., so B = "…" and A ="…" does not need a different lock)

Question 2 - Predicate Locking

(1) How would you implement a predicate lock manager?

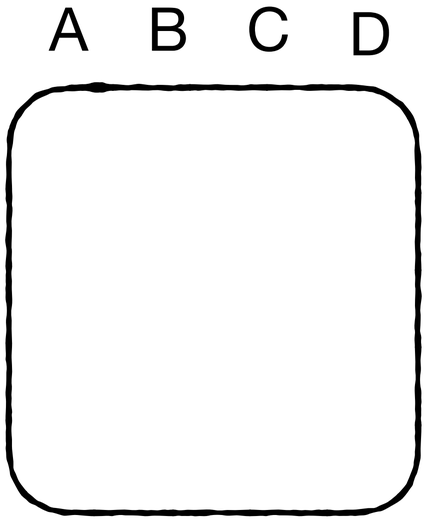


Select * where (A = “a” and B=“b”) or C=“c” or D > “d”

Suppose we now get: “update … A = “a” and C=“c”?

Question 2 - Predicate Locking

(1) How would you implement a predicate lock manager?



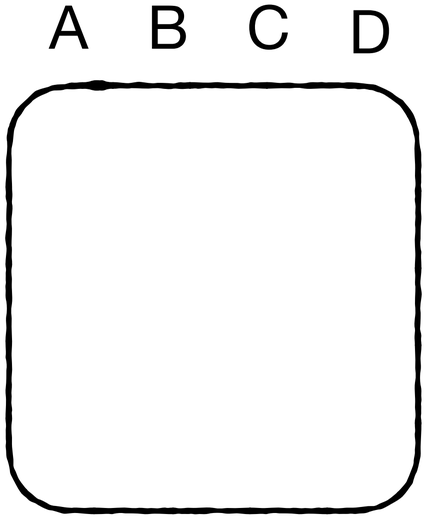
Select * where (A = “a” and B=“b”) or C=“c” or D > “d”

Suppose we now get: “update … A = “a” and C=“c”?

There is overlap between the queries, but if we check the predicate (A = “a” and C=“c”), it is unlocked.

Question 2 - Predicate Locking

(1) How would you implement a predicate lock manager?



Select * where (A = “a” and B=“b”) or C=“c” or D > “d”

Suppose we now get: “update … A = “a” and C=“c”?

There is overlap between the queries, but if we check the predicate (A = “a” and C=“c”), it is unlocked.

We could get unrepeatable read and phantoms anomalies

Question 2 - Predicate Locking

(2) Compare the costs qualitatively to a physical lock manager.

Question 2 - Predicate Locking

(2) Compare the costs qualitatively to a physical lock manager.

Pros of predicate locking

for queries returning a lot of objects, locking each predicate rather than object can be cheaper

Question 2 - Predicate Locking

(2) Compare the costs qualitatively to a physical lock manager.

Cons of predicate locking

Locking as little as possible while while being able to compare the locking coverage of different queries is highly non-trivial.

E.g., lock as little as possible with ensuring no overlap between these queries

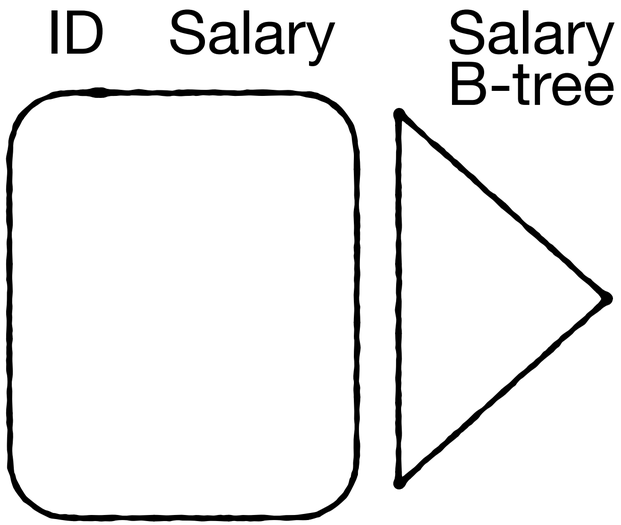
Select * where (A = “a” and B=“b”) or C=“c” or D > ”d”

Update/insert … A = “a” and C=“c”

Question 3 - The Halloween Problem

An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?

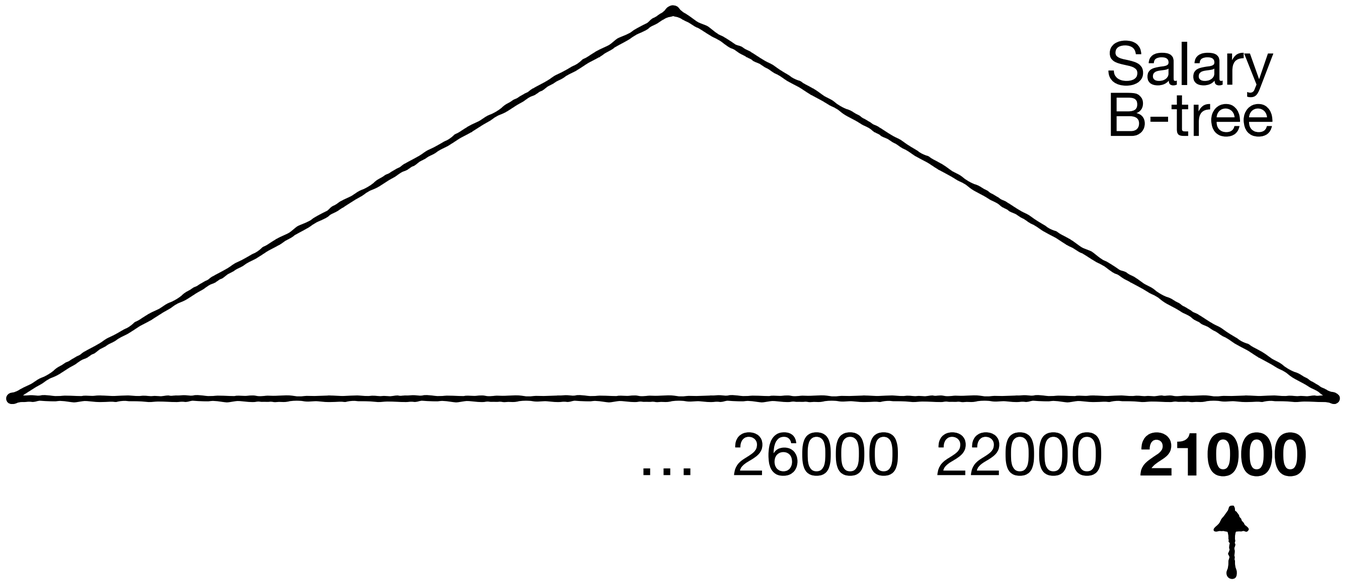
```
UPDATE employees SET salary = salary * 1.1 WHERE salary < 25000;
```



Question 3 - The Halloween Problem

An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?

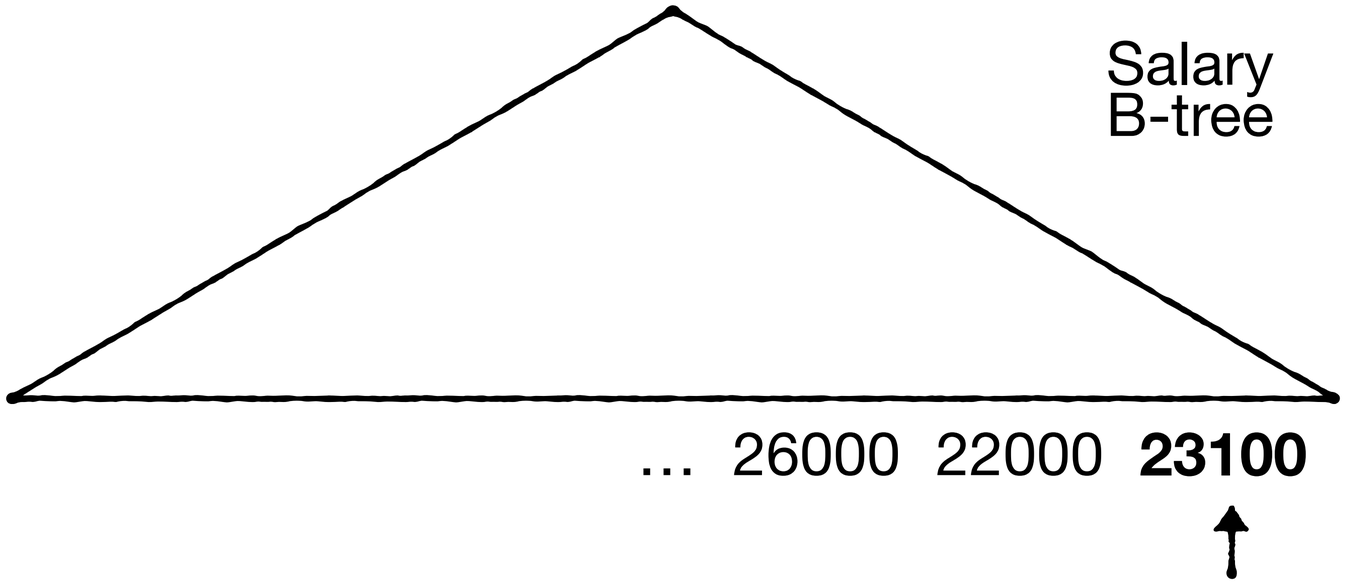
```
UPDATE employees SET salary = salary * 1.1 WHERE salary < 25000;
```



Question 3 - The Halloween Problem

An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?

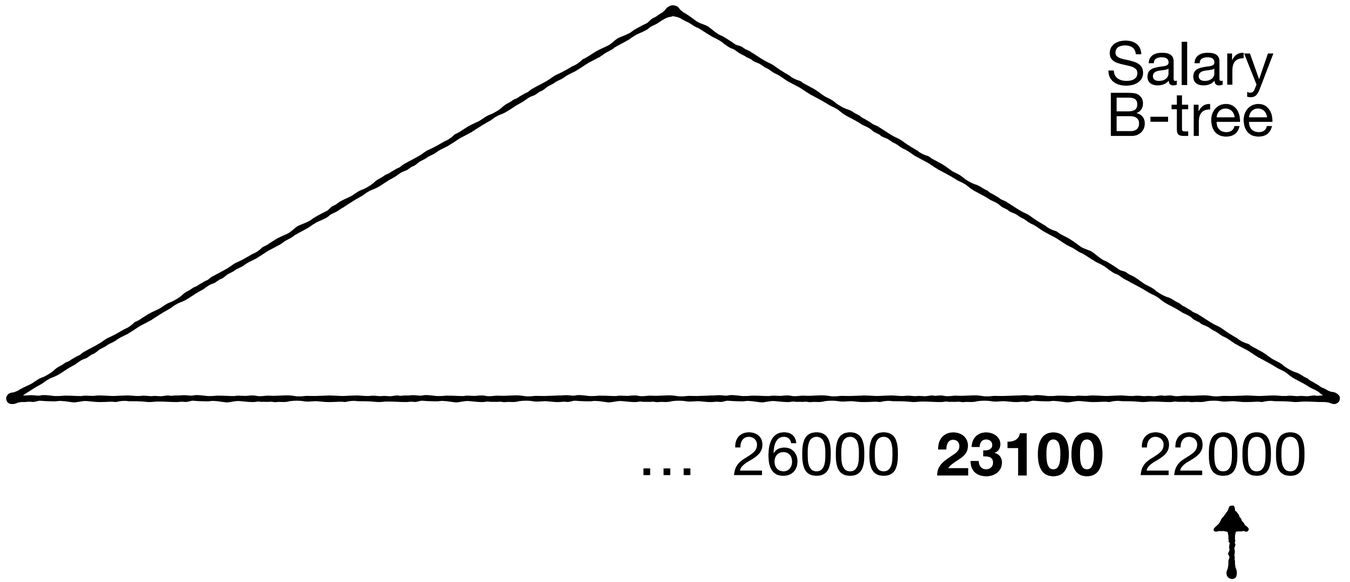
```
UPDATE employees SET salary = salary * 1.1 WHERE salary < 25000;
```



Question 3 - The Halloween Problem

An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?

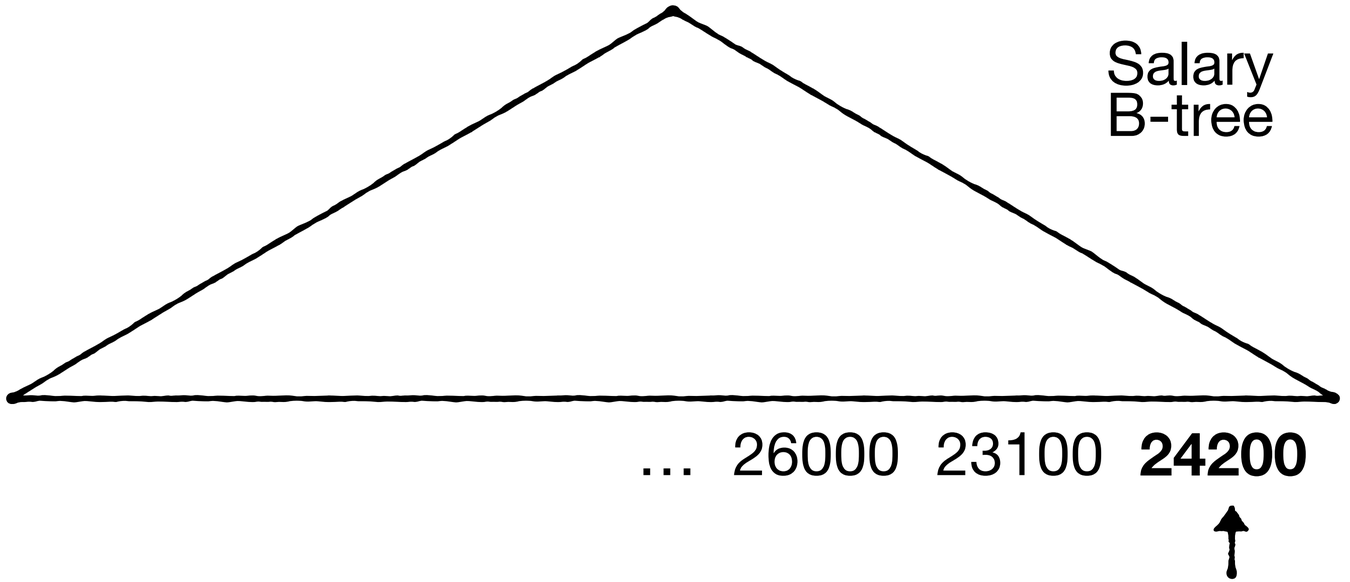
```
UPDATE employees SET salary = salary * 1.1 WHERE salary < 25000;
```



Question 3 - The Halloween Problem

An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?

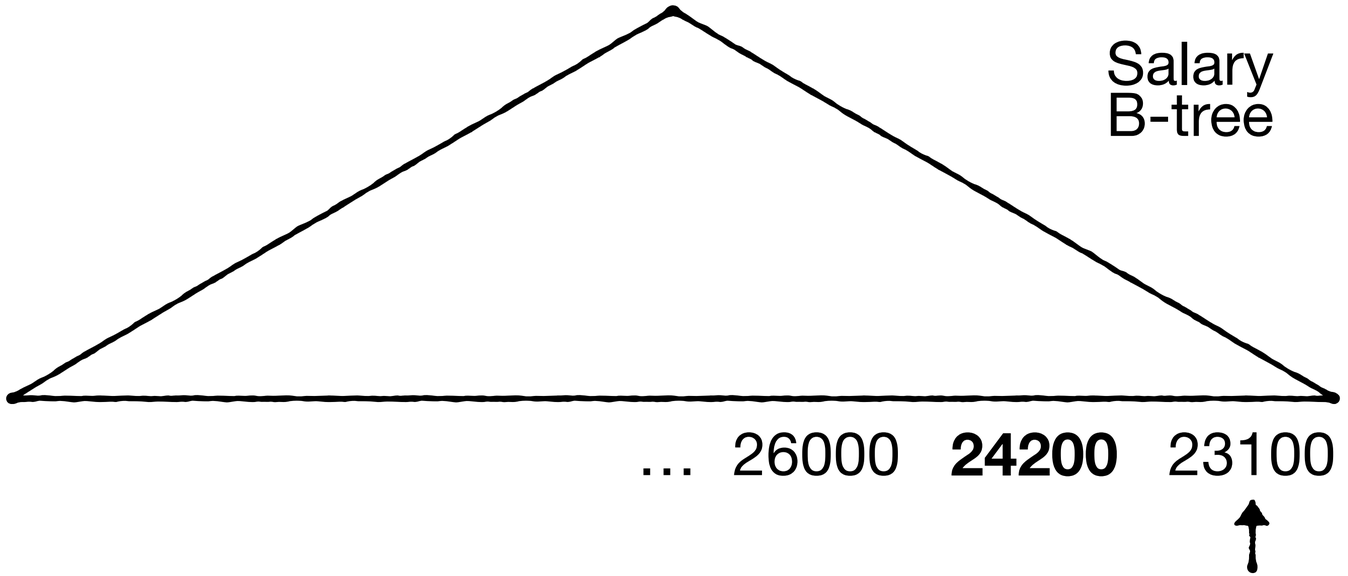
```
UPDATE employees SET salary = salary * 1.1 WHERE salary < 25000;
```



Question 3 - The Halloween Problem

An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?

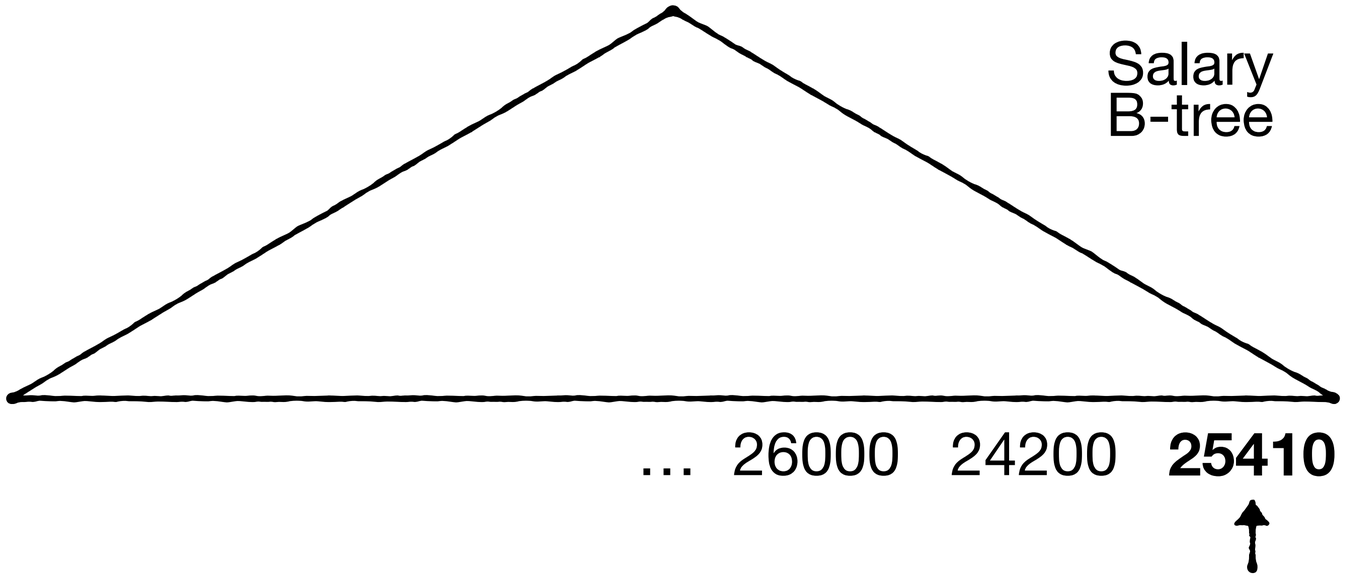
```
UPDATE employees SET salary = salary * 1.1 WHERE salary < 25000;
```



Question 3 - The Halloween Problem

An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?

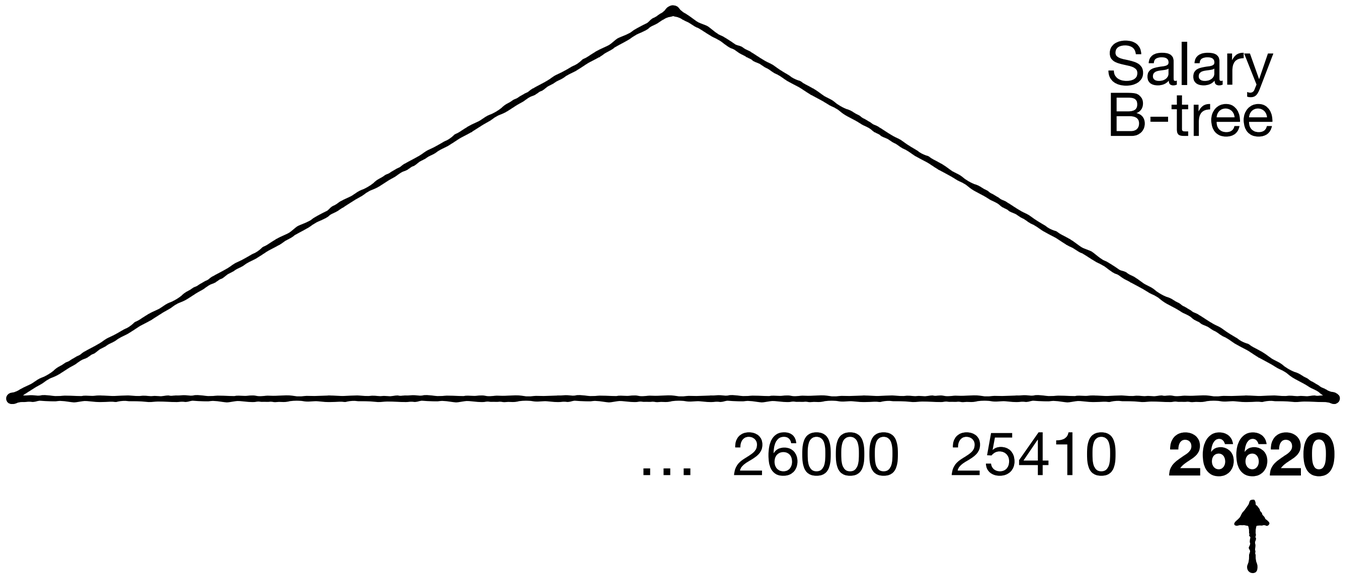
```
UPDATE employees SET salary = salary * 1.1 WHERE salary < 25000;
```



Question 3 - The Halloween Problem

An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?

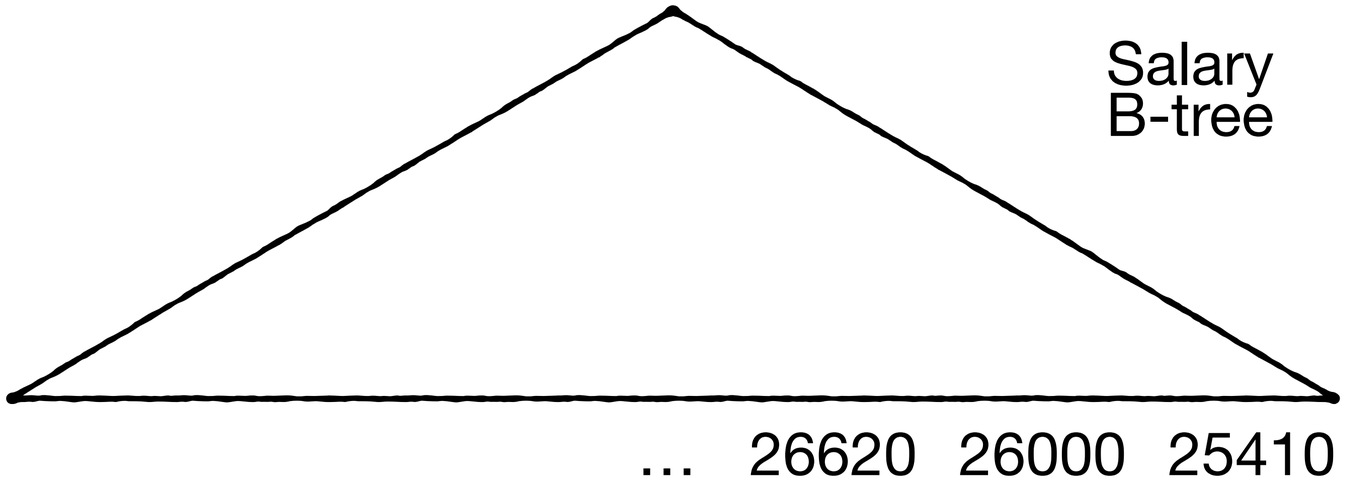
```
UPDATE employees SET salary = salary * 1.1 WHERE salary < 25000;
```



Question 3 - The Halloween Problem

An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?

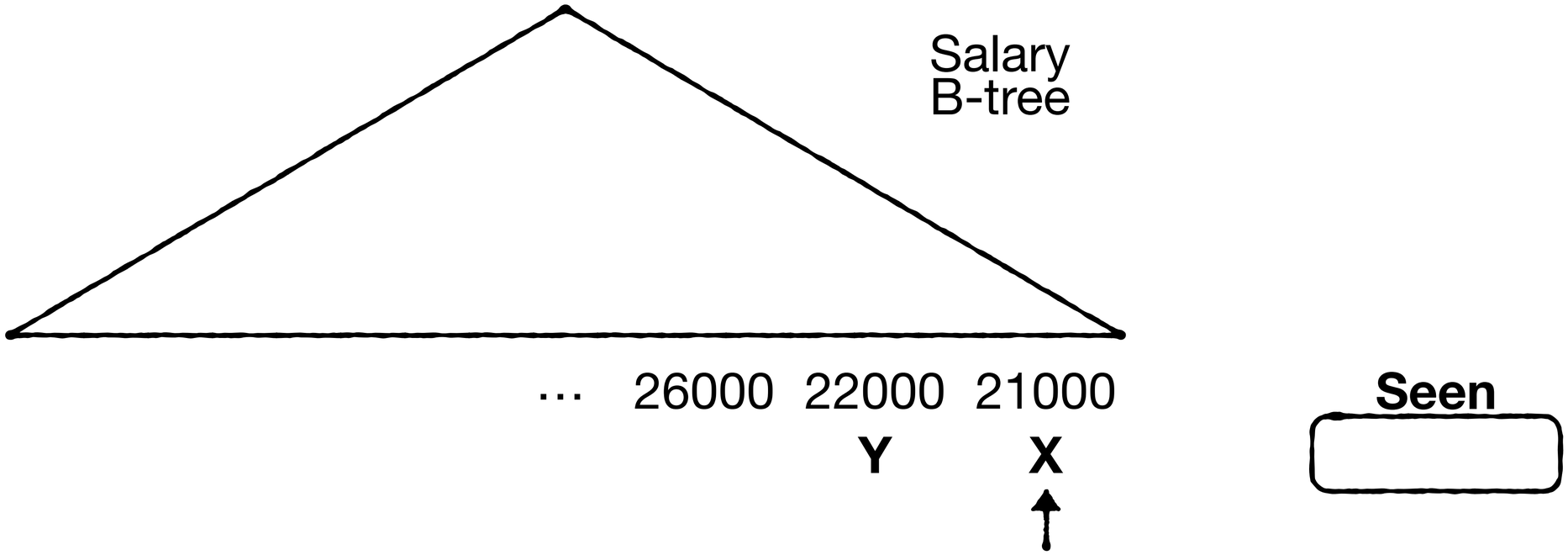
```
UPDATE employees SET salary = salary * 1.1 WHERE salary < 25000;
```



All salaries are now > 25000! This was not the intention :) Consistency is violated.

Question 3 - The Halloween Problem

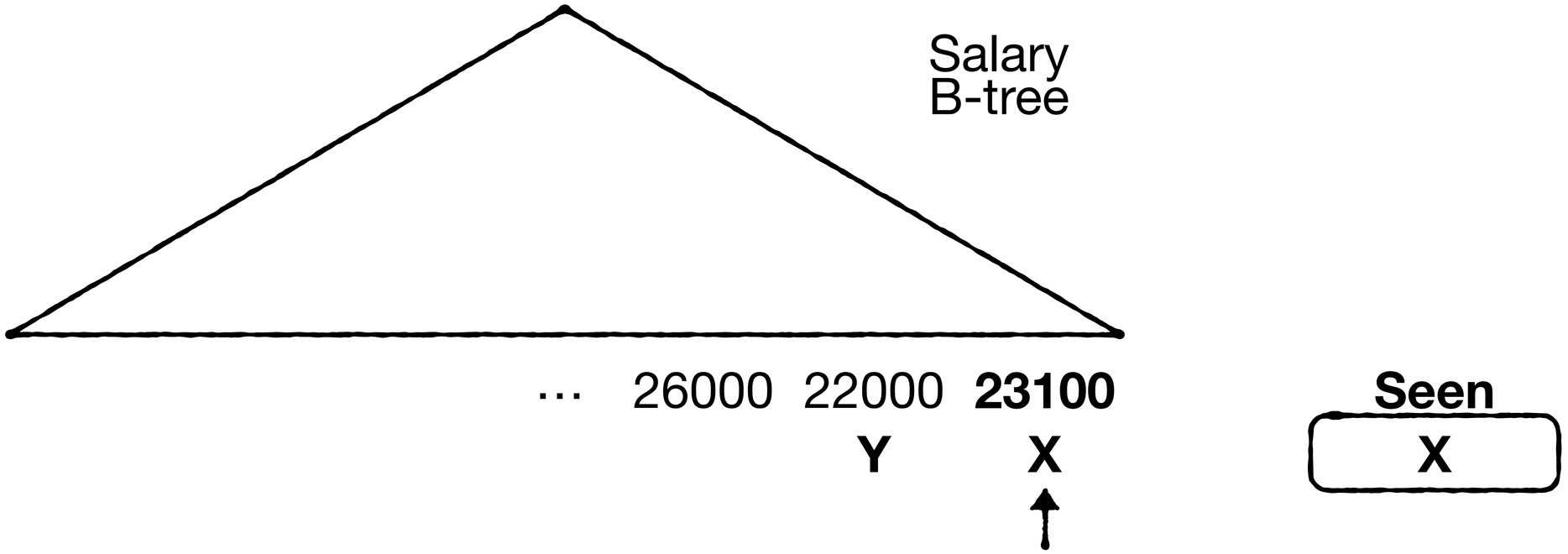
An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?



To solve this, keep track of row IDs that had been modified to ensure they are not modified again

Question 3 - The Halloween Problem

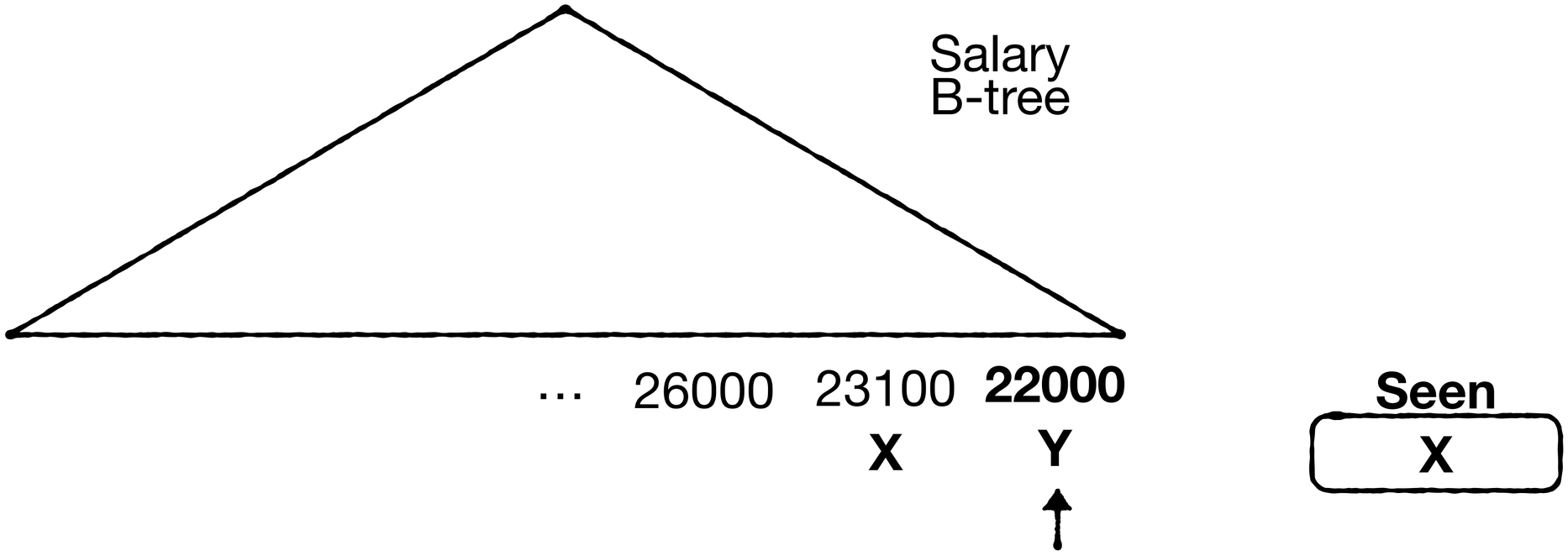
An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?



To solve this, keep track of row IDs that had been modified to ensure they are not modified again

Question 3 - The Halloween Problem

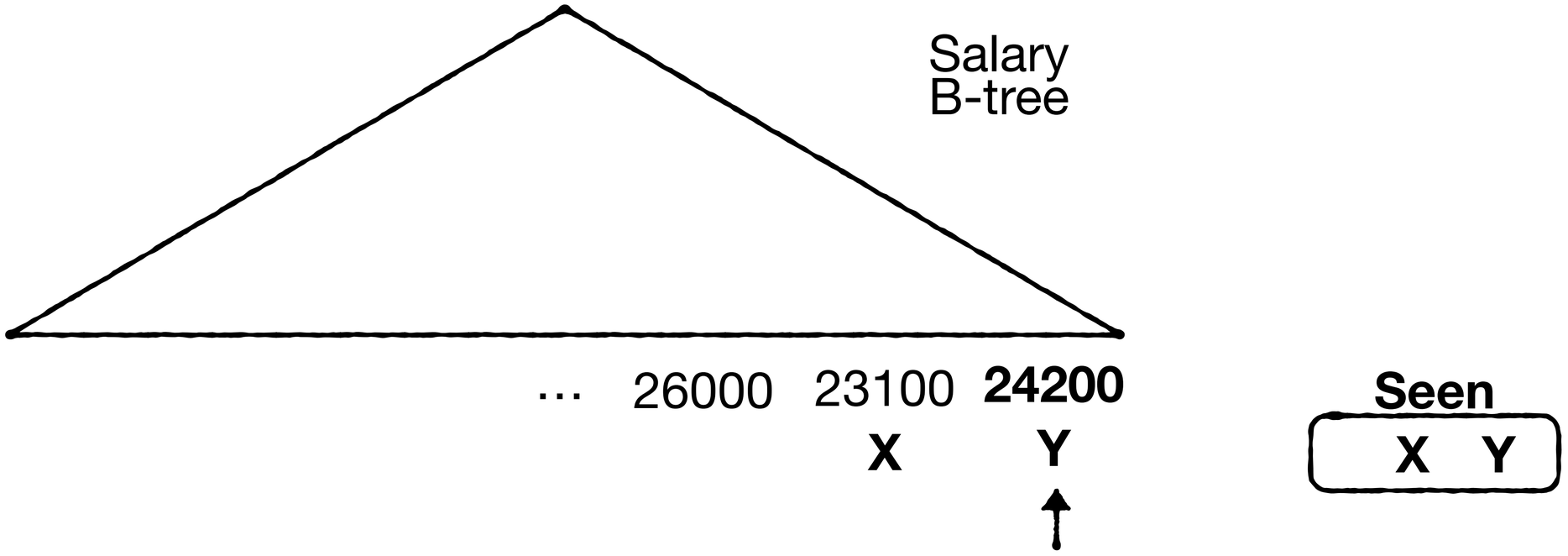
An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?



To solve this, keep track of row IDs that had been modified to ensure they are not modified again

Question 3 - The Halloween Problem

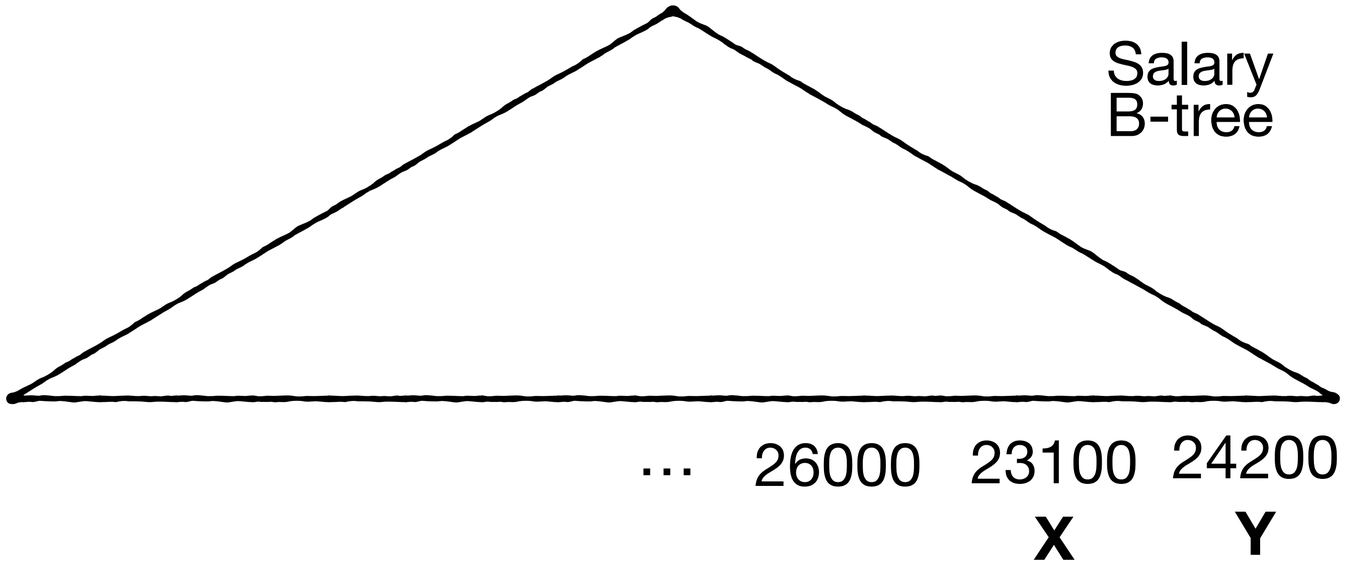
An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?



To solve this, keep track of row IDs that had been modified to ensure they are not modified again

Question 3 - The Halloween Problem

An update is designed to increase the salary of all employees who earn less than $25000 by 10%. What can go wrong when running this query, and how can we fix it?



To solve this, keep track of row IDs that had been modified to ensure they are not modified again