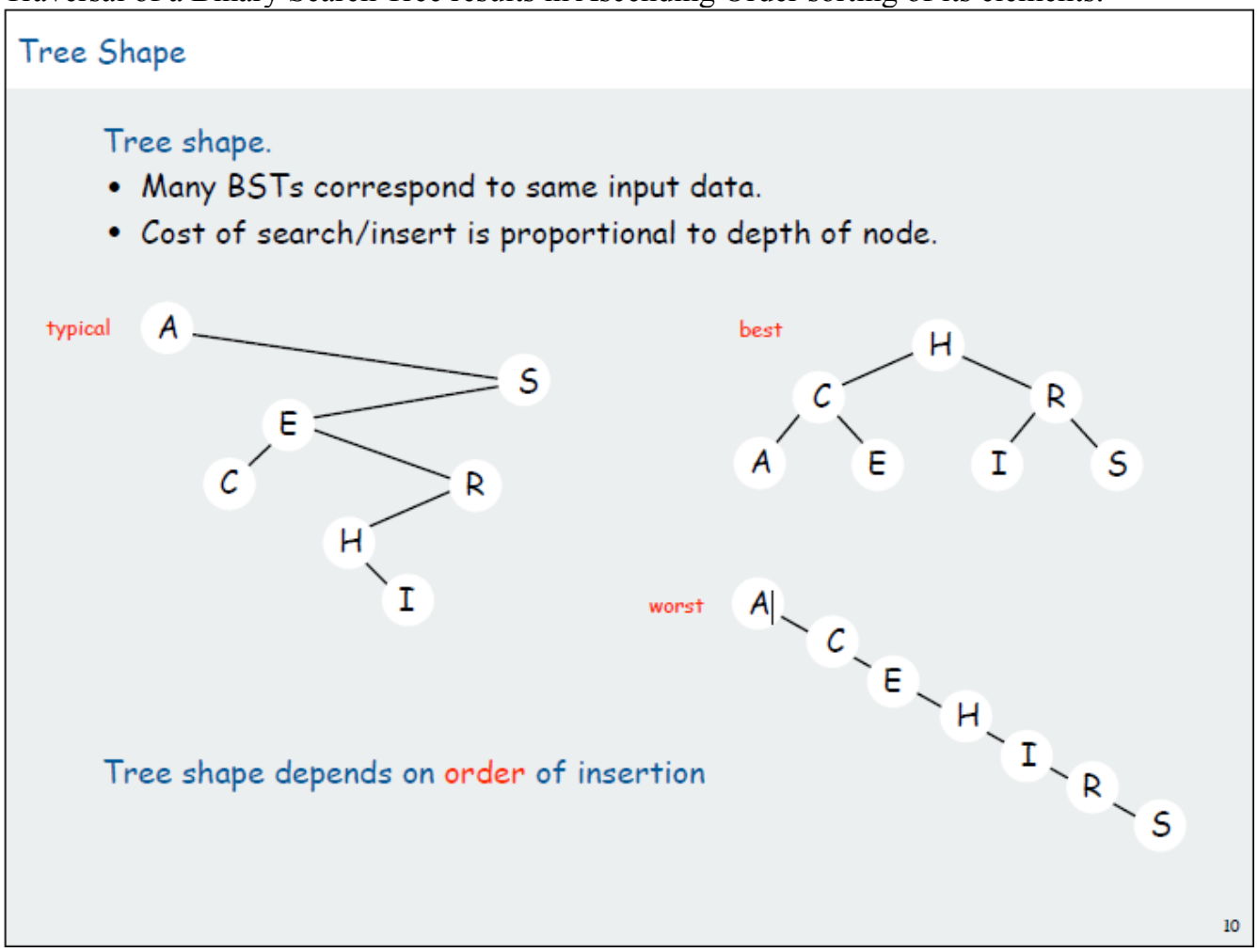
What is a Binary Tree? Why and where is this used in Java ?

Binary Tree is a tree data structure made up of nodes. Each node has utmost two children.

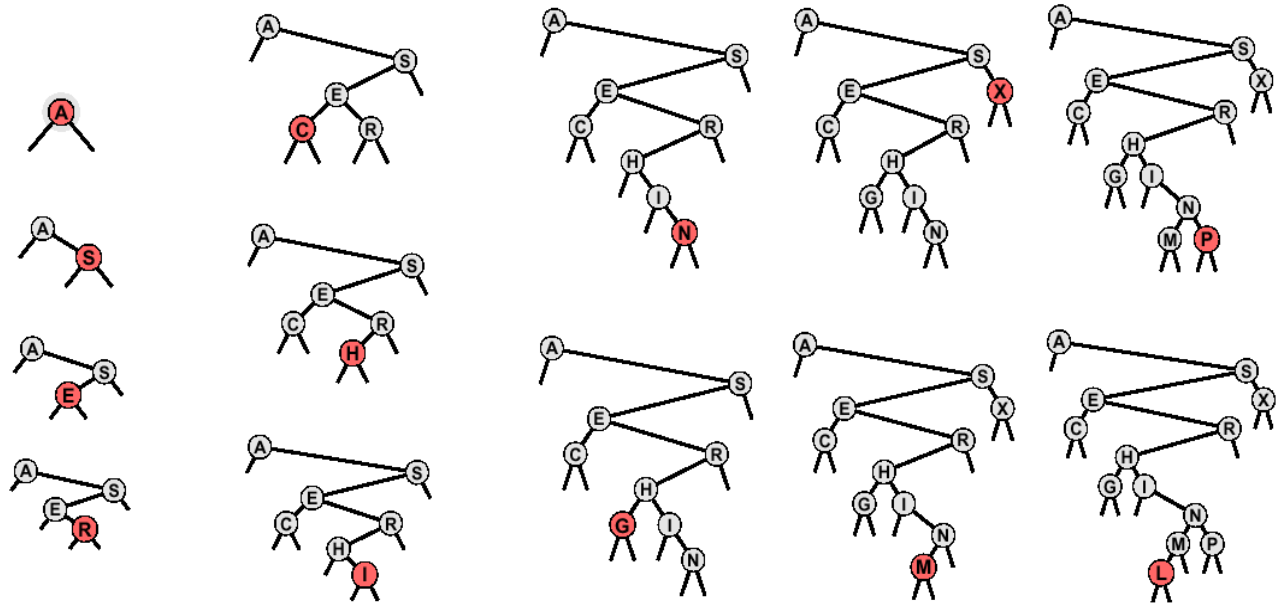
Binary trees are a very good option (not the best) for storing data where faster search/retrieval is required based on certain criteria. It does so by storing its elements in sorted order offering low time complexity compared to any other linear data structure. Any un-sorted collection can be inserted into Binary Search Tree in O (n log n) time complexity. Though the insertion time is increased per element from O(1) in Random Access array to O(log n) in Binary Search Tree, but we get a major advantage when we want to search/retrieve a particular element from the tree data structure.

Binary Tree is useful only when the tree is balanced, because only in that case a Binary Tree provides O(log n) search complexity, otherwise a binary tree will behave more like a linear data structure with O(n) time complexity for searching. A tree is called balanced when the height of the tree is logarithmic compared to number of its elements.

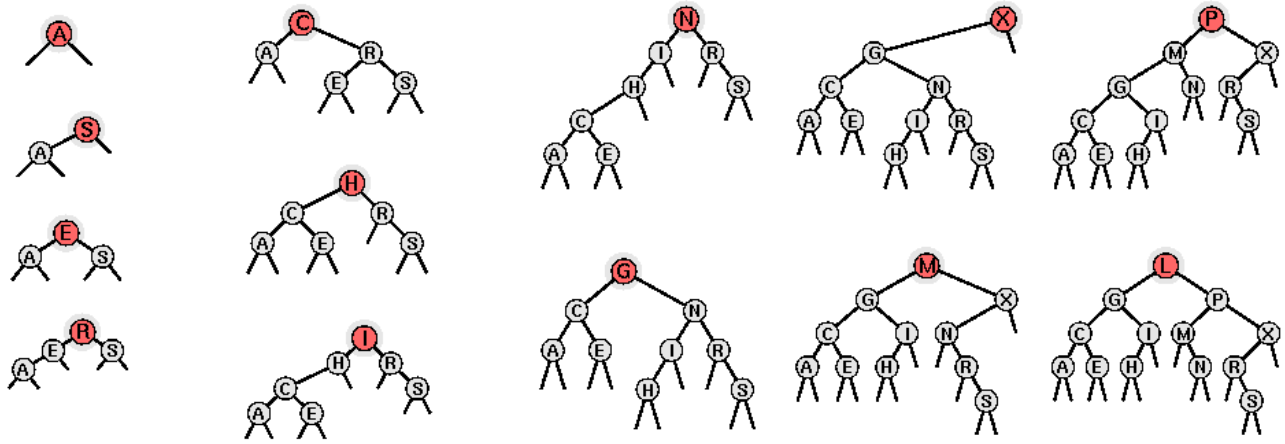
Left child of root is less in value than the right child. And the same is true for left and right sub tree in case of Binary Search Tree. BST is build for efficiently sorting & searching. In Order Traversal of a Binary Search Tree results in Ascending Order sorting of its elements.



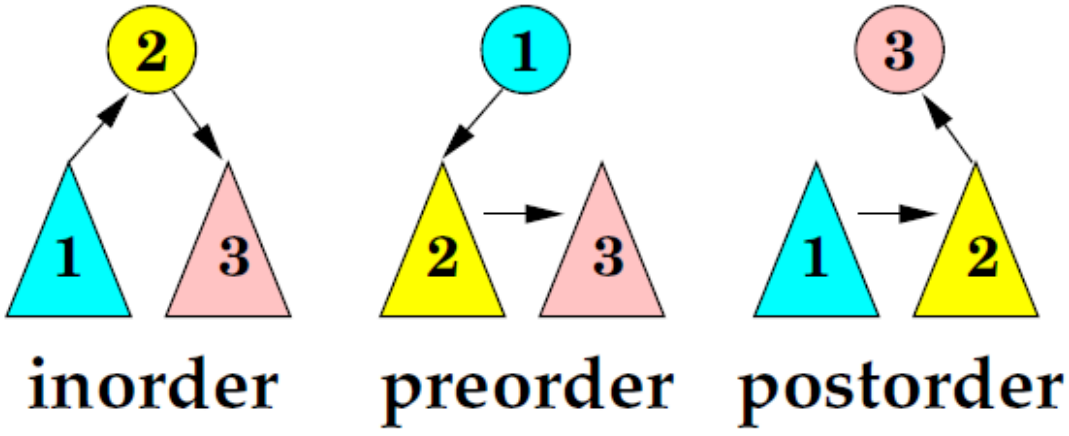
Insert the following keys into BST. A S E R C H I N G X M P L

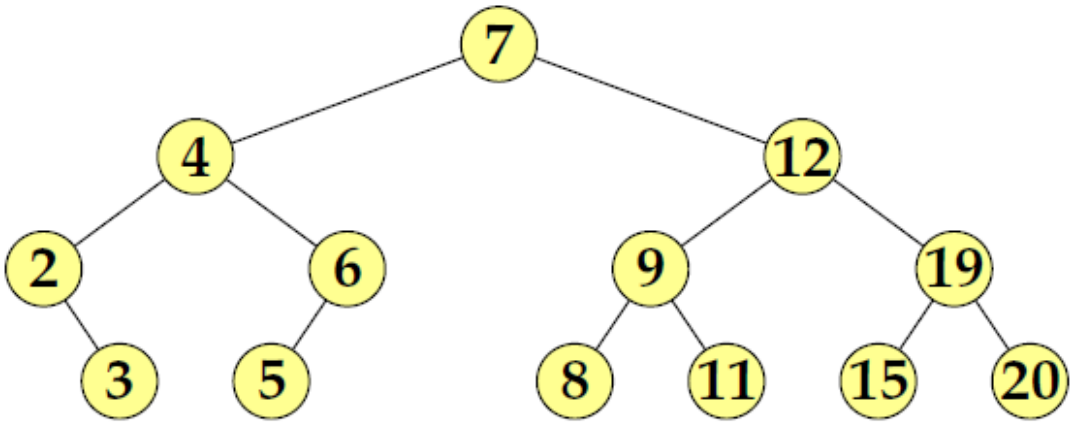


Constructing a BST with root insertion
Ex. **A S E R C H I N G X M P L**



A Binary Search Tree
(also known as sorted binary tree) is a node based binary tree data structure which has the following properties,
All elements in the left subtree are less than the root element.
All elements in the right subtree are greater than the root element.
Both, left and right subtree must also be binary search trees.
There can not be any duplicate element in the entire tree.





Inorder traversal gives:
2, 3, 4, 5, 6, 7, 8 , 9, 11, 12, 15, 19, 20.
Preorder traversal gives:
7, 4, 2, 3, 6, 5, 12, 9, 8, 11, 19, 15, 20.
Postorder traversal gives:
3, 2, 5, 6, 4, 8, 11, 9, 15, 20, 19, 12, 7.
TreeMap in Java 6
Java provides its Binary Search Tree implementation in TreeMap class. TreeMap is special kind of BST which is height balanced and known as red-black-tree.
Tree Traversal
There are three types of depth-first traversal, namely pre-order, in-order and post-order.
Pre-order Traversal
Visit the root node
Traverse the left subtree
Traverse the right subtree
In-order Traversal
Traverse the left subtree
Visit the root node
Traverse the right subtree

Post-order Traversal
Traverse the left subtree
Traverse the right subtree
Visit the node

Pseudo Code for Traversal
Given a Node with definition, Node{data, Node left, Node right}

Pre-order

```
preOrder(node){
if(node == null)
return
visit(node)
preOrder(node.left)
preOrder(node.right)
}
```

In-order

```
inOrder(node){
if(node==null)
return;
inOrder(node.left)
visit(node)
inOrder(node.right)
}
```

Post-order

```
postOrder(node){
if(node==null)
return;
postOrder(node.left)
postOrder(node.right)
visit(node)
}
```

```
public class Person implements Comparable<Person>{
  private String name;
  private int id;
  private Date dob;

  @Override
  public boolean equals(Object other){
     if(this == other) return true;
     if(!(other instanceof) Person) return false;
     Person guest = (Person) other;
     return (this.id == guest.id) &&
           (this.name != null && name.equals(guest.name)) &&
           (this.dob != null && dob.equals(guest.dob));
  }
```

```
    @Override
   public int hashCode(){
      int result = 0;
      result = 31*result + id;
      result = 31*result + (name !=null ? name.hashCode() : 0);
      result = 31*result + (dob  !=null ? dob.hashCode() : 0);

      return result;
   }
    @Override
    public int compareTo(Person o) {
       return this.id - o.id;
    }
}
```

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application. When we write a complex application ,application class should be as independent as possible of other classes so that we can write unit testing independently .Dependency Injection makes our programming code loosely coupled.
For example Class A is dependent to class B then we are able to write unit test case for both class and class B will get injected into class A (setter based or constructor based.)
Best example :- Using Jdbc properties separate from code.
Advantages

- Separation of Concerns
- Dependency injection can be used to externalize a system's configuration details into configuration files allowing the system to be reconfigured without recompilation. Separate configurations can be written for different situations that require different implementations of components. This includes, but is not limited to, testing.
- Reduction of boilerplate code in the application objects since all work to initialize or set up dependencies is handled by a provider component.
- Dependency injection allows concurrent or independent development. Two developers can independently develop classes that use each other, while only needing to know the interface the classes will communicate through. Plugins are often developed by third party shops that never even talk to the developers who created the product that uses the plugins.

Disadvantages

Dependency injection can make code difficult to trace (read) because it separates behavior from construction. This means developers must refer to more files to follow how a system performs.

Dependency injection typically requires more lines of code to accomplish the same behavior legacy code would.
http://www.journaldev.com/2394/dependency-injection-design-pattern-in-java-example-tutorial
when to use singleton and prototype scope in spring give me real time example.

Scope	Description
singleton	This scopes the bean definition to a single instance per Spring IoC container (default).
prototype	This scopes a single bean definition to have any number of object instances.
request	This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
session	This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
global-session	This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

We can said singleton bean is stateless and prototype is state full bean.

Case 1:- Suppose you want to make "CoinsVendingMachine" who take coins like 1 ,2,5 ,10 rs and deliver the product as per some coins based some logic in this case "CoinsVendingMachine" should be singleton and delivered product should be prototype scope.

Case 2:-Suppose there is a abstract class CommandManager who have process_Command and create_command method command can be email,sms,mms,audio ,video any type who have different characteristic. In this case we should define CommandManager is singleton bean and command should be prototype scope.

Case 3:- Call-center provide the customer service to customer but their is many way to convey message to customer like sms ,direct call , visiting in his home. In that case "CustomerService" is a Singleton scope but "conveyMessage" should be prototype scope.

Case 4 : ItemFactory (Singleton) item (Prototype) scope.

Describes the invocation semantics used to call a Web service operation: synchronous or asynchronous.

When you invoke a Web service synchronously, the invoking client application waits for the response to return before it can continue with its work. In cases where the response returns immediately, this method of invoking the Web service might be adequate. However, because request processing can be delayed due to connection timeout, read timeout, it is often useful for the client application to continue its work and handle the response later on.

By calling a Web service asynchronously, the client can continue its processing, without interruption, and be notified when the asynchronous response is returned.

Advantages and Disadvantages of Synchronous vs Asynchronous Messaging

There are some key advantages to asynchronous messaging. They enable flexibility and offer higher availability– There's less pressure on the system to act on the information or immediately respond in some way. Also, one system being down does not impact the other system. For example, emails – you can send thousands of emails to your friend without her having to revert back to you.

The drawbacks of asynchronous collaboration are that they can lack a sense of immediacy. There's less immediate interaction. Think about doing a chat with your friend on an instant messenger or over phone – it isn't a chat or a conversation unless your friend is replying back to you promptly.

Asynchronous message passing allows more parallelism. Since a process does not block, it can do some computation while the message is in transit. In the case of receive, this means a process can express its interest in receving messages on multiple ports simultaneously. Asynchronous message passing introduces several problems. What happens if a message cannot be delivered? What if the message was lost in the transmission?

What are database transaction Isolation level (Spring transaction isolation level)

```
@Transactional(isolation=Isolation.READ_COMMITTED)
public void someTransactionalMethod(User user) {...}
```

Inserts, updates, and deletes always behave the same no matter what the isolation level is. Only the behavior of select statements varies.

Dirty Reads :- A dirty read happens when a transaction reads data that is being modified by another transaction that has not yet committed.

Transaction 1	Transaction 2
/* Query 1 */ SELECT age FROM users WHERE id = 1; /* will read 20 */	
	/* Query 2 */ UPDATE users SET age = 21 WHERE id = 1; /* No commit here */
/* Query 1 */ SELECT age FROM users WHERE id = 1; /* will read 21 */	
	ROLLBACK; /* lock-based DIRTY READ */

But in this case no row exists that has an id of 1 and an age of 21.

Non-repeatable reads: Non-repeatable reads happen when a query returns data that would be different if the query were repeated within the same transaction. Non-repeatable reads can occur when other transactions are modifying data that a transaction is reading.

A *non-repeatable read* occurs, when during the course of a transaction, a row is retrieved twice and the values within the row differ between reads.

Transaction 1	Transaction 2
/* Query 1 */ SELECT * FROM users WHERE id = 1;	
	/* Query 2 */ UPDATE users SET age = 21 WHERE id = 1; COMMIT; /* in multiversion concurrency control, or lock-based READ COMMITTED */
/* Query 1 */ SELECT * FROM users WHERE id = 1; COMMIT; /* lock-based REPEATABLE	

READ */

Non-repeatable reads phenomenon may occur in a lock-based concurrency control method when read locks are not acquired when performing a SELECT, or when the acquired locks on affected rows are released as soon as the SELECT operation is performed. Under the multiversion concurrency control method, non-repeatable reads may occur when the requirement that a transaction affected by a commit conflict must roll back is relaxed.

There are two basic strategies used to prevent non-repeatable reads. The first is to delay the execution of Transaction 2 until Transaction 1 has committed or rolled back. This method is used when locking is used, and produces the serial schedule T1, T2. A serial schedule exhibits repeatable reads behaviour.

In the other strategy, as used in multiversion concurrency control, Transaction 2 is permitted to commit first, which provides for better concurrency. However, Transaction 1, which commenced prior to Transaction 2, must continue to operate on a past version of the database — a snapshot of the moment it was started. When Transaction 1 eventually tries to commit, the DBMS checks if the result of committing Transaction 1 would be equivalent to the schedule T1, T2. If it is, then Transaction 1 can proceed. If it cannot be seen to be equivalent, however, Transaction 1 must roll back with a serialization failure.

Phantom reads: Records that appear in a set being read by another transaction. Phantom reads can occur when other transactions insert rows that would satisfy the WHERE clause of another transaction's statement.

Transaction 1	**Transaction 2**
/* Query 1 */ SELECT * FROM users WHERE age BETWEEN 10 AND 30;	
	/* Query 2 */ INSERT INTO users VALUES (3, 'Bob', 27); COMMIT;
/* Query 1 */ SELECT * FROM users WHERE age BETWEEN 10 AND 30; COMMIT;	

Isolation Level	Transactions	Dirty Reads	Non-Repeatable Reads	Phantom Reads
TRANSACTION_NONE	Not supported	*Not applicable*	*Not applicable*	*Not applicable*
TRANSACTION_READ_COMMITTED	Supported	Prevented	Allowed	Allowed
TRANSACTION_READ_UNC	Supported	Allowed	Allowed	Allowed

OMMITTED				
TRANSACTION_REPEATABLE_READ	Supported	Prevented	Prevented	Allowed
TRANSACTION_SERIALIZABLE	Supported	Prevented	Prevented	Prevented

Isolation Levels vs Lock Duration

In lock-based concurrency control, isolation level determines the duration that locks are held.
"C" - Denotes that locks are held until the transaction commits.
"S"- Denotes that the locks are held only during the currently executing statement. Note that if locks are released after a statement, the underlying data could be changed by another transaction before the current transaction commits, thus creating a violation.

Isolation level	Write Operation	Read Operation	Range Operation (...where...)
Read Uncommitted	S	S	S
Read Committed	C	S	S
Repeatable Read	C	C	S
Serializable	C	C	C

Database ACID (Atomicity, Consistency, Isolation, Durability) Properties

There are a set of properties that guarantee that database transactions are processed reliably, referred to as ACID (Atomicity, Consistency, Isolation, Durability).

Atomicity : Atomicity refers to the ability of the database to guarantee that either all of the tasks of a transaction are performed or none of them are. Database modifications must follow an all or nothing rule. Each transaction is said to be atomic if when one part of the transaction fails, the entire transaction fails.

Consistency : The consistency property ensures that the database remains in a consistent state before the start of the transaction and after the transaction is over (whether successful or not). For example, in a storefront there is an inconsistent view of what is truly available for purchase if inventory is allowed to fall below 0, making it impossible to provide more than an intent to complete a transaction at checkout time. An example in a double-entry accounting system illustrates the concept of a true transaction. Every debit requires an associated credit. Both of these happen or neither happen.

A distributed data system is either strongly consistent or has some form of weak consistency. Once again, using the storefront example, a database needs to provide consistency and isolation, so that when one customer is reducing an item in stock and in parallel is increasing the basket by one, this is isolated from another customer who will have to wait while the data store catches up. At the other end of the spectrum is BASE (Basically Available Soft-state Eventual consistency).

Isolation : Isolation refers to the requirement that other operations cannot access or see the data in an intermediate state during a transaction. This constraint is required to maintain the performance as well

as the consistency between transactions in a database. Thus, each transaction is unaware of another transactions executing concurrently in the system.

Durability: Durability refers to the guarantee that once the user has been notified of success, the transaction will persist, and not be undone. This means it will survive system failure, and that the database system has checked the integrity constraints and won't need to abort the transaction. Many databases implement durability by writing all transactions into a transaction log that can be played back to recreate the system state right before a failure. A transaction can only be deemed committed after it is safely in the log.