Compare and Swap. ⇒

* Lock Free Algorithm → Based on non-blocking algorithms which Provides Global Progress inspite of individual Stop

* A Technique used when Designing Concurrent Algorithms. Basically CAS Compares an expected value to the Concrete value of a Variable, and if the Concrete value of the variable is equals to the expected value, Swaps the value of the variable for a new Variable.

```
Class MyLock {
    Private boolean locked = false;
    Public boolean lock () {
        if (! locked ) {
            locked = true;
            return true;
        }
        return false;
    }
}
```

CAS as Atomic Operation :

```
Public Static Class MyLock {
    Private AtomicBoolean locked = new AtomicBoolean(false);
    Public boolean lock () {
        return locked . CompareAndSet ( false, true);
    }
}
```

* Here it Compares the value of locked to false and if it is false it sets the new value of AtomicBoolean to true.

The CompareandSet () method returns true if the value is Swapped, and false if not.

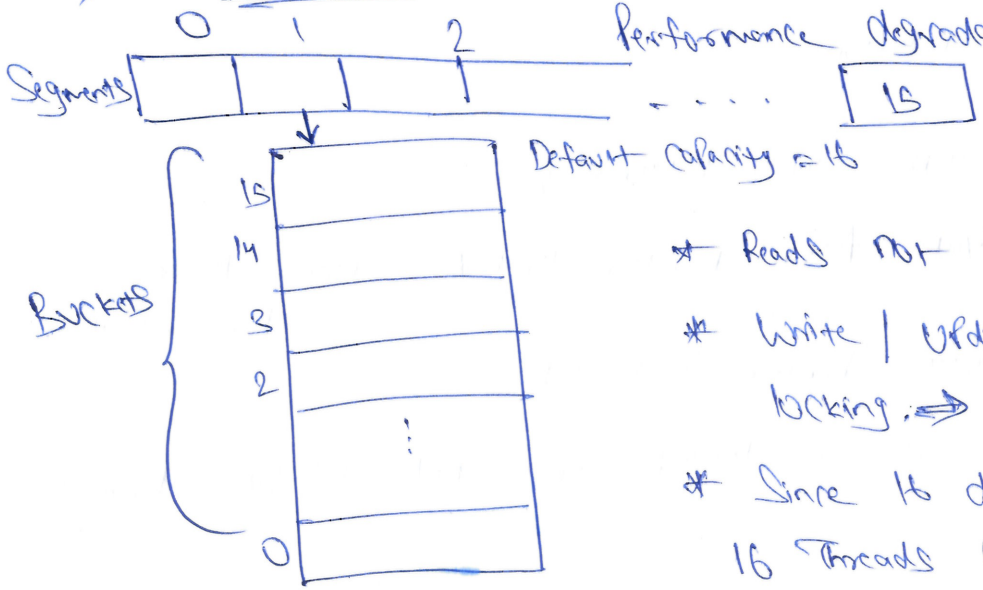
* CAS is generally much faster than locking, but it does depend on the degree of contention. Because CAS may force a retry if the value changes between reading and comparing, a thread can get stuck in a busy-wait if the variable in question is hit hard by many other threads.

Synchronized Collections vs Concurrent Collections.

* Collections.synchronizedMap() and Collections.synchronizedList() provides a basic conditionally thread-safe implementation of Map and List. (fail-fast Iterator)

* Single collection-wide lock is an impediment to scalability and it often becomes necessary to lock a collection for a considerable time during iteration to prevent ConcurrentModificationException.

⇒ ConcurrentHashMap → Introduced a concept of Segmentation so performance degradation won't happen.



* Reads not having lock
* Write / Update having segment level locking. ⇒ lock Striping.
* Since its default capacity by default 16 Threads can write/update simultaneously.
* Concurrency-level is 16.

* HashTable all methods including get is synchronized.
* CHM - Put & remove use lock Striping → find value always cached.
  - Get : value is volatile, next is final - Hence no stale reads.

	Re ordering	Caching	happens-before	Mutual-exclusion
Sync	✓	✓	✓	✓ (Pessimistic lock)
Volatile	✓	✓	✓	X
Atomic (CAS)	✓	✓	✓	✓ (Optimistic lock)

* Optimistic lock → Read is allowed across thread. Write is not allowed.
* Pessimistic lock → no read, write till lock is released.

Issues with Synchronization:

1. One condition per lock
2. No lock polling. i.e Thread which does not get a lock can't do anything else. [tryLock() method]
3. No Timed lock waits. [tryLock(longtime, TimeUnit unit)]
4. No Interrupted lock waits.
5. Block Structured lock (JVM is doing an intrinsic lock. Ex: Calling an inner method lock)

wait() / notify():

1. In Java, the Object is the entity that is shared between threads which allows them to communicate with each other. The threads have no knowledge of each other and they run Asynchronously.
2. In Java, you wait() on a particular instance of an Object called the monitor. notify() will notify other threads waiting on same Object.

* If wait() and notify() was on a Thread, each thread would have to know the status of every other thread. How would thread1 know that thread 2 was waiting for access to particular resource?

* JVM has a hidden wait queue() which puts all threads in a queue waiting for same object and then notify() will dequeue & put to running state.

* notify() → Printer example
  notifyAll() → Application cache clearing → Objects wake up all after this.
  Ex: Parallel Garbage Collector.

⇒ Transfer Queue

* Extends Blocking Queue.
* transfer() method is like a Synchronous Put() method.
* tryTransfer() → returns boolean
  ie weather consumer able to consume now or not?

Linked Transfer Queue: implements Transfer Queue.

⇒ Object & Thread itself in a Queue, hence called Dual Queue or Dual blocking Queue

transfer	put	tryTransfer
Blocking	Normal	Depends on consumer

* hasWaitingConsumer(count()) → return blocking count
* hasWaitingConsumer() → true/false.

* Fastest blocking Queue:
  → LinkedBlockingQueue (JDK 5 & 6)
  → LinkedTransferQueue (JDK 7)