```
trait LatencyBound {
  // execute readOp with strongest consistency possible
  // within the latency bound
  def rush[T](bound: Duration,
              readOp: ConsistencyLevel => T): Rushed[T]
}
/* Generic reservaton pool, one per ADT instance.
   `max` recomputed as needed (e.g. for % error) */
class ReservationPool(max: () => Int) {
  def take(n: Int): Boolean // try to take tokens
  def sync(): Unit      // sync to regain used tokens
  def delta(): Int      // # possible ops outstanding
}
/* Counter with ErrorBound (simplified) */
class Counter(key: UUID) with ErrorTolerance {
  def error: Float // % tolerance (defined by instance)
  def maxDelta() = (cassandra.read(key) * error).toInt
  val pool = ReservationPool(maxDelta)

  def read(): Interval[Int] = {
    val v = cassandra.read(key)
    Interval(v - pool.delta, v + pool.delta)
  }
  def incr(n: Int): Unit =
    waitFor(pool.take(n)) { cassandra.incr(key, n) }
}
```