```
// creates a table of pools, so each event gets its own
// 5% error tolerance on `remaining` method, weak otherwise
val tickets = UUIDPool() with Consistency(Weak)
                with Remaining(ErrorTolerance(0.05))

// called from displayEvent (& purchaseTicket)
def getTicketCount(event: UUID): Interval[Int] =
  tickets(event).remaining()

def purchaseTicket(event: UUID) = {
  // UUIDPool is safe even with weak consistency (CRDT)
  endorse(tickets(event).take()) match {
    case Some(ticket) =>
      // imprecise count returned due to error tolerance
      val remaining = getTicketCount(event)
      // use maximum count possible to be fair
      val price = computePrice(remaining.max)
      display("Ticket reserved. Price: $" + price)
      prompt_for_payment_info(price)
    case None =>
      display("Sorry, all sold out.")
  }
}
```