```
def kMeansIterate(partitionedPoints: Seq[SiloRef[Array[Point]]],
                  centroids: Array[Point],
                  iteration: Int): Array[Point] = {
  val clusterParts =
    partitionedPoints.map(silo => silo.apply(
      spore {
        val lCentroids = centroids // spore header
        (points: Array[Point]) =>
          SiloRef.populate(currentHost, kmeansLocal(points, lCentroids))
      }
    ).send())

  val newCentroids =
    Await.result(Future.sequence(clusterParts).map(seq => {
      seq.reduce((x, y) => x ++ y)
         .groupBy(x => x._1)
         .toSeq
         .sortBy(x => x._1)
         .map(x => x._2)
         .map(clp => clp.map(x => x._2).toArray.unzip)
         .map({case (ns, points) => (ns.sum, sumPoints(points)) })
         .map({case (n, sum) => divPoint(sum, n) })
    }), Duration.Inf).toArray


  val diff =
    newCentroids.zip(centroids).map({case (p1, p2) => dist(p1, p2) }).max
    if (diff < epsilon) // check if converged, else iterate again
      newCentroids
    else
      kMeansIterate(partitionedPoints, newCentroids, iteration + 1)
  }
```

apply returns a SiloRef

type: SiloRef[Array[Point]] parameter of fn passed to map on Seq[SiloRef[Array[Point]]]

Spore passed to apply method on SiloRef

send returns a Future

Await.result is a barrier which blocks until the argument future is resolved

type: Seq[Array[Point]]] parameter of fn passed to map on Future[Seq[Array[Point]]]

Chained higher-order functions on standard Scala collections

Pattern match deconstructing a pair and assigning names to each component.