```
val leader = branch(isStationary) { S(grain) } { false } //(1.) leader election
//.. (2.) sense animals in danger ..
val noAnimal : Map[ID, P] = Map.empty
val animalsInDanger = {
  foldhood(noAnimal)(combineDangerMap){
    mux(nbr(isDanger)) {
      Map(nbr(mid() -> currentPosition()))
    } { noAnimal }
  }
}
val potential = distanceTo(leader)
//(2.) send information to the leader
val animalDangerInArea = C(potential, combineDangerMap, animalsInDanger, noAnimal)
//(3.) leader chooses an animal to rescue
val leaderHealTask = animalDangerInArea.toSeq.sortBy {
  case (_, p) => p.distance(currentPosition()) //choosing policy
}.headOption.map {
  case (id, p) => HealTask(mid(), id, p)
}
val exploreArea = ExploreTask(mid(), currentPosition(), grain) //for explorers
//(4.) and share its choice via broadcast
val healTaskForSlave =  broadcast(leader, leaderHealTask)
val exploreAreaToSlave = broadcast(leader, exploreArea)
//(5.) slave choose the task according its role, intentions and leader choice
val collectiveTasks = Seq(healTaskForSlave, exploreAreaToSlave)
val localTask = Planner.eval(collectiveTasks) //the task choice is encapsulated here..
val actuation = localTask.call(this) // produces data in order to achieve the task chosen
```