Algorithm 3: `Propulate` with real migration.

Input: Search-space limits, *num_islands*, island sizes P_i ($i = 1, \ldots,$ *num_islands*), number of iterations *self.generations*, evolutionary operators (including *selection_policy*, *crossover_probability*, *mutation_probability* etc.), *migration_probability*, *migration_topology*, *emigration_policy*.

Configure parallel setup of *num_islands* evolutionary islands with P_i co-workers each and respective intra-island communicators. Intra-island ranks correspond to an island's co-workers. Each co-worker evaluates one individual at a time and maintains its own population list *self.population* of evaluated and migrated individuals on the island.

`/* START OPTIMIZATION. */`
for each worker **do in parallel**
  while *generation* $\leq$ *self.generations* **do** `// Loop over generations.`
    `/* BREEDING, EVALUATION, AND ASYNCHRONOUS INTRA-ISLAND SYNCHRONIZATION */`
    Breed and evaluate new individual. Append it to *self.population*. Send it to co-workers to update their populations lists: `self.evaluate_individual()`
    Check for and possibly receive individuals bred and evaluated by co-workers. Append them to *self.population*: `self.receive_intra_isle_individuals()`
    `/* EMIGRATION */`
    if *random.random()* $\leq$ *self.migration_probability* **then**
      Choose emigrants from worker-exclusive subset of currently active individuals on the island according to *emigration_policy*. Send emigrant(s) to workers of other islands according to *migration_topology*. For real migration, an individual can only exist actively on one island at a time. Send emigrants to co-workers for deactivation. Deactivate emigrants in *self.population*: `self.send_emigrants()`
    end
    `/* IMMIGRATION */`
    Check for and possibly receive incoming migrants sent by workers from other islands. Add them to *self.population*: `self.receive_immigrants()`
    `/* ASYNCHRONOUS INTRA-ISLAND SYNCHRONIZATION */`
    Check for and possibly receive individuals emigrated by co-workers. Try to deactivate them in *self.population*. If an individual to be deactivated was bred and sent out by a co-worker but has not yet been received and added to *self.population*, append it to a history list *self.emigrated* and try again in the next generation: `self.deactivate_emigrants()`
    Go to next generation: `generation += 1`
  end
  `/* OPTIMIZATION DONE: FINAL SYNCHRONIZATION */`
  Having completed all generations, wait for all other workers to finish:
  `MPI.COMM_WORLD.barrier()`
  Check for incoming messages so that each worker finally holds the complete population.
  Final check for individuals evaluated by co-workers: `self.receive_intra_isle_individuals()`
  `MPI.COMM_WORLD.barrier()`
  Final check for incoming individuals immigrating from other islands: `self.receive_immigrants()`
  `MPI.COMM_WORLD.barrier()`
  Final check for emigrants from co-workers to be deactivated: `self.deactivate_emigrants()`
  `MPI.COMM_WORLD.barrier()`
end

Result: n best individuals.