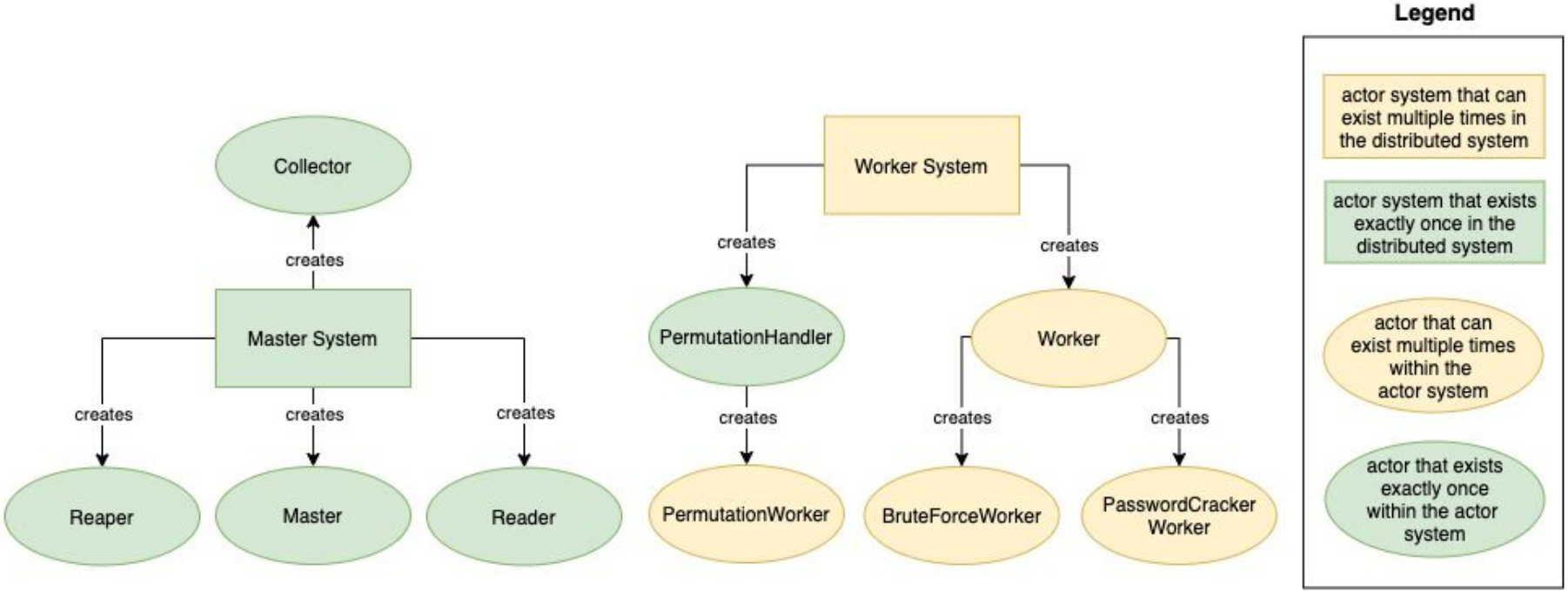
Task 3: Password Cracking

Team Dally

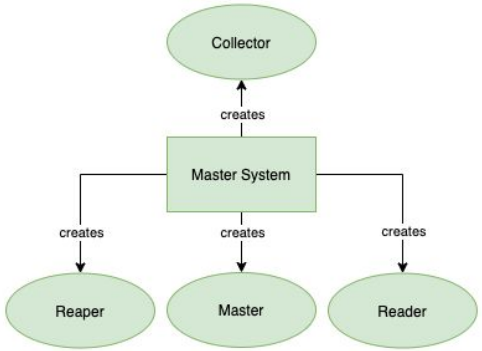
Contents

1) Actor Hierarchies
2) Solution Concept
3) Structures
4) Actor States
5) Initial Actor System Start-Up
6) Message Passing
7) Critical Reflection of the Solution

1) Actor Hierarchies

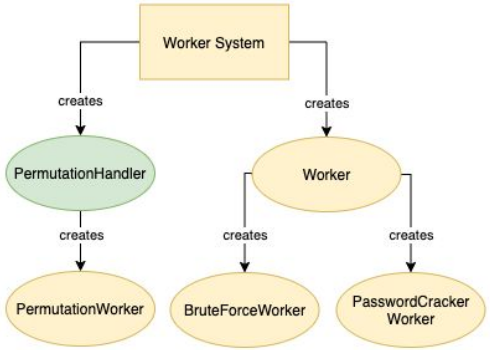


2) Solution Concept



MasterSystem

1) Reads file contents and creates work packages
2) Gets results from Workers and sends them to the Collector
3) System Shutdown



WorkerSystem

1) PermutationHandler and PermutationWorkers calculate all permutations to solve the hints (Data Parallelism)
2) Workers start cracking passwords (Data Parallelism)
   a) BruteForceWorkers crack Hints (Data Parallelism)
   b) PasswordCrackerWorkers crack Passwords (Data Parallelism)

→ a) & b) are partly parallel too (Task Parallelism)

3) Structures - PasswordWorkPackage

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class PasswordWorkPackage implements Serializable {
    private int id;
    private String name;
    private String passwordCharacters;
    private int passwordLength;
    private String password;
    private String [] hints;
}
```

- Created by Master
- Solved by Worker with help of its child actors
- Contains all information on one password that should be cracked
- All information is supplied by the Reader to the Master as BatchMessages (file contents)

Parallelization of Password Solving:

- Each work package contains exactly one password
- Multiple Workers are solving different passwords (data parallelism)

3) Structures - PermutationWorkPackage

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class PermutationWorkPackage implements Serializable {
    private char head;
    private char head2;
    private String passwordChars;
}
```

- Created by Master
- Solved by PermutationHandler with the help of its children
- passwordChars - all characters that could appear in the password
- head & head2 - specific characters

Parallelization of Permutation Calculation:

- All permutations of the passwordChars excluding the characters head and head2 are calculated
- head and head2 are prepended to the calculated permutations

→ one part of all permutations of all passwordChars is calculated and allows parallelization

- Multiple PermutationWorkers work on different permutations (data parallelism)

3) Structures - BruteForceWorkPackage

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class BruteForceWorkPackage implements Serializable {
    private int passwordId;
    private String passwordChars;
    private String hint;
}
```

- Created by Worker
- Solved by BruteForceWorkers
- Contains the password ID, the passwordChars, and the hint that should be solved

Parallelization of Hint Solving:

- Each work package contains exactly one hint
- Multiple BruteForceWorkers work on solving different hints for different passwords (data parallelism)

3) Structures - PermutationSingleton

```java
public class PermutationSingleton {
    private static final List<String> permutations = synchronizedList(new ArrayList<>());

    public static List<String> getPermutations() {
        return permutations;
    }

    public static void addPermutation(String permutationsPart) {
        permutations.add(permutationsPart);
    }
}
```

- Data structure to store hint permutations within the actor system
- Exists once per actor system → permutations are calculated once per WorkerSystem
- PermutationWorkers write permutations to Singleton while calculating

→ Prevents redundant calculation of permutations for solving each hint

4) Actor State - Master

Tasks: gets file contents from reader; creates and distributes work packages; sends results to collector

Type	Name	Description
ActorRef	reader	Reference to Reader actor (created by MasterSystem)
ActorRef	collector	Reference to Reader actor (created by MasterSystem)
ActorRef	largeMessageProxy	Reference to the LargeMessageProxy actor (child)
List<ActorRef>	workers	References to all workers that are registered with the Master
List<PasswordWorkPackage>	passwordWorkPackages	Unassigned work packages for not yet solved passwords
List<PermutationWorkPackage>	permutationWorkPackages	All work packages for calculating permutations
Map<Integer, Boolean>	resultTracker	Mapping password ID to whether the master has already received a result for the password or not

4) Actor State - PermutationHandler

Tasks: controls the parallel calculation of hint permutations for its actor system

Type	Name	Description
List<ActorRef>	permutationWorkers	References to all PermutationWorkers that register to the PermutationHandler
Cancellable	workRequest	Retries getting PermutationWorkPackages from Master until it receives them
List<PermutationWorkPackage>	permutationWorkPackages	Unassigned work packages for not yet calculated permutations (received from Master)
Map<String, Boolean>	resultTracker	Tracks the permutationWorkPackages in Map to validate if calculations are done
ActorRef	largeMessageProxy	Reference to the LargeMessageProxy actor (child)

4) Actor State - Worker

Tasks: solves a given PasswordWorkPackage with the help of its child actors

Type	Name	Description
ActorRef	largeMessageProxy	Reference to the LargeMessageProxy actor (child)
List<ActorRef>	bruteForceWorkers	References to all bruteForceWorkers registered with Worker
List<ActorRef>	passwordCrackerWorkers	References to all passwordCrackerWorkers registered with Worker
List<BruteForceWorkPackage>	bruteForceWorkPackages	Unassigned work packages containing unsolved hints (received from Master)
Map<Integer, List<HintResult>>	hintResults	Map of password ID to list of hint results the Worker has already received
Map<Integer, PasswordWorkPackage>	passwordWorkPackages	Map of password ID to work package for all passwords the worker is working on
List <Integer>	workPackagesReadyFor PasswordCracker	Password IDs of passwords that are ready to be cracked (all hints solved)
Int	numberOfHints	Total number of hints per password, used to check if calculations are done

4) Actor State

PermutationWorker:

- **Tasks:** calculates hint permutations for a given PermutationWorkPackage

BruteForceWorker:

- **Tasks:** solves a given BruteForceWorkPackage

PasswordCrackerWorker:

- **Tasks:** cracks a given password with the help of the solved hints

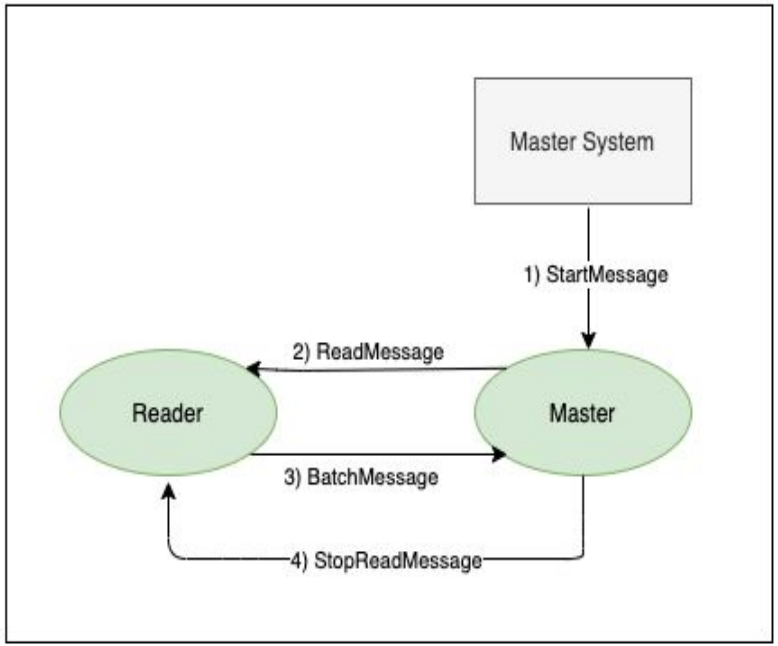
Type	Name	Description
ActorRef	largeMessageProxy	Reference to the LargeMessageProxy actor (child)

5) Initial Actor System Start-Up

- MasterSytem creates Master, Reader, Reaper, Collector, MetricsListener, ClusterListener
- MasterSystem can also have workers: creates PermutationHandler + Workers
- Reader starts to read file on start-up
- MasterSystem sends StartMessage to Master

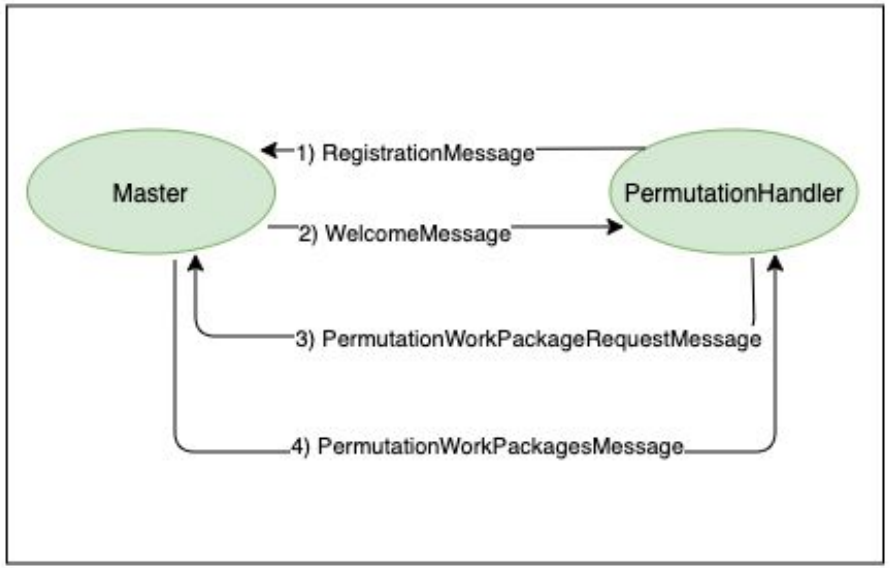
- WorkerSystem creates Reaper, MetricsListener, ClusterListener, multiple Workers, and exactly one PermutationHandler
- PermutationHandler and Workers register with Master on start-up

6) Message Passing - Reading & Work Packages



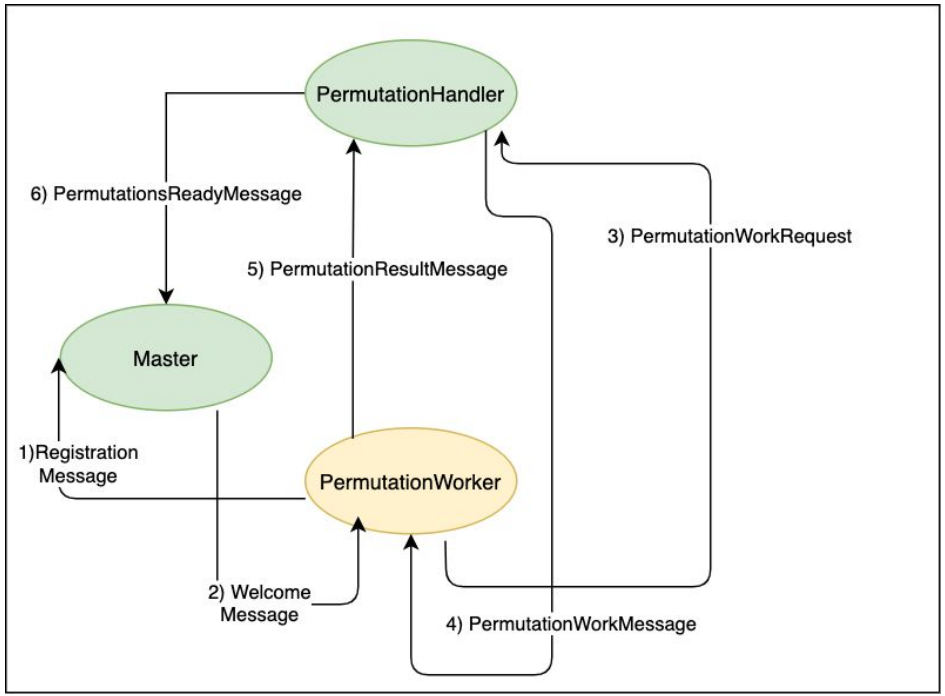
- Receiving the StartMessage triggers the Master to send first ReadMessage to Reader
- Reader responds with BatchMessage via LargeMessageProxy containing the already read file contents, clears its buffer, and continues reading
- Master processes BatchMessage by creating the work packages and sends next ReadMessage to Reader
- Once the Master receives an empty BatchMessage it sends StopReadMessage to Reader which then stops reading

6) Message Passing - Permutation Calculation



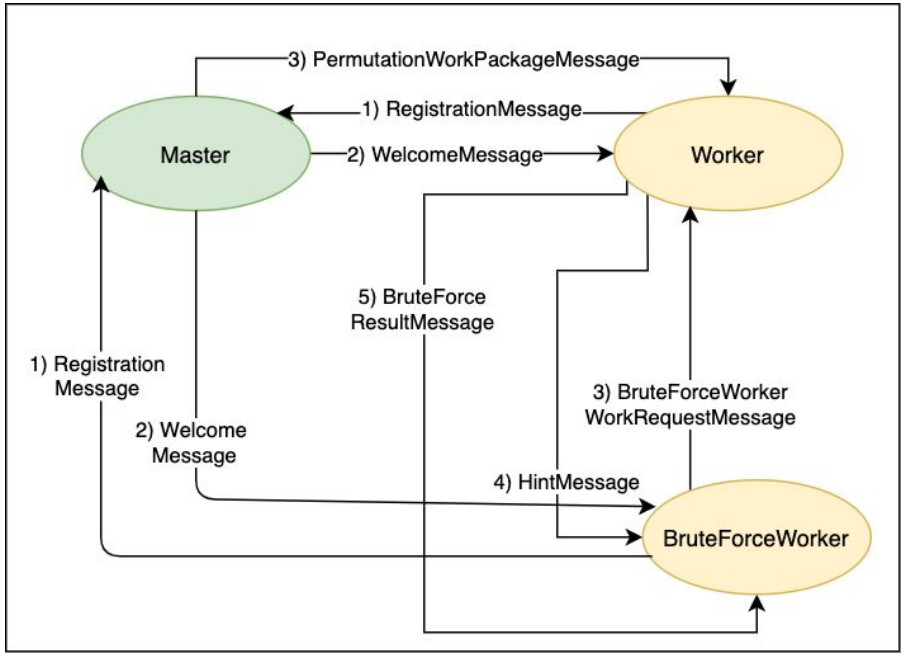
- PermutationHandler sends RegistrationMessage to Master on start-up
- Master responds with WelcomeMessage
- Receiving the WelcomeMessage triggers to PermutationHandler to send a PermutationWorkPackageRequestMessage to Master (pull, retried until fulfilled)
- Master responds with PermutationWorkPackagesMessage via LargeMessageProxy containing all PermutationWorkPackages

6) Message Passing - Permutation Calculation



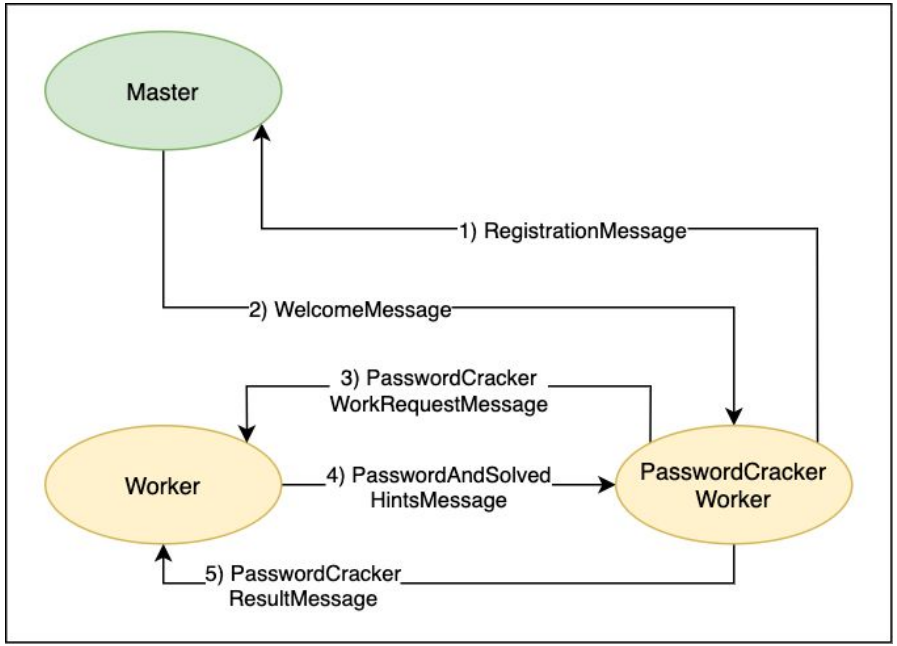
- Receiving the PermutationWorkPackages triggers the PermutationHandler to create PermutationWorkers (children)
- Each PermutationWorker registers with Master and is welcomed
- PermutationWorker sends work request to Handler (pull) and Handler responds with work package
- PermutationWorker calculates permutations, writes them into PermutationSingleton, tells Handler it's done
- Handler checks if all calculations are done, tells master if yes, responds with next work package if not

6) Message Passing - Hint Cracking



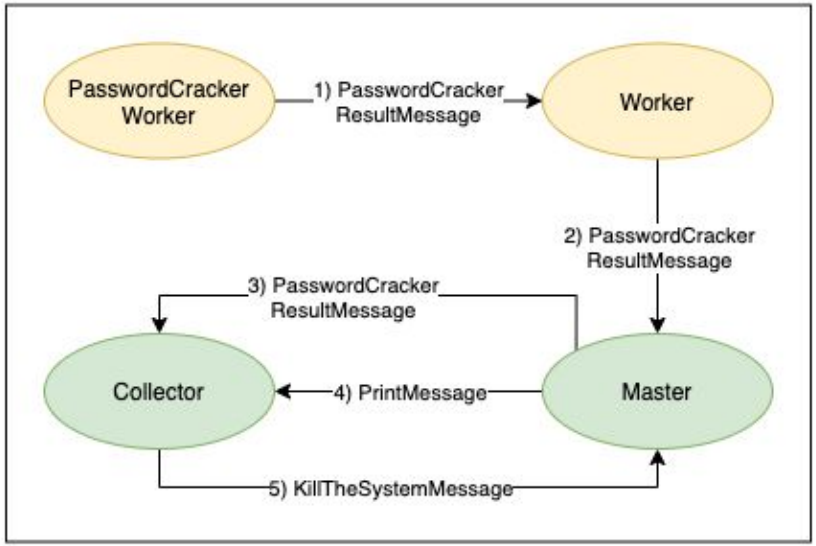
- Workers registered with Master and received WelcomeMessage on start-up and then waited for work
- Receiving the PermutationsReadyMessage triggers the Master to send work packages to Workers
- Receiving work triggers the Workers to create BruteForceWorkers (children)
- They register with the master, get welcomed
- They then ask parent Worker for work (pull) and the Worker gives them a hint to crack (HintMessage)
- BruteForceWorker compares hint to hashed permutations (stored in PermutationSingleton) to crack the encryption, then compares the hint characters to all password characters to find the solution character and sends it to parent Worker
- Worker checks if all hints are solved, responds with next work package if not

6) Message Passing - Password Cracking



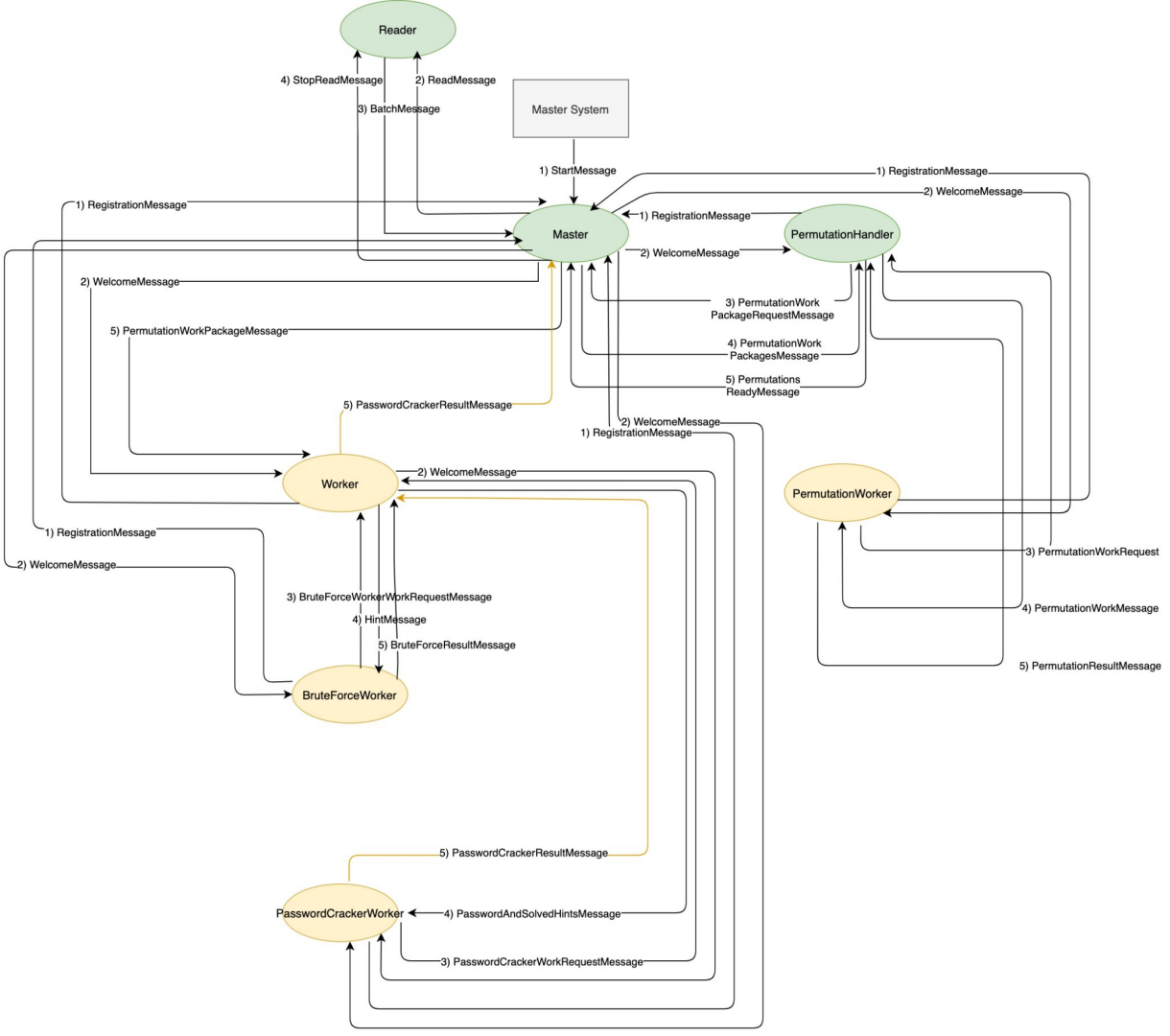
- If all hints for the password are solved, Worker adds password to list of work packages ready for cracking and creates PasswordCrackerWorkers & asks Master for next password to crack (pull)
- PasswordCrackerWorkers register with the Master, get welcomed, ask parent Worker for work (pull)
- Worker responds with encrypted password to crack and solved hints
- PasswordCrackerWorker deducts characters that are actually in the password from hints, generates corresponding permutations, compares encrypted password until a match is found

6) Message Passing - Result Collection



- When the Worker receives a cracked password from the PasswordCrackerWorker, it forwards the message to the Master which forwards it to the collector
- The master checks if all passwords are solved on receiving the result and if yes, sends PrintMessage to Collector
- Collector prints all results, tells the Master to terminate the system

6) Message Passing - Entire Graph



7) Critical Reflection of the Solution

- To improve fault tolerance and reliability:
  - Make more use of schedulers
  - Custom supervision strategies
  - Write UnitTests
  - Monitor task completion more thoroughly (more retries)
- To remove bottleneck at permutation calculation:
  - Problem: password cracking waits for the completion of the calculation of all permutations for solving the hints → high load & resource usage (cpu, heap space), blocking & very slow
  - Option 1:
    - Don’t calculate permutations up front but everytime a hint is solved → redundant calculations but less heap space used permanently and load of calculating stretched over a larger time frame, non-blocking
  - Option 2:
    - Calculate less permutations and only solve hints that can be solved with those → not all hints can be solved per password, more permutation calculation when solving password

7) Critical Reflection of the Solution

Because of the permutation calculation bottleneck and the high usage of heap space we think that our solution might not complete successfully on systems with less computational power and memory.

1) We *successfully* tested:

- MasterSystem without any workers on MacOs Catalina 10.15.4, 16GB RAM, Prozessor: 2,2 GHz 6-Core Intel Core i7
  → see file log-master.txt
- WorkerSystem on MacOs BigSur 11.3.1 32GB RAM, Prozessor: 2,3 GHz Quad-Core Intel Core i7
  → see file log-worker.txt

2) We *successfully* tested:

- MasterSystem with 4 workers and WorkersSystem with 4 workers both on the same laptop started in two terminals: MacOs BigSur 11.3.1 32GB RAM, Prozessor: 2,3 GHz Quad-Core Intel Core i7

7) Critical Reflection of the Solution

3) We *unsuccessfully* tested:

- MasterSystem with multiple workers on MacOs Catalina 10.15.4, 16GB RAM, Prozessor: 2,2 GHz 6-Core Intel Core i7

  → **java.lang.OutOfMemoryError: Java heap space error**
  Solution Attempts: used PermutationSingleton instead of passing the whole Map/ List around via LMP, used List instead of Map inside PermutationSingleton

  → **OutOfMemoryError: GC overhead limit exceeded error**
  Solution Attempts: tried using different GC, calling GC manually

  → **Scheduled sending of heartbeat was delayed + heartbeat interval is growing too large**
  Solution Attempts: tried using custom dispatchers, split up permutation calculation work packages even further so that the computation of each work package takes a shorter amount of time

Unfortunately we couldn't find a way to solve this without completely redesigning our whole solution (e.g., not computing all permutations upfront). Since we've put a lot of work into it, we decided to leave it as it is and to critically reflect instead.