Hasso Plattner Institut
Digital Engineering · Universität Potsdam

Assignment 3

Distributed Data Management

Problemstellung

Der Grundgedanke unsere Implementierung ist es, Mehrfachberechnungen auszuschließen, um schnellst möglich alle 100 Passwörter zu cracken. Wir identifizierten hierfür folgende Hauptbottlenecks die der Lösung des Problems im Wege stehen:

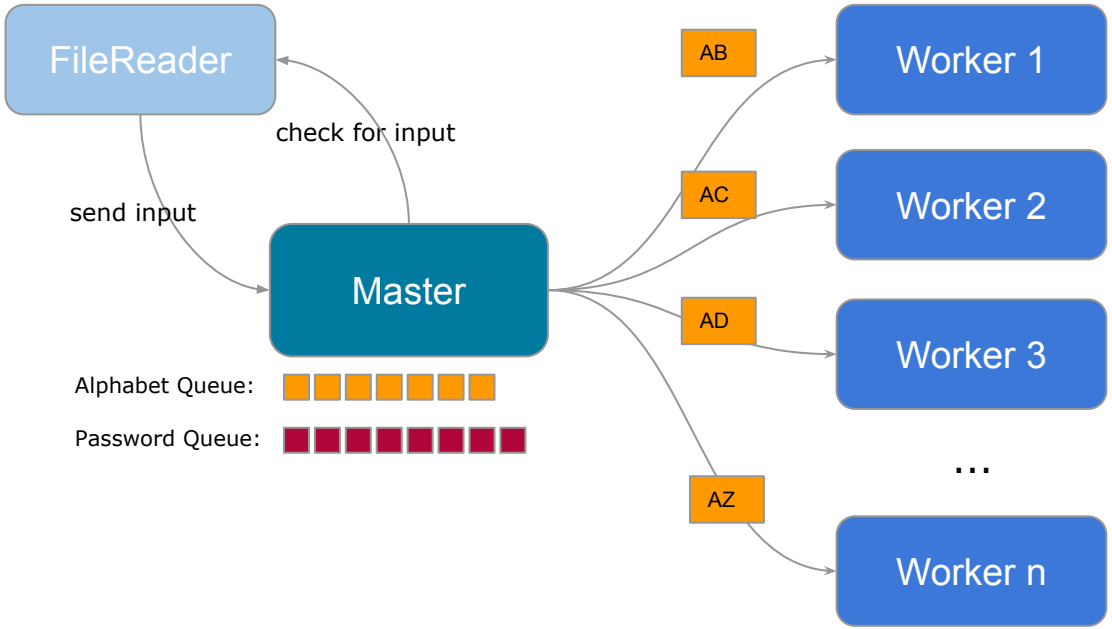
1. Berechnung der Hints
2. Berechnung der Passwörter mithilfe der Hints.
3. Netzwerkkommunikation

Mithilfe der Hints ist es möglich, die Anzahl der möglichen Lösungen eines Passwortes drastisch zu reduzieren. Zu Beginn ist nur bekannt, dass ein Passwort aus 10 Zeichen 10 unterschiedlicher Ziffern besteht. Mithilfe der Hints können dann zwei Ziffern gefunden werden, aus denen das Passwort besteht. Dadurch müssen nur noch 2^{10} =1024 Möglichkeiten ausprobiert werden, was für einen Computer sehr schnell lösbar ist. Die Lösungsraum der Hints ist dagegen sehr viel größer, naiv betrachtet besteht jeder aus 10 Zeichen mit 10 möglichen Ziffern, wobei jeder allerdings nur einmal vorkommt. Das macht insgesamt 10!=3628800 mögliche Lösungen, ist also immernoch bedeutend kleiner als 10^{10} möglichkeiten für ein Passwort. Deswegen konzentriert sich unsere Lösung die Optimierung von Bottleneck #1, der Berechnung der Hints, unter Berücksichtigung der Netzwerkauslastung.

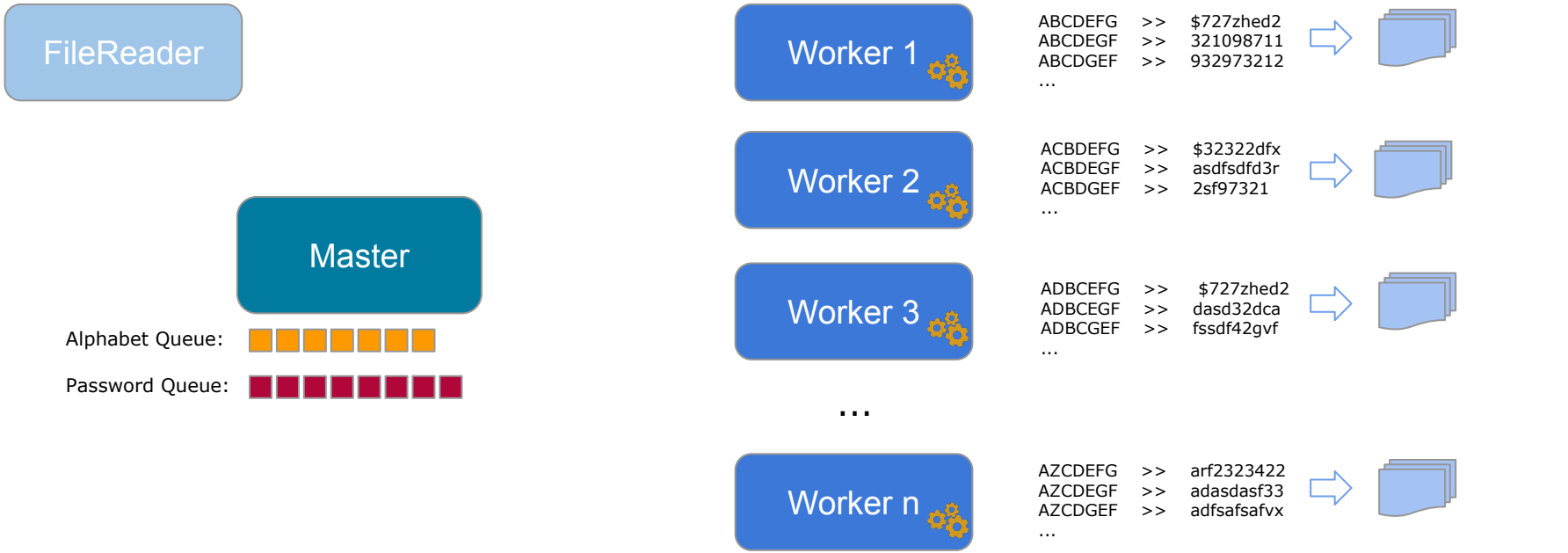
Generelles Konzept

Um die Berechnung der Hashes möglichst effizient zu gestalten, haben wir mehrere Ansätze verfolgt. Der wichtigste davon ist, dass die Berechnung der Hints Hashes wiederverwendbar ist. Statt sie also wegzuwerfen, werden sie in unserer Lösung in einer verteilten HashMap gespeichert. Das erlaubt es, direkt mithilfe des gesuchten Hashes den korrespondierenden Hint abzufragen. Anstatt die Hashes also für alle 100 Passwörter zu berechnen, ist nur eine einmalige Berechnung nötig. Das ist möglich, da sich der Speicherbedarf noch im technisch möglichen hält (<8GB). Dieser wird unter den teilnehmenden Worker Actors aufgeteilt, jeder bekommt einen Teilbereich aller möglichen Hashes für den der Worker eine HashMap baut und verwaltet. Nachdem jeder Worker seinen Bereich fertig berechnet hat, beginnt die Cracking Phase: jeder Worker versucht ein Passwort zu cracken und queried dafür erst seine eigene HashMap und dann die der anderen. Dafür sendet er eine einfache Anfrage, die nur beantwortet wird falls der Wert gefunden wird. Sobald er alle Hints von seinen Co-Workern erhalten hat, wird das Passwort gecrackt und danach das nächste zu crackende vom Master gefetcht. Somit wird großer Networktraffic vermieden, da eine gespeicherte HashMap niemals seinen Besitzer wechselt. Allerdings skaliert dieser quadratisch, da jeder Worker alle seine Coworker frägt, das kann bei vielen Knoten problematisch werden.

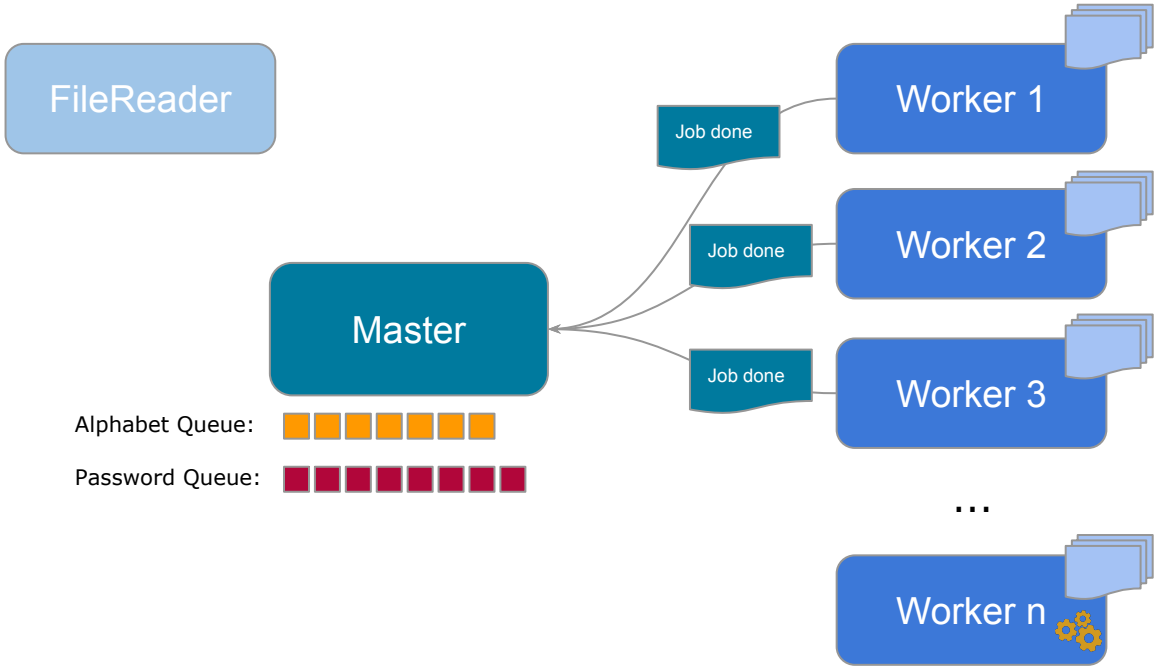
Hashing Phase - assign work



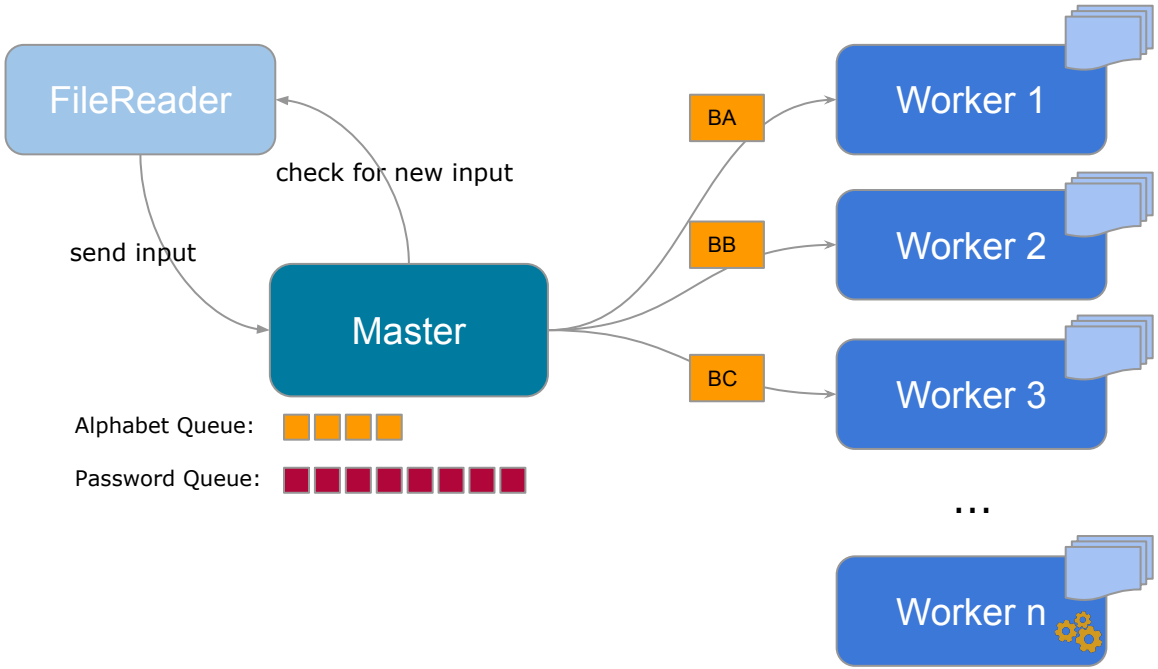
Hashing Phase - permutate and hash



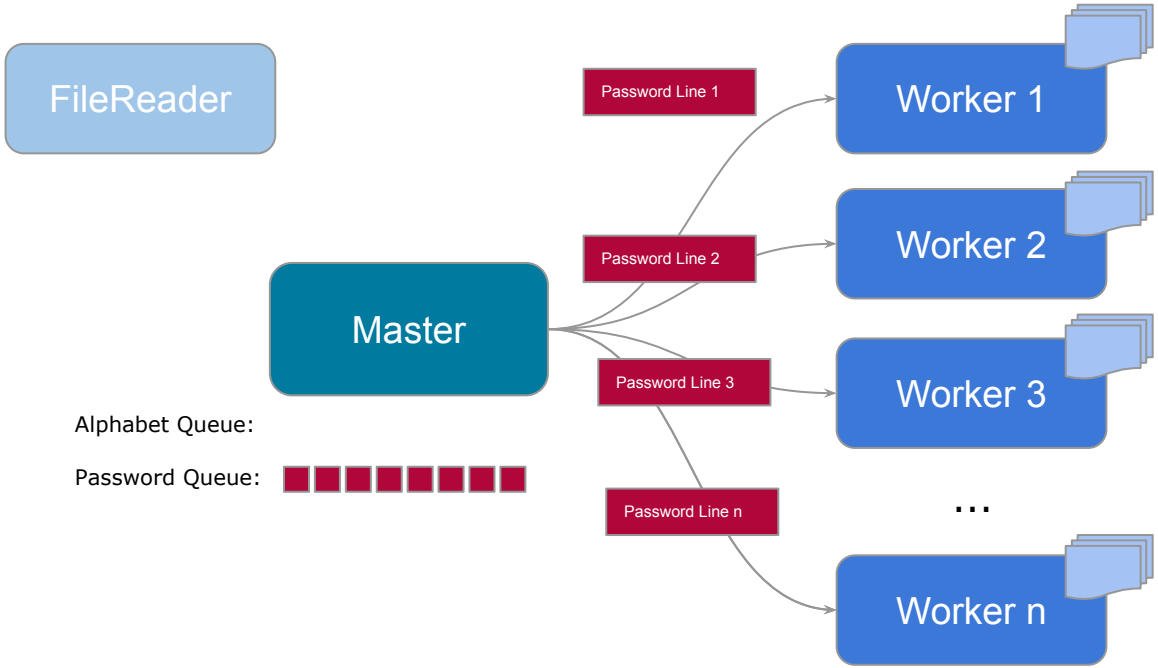
Hashing Phase - ask for new work



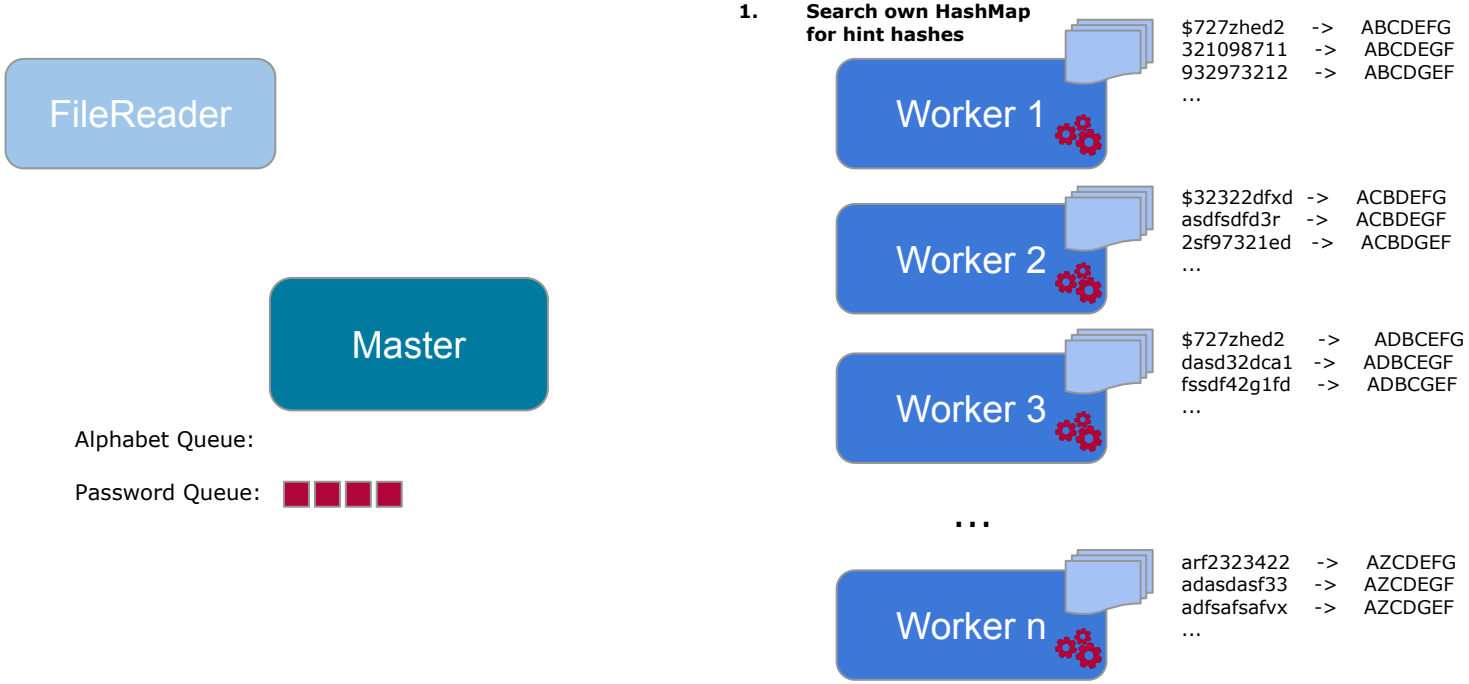
Hashing Phase - assign work



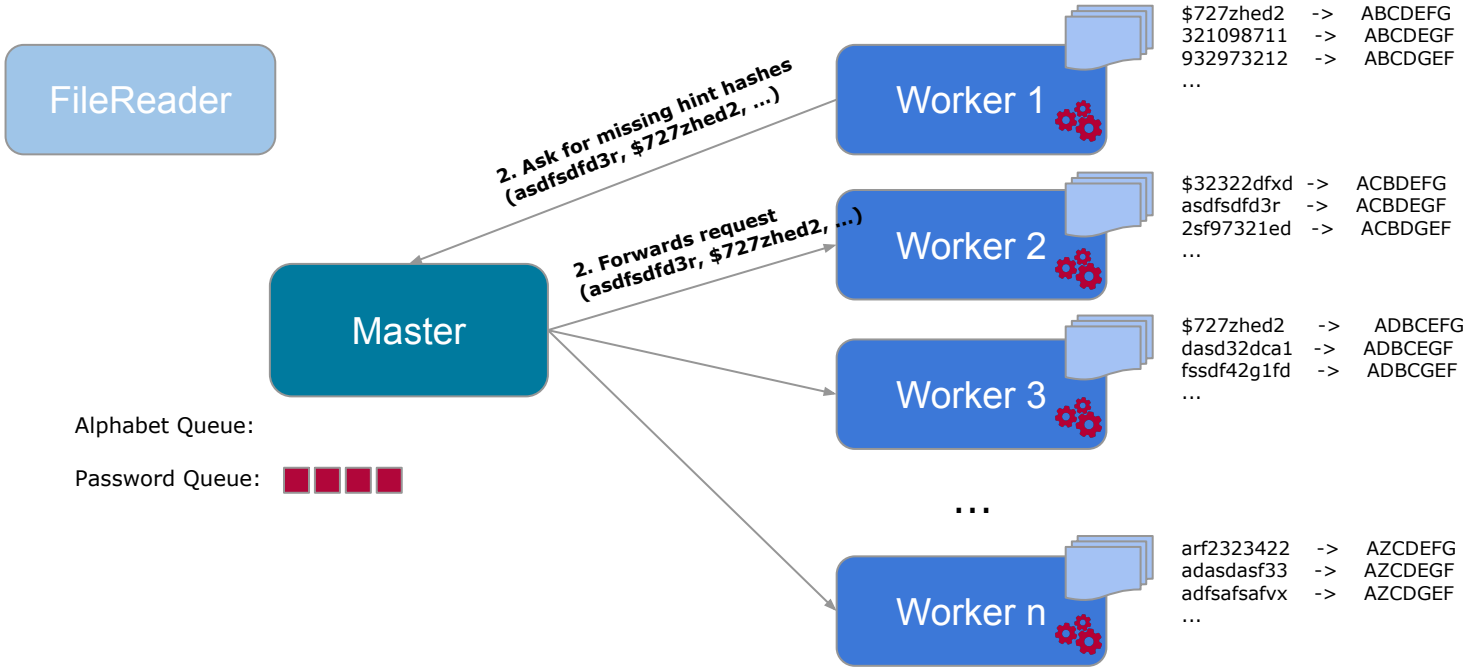
Cracking Phase



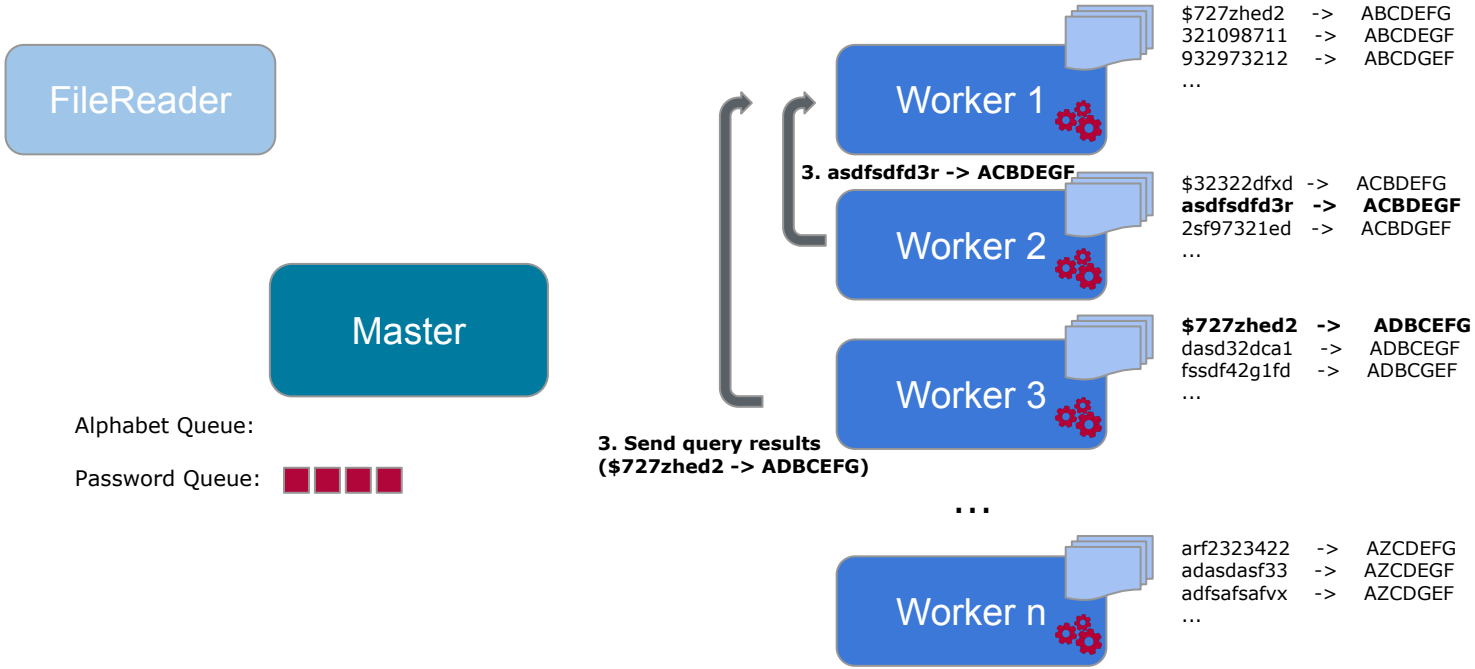
Cracking Phase - Workflow



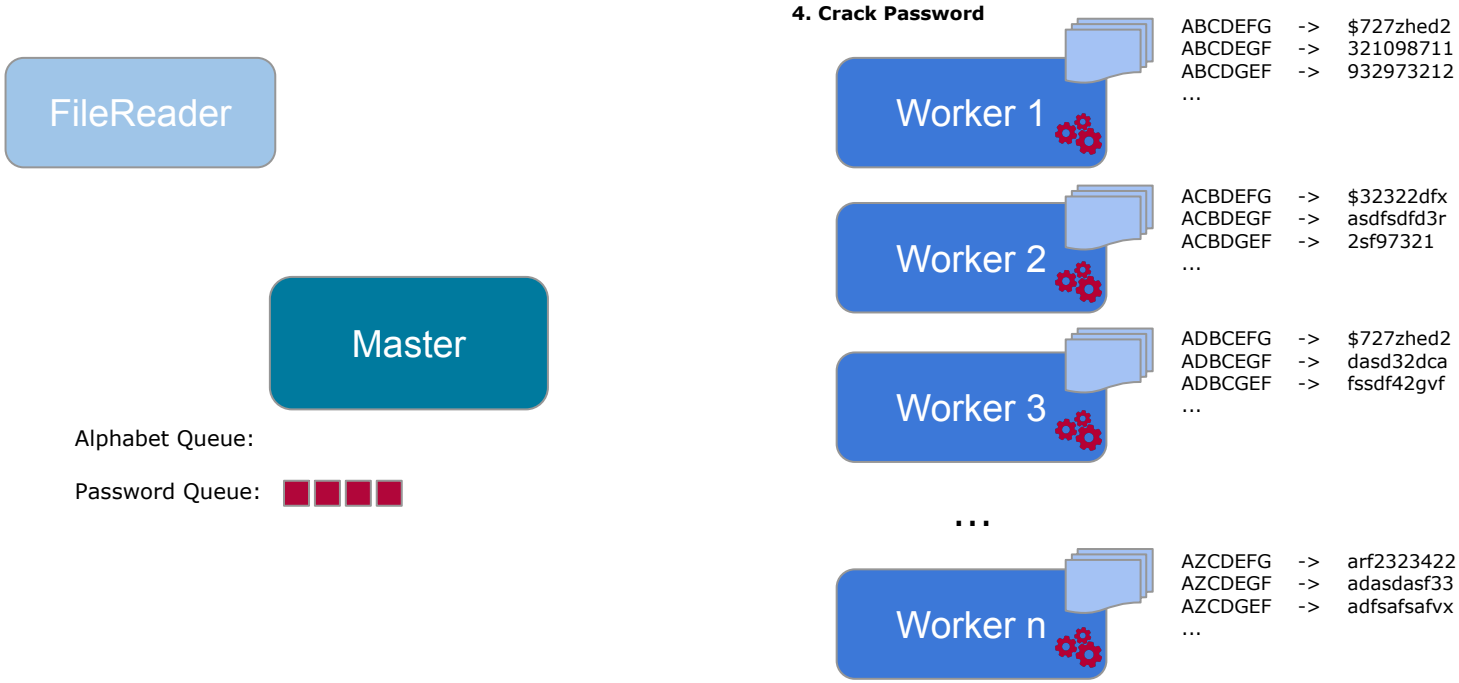
Cracking Phase - Workflow



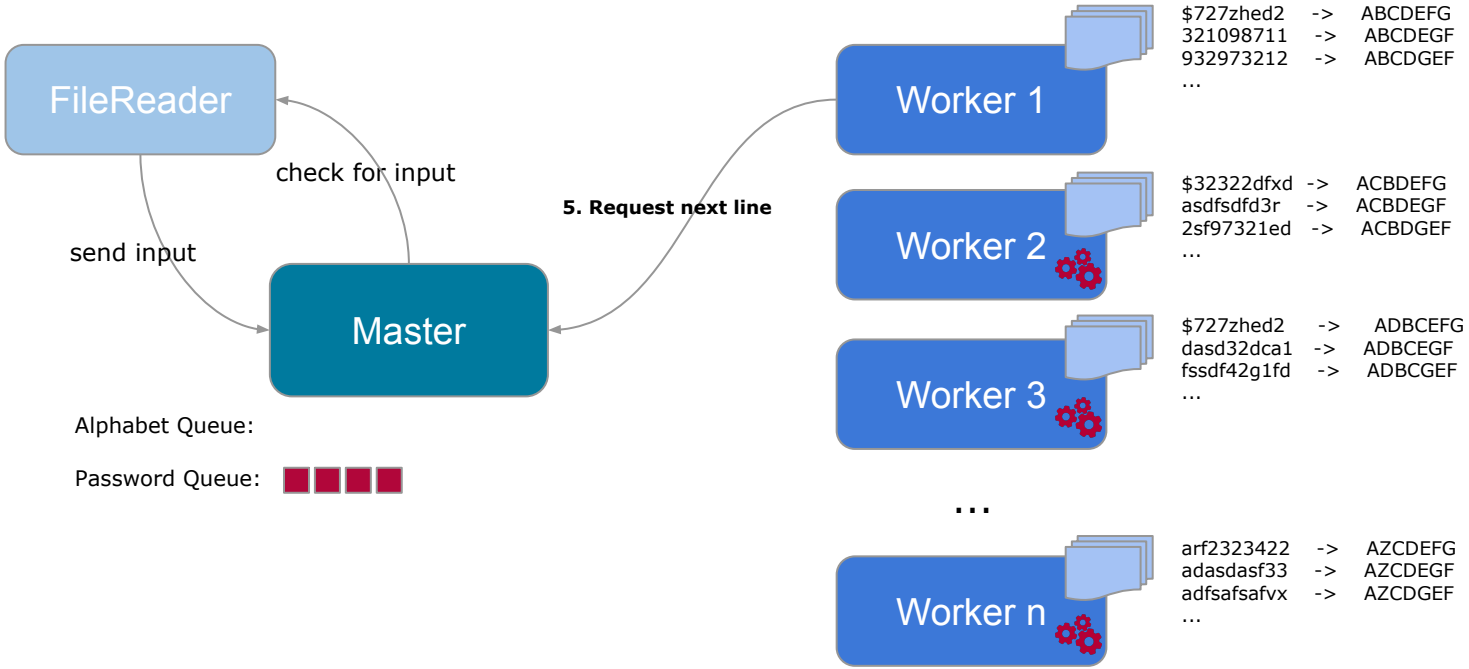
Cracking Phase - Workflow



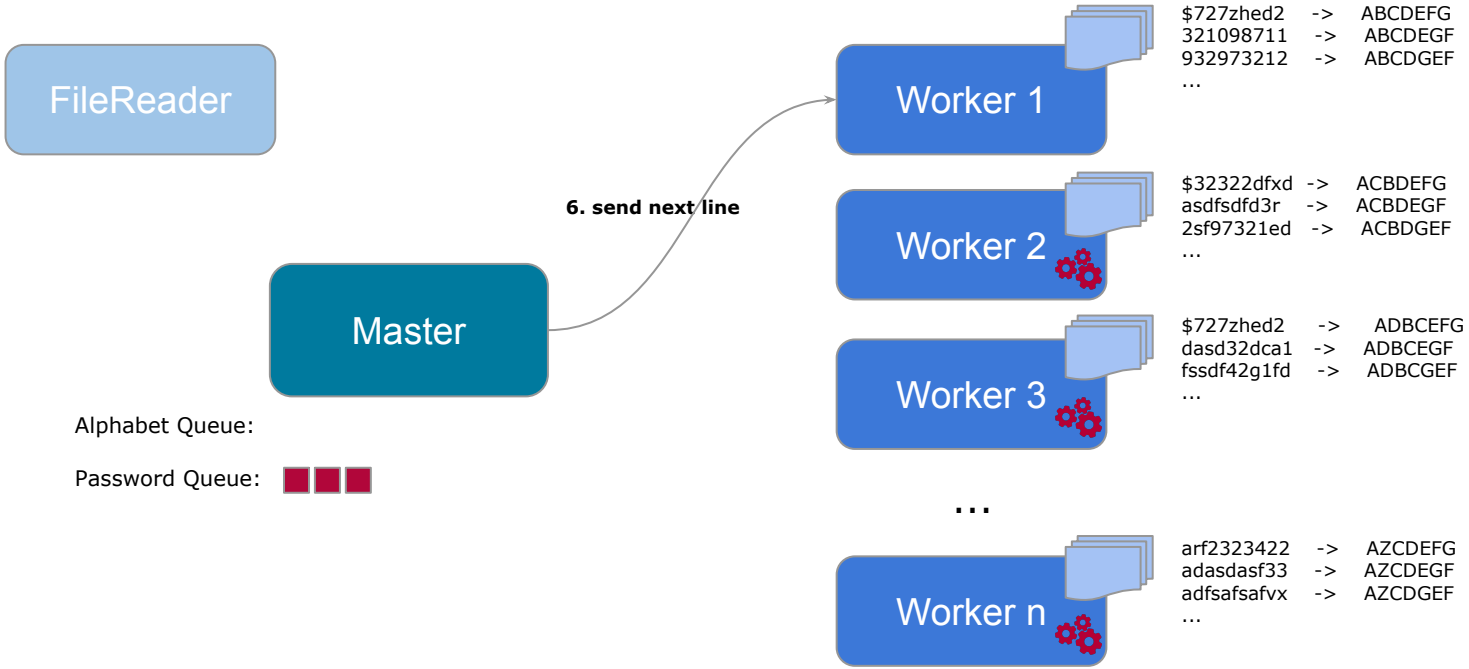
Cracking Phase - Workflow



Cracking Phase - Workflow



Cracking Phase - Workflow



Cracking Phase - Workflow