BIG DATA ET SCIENCE DE DONNÉES

TECHNOLOGIE HADOOP

Master 1 Intelligence Artificielle

Université de M'sila, Département d'Informatique

Dr Mehenni Tahar

2020-2021

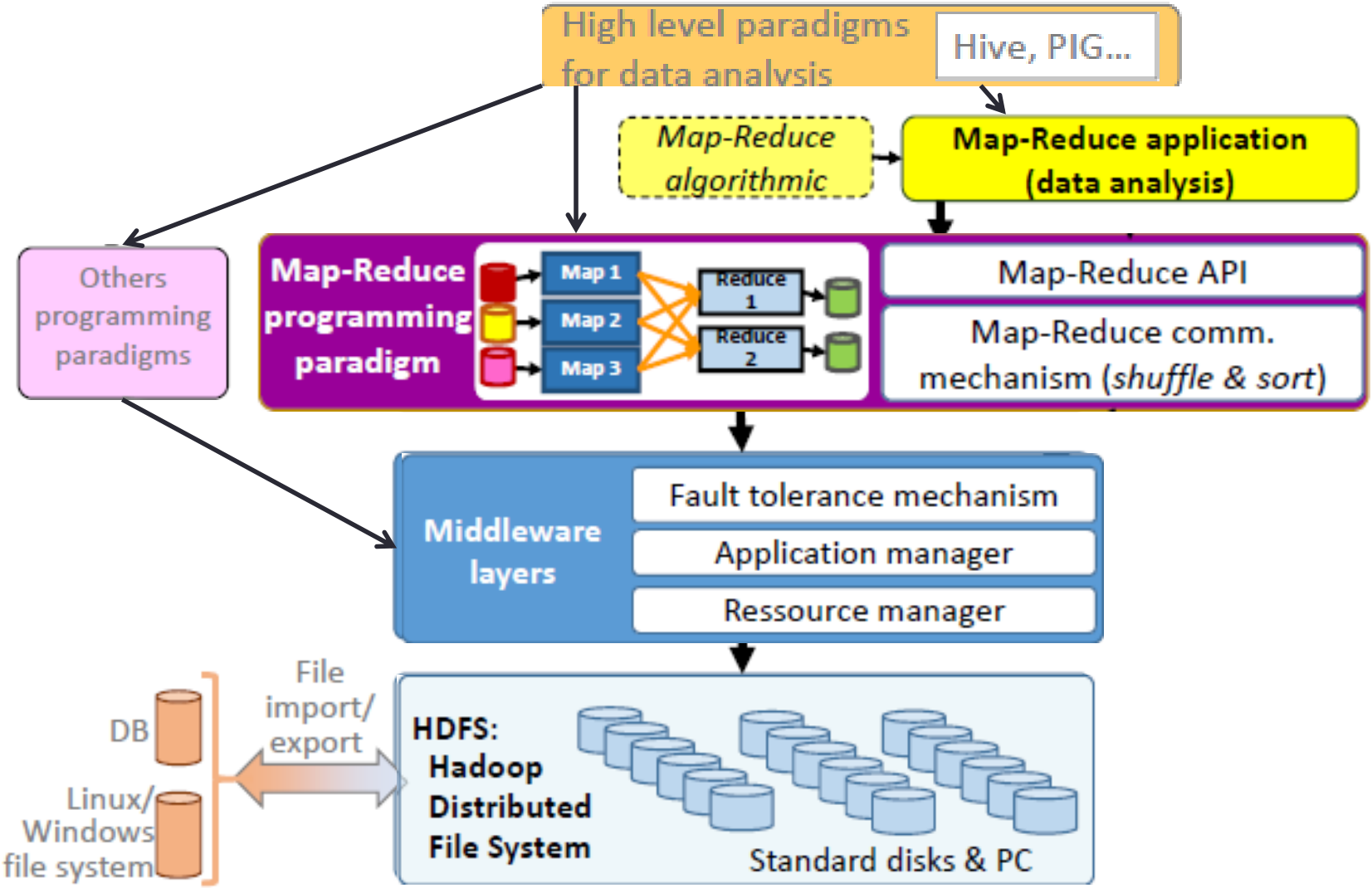
Introduction

- La programmation distribuée, sous la forme de processus répartis sur un ensemble (cluster, cloud, …) de machines, est la seule solution qui permet de traiter en temps raisonnable de gros problèmes et de gros volumes de données.
- Hadoop est:
- **Une plateforme** « *Big Data* » *open source* très mature destiné à faciliter la création d'applications distribuées et échelonnables à des milliers de machines. Les composantes principales de la plateforme sont: HDFS (*Hadoop's Distributed File System*) et MapReduce
- **Un éco-système** avec 100+ de sous-projets dont: Hbase, Hive, Pig; Sqoop, Zookeeper, Impala, ….

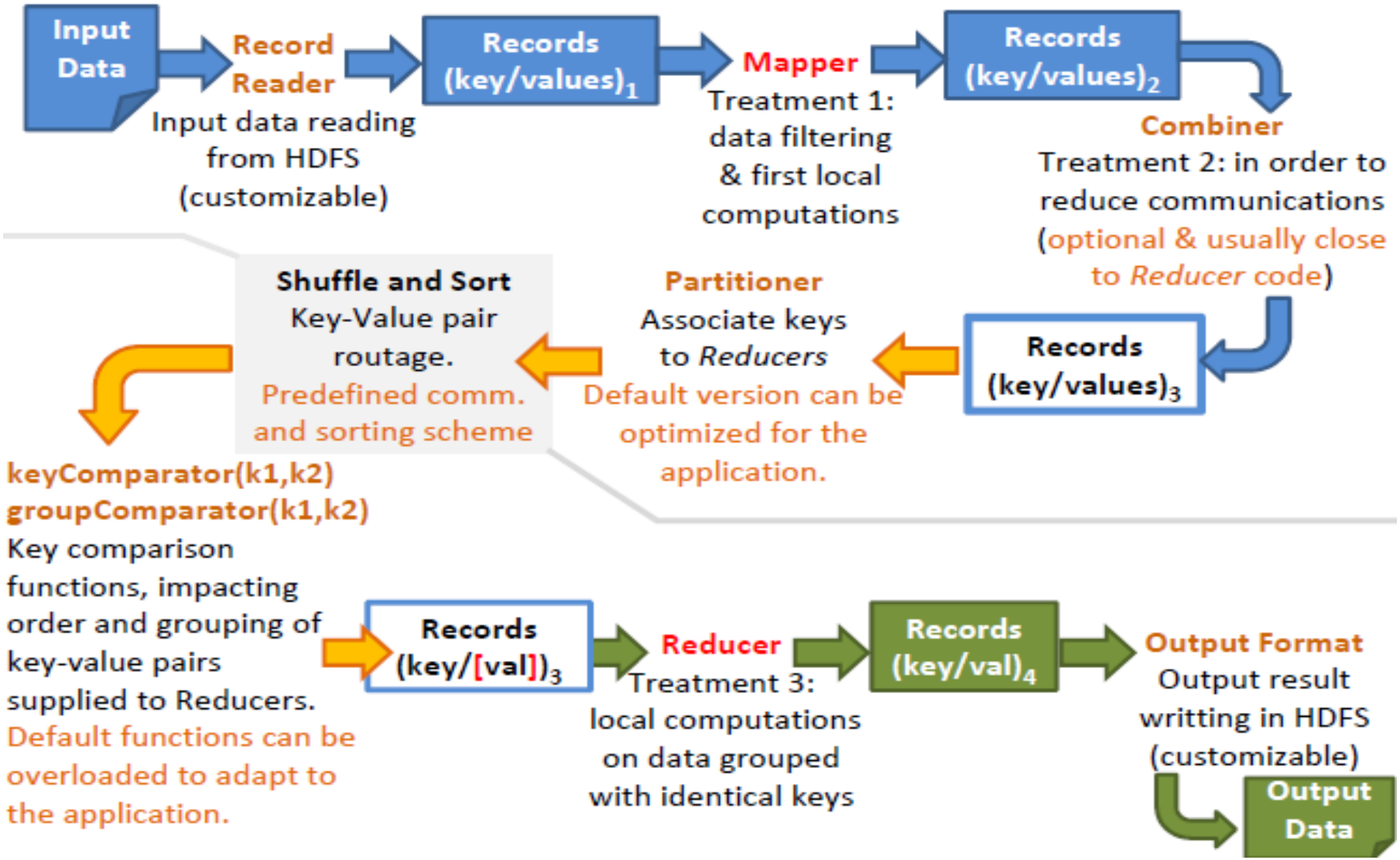
Bref historique

- **2002** **Doug Cutting**, Internet Archive search director, and Mike Cafarella, étudiant à la maitrise, démarrent **Nutch**, un *crawler* web *open-source* à l'échelle de l'internet. Nutch s'exécute sur 4 machines, mais requière surveillance humaine 24h / 24h.
- **2003** Google publie ***GFS: The Google File System***, un système de fichiers distribué pour applications big data.
- **2004** Google publie ***MapReduce: Simplified Data Processing on Large Clusters***, permettant la distribution du traitement sur une grappe de serveurs.
- **+ qq. mois** Cutting et Cafarella implémentent les publications de Google. Nutch s'exécute maintenant sur 20 – 40 machines.
- **2006** Doug Cutting est engagé par **Yahoo!**. Extraction des composantes de stockage (HDFS) et de traitement (MapReduce) pour former **Hadoop**.
- **2009** Hadoop est un **projet racine** (*top level*) d'Apache Software Foundation
- **2013** Yahoo! a un cluster de 42000 noeuds Hadoop

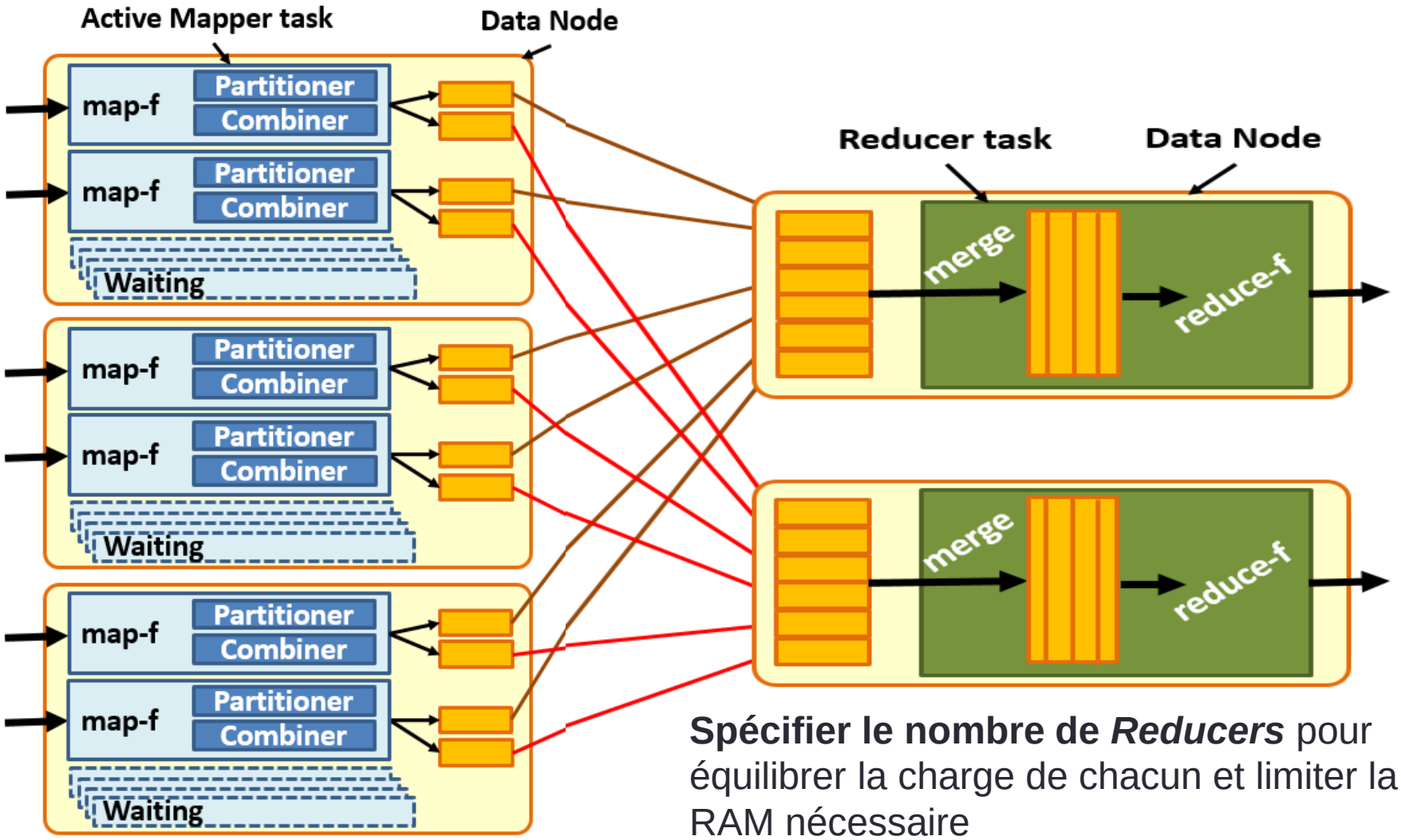
Principaux composants



Chaîne MapReduce d'Hadoop



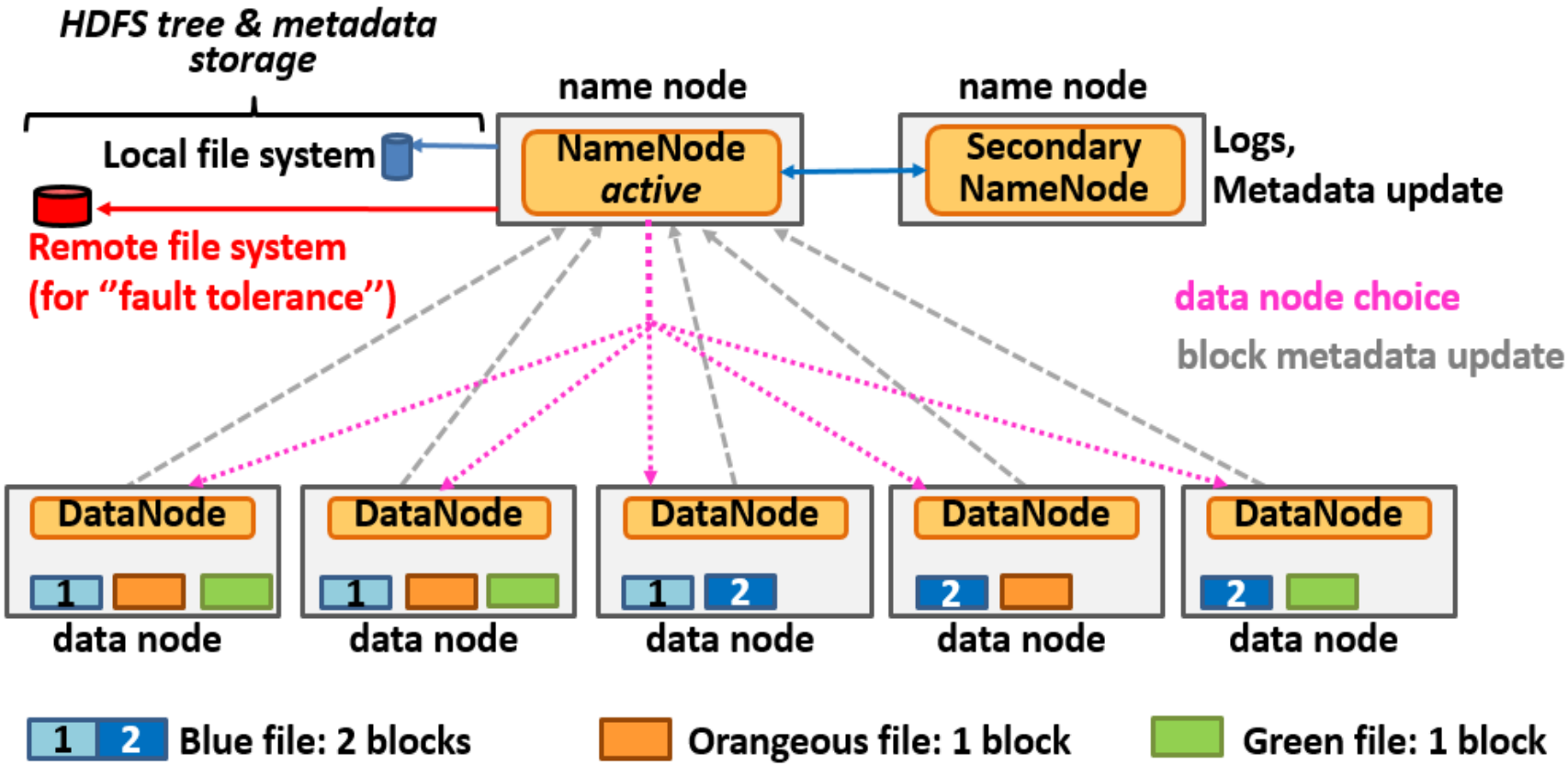
Déploiement d'un Map-Reduce en Hadoop



Spécifier le nombre de *Reducers* pour équilibrer la charge de chacun et limiter la RAM nécessaire

Système de fichiers distribué HDFS

- **Principes d'HDFS**



Système de fichiers distribué HDFS

- **Principes d'HDFS**

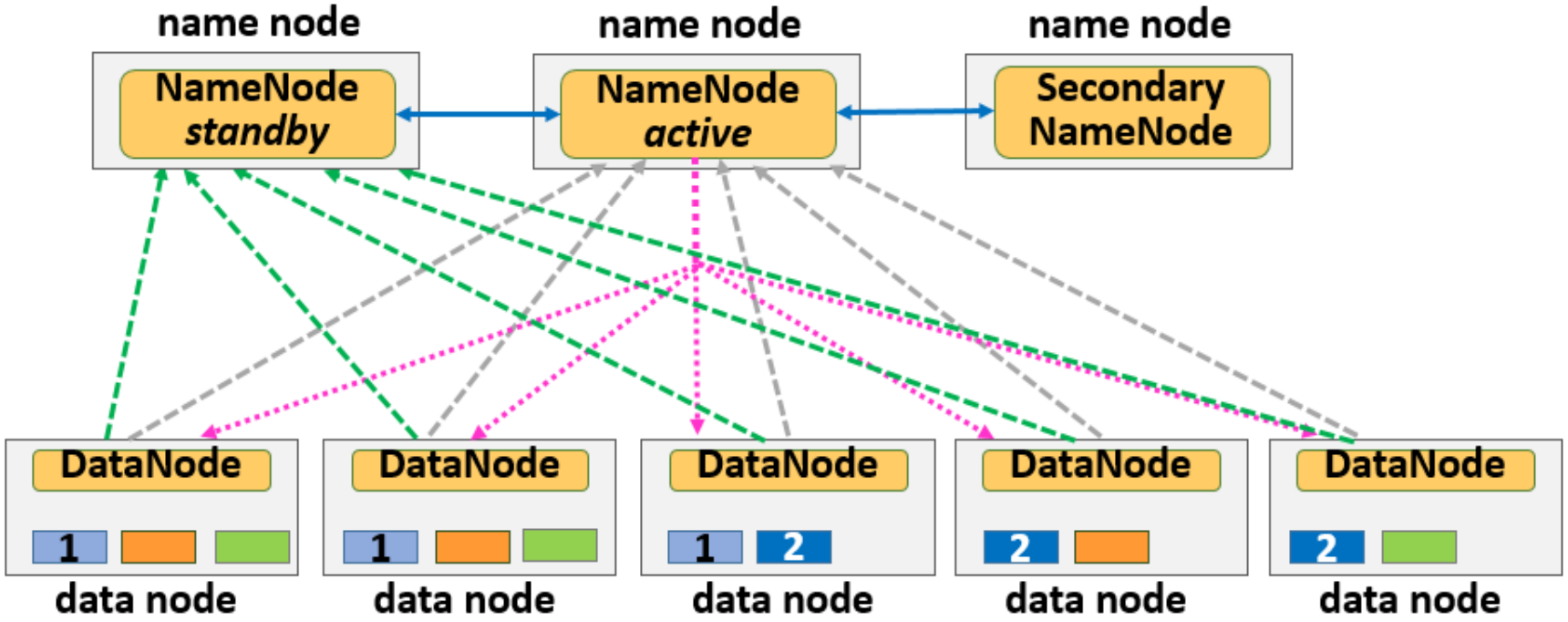
- Découpage en blocs, réplication et le stockage distribué des fichiers HDFS → performances d'accès et forte tolérance aux pannes.
- *NameNode actif* établit et conserve une cartographie de la répartition de tous les fichiers stockés dans HDFS.
- Les insertions de nouveaux fichiers et suppressions d'anciens donnent lieu à des modifications de la cartographie qui sont stockées sous forme de *logs* à la fois dans le *NameNode actif* et dans le *NameNode secondaire*.
- Une cartographie à jour est obtenue en appliquant les évolutions décrites dans les *logs* aux meta-données du *NameNode actif*.
- Chaque fichier est découpé en blocs, typiquement de 64Mo ou 128Mo, et chaque bloc est répliqué n fois, habituellement 3 fois. (fichiers bleu, orange et vert sur la figure).
- Dans un cluster Hadoop de grande taille, les réplicats d'un même bloc doivent être stockés sur des noeuds situés dans des racks différents (donc électriquement indépendants).
- Lors de la création d'un nouveau fichier le NameNode actif va répartir les réplicats de ses blocs sur les différents noeuds de données disponibles (flèches roses sur la figure), et chaque noeuds de données va maintenir le NameNode actif au courant de son état, et du succès ou de l'échec de ses créations de blocs (flèches grises sur la figure). Le NameNode actif maintiendra ainsi une connaissance à jour du système de fichier HDFS.

Système de fichiers distribué HDFS

- **Tolérance aux pannes et haute disponibilité d'HDFS**
- Une panne du *NameNode actif* pourrait donc rendre le système de fichiers HDFS totalement inexploitable.
- **Deux mécanismes** complémentaires:
  - **Tolérance aux pannes :** Les meta-données du *NameNode actif* sont régulièrement sauvegardées sur un système de fichiers local (accès rapide) mais aussi sur un système distant (voir figure précédent)
  - **Haute disponibilité :** il est possible de dupliquer le *NameNode actif*, et de créer un *Name-Node standby* qui reçoit et stocke en permanence les mêmes meta-données et logs que le *NameNode actif.*

Système de fichiers distribué HDFS

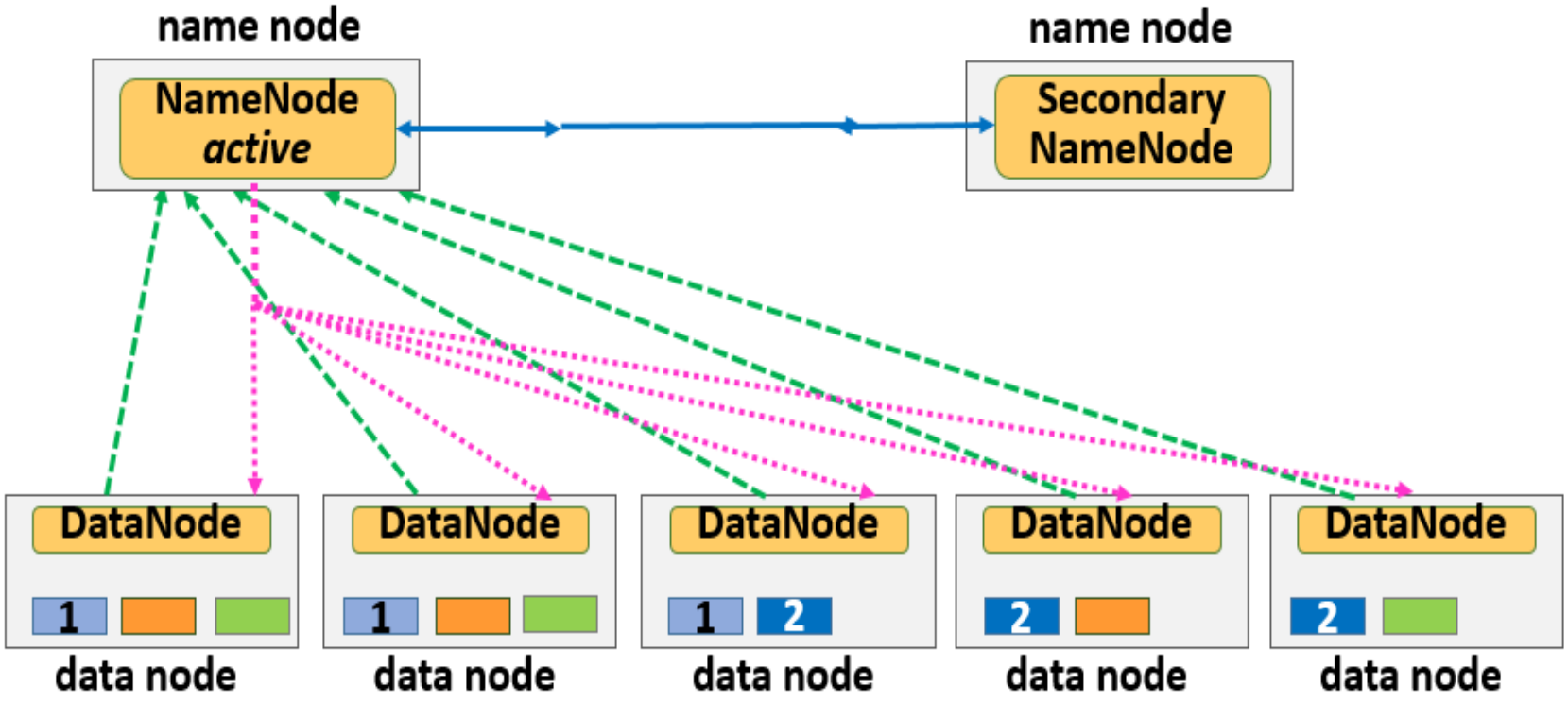
- **Tolérance aux pannes et haute disponibilité d'HDFS**



Solution à haute disponibilité pour HDFS en cas de panne sur le NameNode

Système de fichiers distribué HDFS

- **Tolérance aux pannes et haute disponibilité d'HDFS**



Solution à haute disponibilité pour HDFS après apparition d'une panne sur le NameNode

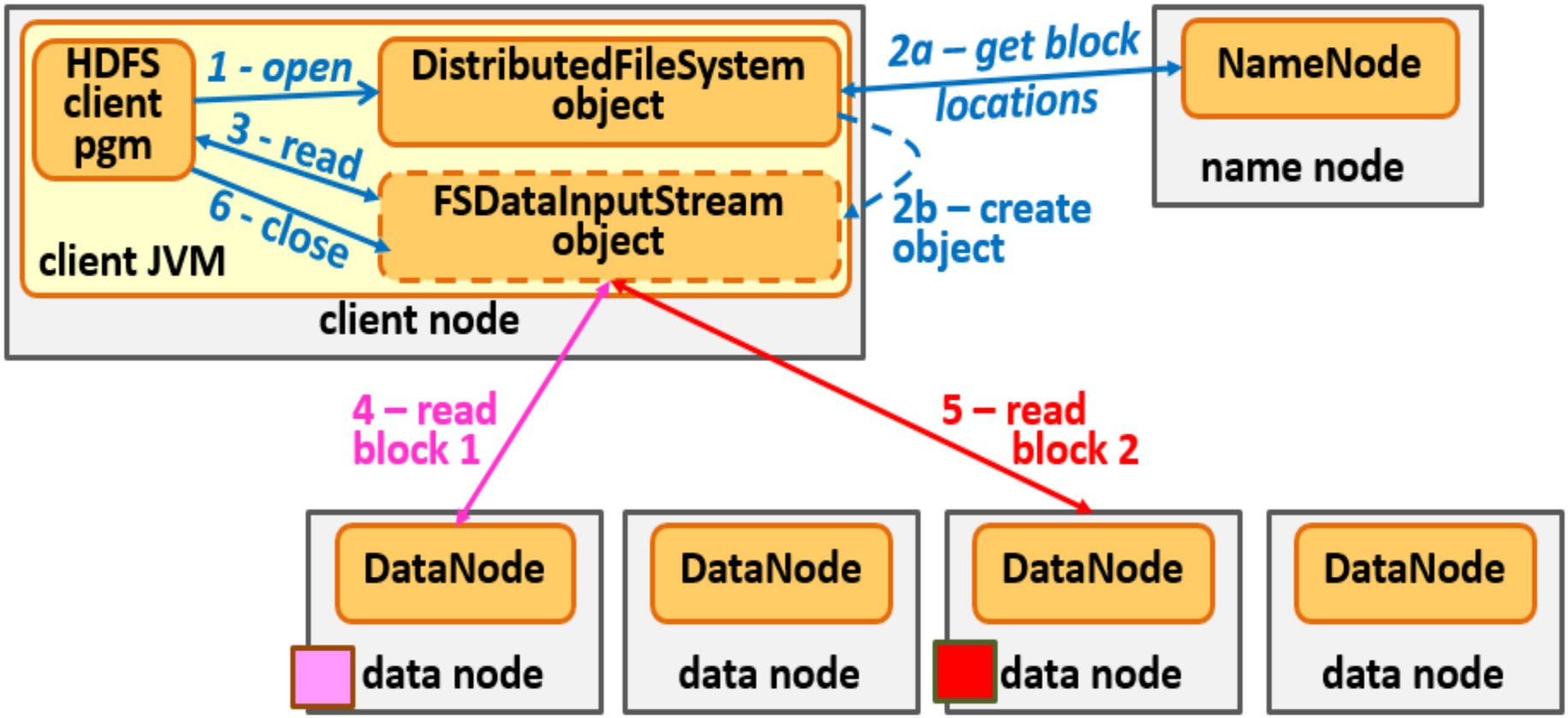
Système de fichiers distribué HDFS

- **Mécanismes de lecture d'HDFS**

- **Etape 0:** Le code client commence par créer un objet local de la classe *DistributedFileSystem*, qui va agir comme un *stub* ou *proxy* avec le système de fichier HDFS.
- **Etape 1:** Le code client s'adresse donc à cet objet local et demande à ouvrir un fichier
- **Etape 2:** Demande de localisation du fichier:
  - **Etape 2a:** Le *stub* s'adresse alors au *NameNode* d'HDFS pour connaître l'emplacement des réplicats de tous les blocs du fichier.
  - **Etape 2b:** Ensuite le *stub* crée un autre objet local, de la classe *FSDataInputStream*, qui va agir comme un lecteur spécialisé dans la lecture du fichier visé, ayant connaissance des noeuds à contacter.
- **Etape 3:** Le client va faire alors des opérations *read* sur ce lecteur spécialisé.
- **Etape 4:** Le lecteur va interroger un des noeuds stockant le premier bloc du fichier.
- **Etape 5:** Une fois le premier bloc lu, si les opérations de lecture continuent de la part du client, le lecteur spécialisé va interroger un nœud contenant le second bloc et ainsi de suite.
- **Etape 6:** Finalement, le client demande à fermer le fichier ouvert en lecture auprès du lecteur spécialisé. A noter que le lecteur spécialisé vérifie l'intégrité des données lues (calculs de checksum) et signale toute anomalie au *stub*, qui les retransmet au *NameNode*.

Système de fichiers distribué HDFS

- **Mécanismes de lecture d'HDFS**



Système de fichiers distribué HDFS

- **Mécanismes d'écriture d'HDFS**

- **Etape 0:** au début, le client crée un objet local *DistributedFileSystem* servant de *stub* ou *proxy* envers le système de fichier HDFS.
- **Etape 1:** Le client peut demander au *stub* de créer un nouveau fichier HDFS (opération *create*).
- **Etape 2:** Demande de localisation des noeuds d'accueil des futurs blocs du fichier
- **Etape 2a:** Le *stub* interroge le *NameNode* pour savoir où écrire les blocs du fichier, le *NameNode* répond au *stub*
- **Etape 2b:** Le *stub* crée en local un objet *FSDataOutputStream* qui va jouer le rôle d'un écrivain spécialisé dans l'écriture du fichier.
- **Etape 3:** Par la suite, le client s'adresse localement à cet écrivain spécialisé pour lui demander d'écrire des données dans ce fichier (opération write).

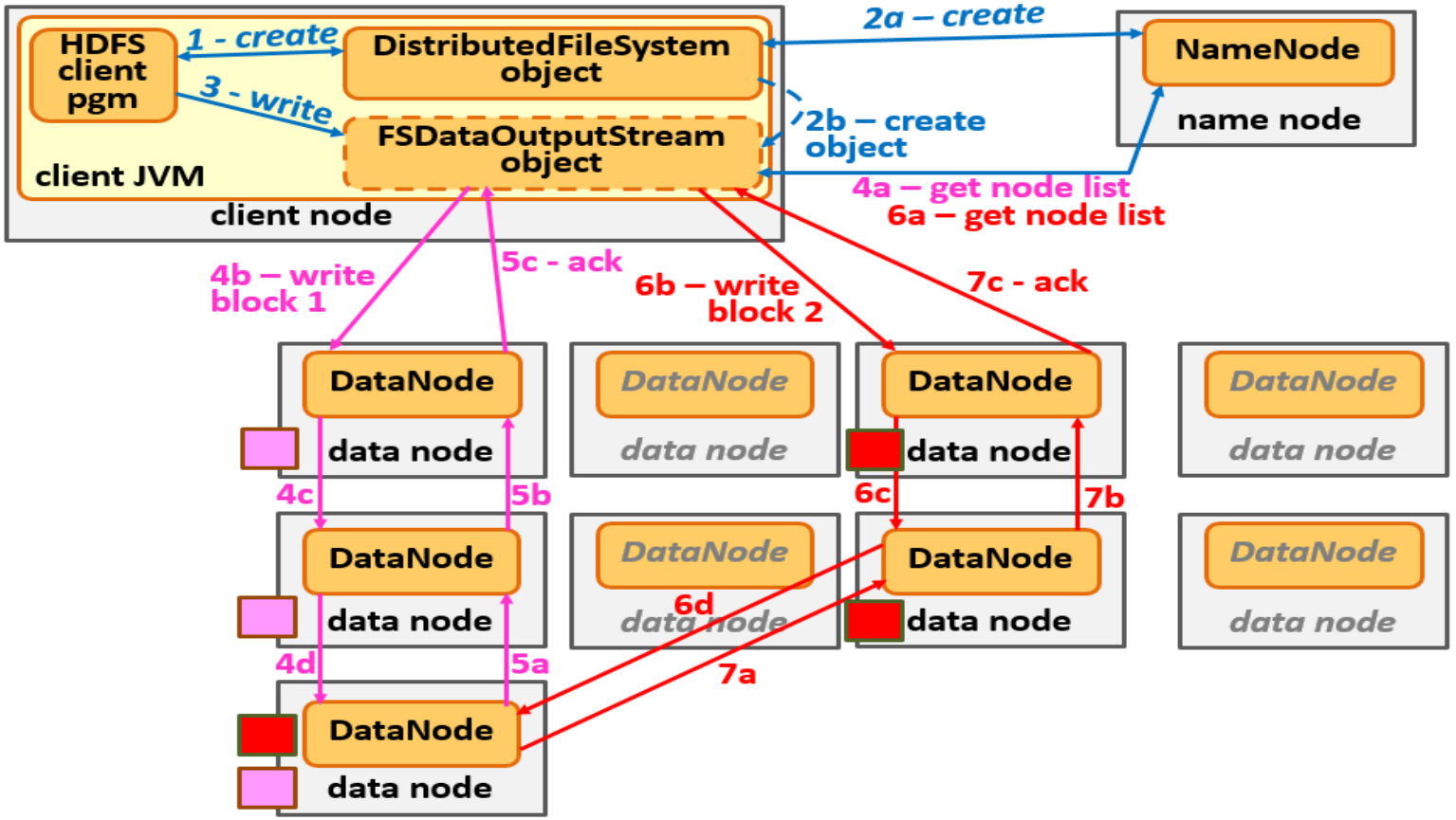
Système de fichiers distribué HDFS

- **Mécanismes d'écriture d'HDFS**

- **Etape 4**: L'écrivain dialogue alors avec le NameNode d'HDFS pour savoir où créer un premier bloc et ses réplicats **(étape 4a)**, puis commence à écrire le premier réplicat du premier bloc sur un noeud de données **(étape 4b)**. Un mécanisme de pipeline se met alors en place : le noeud du premier réplicat retransmet ses données au noeud choisi pour héberger le deuxième réplicat **(étape 4c)**, qui les retransmet lui même au noeud choisi pour héberger le troisième réplicat **(étape 4d)**, et le processus se poursuit si plus de 3 réplicats par bloc sont spécifiés.
- **Etape 5:** Quand l'écriture sur le dernier réplicat est terminée, un message d'*acknowledge* remonte vers le noeud du réplicat précédent **(étape 5a)**, et ainsi de suite jusqu'à atteindre le noeud du premier réplicat **(étapes 5b)**. Ce noeud retourne alors un *acknowledge* à l'écrivain spécialisé **(étape 5c)**, et si tout s'est bien passé, l'écrivain enchaînera en traitant la demande suivante d'écriture de données.
- **Etapes 6 et 7:** Quand le premier bloc est plein, l'écrivain demande à nouveau au *NameNode* d'HDFS un ensemble de noeuds où écrire un deuxième bloc et ses réplicats **(étape 6a)**, et le processus d'écriture pipelinée se reproduit **(étapes 6b à 6d, puis 7a à 7c)**.
- **Etape8:** Une fois toutes les écritures de données terminées, le client demande à l'écrivain de refermer le nouveau fichier (opération *close*).
- **Etape 9:** Celui-ci en informe alors le *stub*, qui en informe à son tour le *NameNod*e d'HDFS, qui met à jour ses metadonnées avec un nouveau fichier dans sa cartographie.

Système de fichiers distribué HDFS

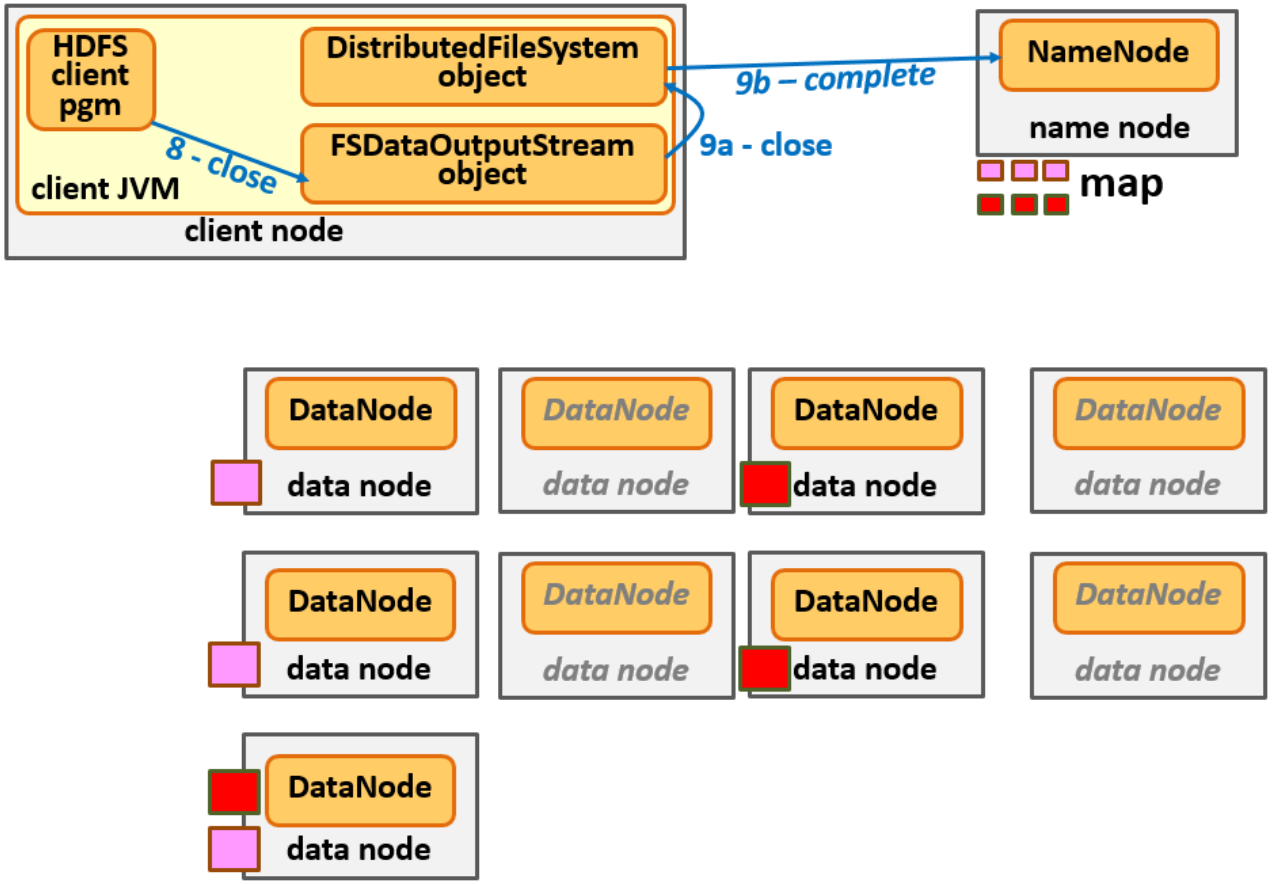
- **Mécanismes d'écriture d'HDFS**



Fonctionnement d'HDFS pour l'écriture de nouvelles données dans HDFS

Système de fichiers distribué HDFS

- **Mécanismes d'écriture d'HDFS**



Fonctionnement d'HDFS à la fin de l'écriture d'un nouveau fichier

Système de fichiers distribué HDFS

- **Accès séquentiel ou parallèle aux données stockées en HDFS?**
- Les mécanismes de lecture et d'écriture de fichiers HDFS présentées précédemment sont très séquentiels, menés par un programme client qui écrit ou lit des données dans le système HDFS les unes après les autres.
- Cette démarche permet par exemple d'importer ou d'exporter des données entre le système de fichier d'Hadoop et celui d'un simple PC,
- **MAIS** elle n'est pas adaptée pour exploiter intensivement des données déjà présentes dans des fichier HDFS.
- **Au contraire,** une application Map-Reduce va déployer plusieurs processus sur différents nœuds de données et accéder en parallèle à divers blocs d'un même fichier HDFS (ou de plusieurs fichiers HDFS).
- L'architecture d'une application Map-Reduce est conçue pour tirer un maximum de performances d'un système de fichier distribué comme HDFS.

Système de fichiers distribué HDFS

- **Gestion de ressources d'Hadoop - version 1**
- Lors des lectures et écritures de fichiers HDFS, le programme client crée un objet local agissant comme un *stub* ou *proxy*.
- Avec la Version 1 de l'architecture Hadoop, il s'agit d'un objet de la classe *Job*, à partir duquel le programme client configure, lance et attend la fin d'une opération complète de Map-Reduce.
- Lorsqu'un client Hadoop soumet un job pour exécution :
  - Le JobTracker demande au NameNode où se trouvent les blocs correspondants aux noms de fichiers donné par le client.
  - Le job tracker détermine quels sont les noeuds les plus appropriés pour exécuter les traitements en tenant compte de leur Co-localisation ou proximité.
  - Le Job Tracker envoie au Task Tracker sélectionné le travail à effectuer.
  - Exécution des tâches Map et Reduce.
- **Limitation du Map-Reduce d'Hadoop version 1:** Le *JobTracker* doit gérer tous les jobs en cours d'exécution, en plus de choisir les ressources des nouveaux jobs soumis. Au final, le JobTracker constitue un goulot d'étranglement quand la taille du cluster et le nombre de jobs augmentent.

- Une seconde version d'Hadoop a donc été développée avec une couche plus évoluée de gestion des applications distribuées (voir section suivante).

Système de fichiers distribué HDFS

- **Gestion de ressources d'Hadoop - version 2 (YARN)**

- La version 1 est limitée pour des clusters Hadoop avec plusieurs milliers de nœuds → Nécessité de disposer d'un nouvel environnement qui permet :
  - d'éviter un goulot d'étranglement au niveau du *JobTracker* qui gère toutes les tâches en cours d'exécution et centralise leurs informations de monitoring, afin de renforcer la capacité de passage à l'échelle,
  - de supporter l'exécution d'autres types d'applications distribuées qu'uniquement des MapReduce, donc de pouvoir déployer des gestionnaires d'applications variés.
- En 2010 Yahoo ! a démarré le développement d'une nouvelle couche d'exécution et de contrôle des applications Hadoop au dessus d'HDFS. Cette version fut appelé YARN : *Yet Another Resource Negotiator*.
- Elle distingue clairement :
  - les fonctionnalités de gestion et d'allocation de ressources de calculs, c'est-à-dire l'identification des noeuds de données qui hébergeront les traitements,
  - les fonctionnalités de lancement et de monitoring des tâches d'une application distribuée, sur les ressources allouées.
- Comparé à la version 1, le *JobTracker* et les *TaskTrackers* disparaissent, au profit d'un *ResourceManager* et de plusieurs *MRAppManager* (si on considère des applications Map-Reduce).

Système de fichiers distribué HDFS

- **Différences entre MapReduce V1 (MRv1) et MapReduce V2 (MRv2)**
- Dans MRv1, Map Reduce est responsable à la fois du traitement et de la gestion des clusters, tandis que dans MRv2, le traitement est pris en charge par d'autres modèles de traitement et YARN est responsable de la gestion des clusters.
- MRv2 est plus performant que MRv1 avec près de 10 000 noeuds par cluster, contrairement à MRv1 qui est limité à 4 000.
- Dans MRv1, le NameNode est seul point d'échec (Problème de disponibilité -SPoF-). Cependant, dans le cas de MRv2, chaque fois qu'un NameNode échoue, il est configuré pour la récupération automatique.
- MRv1 fonctionne sur le concept des slots alors que MRv2 travaille sur le concept des conteneurs et peut également exécuter des tâches génériques.