BIG DATA

Siham Yousfi 2022

Ecole d'été: Patrimoine et numérique. 9-15 juin 2022

PLAN

- Big Data Use case
- Big Data: Contexte d'émergence
- Les caractéristiques des Big Data
- Le traitement des Big Data
- Framework de traitement des Big Data: Hadoop
  - HDFS
  - MapReduce

USE CASE

- Superviser le trafic
  - **Description**: identifier les points et les heures précis de congestions
  - **Opportunités**: plusieurs sources de données
  - **Objectif**: analyser les données sur le trafic pour identifier les congestions

USE CASE

- Superviser le trafic
  - **Description**: identifier les points et les heures précis de congestions
  - **Opportunités**: plusieurs sources de données
  - **Objectif**: analyser les données sur le trafic pour identifier les congestions
  - **Contraintes**:

USE CASE

- Superviser le trafic
  - **Description**: identifier les points et les heures précis de congestions
  - **Opportunités**: plusieurs sources de données
  - **Objectif**: analyser les données sur le trafic pour identifier les congestions
  - **Contraintes**:
    - Différents types de données

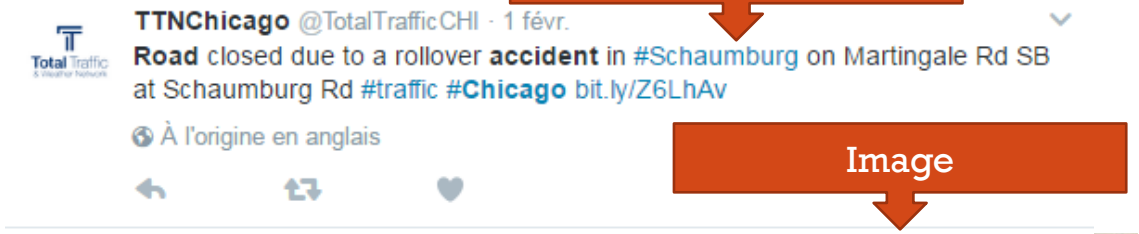
Loop detectors

Texte semi-structuré

Id	Date	Heure	Occupation	Débit	Charge	Vitesse
1001	01/02/2017	07:15	6	1380	29	47
1001	01/02/2017	07:30	11	1932	41	28
1001	01/02/2017	07:45	51	2552	73	8
1001	01/02/2017	08:00	52	2716	75	8
1001	01/02/2017	08:15	41	2736	69	11
1001	01/02/2017	08:30	38	2724	68	12

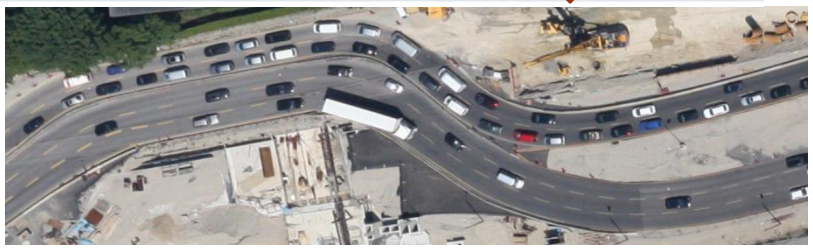
Twitter

Texte non-structuré



Image

Images UAV

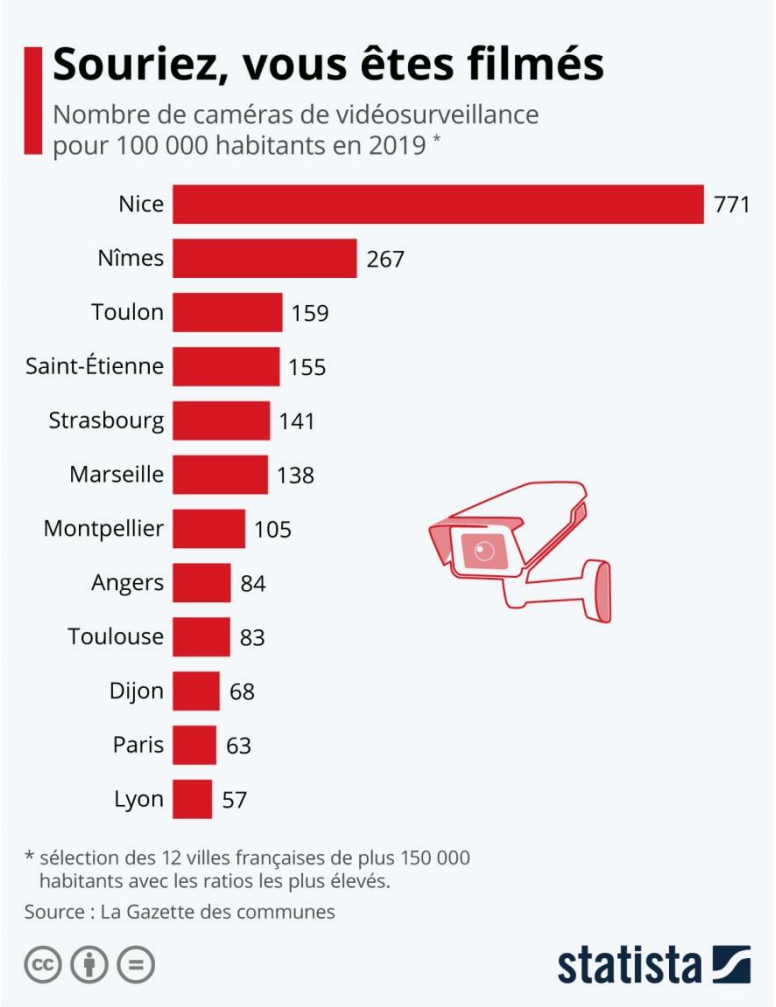


Vidéo

CCTV

USE CASE

- Superviser le trafic
  - **Description**: identifier les points et les heures précis de congestions
  - **Opportunités**: plusieurs sources de données
  - **Objectif**: analyser les données sur le trafic pour identifier les congestions
  - **Contraintes**:
    - Différents types de données
    - Grande quantité de données



Volume selon le nombre et la résolution des caméras

USE CASE

- Superviser le trafic
  - **Description**: identifier les points et les heures précis de congestions
  - **Opportunités**: plusieurs sources de données
  - **Objectif**: analyser les données sur le trafic pour identifier les congestions
  - **Contraintes**:
    - Différents types de données
    - Grande quantité de données
    - Vitesse de données

CCTV
25 frames par seconde

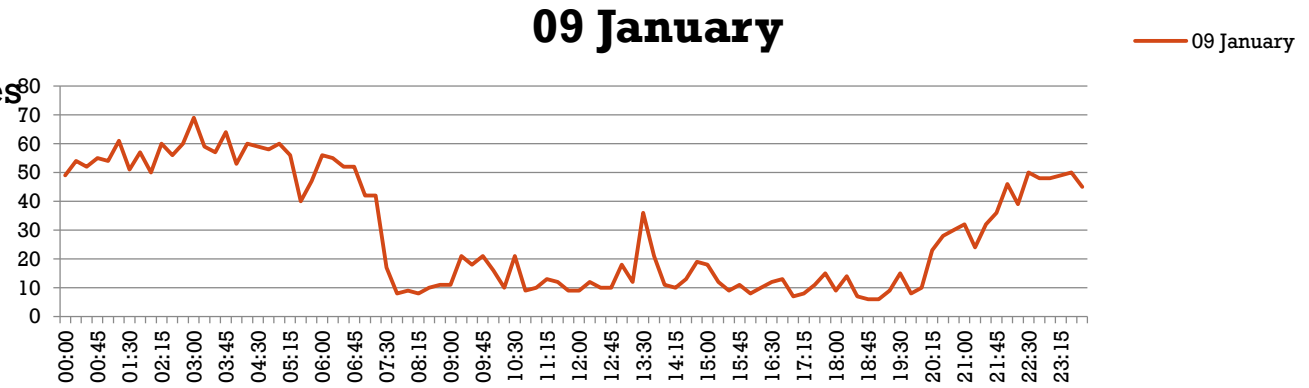
Capteurs de trafic
Toutes les 5 à 15 min

GPS
Toutes les secondes

Réseau social
??

USE CASE

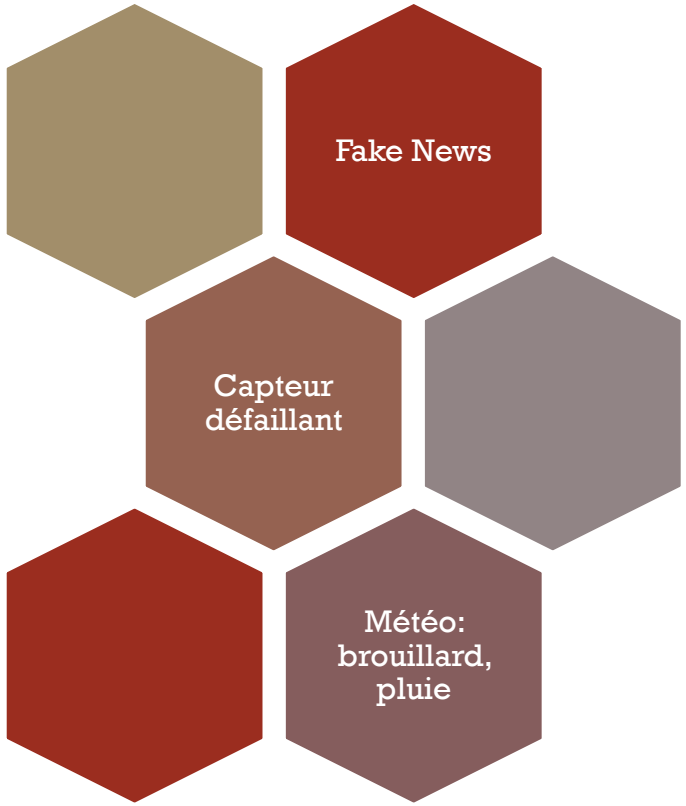
- Superviser le trafic
  - **Description**: identifier les points et les heures précis de congestions
  - **Opportunités**: plusieurs sources de données
  - **Objectif**: analyser les données sur le trafic pour identifier les congestions
  - **Contraintes**:
    - Différents types de données
    - Grande quantité de données
    - Vitesse de données
    - Évolution des données dans le temps



Vitesse de trafic à BD zrektouni à Casablanca un jour ouvrable (09/10/2019)

USE CASE

- Superviser le trafic
  - **Description**: identifier les points et les heures précis de congestions
  - **Opportunités**: plusieurs sources de données
  - **Objectif**: analyser les données sur le trafic pour identifier les congestions
  - **Contraintes**:
    - Différents types de données
    - Grande quantité de données
    - Vitesse de données
    - Évolution des données dans le temps
    - Peut-on faire confiance à toutes les donnée?



BIG DATA: CONTEXTE D'ÉMERGENCE

- Systèmes traditionnels existants

1980-1990: SGBD

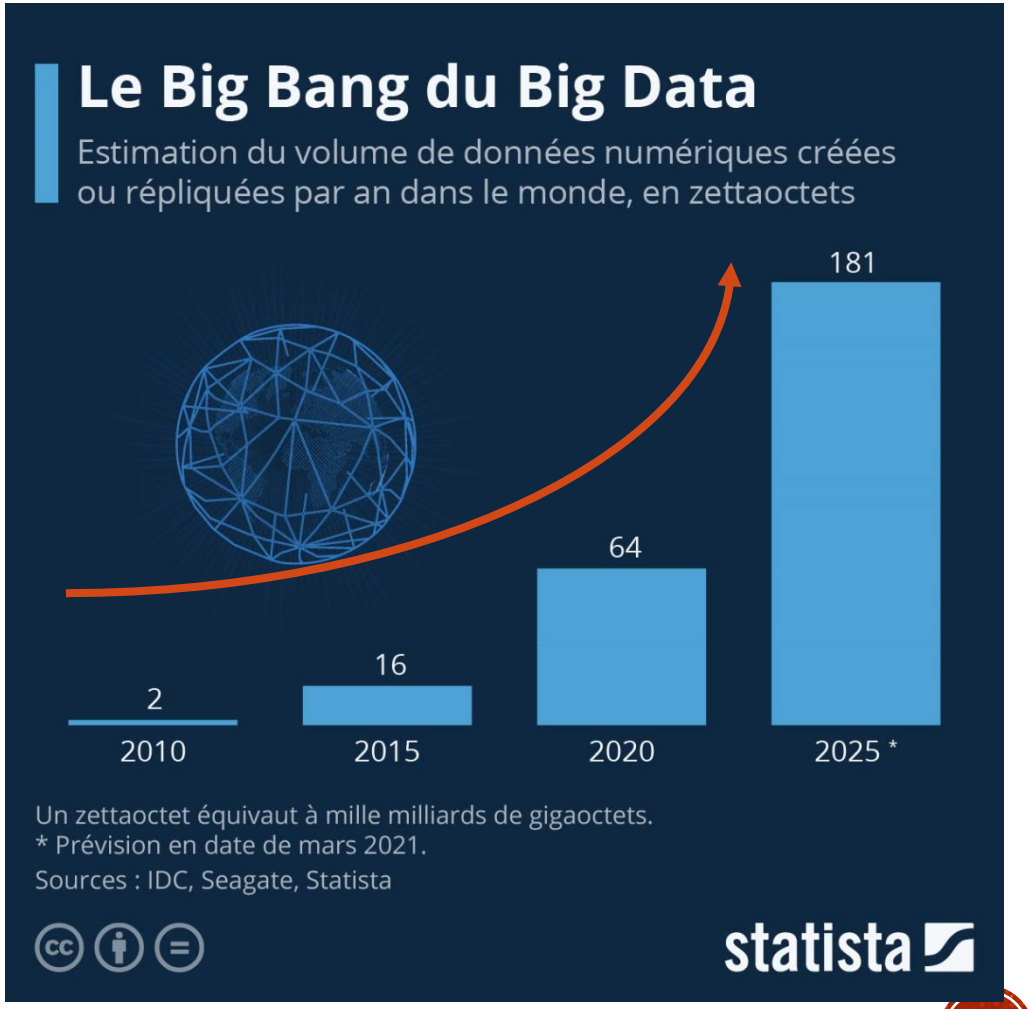
- Simples (SQL+ modèle relationnel)
- Propriétés ACID
- Rapide grâce aux index
- Données structurées
- Limites à partir de quelques téraoctet
- Incapables de gérer des débits extrêmes
- Accès aux disques + ACID entrainent un surcoûts en latence

1990-2010: Entrepôts de donnée

- Requêtes SQL
- analyse décisionnelle
- Processus ETL
- stockage et à l'intégration des données
- Données structurées
- Latence et complexité du processus ETL
- Incapable de gérer des débits extrêmes

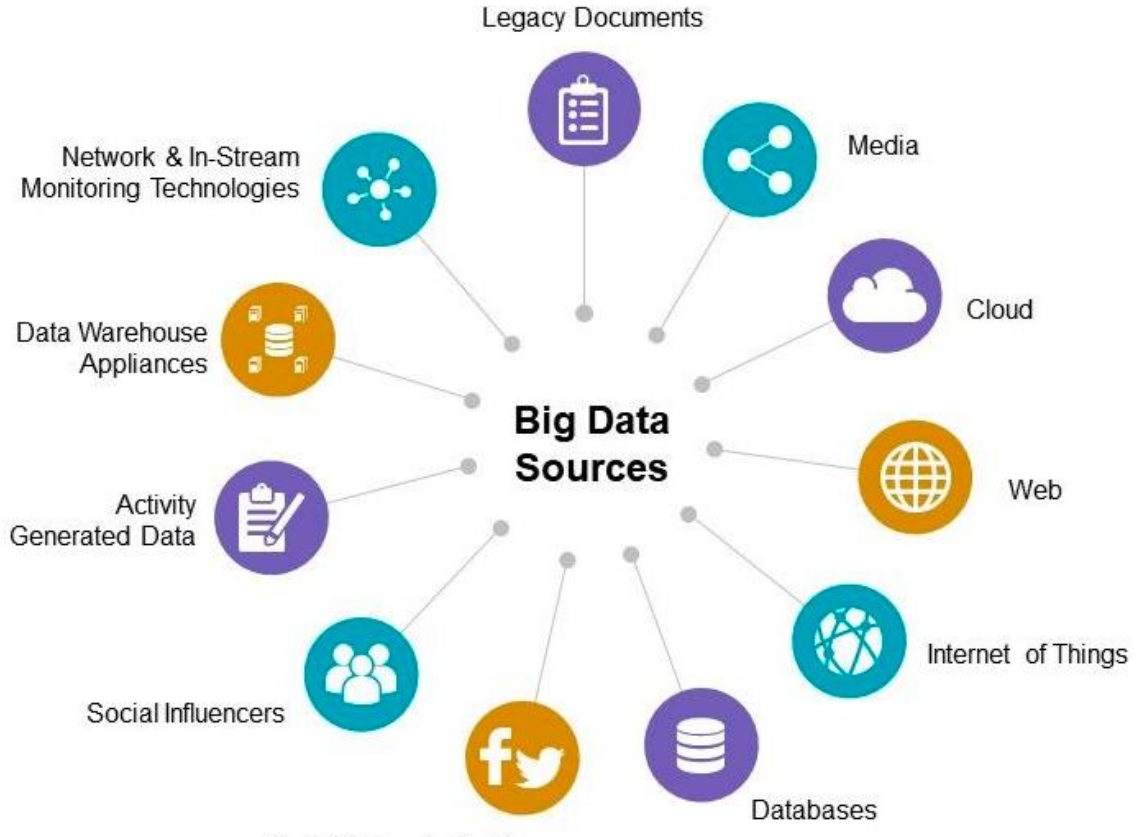
BIG DATA: CONTEXTE D'ÉMERGENCE

- Évolution exponentielle du volume de données
- 2010 : dominance des données structurées
- 2025: dominance des données non structurées
- le pourcentage de données capable d'être traitées diminue. (Moins de 30% en 2021)



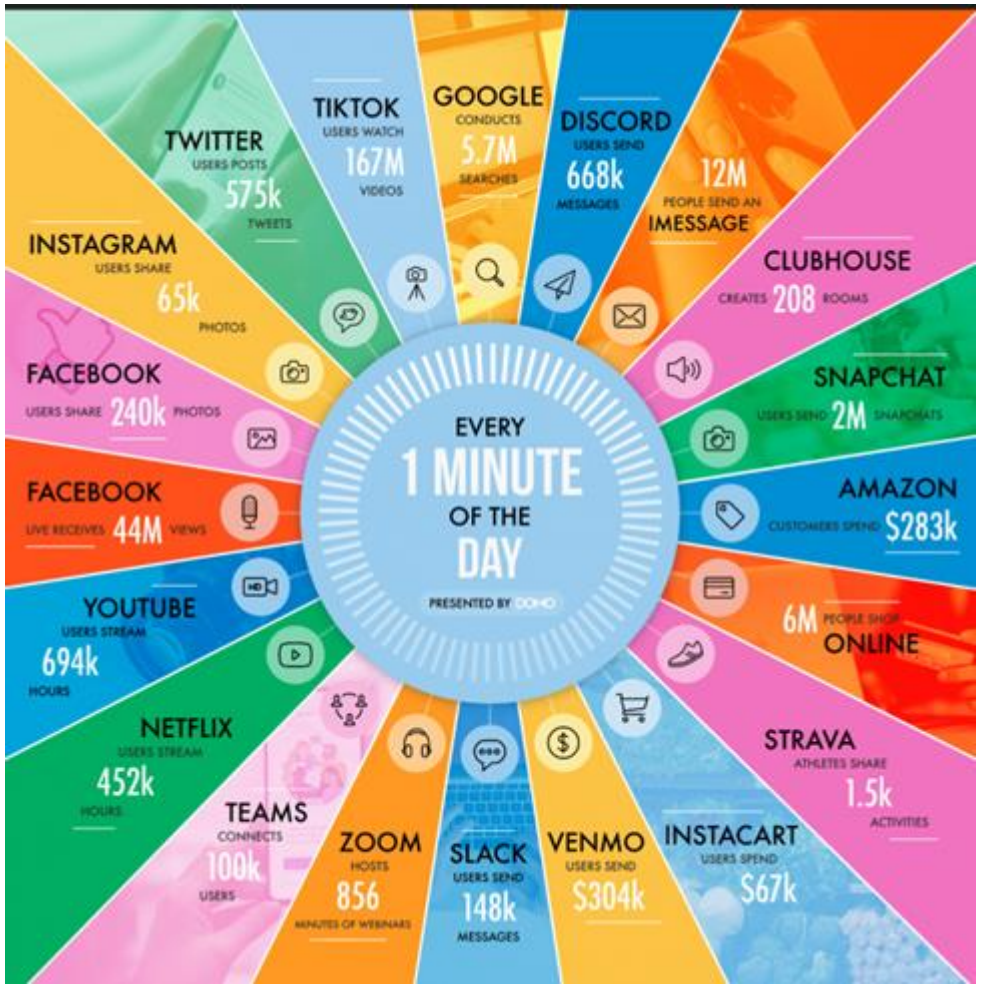
BIG DATA: CONTEXTE D'ÉMERGENCE

- Sources de Big Data



BIG DATA: CONTEXTE D'ÉMERGENCE

Quelle est la taille des données générées sur internet chaque minute?



https://www.domo.com/solution/data-never-sleeps-9 - (2021)

BIG DATA: DÉFINITION

- 1990 John Mashey à Silicon Graphics était le premier à avoir utilisé le terme.
- En 2001 Doug Laney (Meta Group Inc.) a décrit les Big data en utilisant les 3Vs
  - Volume
  - Vitesse
  - Variété
- Big Data est un concept abstrait.
- Plusieurs définitions proposées, aucune définition officielle n'a été adoptée jusqu'à présent

BIG DATA: DÉFINITION

- Orientée traitement:
  - désignent les ensembles de données qu'on est incapable de percevoir, acquérir, stocker, gérer et traiter par les outils informatiques et logiciels / matériels traditionnels dans un délai tolérable

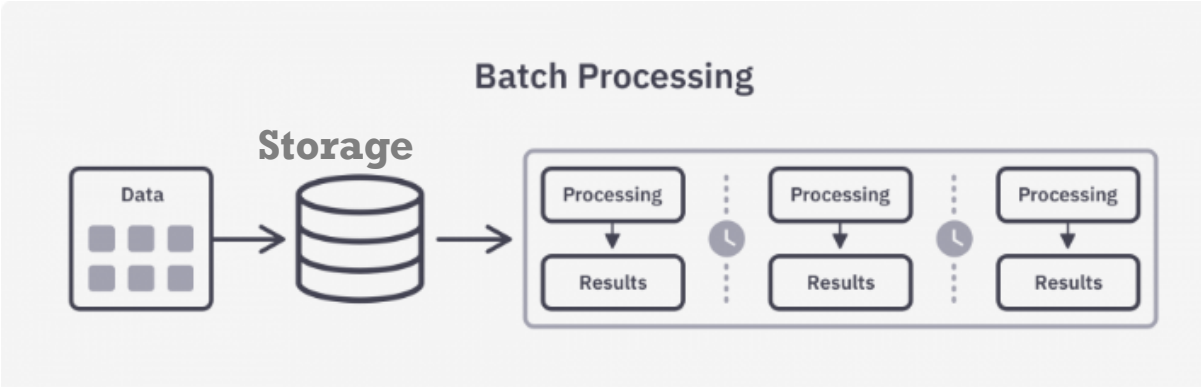
- Orientée caractéristiques:
  - En 2001 Doug Laney a décrit les Big data en utilisant les 3Vs [2] (Volume, Vitesse, Variété)
  - IBM introduit en 2012 le 4^{ième} V : Véracity

CARACTÉRISTIQUES DES BIG DATA

- 3Vs? 4Vs? 11Vs? 17Vs Ou 42Vs?
  - **Value:** est ce que les données sont pertinentes pour le domaine ou le traitement?
  - **Variability:** l'incohérence dans les données et l'incohérences dans la vitesse des données
  - **Validity:** la précision et l'exactitude des données pour l'usage auquel elles sont destinées. Sont elles prêtes à l'emploi?
  - **Vulnerability:** problèmes de sécurité
  - **Volatility:** Quel âge doivent avoir vos données avant qu'elles ne soient considérées comme non pertinentes
  - **Visualization:** à quel point il est difficile de visualiser
  - **…..**

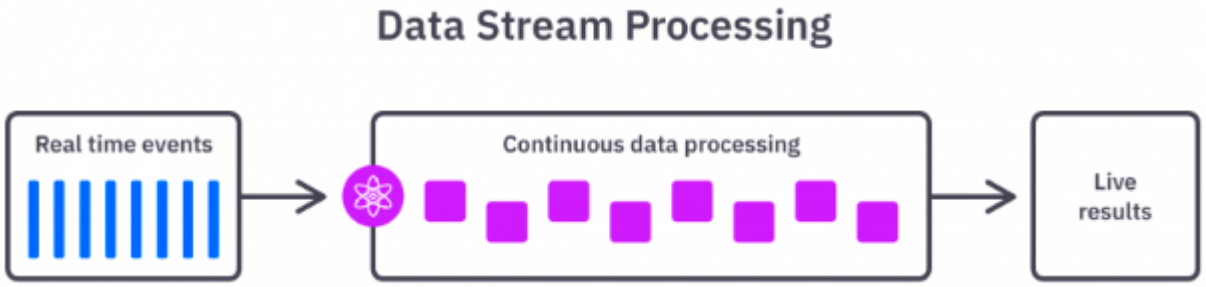
TRAITEMENT DES BIG DATA

- Modes de traitement des données
  - Batch processing: Traitement en lot des données.
  - objectif
    - Apprendre du passé
      - Prédire la vitesse de trafic à un instant t
    - Prendre des décisions globales
      - Élargir la route
  - Avantage
    - Traitement de gros volumes de données
  - Inconvénient
    - Latence



TRAITEMENT DES BIG DATA

- Mode de traitement des données
  - Stream processing: analyse en continu des données
  - Objectif
    - Surveiller et suivre les événements
      - Identifier une congestion en temps réel
    - Prendre des décisions rapides
      - Ré-routage des véhicules
  - Avantage
    - Rapidité de réponse
  - Inconvénient
    - Faible quantité de données



TRAITEMENT DES BIG DATA

- Architecture Lambda: Fait coexister deux modes de traitement
  - Couche batch:
    - Traite une masse importante de données **brutes immutables horodatées**
    - Génère et stocke périodiquement des batch views
    - Fournit des vues avec accès séquentiel

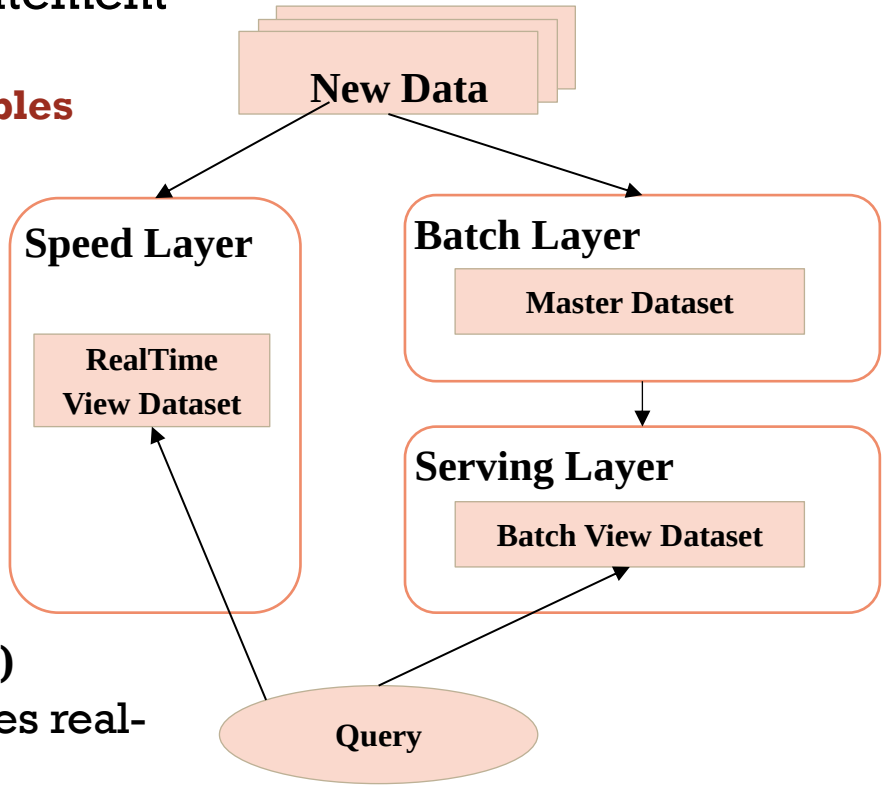
batch view = function(all data)

  - Couche de service
    - Accès direct sur les vues batch
  - Couche vitesse
    - Analyse les données reçues pendant l'exécution du batch
    - Fournit des real-time views

realtime view = function(realtime view, new data)

  - La requête utilisateur est exécutée sur les batch views et les real-time views

query = function(batch view. realtime view)



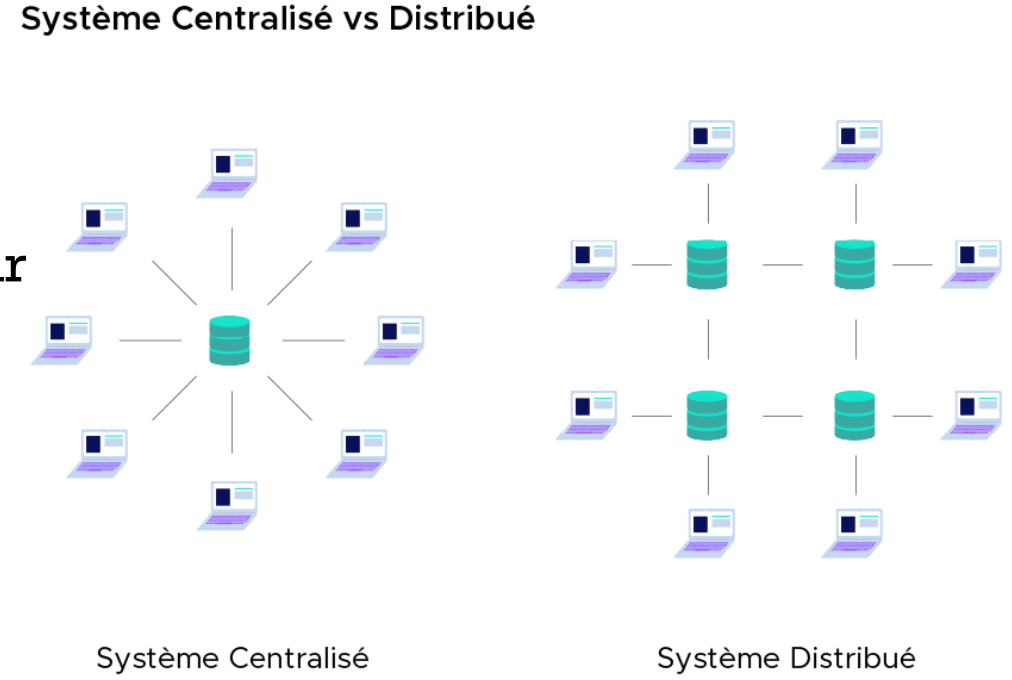
FRAMEWORKS DE TRAITEMENT DES BIG DATA

- Hadoop :
  - batch processing
- Spark :
  - batch processing
  - Near-real time processing
  - Implementation de l'architecture lambda

HADOOP

TRAITEMENT DISTRIBUÉ

- Plusieurs ordinateurs
  - fonctionnent ensemble pour atteindre un objectif commun
  - communiquent entre eux par la transmission de messages
  - apparaissent comme un seul super ordinateur
- **Problèmes**
  - Hétérogénéité
  - Sécurité
  - Scalabilité
  - Concurrence
  - Tolérance aux pannes



FRAMEWORKS DE TRAITEMENT DES BIG DATA

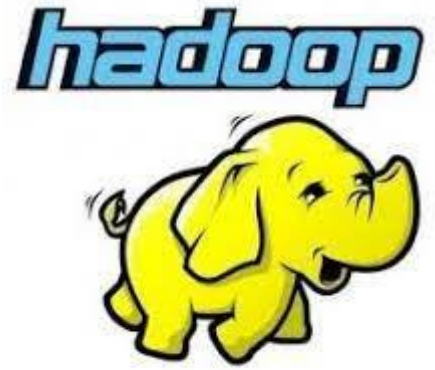
- Hadoop est un framework open-source, utilisé pour le stockage et le traitement des données massives dans un environnement distribué.
- Un framework développé en Java par Yahoo! (2007) sur la base de papiers publiés par Google (GFS, MapReduce).
- Hadoop cache les détails et la complexité du système à l'utilisateur
- Conçu pour un matériel simple/de base hétérogène

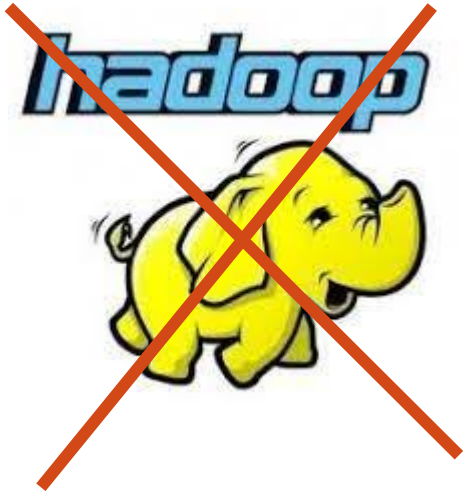
HADOOP

- Objectifs
  - Le système se charge de la plupart des problèmes automatiquement
    - Reprise après panne
    - Réduire la communication
    - …
  - Permet aux applications de fonctionner avec des milliers de nœuds et des pétaoctets de données de manière parallèle et rentable
    - nœud= unité de calcul (CPU) + unité de stockage (disques)
    - Les nœuds peuvent être combinés en clusters
  - Évolutivité : de nouveaux nœuds peuvent être ajoutés au besoin sans changer
    - Format de données
    - Comment les données sont chargées
    - Comment les job sont rédigés
  - Tolérance aux pannes: fournie par la réplication

HADOOP

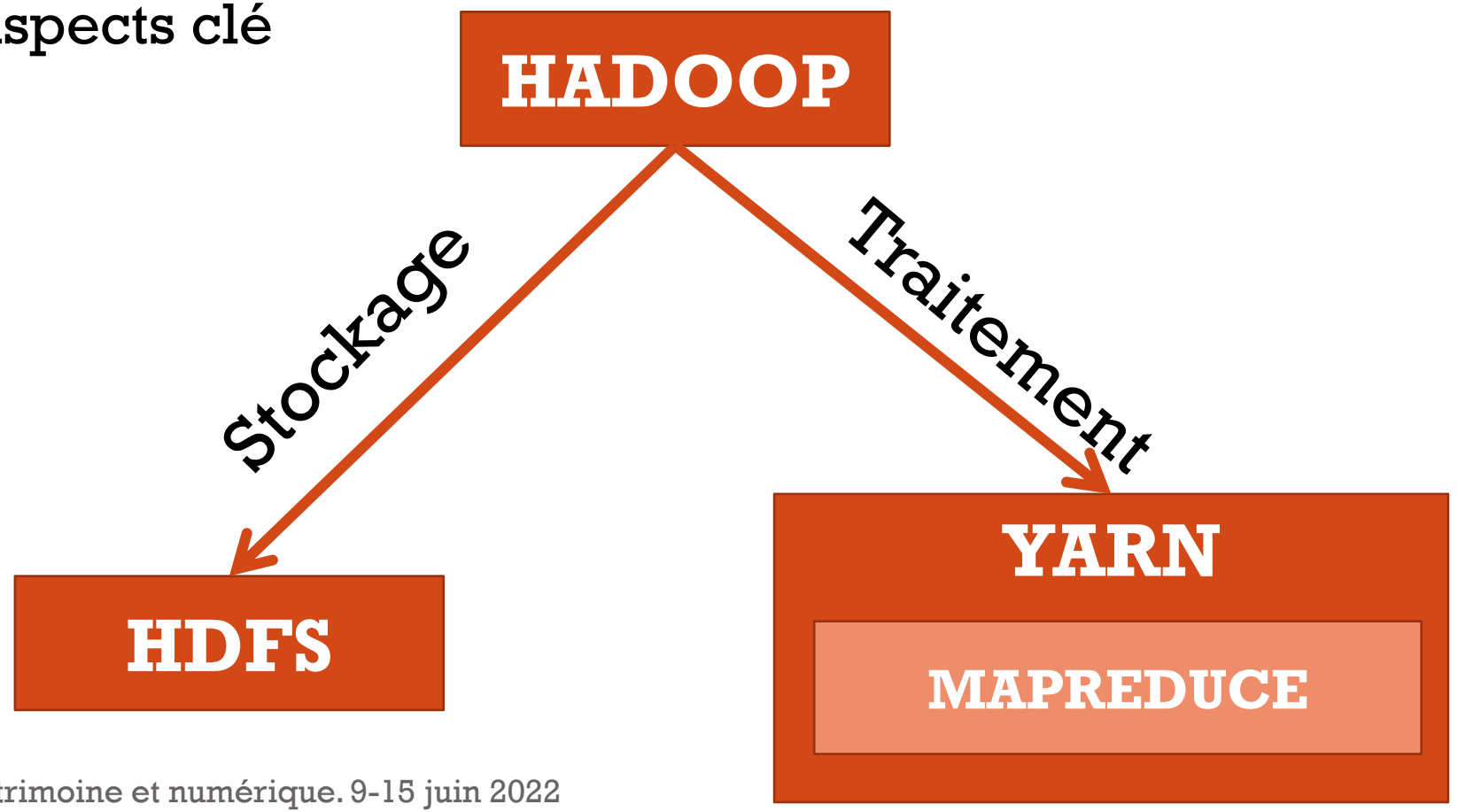
- Optimal pour:
  - Gérer des quantités massives de données grâce au parallélisme
  - Stocker et analyser une variété de données (structurées, non structurées, semi-structurées)
  - Utilisation de matériel de base bon marché
- Non optimal pour:
  - traiter les transactions (accès aléatoire)
  - lorsque le travail ne peut pas être parallélisé (boucles, analyses des graphes)
  - l'accès aux données à faible latence (accès rapide)
  - traiter beaucoup de petits fichiers
  - les calculs intensifs avec peu de données





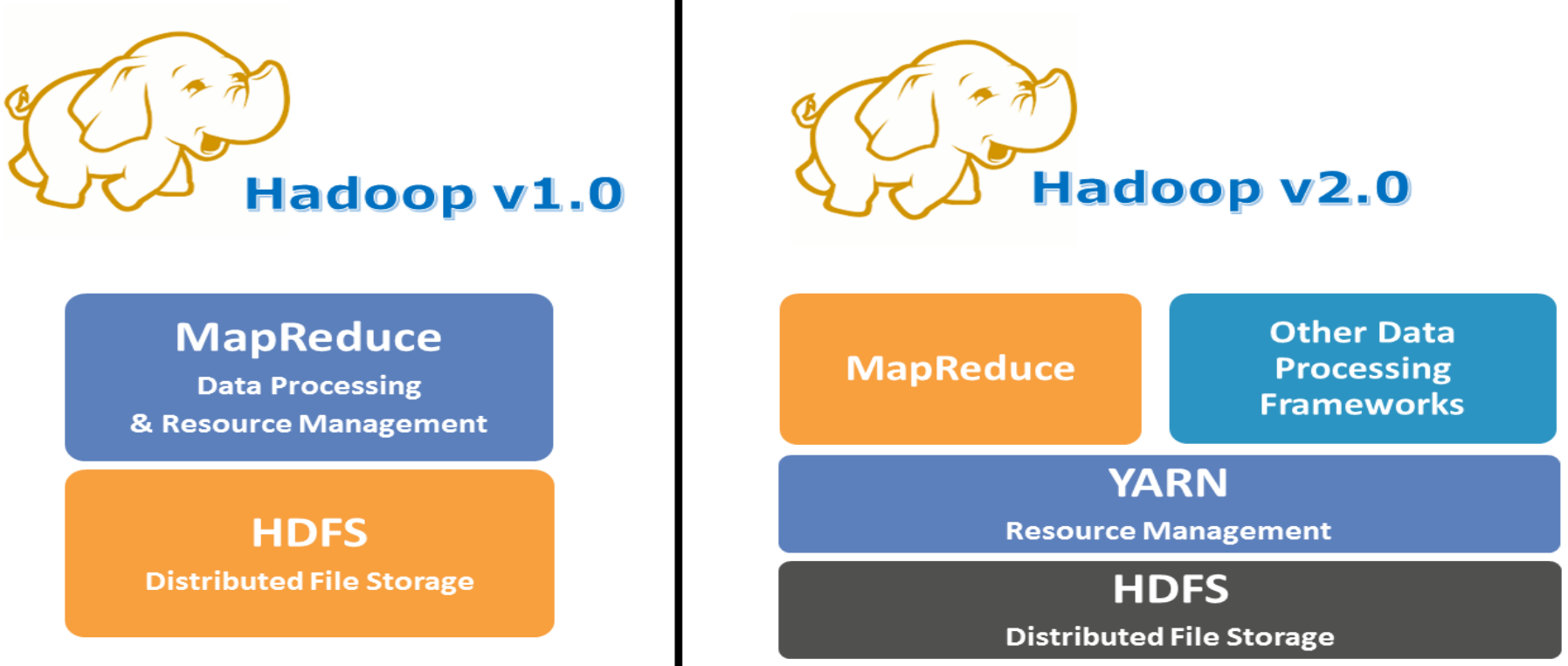
HADOOP

- Deux aspects clé



À partir de Hadoop 2.0

HADOOP



https://www.edureka.co/blog/hadoop-yarn-tutorial/

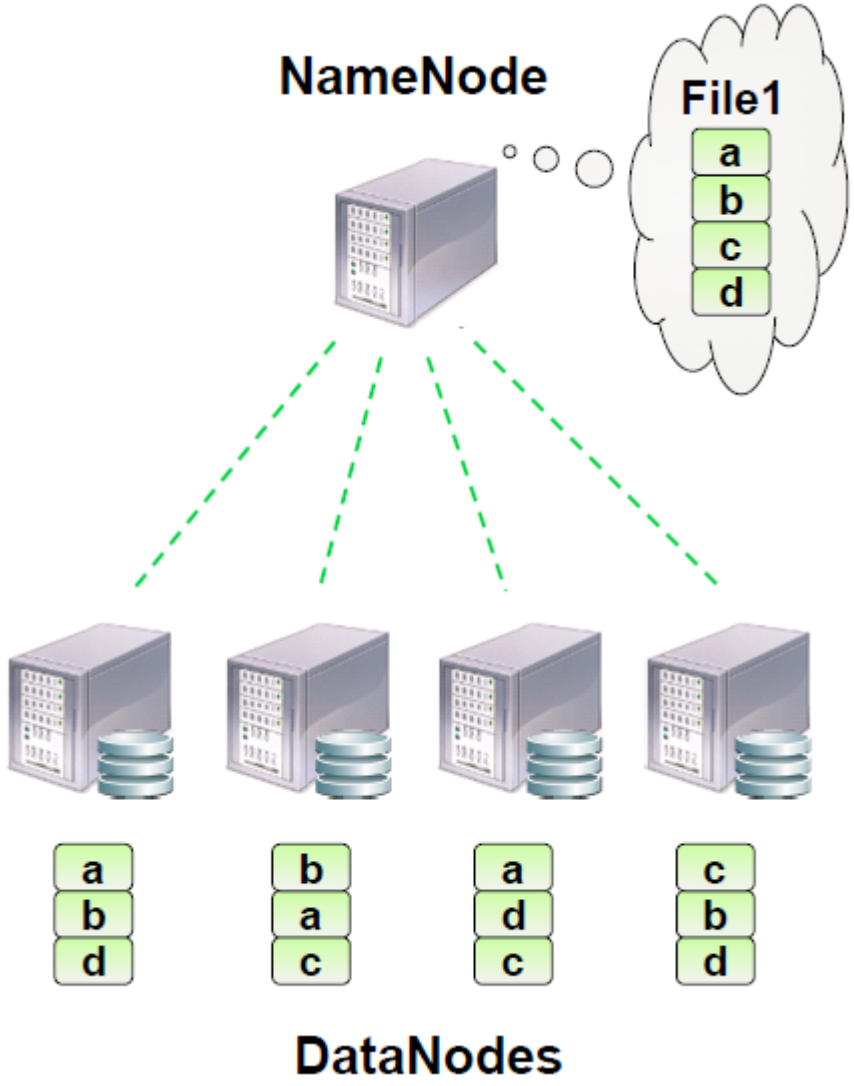
HDFS

HADOOP-HDFS

- Distribué, évolutif, tolérant aux pannes, haut débit
- Accès aux données via MapReduce
- Les fichiers sont divisés en blocs
  - Par défaut 3 réplicas pour chaque bloc
  - Peut créer, supprimer, copier, mais PAS mettre à jour
- Conçu pour la lecture séquentielle, non pour l'accès aléatoire
- Data locality: traitement des données sur ou à proximité du stockage physique pour diminuer la transmission de données

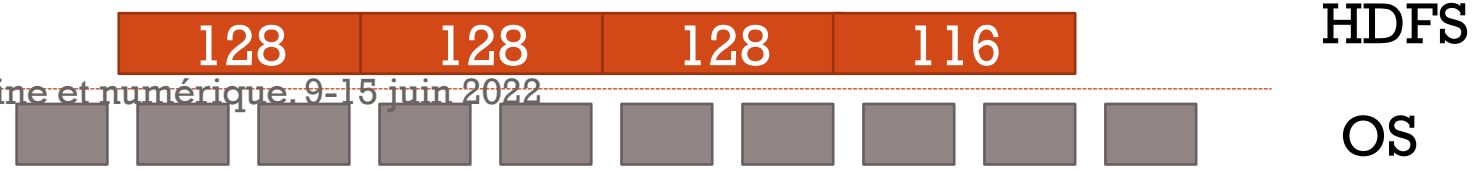
HADOOP-HDFS

- **Architecture maître / esclave**
- Maître: NameNode
  - gère l'espace de noms et les métadonnées du système de fichiers
    - FsImage
    - EditLog
  - régule l'accès des clients aux fichiers
  - Maintient et gère les datanodes
  - Reçoit les heartbeats et les informations sur les blocs à partir des Datanodes
- Esclave: DataNode
  - plusieurs par cluster
  - gère le stockage attaché aux nœuds
  - rapporte périodiquement l'état à NameNode



HADOOP-HDFS

- HDFS est conçu pour prendre en charge de très gros fichiers
- Chaque fichier est divisé en blocs de 128MB
- Les blocs résident sur différents DataNode physiques
- Dans les coulisses, 1 bloc HDFS est pris en charge par plusieurs blocs du système d'exploitation
- Si un fichier ou une partie du fichier sont plus petits que la taille du bloc, seul l'espace nécessaire est utilisé.
- Exemple: un fichier de 500 Mo est divisé en 3 blocs de 128 et 1 bloc de 116



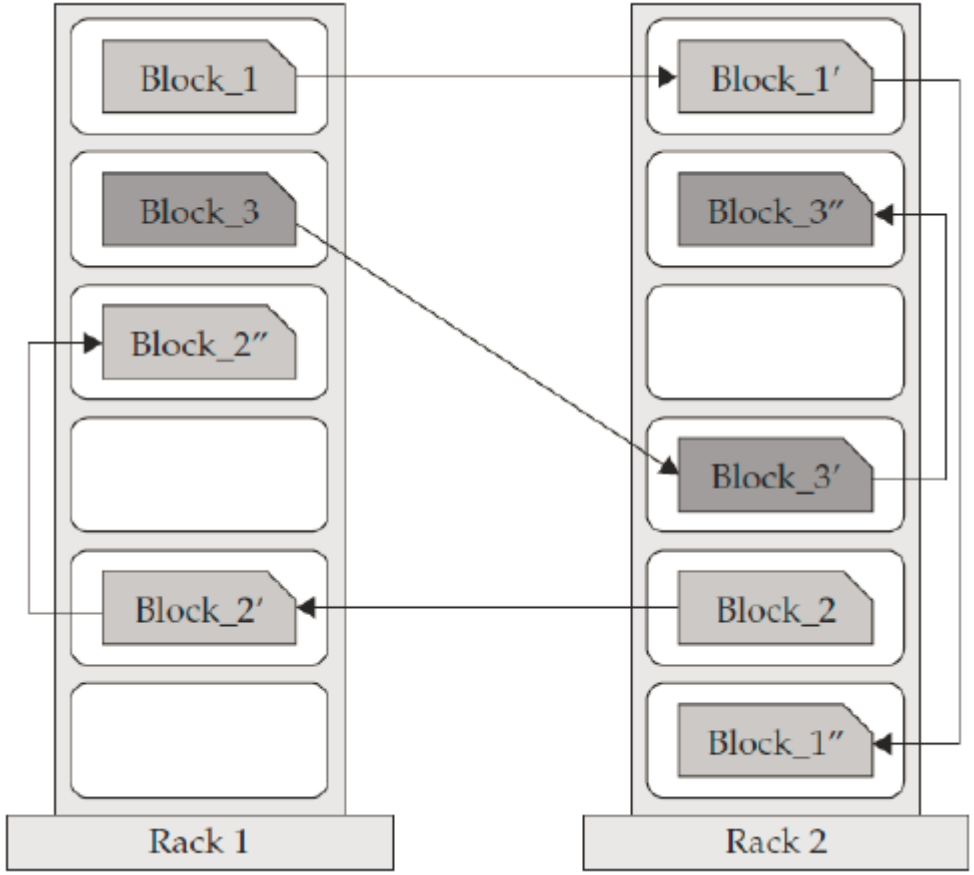
HADOOP-HDFS

- Les blocs de données sont répliqués sur plusieurs nœuds
- Le facteur de réplication est configurable dans le fichier (**hdfs-site.xml**)
- La valeur par défaut est 3 réplicas

```
<property>
 <name>dfs.replication</name>
 <value>3</value>
 </property>
```

- Rack Awareness: le même bloc ne peut pas être présent sur le même rack.
- Rack Awareness est configurable sur **core-site.xml**

```
<property>
<name>topology.script.file.name</name>
<value>/opt/ibm/biginsights/hadoop-conf/rack-aware.sh</value>
</property>
```



HADOOP-HDFS

- Pour la gestion des métadonnées, le NameNode utilise deux fichiers :

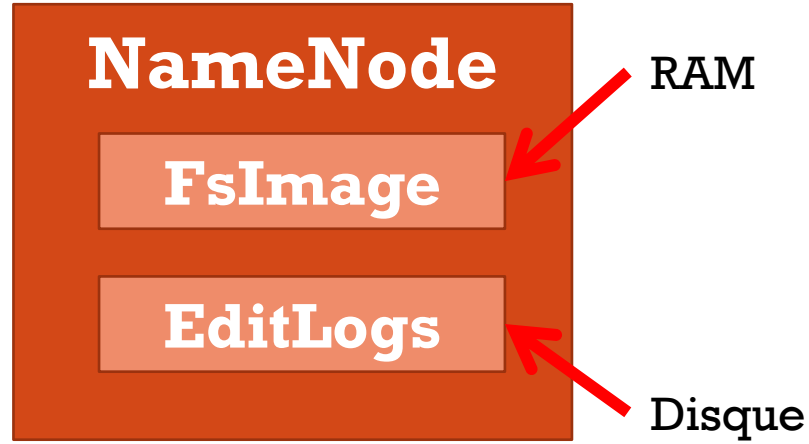
- **FsImage** : stocké sur la RAM, il contient toutes le données sur l'état du système depuis le démarrage du NameNode. (volumineux)

rwxr-xr-- 1 user user 174 3/03/2020 **file1.doc**…

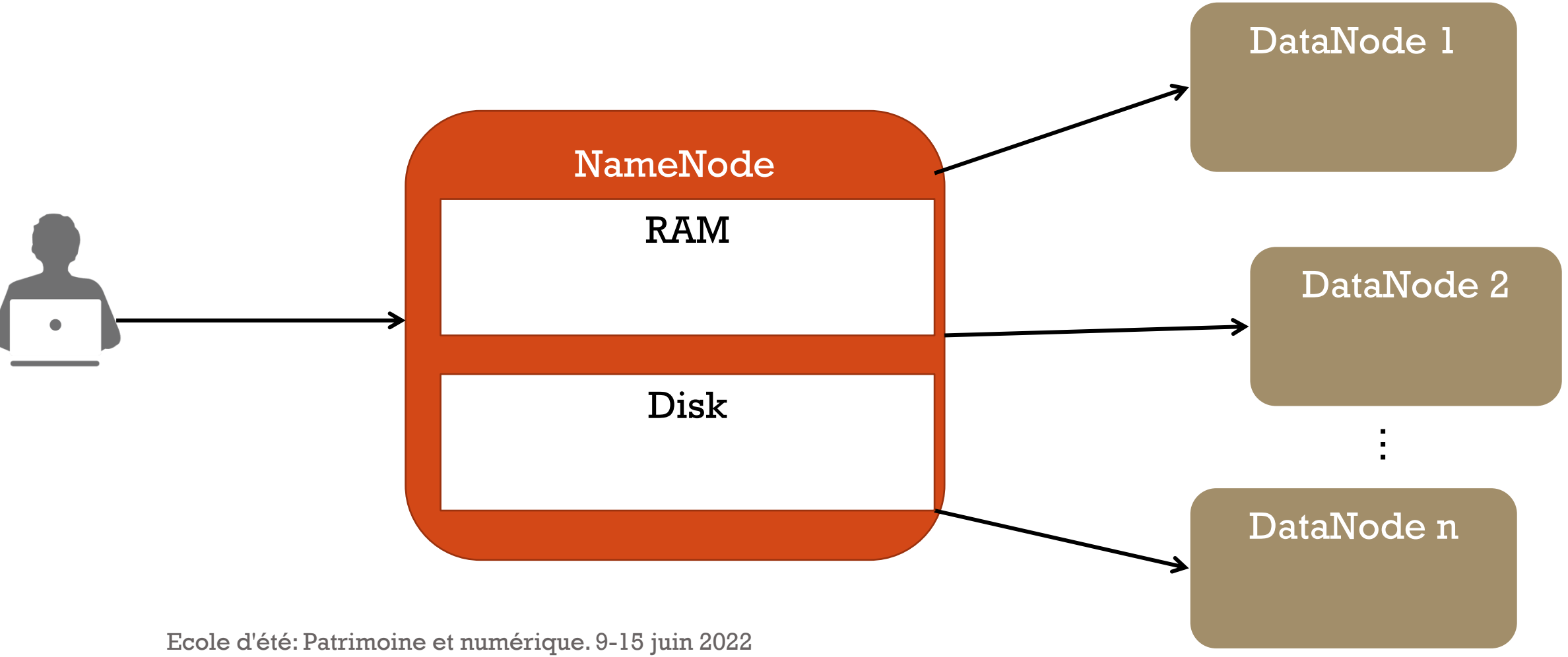
- **EditLogs** : stocké dans le disque, il contient les modifications survenues dans le système.

put file1.doc

Chmod 777 file1.doc

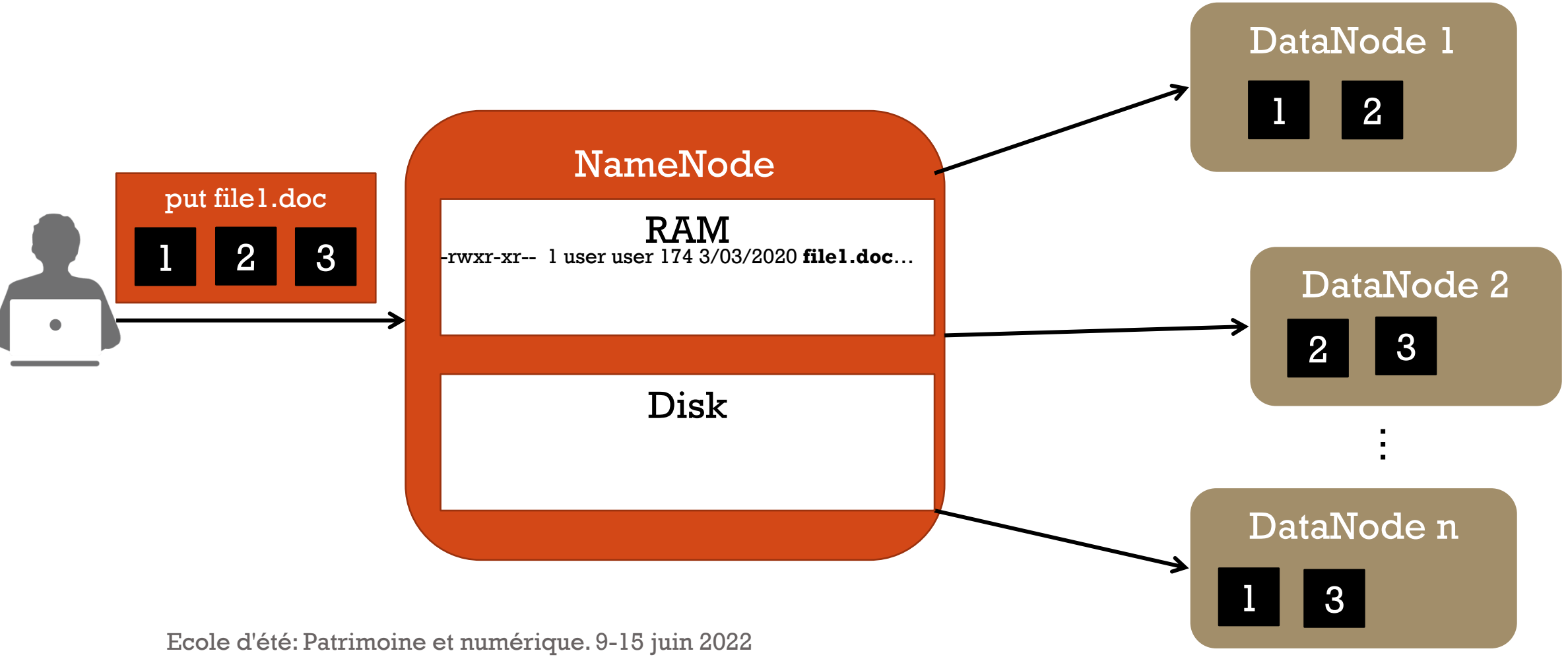


HDFS-1



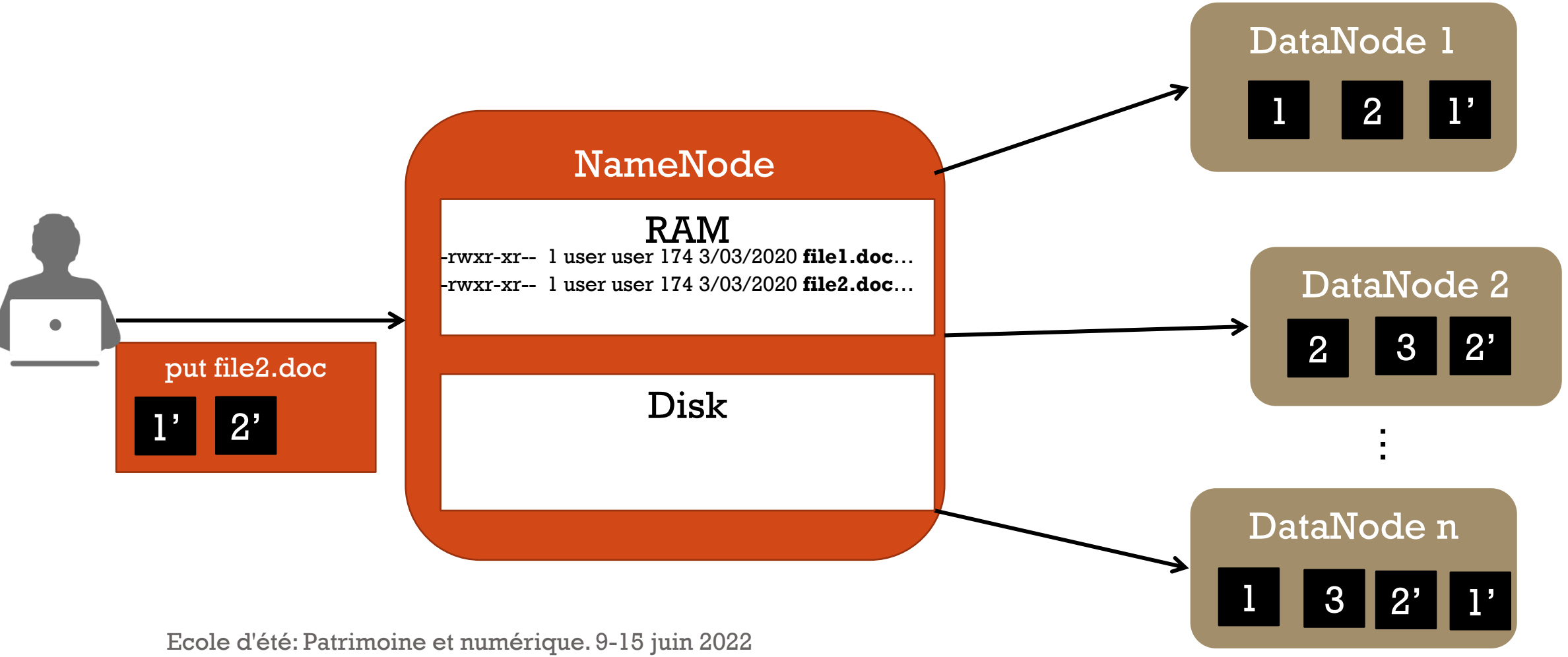
HDFS-1

Facteur de réplication = 2

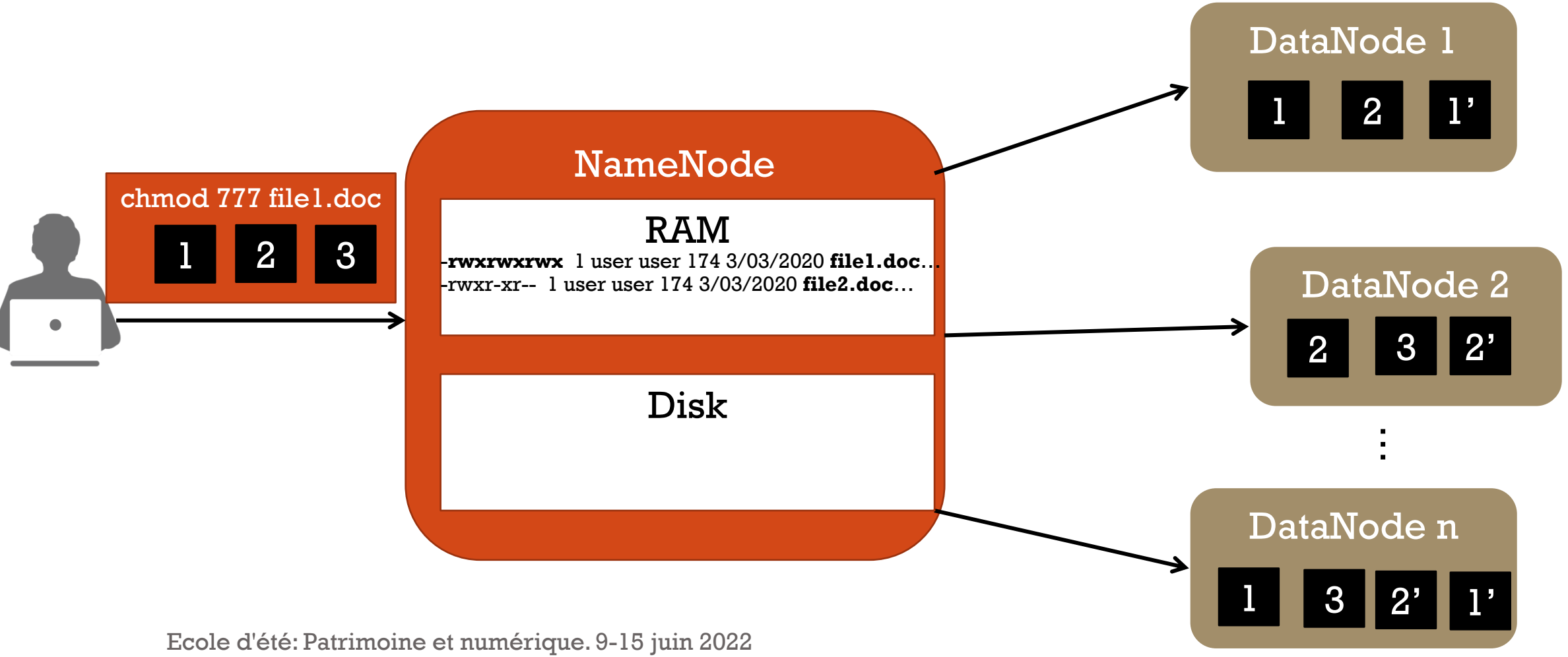


HDFS-1

Facteur de réplication = 2



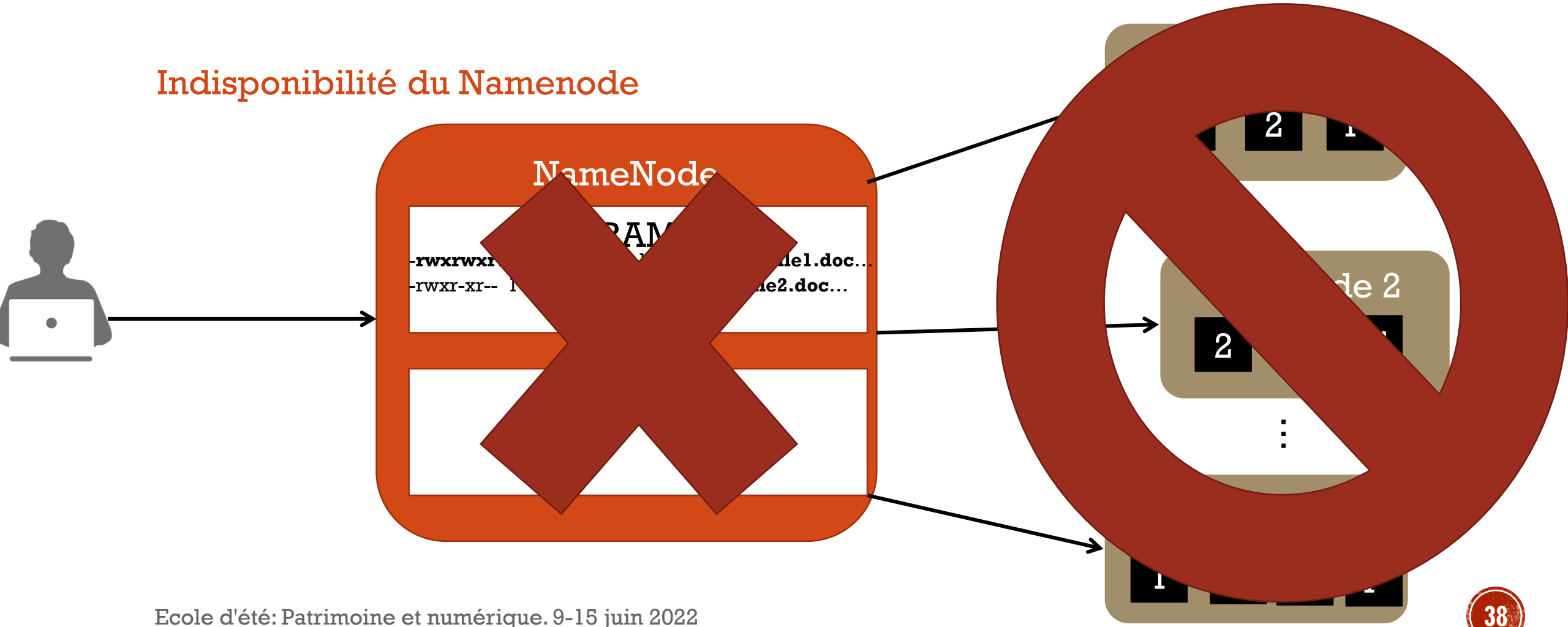
HDFS-1


chmod 777 file1.doc
1
2
3
NameNode
RAM
-rwxrwxrwx 1 user user 174 3/03/2020 file1.doc…
-rwxr-xr-- 1 user user 174 3/03/2020 file2.doc…
Disk
DataNode 1
1
2
1'
DataNode 2
2
3
2'
DataNode n
1
3
2'
1'

HDFS-1

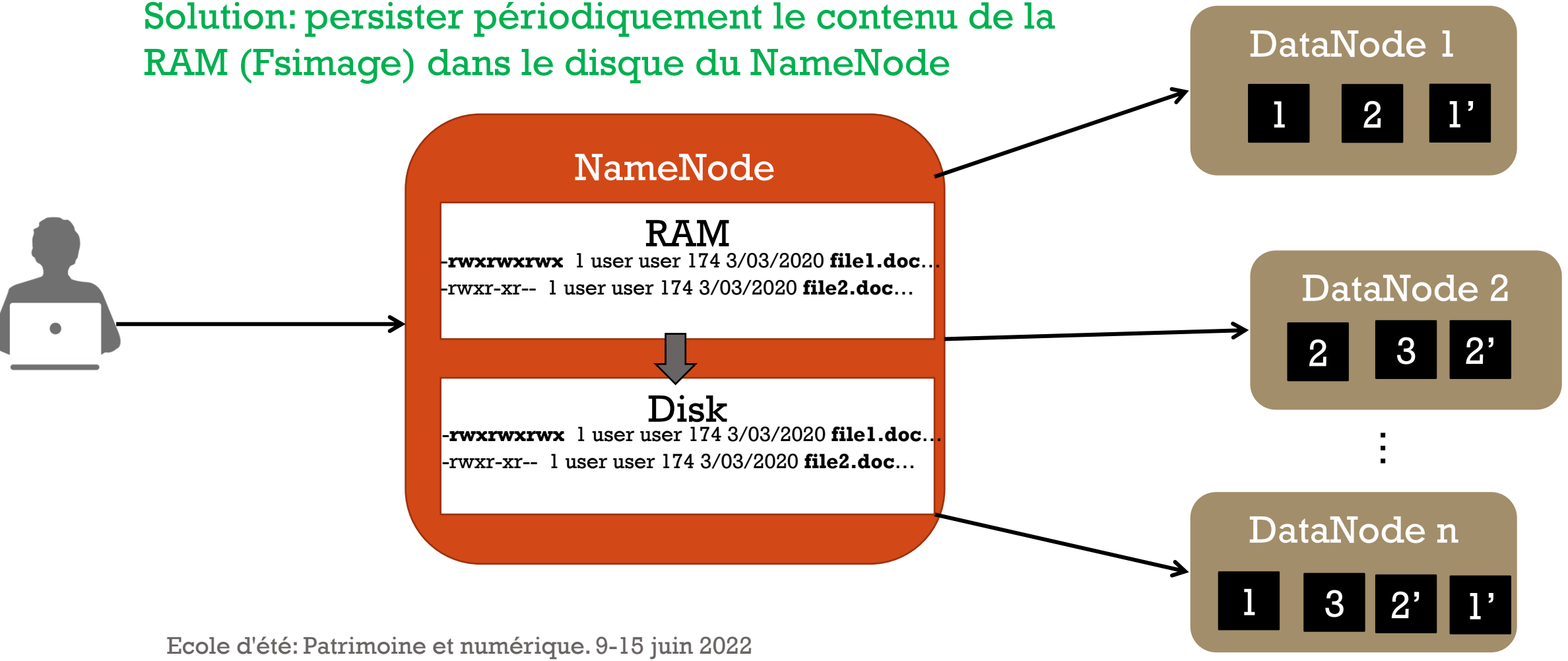
Perte de toutes les informations sur les Datanodes et blocs

Indisponibilité du Namenode

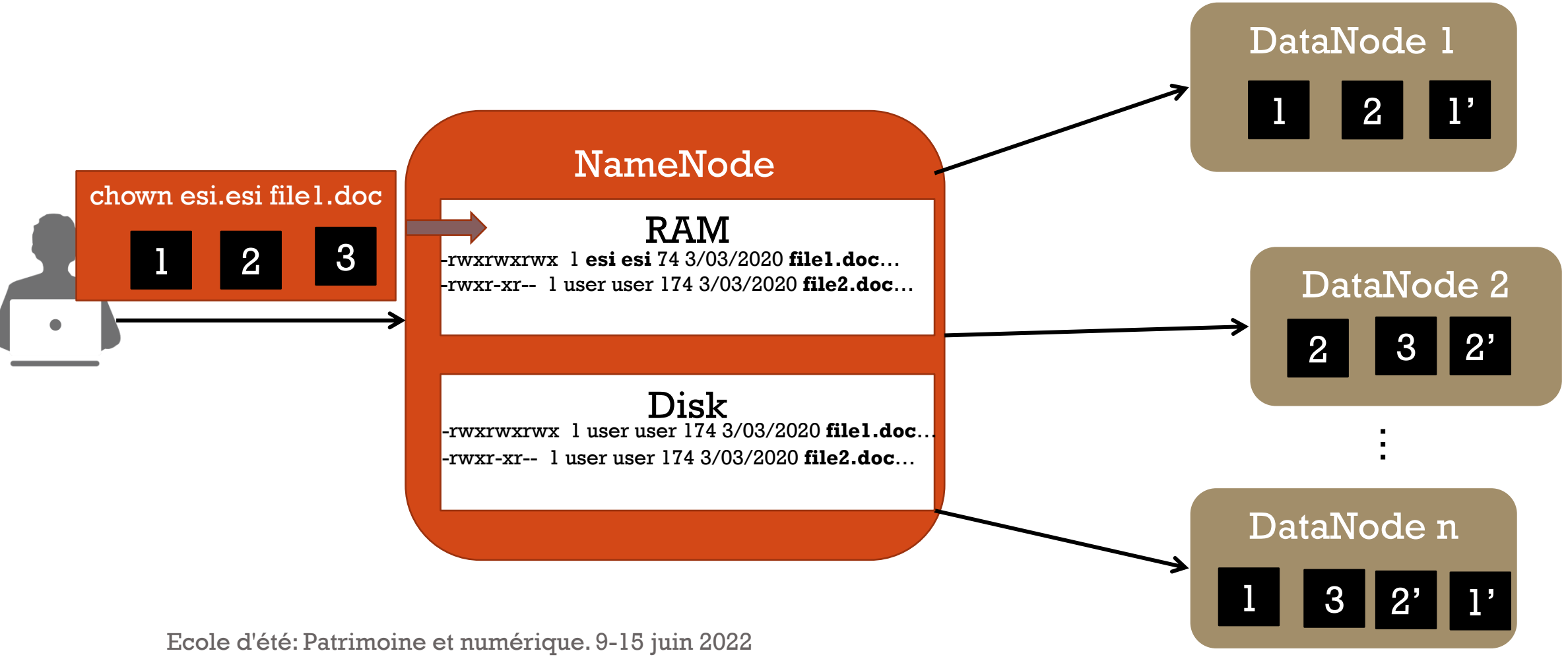


HDFS-1

Solution: persister périodiquement le contenu de la RAM (Fsimage) dans le disque du NameNode

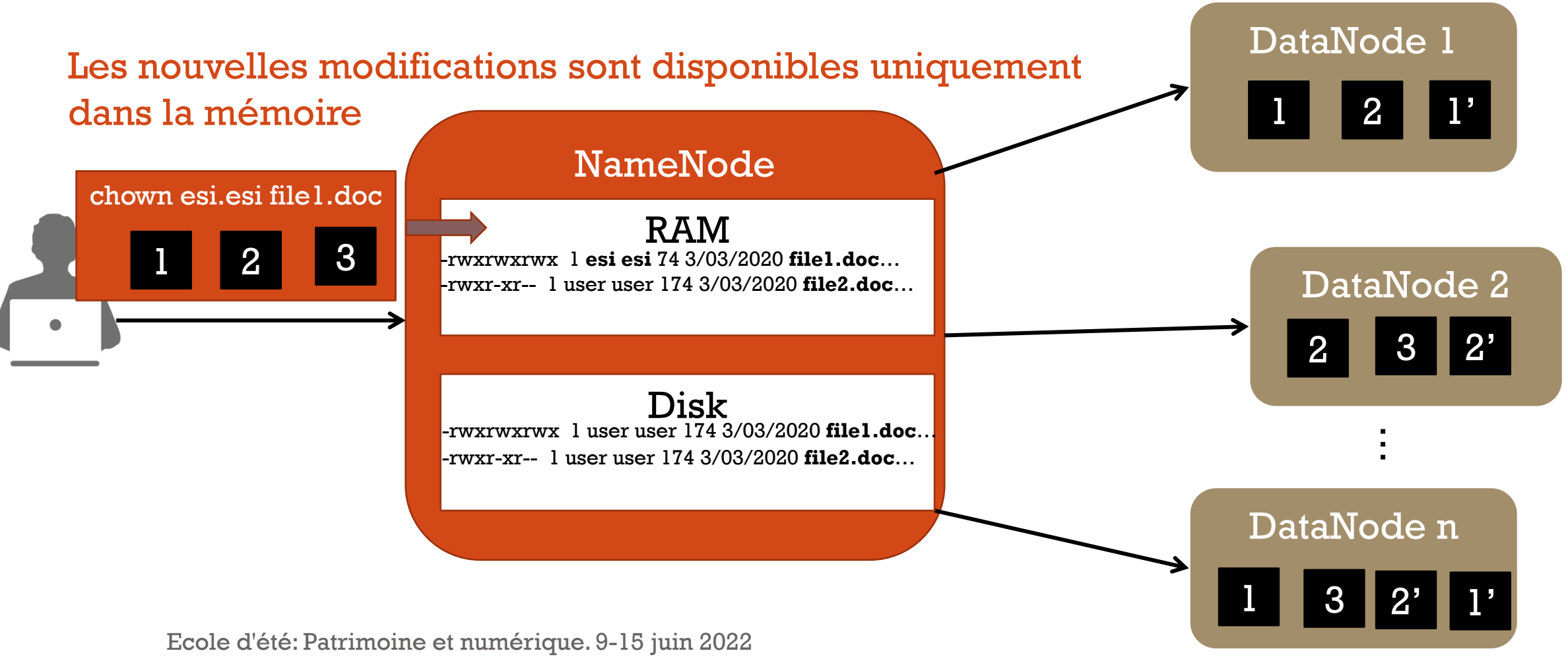


HDFS-1


chown esi.esi file1.doc
1
2
3
NameNode
RAM
-rwxrwxrwx 1 esi esi 74 3/03/2020 file1.doc…
-rwxr-xr-- 1 user user 174 3/03/2020 file2.doc…
Disk
-rwxrwxrwx 1 user user 174 3/03/2020 file1.doc…
-rwxr-xr-- 1 user user 174 3/03/2020 file2.doc…
DataNode 1
1
2
1'
DataNode 2
2
3
2'
DataNode n
1
3
2'
1'

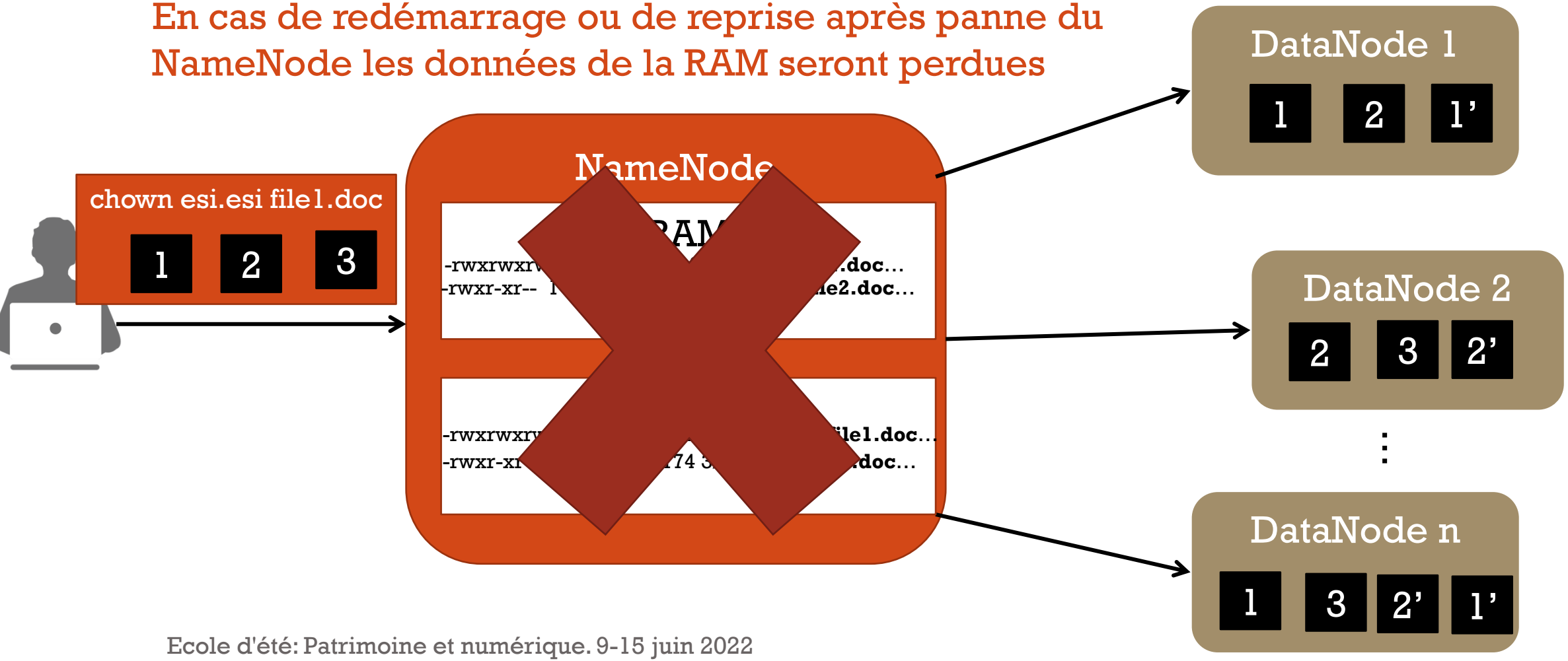
HDFS-1

Les nouvelles modifications sont disponibles uniquement dans la mémoire



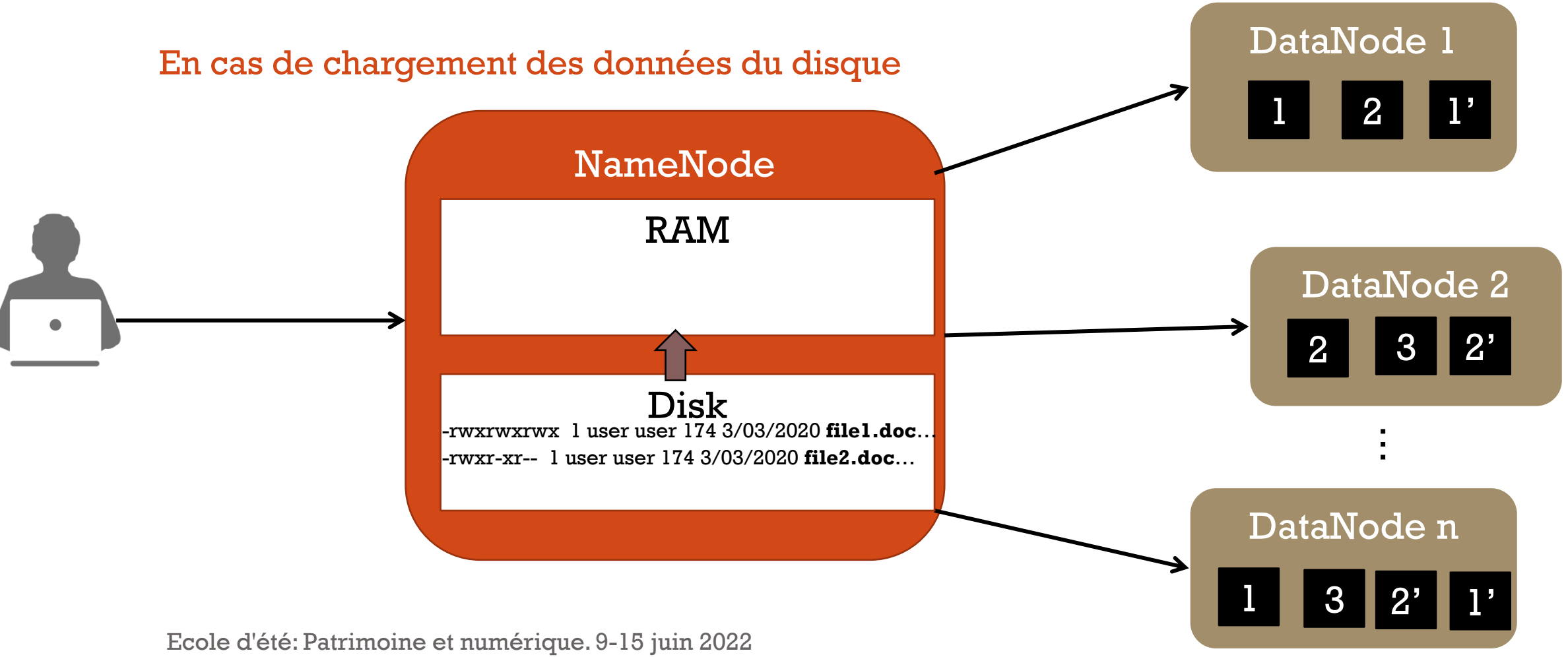
HDFS-1

En cas de redémarrage ou de reprise après panne du NameNode les données de la RAM seront perdues



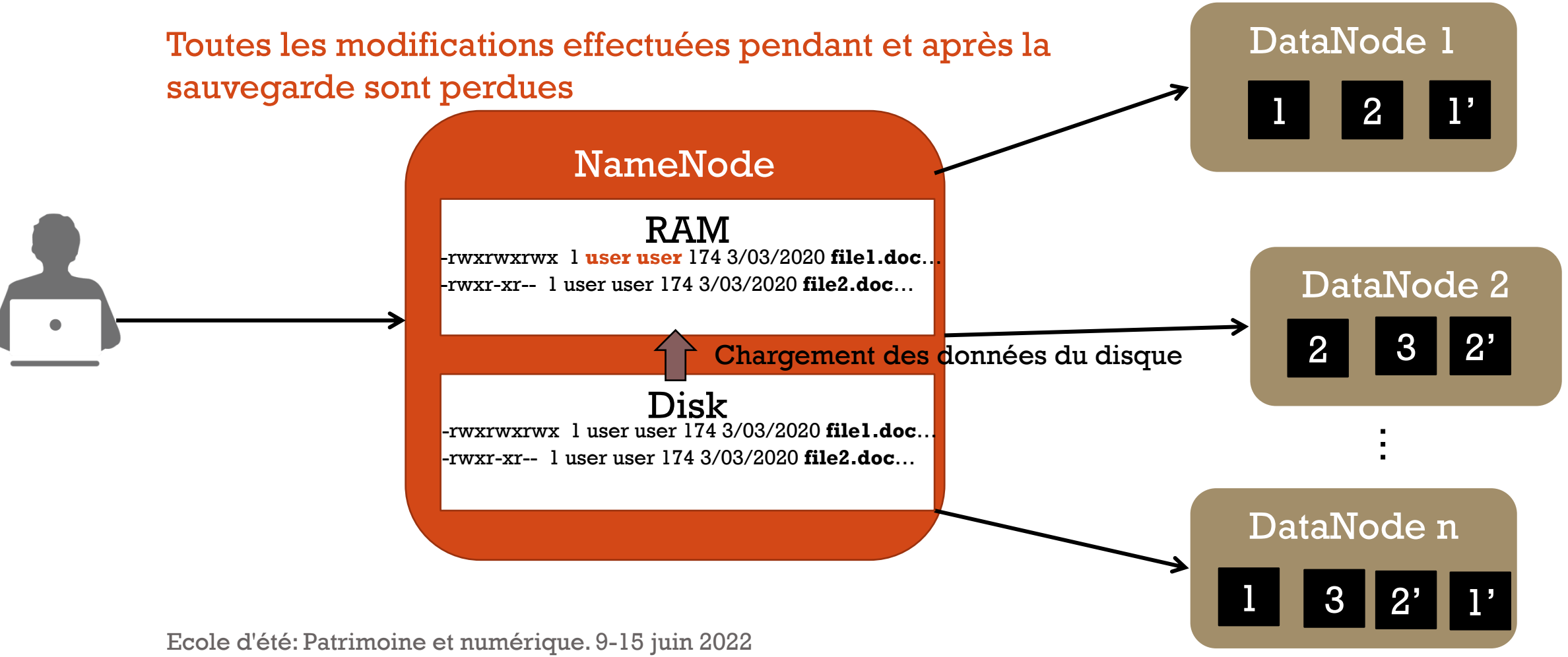
HDFS-1

En cas de chargement des données du disque

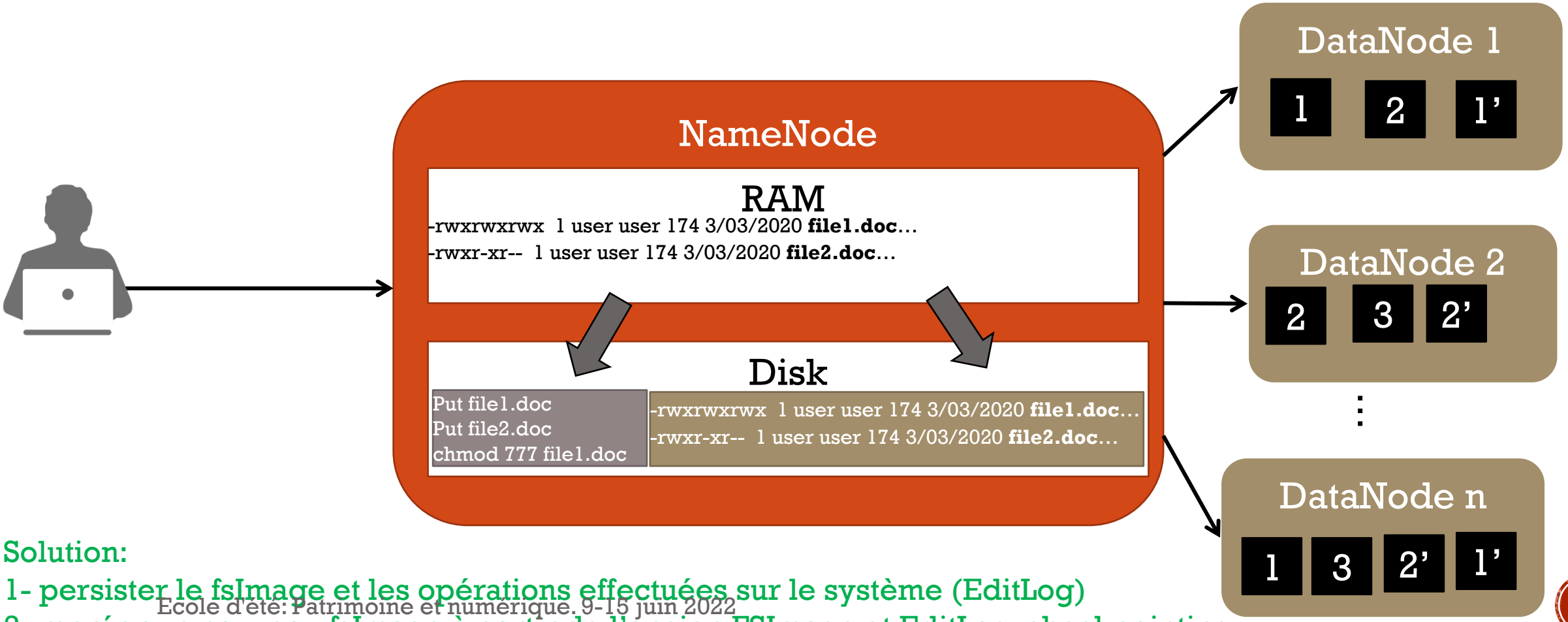


HDFS-1

Toutes les modifications effectuées pendant et après la sauvegarde sont perdues



HDFS-1



Solution:

1- persister le fsImage et les opérations effectuées sur le système (EditLog)

2- recréer un nouveau fsImage à partir de l'ancien FSImage et EditLog: checkpointing

DÉMARRAGE DU NAMENODE (SAFEMODE)

Charger le fichier FsImage stocké dans le disque

Appliquer les modifications du journal d'édition (EditLog)

NewFsImage = EditLogs + OldFsImage

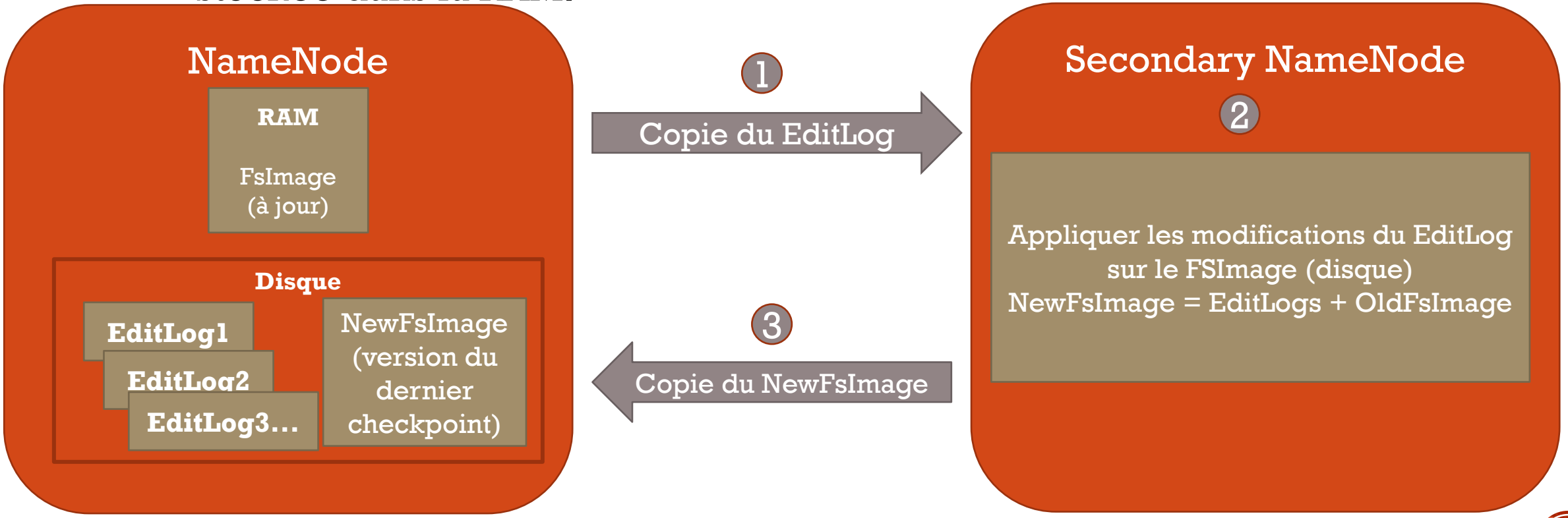
Stocker NewFsImag (le nouveau FsImage) sur le disque

CHECKPOINTING

- Problèmes
  - Le fichier editLog devient très grand
  - Le démarrage en SafeMode est lent
  - Le checkpointing est une opération couteuse en termes de temps
  - Le Namenode est indisponible pendant le checkpointing
- Solution
  - Introduire un noeud responsable de l'opération checkpointing "Secondary NameNode"
  - Secondary NameNode effectue un checkpoint après une durée configurée (1h par défaut)
  - Attention Secondary NameNode n'est pas un noeud de backup!! Il ne gère ni les blocs ni les datanodes!!!
  - Secondary NameNode permet un démarrage plus rapide

HDFS-2

Secondary NameNode permet un démarrage plus rapide car la version du FsImage stockée dans le disque est proche de celle stockée dans la RAM.

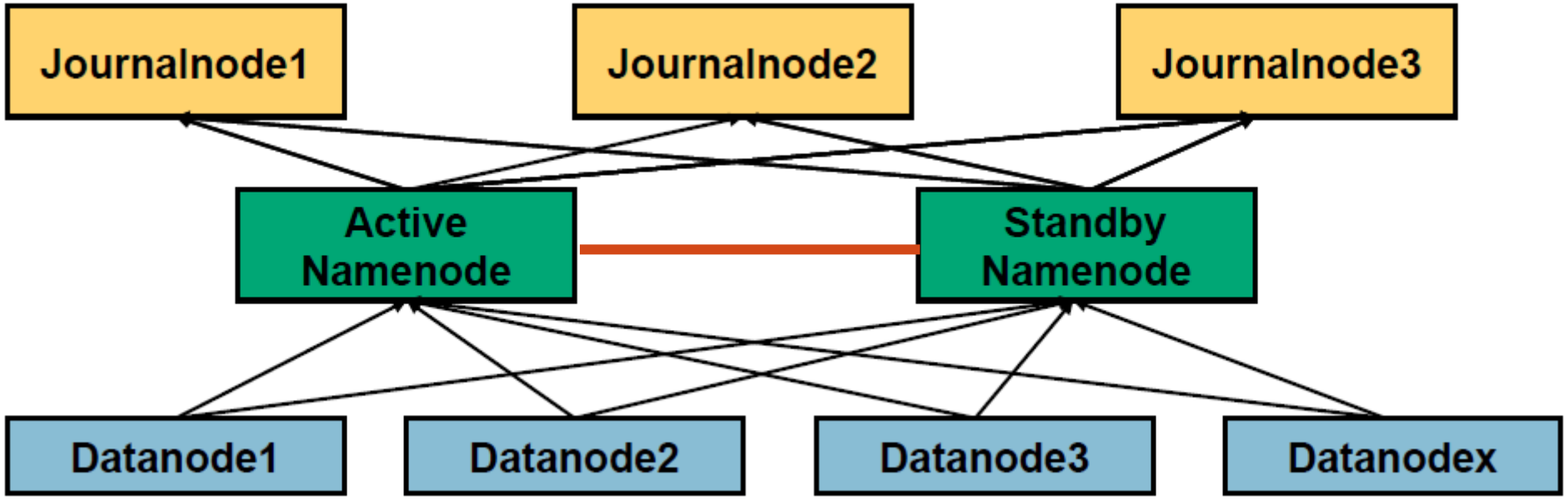


Un nouveau fichier EditLog est utilisé au début de chaque opération checkPoint

HDFS-2

- Le namenode est un SPOF (Point de défaillance unique): si il est indisponible tout le système est indisponible
- HDFS-2 a introduit NameNode HA (high availability)
- Introduction du standbyNode (noeud de backup)
- Le NameNode actif crée plusieurs réplicas du fichier EditLog (3 fois) dans les datanodes.
- Le StandByNode applique les modifications des nœuds de journal au fur et à mesure qu'elles se produisent
- Les DataNodes envoient des informations sur les blocs et les heartbeat aux aux NameNodes et StandByNode
- L'état de la mémoire du StandbyNode est très proche du NameNode actif

HDFS-2 NAMENODE HA



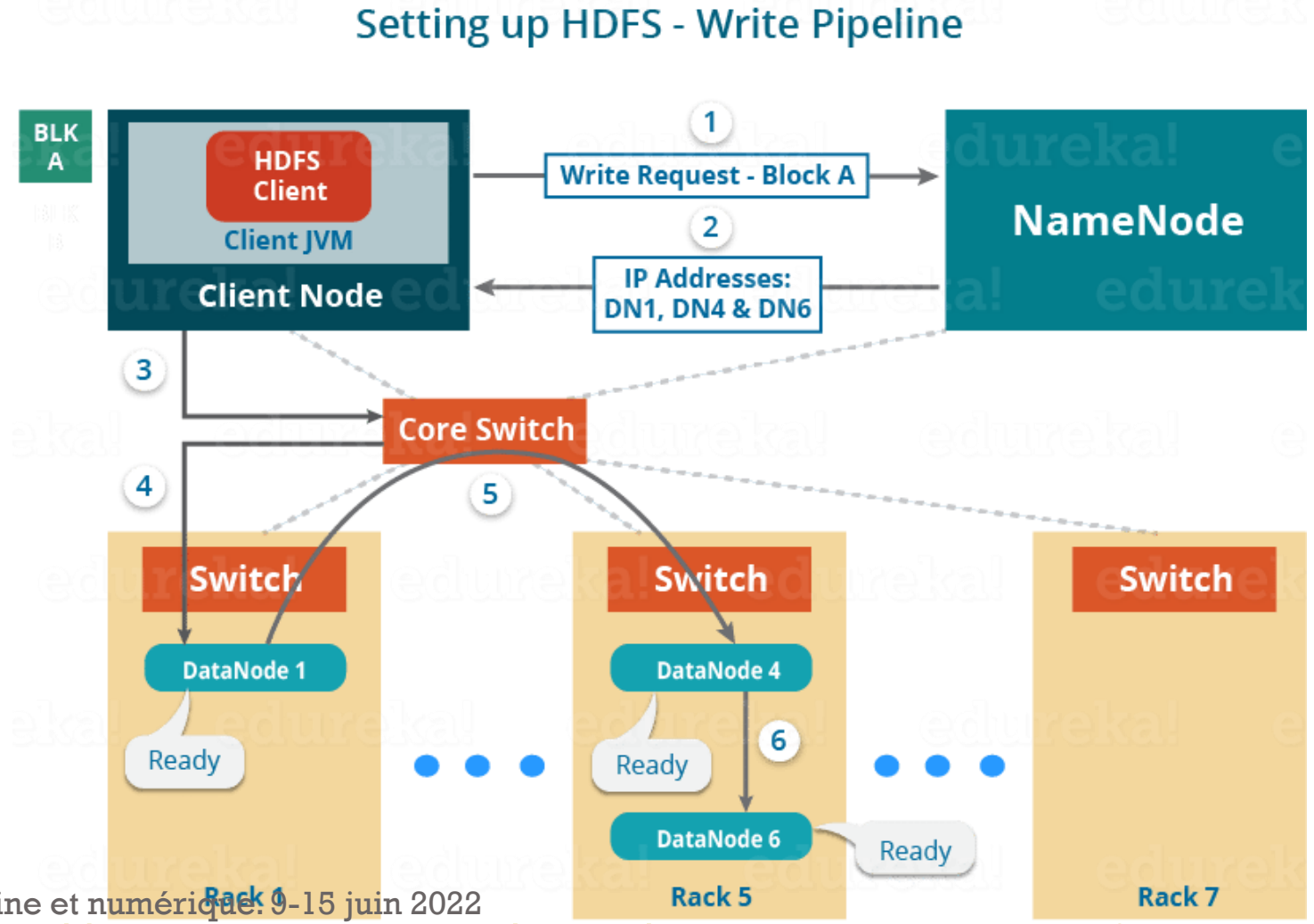
OPERATIONS-HDFS: AJOUT DES FICHIERS

- Le fichier est ajouté à la mémoire de NameNode
- L'opération d'ajout est conservée dans le journal d'édition (exemple: nano file1.doc)
- Les données sont écrites en blocs dans les DataNodes
- La copie vers les DataNodes est chaînée
- Si au moins une écriture pour chaque bloc réussit, l'écriture est réussie (La réplication peut être gérée ultérieurement)

OPERATIONS-HDFS: ECRITURE DES DONNÉES

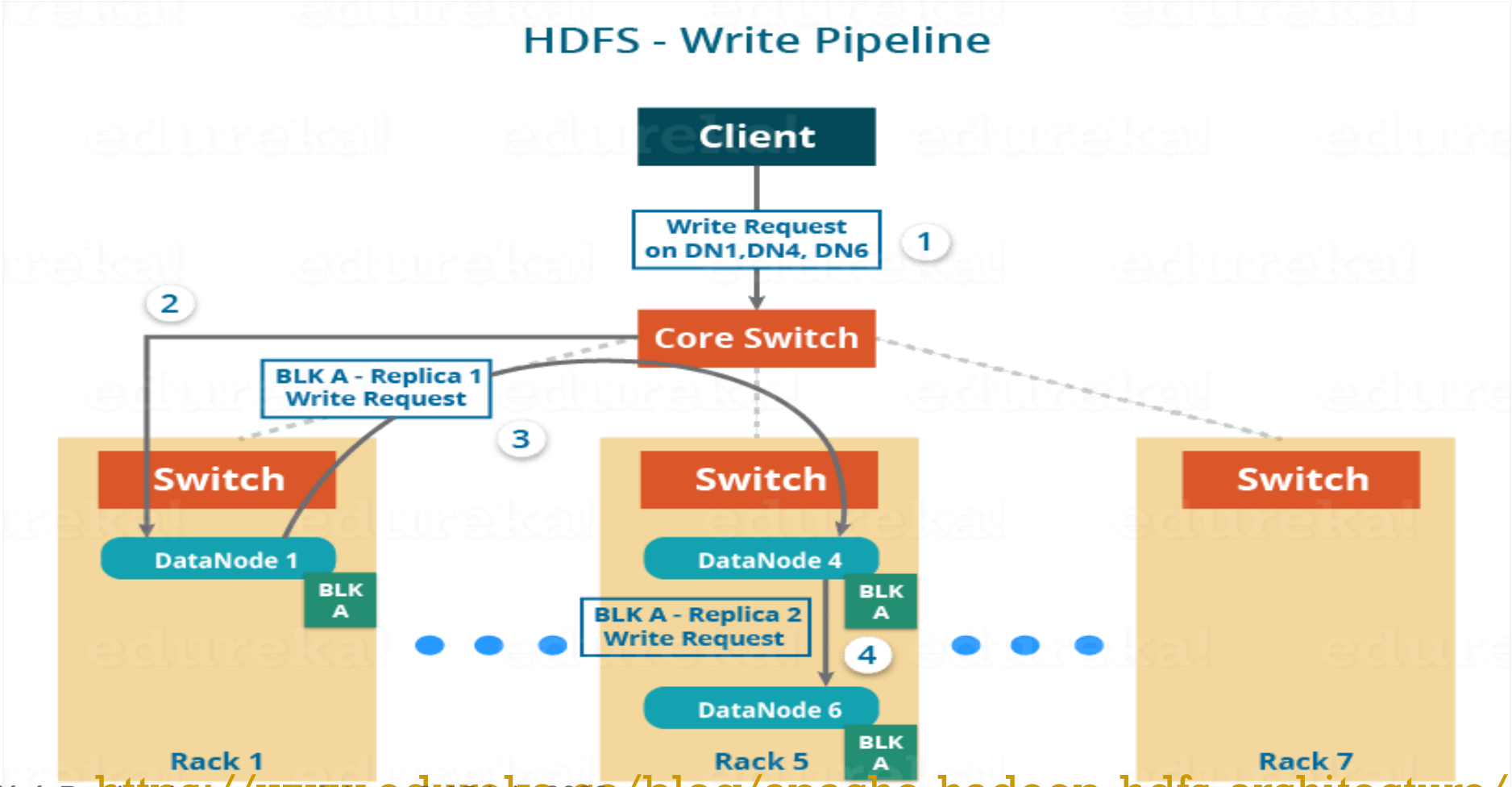
- 3 étapes essentielles
  - Pipeline d'écriture
  - Copie
  - Notification

PIPELINE D'ÉCRITURE



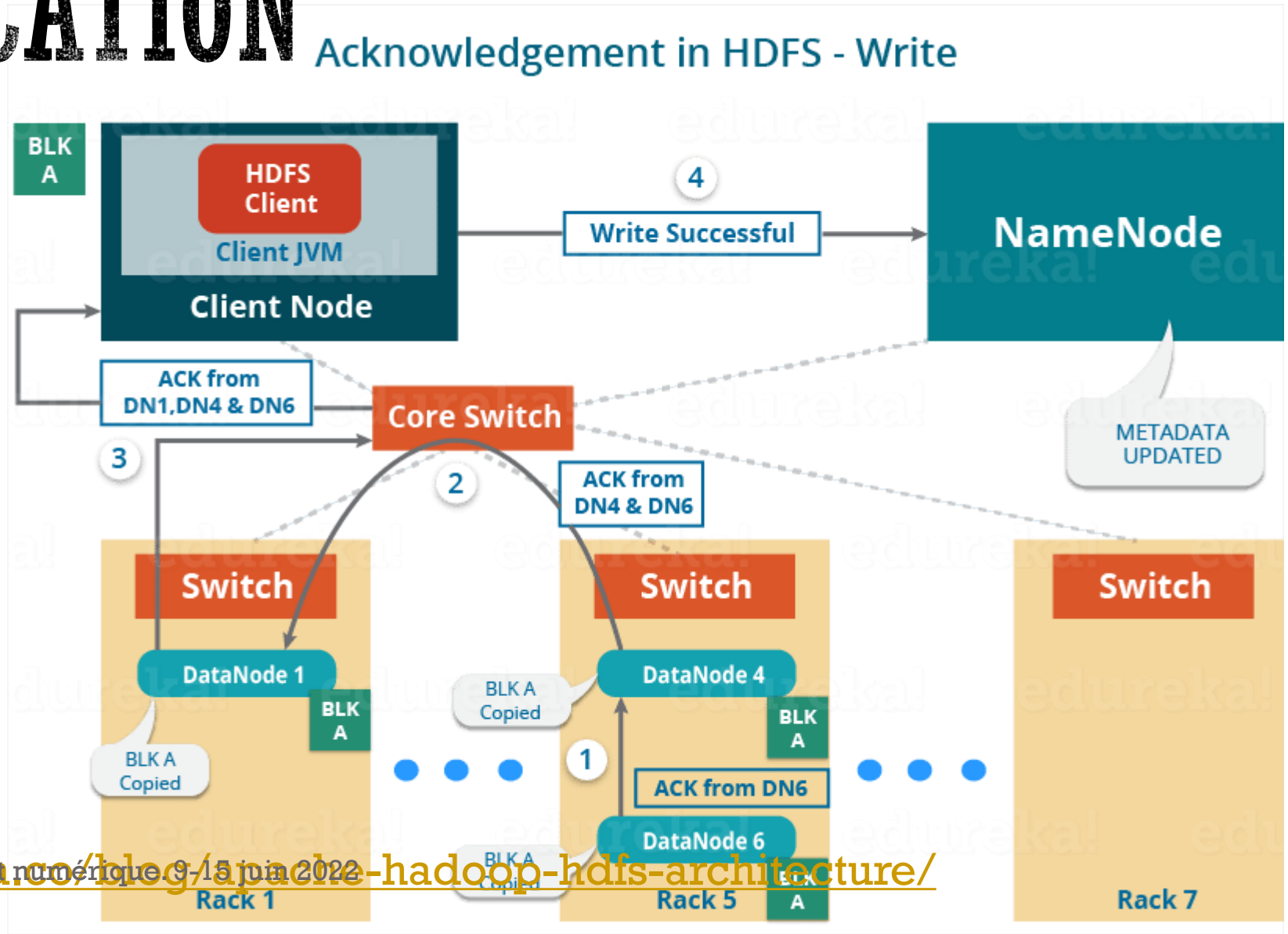
https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/

COPIE



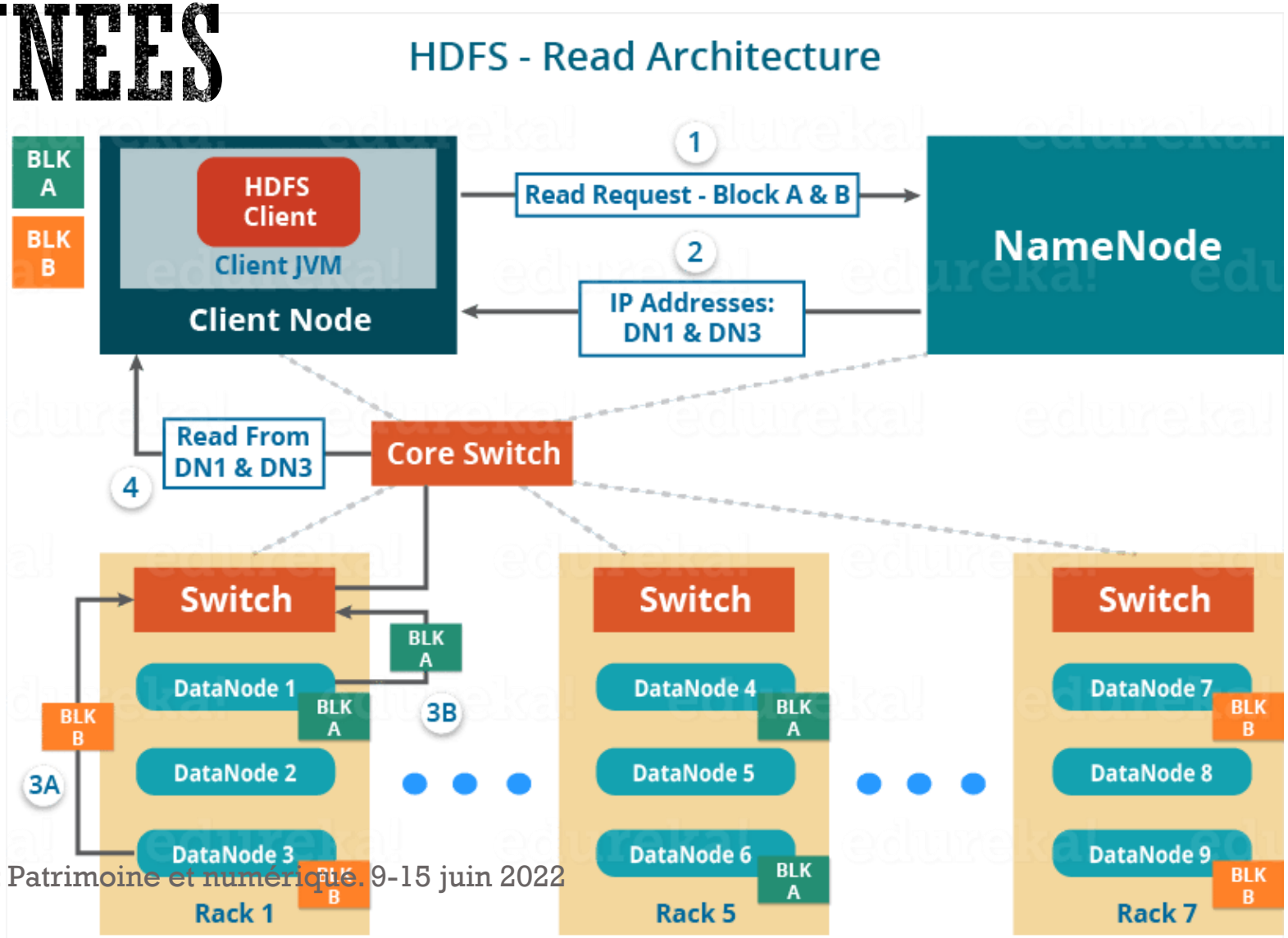
https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/

NOTIFICATION



https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/

OPERATIONS-HDFS: LECTURE DE DONNÉES

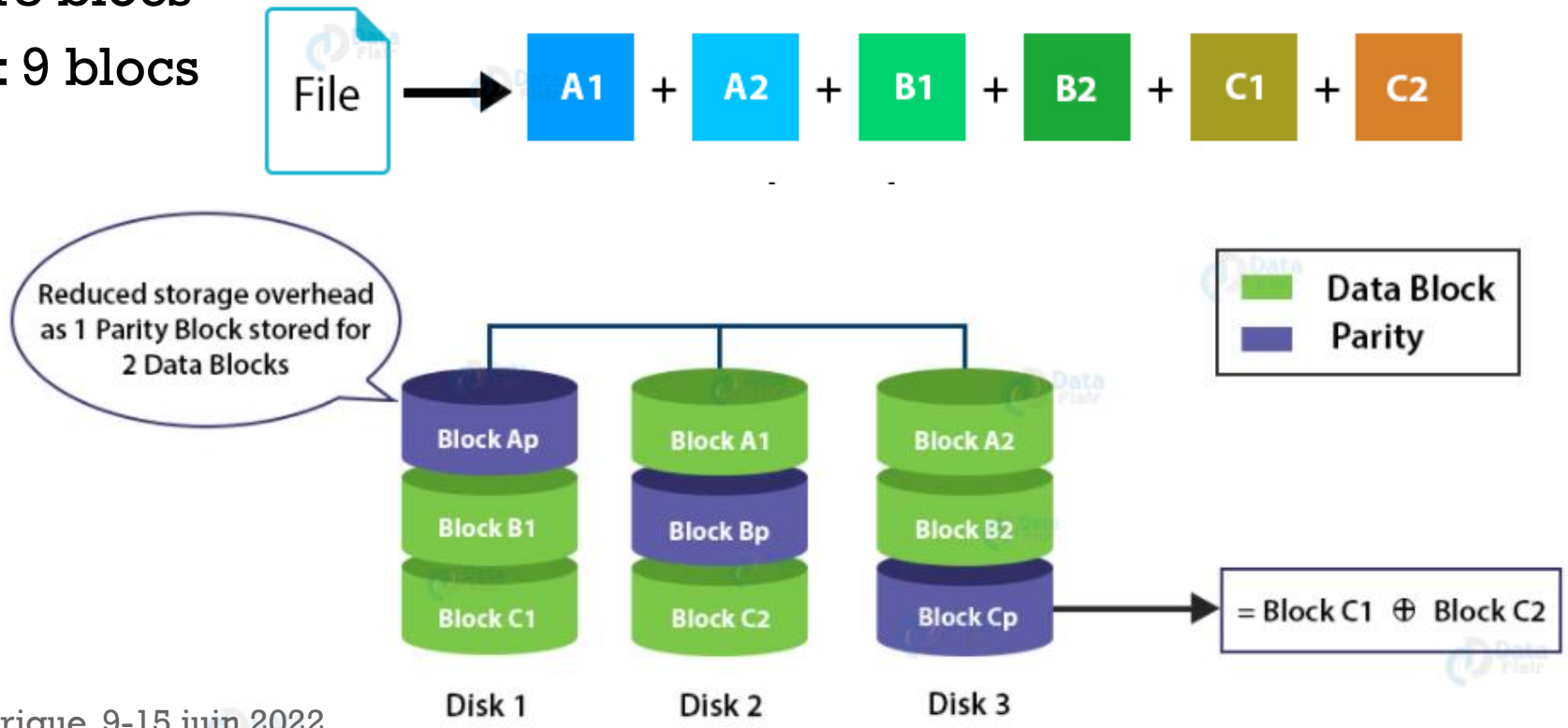


HDFS-3

- Réplication entraine **200% storage overhead**
- Hdfs erasure coding
  - Assurer la tolérance aux panne et réduit le stockage par 50%
  - Utilise un contrôleur RAID (redundant Array of Inexpensive Disks) qui contrôle comment les données sont stockées dans les disques
    - Striping: diviser logiquement les fichiers en unités plus petites
    - Les unités consécutives sur différents disques.
    - Les blocs de parité sont calculées en fonction de l'algorithme « erasure coding »
    - L'erreur dans n'importe quelle bloc récupérée à partir des données restantes des blocs de parité;

HDFS-3

- erasure coding exemple: 1 fichier de 6 blocs
  - Réplication *3: 18 blocs
  - erasure coding: 9 blocs



https://data-flair.training/blogs/hadoop-hdfs-erasure-coding/

HDFS-COMMANDES

- Syntaxe **hdfs dfs <args>**
- Hadoop accepte plusieurs commandes posix-like
  - cat, chgrp, chmod, chown, cp, du, ls, mkdir, mv, rm, stat, tail
- Exemples:
  - hdfs dfs –ls
  - hdfs dfs –chown esi. /myfile.txt
  - hdfs dfs –cat /myfile.txt
- Hadoop dispose également de ses propres commandes
- Examples:
  - Hdfs dfs –copyFromLocal myfile.txt /myfile.txt
  - Hdfs dfs –copyToLocal myfile.txt /myfile.txt

MAPREDUCE

OBJECTIFS

- Hadoop-mapreduce: unité de traitement des données présentes dans HDFS
- Les données dans HDFS sont
  - découpées en blocs de 128 MB
  - Stockées séparément dans les DataNodes (selon disponibilité)
- Les applications natives ne sont pas capables d'accéder et de traiter ces données
- Il est nécessaire d'utiliser un Framework capable de:
  - Trouver les données dans le cluster: comprendre la logique HDFS
  - Paralléliser le traitement sur les nœuds
  - Récupérer le résultat

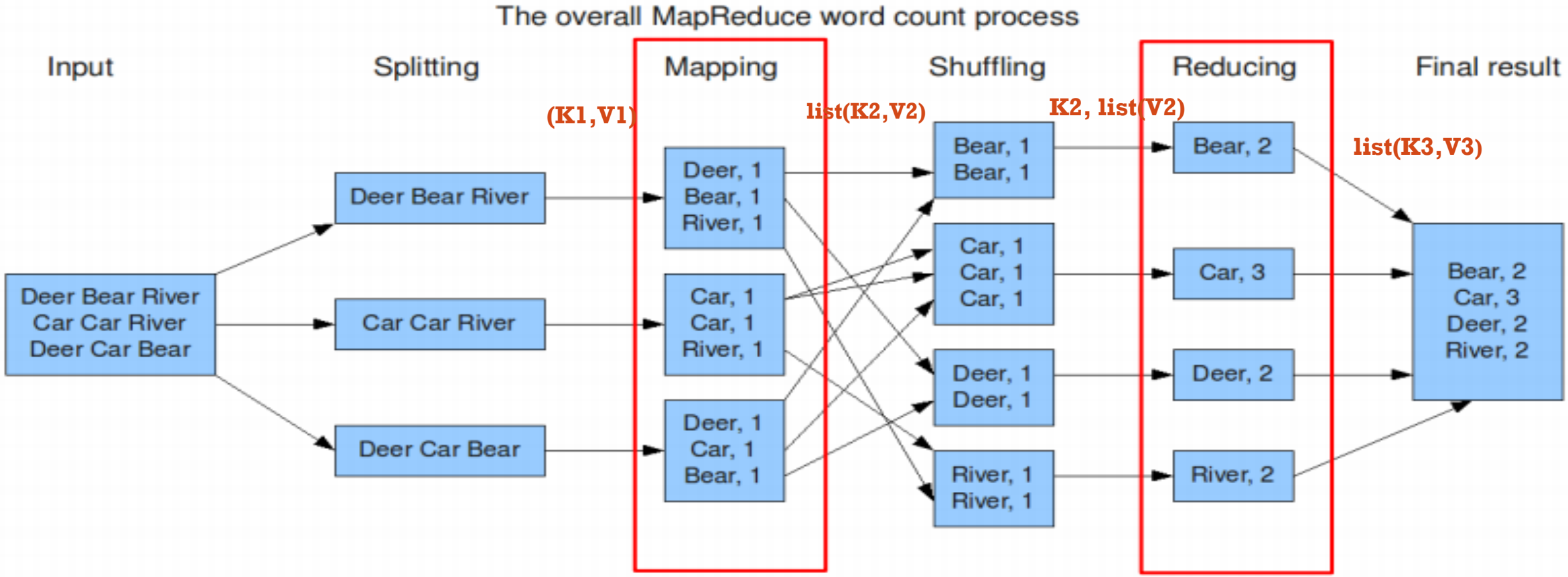
AVANTAGES

- Traitement parallèle
  - Le temps de traitement est réduit
- Data locality
  - Déplacer les données vers les unités de traitement est un processus couteux (Big Data)
  - MapReduce déplace les traitements vers les données
    - Le traitement est plus petit que les data
    - Le même traitement est exécuté sur les blocs de données, en plusieurs emplacement + en parallèle
    - Permet d'éviter le déplacement des données
    - Les résultats sont centralisés

MAPREDUCE

- Un paradigme de traitement distribué qui s'applique sur de gros volume de données.
- Son concept est simple : diviser pour régner.
- Ce paradigme vise à assurer une programmation parallèle via la répartition de la charge sur un grand nombre de nœuds de calcul.
- Mapreduce est basé sur deux fonctions Map() et Reduce()
- Map() :
  - Prend en entrée un ensemble de « clé, valeurs » (K,V)
  - Divise les données d'entrée en associant à chaque élément une clé
  - Retourne une liste intermédiaire de « clé, valeurs »
  - **Map(key, value)** -> **list(key1,value1)**
- Reduce():
  - Prend en entrée une liste intermédiaire de « **Clé1, Valeur1** »
  - fusionne les valeurs intermédiaires
  - Fournit en sortie un ensemble de « **Clé1,Valeur2** »
  - **reduce(key1, list(value1))** -> **value2**

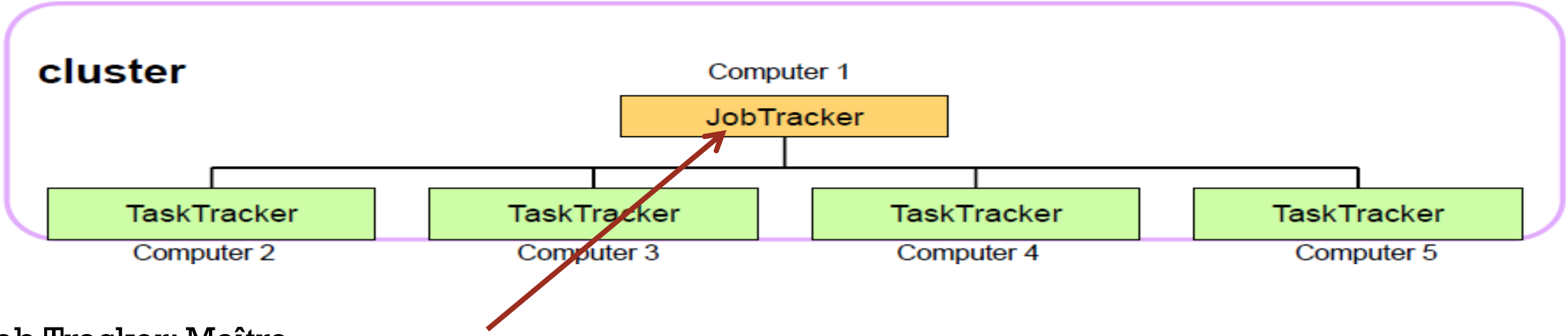
MAPREDUCE: EXEMPLE



EXEMPLES D'UTILISATION

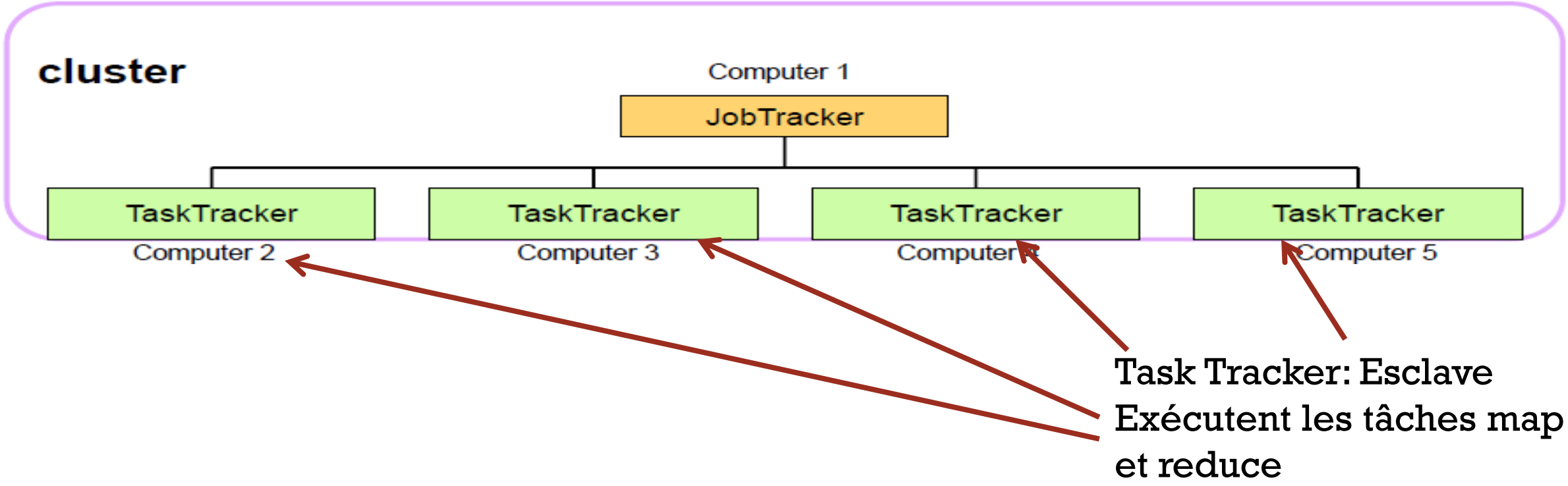
- Google: Google search: construction des index, Google news: regroupement des articles
- Yahoo!; Yahoo! Mail: détection des spam
- Facebook: Datamining

ARCHITECTURE HADOOP MAPREDUCE



Job Tracker: Maître
1- coordonne l'exécution des jobs,
2- décompose le job en tâches map et reduce destinées à chaque nœud
3-attribue et suit la progression de l'exécution des tâches

ARCHITECTURE HADOOP MAPREDUCE



LIMITES DE HADOOP/MAPREDUCE

- Montée en charge:
  - Le JobTracker effectue plusieurs types d'opérations:
    - Gestion des ressources: assigne les ressources, surveille l'utilisation des ressources……
    - Gestion des tâches: répartie les tâches, surveille l'exécution des tâches….
    - Planifications des tâches
  - Atteint ses limites entre 4000 et 5000 nœuds (40000 parallel tasks)

LIMITES DE HADOOP/MAPREDUCE

- Single Point of failure:
  - Un seul master: JobTracker
  - L'indisponibilité du JobTracker rend tout le cluster indisponible

LIMITES DE HADOOP/MAPREDUCE

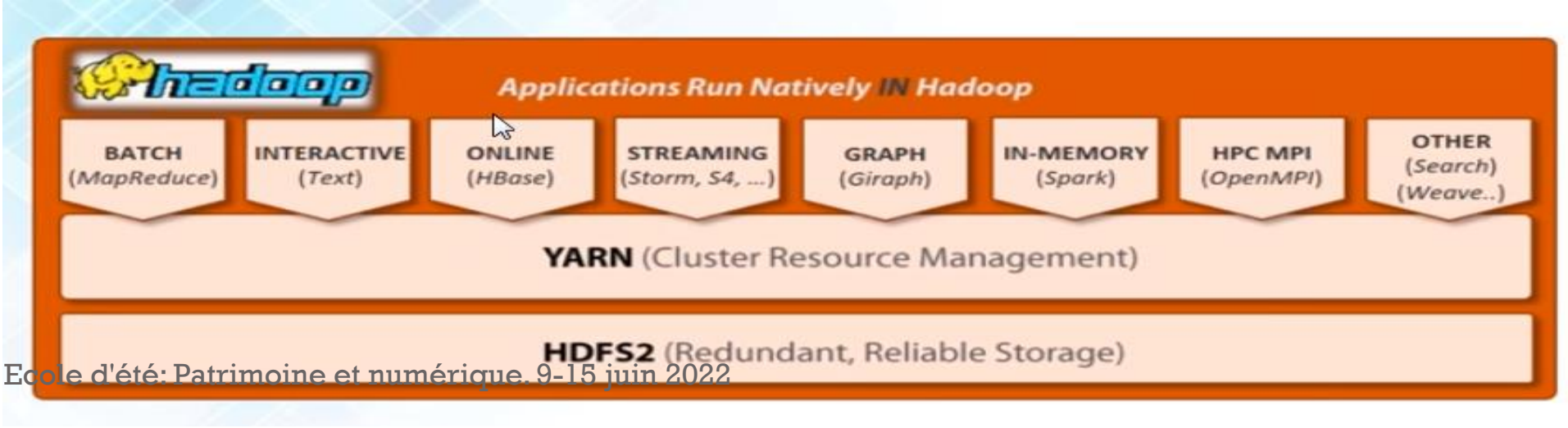
- Des ressources fixes
  - Les ressources de chaque nœud (RAM, CPU, HDD…) sont divisées en un nombre fixe entre les opérations map et reduce (slots)
    - Un nœud ne peut pas lancer plus de tâches map ou reduce que le nombre préalablement fixé
    - Si le nombre map de slots map est atteint aucune autre opération map ne sera lancée même si les slots reduce sont disponibles

LIMITES DE HADOOP/MAPREDUCE

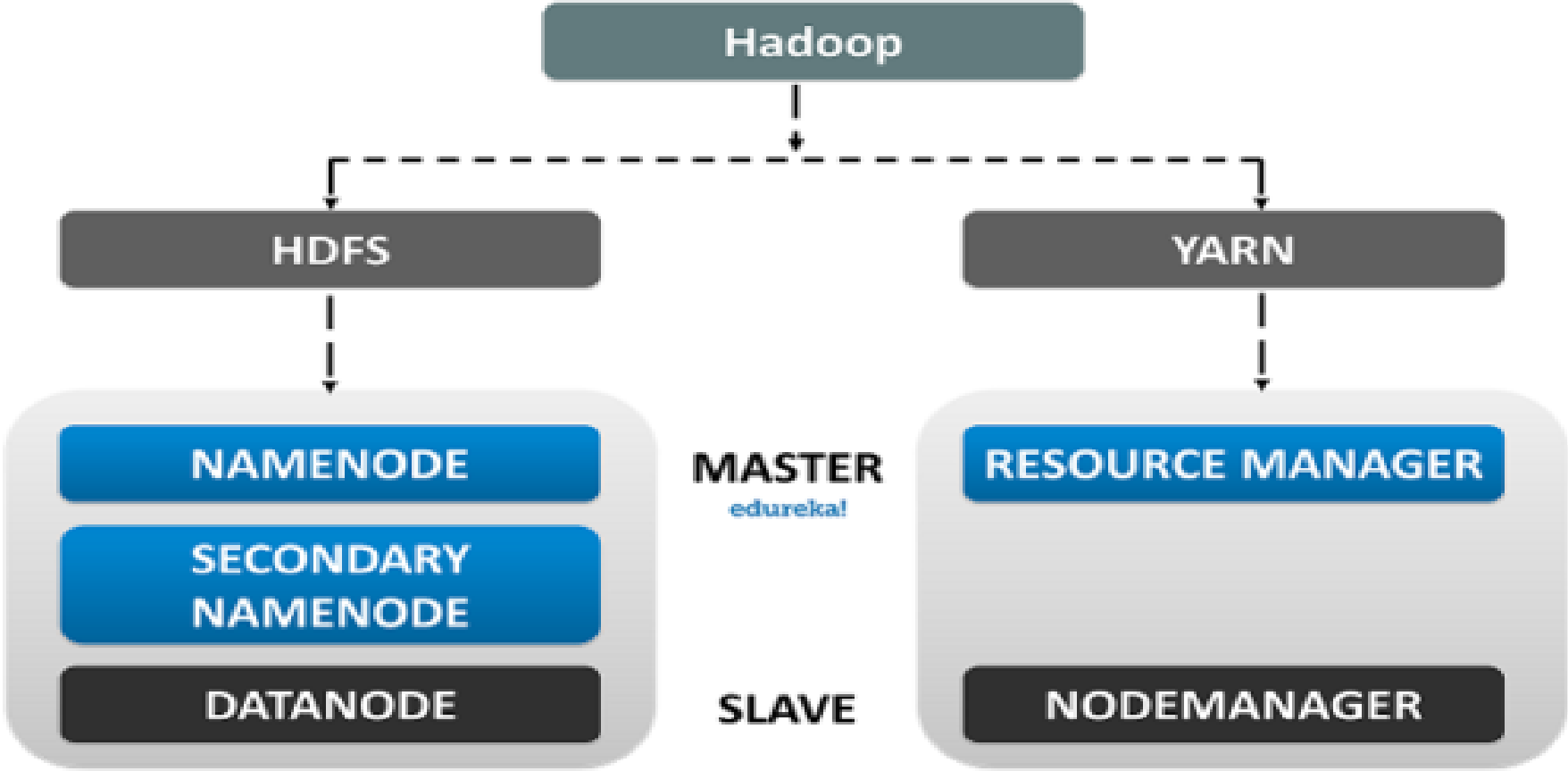
- runs only Map/Reduce jobs
  - Hadoop a été conçu pour exécuter uniquement des tâches MapReduce.
  - JobTracker et une application créée uniquement pour MapReduce et ne peut pas créer, assigner, suivre des tâches non Mapreduce

YARN: YET ANOTHER RESOURCE NEGOTIATOR

- Introduit à partir de la version Hadoop 2.0
- Une couche qui sépare la couche de gestion des ressources de la couche des composants de traitement
- Ouvre le chemin pour les autres Framework de l'écosystème



HADOOP 2.X DEAMONS



https://www.edureka.co/blog/hadoop-tutorial/

COMPOSANTS YARN

- Ressource manager: Gère les ressources du cluster
  - Reçoit les requêtes clientes
  - Alloue les ressources aux jobs qui s'exécutent dans le cluster Hadoop
  - Tourne dans une machine performante: master machine
  - Possède deux composants: scheduler et Application manager (s'assure que chaque tâche est exécutée)
  - Gère la liste des ApplicationMaster lancée dans le cluster
- Application manager: un composant du ressource manager
  - valide d'abord les spécifications de l'application et rejette toute application qui demande des ressources non disponibles
  - s'assure qu'aucune autre application n'a déjà été soumise avec le même ID d'application,
  - Enregistre et gère les applications terminées
  - Assure qu'un application master a été assigné pour chaque tâche.

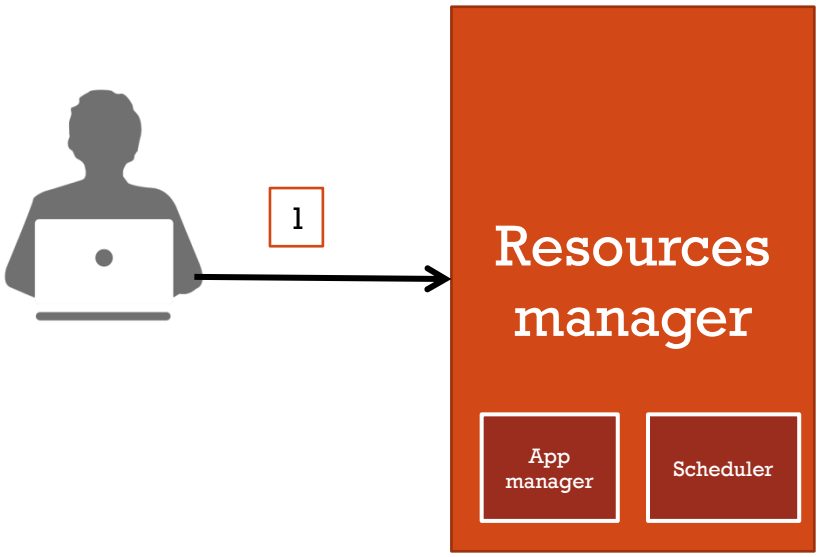
COMPOSANTS YARN

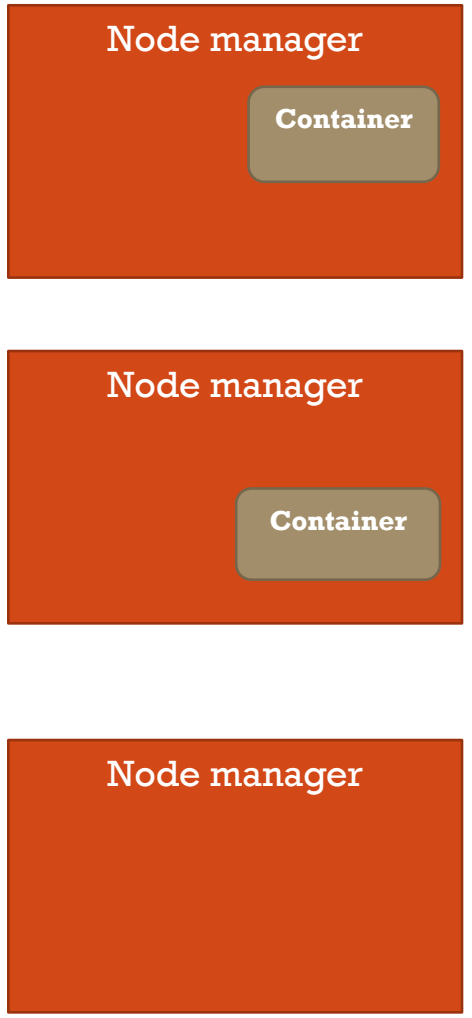
- Node manager: un task tracker généralisé
  - L'esclave
  - Responsable du Application master et du container
  - Un pour chaque nœud
  - Supervise les ressources du dataNode
  - Gère les ressources du nœud
- Application master: Gère les ressources nécessaires pour chaque application
  - Un seul par job
  - Processus qui s'exécute sur machine esclave
  - Il est créé par ressource manager pour exécuter un job
  - Négocie les ressources avec RM et coordonne avec NM pour exécuter la tâche
- Container: un ensemble de ressources (RAM, CPU, HDD, etc.)
  - exécute un processus spécifique à l'application
  - Créé par NM
  - Alloue les ressources dans l'esclave
  - Exécute les jobs

PROCESSUS

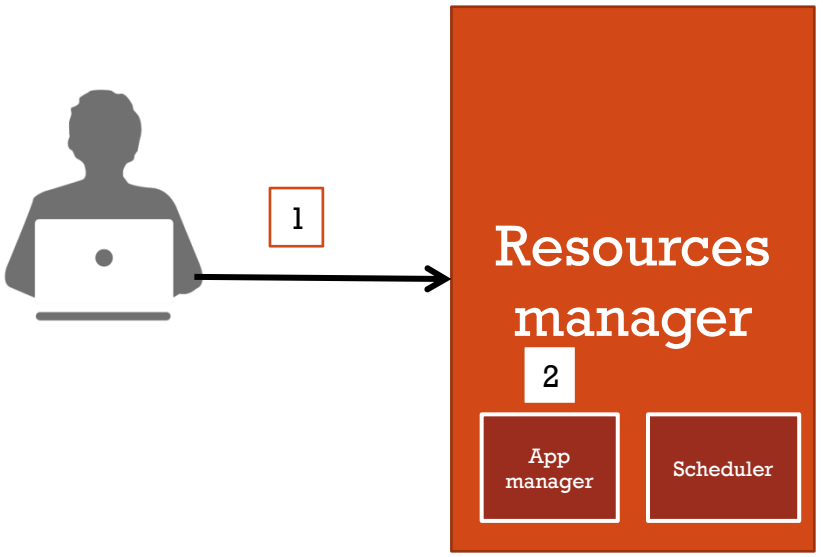
- 5 étapes pour lancer une application
  1. Le client soumet une application au Ressources manager (RM)
  2. L'application manager(AManager) vérifie l'ID de l'application et valide les spécifications
  3. Le RM trouve un node manager (NM) avec un container libre et lance l'application master (AMaster)
  4. L'application master demande les ressources spécifiques au RM
  5. Le RM alloue un container
  6. L' AMaster contacte le NodeManager pour exécuter les tâches map et reduce
  7. À la fin de l'exécution l'application master libère les containers sur le NM.
  8. L' Amanager enregistre la fin de l'application

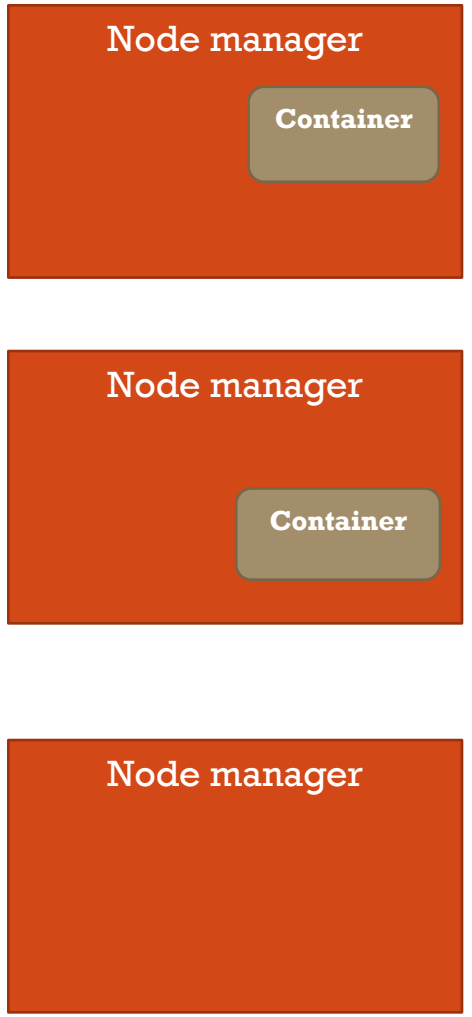
PROCESSUS


1
Resources manager
App manager
Scheduler

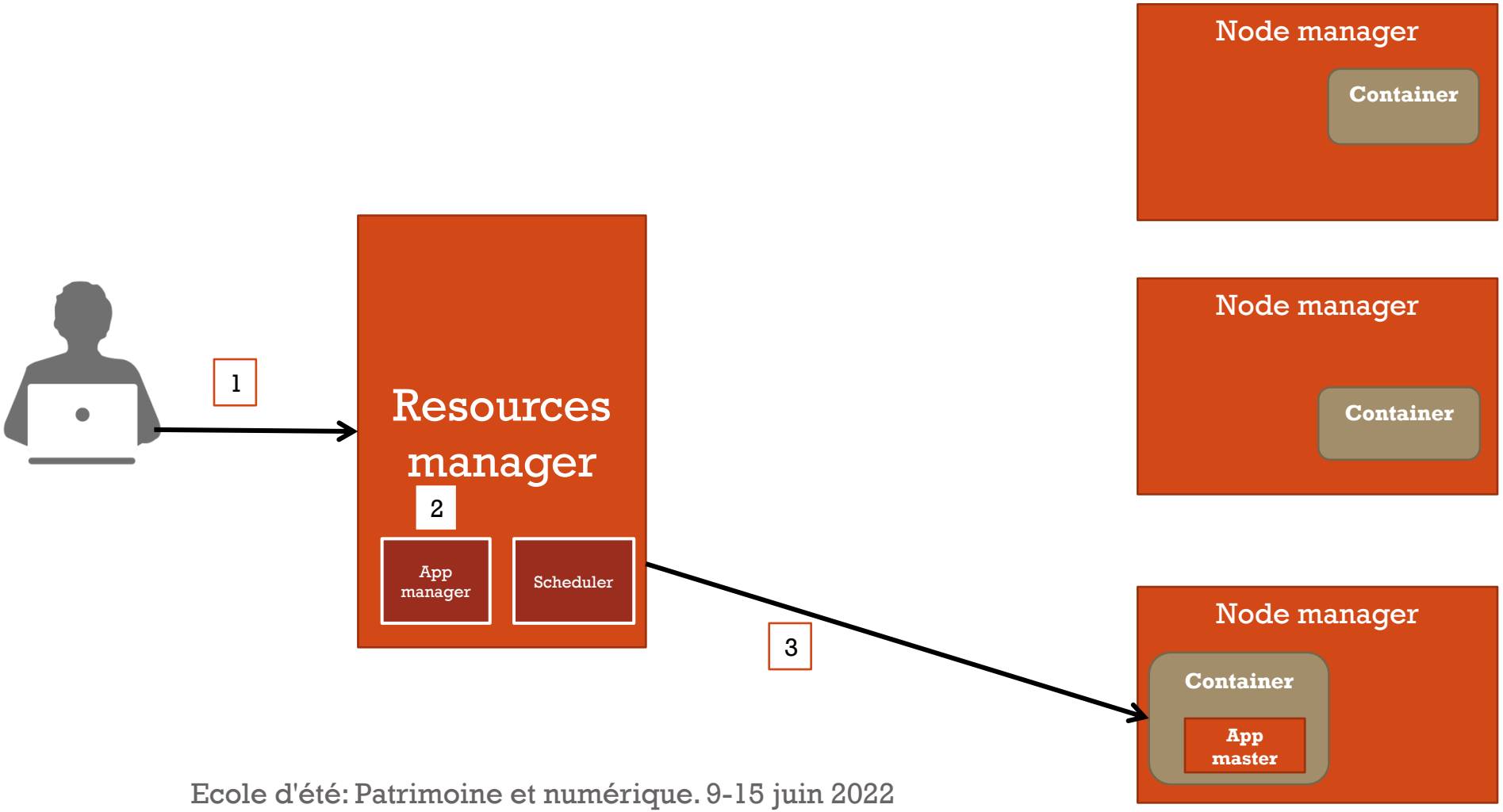

Node manager
Container
Node manager
Container
Node manager

PROCESSUS

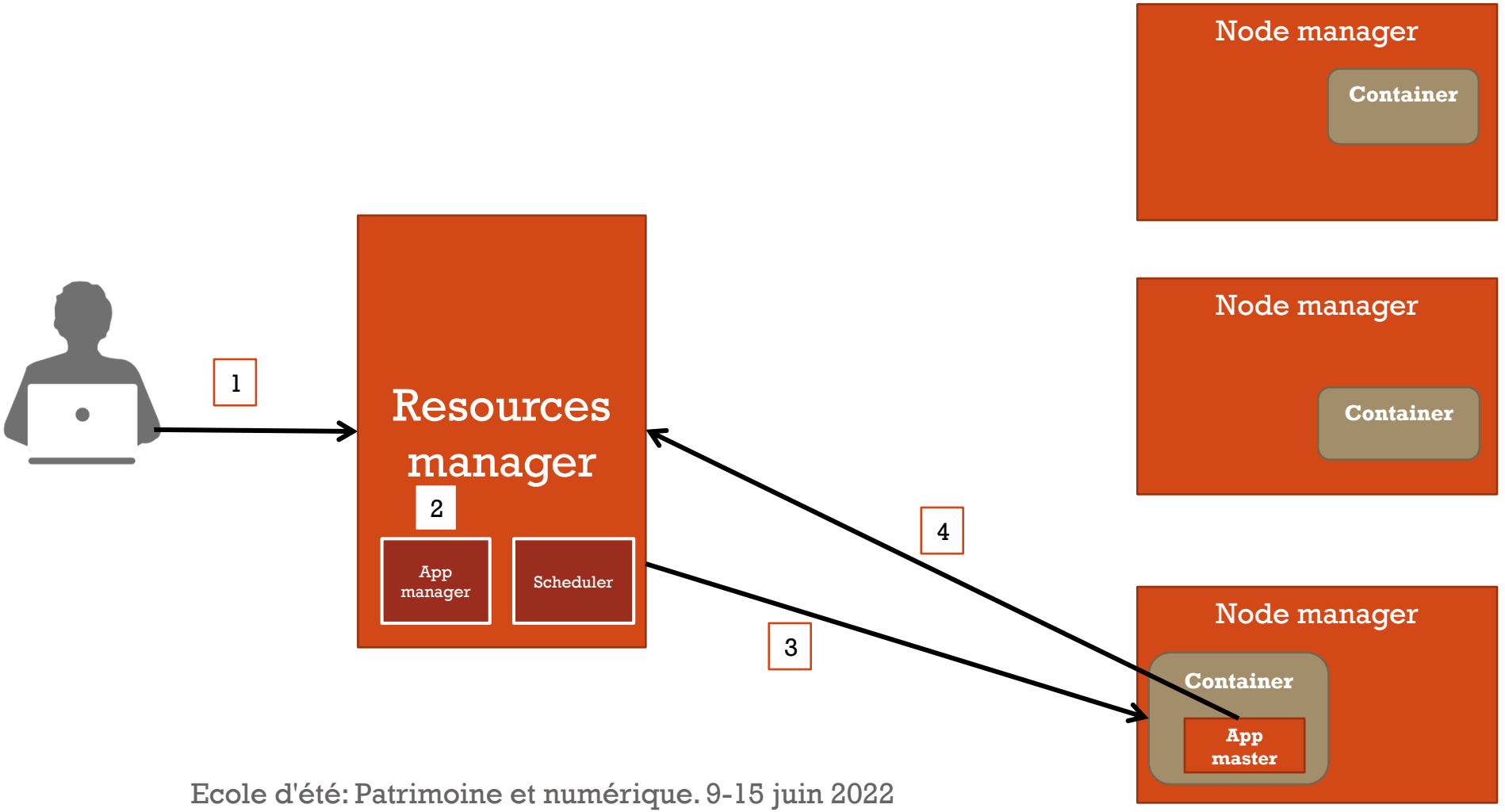

1
Resources manager
2
App manager
Scheduler


Node manager
Container
Node manager
Container
Node manager

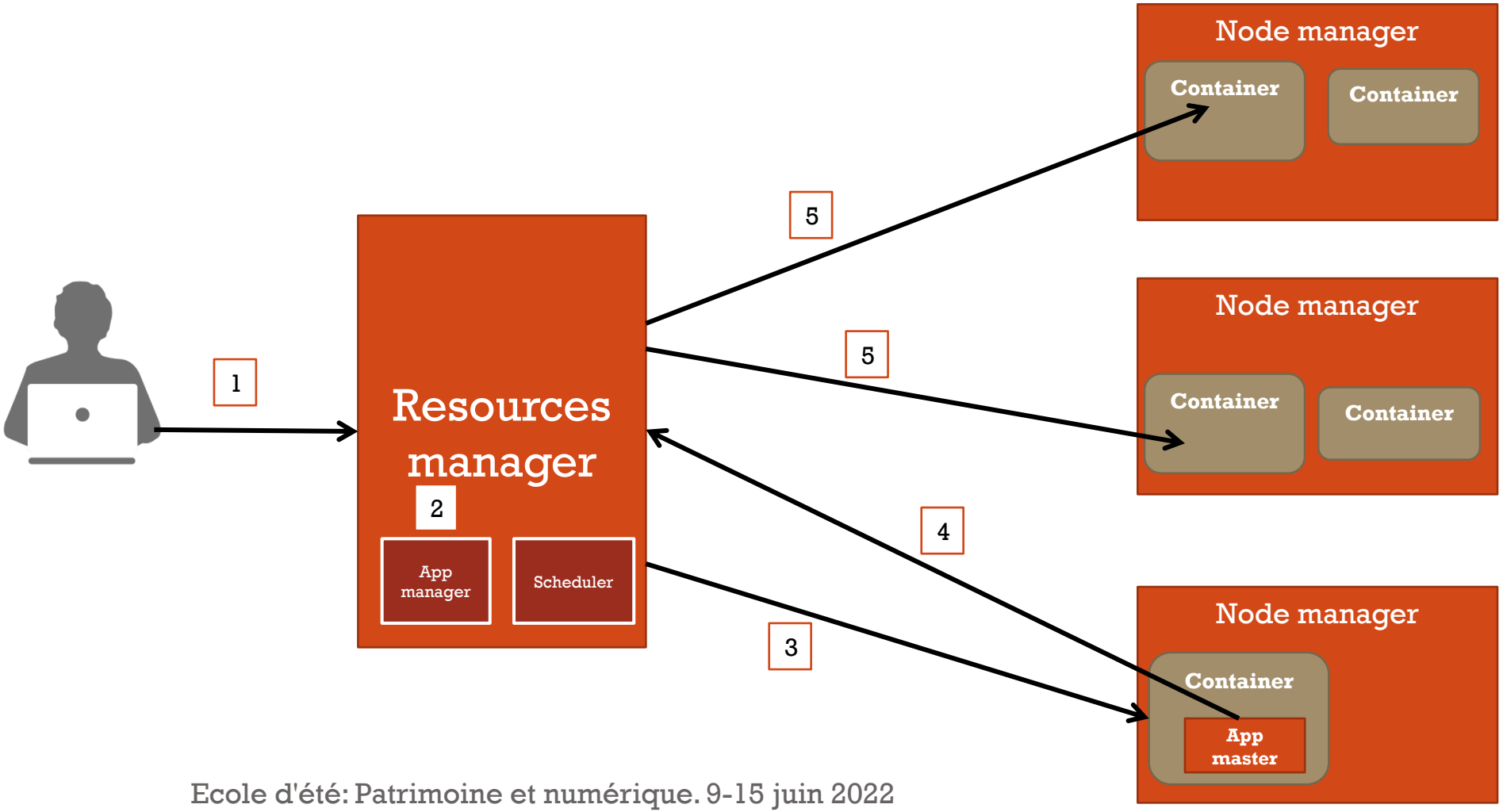
PROCESSUS


Node manager
Container
Node manager
Container
Resources manager
2
App manager
Scheduler
1
3
Node manager
Container
App master

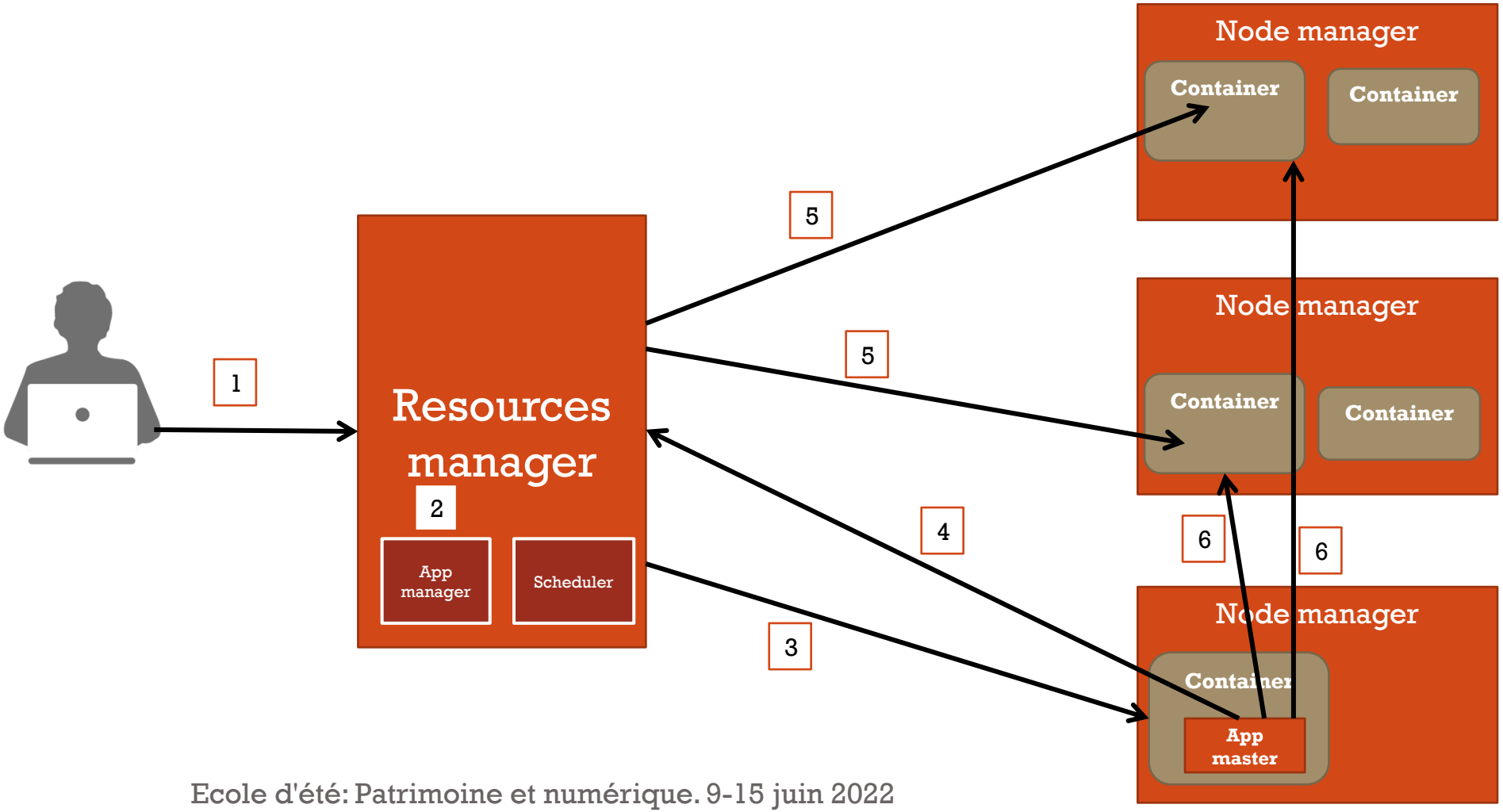
PROCESSUS


Node manager
Container
Node manager
Container
Resources manager
1
2
App manager
Scheduler
3
4
Node manager
Container
App master

PROCESSUS


Node manager
Container
Container
Node manager
Container
Container
Node manager
Container
App master
Resources manager
2
App manager
Scheduler
1
5
5
4
3

PROCESSUS


Node manager
Container
Container
Node manager
Container
Container
Node manager
Container
App master
Resources manager
2
App manager
Scheduler
1
3
4
5
5
6
6


```
public class WordCount {
public static void main(String [] args) throws Exception
{
Configuration c=new Configuration();
String[] files=new GenericOptionsParser(c,args).getRemainingArgs();
Path input=new Path(files[0]);
Path output=new Path(files[1]);
Job j=new Job(c,"wordcount");
j.setJarByClass(WordCount.class);
j.setMapperClass(MapForWordCount.class);
j.setReducerClass(ReduceForWordCount.class);
j.setOutputKeyClass(Text.class);
j.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(j, input);
FileOutputFormat.setOutputPath(j, output);
System.exit(j.waitForCompletion(true)?0:1);
}
public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable>{
public void map(LongWritable key, Text value, Context con) throws IOException, InterruptedException
{
String line = value.toString();
String[] words=line.split(",");
for(String word: words )
{
      Text outputKey = new Text(word.toUpperCase().trim());
  IntWritable outputValue = new IntWritable(1);
  con.write(outputKey, outputValue);
}
}
}

public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>
{
public void reduce(Text word, Iterable<IntWritable> values, Context con) throws IOException, InterruptedException
{
int sum = 0;
   for(IntWritable value : values)
   {
   sum += value.get();
   }
   con.write(word, new IntWritable(sum));
}

}
```

Apache Pig

```
lines = LOAD '/user/hadoop/HDFS_File.txt' AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;
grouped = GROUP words BY word;
wordcount = FOREACH grouped GENERATE group, COUNT(words);
DUMP wordcount;
```

HIVE

```
CREATE TABLE FILES (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE FILES;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, ' ')) AS word FROM FILES) w
GROUP BY word
ORDER BY word;
```

ECOSYSTÈME HADOOP