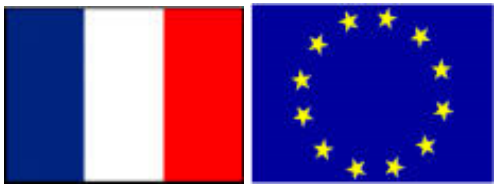


Big Data

Technologies Internes d'Hadoop

Stéphane Vialle **&** **Gianluca Quercini**

POLYTECH PARIS-SUD

Principes et technologie d'Hadoop

1. **Localité des données et des traitements**
2. Framework d'Hadoop
3. Mécanismes du Map-Reduce d'Hadoop
4. Système de fichiers distribué d'Hadoop (HDFS)
5. Allocation et gestion de ressources d'Hadoop

Localité des données et des traitements

C'est toujours l'accès aux données qui coute cher, pas le calcul lui-même (une fois les données arrivées dans l'unité de calcul)

Big Data *façon Hadoop* :

Amener les codes de traitements aux données

→ Transformer momentanément en nœuds de traitement les nœuds de stockage des données traitées

→ Eviter de déplacer des données (très volumineuses) … mais …

→ les relire et les réécrire localement chaque fois que la RAM est pleine

Big Data *façon Spark* :

Amener les données à toutes les unités de calcul disponibles

→ Mais … lire les données et les écrire sur disque une seule fois

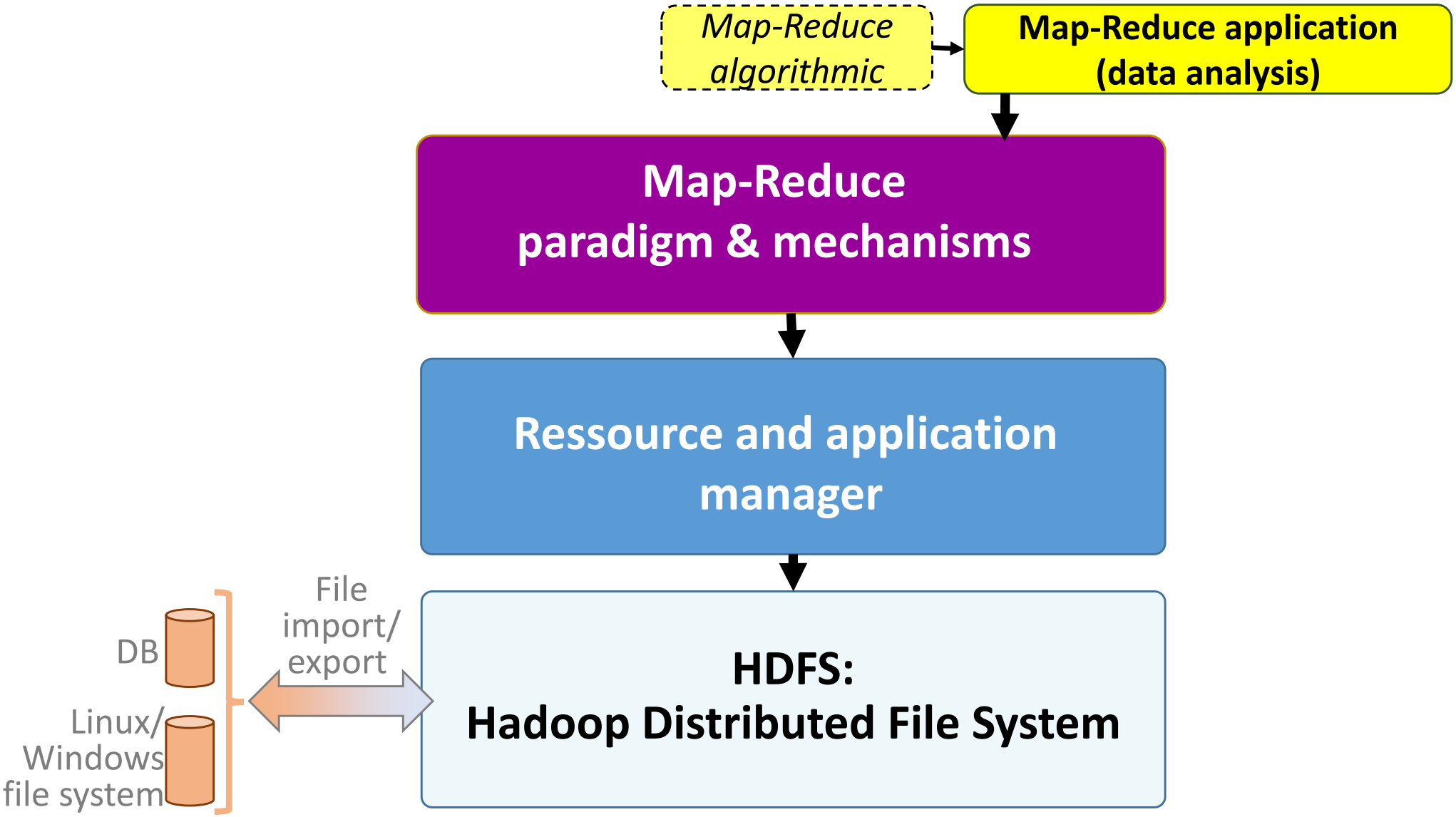
→ et … garder les données en RAM durant tout le traitement

POLYTECH PARIS-SUD

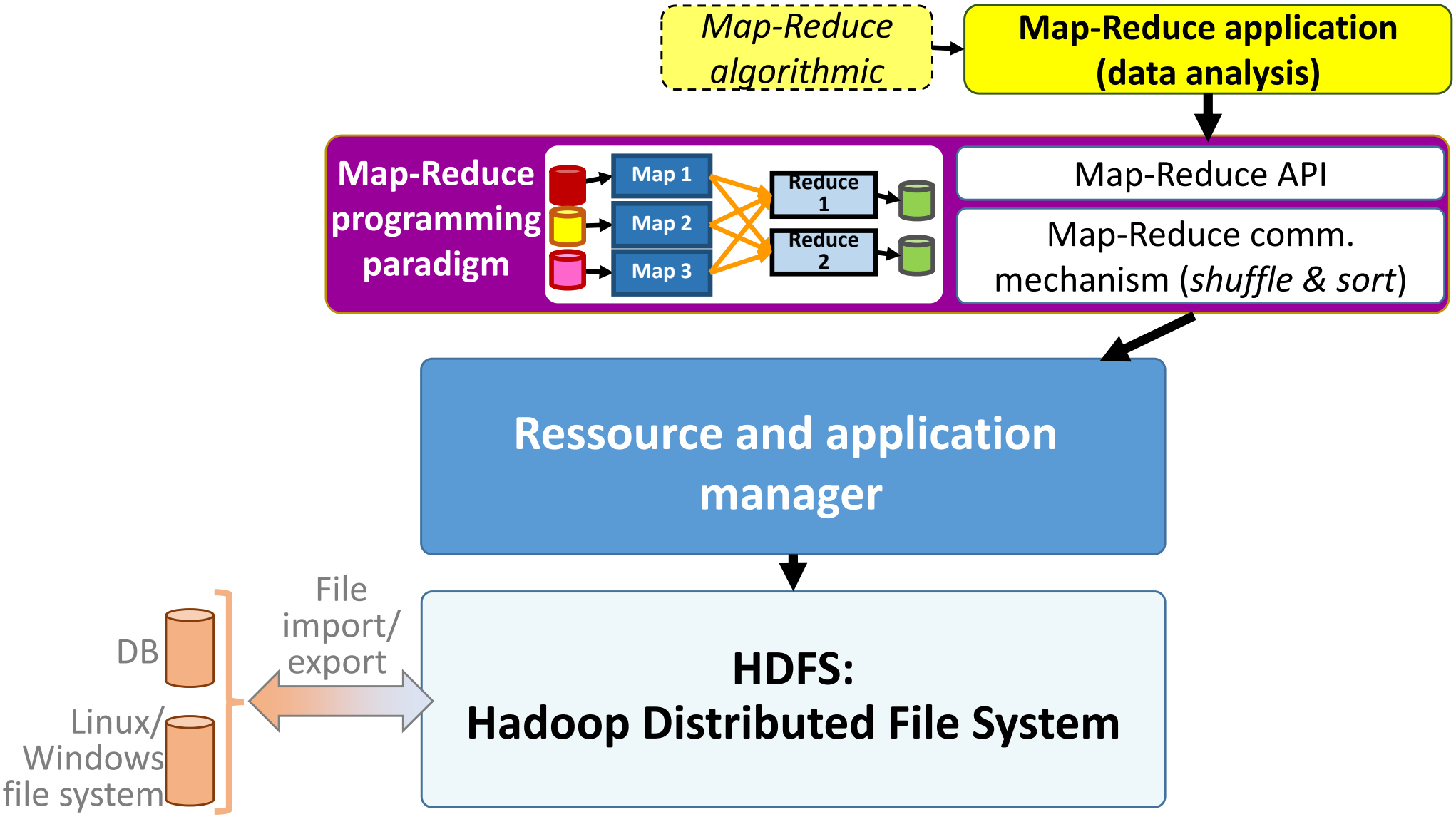
Principes et technologie d'Hadoop

1. Localité des données et des traitements
2. **Framework d'Hadoop**
3. Mécanismes du Map-Reduce d'Hadoop
4. Système de fichiers distribué d'Hadoop (HDFS)
5. Allocation et gestion de ressources d'Hadoop

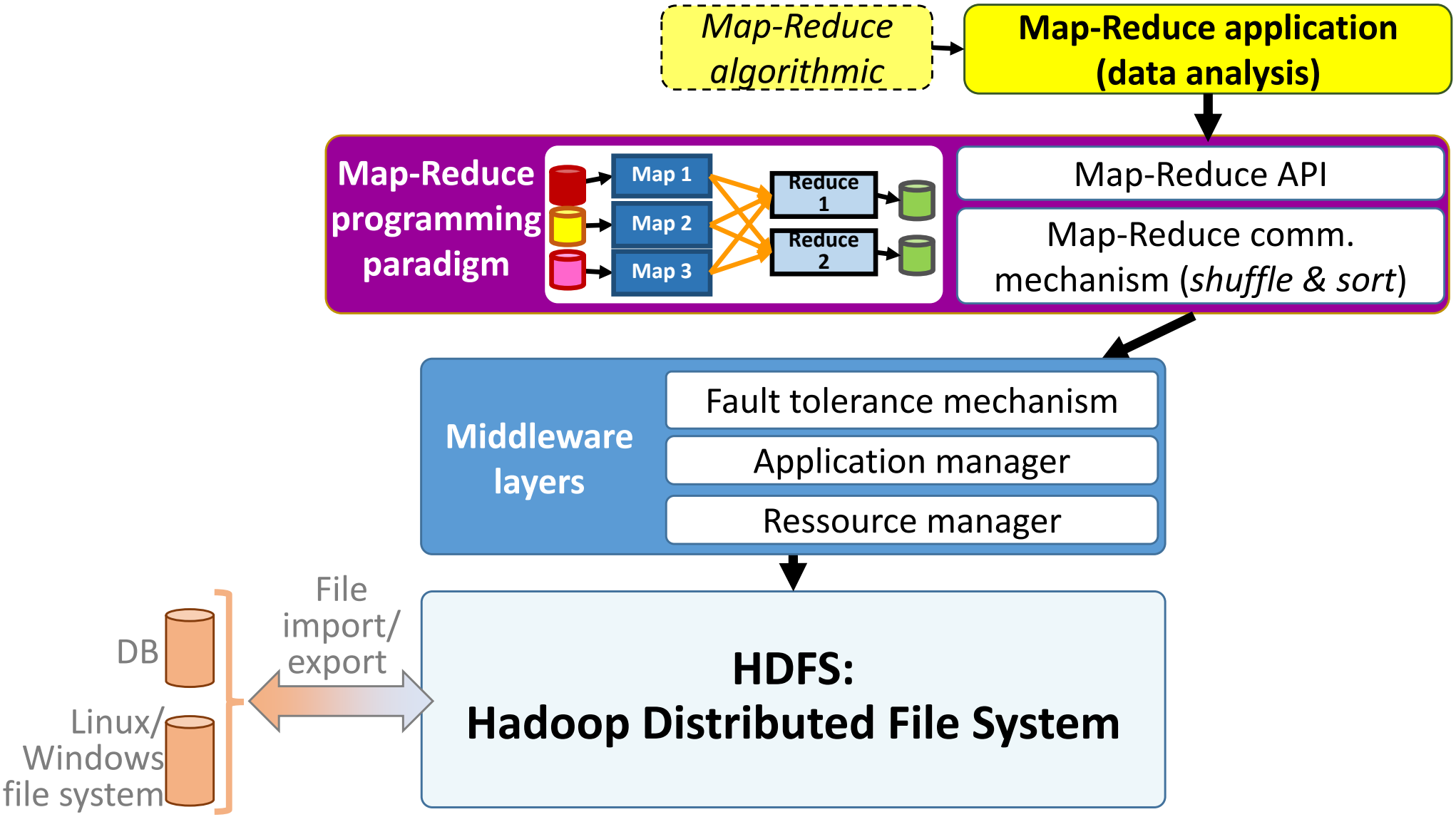
Concepts et pile logicielle



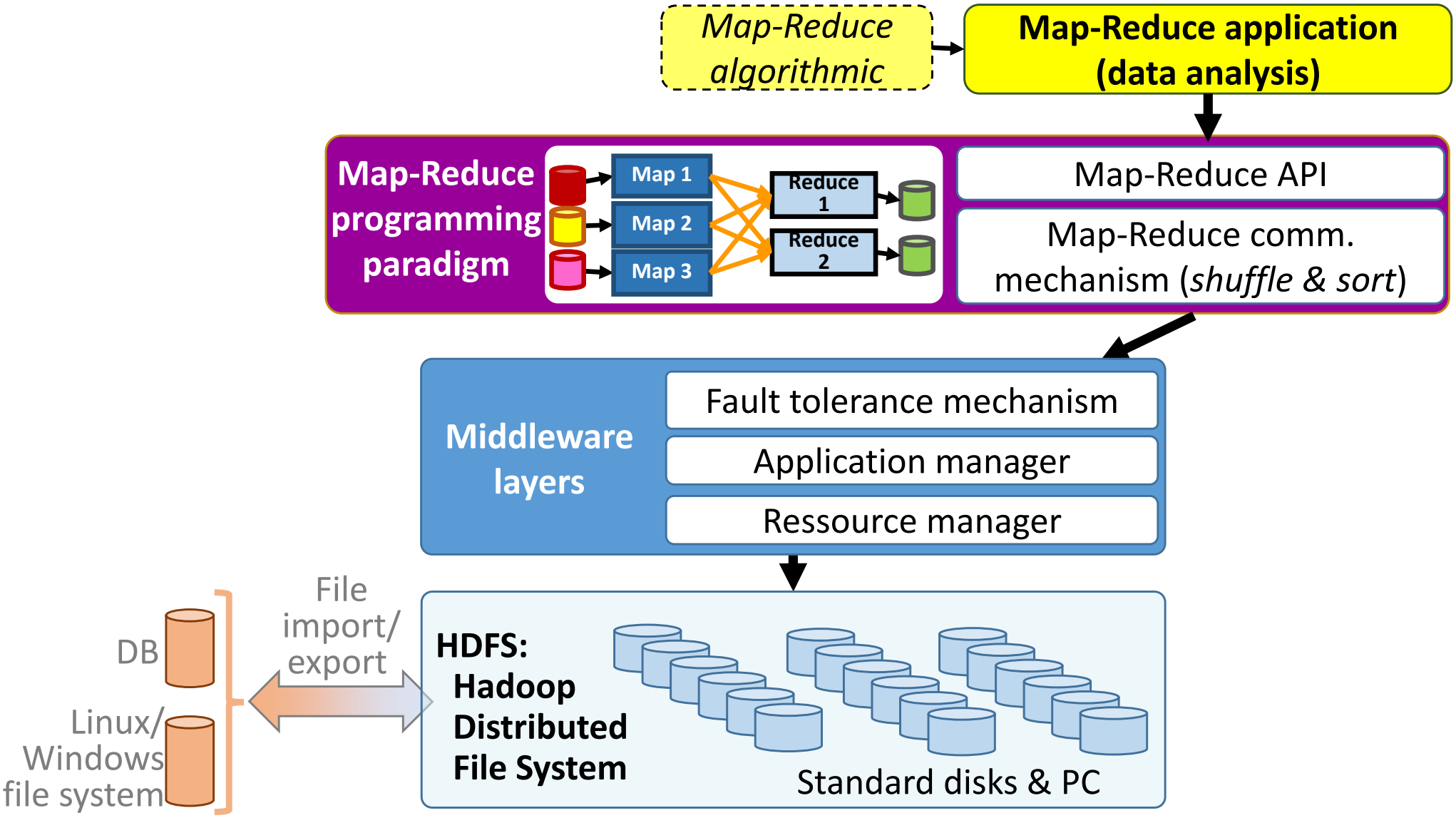
Concepts et pile logicielle



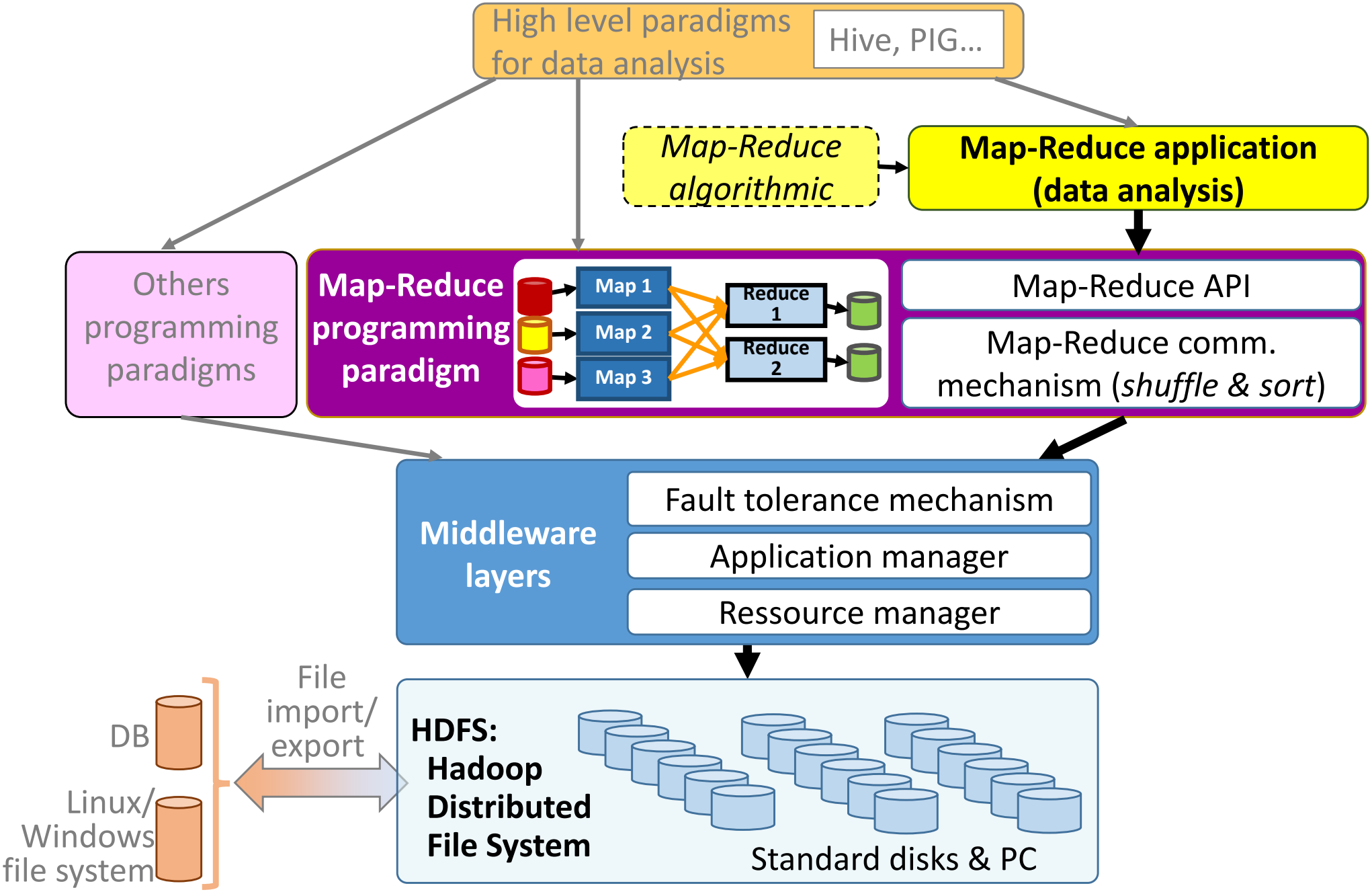
Concepts et pile logicielle



Concepts et pile logicielle



Concepts et pile logicielle



POLYTECH PARIS-SUD

Principes et technologie d'Hadoop

1. Localité des données et des traitements
2. Framework d'Hadoop
3. **Mécanismes du Map-Reduce d'Hadoop**
4. Système de fichiers distribué d'Hadoop (HDFS)
5. Allocation et gestion de ressources d'Hadoop

Chaîne d'opérations Map-Reduce

(17001, Pignon)
(16995, Martin)
(17012, Durand)
(16200, Dupont)
(17003, Dupond)
(16158, Martin)

Map function

```
f_map(key,val) {
  if key < 17000
    write (val, 1)
}
```

(Martin, 1)
(Dupont, 1)
(Martin, 1)

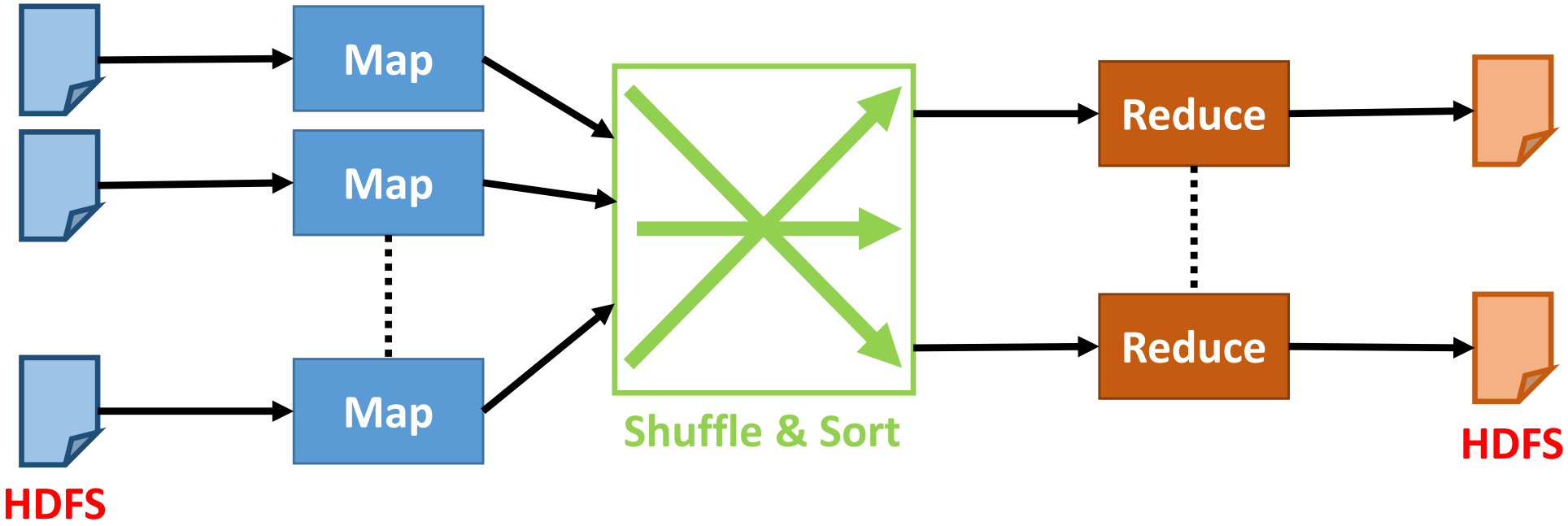
Shuffle & Sort

(Martin, (1,1))
(Dupont, (1))

Reduce function

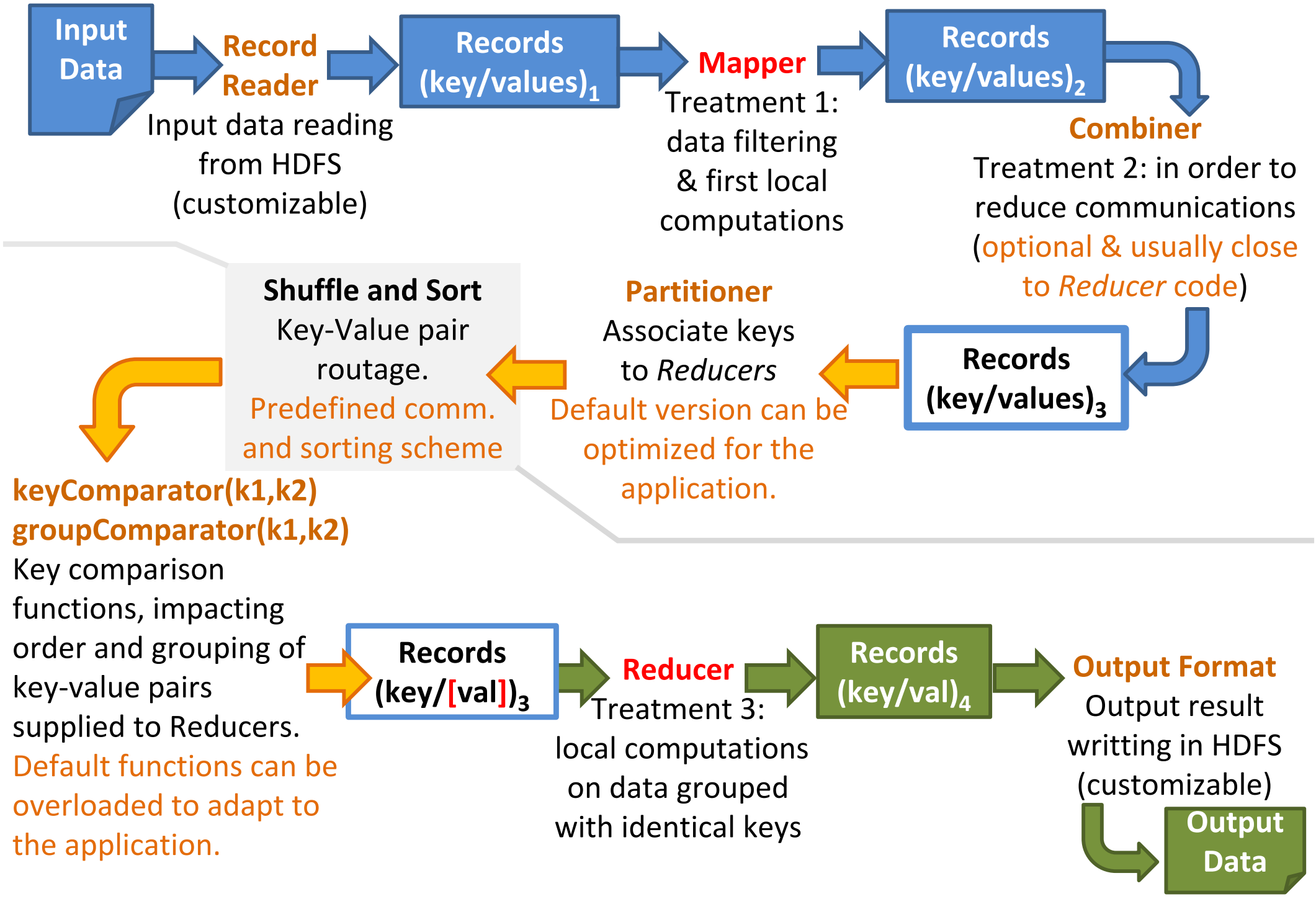
```
g_reduce(key, list_vals) {
  sum = 0
  for each val in list_vals
     sum += val
  write (key, sum)
}
```

(Martin, 2)
(Dupont, 1)

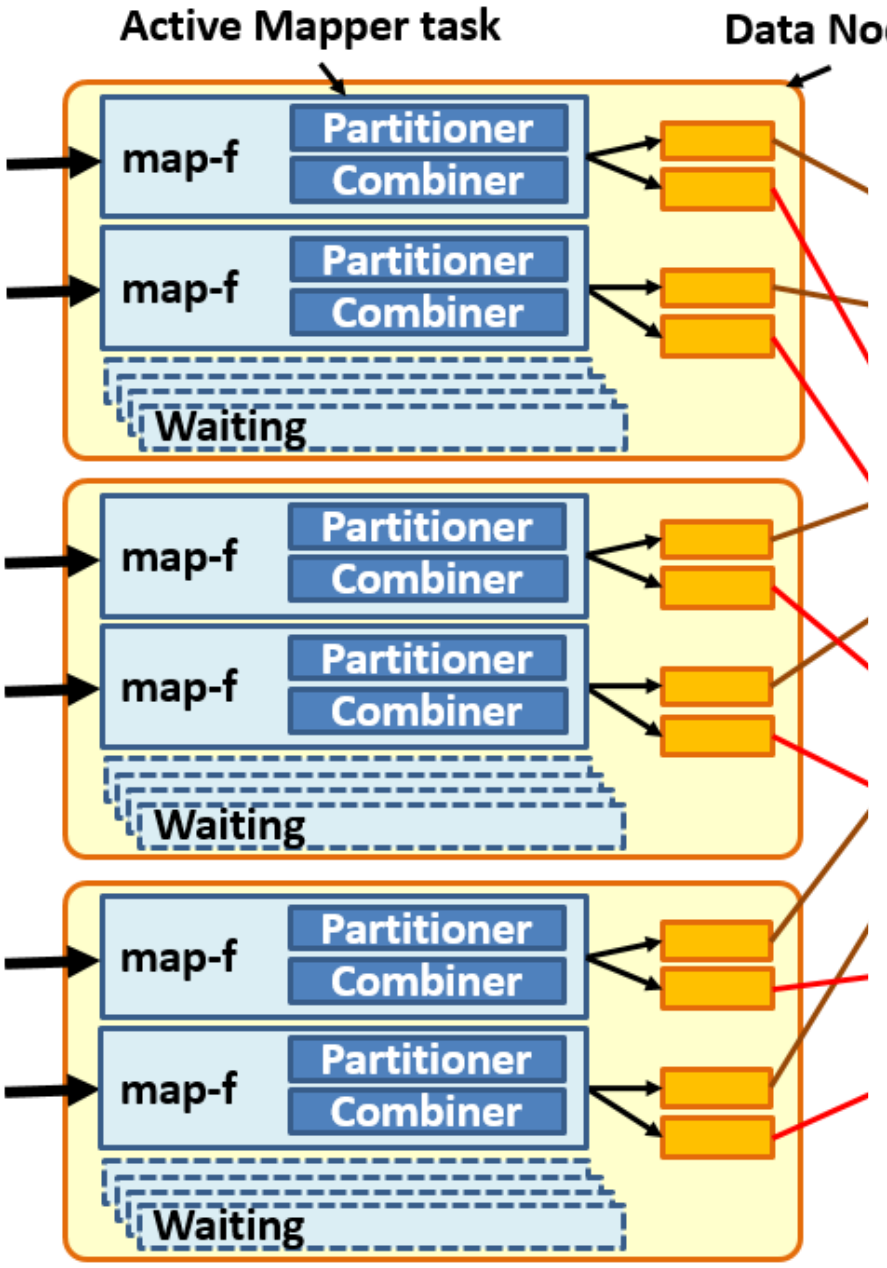


Suite de transformations de paires « clé-valeur »

Chaîne d'opérations Map-Reduce



Déploiement des tâches



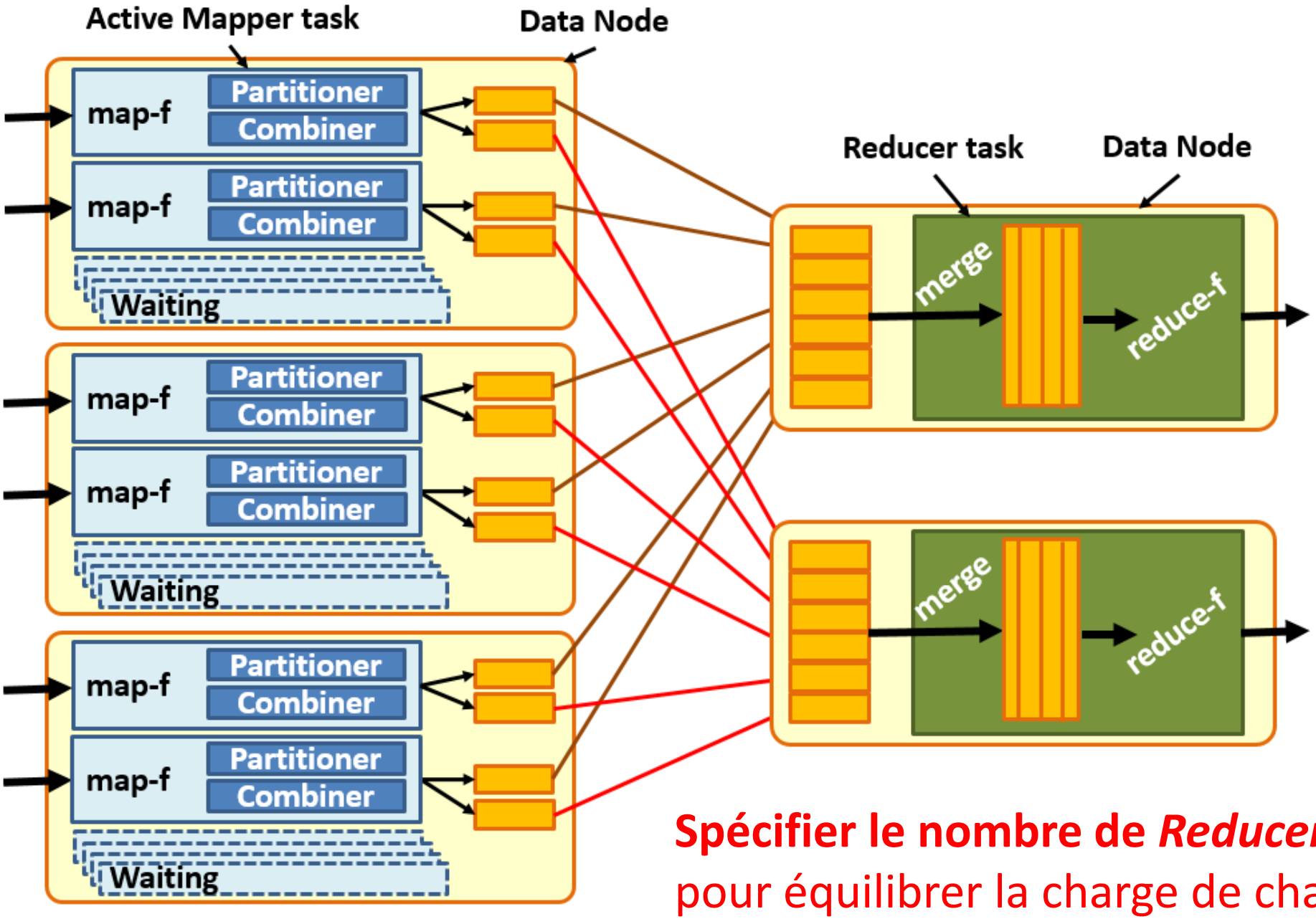
Hadoop installe des *Mappers* sur les nœuds où sont les données accédées (de préférence)

Hadoop crée un *Mapper* par bloc de fichier à lire (bloc classique : tranche de fichier de 64 Mo/128Mo)

Possibilité de limiter le nombre de *Mappers* actifs simultanément sur un PC en fonction de ses caractéristiques matérielles

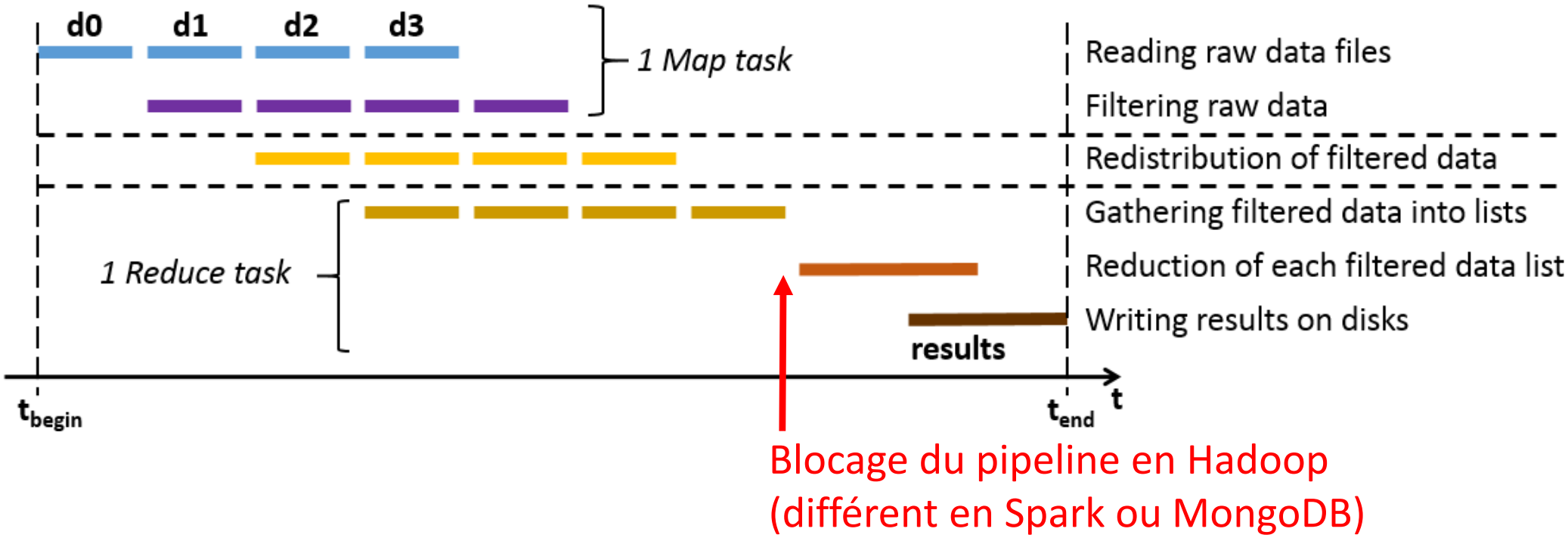
Possibilité de limitations spécifiques à chaque nœud

Déploiement des tâches



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

Pipelining des opérations



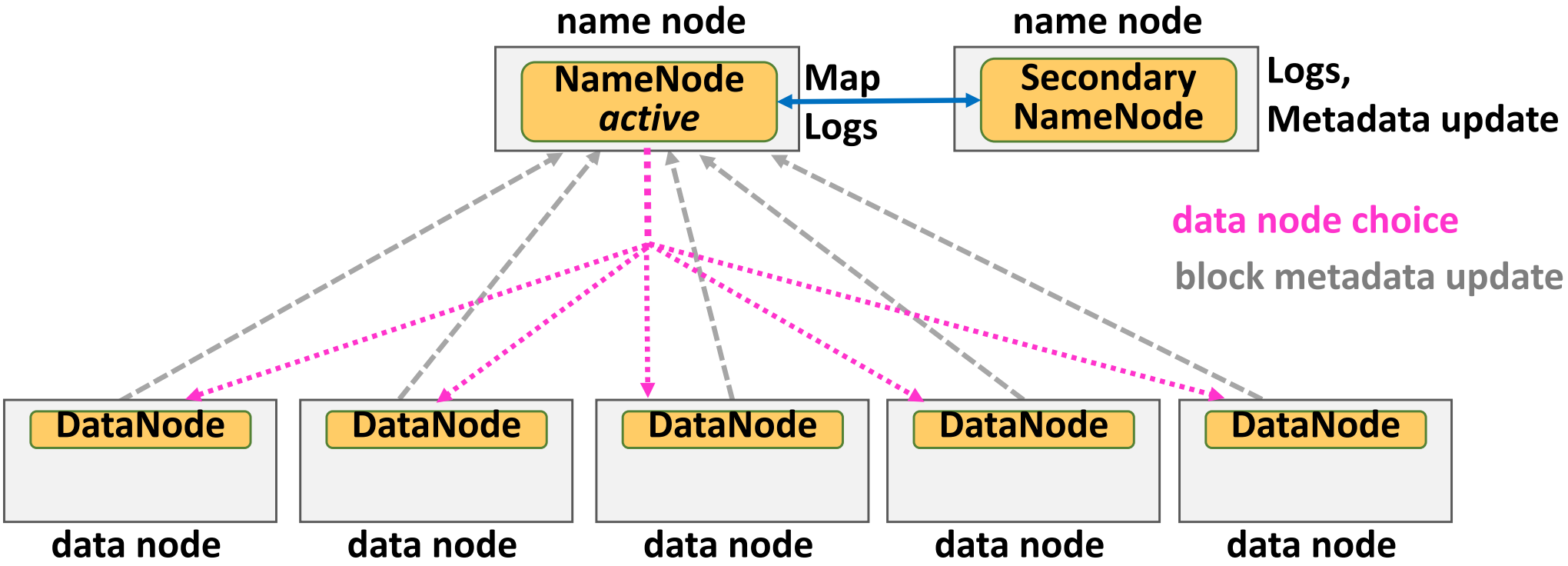
- Pour augmenter le parallélisme, et réduire le temps d'exécution
- Pour tenter de masquer les temps d'IO
  (masquer les lectures et écritures de fichiers temporaires)

POLYTECH PARIS-SUD

Principes et technologie d'Hadoop

1. Localité des données et des traitements
2. Framework d'Hadoop
3. Mécanismes du Map-Reduce d'Hadoop
4. **Système de fichiers distribué d'Hadoop (HDFS)**
   - **Distribution et réplication des fichiers sous HDFS**
   - Lecture de fichiers sous HDFS
   - Ecriture de fichiers sous HDFS
5. Allocation et gestion de ressources d'Hadoop

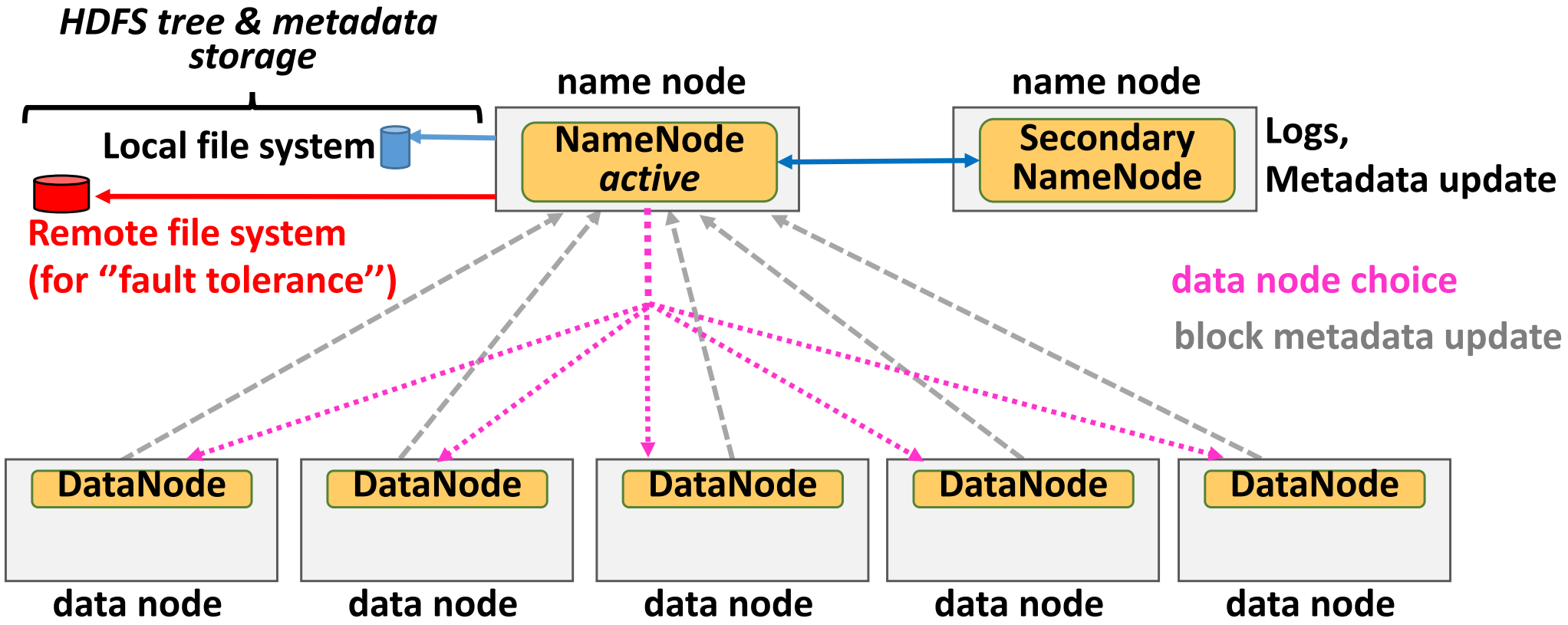
Mécanismes d'HDFS



Le *NameNode* conserve la cartographie du HDFS + les évolutions (les « logs »)
→ Permet de savoir où sont les fichiers

Le *secondary NameNode* stocke les logs et met à jour la cartographie (qd bcp de logs)
→ Sur un nœud à part pour pouvoir calculer la mise à jour sans ralentir le système

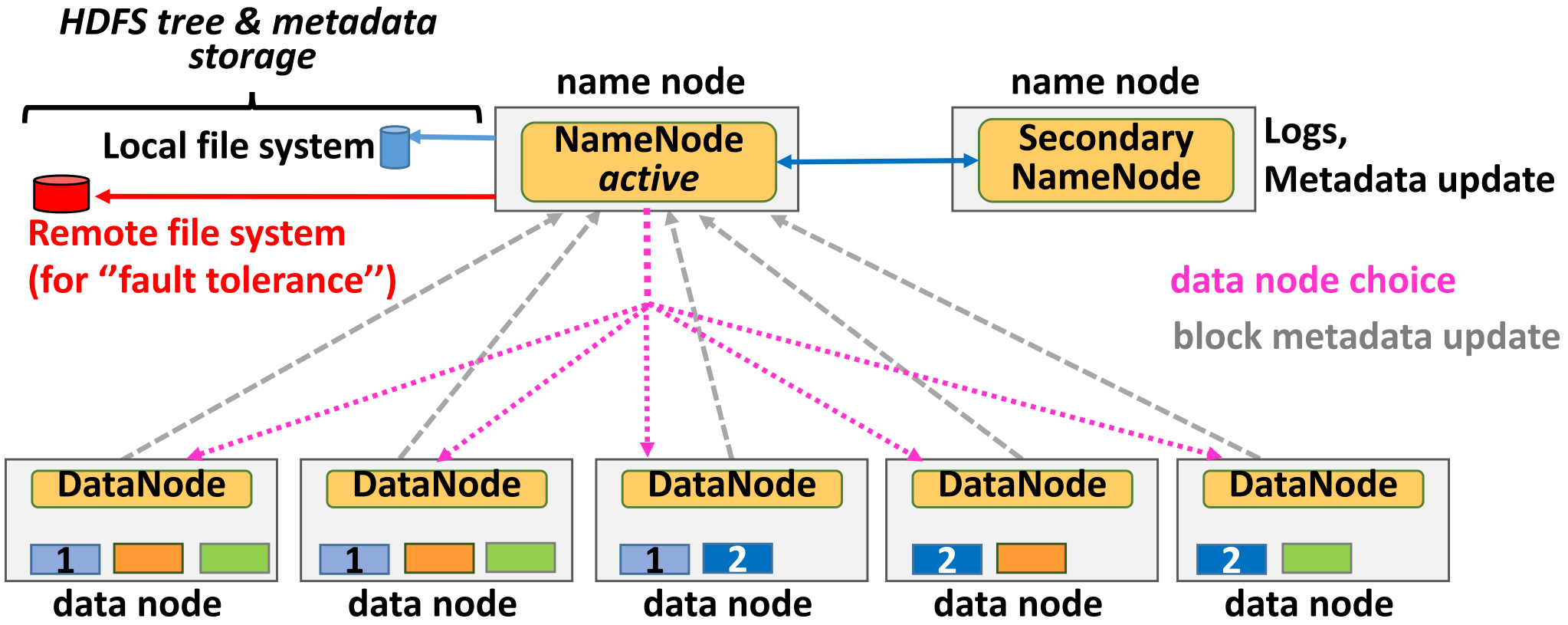
Mécanismes d'HDFS



Les metadonnées du HDFS sont stockés sur le file system classique, localement.

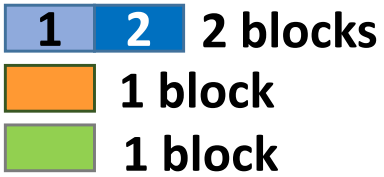
Un stockage distant permet de renforcer la tolérance aux pannes
(sans ses metadonnées, le HDFS est inexploitable)

Mécanismes d'HDFS

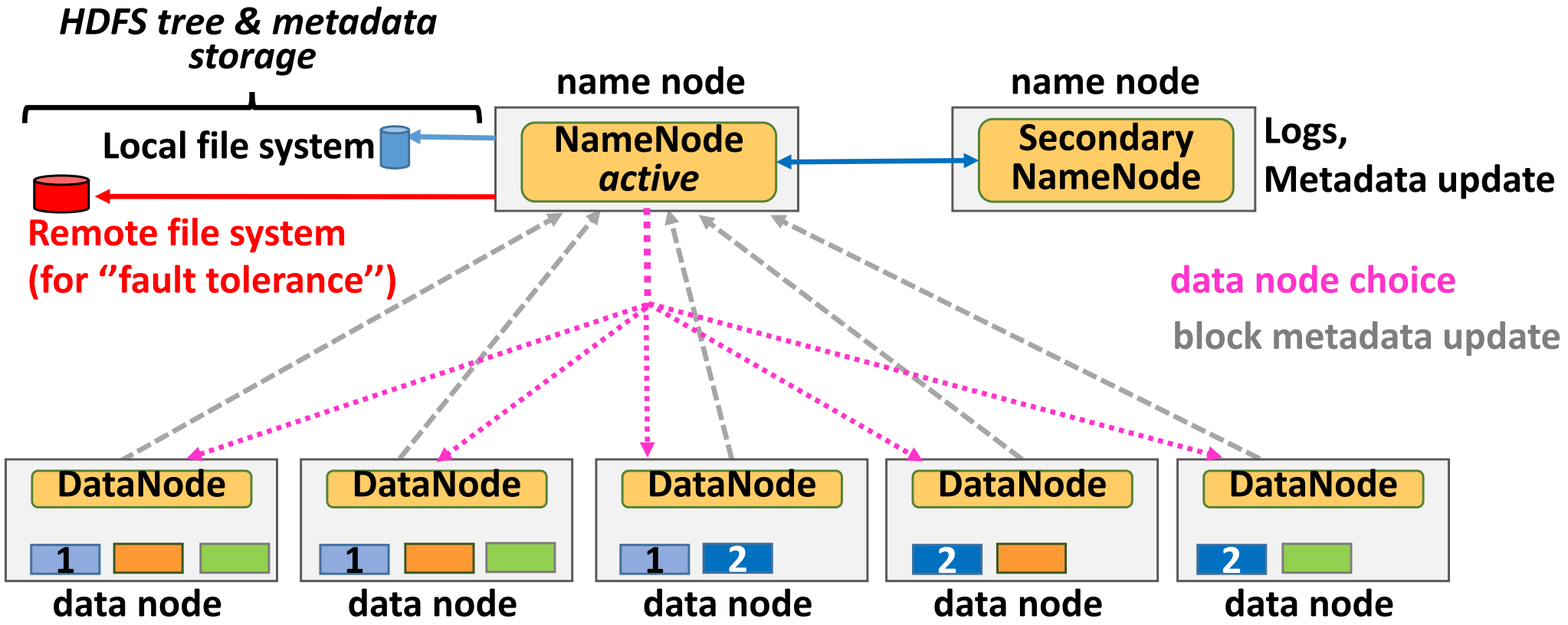


Chaque fichier est découpé en blocs de 64 ou 128 Mo

- **Distribués sur les différents nœuds**
  pour le passage à l'échelle en taille,
  pour la vitesse d'accès
- **Répliqués en n exemplaires (recopiés d'un nœud à un autre)**
  pour la tolérance aux pannes (habituellement n = 3).



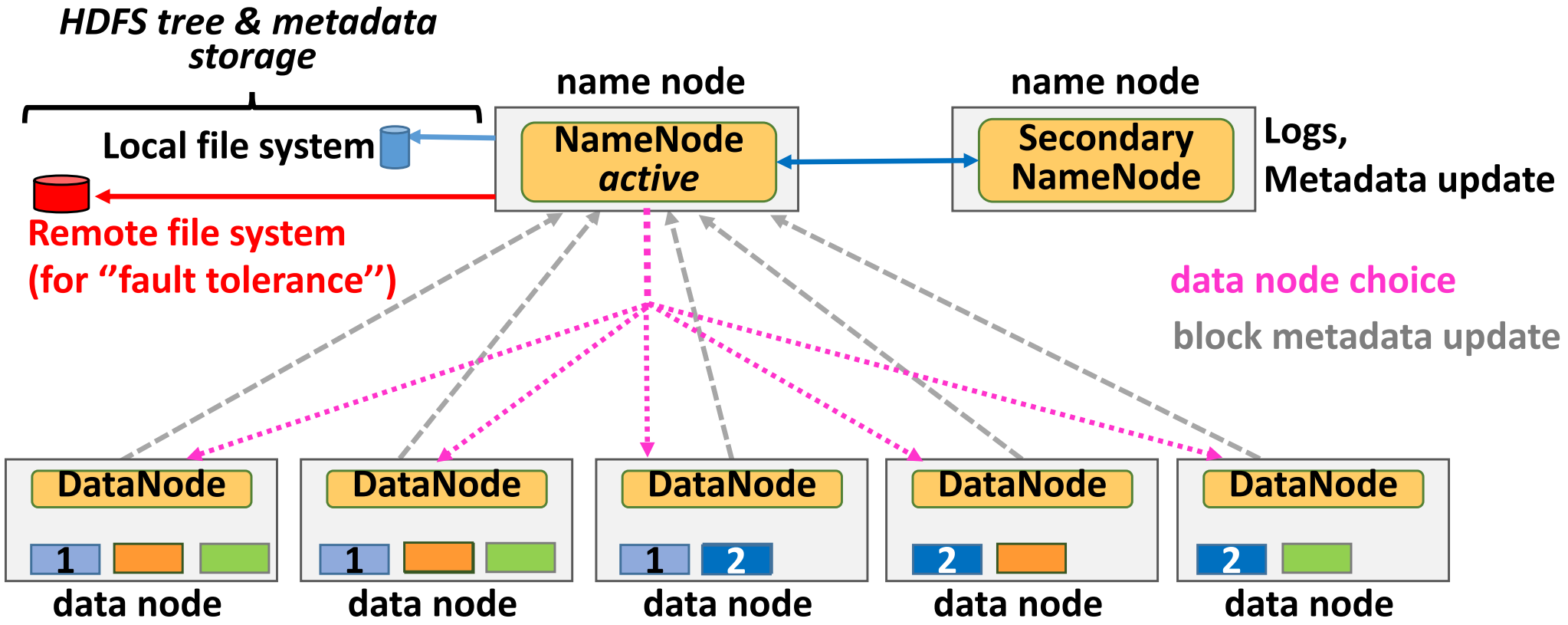
Mécanismes d'HDFS



Règles de choix des nœuds :

- Jamais deux réplicats (du même bloc) sur le même nœud
- Des réplicats dans des « raks » différents
  (grands systèmes d'ensembles de raks)

Mécanismes d'HDFS



Si un nœud tombe :

- Copie et transfert d'autres réplicats
  (reconstitution des n réplicats de chaque bloc)
- Envoie des mise à jour des nœuds vers les NameNode

Mécanismes d'HDFS

Pourquoi des blocs de 64 Mo ?

Temps de « seek » : temps de positionnement au début du fichier sur le disque (disque standard, rotatif)

Tseek = 10ms

Bande passante disque std : Bw = 100 Mo/s

Tread = Q / Bw

On veut : Tseek < 1% Tread

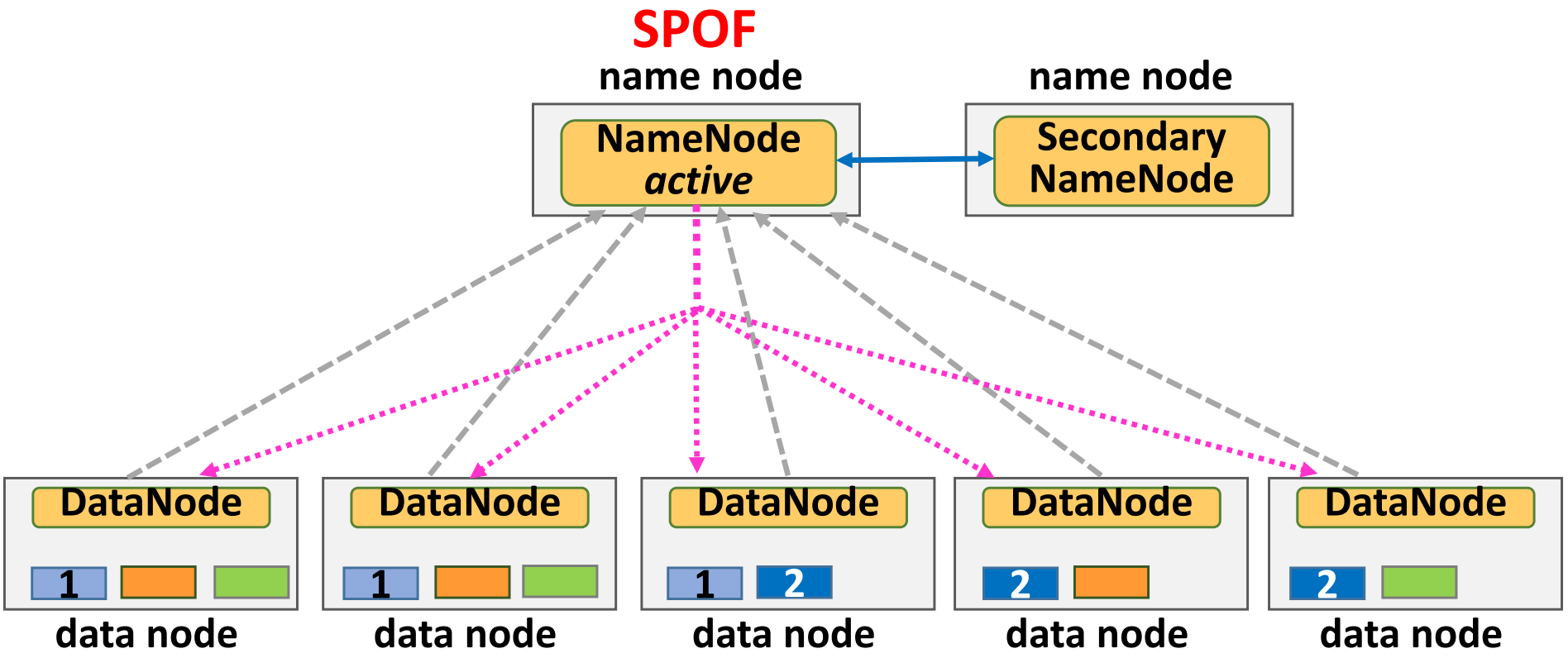
$\Leftrightarrow 10.10^{-3}$ s < (1/100).(Q/100) s

$\Leftrightarrow$ 100 (Mo) < Q

→ Des blocs de 64 Mo ou 128 Mo permettent de masquer les temps de seek

→ Au-delà : pas plus de gain, mais moins de distribution des fichiers, moins de vitesse de lecture

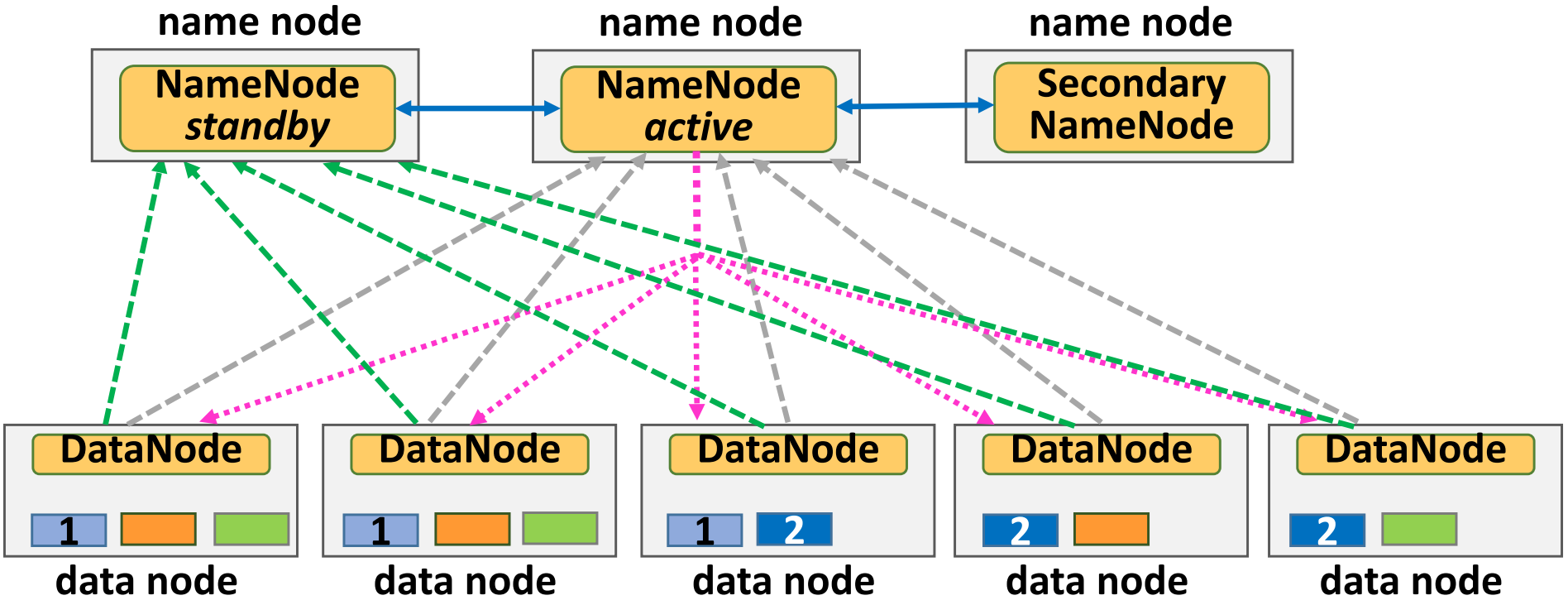
Mécanismes d'HDFS : « haute disponibilité »



Le NameNode est un SPOF : Single Point Of Failure

→ Si on perd le NameNode alors le File System d'Hadoop ne marche plus !

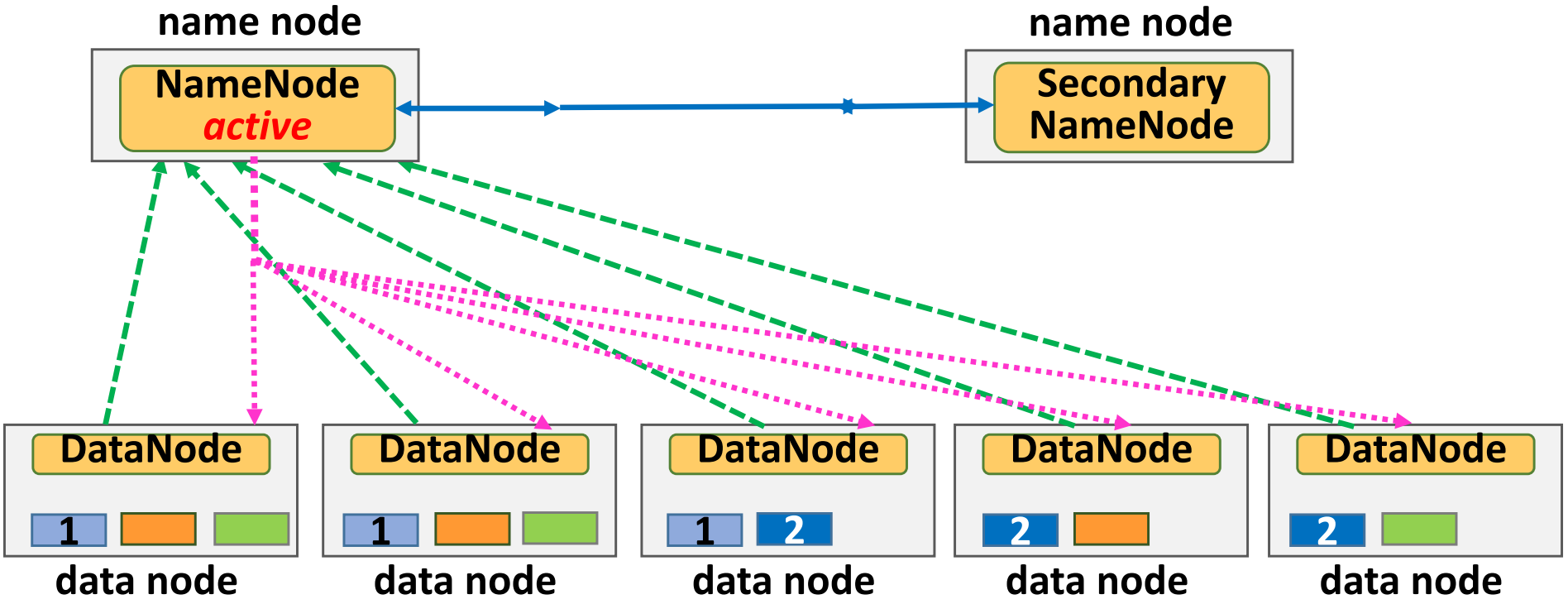
Mécanismes d'HDFS : « haute disponibilité »



Si on perd le NameNode alors le File System d'Hadoop ne marche plus !

→ On duplique le NameNode

Mécanismes d'HDFS : « haute disponibilité »



- Le NameNode en *standy* est passé *actif* « très rapidement »
- Il n'y a plus de SPOF

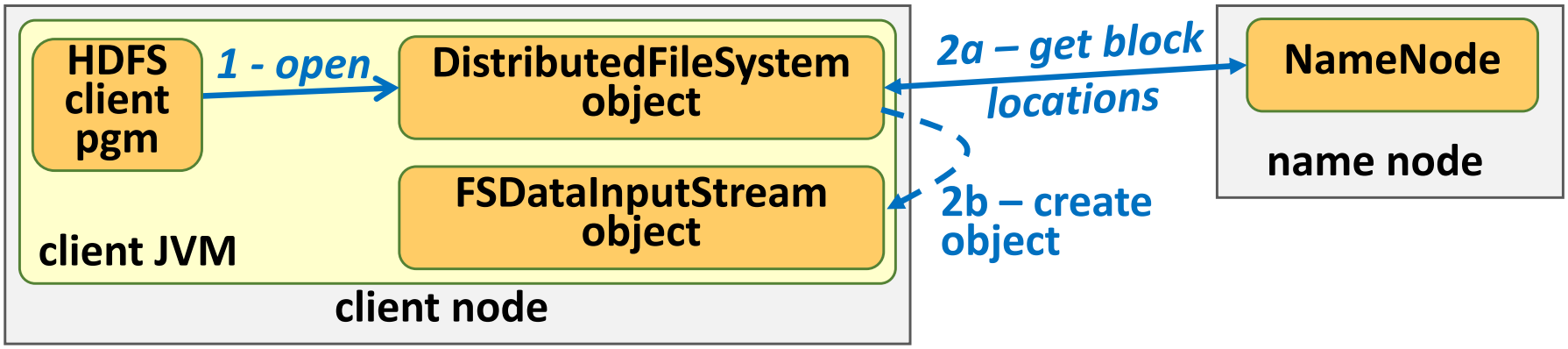
→ **On « ne sent plus passer la panne » : Haute Disponibilité**

POLYTECH PARIS-SUD

Principes et technologie d'Hadoop

1. Localité des données et des traitements
2. Framework d'Hadoop
3. Mécanismes du Map-Reduce d'Hadoop
4. **Système de fichiers distribué d'Hadoop (HDFS)**
   - Distribution et réplication des fichiers sous HDFS
   - **Lecture de fichiers sous HDFS**
   - Ecriture de fichiers sous HDFS
5. Allocation et gestion de ressources d'Hadoop

Mécanismes de lecture d'HDFS



0 – Création d'un objet permettant de s'interfacer à HDFS (un stub/proxy d'HDFS)

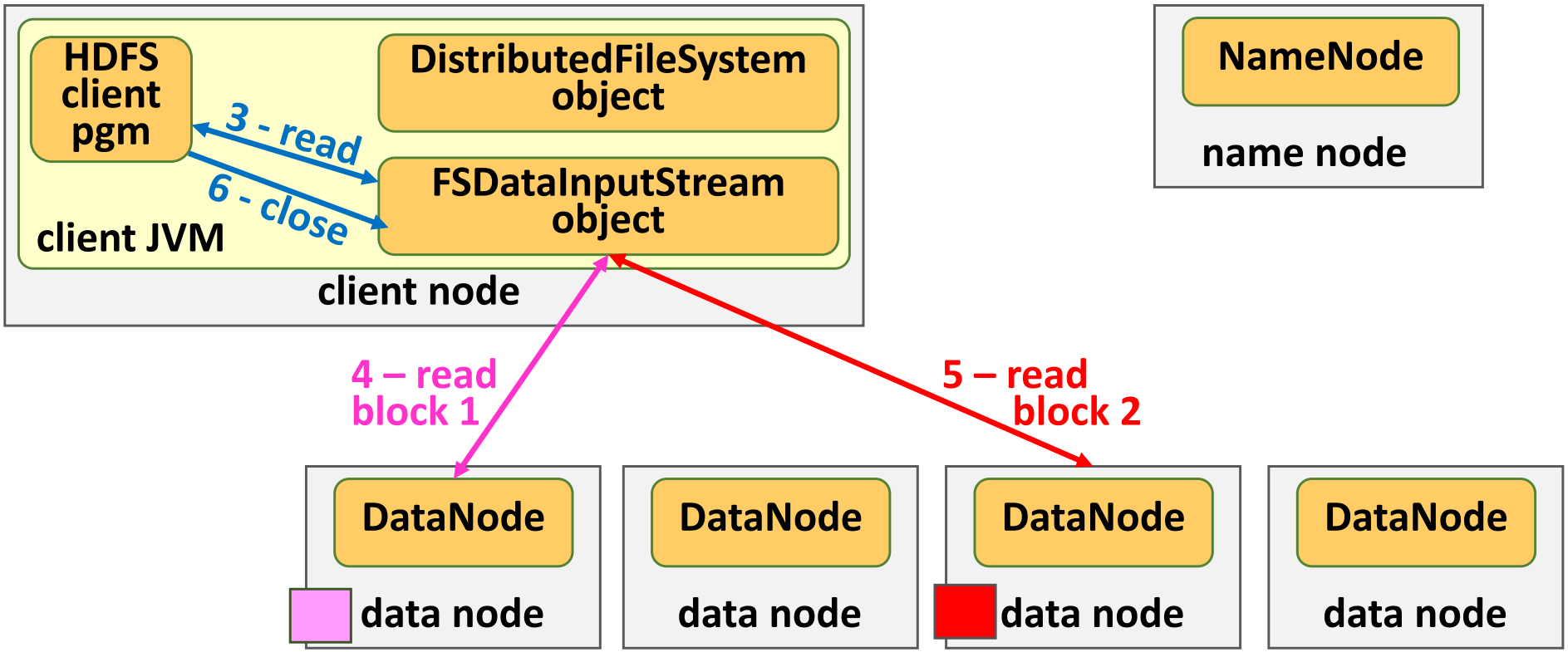
1 – Demande d'ouverture d'un fichier HDFS en lecture (« open »)

2 – Demande de localisation du fichier

2a - Le proxy interroge le NameNode : pour savoir quels blocs lire et où les lire
Le NameNode répond au proxy

2b - Le proxy crée et retourne un objet lecteur spécialisé sur le fichier ciblé

Mécanismes de lecture d'HDFS



3 – Le client exploite le lecteur du fichier

4 – Le lecteur accède au data node du premier bloc et récupère ce bloc

5 – Le lecteur accède au data node du second bloc et récupère ce bloc

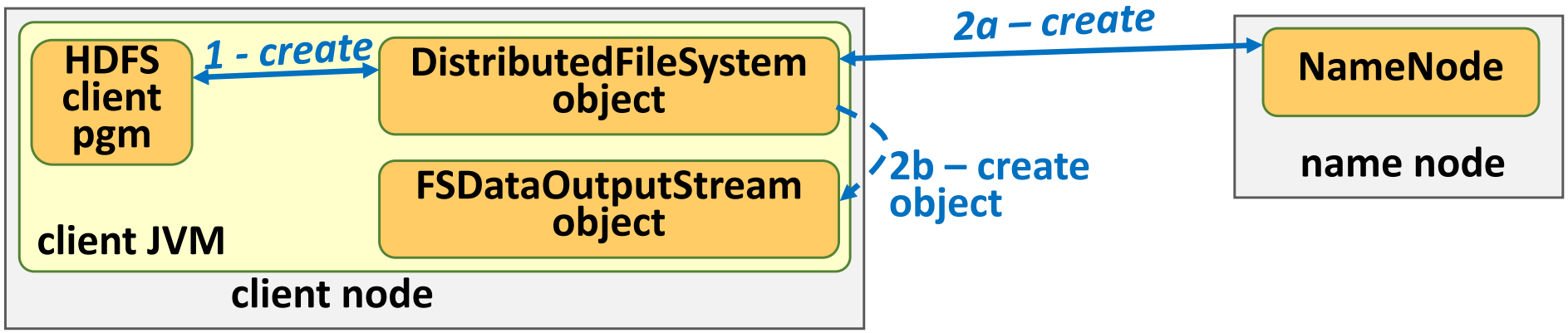
6 – Le client referme le fichier en s'adressant au lecteur du fichier

POLYTECH PARIS-SUD

Principes et technologie d'Hadoop

1. Localité des données et des traitements
2. Framework d'Hadoop
3. Mécanismes du Map-Reduce d'Hadoop
4. **Système de fichiers distribué d'Hadoop (HDFS)**
   - Distribution et réplication des fichiers sous HDFS
   - Lecture de fichiers sous HDFS
   - **Ecriture de fichiers sous HDFS**
5. Allocation et gestion de ressources d'Hadoop

Mécanisme d'écriture d'HDFS



0 – Création d'un objet permettant de s'interfacer à HDFS (un stub/proxy d'HDFS)

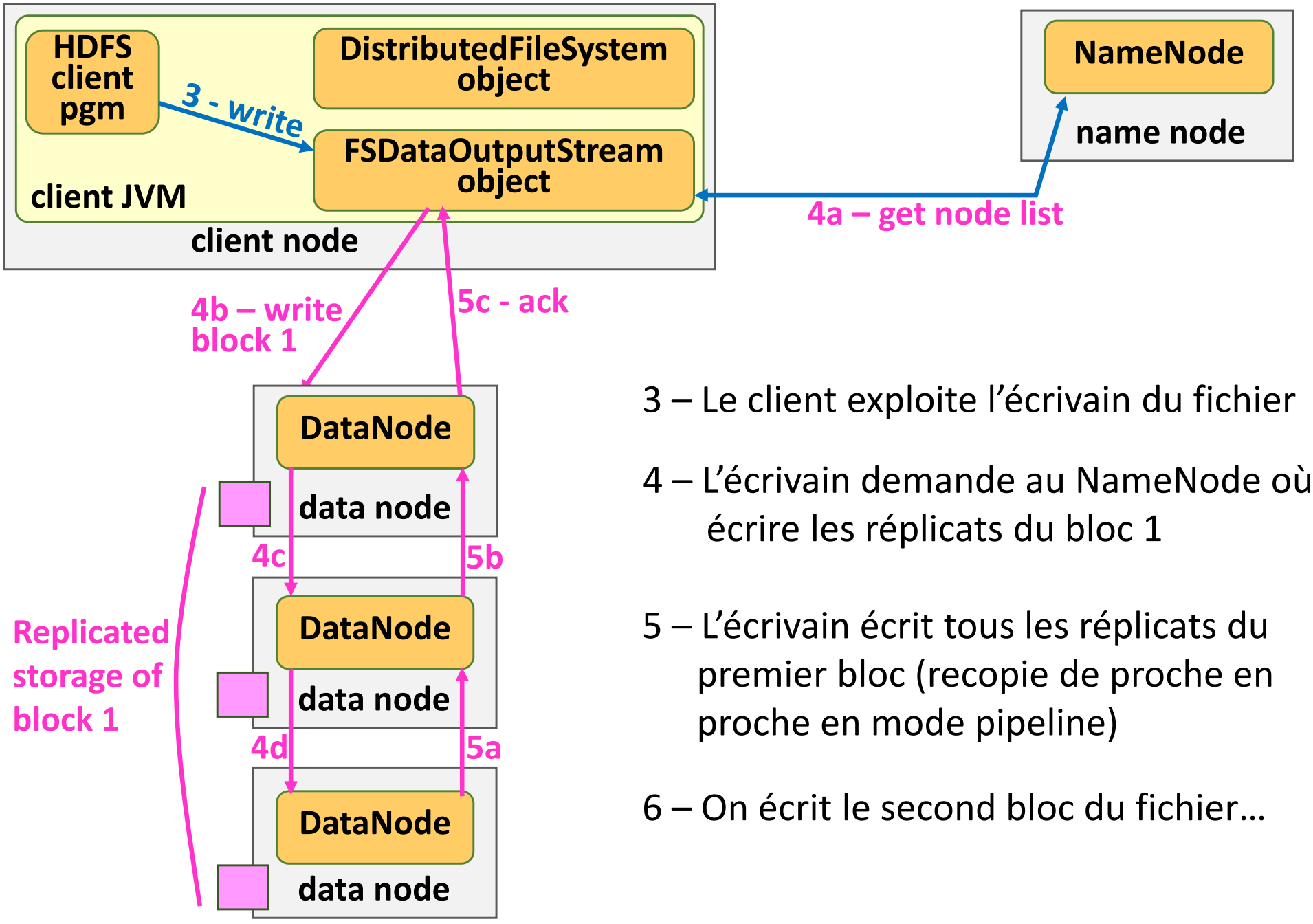
1 – Demande d'ouverture d'un fichier HDFS en écriture (« create »)

2 – Demande de localisation des nœuds d'accueil des futurs blocs du fichier

2a - Le proxy interroge le NameNode : pour savoir où écrire les blocs
Le NameNode répond au proxy

2b - Le proxy crée et retourne un objet écrivain spécialisé sur le fichier ciblé

Mécanisme d'écriture d'HDFS



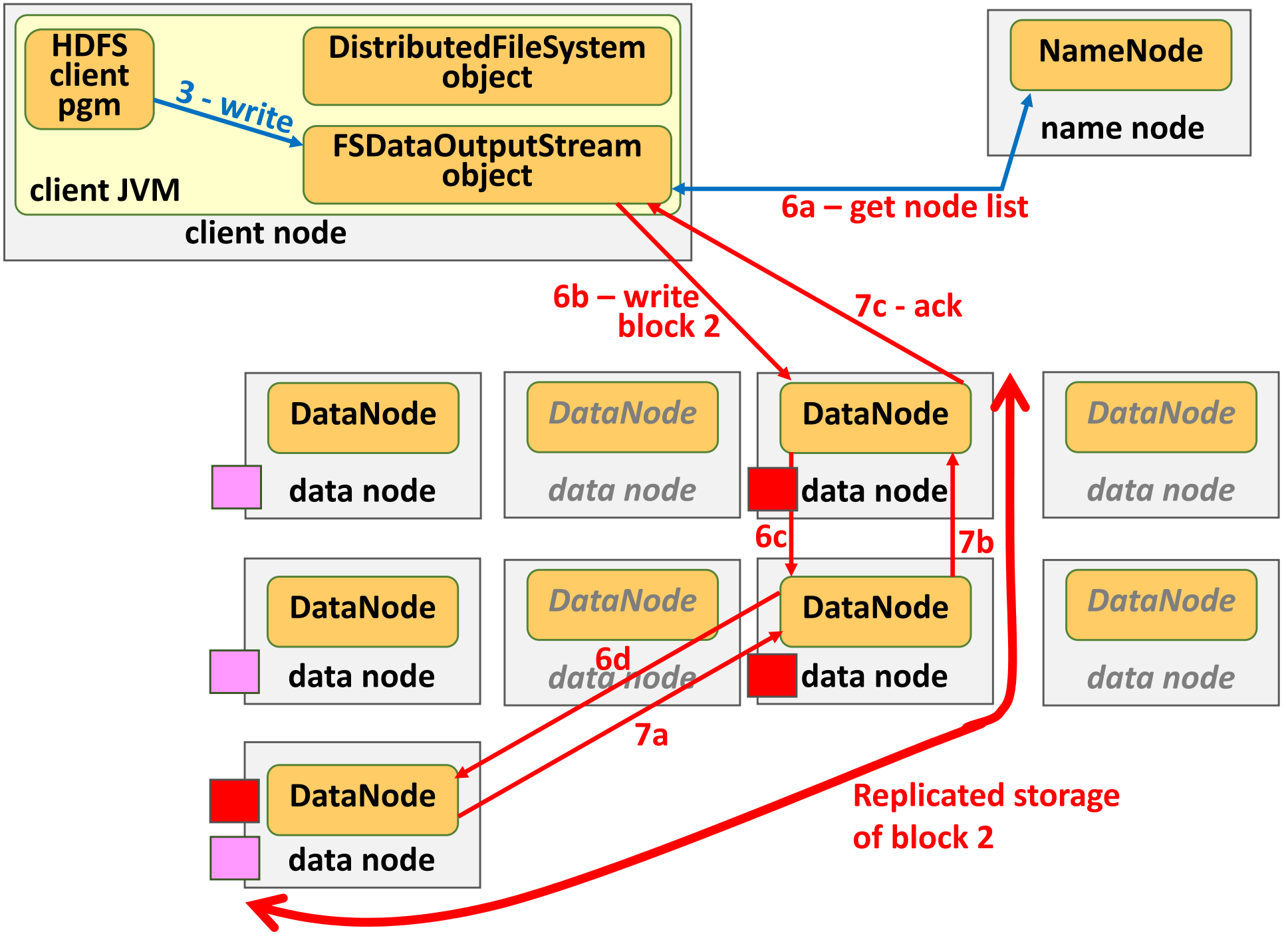
3 – Le client exploite l'écrivain du fichier

4 – L'écrivain demande au NameNode où écrire les réplicats du bloc 1

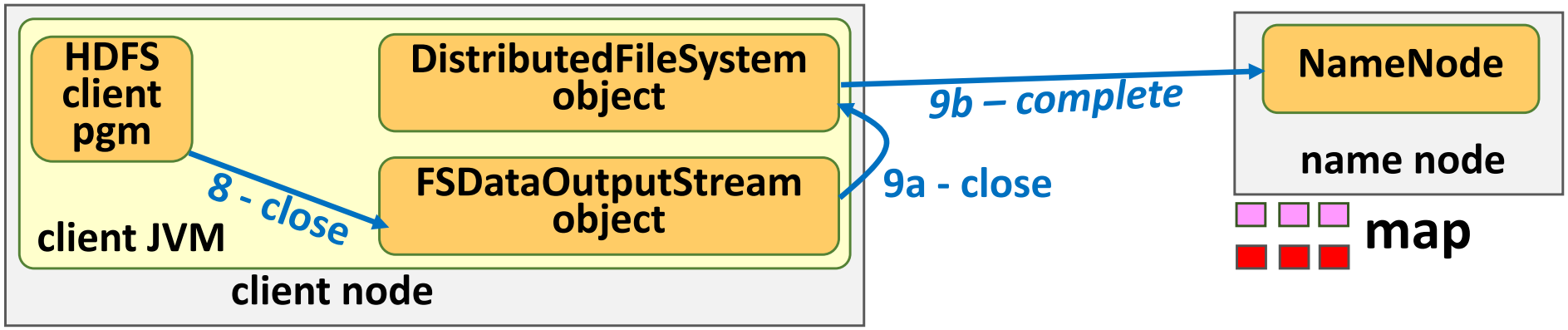
5 – L'écrivain écrit tous les réplicats du premier bloc (recopie de proche en proche en mode pipeline)

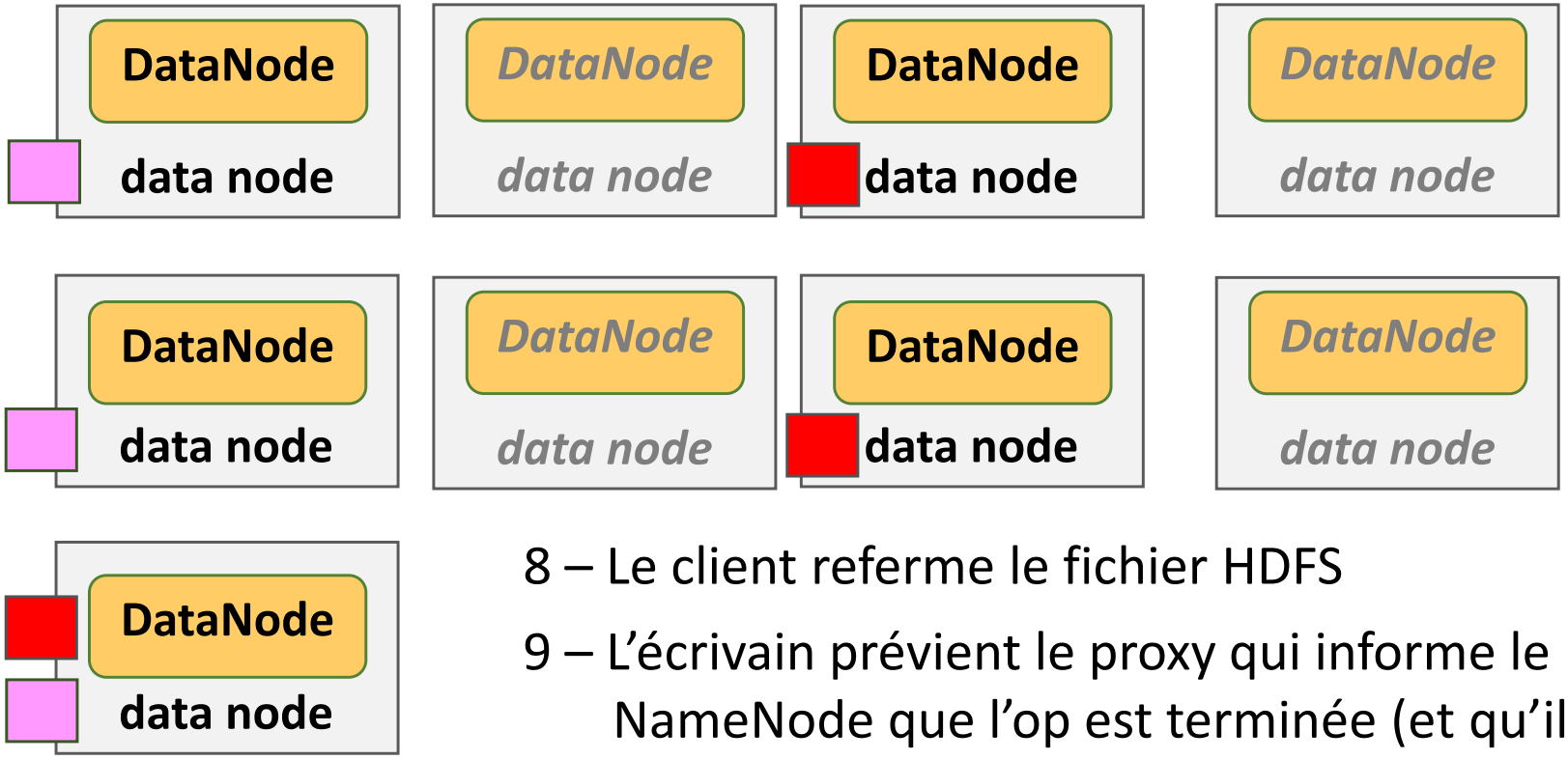
6 – On écrit le second bloc du fichier…

Mécanisme d'écriture d'HDFS



Mécanisme d'écriture d'HDFS





8 – Le client referme le fichier HDFS

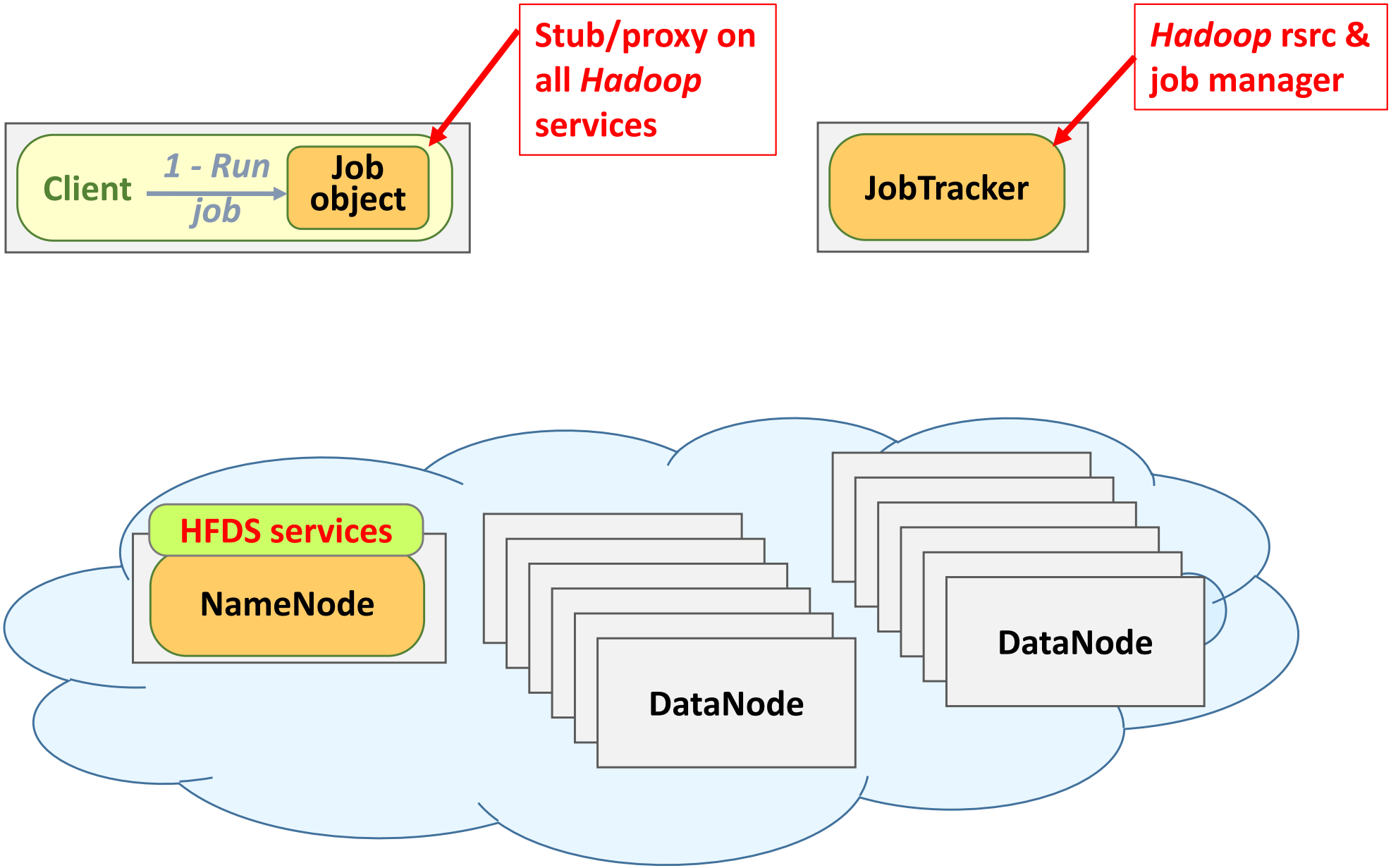
9 – L'écrivain prévient le proxy qui informe le NameNode que l'op est terminée (et qu'il peut rouvrir le fichier pour un autre client)

POLYTECH PARIS-SUD

Principes et technologie d'Hadoop

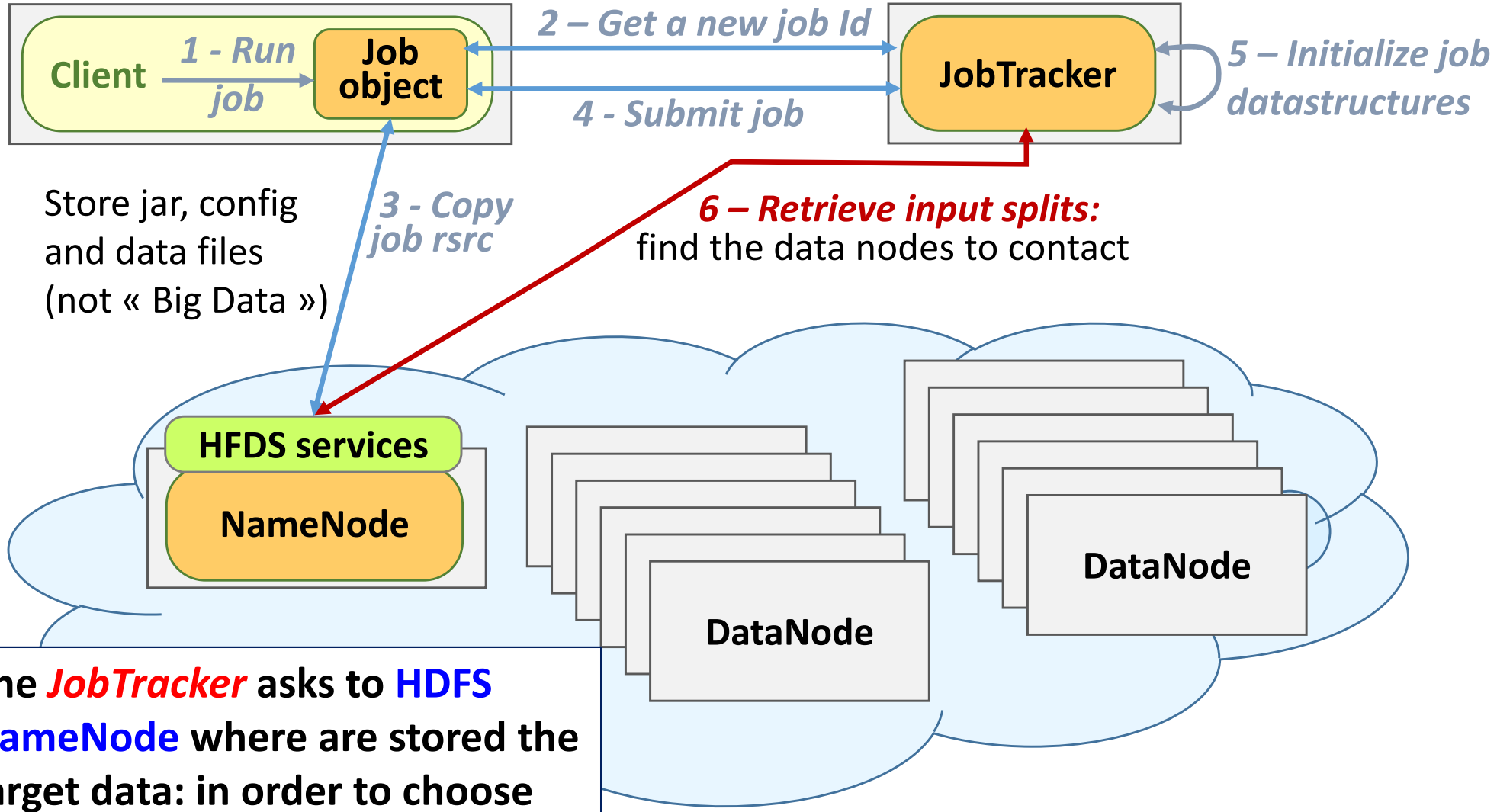
1. Localité des données et des traitements
2. Framework d'Hadoop
3. Mécanismes du Map-Reduce d'Hadoop
4. Système de fichiers distribué d'Hadoop (HDFS)
5. **Allocation et gestion de ressources d'Hadoop**
   - **Hadoop v1**
   - Hadoop v2 (YARN) : passage à l'échelle amélioré

Gestion des ressources v1



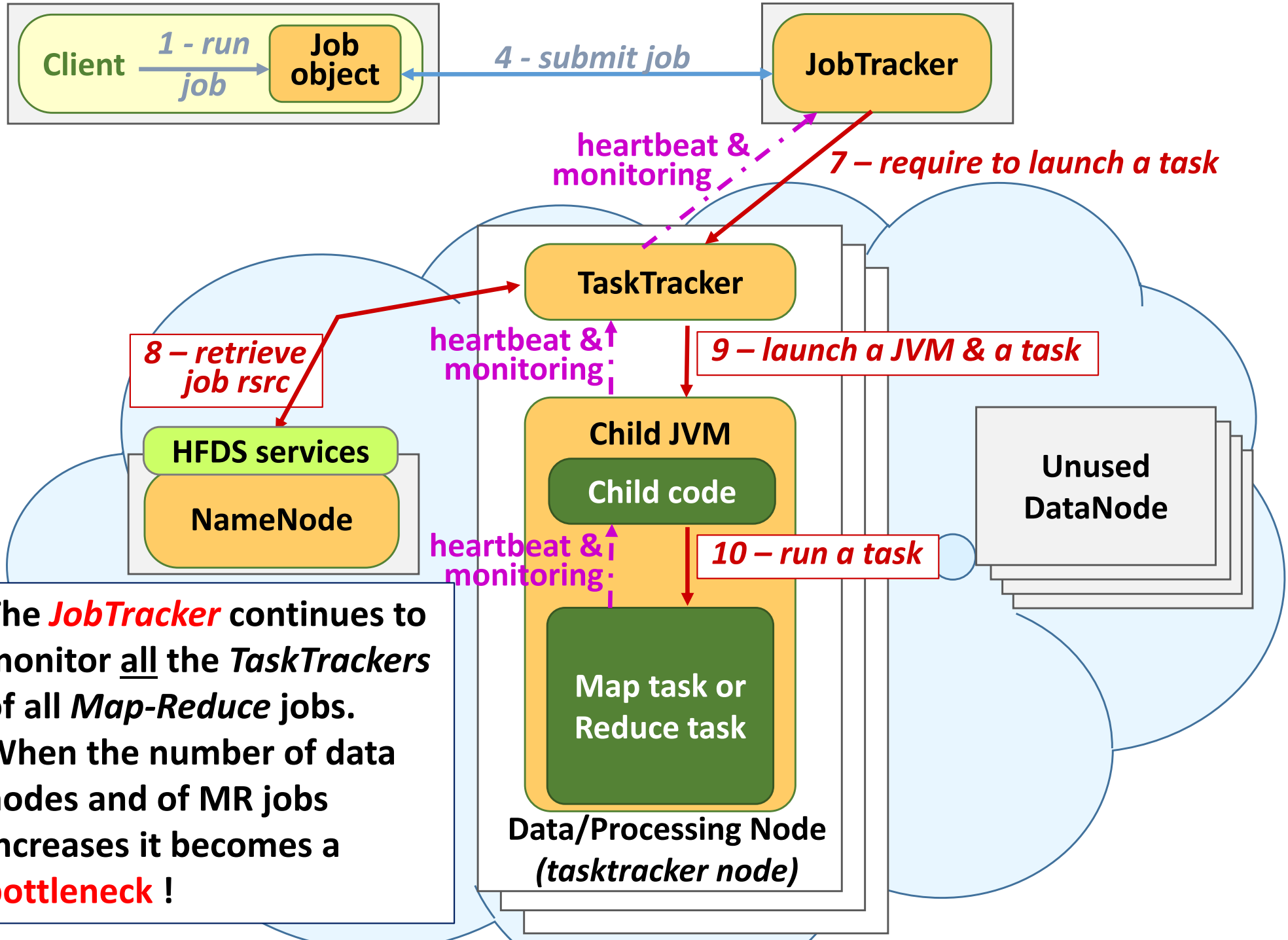
The client creates a proxy local object to ensure all communications with *Hadoop rsrc & job* manager, and with *HDFS* services

Gestion des ressources v1



The *JobTracker* asks to HDFS NameNode where are stored the target data: in order to choose the data nodes that will support Map and Reduce tasks →

Gestion des ressources v1



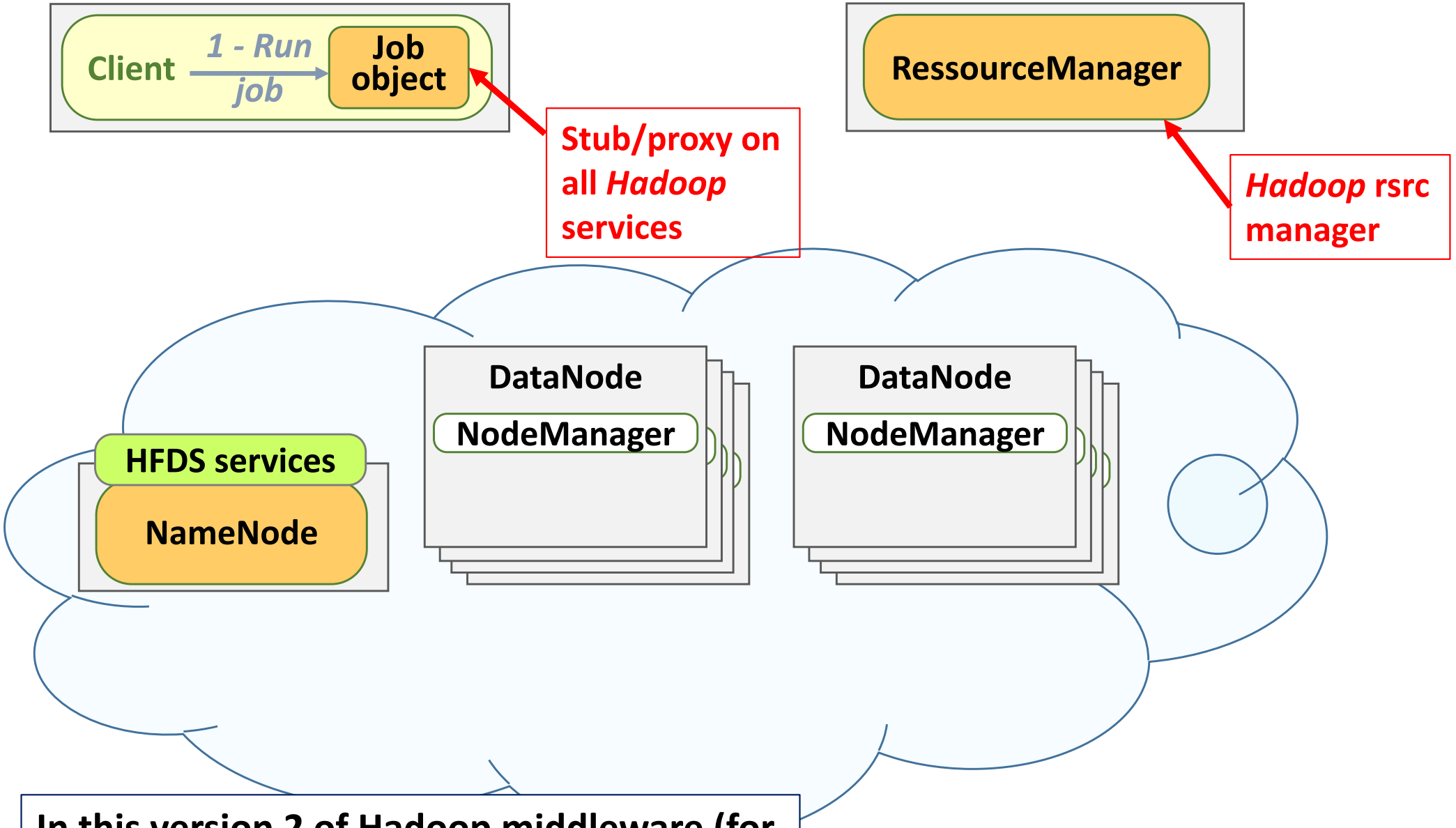
The *JobTracker* continues to monitor <u>all</u> the *TaskTrackers* of all *Map-Reduce* jobs. When the number of data nodes and of MR jobs increases it becomes a bottleneck !

POLYTECH PARIS-SUD

Principes et technologie d'Hadoop

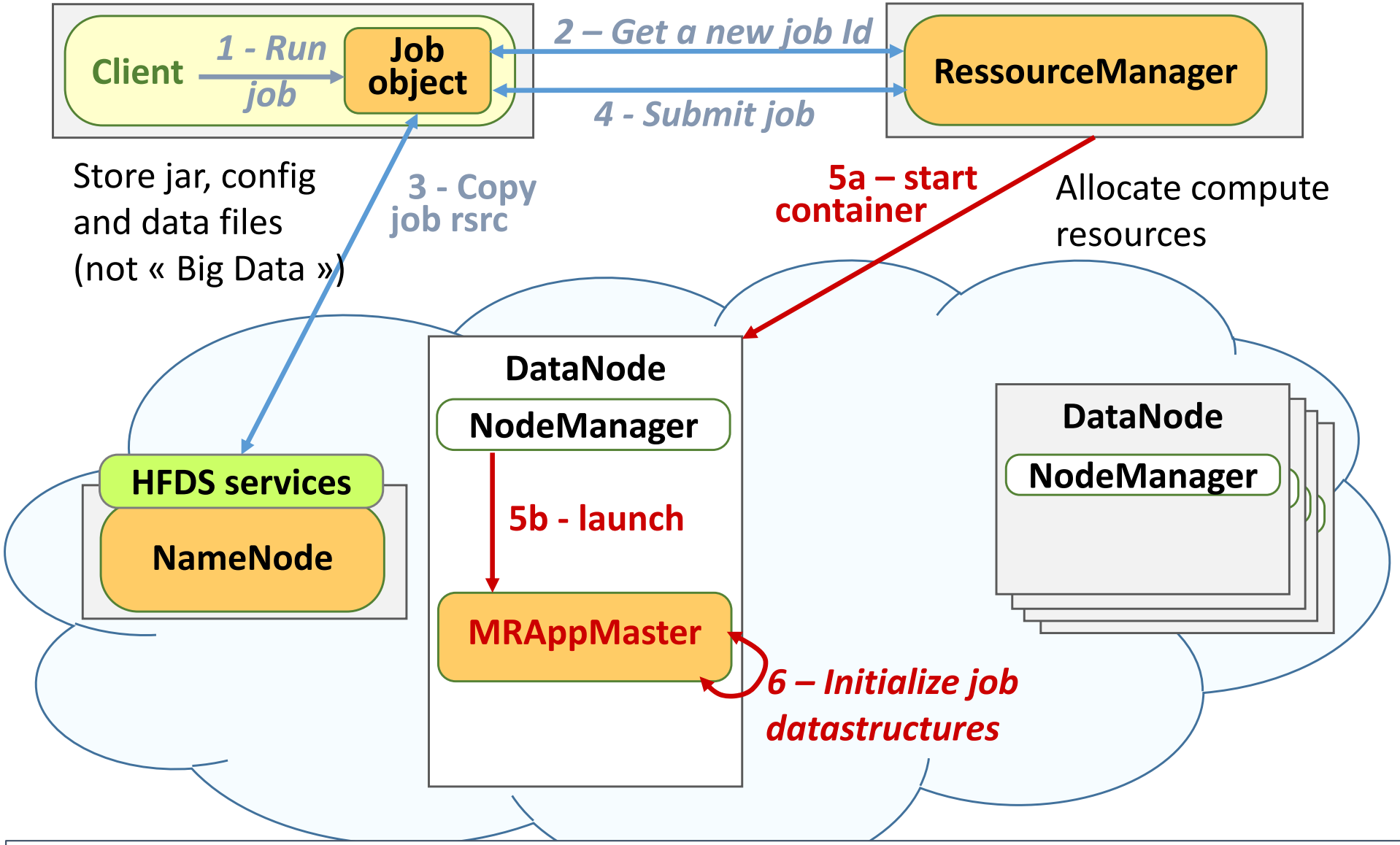
1. Localité des données et des traitements
2. Framework d'Hadoop
3. Mécanismes du Map-Reduce d'Hadoop
4. Système de fichiers distribué d'Hadoop (HDFS)
5. **Allocation et gestion de ressources d'Hadoop**
   - Hadoop v1
   - **Hadoop v2 (YARN) : passage à l'échelle amélioré**

Gestion des ressources v2 (YARN)



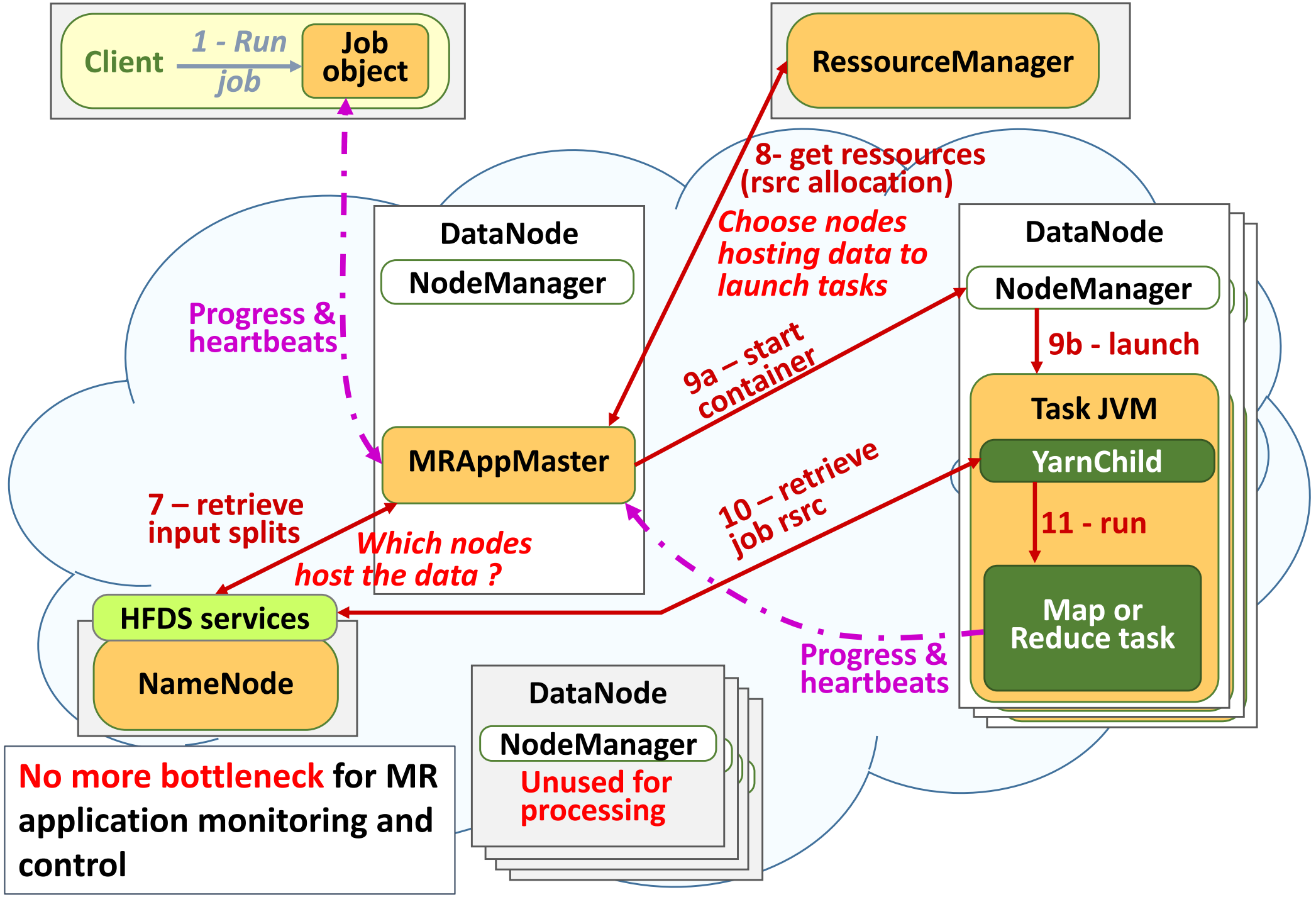
In this version 2 of Hadoop middleware (for Map-Reduce), the RessourceManager is no longer a Job manager (just a rsrc manager)

Gestion des ressources v2 (YARN)



When processing a *Map-Reduce* job submission, the *RessourceManager* chooses a Data Node to launch a *Map-Reduce Application Master* that will manage and monitor the set of MR application tasks.

Gestion des ressources v2 (YARN)



Exécution spéculative

Hadoop lance de nombreuses tâches (map, reduce, …).
Certaines peuvent « planter » ou « ralentir ».

Le TaskTracker (MR v1) ou l'ApplicationManager (MR v2) monitore fortement les exécutions des tâches, et détectent ces « ralentissements ».

Hadoop peut faire une exécution spéculative : il lance de nouvelles instances des tâches « en retard », et dès qu'il obtient des résultats d'une tâche, il tue son doublon.

Mais cette démarche a un coût :

- en charge de calcul (création fréquentes de tâches redondantes)
- en déplacement de données (surtout si on redonde un *reducer*)

On peut débrayer ce comportement (ex : cluster HPC très fiable)

POLYTECH PARIS-SUD

QUIZ

Q1: a Hadoop Map-Reduce program ends up generating a huge number of key-value pairs with (always) the same key

→ Will the HDFS output file be stored in one large block or in several small ones?

→ Will the “reduce” treatment be processed in parallel or sequentially?

POLYTECH PARIS-SUD

QUIZ

Q2: a user connects his client program, running on his laptop, to a 100-node Hadoop cluster, and submits Map-Reduce queries, to compute the histogram of the age of the French (with one-year increments)

→ Technically, can he download the results to his laptop?

→ Technically, can he upload new input data to the HDFS of the 100-node cluster?

→ Technically, can he download the input data to his laptop and then load it into a second Hadoop cluster?

→ Is it possible to copy data from the HDFS of a first Hadoop cluster directly to the HDFS of a second?

POLYTECH PARIS-SUD

QUIZ

Q3: a failure occurs on a Hadoop data node used during the execution of a Map-Reduce program (the node disappears)

→ Does the user have to resubmit the Map-Reduce request?

→ Does the user get the result later when a failure occurs?

Q4: to improve fault tolerance, you can install HDFS on top of a RAID-enabled storage array (*Redundant Array of Independent Disks*)

→ Do you think this is a logical approach?

POLYTECH PARIS-SUD

Technologies Internes d'Hadoop