Spark Streaming

Jonathan Lejeune

Sorbonne Université/LIP6-INRIA

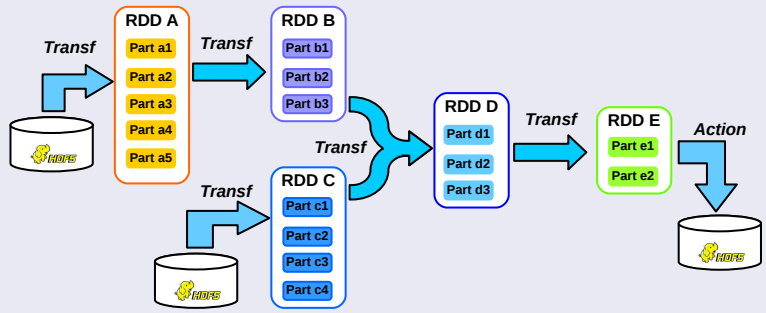
Introduction

Définition du streaming

Diffusion continue de données.

Les traitements Spark Core

- Les traitements se font par lot
- Données d'entrée d'un job sont statiques
  (chargement à partir d'un fichier).



⇒ Inadapté à un changement en continue des données

Problématique

Traiter efficacement un flux de données continu dans Spark

Aperçu Spark Streaming

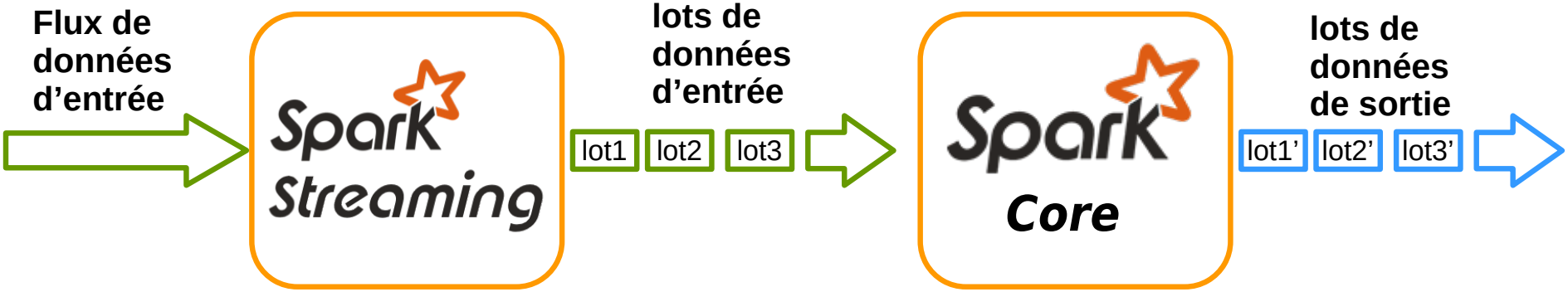
Caractéristiques

- Extension de Spark Core pour les traitements de flux continus
- Tolérant aux pannes : les données reçues ne sont jamais perdues
- Interfaçable avec de multiples sources de données
- API de traitement haut niveau (`map`, `reduce`, `join`, ...)
- Le résultat des traitements peuvent être envoyés vers des fichiers HDFS, des bases de données ou bien des Dashboard

Interaction Spark Streaming et Spark Core

Principe

- Réception continue du flux de données d'entrée depuis une source
- Découpage temporel des données reçues en lots
  $\Rightarrow$ **discrétisation du flux**
- Appel de Spark Core pour traiter chaque lot de données

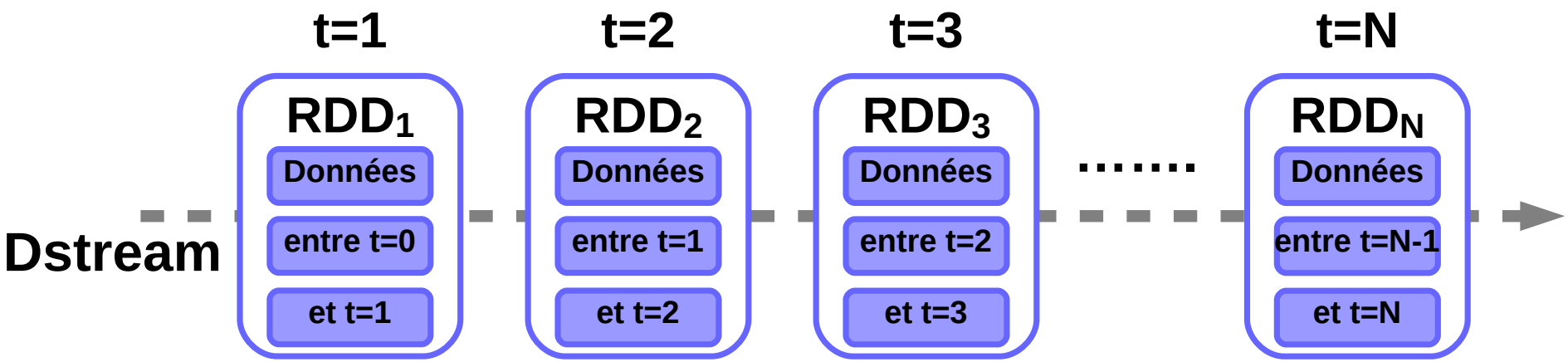


Les DStream

Définition

Un Discretized Stream (DStream) est :

- l'abstraction basique de Spark Streaming
- une représentation d'un flux de données discrétisé par le temps.
- associé à un type de donnée `T`
- une séquence temporelle de `RDD[T]`, où chaque RDD
  - contient les données du flux pour un intervalle de temps
  - représente l'état du flux à un instant donné



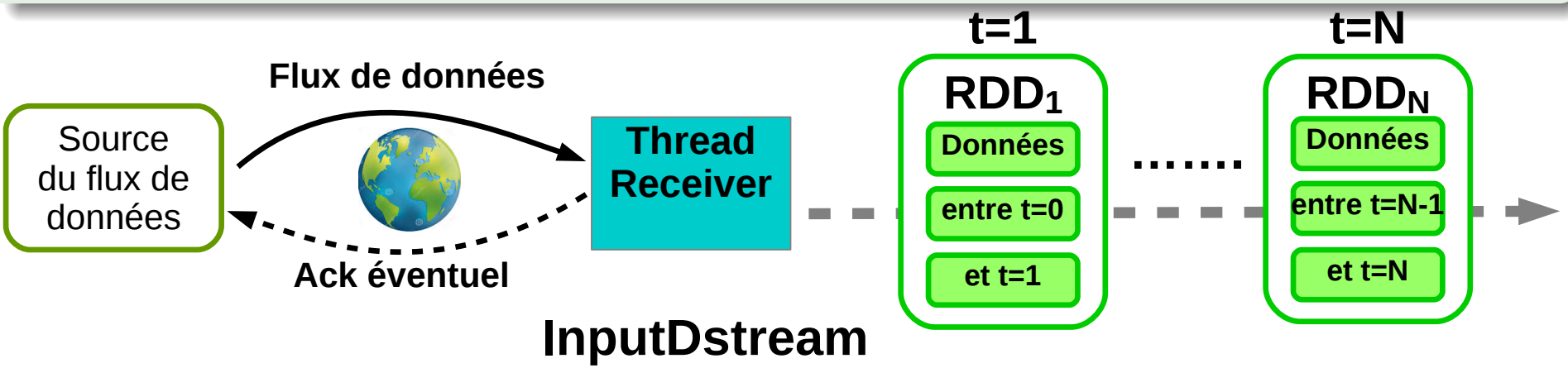
Les DStream d'entrée (InputDStream)

Caractéristiques

Un InputDStream :

- représente un flux des données d'entrée du programme à partir d'une source externe (TCP/Socket, Kafka, HDFS, ...)
- est associé à un Thread `Receiver` :
  - maintient la connexion avec la source
  - reçoit les données de la source et les stocke en mémoire
  - peut renvoyer des acquittements à la sources lorsque les données reçues sont stockées et répliquées

$\Rightarrow$ Chaque Receiver doit avoir son cœur de calcul dédié



Le streaming context

Caractéristiques des `StreamingContext`

Point d'entrée principal pour les fonctionnalités de Spark Streaming

- Trois attributs principaux :
  - un `SparkContext`
  - un dossier HDFS pour les sauvegardes de checkpoint (optionnel)
  - Une période de temps qui définit l'intervalle de discrétisation des flux d'entrée
- Création d'InputDStream
- Démarrage/arrêt du programme (réception + traitement)

```
object MonProgrammeStreaming extends App {
 val conf = new SparkConf().setAppName("Mon programme Streaming");
  val ssc = new StreamingContext(new SparkContext(conf), Seconds(5));

  //création, transformation de DStream

 ssc.start();//lancement des Receivers
 ssc.awaitTermination();// attente d'une exception ou d'un ssc.stop()
}
```

Le streaming context

Contraintes

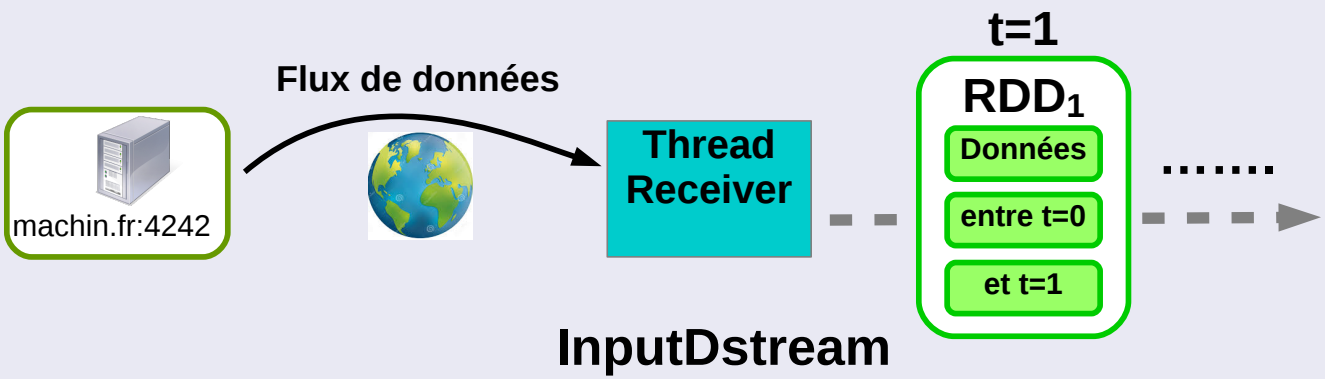
- Une fois que le Streaming Context a été démarré, il n'est plus possible de modifier le programme
- Un Streaming Context stoppé ne peut être redémarré
- Un seul Streaming Context peut être actif à un instant donné dans une JVM
- Un `stop()` sur un Streaming Context arrête également le Spark Context sous-jacent
- Un Spark Context peut être réutilisé pour d'autres Streaming Context

Création d'InputDStream à partir du Streaming Context

Depuis une socket de connexion TCP vers un serveur

```
def socketTextStream(host: String, port: Int,
        storageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK_SER_2
        ): ReceiverInputDStream[String]

def socketStream[T](hostname: String, port: Int,
    converter: (InputStream) => Iterator[T],
    storageLevel: StorageLevel
  ): ReceiverInputDStream[T]
```



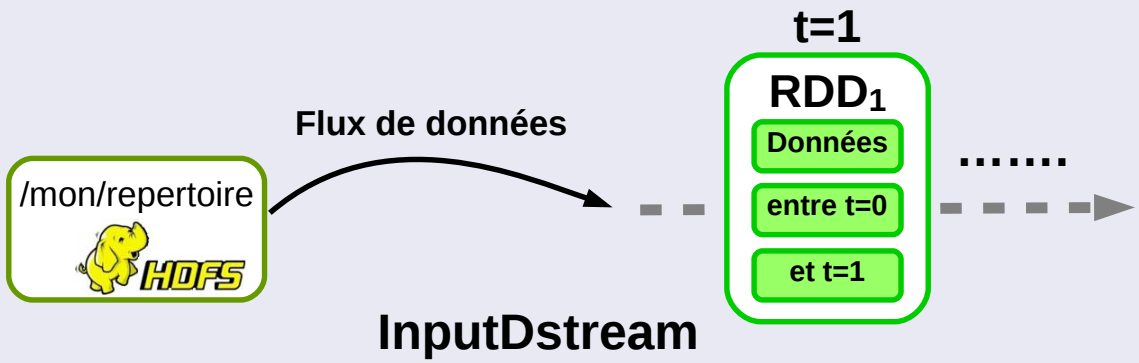
Création d'InputDStream à partir du Streaming Context

Depuis un dossier HDFS

Tous les fichiers doivent avoir le même format

```
def textFileStream(dir: String): DStream[String]
```

```
def fileStream[K,V,F<:NewInputFormat[K, V]](dir:String):InputDStream[(K, V)]
```



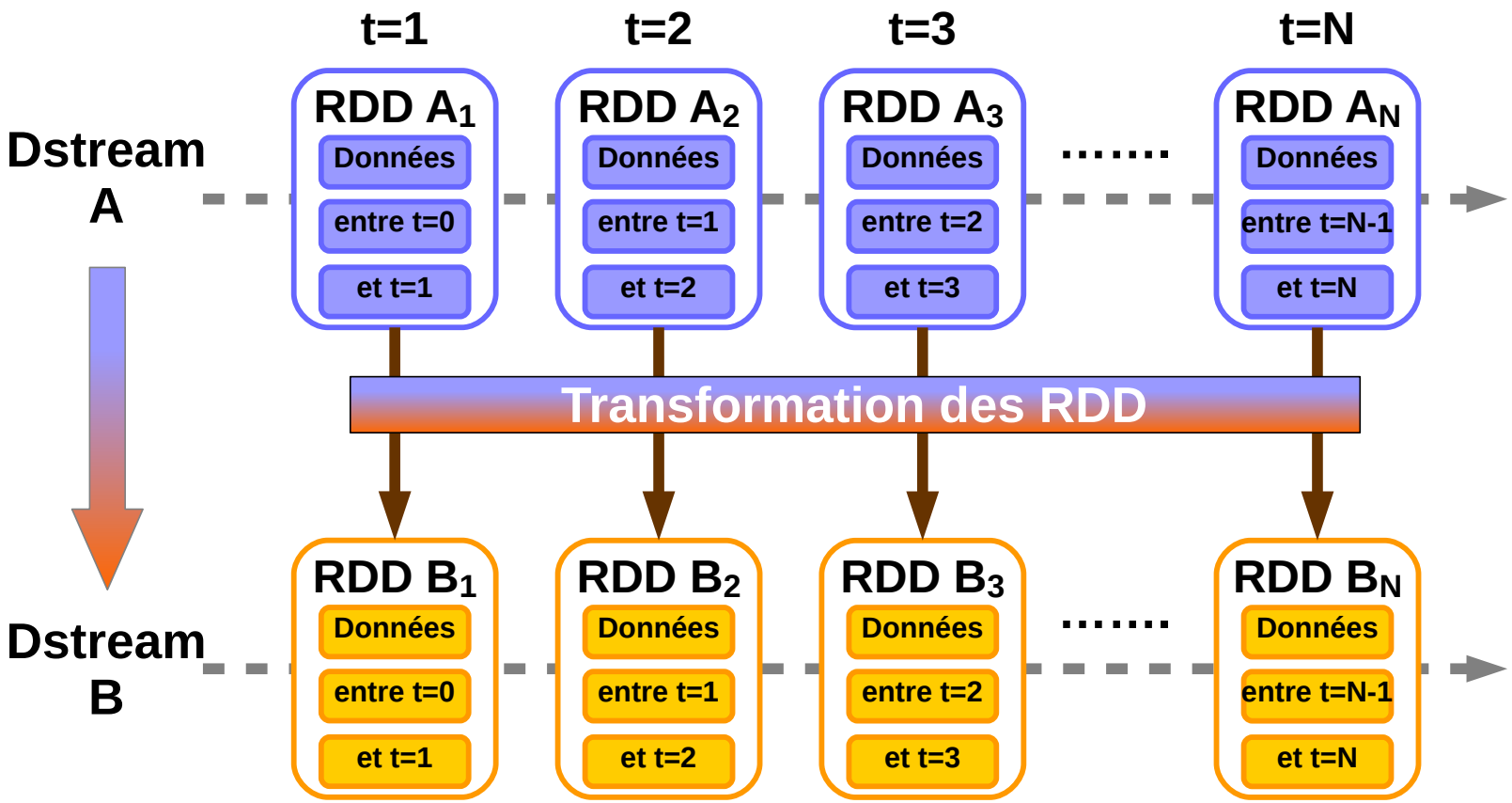
Depuis des sources plus avancées

- **Flume** : Service distribué de collecte, d'agrégation de grandes quantités de logs
- **Kafka** : Système producteur/consommateur de messages en temps réel

Opérations de transformation de DStream

Principe

Une transformation d'un Dstream A vers un Dstream B revient à transformer pour chaque instant t un RDD A vers un RDD B



Opérations de transformation de DStream

Pour n'importe quel type T

- Transformations issues de Spark Core

```
mapPartitions    filter          flatmap     glom
repartition      union           map
```

- Transformations propres à Spark Streaming

```
count       countByValue     countByValueAndWindow   reduce
transform   countByWindow    reduceByWindow
window
```

Pour T étant un couple (K,V)

- Transformations issues de Spark Core

```
fullOuterJoin   rightOuterJoin   leftOuterJoin   join
cogroup         combineByKey     flatMapValues   groupByKey
mapValues       reduceByKey
```

- Transformations propres à Spark Streaming

```
groupByKeyAndWindow reduceByKeyAndWindow updateStateByKey
```

Opération de sorties

Caractéristiques

- Permet l'envoi des données du DStream sur des systèmes externes
- Obligatoire d'en avoir au moins une dans le programme

Liste des opérations

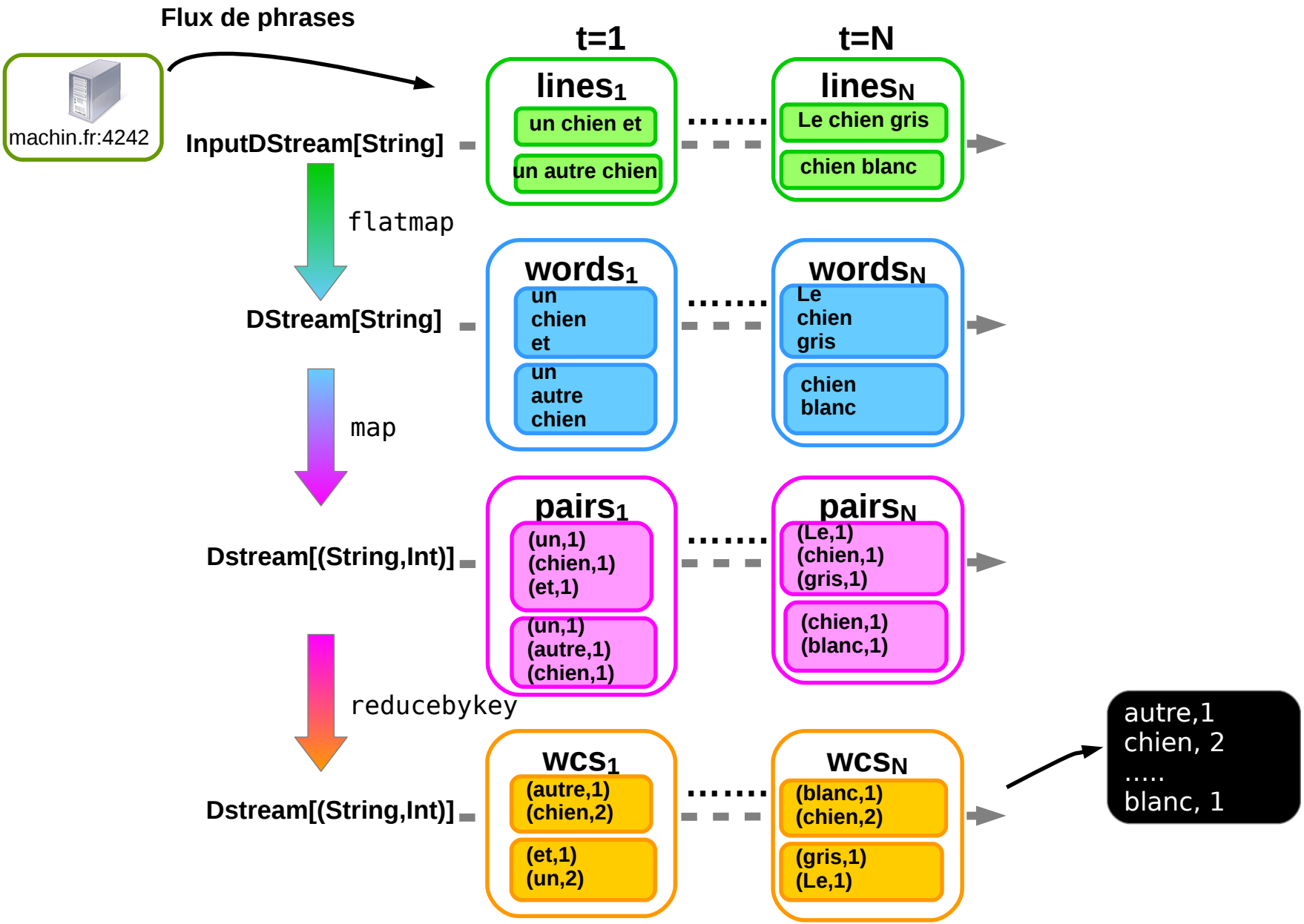
- Écriture d'un fichier pour chaque RDD du flux :
  - `saveAsObjectFiles(prefix, suffix)`
  - `saveAsTextFiles(prefix, suffix)`
  - `saveAsHadoopFiles(prefix, suffix)`

  $\Rightarrow$ Format des noms de fichiers = `prefix-TIME_IN_MS.suffix`
- Affichage des RDD sur la sortie standard du driver
  - `print()` : affiche les 10 premiers éléments de chaque RDD
  - `print(nb)` : affiche les `nb` premiers éléments de chaque RDD
- Opération Générique
  - `def foreachRDD(foreachFunc: RDD[T] =>Unit): Unit`

Exemple wordCound Streaming

```
val sc  = new SparkContext(conf)
val ssc = new StreamingContext(sc, Seconds(1))
val lines = ssc.socketTextStream("machin.fr", 4242)
val words = lines.flatMap(_.split("␣"))
val pairs = words.map(word => (word, 1))
val wcs = pairs.reduceByKey(_ + _)
wcs.print()
ssc.start()
ssc.awaitTermination()
```

Exemple wordCound Streaming



La transformation `Transform`

Principe

```
def transform[U](transformFunc: RDD[T] => RDD[U]): DStream[U]
```

- Permet d'appliquer au flux une fonction de transformation arbitraire de RDD
- Utilisable si la transformation voulue n'existe pas dans l'API des transformation des DStream

Transformation à états

Définition

Transformation qui dépend de RDD calculés dans le passé.

Exemples

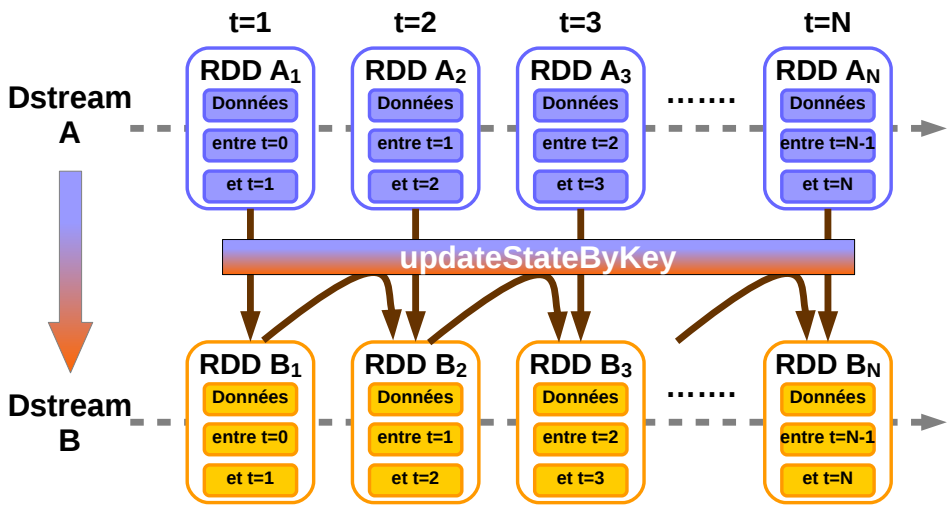
`updateStateByKey`	`window`	`countByWindow`
`countByValueAndWindow`	`reduceByWindow`	
`reduceByKeyAndWindow`	`groupByKeyAndWindow`	

Transformations à états : updateStateByKey

Pour un flux (K,V), maintenir pour chaque clé un état S

```
def updateStateByKey[S](update: (Seq[V], Option[S]) => Option[S]): DStream[(K, S)]
```

- S : le type de l'état
- update définit pour chaque clé la transition de son état de $t-1$ à t
  - Seq[V] : les nouvelles valeurs associées à la clé à l'instant t
  - Option[S] : l'état de la clé K à l'instant $t-1$
- En sortie : le nouvel état associée à la clé K. (None $\Rightarrow$ clé supprimée)

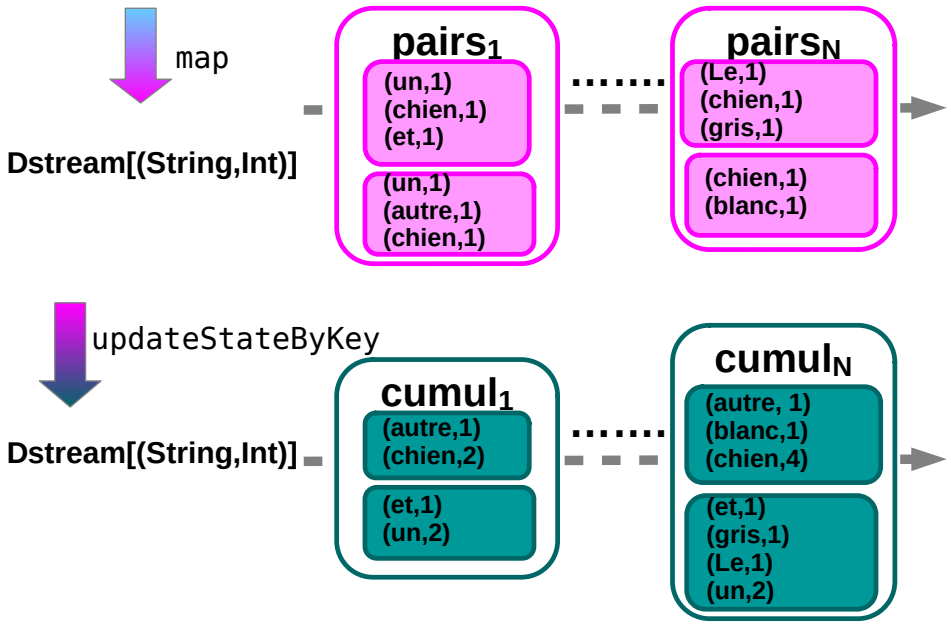


Transformations à états : updateStateByKey

Application au wordcount

Construire un flux qui cumule les compteurs pour chaque mot :

```
val cumul = pairs.updateStateByKey((vals,state:Option[Int]) => state match {
    case None => if(vals.length == 0) Some(0) else Some(vals.reduce(_+_))
    case Some(n) => if(vals.length == 0) Some(n) else Some(n+vals.reduce(_+_))
    })
```

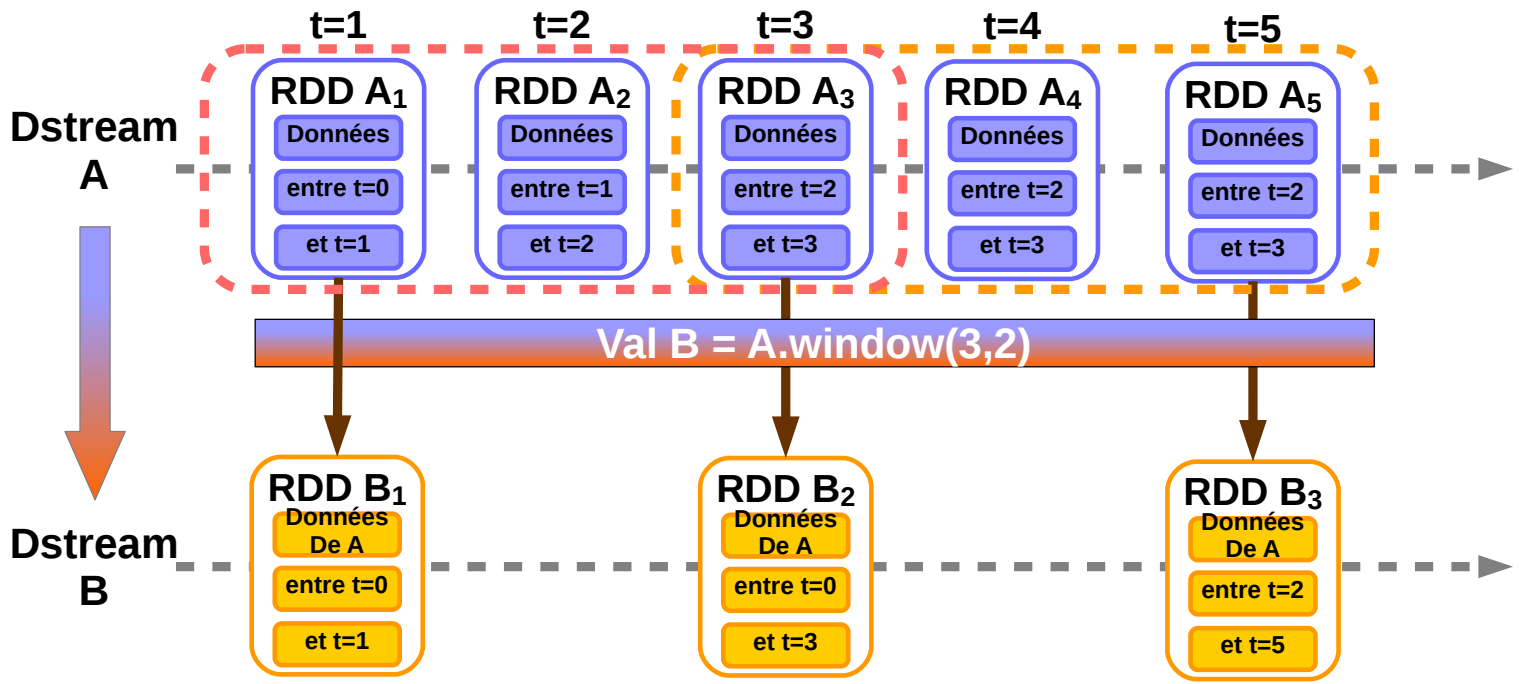


Transformation à états : Windowing

Spécifications

```
def window(windowDur: Duration, slideDur:Duration): DStream[T]
```

- Permet de définir une fenêtre glissante sur un flux
- Union de tous les RDD de la fenêtre
- Les durées doivent être un multiple de la période du flux d'entrée.



Sauvegarde

Persistence

- Comme les RDD, les Dstream peuvent persister/être répliqués en mémoire

  $\Rightarrow$ méthode `persist()`

- La persistance est implicite pour toute les opération de windowing et de sauvegarde d'états.
- persistance par défaut des InputDStream = réplication x2

Checkpointing

Principe

Sauvegarder régulièrement sur un support stable un état du streaming.

`streamingContext.checkpoint(checkpointDirectory)`

Informations sauvegardées

- Méta-données : toute donnée sur le contexte du job :

  configuration, op. sur les DStream, traitements incomplets

  $\Rightarrow$ nécessaire en cas de panne du driver

- Données : les RDD.

  $\Rightarrow$ indispensable pour les transformations à états

Difficulté de régler la fréquence de checkpoints

- Fréquence trop petite $\Rightarrow$ Ralentissement du job
- Fréquence trop grande $\Rightarrow$ Augmentation de la taille des tâches

Recom : période de checkpointing = 5 à 10 fois la période du Stream