TP MAP-REDUCE SOUS PYTHON

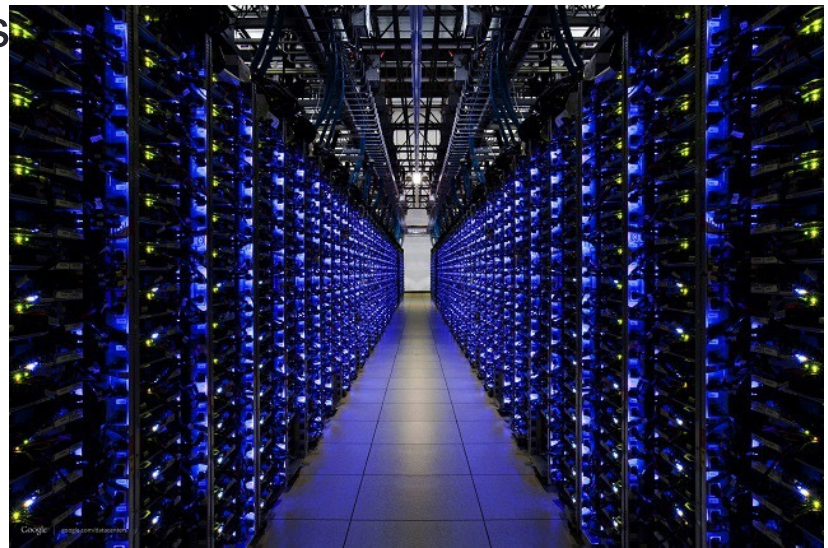
MOD 2.1 : « Défis informatique du Big Data »

« Big Data »

- Exemples:
  - Google, 2008: 20 PB / jour, 180 GB / job
  - Web index : 50 Milliard de pages, 15 PB
  - Large Hadron Collider (LHC)@CERN : 15PB / année

- Capacité d'un (gros) serveur
  - RAM : 256 GB
  - DD : 24 TB
  - Vitesse de transfert du DD : 100 MB/s

- Solution : parallélisme
  - 1 serveur : lit le web en 230 jours
  - Hadoop cluster @ Yahoo : 4000 serveurs, lit le web en // = 1h20

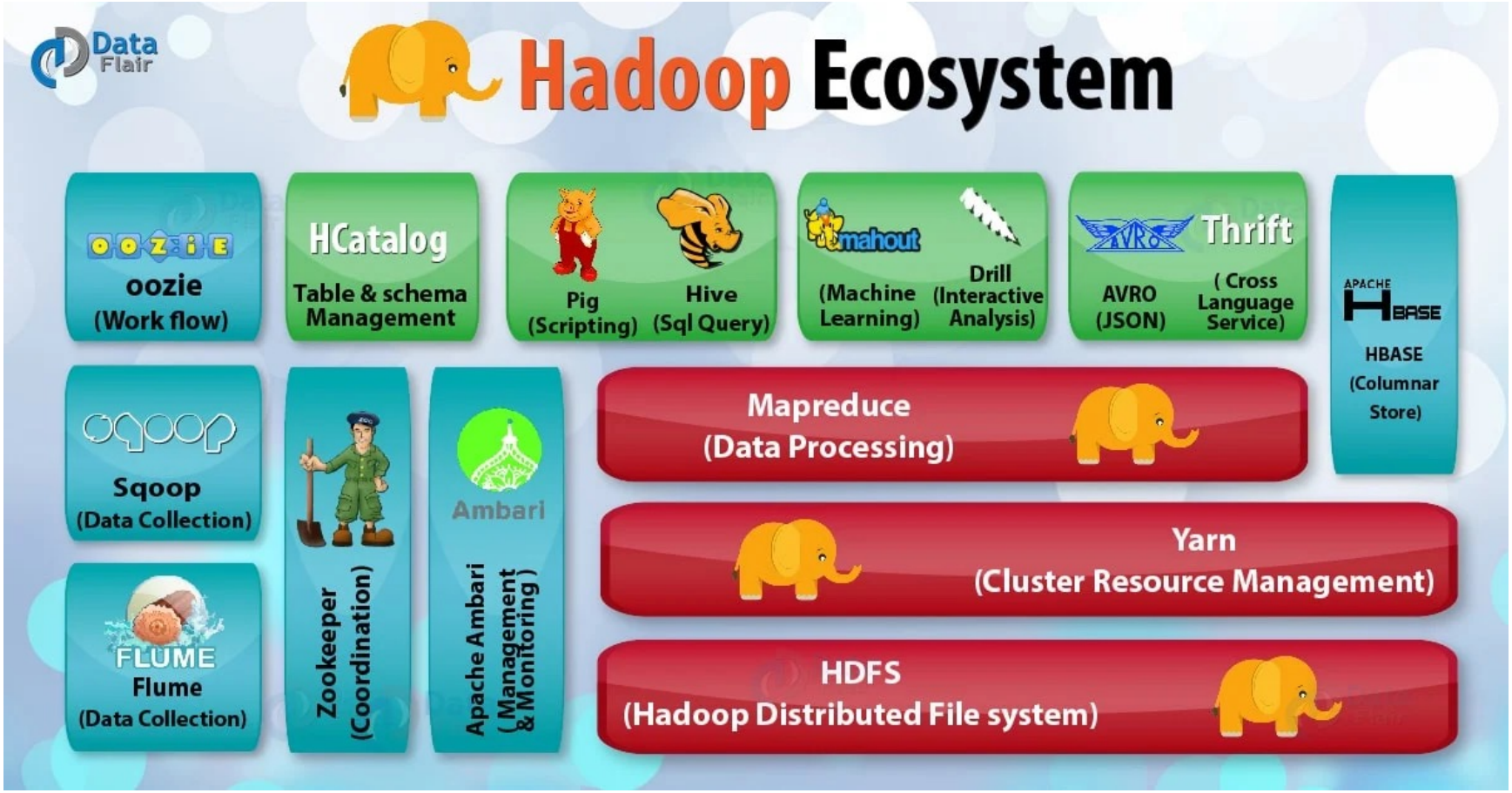


Google data center

Tolérance aux erreurs

- Le problème du parallélisme
  - 1 serveurs bug tous les quelques mois
  - 1000 serveurs -> temps moyen avant bug < 1 jour
- Un « gros » job peut prendre plusieurs jours
  - Une panne matériel : c'est donc la normalité!
  - Parallélisme : impossible de relancer partiellement en cas de panne
  - Point de contrôle, réplication : difficile à implémenter correctement
- Plateformes Big data : tout le monde doit pouvoir écrire des programmes
  - Encapsule le parallélisme
  - Encapsule la tolérances aux pannes
  - Codé une fois par des experts, profitables à tous (non experts)

Environnement Hadoop



Map-Reduce

Deux fonctions très simples inspirées de la programmation fonctionnelle:

- **Transformation : map**
  - map(f, [x_1, …, x_n]) = [f(x_1), …, f(x_n)]
  - Exemple : map(2*, [1, 2, 3]) = [(2*, 1), (2*, 2), (2*, 3)] = [2, 4, 6]
- **Agrégation : reduce**
  - reduce(f, [x_1, …, x_n]) = f(x_1, f(x_2, …, f(x_{n-1}, x_n))
  - Exemple : reduce(+, [2, 4, 6]) = (+2 (+4 6)) = 12

Ces fonctions sont génériques car elles prennent en paramètre une fonction: le développeur fournit les fonctions.

- map(toUpperCase, [‘hello’, ‘data’]) = [‘HELLO’, ‘DATA’]
- reduce(max, [3, 45, 27]) = 45

Données = paires (clé, valeur)

Une clé peut être de n'importe quel type

- **('Hello', 17)**
  - 'Hello' est la clé (text)
  - 17 est la valeur (int)

Lorsque les données ne sont pas des couples (clé, valeur)

- Un texte est représenté par (numéro de ligne, contenu de la ligne)

Map-Reduce appliqué à des couples (clé, valeur)

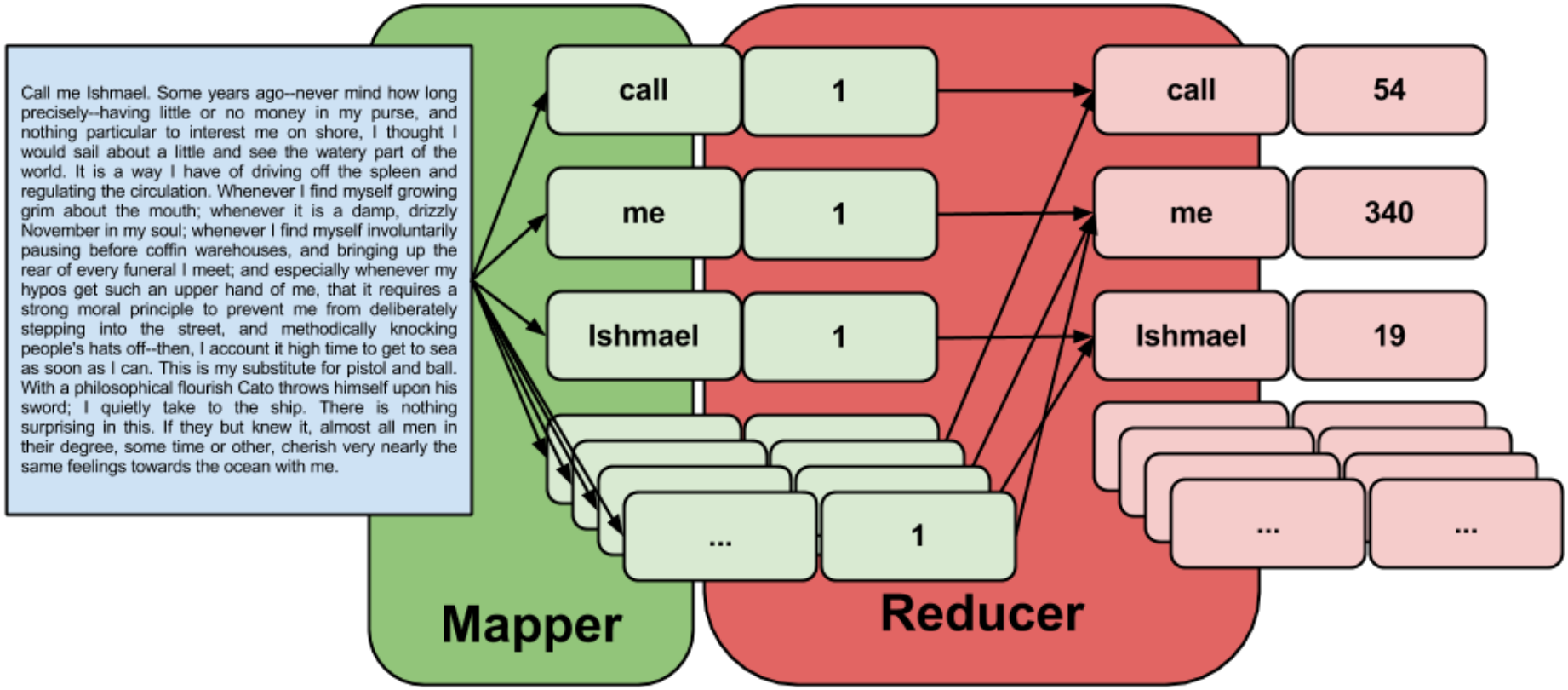
- **Map**, *f* est appliquée sur chaque couple **indépendamment**
  f(clé, valeur)→ list(clé, valeur) (liste de couples)

- **Reduce**, f est appliquée sur **toute les valeurs** de même clé
  f(clé, liste(valeur)) → (clé, valeur) (1 seul couple)

Les types des clés et des valeurs n'ont pas besoin d'être les mêmes en entrée et en sortie.

Map-Reduce famous Word Count

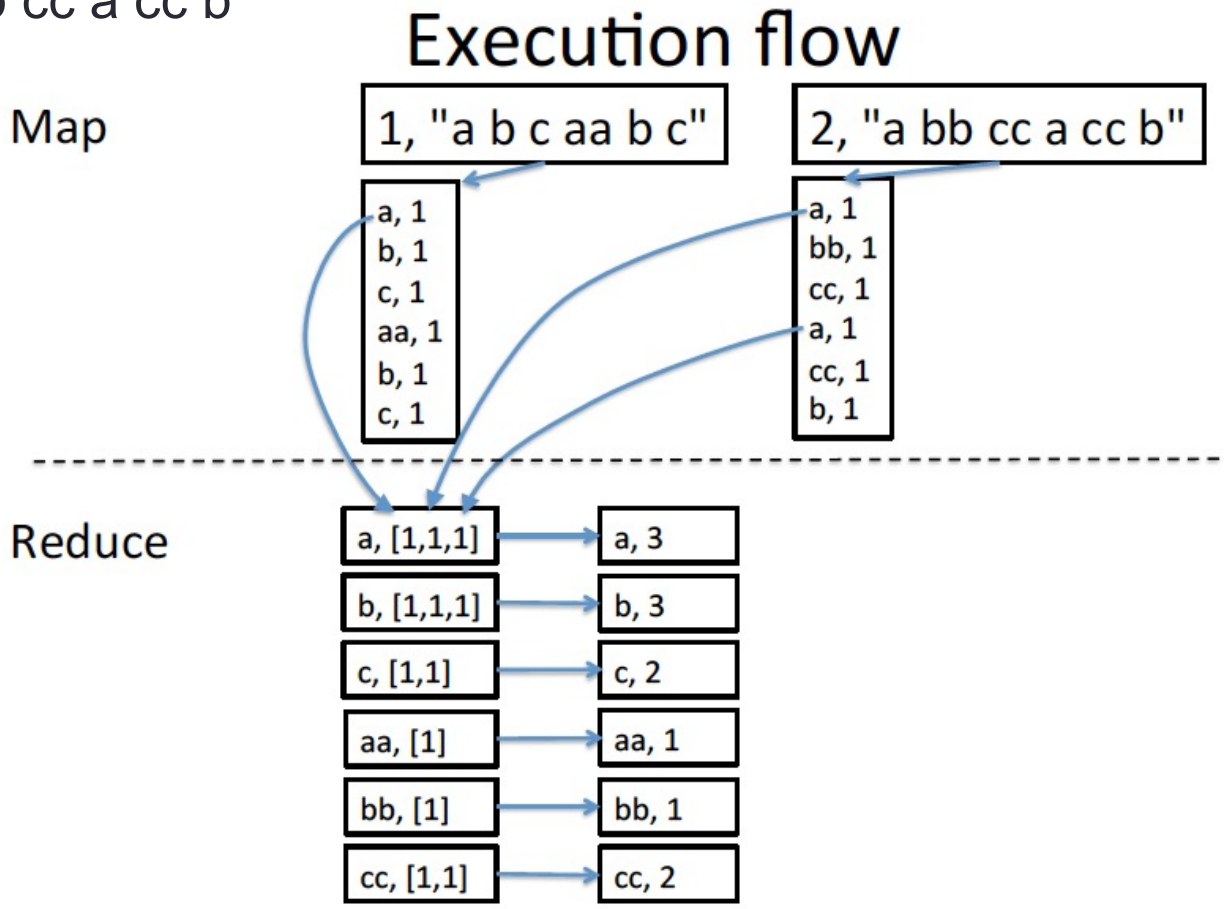
Le ‘Hello world’ du map-reduce



Hadoop map-reduce : natif en Java mais connecteur Python

Exemple : comptage de la fréquence d'un mot

- Données d'entrée: un fichier de 2 lignes
  - 1, 'a b c aa b c'
  - 2, 'a bb cc a cc b'



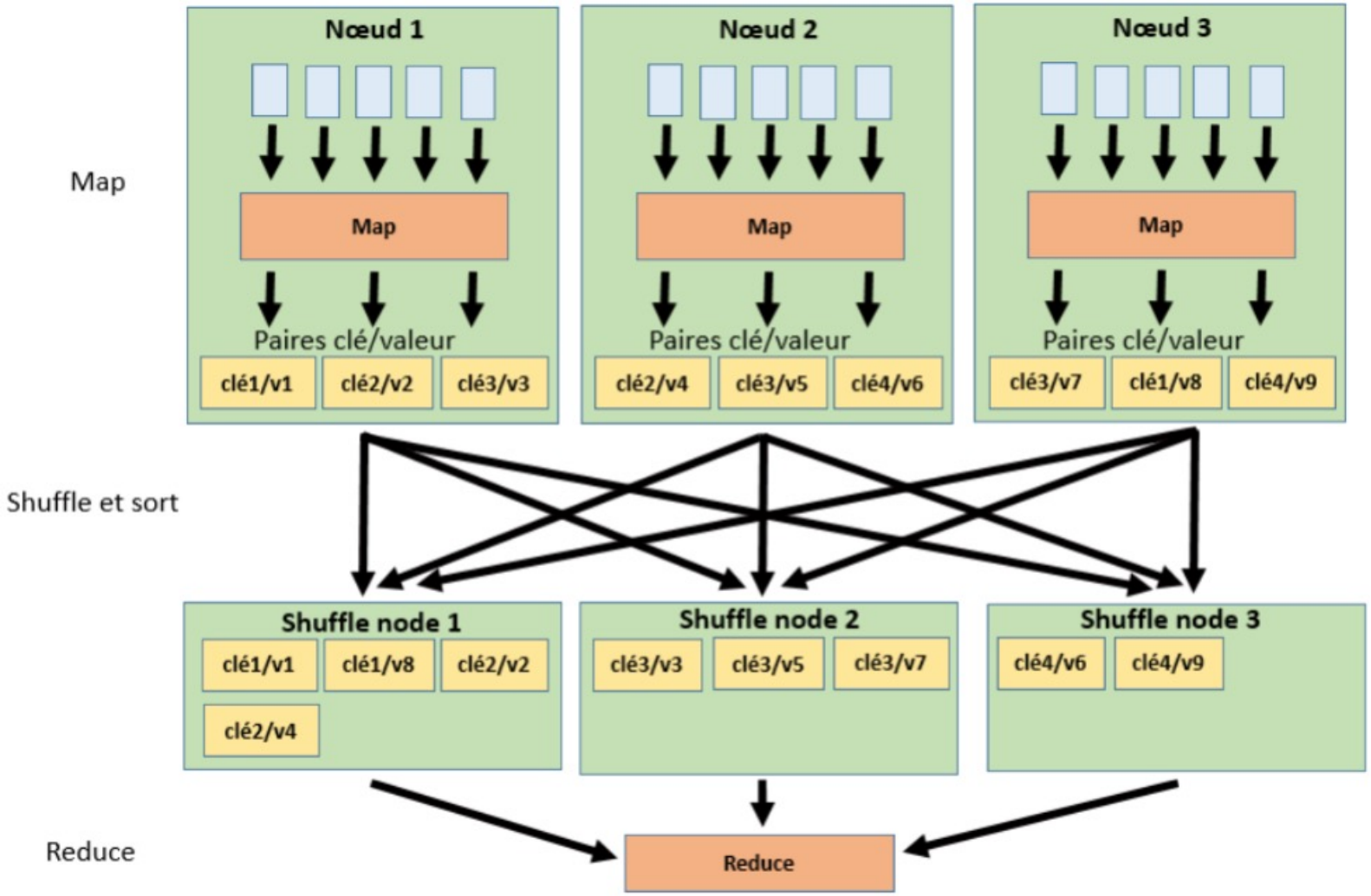
Word Count in python (map)

```
import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line=line.strip()
    # split the line into words
    words=line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        # tab-delimited; the trivial word count is 1
        print(word,'\t1')
```

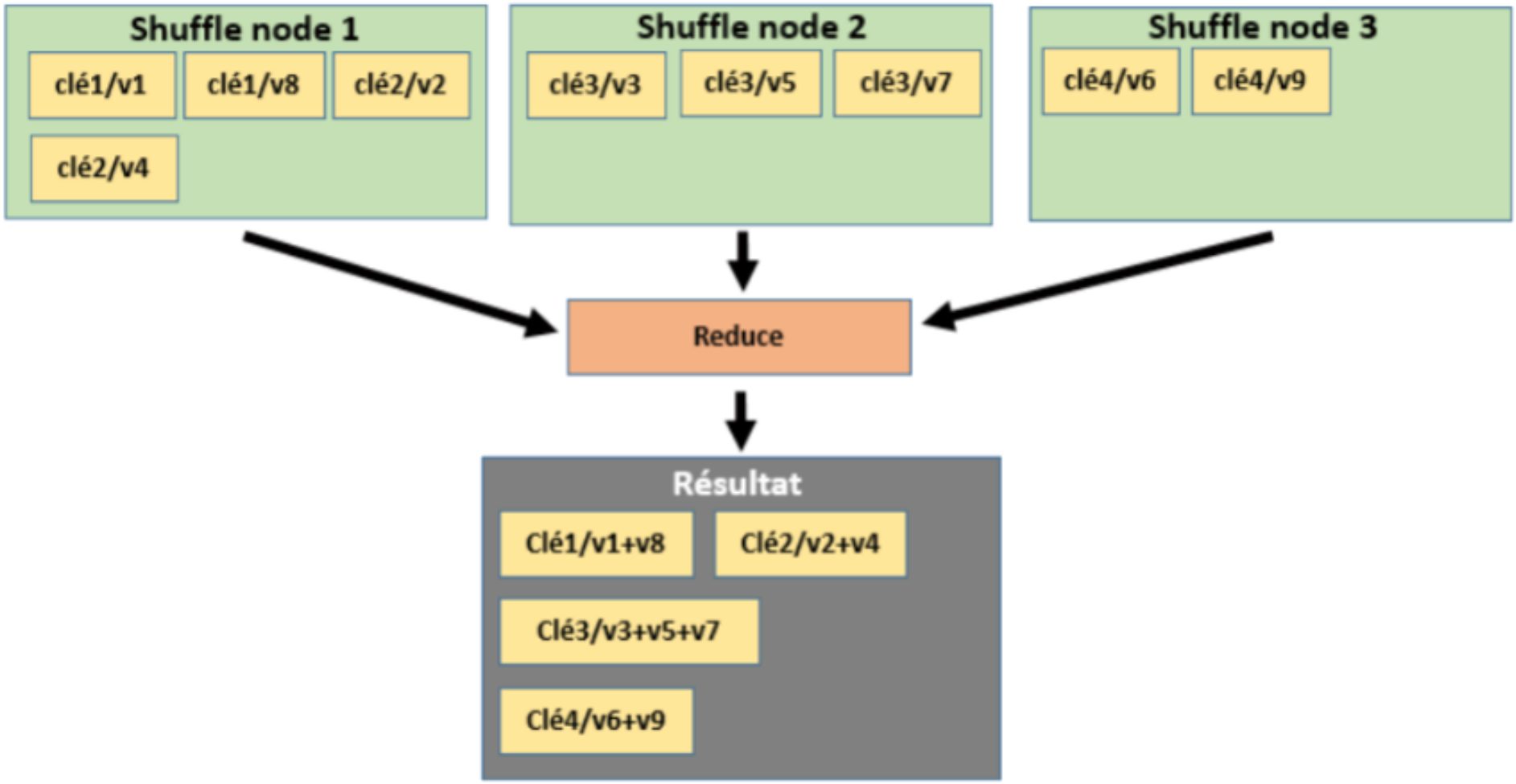
Word Count in python (reduce)

```
import sys

current_word=None
current_count=0
word=None
for line in sys.stdin:
    line=line.strip()
    word, count=line.split('\t', 1)
    try:
        count=int(count)
    except ValueError:
        continue

    if current_word==word:
        current_count+=count
    else:
        if current_word != None:
            print(current_word, '\t', current_count)
        current_count=count
        current_word=word

if current_word==word:
    print(current_word, '\t', current_count)
```

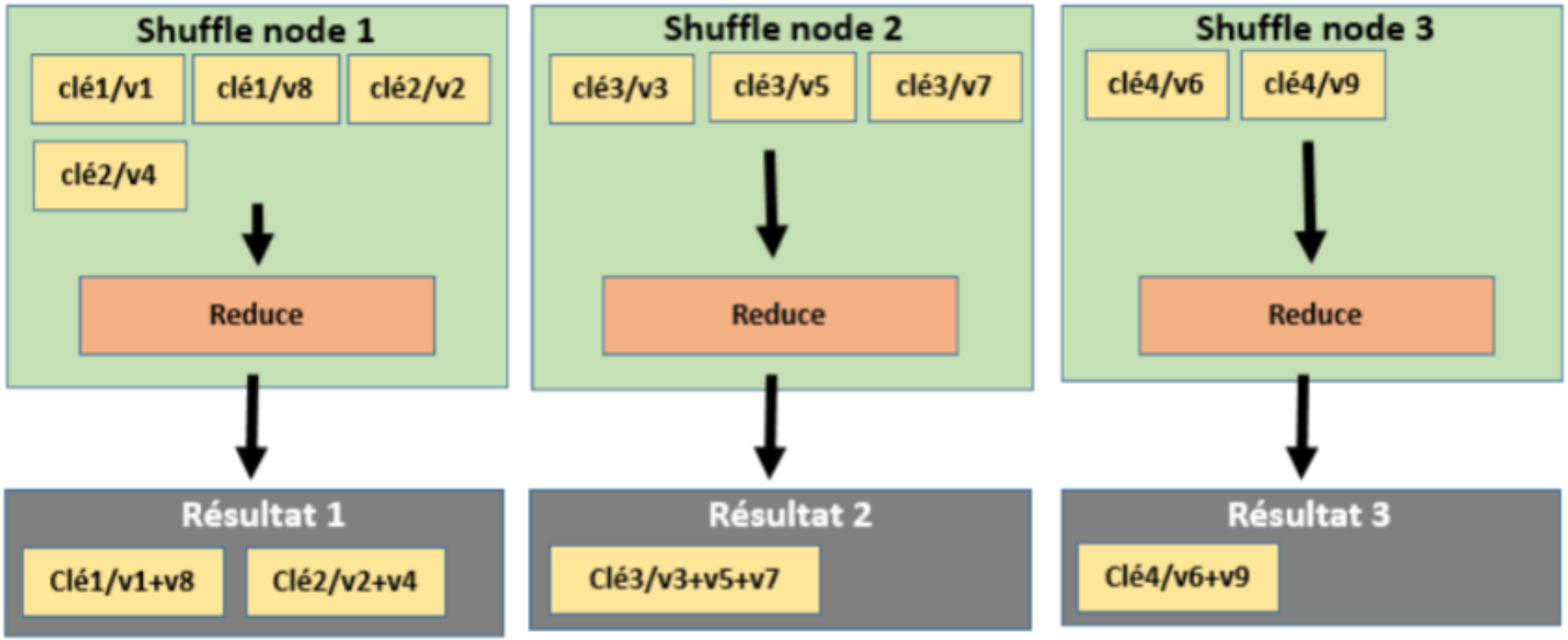
Map Reduce : shuffling and sorting



Map Reduce : reducing



Map Reduce : multiple reducing



Qui utilise Hadoop?

Distribution Apache (libre): https://hadoop.apache.org

Massachusetts Institute of Technology