mapReduce

Big Data

Données et traitements

Les outils de *Big Data* sont...

- capables de stocker et de traiter de très forts volumes de données sur du matériel banalisé

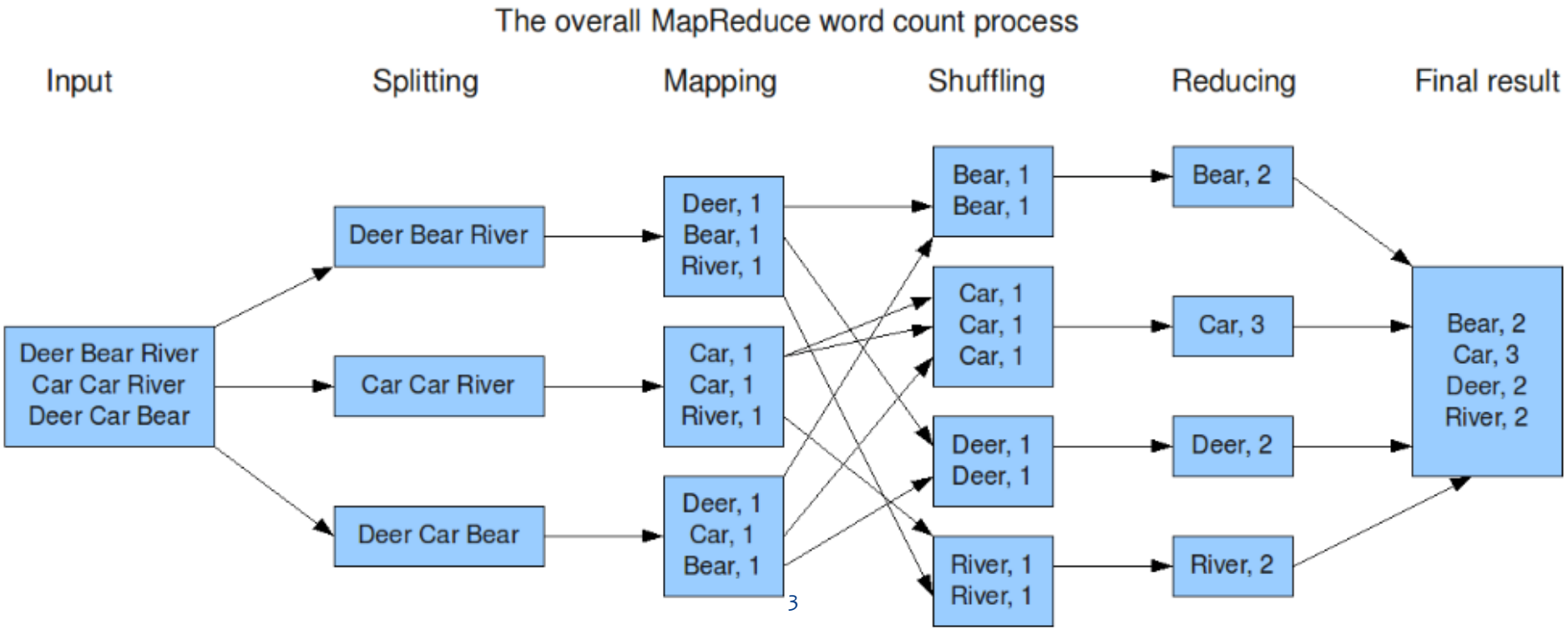
- peu ou pas outillés en algorithmes statistiques ou de *machine learning*

- des outils aux interfaces utilisateurs rudimentaires ou inexistantes, pour programmeurs

- majoritairement des solutions Open Source, avec des communautés de contributeurs & de sponsors

- très robustes, de bonne qualité et **évoluant vite !**

mapReduce


The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

mapReduce

* Technologie apparue en 2004
* Popularisée par Google
* Concept issu de la programmation fonctionnelle (F#, …)
* Utilise la programmation parallèle
* Nécessite plusieurs calculs itératifs
* Scalable technology !
* Dominante open-source (grandes échelles = économie)
* Résistance aux pannes
* Mouvance Big Data et noSQL

mapReduce

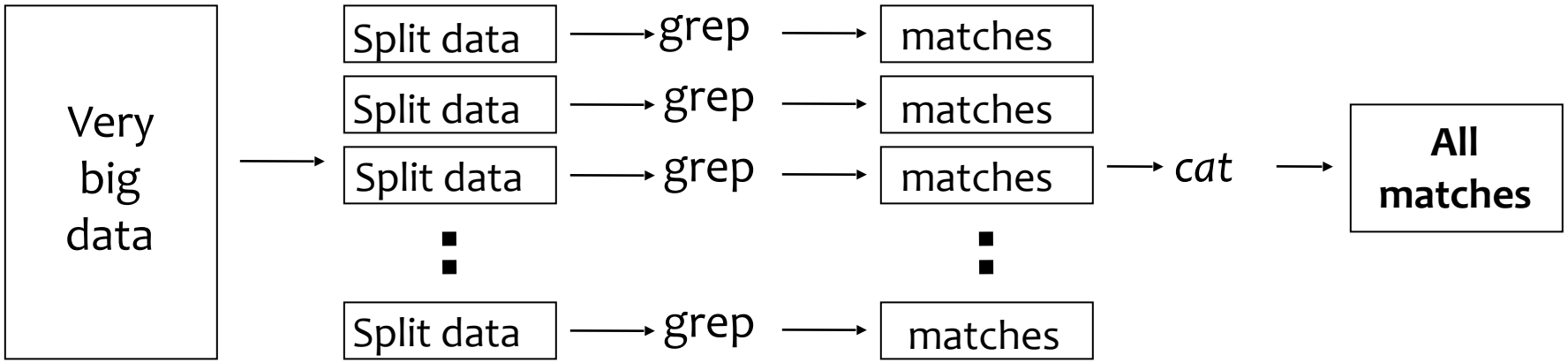
* Fait appel à deux fonctions essentielles :
  * Map : travaille sur un ensemble clés/valeurs
  * Reduce : permet d'agréger le travail des différents nœuds utilisés
* Fiabilité :
  * Programmes simples
  * Machines fiables
  * Redondance, tolérance aux pannes

mapReduce

* Scalable :
  * Adaptable à la charge
  * sans modifier le code
  * En augmentant le nombre de nœuds (1, 2, …)

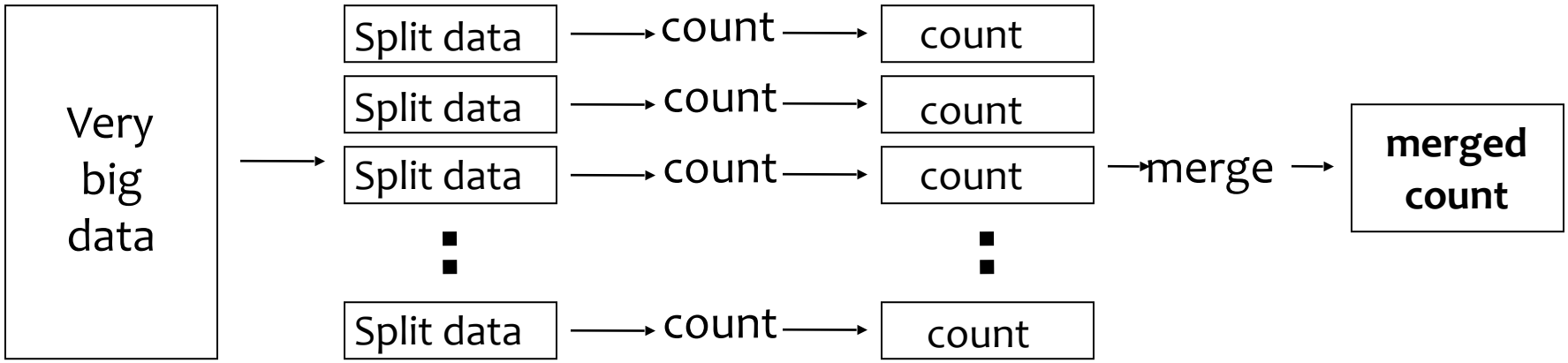
Applications

* Distributed Grep :



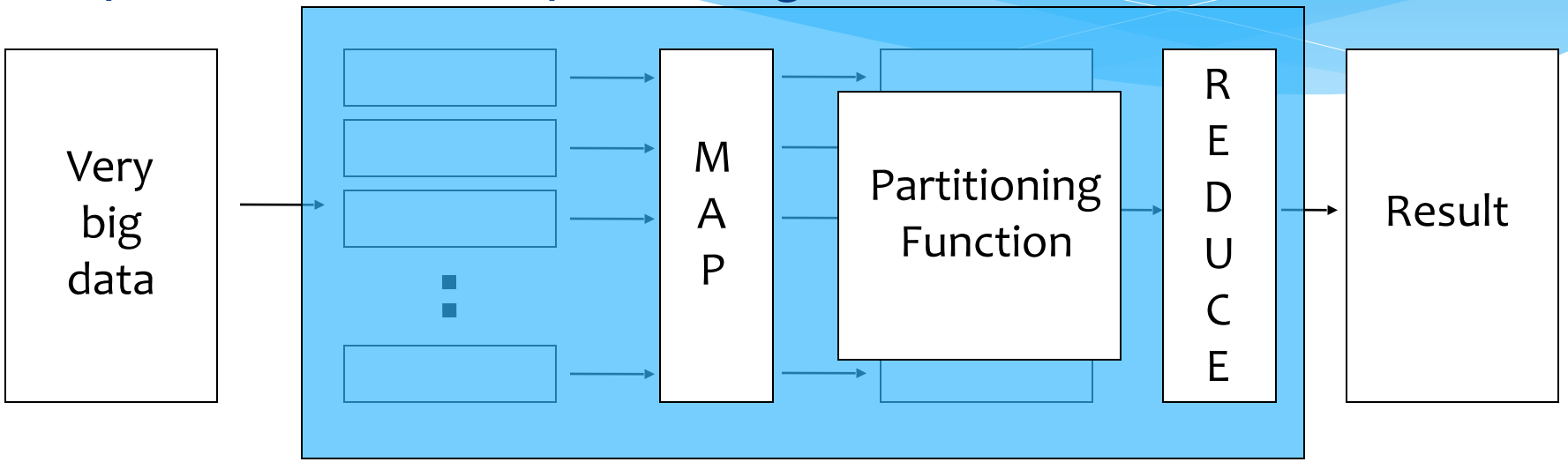
Applications

* Distributed Word Count :



Applications

* Map Reduce Scalable processing :



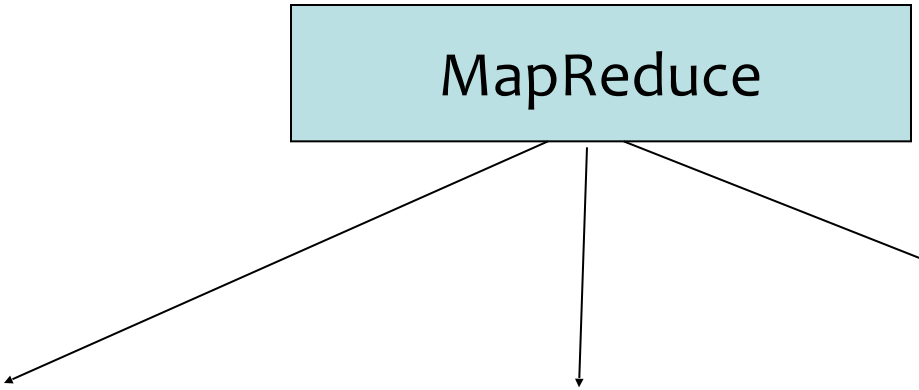
* Map:
  * Accepts *input* key/value pair
  * Emits *intermediate* key/ value pair
* Reduce :
  * Accepts *intermediate* key/ value pair
  * Emits *output* key/value pair

Applications

* String Match, such as Grep
* Reverse index
* Count URL access frequency
* Lots of examples in data mining

Implémentations

* Au niveau matériel :



Cluster,
1, Google
2, Apache Hadoop

Multicore CPU,
Phoenix @ stanford

GPU,
Mars@HKUST

Phoenix

* MapReduce for multiprocessor systems
* Shared-memory implementation of MapReduce
  * SMP, Multi-core
* Features
  * Uses thread instead of cluster nodes for parallelism
  * Communicate through shared memory instead of network messages
  * Dynamic scheduling, locality management, fault recovery

Phoenix API

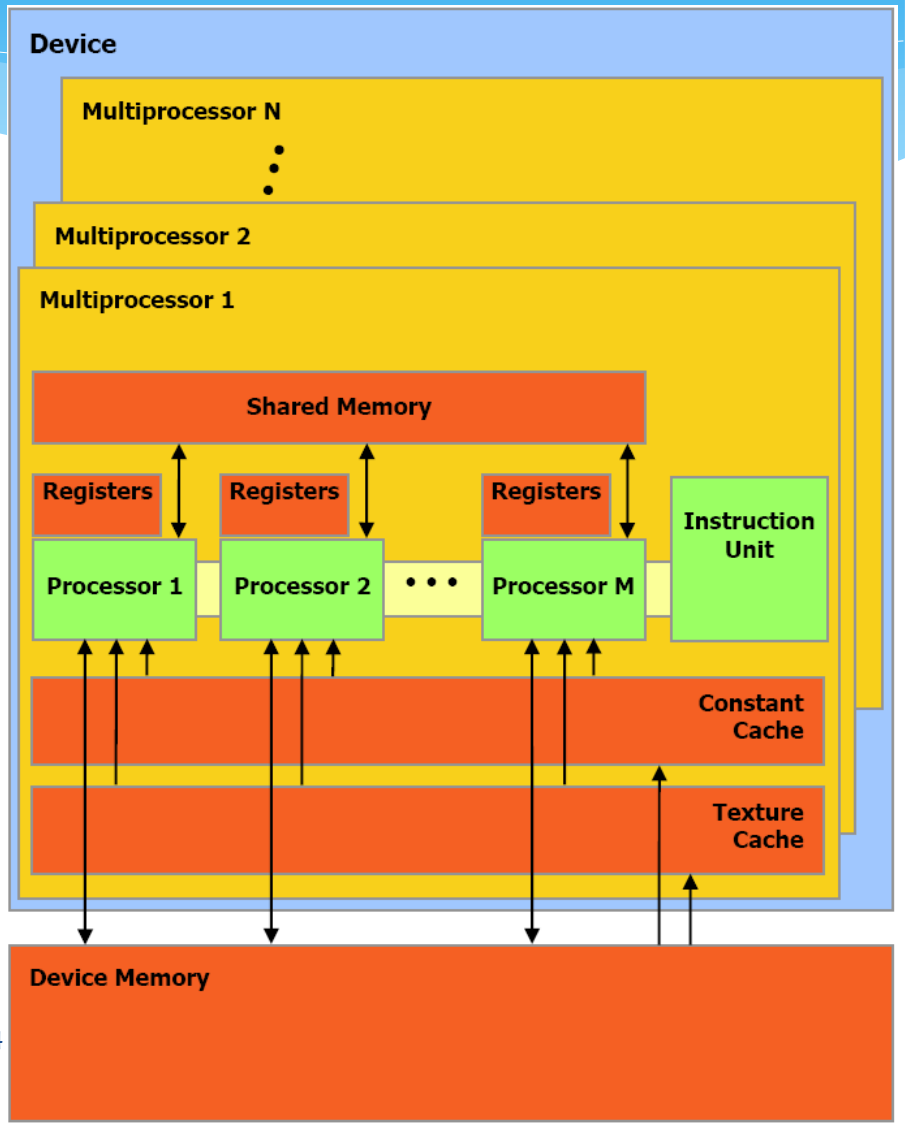
* System-defined functions
  - `int phoenix_scheduler (scheduler_args_t *args)`
    - Initializes the runtime system
  - `void emit_intermediate (void *key, void *val, int key_size)`
  - `void emit (void *key, void *val)`
* User-defined functions

- `void (*map_t) (map_args_t *args)`
  - Map function applied on each input element
- `void (*reduce_t) (void *key, void **buffer, int count)`
  - Reduce function applied on intermediate pairs with same key
- `int (*key_cmp_t) (const void *key1, const void *key2)`
  - Function that compares two keys
- `int (*splitter_t) (void *input, int size, map_args_t *args)`
  - Splits input data across Map tasks (optional)

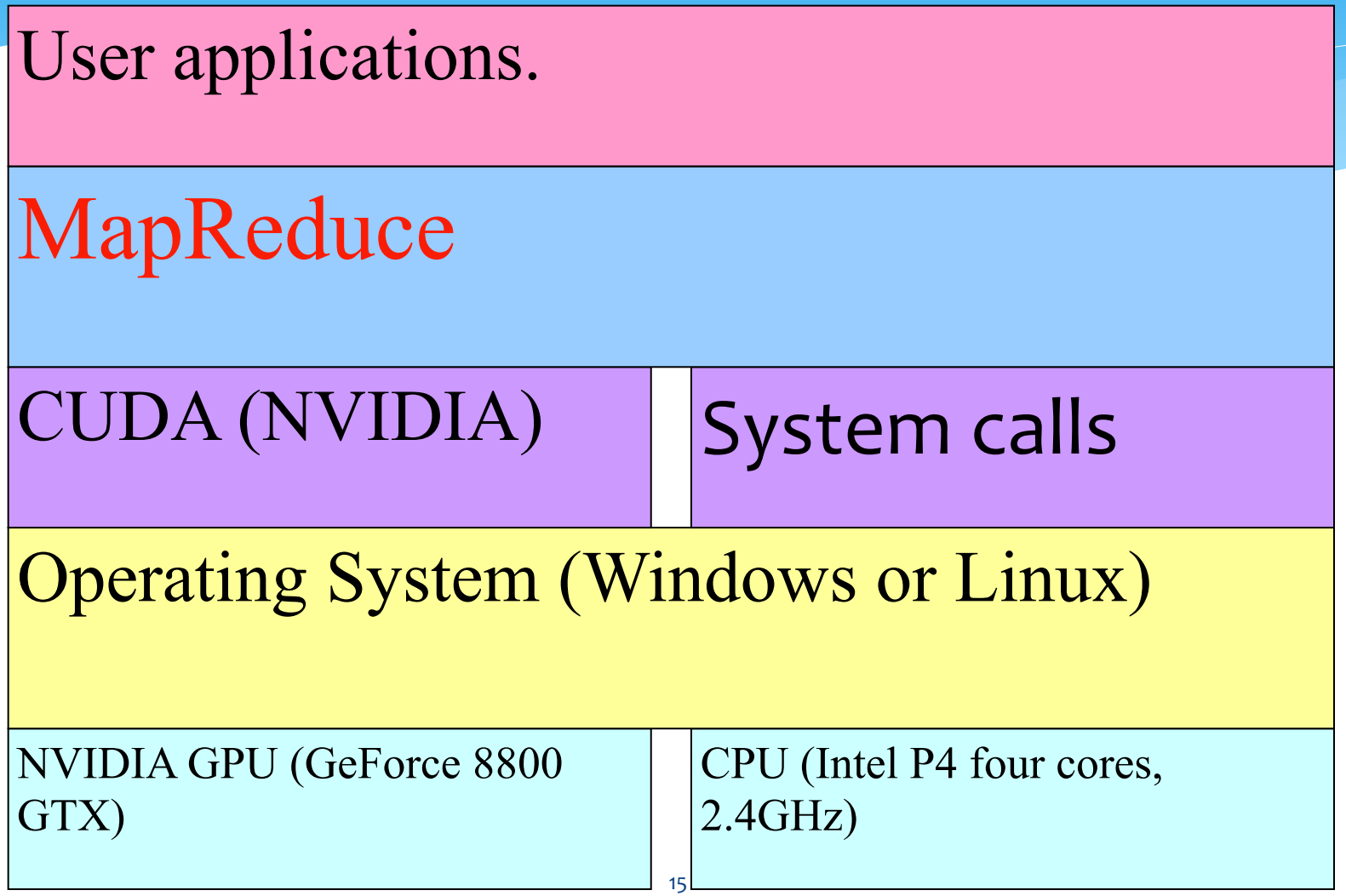
Mars : MapReduce on GPU

* PACT'08

GeForce 8800 GTX, PS3, Xbox360



Implementation of Mars

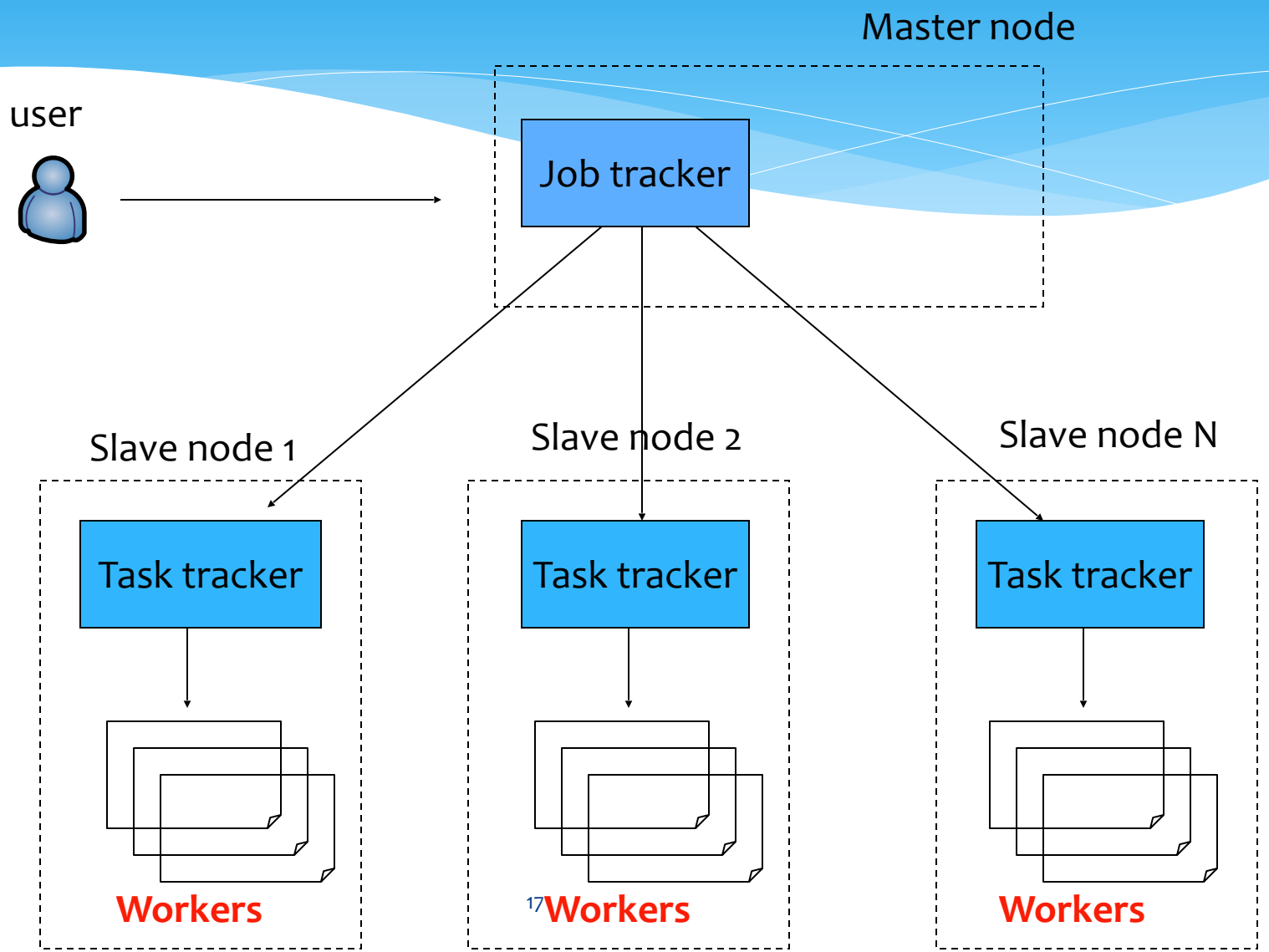


Implémentations

* Au niveau logiciel :
* noSQL, clusters maître/esclave, SGF spécifiques

  * Google : mapReduce, GFS, BigTable (noSQL), Chubby (gestion des nœuds)
  * Yahoo : Hadoop, HDFS, Hbase (noSQL), zooKeeper (gestion des nœuds)

mapReduce overview



Fault Tolerance

* Reactive way
  * Worker failure
    * Heartbeat, Workers are periodically pinged by master
      * NO response = failed worker
    * If the processor of a worker fails, the tasks of that worker are reassigned to another worker.
  * Master failure
    * Master writes periodic checkpoints
    * Another master can be started from the last checkpointed state
    * If eventually the master dies, the job will be aborted

Implémentations

* Frameworks & API :
  * JAVA
  * C++, C#
* Solutions cloud :
  * Cloudera : http://www.cloudera.com/content/cloudera/en/why-cloudera/hadoop-and-big-data.html
  * WindowAzure : https://www.hadooponazure.com
* Solutions :
  * hortonsWorks : http://hortonworks.com
  * Hadoop : http://hadoop.apache.org
* Stacks solutions :
  * VM : http://hortonworks.com/products/hortonworks-sandbox/

Implémentations

* Offres SandBox hadoop (horton) :

Installation Steps

1. Install a virtualization environment (3 Options)
2. Download & Import the respective Sandbox Image

VirtualBox (Recommended)	VMWare Fusion or Player	Hyper-V	HDP 1.3 Sandbox
Runs on Oracle VirtualBox 4.2+ - Available as a free download.	Runs on VMware Fusion 5.0 + (Mac) or VMware Player (Win)	Runs on Microsoft Hyper-V	For legacy testing. Some partner tutorials are also written for 1.3
Sandbox 2 for Virtualbox (2.5 GB)	Sandbox 2 for VMWare (2.6 GB)	Sandbox 2 for Hyper-V (2.5 GB)	on VirtualBox »
Instructions : Mac | Windows	Instructions : Mac	Instructions for Hyper-V (Win)	for VMWare »
			for Hyper-V »
			HDP 1.3 Documentation

Programmation

* Utilisation d'un nœud disponible (Cloud ou local):
  * Fichier en entrée à formater
  * Ecriture du mapper et du reducer
  * Soumission du mapper et du reducer en C# ou Java
  * Lancement du calcul
  * Recueil des résultats
* Aucunes compétences nécessaires

Write Mapper

```
01. //Simple mapper that extracts namespace declarations
02.
03. using System.IO;
04. using System;
05. using System.Collections.Generic;
06. using System.Linq;
07. using System.Text;
08. using System.Text.RegularExpressions;
09.
10. namespace StackOverflowAnalyzer.Mapper
11. {
12.     class Program
13.     {
14.
15.         static void Main(string[] args)
16.         {
17.
18.             string line;
19.             Regex reg=new Regex(@"(using)\s[A-za-z0-9_\.]*\;");
20.
21.             while ((line = Console.ReadLine()) != null)
22.             {
23.                 var matches = reg.Matches(line);
24.                 foreach (Match match in matches)
25.                 {
26.                     Console.WriteLine("{0}", match.Value);
27.                 }
28.             }
29.         }
30.     }
31. }
```

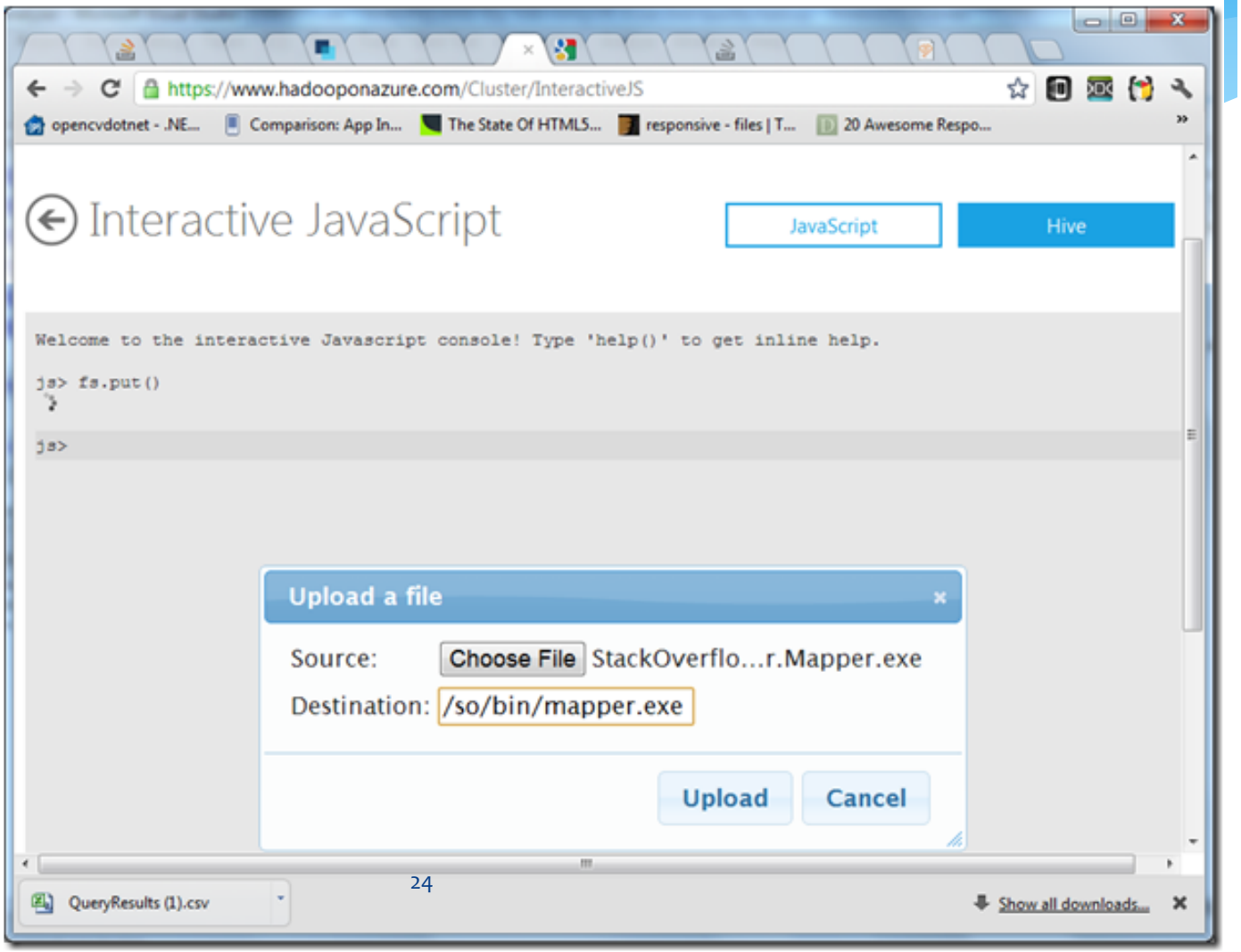
Write Reducer

```
01. using System.IO;
02. using System;
03. using System.Collections.Generic;
04. using System.Linq;
05. using System.Text;
06. using System.Threading.Tasks;
07.
08. namespace StackOverflowAnalyzer.Reducer
09. {
10.     class Program
11.     {
12.         static void Main(string[] args)
13.         {
14.             string ns, prevns = null;
15.             int nscount = 0;
16.
17.             while ((ns = Console.ReadLine()) != null)
18.             {
19.                 if (prevns != ns)
20.                 {
21.                     if (prevns != null)
22.                     {
23.                         Console.WriteLine("{0} {1}", prevns, nscount);
24.                     }
25.                     prevns = ns;
26.                     nscount = 1;
27.                 }
28.
29.                 else
30.                 {
31.                     nscount += 1;
32.                 }
33.             }
34.             Console.WriteLine("{0} {1}", prevns, nscount);
35.         }
36.     }
37. }
```

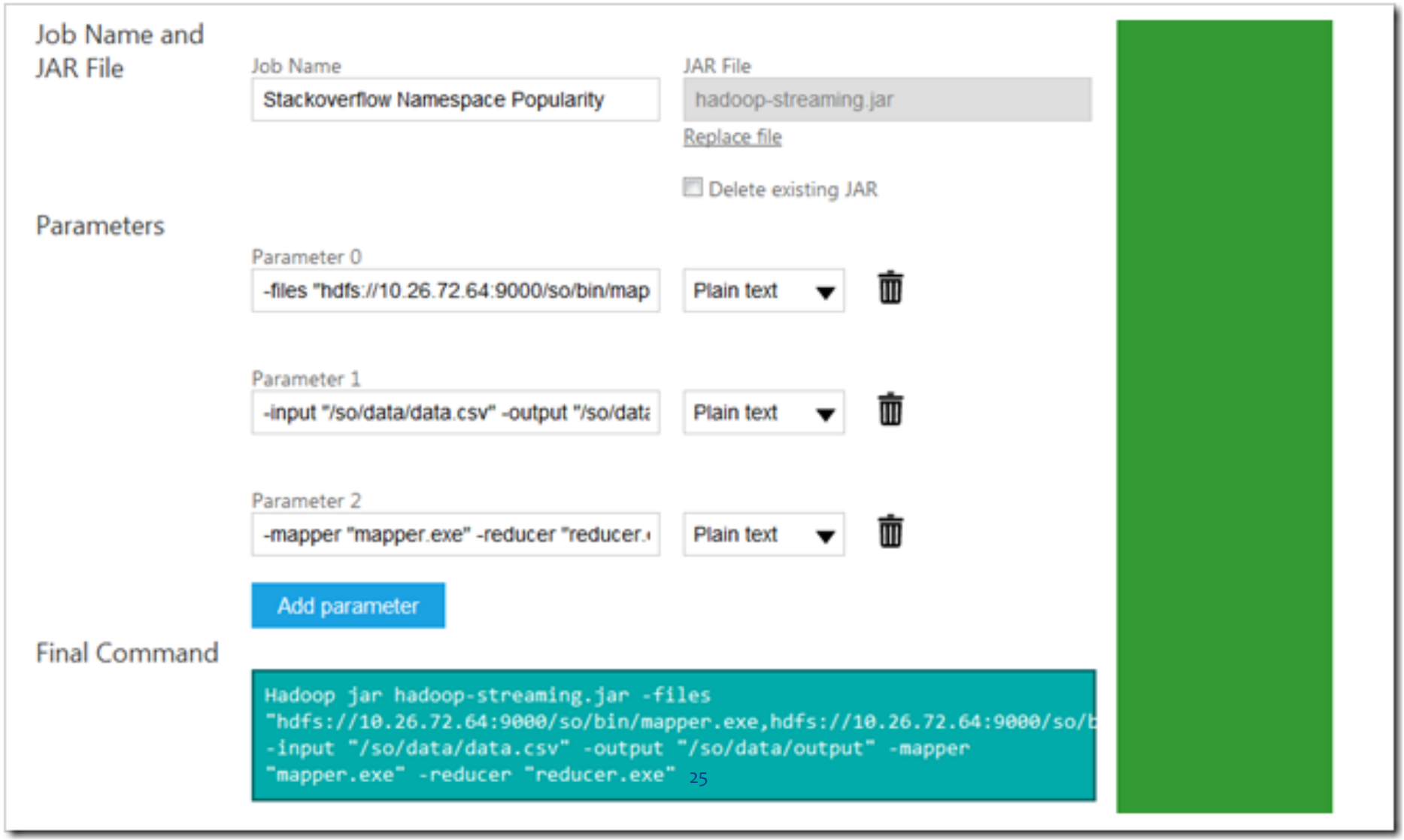
Upload files

* Let's upload :

* Mapper
* Reducer
* Csv



Parameters


Job Name and JAR File
Job Name
Stackoverflow Namespace Popularity
JAR File
hadoop-streaming.jar
Replace file
Delete existing JAR
Parameters
Parameter 0
-files "hdfs://10.26.72.64:9000/so/bin/map
Plain text
Parameter 1
-input "/so/data/data.csv" -output "/so/data
Plain text
Parameter 2
-mapper "mapper.exe" -reducer "reducer.
Plain text
Add parameter
Final Command
Hadoop jar hadoop-streaming.jar -files
"hdfs://10.26.72.64:9000/so/bin/mapper.exe,hdfs://10.26.72.64:9000/so/b
-input "/so/data/data.csv" -output "/so/data/output" -mapper
"mapper.exe" -reducer "reducer.exe"

Go !

Stackoverflow Namespace Popularity

Job Info

Status: Completed Sucessfully
Type: jar
Start time (UTC): 6/5/2012 8:30:59 AM
End time (UTC): 6/5/2012 8:32:00 AM
Exit code: 0

Command

```
call hadoop.cmd jar hadoop-streaming.jar -files
"hdfs://10.26.72.64:9000/so/bin/mapper.exe,hdfs://10.26.72.64:9000/so/bin/reducer.exe" -input
"/so/data/data.csv" -output "/so/data/output.txt" -mapper "mapper.exe" -reducer "reducer.exe"
```

Output (stdo image

packageJobJar: [] [/C:/Apps/dist/lib/hadoop-streaming.jar] D:\Users\AMAZED~1\AppData\Local\Temp\streamjob2237188379918920714.jar tmpDir=null

Results

```
using ;  1
using B;         1
using BerkeleyDB;         1
using Castle.Core.Interceptor;    1
using Castle.DynamicProxy;        1
using ClassLibrary1;      1
using ClassLibrary2;      1
using Dates;      1
using EnvDTE;     2
```

Programmation

* Le concept est réexploitable dans d'autres langages :
  * En C, à la norme POSIX
  * En ObjC, Swift, …
* Proposer une solution locale :
  * En C,
  * Utilisant des données ouvertes …

Données ouvertes

* Open.data.gouv
* Open.data.gov

* Trouver des données au format .txt à exploiter

* Attention aucune intégrité n'est demandée
* Le jeu de données est souple.

Calcul parallèle

* Utilisation de threads :
  * Répartition du calcul (à la charge du programmeur)
  * Optimisation
  * Problème de changement de contexte

* Un seul nœud :
  * Plusieurs processus
  * Différent de la réalité (plusieurs nœuds)

Problème à traiter

* Un wordCount fera l'affaire :
  * Mappage des occurrences
  * Reduce des occurrences
  * Affichage des résultats

  * Attention : il faut gérer la répartition du travail sur les processus.
  * Ce n'est pas le cas avec le framework hadoop.

Problème de contexte

* Parallélisation :
  * Contexte mémoire à partager, gestion du temps
  * Partage des données entre processus
  * Faire le bon choix …

Processus

* Fork()
  * Mémoire non partagée
  * Processus fils et père séparé

* Partage avec Fork():
  * Pipe : synchrone
  * IPC (inter Processus Communication) : asynchrone, SHM (shared memory)

Processus

* Threads (pthread):
  * Mémoire partagée : gestion plus simple, mais aussi plus dangereuse
  * Processus Synchrone et Asynchrone : mutex, sémaphore,
  * Utilisation de pointeurs possible
  * Pas de variables globales !
  * Norme POSIX : API disponibles et standards (*N_X)
  * Pour Windows : https://sourceware.org/pthreads-win32/

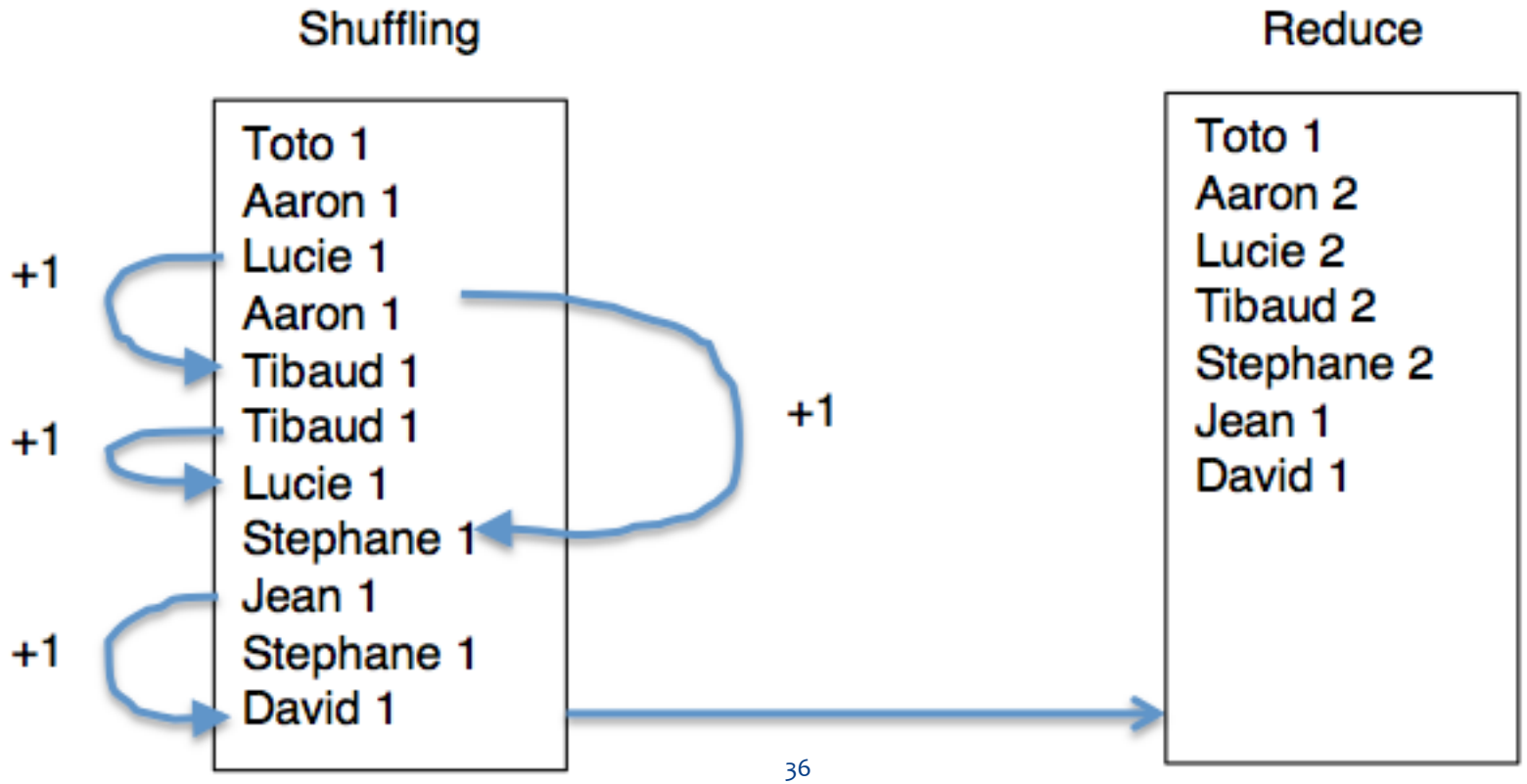
Etapes

* Traitement mono-processus :
  * Lecture du fichier
  * Map & Reduce combiné
  * Affichage

* Passage à plusieurs processus :
  * Map sur les processus (parallèle)
  * Reduce à la fin des processus (combination)

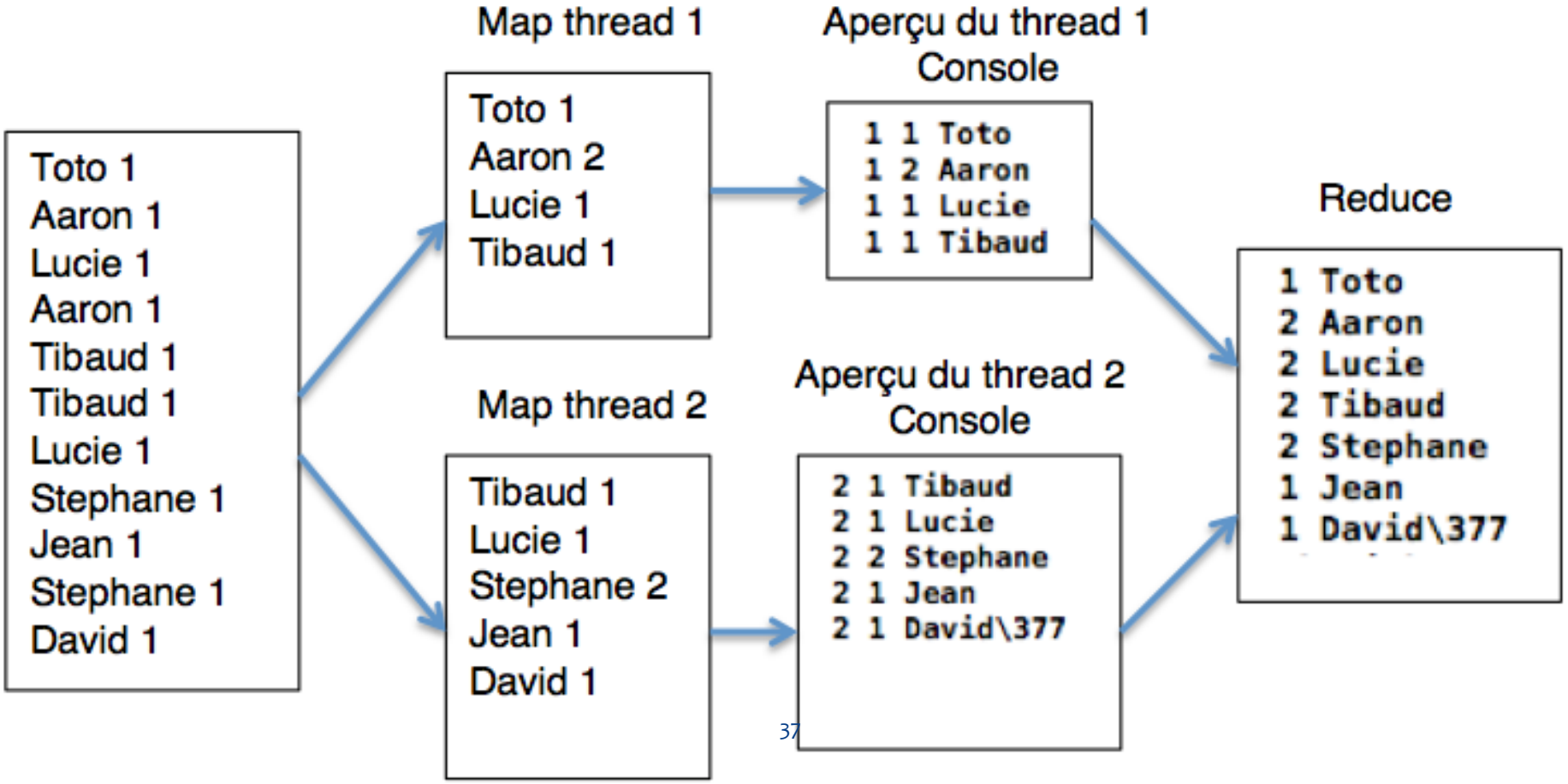
Problème à traiter

* Mono-thread :



Problème de contexte

* Multithreads :



Etapes

* Interface et implémentation :
  * Séparées
  * Aucune variable globale (gain en global ?)
  * Passage de paramètres a chaque sous programme

* Mesure du temps d'exécution :
  * Appréciation des temps de traitement
  * Comparaison

* Pas d'interface requise !

Exemple sur jeu simple

```
Nb ticks/seconde = 1000000,  Nb ticks depart : 1580, Nb
ticks final : 2165
Temps consomme (s) : 0.000585
```

```
prenoms
Aaron
Abdallah
Aaron
Aaron
Abel
Abigail
Aboubacar
Adele
Aude
Jerome
```

```
Toto
Aaron
Lucie
Aaron
Tibaud
Tibaud
Lucie
Stephane
Jean
Stephane
David
```

* Données de départ :
  * Liste de mots
* Données d'arrivée :
  * Map (key, value) +
  * reduce(key, value)

```
1 prenoms
3 Aaron
1 Abdallah
1 Abel
1 Abigail
1 Aboubacar
1 Adele
1 Aude
1 Jerome\377
```

```
1 Toto
2 Aaron
2 Lucie
2 Tibaud
2 Stephane
1 Jean
1 David\377
```

* Prévoir plusieurs jeux d'essai, avec des combinaisons différentes
* La gestion des accents est à votre charge …

Jeu de données massif

* La fiabilité sur jeu simple, permet de passer à un jeu conséquent,
* Impossibilité dans ce cas de vérifier le résultat.
* Suivant la machine, compter 1 seconde pour 10 000 occurrences.
* Pour 10130 occurrences :

```
Nb ticks/seconde = 1000000,  Nb ticks depart : 1655, Nb
ticks final : 1430446
Temps consomme (s) : 1.428791
```