# Big Data & Analytics

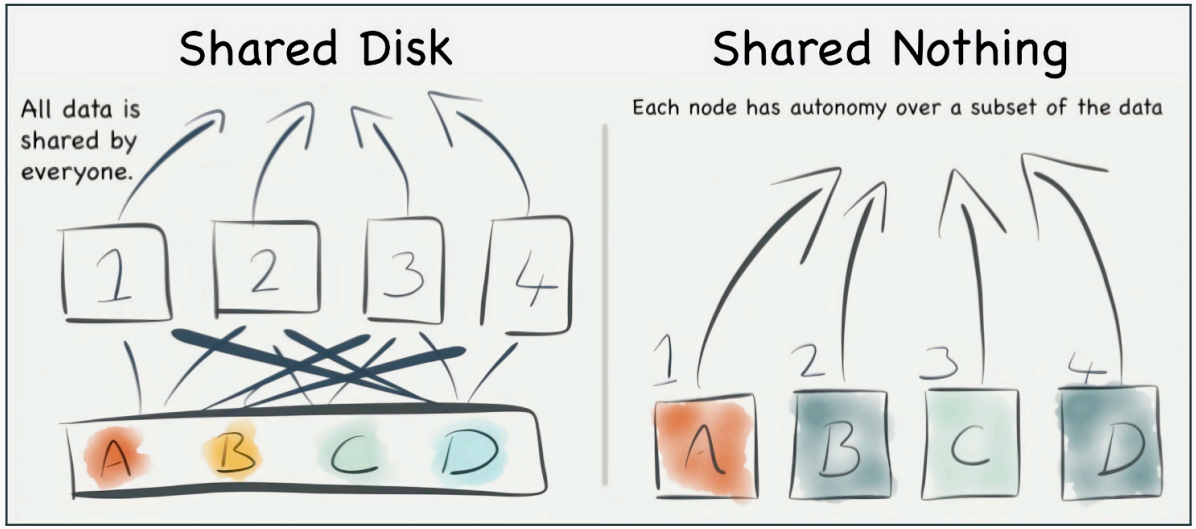
## MapReduce - Introduzione

Pierluca Ferraro
*pierluca.ferraro@unipa.it*

**Università degli Studi di Palermo**

# MapReduce

# Paradigma shared nothing



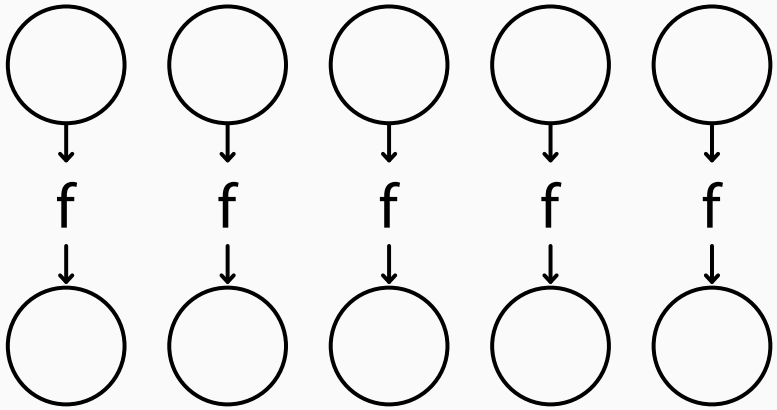

- Nella programmazione multi-threading, i dati delle elaborazioni sono condivisi (shared memory).
- MapReduce adotta invece il paradigma shared nothing.
- La condivisione dei dati è eliminata, e questi sono passati tra le funzioni come parametri o valori di ritorno.
- Alcuni concetti sono ispirati dalla programmazione funzionale.

# Hadoop MapReduce

- Deriva da Google MapReduce[1].
- Prevede due fasi principali: map e reduce.
- Hadoop divide i dati di input in parti (tipicamente della stessa dimensione di un blocco HDFS) e assegna ciascuna parte ad un mapper.
- Caratteristiche delle funzioni map e reduce:
  - mancanza di effetti collaterali (no side-effect);
  - il risultato della funzione dipende solo dall'input;
  - applicando più volte una funzione sugli stessi dati, il risultato non cambia.
- In caso di malfunzionamento di un mapper, è possibile ripetere solo la sua porzione di calcoli.

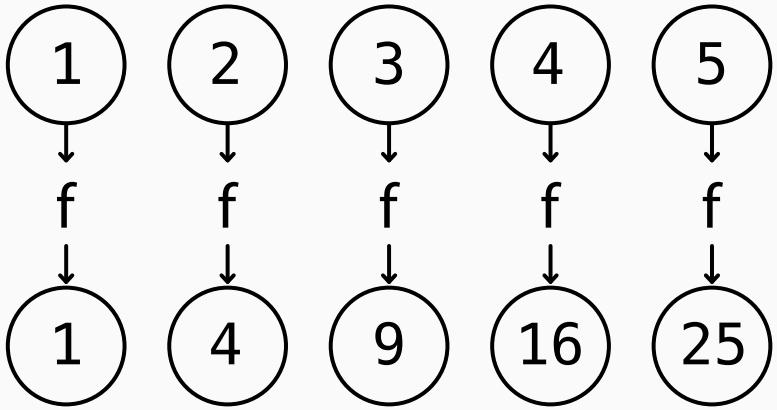
[1] https://www3.nd.edu/~dthain/courses/cse40822/fall2014/papers/mapreduce.pdf

# Funzione map



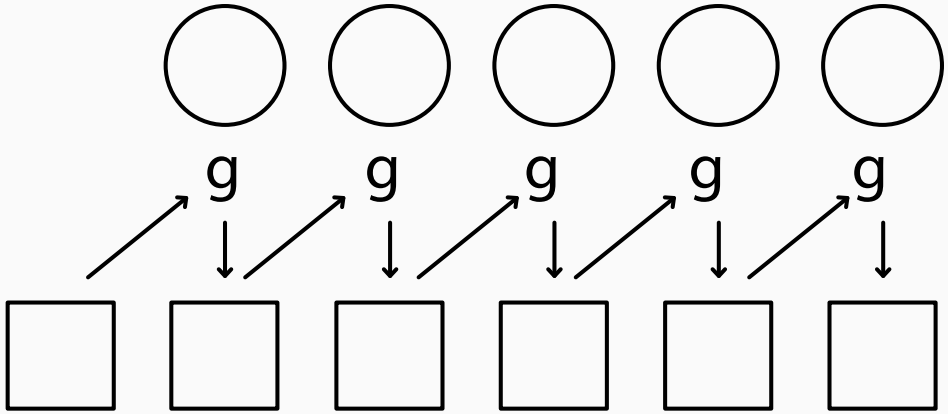

- Applica la stessa funzione a tutti gli elementi dell'input.
- L'ordine in cui vengono elaborati i valori non è importante.
- Ogni elemento dell'input può essere elaborato in modo indipendente.
- Possiamo assegnare ad ogni nodo del cluster il compito di applicare la funzione map su una parte dell'input.

# Funzione map



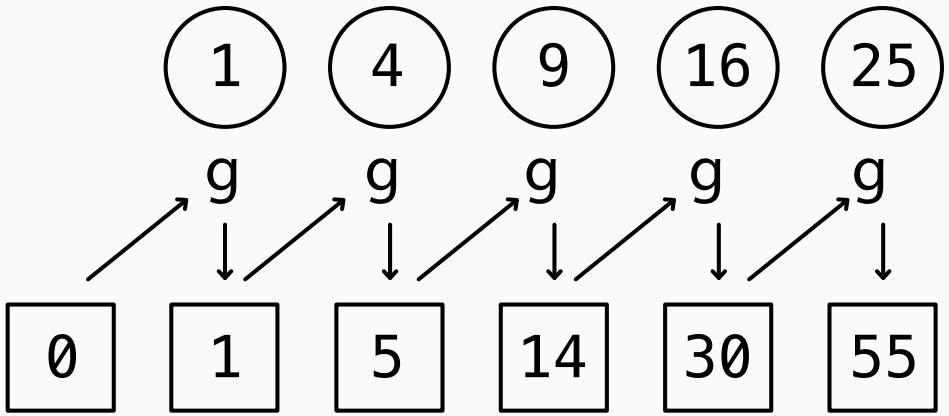

- Applica la stessa funzione a tutti gli elementi dell'input.
- L'ordine in cui vengono elaborati i valori non è importante.
- Ogni elemento dell'input può essere elaborato in modo indipendente.
- Possiamo assegnare ad ogni nodo del cluster il compito di applicare la funzione map su una parte dell'input.
- Esempio funzione f: elevamento al quadrato.

# Funzione reduce




- Riduce gli elementi di una lista, ottenendo un unico risultato.

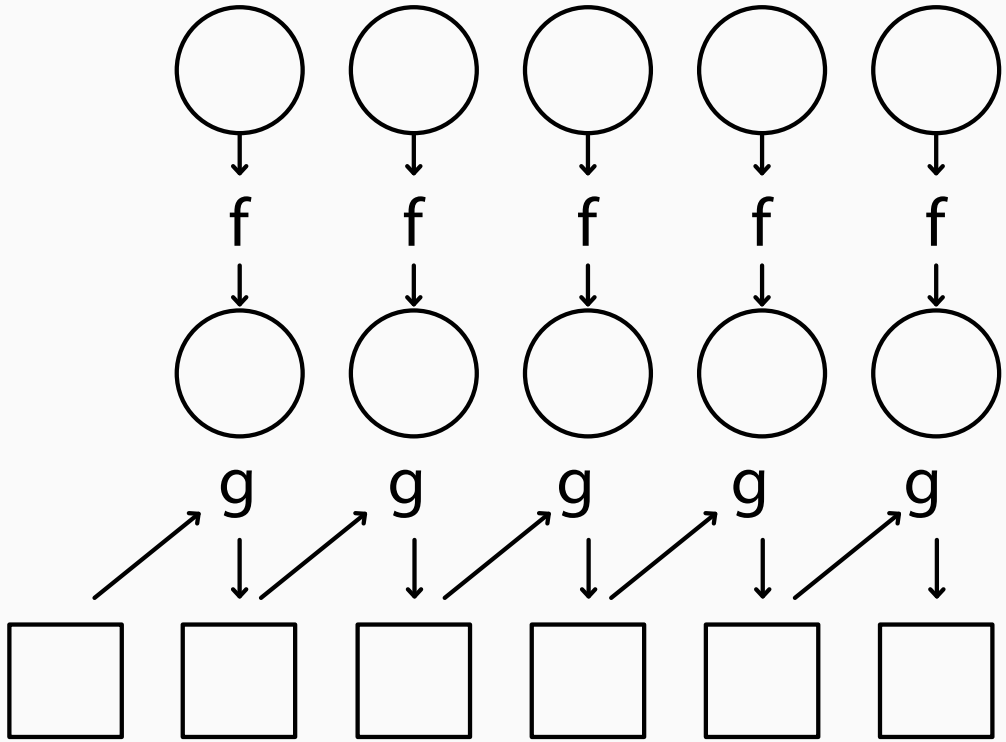
# Funzione reduce



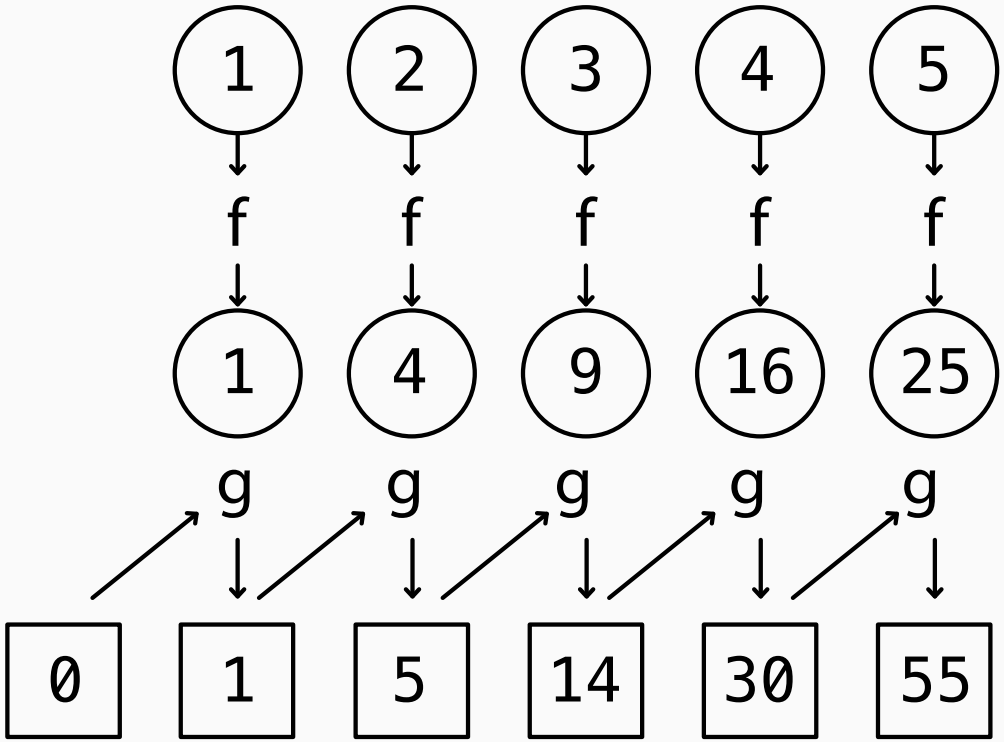

- Riduce gli elementi di una lista, ottenendo un unico risultato.
- Esempio funzione g: somma.

```
 0 +  1 =  1
 1 +  4 =  5
 5 +  9 = 14
14 + 16 = 30
30 + 25 = 55
```

# Map + reduce



# Map + reduce - Somma dei quadrati



# Primo programma MapReduce

# Partiamo dai dati

| Ora (CET) | Temp. | Punto di rugiada | Umidità | Pressione | Visibilità | Wind Dir | Velocità del vento | Velocità Raffica | Precip | Eventi | Condizioni |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 12:20 AM | **18.0** °C | **7.0** °C | 49% | **1009** hPa | **10.0** Km | Sud | **24.1** km/h / **6.7** m/s | - | N/A | | Nubi sparse |
| 12:50 AM | **18.0** °C | **7.0** °C | 49% | **1008** hPa | **10.0** Km | Sud | **37.0** km/h / **10.3** m/s | **55.6** km/h / **15.4** m/s | N/A | | Parzialmente nuvoloso |
| 1:00 AM | **18** °C | **6** °C | 34% | **1008** hPa | **10** Km | Sud | **40.7** km/h / | - | - | | Nubi sparse |
| 1:20 AM | **18.0** °C | **6.0** °C | 45% | **1007** hPa | **10.0** Km | Sud | **46.3** km/h / **12.9** m/s | **74.1** km/h / **20.6** m/s | N/A | | Parzialmente nuvoloso |

- Abbiamo a disposizione un dataset contenente dati rilevati da varie stazioni meteorologiche per un particolare luogo.
- Consideriamo una versione semplificata del dataset che include soltanto data, ora e temperatura.
- I dati sono organizzati in file csv (comma-separated values), divisi per anno.
- Esempio di alcune righe di input:

```
2016-06-09,19:00:00,23.3
2016-06-09,20:00:00,22.5
2016-06-09,21:00:00,22.0
2016-06-09,22:00:00,22.0
```

# Divisione del lavoro in più task - Singolo computer

- Vogliamo calcolare la temperatura massima di ogni anno.
- Il dataset può essere grande, quindi vogliamo parallelizzare il calcolo.
- Possibile approccio: usare tutti i core su una macchina, assegnando una parte del lavoro a ciascun core.

- Problema: non è facile dividere il lavoro in parti uguali.
- Se ci sono parti che hanno molti più record di altre, il tempo di esecuzione totale dipenderà dalla lunghezza del task più lungo.

# Divisione del lavoro in più task - Singolo computer

- Non è detto che ogni file abbia lo stesso numero di record, quindi non possiamo semplicemente dividere il lavoro per anno.
- Se dividiamo i dati in parti uguali in base alla dimensione, il calcolo potrebbe diventare difficile:
  - i dati che ci servono potrebbero essere assegnati a processi diversi.
- Siamo comunque limitati dalla capacità computazionale di una singola macchina (numero di core, dimensione hard disk...)

# Divisione del lavoro in più task - Cluster di computer

**Nuovi problemi se pensiamo di utilizzare più macchine**

- Chi coordina l'intero job?
- Chi divide il lavoro in task, bilanciando il carico tra i nodi?
- Chi si occupa di gestire il fallimento di un task (e come)?

# Divisione del lavoro in più task - Cluster di computer

**Nuovi problemi se pensiamo di utilizzare più macchine**

- Chi coordina l'intero job?
- Chi divide il lavoro in task, bilanciando il carico tra i nodi?
- Chi si occupa di gestire il fallimento di un task (e come)?

**Hadoop** si occupa automaticamente di tutti questi problemi.

- Dobbiamo ripensare il lavoro che vogliamo svolgere per adattarlo al paradigma MapReduce.

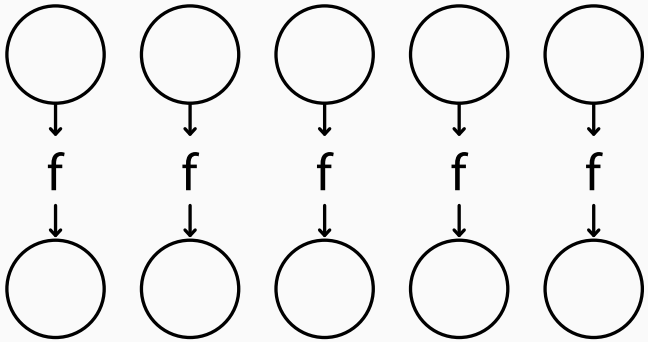
# Job MapReduce

Un job MapReduce, nella sua forma più semplice, è costituito da quattro componenti:

- I dati di input, salvati in una directory su HDFS.
- Una funzione map, che trasforma i dati di input in una serie di coppie chiave-valore.
- Una funzione reduce che, per ogni chiave, elabora i valori ad essa associati.
- I dati di output, salvati su uno o più file in una directory su HDFS.

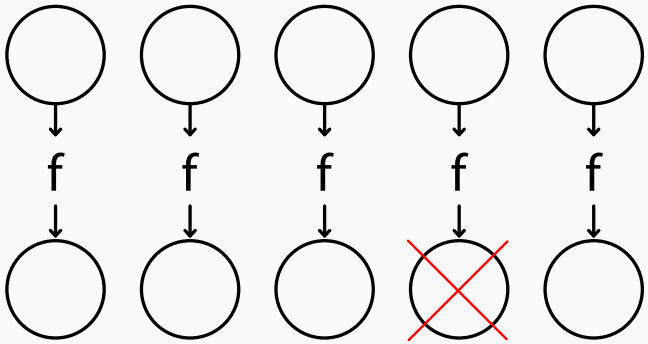
Tra la fase map e quella reduce avviene la cosiddetta fase shuffle and sort.

# Fase map




- I dati di input vengono suddivisi e assegnati ai mapper.
- L'input e output della fase di map sono coppie chiave-valore.
- In molti casi è necessario trasformare l'input per individuare chiavi e valori, scartando i dati che non ci interessano.
- In questa fase possono anche essere filtrati i dati errati o considerati poco affidabili.
- I dati verranno, in seguito, raggruppati in base alla chiave. È quindi necessario scegliere la chiave con attenzione, in modo che i reducer possano svolgere il loro lavoro.

# Fase map




Input:

```
2016-06-09,16:00:00,23.3
2014-07-05,20:00:00,32.5
2017-02-01,14:00:00,11.0
2016-06-09,23:00:00,80.0
2016-06-09,21:00:00,22.1
```

Output:

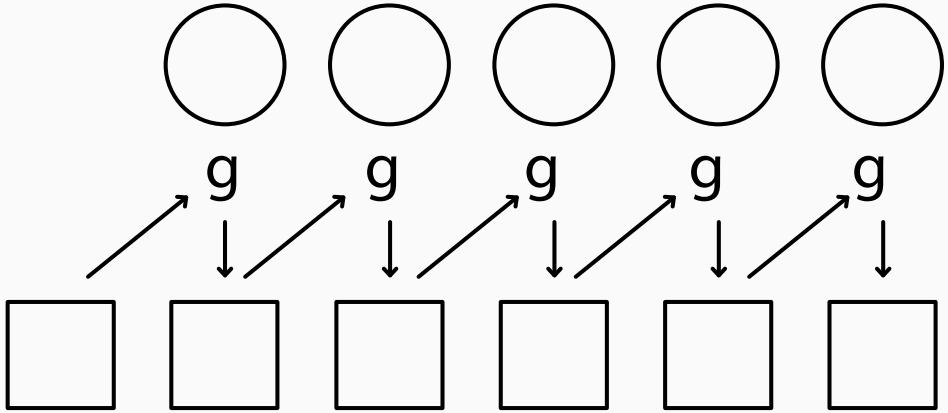
```
(2016, 23.3)
(2014, 32.5)
(2017, 11.0)
(2016, 22.1)
```

- Scegliamo l'anno come chiave, ignorando tutte le altre informazioni che riguardano data e ora.
- Filtriamo i dati palesemente errati (temperatura di 80°C nell'esempio).
- Restituiamo coppie del tipo `(anno, temperatura)`.

# Fase shuffle and sort

- Durante questa fase, le coppie chiave-valore restituite dai mapper vengono suddivise in base alla chiave e assegnate ad un reducer.
- Tutte le coppie con una particolare chiave saranno assegnate ad un singolo reducer.
- Le coppie saranno anche ordinate, in modo tale che tutti i valori con la stessa chiave siano elaborati consecutivamente dal reducer.
- È possibile intervenire sul criterio di partizionamento e sul tipo di ordinamento.

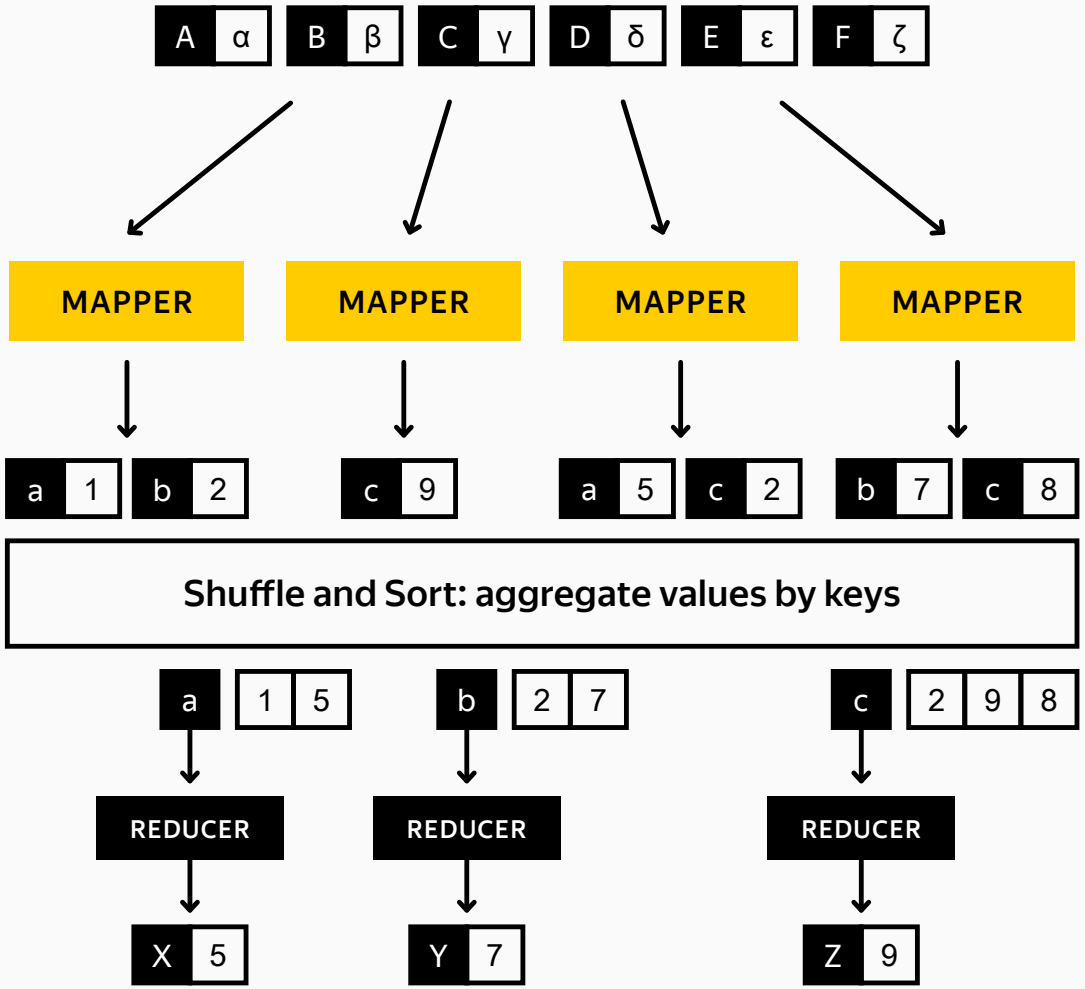
# Fase reduce



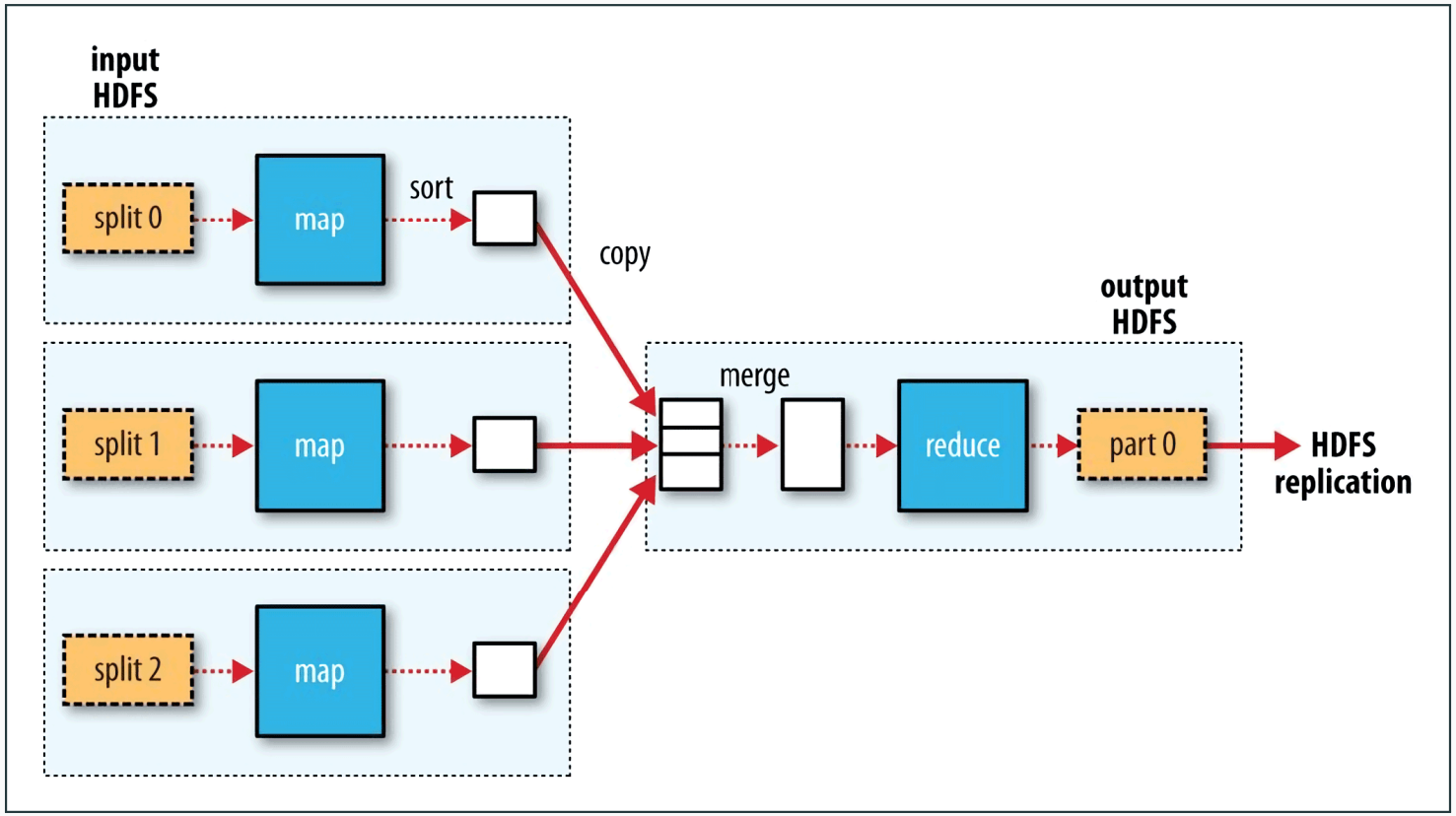

- Il reducer può lavorare su tutti i valori con la stessa chiave, effettuando le elaborazioni necessarie in base al compito da svolgere.
- Input e output sono, anche in questo caso, coppie chiave-valore.
- Nel caso di esempio, possiamo calcolare il massimo per ogni chiave (anno).
- Esempio di output:

```
(2014, 37.0)
(2015, 36.3)
(2016, 41.0)
(2017, 38.0)
```

# MapReduce workflow

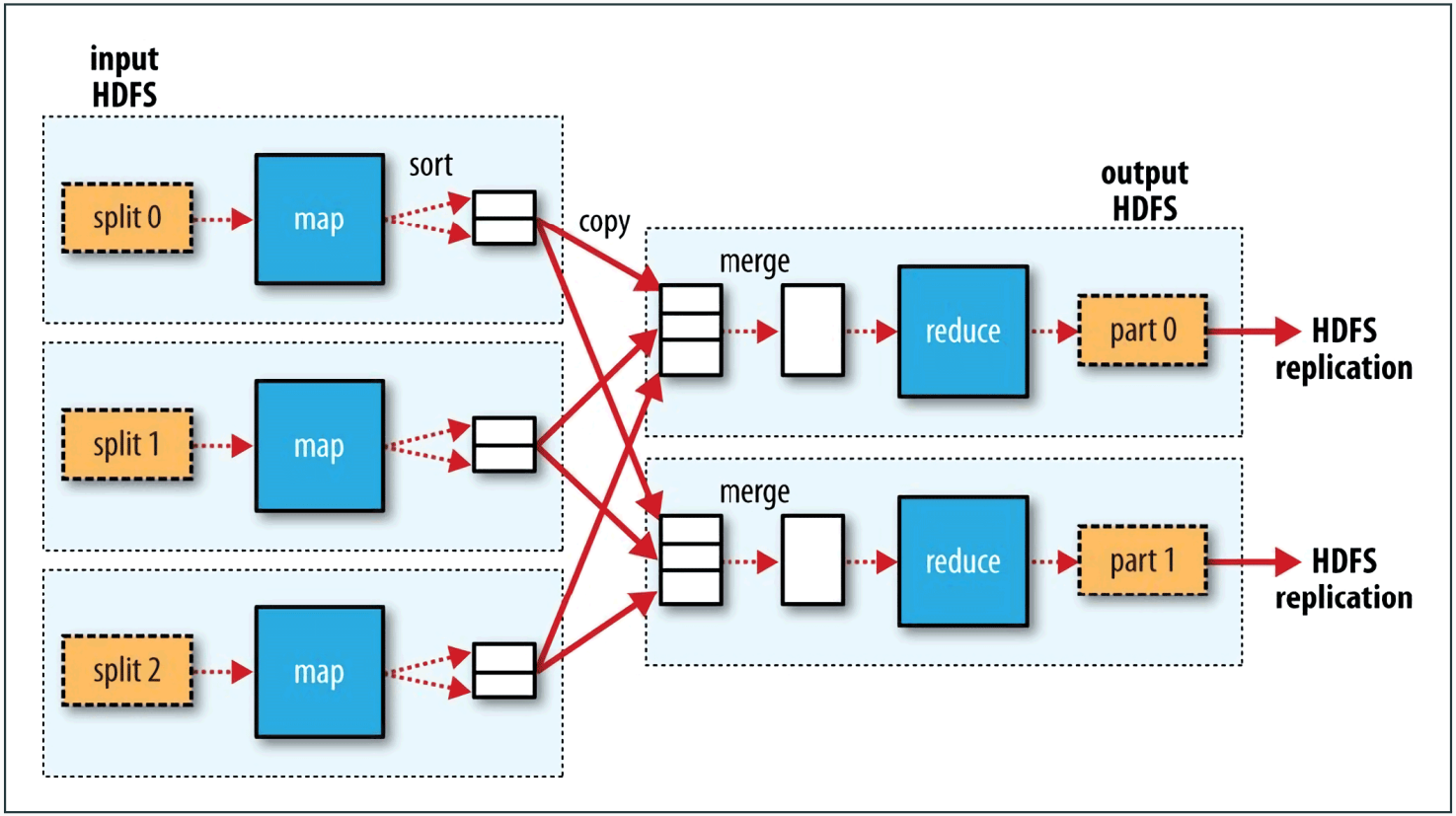


# Map con un reducer



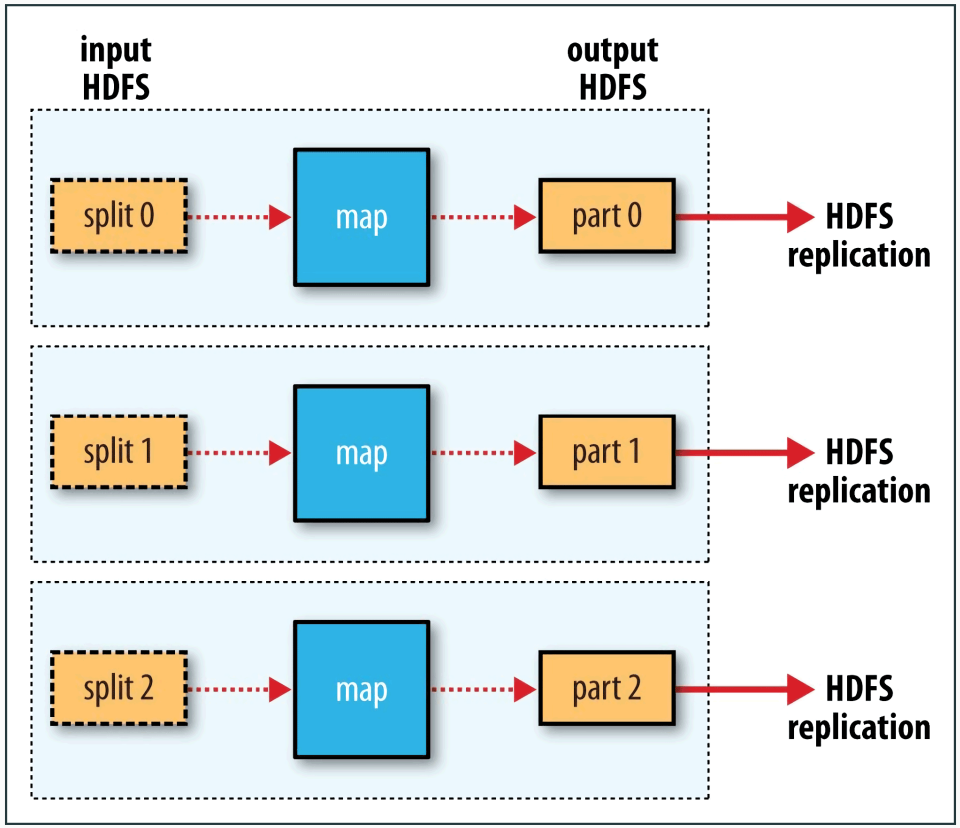

- Utilizzando un solo reducer, avremo un solo file di output.
- Non c'è parallelismo nella fase reduce.
- Paradigma utilizzabile solo se si lavora con pochi dati.

# Map con N reducer




- Se utilizziamo **N** reducer, otterremo **N** file di output.
- Ciascun reducer salva il suo output su un file differente.
- È necessario combinare i risultati ottenuti in qualche modo (ad esempio con un'altra fase MapReduce).

# Map senza reducer

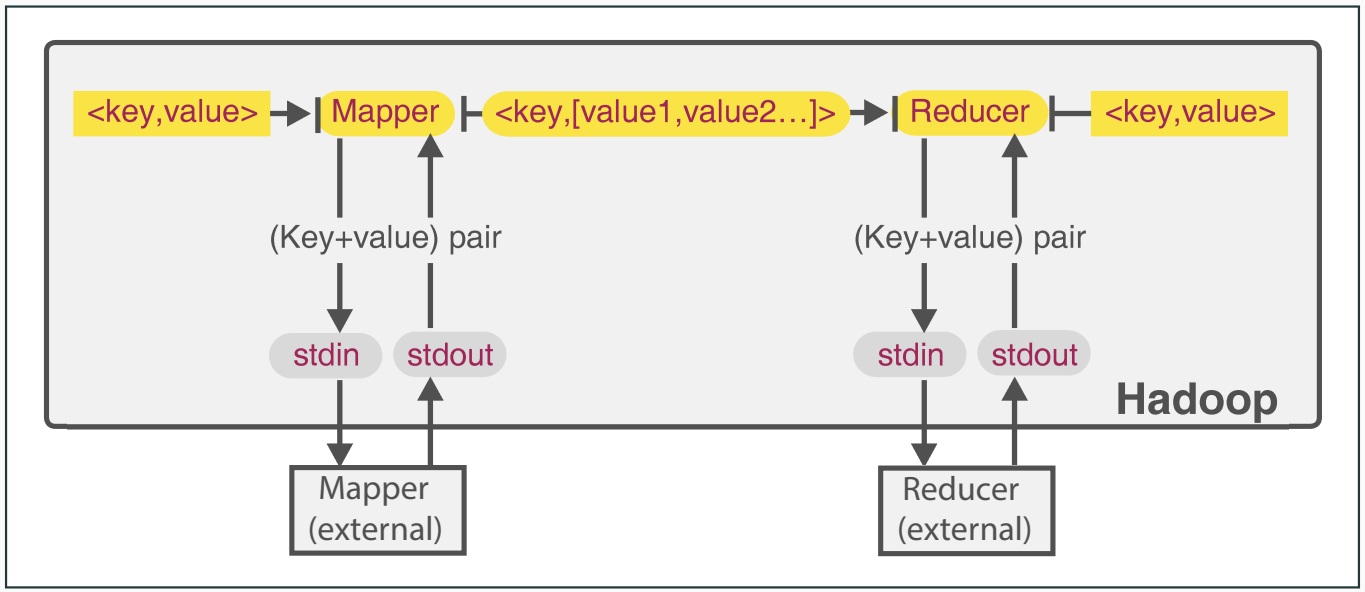



- In certi casi la sola fase di map potrebbe essere sufficiente.
- Ad esempio, potremmo voler solo filtrare o pulire i dati.
- In questo caso avremo un file di output per ciascun mapper.

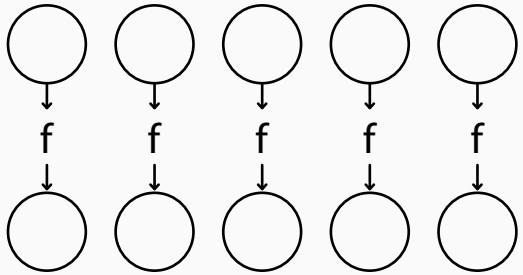
# Hadoop Streaming

# Hadoop Streaming

- Le API standard prevedono che i job MapReduce siano scritti in Java.
- Le API Streaming consentono di utilizzare qualsiasi linguaggio di programmazione (Python, C, script shell...)
- Il programmatore deve fornire due eseguibili (map e reduce) che leggono dati dallo standard input e scrivono sullo standard output.
- Per separare chiavi e valori si utilizza la tabulazione (`\t`).

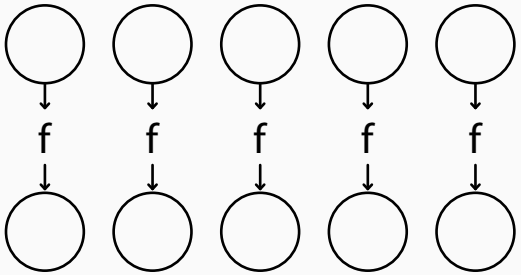


# Mapper... in pratica




- Non sarebbe efficiente richiamare il mapper per ogni singola riga, quindi ogni mapper riceve un certo numero di righe dallo standard input.
- In teoria, ogni riga dovrebbe già rappresentare una coppia chiave-valore...
- In patica, però, l'input dei mapper sono spesso dati non strutturati.
- Uno dei compiti principali dei mapper è quindi quello di pulire i dati e restituire (stampare sullo standard output) coppie chiave-valore.

# Mapper... in pratica




- Non sarebbe efficiente richiamare il mapper per ogni singola riga, quindi ogni mapper riceve un certo numero di righe dallo standard input.
- In teoria, ogni riga dovrebbe già rappresentare una coppia chiave-valore...
- In patica, però, l'input dei mapper sono spesso dati non strutturati.
- Uno dei compiti principali dei mapper è quindi quello di pulire i dati e restituire (stampare sullo standard output) coppie chiave-valore.

Cosa deve fare, praticamente, ciascun mapper, nel nostro esempio?

# Mapper... in pratica

Input:

```
2016-06-09,16:00:00,23.3
2014-07-05,20:00:00,32.5
```

Output:

```
2016    23.3
2014    32.5
```

Il mapper deve...

1. Leggere le righe dallo standard input, una alla volta, fino alla fine.

# Mapper... in pratica

Input:

```
2016-06-09,16:00:00,23.3
2014-07-05,20:00:00,32.5
```

Output:

```
2016    23.3
2014    32.5
```

Il mapper deve...

1. Leggere le righe dallo standard input, una alla volta, fino alla fine.
2. A partire da una riga del tipo "**`2016-06-09,16:00:00,23.3`**", individuare ed estrarre i singoli campi (data, ora, temperatura).

# Mapper... in pratica

Input:

```
2016-06-09,16:00:00,23.3
2014-07-05,20:00:00,32.5
```

Output:

```
2016    23.3
2014    32.5
```

Il mapper deve...

1. Leggere le righe dallo standard input, una alla volta, fino alla fine.
2. A partire da una riga del tipo "**`2016-06-09,16:00:00,23.3`**", individuare ed estrarre i singoli campi (data, ora, temperatura).
3. Estrarre l'anno a partire da una data del tipo "**`2016-06-09`**".

# Mapper... in pratica

Input:

```
2016-06-09,16:00:00,23.3
2014-07-05,20:00:00,32.5
```

Output:

```
2016    23.3
2014    32.5
```

Il mapper deve...

1. Leggere le righe dallo standard input, una alla volta, fino alla fine.
2. A partire da una riga del tipo "**`2016-06-09,16:00:00,23.3`**", individuare ed estrarre i singoli campi (data, ora, temperatura).
3. Estrarre l'anno a partire da una data del tipo "**`2016-06-09`**".
4. Se necessario, convertire il valore di temperatura da stringa a numero.

# Mapper... in pratica

Input:

```
2016-06-09,16:00:00,23.3
2014-07-05,20:00:00,32.5
```

Output:

```
2016    23.3
2014    32.5
```

Il mapper deve...

1. Leggere le righe dallo standard input, una alla volta, fino alla fine.
2. A partire da una riga del tipo "**`2016-06-09,16:00:00,23.3`**", individuare ed estrarre i singoli campi (data, ora, temperatura).
3. Estrarre l'anno a partire da una data del tipo "**`2016-06-09`**".
4. Se necessario, convertire il valore di temperatura da stringa a numero.
5. Verificare che la temperatura sia plausibile (confrontandola con una soglia, ad esempio).

# Mapper... in pratica

Input:

```
2016-06-09,16:00:00,23.3
2014-07-05,20:00:00,32.5
```

Output:

```
2016    23.3
2014    32.5
```

Il mapper deve...

1. Leggere le righe dallo standard input, una alla volta, fino alla fine.
2. A partire da una riga del tipo "**`2016-06-09,16:00:00,23.3`**", individuare ed estrarre i singoli campi (data, ora, temperatura).
3. Estrarre l'anno a partire da una data del tipo "**`2016-06-09`**".
4. Se necessario, convertire il valore di temperatura da stringa a numero.
5. Verificare che la temperatura sia plausibile (confrontandola con una soglia, ad esempio).
6. Stampare sullo standard output le coppie anno-temperatura, separate dal carattere **`\t`**.

# Mapper - Pseudocodice

```
while ci sono dati sullo standard_input:
    line = riga corrente

    date, hour, temperature = campi della stringa line, in base al carattere ','

    year, month, day = campi della stringa date, in base al carattere '-'

    if temperature <= 50:
        print(year + '\t' + temperature)
```

# Reducer... in pratica

- Per motivi di efficienza, anche il reducer riceve come input un certo numero di righe, che non sono separate in base alla chiave.
- Il reducer ha due certezze:
  1. riceverà tutti i valori associati ad una particolare chiave;
  2. le coppie chiave-valore saranno ordinate in base alla chiave.
- Il punto 2 implica che il reducer riceverà tutti i valori associati ad una particolare chiave in modo consecutivo.
- Compito del reducer sarà quindi individuare il passaggio da una chiave a quella successiva.
- Nell'esempio, supponendo che il reducer riceva i valori di temperatura del 2014 e 2016, potrà stampare sullo standard output la temperatura massima del 2014 solo dopo aver letto la prima riga del 2016.

# Reducer... in pratica

Input:

```
2014    24.6
...
2014    18.5
2016    21.9
...
```

Output:

```
2014    36.3
2016    39.1
```

Il reducer deve...

1. Memorizzare l'ultimo anno letto (per individuare il passaggio da un anno all'altro).

# Reducer... in pratica

Input:

```
2014    24.6
...
2014    18.5
2016    21.9
...
```

Output:

```
2014    36.3
2016    39.1
```

Il reducer deve...

1. Memorizzare l'ultimo anno letto (per individuare il passaggio da un anno all'altro).
2. Memorizzare la temperatura massima calcolata fino a questo momento per l'anno corrente.

# Reducer... in pratica

Input:

```
2014    24.6
...
2014    18.5
2016    21.9
...
```

Output:

```
2014    36.3
2016    39.1
```

Il reducer deve...

1. Memorizzare l'ultimo anno letto (per individuare il passaggio da un anno all'altro).
2. Memorizzare la temperatura massima calcolata fino a questo momento per l'anno corrente.
3. Leggere le righe dallo standard input, una alla volta, fino alla fine.

# Reducer... in pratica

Input:

```
2014    24.6
...
2014    18.5
2016    21.9
...
```

Output:

```
2014    36.3
2016    39.1
```

Il reducer deve...

1. Memorizzare l'ultimo anno letto (per individuare il passaggio da un anno all'altro).
2. Memorizzare la temperatura massima calcolata fino a questo momento per l'anno corrente.
3. Leggere le righe dallo standard input, una alla volta, fino alla fine.
4. A partire da una riga del tipo "**`2014    24.6`**", individuare ed estrarre i singoli campi (anno, temperatura).

# Reducer... in pratica

Input:

```
2014    24.6
...
2014    18.5
2016    21.9
```

Output:

```
2014    36.3
2016    39.1
```

5. Se l'anno della riga corrente è uguale all'ultimo letto, aggiornare il massimo per quell'anno.

# Reducer... in pratica

Input:

```
2014    24.6
...
2014    18.5
2016    21.9
```

Output:

```
2014    36.3
2016    39.1
```

5. Se l'anno della riga corrente è uguale all'ultimo letto, aggiornare il massimo per quell'anno.
6. Se, invece, l'anno è cambiato, stampare sullo standard output l'anno precedente e il massimo.

# Reducer - Pseudocodice

```
last_year = NULL
max_value = -999

while ci sono dati sullo standard_input:
    line = riga corrente

    year, temperature = campi della stringa line, in base al carattere '\t'

    # cambio di anno, stampiamo la temperatura massima di last_year
    if last_year is not NULL and last_year != year:
        print(last_year + '\t' + max_value)

        # aggiorniamo l'anno e azzeriamo il massimo
        last_year = year
        max_value = temperature

    else: # altrimenti, l'anno non è cambiato e aggiorniamo solo il massimo
        last_year = year
        max_value = massimo tra max_value e temperature

# abbiamo finito di leggere i dati, ma non abbiamo ancora stampato l'ultimo valore
if last_year is not NULL:
    print(last_year + '\t' + max_value)
```

# Simulare un job MapReduce Streaming

- È facile simulare il comportamento di un job MapReduce Streaming (con un solo mapper e un solo reducer) utilizzando le pipe Unix.

```
$ cat dir_input/* | ./mapper | sort | ./reducer
```

# Simulare un job MapReduce Streaming

- È facile simulare il comportamento di un job MapReduce Streaming (con un solo mapper e un solo reducer) utilizzando le pipe Unix.

```
$ cat dir_input/* | ./mapper | sort | ./reducer
```

- `cat` concatena e restituisce, sullo stdout, il contenuto dei file presenti nella directory `dir_input`.
- Il mapper legge i dati dallo stdin e restituisce coppie chiave-valore sullo stdout.
- `sort` legge i dati dal suo stdin (ovvero dallo stdout del mapper), li ordina in base alla chiave, e li restituisce sullo stdout.
- Il reducer legge le coppie chiave-valore dallo stdin e restituisce nuove coppie chiave-valore.

# Simulare un job MapReduce Streaming - cat

```
$ cat dir_input/*

2008-01-01,00:00:00,13.7
2008-01-01,01:00:00,13.0
2008-01-01,02:00:00,13.0
2008-01-01,03:00:00,13.0
2008-01-01,04:00:00,13.5
2008-01-01,05:00:00,14.0
2008-01-01,06:00:00,13.0
...
2017-12-31,21:00:00,11.0
2017-12-31,22:00:00,11.5
2017-12-31,23:00:00,12.5
```

**Fase 1**

`cat` concatena e restituisce, sullo stdout, il contenuto dei file presenti nella directory `dir_input`.

# Simulare un job MapReduce Streaming - mapper

```
$ cat dir_input/* | ./mapper

2008    13.7
2008    13.0
2008    13.0
2008    13.0
2008    13.5
2008    14.0
2008    13.0
...
2017    11.0
2017    11.5
2017    12.5
```

**Fase 2**

Il mapper legge i dati dallo stdin e restituisce coppie chiave-valore sullo stdout.

# Simulare un job MapReduce Streaming - sort

```
$ cat dir_input/* | ./mapper | sort

2008    10.0
2008    10.0
2008    10.0
2008    10.0
2008    10.0
2008    10.0
2008    10.0
...
2017    9.7
2017    9.8
2017    9.8
```

**Fase 3**

`sort` legge i dati dal suo stdin (ovvero dallo stdout del mapper), li ordina in base alla chiave, e li restituisce sullo stdout.

Nell'esempio i valori erano già ordinati in base alla chiave...

# Simulare un job MapReduce Streaming - reducer

```
$ cat dir_input/* | ./mapper | sort | ./reducer

2008    37.5
2009    35.0
2010    40.7
2011    35.0
2012    38.5
2013    33.5
2014    37.0
2015    36.3
2016    41.0
2017    38.0
```

**Fase 4**

Il reducer legge le coppie chiave-valore dallo stdin e restituisce nuove coppie chiave-valore.

# Mapper in C

```c
#include <stdio.h>

#define LINESIZE 100

int main(void)
{
    char line[LINESIZE];

    while(fgets(line, LINESIZE, stdin)) {
        int year, month, day;
        int hour, minute, second;
        float temperature;

        sscanf(line, "%d-%d-%d,%d:%d:%d,%f", &year, &month, &day, &hour, &minute, &
      second, &temperature);

        if (temperature <= 50) {
            printf("%d\t%.1f\n", year, temperature);
        }
    }

    return 0;
}
```

# Reducer in C - 1

```c
#include <stdio.h>

#define LINESIZE 100

int main(void)
{
    char line[LINESIZE];

    int last_year = -1;
    float max_value = -100;

    while (fgets(line, LINESIZE, stdin)) {
        int year;
        float temperature;
        sscanf(line, "%d\t%f", &year, &temperature);
```

Continua →

# Reducer in C - 2

← Continua

```
        if (last_year != -1 && year != last_year) {
            printf("%d\t%.1f\n", last_year, max_value);

            last_year = year;
            max_value = temperature;
        } else {
            last_year = year;

            if (temperature > max_value) {
                max_value = temperature;
            }
        }
    }

    if (last_year != -1) {
        printf("%d\t%.1f\n", last_year, max_value);
    }

    return 0;
}
```

# Come lanciare un job MapReduce

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -files ./mapper,./reducer \
  -input input_dir \
  -output output_dir \
  -mapper './mapper' \
  -reducer './reducer' \
  -numReduceTasks 3
```

# Come lanciare un job MapReduce

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -files ./mapper,./reducer \
  -input input_dir \
  -output output_dir \
  -mapper './mapper' \
  -reducer './reducer' \
  -numReduceTasks 3
```

Invochiamo l'eseguibile di Hadoop passando il percorso di Hadoop Streaming.

# Come lanciare un job MapReduce

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -files ./mapper,./reducer \
  -input input_dir \
  -output output_dir \
  -mapper './mapper' \
  -reducer './reducer' \
  -numReduceTasks 3
```

L'opzione `-files` copia su tutti i nodi i file specificati. Tutti i mapper/reducer devono poter accedere agli eseguibili/script.

# Come lanciare un job MapReduce

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -files ./mapper,./reducer \
  -input input_dir \
  -output output_dir \
  -mapper './mapper' \
  -reducer './reducer' \
  -numReduceTasks 3
```

Le opzioni `-input` e `-output` servono per indicare le directory di input/output su HDFS.

Se la directory di output esiste già, il job fallirà.

# Come lanciare un job MapReduce

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -files ./mapper,./reducer \
  -input input_dir \
  -output output_dir \
  -mapper './mapper' \
  -reducer './reducer' \
  -numReduceTasks 3
```

Le opzioni `-mapper` e `-reducer` servono per indicare i comandi da eseguire come mapper/reducer.

# Come lanciare un job MapReduce

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -files ./mapper,./reducer \
  -input input_dir \
  -output output_dir \
  -mapper './mapper' \
  -reducer './reducer' \
  -numReduceTasks 3
```

L'opzione `-numReduceTasks` imposta il numero di reducer da utilizzare.

# Come lanciare un job MapReduce

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -files ./mapper,./reducer \
  -input input_dir \
  -output output_dir \
  -mapper './mapper' \
  -reducer './reducer' \
  -numReduceTasks 3
```

```
18/03/09 14:11:38 INFO mapreduce.Job: Running job: job_1520527317571_0004
18/03/09 14:11:43 INFO mapreduce.Job:  map 0% reduce 0%
18/03/09 14:11:48 INFO mapreduce.Job:  map 10% reduce 0%
18/03/09 14:11:50 INFO mapreduce.Job:  map 20% reduce 0%
...
18/03/09 14:12:03 INFO mapreduce.Job:  map 100% reduce 0%
18/03/09 14:12:04 INFO mapreduce.Job:  map 100% reduce 67%
18/03/09 14:12:05 INFO mapreduce.Job:  map 100% reduce 100%
18/03/09 14:12:05 INFO mapreduce.Job: Job job_1520527317571_0004 completed
     successfully
...
```

# Mapper in Python

```
import sys

for line in sys.stdin:
    line = line.strip()

    date, time, temperature = line.split(',')
    year, month, day = date.split('-')

    if float(temperature) <= 50:
        print('{0}\t{1}'.format(year, temperature))
```

# Reducer in Python

```
import sys

last_year = None
max_val = -100.0

for line in sys.stdin:
    line = line.strip()
    year, temperature = line.split('\t')

    if last_year and last_year != year:
        print('{0}\t{1}'.format(last_year, max_val))

        last_year = year
        max_val = float(temperature)
    else:
        last_year = year
        max_val = max(max_val, float(temperature))

if last_year:
    print('{0}\t{1}'.format(last_year, max_val))
```