For dei som vil prøve MapReduce, sjå ”Quickstart to installing tools used in TDT4305“ som er lagt ut på Blackboard

MapReduce

- Programmerings-rammeverk for *parallell og skalerbar prosessering av (veldig) store datasett*
- Mål:
  - Prosessere store filer/datasett (mange PB)
  - Mange noder (mange tusen)
  - Lineær skalering (t.d. 10 x nodar ➔ 10 x yting)
  - Skjule kompleksitet i distribusjon, parallellisering og feiltoleranse for applikasjons-utviklarar
  - Skilje mellom kode for distribuert utføring og applikasjonskode ➔ endring i applikasjonar/algoritmar enkelt
  - Hyllevare (maskiner)

MapReduce

- *Jobb* i MapReduce skildra (og utført) som to separate faser:
  - *Map*-funksjon (også kalla ”mapper”): prosesserer key/value-par og genererer *liste med key/value-par*
    (k1, v1) → List(k2, v2)
  - *Reduce*-funksjon (også kalla ”reducer”): prosesserer alle verdiar med same nøkkel
    (k2, List(v2)) → List(k3, v3)
- Ut-data frå *Map*-fase:
  - Partisjonert basert på k2 slik at alle tuplar med same verdi for k2 ender opp på same Reduce-node (shuffle)
  - Sortert slik at ein effektivt kan utføre Reduce på alle tuplar med same k2

Eksempel: Telje førekomst av termar i dokumentsamling

d1= Deer Bear River
d2= Car Car River
d3= Deer Car Bear

Algoritme (konseptuelt):

Map

1. for kvart dokument *d i D*
2. for kvar term *t* i D
3. skriv ut (*t*, 1) til fil TEMP
4. sorter TEMP på attributt #1 (term)

Reduce

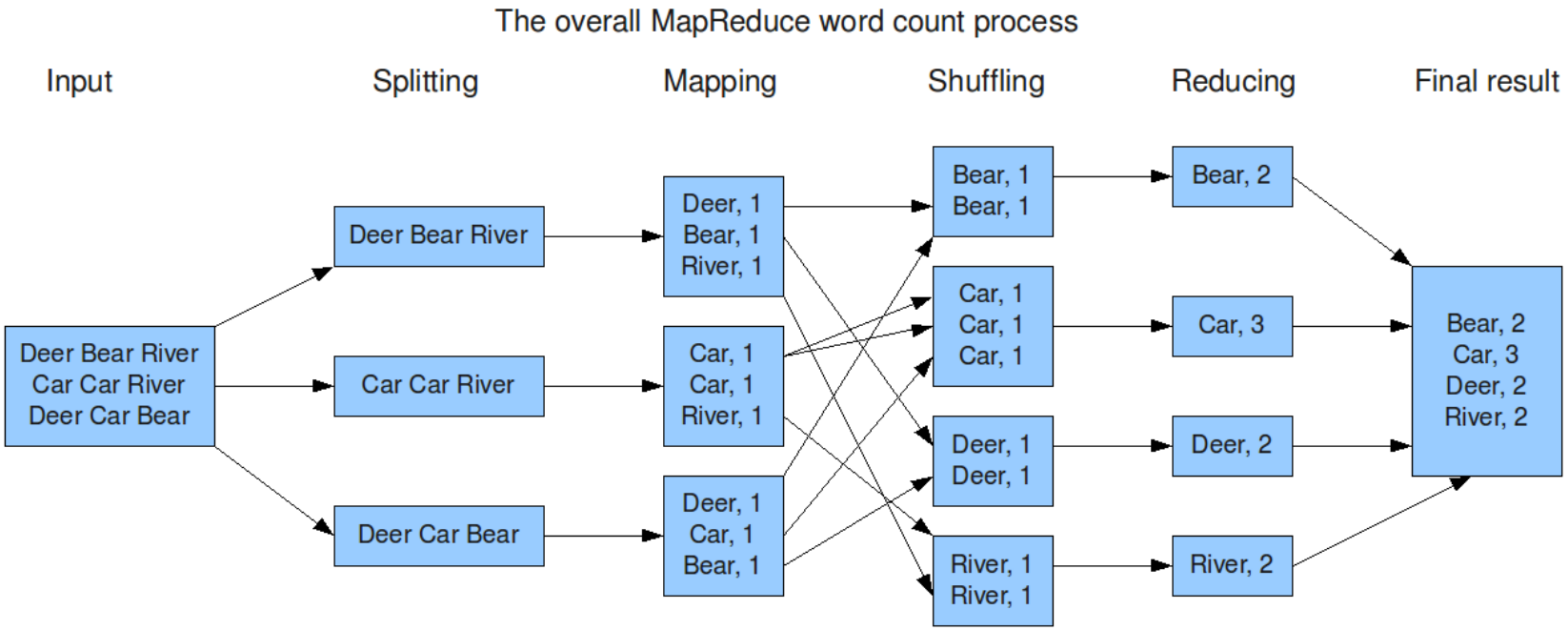
5. for kvar distinkt *t*
6. summér verdiar i attributt #2 (tal på førekomstar) i *s*
7. skriv (t, s)

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

Eksempel: telje førekomst av termar i MapReduce

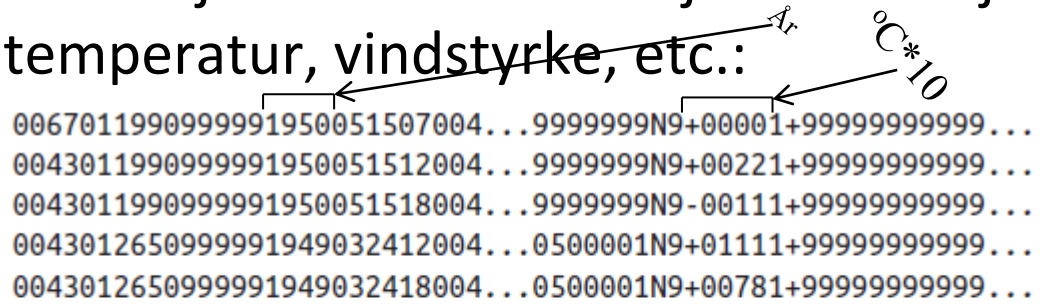


Implementasjon (høgnivå)

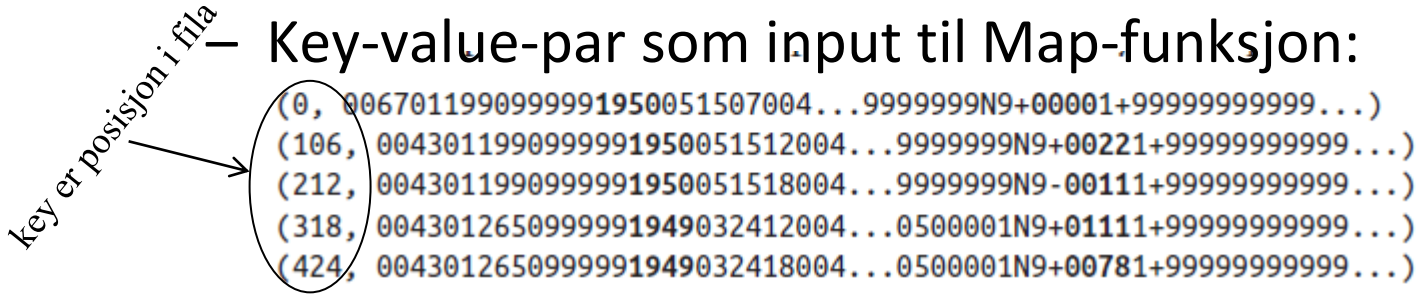
```
mapper (filename, file-contents):
        for each word in file-contents:
                output (word, 1)

reducer (word, values):
        sum = 0
        for each value in values:
                sum = sum + value
        output(word, sum)
```

Eksempel (frå Hadoop-boka)

- Utgangspunkt:
  - Datasett med ver-observasjonar
  - Ei fil for kvar dag for kvar stasjon (> 10K stasjonar)
  - Kvar linje i fil har informasjon om stasjon, data, tid, temperatur, vindstyrke, etc.:



  - Key-value-par som input til Map-funksjon:



  - Ønskjer å finne max-temperatur for kvart år

Dataflyt

- Kun år og temperatur av interesse og ut frå Map som dette:

```
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)
```

- Input til Reduce som dette:

```
(1949, [111, 78])
(1950, [0, 22, -11])
```

- Resultat frå Reduce (max per år):

```
(1949, 111)
(1950, 22)
```

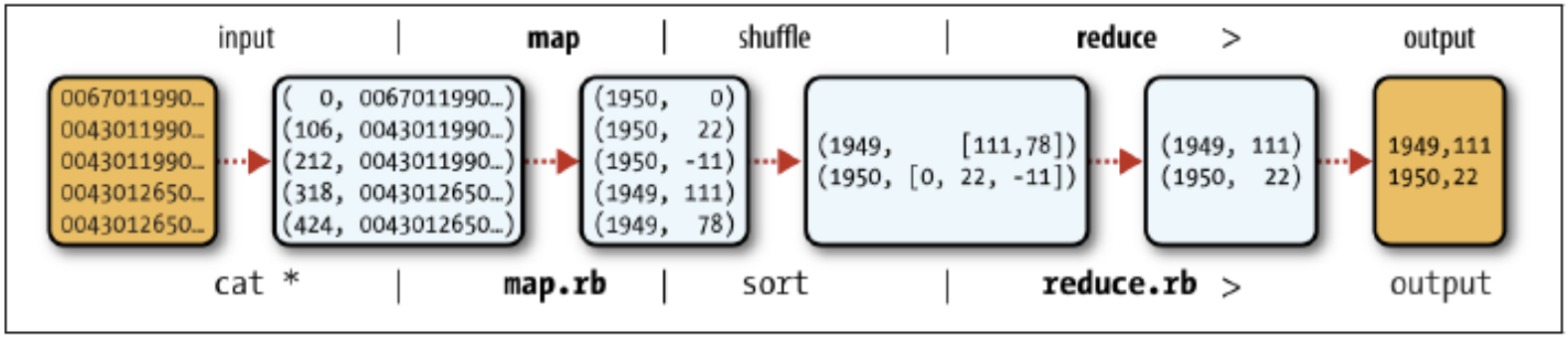
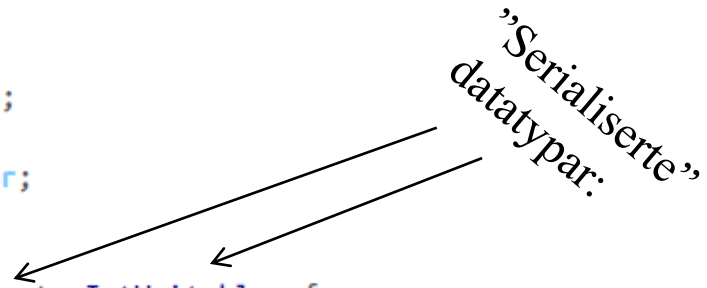


Figure 2-1. MapReduce logical data flow

Mapper i Java

```java
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MaxTemperatureMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

  private static final int MISSING = 9999;

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String line = value.toString();
    String year = line.substring(15, 19);
    int airTemperature;
    if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
      airTemperature = Integer.parseInt(line.substring(88, 92));
    } else {
      airTemperature = Integer.parseInt(line.substring(87, 92));
    }
    String quality = line.substring(92, 93);
    if (airTemperature != MISSING && quality.matches("[01459]")) {
      context.write(new Text(year), new IntWritable(airTemperature));
    }
  }
}
```



Reducer i Java

```java
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {

    int maxValue = Integer.MIN_VALUE;
    for (IntWritable value : values) {
      maxValue = Math.max(maxValue, value.get());
    }
    context.write(key, new IntWritable(maxValue));
  }
}
```

Applikasjon i Java

```java
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperature {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperature <input path> <output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperature.class);
    job.setJobName("Max temperature");

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setReducerClass(MaxTemperatureReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

NB! Til kompilering: hadoop-mapreduce-client-core-3.0.0.jar og hadoop-common-3.0.0.jar eller tilsvarande frå $HADOOP_HOME/share/hadoop

MapReduce dataflyt

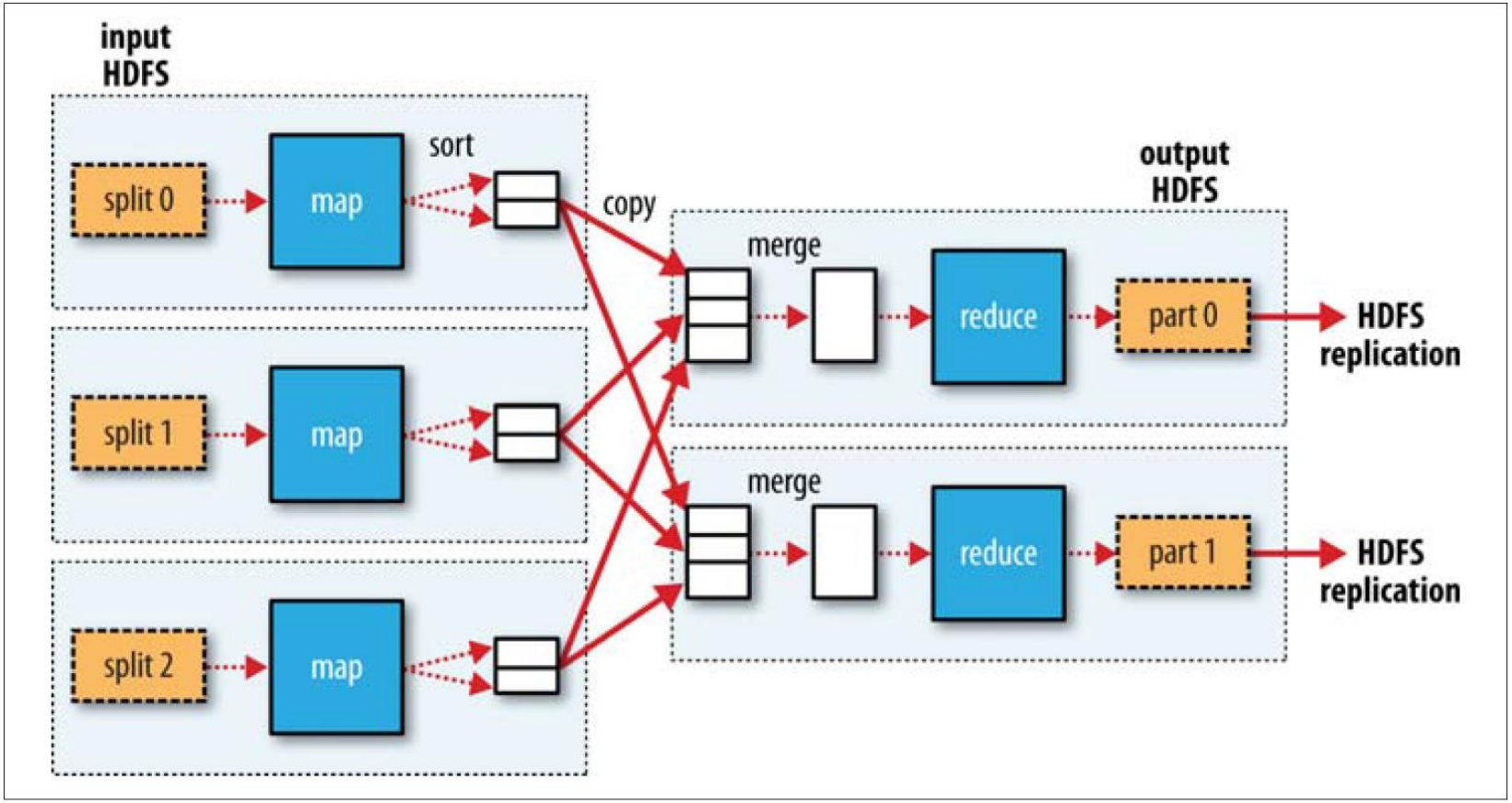
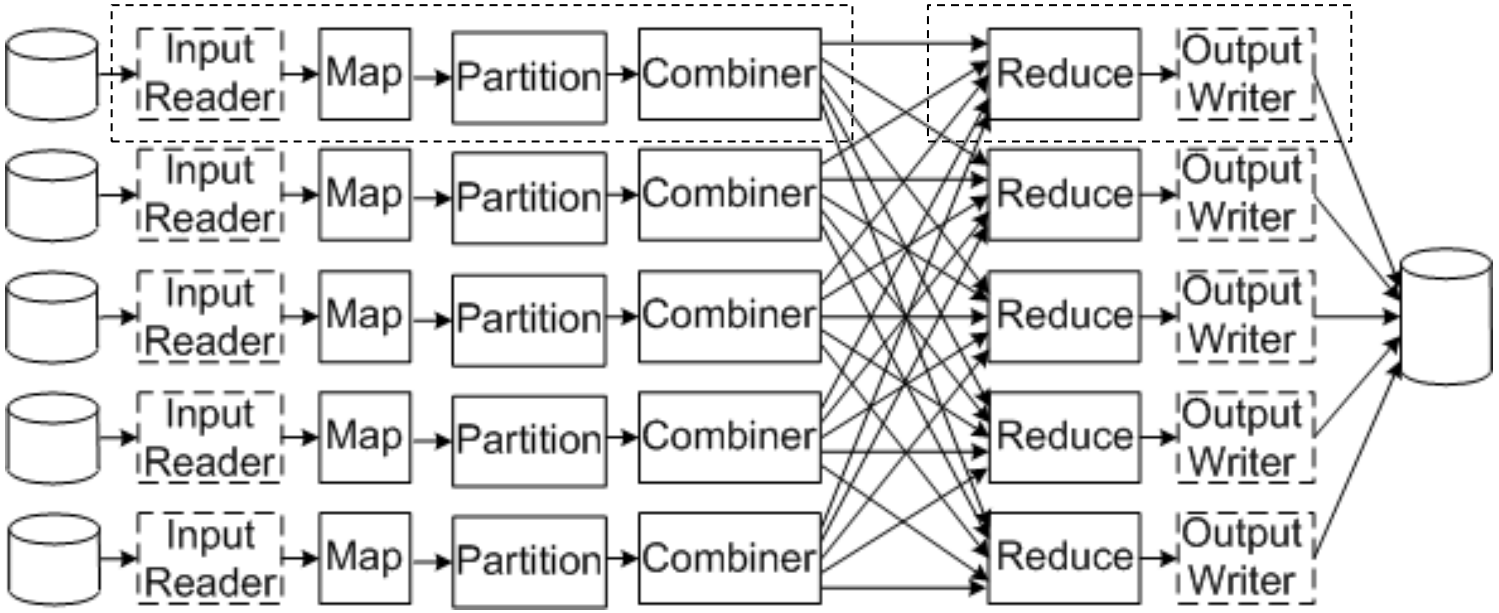


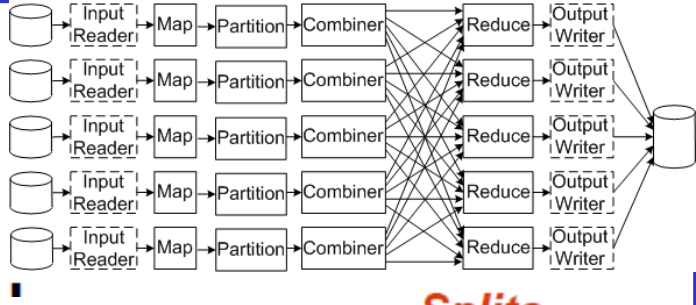
Figure 2-4. MapReduce data flow with multiple reduce tasks

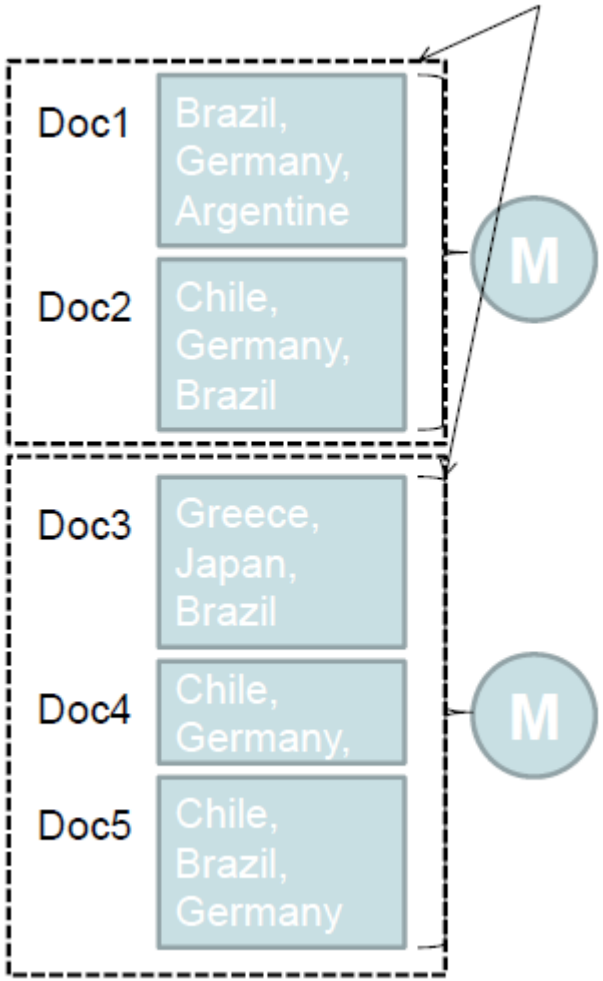
MapReduce/Hadoop: 6 steg



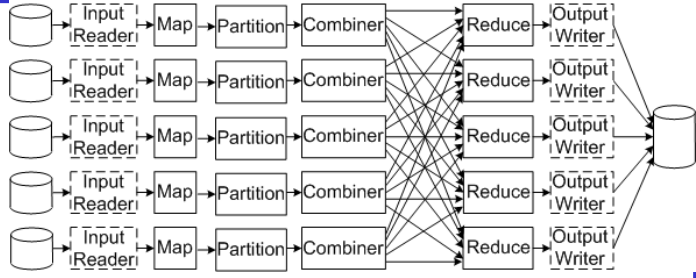
1. Input Reader

- Les data frå filer og transformerer til *key-value*-par
  - Kan også bruke data frå andre kjelder, t.d. databasar
- *Splits* vert generert frå data, *split* er granulariteten til data prosessert av ein *map task*
  - Typisk størrelse på *split* er same som blokk





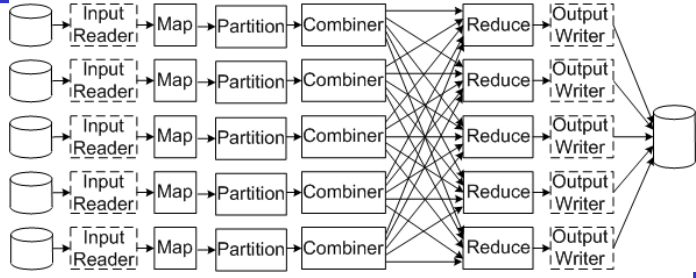
2. Map-funksjon



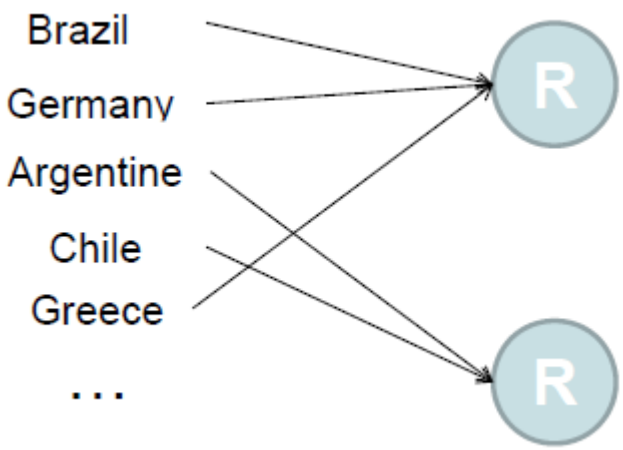
- Input er *key-value*-par frå *Input Reader*
- Køyrer koden i Map-funksjonen på paret
- Produserer nytt *key-value*-par som resultat

- Resultat frå Map-funksjon lagra i buffer i hovudlager, vert skrive til disk (*spill files*) når dette er nesten fullt
- Filene fletta til ei sortert fil

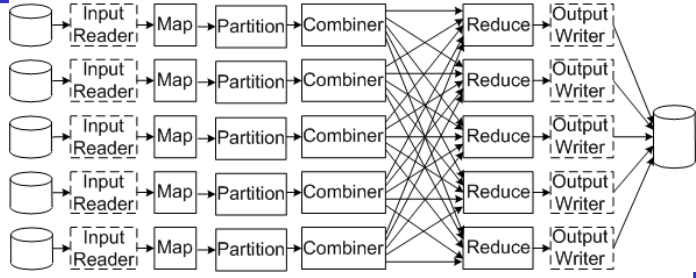
3. Partisjonerings-funksjon



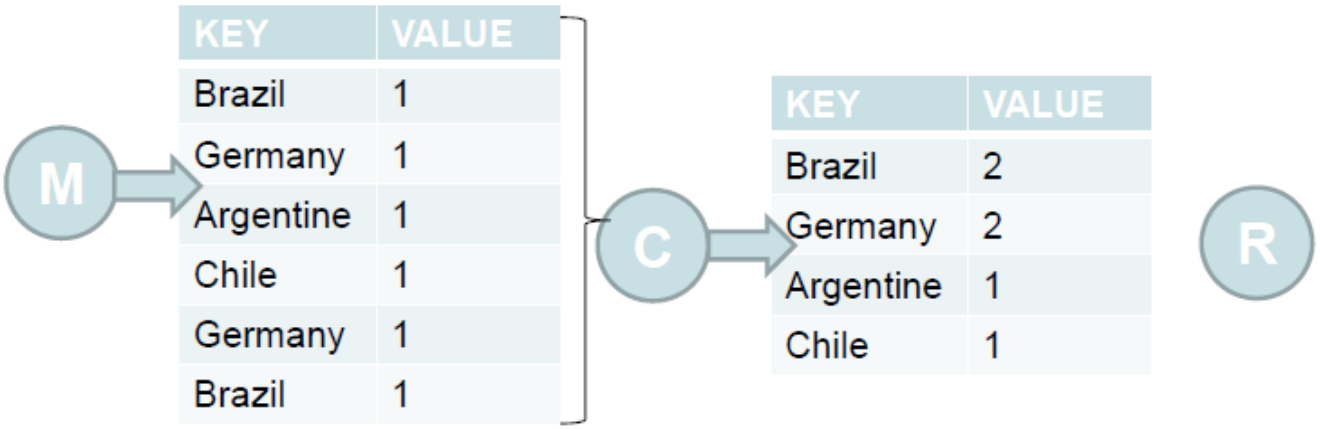
- Default: *hash-funksjon* brukt for å partisjonere resultat av Map-tasks til Reduce-tasks
  - Fungerer vanlegvis bra for *last-balansering*
- Av og til nyttig med annan partisjonerings-funksjon
  - *Brukar-definert*



4. Combiner-funksjon

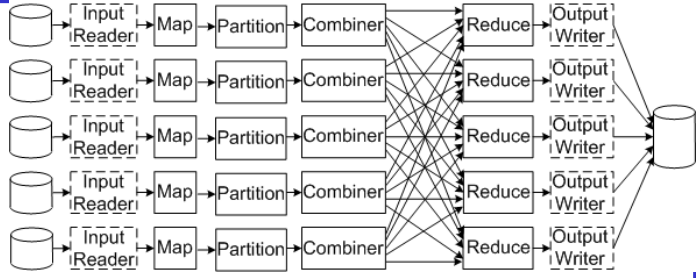


- Bruk av denne er valgfri
- Kan redusere kommunikasjonskostnad
- Bruk når
  - Map-funksjon produserer mange par med same nøkkel
  - Reduce-funksjon er kommutativ (a+b=b+a) og assosiativ (a+b)+c=a+(b+c)
- I eksempelet vil Combiner-funksjon utføre delvis samanslåing slik at *par med same nøkkel vil verte prosessert som gruppe av Reduce-task*



- Ofte vil ein bruke same funksjon til Combiner og Reducer

5. Reduce-funksjon



- Vert kalla *ein gong* for kvar *distinkt nøkkel* og anvendt *på alle verdiar som høyrer til den nøkkelen*
  - Alle par med same nøkkel prosessert som gruppe
- Input til kvar Reduce-task er sortert basert på nøkkel
- Mogleg med eigen-definert *comparator*

Shuffle/sort i meir detalj

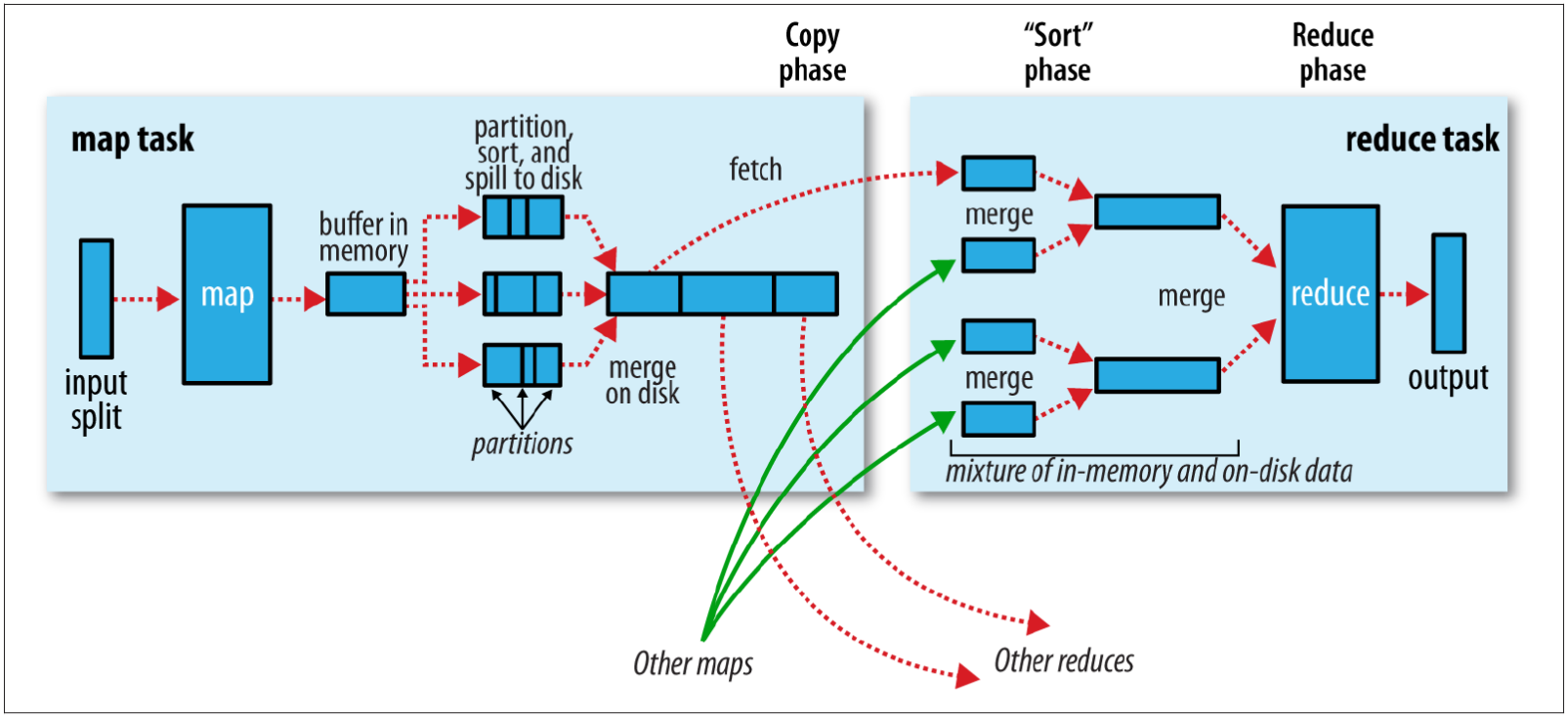


Figure 7-4. Shuffle and sort in MapReduce

- Ein *map task* for kvar *split*
- Prøver å køyre map task på node som lagrar split (HDFS-blokk)

6. Output Writer

- Ansvarleg for å skrive resultat til sekundærlager (disk)
  - Typisk ein katalog, med ei fil for kvar Reducer-task
  - Kan også modifiserast til å lagre i t.d. database

Oppgåve

- Anta at ein har ei fil PersonInfo.txt som inneheld informasjon om namn, alder og løn, dvs. format som dette:

```
Kari 45 450000
Ola 30 200000
Kate 30 500000
Pål 45 550000
```

- Vi ønskjer å finne gjennomsnittsinntekt for kvar alder, dvs. resultat som dette (treng ikkje å vere sortert):

```
45 500000
30 350000
```

- Vis med pseudokode for *mapper og reducer* korleis dette kan gjerast i MapReduce. Anta for enkelheits skyld at value til map er ein post med felta age og salary, dvs. bruk følgjande som utgangspunkt:

```
public void map(key(name), value(age,salary))
public void reduce(key, Iterable values)
```

Mogleg svar

Gjennomsnittsinntekt for kvar alder i MapReduce:

```
public void map(key(name), value(age, salary))
      write(age, salary)

public void reduce(key, Iterable values)
      n = 0
      total = 0
      foreach val in values
            n++
            total = total+ val
      avg = total/n
      write(key, avg)
```

Deklarative språk på toppen av MapReduce: Pig & Hive

```
-- max_temp.pig: Finds the maximum temperature by year
records = LOAD 'input/ncdc/micro-tab/sample.txt'
  AS (year:chararray, temperature:int, quality:int);
filtered_records = FILTER records BY temperature != 9999 AND
  quality IN (0, 1, 4, 5, 9);
grouped_records = GROUP filtered_records BY year;
max_temp = FOREACH grouped_records GENERATE group,
  MAX(filtered_records.temperature);
DUMP max_temp;
```

```
CREATE TABLE records (year STRING, temperature INT, quality INT)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t';
LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'
OVERWRITE INTO TABLE records;
```

```
hive> SELECT year, MAX(temperature)
    > FROM records
    > WHERE temperature != 9999 AND quality IN (0, 1, 4, 5, 9)
    > GROUP BY year;
1949    111
1950    22
```

Generelle svakheiter/begrensningar med MapReduce

- Høg kommunikasjonskostnad
- Støttar dårleg t.d.:
  - Iterasjon
  - ”Early termination” (eks.: top-*k*)
  - Sanntidsprosessering
  - Multi-vegs operatorar (inkl. join)
  - …
- Har gjeve rom til mange forslag til utvidingar, og til nye system som t.d. Spark