MapReduce

Tran Giang Son, tran-giang.son@usth.edu.vn

ICT Department, USTH

What

What?

- A simple programming model that applies to many large-scale computing problem
  - Parallel computation
  - Workload distribution
  - Load balancing
  - Fault tolerance
- Not a language
- Not a library

What?

- Example
  - Count number of students inside USTH building at the moment?
- Traditional way?
- Smart way?
- Smarter way?

What?

- Traditional way

What?

- Traditional way :
  - Place a counting table at the parking entrance of USTH

What?

- Traditional way :
  - Place a counting table at the parking entrance of USTH
  - Announce to everyone to go down there, make a queue, count

What?

- Traditional way :
  - Place a counting table at the parking entrance of USTH
  - Announce to everyone to go down there, make a queue, count

Problem: slow, bottleneck at the counting table

What?

- Smart way

What?

- Smart way : place counting tables at every possible exit of USTH building

What?

- Smart way : place counting tables at every possible exit of USTH building
  - Emergency exit near the museum

What?

- Smart way : place counting tables at every possible exit of USTH building
  - Emergency exit near the museum
  - Parking exits on the ground floor (2)

What?

- Smart way : place counting tables at every possible exit of USTH building
  - Emergency exit near the museum
  - Parking exits on the ground floor (2)
  - Stair exits on the second floor (3)

What?

- Smart way : place counting tables at every possible exit of USTH building
  - Emergency exit near the museum
  - Parking exits on the ground floor (2)
  - Stair exits on the second floor (3)
  - Hit fire alarm

What?

- Smart way : place counting tables at every possible exit of USTH building
  - Emergency exit near the museum
  - Parking exits on the ground floor (2)
  - Stair exits on the second floor (3)
  - Hit fire alarm
- Wait and count

What?

- Smart way : place counting tables at every possible exit of USTH building
  - Emergency exit near the museum
  - Parking exits on the ground floor (2)
  - Stair exits on the second floor (3)
  - Hit fire alarm
- Wait and count
- Still bottleneck at counting tables

What?

- Smarter way

What?

- Smarter way :
  - Come to each classroom

What?

- Smarter way :
  - Come to each classroom
  - Ask the class monitor to count

What?

- Smarter way :
  - Come to each classroom
  - Ask the class monitor to count
  - Aggregate the results in the second time

What?

- Smarter way :
  - Come to each classroom
  - Ask the class monitor to count
  - Aggregate the results in the second time
- Less intrusive, more work done, can be better parallelized

What?

- Two operations
- `map()`: "one to one" transform of each element in a set

$$map_S^f = \{f(x)|x \in S\}$$

- `reduce()`: "many to one" transform of a element set

$$reduce_S^f = f(\{x|x \in S\})$$

map()

- Pre-map()
- Reads data from source
  - Transform

Why

Data explosion

- A lot of data
  - 130+ trillion of webpages (2016)
  - 20KB each
  - 2,600,000+ TB

Data explosion

- Hard drive: 100MB/s sequential read
  - ~824,450,000 **years** to read

Data explosion

- Hard drive: 100MB/s sequential read
  - ~824,450,000 **years** to read
- SSD
  - SATA3 500MB/s sequential read ~ 164,800,000 **years**
  - M.2 3500MB/s sequential read ~ 23,500,000 **years**

Data explosion

- Hard drive: 100MB/s sequential read
  - ~824,450,000 **years** to read
- SSD
  - SATA3 500MB/s sequential read ~ 164,800,000 **years**
  - M.2 3500MB/s sequential read ~ 23,500,000 **years**
- Processing this data
  - Sorting / Searching / Indexing / Classification

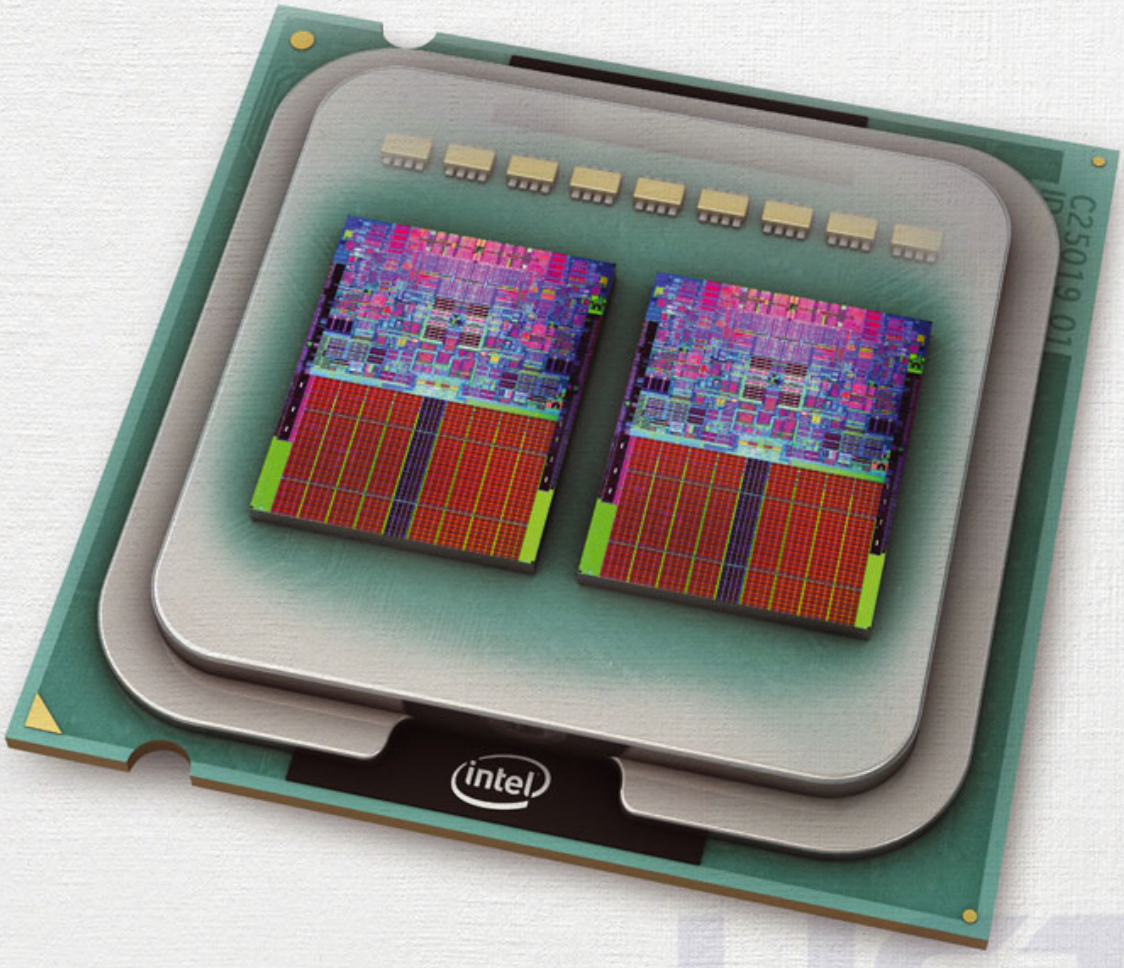
Why MapReduce?

- Traditional programming is serial
- Break processing into independent batches
- Process concurrently
- Aggregate result

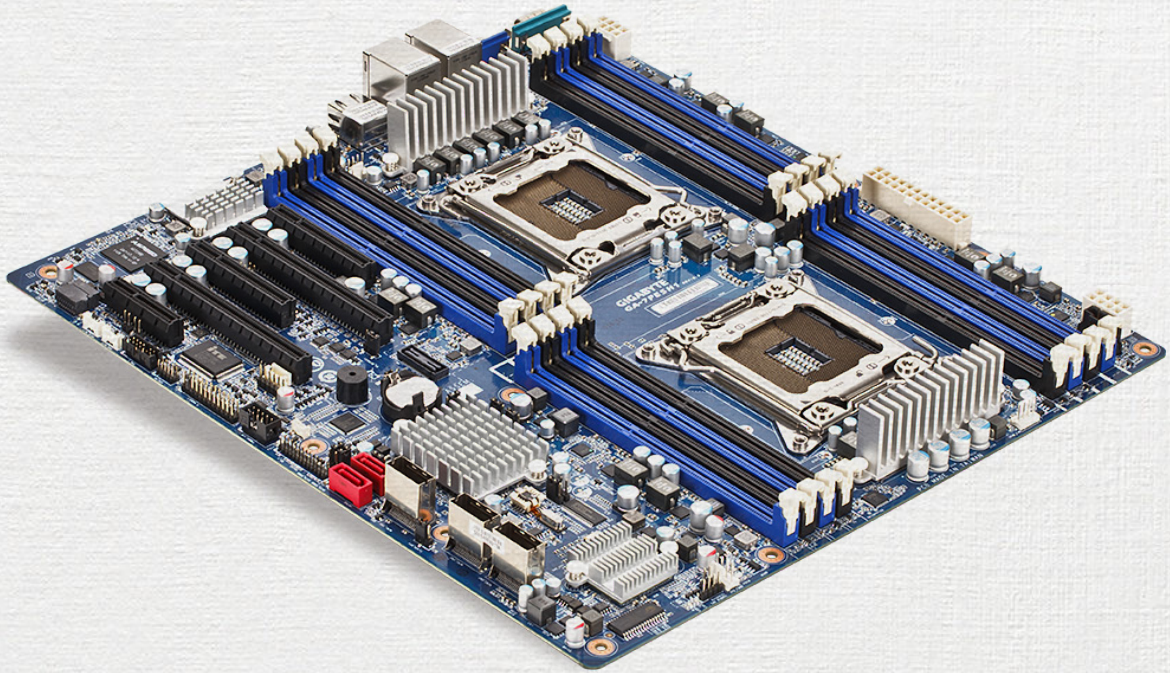
Parallelization

- Multi-core
- Multi-CPU
- Cluster
- Grid

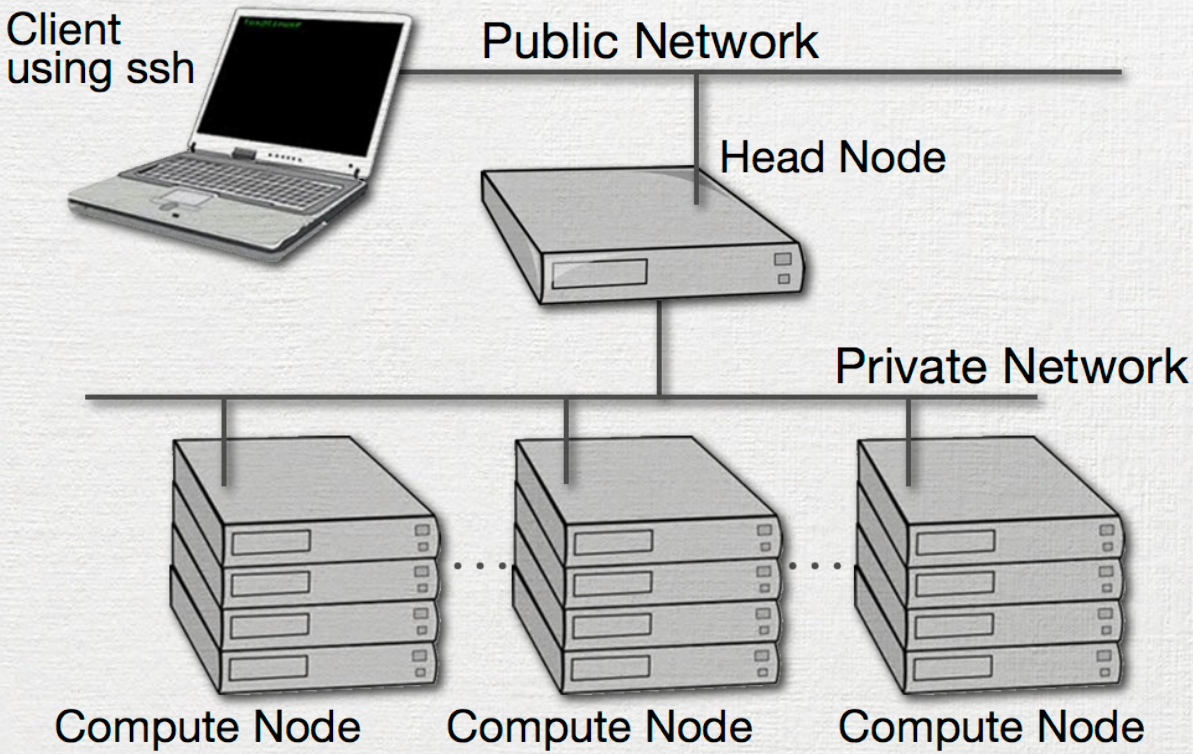
Parallelization



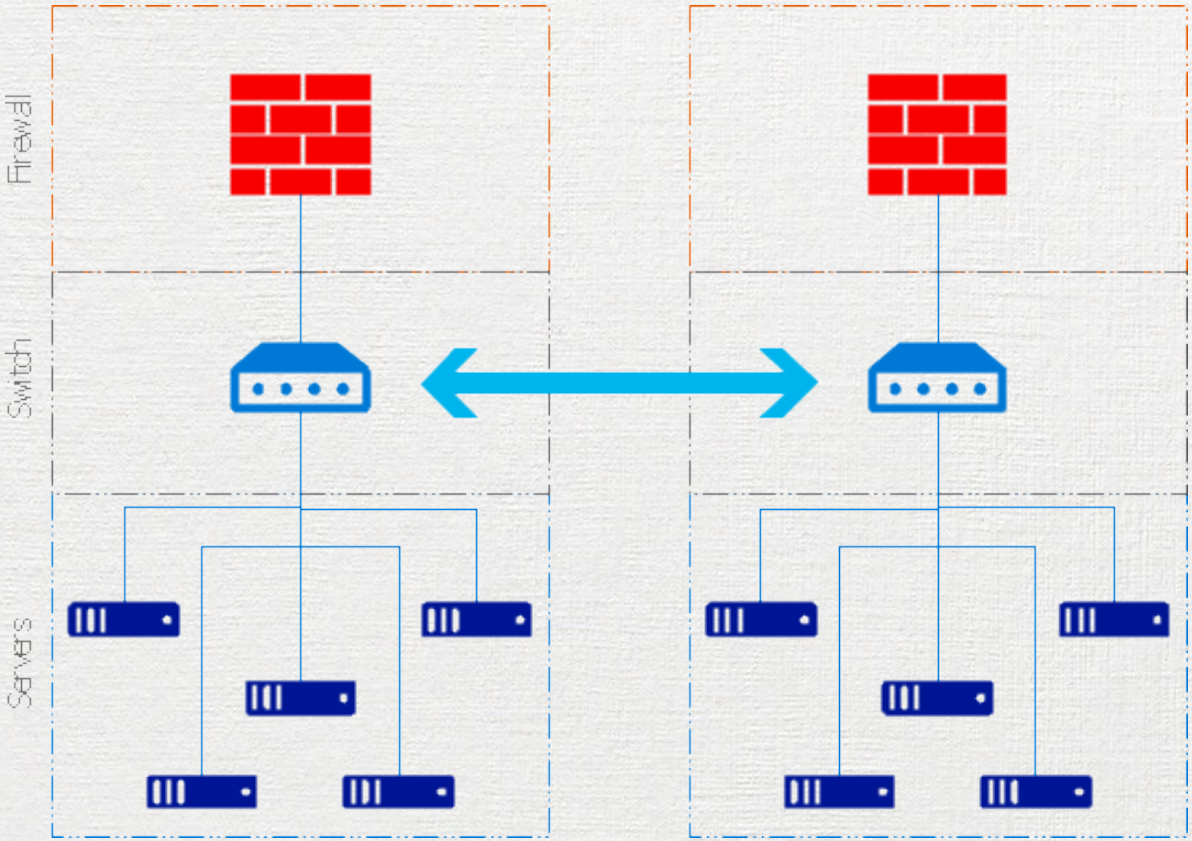
Parallelization



Parallelization



Parallelization



Parallelization

Parallelization

Key	Value
Name	Sunway Taihulight
Nodes	40,960
CPU	SW26010, 256 cores 1.45GHz/node
Cores	10,649,600
Memory	1.31PB (1310TB)
Storage	20PB (20000TB)
Peak	125 PFLOPS
Linpack	93.01 PFLOPS
Power	15MW
Location	National Supercomputer Center, Wuxi, China
Active	June 2016

Why MapReduce?

Challenges:

- Breaking problem into smaller task
- Assigning tasks to machines?
- Partitioning and distributing data?
- Sharing intermediate data?
- Coordinating synchronization? Scheduling? Fault-tolerance?

Why MapReduce?

- Scale "out", not scale "up"
  - E.g. more workers, not more levels of management
- Failure are common
- Process data sequentially and not randomly

How

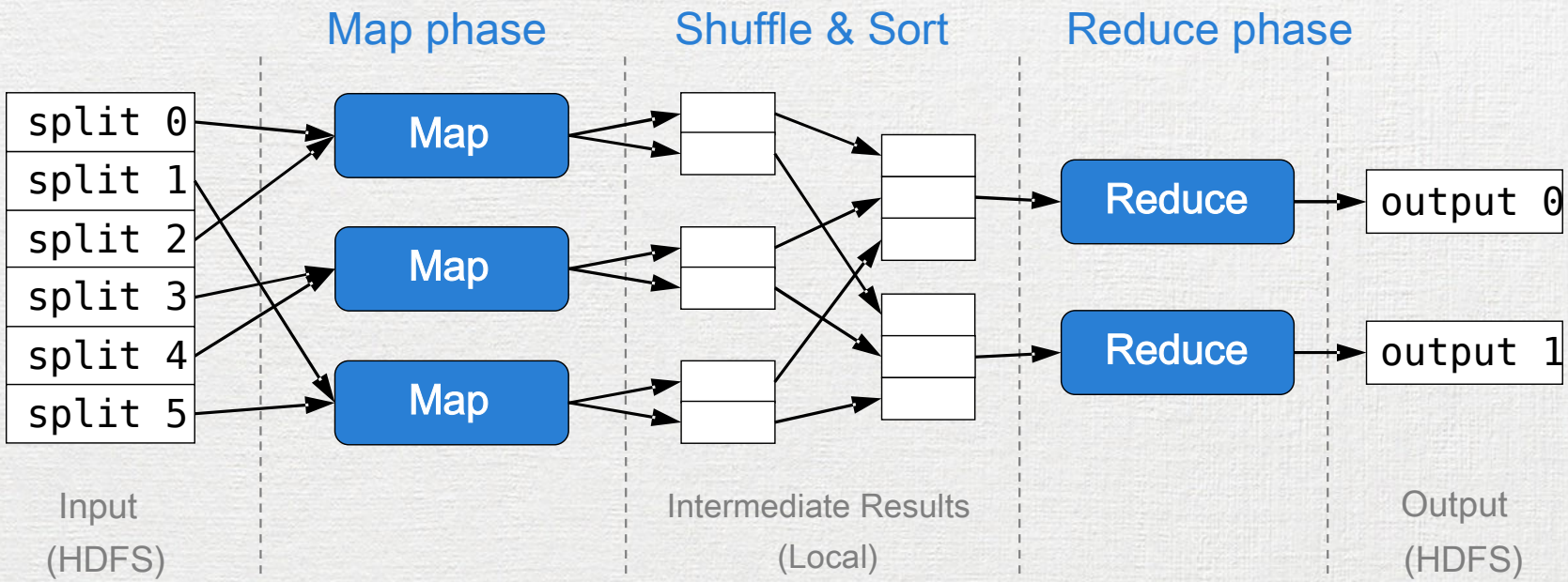
Implementations

- Google
  - Internal
  - Proprietary
- Apache Hadoop MapReduce
  - Most common open source implementation
- Amazon Elastic MapReduce
  - On EC2

Who does what?

- Implement two methods
  - `map()`: `Mapper`
  - `reduce()`: `Reducer`

MapReduce architecture



Execution Framework

- The execution framework (runtime) handles everything else
  - Scheduling: who does `map()`? who does `reduce()`?
  - Data distribution: move data to processes (worker)
  - Synchronization: gathers, sorts,
  - Fault-tolerance: detects failure, restarts
  - Distributed file system

Who does what?

- A "master" controls execution of "slaves"
- Mappers are put near their input block
  - Minimize network usage
- Mappers persist outputs to disk before passing to producer
  - For fault tolerance

Fault Tolerance

- Task crashes
  - Retry on other node
    - `map()`?

Fault Tolerance

- Task crashes
  - Retry on other node
    - `map()`? no deps

Fault Tolerance

- Task crashes
  - Retry on other node
    - `map()`? no deps
    - `reduce()`?

Fault Tolerance

- Task crashes
  - Retry on other node
    - `map()`? no deps
    - `reduce()`? saved on disk

Fault Tolerance

- Task crashes
  - Retry on other node
    - `map()`? no deps
    - `reduce()`? saved on disk
- Important: Task independence

Fault Tolerance

- Node crashes
  - Start tasks on a new node
    - `map()`?

Fault Tolerance

- Node crashes
  - Start tasks on a new node
    - `map()`? restart

Fault Tolerance

- Node crashes
  - Start tasks on a new node
    - `map()`? restart
    - `reduce()`?

Fault Tolerance

- Node crashes
  - Start tasks on a new node
    - `map()`? restart
    - `reduce()`? nothing else

Fault Tolerance

- Task becomes slow
  - Launch same task on another node
  - Use result of whoever finishes first
  - Kill the second one
- Popular in large cluster

Extras

- Extra optional supporting functions
  - `partition()`: divide key space for parallelization
  - `combine()`: mini reducers to combine after map
- Barriers

Example: Word Count

- **The** classic example for MapReduce
- Input: a large text file
- Output : number of occurrence of each word

Example: Word Count

- `map()`: count occurence of word in a single line

1. three witches watch three
   swatch watches

Example: Word Count

- `map()`: count occurence of word in a single line

1. three witches watch three swatch watches

```
<three, 1>
<witches, 1>
<watch, 1>
<three, 1>
<swatch, 1>
<watches, 1>
```

Example: Word Count

- `map()`: count occurence of word in a single line

2. which witch watches which
   swatch watch

Example: Word Count

- `map()`: count occurence of word in a single line

2. which witch watches which swatch watch

```
<which, 1>
<witch, 1>
<watches, 1>
<which, 1>
<swatch, 1>
<watch, 1>
```

Example: Word Count

- `map()`: count occurence of word in a single line

```
<three, 1>
<witches, 1>
<watch, 1>
<three, 1>
<swatch, 1>
<watches, 1>
<which, 1>
<witch, 1>
<watches, 1>
<which, 1>
<swatch, 1>
<watch, 1>
```

Example: Word Count

- Group pairs that have same K

```
<three, 1>
<witches, 1>
<watch, 1>
<three, 1>
<swatch, 1>
<watches, 1>
<which, 1>
<witch, 1>
<watches, 1>
<which, 1>
<swatch, 1>
<watch, 1>
```

Example: Word Count

- Group pairs that have same K

```
<three, 1>
<witches, 1>
<watch, 1>
<three, 1>
<swatch, 1>
<watches, 1>
<which, 1>
<witch, 1>
<watches, 1>
<which, 1>
<swatch, 1>
<watch, 1>
```

```
<three, 1>
<three, 1>
<witches, 1>
<watch, 1>
<watch, 1>
<swatch, 1>
<swatch, 1>
<watches, 1>
<watches, 1>
<which, 1>
<which, 1>
<witch, 1>
```

Example: Word Count

- `reduce()`: combine occurence of word in a single line

```
<three, 1>
<three, 1>
<witches, 1>
<watch, 1>
<watch, 1>
<swatch, 1>
<swatch, 1>
<watches, 1>
<watches, 1>
<which, 1>
<which, 1>
<witch, 1>
```

```
<three, 2>
<witches, 1>
<watch, 2>
<swatch, 2>
<watches, 2>
<which, 2>
<witch, 1>
```

Example: Word Count

1. three
   witches
   watch three
   swatch
   watches
2. which
   witch
   watches
   which
   swatch
   watch

Example: Word Count

1. three
   witches
   watch three
   swatch
   watches
2. which
   witch
   watches
   which
   swatch
   watch

```
<three, 1>
<witches, 1>
<watch, 1>
<three, 1>
<swatch, 1>
<watches, 1>

<which, 1>
<witch, 1>
<watches, 1>
<which, 1>
<swatch, 1>
<watch, 1>
```

```
<three, 1>
<three, 1>
<witches, 1>
<watch, 1>
<watch, 1>
<swatch, 1>
<swatch, 1>
<watches, 1>
<watches, 1>
<which, 1>
<which, 1>
<witch, 1>
```

Example: Word Count

1. three
   witches
   watch three
   swatch
   watches
2. which
   witch
   watches
   which
   swatch
   watch

```
<three, 1>
<witches, 1>
<watch, 1>
<three, 1>
<swatch, 1>
<watches, 1>

<which, 1>
<witch, 1>
<watches, 1>
<which, 1>
<swatch, 1>
<watch, 1>
```

```
<three, 1>
<three, 1>
<witches, 1>
<watch, 1>
<watch, 1>
<swatch, 1>
<swatch, 1>
<watches, 1>
<watches, 1>
<which, 1>
<which, 1>
<witch, 1>
```

```
<three, 2>
<witches, 1>
<watch, 2>
<swatch, 2>
<watches, 2>
<which, 2>
<witch, 1>
```

Example: Word Count

Easy?

Example: Word Count Extra

three swiss witch-bitches, which wished to be switched swiss witch-bitches, watch three swiss swatch watch switches. which swiss witch-bitch, which wishes to be a switched witch-bitch, wishes to watch which swiss swatch watch switch?

Example: Word Count Extra

three swiss witch-bitches, which wished to be switched swiss witch-bitches, watch three swiss swatch watch switches. which swiss witch-bitch, which wishes to be a switched witch-bitch, wishes to watch which swiss swatch watch switch?

```
<swiss, 5>
<witch, 4>
<watch, 4>
<three, 2>
<bitches, 2>
<switched, 2>
<swatch, 2>
<bitch, 2>
<wishes, 2>
<wished, 1>
```

Practical work 4: Word Count

- Create a new directory named «WordCount»
- Use any MapReduce framework of your choice to implement Word Count example
  - Java is OK
  - C/C++ is still preferred
    - No MapReduce framework for C/C++ at the moment
    - Invent yourself

Practical work 4: Word Count

- Write a short report in LaTeX:
  - Name it « 04.word.count.tex »
  - Why you chose your specific MapReduce implementation
  - How your `Mapper` and `Reducer` work. Figure.
  - Who does what.
- Work in your group, in parallel
- Push your report to corresponding forked Github repository

Practical work 5: The Longest Path

- Use any MapReduce framework of your choice to implement LongestPath toy project
  - Input: set of files, one for each of your laptops
    - Each line contain one full path of a file
    - `find /`
  - Output: longest path(s)
- Write a short report in LaTeX:
  - Name it « 05.word.count.tex »
  - How your `Mapper` and `Reducer` work. Figure.
  - Who does what.
- Work in your group, in parallel
- Push your report to corresponding forked Github repository