Spark

Distributed Processing

- Designs for large-scale data processing
- Functional programming
- Map-reduce
- RDD's
- Key-value

Large-scale data processing

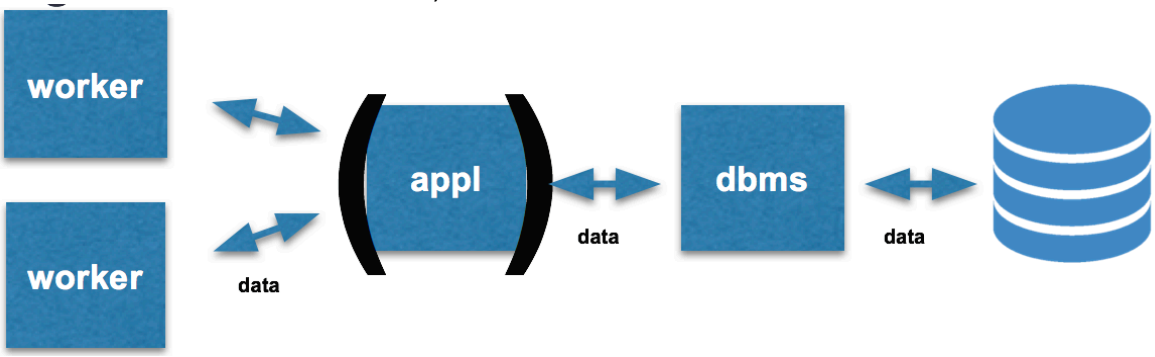
- **Single processor:**
  - Central data repository
  - Bring the data to the code

  - Bottleneck: single processor, data traffic
  - Scale-up expensive

Dit is een typische client-server architectuur waarbij de data van de server wordt opgehaald.

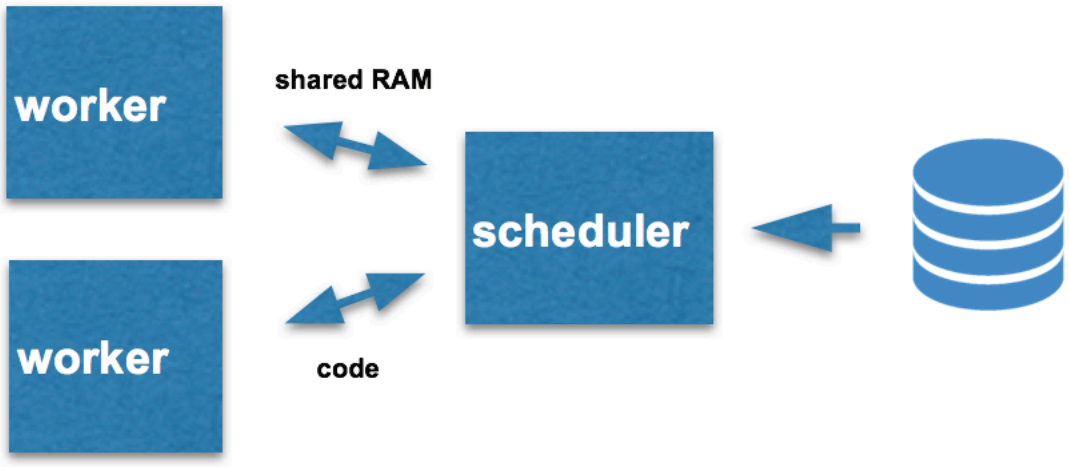
Scale-up, is het neerzetten van een snellere machine waar de applicatie op draait, maar over het algemeen worden snellere machines onevenredig veel duurder.

- **Parallel computing (traditional):**
    - Central data repository
    - Multiple processors
    - Bring the data to the code
      + Task dividable
      - Data traffic over network, central data



Parallel computing kunnen we verschillende workers aan de slag zetten om gezamelijk te werken aan taken. Voor het vinden van priemgetallen werkt het goed, maar ook voor het vinden van hashcodes voor een blockchain. Data komt, omdat de hoeveelheid dataverkeer klein is ten opzichte van de benodigde verwerkingscapaciteit.

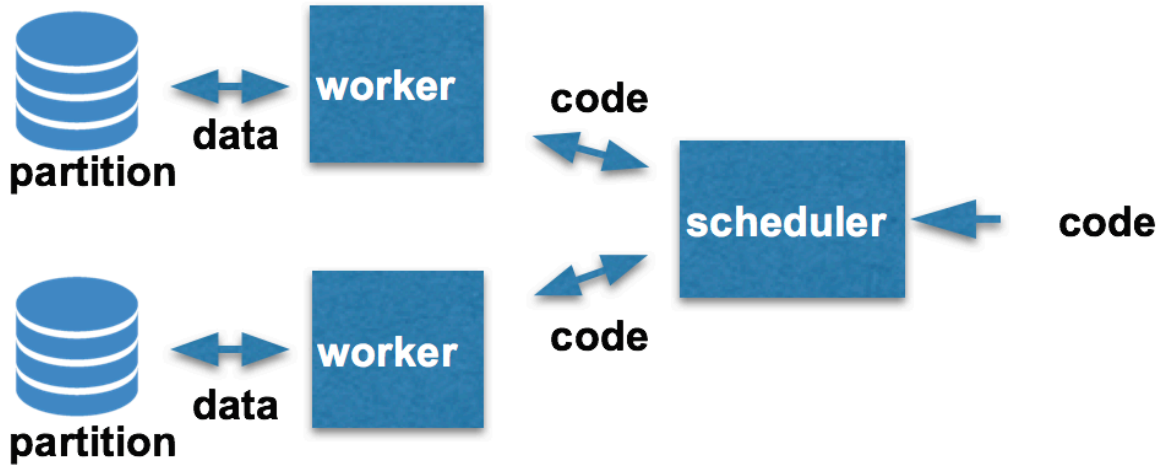
- **Multiprocessing**
    - Shared RAM
    - Multiple cores
      + Exchange data in RAM
      - Limited cores, memory, I/O, expensive, less robust



Multiprocessing wordt gebruikt in iedere computer, waardoor je meerdere cores kan benutten, zowel om te multitasken. Dit wordt ook gebruikt om een bepaalde taak sneller te laten werken (bijv. Video bewerking).

Een groot voodeel is dat alle uitwisseling tussen de machines in RAM gebeurt
Nadeel is wel dat het relatief duur, gelimiteerd en minder robuust is.

- **Distributed computing + MapReduce:**
    - Distributed data
    - Bring (map) the code to the data



- Consistency, availability, partition tolerance (CAP)
+ Cheap (scale-'out' not 'up'), limitless

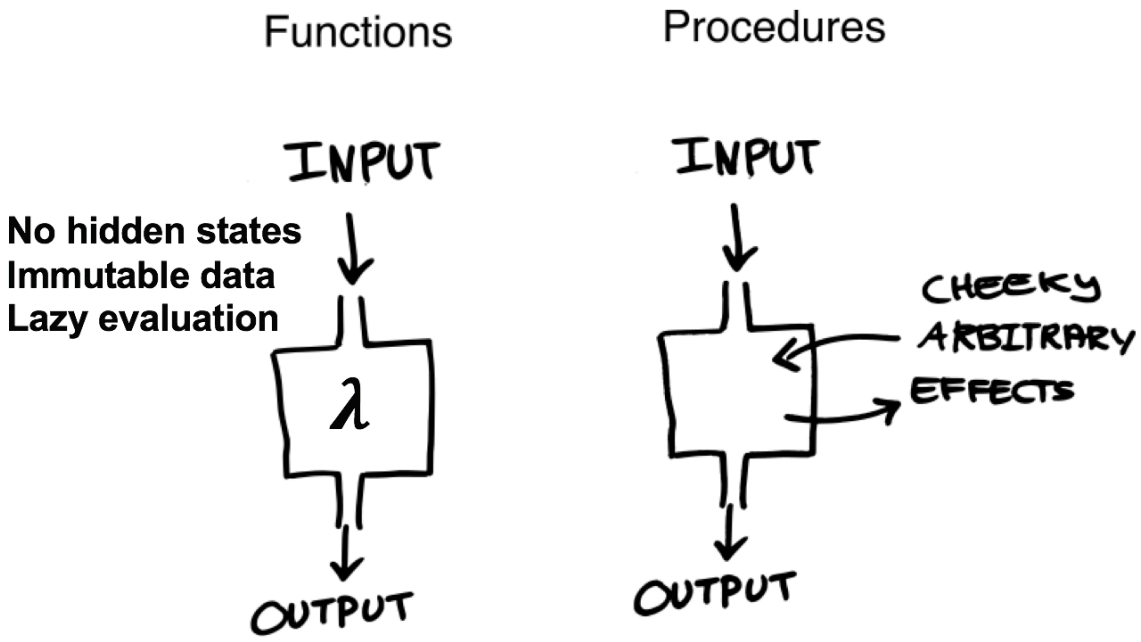
Distributed computing, netwerk van computers die hun acties coördineren door messages uit te wisselen.

MapReduce, programming model waarin operaties op data worden beschreven, die ofwel onafhankelijk van elkaar kunnen worden uitgevoerd (map) of resultaten aggregeren (reduce).

How to approach data engineering challenges for large-scale datasets:

- Computing is becoming increasingly data intensive
- The solution is a "scale-out" architecture
- Bring the computations to the data, rather than data to the computations
- Start with the 20 key questions
- Go from working version to working version

Functional Programming



- High-order functions:
  - Take one or more functions as a parameter
  - Returns a function as its result
- Map, perform a function on every element
- Reduce, aggregate elements

Aan een higher-order function is het mogelijk om een functie als argument meegeven. De map functie, die past een andere functie toe op elk element in de lijst. Het elegante is, dat we door gebruik van de map functie alleen maar een functie hoeven te geven die elk element bewerkt. Deze mate van abstractie maakt het mogelijk voor distributed computing frameworks, om de map niet lokaal maar distributed uit te voeren.

Belangrijk kenmerk van de Map is dat ieder element in isolation kan worden bewerkt.

Map-Reduce

$$\mathbf{map}(\ f,\ [v_0,\ v_1,\ \ldots,\ v_{n-1}\ ]) \Rightarrow [\ f(v_0),\ f(v_1),\ \ldots,\ f(v_{n-1})\ ]$$

$$raise(x) \Rightarrow 1.05 * x$$

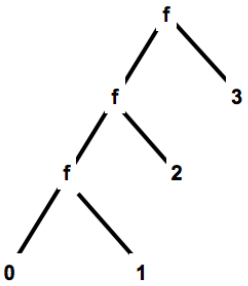
$$\mathbf{map}(\ raise,\ [100,\ 200\ ]) \Rightarrow [\ raise(100),\ raise(200)\ ]$$

- Reduce (fold): aggregate elements

$$sum(x, y) \Rightarrow x + y$$

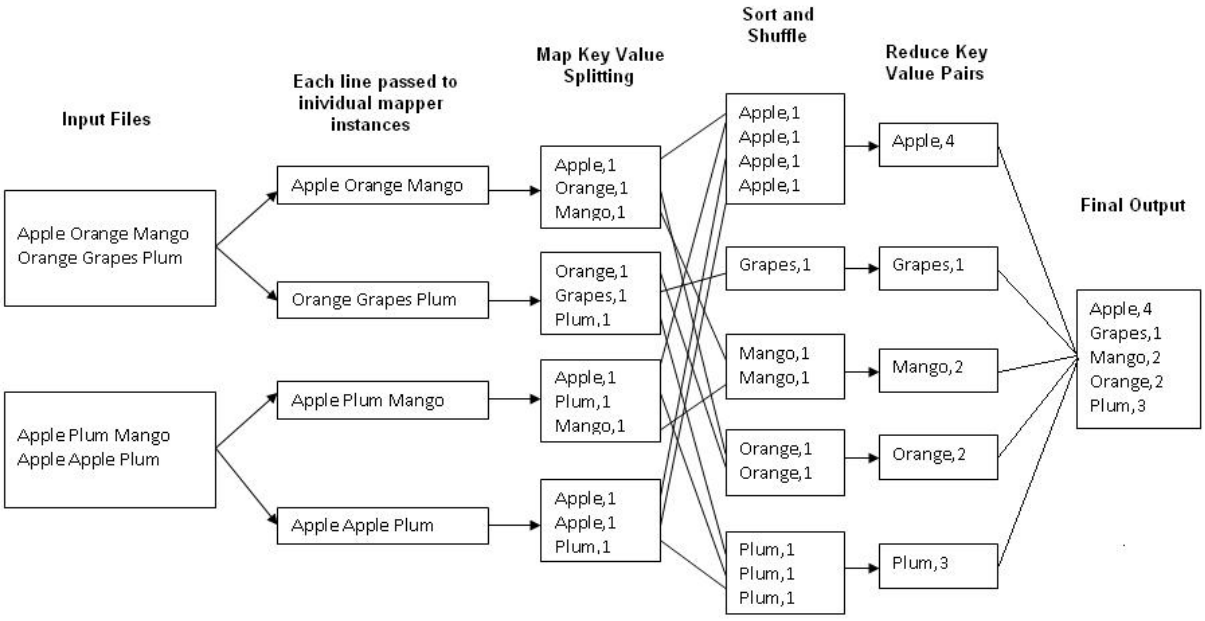
$$\textbf{reduce}(\ f, 0, [v_0, v_1, \ldots, v_{n-1}\ ])$$

$$\textbf{reduce}(\ sum, 0, [\ 1, 2, 3\ ]) \Rightarrow sum(sum(sum(0, 1), 2), 3)$$



De reduce functie aggregeert alle elementen in een groep. In dit voorbeeld, gebruiken we een sum als functie. Je kunt het aggregeren van een lijst dan voorstellen als steeds twee warden aan de functie geven, en het resultaat met de volgende waarde steeds weer aan de functie geven.

Ook hier geldt, deze mate van abstractie maakt het mogelijk voor distributed computing frameworks, om de reduce niet lokaal maar distributed uit te voeren. Het framework kan de optimale strategie bepalen om zo efficient mogelijk tot het resultaat te komen. De volgorde van het reducen maakt niet uit.



Word Count

```
wordcounts = sc.textFile('Wikipedia.gz')
     .flatmap(lambda x: x.split())
     .map(lambda x: (x, 1))
     .aggregateByKey(lambda x, y: x + y)

wordcounts.collect()
```

Efficiente code, niet minder leesbaar.

Resilient Distributed Dataset (RDD)

An **RDD** contains a collection of elements (any type).

Spark automatically distributes the elements over the cluster.

```
numberlist = [1, 2, 3, 4, 5]
numbers = sc.parallelize(numberlist, 2)
numbers.getNumPartitions()
```

```
2
```

```
nameages = sc.parallelize([('Peter', 3), ('Mike', 2), ('John', 5)])
nameages.getNumPartitions()
```

```
32
```

Je kunt het aantal partities handmatig aangeven, anders gebruikt Spark zoveel partities als er cores in de machine zitten.

Transparant Multiprocessing

In [13]:
```
%%time
bignum1 = sc.parallelize(range(1000000000), 1)
bignum1.filter(lambda x: x % 10 == 0).count()
```

```
CPU times: user 12 ms, sys: 4 ms, total: 16 ms
Wall time: 2min 12s
```

In [14]:
```
%%time
bignum32 = sc.parallelize(range(1000000000), 20)
bignum32.filter(lambda x: x % 10 == 0).count()
```

```
CPU times: user 8 ms, sys: 4 ms, total: 12 ms
Wall time: 7.53 s
```

(Key, Value) pairs

```
print(babyrdd.first())
```

```
['2013', 'GAVIN', 'ST LAWRENCE', 'M', '9']
```

```
kv = babyrdd.map(lambda x: (x[3], int(x[4])))
kv.take(5)
```

```
[('M', 9), ('M', 9), ('M', 44), ('M', 49), ('M', 50)]
```

```
kv.reduceByKey(lambda x, y: x + y).collect()
```

```
[('M', 545735), ('F', 397693)]
```

Voor aggregraties en joins moet bekend zijn welke data bij elkaar hoort. Het is gebruikelijk in distributed processing om elementen aan een (key, value) structuur te geven, waarbij de key bepaalt welke elementen worden samengenomen. In plain python is het het meest praktisch om een key uit een enkele waarde te laten bestaan, omdat de key hashable moet zijn. De value mag alles zijn wat je wilt.