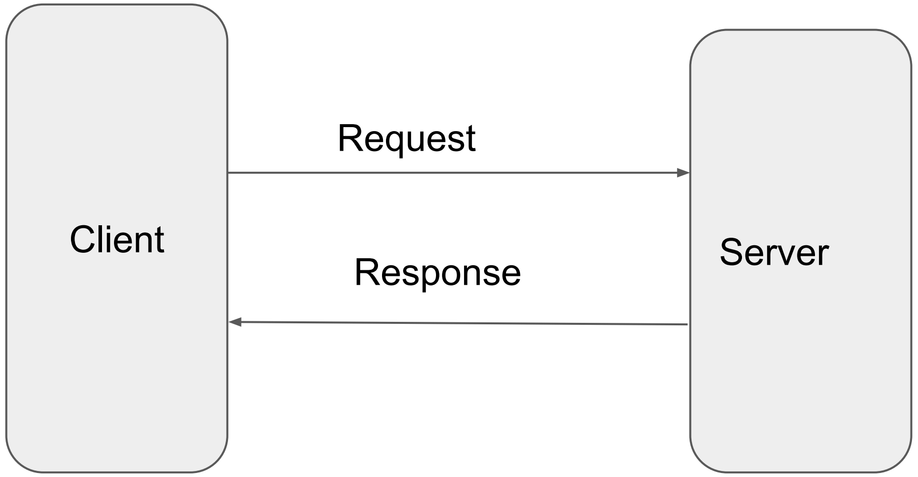
SPARK

Framework for **R**esilient **D**istributed Parallel and Concurrent Computing of **D**ataset

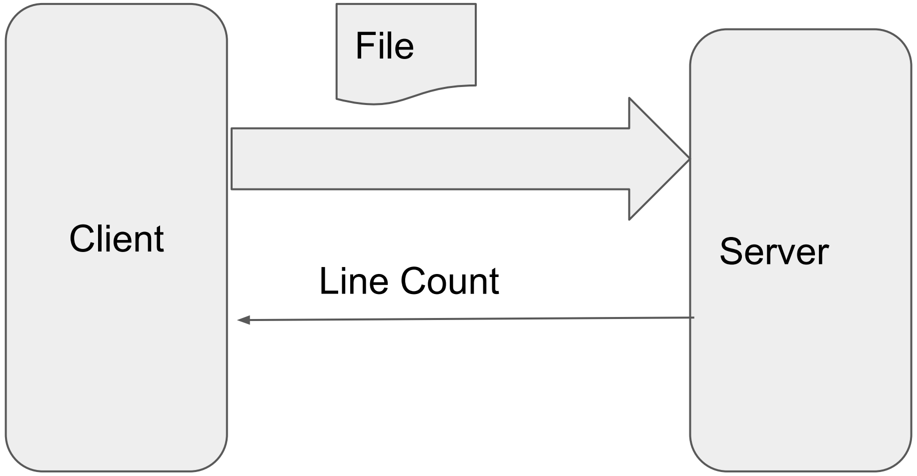
aka

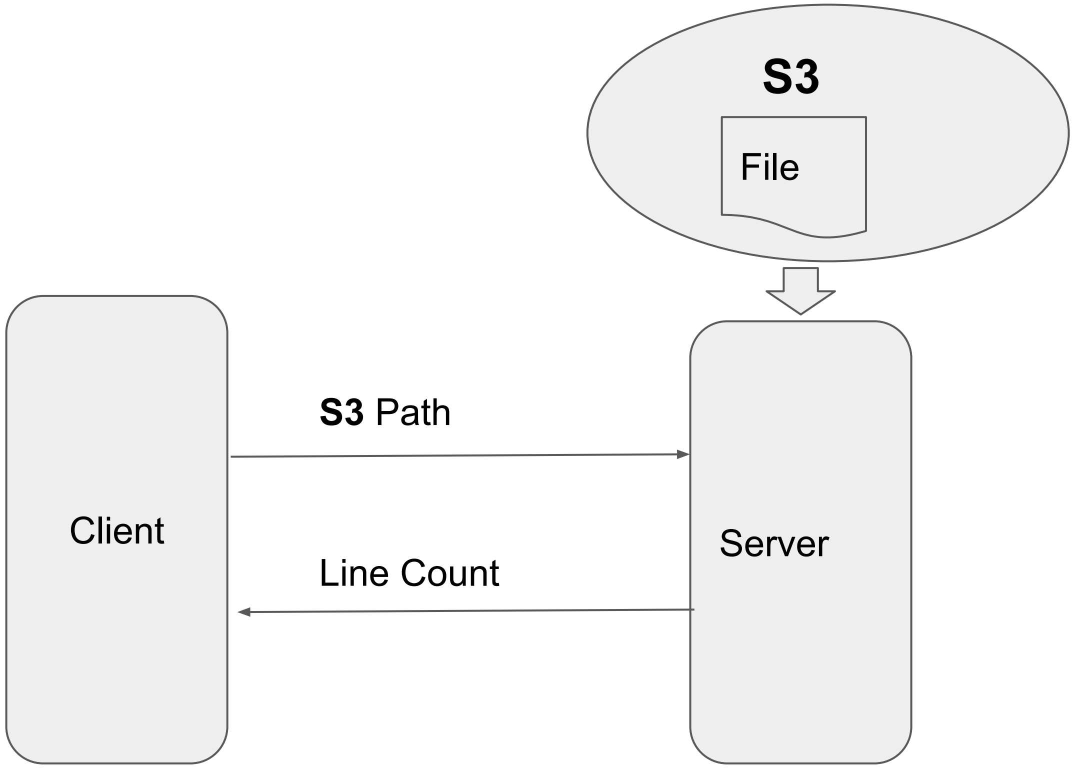
WTF

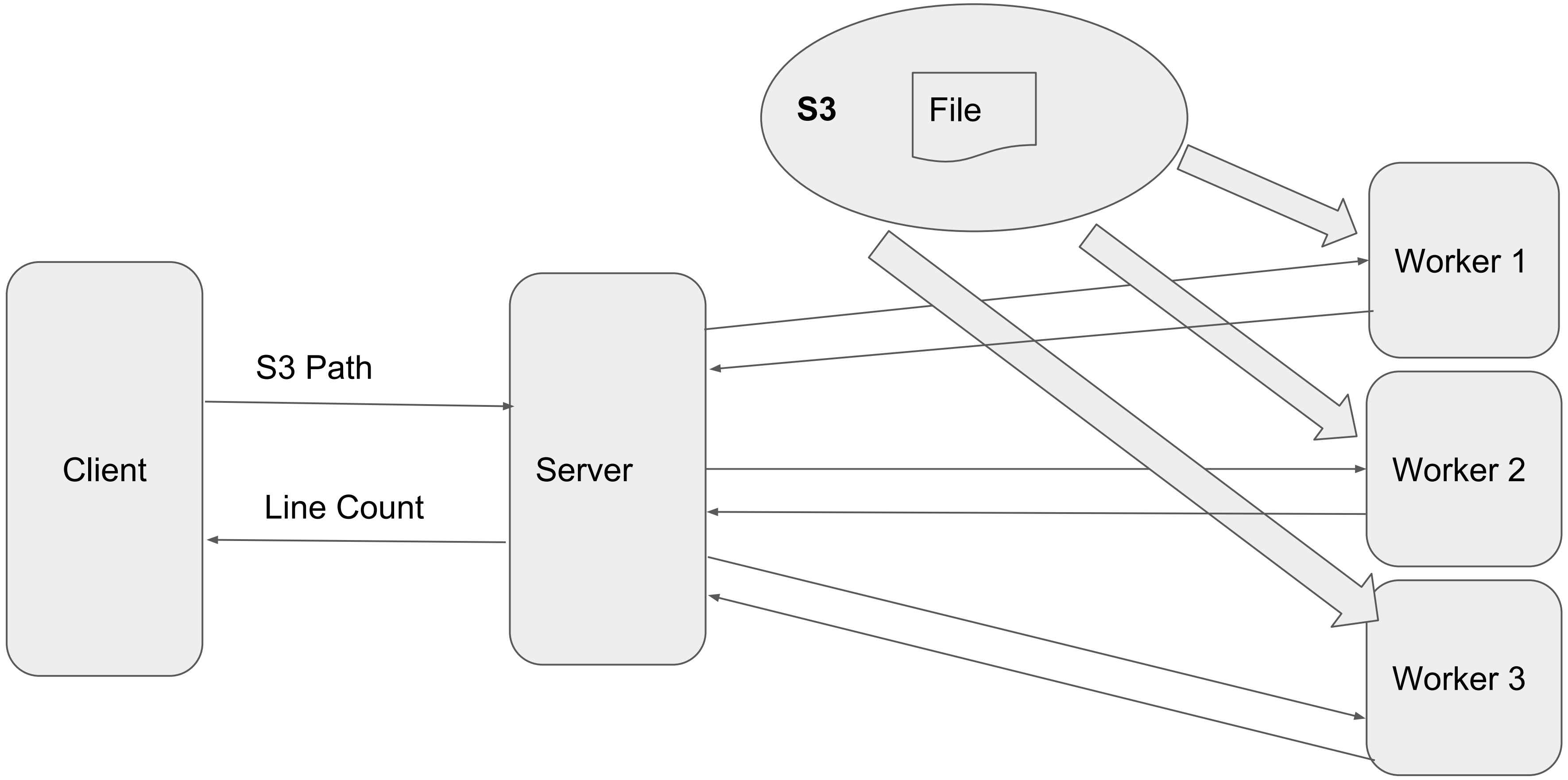
Client Server Architecture



Line Counting on File



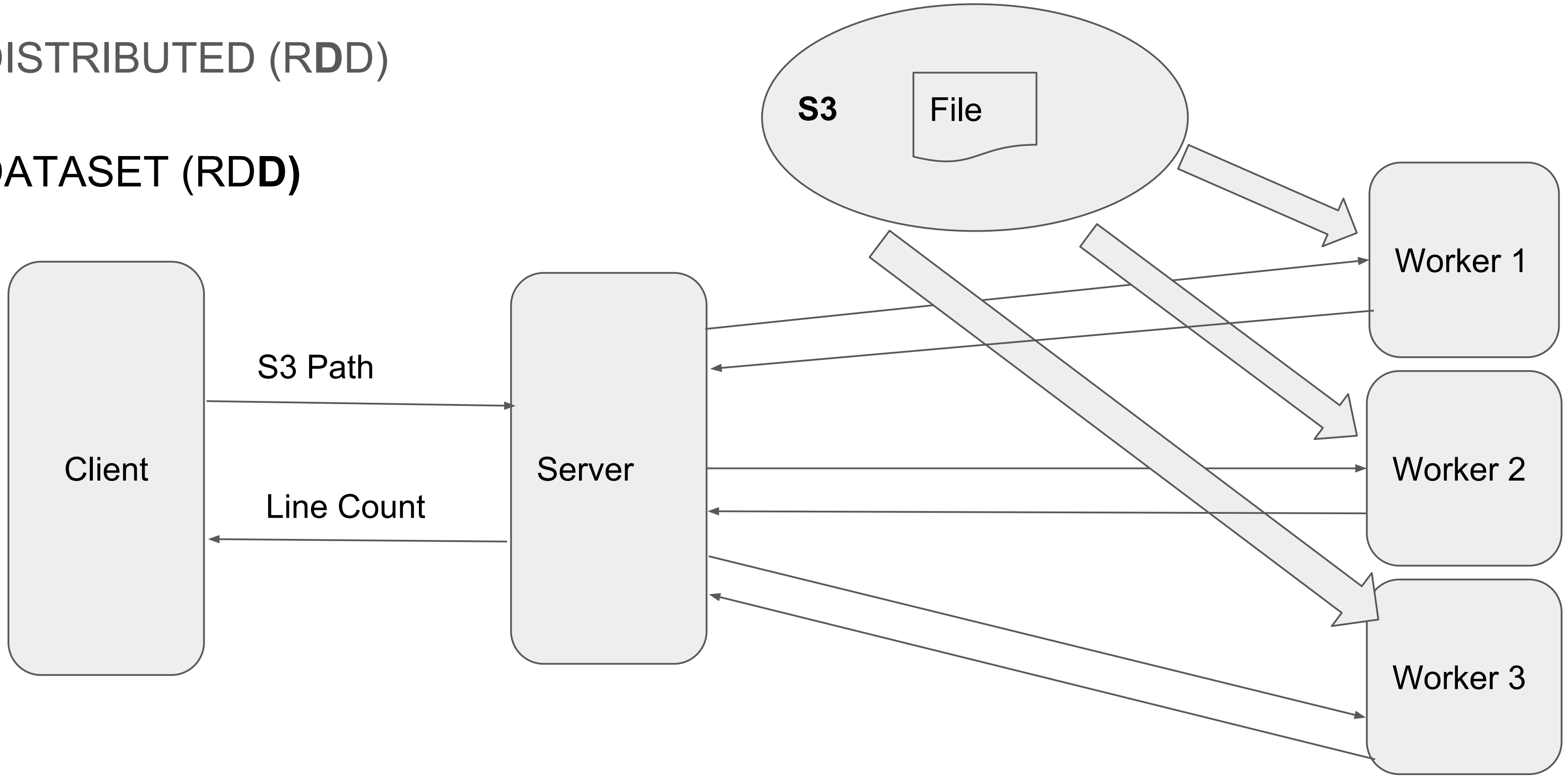

S3
File
S3 Path
Client
Server
Line Count


S3
File
Worker 1
Worker 2
Worker 3
Client
Server
S3 Path
Line Count

RESILIENT (**R**DD)

DISTRIBUTED (R**D**D)

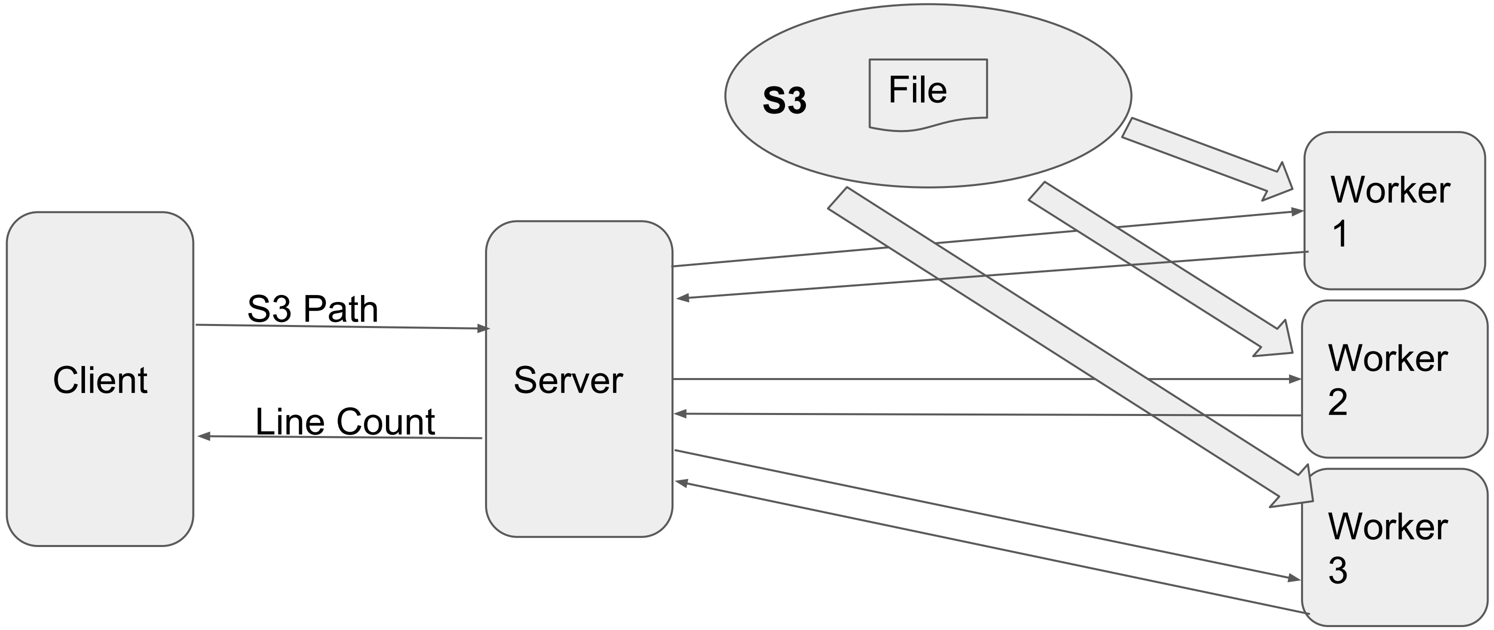
DATASET (RD**D**)



PARTITIONER

Partitions the input data to be run on various workers

Defines the level of parallelism (usually total # of available cores)



Map

f(x) => y

Reduce

Reduces a collection to a single value.

It’s an associative operation (like addition, multiplication, min, max)

f(x1, x2) => Y

f(x2, x1) => Y

Map (Parallel) And Reduce (Concurrent)

$$\begin{tikzcd}
A_1 \arrow[r] & B_1 \arrow[rdd] & \\
A_2 \arrow[r] & B_2 \arrow[rd] & \\
 & & C \\
A_3 \arrow[r] & B_3 \arrow[ru] & \\
A_n \arrow[r] & B_n \arrow[ruu] &
\end{tikzcd}$$

Simplest MapReduce (Do Nothing)

Map: f(x) => x

Reduce: f(x1, x2) => [x1, x2]

MapReduce (Addition)

Map: f(x) => x

Reduce: f(x1, x2) => x1 + x2

MapReduce (Find Max)

Map: f(x) => x

Reduce: f(x1, x2) => max(x1, x2)

MapReduce (Count # of lines)

Map: f(x) => 1

Reduce: f(x1, x2) => x1 + x2

MapReduce (Count lines with Colorado)

```
Map: f(x) => 1          if(line contains Colorado)
          => 0          otherwise
```

Reduce: f(x1, x2) => x1 + x2

MapReduce (Count lines for each State)

```
Map: f(x) => (Colorado, 1)          if(line contains Colorado)
          => (Alaska, 1)            if(line contains Alaska)
          .
          .                          and so on
          .
          => (‘’, 1)                 otherwise
```

Reduce**ByKey**: f(x1, x2) => Sum by Key (State)

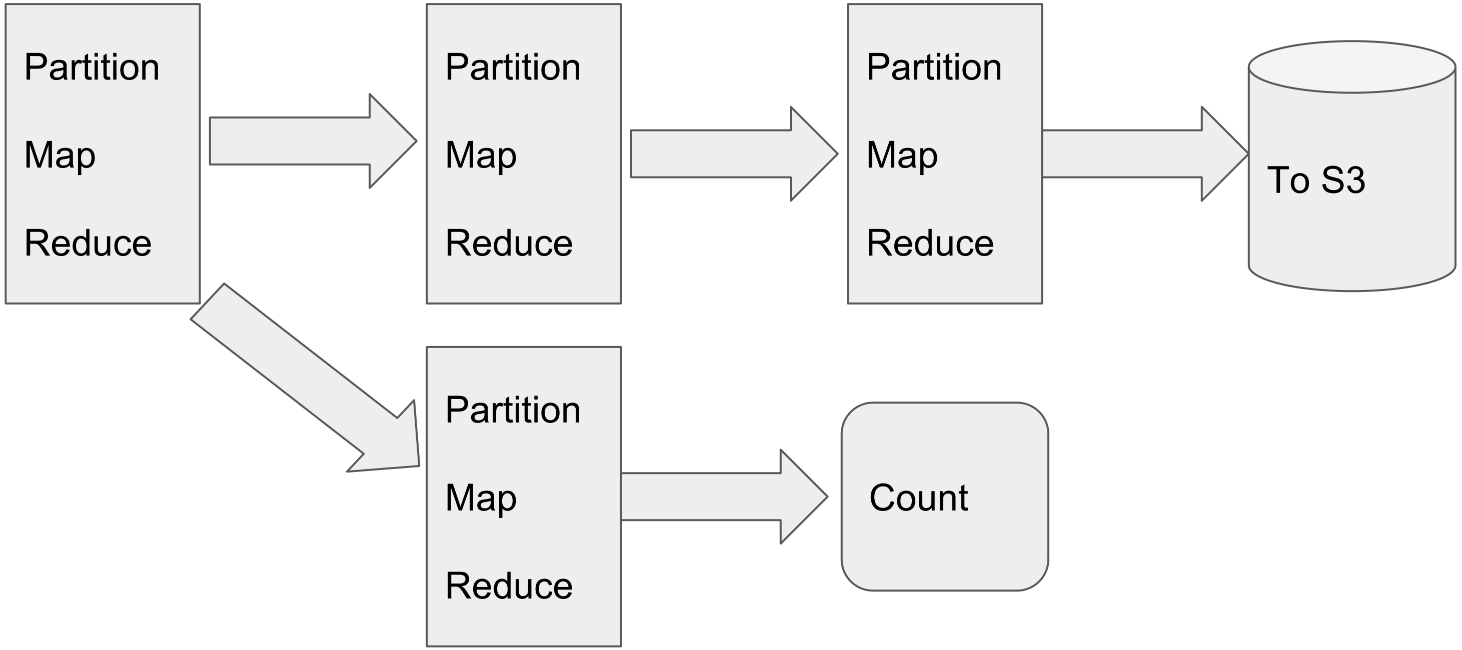
Hadoop

Partition

Map

Reduce

Spark



Chain of functions is evaluated **on demand** (lazily)

Functions can be evaluated multiple times

DataFrame

MapReduce on RDD is great for geeks

Layer on top of RDD to do the usual SQL like operations (90 % of use cases)

DataFrame.count => RDD.map(x => 1).reduce(SUM)

Q&A

Are we done yet?