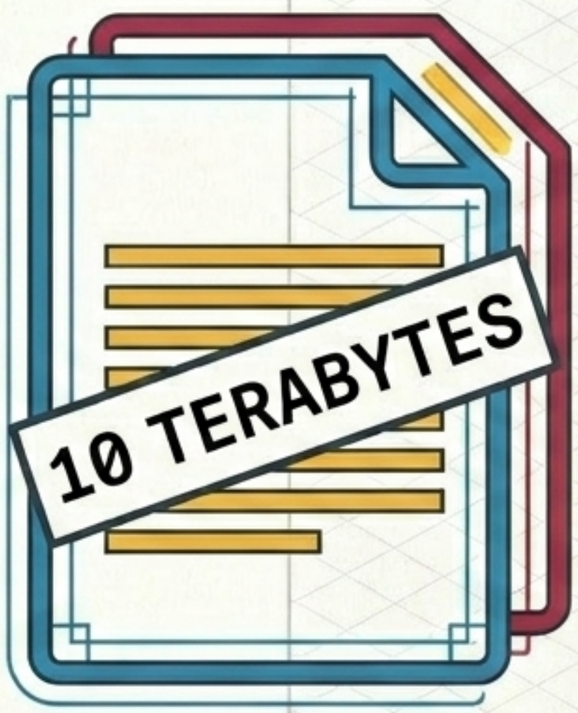
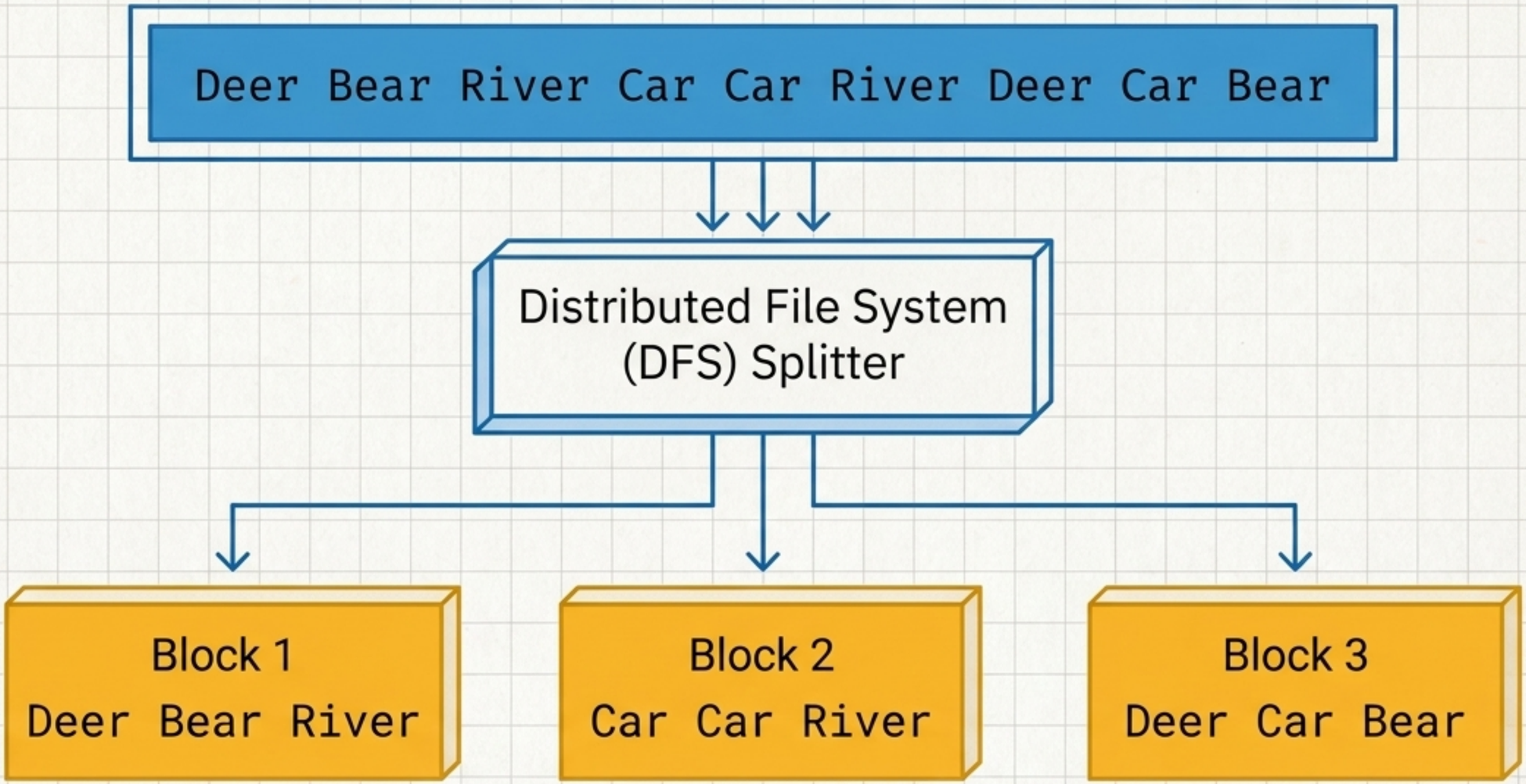
# Execution: The WordCount Benchmark

WordCount is the “Hello World” of distributed computing. The goal is simple: take massive volumes of raw text and count the exact frequency of every unique word.




Doing this sequentially on a 10-terabyte text file would take days. Let’s watch MapReduce do it in parallel.

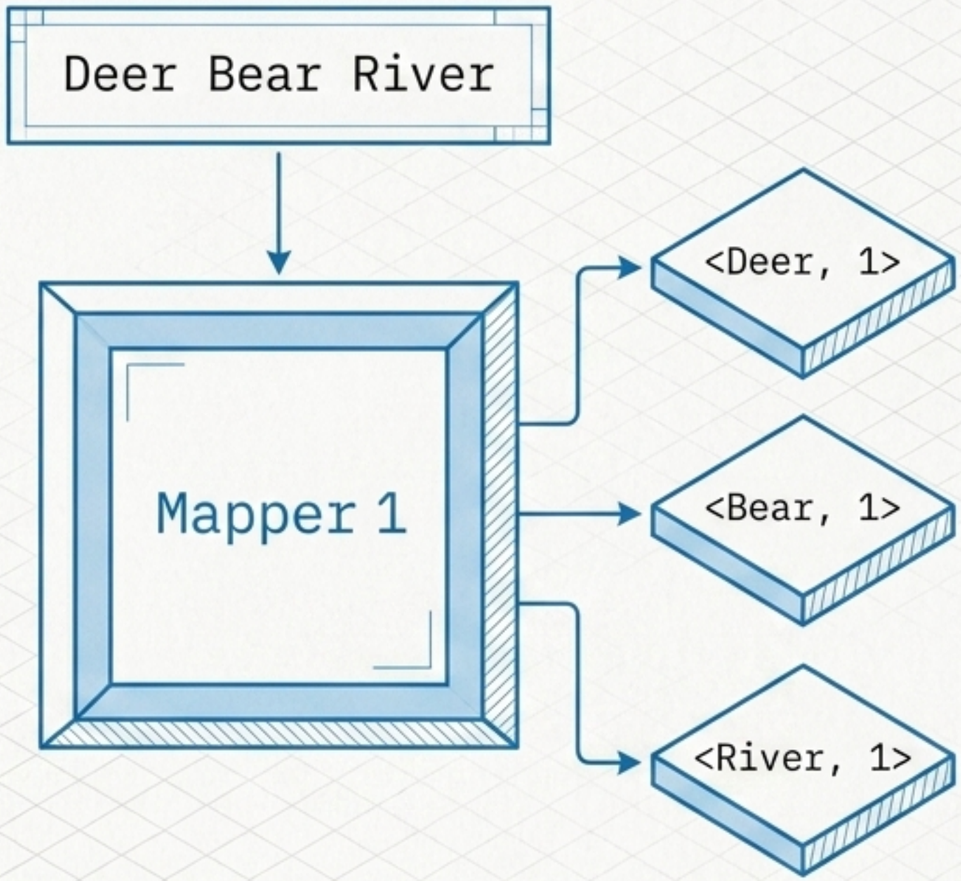



Data is pre-partitioned into discrete blocks ready for parallel Mapper tasks.

# Phase 1: The Map Code

```
function map(key, document):
    for each word in document:
        emit(word, 1)
```

# Visual Execution (Block 1)

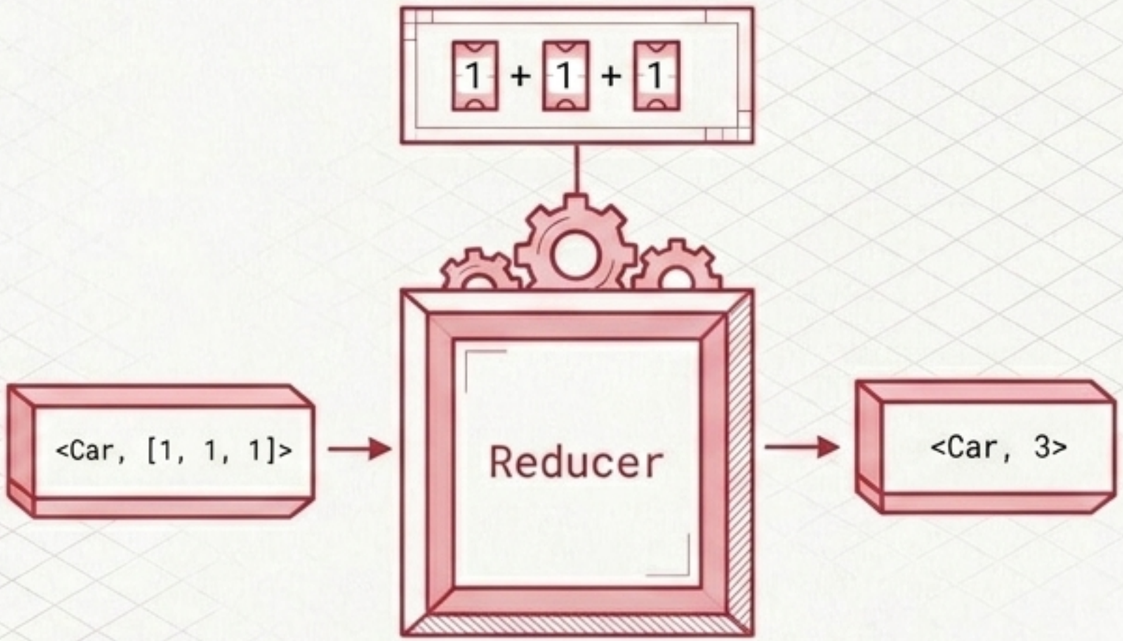


# Phase 2: The Invisible Shuffle & Sort

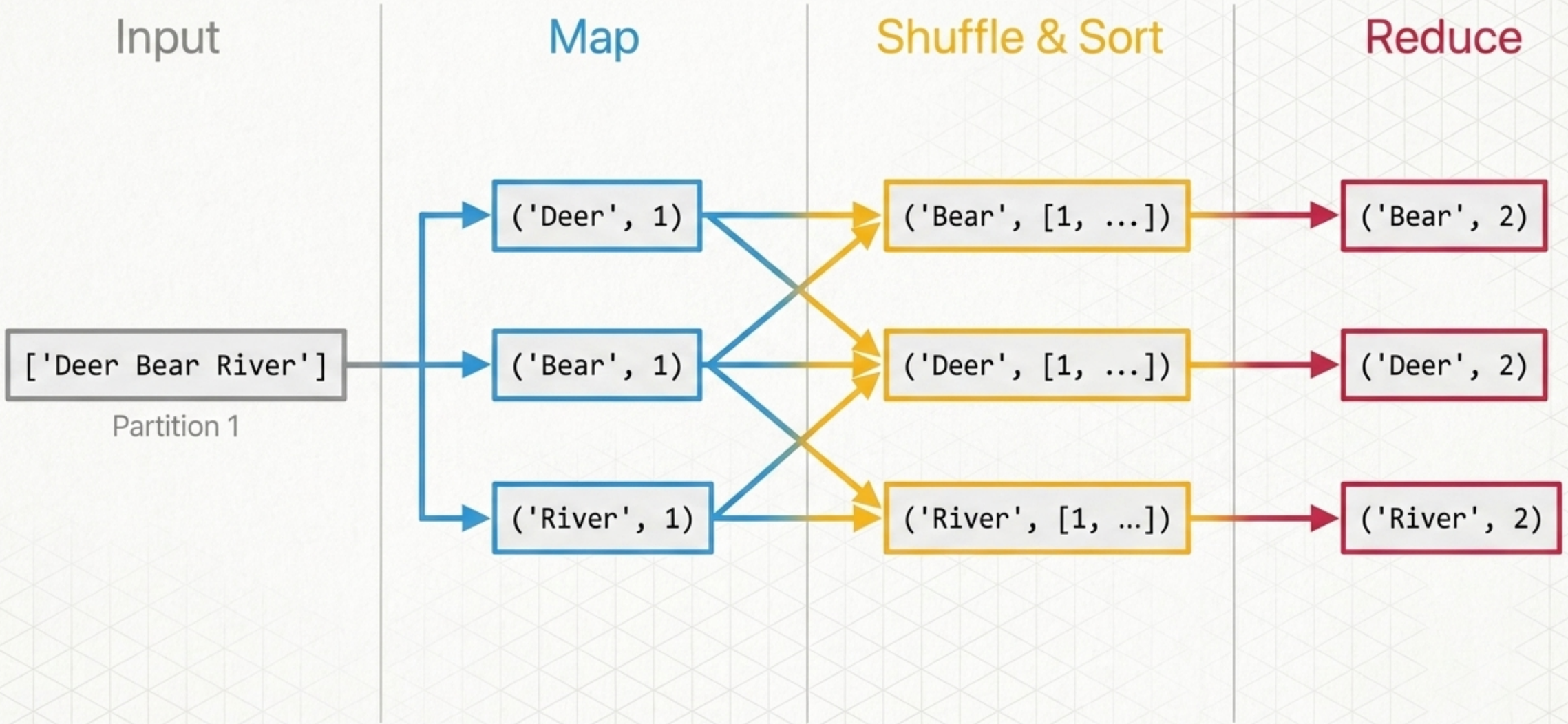
# Phase 3: The Reduce Code

```
function reduce(word, values):
    sum = 0
    for each v in values:
        sum += v
    emit(word, sum)
```

# Visual Execution (Car Group)

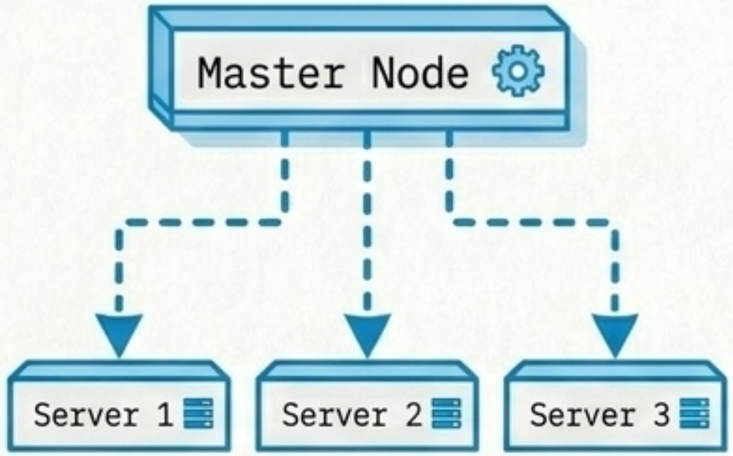


# WordCount Execution Flow

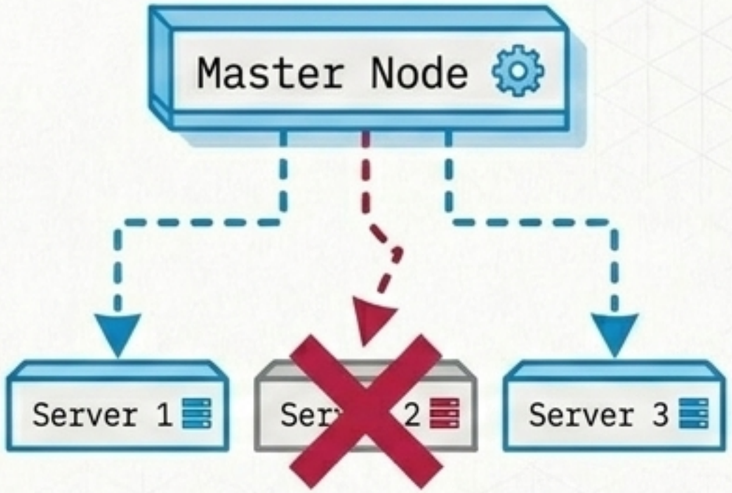


# Fault Tolerance Mechanism

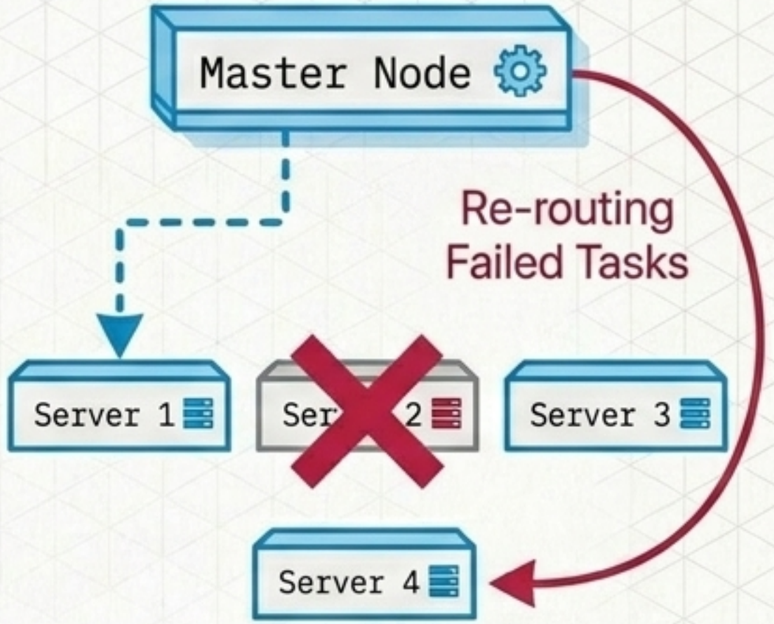
State 1: Health Monitoring




State 2: Node Failure Detected




State 3: Automatic Re-routing




**Invisible Safety Net:** If a node dies mid-computation, MapReduce re-routes the task. The programmer's `WordCount` code never has to change to handle hardware failure.

# Complexity Managed.

1. Move the `computation`, not the `data`.

2. Enforce the `<Key, Value>` contract.

3. Hide system chaos behind the `Map` and `Reduce` APIs.

---

MapReduce remains the defining architecture of the Big Data era because it provided the ultimate abstraction: transforming planetary-scale distributed infrastructure into a simple programming task.