

Map Reduce

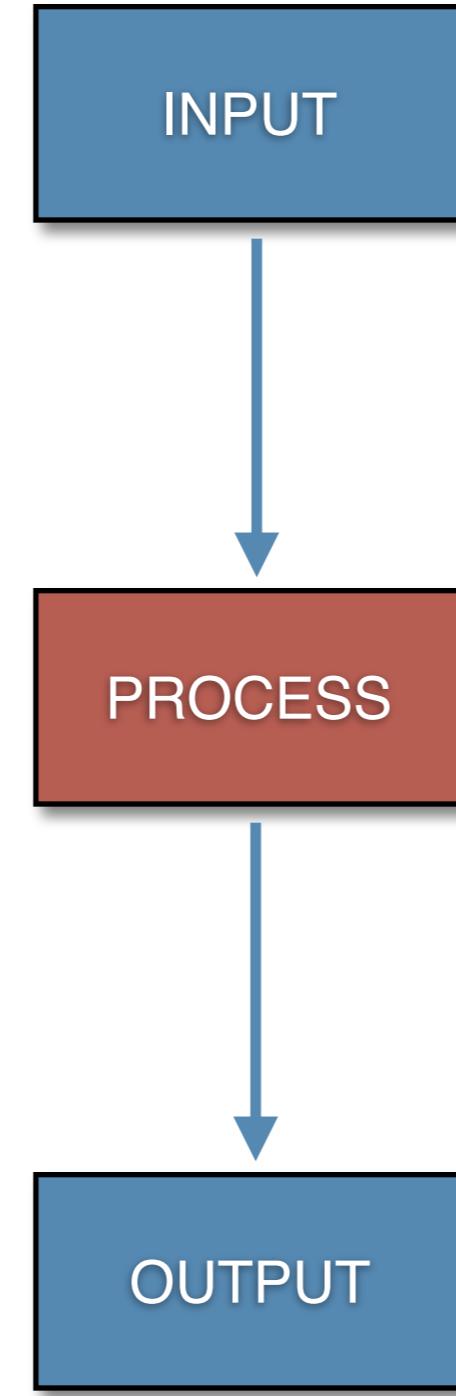


MapReduce inside Google

Googlers' hammer for 80% of our data crunching

- Large-scale web search indexing
- Clustering problems for Google News
- Produce reports for popular queries, e.g. Google Trend
- Processing of satellite imagery data
- Language model processing for statistical machine translation
- Large-scale machine learning problems
- Just a plain tool to reliably spawn large number of tasks
 - e.g. parallel data backup and restore

Typical Application



What if...

INPUT

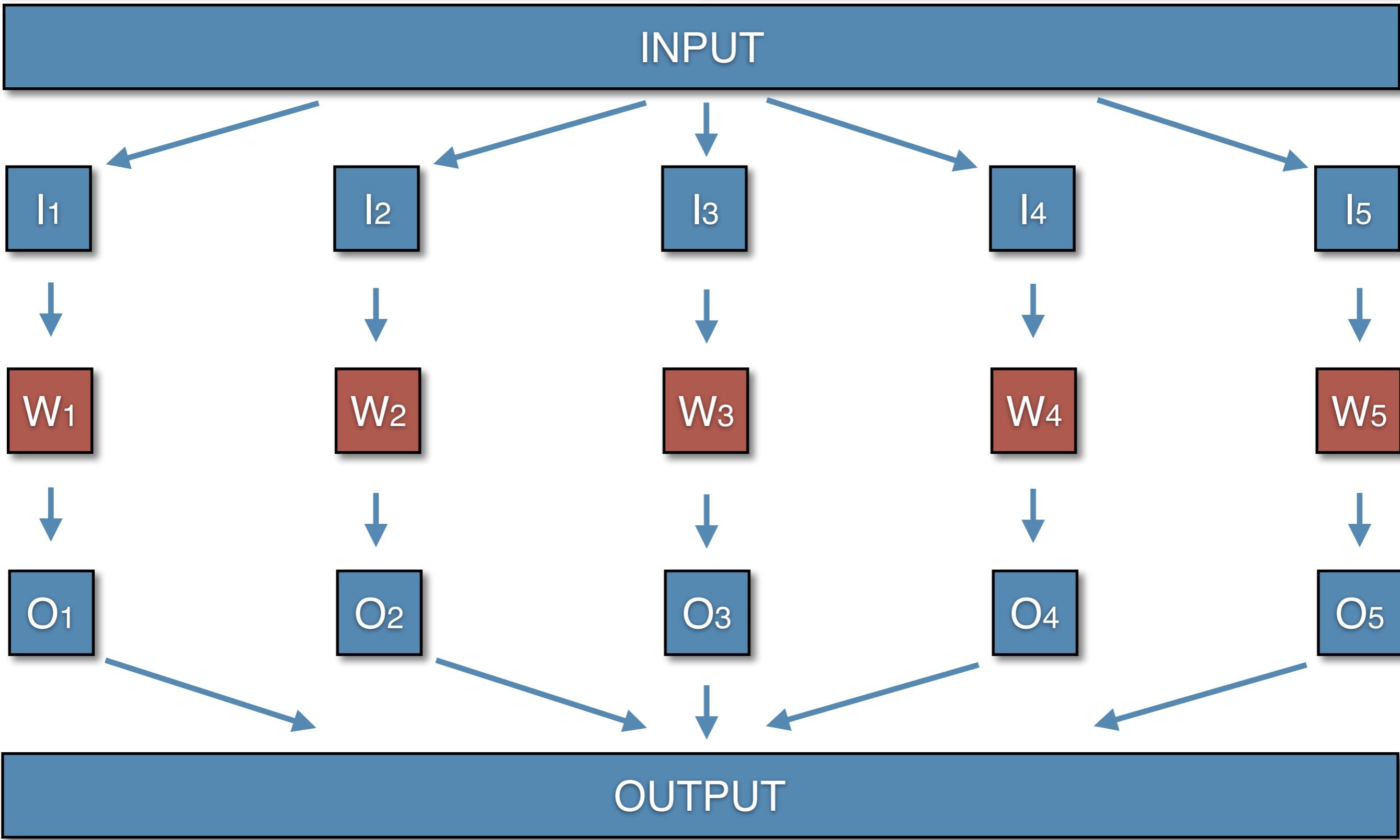


PROCESS



OUTPUT

Divide and Conquer



Challenges

- How do we split the input?
- How do we distribute the input splits?
- How do we collect the output splits?
- How do we aggregate the output?
- How do we coordinate the work?
- What if input splits > num workers?
- What if workers need to share input/output splits?
- What if a worker dies?
- What if we have a new input?

Difficulties

- We don't know the order in which workers run...
- We don't know when workers interrupt each other...
- We don't know when workers need to communicate partial results...
- We don't know the order in which workers access shared resources...

Common Problems

- Communication between workers (e.g., to exchange state)
- Access to shared resources (e.g., data)

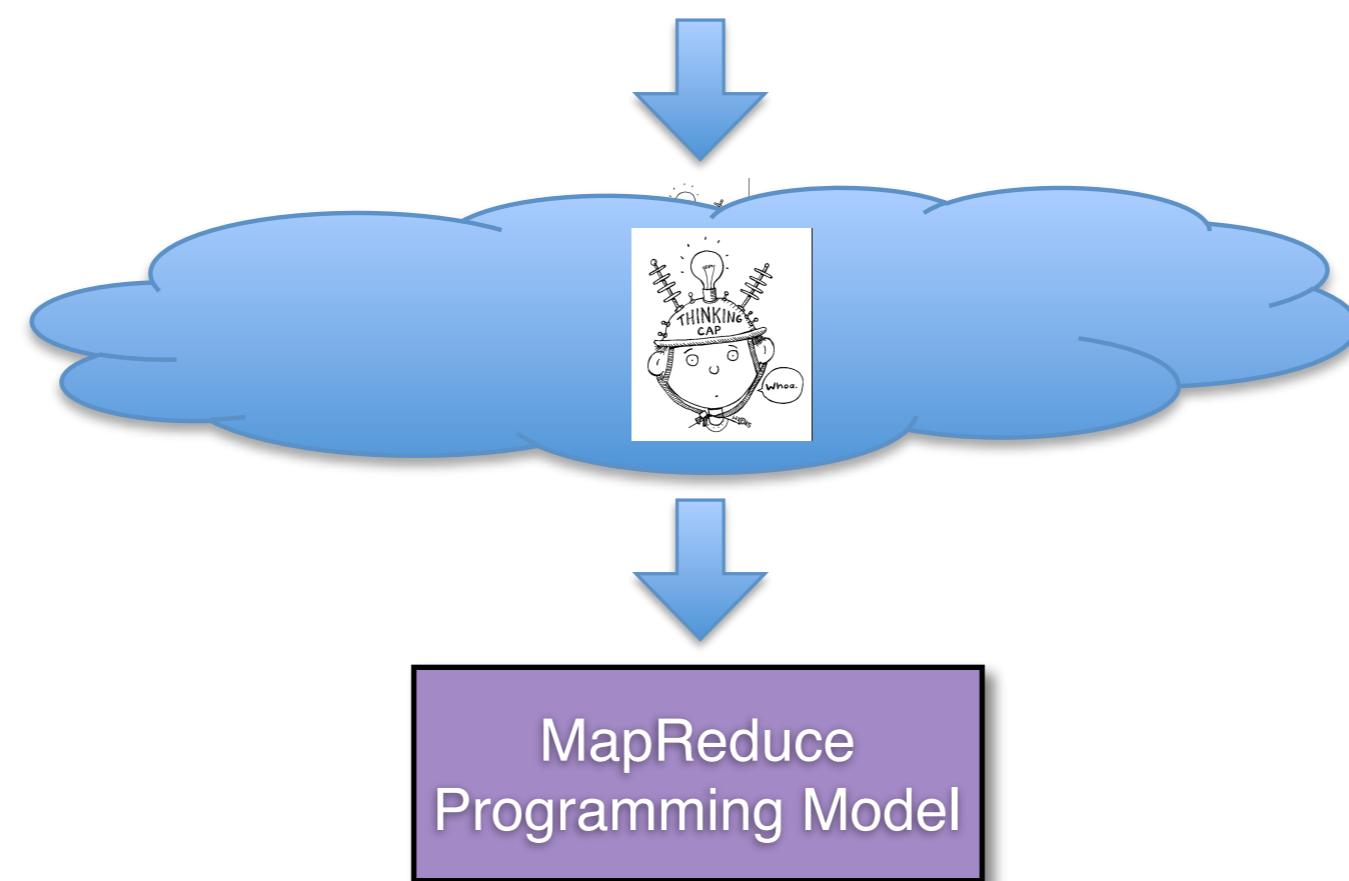


Design Ideas

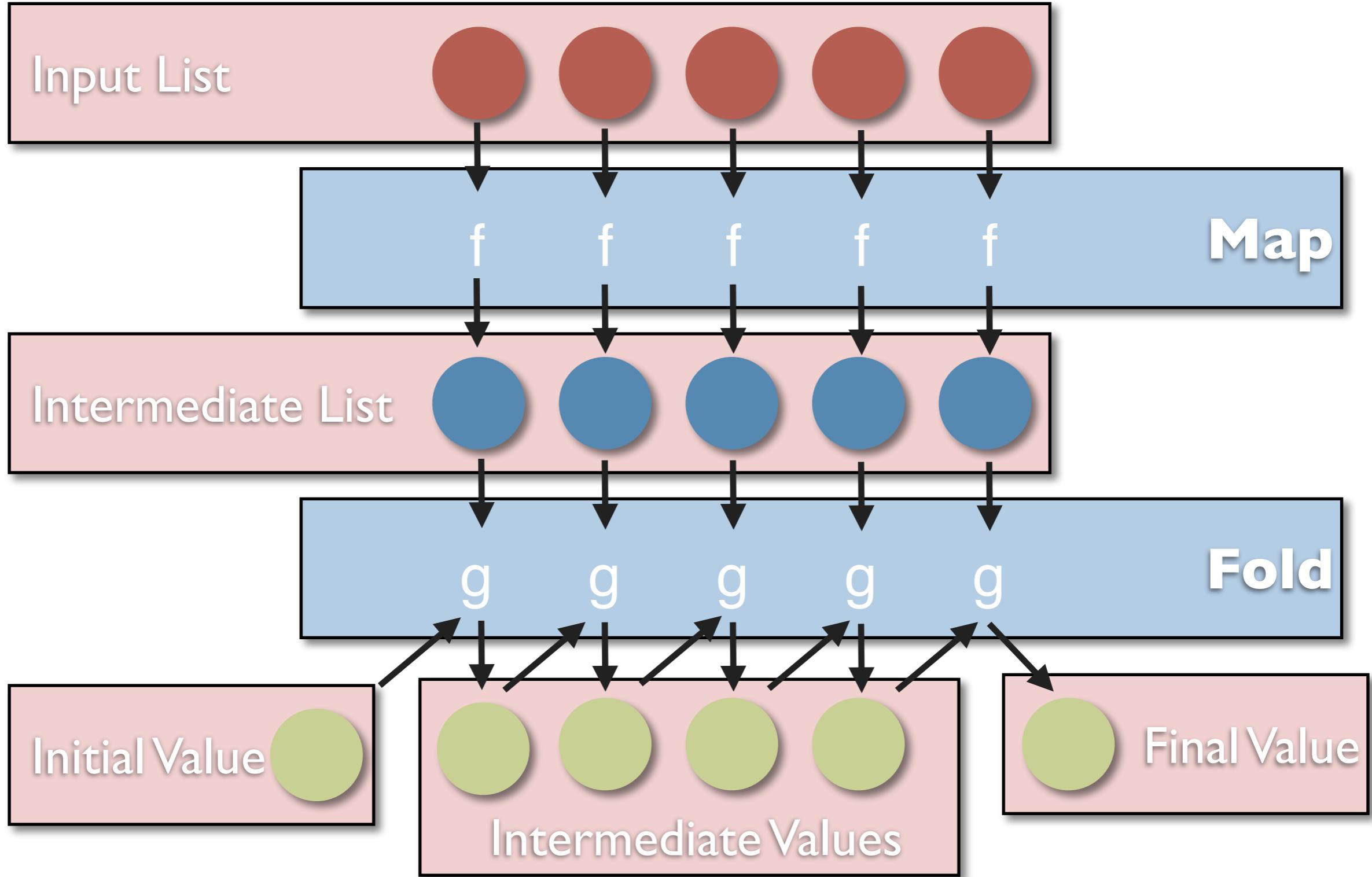
- **Scale “out”, not “up”**
 - Low end machines
- **Move processing to the data**
 - Network bandwidth bottleneck
- **Process data sequentially, avoid random access**
 - Huge data files
 - Write once, read many
- **Seamless scalability**
 - Strive for the unobtainable
- **Right level of abstraction**
 - Hide implementation details from applications development

Typical Large-Data Problem

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output



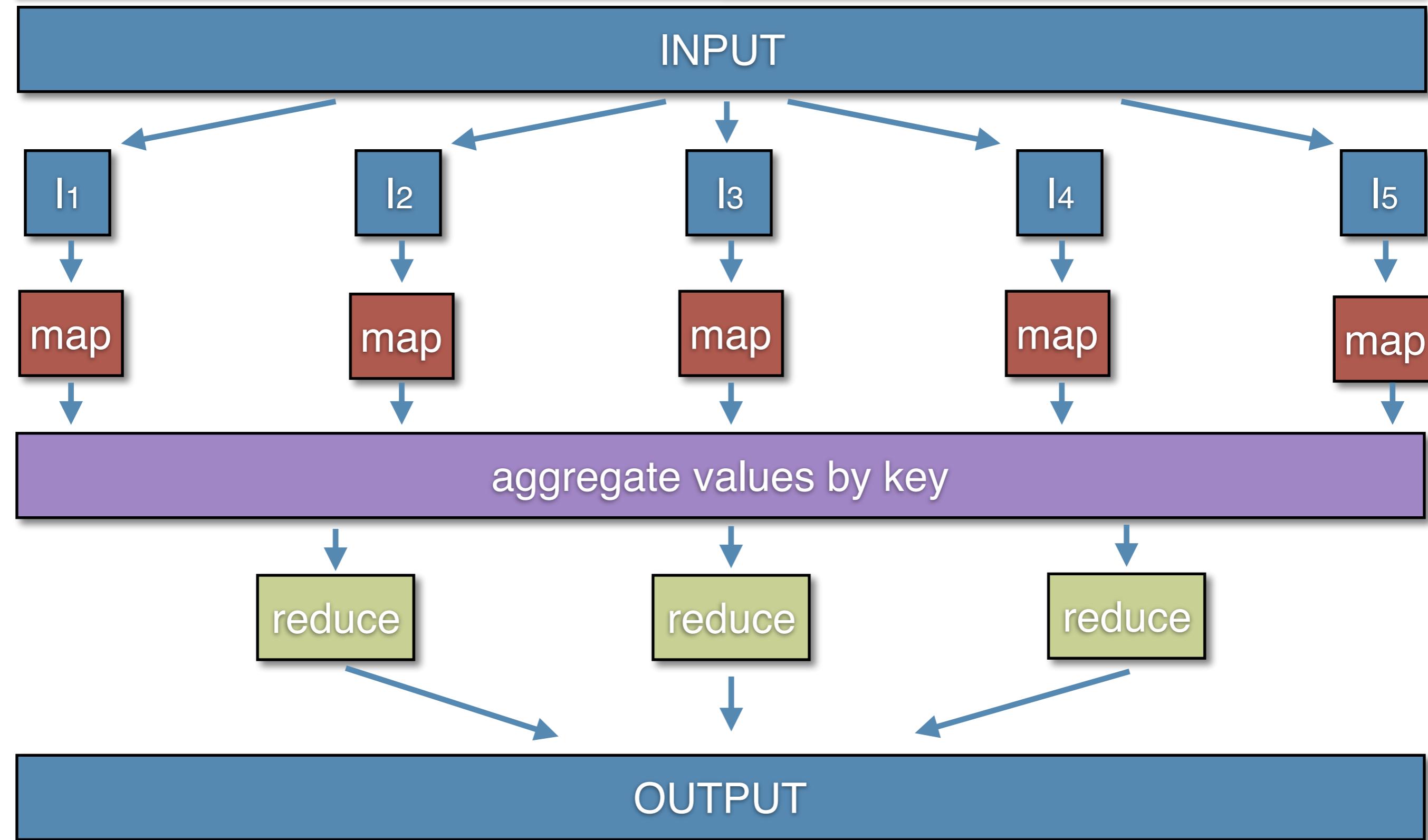
From functional programming...



...to Map Reduce

- Programmers specify two functions
 - **map** (k_1, v_1) --> [(k_2, v_2)]
 - **reduce** ($k_2, [v_2]$) --> [(k_3, v_3)]
- **Map**
 - Receives as input a key-value pair
 - Produces as output a list of key-value pairs
- **Reduce**
 - Receives as input a key-list of values pair
 - Produces as output a list of key-value pairs (typically just one)
- **The runtime support handles everything else...**

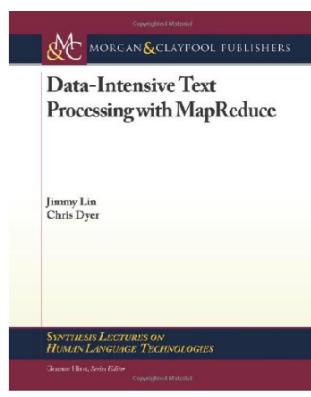
Programming Model (simple)



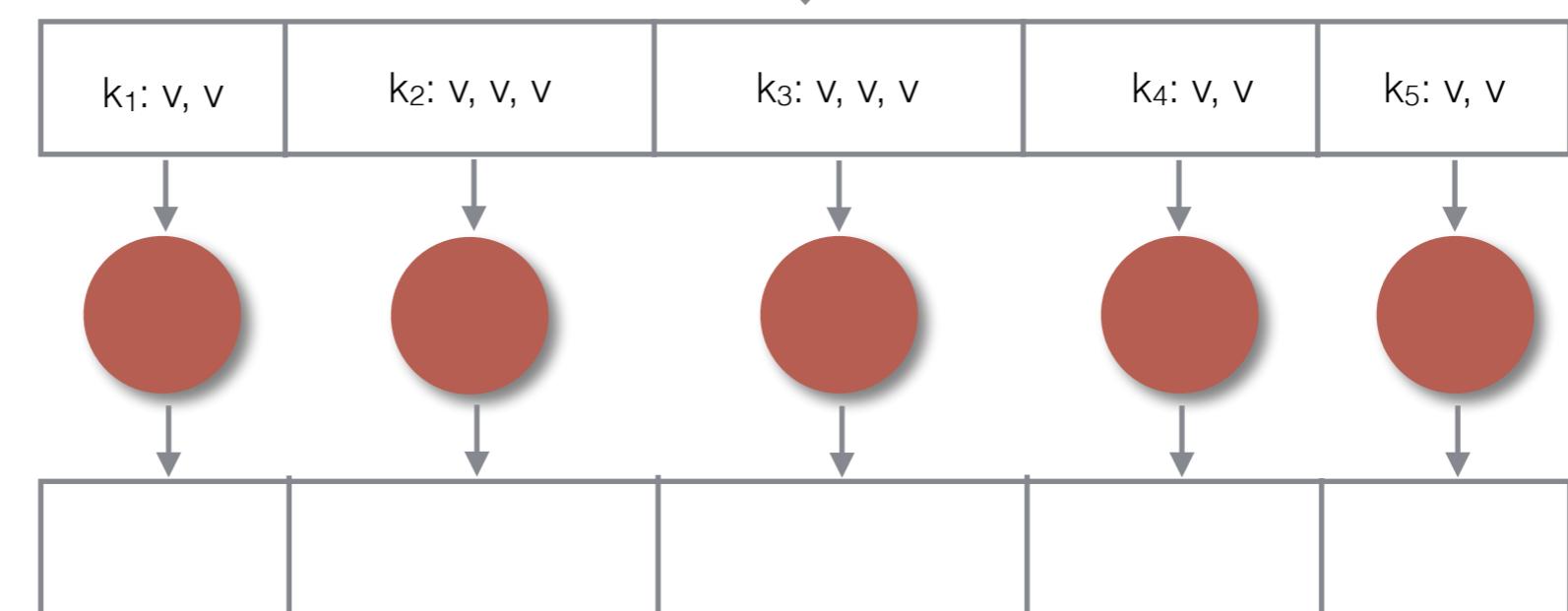
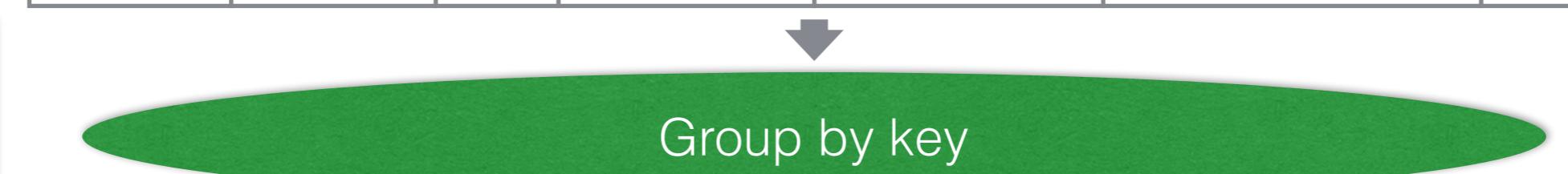
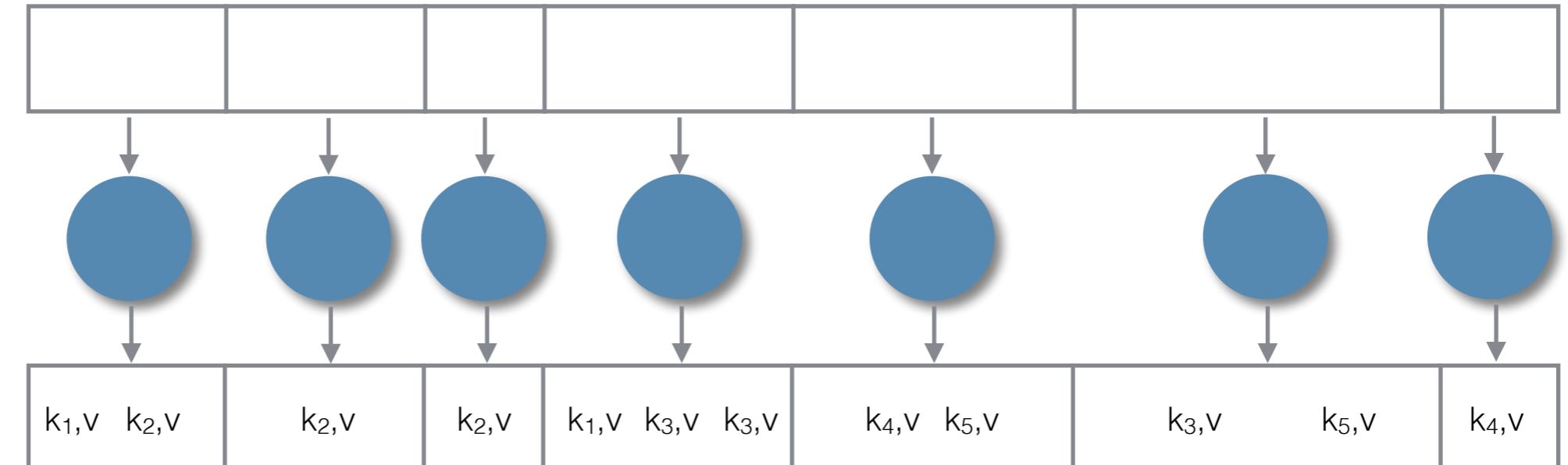
Wordcount Example (I)

```
1: class MAPPER
2:   method MAP(docid a, doc d)
3:     for all term t ∈ doc d do
4:       EMIT(term t, count 1)

1: class REDUCER
2:   method REDUCE(term t, counts [c1, c2, ...])
3:     sum ← 0
4:     for all count c ∈ counts [c1, c2, ...] do
5:       sum ← sum + c
6:     EMIT(term t, count sum)
```



Diagram



Word Frequency Exercise

- What if we want to compute the word **frequency** instead of the word **count**?
- **Input:** large number of text documents
- **Output:** the word frequency of each word across all documents
- **Note:** Frequency is calculated using the **total word count**
- **Hint 1:** We know how to compute the total word count
- **Hint 2:** Can we use the word count output as input?
- **Solution:** Use two MapReduce tasks
 - MR1: count number of all words in the documents
 - MR2: count number of each word and divide it by the total count from MR1