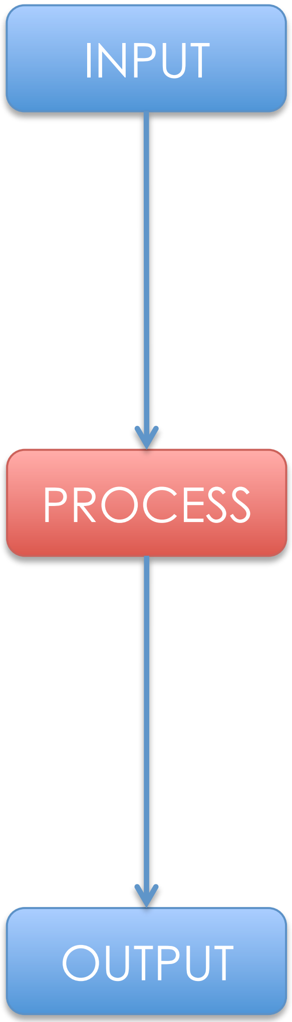
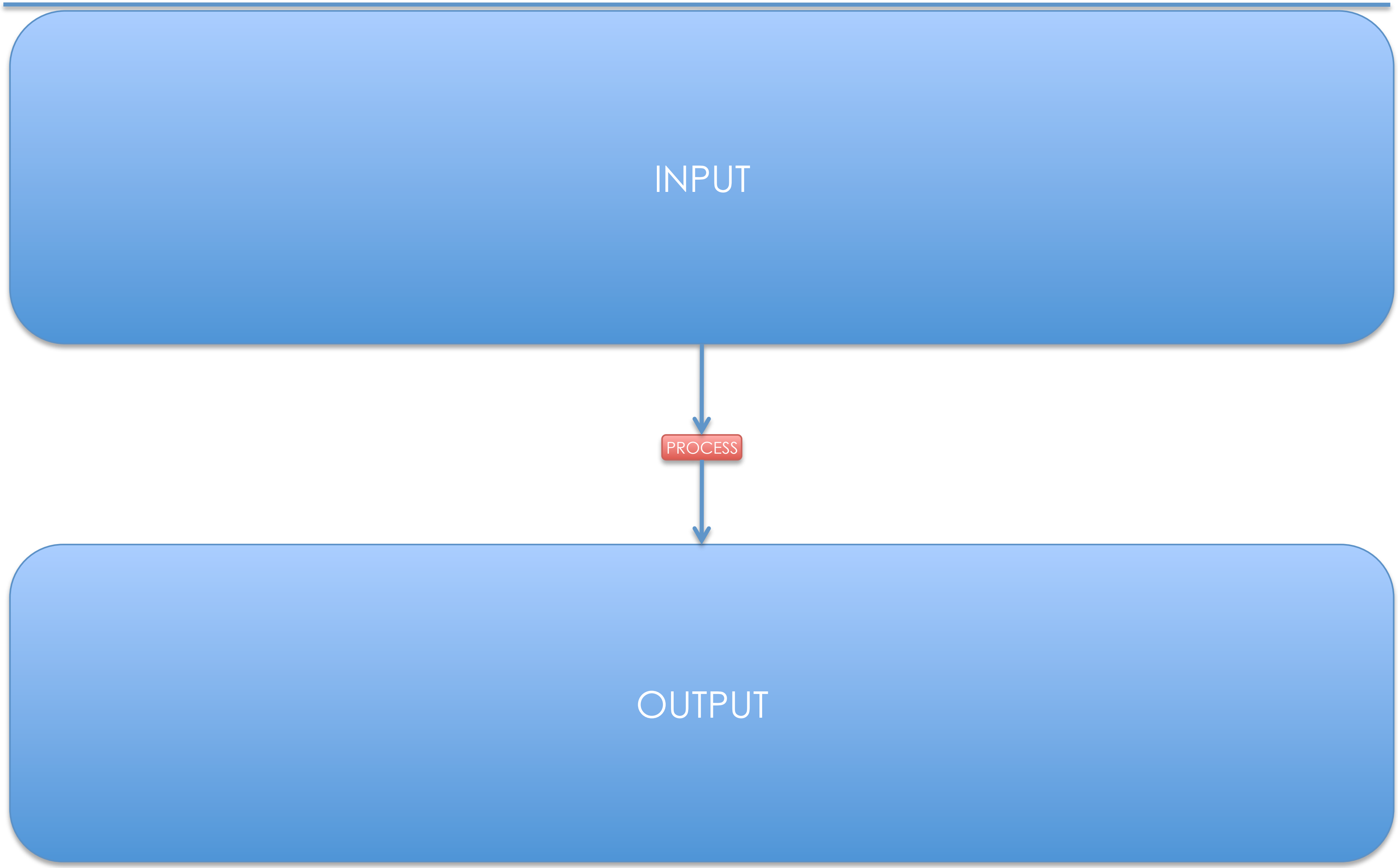
Map Reduce

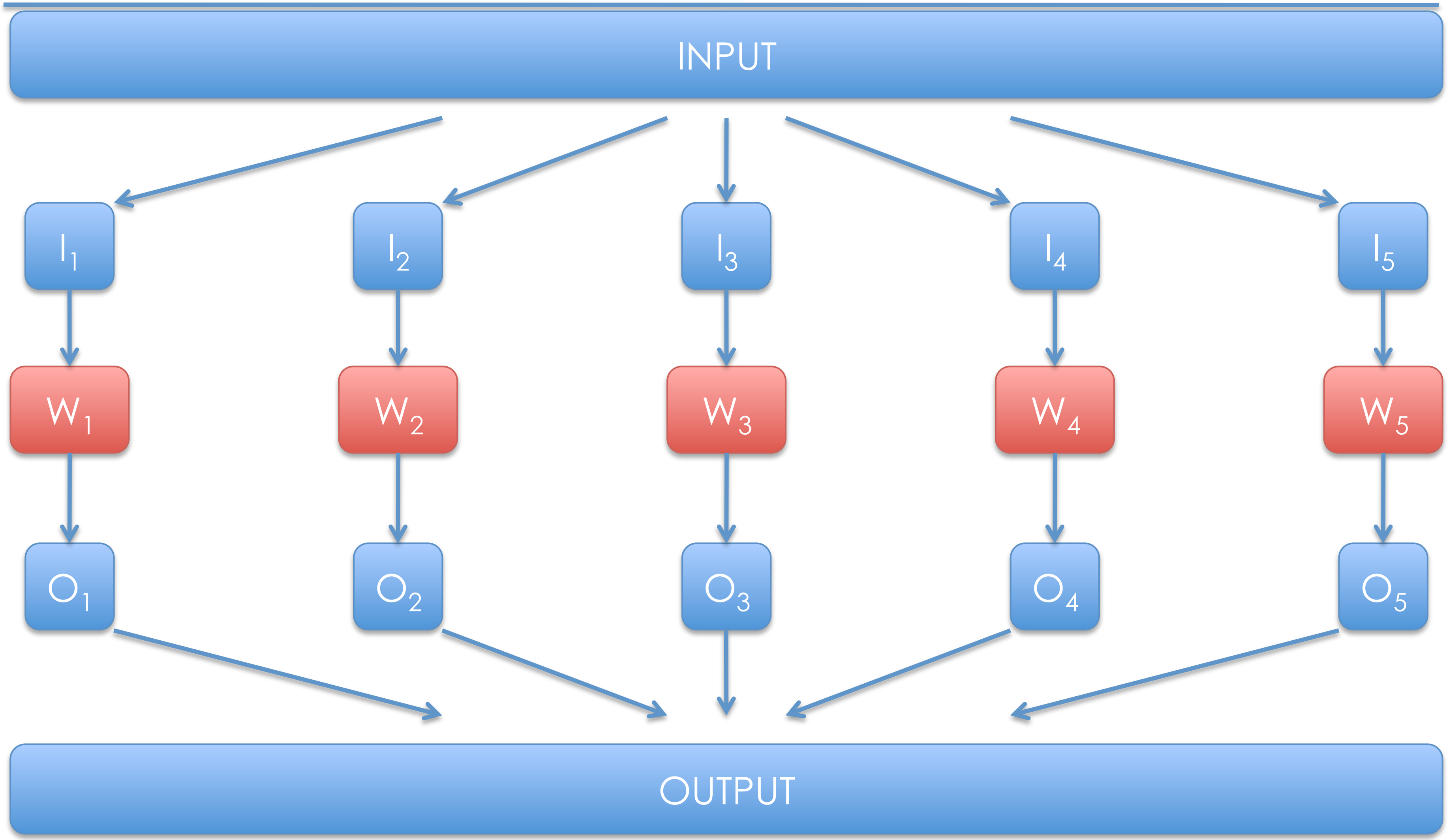
Typical application



What if…



Divide & Conquer



Questions

- 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?

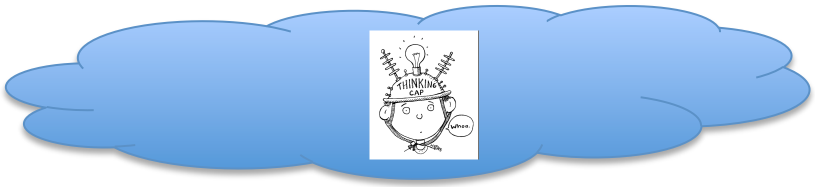

http://www.duiops.net/seresvivos

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

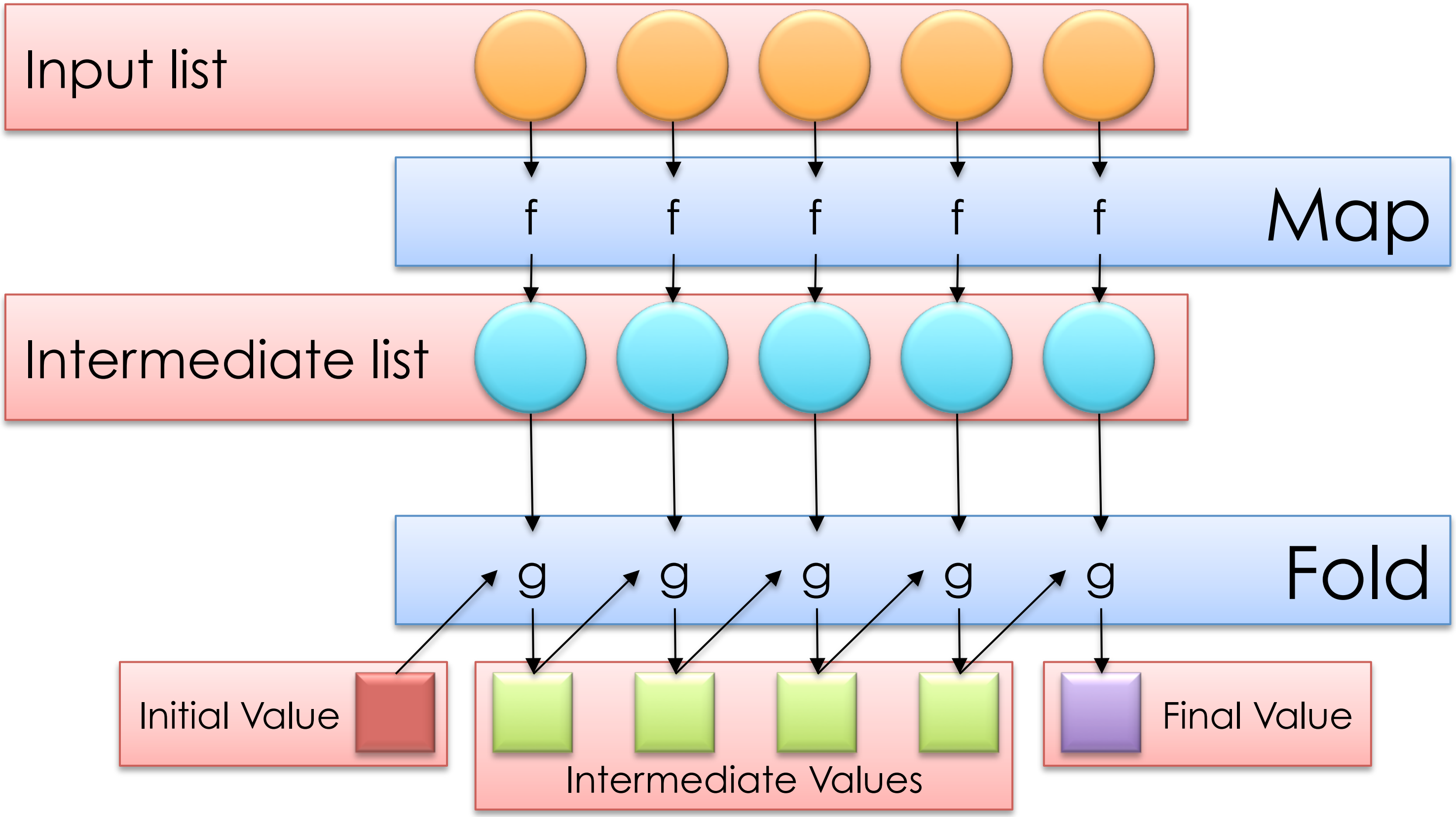
Typical Large-Data Problem

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



Map Reduce Programming Model

From functional programming…



...To MapReduce

- Programmers specify two functions:

  map $(k_1, v_1) \rightarrow [(k_2, v_2)]$

  reduce $(k_2, [v_2]) \rightarrow [(k_3, v_3)]$

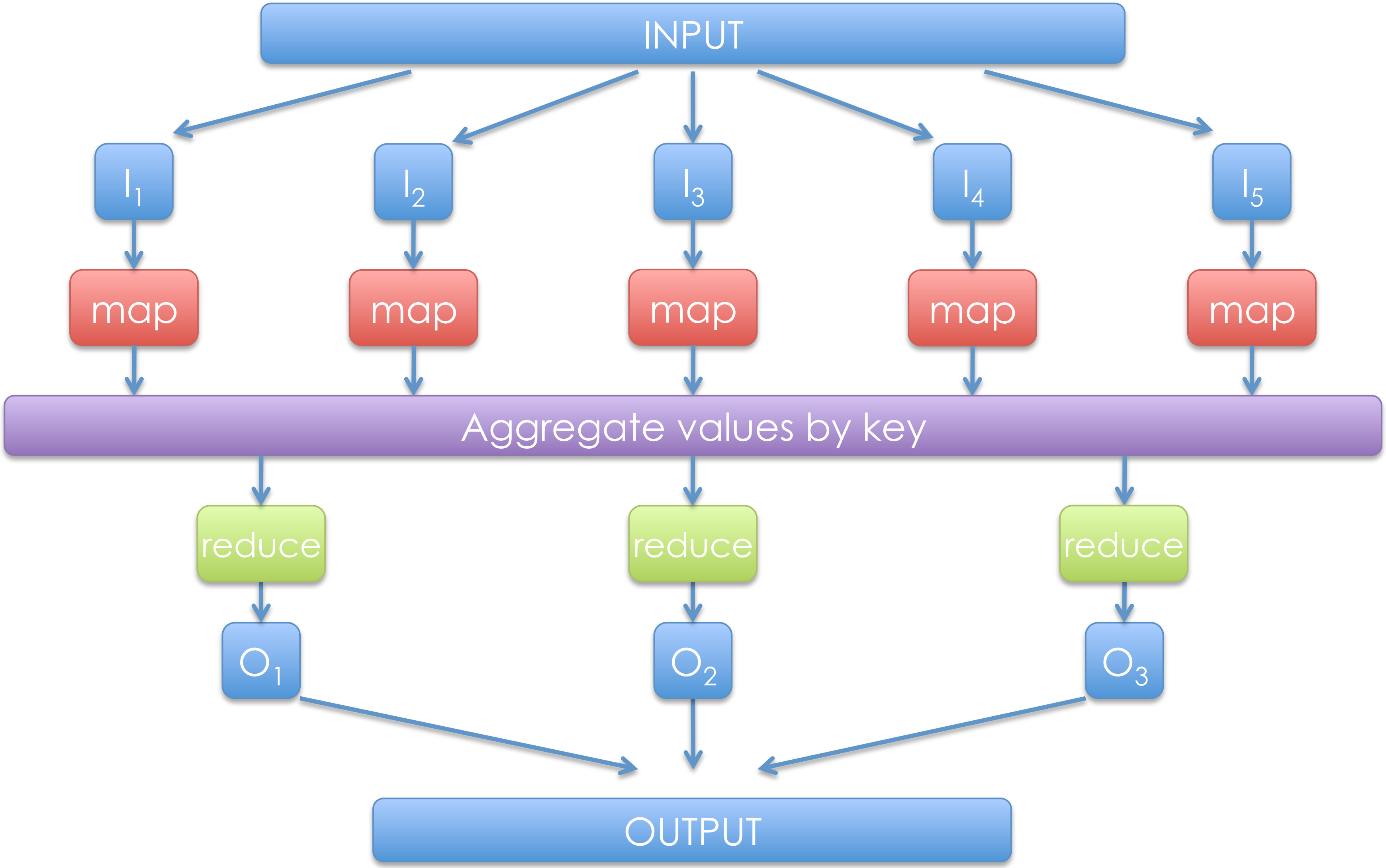
- All values with the same key are sent to the same reducer

- Input keys and values (k_1, v_1) are drawn from different domain than output keys and values (k_3, v_3)

- Intermediate keys (k_2, v_2) and values are from the same domain as the output keys and values (k_3, v_3)

- The runtime handles everything else…

Programming Model (simple)



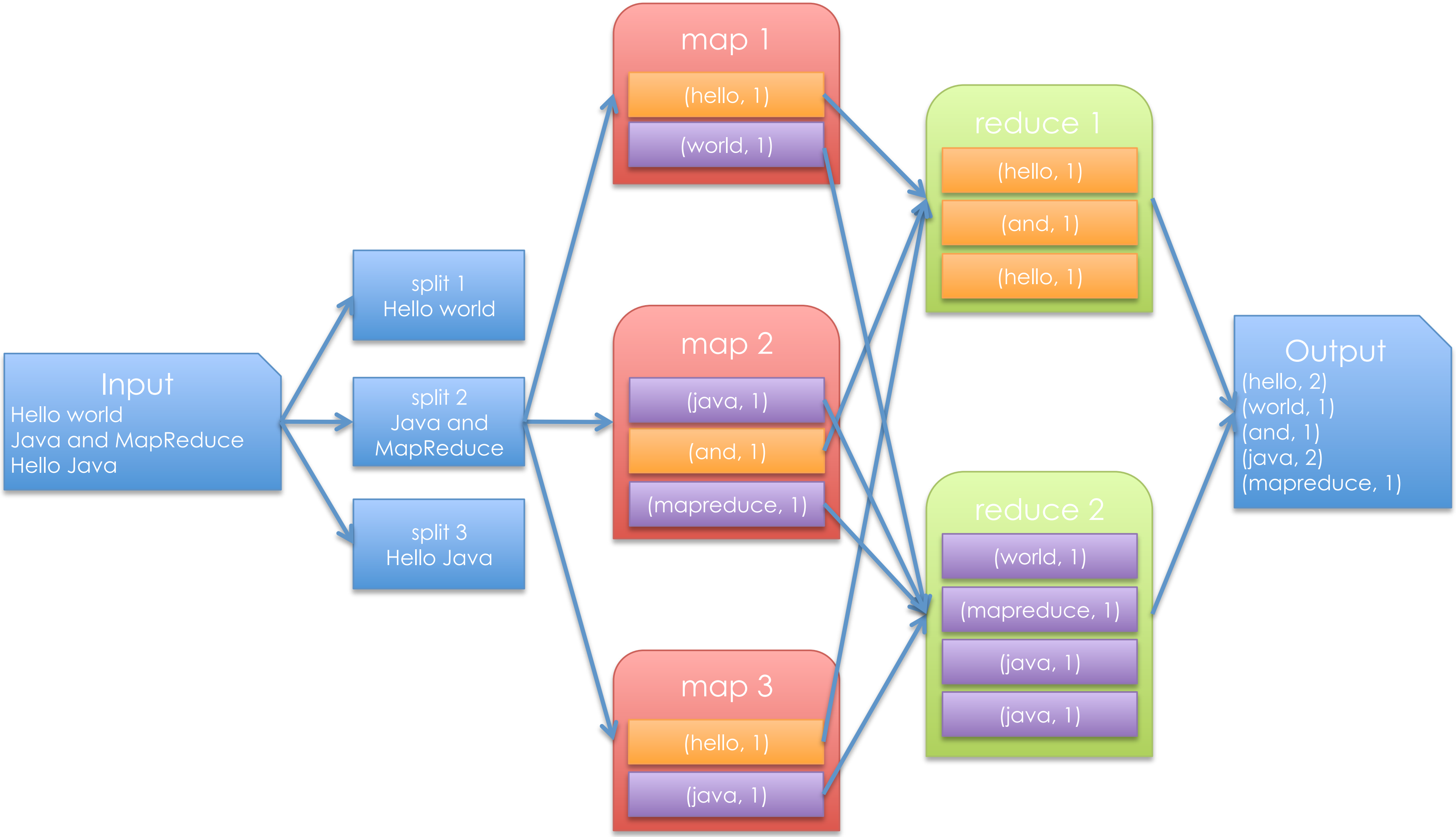
ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"

Example (I)

```
class Mapper
    method Map(docid a, doc d)
        for all term t ∈ doc d do
            Emit(term t, count 1)
```

```
class Reducer
    method Reduce(term t, counts [c1, c2, ...])
        sum ← 0
        for all count c ∈ counts [c1, c2, ...] do
            sum ← sum + c
        Emit(term t, count sum)
```

Example (II)


Input
Hello world
Java and MapReduce
Hello Java
split 1
Hello world
split 2
Java and
MapReduce
split 3
Hello Java
map 1
(hello, 1)
(world, 1)
map 2
(java, 1)
(and, 1)
(mapreduce, 1)
map 3
(hello, 1)
(java, 1)
reduce 1
(hello, 1)
(and, 1)
(hello, 1)
reduce 2
(world, 1)
(mapreduce, 1)
(java, 1)
(java, 1)
Output
(hello, 2)
(world, 1)
(and, 1)
(java, 2)
(mapreduce, 1)

Runtime

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles “data distribution”
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed FS

Partitioners and combiners

- Programmers specify two functions:
  map $(k_1, v_1) \rightarrow [(k_2, v_2)]$
  reduce $(k_2, [v_2]) \rightarrow [(k_3, v_3)]$
  – All values with the same key are reduced together

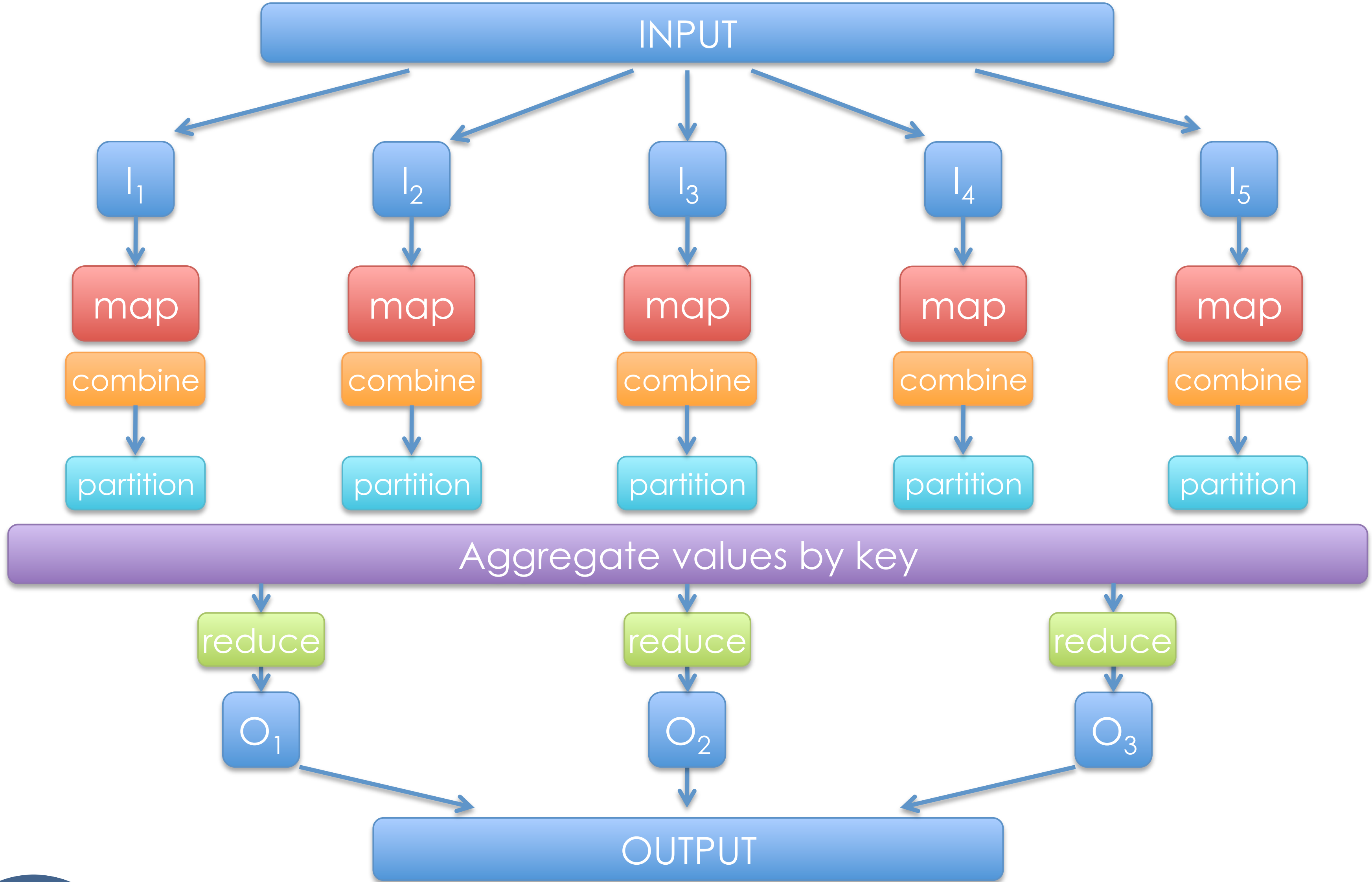
- The execution framework handles everything else…

- Not quite…usually, programmers also specify:

  partition (k_2, number of partitions) → partition for k_2
  – Often a simple hash of the key, e.g., hash(k') mod n
  – Divides up key space for parallel reduce operations

  combine $(k_2, v_2) \rightarrow [(k_2, v_2)]$
  – Mini-reducers that run in memory after the map phase
  – Used as an optimization to reduce network traffic

Programming Model (complete)



ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"

MapReduce can refer to…

- The programming model
- The execution framework (aka “runtime”)
- The specific implementation

MapReduce Implementations

- Google has a proprietary implementation in C++
  - Bindings in Java, Python
- Hadoop is an open-source implementation in Java
  - Development led by Yahoo, used in production
  - Now an Apache project
  - Rapidly expanding software ecosystem
- Lots of custom research implementations
  - For GPUs, cell processors, etc.