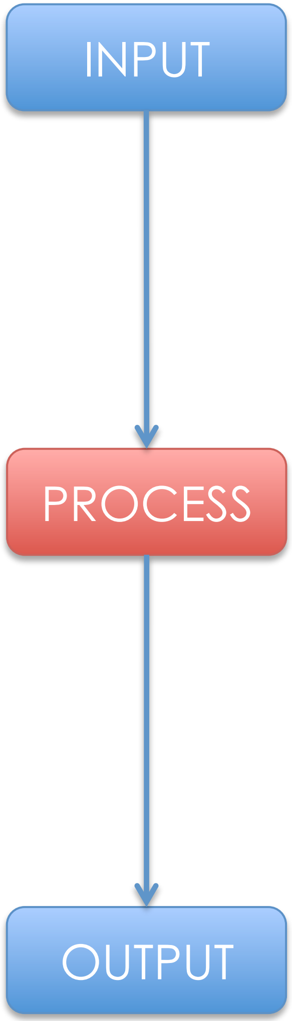
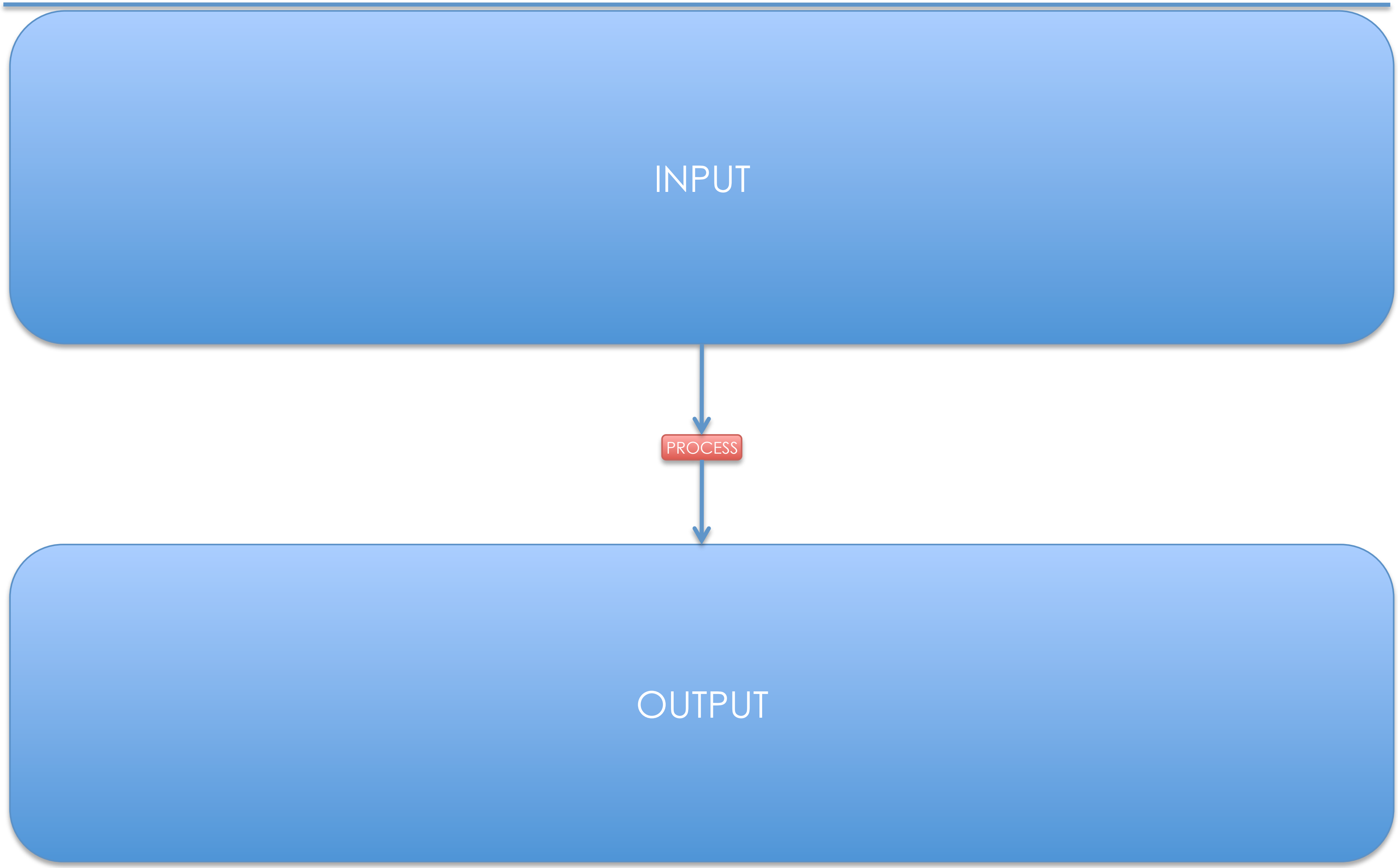
Map Reduce

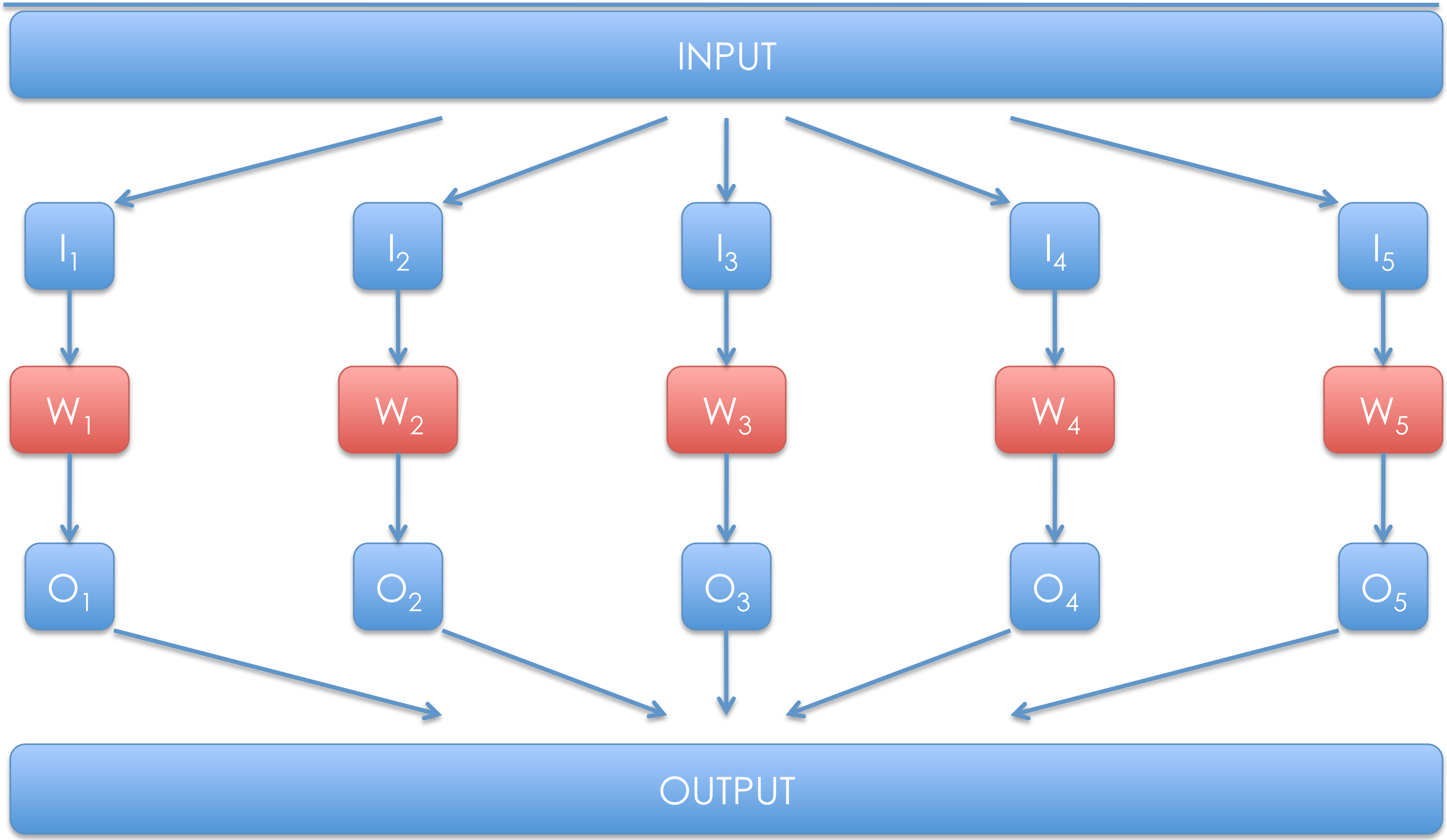
Typical application



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?

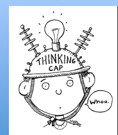

http://www.duiops.net/seresvivos

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

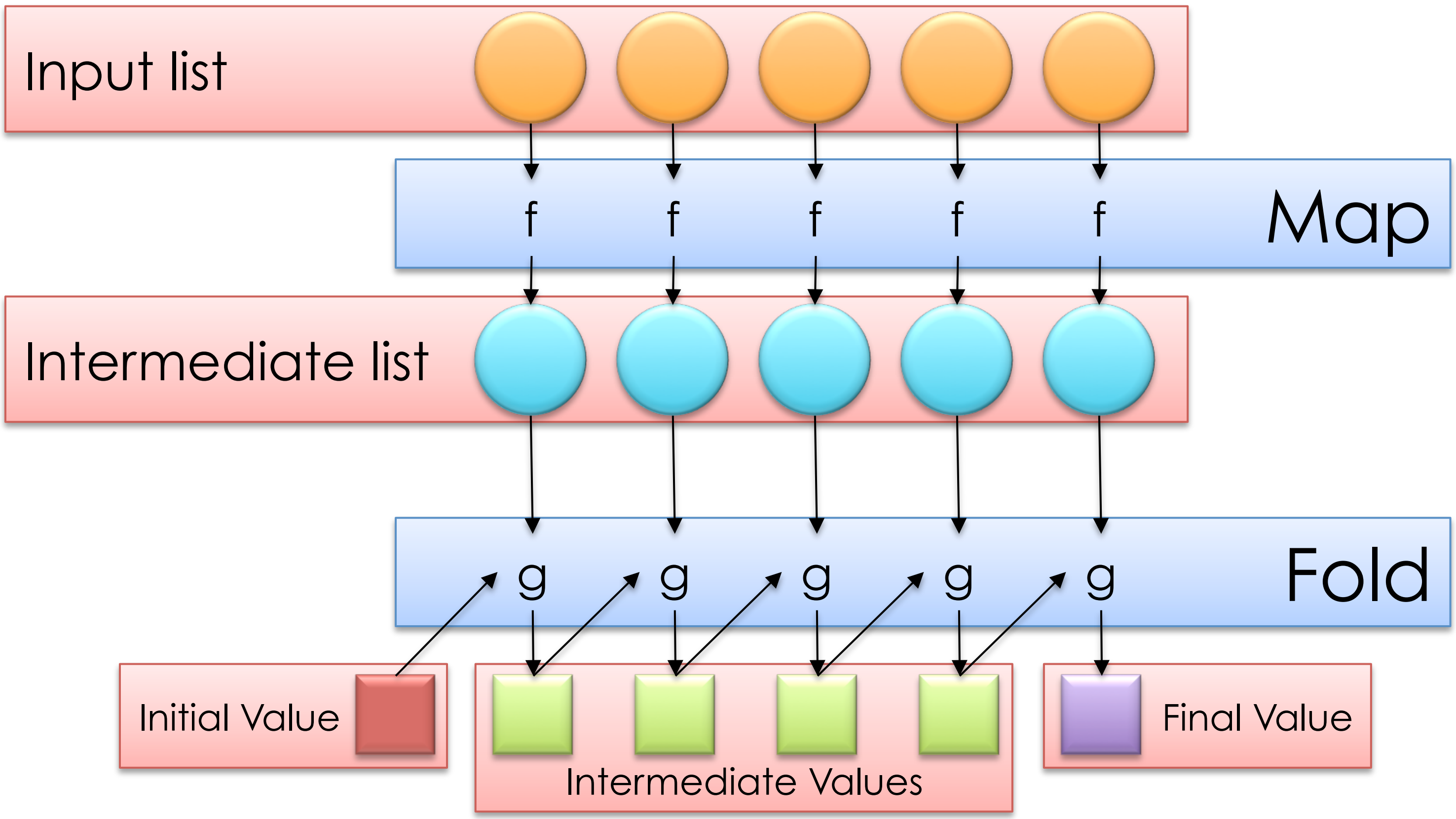


Map Reduce Programming Model

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

UNIVERSITÀ DI PISA

From functional programming…



...To MapReduce

- Programmers specify two functions:

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

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

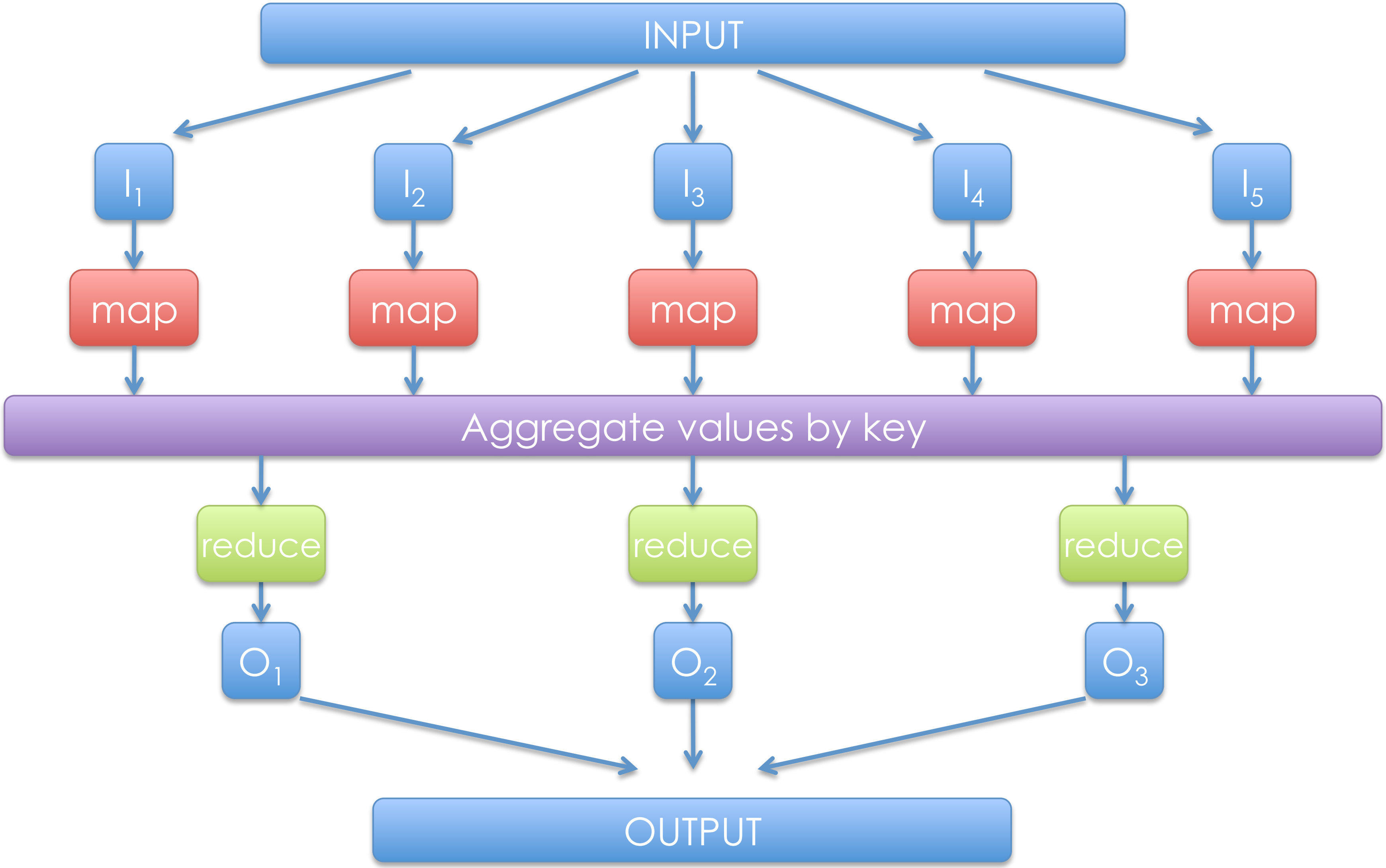
- All values with the same key are sent to the same reducer

- 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)

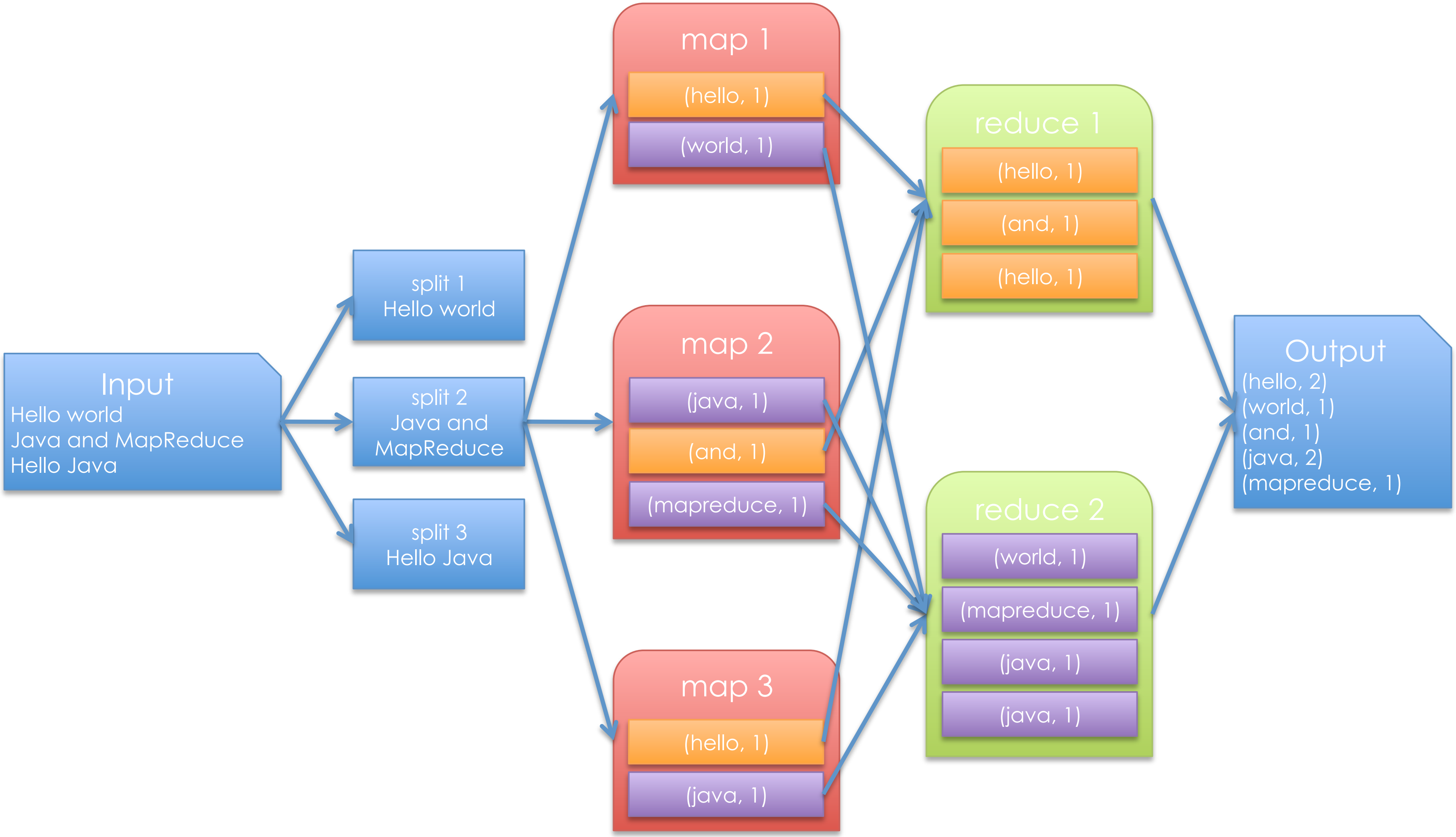


Example (I)

class MAPPER
  method MAP(docid a, doc d)
    for all term $t \in$ doc d **do**
      EMIT(term t, count 1)

class REDUCER
  method REDUCE(term t, counts $[c_1, c_2, \ldots]$)
    $sum \leftarrow 0$
    for all count $c \in$ counts $[c_1, c_2, \ldots]$ **do**
      $sum \leftarrow sum + c$
    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) $\rightarrow$ partition for k_2

  – Often a simple hash of the key, e.g., hash(key) 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

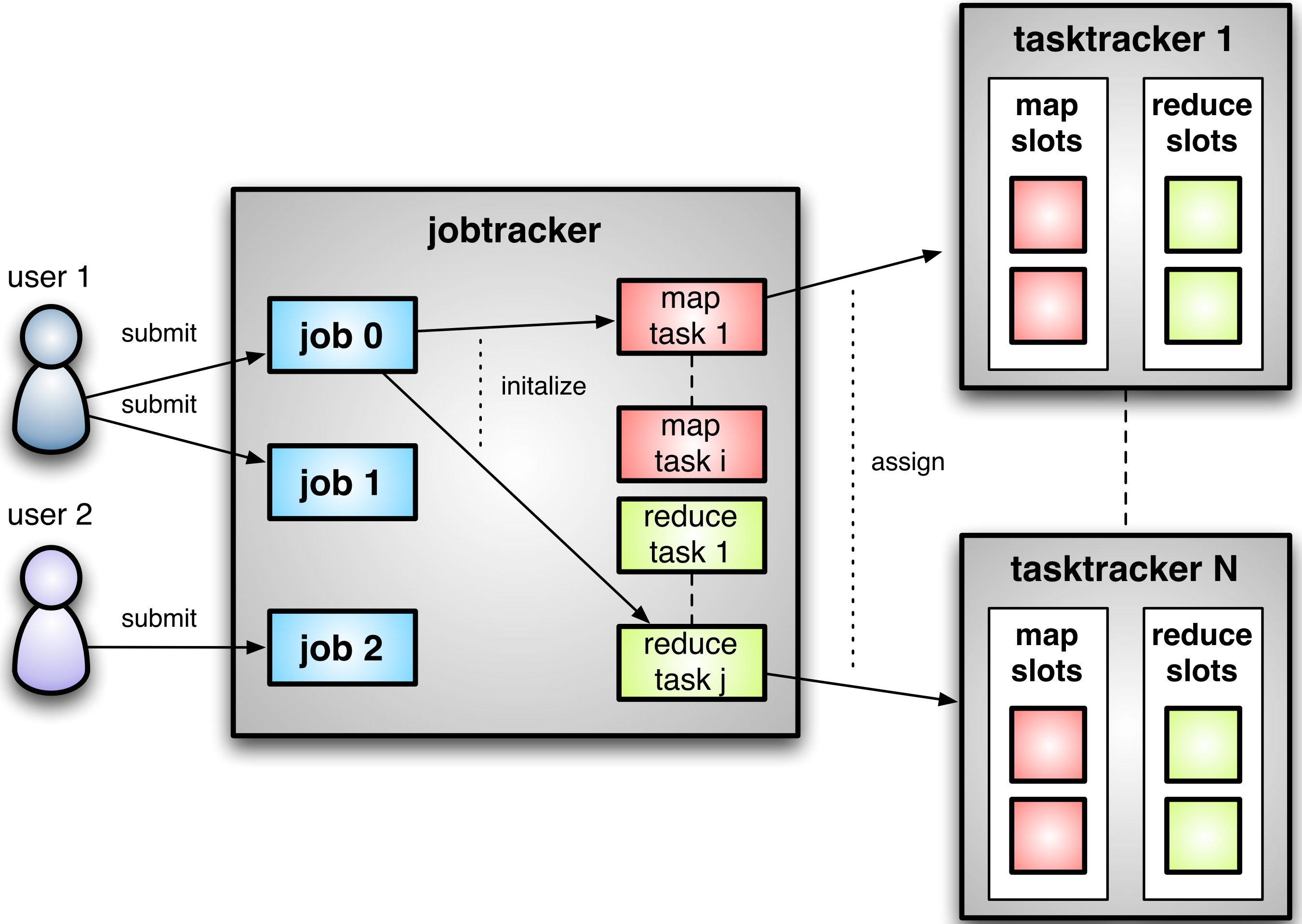
  – Used as an optimization to reduce network traffic

MapReduce Terminology

- Job
- Task
- Slot
- JobTracker
  - Accepts Map/Reduce jobs submitted by users
  - Assigns Map and Reduce tasks to Task Trackers
  - Monitors task and Task Tracker status, re-executes tasks upon failure
- TaskTracker
  - Run Map and Reduce tasks upon instruction from the Job Tracker
  - Manage storage and transmission of intermediate output
- Splits
  - Data locality optimization

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

Runtime Architecture



MapReduce Scheduling

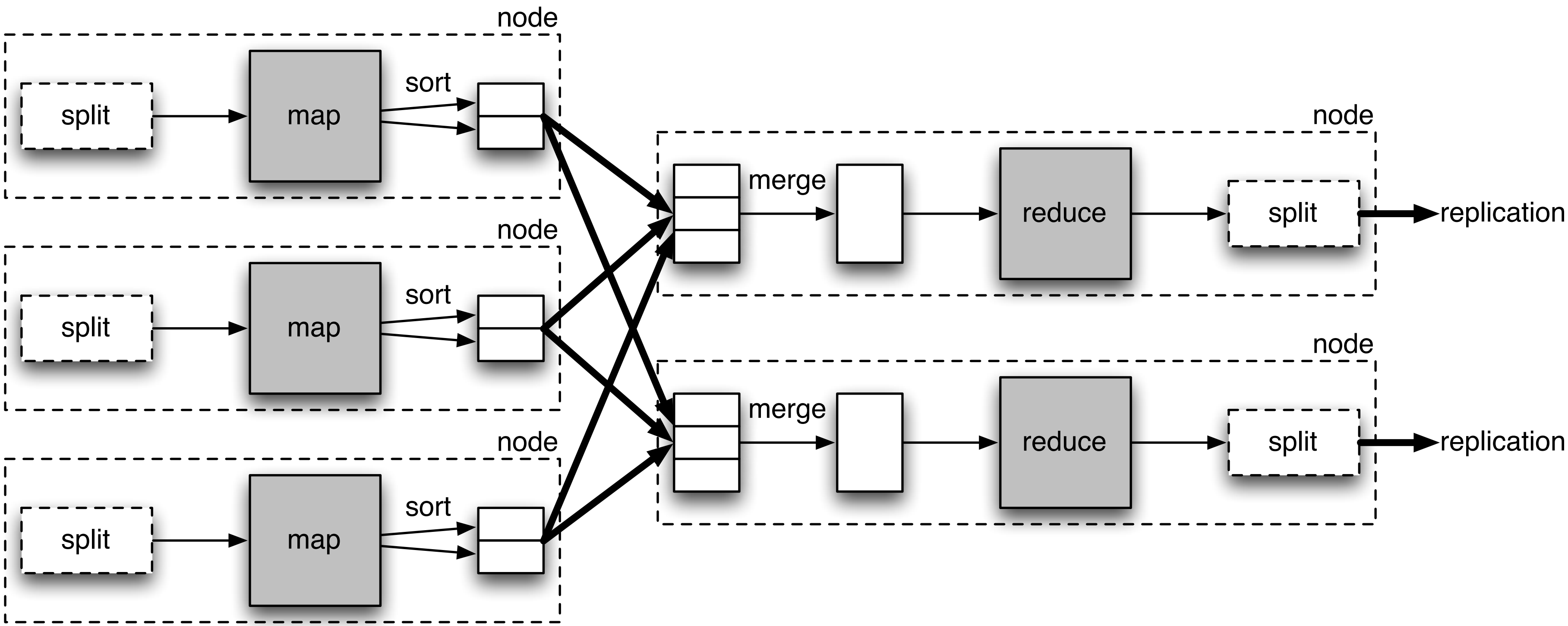
- One master, many workers
  - Input data split into M map tasks (typically 64 MB in size)
  - Reduce phase partitioned into R reduce tasks (hash(k) mod R)
  - Tasks are assigned to workers dynamically
  - Often: M=200,000; R=4000; workers=2000
- Master assigns each map task to a free worker
  - Considers locality of data to worker when assigning a task
  - Worker reads task input (often from local disk)
  - Worker produces R local files containing intermediate k/v pairs
- Master assigns each reduce task to a free worker
  - Worker reads intermediate k/v pairs from map workers
  - Worker sorts & applies user's reduce operation to produce the output

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

MapReduce Speculative Execution

- Problem: Stragglers (i.e., slow workers) significantly lengthen the completion time
  - Other jobs may be consuming resources on machine
  - Bad disks with soft (i.e., correctable) errors transfer data very slowly
  - Other weird things: processor caches disabled at machine init
- Solution: Close to completion, spawn backup copies of the remaining in-progress tasks.
  - Whichever one finishes first, "wins"
- Additional cost: a few percent more resource usage
- Example: A sort program without backup = 44% longer.

Dataflow



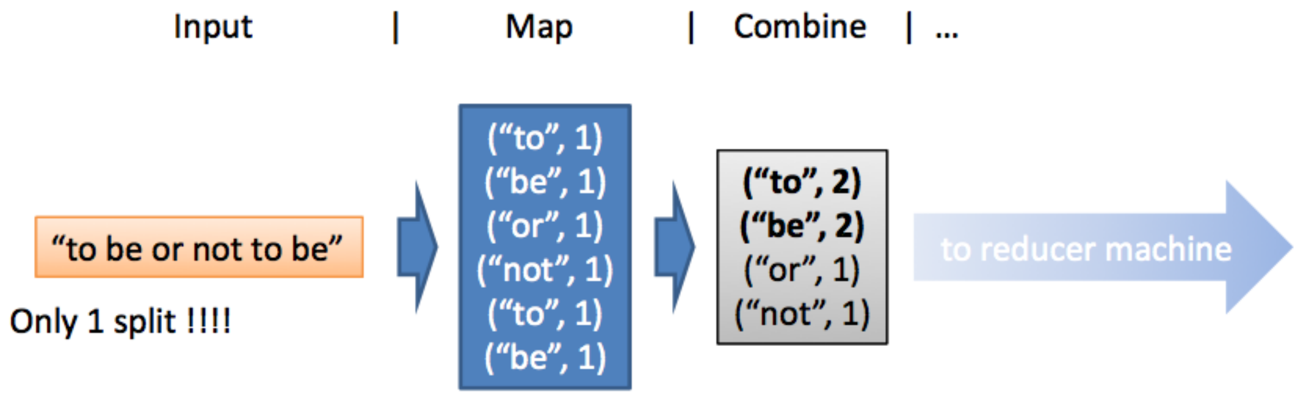
ISTITUTO DI SCIENZA E TECNOLOGIE
DELL'INFORMAZIONE "A. FAEDO"

Optimizations: output ordering

- Applications can define the sort ordering and the partitions of the output (@map)
- Default **partitioner** evenly distributes records
- hashcode(key) mod NR
- Partitioner could be overridden

Optimizations: output aggregation

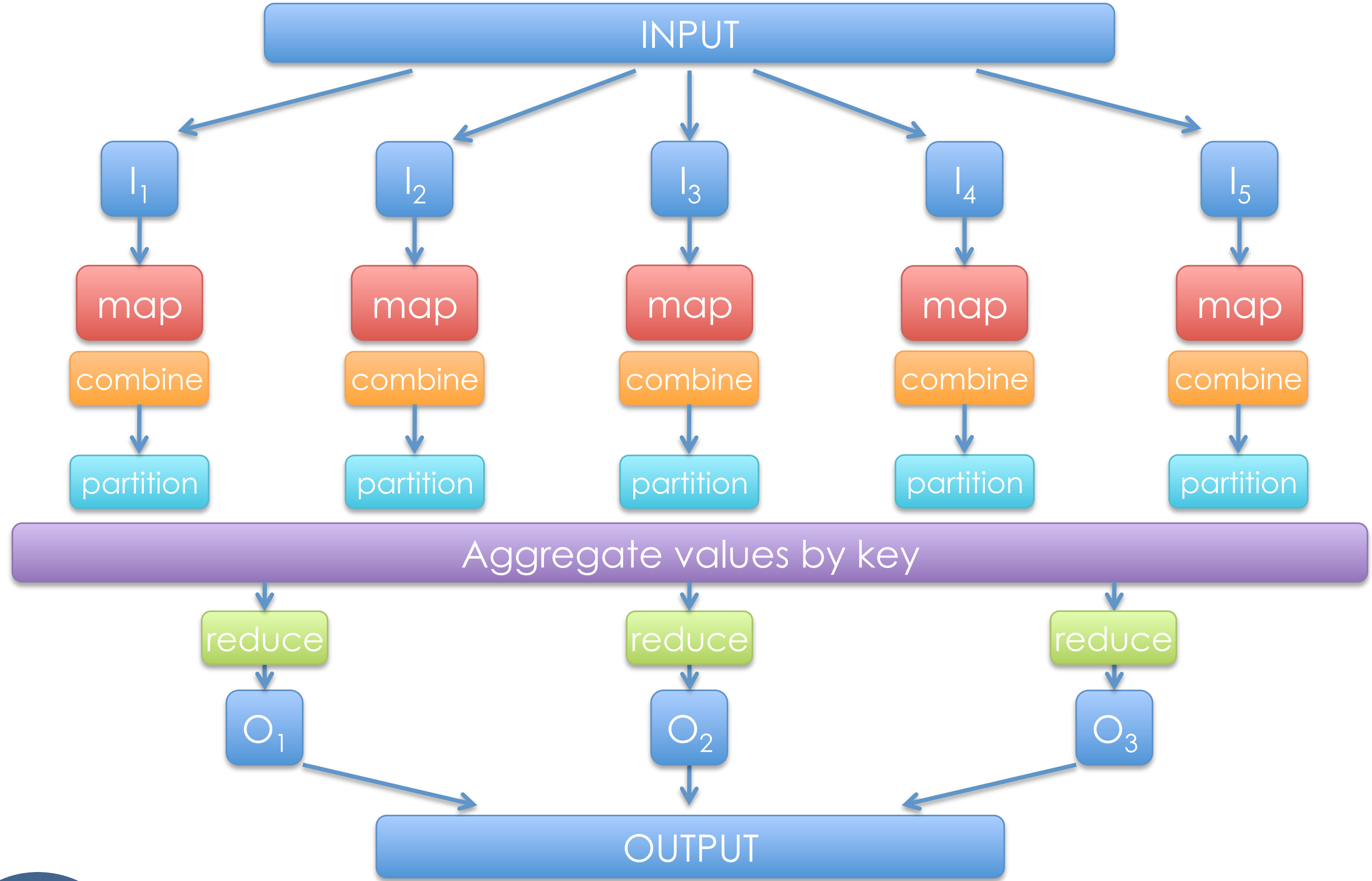
- Aggregation for jobs with reducers that merge values into a single value



- Combiner functions can run on same machine as a mapper
- Causes a mini-reduce phase to occur before the real reduce phase, to save bandwidth

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

Programming Model (complete)



Performance

- Maximizing Map input transfer rate
  - Input Locality
  - Minimal deserialization overhead
- Small intermediate output
  - M x R transfers over the network
  - Minimize/compress transfers
  - Avoid shuffling/sorting if possible (e.g. map-only computations)
  - Use combiners and/or partitioners!!!
  - Compress everything (automatic)
- Opportunity to Load Balance
- Changing algorithm to suit architecture yields best implementation

How many tasks?

- mapred.tasktracker.map.tasks.maximum
- mapred.tasktracker.reduce.tasks.maximum
- Tradeoffs:
  - Number of cores
  - Amount of memory
  - Number of local disks
  - Amount of local scratch space
  - Number of processes
- Consider resources consumed by TaskTracker & Datanode processes

Exercise

- Input data: 15 TB of doubles
- Output data: standard deviation

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

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

Implementation

- Two Map-Reduce stages
  - First stage computes mean
  - Second stage computes std dev
- Stage 1: Compute Mean
  - Map Input: a subset of input data per mapper
  - Map Output: fixed key, mean of the input subset
  - Single Reducer
  - Reduce Input: set of partial means
  - Reduce Output: mean
- Stage 2: Compute Standard deviation
  - Map Input: a subset of input data and mean per mapper
  - Map Output: fixed key, (sum(x_i – mean(x))^ of the input subset
  - Single Reducer
  - Reduce Input: set of partial results
  - Reduce Output: standard deviation

...but...

$$\sigma = \sqrt{\frac{1}{N}\left(\sum_{i=1}^{N} x_i^{\,2} - N\bar{x}^2\right)}$$

- Single Map-Reduce stage
  - Map Input: a subset of input data per mapper
  - Map Output: fixed key, sum(x^2) and mean of the input subset
  - Single Reducer
  - Reduce Input: set of partial results
  - Reduce Output: standard deviation

Only a single pass over large input instead of two!

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