

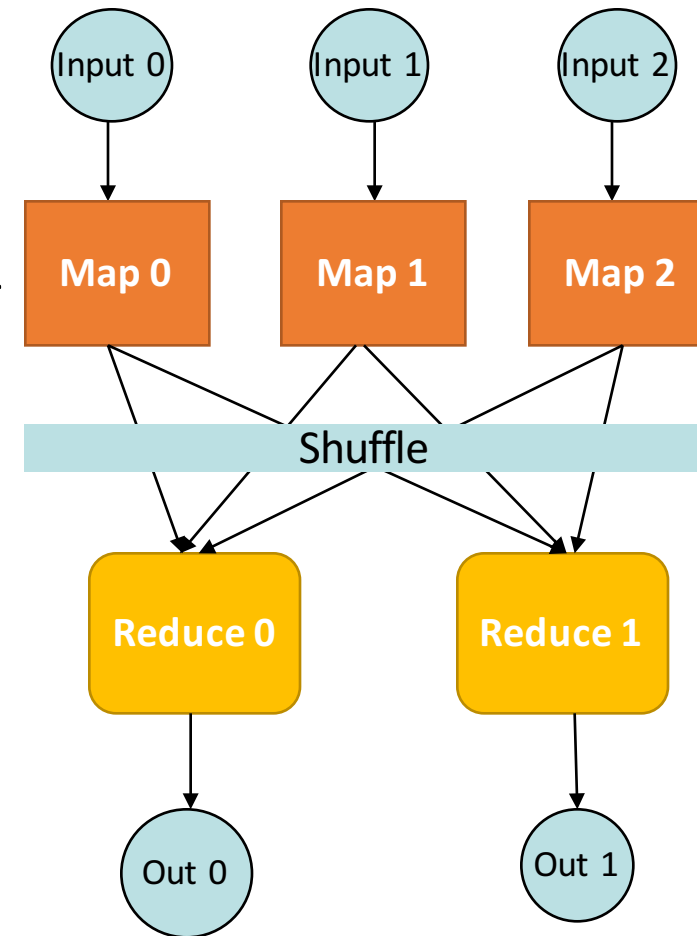
INF 553: Foundations and Applications of Data Mining

Map-Reduce:
Scheduling and Data Flow
Combiners and Partition Functions

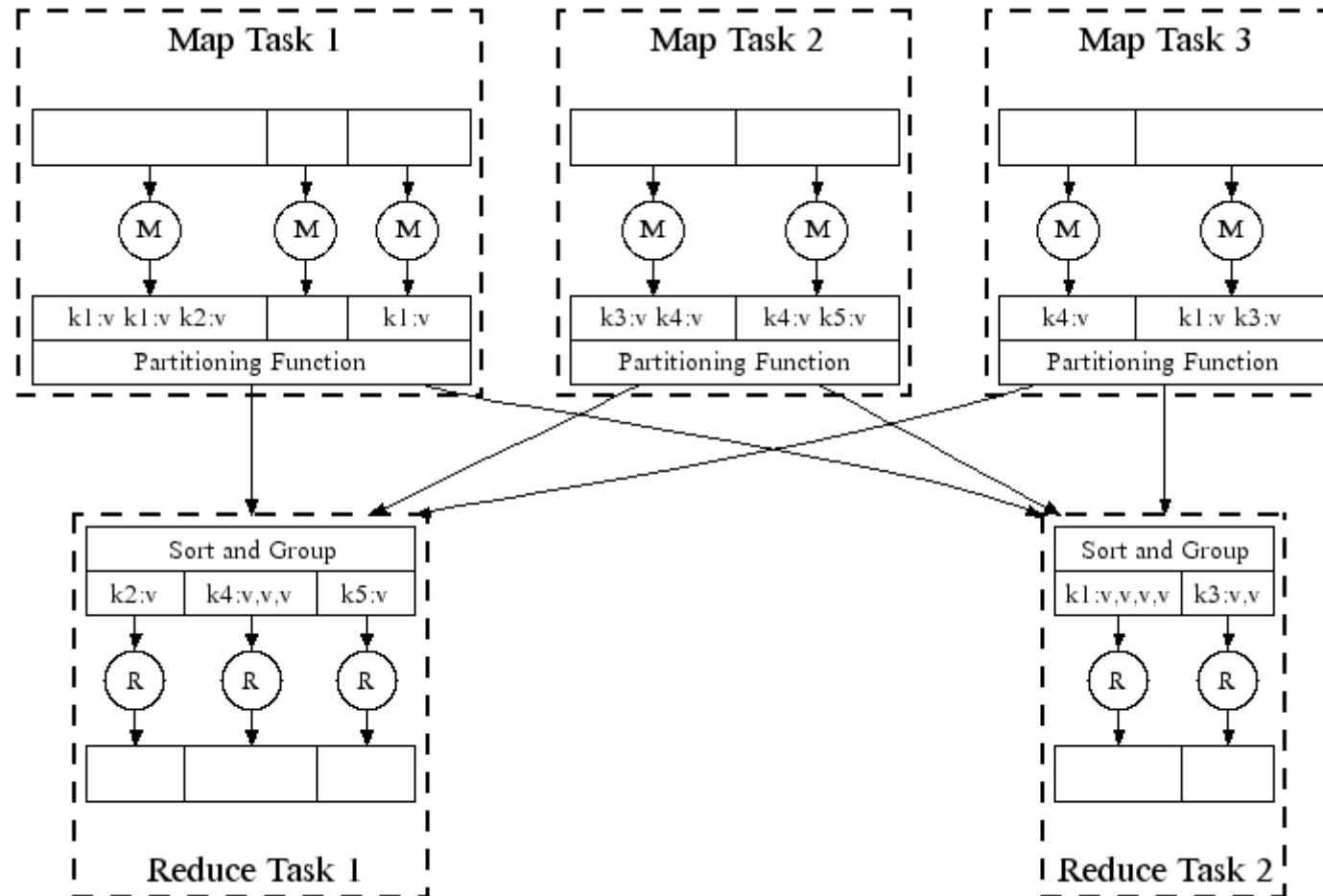
Dr. Anna Farzindar
farzinda@usc.edu

Map-Reduce

- Programmer specifies:
 - Map and Reduce and input files
- **Workflow:**
 - Read inputs as a set of key-value-pairs
 - **Map** transforms input kv-pairs into a new set of k'v'-pairs
 - Sorts & Shuffles the k'v'-pairs to output nodes
 - All k'v'-pairs with a given k' are sent to the same **reduce**
 - **Reduce** processes all k'v'-pairs grouped by key into new k''v''-pairs
 - Write the resulting pairs to files
- All phases are distributed with many tasks doing the work.



Map-Reduce: In Parallel



All phases are distributed with many tasks doing the work

MapReduce: Environment

MapReduce environment takes care of:

- **Partitioning** the input data
- **Scheduling** the program's execution across a set of machines
- Performing the **group by key** step
 - In practice this is the bottleneck
- Handling machine **failures**
- Managing required inter-machine **Communication.**

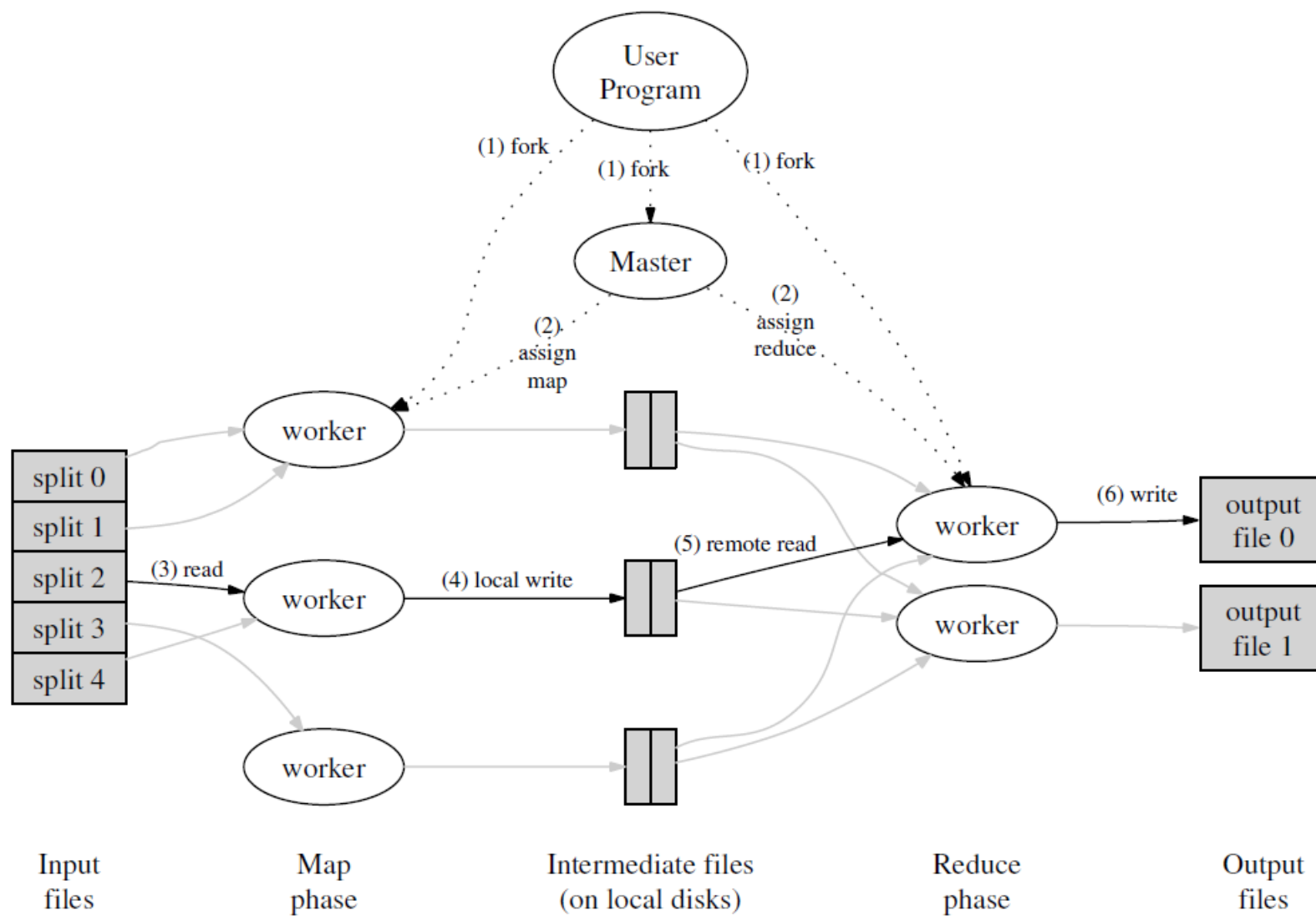
Data Flow

- Input and final output are stored on a distributed file system (HDFS):
 - Scheduler tries to schedule map tasks “close” to physical storage location of input data
- Intermediate results are stored on local FS of Map and Reduce workers
- Output is often input to another MapReduce task.

Coordination: Master

- **Master node takes care of coordination:**
 - **Task status:** (idle, in-progress, completed)
 - **Idle tasks** get scheduled as workers become available
 - When a map task **completes**, it sends the master the **location and sizes** of its intermediate files, one for each reducer
 - Master pushes this info **to reducers**
- **Master pings workers** periodically to detect failures.

Coordination



Dealing with Failures

■ Map worker failure

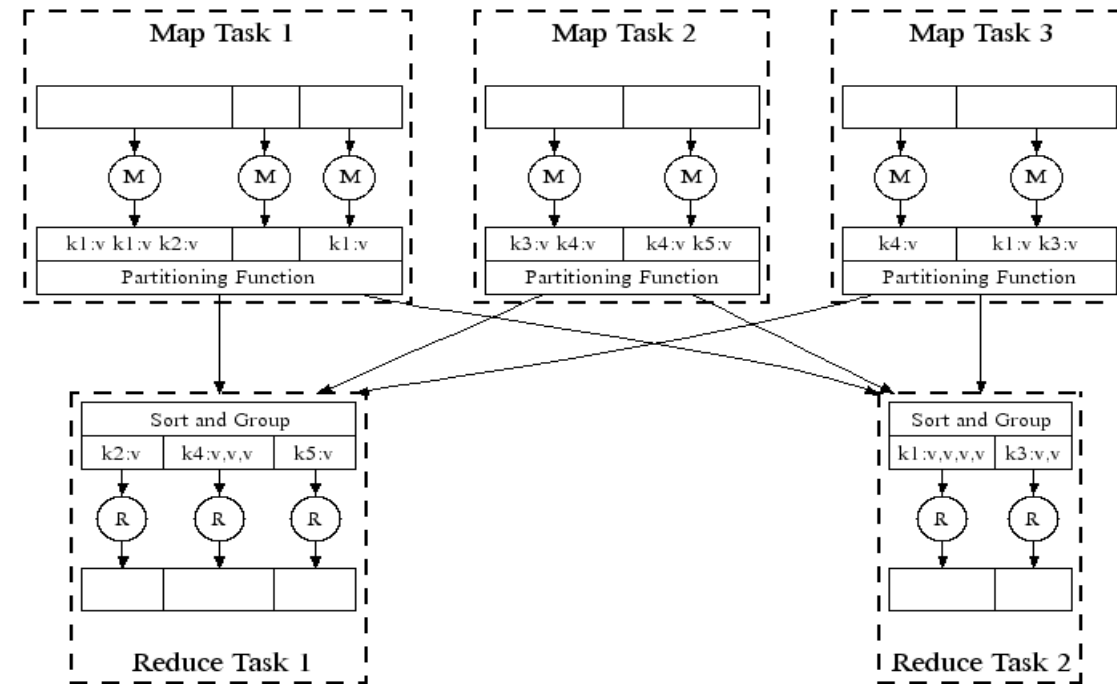
- **Map tasks** completed or in-progress at worker are **reset to idle**
- **Reduce workers** are **notified** when task is rescheduled on another worker

■ Reduce worker failure

- Only **in-progress** tasks are reset to idle
- Reduce task is restarted

■ Master failure Could handle (master failure unlikely, one machine lost in 1000days)

- MapReduce task is aborted and client is notified.

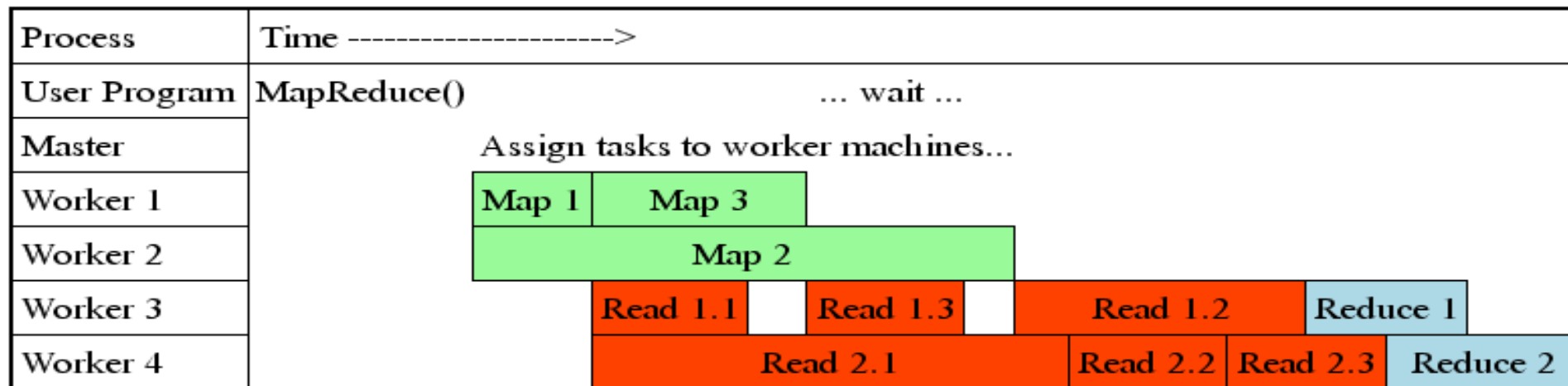


How many Map and Reduce jobs?

- **M** map tasks, **R** reduce tasks
- **Rule of a thumb:**
 - Make **M** much larger than the number of nodes in the cluster
 - One DFS chunk per map is common
 - Improves dynamic load balancing and speeds up recovery from worker failures
- **Usually R is smaller than M**
 - Because output is spread across **R** files
- **Google example:** Often use 200,000 map tasks, 5000 reduce tasks on 2000 machines.

Task Granularity & Pipelining

- **Fine granularity tasks** -> Granularity affects the performance of parallel computers. Using fine grains or small tasks results in more parallelism and hence increases the seedup.
 - -> many more **map tasks** than machines
- Minimizes **time for fault recovery**
- Can do pipeline **shuffling** with map execution
- Better dynamic **load balancing**.



Refinements: Backup Tasks

■ Problem

- Slow workers significantly lengthen the job completion time:
 - Other jobs on the machine
 - Bad disks
 - Weird things

■ Solution

- Near end of phase, spawn backup **copies of tasks**
 - Whichever one finishes first “wins”

■ Effect

- Dramatically shortens job completion time.

Refinement: Combiners

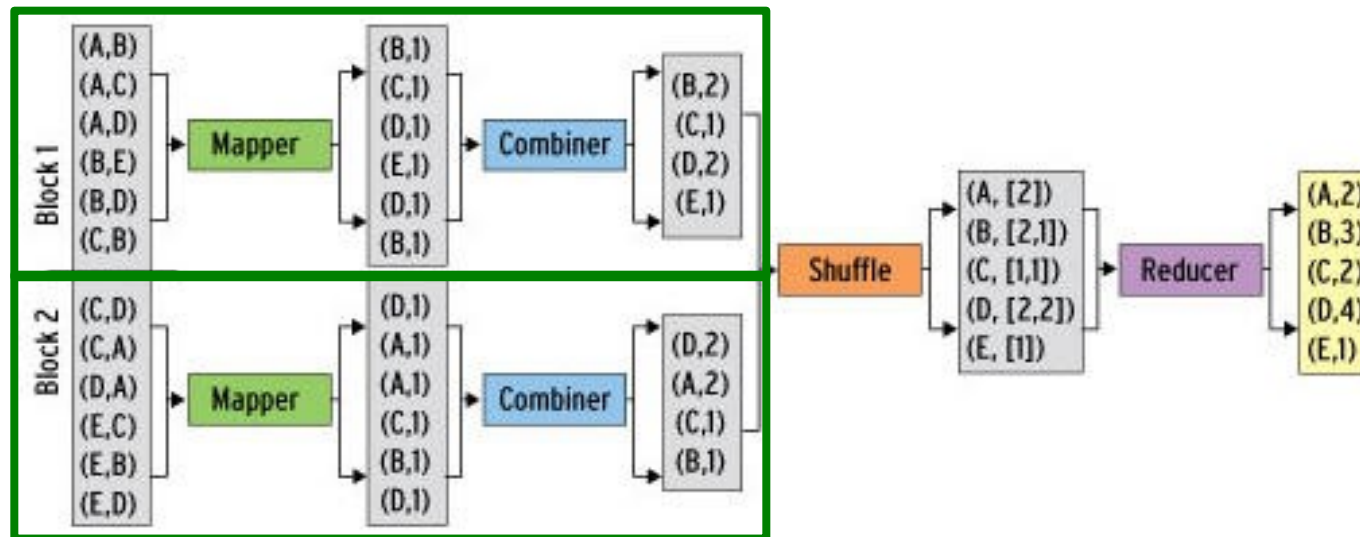
- Combiners are **an optimization** in MapReduce
 - allow for local aggregation **before the shuffle and sort** phase
- When the **map operation outputs its pairs** they are already available in **memory**
- For efficiency reasons, sometimes it makes sense to take advantage of this fact by supplying a combiner class to perform a **reduce-type function**.

Refinement: Combiners

- If a combiner is used then the **map key-value** pairs are **not** immediately written **to the output**
 - They will be **collected in lists**, one list per each key value
- When a certain number of key-value pairs have been written,
- **This buffer** is flushed by passing all the values of each key to the combiner's **reduce method** and
- **Outputting** the key-value pairs of the combine operation as if they were created by the **original map operation**.

Refinement: Combiners

- **Back to our word counting example:**
 - **Combiner** combines the values of all keys of a single mapper (single machine):



- Much less data needs to be copied and shuffled!
- Works if reduce function is commutative and associative.

Commutative and associative

- A binary operation is **commutative** if changing the order of the operands does not change the result.
 - A binary operation on a set S is called commutative if:
 - $x * y = y * x$ for all x, y in S
- A binary operation on a set S is called **associative** if it satisfies the associative law:
$$(x * y) * z = x * (y * z)$$
- **Example:** functions both commutative and associative
 - $\text{Max}()$.

Refinement: Combiner

- Combiner trick works only if reduce function is commutative and associative

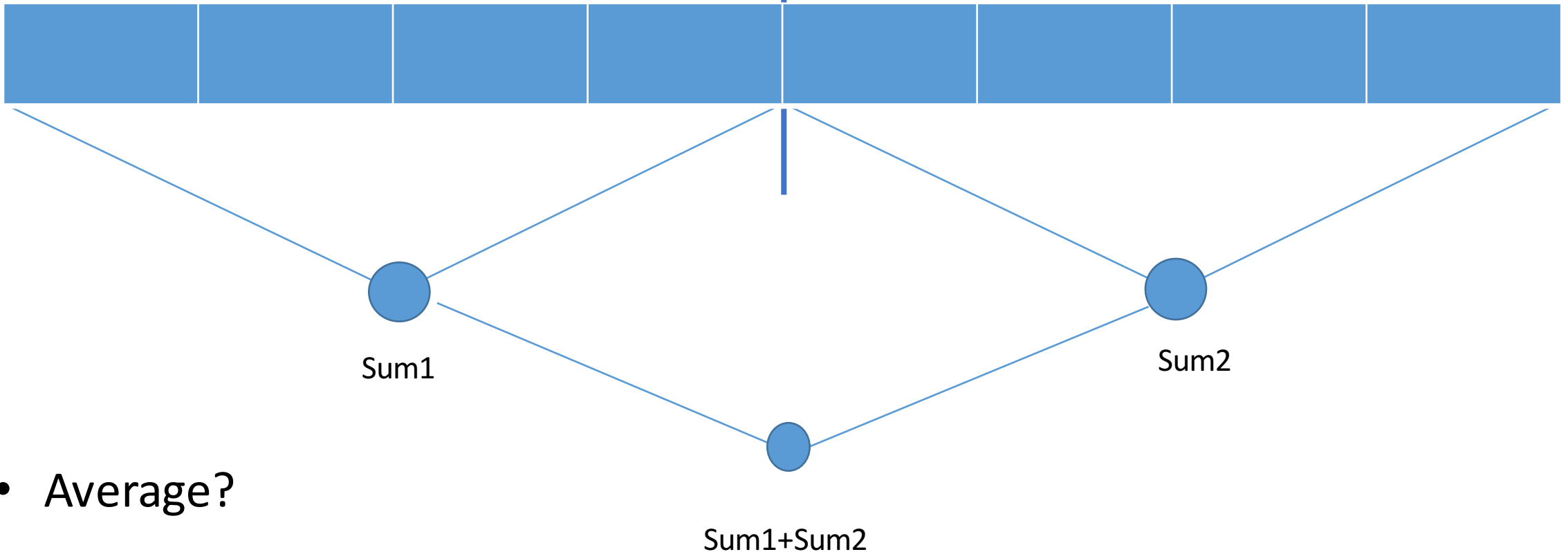
- Sum

$$a+b = b+a$$

$$(a+b) + c = a + (b+c)$$

(commutative)

(associative)

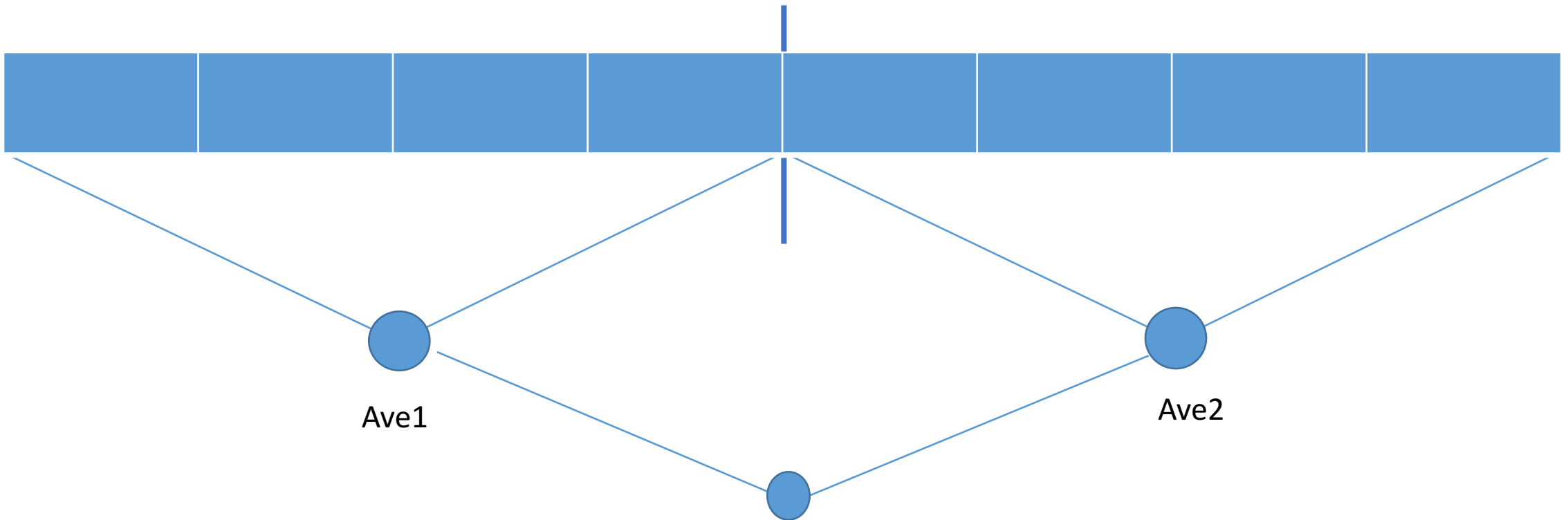


- Average?

Refinement: Combiner

Combiner trick works only if reduce function is commutative and associative

Average:

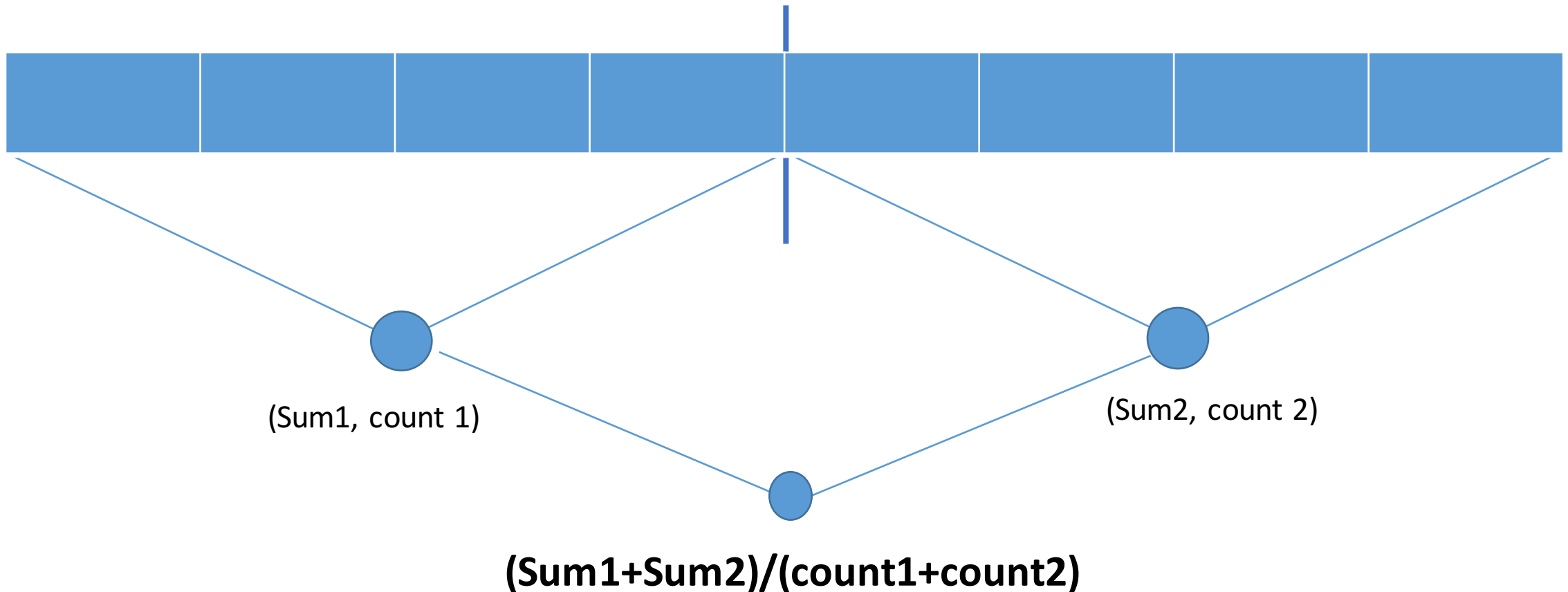


$(Ave1+Ave2)/2$ NOT Correct

Refinement: Combiner

Combiner trick works only if reduce function is commutative and associative

Average: change of (Key, Value)



Refinement: Combiner

- Combiner trick works only if reduce function is commutative and associative
- Sum
- Average
- Median?
 - > Not commutative and associative then can't use combiner

Refinement: Partition Function

- **Want to control how keys get partitioned**
 - Inputs to map tasks are created by contiguous splits of input file
 - **Reducer** needs to ensure that **records with the same intermediate key end up at the same worker**
- **System uses a default partition function:**
 - **$\text{hash}(\text{key}) \bmod R$**
- **Sometimes useful to override the hash function:**
 - E.g., want to have **alphabetical or numeric ranges** going to different Reduce tasks
 - E.g., **$\text{hash}(\text{hostname}(\text{URL})) \bmod R$** ensures URLs from a host end up in the same output file.

Implementations

- **Google's MapReduce**
 - Not available outside Google
- **Hadoop**
 - An open-source implementation in Java
 - Uses HDFS for stable storage
 - Download: <http://hadoop.apache.org/releases.html>
 - Many variations.

Cloud Computing

- Ability to rent computing by the hour
 - Additional services e.g., persistent storage
- Amazon's "Elastic Compute Cloud" ([EC2](#))
 - Aster Data and Hadoop can both be run on [EC2](#)
 - [S3](#) (stable storage)
 - **Amazon S3** has a simple web services interface: to store and retrieve any amount of data, at any time, from anywhere on the web.
 - Elastic Map Reduce ([EMR](#)).

Summary

- MapReduce environment
- Data Flow
- Master node and coordination
- Refinement
- Combiner: useful for saving network bandwidth
- Implementation