Where We Are

Machine Learning Systems

Big Data — 2010 - Now

Cloud — 2000 - 2016

Foundations of Data Systems — 1980 - 2000

Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models
  - Job execution
  - Workflow
- Beyond MapReduce

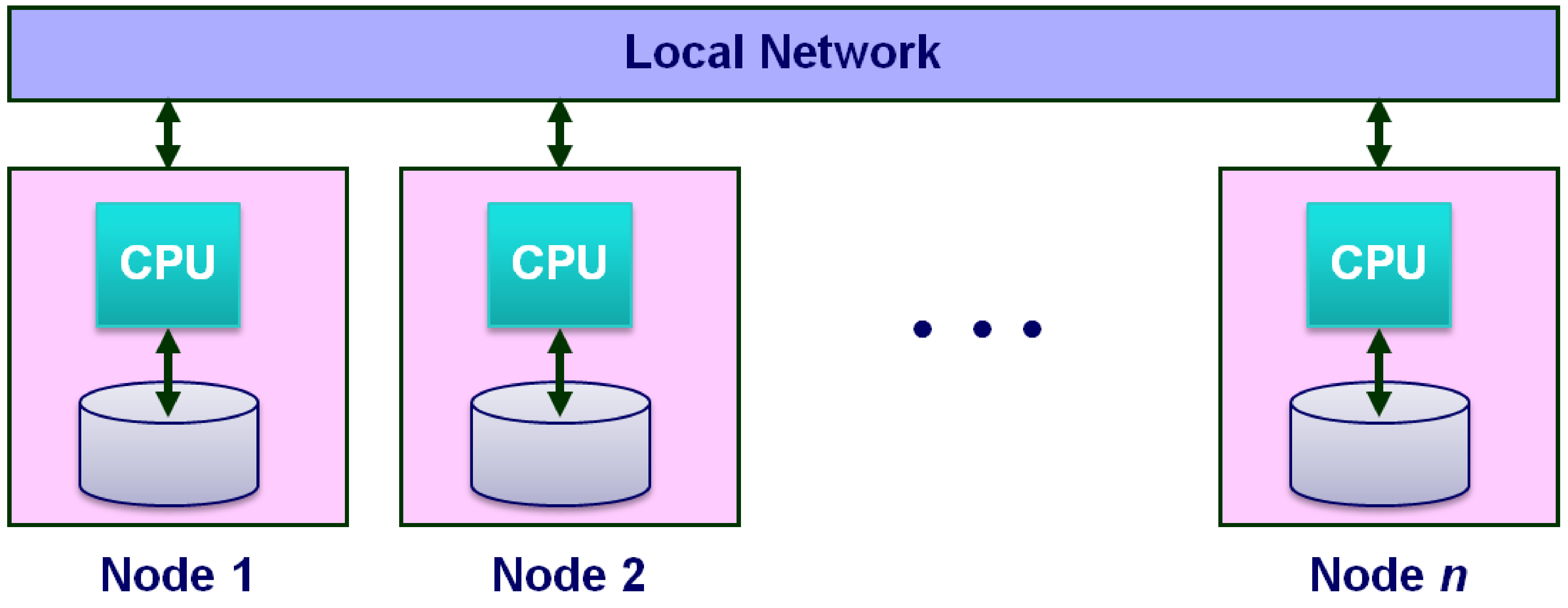
Historical Context of Map-reduce

For Computation That Accesses 1 TB in 5 minutes

- Data distributed over 100+ disks
- Compute using 100+ processors
- Connected by gigabit Ethernet (or equivalent)

System Requirements

- Lots of disks
- Lots of processors
- Located in close proximity
  - Within reach of fast, local-area network

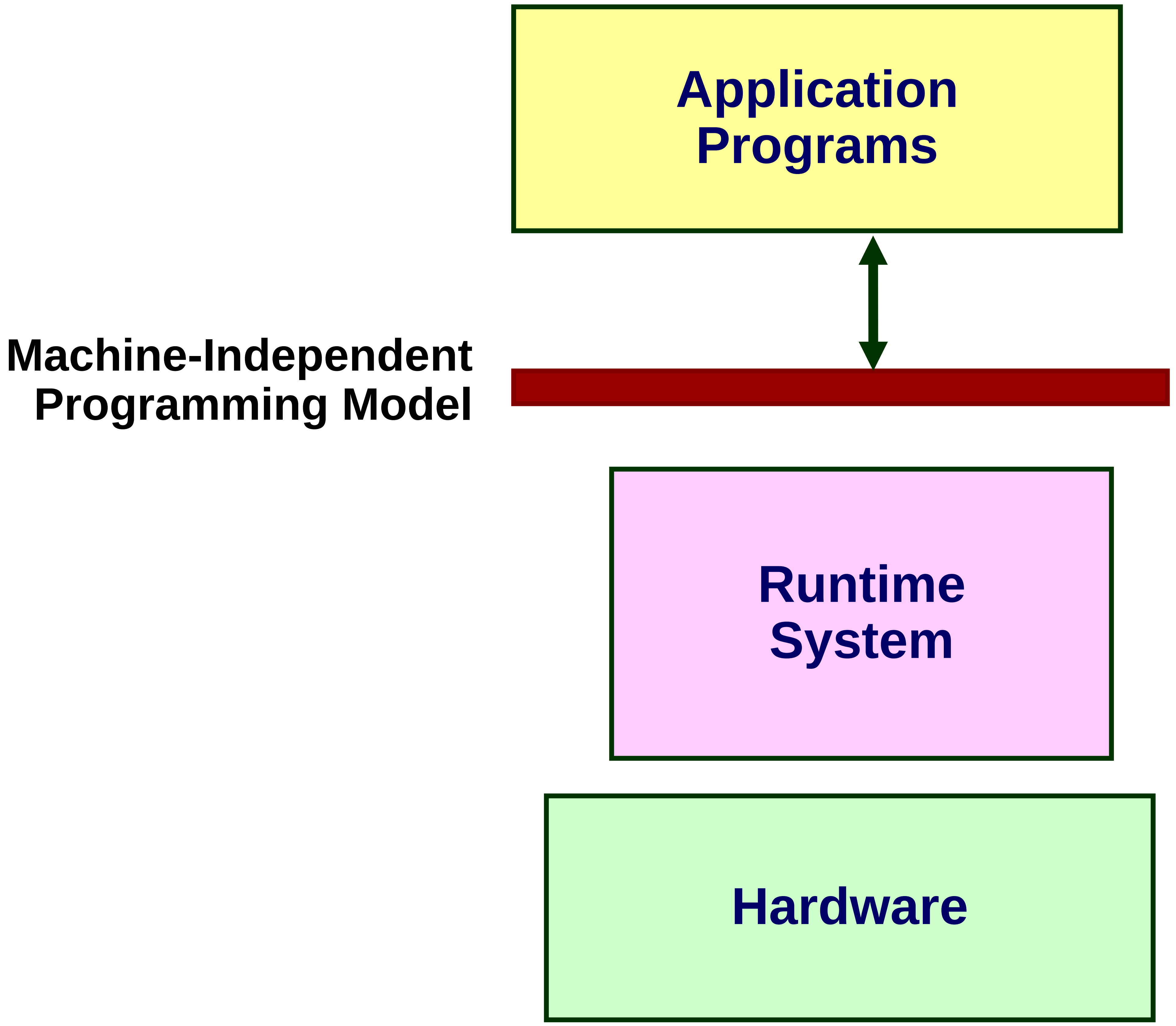


Today’s topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models (API)
  - Job execution (runtime)
  - Workflow
- Beyond MapReduce

Ideal Cluster Programming Model

- Application programs written in terms of high-level operations on data
  - User-facing
- Runtime system controls scheduling, load balancing, …
  - System implementations
- After Map-reduce papers:
  - Many system research papers follow this template



MapReduce: Simplified Data Processing on Large Clusters

by Jeffrey Dean and Sanjay Ghemawat

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world tasks. Users specify the computation in terms of a *map* and a *reduce* function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks. Programmers find the system easy to use: more than ten thousand distinct MapReduce programs have been implemented internally at Google over the past four years, and an average of one hundred thousand MapReduce jobs are executed on Google's clusters every day, processing a total of more than twenty petabytes of data per day.

ANNALS OF TECHNOLOGY

THE FRIENDSHIP THAT MADE GOOGLE HUGE

Coding together at the same computer, Jeff Dean and Sanjay Ghemawat changed the course of the company—and the Internet.

By James Somers

December 3, 2018

https://www.newyorker.com/magazine/2018/12/10/the-friendship-that-made-google-huge

TF-IDF

TF-IDF is a measure of originality of a word by comparing the number of times a word appears in a doc with the number of docs the word appears in.

$$\text{TF-IDF} = \text{TF}(t, d) \times \text{IDF}(t)$$

Term frequency

Number of times term t appears in a doc, d

Inverse document frequency

$$\log \frac{1 + n}{1 + df(d, t)} + 1$$

# of documents

Document frequency of the term t

TF-IDF Examples

Text 1	i love natural language processing but i hate python
Text 2	i like image processing
Text 3	i like signal processing and image processing

Term	and	but	hate	i	image	language	like	love	natural	processing	python	signal
IDF	0.47712	0.47712	0.4771	0	0.1760913	0.477121	0.1760913	0.477121	0.47712125	0	0.477121	0.477121

	and	but	hate	i	image	language	like	love	natural	processing	python	signal
Text 1	0	0.47712	0.4771	0	0	0.477121	0	0.477121	0.47712125	0	0.477121	0
Text 2	0	0	0	0	0.1760913	0	0.1760913	0	0	0	0	0
Text 3	0.47712	0	0	0	0.1760913	0	0.1760913	0	0	0	0	0.477121

Count the number of occurrences of word in a large collection of documents

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

- Functional programming
- Functions are stateless
- They takes an input, processes and output a result.
- Pros and Cons?

Data models

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

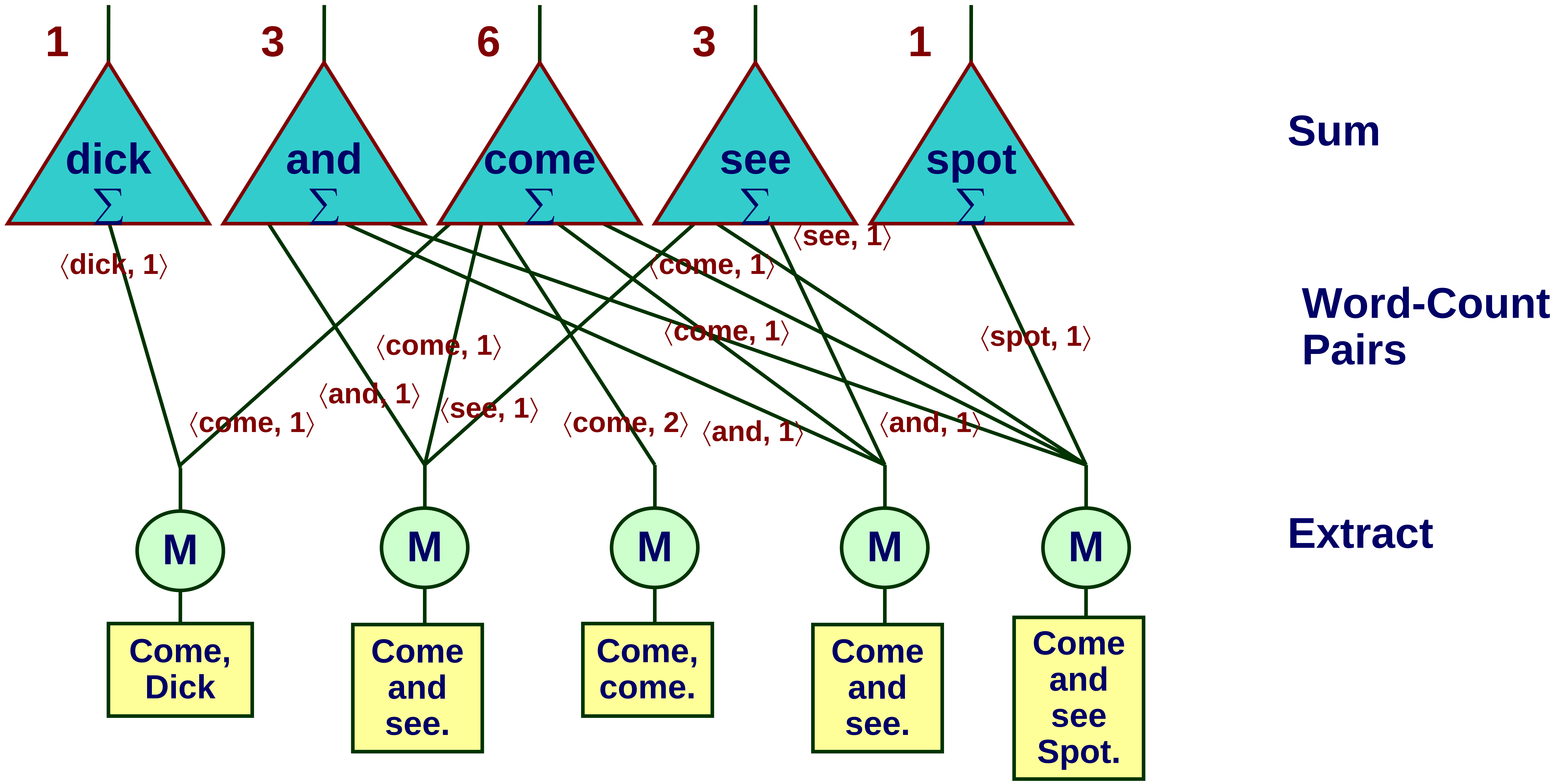
→

map	(k1,v1)	→	list(k2,v2)
reduce	(k2,list(v2))	→	list(v2)

MapReduce Example

- Create a word index of set of documents

Come, Dick	**Come and see.**	**Come, come.**	**Come and see.**	**Come and see Spot.**



- Map: generate ⟨word, count⟩ pairs for all words in document
- Reduce: sum word counts across documents

Discussion:

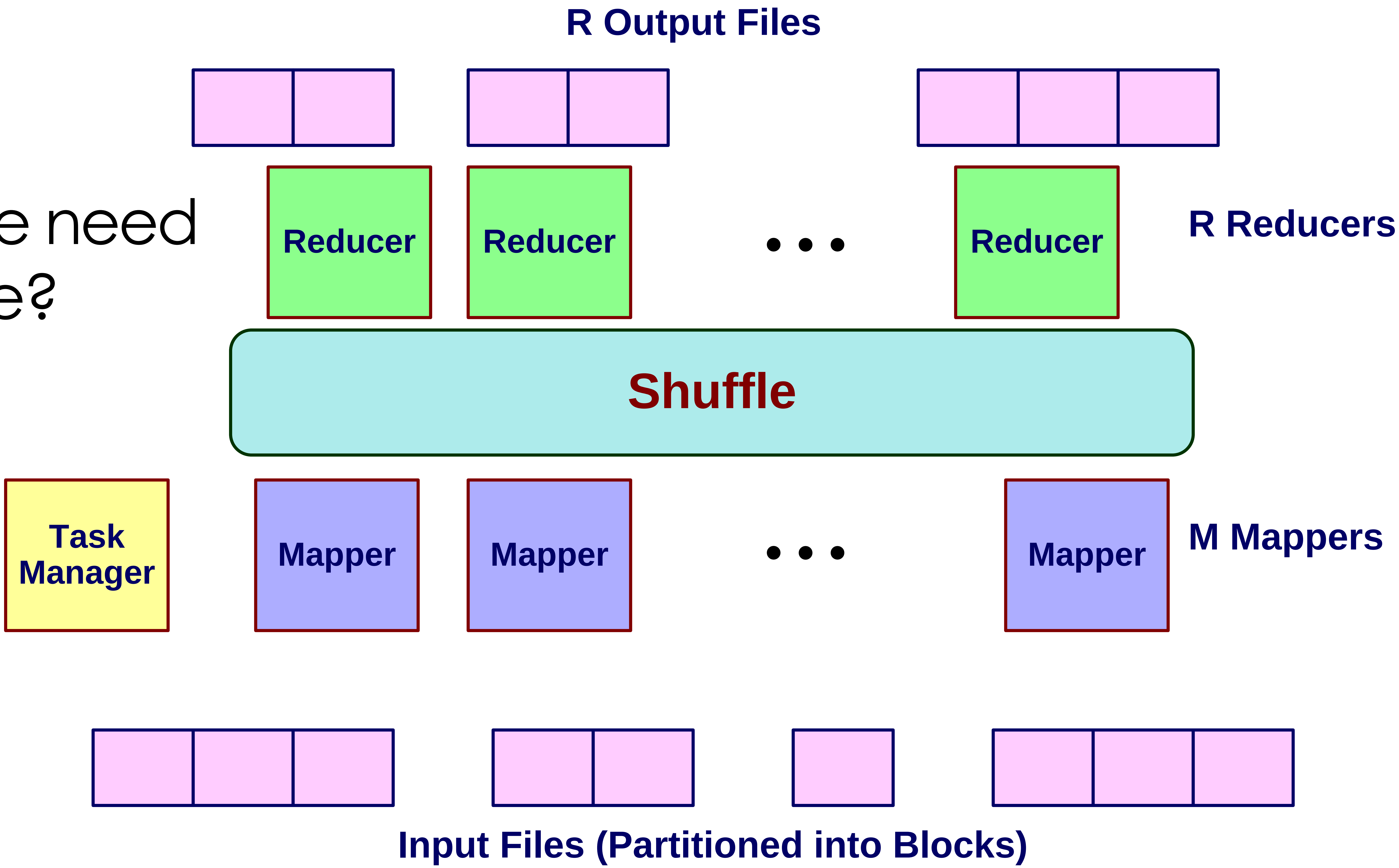
Other possible way to implement this using map-reduce?

Today’s topic: Batch Processing

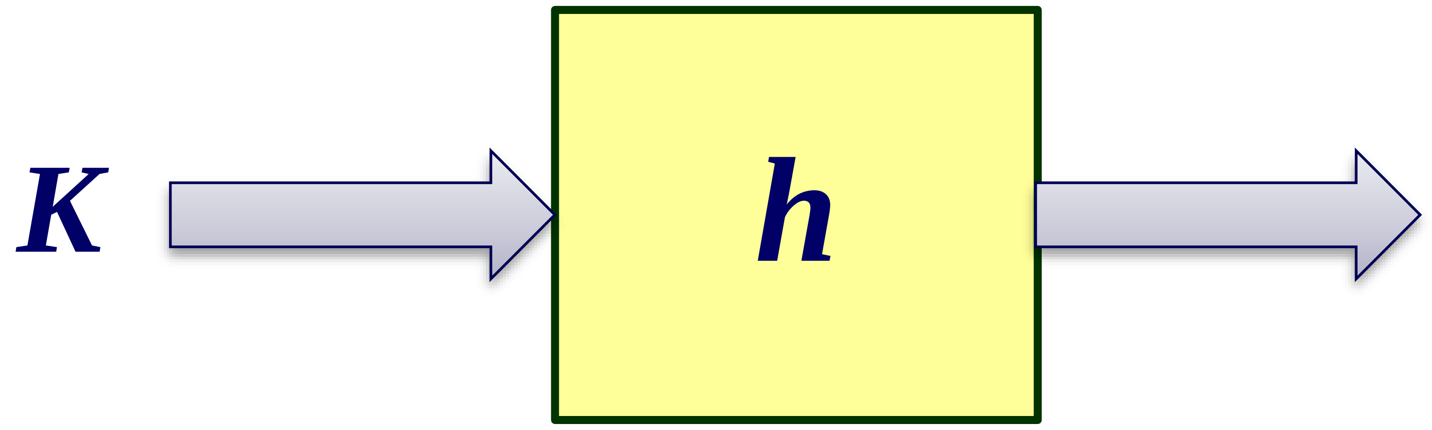
- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models
  - Job execution
  - Workflow
- Beyond MapReduce

MapReduce Execution (Runtime)

Why do we need shuffle?



Single Mapper



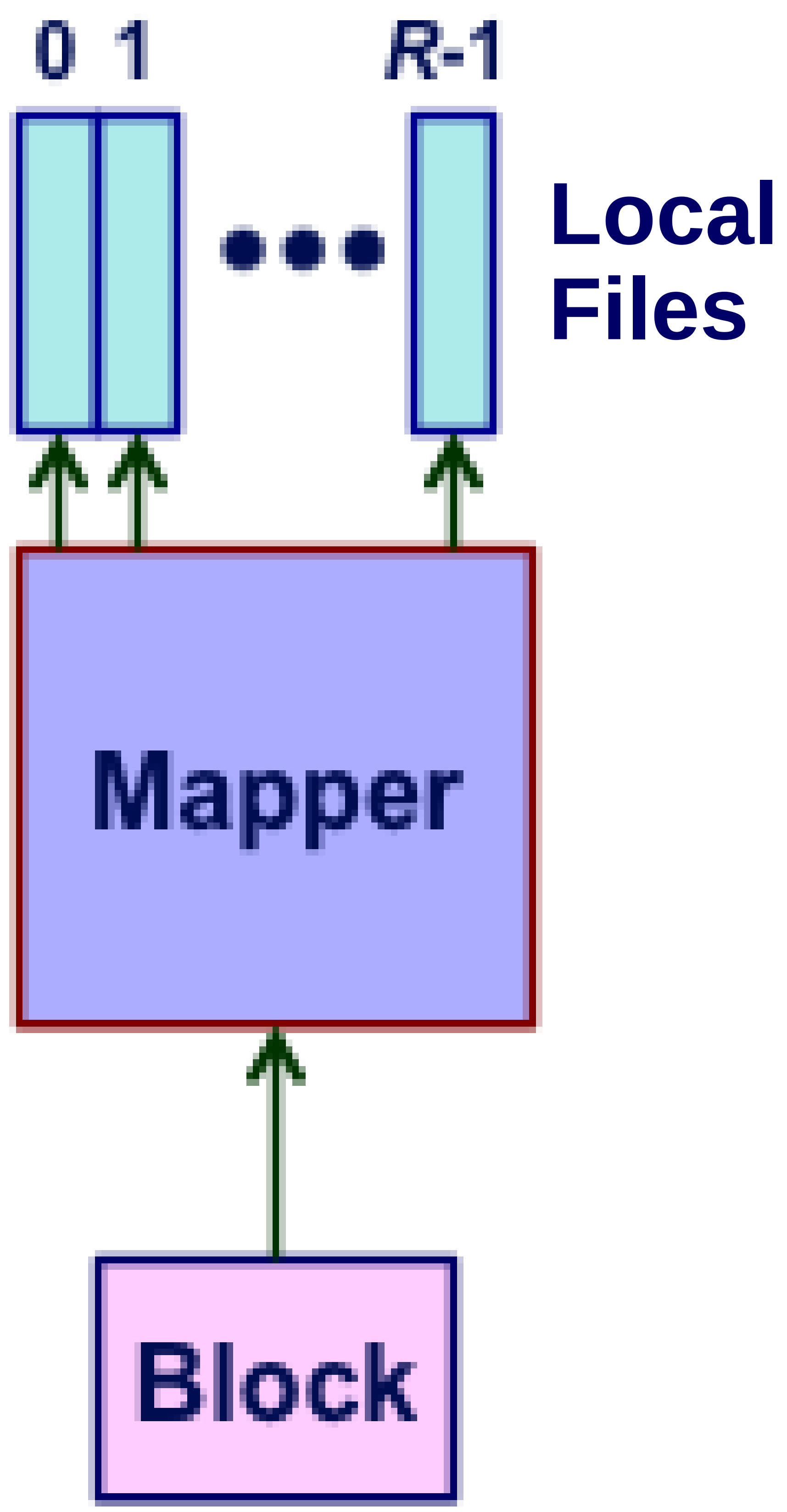
$$h(K) \in \{0, \ldots, R-1\}$$

Hash Function *h*

- Maps each key *K* to integer *i* such that $0 \le i < R$

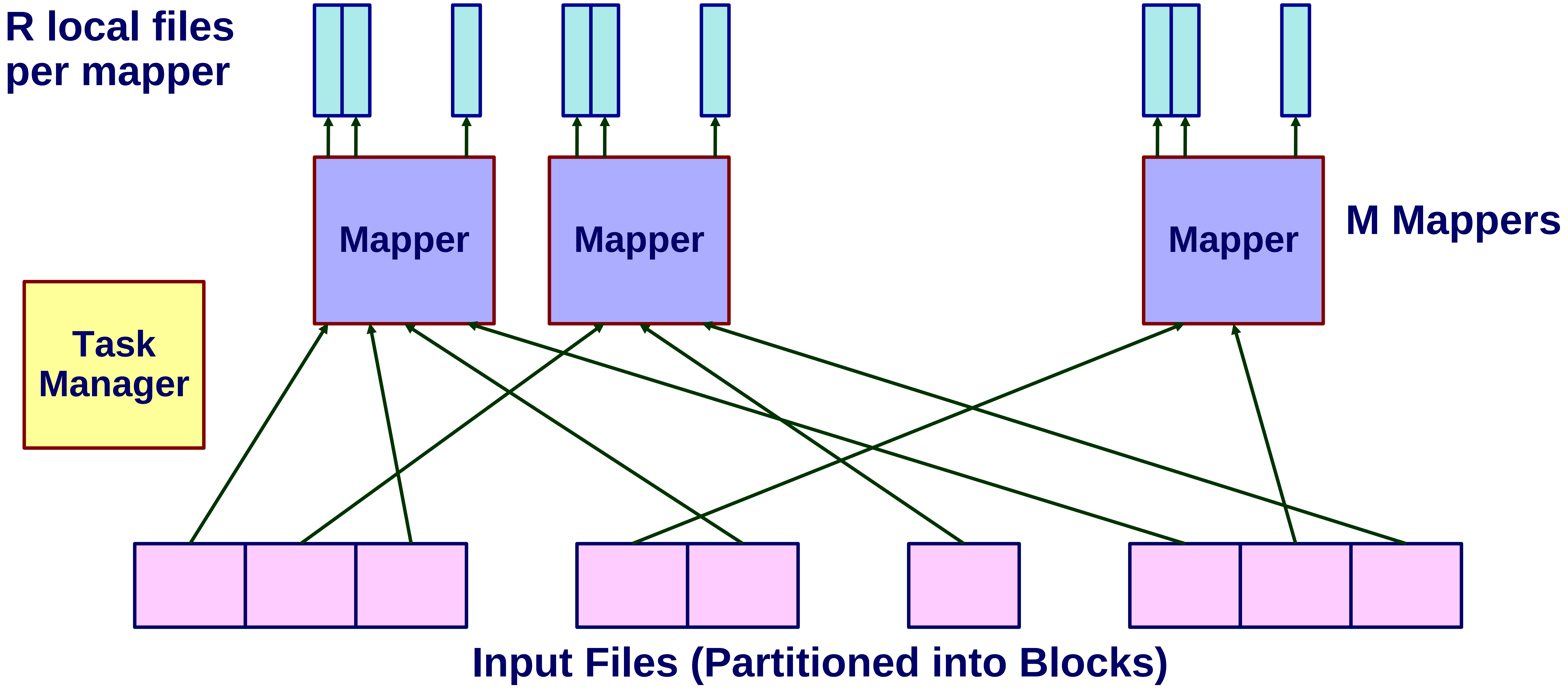
Mapper Operation

- Reads input file blocks
- Generates pairs $\langle K, V \rangle$
- Writes to local file $h(K)$



Distributed Mapper

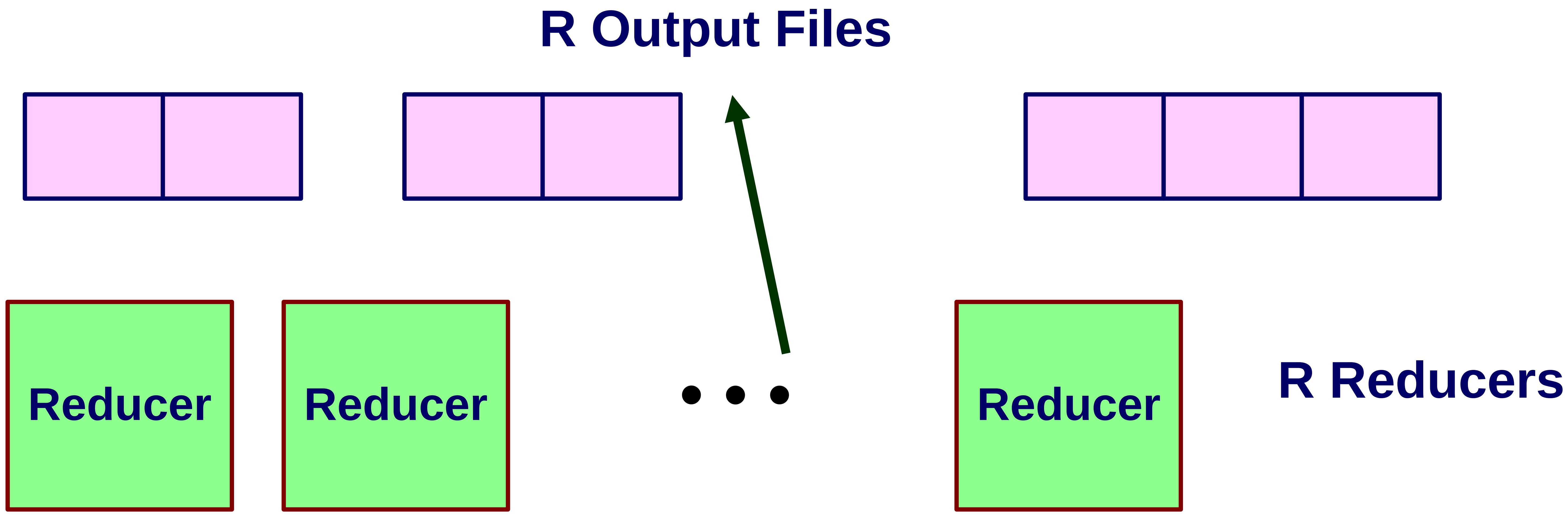
- Dynamically map input file blocks onto mappers
- Each generates key/value pairs from its blocks
- Each writes R files on local file system



Reducer

Each Reducer:

- Executes reducer function for each key
- Writes output values to parallel file system

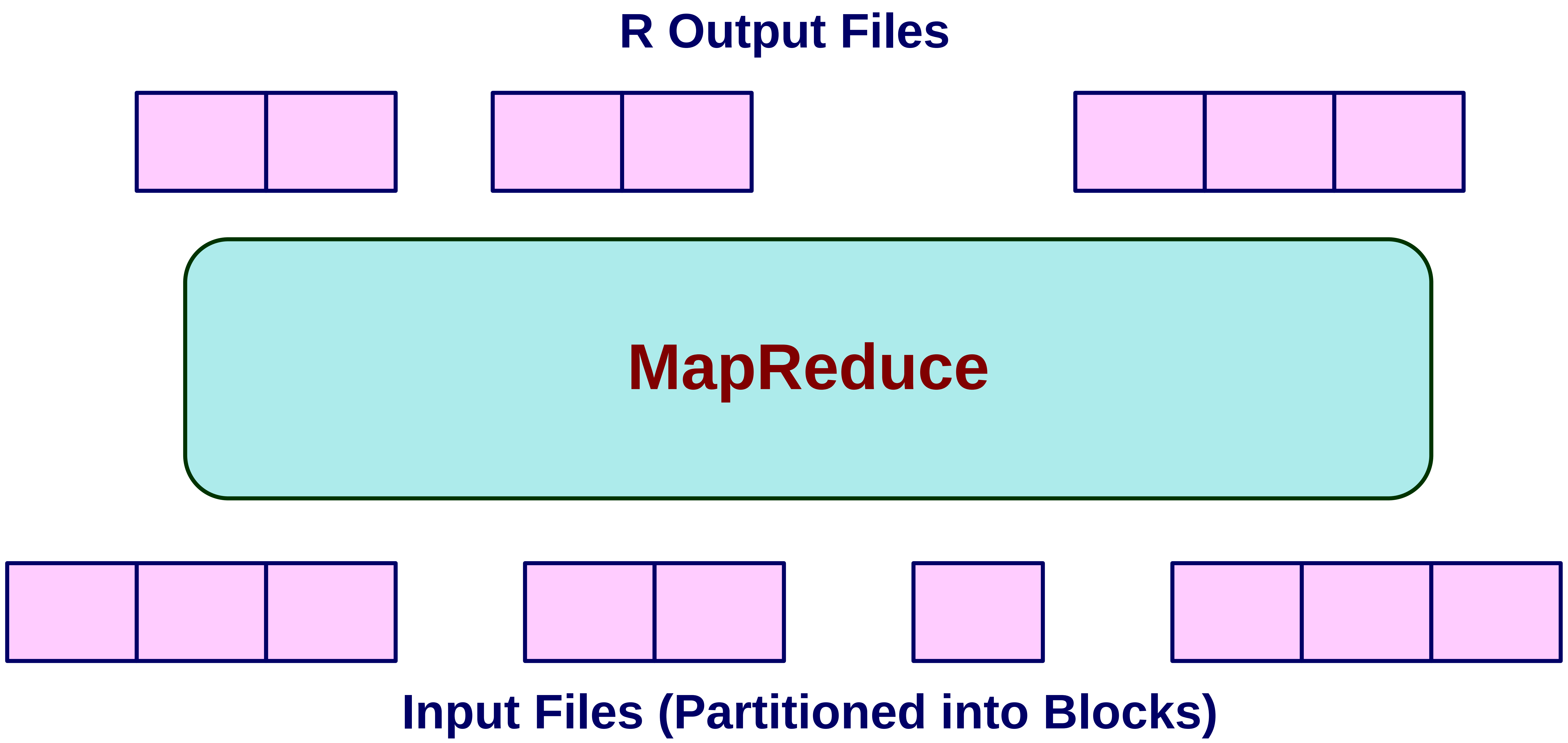


MapReduce Effect

MapReduce Step

- Reads set of files from file system
- Generates new set of files

Can iterate to do more complex processing



Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models (API)
  - Job execution (runtime)
  - MapReduce dataflow
- Beyond MapReduce

Example: Sparse Matrices with Map/Reduce

$$\overset{\mathbf{A}}{\begin{bmatrix} 10 & & 20 \\ & 30 & 40 \\ 50 & 60 & 70 \end{bmatrix}} \times \overset{\mathbf{B}}{\begin{bmatrix} -1 & \\ -2 & -3 \\ & -4 \end{bmatrix}} = \overset{\mathbf{C}}{\begin{bmatrix} -10 & -80 \\ -60 & -250 \\ -170 & -460 \end{bmatrix}}$$

- Task: Compute product C = A ·B
- Assume most matrix entries are 0

Motivation

- Core problem in scientific computing
- Challenging for parallel execution

Computing Sparse Matrix Product

A

$$\begin{bmatrix} 10 & & 20 \\ & 30 & 40 \\ 50 & 60 & 70 \end{bmatrix}$$

$1 \xrightarrow[A]{10} 1$

$1 \xrightarrow[A]{20} 3$

$2 \xrightarrow[A]{30} 2$

$2 \xrightarrow[A]{40} 3$

$3 \xrightarrow[A]{50} 1$

$3 \xrightarrow[A]{60} 2$

$3 \xrightarrow[A]{70} 3$

B

$$\begin{bmatrix} -1 & \\ -2 & -3 \\ & -4 \end{bmatrix}$$

$1 \xrightarrow[B]{-1} 1$

$2 \xrightarrow[B]{-2} 1$

$2 \xrightarrow[B]{-3} 2$

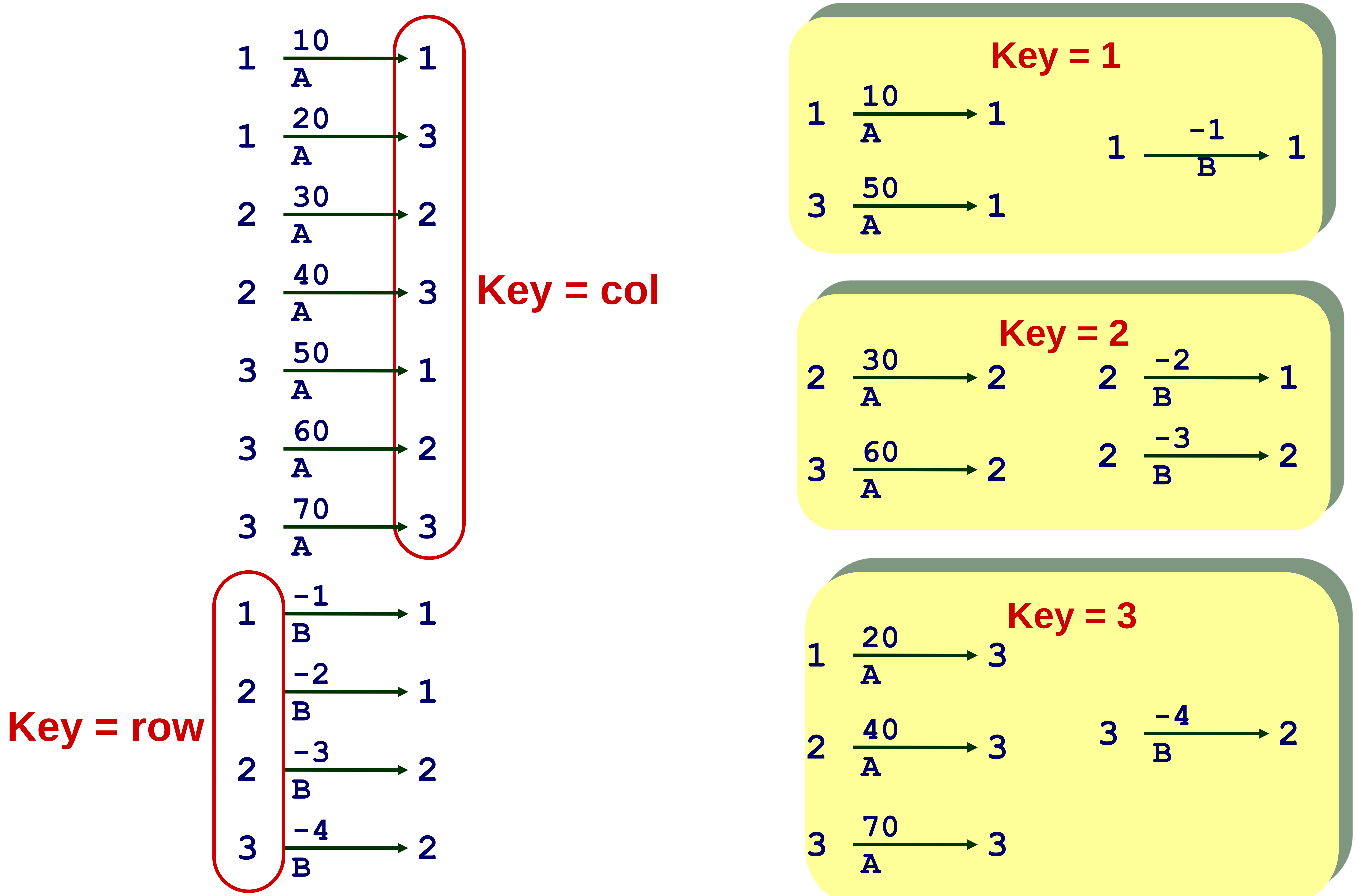
$3 \xrightarrow[B]{-4} 2$

- Represent matrix as list of nonzero entries

  ⟨row, col, value, matrixID⟩

- How to represent the computation as map-reduce?
  - Phase 1: Compute all products $a_{i,k} \cdot b_{k,j}$
  - Phase 2: Sum products for each entry i,j
  - Each phase involves a Map/Reduce

Phase 1 Map of Matrix Multiply



- Group values $a_{i,k}$ and $b_{k,j}$ according to key k

Phase 1 "Reduce" of Matrix Multiply

Key = 1

$1 \xrightarrow[A]{10} 1$

$3 \xrightarrow[A]{50} 1$

X

$1 \xrightarrow[B]{-1} 1$

Key = 2

$2 \xrightarrow[A]{30} 2$

$3 \xrightarrow[A]{60} 2$

X

$2 \xrightarrow[B]{-2} 1$

$2 \xrightarrow[B]{-3} 2$

Key = 3

$1 \xrightarrow[A]{20} 3$

$2 \xrightarrow[A]{40} 3$

$3 \xrightarrow[A]{70} 3$

X

$3 \xrightarrow[B]{-4} 2$

$1 \xrightarrow[C]{-10} 1$

$3 \xrightarrow[A]{-50} 1$

$2 \xrightarrow[C]{-60} 1$

$2 \xrightarrow[C]{-90} 2$

$3 \xrightarrow[C]{-120} 1$

$3 \xrightarrow[C]{-180} 2$

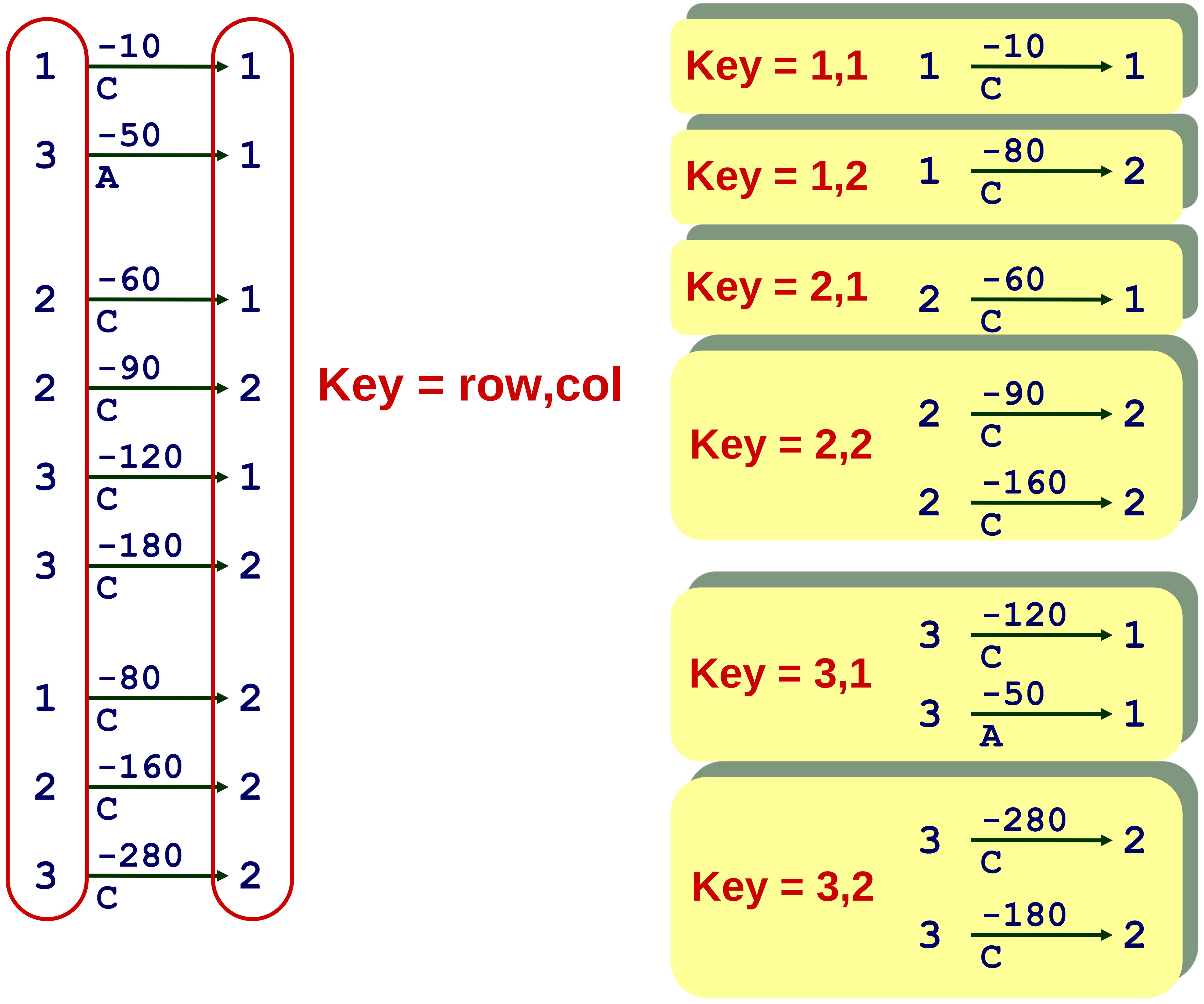
$1 \xrightarrow[C]{-80} 2$

$2 \xrightarrow[C]{-160} 2$

$3 \xrightarrow[C]{-280} 2$

- Generate all products $a_{i,k} \cdot b_{k,j}$

Phase 2 Map of Matrix Multiply



- Group products $a_{i,k} \cdot b_{k,j}$ with matching values of i and j

Phase 2 Reduce of Matrix Multiply

Key = 1,1	$1 \xrightarrow[C]{-10} 1$	$1 \xrightarrow[C]{-10} 1$
Key = 1,2	$1 \xrightarrow[C]{-80} 2$	$1 \xrightarrow[C]{-80} 2$
Key = 2,1	$2 \xrightarrow[C]{-60} 1$	$2 \xrightarrow[C]{-60} 1$
Key = 2,2	$2 \xrightarrow[C]{-90} 2$ $2 \xrightarrow[C]{-160} 2$	$2 \xrightarrow[C]{-250} 2$
Key = 3,1	$3 \xrightarrow[C]{-120} 1$ $3 \xrightarrow[A]{-50} 1$	$3 \xrightarrow[C]{-170} 1$
Key = 3,2	$3 \xrightarrow[C]{-280} 2$ $3 \xrightarrow[C]{-180} 2$	$3 \xrightarrow[C]{-460} 2$

C

$$\begin{bmatrix} -10 & -80 \\ -60 & -250 \\ -170 & -460 \end{bmatrix}$$

- Sum products to get final entries

Recap: MapReduce Implementation

Built on Top of Parallel File System

- Google: GFS, Hadoop: HDFS
- Provides global naming
- Reliability via replication (typically 3 copies)

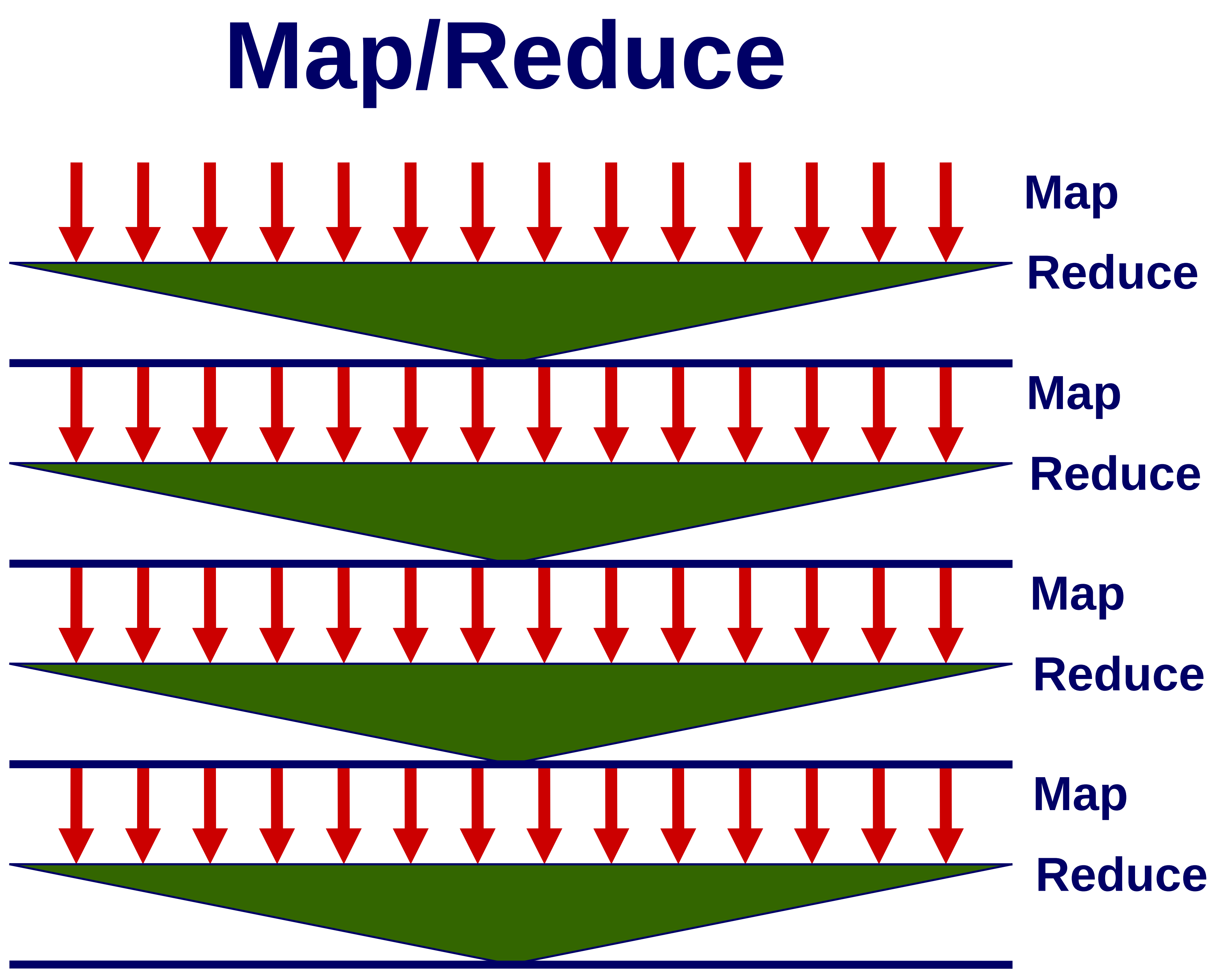
Breaks work into tasks

- Master schedules tasks on workers dynamically
- Typically #tasks >> #processors

Net Effect

- Input: Set of files in reliable file system
- Output: Set of files in reliable file system

Analyzing Pros and Cons of Map/Reduce



Characteristics

- Computation broken into many, short-lived tasks
  - Mapping, reducing
- Use disk storage to hold intermediate results

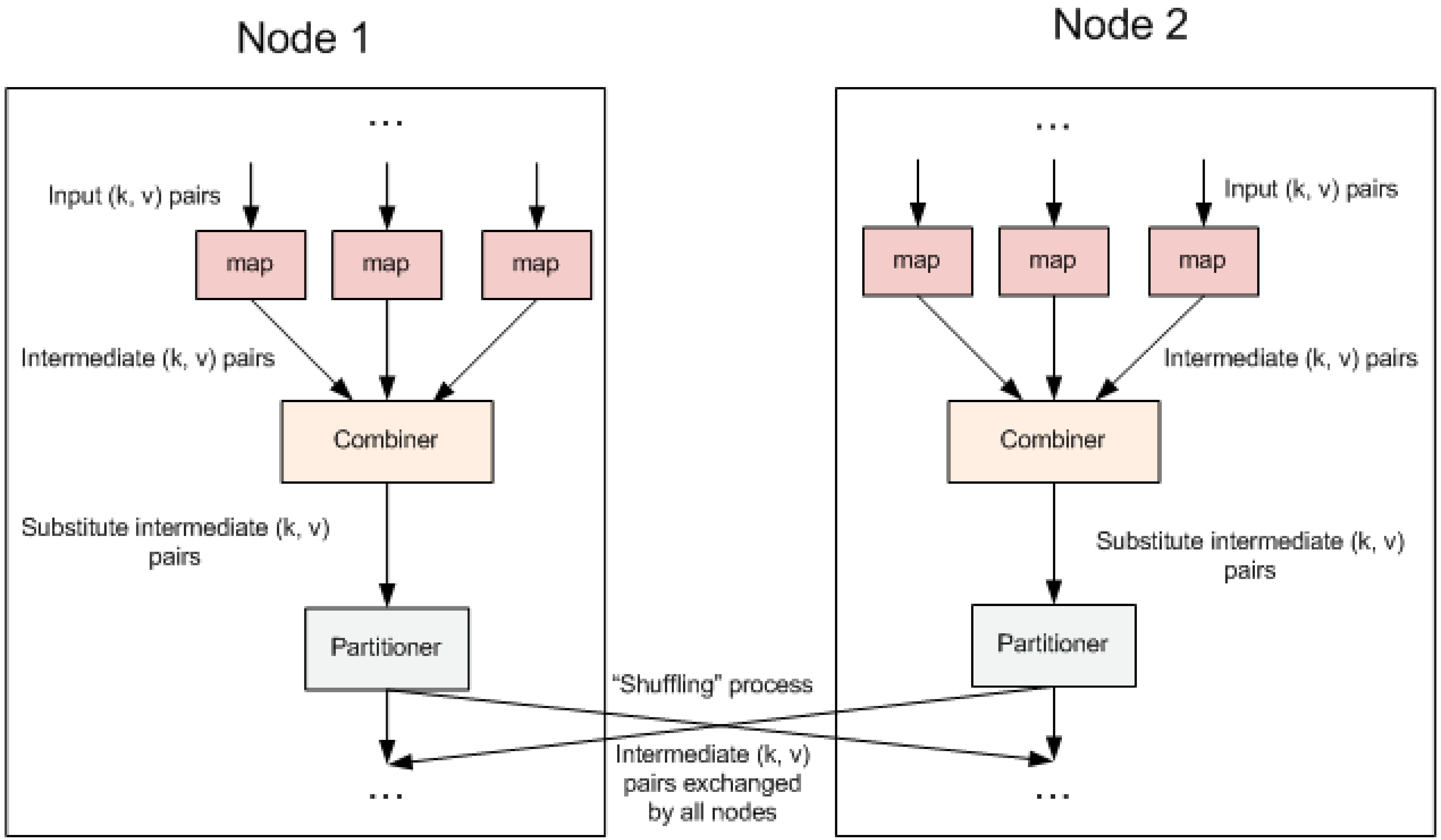
Strengths

- Great flexibility in placement, scheduling, and load balancing
- Can access large data sets

Weaknesses

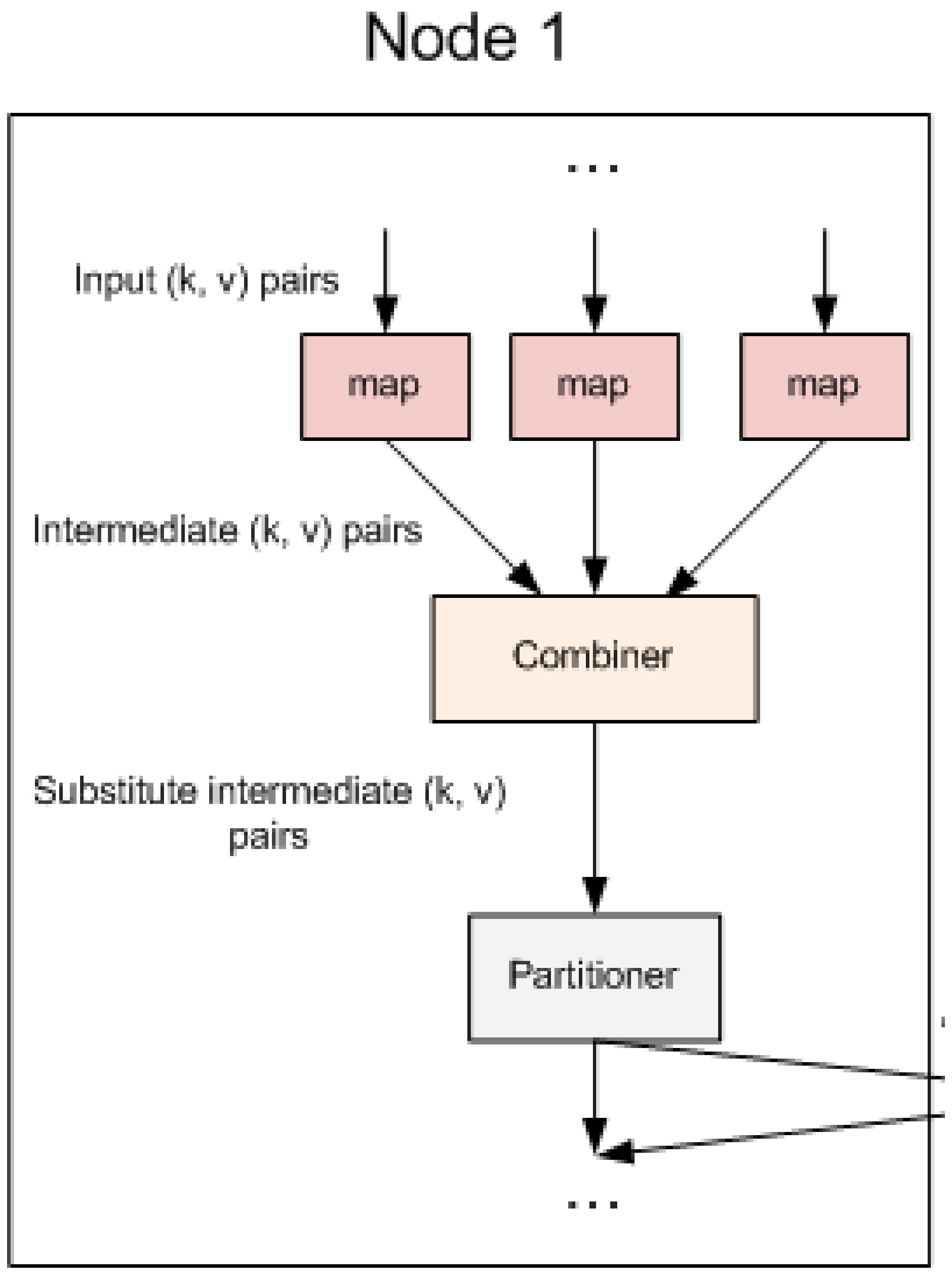
- Higher overhead due to disk read/write
- Lower raw performance (each map / reduce task takes long to invoke)
- Learning Functional programming is non-trivial!

Beyond Map/Reduce: Combiner and Partitioner



Combiners & Partitioners are optional.

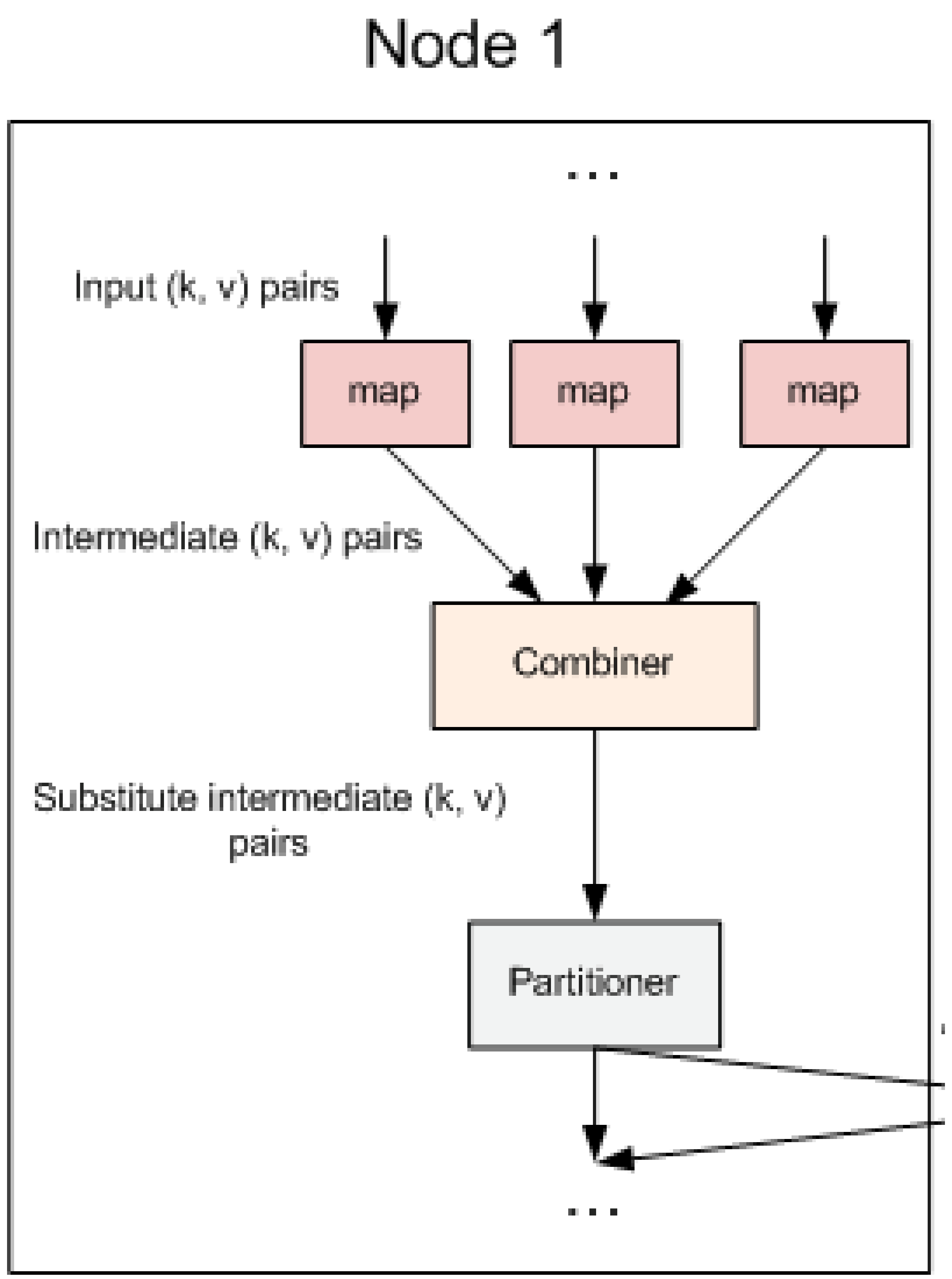
Input → Map → Combiner → Partitioner → Reducer → Output



Input:
What do you mean by Object
What do you know about Java
What is Java Virtual Machine
How Java enabled High Performance

Record reader:
<1, What do you mean by Object>
<2, What do you know about Java>
<3, What is Java Virtual Machine>
<4, How Java enabled High Performance>

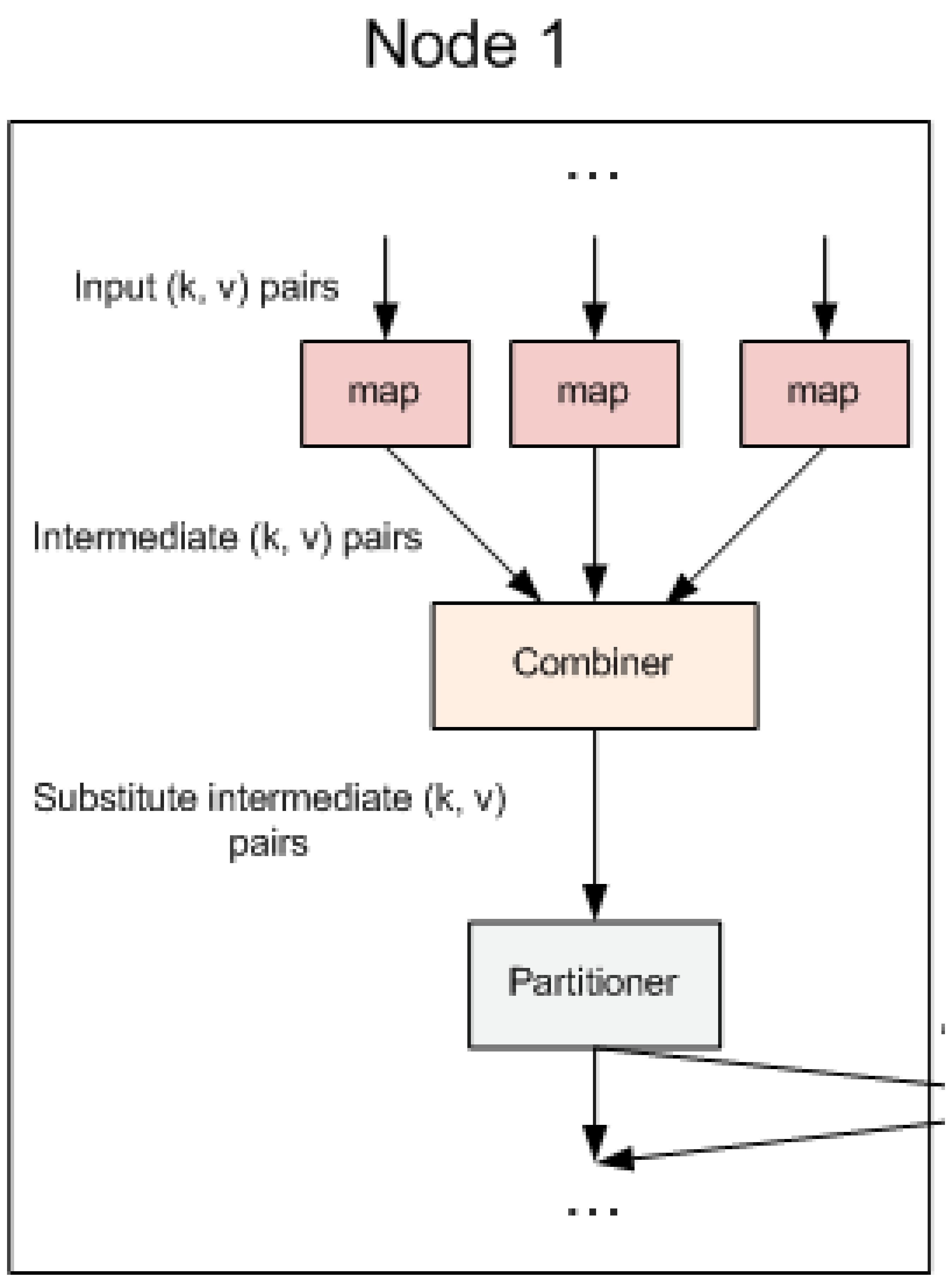
Combiner (mini-reducer):
optional, to summarize the map output records with the same key



Map output:
<What,1> <do,1> <you,1> <mean,1> <by,1>
<Object,1> <What,1> <do,1> <you,1> <know,1>
<about,1> <Java,1> <What,1> <is,1> <Java,1>
<Virtual,1> <Machine,1> <How,1> <Java,1>
<enabled,1> <High,1> <Performance,1>

Combiner output:
<What,1,1,1> <do,1,1> <you,1,1> <mean,1> <by,1>
<Object,1> <know,1> <about,1> <Java,1,1,1> <is,1>
<Virtual,1> <Machine,1> <How,1> <enabled,1>
<High,1> <Performance,1>

Partitioner:
optional, a condition in processing an input dataset



Combiner output:
<What,1,1,1> <do,1,1> <you,1,1> <mean,1> <by,1> <Object,1> <know,1> <about,1> <Java,1,1,1> <is,1> <Virtual,1> <Machine,1> <How,1> <enabled,1> <High,1> <Performance,1>

The number of partitioners is equal to the number of reducers.

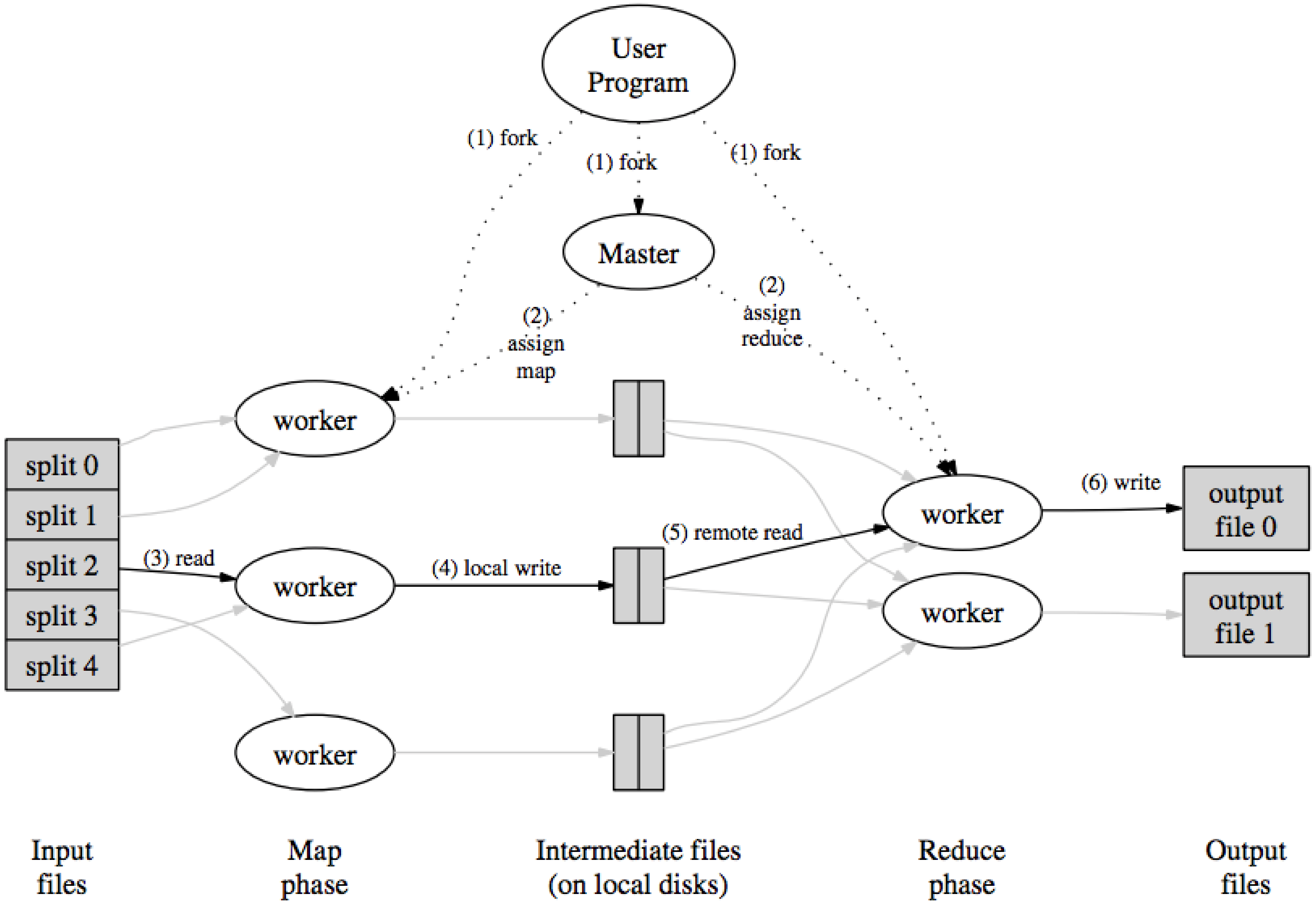
Partitioner output:

<What,1,1,1>: long sentence,
<do,1,1>: long sentence,
……

Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models (API)
  - Job execution (runtime)
  - Workflow
  - MapReduce Recap
- Beyond MapReduce

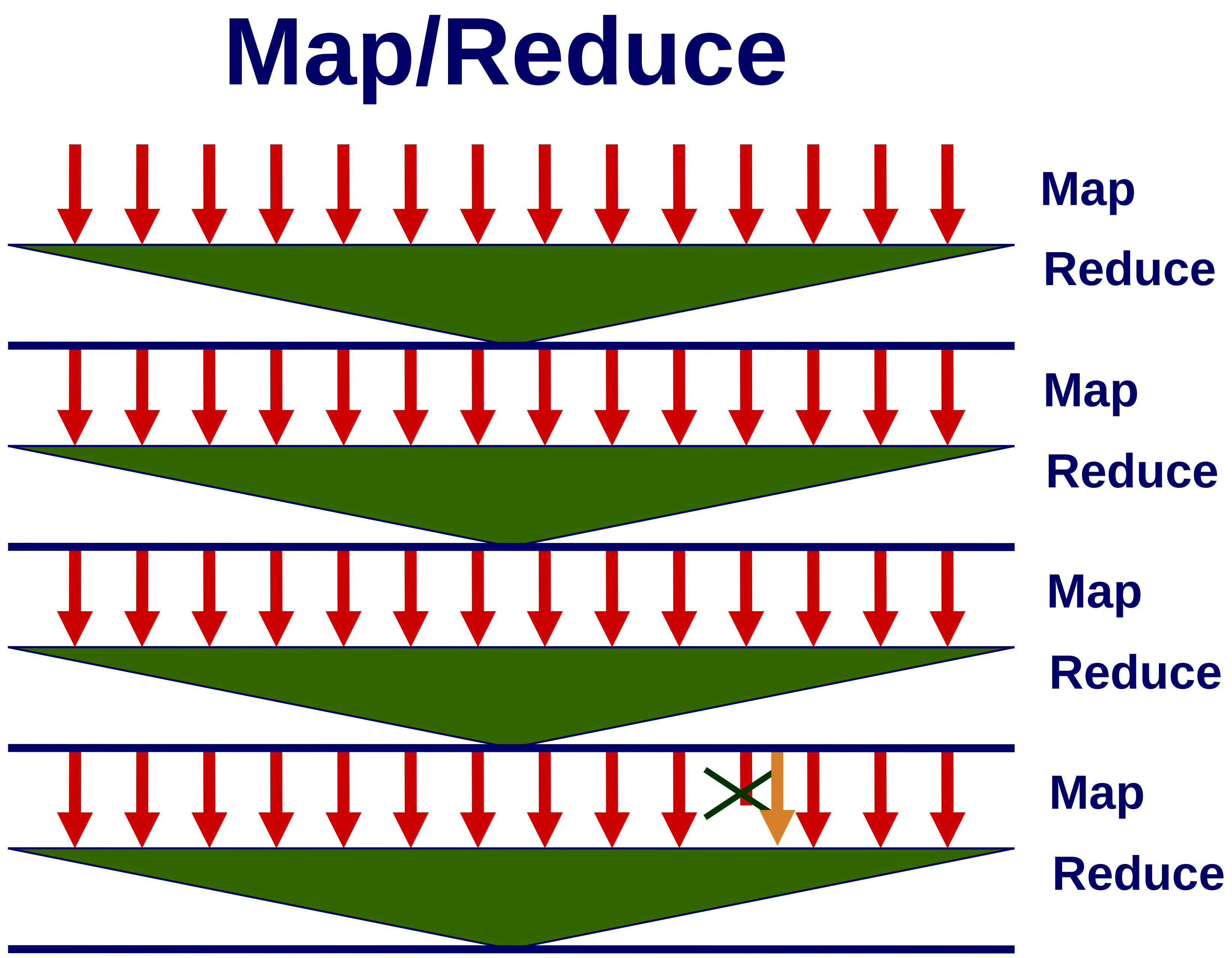
MapReduce System architecture (Paper)



Fault Tolerance and Straggler Mitigation

- Fault Tolerance
  - Assume reliable file system
  - Detect failed worker
    - Heartbeat mechanism
  - Reschedule failed task
- Dealing with Stragglers
  - Tasks that take long time to execute
  - Might be bug, flaky hardware, or poor partitioning
  - When done with most tasks, reschedule any remaining executing tasks
    - Keep track of redundant executions
    - Significantly reduces overall run time

Fault Tolerance



Data Integrity

- Store multiple copies of each file
- Including intermediate results of each Map / Reduc
  - Continuous checkpointing

Recovering from Failure

- Simply recompute lost result
  - Localized effect
- Dynamic scheduler keeps all processors busy

Map/Reduce Summary

Typical Map/Reduce Applications

- Sequence of steps, each requiring map & reduce
- Series of data transformations

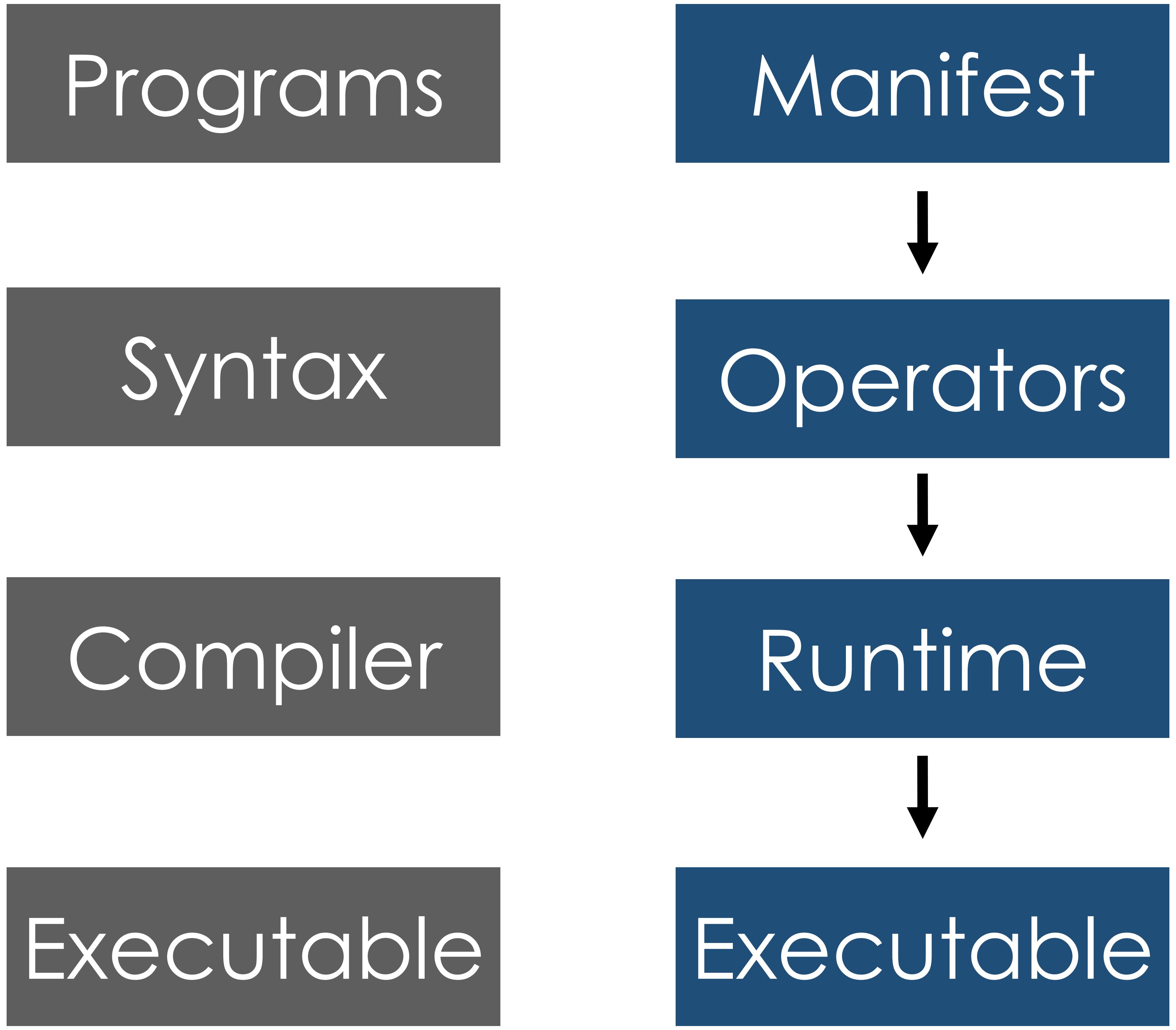
Strengths of Map/Reduce

- User writes simple functions, system manages complexities of mapping, synchronization, fault tolerance
- Very general
- Good for large-scale data analysis

Map Reduce Summary: Cons

- Disk I/O overhead is super high
- Not flexible enough: Each map/reduce step must complete before next begins
- Not suitable for workloads:
  - Iterative processing
  - Real-time processing
- Map-reduce is still difficult to program with

All Modern Data/ML Systems follow the following arch



A fixed set of operators

A trusted runtime with a small set of pre-loaded implementations

PageRank Computation

Initially

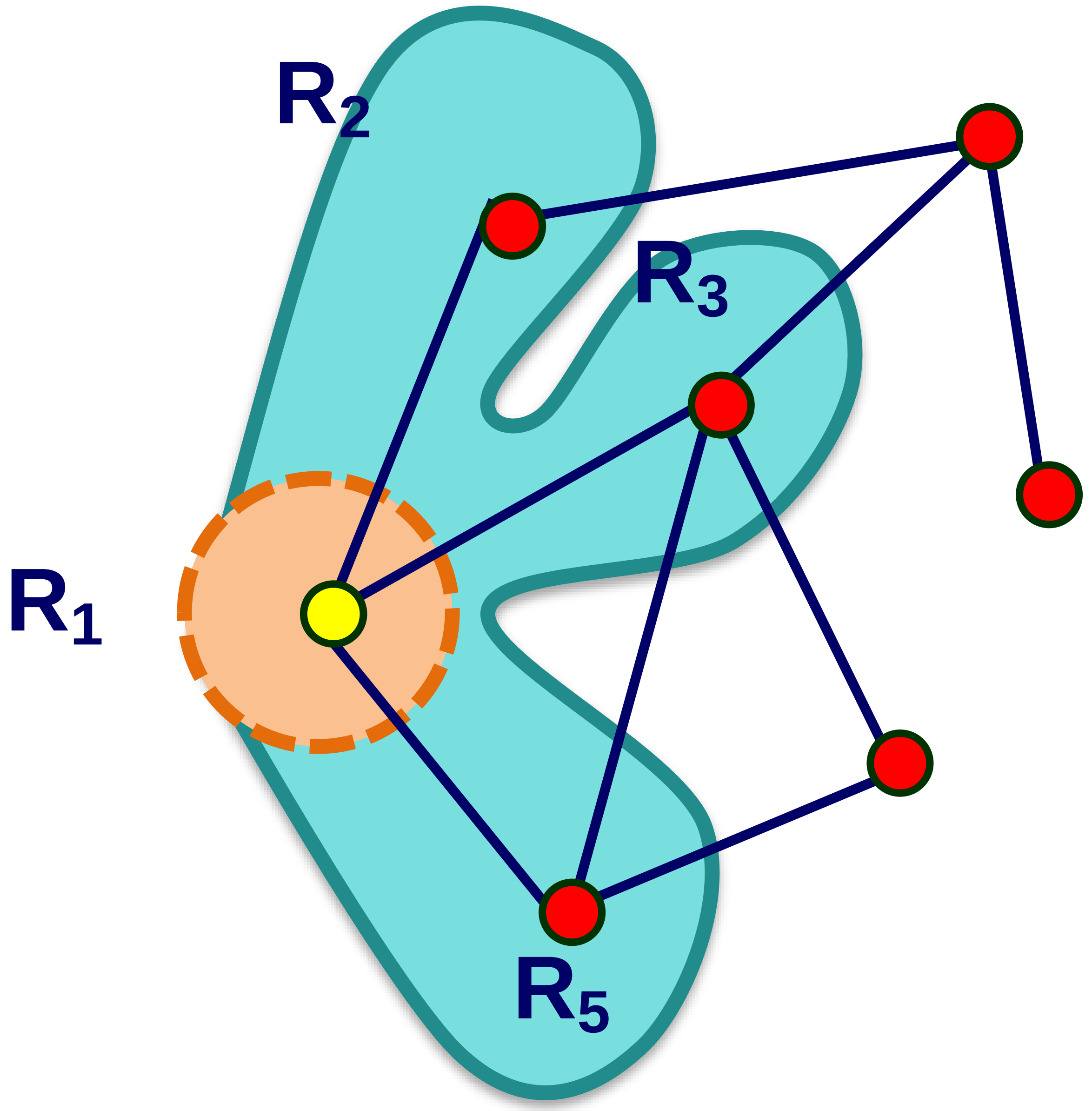
- Assign weight 1.0 to each page

Iteratively

- Select arbitrary node and update its value

Convergence

- Results unique, regardless of selection ordering



$$R_1 \leftarrow 0.1 + 0.9 * (\tfrac{1}{2} R_2 + \tfrac{1}{4} R_3 + \tfrac{1}{3} R_5)$$

Q: how to express pagerank using map-reduce?