https://hao-ai-lab.github.io/dsc204a-f25/

DSC 204A: Scalable Data Systems
Fall 2025

Staff
Instructor: Hao Zhang
TAs: Mingjia Huo, Yuxuan Zhang

@haozhangml @haoailab
haozhang@ucsd.edu

Where We Are

Motivations, Economics, Ecosystems, Trends

Networking

Storage

Part3: Compute

~~Datacenter networking~~

~~Collective communication~~

(Distributed) File Systems / Database

Cloud storage

Distributed Computing

Big data processing

Let's Focus On: Multi-node Distributed Systems

We have two primary problems to solve in real systems

1. How to Distribute Data (we'll not cover, also not in exam)
   - Read DDIA
   - Read GFS paper
   - Read BigTable paper
2. How to Distribute Compute (we'll cover in lectures)
   - Batching Processing
   - Streaming Processing

How to Analyze Distributed Systems

- Scalability
  - Data volume
  - Read/Write/Compute load

- Consistency and correctness
  - Read / Write sees consistency data
  - Compute produce correct results

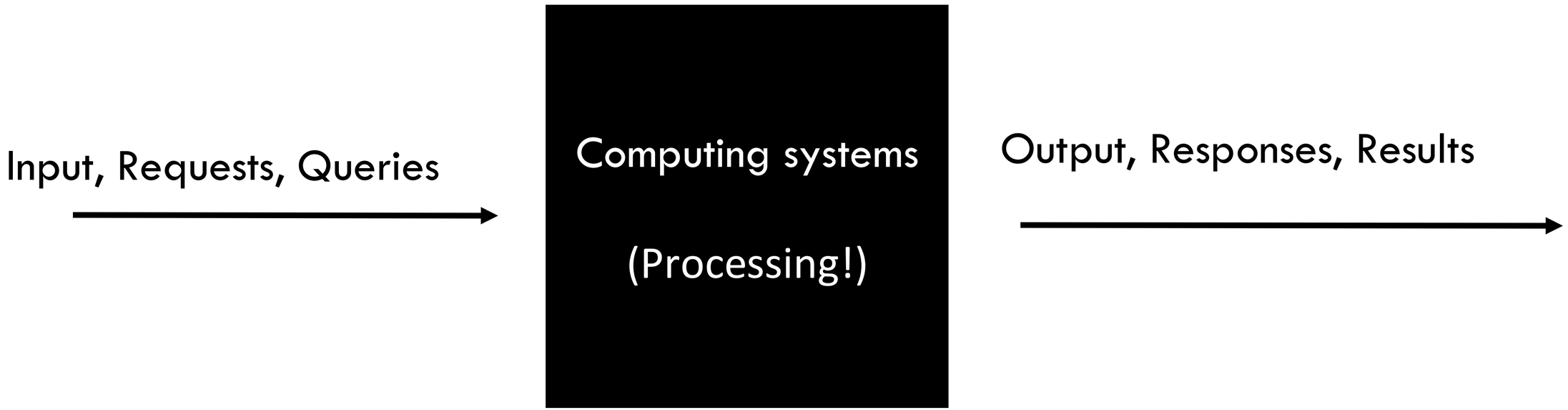
- Fault tolerance / high availability
  - When one fails, another can take over.

- Latency and throughput
  - Distribute machines worldwide.
  - Reduce network latency.

Today's topic: Batch Processing

- Overview
- IO & Unix pipes
- MapReduce
- Beyond MapReduce

Basic Computing System Paradigm



Processing latency

feedback cycle time

direct manipulation
no visible lag → Interactive data science!

→ Online system: handle request ASAP

turn-taking
minutes to seconds → Stream processing systems (near-real-time systems)

batch-processing
hours or overnight → Batch processing systems (Offline systems)

Figure from https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/deliver/index/docId/10090/file/mueller_diss.pdf

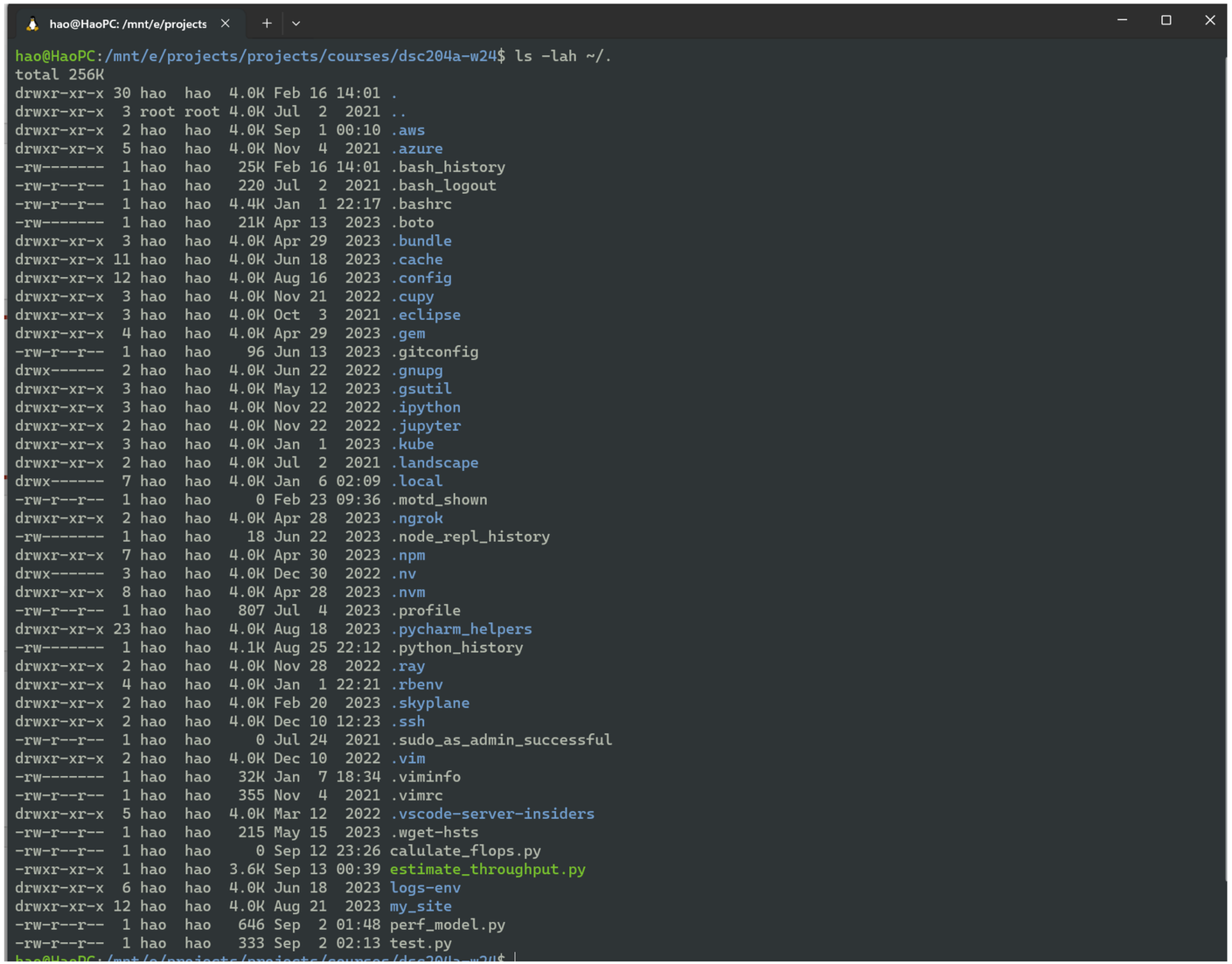
Today's topic: Batch Processing

- Overview
- IO & Unix pipes: the first batching processing system
- MapReduce
- Beyond MapReduce

Shell example

“.****rc”

Run commands



Useful shell commands

- Shell already has a collection of rich commands
  - Some Useful commands
    - uptime, cut, date, cat, finger, hexdump, man, md5sum, quota,
    - mkdir, rmdir, rm, mv, du, df, find, cp, chmod, cd
    - uname, zip, unzip, gzip, tar
    - tr, sed, sort, uniq, ascii
    - Type “man command” to read about shell commands

What do these shell commands do?

- cat dups.txt | sort | uniq
- cat dups.txt | sort -V | uniq
- cat dups.txt | sort -V | uniq > outfile.txt
- tr "a" "e" < z.txt
- cat z.txt | tr a e

Batch processing with Unix Tools

```
cat /var/log/nginx/access.log | ❶
  awk '{print $7}' | ❷
  sort             | ❸
  uniq -c          | ❹
  sort -r -n       | ❺
  head -n 5          ❻
```

```
4189 /favicon.ico
3631 /2013/05/24/improving-security-of-ssh-private-keys.html
2124 /2012/12/05/schema-evolution-in-avro-protocol-buffers-thrift.html
1369 /
 915 /css/typography.css
```

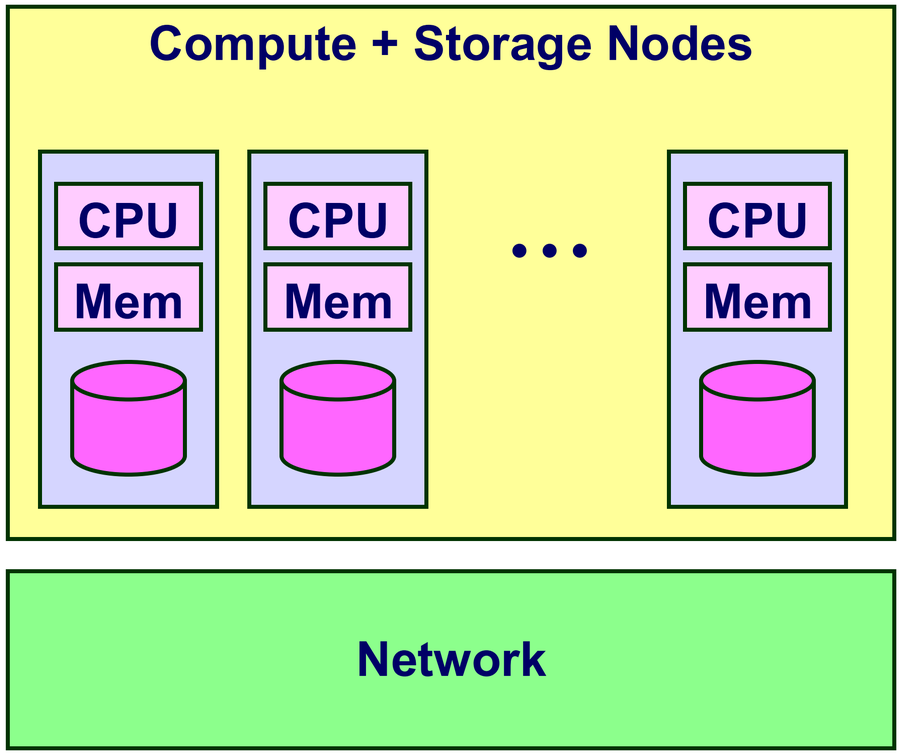
- Read the log file.
- Split each line into fields by white space, output only the 7th element (requested URL).
- Alphabetically sort
- Filter out repeated lines.
- Sort it again based on the line number (-n)
- Out put the first five lines.

The biggest limitation of Unix tools is that they run only on a single machine — and that's where tools like Hadoop come in.

Today's topic: Batch Processing

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

History of MapReduce/Hadoop



Compute + Storage Nodes

- Medium-performance processors
- Modest memory
- 1-2 disks

Network

- Conventional Ethernet switches
  - 10 Gb/s within rack
  - 100 Gb/s across racks

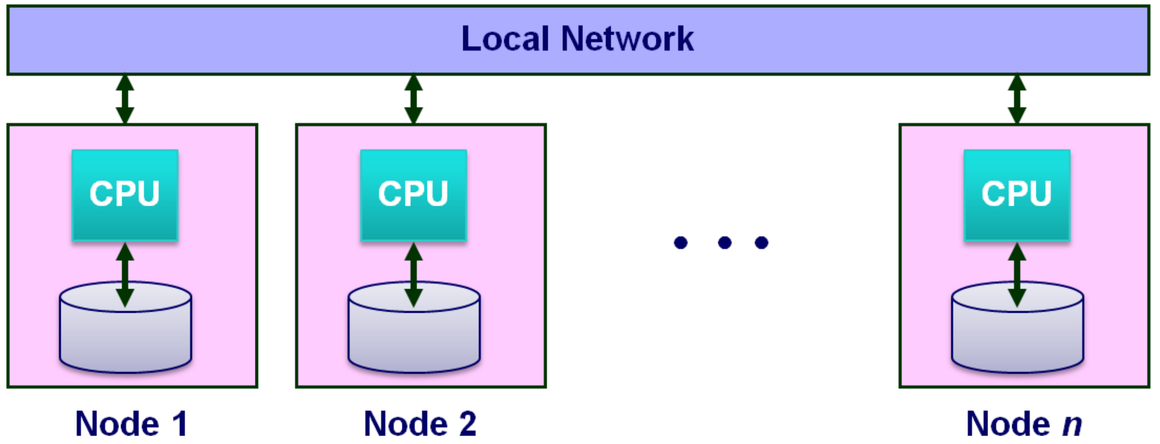
Data-Intensive System Challenge

For Computation That Accesses 1 TB in 5 minutes

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

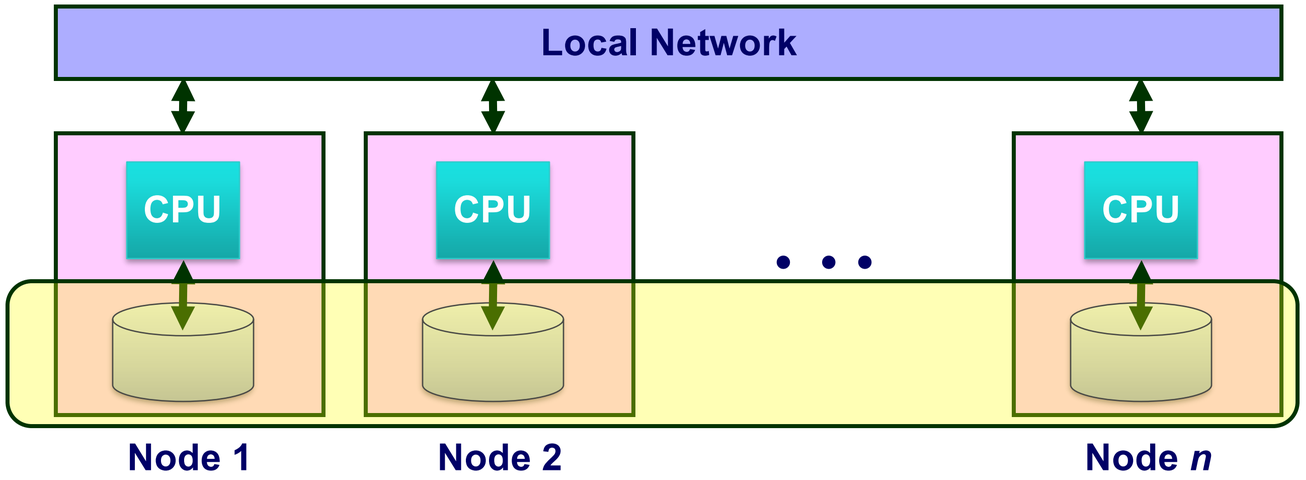
System Requirements

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



Hadoop Project

File system with files distributed across nodes



- Store multiple (typically 3 copies of each file)
  - If one node fails, data still available
- Logically, any node has access to any file
  - May need to fetch across network (ideally, leverage locality for perf.)

Map / Reduce programming environment

- Software manages execution of tasks on nodes

Today's topic: Batch Processing

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

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.

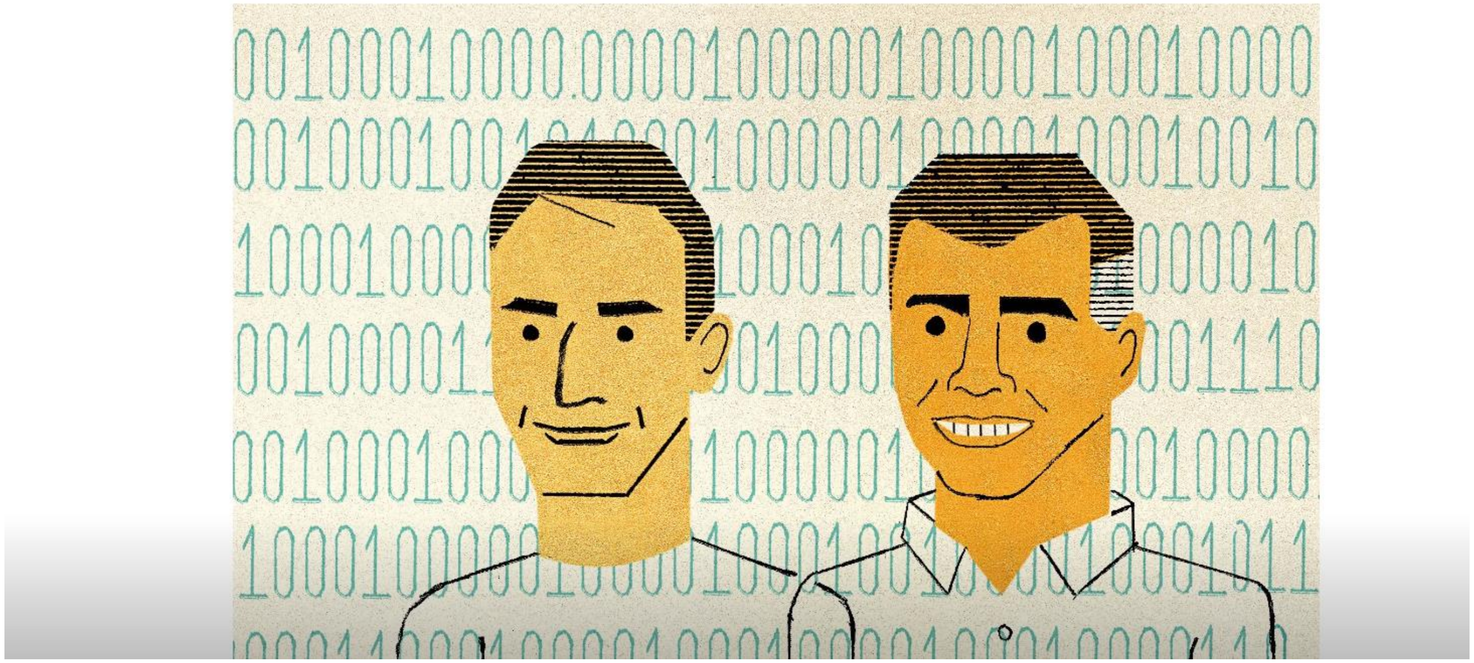
ANNALS OF TECHNOLOGY

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

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

n ← # of documents

df(d, t) ← Document frequency of the term t

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

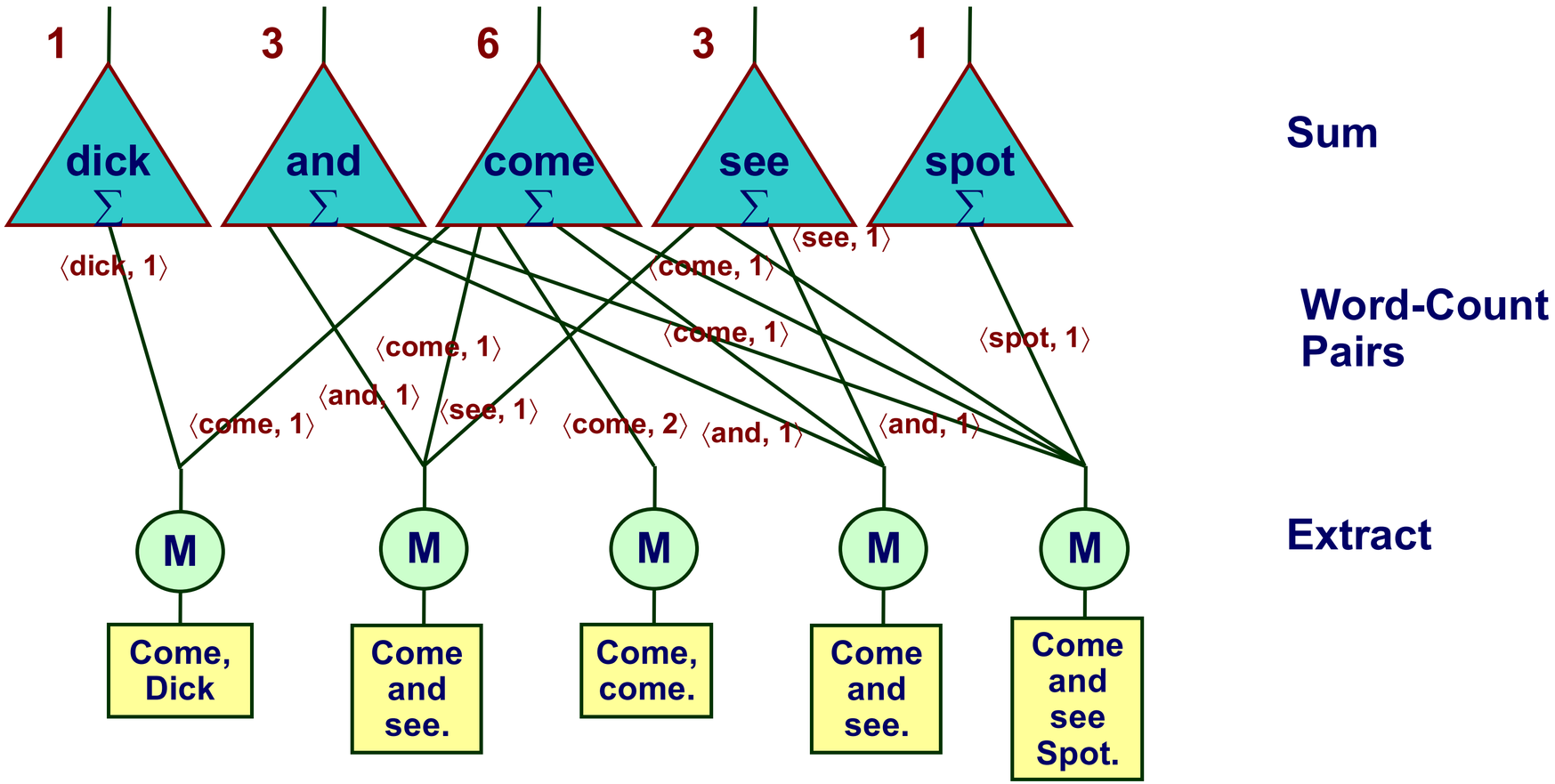
Come, Dick

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

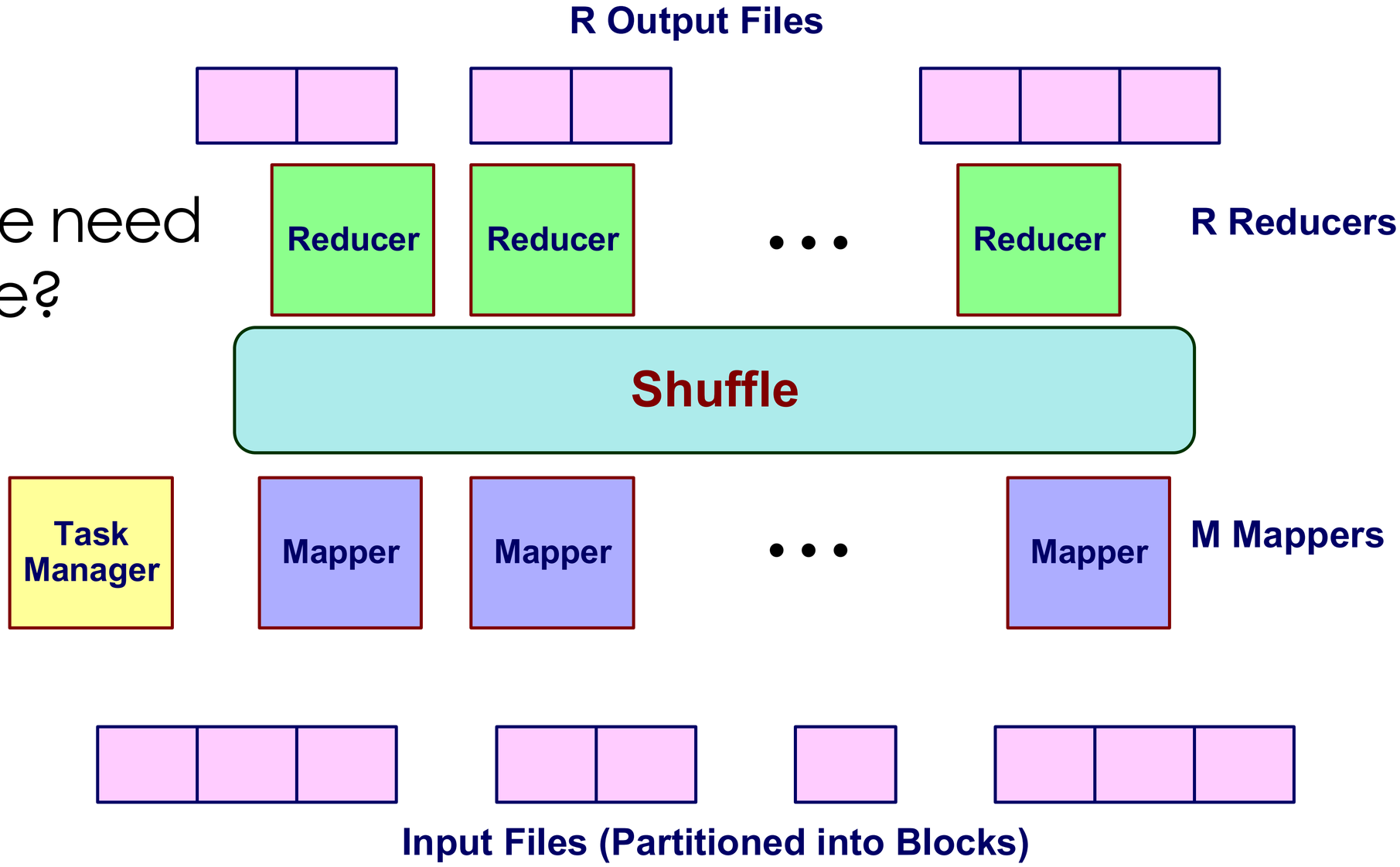
Discussion:
Other possible way to implement this using map-reduce?

Today's topic: Batch Processing

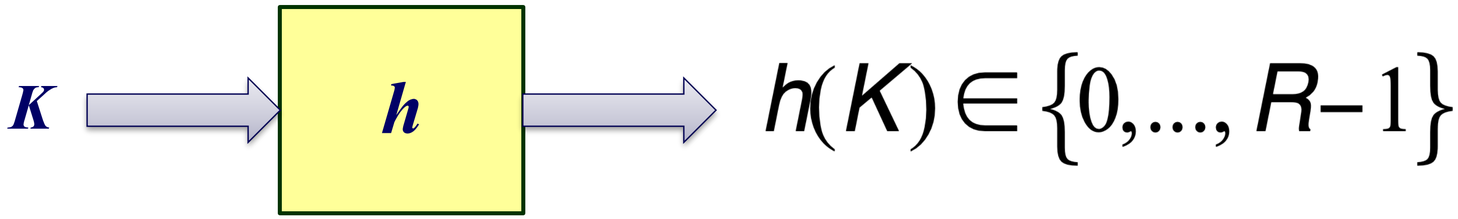
- Overview
- IO & Unix pipes
- MapReduce
  - HDFS - infrastructure
  - Programming models
  - Job execution
  - Workflow
- Beyond MapReduce

MapReduce Execution (Runtime)

Why do we need shuffle?



Single Mapper

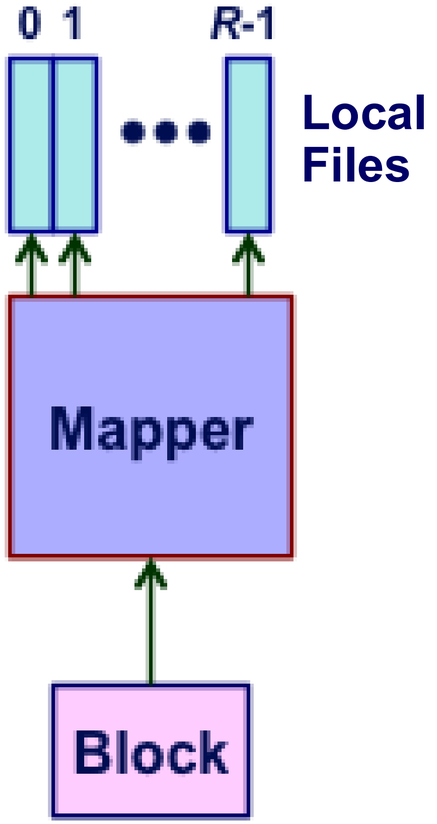


Hash Function *h*

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

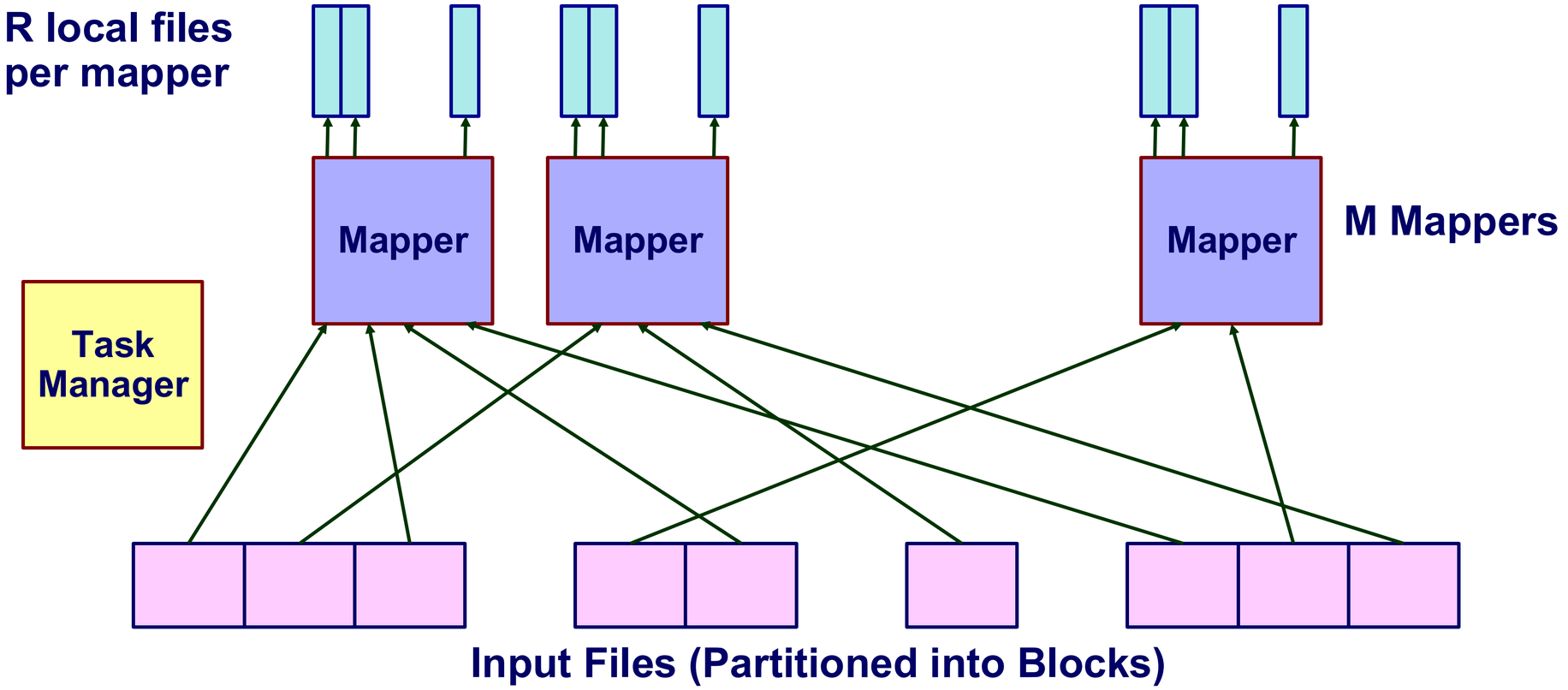
Mapper Operation

- Reads input file blocks
- Generates pairs ⟨*K*, *V*⟩
- Writes to local file *h*(*K*)



Distributed Mapper

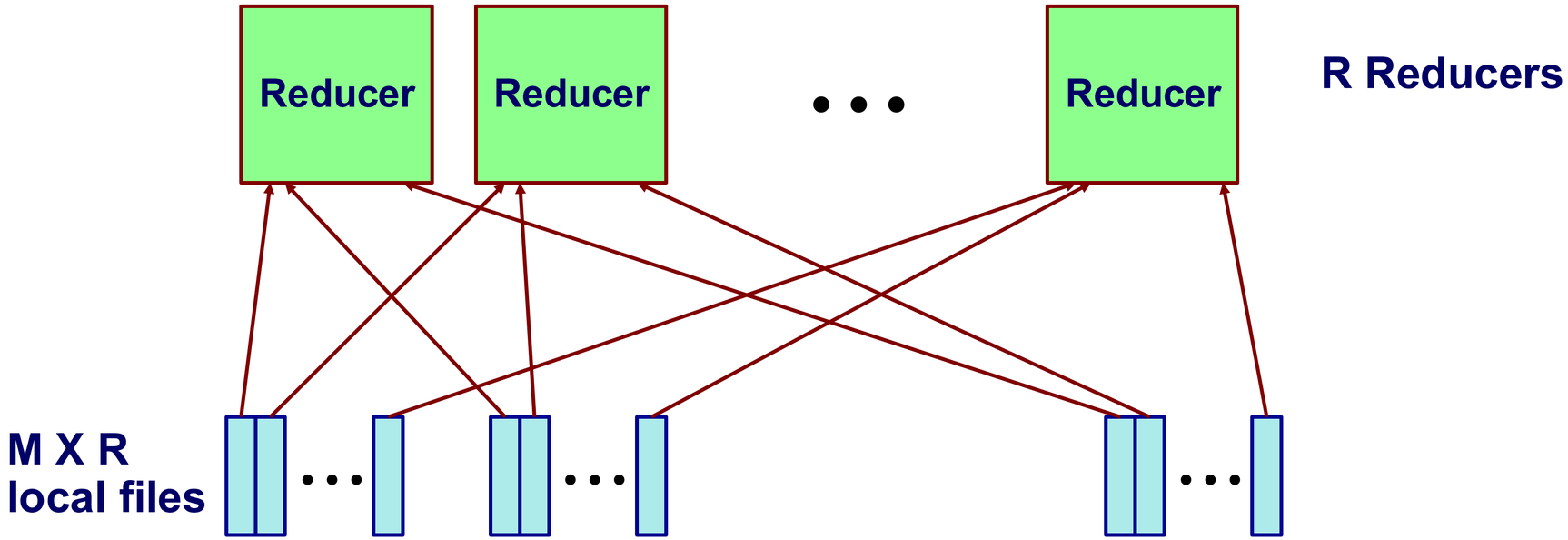
- Dynamically map input file blocks onto mappers
- Each generates key/value pairs from its blocks
- Each writes R files on local file system



Shuffling

Each Reducer:

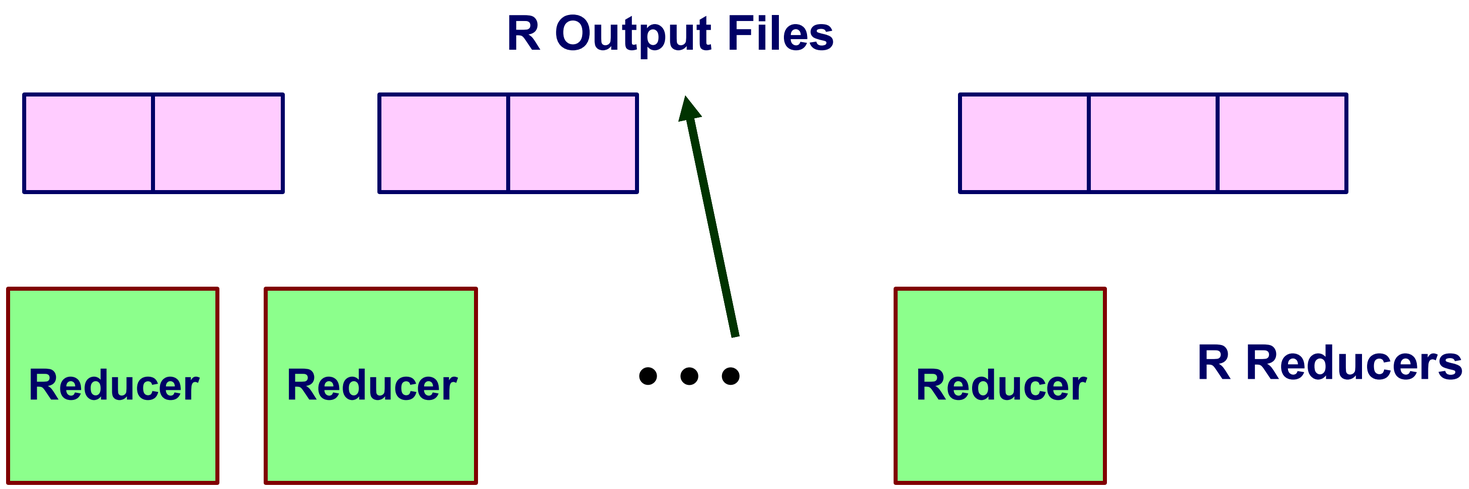
- Handles 1/R of the possible key values
- Most cases: just a hash function
- Need to handle fault tolerance



Reducer

Each Reducer:

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

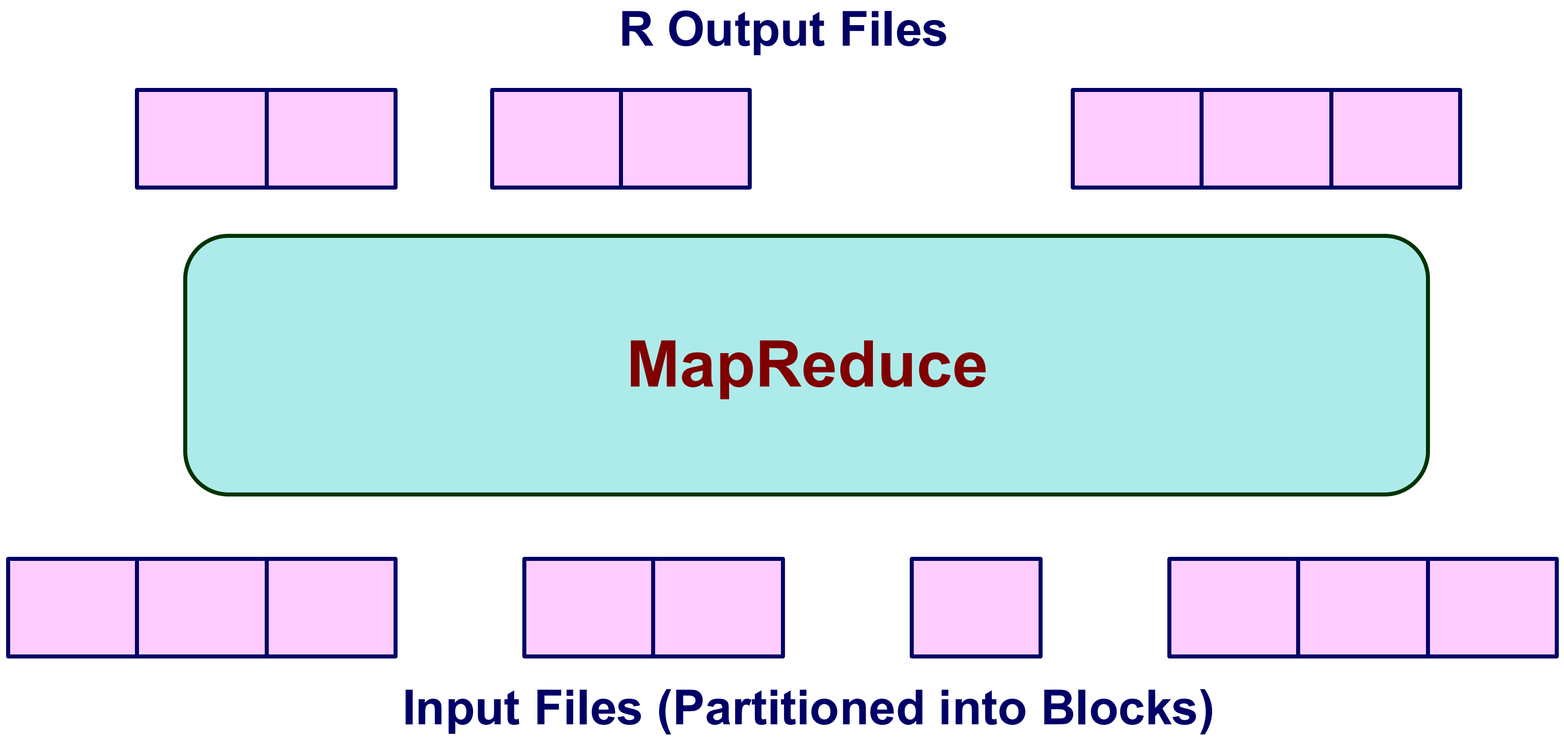


MapReduce Effect

MapReduce Step

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

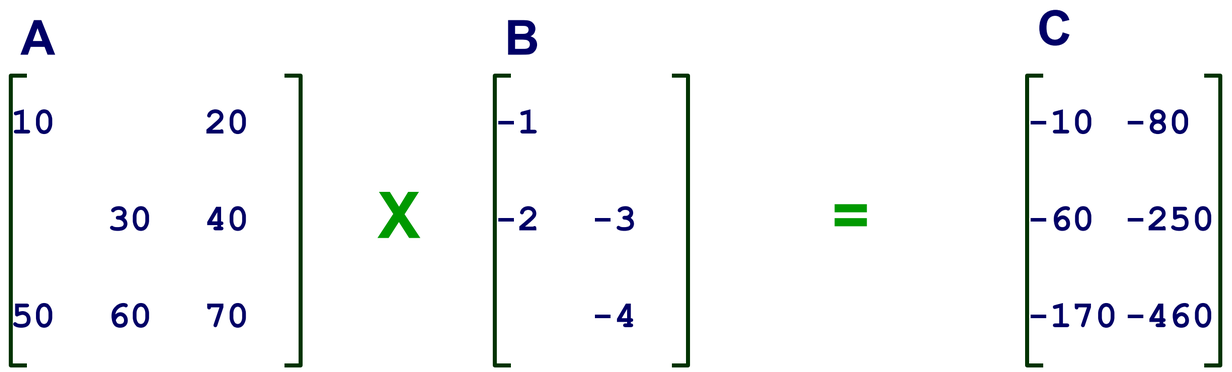
Can iterate to do more complex processing



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

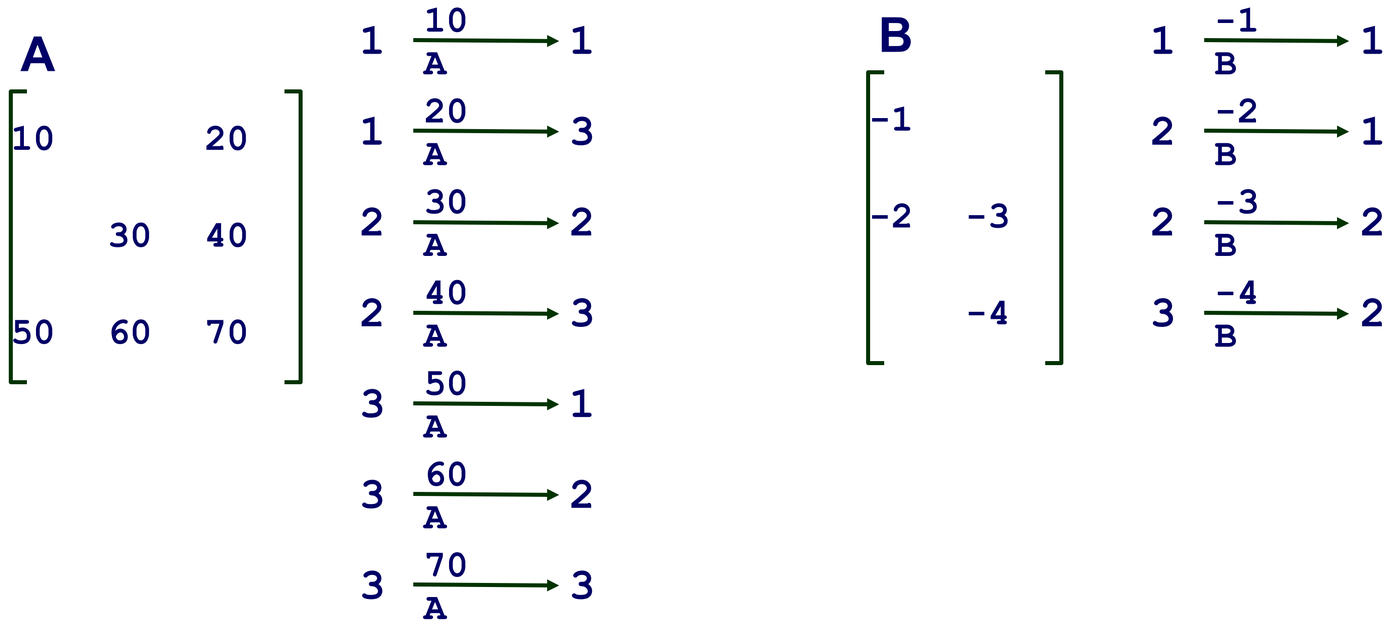


- Task: Compute product C = A·B
- Assume most matrix entries are 0

Motivation

- Core problem in scientific computing
- Challenging for parallel execution
- Demonstrate expressiveness of Map/Reduce

Computing Sparse Matrix Product

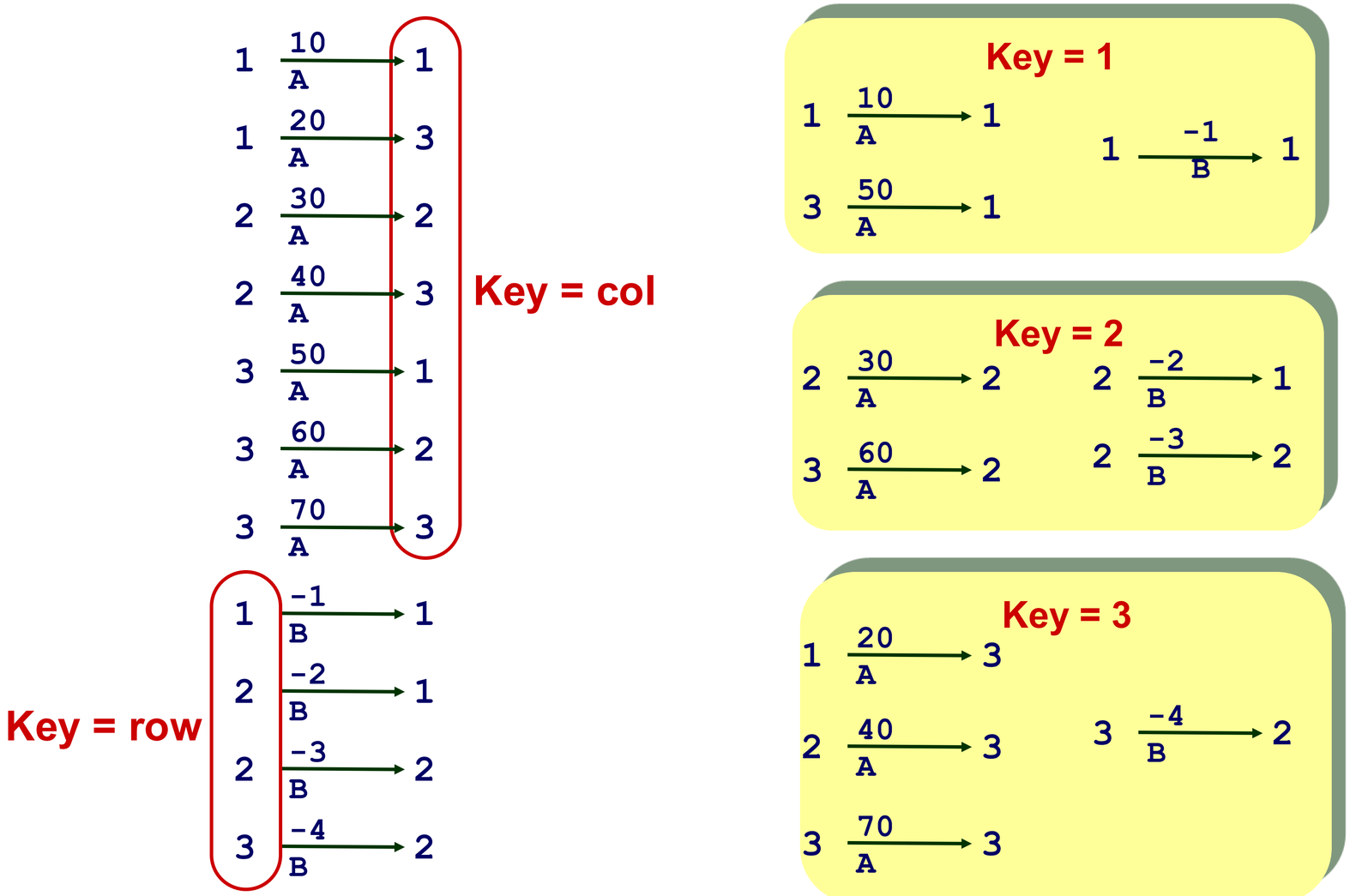


- Represent matrix as list of nonzero entries

  <row, col, value, matrixID>

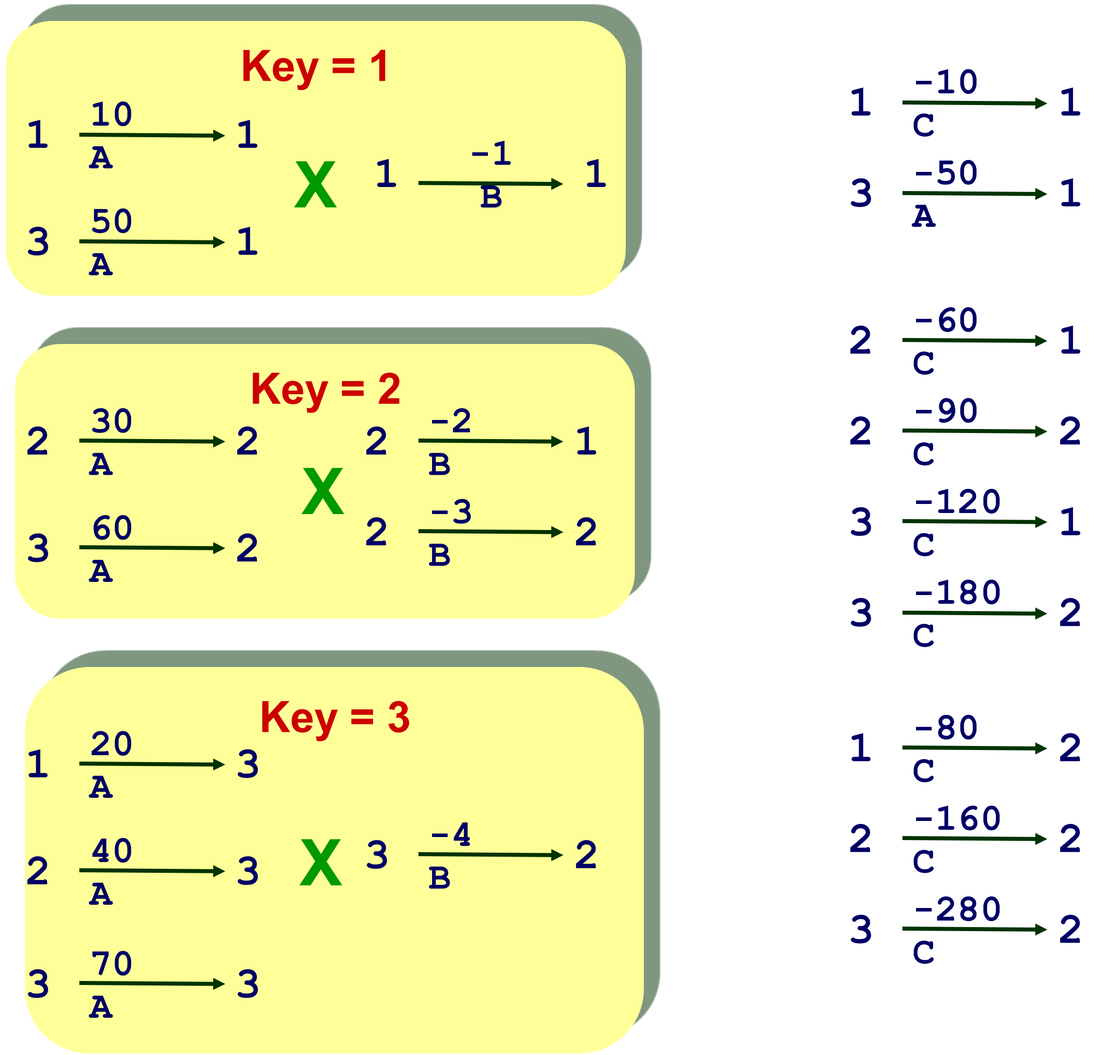
- How to represent the computation as map-reduce?
  - Phase 1: Compute all products $a_{i,k} \cdot b_{k,j}$
  - Phase 2: Sum products for each entry i,j
  - Each phase involves a Map/Reduce

Phase 1 Map of Matrix Multiply



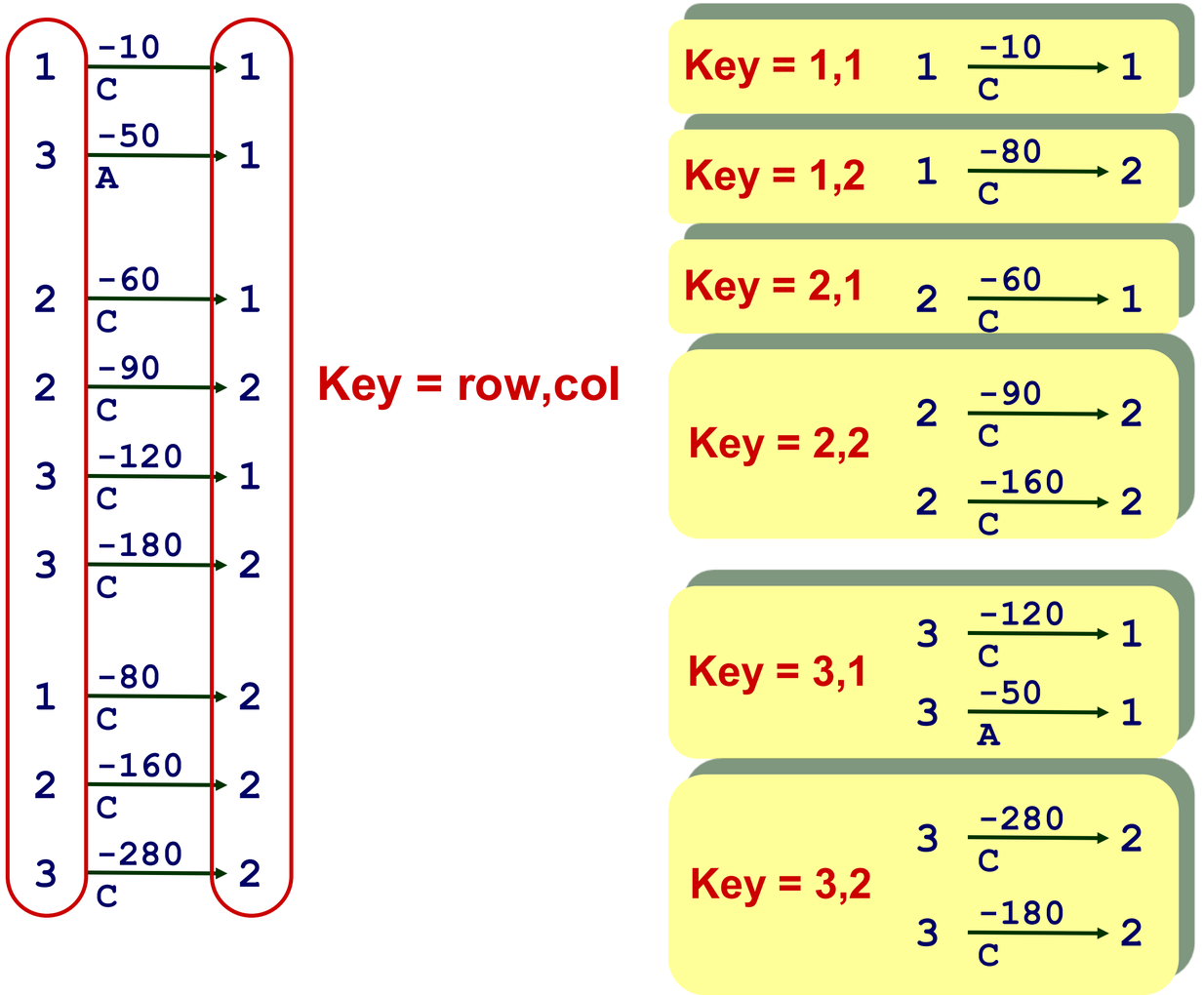
- Group values $a_{i,k}$ and $b_{k,j}$ according to key k

Phase 1 "Reduce" of Matrix Multiply



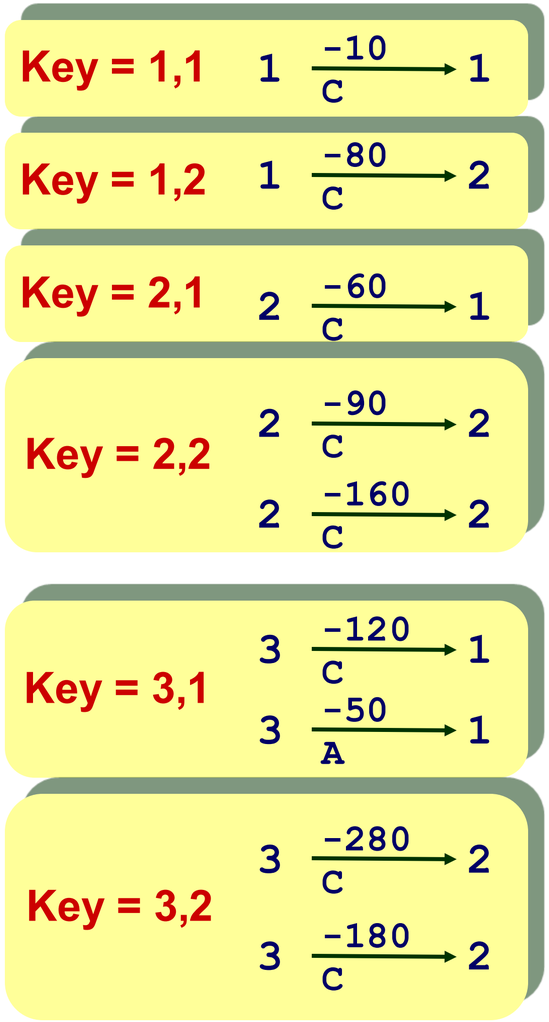
- Generate all products $a_{i,k} \cdot b_{k,j}$

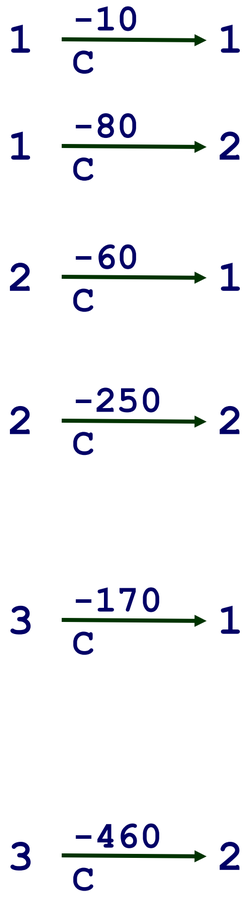
Phase 2 Map of Matrix Multiply



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

Phase 2 Reduce of Matrix Multiply





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)

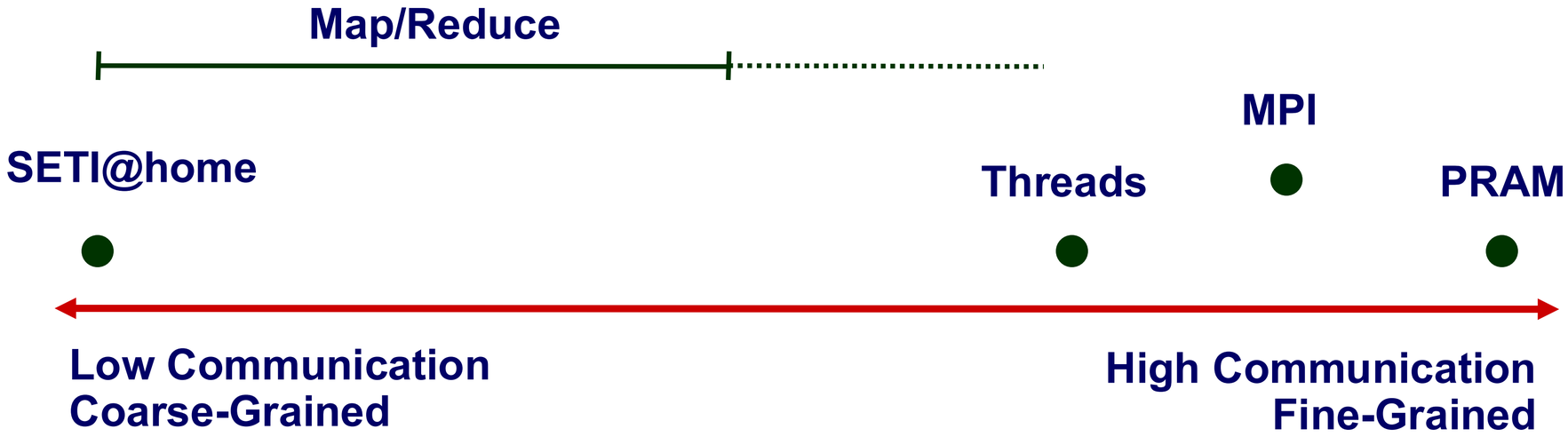
Breaks work into tasks

- 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

Exploring Parallel Computation Models



Map/Reduce Provides Coarse-Grained Parallelism

- Computation done by independent processes
- File-based communication

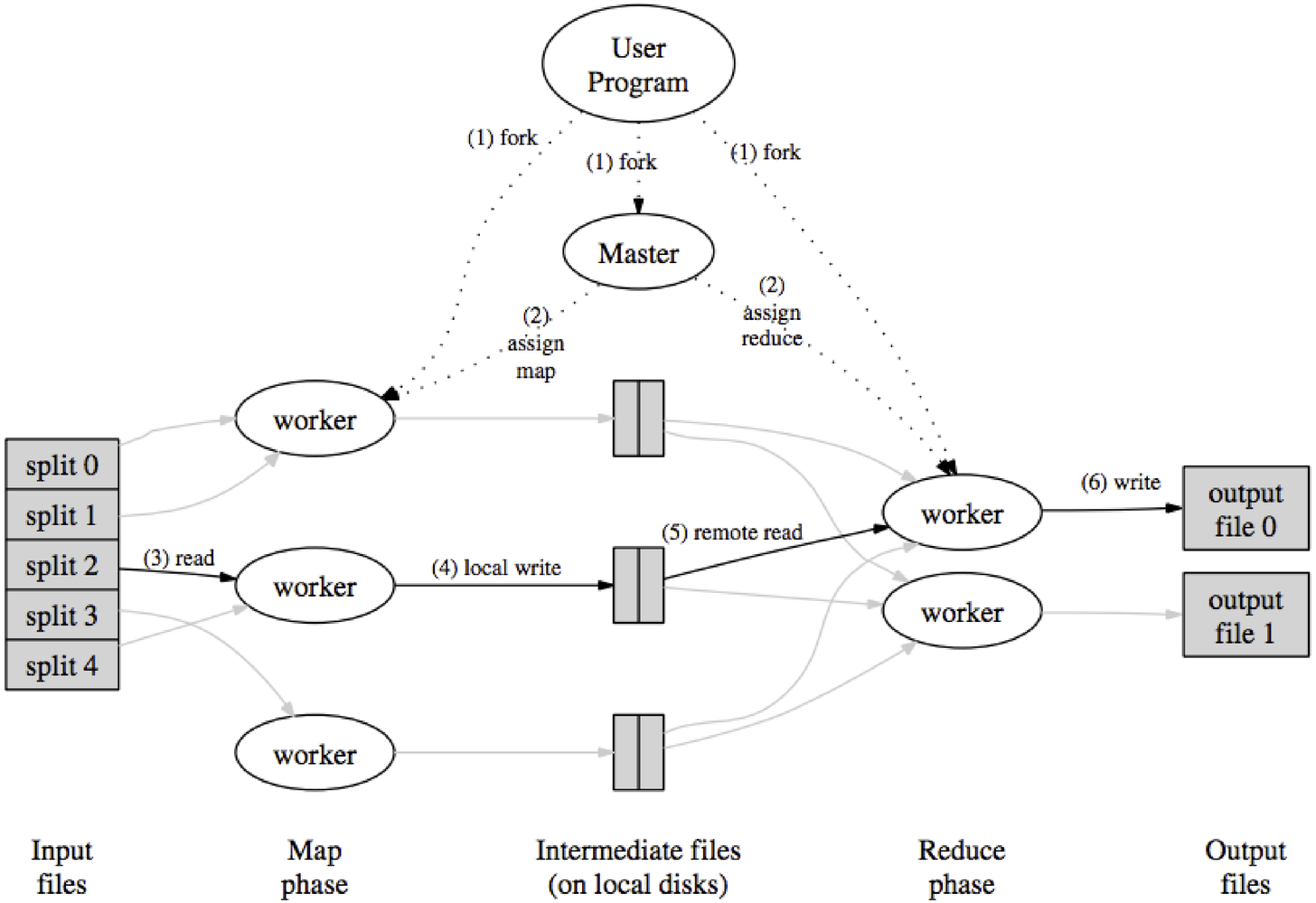
Observations

- Relatively "natural" programming model
- Research issue to explore full potential and limits

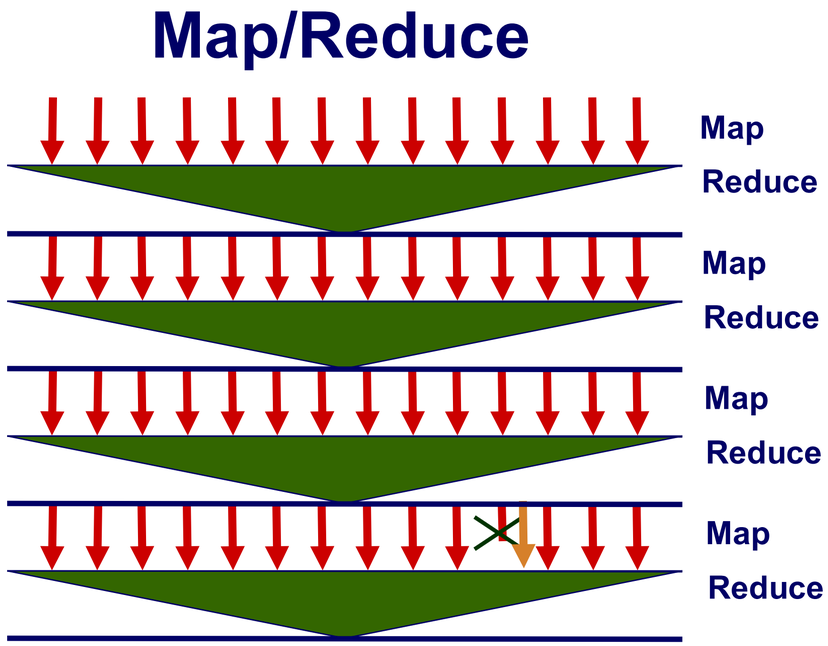
Today's topic: Batch Processing

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

MapReduce System architecture (Paper)



Fault Tolerance



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

PageRank Computation

Initially

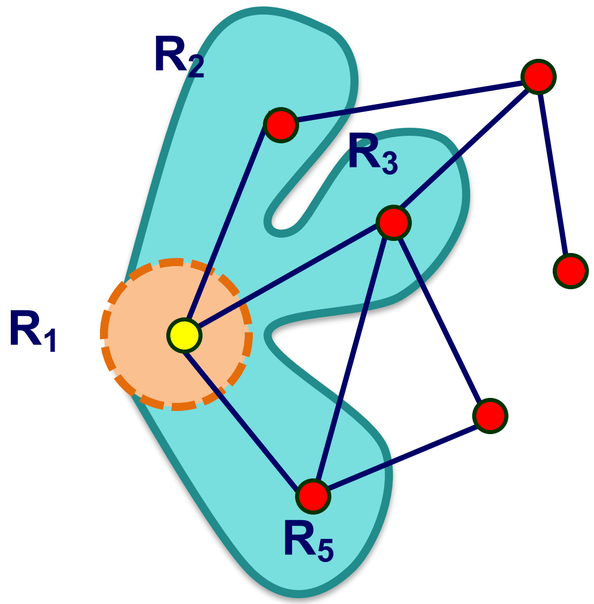
- Assign weight 1.0 to each page

Iteratively

- Select arbitrary node and update its value

Convergence

- Results unique, regardless of selection ordering



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

Q: how to express pagerank using map-reduce?