CS 398 ACC
MapReduce Part 1

Prof. Robert J. Brunner

Ben Congdon
Tyler Kim

Data Science Projects for iDSI

- Looking for people interested in working with City of Champaign Data (outside of this class)
- If interested, please contact Professor Brunner directly
- Prerequisite: INFO490 I & II or equivalent.

Administrative Reminders

- This course is experimental / new in its structure
  - An attempt to fill a niche, and *would not exist* if not for the current format
  - It’s also <u>not</u> a required course
  - We welcome feedback!

- Questions/concerns about:
  - Course content / MPs?
    - Piazza, Email list, after lecture office hours

  - Course administration?
    - Professor Brunner Office hours:
      - 12pm-1pm Tuesday, 226 Astronomy Building

Administrative Reminders

- Check Piazza for announcements
  - Some Wednesday lectures will be optional
    - i.e. Tutorial session / office hours
  - This week's lecture is not optional :)

- More on MP1 at the end of the lecture…

MP 1 & Quiz 1

MP 1 will be released later tonight.

- Due January 30th 11:59 pm

Quiz 1 will be released tomorrow.

- Due this Friday 11:55 pm

Outline

- A bit about Distributed Systems
- MapReduce Overview
- MapReduce in Industry
- Programming Hadoop MapReduce Jobs
  - Mappers and Reducers
  - Operating Model

Outline

- **A bit about Distributed Systems**
- MapReduce Overview
- MapReduce in Industry
- Programming Hadoop MapReduce Jobs
    - Mappers and Reducers
    - Operating Model

Our Primary Concerns:

- Running computation on large amounts of data
    - Want a Framework that scales from 10GB => 10TB => 10PB
- High throughput data processing
    - Not only processing lots of data, but doing so in a reasonable timeframe
- Cost efficiency in data processing
    - Workloads typically run weekly/daily/hourly (not one-off)
    - Need to be mindful of costs (hardware or otherwise)

What traditionally restricts performance?

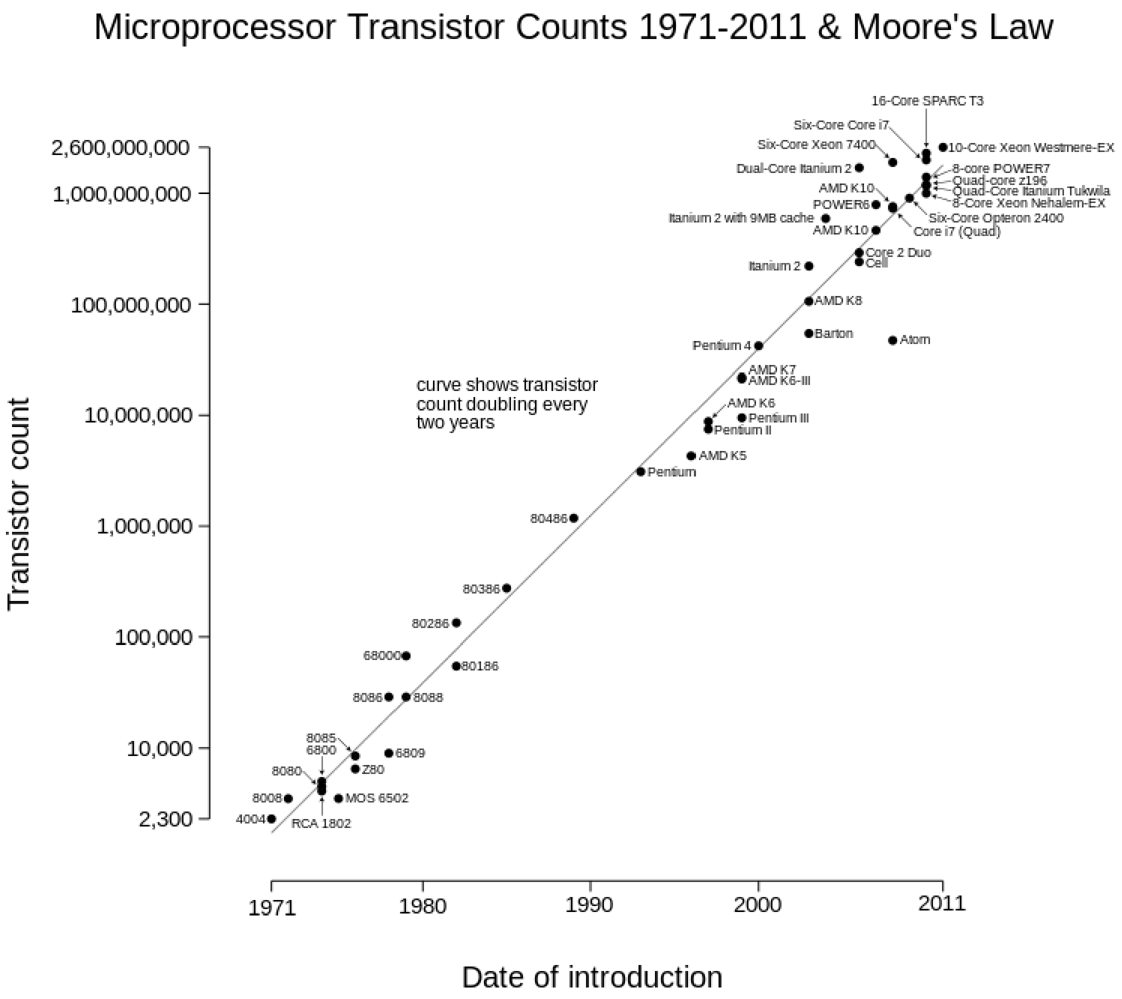
- **Processor frequency** (Computation-intensive tasks)
  - Fastest commodity processor runs at 3.7 - 4.0 Ghz
  - Rough correlation with instruction throughput

- **Network/Disk bandwidth** (Data-intensive tasks)
  - Often, data processing is computationally simple
  - Jobs become bottlenecked by network performance, instead of computational resources

Moore's Law

- The number of transistors in a dense integrated circuit doubles approximately every two years

- It's failing!



Parallelism

- If Moore's law is slowing down how can we process more data at local scale?
  - More CPU cores per processor
  - More efficient multithreading / multiprocessing

- However, there are limits to local parallelism...
  - Physical limits: CPU heat distribution, processor complexity
  - Pragmatic limits: Price per processor, what if the workload isn't CPU limited?

Distributed Systems from a Cloud Perspective

- Mindset shift from **vertical scaling** to **horizontal scaling**
    - Don’t increase performance of each computer
    - Instead, use a pool of computers - (a datacenter, “the cloud”)
    - Increase performance by adding new computer to pool
        - (Or, buy purchasing more resources from a cloud vendor)

Distributed Systems from a Cloud Perspective

- **Vertical Scaling** - “The old way”
    - Need more processing power?
        - Add more CPU cores to your existing machines
    - Need more memory?
        - Add more physical memory to your existing machines
    - Need more network bandwidth?
        - Buy/install more expensive networking equipment

Distributed Systems from a Cloud Perspective

- **Horizontal Scaling**
  - Standardize on commodity hardware
    - Still server-grade, but before diminishing returns kicks in
  - Need more CPUs / Memory / Bandwidth?
    - Add more (similarly spec'd) machines to your total resource pool
  - Still need to invest in good core infrastructure (machine interconnection)
    - However, commercial clouds are willing to do this work for you

- Empirically, horizontal scaling works really well if done right:
  - This is how Google, Facebook, Amazon, Twitter, et al. achieve high performance
  - Also changes how we write code
    - We can no longer consider our code to only run sequentially on one computer

Outline

- A bit about Distributed Systems
- **MapReduce Overview**
- MapReduce in Industry
- Programming Hadoop MapReduce Jobs
  - Mappers and Reducers
  - Operating Model

MapReduce

- **What it is:**
    - A programming paradigm to break data processing jobs into distinct stages which can be run in a distributed setting
- **Big Idea:**
    - Restrict programming model to get parallelism “for free”

- Most large-scale data processing is free of “data dependencies”
    - Results of processing one piece of data not tightly coupled with results of processing another piece of data
    - Increase throughput by distributing chunks of the input dataset to different machines, so the job can execute in parallel

MapReduce

- A job is defined by 2 distinct stages:
  - **Map** - Transformation / Filtering
  - **Reduce** - Aggregation

- Data is described by key/value pairs
  - **Key** - An identifier of data
    - I.e. User ID, time period, record identifier, etc.
  - **Value** - Workload specific data associated with key
    - I.e. number of occurences, text, measurement, etc.

Map & Reduce

Map

- A function to process **input** key/value pairs to generate a set of **intermediate** key/value pairs.
- Values are grouped together by **intermediate key** and sent to the Reduce function.

Reduce

- A function that merges all the **intermediate** values associated with the same intermediate key into some **output** key/value per intermediate key

<key_input, val_input> ⇒ <key_inter, val_inter> ⇒ <key_out, val_out>

Map *Reduce*

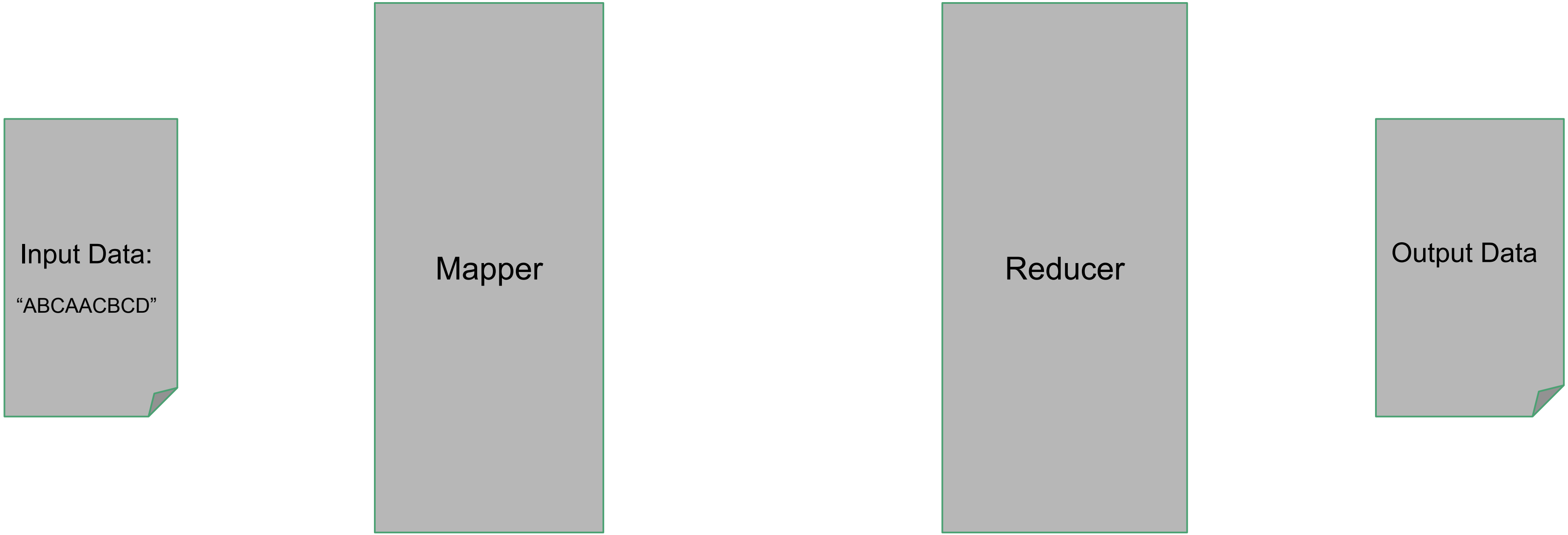
Map & Reduce - Word Count

- **Problem:** Given a “large” amount of text data, how many occurences of each individual word are there?
    - Essentially a “count by key” operation

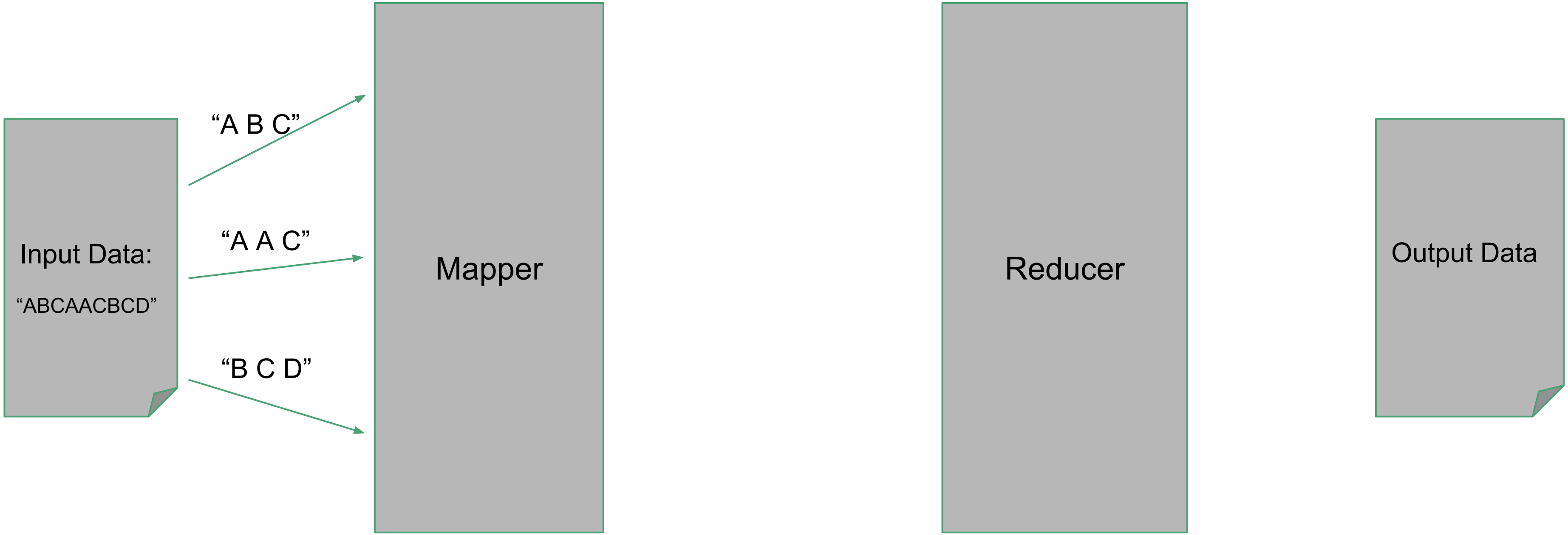
- Generalizes to other tasks:
    - Counting user engagements, aggregating log entries by machine, etc.

- Map Phase:
    - Split text into words, emitting (“word”, 1) pairs
- Reduce Phase:
    - Calculate the sum of occurrences per word

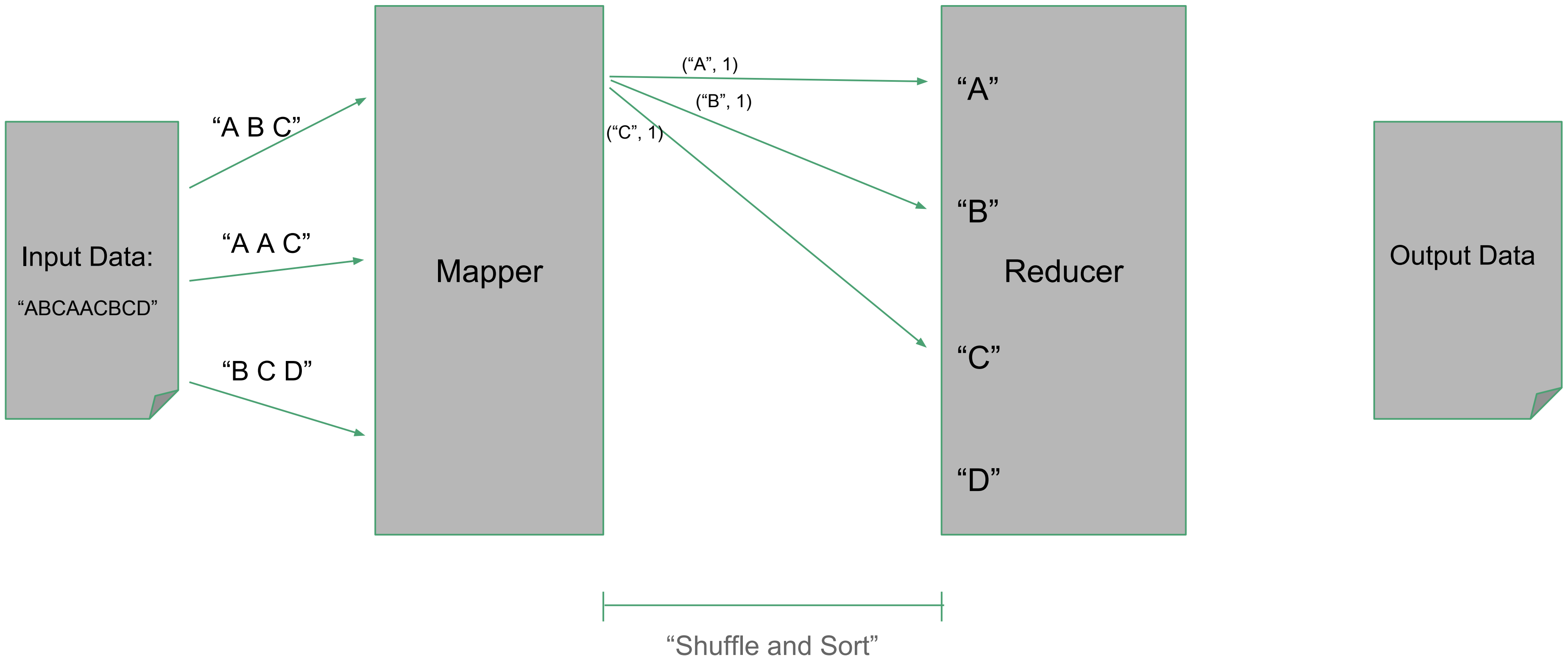
Map & Reduce - Word Count



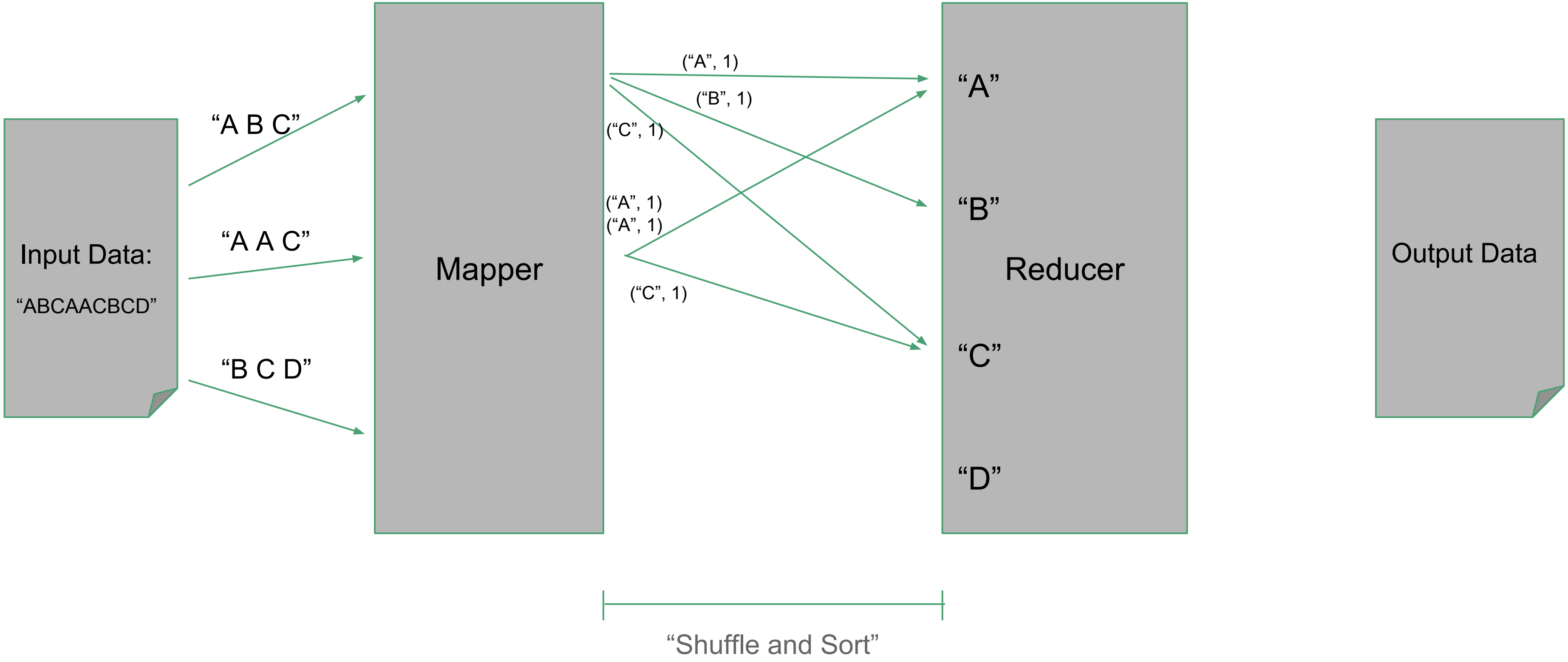
Map & Reduce - Word Count



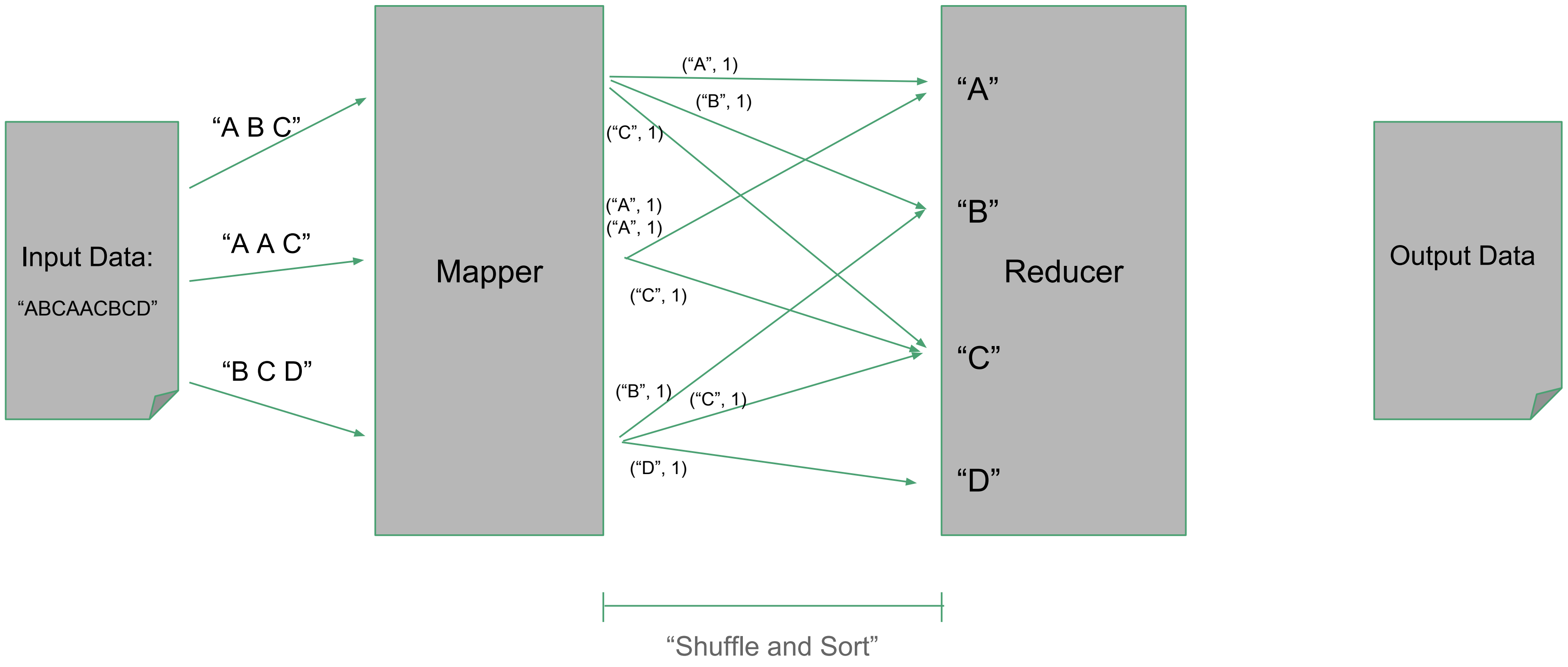
Map & Reduce - Word Count



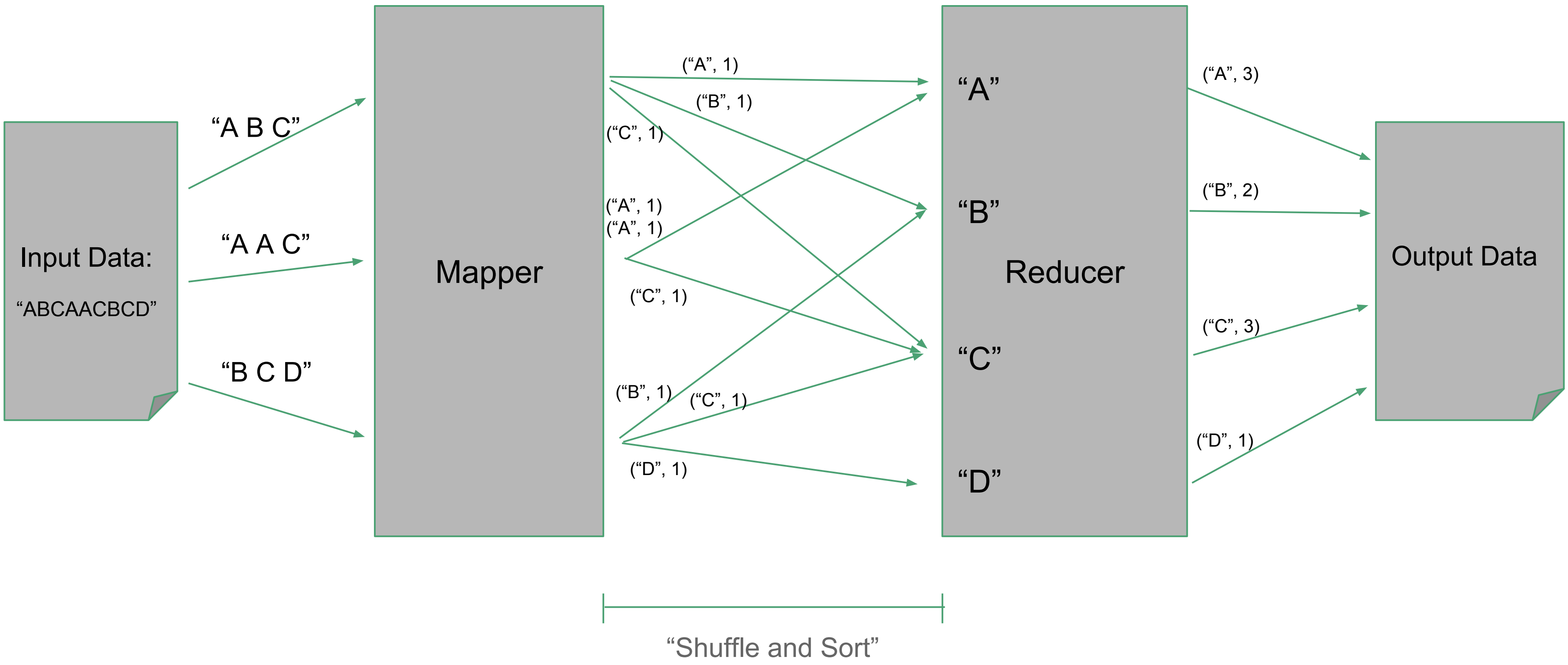
Map & Reduce - Word Count



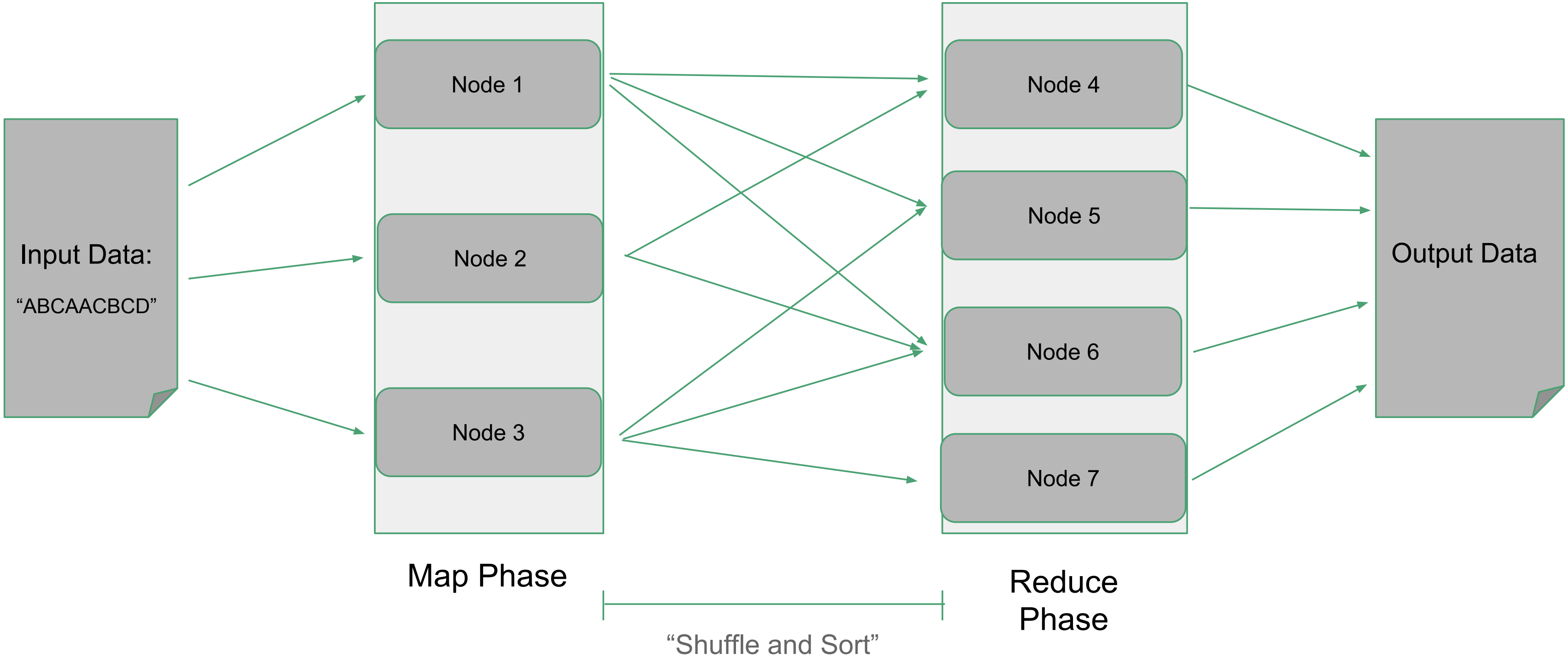
Map & Reduce - Word Count



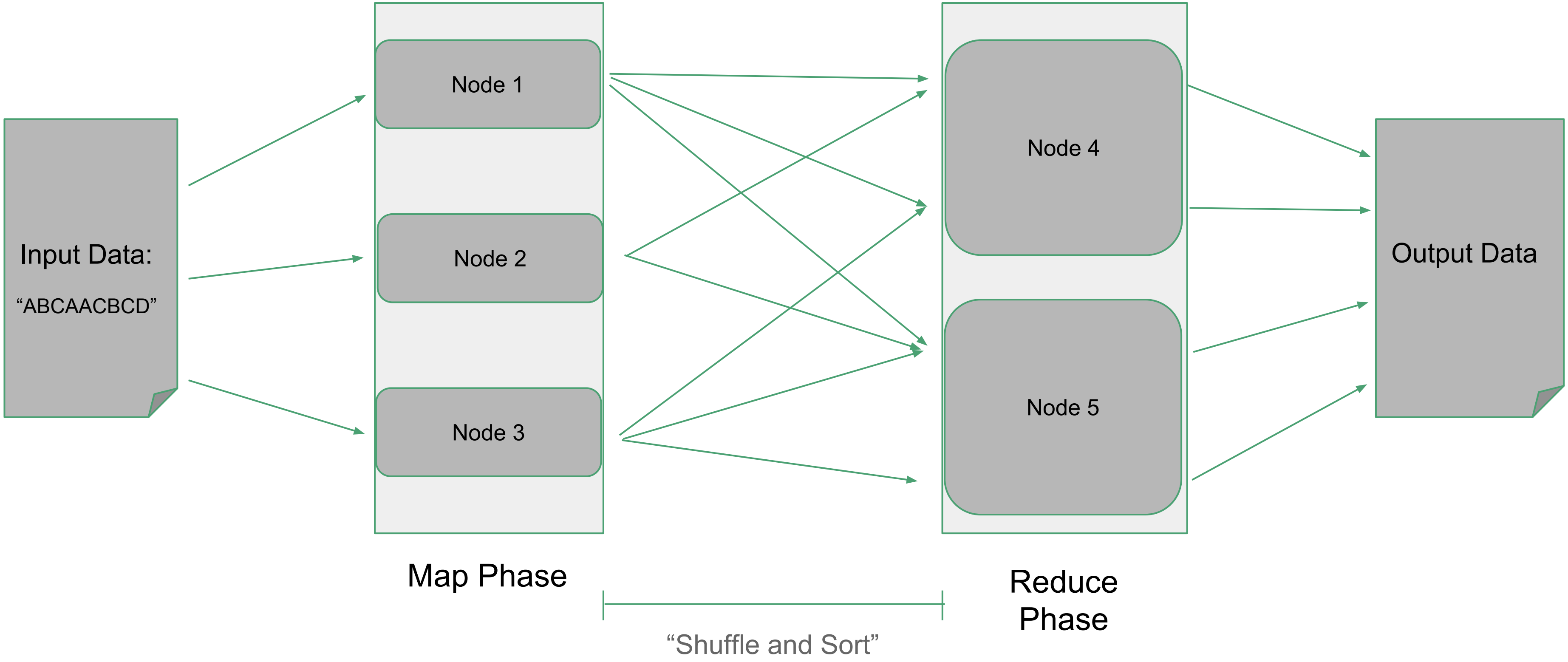
Map & Reduce - Word Count



Map & Reduce - Word Count



Map & Reduce - Word Count



Map & Reduce

- Why is **Map** parallelizable?
  - Input data split into *independent* chunks which can be transformed / filtered *independently* of other data

- Why is **Reduce** parallelizable?
  - The aggregate value *per key* is only dependent on values associated with *that key*
  - All values associated with a certain key are processed on the same node

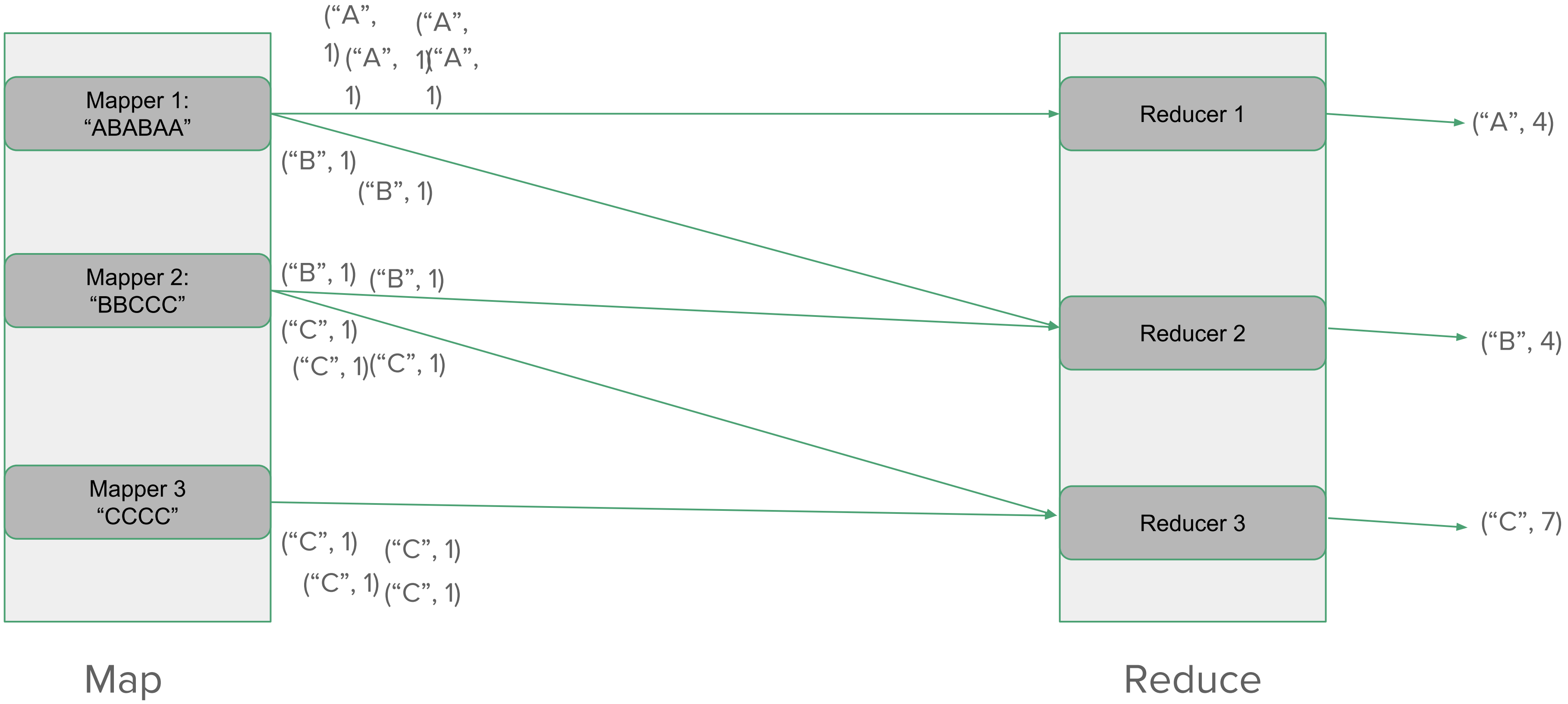
- What do we give up in using MR?
  - Can’t “cheat” and have results depend on side-effects, global state, or partial results of another key

Map & Reduce - Shuffle/Sort In-Depth

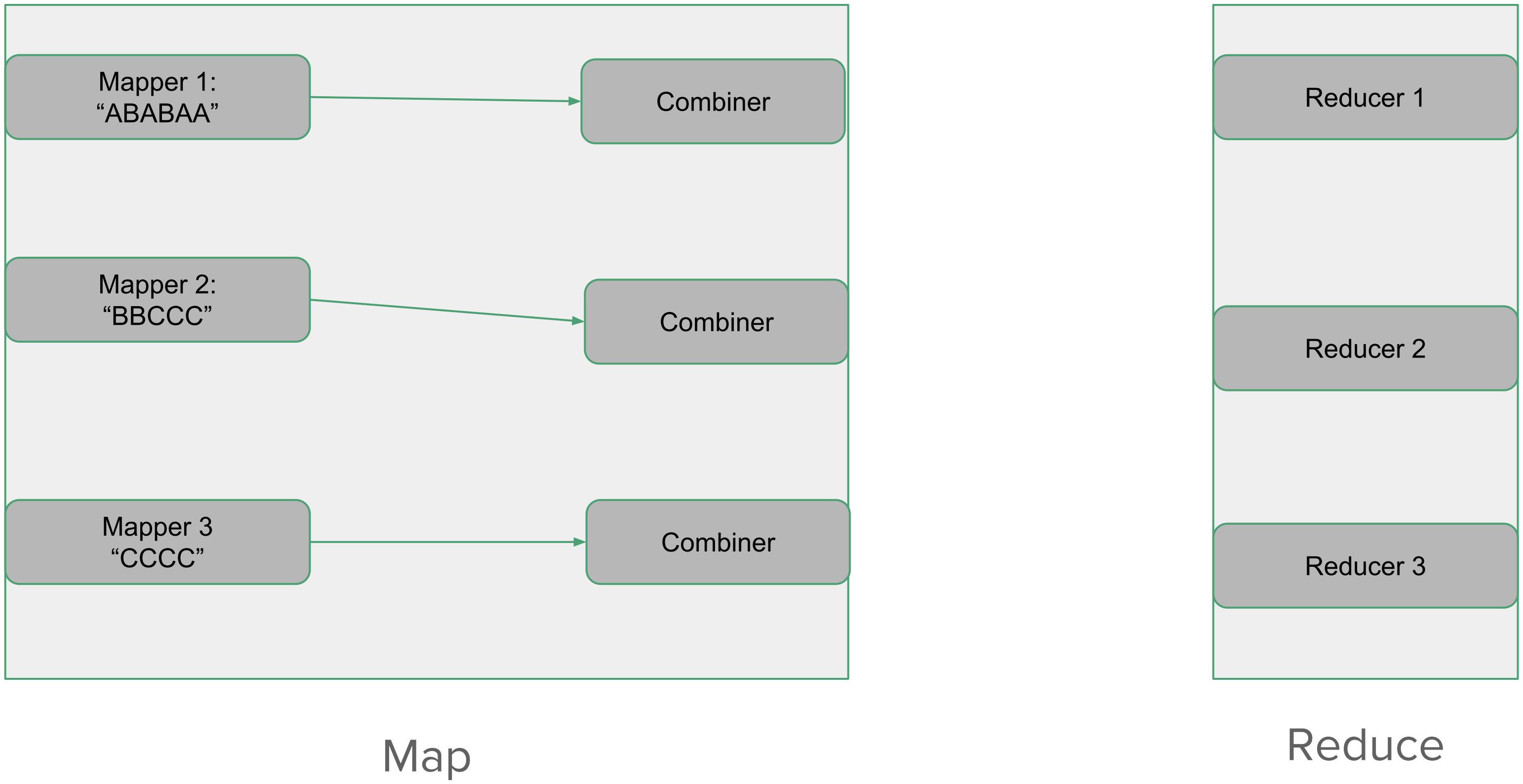
1. **Combiner** - Optional
    - Optional step at end of Map Phase to pre-combine intermediate values before sending to reducer
    - Like a reducer, but run by the mapper (usually to reduce bandwidth)
2. **Partition / Shuffle**
    - Mappers send intermediate data to reducers by key (key determines which reducer is the recipient)
    - "Shuffle" because intermediate output of each mapper is broken up by key and redistributed to reducers
3. **Secondary Sort** - Optional
    - Sort within keys by value
    - Value stream to reducers will be in sorted order

Map & Reduce - Shuffle/Sort - Combiner

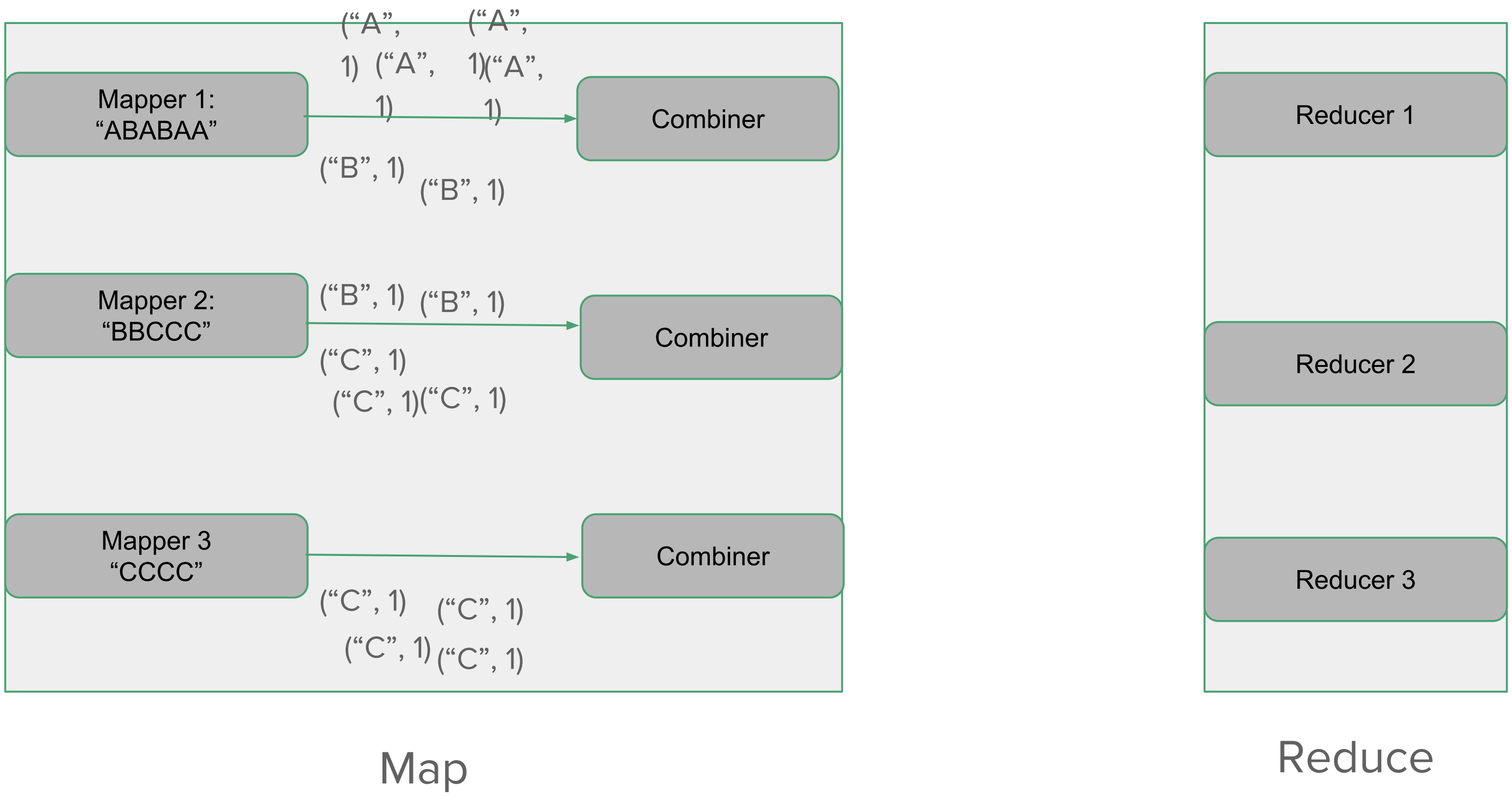
Map & Reduce - Shuffle/Sort - Combiner



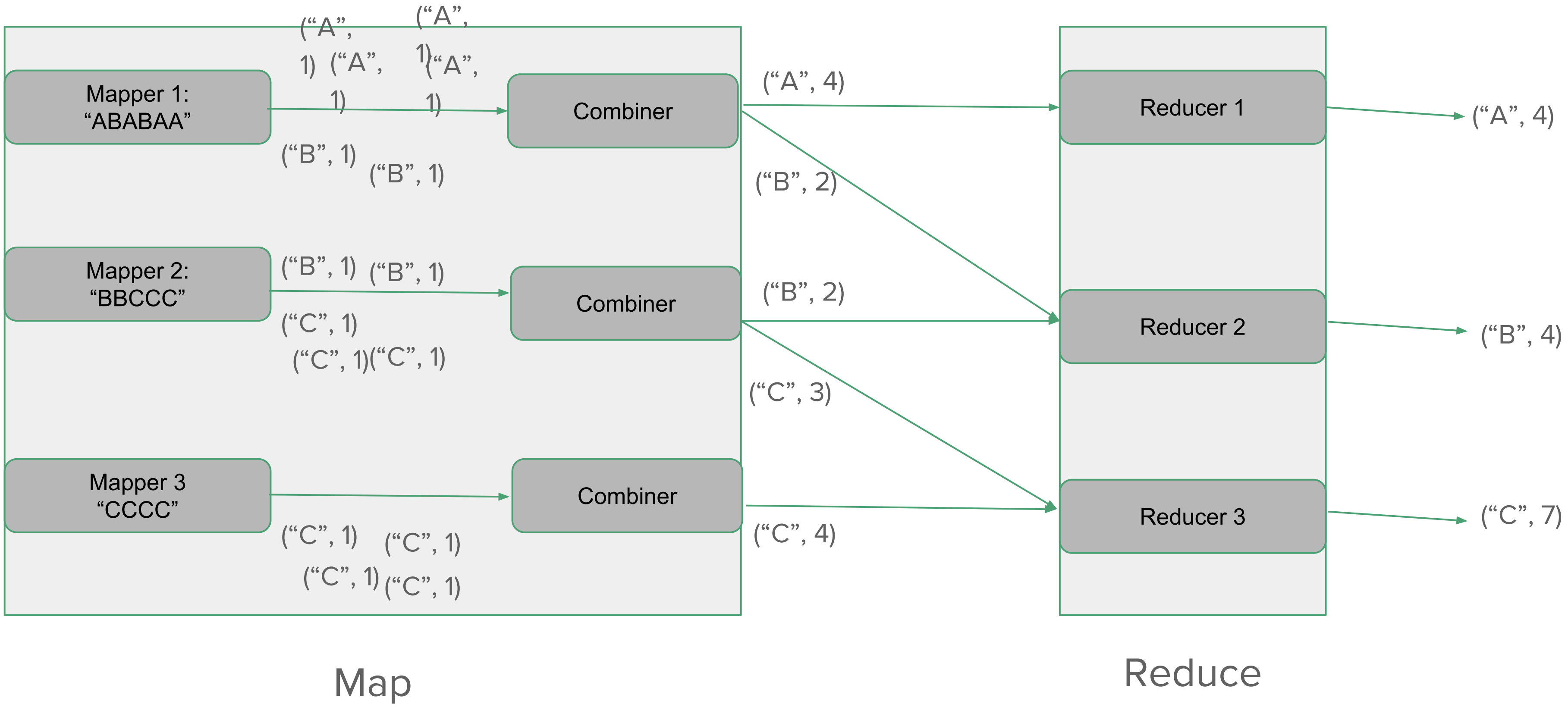
Map & Reduce - Shuffle/Sort - Combiner



Map & Reduce - Shuffle/Sort - Combiner



Map & Reduce - Shuffle/Sort - Combiner



Map & Reduce - Caveats

- What if we need data that is dependent on another key?
  - Solution: Chain MapReduce jobs together
  - Job 1: Calculate necessary subconditions per each key
  - Job 2: Determine final aggregate value

- Chaining MapReduce Jobs
  - Output of n^{th} job is the input to the $(n+1)^{th}$ job
  - Very useful in practice!
  - Try to minimize number of stages, because bandwidth overhead per stage is high
    - MapReduce tends to be naive in this area

Outline

- A bit about Distributed Systems
- MapReduce Overview
- **MapReduce in Industry**
- Programming Hadoop MapReduce Jobs
    - Mappers and Reducers
    - Operating Model

Hadoop MapReduce

- **What it is:** Specific implementation of a MapReduce system

- What Hadoop MapReduce gives us:
  - A means of automatically distributing work across machines
  - Scheduling of jobs
  - Fault tolerance
  - Cluster monitoring and job tracking

- How? An underlying resource manager (more on this later)

Hadoop MapReduce

- **How fast is it?**

  - Benchmarks based on sorting large datasets (synthetic load)
  - Hadoop Record: 1.42TB / min
    - Record set in 2013 using a 2100 node cluster
    - Since 2014, Spark (and others) have been faster

MapReduce in Industry

- **Compelling Use Cases:**
  - Batch Processing
    - Analyzing data “at rest” (i.e. daily/hourly jobs, not streaming data)
    - i.e. Log Processing, User data transformation / analysis, web scraping
  - Workloads that can be broken into a single (or few) distinct Map/Reduce phases
    - Poor results on iterative workloads
      - Non-parallizable workloads => Many MR stages => High bandwitdh overhead

MapReduce in Industry

- **Google**
  - Released MapReduce whitepaper in 2004, detailing their use of MR to process large datasets
  - Inspired Hadoop MapReduce (Open source implementation)

- **Twitter**
  - Uses MapReduce to “process tweets, log files, and many other types of data”

MapReduce in Industry

- **Facebook**
  - Maintains 2 Hadoop clusters with 1400 total machines and 10,000+ processing cores, 15PB of storage

- **Spotify**
  - Runs 20k+ Hadoop jobs daily
  - Uses Hadoop for “content generation, data aggregation, reporting, analysis”

Wednesday:

- Writing MapReduce jobs
- Specific MapReduce use cases

MP Logistics

- Please log into the **UIUC Gitlab** if you have not already
    - http://gitlab.engr.illinois.edu

- Make sure you have access to a system with **Python3**
    - Install via Miniconda / Package Manager
    - Use an EWS Workstation

- Post on Piazza if you encounter issues

MP 1

Due next Tuesday (1/30) at 11:59pm

Introduces how to run MapReduce using in Python on a single machine

> Check Piazza for Q&A and Announcements