“Civilization advances by extending the number of important operations which we can perform without thinking about them”

Alfred North Whitehead, Professor at Harvard, 1910s

Hands-on H4
MapReduce Design Patterns

CS205: Computing Foundations for Computational Science
Dr. Ignacio M. Llorente
Spring Term 2020

IACS
INSTITUTE FOR APPLIED COMPUTATIONAL SCIENCE
AT HARVARD UNIVERSITY

HARVARD
School of Engineering
and Applied Sciences

Before We Start

Where We Are

Computing Foundations for Computational and Data Science

How to use modern computing platforms in solving scientific problems

Intro: Large-Scale Computational and Data Science

A. Parallel Processing Fundamentals

B. Parallel Computing

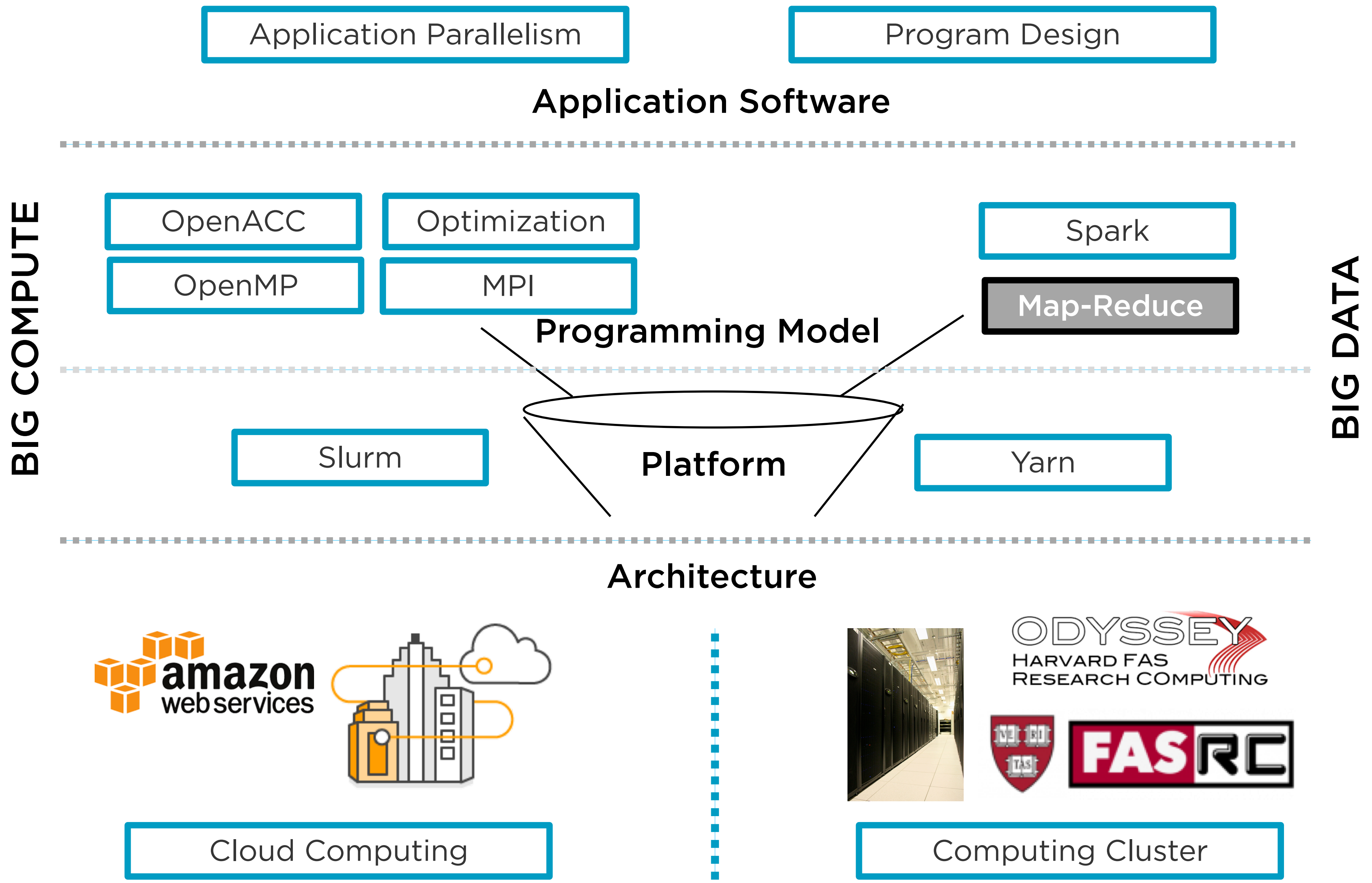
C. Parallel Data Processing

- **C1. Batch Data Processing**
- C2. Dataflow Processing
- C3. Stream Data Processing

Wrap-Up: Advanced Topics

CS205: Contents

APPLICATION SOFTWARE



Application Parallelism

Program Design

Application Software

BIG COMPUTE

OpenACC

Optimization

OpenMP

MPI

Programming Model

Spark

Map-Reduce

BIG DATA

Slurm

Platform

Yarn

Architecture

amazon web services

ODYSSEY HARVARD FAS RESEARCH COMPUTING

FASRC

Cloud Computing

Computing Cluster

Before We Start

Where We Are

Concepts

Platform

Programming

Week 9: **Batch Data Processing** => MapReduce

3/23	3/24	3/25	3/26	3/27
	Lecture C1 Batch Data Processing (Quiz & Reading)	Lab I8 MapReduce Hadoop Cluster	Hands-on H4 MapReduce Programming	

Week 10: **Dataflow Processing => Spark**

3/30	3/31	4/1	4/2	4/3
	Lecture C2 Dataflow Processing (Quiz & Reading)	Lab I9 Spark Single Node	Hands-on H5 Spark Programming	

Context

MapReduce Programming Model

The programmer essentially only specifies two (sequential) functions

STEP 1. MAP: *map(k1,v1) → list(**k2**,v2)*

- Inputs data record and outputs a set of intermediate key-value pairs, each of type k2 and v2
- Types can be simple or complex user-defined objects
- Each map call is **fully independent (no execution ordering, sync or comm)**

STEP 2. SUFFLING: Internal grouping of all intermediate pairs with same key together and passes them to the workers executing reduce

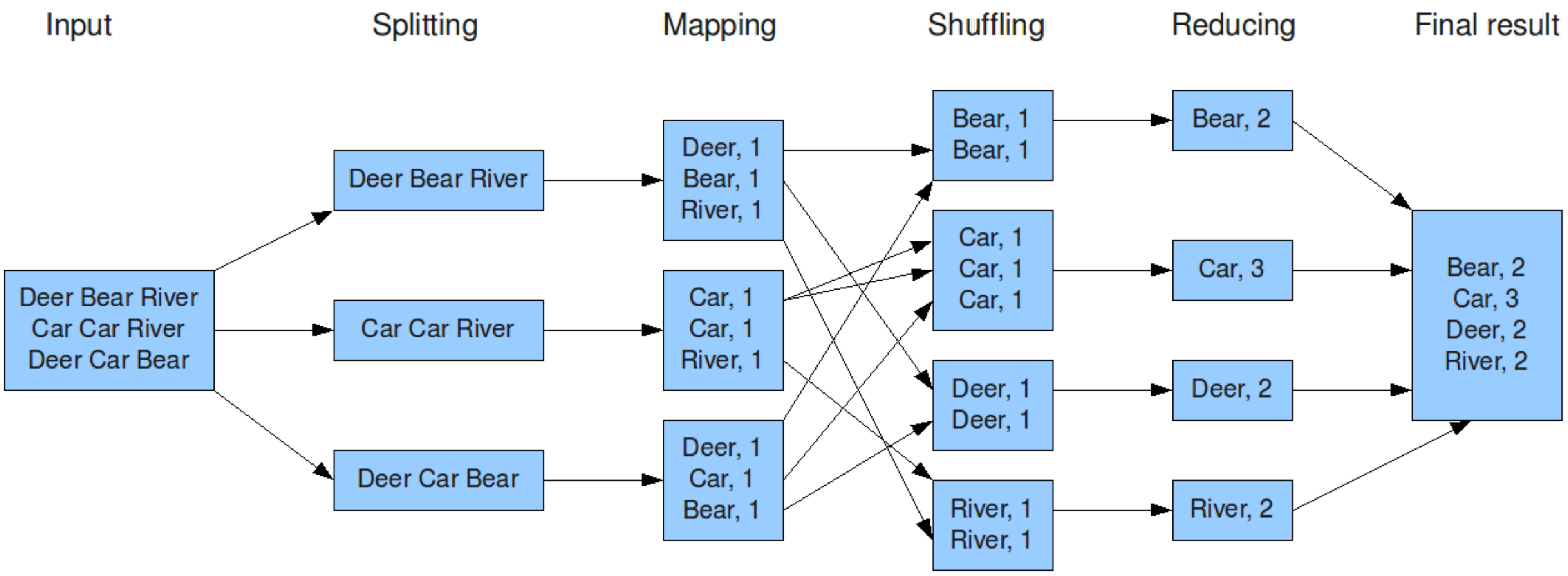
STEP 3. REDUCE: *reduce(**k2**,list(v2)) → list(k3,v3)*

- Combines information across records that share this same **intermediate key**
- Each reduce call is **fully independent**

Context

MapReduce Programming Model

The overall MapReduce word count process



Hands-on Examples

Requirements

Both the mapper and the reducer should be python executable scripts that read the input from stdin (line by line) and emit the output to stdout

```
$ cat files | mapper.py| sort | reducer.py
```

1. Unix-like shell (Linux, Mac OS or Windows/Cygwin)
2. Python installed

Roadmap

MapReduce Design Patterns

Design Patterns

Summarization

Inverted Index

Filtering

Other Patterns

Design Patterns

What Are Design Patterns?

- ✓ **Reusable** solutions to problems (**HWC!**)
- ✓ Domain **independent**
- ✓ Not a cookbook
- ✓ Not a guide
- ✓ Not a finished solution

Design Patterns

Why Design Patterns?

- ✓ Makes the intent of model and platform **easier to understand**
- ✓ Provides a **common language** for solutions
- ✓ Be able to **reuse code**
- ✓ Describes known **performance profiles and limitations** of solutions

Design Patterns

When Should I Use MapReduce?

Query

- Index and Search: inverted index
- Filtering
- Classification

Analytics

- Summarization and statistics
- Sorting and merging
- Frequency distribution
- SQL-based queries: group-by, having, etc.
- Generation of graphics: histograms, scatter plots.

. . . large datasets in off-line mode for boosting other on-line processes

Main Patterns

1. Summarization
2. Inverted Index
3. Filtering

Summarization

Calculating Aggregate Statistical Values

Description

- A general pattern for calculating aggregate statistical values over your data

Intent

- Group records together by a key field and calculate a numerical aggregate per group to get a top-level view of the larger data set

Examples

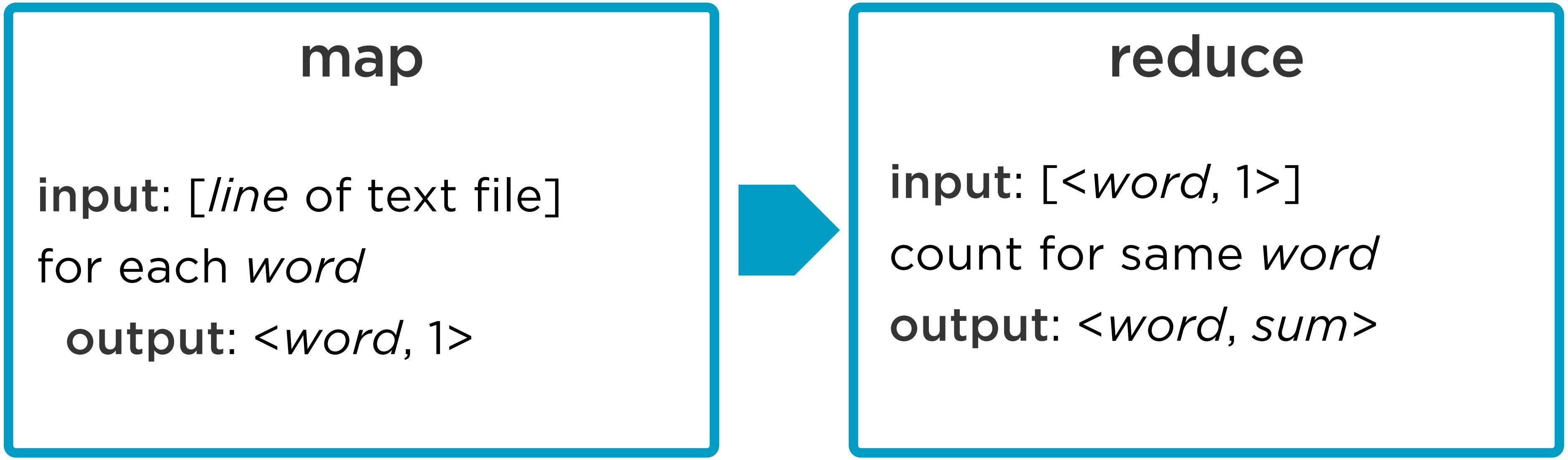
1. Word count
2. Record count
3. Min/Max/Count
4. Average/Median/Standard deviation
5. …

Summarization

Word Count

Find the frequency of each word in text files

- **Map**: Process lines and generate as output <*word*, 1>
- **Reduce**: Add all values for the same *word*



map

input: [*line* of text file]

for each *word*

  output: <*word*, 1>

reduce

input: [<*word*, 1>]

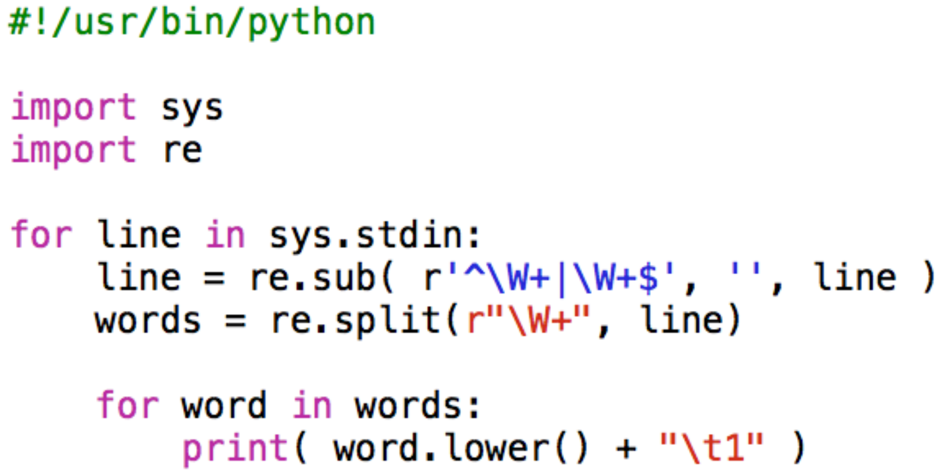
count for same *word*

output: <*word*, *sum*>

Summarization

Word Count

mapper.py



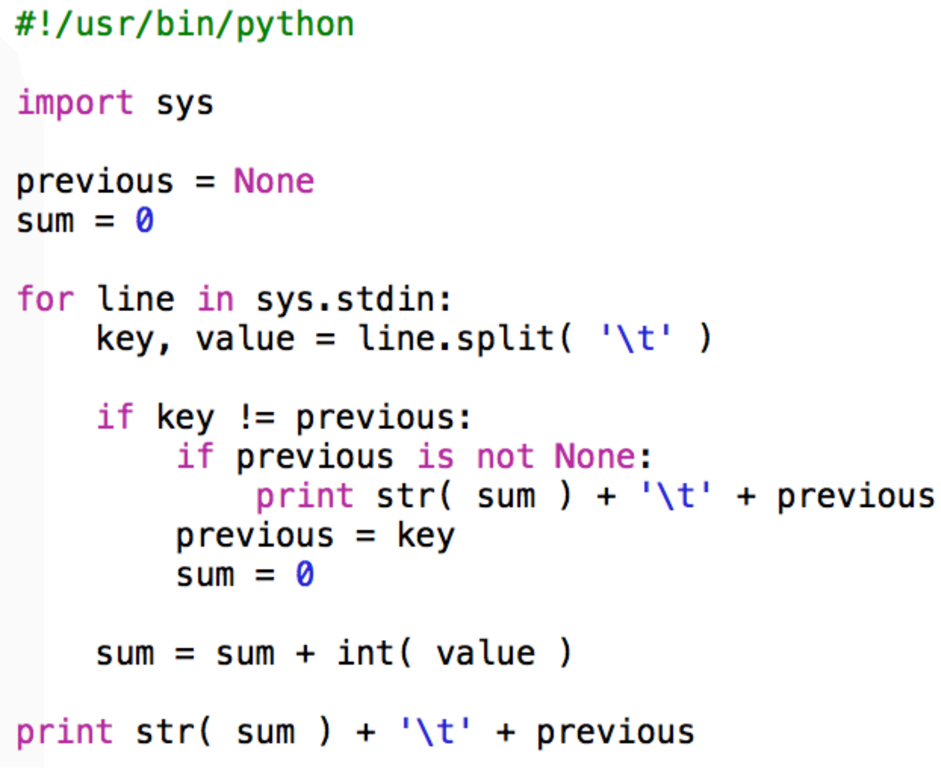
```
#!/usr/bin/python

import sys
import re

for line in sys.stdin:
    line = re.sub( r'^\W+|\W+$', '', line )
    words = re.split(r"\W+", line)

    for word in words:
        print( word.lower() + "\t1" )
```

reducer.py



```
#!/usr/bin/python

import sys

previous = None
sum = 0

for line in sys.stdin:
    key, value = line.split( '\t' )

    if key != previous:
        if previous is not None:
            print str( sum ) + '\t' + previous
        previous = key
        sum = 0

    sum = sum + int( value )

print str( sum ) + '\t' + previous
```

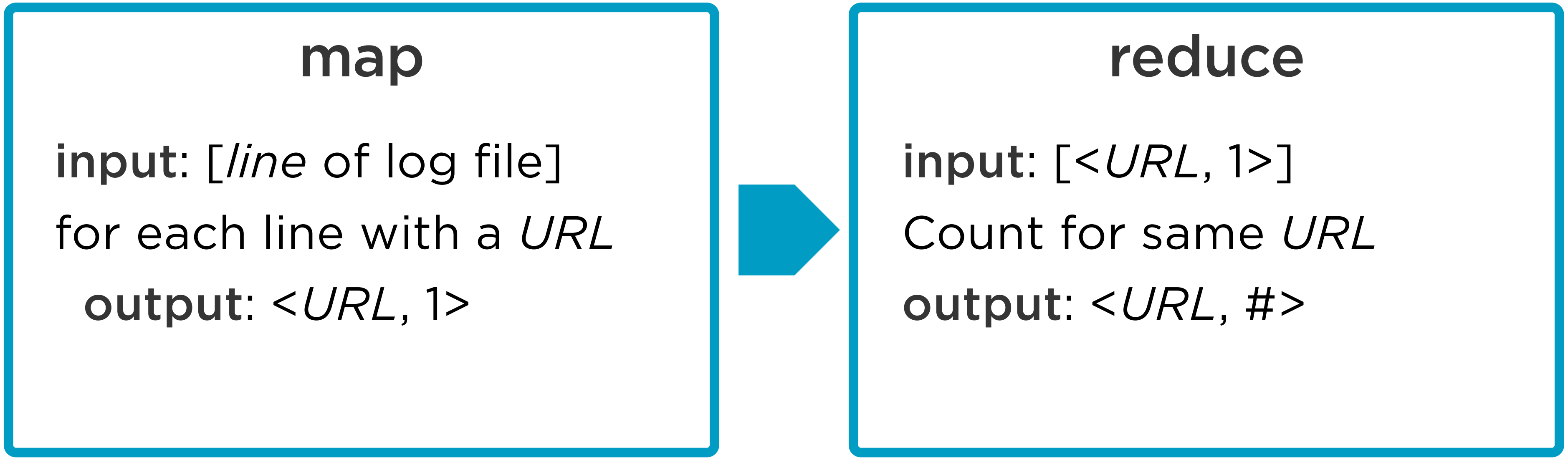
Summarization

Record Count

Find the frequency of each URL in web logs

- **Map**: Process web page access logs and generate <*URL*, 1> as output
- **Reduce**: Add all values for the same *URL*

```
64.242.88.10 - - [07/Mar/2004:16:37:27 -0800] "GET /twiki/bin/view/TWiki/DontNotify HTTP/1.1" 200 4140
64.242.88.10 - - [07/Mar/2004:16:39:24 -0800] "GET /twiki/bin/view/Main/TokyoOffice HTTP/1.1" 200 3853
…
```



map

input: [*line* of log file]

for each line with a *URL*

output: <*URL*, 1>

reduce

input: [<*URL*, 1>]

Count for same *URL*

output: <*URL*, #>

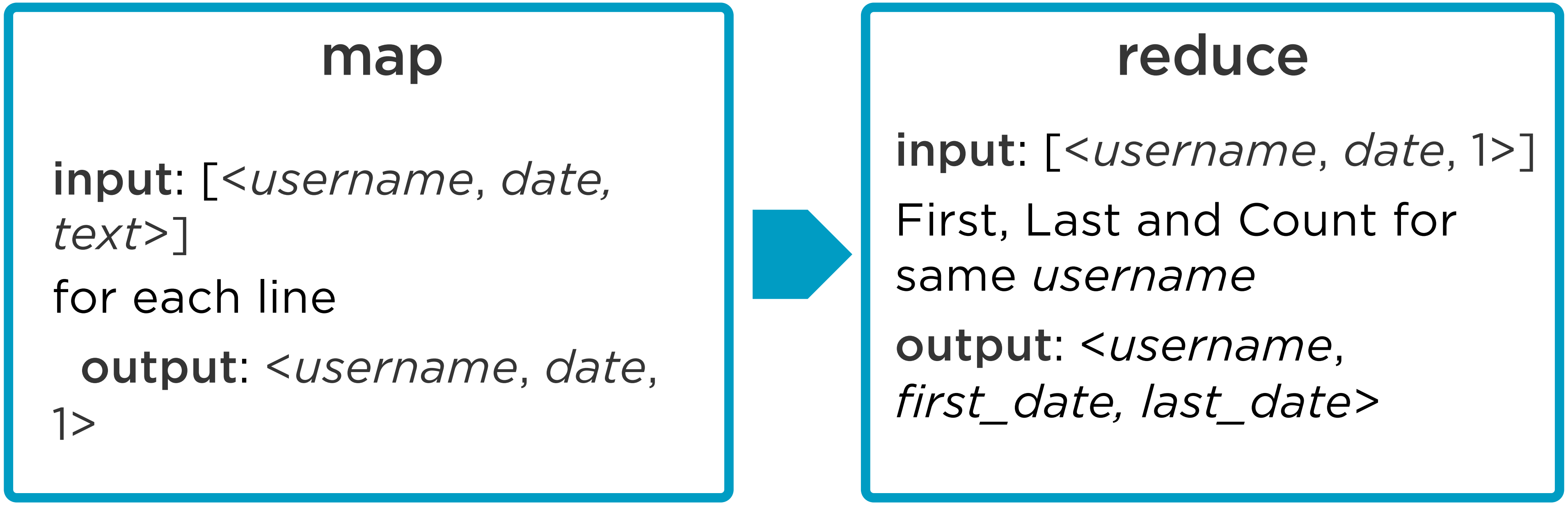
Summarization

Max-Min

Given a list of tweets determine first and last time an user commented and the number of times.

- Data is a set of lines < *username*, *date*, *text* >

```
Peter [07/Mar/2020:16:39:24 -0800] “Stay at home”
John  [07/Mar/2020:16:39:25 -0800] “Me too”
…
```



map

input: [<*username*, *date*, *text*>]

for each line

  output: <*username*, *date*, 1>

reduce

input: [<*username*, *date*, 1>]

First, Last and Count for same *username*

output: <*username*, *first_date, last_date*>

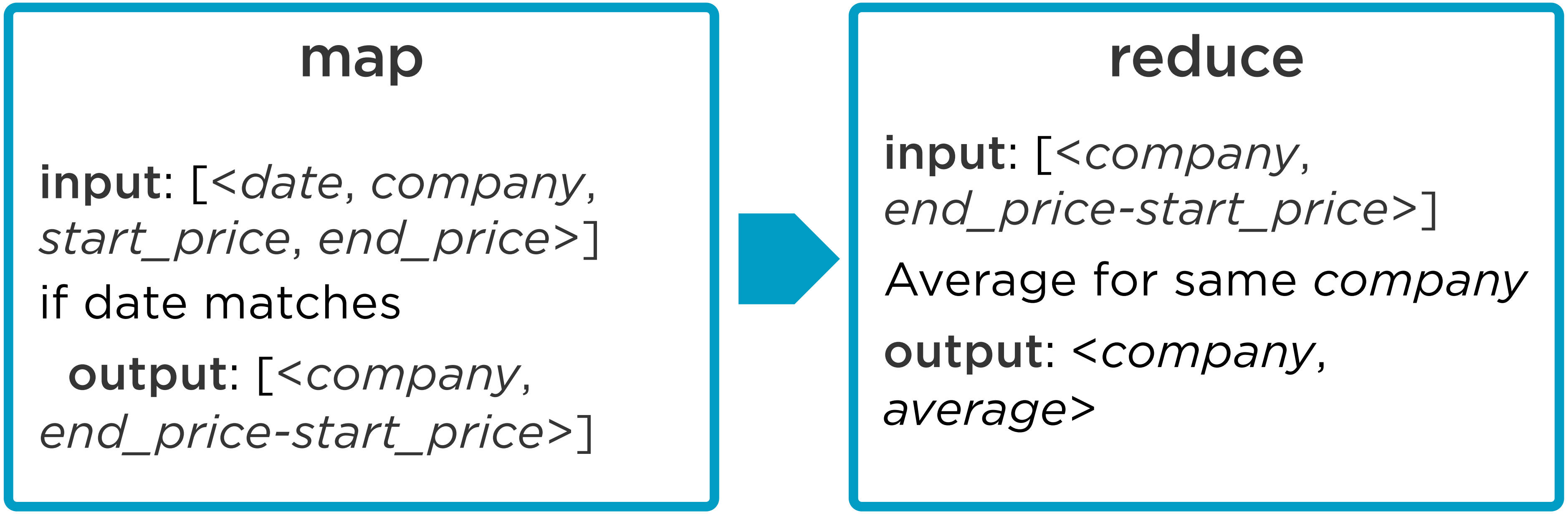
Summarization

Average

Find average daily gains in stock for each company

- Data is a set of lines <*date*, *company*, *start_price*, *end_price*>
- This example is for company from 1/1/2000 – 12/31/2015

```
Date,Company,Open,Close
2009-01-02,Alphabet,153.302917,159.870193,159.621811
…
```



map

input: [<*date*, *company*, *start_price*, *end_price*>]

if date matches

output: [<*company*, *end_price-start_price*>]

reduce

input: [<*company*, *end_price-start_price*>]

Average for same *company*

output: <*company*, *average*>

Inverted Index

Mapping Content to Location

Description

- A general pattern for mapping content to locations such as words or numbers, to its locations in a database file or in a document or a set of documents

Intent

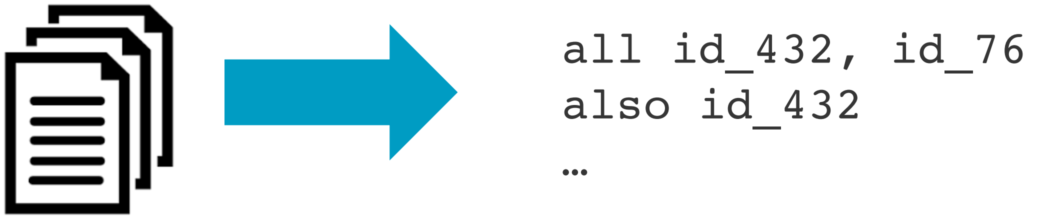
- Most of the text searching systems rely on inverted index to search the documents that contains a given word or a term

Inverted Index

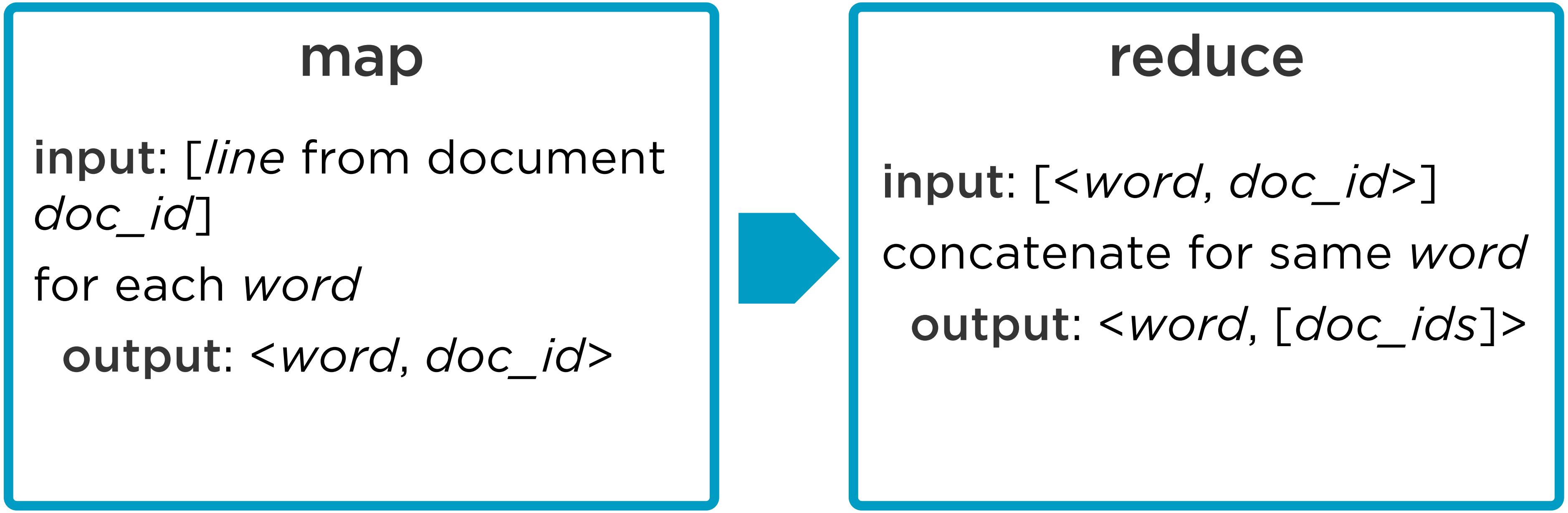
Word to Documents

Find what documents contain a specific word

- **Map**: Parse document and generate <*word*, *doc_id*> pairs
- **Reduce**: For each word, sort the corresponding document IDs



```
all id_432, id_76
also id_432
…
```



map

input: [*line* from document *doc_id*]

for each *word*

  output: <*word*, *doc_id*>

reduce

input: [<*word*, *doc_id*>]

concatenate for same *word*

  output: <*word*, [*doc_ids*]>

Hands-on

Word to Documents – Inverted Index

- ✓ Implement word to documents
- ✓ Adapt `mapper` and `reducer` from wordcount
- ✓ Pre-create a file with the file name as first item in each line

`https://goo.gl/dX1Kn7`

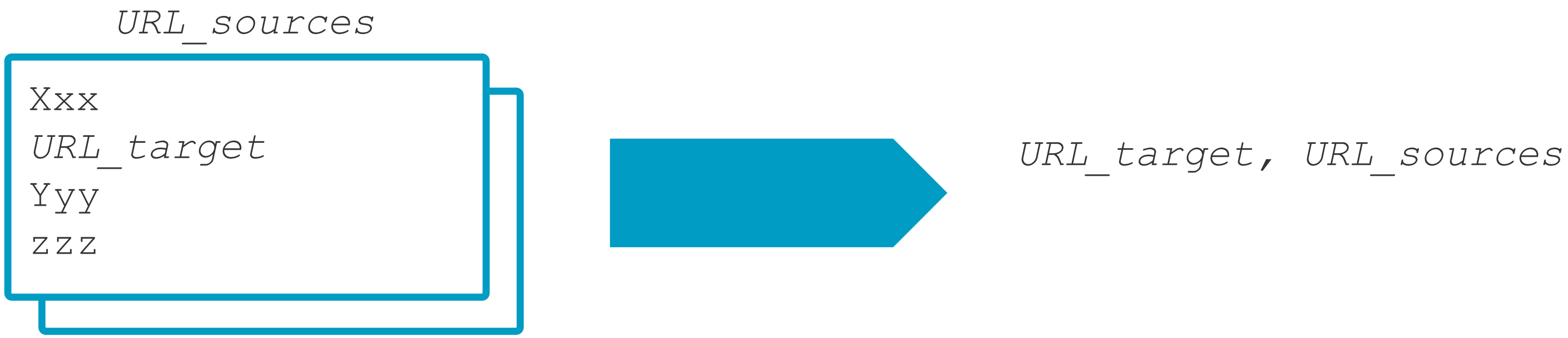
- ✓ Extend to see the number of occurrences per file

Inverted Index

Reverse Web-link Graph

Find where page links come from

- **Map**: Output <target, source> for each link to target in a page source
- **Reduce**: Concatenate the list of all source URLs associated with a target



map

input: [*line* of HTML file *URL_source*]

for each *URL_target*

output: <*URL_target*, *URL_source*>

reduce

input: [<*URL_target*, *URL_source*>]

concatenate for same *URL_target*

output: <*URL_target*, [*URL_sources*]>

Filtering

Filtering Out Records

Description

- It evaluates each record separately and decides, based on some condition, whether it should stay or go

Intent

- Filter out records that are not of interest and keep ones that are.

Examples

1. Closer view of dataset
2. Data cleansing
3. Tracking a thread of events
4. Simple random sampling
5. Distributed Grep
6. Removing low scoring dataset
7. Log Analysis
8. Data Querying and Validation

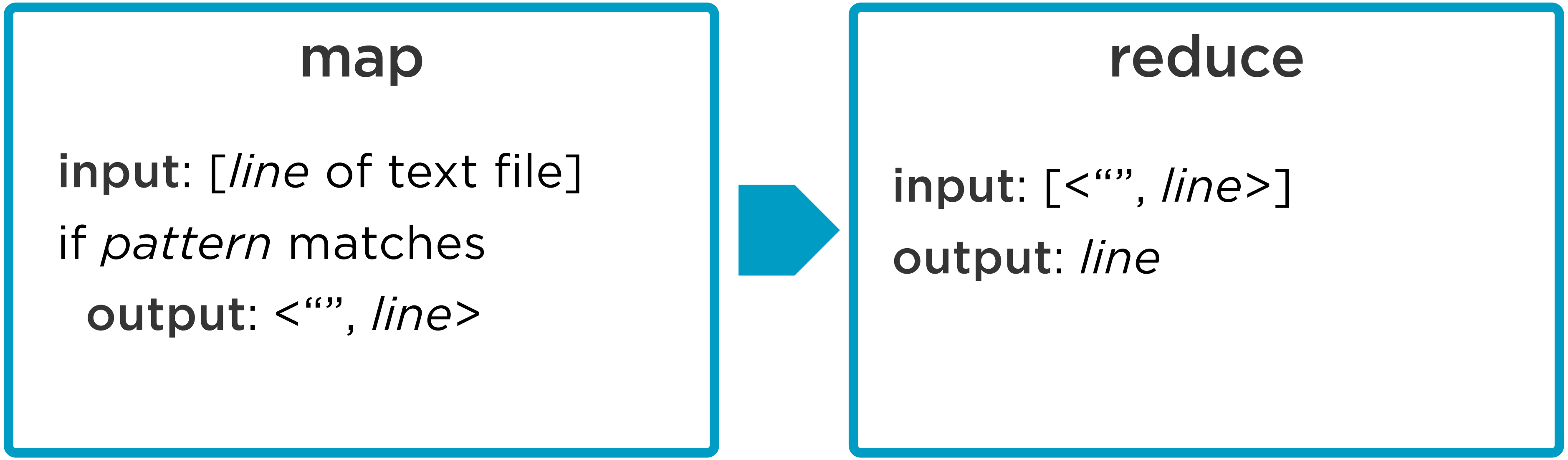
9.…

Filtering

Distributed Grep

Search for words in a document

- **Map**: Generate a line if it matches a given *pattern*
- **Reduce**: Just copy the intermediate data to the output



map

input: [*line* of text file]

if *pattern* matches

  output: <“”, *line*>

reduce

input: [<“”, *line*>]

output: *line*

Building Effective Algorithms and Analytics for Hadoop and Other Systems
MapReduce Design Patterns
O'REILLY
Donald Miner & Adam Shook

Other Patterns

Organization, Join and Input/Output

✓ **Summarization patterns**: Get a top-level view by summarizing and grouping data

✓ **Filtering patterns**: View data subsets such as records generated from one user

✓ **Data organization patterns**: Reorganize data to work with other systems, or to make MapReduce analysis easier

✓ **Join patterns**: Analyze different datasets together to discover interesting relationships

✓ **Metapatterns**: Piece together several patterns to solve multi-stage problems, or to perform several analytics in the same job

✓ **Input and output patterns**: Customize the way you use Hadoop to load or store data

Next Steps

- Get ready for next **lecture:**

  C2. Dataflow Processing (Tuesday 3/31)

Questions

MapReduce Design Patterns

http://piazza.com/harvard/spring2020/cs205/home