Intro, MapReduce & Spark

CS547 Machine Learning for Big Data

Tim Althoff

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Data contains value and knowledge

Data Mining

- **But to extract the knowledge data needs to be**
  - **Stored (systems)**
  - **Managed (databases)**
  - **And ANALYZED ← this class**

Data Mining ≈ Big Data ≈
Predictive Analytics ≈
Data Science ≈ Machine Learning

What This Course Is About

- ***Data mining*** **= extraction of actionable information from (usually) very large datasets, is the subject of extreme hype, fear, and interest**

- It's not all about machine learning
- But some of it is
- **New updated course!**

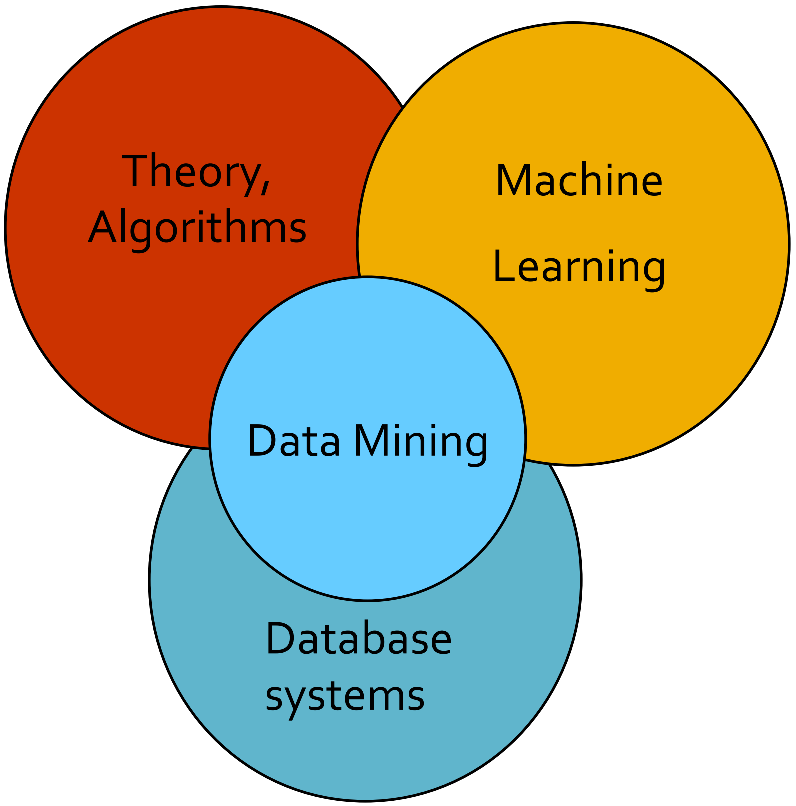
- Emphasis in CS547 on algorithms that **scale**
  - Parallelization often essential

Data Mining Methods

- **Descriptive methods**
  - Find human-interpretable patterns that describe the data
    - **Example:** Clustering
- **Predictive methods**
  - Use some variables to predict unknown or future values of other variables
    - **Example:** Recommender systems

This Class: CS547

- **This combines best of machine learning, statistics, artificial intelligence, databases but emphasis on**
  - **Scalability** (big data)
  - **Algorithms**
  - **Computing architectures**
  - Automation for handling **large data**



What will we learn?

- **We will learn to mine different types of data:**
  - Data is high dimensional
  - Data is a graph
  - Data is infinite/never-ending
  - Data is labeled
- **We will learn to use different models of computation:**
  - MapReduce
  - Streams and online algorithms
  - Single machine in-memory

What will we learn?

- **We will learn to solve real-world problems:**
  - Recommender systems
  - Market Basket Analysis
  - Spam detection
  - Duplicate document detection
- **We will learn various "tools":**
  - Linear algebra (SVD, Rec. Sys., Communities)
  - Optimization (stochastic gradient descent)
  - Dynamic programming (frequent itemsets)
  - Hashing (LSH, Bloom filters)

How the Class Fits Together

High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	PageRank, SimRank	Sampling data streams	SVM	Recommen der systems
Clustering	Network Analysis	Filtering data streams	Decision Trees	Association Rules
Dimensional ity reduction	Spam Detection	Queries on streams	Perceptron, kNN	Duplicate document detection

How do you want that data?

Course Logistics

Course Staff

Teaching Assistants

Alex Okeson | Alon Milchgrub | Jessica Perry | Mathew Luo | Nicasia Beebe-Wang | Swati Padmanabhan

CS547 Course Staff

- **Office hours:**
  - See course website www.cs.washington.edu/cse547 for TA office hours
    - **We start Office Hours next week (April 8)**
  - **Tim:** Tuesdays 11:30-12:30am, GC 313
  - **TA office hours:** see website and calendar

Resources

- **Course website: www.cs.washington.edu/cse547**
  - Lecture slides (at least 30min before the lecture)
  - Homeworks, readings
- **Class textbook: Mining of Massive Datasets** by A. Rajaraman, J. Ullman, and J. Leskovec
  - **Sold by Cambridge Uni. Press but available for free at http://mmds.org**
  - Course based on textbook and Stanford CS246 course by Jure Leskovec

Logistics: Communication

- **Piazza Q&A website:**
  - https://piazza.com/washington/spring2019/cs547 Use Piazza for **all questions** and public communication & announcements
    - Search the forum before asking a question
    - Please tag your posts and please no one-liners
    - *Participation in Piazza Careers is not required, can opt out any time*
    - ***Please sign up for Piazza through Canvas (access code on Canvas)***
- **For emergencies & personal matters, email course staff always at:**
  - *cse547-instructors@cs.washington.edu*
- **We will post course announcements to Piazza (make sure you check it regularly)**

Auditors are welcome to sit-in & audit the class

Special Tutorials

- **Spark tutorial and help session:**
  - Thursday, April 4, 2:30-4:20 PM, GWN 201
- **Review of basic probability and proof techniques**
  - Tuesday, April 9, 3:30-5:20 PM, PAA A102
- **Review of linear algebra:**
  - Thursday, April 11, 3:30-5:20 PM, SIG 134

Work for the Course: Homeworks

- **4 longer homeworks: 50%**
  - Four major assignments, involving programming, proofs, algorithm development.
  - "Warmup" assignment, called "HW0," to introduce everyone to Spark has just been posted
  - Assignments take lots of time (+20h). **Start early!!**
- **How to submit?**
  - **Homework write-up:**
    - Submit via Gradescope
    - Course code: **97EWEW**
  - **Everyone uploads code:**
    - Put all the code for 1 question into 1 file and submit via Gradescope

Homework Calendar

- **Homework schedule:**

Date (23:59 PT)	Out	In
Today	HW0, *Course Proj.*	
04/04, Thu	HW1	
04/18, Thu	HW2	HW0, HW1
04/25, Thu		*Project Proposal*
05/02, Thu	HW3	HW2
05/09, Thu		*Project Milestone*
05/16, Thu	HW4	HW3
06/01, Sat		HW4
06/09, Sun		*Project Report*
06/10, Mon		*Poster Presentation*

- **Two late periods for HWs for the quarter:**
  - **Late period expires 48 hours after the original deadline**
  - **Can use max 1 late period per HW (not for Final Report/Poster)**

Work for the Course: Course Project

- **Course Project: 50%**
  - Project proposal (10%)
  - Project milestone report (10%)
  - Final project report (25%)
  - Poster Presentation (5%)
  - *More details on course website*

- Teams of (up to) **three students each**
  - Start planning now
  - Find students in class or through Piazza
  - Find dataset to work on – also see course website

Work for the Course: Course Project

- **Poster Presentation**
  - **Monday, June 10,** 10:00am-1:00pm
  - Location: Allen Center Atrium
  - **You have to be present**
- **Extra credit:** Up to 2% of your grade
  - For participating in Piazza discussions
    - Especially valuable are answers to questions posed by other students
  - Reporting bugs in course materials
  - See course website for details

Prerequisites

- Programming: Python
- Basic Algorithms: e.g., CS332/CS373 or CS417/CS421
- Probability: any introductory course
  - There will be a review session and a review doc is linked from the class home page
- Linear algebra: (e.g., Math 308 or equivalent)
  - Another review doc + review session is available
- Rigorous proofs & Multivariable calculus (e.g., CS311 or equivalent)
- Database systems (SQL, relational algebra)

What If I Don't Know All This Stuff?

- **Each of the topics listed is important for a small part of the course:**
  - If you are missing an item of background, you could consider just-in-time learning of the needed material
- **The exception is programming:**
  - To do well in this course, you really need to be comfortable with writing code in Python

Collaboration Policy & Academic Integrity

- **We'll follow the standard CS Dept. approach:** You can get help, but you *MUST* acknowledge the help on the work you hand in
  - www.cs.washington.edu/academics/misconduct
- Failure to acknowledge your sources is a *violation of academic integrity*
- We use plagiarism tools to check the originality of your code

Collaboration Policy & Academic Integrity

- **You can talk to others about the algorithm(s) to be used to solve a homework problem;**
  - As long as you then mention their name(s) on the work you submit
- **You should not use code of others or be looking at code of others when you write your own:**
  - You can talk to people but have to write your own solution/code
  - If you fail to mention your sources, plagiarism tools will catch you, and you will be charged with a academic integrity violation

Final Thoughts

- **CS547 is fast paced!**
  - Requires programming maturity
  - Strong math skills
    - Some students tend to be rusty on math/theory
- **Course time commitment:**
  - Homeworks take +20h
  - Significant course project
- Form study groups
- Form project groups

- **It's going to be <u>fun</u> and <u>hard</u> work. ☺**

4 To-do items

- **4 to-do items for you:**
  - **Register to Piazza through Canvas**
  - **Register to Gradescope**
  - **Complete HW0**
    - HW0 should take you about 1-2 hours to complete (Note this is a "toy" homework to get you started. Real homeworks will be much more challenging and longer)
  - **Start planning course project (topic, team, dataset)**

- **Additional details/instructions at http://www.cs.washington.edu/cse547**

Distributed Computing for Data Mining

Large-scale Computing

- **Large-scale computing** for **data mining** problems on **commodity hardware**
- **Challenges:**
  - **How do you distribute computation?**
  - **How can we make it easy to write distributed programs?**
  - **Machines fail:**
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to lose 1/day
    - With 1M machines 1,000 machines fail every day!

An Idea and a Solution

- **Issue:**
  Copying data over a network takes time
- **Idea:**
  - Bring computation to data
  - Store files multiple times for reliability
- **Spark/Hadoop** address these problems
  - **Storage Infrastructure – File system**
    - Google: GFS. Hadoop: HDFS
  - **Programming model**
    - MapReduce
    - Spark

Storage Infrastructure

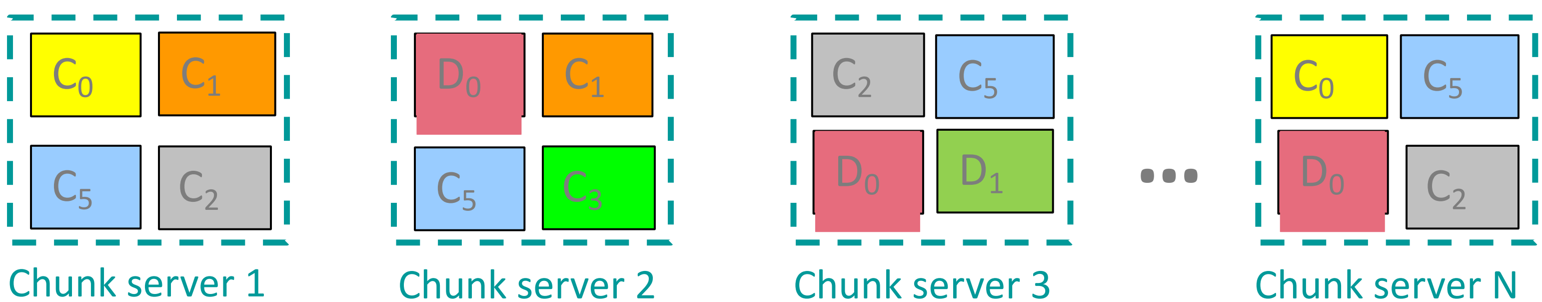
- **Problem:**
  - If nodes fail, how to store data persistently?
- **Answer:**
  - **Distributed File System**
    - Provides global file namespace
- **Typical usage pattern:**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

Distributed File System

- **Chunk servers**
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks
- **Master node**
  - a.k.a. Name Node in Hadoop's HDFS
  - Stores metadata about where files are stored
  - Might be replicated
- **Client library for file access**
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data

Distributed File System

- **Reliable distributed file system**
- Data kept in "chunks" spread across machines
- Each chunk **replicated** on different machines
  - Seamless recovery from disk or machine failure



Bring computation directly to the data!

Chunk servers also serve as compute servers

Programming Model

- **MapReduce is a style of programming designed for:**
  1. Easy parallel programming
  2. Invisible management of hardware and software failures
  3. Easy management of very-large-scale data

- It has several implementations, including Hadoop, Spark (used in this class), Flink, and the original Google implementation just called "MapReduce"

MapReduce: Overview

3 steps of MapReduce

- **Map:**
  - Apply a user-written *Map function* to each input element
    - *Mapper* applies the Map function to a single element
      - Many mappers grouped in a *Map task* (the unit of parallelism)
  - The output of the Map function is a set of 0, 1, or more *key-value pairs*.
- **Group by key:** Sort and shuffle
  - System sorts all the key-value pairs by key, and outputs key-(list of values) pairs
- **Reduce:**
  - User-written *Reduce function* is applied to each key-(list of values)

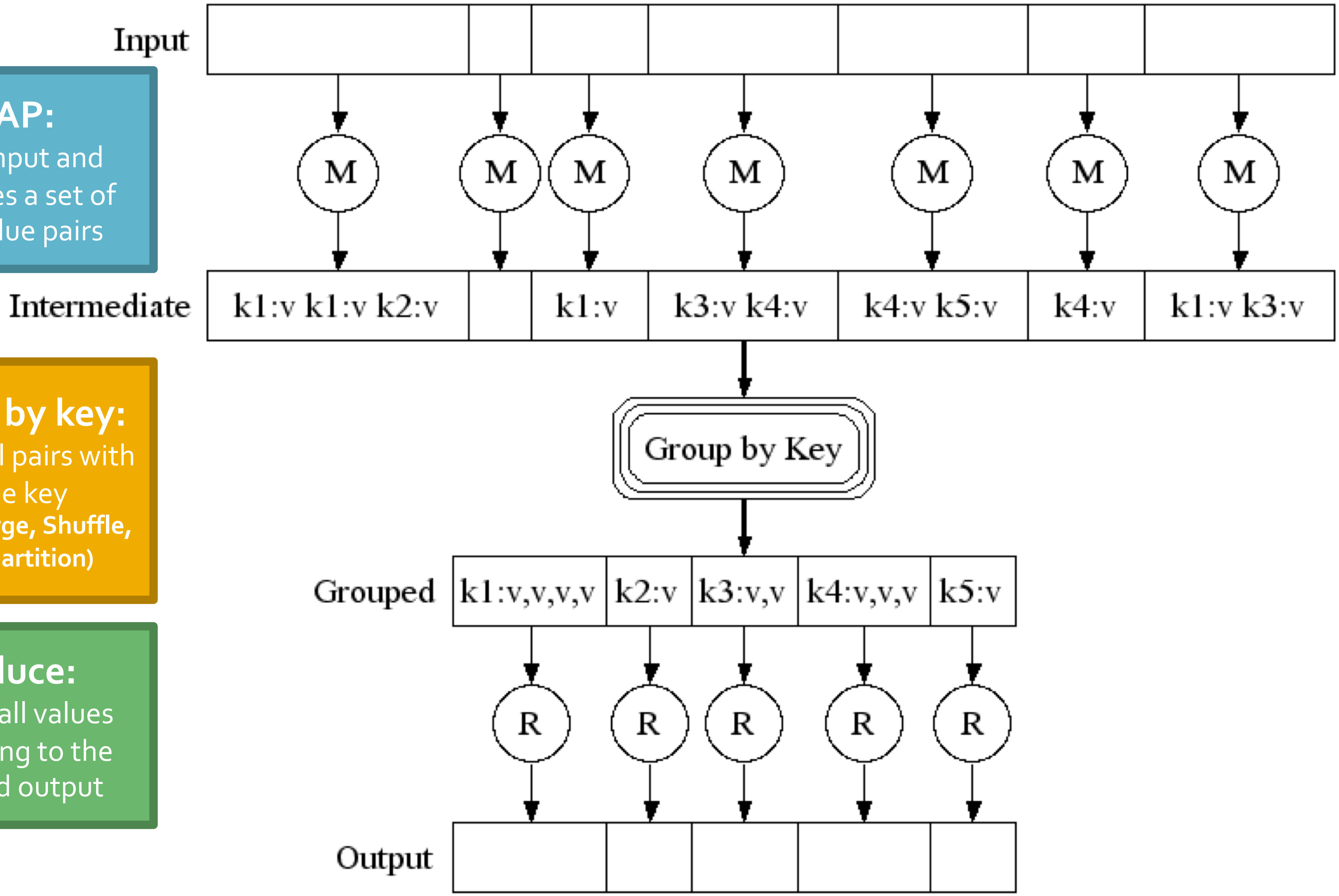
Outline stays the same, **Map** and **Reduce** change to fit the problem

Map-Reduce: A diagram

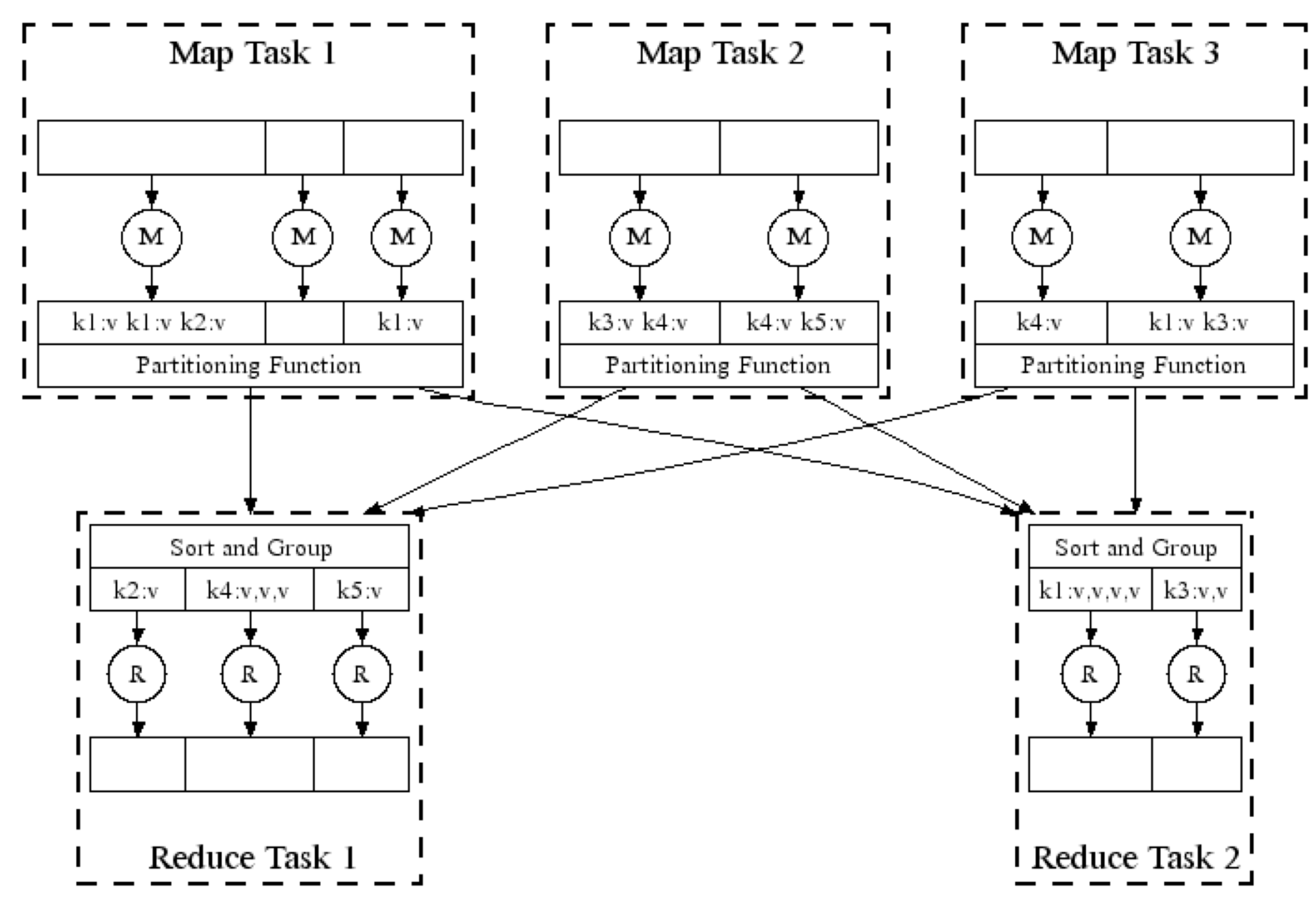
MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output

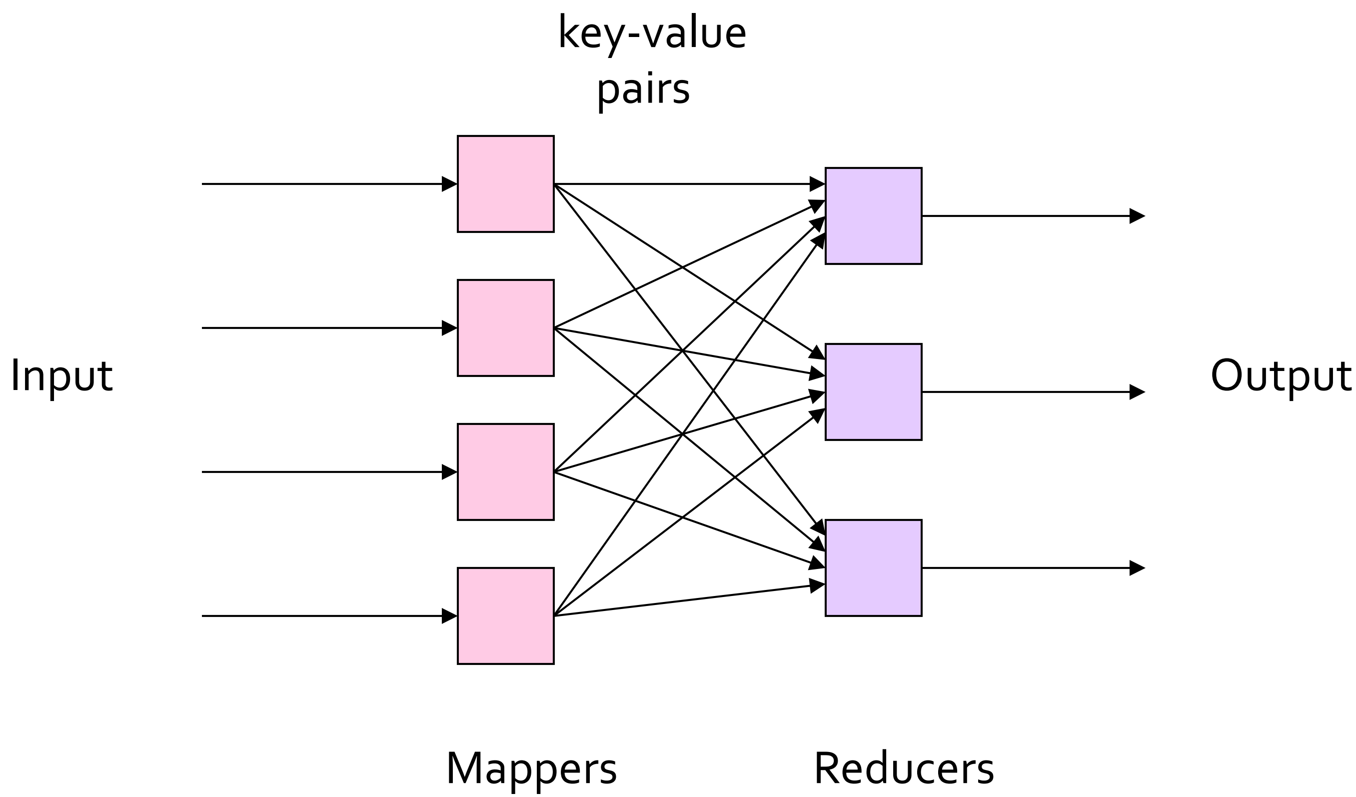


Map-Reduce: In Parallel



All phases are distributed with many tasks doing the work

MapReduce Pattern



Example: Word Counting

Example MapReduce task:

- We have a huge text document
- Count the number of times each distinct word appears in the file

- **Many applications of this:**
  - Analyze web server logs to find popular URLs
  - Statistical machine translation:
    - Need to count number of times every 5-word sequence occurs in a large corpus of documents

MapReduce: Word Counting

Provided by the programmer

MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key

Provided by the programmer

Reduce:
Collect all values belonging to the key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/mache partnership. '"The work we're doing now -- the robotics we're doing -- is what we're going to need ……………………..

Big document

(The, 1)
(crew, 1)
(of, 1)
(the, 1)
(space, 1)
(shuttle, 1)
(Endeavor, 1)
(recently, 1)
....

(key, value)

(crew, 1)
(crew, 1)
(space, 1)
(the, 1)
(the, 1)
(the, 1)
(shuttle, 1)
(recently, 1)
…

(key, value)

(crew, 2)
(space, 1)
(the, 3)
(shuttle, 1)
(recently, 1)
…

(key, value)

Only sequential reads

Word Count Using MapReduce

```
map(key, value):
# key: document name; value: text of the document
  for each word w in value:
     emit(w, 1)
```

```
reduce(key, values):
# key: a word; value: an iterator over counts
      result = 0
      for each count v in values:
            result += v
      emit(key, result)
```

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 is the bottleneck
- Handling machine **failures**
- Managing required inter-machine **communication**

Dealing with Failures

- **Map worker failure**
  - Map tasks completed or in-progress at worker are reset to idle and rescheduled
  - Reduce workers are notified when map task is rescheduled on another worker
- **Reduce worker failure**
  - Only in-progress tasks are reset to idle and the reduce task is restarted

Spark

Problems with MapReduce

- **Two major limitations of MapReduce:**
  - Difficulty of programming directly in MR
    - Many problems aren't easily described as map-reduce
  - Performance bottlenecks, or batch not fitting the use cases
    - Persistence to disk typically slower than in-memory work
- **In short, MR doesn't compose well for large applications**
  - Many times one needs to chain multiple map-reduce steps

Data-Flow Systems

- **MapReduce uses two "ranks" of tasks:** One for Map the second for Reduce
  - Data flows from the first rank to the second
- **Data-Flow Systems generalize this in two ways:**
  1. Allow any number of tasks/ranks
  2. Allow functions other than Map and Reduce
  - As long as data flow is in one direction only, we can have the blocking property and allow recovery of tasks rather than whole jobs

Spark: Most Popular Data-Flow System

- **Expressive computing system, not limited to the map-reduce model**
- **Additions to MapReduce model:**
  - Fast data sharing
    - Avoids saving intermediate results to disk
    - Caches data for repetitive queries (e.g. for machine learning)
  - General execution graphs (DAGs)
  - Richer functions than just map and reduce
- Compatible with Hadoop

Spark: Overview

- Open source software (Apache Foundation)
- Supports **Java, Scala and Python**

- **Key construct/idea:** Resilient Distributed Dataset (RDD)

- **Higher-level APIs**: DataFrames & DataSets
  - Introduced in more recent versions of Spark
  - Different APIs for aggregate data, which allowed to introduce SQL support

Spark: RDD

Key concept ***Resilient Distributed Dataset*** **(RDD)**

- Partitioned collection of records
  - Generalizes (key-value) pairs
- Spread across the cluster, Read-only
- Caching dataset in memory
  - Different storage levels available
  - Fallback to disk possible

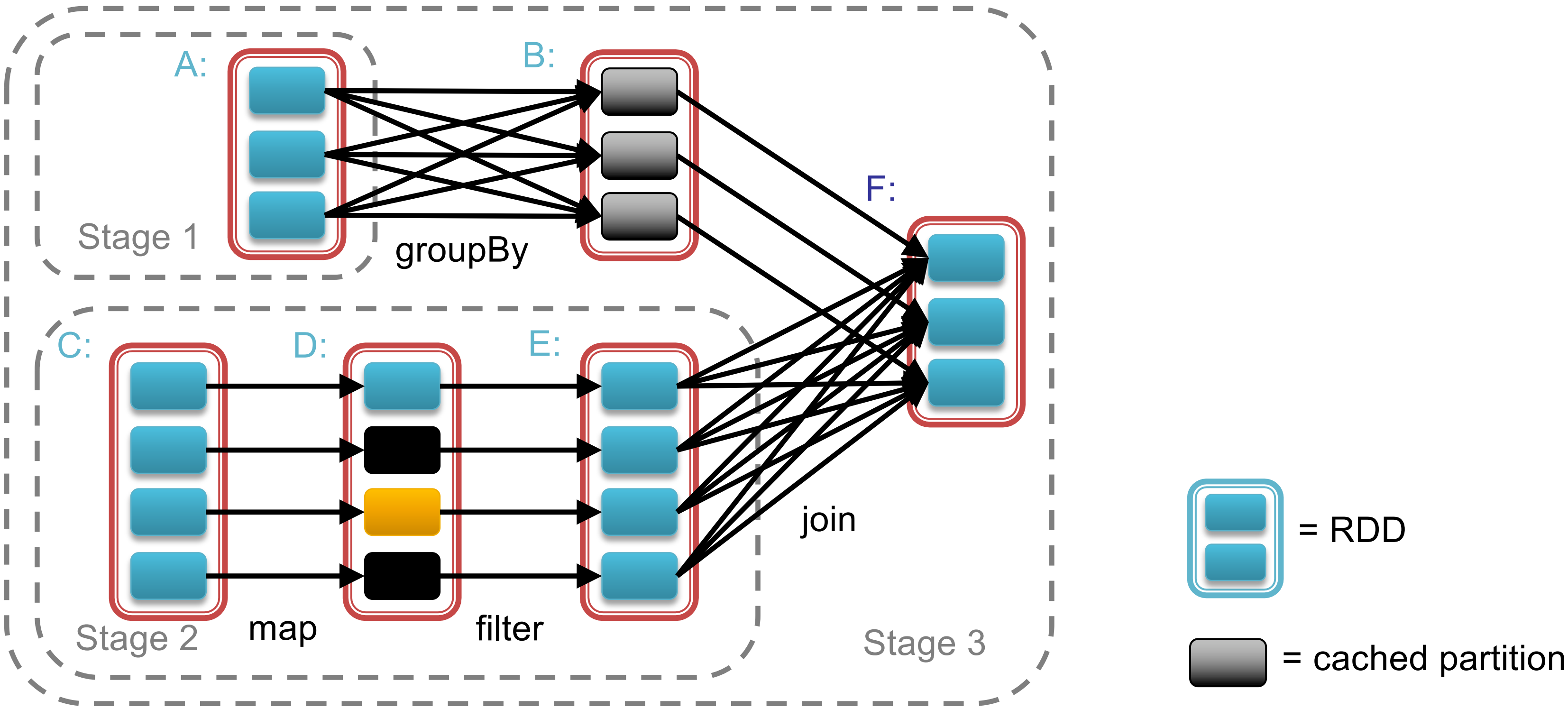
- RDDs can be created from Hadoop, or by transforming other RDDs (you can stack RDDs)

- RDDs are best suited for applications that apply the same operation to all elements of a dataset

Spark RDD Operations

- **Transformations** build RDDs through deterministic operations on other RDDs:
  - Transformations include *map*, *filter*, *join, union, intersection, distinct*
  - **Lazy evaluation:** Nothing computed until an action requires it
- **Actions** to return value or export data
  - Actions include *count, collect, reduce, save*
  - Actions can be applied to RDDs; actions force calculations and return values

Task Scheduler: General DAGs



- Supports general task graphs
- Pipelines functions where possible
- Cache-aware data reuse & locality
- Partitioning-aware to avoid shuffles

DataFrame & Dataset

- DataFrame:
  - Unlike an RDD, data organized into named columns, e.g. a **table in a relational database**.
  - Imposes a structure onto a distributed collection of data, allowing higher-level abstraction
- Dataset:
  - Extension of DataFrame API which provides **type-safe, object-oriented programming interface** (compile-time error detection)

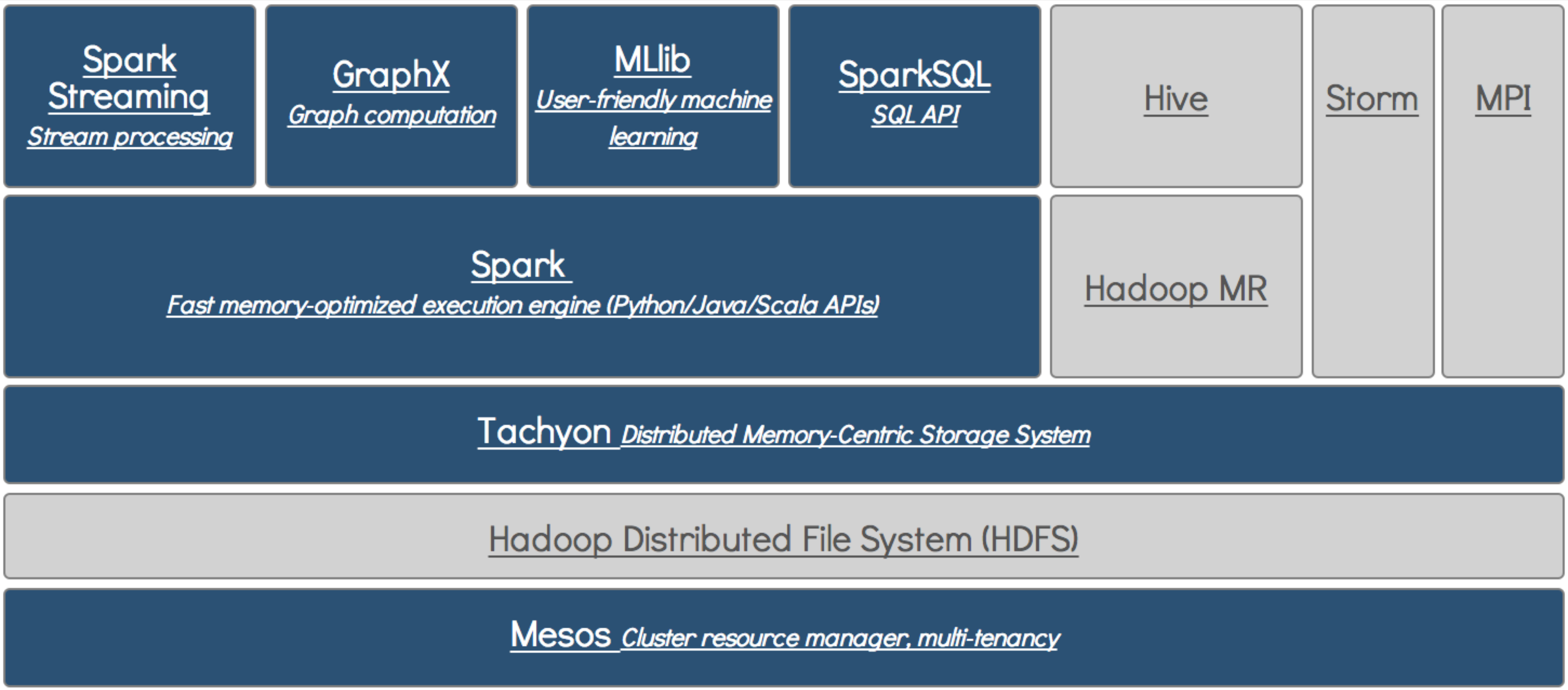
Both built on Spark SQL engine. Both can be converted back to an RDD

Useful Libraries for Spark

- Spark SQL

- Spark Streaming – **stream processing of live datastreams**
- MLlib – **scalable machine learning**
- GraphX – **graph manipulation**
  - extends Spark RDD with Graph abstraction: a directed multigraph with properties attached to each vertex and edge

Data Analytics Software Stack



Spark vs. Hadoop MapReduce

- Performance: Spark normally faster but with caveats
  - Spark can process data in-memory; Hadoop MapReduce persists back to the disk after a map or reduce action
  - Spark generally outperforms MapReduce, but it **often needs lots of memory to perform well**; if there are other resource-demanding services or can't fit in memory, Spark degrades
  - MapReduce easily runs alongside other services with minor performance differences, & works well with the 1-pass jobs it was designed for
- Ease of use: **Spark is easier to program** (higher-level APIs)
- Data processing: **Spark more general**

Problems Suited for MapReduce

Example: Host size

- **Suppose we have a large web corpus**
- Look at the metadata file
  - Lines of the form: (URL, size, date, …)
- **For each host, find the total number of bytes**
  - That is, the sum of the page sizes for all URLs from that particular host

- **Other examples:**
  - Link analysis and graph processing
  - Machine Learning algorithms

Example: Language Model

- **Statistical machine translation:**
  - Need to count number of times every 5-word sequence occurs in a large corpus of documents
- **Very easy with MapReduce:**
  - **Map:**
    - Extract (5-word sequence, count) from document
  - **Reduce:**
    - Combine the counts

Example: Join By Map-Reduce

- **Compute the natural join *R(A,B)* ⋈ *S(B,C)***
- *R* and *S* are each stored in files
- Tuples are pairs *(a,b)* or *(b,c)*

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

⋈

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

=

A	C
a_3	c_1
a_3	c_2
a_4	c_3

Map-Reduce Join

- **Use a hash function *h* from B-values to *1...k***
- **A Map process turns:**
  - Each input tuple *R(a,b)* into key-value pair *(b,(a,R))*
  - Each input tuple *S(b,c)* into *(b,(c,S))*
- **Map processes** send each key-value pair with key *b* to Reduce process *h(b)*
  - Hadoop does this automatically; just tell it what *k* is.
- Each **Reduce process** matches all the pairs *(b,(a,R))* with all *(b,(c,S))* and outputs *(a,b,c)*.

Problems NOT suitable for MapReduce

- **MapReduce is great for:**
  - Problems that require sequential data access
  - Large batch jobs (**not** interactive, real-time)
- **MapReduce is inefficient for problems where random (or irregular) access to data required:**
  - **Graphs**
  - Interdependent data
    - Machine learning
    - Comparisons of many pairs of items

Cost Measures for Algorithms

- **In MapReduce we quantify the cost of an algorithm using**

1. *Communication cost* = total I/O of all processes
2. *Elapsed communication cost* = max of I/O along any path
3. (*Elapsed*) *computation cost* analogous, but count only running time of processes

Note that here the big-O notation is not the most useful (adding more machines is always an option)

Example: Cost Measures

- **For a map-reduce algorithm:**
  - **Communication cost =** input file size + 2 × (sum of the sizes of all files passed from Map processes to Reduce processes) + the sum of the output sizes of the Reduce processes.
  - **Elapsed communication cost** is the sum of the largest input + output for any map process, plus the same for any reduce process

What Cost Measures Mean

- Either the I/O (communication) or processing (computation) cost dominates
  - Ignore one or the other
- Total cost tells what you pay in rent from your friendly neighborhood cloud
- Elapsed cost is wall-clock time using parallelism

Cost of Map-Reduce Join

- **Total communication cost** $= O(|R|+|S|+|R \bowtie S|)$
- **Elapsed communication cost** $= O(s)$
  - We're going to pick ***k*** and the number of Map processes so that the I/O limit ***s*** is respected
  - We put a limit ***s*** on the amount of input or output that any one process can have. ***s* could be:**
    - What fits in main memory
    - What fits on local disk
- With proper indexes, computation cost is linear in the input + output size
  - So computation cost is like comm. cost

Waitlisted?
Please fill out form to let us know you're here and still interested.
We will distribute add codes for any slot that opens up.

Grab a handout at the back of the room