JOHNS HOPKINS
CAREY BUSINESS SCHOOL

Lecture 3

BU.330.740 Large Scale Computing on the Cloud

Minghong Xu, PhD.
Associate Professor

Reflections on last week

- MapReduce: parallelism framework, divide and conquer
  - Map: the divide step
  - Reduce: the aggregate step
  - **Reason: we need to use multiple machines to speed up**

- Frequent itemset mining problem
  - Association patterns
  - Two products, two movies, two medicines, etc.
  - “co-occurrence”, “co-appearing”

One More Example

- Mine your own business: Market-structure surveillance through text mining (Netzer, Oded, et al. 2012 Marketing Science)
  - Explore online user-generated content and “listen” to what customers write about their and their competitors' products

Figure 8 Terms Commonly Appearing with the Honda Civic, Nissan Sentra, and Toyota Corolla

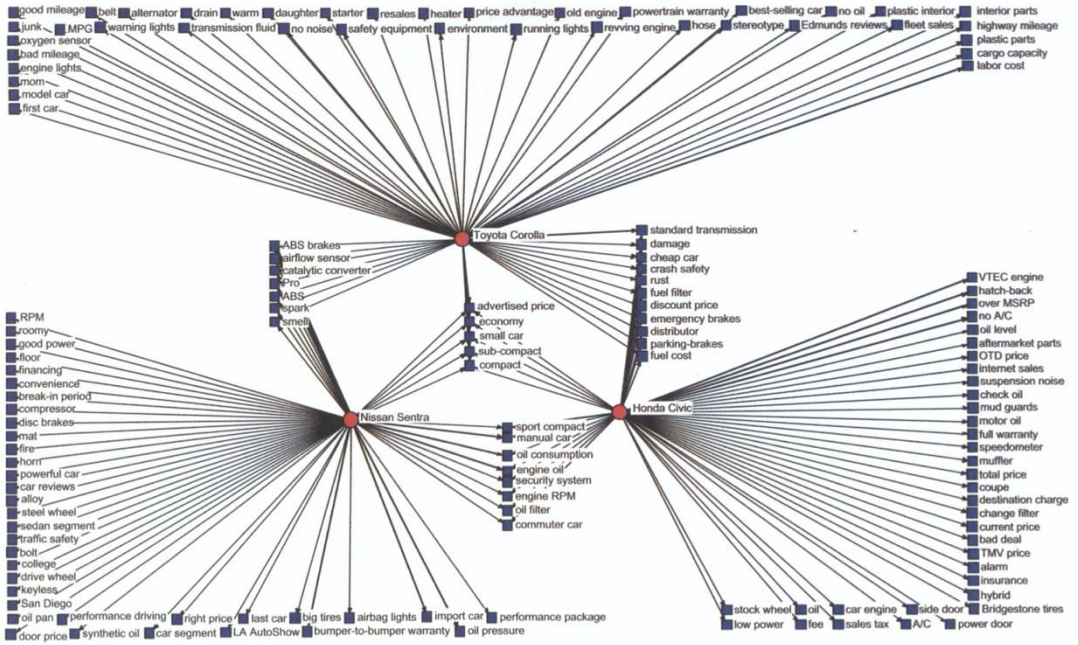
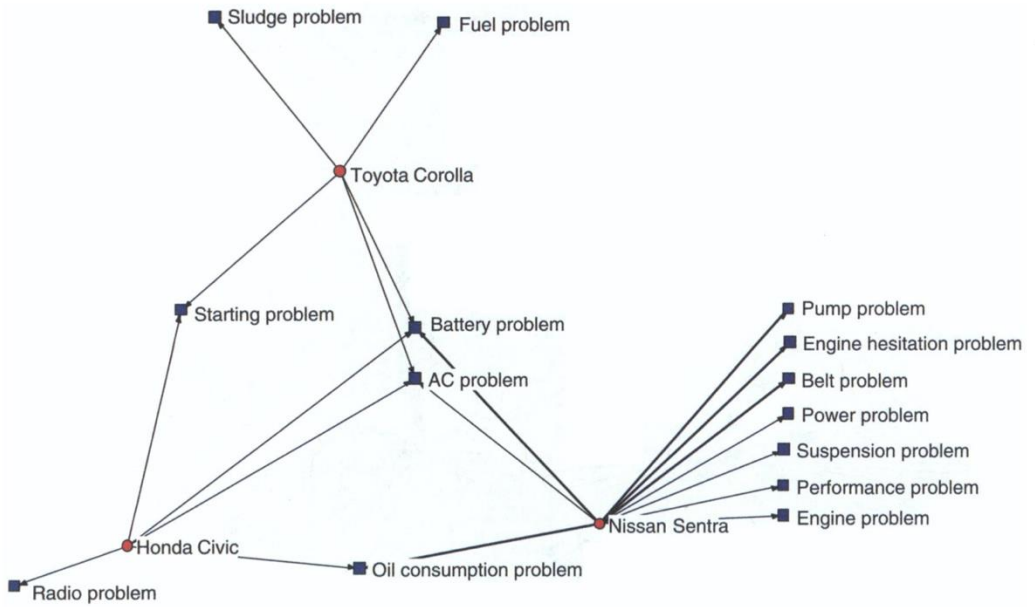
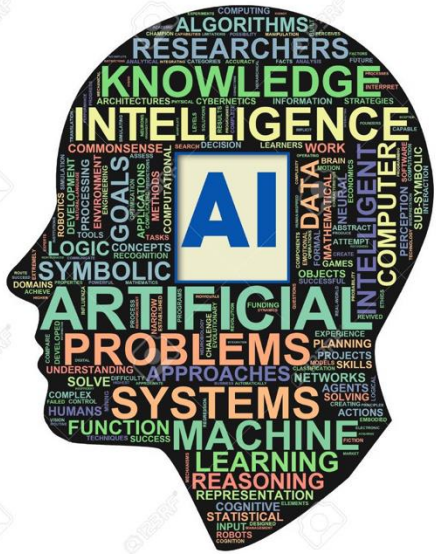


Figure 9 Problems Commonly Appearing with the Honda Civic, Nissan Sentra, and Toyota Corolla



Word Cloud

- Visualize text data, perceive the most prominent terms to determine its relative prominence
  - Bigger term means greater weight
- Frequency of each item, show the top n words/categories



Today's Agenda

- Advanced Big Data Framework: Spark
- Data Engineering and Architectures
- Lab2: PySpark in Google Colab
  - You need a Google account for today's exercises

- Optional materials: Apache Hive
  - Apache Pig will be introduced next week

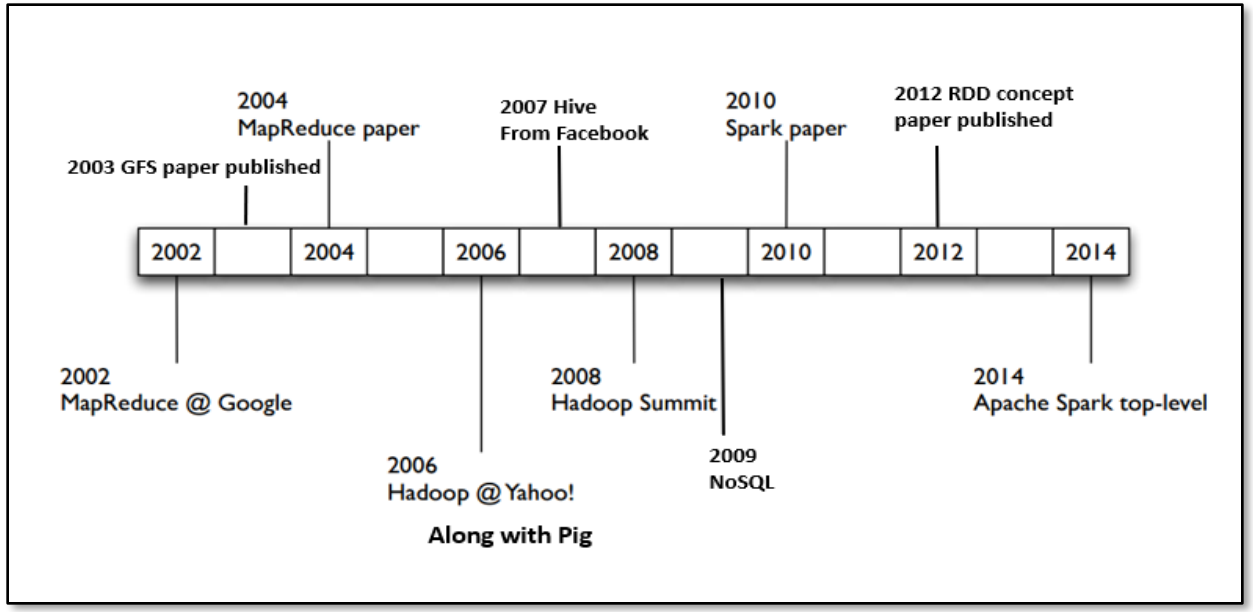
Spark: Improved MapReduce

Limitations about MapReduce

- Limited control over data and execution flow
- All algorithms must be expressed in map reduce format
- You CAN NOT control:
  - When a mapper or reducer begins or finishes
  - Which input a particular mapper is processing
  - Which keys a particular reducer is processing
  - Number of reducers
    - Refer to page 18 in Lecture 2 optional notes on YARN

Spark Introduction

- Cluster computing platform designed to be fast and general purpose
  - Speed: much faster than MapReduce due to in-memory processing
  - Generality: combine SQL queries, ML, streaming…
- Start in 2009 as a class project by Matei Zaharia

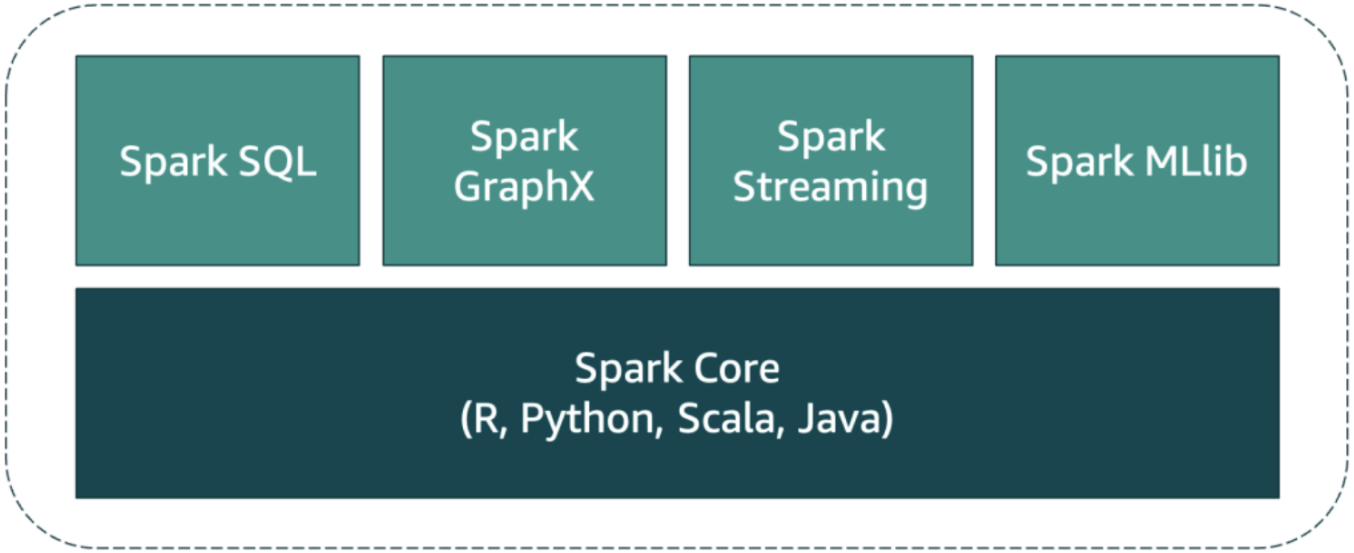


Spark Use Cases

- Big Data ETL
  - Uber: https://youtu.be/AiinFkL-pmw?si=StrwWb6B6fnn8BVc
- Real-time data analytics, such as stock market, IoT sensors
- Machine learning & AI, such as recommendation systems
- Bioinformatics & genome sequencing

- Not to use:
  - When you have only little data
  - When you have only simple operations

Spark Components



- Spark Core
  - Basic functionality, task scheduling, memory management, faulty recovery, interaction with storage systems

Components (Cont.)

- Spark SQL
  - Structured data processing with DataFrames
- Spark Streaming
  - Process live streams of data, e.g., logfiles, queues of messages
- Spark MLlib
  - Machine learning library
- Spark GraphX
  - Graph mining

Writing and Deploying Spark Applications

- Work in multiple ways, AWS, Google Colab…
- Can also work locally in Python IDLE or Anaconda
  - Install via command: “pip install pyspark” or “conda install pyspark”
  - https://docs.anaconda.com/anaconda-scale/spark/
  - https://gist.github.com/dvainrub/b6178dc0e976e56abe9caa9b72f73d4a
  - Note that Windows system may encounter issues
- We will use the simplest (and free) way: Google Colab

SparkContext

- SparkContext: entry point to Spark functionality
  - First thing created when a Spark application starts
  - Represent connection to a Spark cluster
- Named **sc** by convention
- Introduced in Spark 1.x
- Limitations:
  - No support for DataFrame
  - No support for SQL

SparkSession

- Introduced in Spark 2.0, a unified entry point
- Support
  - DataFrame
  - SQL
  - Hive
- Built-in optimization via Catalyst optimizer → SparkSession

Resilient Distributed Dataset (RDD)

- **R**esilient: if data in memory is lost, it can be recreated
- **D**istributed: processed across the cluster
- **D**ataset: initial data can come from a file or created programmatically

- Legacy, fundamental unit of data in Spark
- A collection of items distributed across many computer nodes, and can be manipulated in parallel

RDD

a data structure to store collection of items.

DataFrame and Dataset

- DataFrame: optimized, structured data with Catalyst optimizer
  - Support distributed processing
  - Convert from a pandas DataFrame using createDataFrame()
- Dataset: "~~DataFrame~~" for Scala & Java only

Creating RDDs

- `sc.textFile("myfile.txt")`
- `sc.textFile("mydata/*.log")`
- `sc.textFile("myfile1.txt,myfile2.txt")`

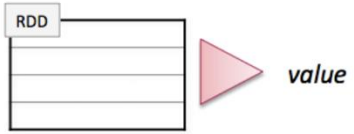
```
> myData = ["Alice","Carlos","Frank","Barbara"]
> myRdd = sc.parallelize(myData)
> myRdd.take(2)
['Alice', 'Carlos']
```

Two types of RDD Operations

- Actions
  - Return values

Some common actions

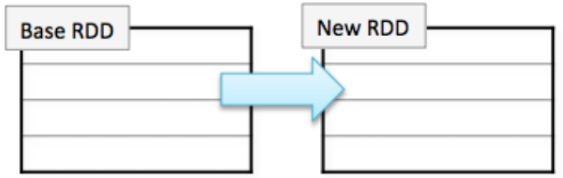
- `count()` – return the number of elements
- `take(n)` – return an array of the first *n* elements
- `collect()` – return an array of all elements
- `saveAsTextFile(file)` – save to text file(s)



- Transformations
  - Define new RDD based on the current ones

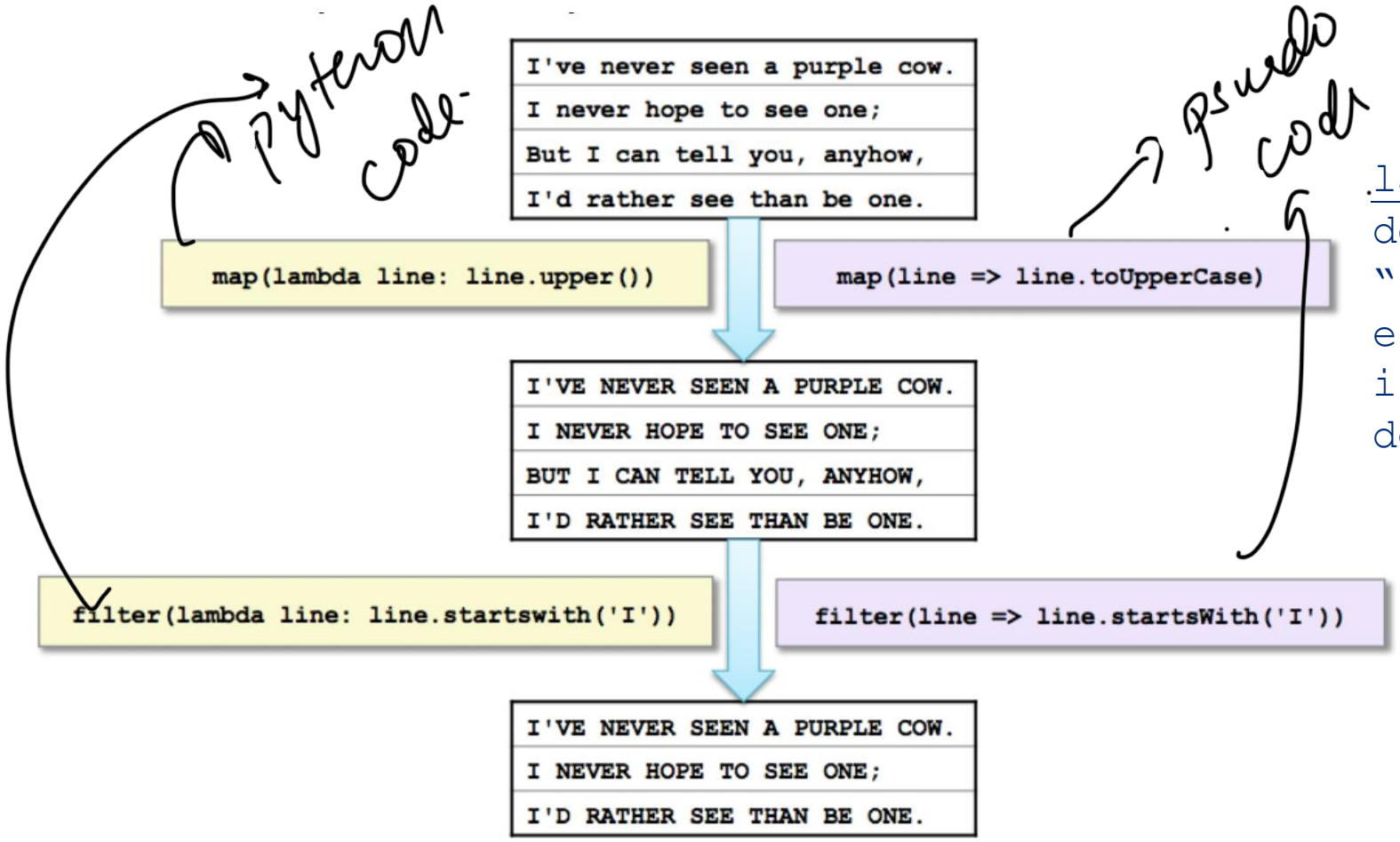
Some common transformations

- `map(function)` – creates a new RDD by performing a function on each record in the base RDD
- `filter(function)` – creates a new RDD by including or excluding each record in the base RDD according to a boolean function



- Lazy execution: Data in RDDs is not processed until an action is performed

Example: map and filter

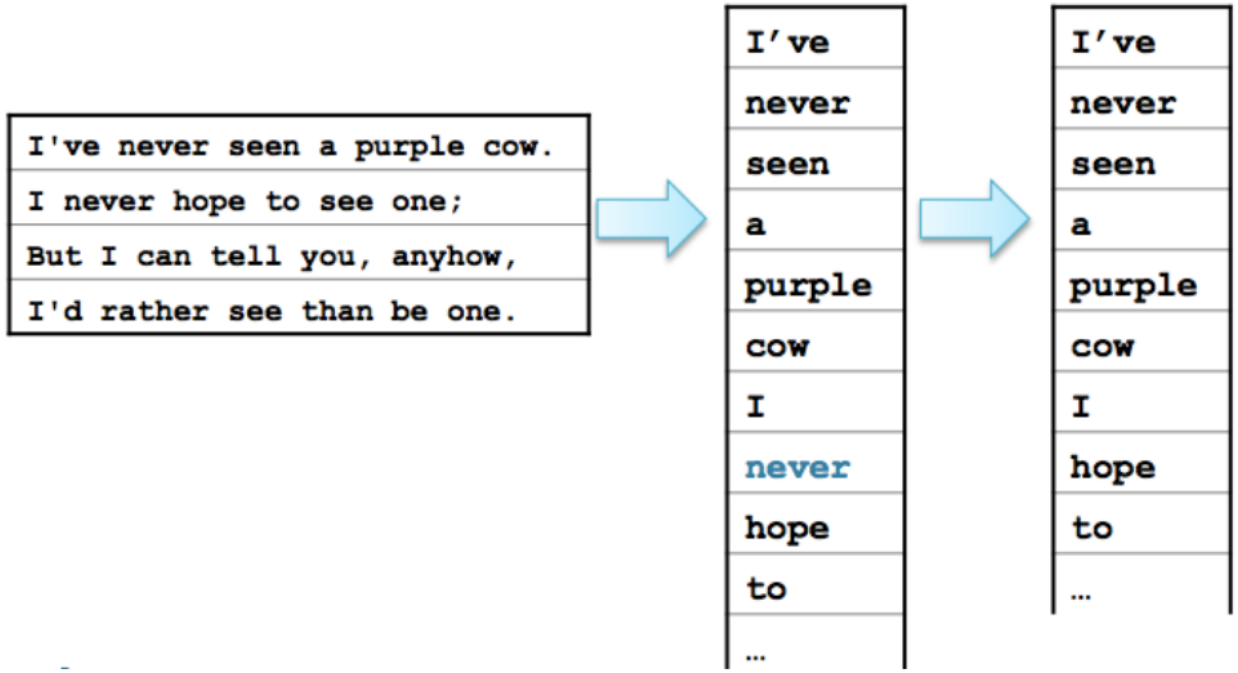


lambda: Python in-line function definition in the format "lambda arguments: return"
e.g. lambda line: line.split(" ")
is equivalent to

```
def split_word(line):
    return line.split(" ")
```

Example: flatMap and distinct

```
> sc.textFile(file) \
  .flatMap(lambda line: line.split()) \
  .distinct()
```



Chaining transformation

```
> mydata = sc.textFile("purplecow.txt")
> mydata_uc = mydata.map(lambda s: s.upper())
> mydata_filt = mydata_uc.filter(lambda s: s.startswith('I'))
> mydata_filt.count()
3
```

is exactly equivalent to

```
> sc.textFile("purplecow.txt").map(lambda line: line.upper()) \
   .filter(lambda line: line.startswith('I')).count()
3
```

\: Python line continuation

Single-RDD vs Multi-RDD

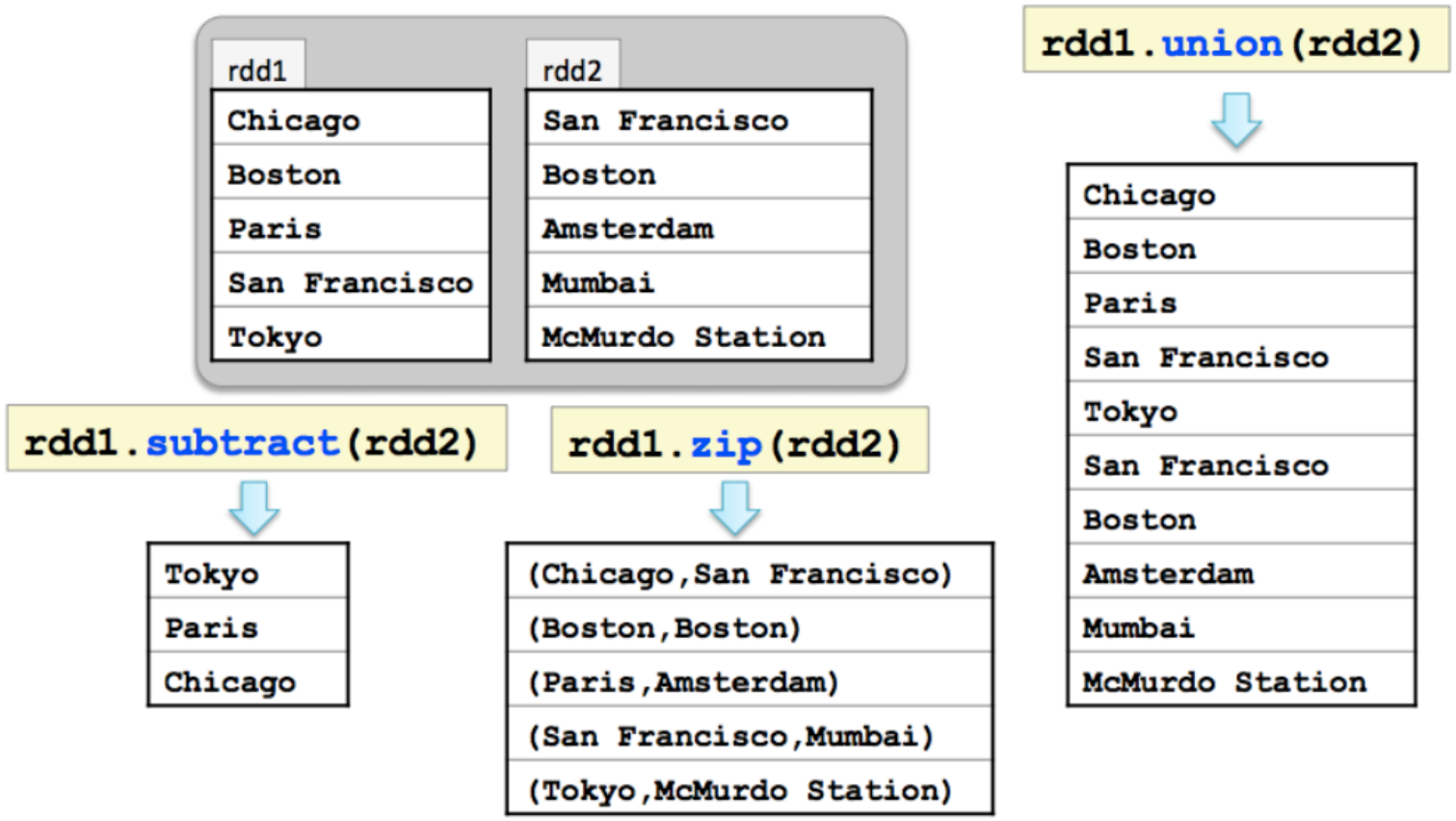
Single-RDD Transformations

- **`flatMap`** – maps one element in the base RDD to multiple elements
- **`distinct`** – filter out duplicates
- **`sortBy`** – use provided function to sort

Multi-RDD Transformations

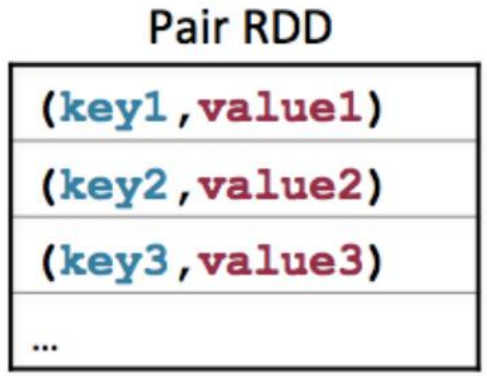
- **`intersection`** – create a new RDD with all elements in both original RDDs
- **`union`** – add all elements of two RDDs into a single new RDD
- **`zip`** – pair each element of the first RDD with the corresponding element of the second

Multi-RDD transformations (Optional)



Pair RDDs

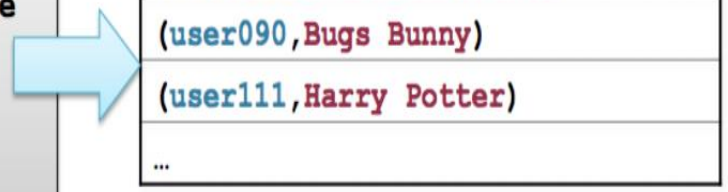
- RDDs consisting of key-value pairs (two-element tuple)
- Special form of RDD
- Keys and values can be any type
- Commonly used functions to create pair RDDs
  - map
  - flatMap
  - keyBy



Example: a simple pair RDDs

```
> users = sc.textFile(file) \
    .map(lambda line: line.split('\t')) \
    .map(lambda fields: (fields[0],fields[1]))
```

```
user001\tFred Flintstone
user090\tBugs Bunny
user111\tHarry Potter
…
```



```
(user001,Fred Flintstone)
(user090,Bugs Bunny)
(user111,Harry Potter)
…
```

```
> sc.textFile(logfile) \
    .keyBy(lambda line: line.split(' ')[2])
```

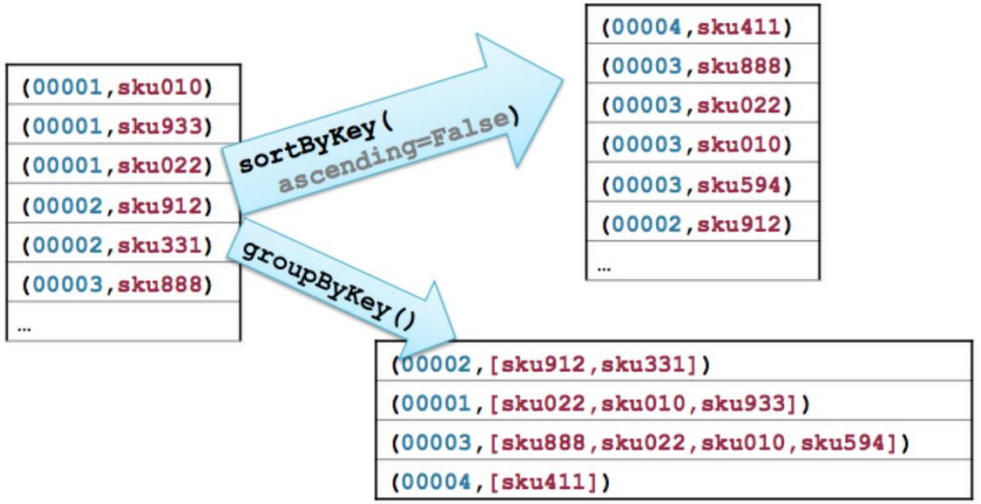
User ID

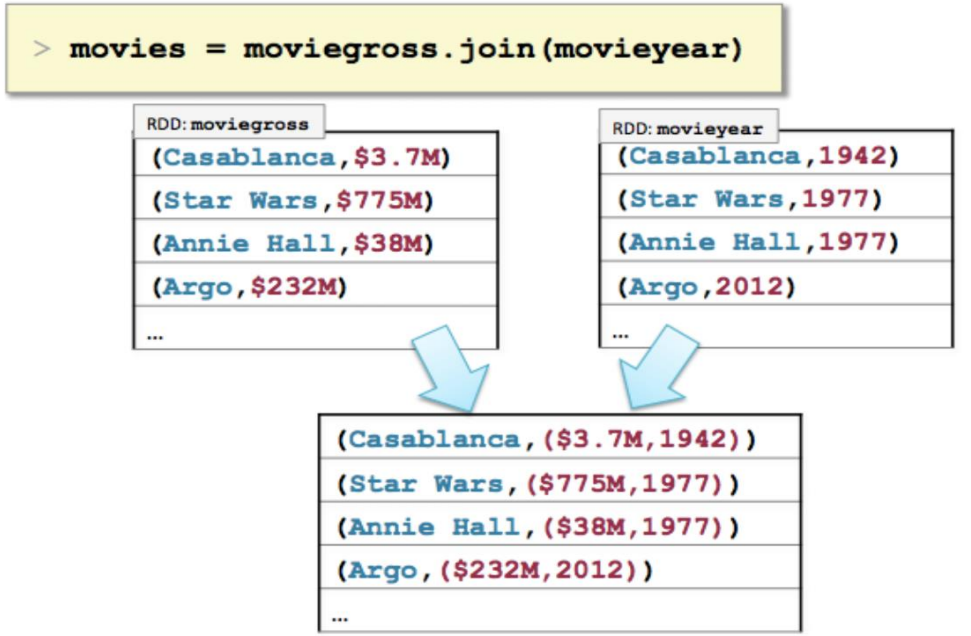
```
56.38.234.188 - 99788 "GET /KBDOC-00157.html HTTP/1.0" …
56.38.234.188 - 99788 "GET /theme.css HTTP/1.0" …
203.146.17.59 - 25254 "GET /KBDOC-00230.html HTTP/1.0" …
…
```

```
(99788,56.38.234.188 - 99788 "GET /KBDOC-00157.html…)
(99788,56.38.234.188 - 99788 "GET /theme.css…)
(25254,203.146.17.59 - 25254 "GET /KBDOC-00230.html…)
…
```

Other pair RDD Operations (Optional)

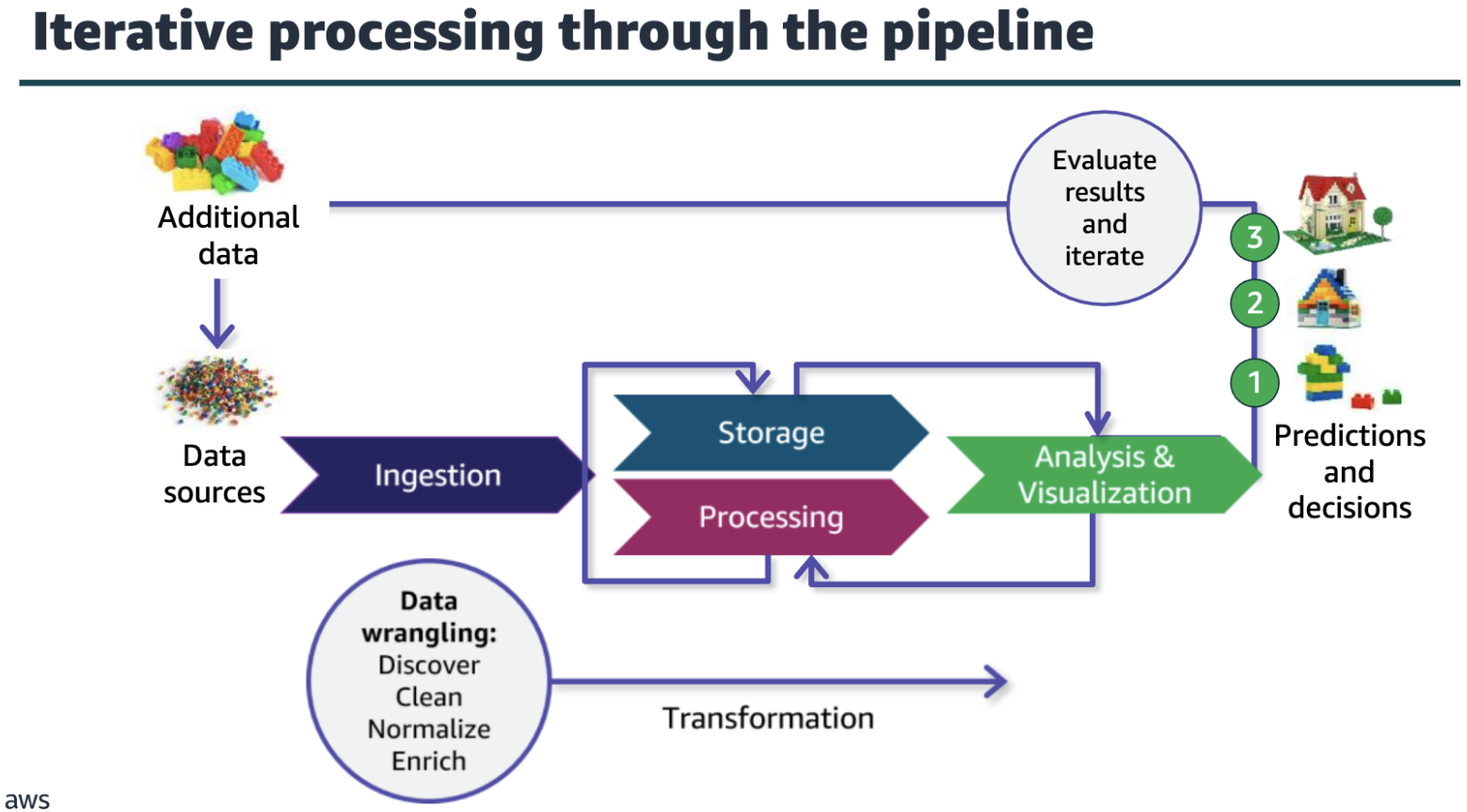
- **groupByKey** – group all the values for each key in an RDD
- **sortByKey** – sort in ascending or descending order
- **join** – return an RDD containing all pairs with matching keys from two RDDs





Data Engineering

Data Pipeline



Role of Data Engineer

Common data pipeline questions

Data engineer

- Does the organization have the data that addresses the need? Where is the data stored and in what format?
- Will I need to combine data from multiple sources?
- What is the type and quality of data? What will be the source of truth?
- What are the security requirements for this data? Who needs access to it and in what state?
- What types of mechanisms are needed to transfer the data from its locations into the pipeline?
- How much data is there, and how frequently is it updated or processed?
- How important is speed when data is requested?

- What can the data tell me?
- How will I evaluate the results?
- What kind of visualization do I need?
- Which formats and tools are the analysts familiar with?
- Do I need a big data framework?
- Which type of AI/ML models fit?
- What is the simplest way to implement AI/ML?

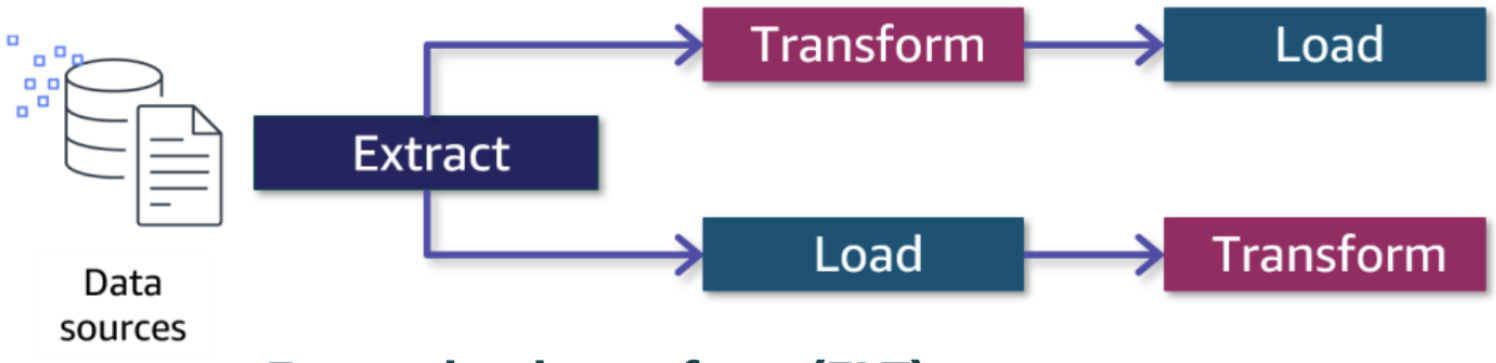
Data scientist

aws

ETL and ELT

Extract transform load (ETL)

1. **Extract** structured data.
2. **Transform** the data into a format that matches the destination.
3. **Load** the data into structured storage for defined analytics.



Extract load transform (ELT)

1. **Extract** unstructured or structured data.
2. **Load** the data into the data lake in the format that is as close to the raw form as possible.
3. **Transform** the data as needed for analytic scenarios.

aws

Benefits of ETL

- Automate routine transformation
- Filter out sensitive data and never store it
- Stored data is ready for analysis
- Complex queries can run more quickly

Benefits of ELT

- The ingestion process can run more quickly
- Don’t need to guess what people will use
- Support ad hoc analysis
- Changes to transformations apply to all historical data

Data Cleaning Tasks (Optional)

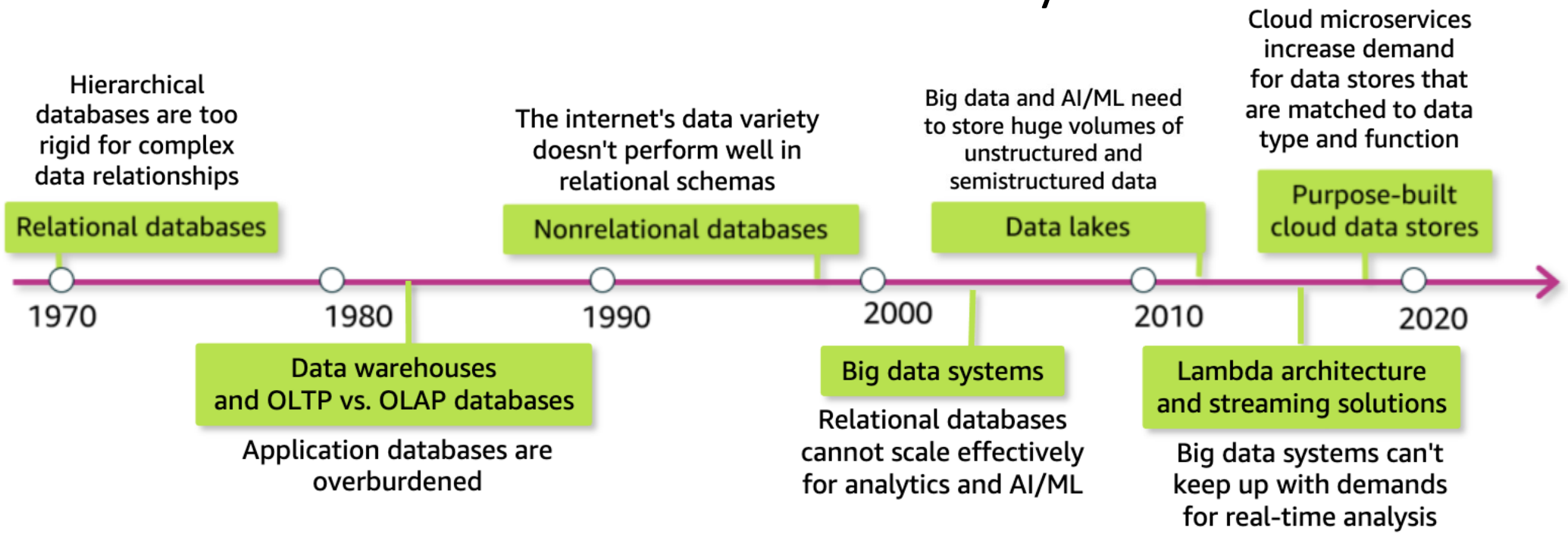
- Remove data
  - Unwanted columns, duplicate values, “garbage” values
- Fill missing data
  - Null columns, missing values for mandatory fields
- Check data types
  - Validate or modify data types
- Fix outliers
  - Identify, remove or fix data as appropriate

Best Practice for Cleaning Data (Optional)

- Define clean
  - Agree on what clean looks like and consistently apply the definition
- Trace errors
  - Return to the source or sources and find the cause
- Change thoughtfully
  - Don't make assumptions about what a data value means
- Retain auditable data
  - Don't discard raw data after cleaning if the raw data has business value

Data Architecture Evolution

- Architecture evolved into more distributed systems



Credit: AWS Academy

Relational vs Nonrelational DBs (Optional)

<table>
<tr><th></th><th>Relational (SQL)</th><th>Non-Relational</th></tr>
<tr><td>Data Storage</td><td>Rows and columns</td><td>Key-value, document, graph</td></tr>
<tr><td>Schemas</td><td>Fixed</td><td>Dynamic</td></tr>
<tr><td>Querying</td><td>Uses SQL</td><td>Focuses on collection of documents</td></tr>
<tr><td>Scalability</td><td>Vertical</td><td>Horizontal</td></tr>
<tr><td>Example</td><td>
<table>
<tr><th>ISBN</th><th>Title</th><th>Author</th><th>Format</th></tr>
<tr><td>3111111223439</td><td>Withering Depths</td><td>Jackson, Mateo</td><td>Paperback</td></tr>
<tr><td>3122222223439</td><td>Wily Willy</td><td>Wang, Xiulan</td><td>Ebook</td></tr>
</table>
</td><td>{
ISBN: 3111111223439,
Title: “Withering Depths”,
Author: ”Jackson, Mateo”,
Format: “Paperback”
}</td></tr>
</table>

Credit: AWS Academy

Data Warehouse (Optional)

- Centralized repository designed to store, manage, and **analyze** large volumes of data from multiple sources
  - For business intelligence (BI) and decision-making purposes
- Structured to facilitate the analysis and reporting

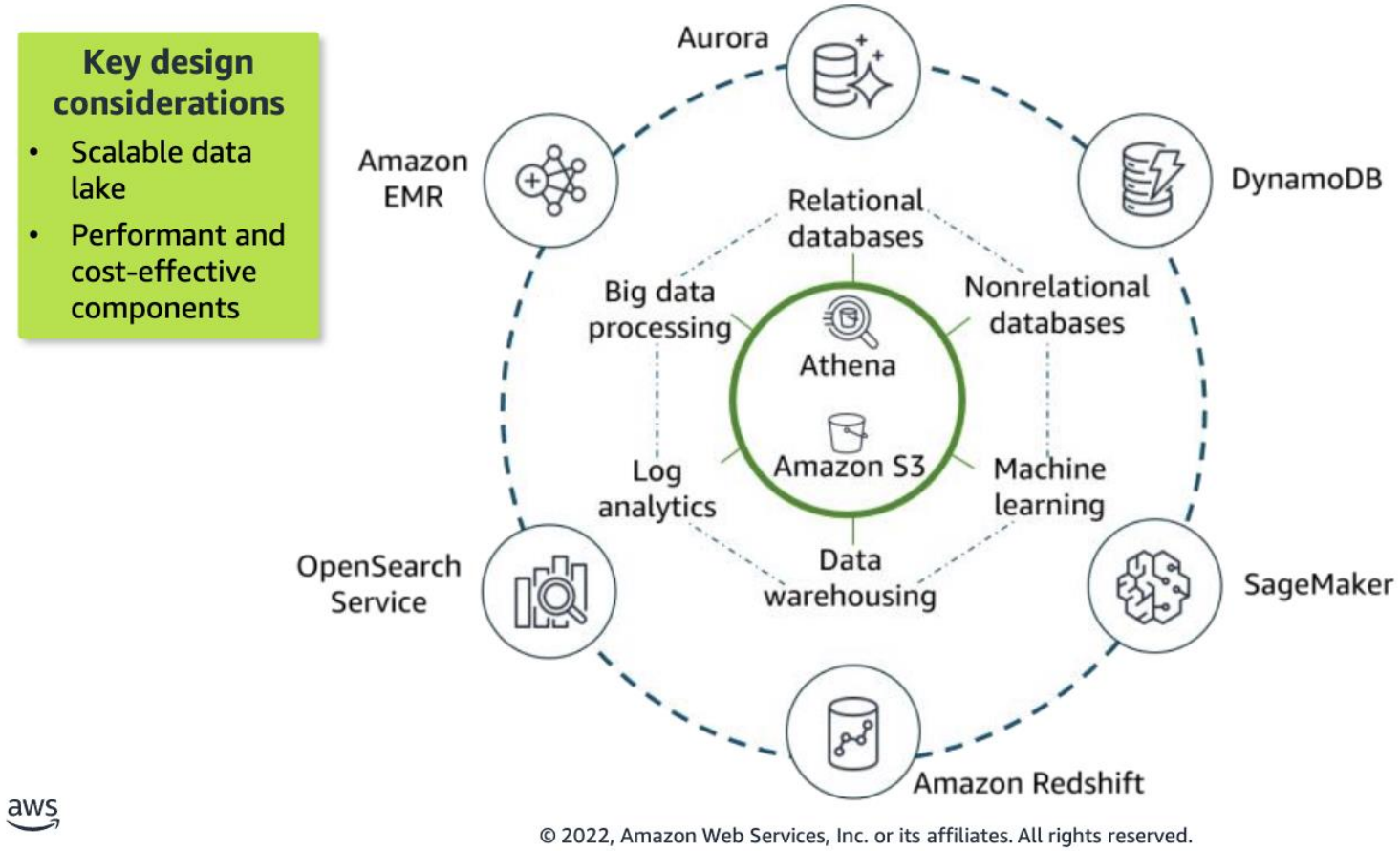
- Database purpose: transaction processing
  - Row-oriented storage
- Data warehouse purpose: analytical processing
  - Column-oriented storage

Data Warehouse vs Data Lake (Optional)

Characteristic	Data Warehouse	Data Lake
Data	Relational data from transactional systems, operational databases, and line of business applications	Nonrelational and relational data from Internet of Things (IoT) devices, websites, mobile apps, social media, and corporate applications
Schema	Designed prior to the data warehouse implementation (schema on write)	Written at the time of analysis (schema on read)
Price and Performance	Fastest query results using higher cost storage	Query results getting faster using low-cost storage
Data Quality	Highly curated data that serves as the central version of the truth	Any data, which might or might not be curated (for example, raw data)
Users	Business analysts	Data scientists, data developers, and business analysts (using curated data)
Analytics	Batch reporting, business intelligence (BI), and visualizations	ML, predictive analytics, and data discovery and profiling

Credit: AWS Academy

AWS Data Stores and Analytics Tools



- Amazon DynamoDB: NoSQL database
- Amazon Redshift: data warehouse
- Will be introduced in week 5 as optional

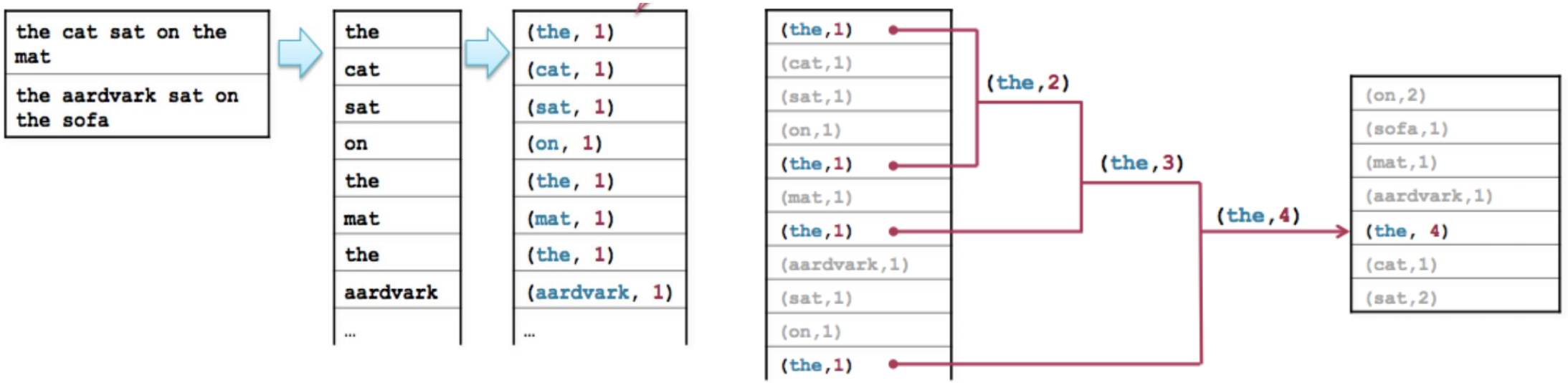
Lab2: PySpark in Google Colab

- You need a Google account to mount Google Drive
- Browser recommendation: don't use Chrome
- Two exercises:
  - Word count using RDD and SparkContext
  - Spam filter using DataFrame, ML and SparkSession
    - 750 spam examples and 750 ham examples
- Homework: sentiment analysis using ML approach
- Extension: sentiment analysis using dictionary approach and Hive

Example: Word Count

- **Pseudocode**

```
Split line into words
Map each word to (word, 1)
Reduce for each word to summation of counts
```



Wordcount PySpark Code

```
#create RDD from a text file
text_file = sc.textFile("Google_CoLab_folder/ebook.txt")
#1st line split line into words
#2nd line map word to (word,1)
#3rd line reduce values to summation
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)



#write output to a text file
#and output file must not exist!
counts.saveAsTextFile("/content/output")
```

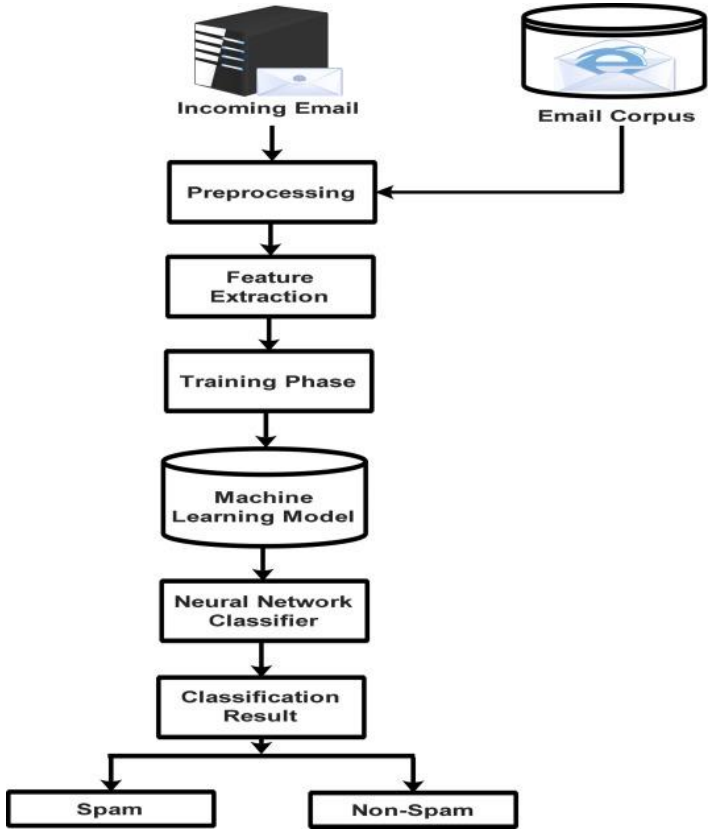
sc: SparkContext

flatmap: map one line to multiple words
lambda: Python in-line function definition in the format "lambda arguments: return"
e.g. lambda line: line.split(" ")
is equivalent to

```
def split_word(line):
    return line.split(" ")
```

\: Python line continuation

Example: Spam Filtering



Pseudocode

Training phase:

1. Get some spam and ham examples
2. Convert to words (features)
3. Use ML to identify spam-related words and ham-related words (classifier)

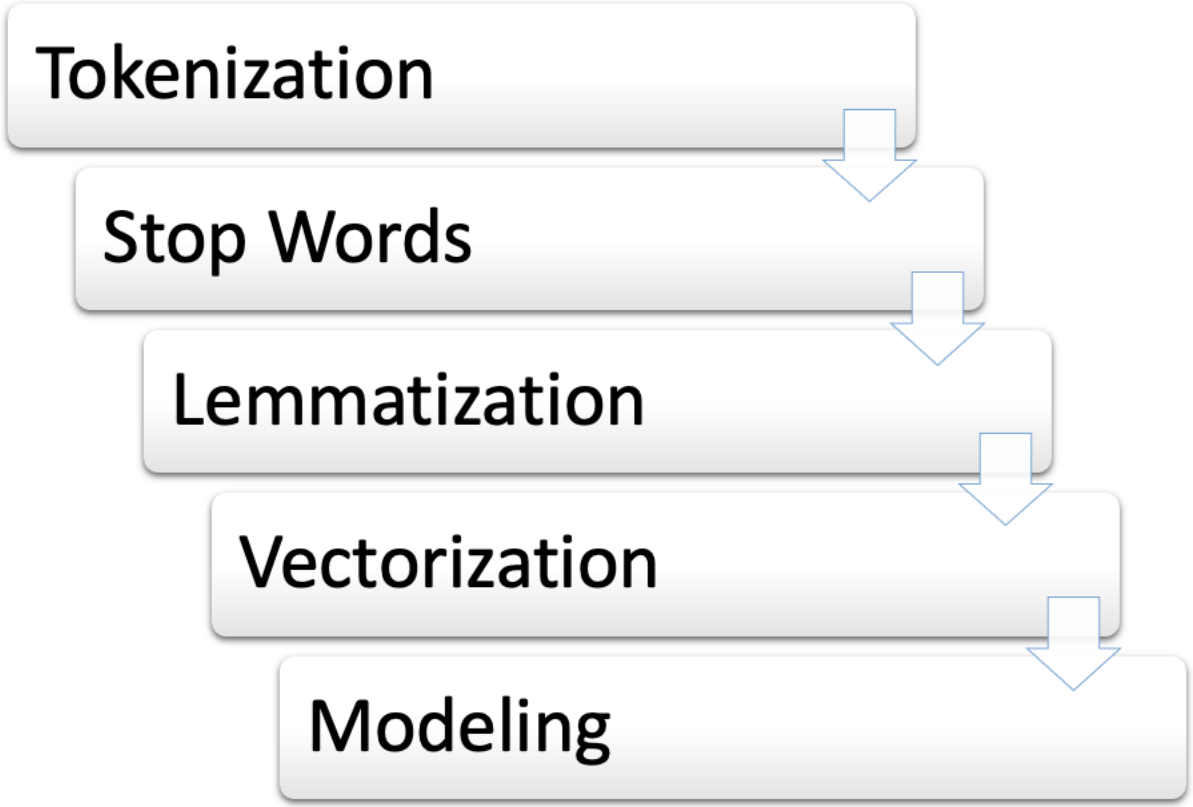
Testing phase:

1. Convert test case to words (features)
2. Use learnt classifier to predict case

Tokenization

- Break a stream of text into meaningful units
- Tokens: words, phrases, symbols
- You can split text using space and/or special characters
  - Such as apostrophe’ hyphen- quotations’”” comma, period. etc
- Depends on language, corpus, or even context
- Example:
  - Input: It’s easy to use Hadoop eco-system in BU.330.740.
  - Output(1): ‘It’s’ ‘easy’ ‘to’ ‘use’ ‘Hadoop’ ‘eco-system’ ‘in’ ‘BU.330.740’
  - Output(2): ‘It’ ‘’’ ‘s’ ‘easy’ ‘to’ ‘use’ ‘Hadoop’ ‘eco’ ‘system’ ‘in’ ‘BU’ ‘330’ ‘740’
- More advanced tokenization in BU.520.710 AI Essentials for Business

NLP Workflow (Optional)



From BU.520.710 AI Essentials for Business

Spam Filtering PySpark Code

```
# Import ml package since we will use ML models
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.feature import Tokenizer, HashingTF
from pyspark.ml import Pipeline
from pyspark.sql.functions import lit
# Training: load spam and ham examples
spam_df = spark.read.text("Google_CoLab_folder/spam.txt")
ham_df = spark.read.text("Google_CoLab_folder/ham.txt")
# Label spam columns with value 1, ham columns with value 0
spam_df = spam_df.withColumn("label", lit(1))
ham_df = ham_df.withColumn("label", lit(0))
# Combine the spam and ham datasets into a single dataset
combined_df = spam_df.union(ham_df)
```

adds a new column.

HashingTF: transform words into fixed-length feature vectors using hashing function. e.g. “Hadoop class uses Hadoop” can be converted to [(00,1),(01,2),(11,1)]
Here, “class” is hashed to 00, with count 1
“Hadoop” is hashed to 01, with count 2
“uses” is hashed to 11, with count 1
https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.HashingTF.html

lit() function: create a column with a literal value. It's helpful when you want to add a constant value to a DataFrame

Spam Filtering PySpark Code (Cont.)

```
# Create a ML pipeline
tokenizer = Tokenizer(inputCol="value", outputCol="words")
hashingTF = HashingTF(inputCol="words", outputCol="features")
lr = LogisticRegression(featuresCol="features", labelCol="label")
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```

from raw text to tokenized words

from tokenized words to numerical feature vectors

```
# Split data into training and testing sets
(trainingData, testData) = combined_df.randomSplit([0.95, 0.05], seed=100)
print(f"Training Data Count: {trainingData.count()}")
print(f"Testing Data Count: {testData.count()}")
```

Spam Filtering PySpark Code (Cont.)

```
# Train the model
model = pipeline.fit(trainingData)

# Make predictions on the test set
predictions = model.transform(testData)

# Evaluate the model
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction", labelCol="label")
accuracy = evaluator.evaluate(predictions)
print(f"Accuracy: {accuracy}")

# Print 70 records
predictions.select("value", "label", "prediction").show(70, truncate = False)
```

Sentiment Analysis

- Sentiment analysis is the detection of attitudes
- Other names include Opinion Extraction, Opinion Mining, Sentiment Mining, Subjectivity Analysis
- Use cases include
  - Movie: is this review positive or negative?
  - Products: what do people think about the new iPhone?
  - Politics: what do people think about this candidate or issue?
- We will see “sentiment analysis” on images in week 5
- Two main techniques: machine learning approach and dictionary-based approach

ML Approach: Baseline Algorithm

1. Tokenization
2. Feature extraction
3. Classification

Pseudocode

Training phase:

1. Get some positive and negative examples
2. Convert to words (features)
3. Use ML to identify positive-related words and negative-related words (classifier)

Testing phase:

1. Convert test case to words (features)
2. Use learnt classifier to predict case

Next Week

- Scaled machine learning pipeline
- Recommendation system
- AWS SageMaker

References

- Kunpeng Zhang's notes (2019)
- AWS Academy Data Engineering