



Industry Leader in Customized IT & Management Training



COURSE NOTES

Module 2:

Global Technical Analyst Program





ROI TRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Module 2: Global Technical Analyst Program

INTRODUCTION

Welcome!

- ◆ Meet your instructor
 - Name
 - Background
 - Contact info
- ◆ Let's get started!



Course Objectives

In this course, we will:

- ◆ Revise our knowledge of Java
- ◆ Highlight successful software team practices
- ◆ Investigate the principles of blockchain algorithms
- ◆ Understand the concepts of MapReduce and Big Data
- ◆ Leverage Hadoop as a reliable, scalable MapReduce framework
- ◆ Utilize the Hadoop distributed file system (HDFS) for storing big data files
- ◆ Develop MapReduce applications with Apache Spark
- ◆ Complete a substantial team project using the technologies discussed

Course Contents

- Chapter 0 Introduction
- Chapter 1 Effective Software Development
- Chapter 2 Java Foundations—Recap
- Chapter 3 Blockchain Fundamentals
- Chapter 4 Introducing MapReduce with Hadoop
- Chapter 5 Hadoop Distributed File System (HDFS)
- Chapter 6 Hadoop MapReduce
- Chapter 7 Apache Spark with Java
- Chapter 8 Introduction to Machine Learning
- Chapter 9 Cloud Computing
- Chapter 10 Course Summary

Class Schedule

◆ Start of class _____

◆ Morning breaks approximately on the hour

◆ Lunch _____

◆ Afternoon breaks approximately on the hour

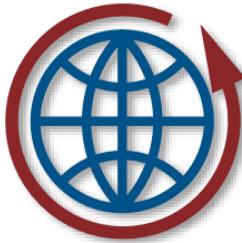
◆ Class end _____

Student Introductions

Please introduce yourself stating:

- ◆ Name
- ◆ Position or role
- ◆ Expectations or a question you'd like answered during this class





ROI TRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Module 2: Global Technical Analyst Program

CHAPTER 1: EFFECTIVE SOFTWARE DEVELOPMENT

Chapter Objectives

In this chapter, we will introduce:

- ◆ The Software Development Life Cycle (SDLC)
- ◆ The role of an architect and technical lead in software delivery

Chapter Concepts



Software Development Life Cycle

Agile Development

Continuous Delivery

Chapter Summary

What Is SDLC?

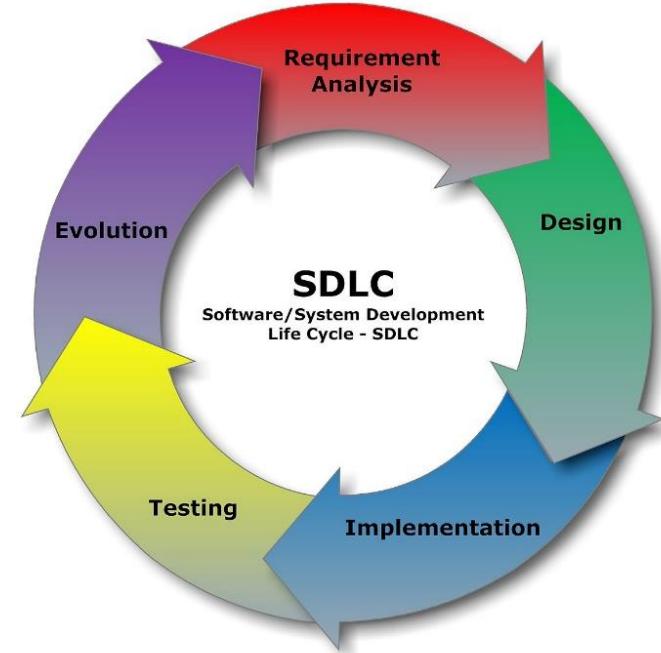
- ◆ Software Development Life Cycle (SDLC) is a process for developing, maintaining, replacing software
 - Aims to produce a high-quality software that:
 - ◆ Meets or exceeds customer expectations
 - ◆ Reaches completion within times and cost estimates

SDLC Models

- ◆ Following are the most important and popular SDLC models followed in the industry:
 - Waterfall Model
 - Iterative Model
 - Spiral Model
 - V-Model
 - Big Bang Model
- ◆ There is also a consideration of the role and makeup of teams
 - Distinct teams/departments
 - ↳ E.g., Development, test/QA, operations/support
 - DevOps
 - ↳ Those that build it, deploy, and support it

Stages of SDLC

- ◆ All models have a number of key stages shown on the diagram
- ◆ The main difference between models is the size of the work units performed on each cycle around the loop
- ◆ Focus and key tasks to be performed depend on project management approach as well as the type and size of the project



Agile Projects

- ◆ Using Scrum/Kanban, XP Agile principles and processes
- ◆ Focus around self-organizing teams with lightweight processes and frequent delivery
 - Align all architecture work with the culture of development teams
 - Deliver often, in small iterations
 - Give more power to the teams
 - Favor individuals and interactions over processes and tools, working software over comprehensive documentation
- ◆ Agile != Chaos
 - Architecture is still very important
 - Done in iterative fashion as well
 - Long term planning and roadmaps are useful
 - Changing direction by 180° is expensive

Chapter Concepts

Software Development Life Cycle

➤ **Agile Development**

Continuous Delivery

Chapter Summary

Agile Framework Collaborators

- ◆ **Product Owner** maintains all product requirements, features, etc. in **Product Backlog**
- ◆ **Scrum Team** selects a subset of items from Product Backlog (**Sprint Backlog**) based on priority and other factors and implements them in iterative fashion, through **Sprints**
 - A time box of one month or less during which a “Done”, useable, and potentially releasable ‘**product increment**’ is created
- ◆ During Sprint
 - Work effort is planned and coordinated through short, 15-minute **Daily Scrum (Standup)** meetings
 - **Scrum Master** facilitates activities, removes obstacles, and protects the team from outside interruptions and distractions
- ◆ At the end of the Sprint:
 - Scrum team demonstrates product to key stakeholders during **Sprint Review** meeting
 - Reflects on ways to improve during **Sprint Retrospective** meeting

The Agile Scrum Framework at a glance

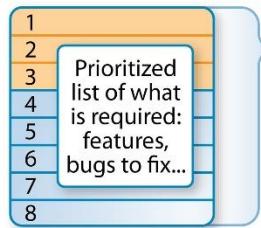
Inputs from
Customers, Team,
Managers, Execs



Product Owner



The Team



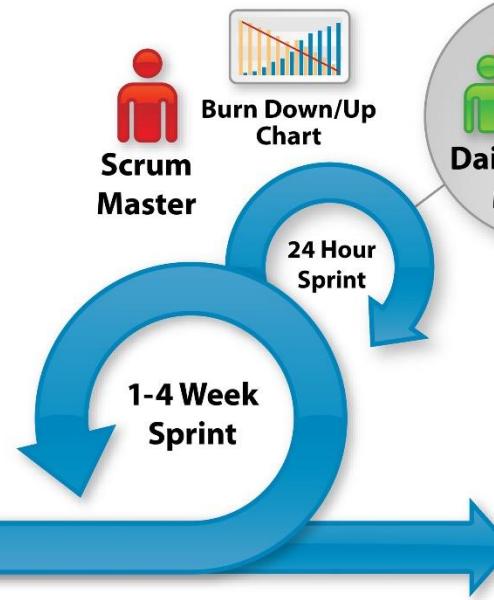
Product Backlog

Team selects starting at top as much as it can commit to deliver by end of Sprint

Sprint Planning Meeting

Task Breakout

Sprint Backlog



Sprint end date and team deliverable do not change



Finished Work



Sprint Retrospective

Creative commons license: <https://creativecommons.org/licenses/by-nd/3.0/nz/>

Overview of Scrum Process

- ◆ Starting point is list of all things system should include and address
 - Functionality
 - Features
 - Technology
- ◆ Work is undertaken by Scrum teams
 - Small, cross-functional teams
 - Take on part of product backlog to turn into increment in allocated days
 - Allocated day period is known as *Sprint*
- ◆ Multiple teams can develop increments in parallel
 - Scrum of Scrums

Meet the Team

- ◆ Cross-functional team responsible for *all* aspects of development
 - Analysis
 - Design
 - Implementation
 - Testing
- ◆ Balance of team member skills is vital in achieving successful outcomes
 - Technical skills
 - Domain knowledge
 - Communication skills
- ◆ Commits to achieving Sprint goal
 - Has full authority to do whatever is necessary to achieve the goal
 - Only constraints are organizational standards and conventions



Product Backlog

Evolving, prioritized queue of business and technical functionality that needs to be developed into a system

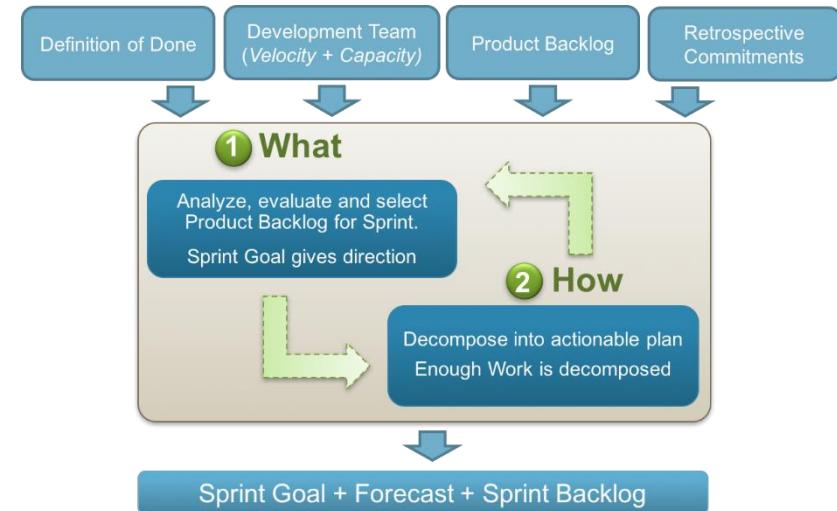
- ◆ Product Owner maintains product backlog based on requirements from various stakeholders
 - Dev Team and Scrum Master can contribute to the backlog
- ◆ List of:
 - Features
 - Functions
 - Technologies
 - Enhancements
 - Bug fixes
 - Anything that represents 'work to be done'
- ◆ Backlog is dynamic
 - Requirements emerge/evolve
 - Exists as long as product exists

Estimating Product Backlog Effort

- ◆ Scrum team estimates the effort of the items in the backlog
- ◆ Estimating is an iterative process
 - Estimates change as items become better understood
 - Initial estimate can be done as part of 'backlog grooming'
- ◆ Estimate is not binding
 - It is a starting point—best guess
 - Continually refined
- ◆ Product Owner is prioritizing backlog items
 - Ultimate decision maker in terms of what is important
 - ◆ Team can provide feedback and suggest
- ◆ Includes time to:
 - Define architecture, design, test, and build

Planning a Sprint

- ◆ The work to be performed in the Sprint is planned at the Sprint Planning
 - This plan is created by the collaborative work of the entire Scrum team
 - Meeting is normally 8 hours or less
- ◆ During Sprint Planning, Scrum team decides on:
 - **What** to deliver in the next Sprint
 - **How** will the work be achieved
- ◆ Two key outcomes of Sprint Planning:
 - Sprint Goal (the 'what')
 - ↳ A short statement of what the team plans to achieve during the sprint
 - Sprint Backlog (the 'how')
 - ↳ List of the product backlog items the team commits to delivering
 - ↳ Breakdown of the backlog items into tasks, with more detailed estimates



Sprint Constraints

- ◆ Every product development is constrained by four variables
 - Time
 - Cost (people and resources)
 - Delivered quality
 - Delivered functionality
- ◆ During Sprint, the first three are fixed
 - Fourth can be changed
 - ◆ As long as it still meets the Sprint goal

Sprint Meetings

- ◆ Scrum team meets daily (Daily Scrum)
 - Progress reviewed
 - Impediments identified
 - Provides feedback on progress
- ◆ At the end of Sprint, Scrum team meets with management (sprint review meeting, retrospective)
 - Inspect product increment
 - Product backlog rearranged based on Sprint results

Daily Scrum Meetings

- ◆ During the Daily Scrum, each team member typically answers three questions
 - What did I complete yesterday?
 - What do I plan to complete today?
 - Do I see any impediment for the team or myself?
- ◆ The Daily Scrum
 - Starts precisely on time even if some development team members are missing
 - Should happen at the same time and place every day
 - Is limited to 15 minutes
- ◆ Well organized Daily Scrums:
 - Improve communication
 - Enable progress to be accurately tracked
 - Promote quick decision making
 - Eliminate other meetings
 - Identify and remove impediments

What Is NOT a Daily Scrum Meeting?

- ◆ NOT a venue to discuss technical details, argue, or design a solution
 - Should be taken offline and separate meetings are to be set up
- ◆ NOT a project status update
 - Primary goal of Daily Scrum is to plan future work, not to discuss work that has been done
- ◆ It is for the TEAM and NOT for Scrum Master
 - Scrum Master sets up the meeting and facilitates, but team does NOT report to him
- ◆ NOT a planning meeting
 - Any big changes that can affect achieving sprint goal should be discussed in a separate meeting

Follow-Up Meetings

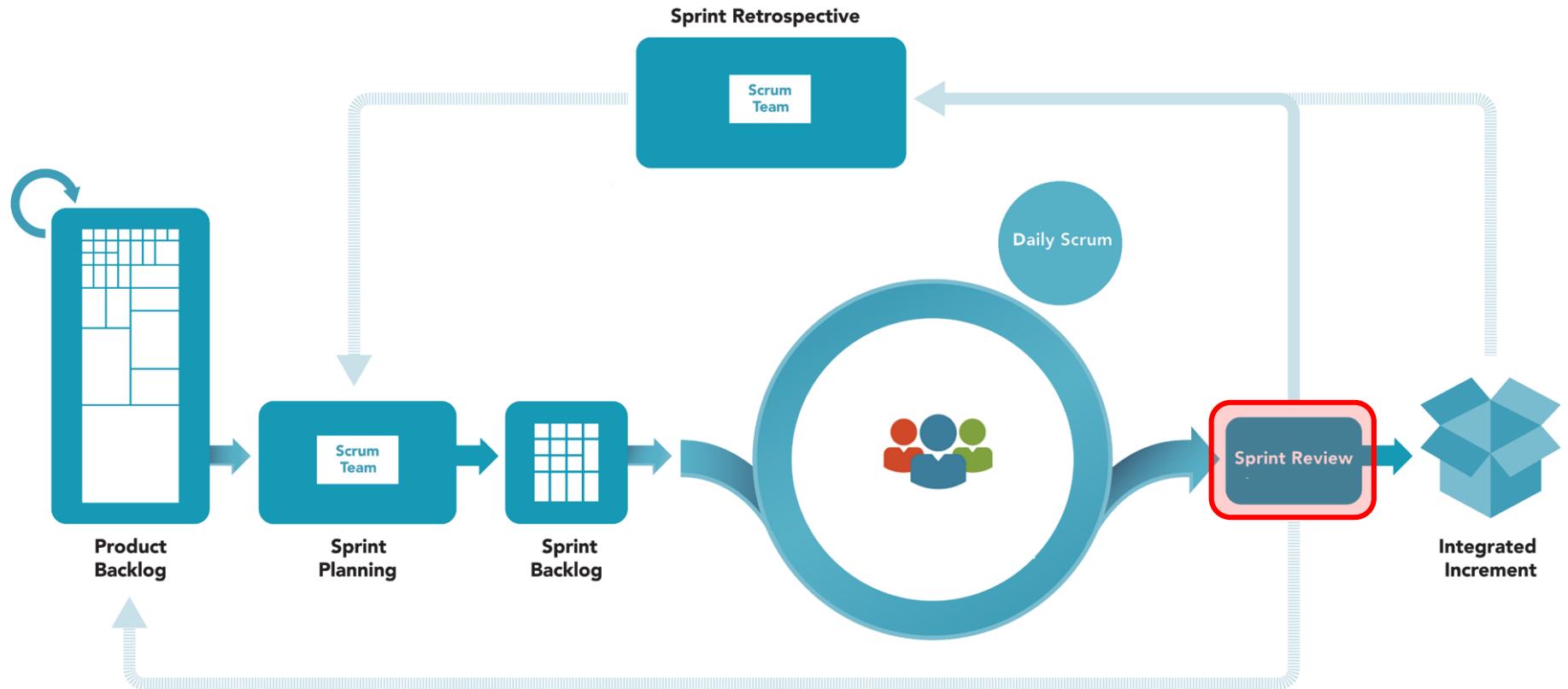
- ◆ Team members can request follow-up meetings during Daily Scrum
 - Referred to as working sessions
- ◆ Number of reasons for these sessions including:
 - One team member knows a solution to an impediment
 - A team member wants to share information
 - A design decision needs to be made
 - Technology choice must be made
- ◆ Any team member is welcome to attend
 - Held immediately after Daily Scrum where session was requested
- ◆ Helps maintain Daily Scrum as focused and short

Scrum Master's Role in Daily Scrum

- ◆ Responsible for conducting Daily Scrum
 - Time and location
- ◆ Establishing layout of the meeting room
 - Ideally, table with just enough chairs for team members
 - Whiteboards for recording impediments, progress, etc.
 - Door to be closed to prevent interruptions!
- ◆ Meeting enables Scrum Master to sense how team is progressing
 - Individuals becoming swamped
 - Members lost interest
 - Somebody not working efficiently
 - Team chemistry

Identifying Impediments

- ◆ Scrum Master is responsible for removing impediments
 - Should be recorded on whiteboard
- ◆ Common impediments include:
 - Unsure of how to best use technology
 - Design decision needs to be made
 - Asked by management to undertake a different task
 - Infrastructure (server, network, workstation) slow
- ◆ Scrum Master must do all that is possible to remove impediments
 - If not removed, should be reported next day



Sprint Review Meeting

- ◆ A meeting held to review and demo the *product increment* to the team as well as to external stakeholders
 - I.e., meeting to review WHAT has been done
- ◆ Usually four hours allocated to meeting on last day of Sprint
- ◆ Scrum Master responsible for conducting meeting
- ◆ Attended by:
 - Product owner
 - Management
 - Customers
 - Users
- ◆ Product increment is presented and demonstrated

Sprint Retrospective

- ◆ A meeting to review and discuss things that did or did not work *within team* during the last sprint
 - The focus is on HOW things were done and find areas of improvement
 - Meeting is 'inward' looking
 - ◆ This is the time and the place for team to praise each other or bring up some grudges in a safe, non-judgmental environment
- ◆ Usually the last thing done in a sprint
 - Many teams will do it immediately after the sprint review
 - Normally takes about an hour, though may take longer
- ◆ The entire Scrum team normally participates
 - This does NOT include external stakeholders or anyone else outside of Scrum team

Chapter Concepts

Software Development Life Cycle

Agile Development

➤ **Continuous Delivery**

Chapter Summary

Continuous Delivery

- ◆ A software strategy that enables delivery of new features to users as fast and efficiently as possible
 - Create a repeatable and reliable incremental process
 - Takes software from concept to customer
- ◆ A delivery pipeline makes this happen
 - Breaks the delivery process into stages
 - Each stage verifies software quality from a different angle
 - Validates new functionality
 - Prevents regressions
 - Simplifies deployment

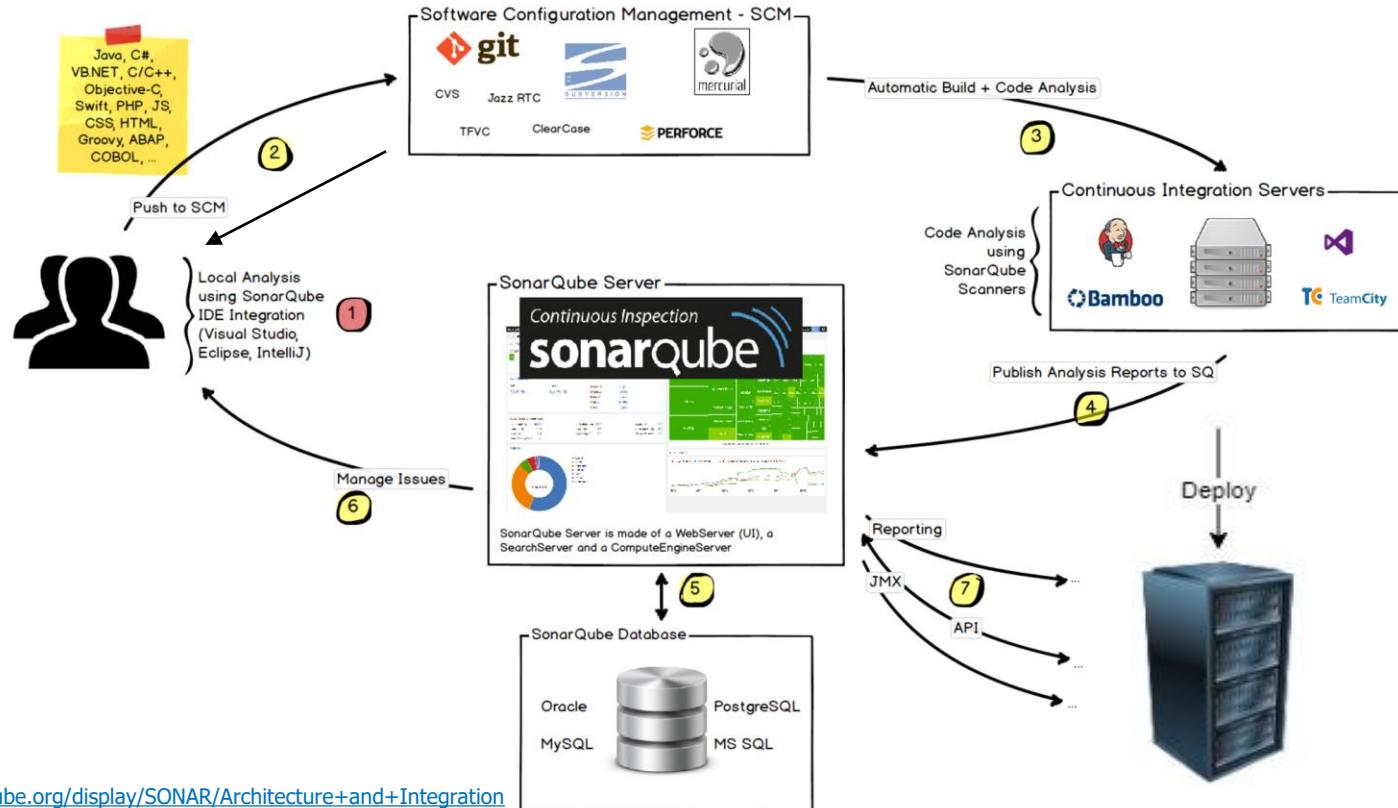
Major Stages of Continuous Delivery

- ◆ Build automation and continuous integration
 - Builds software by integrating changes into central code base continually
- ◆ Test automation
 - Rigorous automated tests to meet desired system qualities
 - All aspects, functional, performance, compliance are verified
- ◆ Deployment automation
 - Automated deployment to live environment
- ◆ Orchestration
 - Multiple stages in deployment pipeline automated by tools

Key Components

- ◆ Implementing a continuous delivery pipeline requires core components
 - Source code repository
 - Code quality analysis
 - Continuous integration testing
 - Automated deployment

Continuous Delivery Cycle



Source Configuration Management

- ❖ We need to track changes in our projects
 - New features added
 - Bugs fixed
- ❖ Bug/feature tracking
 - Bitbucket
 - Jira
 - GitHub
- ❖ Source code versioning
 - Subversion
 - Mercurial
 - Git



Continuous Integration

- ◆ Continuous Integration (CI)
- ◆ Typical flow controlled by CI server
 - Developer commits code
 - CI server clones repository
 - Runs tests
 - Runs quality analysis
 - Generate reports and raise alarms
 - May also deploy checked systems to production
- ◆ Many to choose from
 - TeamCity
 - Jenkins
 - Bamboo
 - Travis

Static Code Analysis

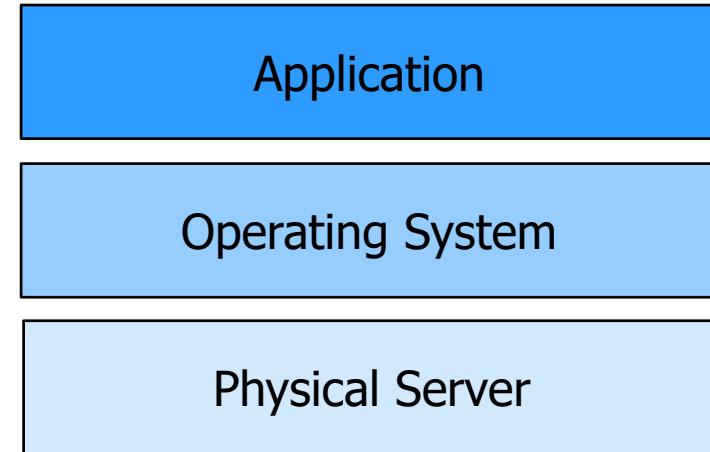
- ◆ Static code analysis tool
 - Can be used as a quality gate in a CD pipeline
 - ◆ Provides feedback on:
 - Code quality against industry standards
 - Technical debt
 - Complexity
 - Code test coverage
 - Used to detect potential bugs

Automated Deployment

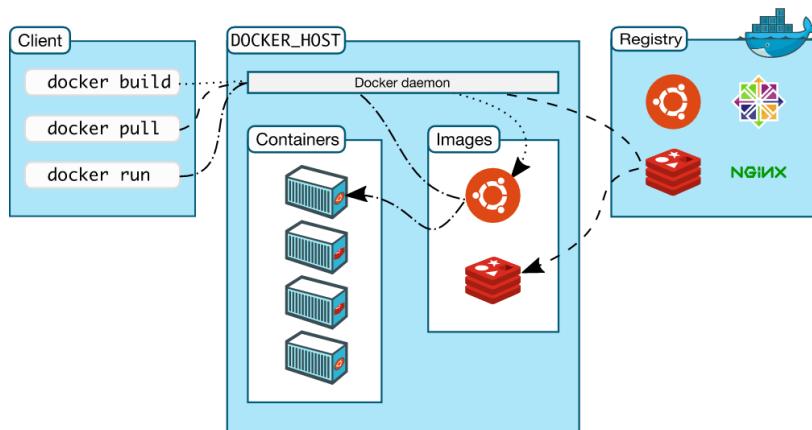
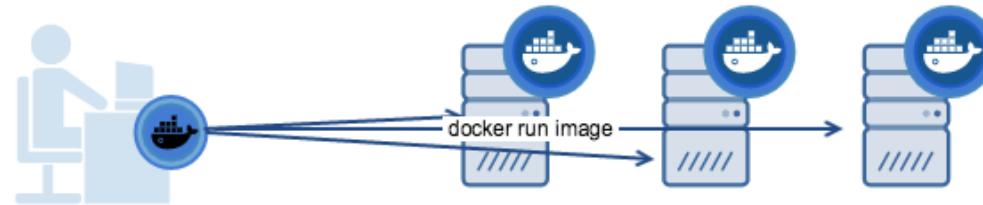
- ◆ Manually deploying software is error prone
 - Repetitive tasks
 - Time consuming
 - Potentially complex for large systems
 - **Developers often involved**
 - ↳ Security concerns
- ◆ Automatically deploying versions of systems has many advantages
 - Use off-the-shelf solutions or build in-house
 - Deployments can be:
 - ↳ Versioned
 - ↳ Rolled back
 - **DevOps take responsibility**
- ◆ Can be very complex
 - Deal with different hardware, operating systems, packaging, programming languages

What Is Docker?

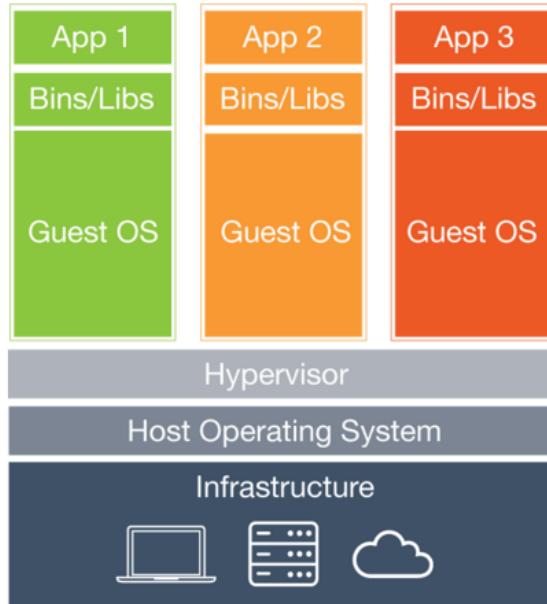
- ◆ Consider deploying an application on a server
- ◆ Choices include:
 - Traditional servers
 - Virtual machines
 - Containers



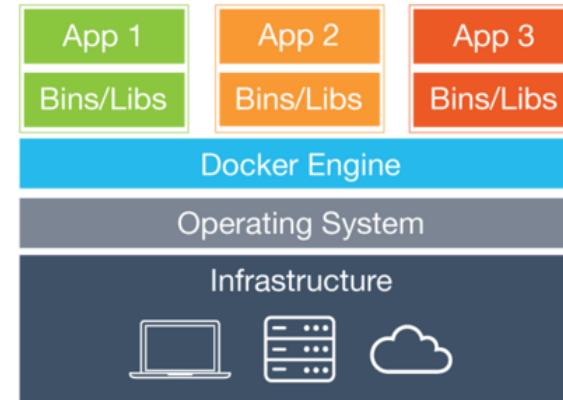
Introducing Docker



Virtual Machines and Containers



Virtual Machines



Containers

Container Setup

- ◆ Key difference of containers from virtual machines is:
 - Container uses kernel of host OS to run multiple isolated guest instances
 - Guest instances are called containers
 - More lightweight
- ◆ Host can be a physical server or virtual machine
- ◆ Enables isolation of resources
 - Process
 - File system
 - Network
 - Memory/CPU
- ◆ Run locally to test
- ◆ Deploy to different servers



Docker Demo

- ◆ Your instructor will now demonstrate how a package can be installed with Docker

Chapter Concepts

Software Development Life Cycle

Agile Development

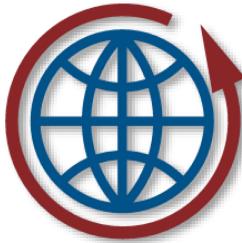
Continuous Delivery

Chapter Summary

Chapter Summary

In this chapter, we have introduced:

- ◆ The Software Development Life Cycle (SDLC)
- ◆ The role of an architect and technical lead in software delivery



ROI TRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Module 2: Global Technical Analyst Program

CHAPTER 2: JAVA FOUNDATIONS—RECAP

Chapter Objectives

In this chapter, we will:

- ◆ Refresh our knowledge of Java, its key constructs and features

Chapter Concepts

➤ Java Introduction

Unit Tests

Inheritance and Polymorphism

Handling Exceptions

Interfaces

Collections

Chapter Summary

Introduction to VM Environment



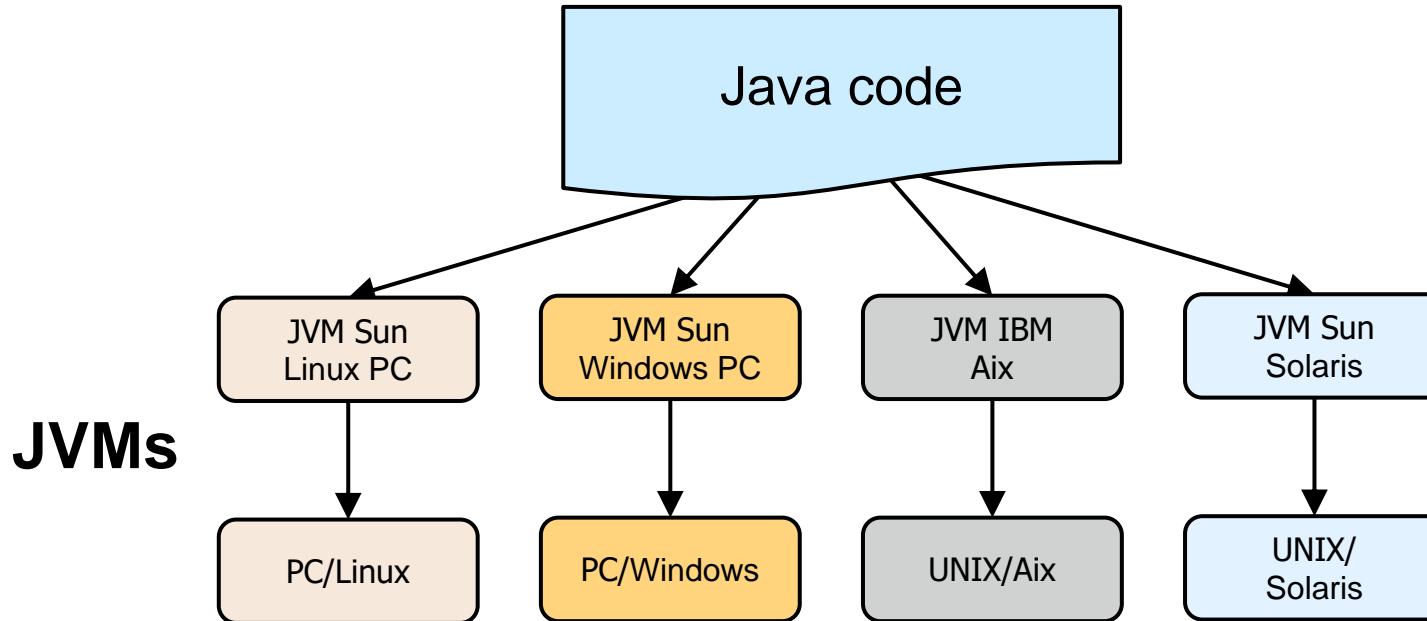
- ◆ All hands-on exercises in the class will be performed within virtual environment running on the laptop
- ◆ Your instructor will now guide you through the setup

What Is Java?

- ◆ Java is an object-oriented language designed by James Gosling
 - Created by Sun Microsystems (now part of Oracle)
 - Designed to be platform-independent
- ◆ Enhancements to Java are coordinated through a community model
 - Representatives from major Java tool and services providers

Java Code Executes in a Java Virtual Machine

- In order to run Java programs, need to have a Java virtual machine installed

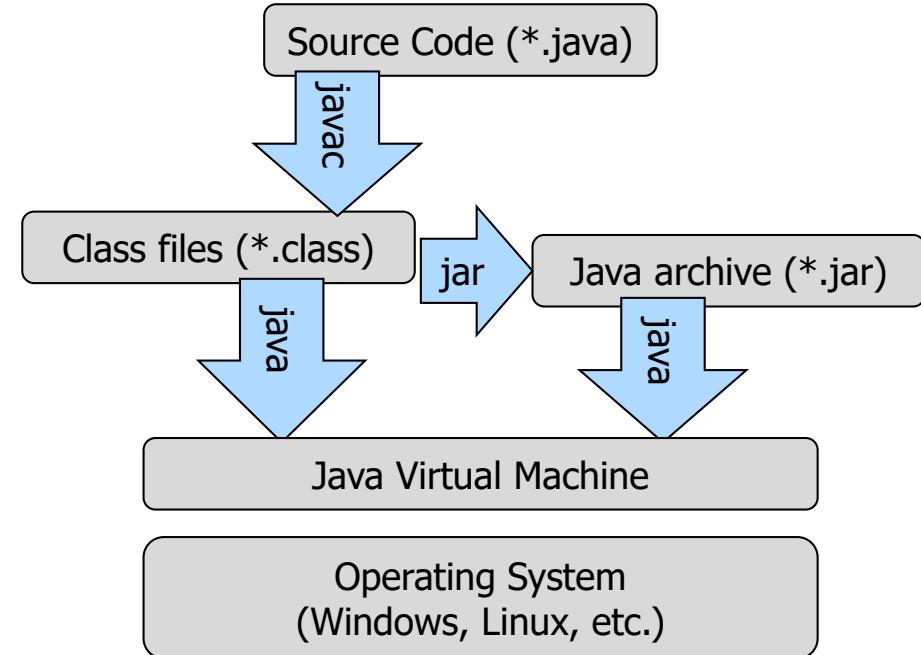


What Software Is Needed for Development?

- ◆ Any text editor to create source code
 - TextPad, Notepad, Notepad++ on Windows
 - vi or emacs on Linux
 - TextEdit, Sublime Text 2 on Mac
- ◆ A Java Development Kit (JDK)
 - aka [Java SE Development Kit \(SDK\)](#)
 - Contains a Java Runtime Environment
 - Other tools: *java, javac, javadoc, jar*
- ◆ For convenience, an Integrated Development Environment (IDE) is used for editing and executing the code
- ◆ Most popular IDEs:
 - Eclipse
 - IntelliJ
 - NetBeans

Steps to Create Programs

- ◆ We create text files/source code (*.java)
- ◆ Compile source code into bytecode (*.class)
 - Using **javac** tool from JDK
- ◆ Optionally, combine files into archives (*.jar)
 - Using **jar** tool from JDK
- ◆ Execute code
 - Using **java** NameOfClass (no extension)



What Type of Source Code Do We Write?

class

MyFirstApplication.java

```
public class MyFirstApplication {  
    public static void main(String[] args) {  
        // Programs Start Here  
    }  
}
```

Customer.java

```
public class Customer {  
    //member variables aka fields aka data  
  
    //methods  
}
```

interface

Payable.java

```
interface Payable {  
    //public static methods  
    double calculatePay();  
}
```

Rules for a class to follow

enum

CoffeeSize.java

```
enum CoffeeSize {  
    //constants  
    SMALL, MEDIUM, LARGE;  
}
```

Enum for constant values

Applications Start in main()

- ◆ The 'main' method should be defined exactly as follows:
 - Everything in Java is case sensitive!
 - Whitespace does not matter
- ◆ The name of variable "args" is up to you
- ◆ The code within the main method will be executed
 - Program exits once all the lines of code in main have been executed
 - Unless a separate thread was launched by application

```
package com.roi.hello;

public class HelloWorld {

    public static void main(String[] args) {
        // the obligatory first program
        System.out.println("Hello World!");
    }
}
```



OOD Refresher

- ◆ Java is an object-oriented programming language
- ◆ What do you know about OOD/OOA principles?
 - Discuss within your group and report to the class
 - Make sure everyone within your group understands the concepts
 - Use internet to discover interesting facts
 - ◆ Pros & cons
 - ◆ What other OOP languages are out there
 - ◆ What are the alternatives to OOP
- ◆ You have 20 minutes to complete the exercise

Object Think

- ◆ Objects represent important “things” in the problem domain
 - Contain data
 - Provide functionality
 - Each object has unique identity
- ◆ Objects have responsibilities
 - To maintain the integrity of their data
 - Control access by the rest of the system to that data
- ◆ Objects collaborate to accomplish goals
 - By requesting other objects to perform an operation

The PIE Principle

- ◆ Three fundamental OO concepts
- ◆ Encapsulation
 - Each object protects its data
 - Does not allow direct manipulation of its data
- ◆ Inheritance
 - A parent class defines common data and behavior
 - Child classes inherit the parent features
 - ◆ And can extend or override these features
- ◆ Polymorphism
 - Child class can override a parent class method
 - Child class method will be called when referenced through a parent class reference
 - Aka “run-time binding”



Classes

- ◆ An OO class defines an object type
 - Describes what it means to be an object of this type
 - Defines data members
 - Defines methods (behavior)
- ◆ Java and C# require all behavior to be defined in a class
 - C++ allows functions to be defined outside of classes
- ◆ A class is like a template for creating objects
 - It is the “cookie cutter”
 - Objects are the cookies
- ◆ Each class has to be responsible for one thing and one thing only
 - Known as ‘single responsibility principle’



Objects

- ◆ Objects are instances of a class
 - The “cookies” created from a “cookie cutter”
- ◆ Each object contains its own set of data
 - And is responsible for maintaining it
 - And encapsulating it
- ◆ Every object of a given class type provides the same behavior
 - Implements the same set of methods
 - Operating on its own set of data

Java Demo



- ◆ Let's create our first application – a simple Bank Account class
- ◆ Instructor will demo IDE for you and guide you through creating Account class with a single method – deposit(int amount)



Exercise: Java

- ◆ Finish up the class by implementing withdraw(int amount) and getBalance () methods
- ◆ Use 'main ()' method to test your code
- ◆ Once you finish, review each other's code and provide constructive feedback
 - Ask questions and explore more as time allows if you are new to Java
 - If you are stuck, ask someone within your group or the instructor
- ◆ You have 30 minutes to complete the exercise



Java Demo Follow Up

- ❖ What property does the Account class encapsulates?
- ❖ What can break the encapsulation?

Chapter Concepts

Java Introduction



Unit Tests

Inheritance and Polymorphism

Handling Exceptions

Interfaces

Collections

Chapter Summary

Different Views of Testing

- ◆ Acceptance testing verifies implementation of user stories
 - Indicate whether software is broken
 - Help build the **right code**
 - ◆ What the customer wants
- ◆ Unit and integration testing verifies individual components
 - Indicate where software is broken
 - Help build the **code right**
 - ◆ From a developer perspective
 - Maintainable, extensible

Overall Aim of Testing Code

- ◆ Test-driven development is often aiming for test-first code
 - Real aim is **self-testing code**
 - We should always be able to test the code
- ◆ Not always possible to write tests for code before writing the code
 - In these scenarios, it's OK to write the tests after
- ◆ Most important thing is to test EARLY
 - As soon as it is practical to do so

Good Unit Tests

- ◆ Are **fully automated**; i.e., write code to test code
- ◆ Offer good coverage of the code under test, **including discontinuities and error-handling paths**
- ◆ Express the intent of the code under test – they do more than just check it
 - Behavioral specifications, where functionality is illustrated by example
- ◆ Should focus on and express the interface contract, not the private implementation
- ◆ Should aim to demonstrate and test **one** behavior only
 - The behavior should be clear from the test name
 - ◆ E.g., 'depositPositiveAmount', 'overdrawAccountWithZeroInitialBalance'
 - ◆ **not** 'test1', 'test2' or simply 'testDeposit' or 'testWithdraw'
 - The behavior should be reflected in the assertions in the test
 - But there may be more than one actual assertion within the test

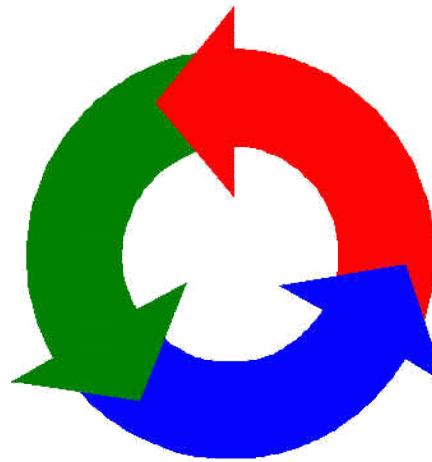
Problematic Tests

- ◆ Problematic test styles include:
 - *Monolithic tests*: all test cases in a single method; e.g., *Test* or *Main*
 - *Ad hoc tests*: test cases arbitrarily scattered across test functions; e.g., *Test1*, *Test2*, ...
 - *Procedural tests*: test cases bundled into a test method that correspond to target method; e.g., *testFoo* tests *foo*

TDD Cycle

Green

**Write enough
code to pass
the test**



Red

**Write a failing
test for a new
feature**

Refactor

**Simplify, consolidate, and
generalize the code**

What Tests to Write

- ◆ There are a number of things that should appear in tests
 - Simple cases, because you have to start somewhere
 - Common cases, using equivalence partitioning to identify representatives
 - Boundary cases, testing at and around
 - Contractual error cases; i.e., test rainy-day as well as happy-day scenarios
- ◆ There are a number of things that should not appear in tests
 - Do not assert on behaviors that are standard for the platform or language—the tests should be on your code
 - Do not assert on implementation specifics
 - ◆ A comparison may return 1 but test for > 0
 - Or incidental presentation—spacing, spelling, etc.

Structure of a Test

- ◆ A test case should have linear flow: arrange, act, assert
 - *Given*: declare and set up data
 - *When*: perform the action to be tested
 - *Then*: assert desired outcome
- ◆ Note that these sections may vary depending on what is being tested
 - But tests should rarely lack assertions
 - ◆ Assert-less tests are likely to be very brittle!

```
public class AccountTest {  
    @Test  
    public void depositPositiveAmount() {  
        Account a = new Account(100);  
        a.deposit(10);  
        assertEquals(110, a.getBalance());  
    }  
}
```

Given

When

Then

Fixtures, setUp, and tearDown

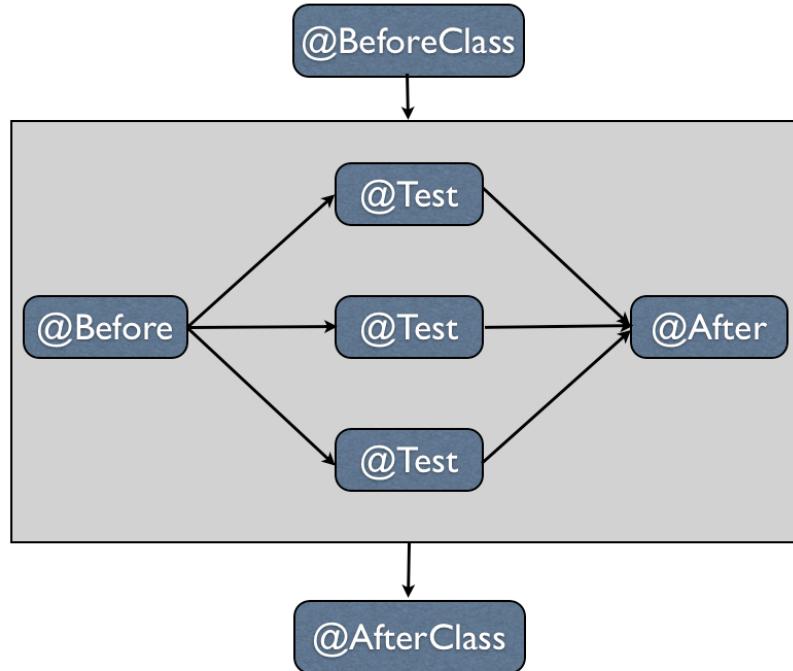
- ◆ Fields can be used to share fixture code
- ◆ Can mark initialization methods with @Before to create objects, files, database connections
- ◆ Can mark disposal methods with @After and use it to close files, database connections, etc.
- ◆ Commonly, these methods are named setUp and tearDown()

```
public class FullTimeEmployeeTest {  
    FullTimeEmployee employee;  
  
    @Before  
    public void setup() {  
        employee =  
            new FullTimeEmployee("Bob", new BigDecimal(10_000.00));  
    }  
  
    @Test  
    public void testBelowAverage() {  
        employee.setPerformanceReview(PerformanceReview.BELOW);  
        BigDecimal expected = new BigDecimal(1000.00);  
        BigDecimal actual = employee.calculateRaise();  
  
        assertEquals("The calculated raise is not correct",  
                    expected, actual);  
    }  
}
```

Class-Wide Setup

- ◆ Sometimes, need to do a common setup for all tests
 - No need to setup before each individual test method
- ◆ A method marked **@BeforeClass**
 - Run once before any tests are run
- ◆ A method marked **@AfterClass**
 - Run once after all tests have completed

JUnit Test Lifecycle





Unit Test Demo

- Let's create a couple of unit tests for our simple Bank Account class
- Instructor will guide you through one TDD cycle



Exercise: Unit Test

- ◆ Create more unit tests for other cases
- ◆ Once you finish, review each other's code and provide constructive feedback
 - Ask questions and explore more as time allows if you are new to Java
 - If you are stuck, ask someone within your group or the instructor
- ◆ You have 30 minutes to complete the exercise
- ◆ When done, research and discuss within the group the meaning of '@Before', '@After', '@BeforeClass' and '@AfterClass' annotations

Chapter Concepts

Java Introduction

Unit Tests

➤ **Inheritance and Polymorphism**

Handling Exceptions

Interfaces

Collections

Chapter Summary

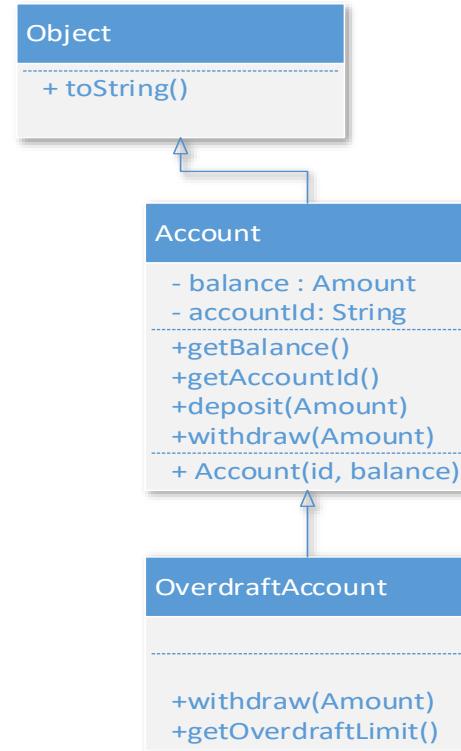
Java Syntax for Inheritance

- The `extends` keyword signifies inheritance in Java

```
3 public class OverdraftAccount extends Account {  
4 }
```

- By default, the superclass is `java.lang.Object`

```
3 public class Account extends Object {  
4 }
```



Inherited Methods

- Subclasses inherit methods and variables from the superclass, if:
 - They are **public** or **protected** (even if in different package)
 - They are **package** (default) and in same package

```
3 public class Account extends Object {  
4  
5     protected int accountID;  
6     double balance;
```

- Users can call the superclass methods on the subclass object
 - As if it was defined and implemented in the subclass

```
24 public static void main(String[] args) {  
25     OverdraftAccount overdraftAccount = new OverdraftAccount(12345, 4298.67);  
26     double amount = overdraftAccount.getBalance();  
27     System.out.println(amount);  
28 }
```

@Override

- ◆ Java 5 provides *annotations*
 - Meta information about the code
 - `@Override` is an annotation
- ◆ If `@Override` is provided:
 - The compiler will check to make sure that the method really does have the same signature
 - Prevents subtle maintenance problems later on

```
3 public class OverdraftAccount extends Account {  
4     @Override  
5     public boolean withdraw(double amount){  
6         balance -= amount;  
7         return true;  
8     }
```

In Java, “final” Means That Not to Be Changed

- ◆ A final field is a constant: once set, it cannot be changed
- ◆ A final method cannot be overridden
- ◆ A final class cannot be subclassed

```
3  public final class AccountOwner {  
4      private final String name;  
5      private final boolean taxable;  
6      public AccountOwner(String name, boolean taxable){  
7          this.name = name;  
8          this.taxable = taxable;  
9      }  
10     public final String getName() { // redundant (why?)  
11         return name;  
12     }  
13     public final boolean isTaxable() {  
14         return taxable;  
15     }  
16     // No setter methods: (why?)  
17 }
```

super.

- ◆ Overriding replaces the entire method
 - Use “super” to obtain the superclass’ implementation of the method
- ◆ Here, the OverdraftAccount reverts to Account behavior if the account has been overdrawn more than 10 times:

```
3 public class OverdraftAccount extends Account {  
4     private int numTimesOverdrawn;  
5     private static final int MAX_TIMES_OVERDRAWN = 10;  
6  
7     @Override  
8     public boolean withdraw(double amount) {  
9         if ( numTimesOverdrawn >= MAX_TIMES_OVERDRAWN ){  
10             return super.withdraw(amount);  
11         }  
12         balance -= amount;  
13         if ( balance < 0 ){  
14             numTimesOverdrawn++;  
15         }  
16         return true;  
17     }  
18 }
```

static final

- ◆ Note the use of **static** and **final** here:

```
public class OverdraftAccount extends Account {  
    private int numTimesOverdrawn;  
    private static final int MAX_TIMES_OVERDRAWN = 10;  
}
```

- ◆ The `numTimesOverdrawn` is an *instance* variable
 - This means that different `OverdraftAccount` objects have different values for this variable
 - John's account may have been overdrawn four times
 - Susan's account may have been overdrawn two times
- ◆ The `MAX_TIMES_OVERDRAWN` is a *class* variable
 - All `OverdraftAccount` objects have the same value for this variable
 - Denoted by marking the field as being *static*
- ◆ The `MAX_TIMES_OVERDRAWN` is also a constant (will always be 10)
 - Denoted by marking the field as being *final*

super()

- ◆ The `super` keyword is also used to call superclass' constructor
 - Has to be first line of constructor

```
public class Account {  
    public Account(String owner, double balance) {  
        // init Account fields  
    }  
}
```

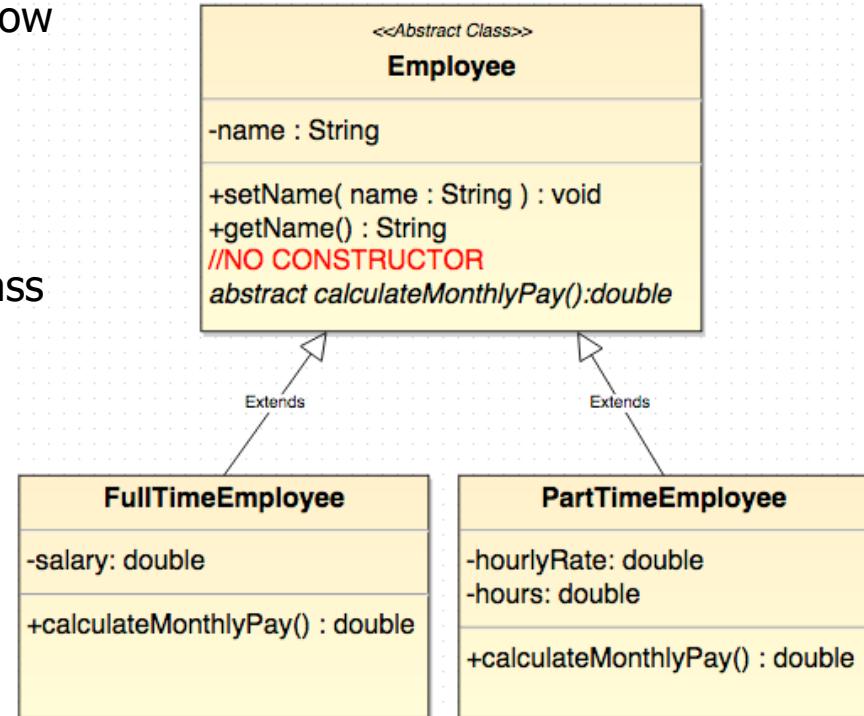
```
public class OverdraftAccount extends Account {  
    public OverdraftAccount(String owner, double balance, int numTimesOverdrawn) {  
        super(owner, balance);  
        // init OverdraftAccount fields  
    }  
}
```

- ◆ If call to `super()` is omitted, compiler will insert call
 - Calls constructor with no parameters

Abstract Methods

- ◆ Sometimes the superclass may not know the implementation of a method
 - But needs to specify that subclasses will know
 - Such methods are abstract
- ◆ A class with one or more abstract methods is itself abstract
- ◆ You cannot create instances of an abstract class

```
//This will NOT COMPILE  
Employee emp1 = new Employee();
```



Inheriting from an Abstract Class

- ◆ If you subclass an abstract class, you have two choices
 - Implement the abstract method
 - Syntax just like overriding an inherited method
 - Write a method with exact same signature
 - Optionally, add `@Override` annotation
 - Make the subclass also abstract
 - ◆ You must implement all the abstract methods
 - Otherwise, subclass is itself abstract

```
//If subclasses provide implementations, these are all okay:  
Employee emp1 = new PartTimeEmployee();  
Employee emp2 = new PartTimeEmployee();  
FullTimeEmployee emp3 = new FullTimeEmployee();  
PartTimeEmployee emp4 = new PartTimeEmployee();
```

Why Abstract Methods?

- ◆ Abstract methods lead to extensible code
 - Allow programmers to use different types of Employee
 - Don't need to know the exact type of Employee
- ◆ If a new type of Employee is added:
 - Implementer provides a `calculateMonthlyPay()` method
 - Existing code continues to work

```
Employee myEmployee = getEmployeeSomehow();  
// could return any subclass  
  
double monthlyPay = myEmployee.calculateMonthlyPay();
```

Inheritance Demo



- ◆ Instructor will now show how to extend a base class Account and create a child class CheckingAccount
- ◆ Note that we can still refer to the created object as 'Account' (parent type) – this is because of polymorphism



Inheritance Example

- ◆ Create two more subclasses – SavingAccount and OverdraftAccount
 - In SavingAccount class, override 'withdraw' method to collect a transaction fee
 - ◆ 2% of the amount being withdrawn OR \$1 – whichever is greater
 - ◆ Fee amount should be rounded to the nearest dollar and stored in the base Account class
 - ◆ Deducted from the balance immediately
 - ◆ Add 'getCollectedFees ()' method to the Account class to return the accumulated fee
 - In OverdraftAccount class, override 'withdraw' method to allow negative balance
 - ◆ Define maximum allowed negative balance amount via 'public static final' field
- ◆ Enhance your test suite with new unit tests
 - Add test cases first and implementation second
- ◆ Create new types of account from 'main' method
- ◆ You have 30 minutes to complete the exercise

Chapter Concepts

Java Introduction

Unit Tests

Inheritance and Polymorphism

➤ **Handling Exceptions**

Interfaces

Collections

Chapter Summary

What Are Exceptions?

- Exceptions are thrown when there are issues executing code
 - We would like to make sure the calling code is aware there was an issue
- What lines of code might have an issue when this code executes?

```
3 public class DoubleException {  
4     public static void main(String[] args) {  
5         displaySum(args);  
6     }  
7  
8     private static void displaySum(String[] args) {  
9         double d1 = new Double(args[0]);  
10        double d2 = new Double(args[1]);  
11  
12        System.out.println("These numbers sum to " + (d1+d2));  
13    }  
14 }
```

When Expecting Numerical Inputs...

- ◆ If no parameters passed in: `ArrayIndexOutOfBoundsException`

```
<terminated> DoubleException [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_25.jdk/Contents/Hon  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0  
    at com.demo._2using_classes.DoubleException.displaySum(DoubleException.java:9)  
    at com.demo._2using_classes.DoubleException.main(DoubleException.java:5)
```

- ◆ If invalid parameters passed in: `NumberFormatException`

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "abcd"  
    at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1241)  
    at java.lang.Double.valueOf(Double.java:504)  
    at java.lang.Double.<init>(Double.java:597)  
    at com.demo._2using_classes.DoubleException.displaySum(DoubleException.java:9)  
    at com.demo._2using_classes.DoubleException.main(DoubleException.java:5)
```

Stack Trace

- ◆ The stack trace shows the complete stack trace
 - Source file and line number
 - Shows chained exceptions if one exception was used to create another
- ◆ In Eclipse, line numbers of method calls are hyperlinks

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "abcd"
  at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1241)
  at java.lang.Double.valueOf(Double.java:504)
  at java.lang.Double.<init>(Double.java:597)
  at com.demo._2using_classes.DoubleException.displaySum(DoubleException.java:9)
  at com.demo._2using_classes.DoubleException.main(DoubleException.java:5)
```

- ◆ What can we do to prevent the program from crashing with a stack trace message?

try and catch

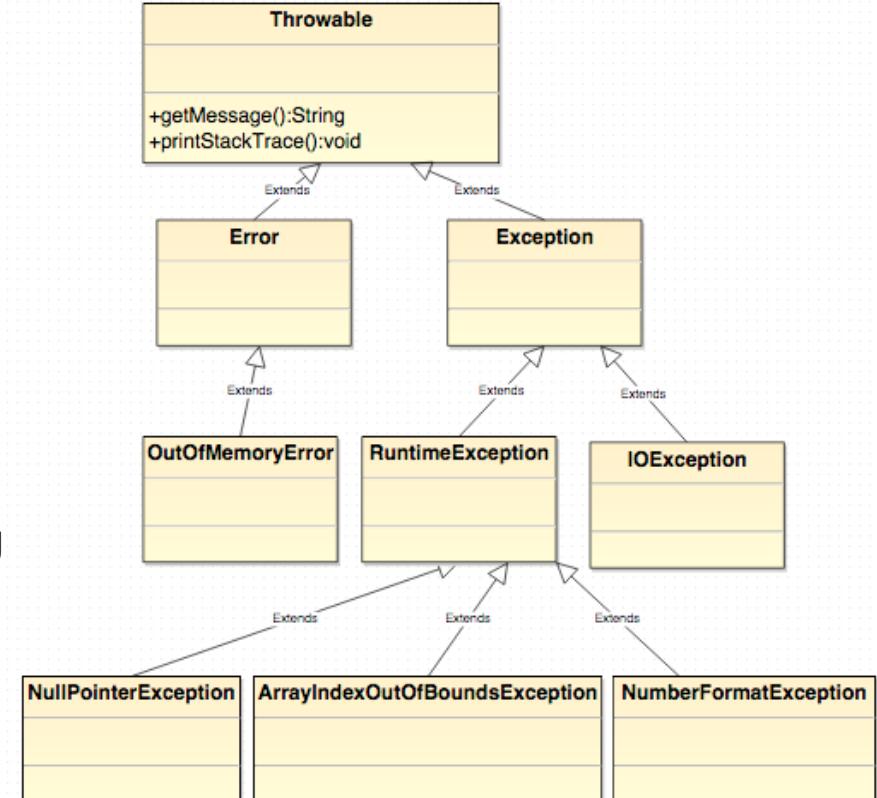
- ◆ If an exception might occur, can surround code with a “try-catch” block
- ◆ If an exception is thrown on line 10 or 11:
 - Execution jumps to line 14 of the catch block, then back to line 6
 - If no exception, the statements in the catch block are not executed

```
3  public class DoubleExceptionTryCatchDemo {  
4      public static void main(String[] args) {  
5          displaySum(args);  
6      }  
7  
8      private static void displaySum(String[] args) {  
9          try {  
10              double d1 = new Double(args[0]);  
11              double d2 = new Double(args[0]);  
12              System.out.println("These numbers sum to " + (d1+d2));  
13          }  
14          catch(Exception e) {  
15              System.out.println("Either no parameter passed in, or " +  
16                              "non numeric data was passed in");  
17          }  
18      }  
19  }
```

What exceptions will
be handled by this
catch block?

Exception Hierarchy

- ❖ Using `catch (Exception e)` will catch all exceptions with `Exception` as a superclass
- ❖ Specific Exception types help make for specific actions or messages
`catch (NumberFormatException e)`
- ❖ Helpful methods inherited from `Throwable`
 - `getMessage()`
 - ↳ Readable explanation of what went wrong
 - ↳ *For input string: "abcd"*
 - `printStackTrace()`
 - ↳ Lists complete set of calls, with line numbers that led to error
 - ↳ Extremely useful for debugging



RuntimeException

- ◆ With the code we have written so far, we were not forced to handle exceptions
- ◆ Errors due to logic flaws typically throw a `RuntimeException`
 - Do not need to be declared
 - Do not need to be caught
 - If thrown, will travel up the call stack to whoever catches it
- ◆ Examples of `RuntimeException`:
 - Arithmetic overflow
 - Calling methods on a null pointer reference
- ◆ In future chapters, we will look at checked exceptions
 - Enforced by the compiler
 - Such as with `IOException` and `SQLException`

Multiple Catch Blocks

- ◆ We can use multiple catch blocks to take specific actions
 - Put in reverse hierarchical order—the first matching catch block handles the exception

```
3 public class DoubleExceptionMultipleCatches {
4     public static void main(String[] args) throws Exception {
5         addNumbers(args);
6     }
7
8     private static void addNumbers(String[] args) throws Exception {
9         try {
10             double d1 = new Double(args[0]);
11             System.out.println("d1 is " + d1);
12         }
13         catch(NumberFormatException e) {
14             System.out.println("Invalid Usage:" + e.getMessage());
15             System.out.println("Please pass in double values");
16         }
17         catch(RuntimeException e) {
18             System.out.println("Caught a runtime exception");
19         }
20         catch(Exception e) {
21             System.out.println("Caught a high level exception");
22         }
23     }
24 }
```

Try-Catch-Finally

- ◆ You can make sure that code executes whether an exception is thrown or not by using a finally block

```
3 public class DoubleExceptionTryCatch {  
4     public static void main(String[] args) {  
5         try {  
6             double d1 = new Double(args[0]);  
7             System.out.println("d1 is " + d1);  
8         }  
9         catch(Exception e) {  
10             e.printStackTrace();  
11         }  
12         finally {  
13             System.out.println("\nNo matter what, I execute");  
14         }  
15     }  
16 }
```

Rules on Using the Try Block

◆ Rules in using the try block

- Can be followed by a single catch block
- Can use multiple catch blocks for different exceptions
- Can use just a finally block (no catch blocks)
- Can use one/more catch blocks and a finally block

```
public void printMatches(String searchTerm) throws IOException {  
    int counter = 0;  
    try{  
        // do stuff, loop, increment counter  
    } catch (IllegalArgumentException e){  
        System.err.println(e);  
    } catch (Exception e2){  
        e2.printStackTrace();  
    }  
    finally{  
        System.out.println("Wanted to print "+ counter + " + regardless of exception");  
    }  
}
```

Throwing an Exception

- ◆ Can generate a new Exception by using the keyword **throw**
- ◆ When choosing to throw an exception, the method signature must be updated to indicate it throws an exception
- ◆ If you catch one type of exception, you can throw it, or create a new exception using the caught exception

```
3 public class ThrowNewExceptionDemo {  
4     public static void main(String[] args) throws Exception {  
5         String name = "", message = "";  
6  
7         if (args.length < 2) {  
8             throw new IllegalArgumentException(  
9                     "You must pass in name and age");  
10        }  
11  
12        message = "Hello, " + name + ". " +  
13                generateAgeMessage(args[1]);  
14        System.out.println(message);  
15    }  
16  
17    private static String generateAgeMessage(String ageString)  
18                      throws Exception {  
19        String returnMessage = "";  
20        int convertedAge = 0;  
21        try {  
22            convertedAge = Integer.parseInt(ageString);  
23        }  
24        catch(Exception e){  
25            throw new IllegalArgumentException(  
26                    "2nd argument was not a valid number", e);  
27        }  
28        // If no exception, code continue to run....
```

Best Practice: When to Throw an Exception

- ◆ Throwing an exception should indicate a serious error condition
 - That the method cannot handle itself
 - So “throw” the problem back to the calling method
- ◆ Code could be called from another program or code
 - Exceptions can help the calling code determine an alternative route
 - To try with different inputs or different resources
 - E.g., if file, database, service is not available
- ◆ Catch an exception as soon as it can be handled correctly
 - More context information is available as we go up the call stack

Creating a Custom Exception

- ◆ Creating a custom exception is easy
 - Eclipse will do most of the work for you
- ◆ A custom Exception is a Java class
 - That extends either Exception or RuntimeException
- ◆ Often may want to catch a standard exception (such as SQLException)
 - And then wrap it inside an application specific (custom) exception
 - Nesting exceptions preserves the stack trace information

A Custom Exception

```
public class PersistenceException extends RuntimeException {  
  
    public PersistenceException(String message, Throwable cause) {  
        super(message, cause);  
        // TODO Auto-generated constructor stub  
    }  
  
    public PersistenceException(String message) {  
        super(message);  
        // TODO Auto-generated constructor stub  
    }  
  
    ...  
}
```

Testing for Exceptions

- ◆ **Do not** catch exceptions in your test code when testing for 'happy path' scenarios
 - If exception is thrown, JUnit will **fail** the test, which is desired behavior
- ◆ **Do** test for 'expected' exceptions when testing for 'alternative flows':
 - If expected exception is thrown, JUnit will **pass** the test
 - JUnit will **fail** test if exception does **not** occur

```
public class CrmTest {  
    @Test(expected = ClientDuplicateException.class)  
    public void addDuplicateClient() {  
        crm.addClient(JOHN_SMITH);  
        crm.addClient(KATE_SMITH);  
        crm.addClient(JOHN_DOE);  
        crm.addClient(JOHN_DOE);  
    }  
}
```

```
public class CRM {  
    ...  
    public void addClient(Client client) {  
        if (contains(client))  
            throw new ClientDuplicateException(  
                "Client ["+client+"] already exists");  
        clients.add(client);  
    }  
}
```

Trying to add client
that already exists



Exceptions Demo

- ◆ Instructor will now demonstrate:
 - How to use standard `IllegalArgumentException` to prevent depositing negative amount
 - How to create custom exception
 - How to test for an exception
- ◆ Note that we can still refer to the created object as 'Account' (parent type) – this is because of polymorphism



Exercise: Handling Exceptions Example

- ◆ Using existing codebase with three types of accounts (Overdraft, Checking, Saving)
 - Implement new business rules on all types of accounts (by throwing an exception)
 - ◆ Do not allow to create accounts with negative balance
 - ◆ Do not allow to set withdraw more than `MAX_WITHDRAWL_LIMIT` (defined as static field on the account class)
 - ◆ Do not allow to overdraw any account but Overdraft account
 - ◆ Validate that the amount that is being withdrawn or deposited is a positive number
 - Add any other validations you feel needed
 - Try to create custom 'business' exceptions for handling the error conditions
- ◆ Follow a TDD approach when extending functionality
 - Add test cases first and implementation second
- ◆ You have 60 minutes to complete the exercise

Chapter Concepts

Java Introduction

Unit Tests

Inheritance and Polymorphism

Handling Exceptions

Interfaces

Collections

Chapter Summary

Interfaces

- ◆ Interface is like a 'contract' which any Java class can fulfill
- ◆ Interfaces specify what a class must do and not how
 - Therefore, the methods declared in interface are abstract and do not have body with exception of 'default' implementations
 - No instance variables (fields)
 - ◆ Public static final fields are allowed

```
public interface Animal {  
    void makeSound();  
    int getNumLegs();  
}
```

- ◆ Any class that wishes to implement an interface must:
 - Implement *all* the abstract methods of the interface
 - OR: declare itself as abstract

Multiple Interfaces

- Interfaces are special
 - A class can *extend* only one class
 - A class may *implement* many interfaces

```
public class Lion extends Object implements Animal, Comparable {  
    public void makeSound() {  
        // code here  
    }  
    public int getNumLegs() {  
        // code here  
    }  
    public int compareTo(Object other) {  
        // code here  
    }  
}
```

Standard Interfaces

- From the last slide, we saw **java.lang.Comparable**
 - Allows an object to be compared to another
 - Declares a single method:

```
public interface Comparable {  
    public int compareTo(Object other);  
}
```

- Implement the interface and provide the method body on your class
 - Used extensively in the Collections API as we'll see later

Default Methods

- ◆ Interfaces can now define default methods
 - And static methods
- ◆ Why would we want to do this?
 - It is a convenient and powerful way to extend an interface
 - Without breaking existing clients

Extending an Interface

- ◆ Consider this interface
 - Which has been implemented by several classes
- ◆ What to do if we need to now add a new method to this interface?
 - And avoid the wrath of all the programmers whose class implements it

```
// example from Oracle JavaDocs
public interface TimeClient {
    void setTime(int hour, int minute, int second);

    void setDate(int day, int month, int year);

    void setDateAndTime(int day, int month, int year, int hour,
                        int minute, int second);

    LocalDateTime getLocalDateTime();
}
```

Using Default Methods

- ◆ Add a static or default method

```
public interface TimeClient {  
    // methods as before  
    static ZoneId getZoneId(String zoneString) {  
        try {  
            return ZoneId.of(zoneString);  
        } catch (DateTimeException e) {  
            System.err.println("Invalid time zone: " + zoneString  
                + "; using default time zone instead.");  
            return ZoneId.systemDefault();  
        }  
    }  
  
    default ZonedDateTime getZonedDateTime(String zoneString) {  
        return ZonedDateTime.of(getLocalDateTime(), getZoneId(zoneString));  
    }  
}
```

Interface or Abstract Class?

- ◆ How to decide whether to use an interface or an abstract class
- ◆ Abstract classes can contain concrete method definitions
- ◆ Interfaces are more flexible
 - A class can implement multiple interfaces
 - But extend only one base class



Introducing Interfaces

- ◆ Instructor will now:
 - Show you the examples of interfaces
 - Demonstrate how to create and implement an interface



Exercise: Introducing Interfaces

- ◆ Currency account
 - Add this interface to your project

```
public interface CurrencyAccount {  
    void deposit(int amount, Currency currency);  
}
```

- Only CheckingAccount **and** OverdraftAccount should accept foreign currency deposits
- For the sake of simplifying this exercise, we make the following assumptions:
 - ↳ The balance field for our account represents an amount in U.S. Dollars
 - ↳ The accounts should accept deposits for Euros (EUR) with FX rate: 0.89
 - ↳ The accounts should accept deposits for Pounds Sterling (GBP) with FX rate: 0.77
- ◆ Your finished code should be fully tested and look something like this:

```
Currency eur = Currency.getInstance("EUR");  
CurrencyAccount a = new CheckingsAccount(); // balance starts $0  
a.deposit(100, eur); // balance now $89
```

Chapter Concepts

Java Introduction

Unit Tests

Inheritance and Polymorphism

Handling Exceptions

Interfaces

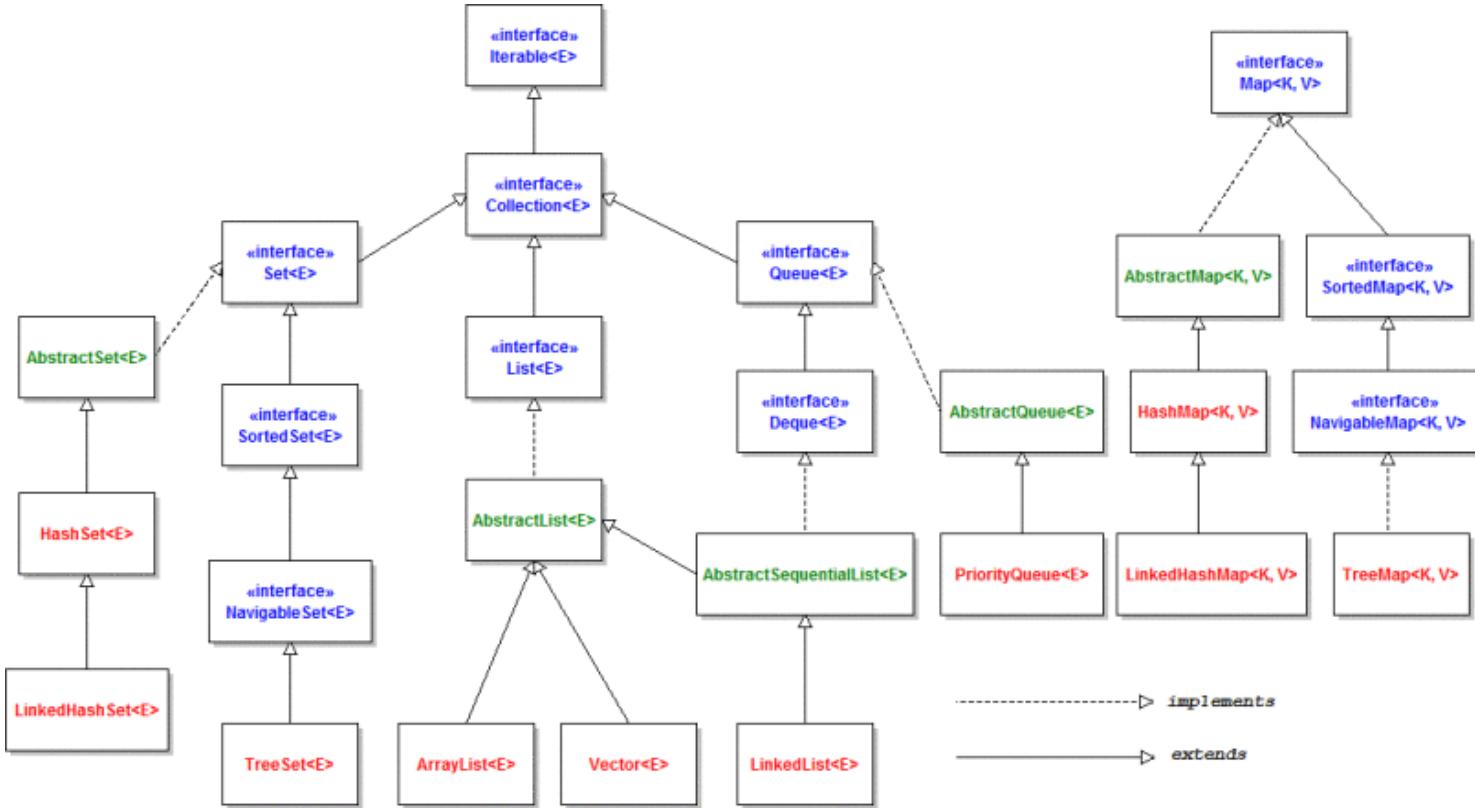


Chapter Summary

Collections

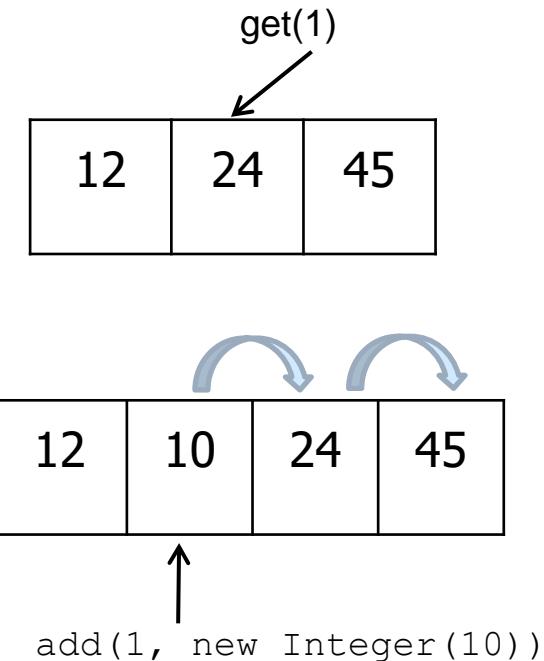
- ◆ Java provides a framework that allows to store and manipulate the group of objects
- ◆ There are also three generic types of collection
 - Ordered lists
 - Dictionaries/maps
 - Sets
- ◆ Java Collection framework provides
 - Many interfaces
 - ◆ Set, List, Queue, Deque, etc.
 - And classes
 - ◆ ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet, etc.

Java Collection APIs



ArrayList

- ◆ The most commonly used collection is ArrayList
 - A list of items stored in the order they are added
 - Grows as items are added
 - No need to know size beforehand
- ◆ Useful methods of ArrayList:
 - `size()` gives you current size of ArrayList
 - `get(i)` gets you the ith item in ArrayList
 - ↳ The index starts at zero
 - `add(Object o)` takes the item to add
 - ↳ Adds to the end of ArrayList
 - `add(int index, Object object)` allows insertion in the middle of the list
 - `remove(Object o)` – removed 1st occurrence of object



ArrayList: Example

```
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class ArrayListInteger {
7
8     public static void main(String args[]){
9         // create empty array list
10        List<Integer> employeeNumberList = new ArrayList<Integer>();
11
12        //populate
13        employeeNumberList.add(new Integer(10));
14        employeeNumberList.add(new Integer(20));
15        employeeNumberList.add(0,new Integer(30));
16        employeeNumberList.add(new Integer(40));
17        employeeNumberList.add(1,new Integer(50));
18
19        //iterate
20        for (int index = 0; index < employeeNumberList.size(); index++) {
21            Integer employeeNumber = employeeNumberList.get(index);
22            System.out.println(employeeNumber);
23        }
24    }
25 }
```

Console

```
<terminated> ArrayListInt
30
50
10
20
40
```

HashMap

- ◆ HashMap is the most commonly used map
 - TreeMap is much less common; it stores data in sort order
 - Each Entry in a Map is a pair
 - ◆ Key
 - ◆ Value
- ◆ Useful methods of HashMap:
 - put() allows you to add items to the map
 - get() allows you to obtain items to the map
 - ◆ Is actually a search operation
- ◆ HashMaps commonly used to cache data
 - Such as from the database or files

Using HashMap: An Example

- Employee is the key; Phone is the value

```
// create empty map
Map<Employee, Phone> directory = new HashMap<Employee, Phone>() ;

// add items to map (database pseudocode)
while ( dataAccessor.hasMoreRecords() ) {
    Employee employee = dataAccessor.getEmployee();
    Phone phone      = dataAccessor.getPhone();
    directory.put(employee, phone);
}

// search for the Phone for a particular employee
Employee janitor = ...;
Phone janitorPhone = directory.get(janitor) ;
```

Iterator

- ◆ An iterator object is optimized for a collection
 - Calling `iterator()` on a Collection returns an Iterator optimized for that Collection

```
27     Iterator<String> iterator = myCollection.iterator();
28     while(iterator.hasNext()){
29         System.out.println(iterator.next());
30     }
```

- ◆ The `hasNext()` method is used to check there are still items to iterate over
- ◆ The `next()` returns the next reference
- ◆ The `remove()` method allows the caller to remove elements during the iteration
 - some collections will throw an `UnsupportedOperationException`
- ◆ Some iterators returned by Collections have more methods
 - List iterator can go backwards

for-each Notation

- ◆ If don't need to remove items, or go backwards, can use the for-each notation

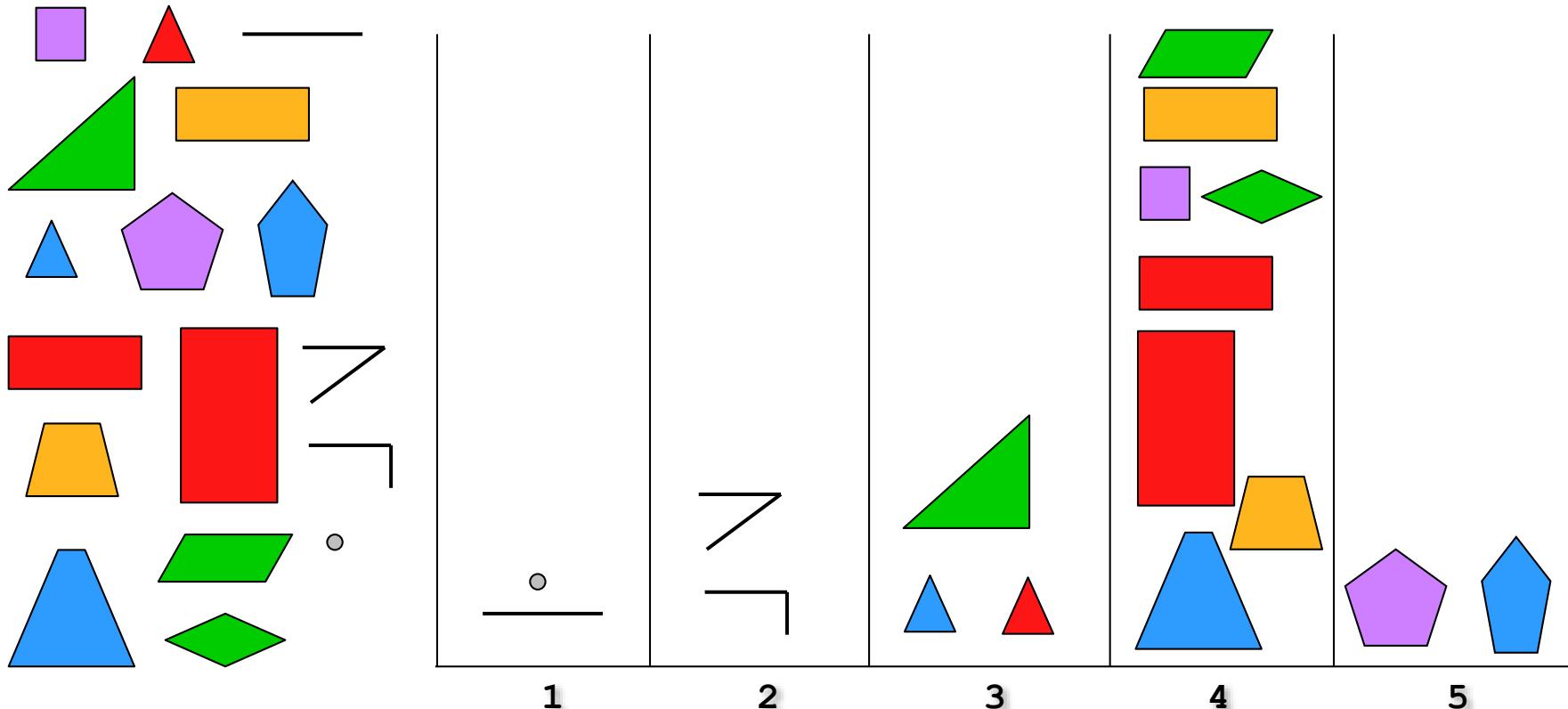
```
for (String item : myCollection) {  
    System.out.println(item);  
}
```

- ◆ Pros:
 - Greater readability, easier to write
- ◆ Cons:
 - Cannot remove items
 - Can only access single item at one time
 - Cannot traverse two collections at once
 - Only accessing, not assigning

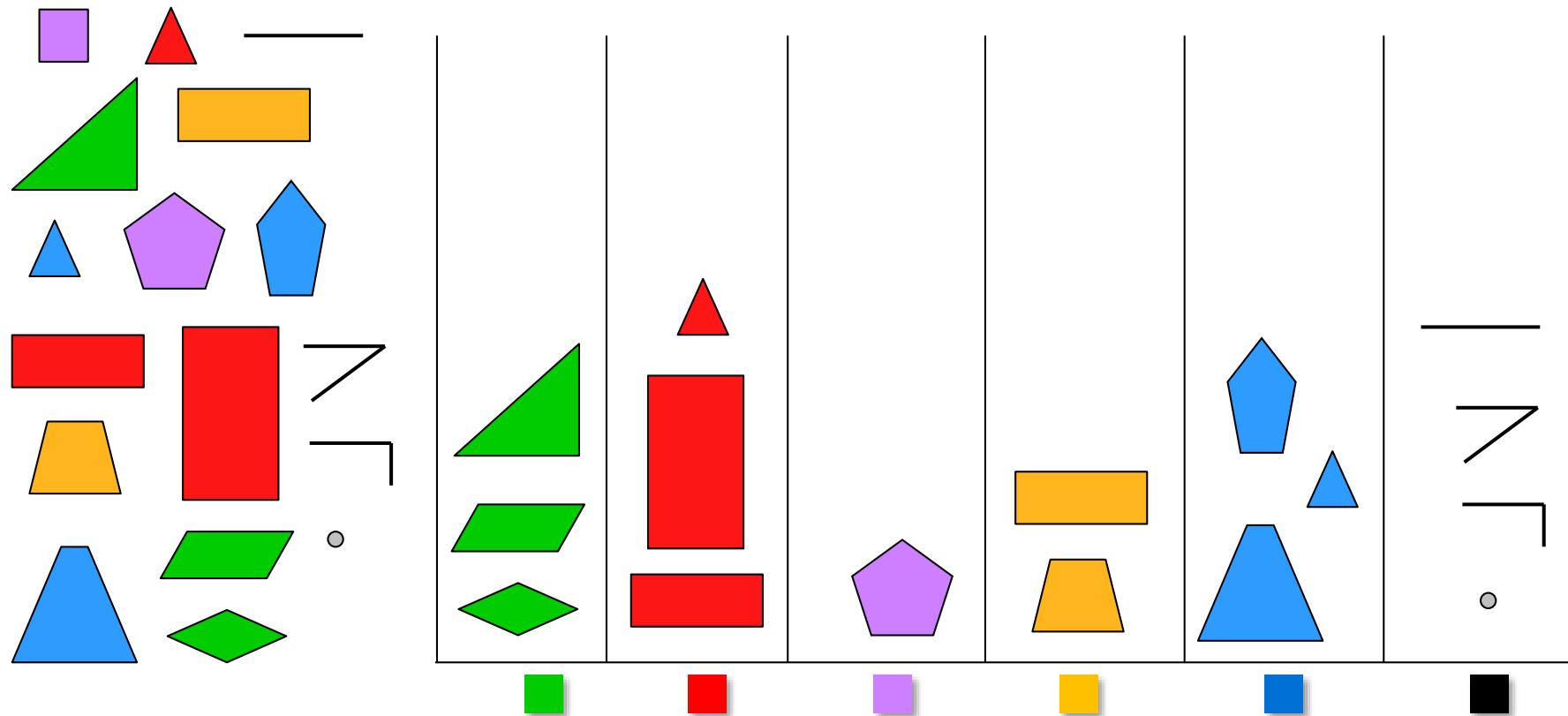
equals and hashCode Method

- ◆ When working with collections, it is important to provide proper implementation for `Object.equals` and `Object.hashCode` methods
- ◆ `boolean equals (Object obj)`
 - Default implementation returns true only if object is compared to itself
 - ◆ `x.equals(y)` same as `x == y`
- ◆ `int hashCode ()`
 - Returns a hash code value for the object
 - If two objects are equal, `hashCode` method called on each object must produce the same result
 - ◆ If two objects are NOT equal, it is allowed, though not advisable, for `hashCode` to produce the same value
 - Default `hashCode` value is the 'virtual address' of the object in memory
 - ◆ `System.identityHashCode (Object obj)`

hashCode & equals in Action



hashCode & equals in Action (continued)



hashCode and equals Implementations

- ◆ Normally, domain objects have their `hashCode` and `equals` method overridden
 - One of the reasons is domain objects are passed around a lot and manipulated as part of Collections
- ◆ IDEs provide autogenerated implementations—DO USE THEM!
- ◆ ‘equality’ of two objects is a subjective thing and depends on the domain
 - Comparable and Comparator interfaces allow to provide more than one definition of equality based on the circumstances the object is used in
- ◆ If ‘equals’ method is overridden, `hashCode` should be overridden as well in most cases
- ◆ `hashCode` calculation must be efficient and fast



Introducing Collections

- ◆ Instructor will now:
 - Show you how to create a simple collection and how to override `equals` and `hashCode` methods



Exercise: Introducing Collections

- ◆ Create a List of accounts and add a variety of different types of bank accounts to it
 - Use the new `for` loop syntax to iterate through the list and:
 - ◆ Deposit \$100 into each account
 - ◆ Withdraw \$50 from each account
 - ◆ Print out the balance of all accounts
 - Do all the accounts behave the same way?
 - Can you add EUR funds to the accounts in the loop?
- ◆ Follow a TDD approach when extending functionality
 - Make sure you enhance your test cases to cover new functionality
- ◆ Review each other's code and provide constructive feedback
- ◆ You have 45 minutes to complete the exercise

Chapter Concepts

Java Introduction

Unit Tests

Inheritance and Polymorphism

Handling Exceptions

Interfaces

Collections

➤ Chapter Summary

Chapter Summary

In this chapter, we have:

- ◆ Refreshed our knowledge of Java, its key constructs and features



ROI TRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Module 2: Global Technical Analyst Program

CHAPTER 3: BLOCKCHAIN FUNDAMENTALS

Chapter Objectives

In this chapter, we will:

- ◆ Gain an insight into how blockchains work
- ◆ Examine the types of cryptography required in a blockchain
- ◆ Build a simple blockchain system

Chapter Contents

➤ Overview

Hashing Cryptography

How It Works

Exercise

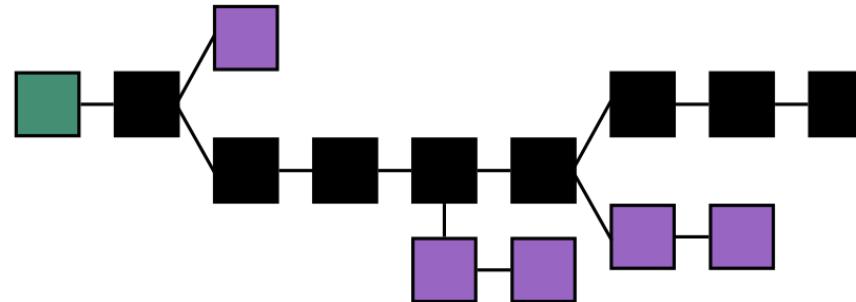
Chapter Summary

Blockchain Overview

- ◆ A growing list of records called blocks
- ◆ Blocks are linked in time order using cryptography
- ◆ Data in the blocks cannot be changed once encrypted
- ◆ Can be implemented as simple stand-alone system to ensure data integrity
- ◆ Can be implemented as a distributed environment across many nodes
 - Data is visible to all the nodes
 - Can be replicated by the nodes
 - Individual nodes can add records to the chain
 - Most common application: distributed ledger

Blockchain Overview (continued)

- ❖ First conceptualized by Satoshi Nakamoto in the design for Bitcoin
 - Used to encode transactions
 - Provides an audit trail of all transactions
 - Bitcoin protocol is distributed by design
 - ◆ The chain is visible to all nodes in the network
 - ◆ No need for a central trusted server to ensure that currency is only spent once
 - ◆ Nodes form a consensus to validate blocks (longest chain wins)
 - ◆ Resolves the double spend problem



Chapter Contents

Overview

➤ **Hashing Cryptography**

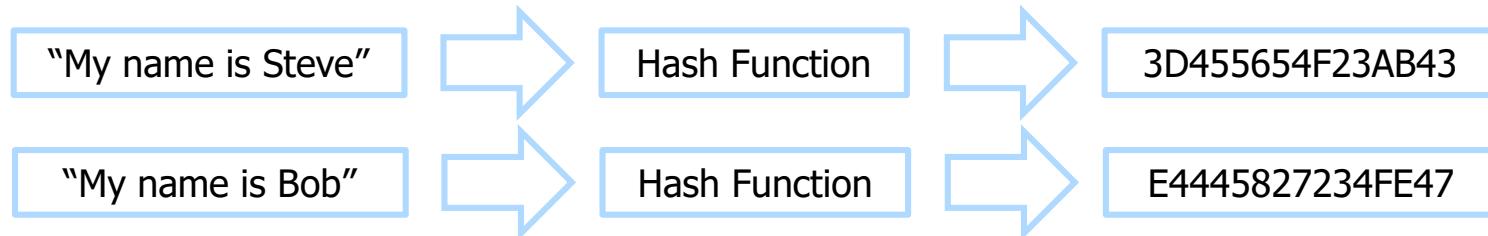
How It Works

Exercise

Chapter Summary

What Is a Hash function?

- ◆ A function that takes an arbitrary size data and returns a fixed size code
 - The result is called the hash value or digest
 - Impossible to calculate the input value from the hash value “one way”
 - Extremely hard to find alternate input values that form the same hash
- ◆ Hashes are central to blockchains
 - Used to digitally “sign” the data
- ◆ Commonly used hash functions:
 - MD5 – password encryption, large document signatures
 - SHA256 – SSL, Bitcoin





Creating a Digest

- ❖ Your instructor will now demonstrate the use of a hashing function
 - Examine the resulting digest for some simple String inputs
 - See the Java API for SHA256 encoding

Chapter Contents

Overview

Hashing Cryptography

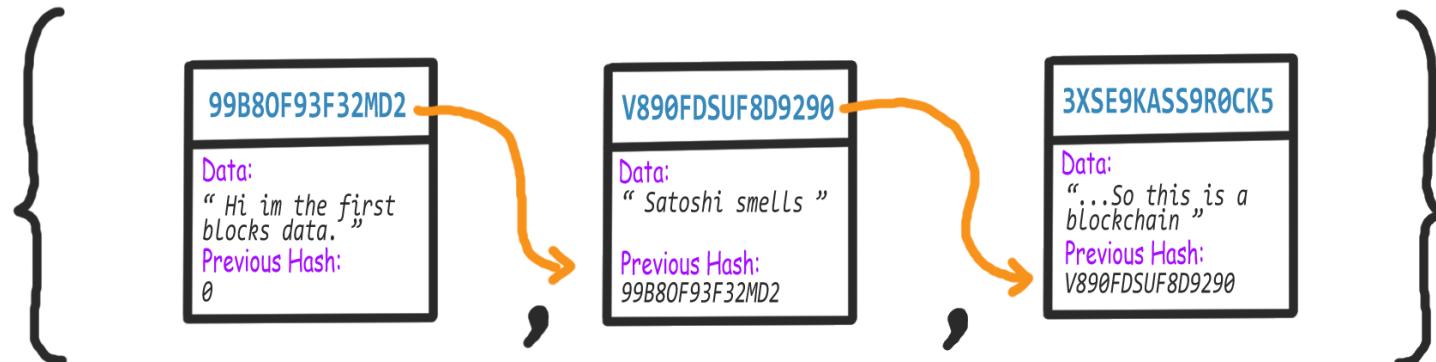
How It Works

Exercise

Chapter Summary

Hashing Blocks

- Each block in a chain contains a **hash value** and some payload **data**
 - The hash for a new block is calculated by hashing:
 - The hash of the previous block
 - And the new block's payload data
 - The chain can be validated at any time by recalculating the hashes
 - Any changes to data will result in a different result for the hash, breaking the chain



- The first block (the genesis block) has an arbitrary hash value – zero in this case

Compromising Data in the Chain

- ◆ An attacker could change the data in one of the nodes if they:
 - Know the hashing algorithm
 - Have access to the entire chain
- ◆ How could this be achieved?
 - Change the data in the node
 - Rehash the node that has been modified to update its hash digest
 - Rehash **EVERY** subsequent node to update all the dependent hash digests
- ◆ Risk can be alleviated by:
 - Replicating the chain between peers in a distributed implementation
 - Impossible to change them all
 - Requiring a **proof of work** when encoding the hashes
 - A key innovation of Bitcoin algorithm

Proof of Work

- ◆ Calculating a SHA256 hash is a relatively cheap and fast process
 - Not possible to predetermine the value given in the hash digest
- ◆ If it were harder to generate a digest, it would be harder to rehash blocks in the chain
- ◆ Limit the range of permitted digest values
 - Keep it low
 - Must start with n zeros
- ◆ The digest becomes much more difficult to calculate
 - Must use a trial and error approach
 - Add a generated **nonce** value to the encrypted data until a low enough digest is found
 - Requires much more processing power
 - Slower = harder to abuse

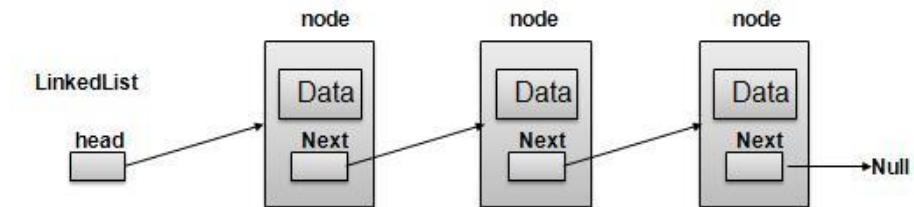
Proof of Work for Bitcoin

- ◆ How does this affect Bitcoin?
 - Massive computing resources required to generate enough digests
 - ↳ Digests are “mined” using dedicated processors on the network
 - ↳ Nodes are paid for each digest they mine
 - Data centers exist with dedicated mining machines
 - Using application specific integrated circuits (ASICs)
 - In regions with cheap electricity

Related Design Pattern: Linked List

- ◆ A blockchain could be implemented as a one-way linked list
- ◆ Objects in the chain are linked to other objects of the same class
- ◆ Methods called on the head of the chain can be delegated **recursively** along the list
 - Head-first recursion
 - Tail-first recursion

```
public void printChain() {  
    if (next != null){  
        next.printChain();  
    }  
    System.out.println(data);  
}
```



Chapter Contents

Overview

Hashing Cryptography

How It Works



Exercise

Chapter Summary



Implementing a Blockchain—I

- ◆ Inspect the code for the HashGenerator class
 - Complete the HashGeneratorTest unit test methods
- ◆ Examine the Block class and note its use of the LinkedList pattern
 - Are there any recursive methods?
 - Do they use head-first or tail-first recursion?
- ◆ In the main method of the BlockChainMain class:
 - Create a blockchain with the following variable names
 - ◆ genesisNode -> secondNode -> thirdNode
 - ◆ You may pass any string data you like into these nodes
 - ◆ Which node is the head of the chain?
 - ◆ Call the printChain method on the head of the chain
 - You should see the data for all nodes printed out



Implementing a Blockchain—II

- ❖ In the Block class constructor:
 - Enable block encoding by uncommenting the line and run your program again
 - Spend some time understanding what this line does!
- ❖ Add a recursive method to the Block class with the following signature:
 - public void validateChain() throws BlockChainException { ... }
 - Make use of the HashGenerator class to complete the implementation
 - Test your method by invoking it on the head of the chain from the main method
 - ◆ It should not generate any errors
 - Add a line of code that changes the data in secondBlock and run the program again
 - ◆ It should now generate an error as the data has been tampered with
 - Add unit tests to ensure that the correct behavior is asserted
- ❖ Experiment with the hash difficulty level
 - What do you feel is a suitable level for our blockchain?

Chapter Contents

Overview

Hashing Cryptography

How It Works

Exercise

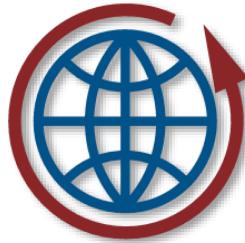


Chapter Summary

Chapter Summary

In this chapter, we have:

- ◆ Gained an insight into how blockchains work
- ◆ Examined the types of cryptography required in a blockchain
- ◆ Built a simple blockchain system



ROI TRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Module 2: Global Technical Analyst Program

CHAPTER 4: INTRODUCING MAPREDUCE WITH HADOOP

Chapter Objectives

In this chapter, we will:

- ◆ Describe Big Data
- ◆ Introduce MapReduce
- ◆ Implement MapReduce with Linux commands
- ◆ Present the benefits and issues of using MapReduce on a cluster
- ◆ Give an overview of the Hadoop framework

Chapter Concepts



About Big Data

Introducing MapReduce

MapReduce with Linux

MapReduce on a Cluster

Introducing Hadoop

Running Hadoop

Installing Hadoop

Chapter Summary

What Is Big Data?

Big data is high-volume, high-velocity, and/or high-variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization

—Doug Laney, Gartner

Volume

- Result of data being continuously gathered
- Terabytes rather than Gigabytes

Velocity

- Require continuously updated analysis

Variety

- Not suitable for traditional databases
- Unstructured text, images, video, citation graphs...

Information for Business Decisions

- ❖ In an ever-changing, complicated environment, businesses seek information that helps them make the best decisions
- ❖ One of the tools traditionally used for obtaining information is statistics
 - Data samples are selected that represent a larger population
 - Information obtained from the data sample potentially provides insight into the larger population
 - ❖ But there is always an uncertainty involved
 - With Big Data, the tools exist to directly analyze the entire population

Big Data Use Cases

- ◆ Big Data is used for:
 - Natural language processing (NLP)
 - Voice recognition, sentiment analysis
 - Image processing
 - Used in self-driving cars
 - Security, facial recognition
 - Medical
 - Stock market trends
 - TCP/IP data for intrusion detection
 - Money laundering detection
 - The Internet of Things (IoT)
 - Monitoring devices for repair or replacement
 - Recommender engines
- ◆ If you were a telephone company, streaming content provider, or social website:
 - Would you have access to large amounts of data?
 - What information could you mine with that data?

Big Data Ecosystem

- ◆ The Big Data Ecosystem is very large
 - In this course, we are focusing on three popular technologies:
 - ◆ Apache Hadoop, Apache Hive, and Apache Spark
- ◆ Big Data can be organized in a couple of key ways
 - Computation vs. storage
 - Batch vs. real time
- ◆ Computation has traditionally centered on MapReduce
 - Popular MapReduce frameworks include Hadoop and Spark
- ◆ Big Data is not usually structured like a 3rd normal form relational model
 - NoSQL technologies, such as Cassandra and MongoDB, provide alternatives to the relational databases for storing massive amounts of data
- ◆ Hadoop performs its computation in batch mode
- ◆ Industry is moving more towards near real-time processing
 - Tools like Apache Spark, Apache Storm, and Apache Flink support this

Chapter Concepts

About Big Data

➤ **Introducing MapReduce**

MapReduce with Linux

MapReduce on a Cluster

Introducing Hadoop

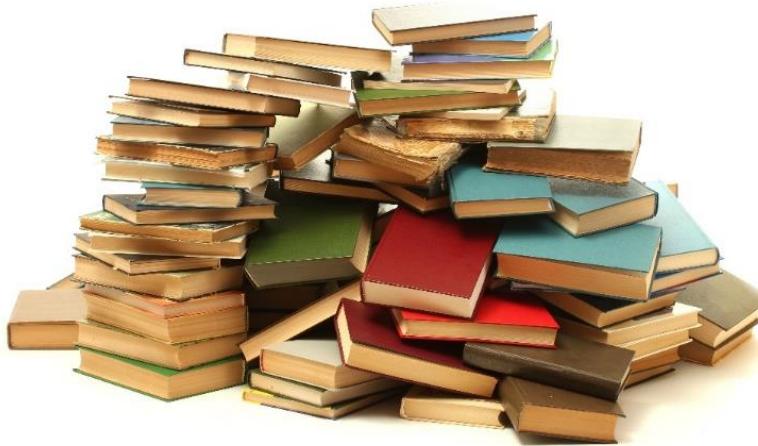
Running Hadoop

Installing Hadoop

Chapter Summary

What Is MapReduce?

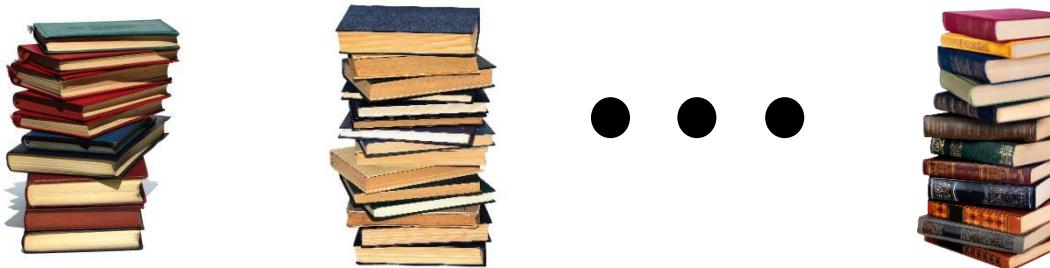
- ◆ You are given a pile of books:



- ◆ You are asked to produce a list of the book's publishers
- ◆ How would you go about solving the problem?

The List of Book Publishers

- ◆ One possible approach would be to create smaller piles of books sorted by publisher:



- ◆ The same approach could be used if the requirement was to produce a list of:
 - Authors
 - Publishing Dates
 - Page Counts
 - Subject Matter (Fiction, History, Cookbooks, Health, etc.)
- ◆ Using the above approach, what other information could be mined from the pile of books?
 - One of our goals is to find useful information in the data available to us

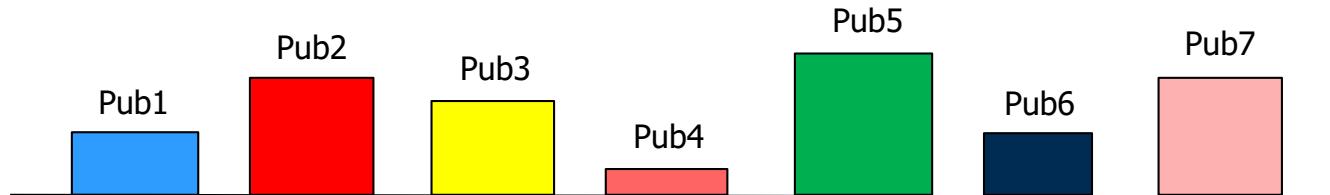
Mapping—I

- ◆ The process of transforming the original pile of books into smaller piles is called mapping:

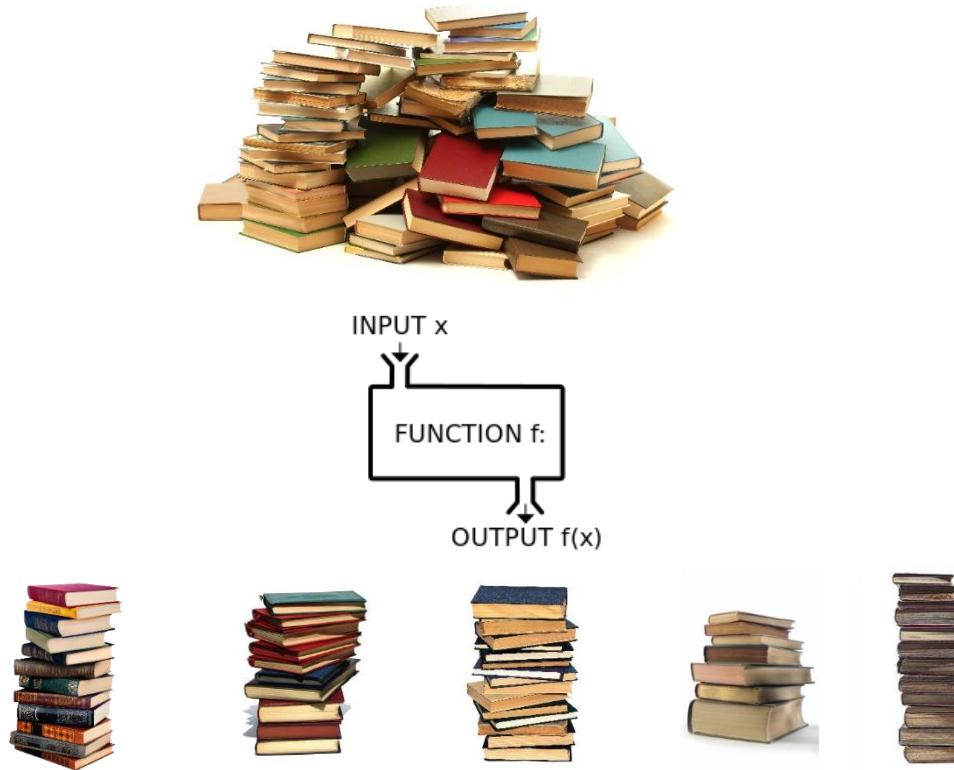


Mapping—II

- ◆ In mathematics, the term **mapping**, usually shortened to **map**, refers to either a:
 - **Function**, often with some sort of special structure, or a
 - **Morphism**, in category theory, which generalizes the idea of a function
 - [https://en.wikipedia.org/wiki/Map_\(mathematics\)](https://en.wikipedia.org/wiki/Map_(mathematics))
- ◆ In mathematics, a **function** is a relation between a set of inputs and a set of permissible outputs with the property that each input is related to exactly one output
 - [https://en.wikipedia.org/wiki/Function_\(mathematics\)](https://en.wikipedia.org/wiki/Function_(mathematics))
- ◆ Mapping can also be viewed as a filtering operation
 - Input items can be placed (filtered) into different **buckets**, sets, collections, queues, etc.
 - ◆ For example, one bucket for each publisher
 - ◆ Items can also be entirely eliminated from further processing

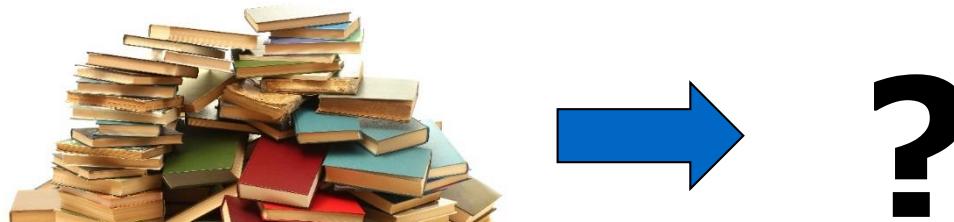


Mapping—III



Counting

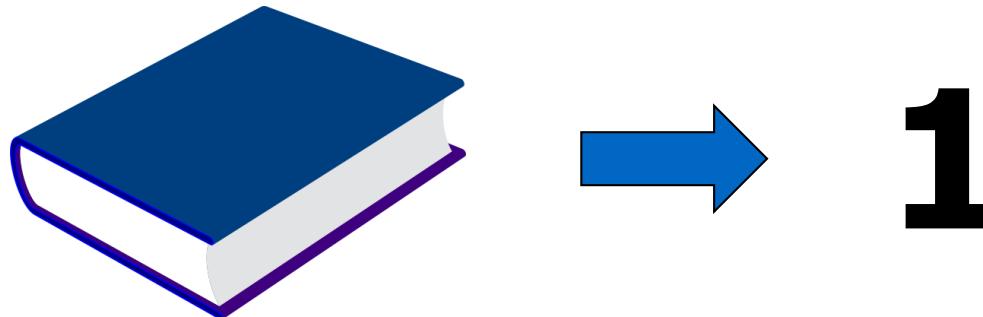
- ❖ If the requirement was to determine how many books were in the original pile, how would you proceed?



- ❖ Is a mapping process involved?

Counting in MapReduce

- There is a somewhat subtle mapping that we perform when we count
 - We **map** each individual item to the number '1'
 - We then add up (sum, aggregate) all the '1's

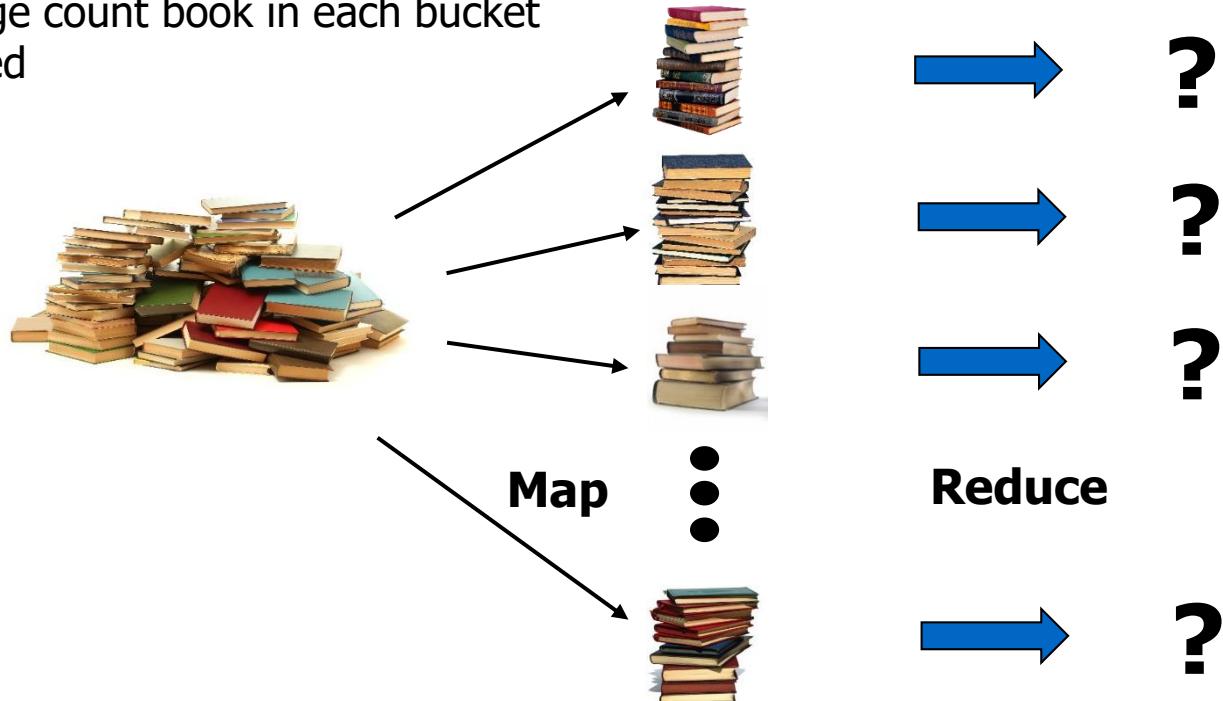


Reducing

- ◆ Adding integers is one type of **reducing** activity
 - Reducing summarizes the information that was found in the mapping operation
 - <https://en.wikipedia.org/wiki/MapReduce>
- ◆ Two other commonly used reduce operations include:
 - Maximum
 - ◆ For example, from the original pile of books, find the book with the largest page count
 - Minimum
 - ◆ For example, from the original pile of books, find the book with the earliest publishing date
- ◆ What if the requirement was to find the book with the largest page count for each publisher
 - How would you do this?

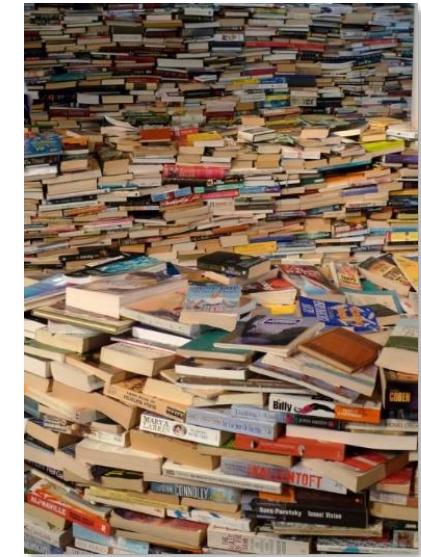
Book with Largest Page Count per Publisher

- ◆ One solution would be to map the original pile of books into publisher-specific buckets and then find the largest page count book in each bucket
 - Sorting will be involved
- ◆ How many books did we start with?
 - How many answers will there be?



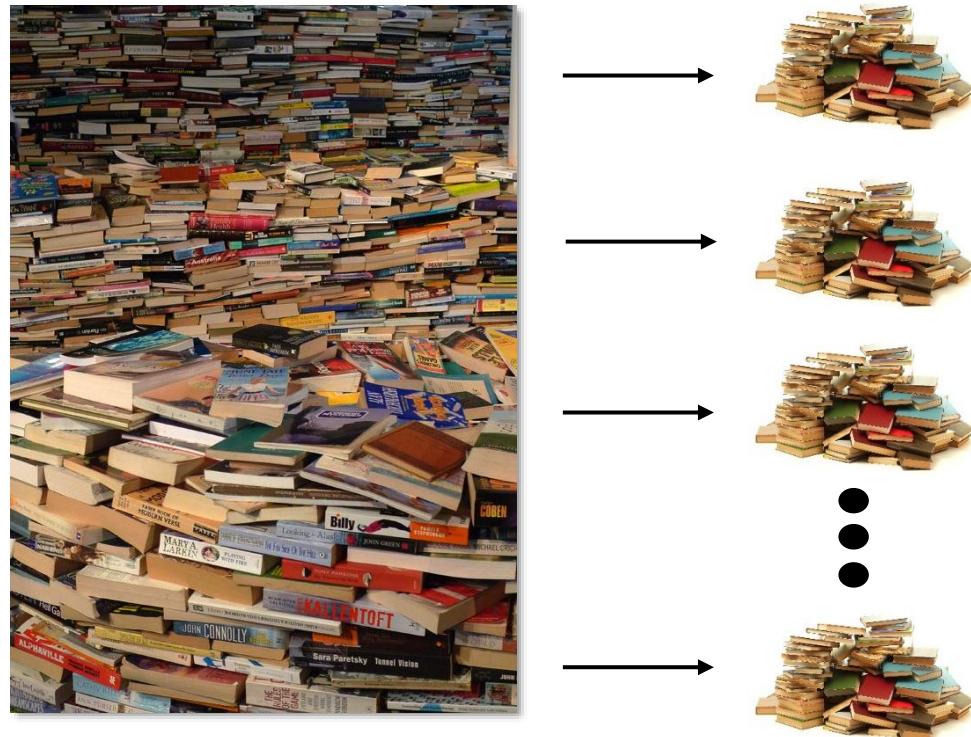
Scaling Out!—I

- ❖ How would you do MapReduce with a **huge** pile of books?
- ❖ Would you do this by yourself?
 - You could do the work faster, more efficiently, etc.
 - This is called **scaling up**
- ❖ Would you invite your friends over to help?
 - The more friends that are involved, the faster the work gets done
 - This is called **scaling out**



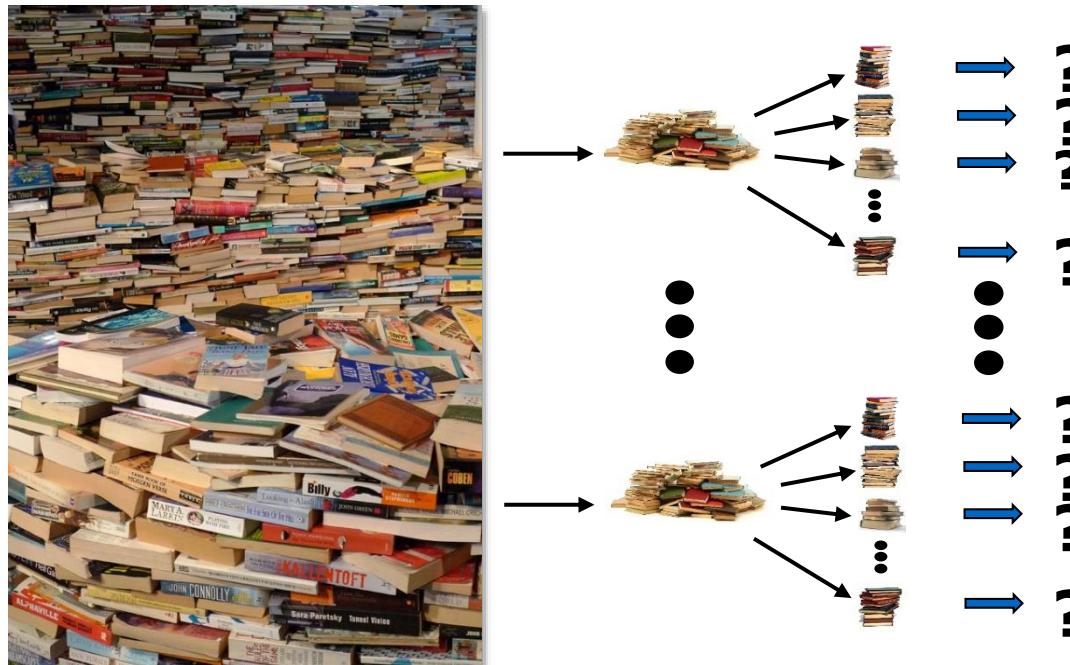
Scaling Out!—II

- ◆ If you invited your friends to help, you might start like this:
 - One smaller pile for each friend
 - They could do MapReduce with their own book pile



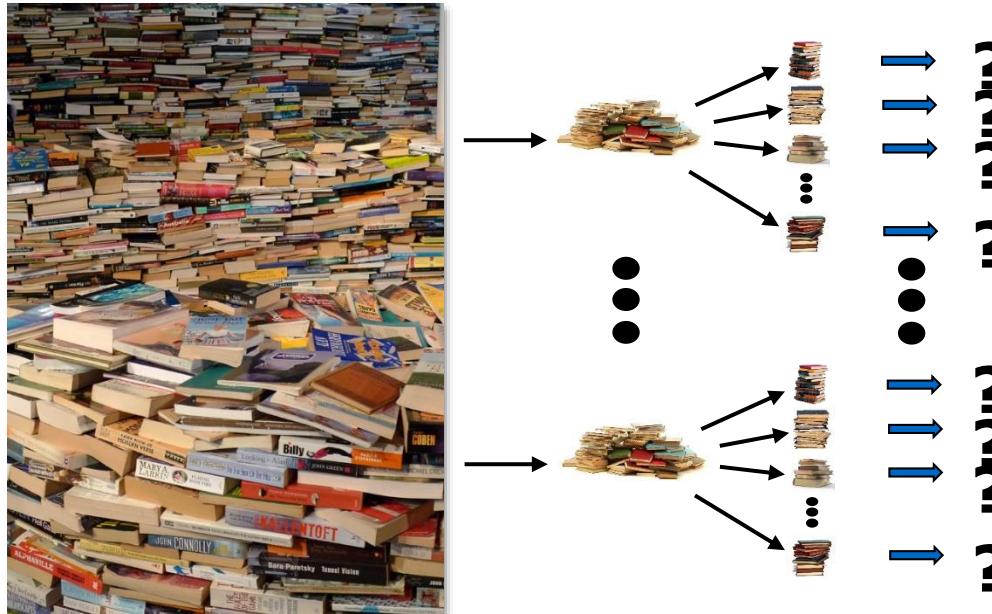
Will This Give Us the Answer?

- ♦ If each of your friends is doing MapReduce with their own pile of books, will that give us the answers we need?



Close, but Not Quite

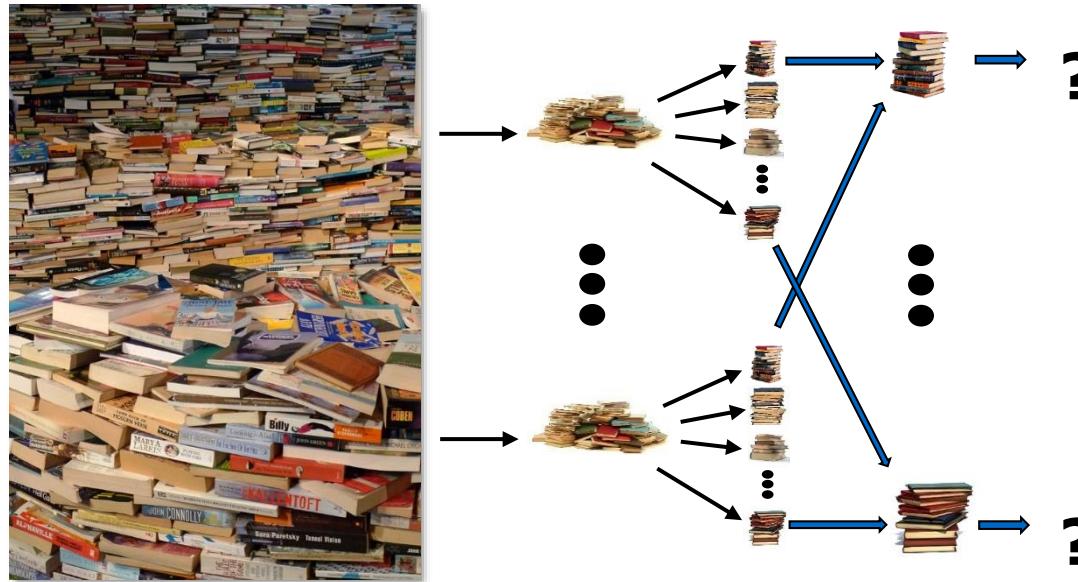
- ◆ Won't each of your friends possibly find books from the same publisher?



- ◆ How do you deal with this possibility?

Shuffle

- Have each of your friends combine their buckets with books from the same publisher—this is called shuffling



- Are there any other issues to consider?

Balancing the Merged Buckets

- ◆ It is possible that after the merge:
 - One bucket contains thousands of books
 - Another bucket is nearly empty
- ◆ We want to keep our friends equally busy
 - Avoid overloading one while another is idle
 - Keeping them equally busy results in task getting completed in the shortest amount of time
- ◆ Our goal is to find a better way to group the books
 - So that each of the buckets is equally full
 - ◆ That way, each of our friends will be equally busy
 - While still obtaining the answers to our question on the largest book per publisher
- ◆ To solve this may take some experimentation
- ◆ In MapReduce, these questions are addressed in the choice of **partitioning algorithm**

Chapter Concepts

About Big Data

Introducing MapReduce

➤ **MapReduce with Linux**

MapReduce on a Cluster

Introducing Hadoop

Running Hadoop

Installing Hadoop

Chapter Summary

MapReduce with Linux

- ◆ MapReduce is a programming model
 - Inspired by the map and reduce functions commonly used in **functional programming** (FP) languages
 - ◆ Lisp, Clojure
 - ◆ ML, OCaml
 - ◆ Haskell
 - ◆ F#
 - ◆ Also hybrid (Object-Oriented/FP) languages including Perl, Python, Java, Scala, C++, C#, etc.
 - <https://en.wikipedia.org/wiki/MapReduce>
- ◆ MapReduce can be performed on the Linux command line



MapReduce for Word Counting—I

- ◆ Search engine companies index the entire web
 - At its most basic, indexing a web page involves identifying all the words
- ◆ The following very simple example shows how it might be done with Linux tools

```
#!/bin/sh  
  
cat somefile.html | grep "some_word" | wc -l
```

- ◆ The `cat` command opens a file and outputs one data record at a time
 - It creates a stream of records (e.g., lines)
- ◆ The `grep` command
 - Reads the records on the input stream
 - **Maps** the records that contain "some_word" to a new output stream
 - The map, in this case, happens to be performing a filter operation
- ◆ The `wc -l` command
 - Reads the records on its input stream
 - Counts the records and outputs a single number, the total count
 - The `wc -l` is a **reduce** operation (specifically aggregation)

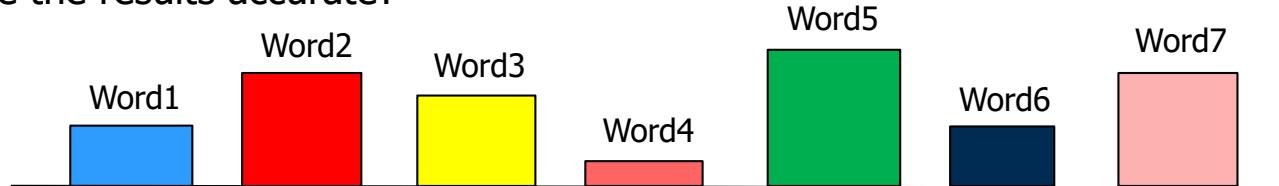


MapReduce for Word Counting—II

- ◆ What if we wanted to find out how many occurrences of each word there are in our pile of books?
 - This problem is a generalization of the previous slide
 - It is similar to grouping books by publisher
 - ◆ We group words together (in a bucket) that are the same (**map**)
 - ◆ We then count the total number of words in each bucket (**reduce**)
 - The difficulty is that Linux text processing commands are generally line oriented
 - ◆ To count words using Linux tools, we have to make sure that each word is on a separate line

```
cat ~/roi_datasets/shakespeare.txt |  
tr -s '[:punct:][:space:]' '\n' | sort | uniq -c
```

- ◆ In the above script, `word_count_1_linux.sh`, which commands are the map and reduce?
 - Is the sort command similar to a shuffle operation?
 - Are the results accurate?





MapReduce for Word Counting—III

- Here is a more refined word counting script, `word_count_2_linux.sh`
 - How many map operations are there?
 - Does this script produce better results than the previous script?
 - How could this script be improved?
 - Would it work on a cluster?

```
cat ~/roi_datasets/shakespeare.txt |  
tail -n +182 |  
head --lines=-418 |  
tr [A-Z] [a-z] |  
tr "\>:, \[\]\.\*\; \'\"?) (" " " |  
fmt -1 |  
sed '/^s*/d' |  
sed 's/^[[[:blank:]]]*//; s/[[[:blank:]]]*$//' |  
sed 's/^--//; s/\-$//; s/!$//' |  
sort |  
uniq -c
```

Career Home Runs with Linux—I



- ◆ Problem: Using the Chadwick Bureau Baseball Databank, calculate the total career home runs for Barry Bonds, Hank Aaron, and Babe Ruth
- ◆ Chadwick Bureau Baseball Databank is located here:
<https://github.com/chadwickbureau/baseballdatabank>
- ◆ The relevant file, `Batting.csv`, has been downloaded to the `~/roi_datasets/baseball` directory
- ◆ View the first 10 lines of `Batting.csv` using the following command:

```
head ~/roi_datasets/baseball/Batting.csv |
```

- ◆ The first line of `Batting.csv` is a header
 - View the description of the headers by using the following command:

```
sed -n '398,420p;421q' ~/roi_datasets/baseball/readme2014.txt
```

- ◆ What is the column header for home runs? _____
 - What is the position of the home run column? _____

Career Home Runs with Linux—II



- ◆ The first column of the Batting.csv file is playerID
 - The playerID for Barry Bonds, Hank Aaron, and Babe Ruth are:
 - ◆ Barry Bonds: bondsba01
 - ◆ Hank Aaron: aaronha01
 - ◆ Babe Ruth: ruthba01
- ◆ View the available batting statistics for each player by running the following commands:

```
grep bondsba01 ~/roi_datasets/baseball/Batting.csv
grep aaronha01 ~/roi_datasets/baseball/Batting.csv
grep ruthba01 ~/roi_datasets/baseball/Batting.csv
```

Career Home Runs with Linux—III



- ◆ Run the following script to calculate the career home runs hit by Barry Bonds:

```
cat ~/roi_datasets/baseball/Batting.csv |  
grep bondsba01 |  
awk -F, '{ s += $12 } END { print s }'
```

- ◆ Run the above command for Hank Aaron and Babe Ruth
- ◆ Compare your results with the information posted here:
 - https://en.wikipedia.org/wiki/List_of_Major_League_Baseball_career_home_run_leaders#List
- ◆ In the above script:
 - Which command is the map operation? _____
 - Which command is the reduce operation? _____

Chapter Concepts

About Big Data

Introducing MapReduce

MapReduce with Linux

➤ **MapReduce on a Cluster**

Introducing Hadoop

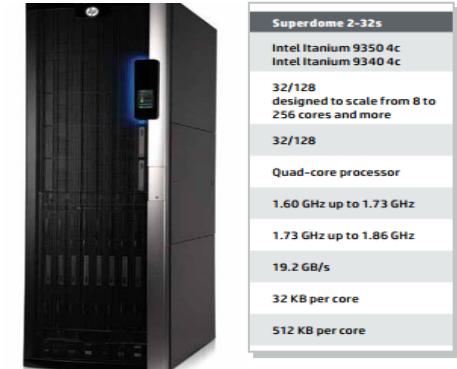
Running Hadoop

Installing Hadoop

Chapter Summary

Scalability of Single grep

- ◆ Linux commands such as `grep` process input data line-by-line
 - The more lines there are, the more processing time is required
 - Increasing I/O and CPU speeds (**scaling up**) will:
 - ◆ Provide only linear improvements
 - 10% faster I/O means that task will complete in 10% less time
 - ◆ Result in increasing costs as the hardware gets more specialized
 - Think mainframe or supercomputer
 - ◆ Eventually reach an upper limit on improvement
 - The machine is the bottleneck
- ◆ Will adding more machines (**scaling out**) help?
 - Referred to as **computer cluster**



Source: Hewlett Packard



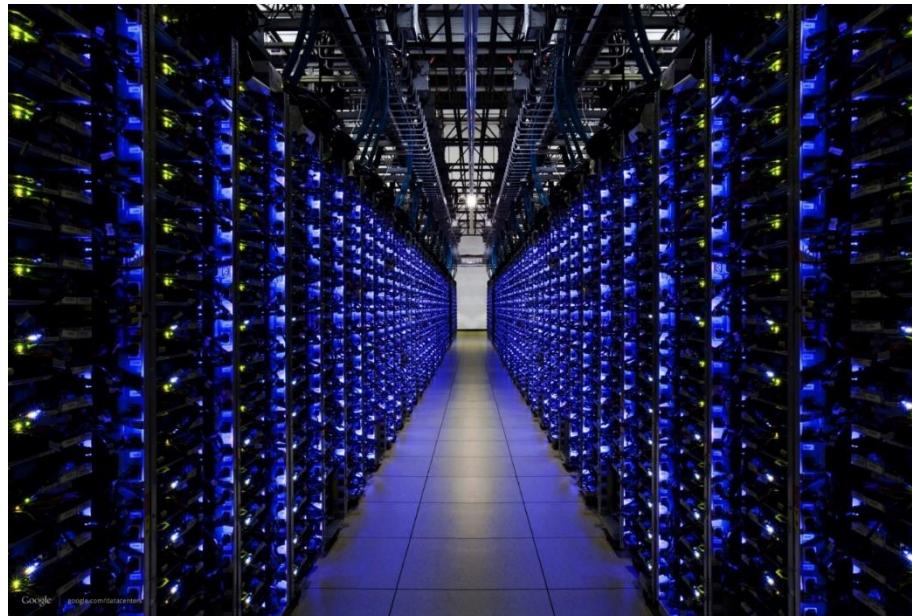
What Is a (Beowulf) Computer Cluster?

- ◆ A **Beowulf cluster** is a computer cluster of what are normally:
 - Identical, commodity-grade computers (CPU, RAM, Disk)
 - Networked into a small local area network
 - With libraries and programs installed which allow processing to be shared among them
 - In our case, the libraries will be the Hadoop framework
 - https://en.wikipedia.org/wiki/Beowulf_cluster
- ◆ The result is a high-performance parallel **computing cluster** from inexpensive personal computer hardware
 - https://en.wikipedia.org/wiki/Computer_cluster
- ◆ Conversationally, the phrase “computer cluster” normally means “Beowulf computer cluster”
 - The picture on the right is representative of an early generation Beowulf cluster



Modern Data Center

- ◆ A computer cluster in a modern data center may have anywhere from 100s to 100,000+ computers
 - The computers are commonly referred to as nodes

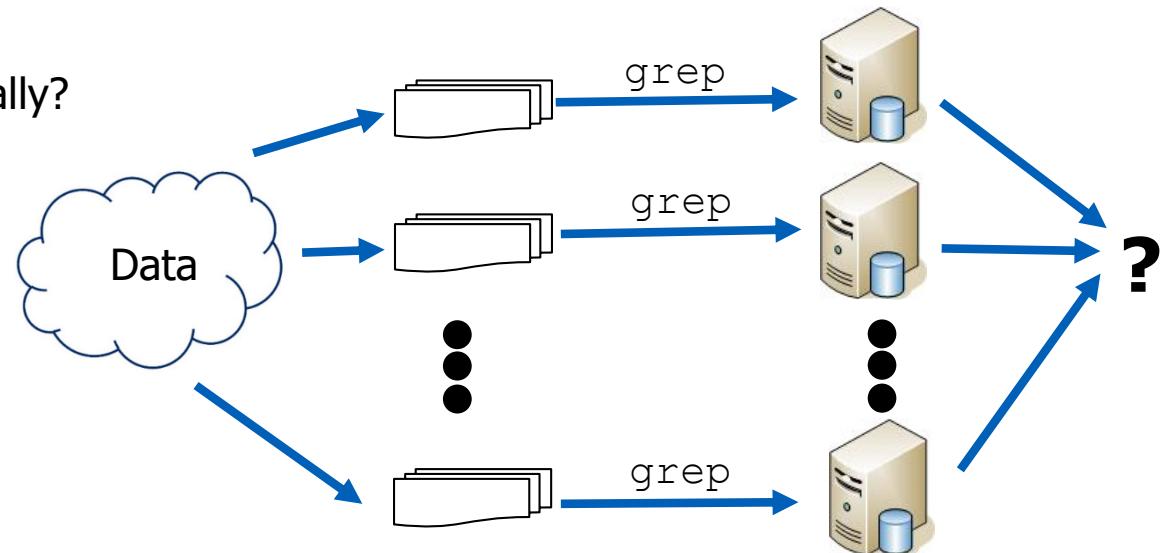


Using grep on a Computer Cluster

- ❖ Using grep on a cluster involves:

1. Splitting the text files being searched into smaller data sets
2. Copying each smaller data set to a separate node on the cluster
3. Running grep on each node that has a set of files to be searched
4. Combining the results

- ❖ Should all this be done manually?



Automating Distributed grep

- ◆ Performing the distributed grep presents several problems and issues
 - Manually launching the grep command on three computers is fine
 - What about launching grep on 100 or 1000 computers?
 - Do we really know the optimal way to split the input data?
 - How do we transfer the data between nodes so that data are local?
 - What is the best way to combine the results
 - If you are using 1000 computers, how will you know if one fails?
- ◆ For small data sets, performing a manual MapReduce by hand might be acceptable
- ◆ With large data sets and frequent executions, a framework is needed to automate as much of the process as possible

Automating Distributed grep (continued)

- ◆ This is exactly what Hadoop does
 - It splits the input data into parts and places each part on a specific computer
 - Assigns the map operation to the computers that have the data
 - Sends the results of the map to computers that will do the reduce operation
 - Recovers from any hardware or software errors
- ◆ Why do we need a framework? Can we not do this manually?

MapReduce Use Cases

- ◆ In this course, we focus on using MapReduce for word counting
 - Word counting is a simple concept and doesn't distract from the details involved using the Hadoop framework
- ◆ However, MapReduce has many use cases including the following:
 - Clustering
 - ↳ Group data that share similarities
 - ↳ Discover hierarchy and order in large data sets
 - ↳ Reveal patterns
 - Classification
 - ↳ Categorize data; e.g., spam for email, fraudulent transactions
 - ↳ Estimate how similar data is to some other data
 - Recommender engines
 - ↳ Provide recommendation based on past actions
 - ↳ Use preferences too

Chapter Concepts

About Big Data

Introducing MapReduce

MapReduce with Linux

MapReduce on a Cluster

➤ **Introducing Hadoop**

Running Hadoop

Installing Hadoop

Chapter Summary

Word Count on the World Wide Web

- ◆ The Hadoop framework arose from the need to index the World Wide Web
 - All the Internet search engine companies face the problem of indexing billions of web pages on a daily basis
 - As part of their service, the search engines report how many times a particular word appears in the entire corpus of the World Wide Web
 - ◆ This is the word count problem
- ◆ Google's solution for indexing and word counting the entire web directly inspired the Hadoop framework



Google's Solution

- ◆ Google realized that many of the data processing problems they had:
 - Dealt with extremely large amounts of data
 - Involved unstructured or semi-structured data
 - Possessed the attribute of being split into smaller jobs that could run concurrently
- ◆ The solution is made up of two components:
 1. The Google File System
 2. The MapReduce framework
- ◆ The Google File System (GFS) automatically:
 - Breaks up large data sets into smaller blocks of data
 - Stores each block on one node in the cluster
 - ◆ Optionally replicates each block to one or more alternate nodes
- ◆ The MapReduce framework executes jobs on the data stored in GFS
 - Jobs are structure into four phases
 - ◆ Map, Combine, Shuffle, and Reduce
- ◆ Google published their solutions in two research papers

The Google File System*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients.

In this paper, we present file system interface extensions designed to support distributed applications, discuss many aspects of our design, and report measurements from both micro-benchmarks and real world use.

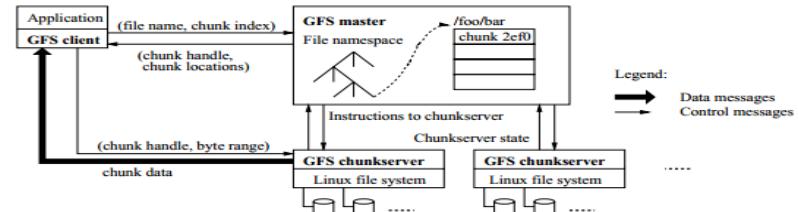


Figure 1: GFS Architecture

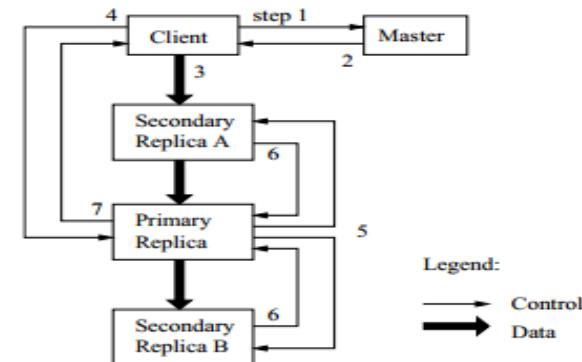


Figure 2: Write Control and Data Flow

*Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, Google, Inc.

MapReduce: Simplified Data Processing on Large Clusters*

ABSTRACT

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real-world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

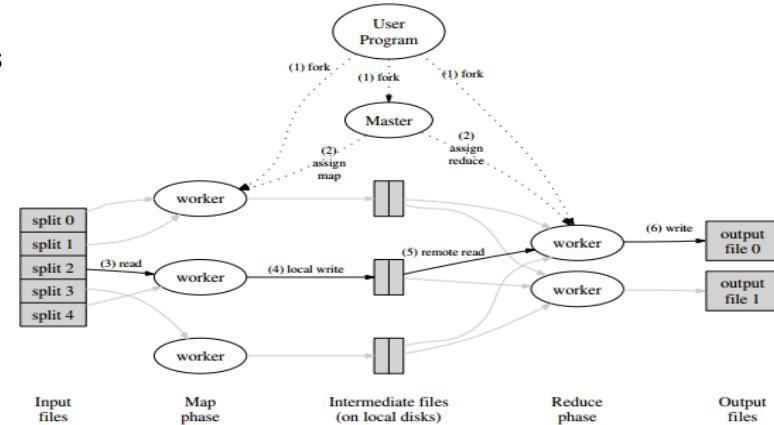


Figure 1: Execution overview

*Jeffrey Dean and Sanjay Ghemawat, Google, Inc.

What Is Hadoop?

- ◆ Apache Hadoop is an open source implementation of Google's GFS and MapReduce framework
 - Mirrors Google's design
 - Hadoop Distributed File System (HDFS) corresponds to GFS
 - Hadoop MapReduce corresponds to Google's MapReduce
 - ◆ This is true in Hadoop 1.X
 - ◆ Hadoop 2 and 3 diverge from the original Google design
- ◆ Designed to scale from single server to thousands of machines
 - Run on off-the-shelf, low-end hardware
 - Be robust to hardware and software failures
 - Be scalable to terabytes and petabytes of data
 - Be simple to program parallel applications in
- ◆ Built-in fault tolerance
 - Achieved by software
 - Failures in machine are detected and handled

Why Hadoop?

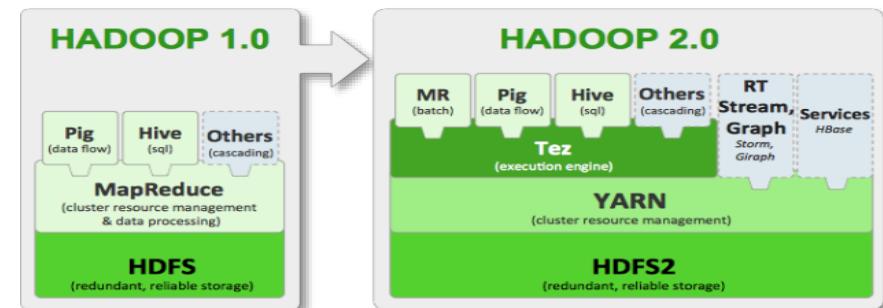
- ◆ Hadoop **changes** the **economics** of large scale computing
 1. *Scalable*: new machines can be added seamlessly
 2. *Cost effective*: very cheap, reliable storage and compute
 3. *Flexible*: data can be in different formats with no schema
 4. *Fault tolerant*: failures of storage and compute are handled dynamically and transparently to applications
- ◆ These characteristics open new opportunities for business
 - Store data that was previously discarded
 - Analyze data in a timely manner

Origins of Apache Hadoop

- ◆ Apache Hadoop was first implemented as part of the Nutch project
 - Inspired by the Google papers on GFS and MapReduce in 2003
 - Became its own project in 2006
 - Named after a toy elephant belonging to Doug Cutting's son
 - Implemented in Java
 - Initial release: December 2011
 - https://en.wikipedia.org/wiki/Apache_Hadoop
- ◆ Apache Nutch is an open source web crawler and indexer
 - A sub-project of Apache Lucene
 - Created by Doug Cutting and Mike Cafarella
 - Faced the same problems indexing the web at scale as Google, Yahoo, and others
 - https://en.wikipedia.org/wiki/Apache_Nutch
- ◆ Apache Lucene is an open-source search engine
 - Created by Doug Cutting
 - <https://en.wikipedia.org/wiki/Lucene>

Hadoop Releases

- ◆ In this course, we are using Hadoop 3.0
 - No significant structural changes but several refinements including:
 - ↳ Support for user-defined resources in addition to CPU and RAM
 - ↳ High availability for the NameNode service
 - ↳ Default ports of some services have changed
 - ↳ Intra-datanode balancing tool
 - ↳ Erasure coding in HDFS
- ◆ Major architectural change between Hadoop 1.X and 2.X
 - Hadoop 2 introduces YARN
 - Hadoop 1 closely tied to MapReduce



What Is HDFS?—I

- ◆ HDFS stands for Hadoop Distributed File System
 - A POSIX-like file system
 - POSIX: Portable Operating System Interface
 - Files can be created and deleted
 - Files can not be modified except by appending new data
 - ◆ New lines can not be inserted
 - ◆ Existing lines can not be modified or deleted
- ◆ HDFS is based on two major services:
 - NameNode
 - ◆ Responsible for maintaining file system metadata
 - Hierarchical directory structure
 - File names, permissions, timestamps, size, etc.
 - ◆ Does not hold data blocks
 - Keeps track of where all primary and back-up data blocks are stored
 - DataNode
 - ◆ Responsible for storing file data in the form of blocks

What Is HDFS?—II

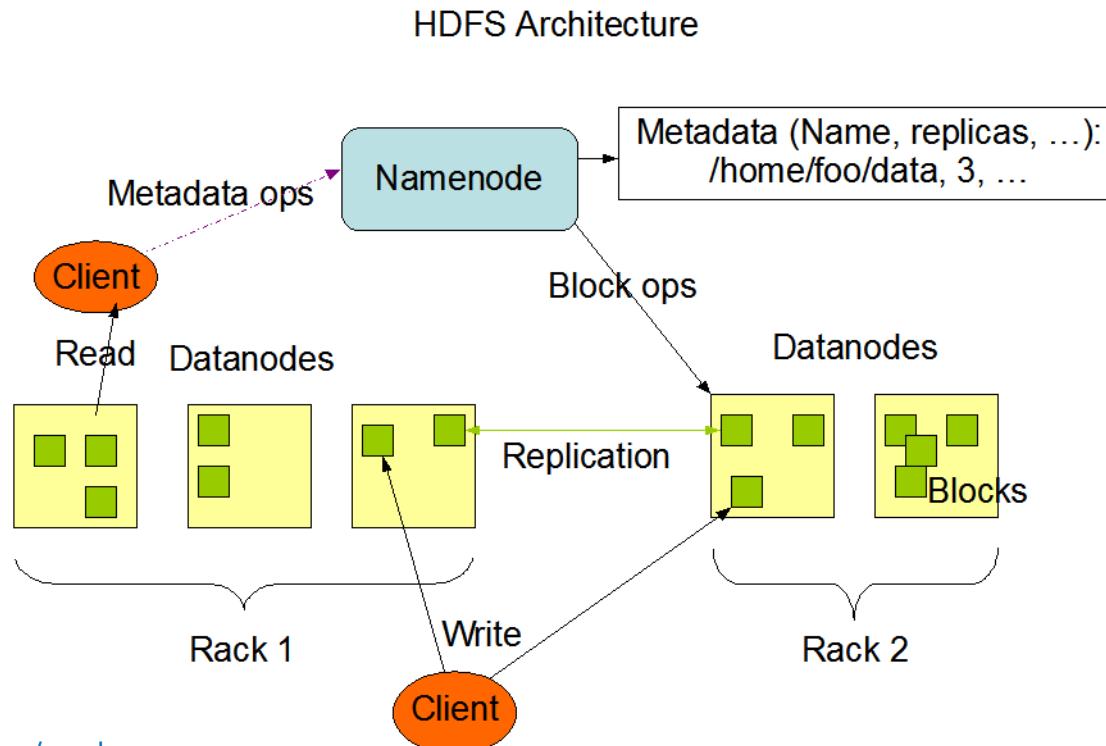
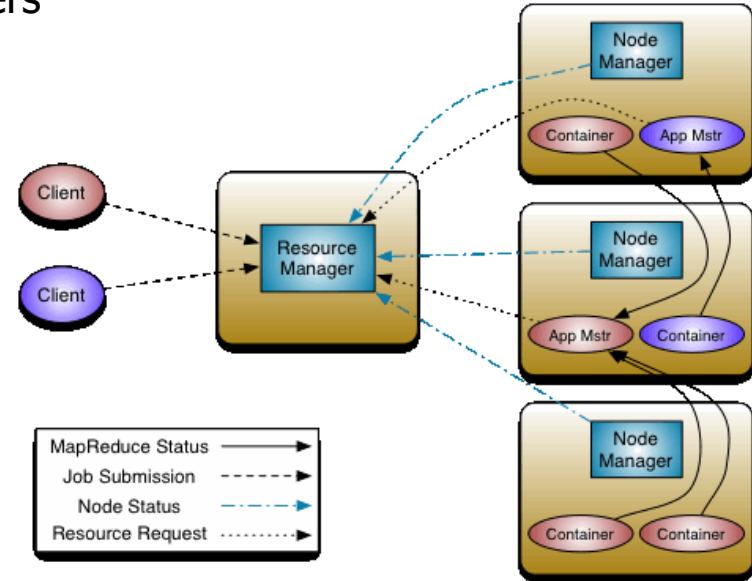


Image from <https://hadoop.apache.org>

What Is YARN?

- ◆ YARN stands for Yet Another Resource Negotiator
 - Queues job requests
 - ↳ MapReduce (MR), Pig, Hive, Spark, Tez, others
 - Assigns jobs to nodes for execution
 - ↳ Based on work load and resource availability
- ◆ YARN is composed of two major services:
 - Resource Manager (RM)
 - Node Manager (NM)



Resource Manager

- ◆ YARN responsibilities include:
 - Resource management (CPU, RAM, user defined)
 - Job scheduling
- ◆ YARN is composed of two major services:
 - Resource Manager
 - Node Managers
- ◆ The Resource Manager operates at the cluster level
 - Has sole authority to allocate resources to applications
 - It allocates resources as containers
 - ➔ A container grants use of specific resources on a specific node
- ◆ There are three types of containers:
 - Application master
 - Map
 - Reduce

Node Manager

◆ Node Manager

- Defines the resources available on its machine
 - ◆ Using `yarn-site.xml` or `noderesources.xml`
- Launches containers
 - ◆ In response to requests from the Resource Manager or an Application Master

Application Master

- ◆ Applications are composed of one or more jobs
 - Jobs are modeled as directed acyclic graphs (DAGs)
 - Clients submit jobs to the Resource Manager
- ◆ The Resource Manager, with the help of a Node Manager, launches one Application Master for each submitted job
- ◆ An Application Master:
 - Is responsible for executing an application
 - Contains the framework-specific libraries needed to execute the job
 - ◆ Such as the libraries for MapReduce, Spark, or Hive
 - Obtains resources (containers) from the Resource Manager
 - Executes and monitors containers via the Node Managers

Application Master (continued)

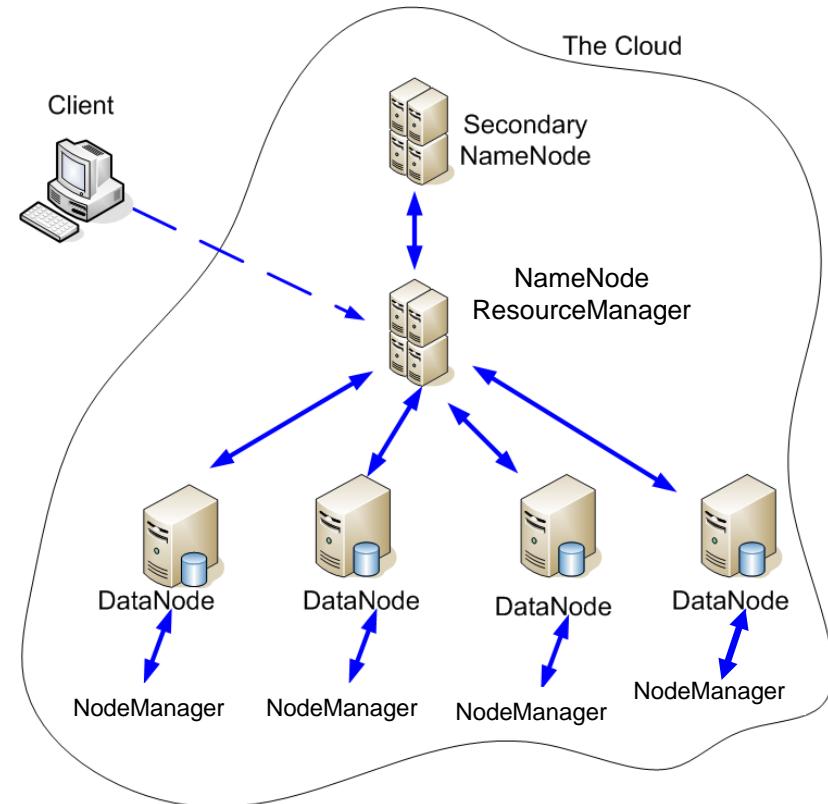
- ◆ The delegation of the responsibility of application execution to Application Masters improves scalability
- ◆ Once it has started, an Application Master may:
 - Request additional containers to execute the submitted job
 - Or decide to execute the job in its own container
 - ◆ Called an *uber job*

Summary of Hadoop Services

Service	Description	Purpose
NameNode	Manager node for HDFS	Coordinates the data nodes
DataNode	Worker node for HDFS	Reads and writes HDFS blocks
SecondaryNameNode (SNN)	Recovery service in case of NameNode failure	Takes snapshots of HDFS metadata (check points) to minimize data loss
ResourceManager	Manager node for YARN	Accepts jobs submitted to cluster, coordinates NodeManagers, monitors tasks
NodeManagers	Worker node for YARN	Executes tasks

Topology of a Small Hadoop Cluster

- Note that the client machine will also have Hadoop installed in order to have the necessary libraries and client programs to interact with HDFS and submit MapReduce jobs to the cluster



Hadoop Execution Modes

- ◆ At a minimum, Hadoop is implemented as four different Linux processes
 - Two are for HDFS: NameNode (exactly one), DataNode (one or more)
 - Two are for YARN: ResourceManager (exactly one), NodeManager (one or more)
- ◆ Additional services can be run to provide support for
 - NameNode Failure Recovery (e.g., SecondaryNameNode)
 - NameNode Federation (e.g., multiple NameNodes for greater scalability)
 - High Availability (e.g., multiple NameNodes and Resource Managers, ZooKeeper)
- ◆ Hadoop can be run in one of two modes
 - **Pseudo distributed mode**
 - ◆ All the Hadoop processes run on a single machine
 - ◆ Useful for development, test, proof of concept
 - **Full cluster mode**
 - ◆ Hadoop processes run on multiple nodes in a cluster

Chapter Concepts

About Big Data

Introducing MapReduce

MapReduce with Linux

MapReduce on a Cluster

Introducing Hadoop



Running Hadoop

Installing Hadoop

Chapter Summary

Launch Class Virtual Machine

- ◆ Your instructor will guide you through the process of starting your virtual machine for this class
- ◆ Depending on the virtual machine, you may be prompted to log in
 - Username: **student**
 - Password: **student**
- ◆ The student account is a member of the `sudo` group
 - Allows running the `sudo` command as root



Start Pseudo Distributed Hadoop

- Once you have logged in, open a terminal window and execute the following commands:

```
$ start_hadoop.sh  
$ jps
```

- The `jps` command displays all of the Java processes running on your virtual machine. These should include the following Hadoop processes:
 - NameNode (HDFS)
 - SecondaryNameNode (HDFS)
 - DataNode (HDFS)
 - ResourceManager (YARN)
 - NodeManager (YARN)
 - JobHistoryServer

The NameNode Web Interface



- ◆ The NameNode and the Resource Manager are also web applications
- ◆ Visit the NameNode using the Firefox bookmark Hadoop → Namenode for Hadoop 3.0.0
- ◆ On the NameNode Overview page, select the Utilities → Browse the file system menu item

A screenshot of a web browser window titled "Namenode information". The address bar shows "localhost:9870/dfshealth.html#tab-overview". The main content area displays the "Overview" page for the NameNode, showing the status of the cluster. At the top, there is a navigation bar with tabs for Datasets, Hadoop, Hive, Neo4j, and Spark. Below this is a sidebar with icons for Namenode for Hadoop 2.7.5, Namenode for Hadoop 3.0.0, and Resource Manager, along with a "Open All in Tabs" option. A red arrow points from the "Utilities" dropdown menu in the top right corner to a submenu that includes "Browse the file system" and "Logs". Another red arrow points from the "Logs" option in the submenu back to the "Logs" link on the overview page itself.

The Resource Manager Web Interface



- Visit the Resource Manager using the Hadoop → Resource Manager bookmark in Firefox
 - We will investigate this interface when some MapReduce jobs have been launched

A screenshot of a web browser window displaying the Hadoop Resource Manager web interface. The title bar shows "All Applications" and the URL "localhost:8088/cluster". The left sidebar has a yellow elephant icon and navigation links like "Datasets", "Hadoop", "Hive", "Neo4j", and "Spark". A dropdown menu is open over the "Resource Manager" link. A large red arrow points from the text "Visit the Resource Manager using the Hadoop → Resource Manager bookmark in Firefox" to this dropdown menu. The main content area is titled "All Applications" and contains several metrics tables:

Cluster Metrics
Apps Submitted: 0
Apps Pending: 0
Apps Running: 0
Apps Completed: 0
Containers Running: 0
Memory Used: 0 B
Memory Total: 8 GB

Cluster Nodes Metrics
Active Nodes: 1
Decommissioning Nodes: 0
Decommissioned Nodes: 0
Lost Nodes: 0

User Metrics for student
Apps Submitted: 0
Apps Pending: 0
Apps Running: 0
Apps Completed: 0
Containers Running: 0
Containers Pending: 0
Containers Reserved: 0
Memory Used: 0 B

Scheduler Metrics
Scheduler Type: Fair Scheduler
Scheduling Resource Type: [memory-mb (unit=Mi), vcores]
Minimum Allocation: <memory:1024, vCores:1>
Maximum Allocation: <memory:8192, vCores:16>

Show 20 entries



Stopping Hadoop

- ◆ In production environments, only administrators start and stop a production Hadoop cluster
 - For the classroom environment, we will have to start and stop it
- ◆ You have already seen how to start Hadoop for this class
 - Here is how to stop Hadoop:

```
$ stop_hadoop.sh  
$ jps
```

- ◆ The `jps` command should show that none of the Hadoop processes are running

Chapter Concepts

About Big Data

Introducing MapReduce

MapReduce with Linux

MapReduce on a Cluster

Introducing Hadoop

Running Hadoop

 **Installing Hadoop**

Chapter Summary

Setting Up Hadoop

- ◆ To set up Hadoop, need to set up:
 - Environment variables
 - Communication
 - Configuration
 - ↳ Core Hadoop
 - ↳ HDFS
 - ↳ Yarn
 - Formatting HDFS

Installing Hadoop

- ◆ HDFS is bundled with Apache Hadoop
 - This chapter uses Hadoop 3.0.0
- ◆ The following steps are based on the “Apache Hadoop Releases” documentation (see: <https://hadoop.apache.org/releases.html>):
 - Download the binary file `hadoop-3.0.0.tar.gz` to `/usr/student/Downloads`
 - Extract to: `/usr/local`
 - ➔ `su -c "tar -zxvf Hadoop-3.0.0.tar.gz -C /usr/local"`
 - Create a symbolic link
 - ➔ `cd /usr/local`
 - ➔ `sudo ln -s hadoop-3.0.0/ hadoop`



Hadoop Environment Variables

- ◆ Several Hadoop variables need to be declared in the environment
 - Often declared in a file like `/etc/profile.d/hadoop.sh` or `~/.bashrc`
 - Execute the following command to see the Hadoop variables:
`cat /etc/profile.d/hadoop.sh`

```
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_YARN_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_CONF_HOME=$HADOOP_HOME/etc/hadoop
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export YARN_CONF_HOME=$HADOOP_CONF_HOME
Export PDSH_RCMD_TYPE=ssh
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
```

- ◆ The list of variables will be slightly different for Hadoop 2.X
 - Check the documentation

Communication Between Services

- ◆ Communication between the Hadoop services is via ssh
 - Need to have password-less ssh set up between nodes
- ◆ For the pseudo-distributed configuration, we should be able to type:

```
ssh localhost
```

- And find ourselves logged in
 - ➔ The first time, we may have to answer “yes” to a prompt
 - ➔ So, try doing ssh manually before running Hadoop
- ◆ Since Hadoop services can not answer prompts or type passwords, ssh login should happen with no prompts or passwords

Password-Less Login with ssh—I

- ◆ Setting up password-less ssh between two machines is straightforward
 - Install OpenSSH from <http://www.openssh.com/>
 - ◆ sudo apt-get install ssh
 - ◆ sudo apt-get install rsync
 - Make sure ssh, sshd, ssh-keygen, etc. are in your PATH
- ◆ On machine1, to ssh into remote machines on the cluster without a password:
 - Generate an ssh key pair using no passphrase
 - ◆ ssh-keygen -t rsa
- ◆ For **pseudo-distributed** mode, copy the key to the authorized_keys file:
 - cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
- ◆ For **full distributed** mode, copy the public key file to each remote machine:
 - ssh-copy-id -i ~/.ssh/id_rsa.pub username@machine2

Password-Less Login with ssh—II

- ◆ Test ssh for pseudo-distributed mode
 - ssh localhost
- ◆ Test ssh for full distributed mode
 - ssh machine2 uname -a
 - Repeat for every pair of machines
 - Including from machine2 to machine1
 - Full details, including troubleshooting help, can be found at:
<http://www.debian-administration.org/articles/152>

Configuring Hadoop

- ◆ Next step is to update the Hadoop configuration files
- ◆ As shipped, the configuration files are in `HADOOP_HOME/etc/hadoop`
 - The following configuration files have empty configuration elements:
 - ◆ `core-site.xml`
 - ◆ `hdfs-site.xml`
 - ◆ `yarn-site.xml`

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
</configuration>
```

- There is a `mapred-site-template.xml` but no `mapred-site.xml`

Basic Hadoop Configuration

- ◆ How does the `hadoop` command running on your machine know where the NameNode is running on the cluster?
 - The `hadoop` command reads a configuration file called `core-site.xml`
 - ◆ Located at `~/roihadoop/setup/hadoopconf/core-site.xml`

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

- ◆ `core-site.xml` and other configuration files, including `hdfs-site.xml`, `mapred-site.xml`, and `yarn.xml`, may have to be adjusted on your machine to work properly with the cluster

Formatting HDFS

- ◆ HDFS needs to be formatted before its very first use
 - We will format HDFS in the next chapter

Chapter Concepts

About Big Data

Introducing MapReduce

MapReduce with Linux

MapReduce on a Cluster

Introducing Hadoop

Running Hadoop

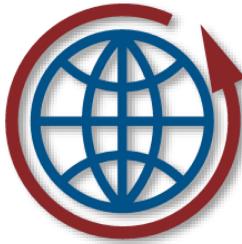
Installing Hadoop

► Chapter Summary

Chapter Summary

In this chapter, we have:

- ◆ Described Big Data
- ◆ Introduced MapReduce
- ◆ Implemented MapReduce with Linux commands
- ◆ Presented the benefits and issues of using MapReduce on a cluster
- ◆ Given an overview of the Hadoop framework



ROI TRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Module 2: Global Technical Analyst Program

CHAPTER 5: HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

Chapter Objectives

In this chapter, we will:

- ◆ Learn about the Hadoop Distributed Files System (HDFS)
- ◆ Run a stand-alone instance of HDFS
- ◆ Create directories and files in HDFS
- ◆ Create a 6-node HDFS cluster using Docker

Chapter Concepts

➤ HDFS Overview

Launching HDFS

HDFS Commands

Fully Distributed HDFS

HDFS Cluster with Docker

Chapter Summary

About HDFS—I

- ◆ The Hadoop Distributed File System (HDFS) is the main storage used by Hadoop MapReduce applications
 - Distributed, POSIX-like file system
 - ↳ Designed to run on commodity hardware
 - ↳ Scales to clusters composed of thousands of nodes
 - Highly fault tolerant
 - ↳ Automatically detects hardware faults
 - ↳ Supports quick recovery
 - Implemented in Java
- ◆ Can be used as a stand-alone general purpose file system, but relaxes certain POSIX filesystem requirements
 - Designed for storing and reading very large files (>TB)
 - ↳ Supports high throughput read and writes
 - ↳ Write once, read many
 - ↳ Aimed at batch processing
 - ↳ Default block size is 128MB
 - Does not support random insertion or modification of data
 - Appending/truncating data is possible

About HDFS-II

- ◆ HDFS is used either directly or indirectly by many BigData and NoSQL applications including:
 - Hadoop
 - Spark
 - HBase
 - Pig
 - Hive
 - Others

Core HDFS Services

- ◆ HDFS is implemented as several services which are usually deployed on a cluster of machines
 - Referred to as an HDFS cluster
 - Arranged in a master/slave architecture
- ◆ Core HDFS Services include:
 - **NameNode** which stores file system metadata
 - **DataNode** which stores file data (data blocks)
- ◆ The NameNode is the master server
 - Implements a POSIX-like hierarchical file system with '/' as the root directory
 - Enforces read/write permissions on files and directories
 - Tracks the location of the data blocks for each file
- ◆ The DataNode is the slave server
 - Handles read and write requests from HDFS clients
 - Performs block creation, deletion, and replication as instructed by the NameNode

Core HDFS Services (continued)

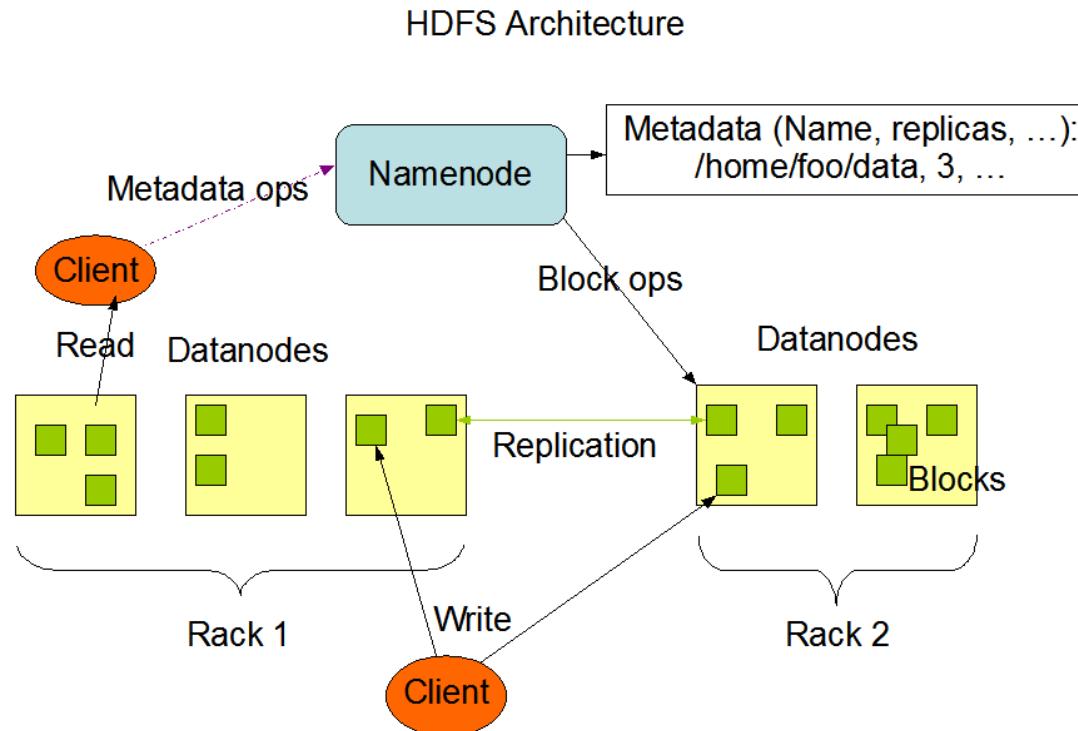


Image from <https://hadoop.apache.org>

Chapter Concepts

HDFS Overview

➤ **Launching HDFS**

HDFS Commands

Fully Distributed HDFS

HDFS Cluster with Docker

Chapter Summary



Launching HDFS—I

- ◆ HDFS is designed to work in a multi-node cluster
 - It can also function with one node, which is where we will start
- ◆ Open a terminal window
- ◆ From the terminal window, enter the following commands:

```
$ cd ~/roi_hadoop/hdfs_exercises/standalone  
$ ls
```

- ◆ You should see the following:

```
core-site.xml      format_hdfs.sh      hdfs-site.xml  
start-standalone-hdfs.sh      stop-standalone-hdfs.sh
```

- ◆ There are two .xml configuration files and three shell scripts

core-site.xml



- ◆ Many settings in Hadoop and HDFS can be configured
- ◆ The **core-site.xml** file holds settings common to the entire Hadoop platform
 - The essential property in this file, `fs.defaultFS`, is the URL that client applications use to access the NameNode
- ◆ We looked at this file in the previous chapter using:

```
cat core-site.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```



hdfs-site.xml

- ◆ The **hdfs-site.xml** file holds settings specific to HDFS
 - There are several essential properties in this file
 - Execute the following command to view its contents:
 - cat -n hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/home/student/hdfs/nn</value>
  </property>
  <property>
    <name>fs.checkpoint.dir</name>
    <value>file:/home/student/hdfs/snn</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/home/student/hdfs/dn</value>
  </property>
</configuration>
```



NameNode Log Files—I

- ◆ Both the NameNode and DataNode write their data to the host (Linux) file system to the locations specified in `hdfs-site.xml`
 - The NameNode will be writing to `/home/student/hdfs/nn`
 - The DataNode will be writing to `/home/student/hdfs/dn`
- ◆ From the terminal window, enter the following command:

```
ls ~/hdfs/nn/current
```

```
edits_00000000000000000001-00000000000000000002 edits_00000000000000000003-00000000000000000003
edits_00000000000000000004-00000000000000000004 edits_00000000000000000005-00000000000000000006
edits_inprogress_00000000000000000007
fsimage_00000000000000000003
fsimage_00000000000000000003.md5
fsimage_00000000000000000006
fsimage_00000000000000000006.md5
seen_txid
VERSION
```

- ◆ There are two file types used by NameNode to **log** HDFS metadata: `edits_*`, and `fs_image_*`

NameNode Log Files—II



- ◆ A NameNode stores HDFS metadata for one HDFS namespace
 - A “federated” HDFS can have several NameNodes each managing their own namespace
- ◆ HDFS metadata includes:
 - Directory tree structure
 - Names, ownership, permissions of all files and directories in the namespace
 - Replication level of each file
 - The locations of each data block managed by the DataNodes
- ◆ To enhance NameNode performance, all metadata is stored in RAM
- ◆ All HDFS metadata, except data block locations, is also logged (persisted) to the file system using the `edits_*` and `fs_image_*` files
 - These files are generally referred to by the names `edits` and `fsimage`, respectively
- ◆ Data block locations are persisted to disk by the DataNodes that are storing them

Safe Mode—I

- ◆ When a NameNode is booted, it determines if it is managing metadata for files with data blocks
 - If so, it enters the **Safe Mode** state
 - Otherwise, it enters normal operating mode
 - ◆ For example, this will occur when HDFS has just been formatted
- ◆ Safe Mode is a window of time during which the DataNodes (that are also booting up) communicate their state to the NameNode
 - A list of data blocks that they are storing
- ◆ Safe Mode
 - Allows (limited) read operations to be performed by clients
 - Doesn't allow write operations
 - ◆ From either clients or the NameNode itself
 - NameNode won't be able to issue block replication or deletion requests
 - Clients won't be able to:
 - ◆ Create or delete directories and files
 - ◆ Change file permissions, ownership, etc.

Safe Mode—II

- ◆ When booting into Safe Mode, the NameNode:
 - Reads the content of the current `fsimage` file into RAM
 - Applies the edits in the current `edits` file to the data in RAM
 - Writes the updated content in RAM to a new `fsimage` disk file
 - Creates a new, empty `edits` file
 - When the DataNodes boot, they send their list of data blocks to the NameNode
 - The NameNode builds an in-memory map of all the data block locations noting how many copies of each data block exist
 - The NameNode does not write this information to the `edits` or `fsimage` files
- ◆ Once the NameNode determines that 99.9% of all data blocks are at their minimum replication level, it exits Safe Mode
- ◆ Once out of Safe Mode, all changes made to the HDFS state are:
 - First logged (appended) to the `edits` file
 - Then updated in RAM



Formatting HDFS—I

- When installing HDFS, it is important to format the NameNode before booting it
- Formatting the NameNode creates the correct directory structure and necessary files under the `dfs.namenode.name.dir` directory
 - Otherwise, the NameNode won't boot
- From the terminal window, enter the following commands:

```
$ cd ~/roi_hadoop/hdfs_exercises/standalone  
$ cat format_hdfs.sh
```

- You should see the following:

```
#!/bin/sh  
  
rm -rf $HOME/hdfs  
mkdir -p $HOME/hdfs/nn  
mkdir -p $HOME/hdfs/snn  
mkdir -p $HOME/hdfs/dn  
  
$HADOOP_HOME/bin/hdfs namenode -format
```



Formatting HDFS—II

- From the terminal window, enter the following command:

```
./format_hdfs.sh
```

- This script will start the NameNode, and in the process of formatting, generate much output
- What is the value of the clusterid: _____
- Who is the fsOwner: _____
- What is the value of the BlockPoolId: _____
- When the script completes, the NameNode is shut down



Newly Formatted NameNode

- From the terminal window, enter the following commands:

```
ll ~/hdfs/nn/current
```

- You should see something like the following:

```
-rw-r--r-- 1 student student 354 Oct  2 15:04 fsimage_0...
-rw-r--r-- 1 student student   62 Oct  2 15:04 fsimage_0...0.md5
-rw-r--r-- 1 student student     2 Oct  2 15:04 seen_txid
-rw-r--r-- 1 student student 201 Oct  2 15:04 VERSION
```

- Notice that there are no edits files
- From the terminal window, enter the following commands:

```
ll ~/hdfs/dn
```

- The directory should be empty



The VERSION File

- From the terminal window, enter the following commands:

```
cat -n ~/hdfs/nn/current/VERSION
```

- You should see something like the following:

```
1 #Sun Oct 02 15:04:45 EDT 2016
2 namespaceID=1837345224
3 clusterID=CID-dca795c7-63db-48eb-9f52-6a1be2ab0f34
4 cTime=0
5 storageType=NAME_NODE
6 blockpoolID=BP-527989250-127.0.1.1-1475435085849
7 layoutVersion=-63
```

- HDFS supports a **federated** mode where two or more NameNodes can run at the same time on the same cluster
 - Each NameNode manages its own unique subset of the HDFS files
 - The subset of files is called the 'namespace' and has its own namespaceID
 - Since NameNodes store metadata in memory, the amount of memory determines the upper bound of the size of HDFS
 - Multiple federated NameNodes results in a larger HDFS



Formatting HDFS—III

- When we previously viewed the `format_hdfs.sh` script, we saw the following:

```
#!/bin/sh

rm -rf $HOME/hdfs
mkdir -p $HOME/hdfs/nn
mkdir -p $HOME/hdfs/snn
mkdir -p $HOME/hdfs/dn

$HADOOP_HOME/bin/hdfs namenode -format
```

- When running the `hdfs namenode -format` command by itself on an existing HDFS system, all the HDFS metadata is destroyed and a new clusterID is created for the NameNode
- Existing DataNodes will have a `VERSION` file with a clusterID that matches the previous NameNode clusterID
 - These DataNodes won't boot due to the mismatched clusterIDs
 - When a NameNode is formatted, the data directory of each DataNode must have its content completely removed in order for it to boot with the newly formatted NameNode



Launching HDFS—II

- ◆ Enter the following command to launch HDFS in stand-alone mode
 - Also known as pseudo-distributed mode

```
$ cd ~/roi_hadoop/hdfs_exercises/standalone  
$ ./start-standalone-hdfs.sh
```

- ◆ You should see something similar to the following:

```
Starting namenodes on [localhost]  
Starting datanodes  
Starting secondary namenodes
```

- ◆ The `start-dfs.sh` script is part of the Hadoop distribution
 - By default, it will launch one instance of the NameNode, DataNode, and SecondaryNameNode on the current machine
- ◆ Execute the following command to see the details:

```
cat -n $HADOOP_HOME/sbin/start-dfs.sh
```



Launching HDFS—III

- ◆ Enter the following command and verify that new files have been added to the `dfs.namenode.name.dir` directory:

```
ll ~/hdfs/nn/current
```

- ◆ You should see something similar to the following:

```
-rw-rw-r-- 1 student student 1048576 Oct  2 15:10
                                         edits_inprogress_0...01
-rw-r--r-- 1 student student      354 Oct  2 15:04 fsimage_0...0
-rw-r--r-- 1 student student      62 Oct  2 15:04 fsimage_0...0.md5
-rw-rw-r-- 1 student student       2 Oct  2 15:10 seen_txid
-rw-r--r-- 1 student student     201 Oct  2 15:04 VERSION
```



Launching HDFS—IV

- ◆ Enter the following command and verify that files have been added to the `dfs.namenode.data.dir` directory:

```
ll ~/hdfs/dn/current
```

```
drwx----- 4 student student 4096 Oct  2 15:10 BP-527989250-
          127.0.1.1-1475435085849/
-rw-rw-r-- 1 student student   229 Oct  2 15:10 VERSION
```

- ◆ Enter the following command:

```
cat -n ~/hdfs/dn/current/VERSION
```

```
1 #Sun Oct  02 15:10:34 EDT 2016
2 storageID=DS-860e214b-b3f8-42dc-bf74-07efce026500
3 clusterID=CID-dca795c7-63db-48eb-9f52-6a1be2ab0f34
4 cTime=0
5 datanodeUuid=83e66858-dbe6-4869-a83d-867bbe89b286
6 storageType=DATA_NODE
7 layoutVersion=-56
```

- ◆ Does the DataNode clusterID match the NameNode clusterID? _____

Chapter Concepts

HDFS Overview

Launching HDFS

➤ **HDFS Commands**

Fully Distributed HDFS

HDFS Cluster with Docker

Chapter Summary



HDFS Commands—I

- Once HDFS is running on your machine, execute the following commands:

```
$ hadoop  
$ hadoop fs  
$ hadoop fs -ls -R /  
$ hadoop fs -copyFromLocal  
    ~/roi_datasets/big_shakespeare.txt  /big_shakespeare.txt  
$ hadoop fs -ls /
```

- The `copyFromLocal` option copies a file from the file system on your machine to the HDFS file system on the cluster
 - Due to the way it is designed, HDFS prefers one large file as opposed to many small files
 - Files are automatically broken into 128MB blocks
 - The data blocks are placed on different DataNodes in the cluster and automatically replicated
- View the results of the command using the NameNode web interface



HDFS Commands—II

- From a command line, enter the following commands:
 - Note that the hadoop and hdfs commands have much overlap

```
$ hdfs  
$ hdfs dfs  
$ hdfs dfs -ls -R /  
$ hdfs dfs -mkdir -p /user/student  
$ hdfs dfs -put ~/roi_datasets/shakespeare.txt  
$ hdfs dfs -chmod og+w /user/student/shakespeare.txt  
$ hdfs dfs -chown <your_name>  
/user/student/shakespeare.txt  
$ hdfs dfs -ls -R /
```

- Using the NameNode web interface, inspect the changes made by the above commands
- When ready, stop the HDFS cluster with the following command:

```
$ ./stop-standalone-hdfs.sh
```

Chapter Concepts

HDFS Overview

Launching HDFS

HDFS Commands

➤ **Fully Distributed HDFS**

HDFS Cluster with Docker

Chapter Summary

Additional HDFS Services

- ◆ Besides the NameNode and DataNode, HDFS provides several additional services including:
 - SecondaryNameNode
 - ↳ Checkpoints the NameNode data
 - ↳ Checkpointed image can be read by NameNode
 - ↳ Supports manual failover
 - Checkpoint Node
 - ↳ Periodically creates check points of the NameNode's data
 - ↳ Improves performance of the NameNode
 - Backup Node
 - ↳ Same checkpointing functionality as Checkpoint Node
 - ↳ Maintains a synchronized, in-memory copy of the NameNode data
 - Quorum Journal Nodes
- ◆ Each of the above services is involved in mitigating the effects of a NameNode failure
 - Historically, the NameNode has been a single point of failure for an HDFS cluster

Checkpointing—I

- ◆ Earlier in the chapter, the NameNode Safe Mode boot process was described
 - It involved creating a new `fsimage` by merging the existing `fsimage` and `edits` files
 - This process only occurs when the NameNode boots
 - On a busy HDFS cluster, the time between reboots could be long (days) resulting in an ever larger `edits` file
 - Larger `edits` files slow down the NameNode boot process
- ◆ The SecondaryNameNode performs a checkpointing process
 - It periodically downloads the `fsimage` and `edits` files from the NameNode
 - Performs the merge locally
 - Stores the results of the merge in a directory accessible to the NameNode
 - With the SecondaryNameNode checkpoints, the NameNode boot time can be maintained at an acceptable level
 - The SecondaryNameNode supports manual failover

Checkpointing—II

- ◆ Checkpoint Node performs the same functions as the SecondaryNameNode
 - In addition, it uploads the merged image back to the NameNode
- ◆ Multiple Checkpoint Nodes are allowed in a cluster
- ◆ Backup Node performs the same functions as the Checkpoint Node
 - It also maintains an in-memory copy of the NameNode's namespace
- ◆ The Backup Node receives a near real-time stream of all the edits processed by the NameNode
 - Uses this data to maintain its in-memory data as well as its copies of the `fsimage` and `edits` files
 - It never needs to download the `fsimage` and `edits` files from the NameNode
 - It never needs to perform a merge, it just periodically writes the in-memory image to a new `fsimage` file and resets the `edits` file

HDFS Deployment Configurations

- ◆ HDFS identifies three different deployment configurations
 - Each configuration allows variations
- ◆ Standard, fully distributed configuration
 - 1 NameNode
 - 1 SecondaryNameNode
 - 1 or more DataNodes
- ◆ Federated configuration
 - ◆ 2 or more NameNode – SecondaryNameNode pairs
 - ◆ 1 or more DataNodes
- ◆ High Availability
 - ◆ 2 NameNodes
 - ◆ 1 or more DataNodes
 - ◆ 3 or more JournalNodes
 - ◆ 3 or more Zookeeper Nodes

Chapter Concepts

HDFS Overview

Launching HDFS

HDFS Commands

Fully Distributed HDFS

➤ **HDFS Cluster with Docker**

Chapter Summary

Launching a Full HDFS Cluster—I



- ◆ An alternative to running the HDFS services directly on your machine in stand-alone mode is to run each service in a Docker container
- ◆ Although it is possible to manually configure multiple Docker containers to work together, it is easier to use a Docker orchestration tool
- ◆ There are several orchestration tools available for Docker images
 - We will use docker-compose
- ◆ From a terminal window, enter the following commands:

```
$ cd ~/roi_hadoop/hdfs_exercises/distributed  
$ docker-compose up
```

- ◆ Note: the HDFS cluster being started is using version 2.7.5 of Hadoop



Launching a Full HDFS Cluster—II

- From a second terminal window, enter the following commands:

```
$ cd ~/roi_hadoop/hdfs_exercises/distributed  
$ docker-compose ps  
$ docker-compose scale datanode=4  
$ docker-compose ps
```

- From the output of the last command, you should see:

Name	Command	State	Ports

distributed_datanode_1	/entrypoint.sh	Up	15000/tcp...
distributed_datanode_2	/entrypoint.sh	Up	15000/tcp...
distributed_datanode_3	/entrypoint.sh	Up	15000/tcp...
distributed_datanode_4	/entrypoint.sh	Up	15000/tcp...
namenode	/entrypoint.sh	Up	14000/tcp...
secondarynamenode	/entrypoint.sh	Up	14000/tcp...

- How many DataNodes have been started? _____



Accessing the Cluster

- ◆ The cluster we have launched is operating on a private Docker network
 - One of the ports on the namenode container has been exposed to the host system
 - It can be accessed by running the following command:

```
$ firefox http://localhost:50070 &
```

- ◆ How many DataNodes have been started? _____
- ◆ What is the NameNode storage directory? _____
- ◆ Are there any directories or files in the HDFS file system? _____

Accessing the Cluster via the CLI—I



- ◆ There are different approaches for accessing the HDFS cluster from the command line
 - One way is to create a new container that is provisioned for use as an HDFS client
- ◆ From a terminal window, execute the following commands:

```
$ cd ~/roi_hadoop/hdfs_exercises/distributed  
$ ./start-hdfs-client.sh
```

- ◆ You should see a new prompt that looks something like:

```
bash-4.3#
```

- ◆ From this new prompt, enter the following commands:

```
bash-4.3# hostname  
bash-4.3# env | grep HADOOP  
bash-4.3# hdfs dfs  
bash-4.3# hdfs dfs -ls -R /
```



Accessing the Cluster via the CLI—II

- From the client container, continue with executing the following commands:

```
bash-4.3# ls
bash-4.3# cat script1.sh
bash-4.3# ls /roi_datasets
bash-4.3# ./script1.sh
bash-4.3# hdfs dfs -ls -R /
bash-4.3# hdfs fsck /user/student/big_shakespeare.txt -files -blocks
bash-4.3# hdfs fsck /user/student/big_shakespeare.txt -files -blocks
    -locations
bash-4.3# exit
$ docker-compose ps
$ docker-compose down
```

- For details on the `fsck` command, see:

<https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSCCommands.html#fsck>

Chapter Concepts

HDFS Overview

Launching HDFS

HDFS Commands

Fully Distributed HDFS

HDFS Cluster with Docker

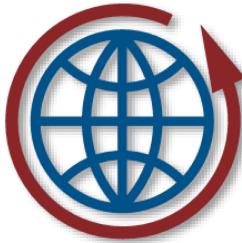


Chapter Summary

Chapter Summary

In this chapter, we have:

- ◆ Learned about the Hadoop Distributed Files System (HDFS)
- ◆ Ran a stand-alone instance of HDFS
- ◆ Created directories and files in HDFS
- ◆ Created a 6-node HDFS cluster using Docker



ROI TRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Module 2: Global Technical Analyst Program

CHAPTER 6: HADOOP MAPREDUCE

Chapter Objectives

In this chapter, we will:

- ◆ Review the data flow structure of MapReduce
- ◆ Design MapReduce applications
- ◆ Run simple MapReduce jobs using the Hadoop Streaming API
- ◆ Use the Hadoop Java API for MapReduce

Chapter Concepts

➤ MapReduce with Hadoop

Planning MapReduce

Streaming in Hadoop

Advanced Streaming

Beyond Streaming with Java

The Java API

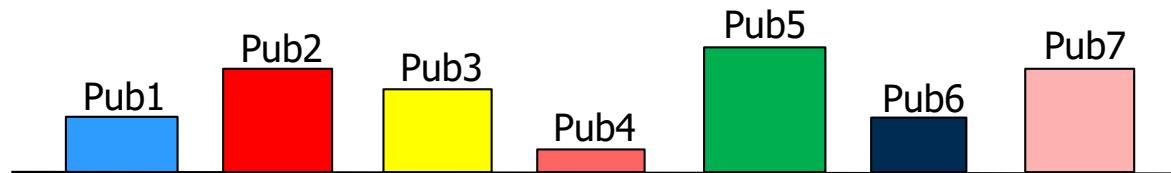
Chapter Summary

MapReduce with Hadoop—I

- ◆ Uploading a data file to HDFS is a distinct and separate operation from running MapReduce jobs on YARN
 - Large files will tend to stay on HDFS for extended periods of time
 - ◆ Over that time, many different MapReduce jobs can be run against the one large data file
- ◆ We remember that HDFS automatically breaks a large data file into 128MB blocks
 - Ignoring replication, let's assume that each block is being saved on a different DataNode in our cluster
 - Let's also assume that we are working with text data
 - ◆ Hadoop can also process binary data
- ◆ The MapReduce jobs that we submit to YARN will be run on the nodes where the data blocks are located

MapReduce with Hadoop—II

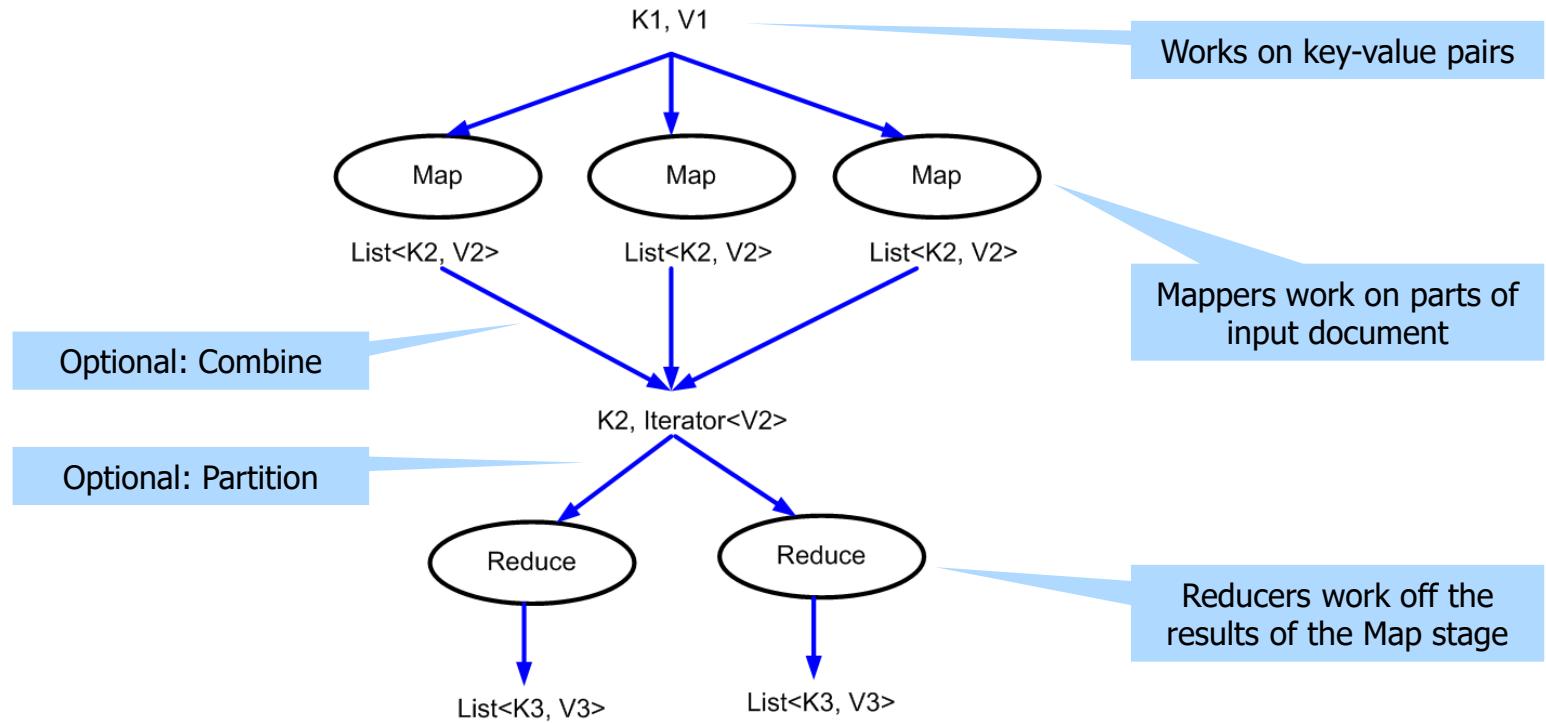
- ◆ On a given node, a MapReduce job on Hadoop operation consists of the following operations:
 - The data block on the node will be split into smaller parts called “splits”
 - One thread for executing the **Map** logic will be created for each split
 - The Map thread will read one line at a time from its split
 - ◆ It views each line of input as a <Key, Value> pair
 - For text data, the Key is the line number (commonly ignored) and the Value is the complete line of text
 - ◆ The Map thread is responsible for generating output that is also in the form the a <Key, Value> pair
 - For the book publisher problem, the key might be the publisher name and the value might be the complete book data
 - For the word count problem, the key might be the word, and the value might be the count
 - Input: <K1,V1> → Output: List<K2,V2>
 - In the picture below, Pub1, Pub2, etc. are the keys (K2)



MapReduce with Hadoop—III

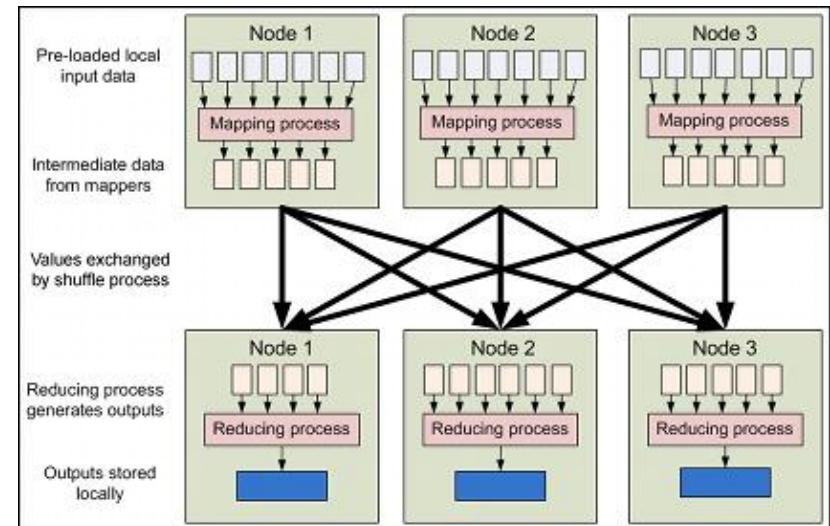
- ◆ On a given node, a MapReduce job on Hadoop operation consists of the following operations (continued from the previous slide):
 - Optional: The output of the Mappers is **combined**
 - ↳ The Combiner often has the same functionality as the Reducer
 - ↳ The Combiner is used to balance the work load between the Mappers, Reducers, and network
 - Optional: The output of the Mappers is **partitioned**
 - ↳ Partitioning is used to keep the Reducers equally busy
 - ↳ Partitioning assigns all the values associated with a specific key to exactly one Reducer
 - ↳ A bad partitioning algorithm will assign all the values for all the keys to one Reducer leaving the other Reducers idle
 - A **Reduce** task processes the output of the various Map tasks
 - ↳ Input: $\langle K2, \text{Iterator}\langle V2 \rangle \rangle$
 - ↳ Output: $\text{List}\langle K3, V3 \rangle$
- ◆ Terminology:
 - $K1, V1$ are the input key and value
 - $K2, V2$ are the intermediate key and value
 - $K3, V3$ are the output key and value

MapReduce in Generic Terms



Another View of MapReduce

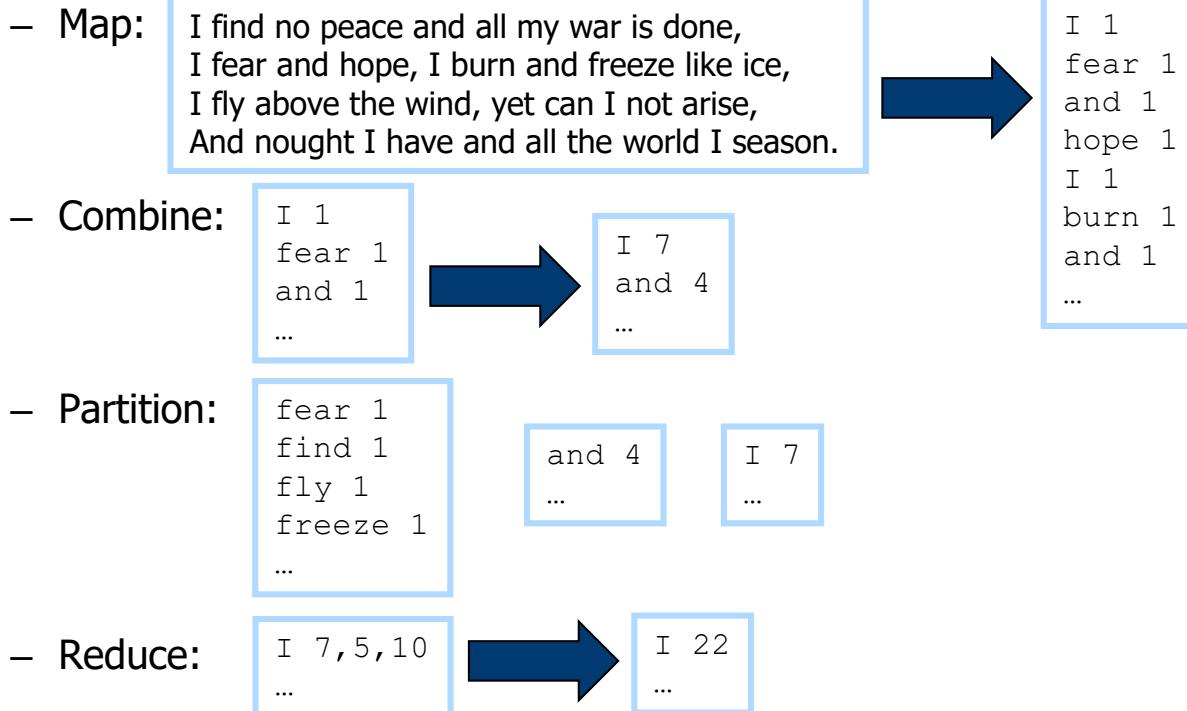
- ◆ Hadoop reads and writes to HDFS
 - This is not so obvious in this picture
 - Shakespeare.txt has already been uploaded to HDFS
 - Different data blocks on different nodes
- ◆ There will usually be many more nodes doing Mapping than Reducing
 - Combining and partitioning are not shown explicitly
 - They have default operations for text files



Source: Horton Works

Word Count with Hadoop MapReduce

- ◆ The steps for word count go like this:



Hadoop and Statistics

- ◆ Counting is a type of statistics operation referred to as distributive statistics
- ◆ It is helpful to divide statistics on large datasets into three categories
 - **Distributive** statistics like maximum, minimum, count, sum, etc.
 - ◆ You can compute these statistics on small chunks of data, and then recompute the statistics from the chunks to the whole dataset

```
Min_overall = Min ( Min_chunk1, Min_chunk2, ... )
```

- ◆ Distributive statistics are what MapReduce is designed for
- **Algebraic** statistics like average and variance
 - ◆ Although not distributive, there is a simple trick available

```
Avg_overall = Sum_overall / Count_overall
```

- Sum and Count are distributive statistics
- **Holistic** statistics like the Nth percentile (e.g., median)
 - ◆ These statistics require the complete dataset
 - ◆ Hard to break down and compute in a map/reduce framework
 - But not impossible!

Algebraic Statistics

- ◆ To compute an algebraic statistic:
 - Have the mappers output what's needed for each component
 - Have the reducers compute the component
 - Outside of Hadoop, carry out the final step

Chapter Concepts

MapReduce with Hadoop

➤ **Planning MapReduce**

Streaming in Hadoop

Advanced Streaming

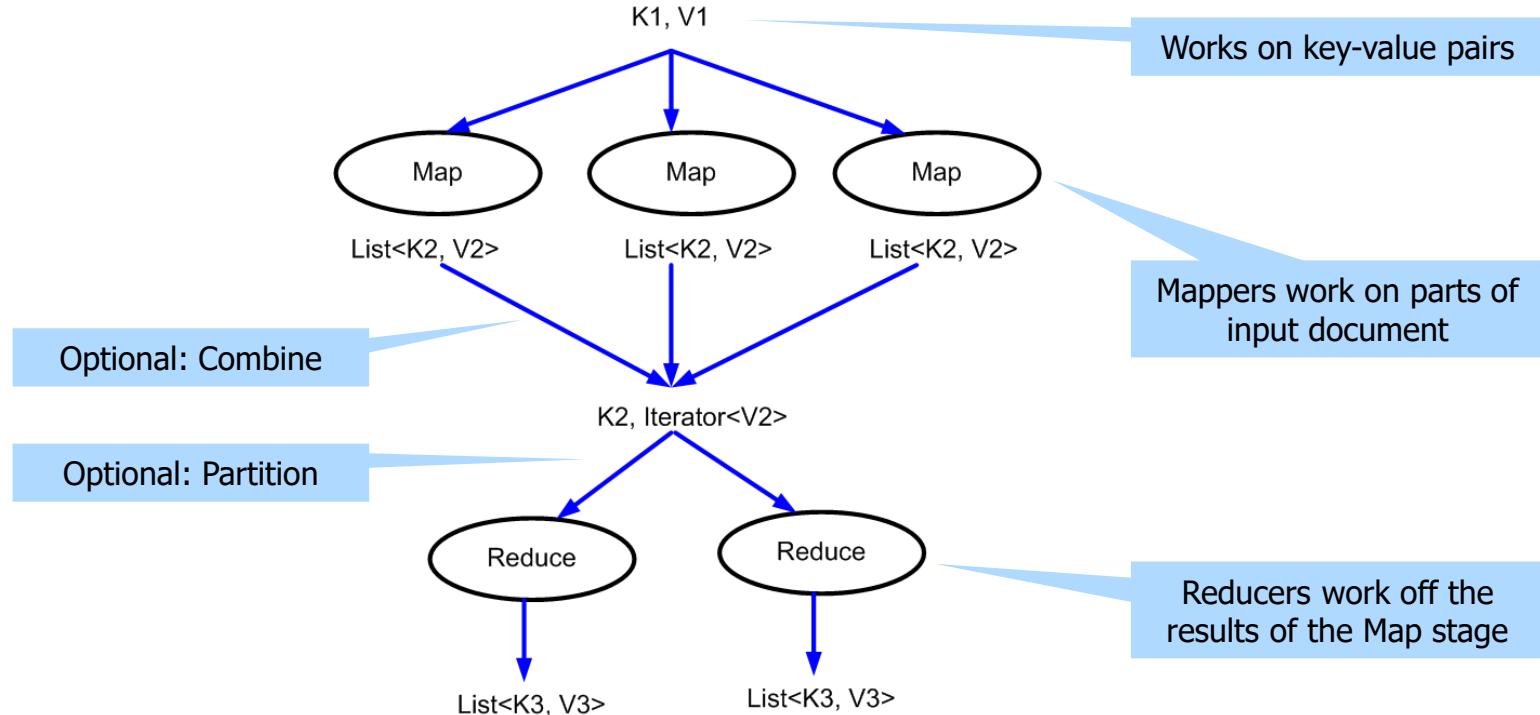
Beyond Streaming with Java

The Java API

Chapter Summary

A MapReduce Program

- Recall that MapReduce is a data processing algorithm



Planning MapReduce

- It is important to be comfortable with how MapReduce uses key-value pairs
 - This is independent of the framework such as Pig, Hive, or Spark

The diagram illustrates the flow of data through a MapReduce pipeline. A blue arrow labeled "Initial input" points to the first column of the table. A blue arrow labeled "Final output" points from the last column back towards the right.

Task	Input Type and Example	Description of Task	Output Type and Example
Map			
Combine			
Partition			
Reduce			

- By default, Hadoop uses an identity transformation for combine, partition, and reduce; i.e., it simply passes the data through
 - Often, we use these defaults

Word Count Example

- Consider the canonical word count example:

Task	Input Type and Example	Description of Task	Output Type and Example
Map	String I find no peace and all my war is done,	Split the line and emit the value 1 for each word	<String, int> I 1 find 1 no 1 peace 1
Combine	From Map	Add up counts	<String, int> find 3
Partition	From Combine	Split by starting letter	<String, int> find 3 fine 14
Reduce	From partition	Add up counts	<String, int>

Out of Stock Example

- Web server log files are frequently mined using MapReduce
 - Each log file line corresponds to one HTTP request as follows:

```
timestamp    fromIpAddress    requestedUrl    some-textual-log-message
```

- For a simple MapReduce example, consider that:
 - When a customer requests an out-of-stock product, the log message includes the key word OUTOFSOCK
 - You want to find which URLs lead to out-of-stock messages; i.e., you want output of the form:

```
/product/23403    18  
/product/15991    34
```

- The 18, 34, etc. are the number of times that the URL has been accessed and an OUTOFSOCK response was sent

Out of Stock Plan

Task	Input Type and Example	Description of Task	Output Type and Example
Map	String A line of the semi-structured log file	If line contains OUTOFSOCK, split the line and pull out 3 rd field	<String, int> /product/32 1
Combine			
Partition			
Reduce	<String, 1>	Add up counts	/product/32 3 etc.

Chapter Concepts

MapReduce with Hadoop

Planning MapReduce

➤ **Streaming in Hadoop**

Advanced Streaming

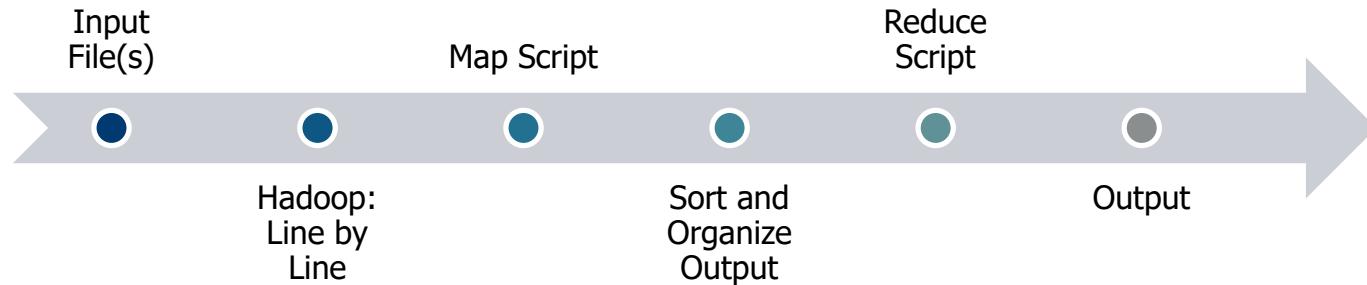
Beyond Streaming with Java

The Java API

Chapter Summary

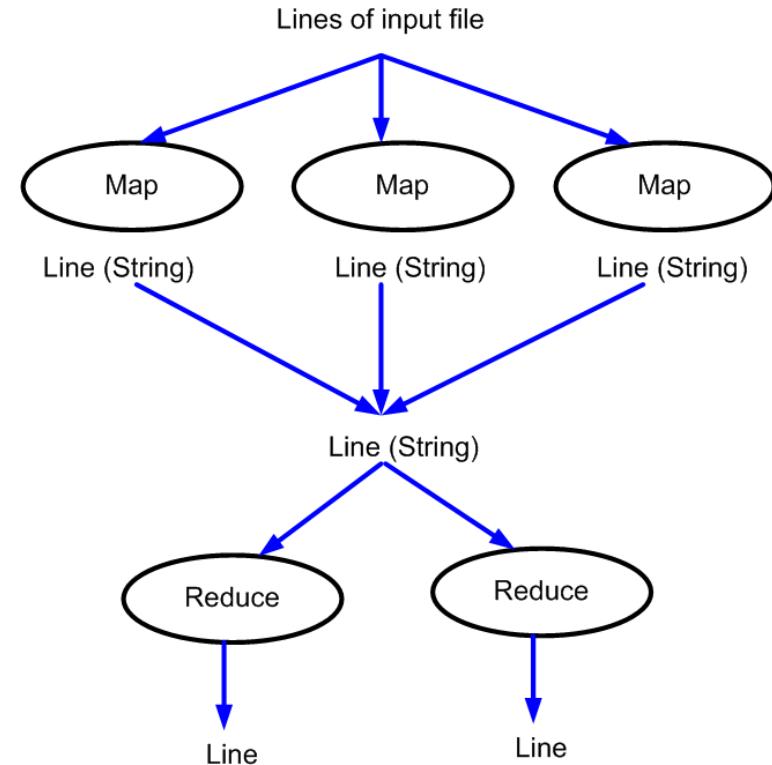
About the Hadoop Streaming API

- ◆ It is possible to execute some of the Linux shell scripts we saw earlier directly on Hadoop
 - Using the streaming API
- ◆ The streaming API supports other scripting languages
 - Python, Perl, Ruby
- ◆ The streaming API adopts Linux command line idioms
 - We provide map and reduce scripts that are Linux filter programs
 - Linux filter programs read from standard input and write to standard output



Default Key-Value Separator

- Streaming API uses tab as the default separator between keys and values
 - Until the first tab is the key; rest of line is the value
 - The key is the line number the value is the line of text



Out of Stock Streaming Solution

- ◆ Our task can be accomplished using `grep`, `awk`, and `uniq`
 - No Java code or scripts to write!

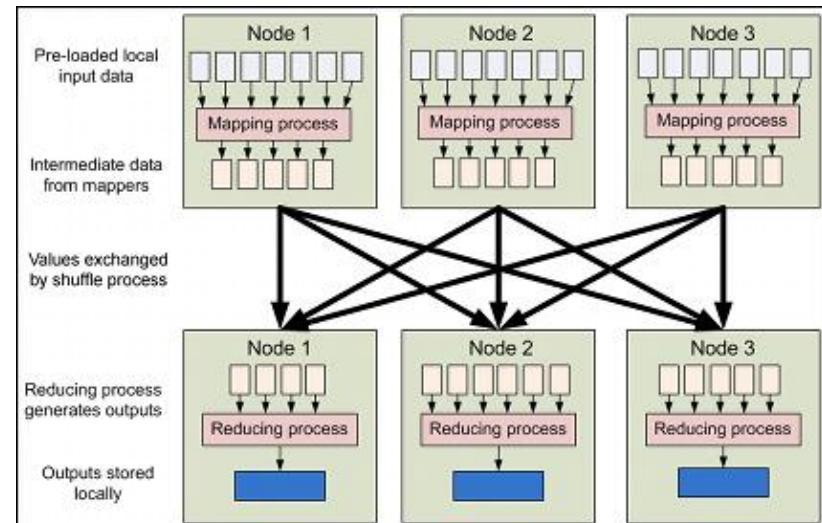
```
| grep -w OUTOFOFSTOCK | awk '{print $3"\t"1}'
```

Task	Input Type and Example	Description of Task	Output Type and Example
Map	String A line of the semi-structured log file	If line contains OUTOFOFSTOCK, split the line and pull out 3 rd field	<String, int> /product/32 1
Reduce	<String, 1>	Add up counts	/product/32 3 etc.

```
| uniq -c | awk '{print $2,$1}'
```

Preparing for MapReduce on Hadoop—I

- Remember that Hadoop streaming reads and writes to HDFS
 - Datasets must be copied to HDFS prior to running MapReduce jobs to process them
 - Reducers write their results to a directory specified in the MapReduce job
 - The directory will be created by Hadoop at the time the job is submitted
 - If the directory already exists, the MapReduce job will fail



Source: Horton Works

Preparing for MapReduce on Hadoop—II

- Given a script that we've tested locally, identify the parts that are the:
 - Mapper
 - Reducer
 - Performed by Hadoop

```
#!/usr/bin/bash
cat logfile | grep -w OUTOFSOCK | awk '{print $3}' |
sort | uniq -c | awk '{print $2 $1}' > output_file
```

Map

Reduce

- The bolded parts (`cat`, `sort`, `output`) will be performed by Hadoop

```
cat logfile | mapper | sort | reducer > output_file
```

- The parts we need to give to Hadoop streaming are:

- Mapper: `grep -w OUTOFSOCK | awk '{print $3}'`
- Reducer: `uniq -c | awk '{print $2 $1}'`

Submit a MapReduce Job to Hadoop

- ◆ To submit a MapReduce job to Hadoop, use the `hadoop jar` command
 - The `hadoop jar` command takes a Java archive (JAR) file as its argument

```
hadoop jar \
$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming*.jar \
-input /logfile.txt \
-output /tmp/output \
-mapper "grep -w OUTOFSTOCK" \
-reducer "uniq -c"
```

- ◆ The Hadoop streaming jar is a standard part of the Hadoop distribution
 - The input can be a file or a directory (in which case all files in that directory are processed), but it is usually a single file
 - Hadoop will not overwrite an existing output directory
 - ◆ In the above example, `/tmp/output` should NOT exist



Word Count 1 with Hadoop—I

- In Chapter 4, we executed the following script, `word_count_1_linux.sh`, to count words in `shakespeare.txt`

```
cat ~/roi_datasets/shakespeare.txt |  
    tr -s '[[[:punct:][:space:]]]' '\n' | sort | uniq -c
```

- Execute the following commands to perform the same task on Hadoop:

```
hadoop fs -copyFromLocal ~/roi_datasets/shakespeare.txt /  
  
hadoop fs -rm -r /output  
  
hadoop jar \  
/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.0.0.jar \  
-input /shakespeare.txt \  
-output /output \  
-mapper "tr -s '[[[:punct:][:space:]]]' '\n'" \  
-reducer "uniq -c"  
  
hadoop fs -cat /output/part-00000
```

Word Count 1 with Hadoop—II



- ❖ If the previous map reduce job is going to be executed frequently, it makes sense to create a script
- ❖ Execute the following command to view a candidate script
 - The expected output is on the next slide

```
cd ~/roi_hadoop/solutions  
cat word_count_1_hadoop.sh
```



Word Count 1 with Hadoop—III

- ◆ You should see the following:

```
#!/bin/sh
OUTDIR_REMOTE=/output
OUTDIR_LOCAL=./output

#copy the input (normally this will have to be done only once)
hadoop fs -copyFromLocal ~/roi_datasets/shakespeare.txt /
hadoop fs -rm -r $OUTDIR_REMOTE

#run
hadoop jar \
/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.0.0.jar \
-input /shakespeare.txt \
-output /output \
-mapper "tr -s '[:punct:] [:space:]' '\n'" \
-reducer "uniq -c"

# copy output from HDFS to local machine
rm -rf $OUTDIR_LOCAL
hadoop fs -copyToLocal $OUTDIR_REMOTE $OUTDIR_LOCAL
```



Word Count 1 with Hadoop—IV

- ❖ Execute the following to execute the word count job on Hadoop:

```
./word_count_1_hadoop.sh
```

- ❖ Execute the following to compute the word counts locally:

```
./word_count_1_linux.sh >> temp1
```

- ❖ Compare the word counts from the Hadoop job and the local script
 - The two output files will not have the same structure
 - But the word counts should generally match

```
grep -w king output/part-00000 temp1
```

```
output/part-00000: 388 king  
temp1: 388 king
```

- ❖ You should see:

- Try some other words, e.g., happy, peace, hungry



Word Count 2 with Hadoop—I

- ◆ In Chapter 4, we also executed the following more refined script, `word_count_2_linux.sh`, to count words in `shakespeare.txt`

```
cat ~/roi_datasets/shakespeare.txt |  
tail -n +182 |  
head --lines=-418 |  
tr [A-Z] [a-z] |  
tr ">:, [\]\.\*\;'\") (" " " |  
fmt -1 |  
sed '/^s*$/d' |  
sed 's/^[:blank:]*//; s/[:blank:]*$//' |  
sed 's/-//; s/\-$//; s/!$//' |  
sort |  
uniq -c
```

- ◆ Which parts of the script are the mapper and reducer components? _____
- ◆ Unfortunately, the Hadoop streaming API cannot accept arguments for the mapper and reducer options that contain the pipe symbol '|'
 - To get around this restriction, place bash shell pipelines in their own script file
 - These script files are passed to the mapper and reducer options as arguments



Word Count 2 with Hadoop—II

- Here is the mapper script, `word_count_2_hadoop_mapper.sh`
 - Note that this script must be executable
 - Note also that the script must read from `stdin` and write to `stdout`

```
#!/bin/bash

tail -n +182 |
head --lines=-418 |
tr [A-Z] [a-z] |
tr ">:, \[\].\*; '\\"?) (" " " |
fmt -1 |
sed '/^s*/d' |
sed 's/^[:blank:]*//; s/[:blank:]*$/'' |
sed 's/^--//; s/\-$//; s/!$/''
```



Word Count 2 with Hadoop—III

- Here is the driver script, `word_count_2_hadoop_driver.sh`, that invokes the streaming API
 - Note the new argument to `-mapper` and the new `-file` option

```
#!/bin/sh
OUTDIR_REMOTE=/output
OUTDIR_LOCAL=./output

#copy the input (normally this will have to be done only once)
hadoop fs -copyFromLocal ~/roi_datasets/shakespeare.txt /
hadoop fs -rm -r $OUTDIR_REMOTE

#run
hadoop jar \
/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.0.0.jar \
-input /shakespeare.txt \
-output /output \
-mapper word_count_2_hadoop_mapper.sh \
-file word_count_2_hadoop_mapper.sh \
-reducer "uniq -c" ←

# copy output from HDFS to local machine
rm -rf $OUTDIR_LOCAL
hadoop fs -copyToLocal $OUTDIR_REMOTE $OUTDIR_LOCAL
```



Career Home Runs with Hadoop—I

- ◆ In Chapter 4, we executed the following script, `career_home_runs_with_linux.sh`, to calculate the career home runs for Barry Bonds
 - The dataset used is `~/roi_datasets/baseball/Batting.csv`

```
cat ~/roi_datasets/baseball/Batting.csv |  
grep bondsba01 |  
awk -F, '{ s += $12 } END { print s }'
```

- ◆ Adapt this script so that it can run on Hadoop
 - Remember that the streaming API uses '\t' as the default field separator
 - Create a mapper, reducer, and driver file
 - Start by creating a very simple mapper and reducer
 - ◆ Test the driver with your simple mapper and reducer
 - ◆ Incrementally improve the mapper and reducer





Career Home Runs with Hadoop—II

- Here is a candidate solution for the driver: `career_home_runs_hadoop_driver.sh`

```
#!/bin/sh
OUTDIR_REMOTE=/output
OUTDIR_LOCAL=./output

#copy the input (normally this will have to be done only once)
hadoop fs -copyFromLocal ~/roi_datasets/baseball/Batting.csv /
hadoop fs -rm -r $OUTDIR_REMOTE

#run
hadoop jar \
/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.0.0.jar \
-input /Batting.csv \
-output /output \
-mapper career_home_runs_hadoop_mapper.sh \
-file career_home_runs_hadoop_mapper.sh \
-reducer career_home_runs_hadoop_reducer.sh \
-file career_home_runs_hadoop_reducer.sh

# copy output from HDFS to local machine
rm -rf $OUTDIR_LOCAL
hadoop fs -copyToLocal $OUTDIR_REMOTE $OUTDIR_LOCAL
```

Career Home Runs with Hadoop—III



- ◆ Here is a candidate solution for the mapper: `career_home_runs_hadoop_mapper.sh`

```
#!/bin/bash

tr -s ',' '\t' | grep -e 'bondsba01' -e 'aaronha01' -e 'ruthba01'
```

- ◆ Here is a candidate solution for the reducer: `career_home_runs_hadoop_reducer.sh`

```
#!/bin/bash

awk '{ sum[$1] += $12 } END { for (i in sum) print i, sum[i] }'
```

Chapter Concepts

MapReduce with Hadoop

Planning MapReduce

Streaming in Hadoop

➤ Advanced Streaming

Beyond Streaming with Java

The Java API

Chapter Summary

Python Scripts

- ◆ As was mentioned, the streaming API works with any program that reads `stdin` and writes to `stdout`
 - The program can be written in any language, e.g., Python
- ◆ There is one caveat though
 - UNIX commands such as `uniq` and `wc` are present on all machines in the cluster
 - The interpreter and libraries for the scripting language, e.g., Python, must be installed on all the Node Managers

A Python Map Script

- ◆ The following script, `startswith.py`, searches lines of text for words that begin with a specific pattern
 - All matches are sent to standard output
 - The script `startswith.py` needs to be executable (`chmod +x`)

```
#!/usr/bin/python
import sys
import re

if len(sys.argv) < 2:
    print "Usage: cat shakespeare.txt | startswith.py searchterm\n"
    sys.exit()
term = sys.argv[1]
startswith = re.compile("^" + term)
nonletters = re.compile("[^a-z A-Z]")

for line in sys.stdin:  # FROM STDIN
    line = line[:-1] # remove '\n'
    line = nonletters.sub("", line)
    words = line.split()
    for word in words:
        if startswith.match(word):
            print word  # TO STDOUT
```



Hadoop Driver for Python Script

- ◆ Here is the candidate driver, `startswith_hadoop_driver.sh`:

```
#!/bin/sh
OUTDIR_REMOTE=/output
OUTDIR_LOCAL=./output

#copy the input (normally this will have to be done only once)
hadoop fs -copyFromLocal ~/roi_datasets/shakespeare.txt /
hadoop fs -rm -r $OUTDIR_REMOTE

#run
hadoop jar \
/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.0.0.jar \
-input /shakespeare.txt \
-output $OUTDIR_REMOTE \
-mapper 'startswith.py love' \
-file 'startswith.py' \
-reducer 'cat'

# copy output from HDFS to local machine
rm -rf $OUTDIR_LOCAL
hadoop fs -copyToLocal $OUTDIR_REMOTE $OUTDIR_LOCAL
```

- ◆ Execute the driver `./startswith_hadoop_driver.sh`

Chapter Concepts

MapReduce with Hadoop

Planning MapReduce

Streaming in Hadoop

Advanced Streaming

➤ **Beyond Streaming with Java**

The Java API

Chapter Summary

Beyond Streaming

- ◆ We have seen two ways of writing MapReduce programs:
 - String together UNIX tools like grep, wc, etc. to yield Map and Reduce operations
 - Write scripts or programs in Perl, Python, C++, etc. and use these scripts as Map and Reduce
- ◆ Both of these techniques rely on the Streaming API
 - Reading from standard input and writing to standard output
 - Quite flexible and powerful
- ◆ So why do we need to go lower-level with Java?

Why Java?

- ◆ Hadoop is implemented in Java
- ◆ MapReduce jobs written in Java have access to features not available via the Streaming API including, in no particular order:
 - Custom Combiners
 - Custom Partition Algorithms
 - Binary Data
 - Use of non-HDFS data sources and sinks, e.g., HBase, Cassandra
 - Custom file formats
 - Custom data types for keys
 - Custom Counters
 - Higher performance
- ◆ A downside to coding in Java is that it presents a low-level, detailed API aimed at programmers

Chapter Concepts

MapReduce with Hadoop

Planning MapReduce

Streaming in Hadoop

Advanced Streaming

Beyond Streaming with Java

➤ **The Java API**

Chapter Summary

An Example Java MapReduce App

- ◆ The following Java MapReduce program finds the maximum number of words in a line by Shakespeare given that the line contains a given word
 - For example, of all the lines in Shakespeare that contain the word “hungry”, the longest one has 17 words
- ◆ To more effectively compare the streaming and Java APIs:
 - The Java program will only implement map and reduce functionality
 - No custom combiner, partitioner, etc.

Key-Value Pairs

- Here is how the Java program might be invoked in stand-alone mode, not on a Hadoop cluster

```
grepnumwords shakespeare.txt outputdir lovely fair hungry
```

- Example output:

```
lovely 17  
fair 15  
hungry 23
```

- Can you fill out this table? (remember that these are key-value pairs)

Task	Input Type and Example	Description of Task	Output Type and Example
Map			
Combine			
Partition			
Reduce			

MapReduce Operations

◆ Input to mapper:

QUEEN MARGARET. Bear with me; I am hungry for revenge,

- Key: ignored
- Value: a line of text

◆ Output of mapper:

hungry 10

- Key: text (search term)
- Value: integer (number of words in line)
- Description: strip out punctuation; if any word in the line matches the search term, print out the number of words in that line

◆ Input of reducer is the output of the mapper sorted:

hungry 10, 9, 17, 32, etc.

◆ Output of reducer is the maximum of the number of words:

hungry 32

Hadoop Writables

- ◆ There are Java classes for all our key and value types
 - String
 - Integer (or int, which is a primitive)
- ◆ Cannot use these in Hadoop
 - Hadoop serializes data differently from Java
- ◆ Will need to use Hadoop's writable classes
 - Text
 - IntWritable

Steps to Write MapReduce in Java

◆ Steps:

- Write a Mapper
- Write a Reducer
- Write a Main program to configure the job
- Submit the job to Hadoop

Step 1: Mapper

- ◆ Our Mapper extends Hadoop's abstract Mapper class

```
package com.roi.hadoop.grepnumwords;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
import java.util.regex.Pattern;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class SearchTermsMapper
    extends Mapper<Object, Text, Text, IntWritable> {

}
```

Input key: Object
Input value: Text

Output key: Text
Output value: IntWritable

Map Method

- ◆ The key method we override in a Mapper is the map method
 - Does the operation on <K1, V1>
 - Writes out <K2, V2>

```
@Override
public void map(Object ignoredKey, Text value, Context context)
    throws IOException, InterruptedException {
    // search for the words
    Set<String> words = findWords(value.toString());
    for (String term : getSearchTerms(context)) {
        if (words.contains(term)) {
            // word \t count=1
            context.write(new Text(term), new IntWritable(words.size()));
        }
    }
}
```

- Requires two methods:
 - ◆ `findWords()`: a way to find the words in a line
 - ◆ `getSearchTerms()`: a way to find the search terms specified by user

Finding Words in a Line

- ◆ Finding words in a line is pure Java code
 - For search efficiency in `map()`, we place the words in a Set

```
private final static Pattern NON LETTERS = Pattern.compile("[^a-zA-Z]");
private final static String SPACE = " ";
private final static String SPACES = " +";

public static Set<String> findWords(String line) {
    line = NON LETTERS.matcher(line).replaceAll(SPACE);
    line = line.toLowerCase(); // case-insensitive
    String[] words = line.split(SPACES);
    // for search efficiency, use a Set
    Set<String> wordSet = new HashSet<String>();
    for (String word : words) {
        wordSet.add(word);
    }
    return wordSet;
}
```

- ◆ But obtaining search terms requires sending along command-line parameters to Hadoop
 - Let's defer it for now

Step 2: Reducer

- ◆ Our Reducer has to extend Hadoop's abstract Reducer class

```
package com.roi.hadoop.grepnumwords;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class MaxCountReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {
}
```

Input key: Text
Input value: IntWritable

Output key: Text
Output value: IntWritable

- ◆ The key method to implement is called `reduce()`

The reduce Method

- ◆ In the `reduce()` method, we reduce the `<K2, List<V2> >` to `<K3, V3>`

```
public void reduce(Text key, Iterable<IntWritable> values, Context context)
                  throws IOException, InterruptedException {
    // compute the max for this key
    int max = 0;
    for (IntWritable val : values) {
        if (val.get() > max) {
            max = val.get();
        }
    }
    // write it out
    context.write(key, new IntWritable(max));
}
```

Step 3: Main

◀ Four key steps

```
package com.roi.hadoop.grepnumwords;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class Main {
    public static void main(String[] args) throws Exception {
        // (a) parse command-line

        // (b) set up parameters to Mapper and Reducer

        // (c) configure the job

        // (d) run the job
    }
}
```

3a: Parsing Command-Line

- ◆ To parse the command-line, use a Hadoop utility class

```
// parse command-line
Configuration conf = new Configuration();
args = new GenericOptionsParser(conf, args).getRemainingArgs();
if (args.length <= 2) {
    System.err.println("Usage: grepnumwords <in> <out> <term1> <term2>");
    System.exit(2);
}
```

- ◆ Tells Hadoop to pull out its parameters
 - For example, can use `-D property.name=property.value` to add or override Hadoop configuration setting

```
hadoop jar somejar.jar -D mapred.map.tasks=3
```

 - ◆ Note space between D and property name
 - ◆ `-Dproperty=value` would be a JVM setting, not a Hadoop one
- ◆ Once Hadoop has pulled out all its parameters, the rest are for our application

3b: Application Parameters

- ◆ Command-line parameters (here the search terms) have to be passed on to later stages (here to the Mapper):

```
// set up parameters to Mapper and Reducer
String[] searchTerms = new String[args.length - 2];
for (int i = 2; i < args.length; ++i) {
    searchTerms[i - 2] = args[i];
}
conf.setStrings("searchTerms", searchTerms);
```

- ◆ Can set a single primitive value
- ◆ The configuration acts like a HashMap
 - Provides access to configuration setting through the Context that is the third parameter to the `map()` and `reduce()` methods
 - Will discuss this shortly

3c: Configuring Job

- ◆ Configure the job as follows:

```
// configure the job
Job job = Job.getInstance(conf);
job.setJarByClass(Main.class);
job.setMapperClass(SearchTermsMapper.class);
job.setReducerClass(MaxCountReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

- Specify the jar that Hadoop needs to distribute to different nodes
- Specify the Mapper and Reducer classes
 - ➔ Hadoop will create these objects
- In Java, generics are only a compile-time convenience, so need to re-specify the key-value types here
- Tell Hadoop to look in input directory and write to output directory

3d: Launch Job

- ◆ Launch the job as follows:

```
// run the job  
boolean success = job.waitForCompletion(true);  
System.exit( success ? 0 : 1);
```

- Follow the UNIX convention of programs returning non-zero status on error

Using Command-Line Parameters

- Recall that we set command-line parameters in Main:

```
// set up parameters to Mapper and Reducer
String[] searchTerms = new String[args.length - 2];
for (int i = 2; i < args.length; ++i) {
    searchTerms[i - 2] = args[i];
}
conf.setStrings("searchTerms", searchTerms);
```

- We can get that back in the Mapper or the Reducer:

```
public void map(Object ignoredKey, Text value, Context context)
    throws IOException, InterruptedException {
    String[] searchTerms = context.getConfiguration().getStrings("searchTerms");
}
```



MapReduce in Java

- ◆ Open Eclipse by double clicking its desktop icon
 - When prompted, select the /home/student/roi_hadoop workspace
 - Open the grepnumwords project
- ◆ Examine the Java files under src/main/java
 - Open a file in an editor by double-clicking the file of interest
- ◆ Open grepnumwords.sh in an editor
 - Note the arguments to the hadoop jar command

```
#!/bin/sh
hadoop fs -copyFromLocal ~/roi_datasets/shakespeare.txt /shakespeare.txt
hadoop fs -rm -r /javaoutput

hadoop jar ~/roi_hadoop/solutions/grepnumwords/target/grepnumwords-1.0.0.jar \
    com.roi.hadoop.grepnumwords.Main /Shakespeare.txt /javaoutput hungry fair

hadoop fs -cat /javaoutput/*
```

- ◆ Execute the grepnumwords.sh script
 - How long are the longest lines containing the words hungry and fair?
 - _____

Chapter Concepts

MapReduce with Hadoop

Planning MapReduce

Streaming in Hadoop

Advanced Streaming

Beyond Streaming with Java

The Java API

➤ Chapter Summary

Chapter Summary

In this chapter, we have:

- ◆ Reviewed the data flow structure of MapReduce
- ◆ Designed MapReduce applications
- ◆ Run simple MapReduce jobs using the Hadoop Streaming API
- ◆ Used the Hadoop Java API for MapReduce



ROI TRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Module 2: Global Technical Analyst Program

CHAPTER 7: APACHE SPARK WITH JAVA

Chapter Objectives

In this chapter, we will:

- ◆ Present an overview of Apache Spark
- ◆ Investigate the ways in which Spark applications are executed
- ◆ Build powerful data processing applications with Spark
- ◆ Make incremental changes to data using RDDs
- ◆ Simplify complex operations with SQL
- ◆ React to new data using Streams

Chapter Concepts



Overview of Spark

Running Spark Applications

Java Development on Spark

Resilient Distributed Datasets

Spark SQL

Spark Streaming

Chapter Summary

What Is Spark?

- ◆ General purpose cluster computing framework
 - Implicitly parallel programming model
 - Abstracts parallelism away from developer
 - Highly fault tolerant
 - Open-source project from Apache
 - Often used in combination with Hadoop's HDFS
- ◆ Modular
 - Spark Core
 - Spark SQL
 - Spark MLlib
 - Spark Streaming
 - Spark GraphX
- ◆ Implemented in Scala
 - Python and Java APIs available

Chapter Concepts

Overview of Spark

➤ **Running Spark Applications**

Java Development on Spark

Resilient Distributed Datasets

Spark SQL

Spark Streaming

Chapter Summary

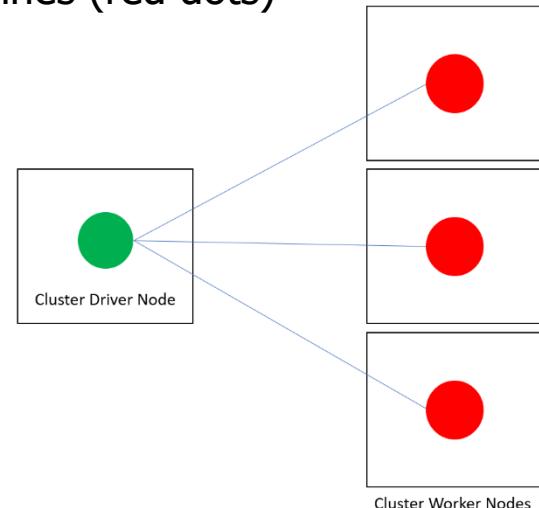
How Does Code Run on Spark?

- ◆ A Spark application consists of multiple processes
 - Spark Driver
 - Spark Executor
 - Cluster Manager
- ◆ Spark Driver controls the execution of the program
 - A single process for each application
 - Interacts with the Cluster Manager to create executors
 - Assigns tasks to the executors
 - Maintains state across the cluster
 - Handles node failures
- ◆ Spark Executor
 - Does the work assigned by the driver
- ◆ Cluster Manager
 - Assigns resources within the cluster
 - Has its own Cluster Driver and Cluster Worker nodes

The Cluster Manager

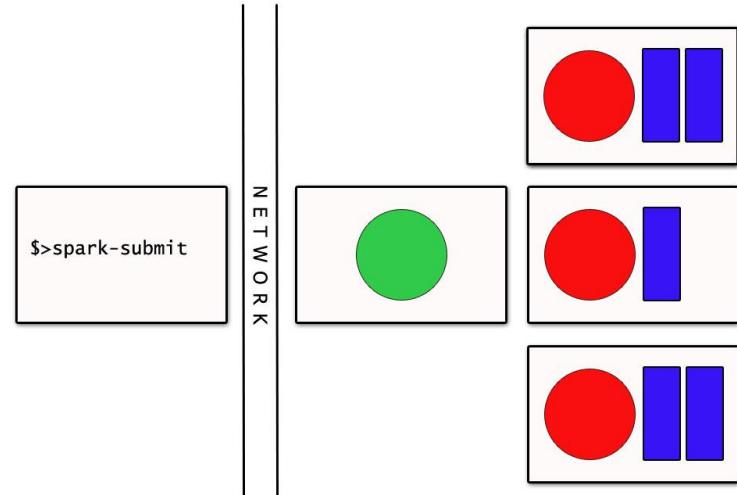
Cluster Manager

- Assigns resources within the cluster
- Has its own Cluster Driver and Cluster Worker nodes
 - ↳ Cluster Drivers will nominate a leader
 - ↳ Workers run as daemons processes on individual machines (red dots)
- Different managers can be configured
 - ↳ Build-in standard manager
 - ↳ YARN
 - ↳ Apache Mesos



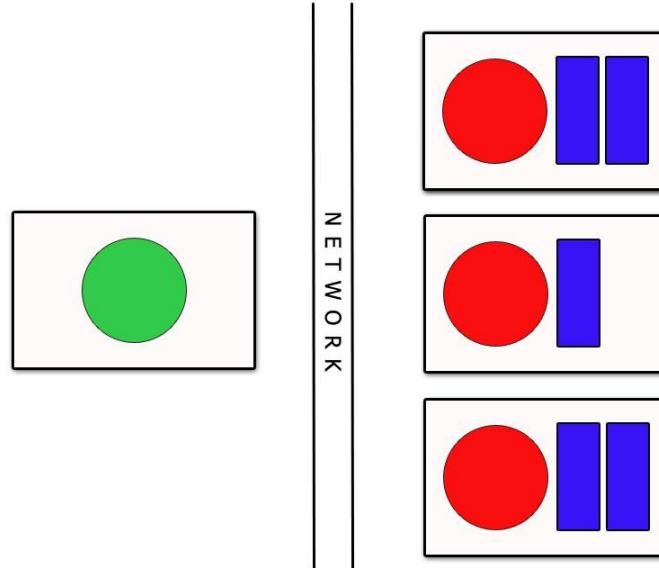
Running Application – Cluster Mode

- Spark driver is 'submitted' to cluster to be executed
 - Runs locally to the executors
 - Best performance
 - Most common in production environment
 - Default mode: master=cluster



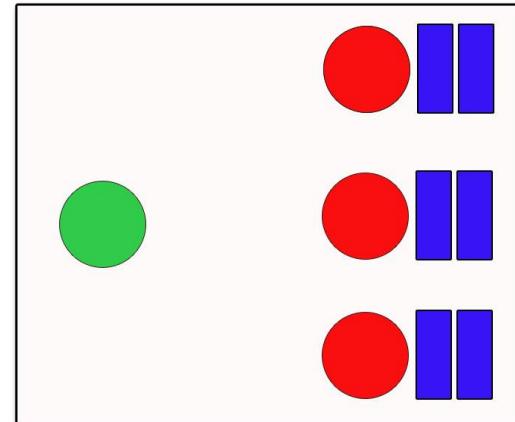
Running Application – Client Mode

- Spark driver is run on local machine
 - Communication overhead with executors
 - Works well if storing small results locally
 - Run with: master=client



Running Application – Local Mode

- ◆ Everything runs on single machine
 - Good for development
 - Run directly within your IDE
 - Achieves parallelism using threads and local cores
 - Don't use for production deployment
 - Run with: master=local



Running with Client or Cluster Mode

- ◆ Code must be distributed over the network to be run on cluster nodes
 - Packaged as a jar file
 - Maven will build this for you: use the package task
- ◆ Spark provides a convenient script for deploying jar files and running programs:

```
$> spark-submit  
      --class demos.HelloSpark  
      --deploy-mode cluster  
      --master spark://192.168.8.101:7077  
      demos-1.0.jar
```

Chapter Concepts

Overview of Spark

Running Spark Applications

➤ **Java Development on Spark**

Resilient Distributed Datasets

Spark SQL

Spark Streaming

Chapter Summary

Java API Setup

- Include Spark using Maven or Gradle

```
<dependencies>
  <dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.12</artifactId>
    <version>2.4.3</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

- Or use transitive dependencies to include more modules:

```
<!-- sql module includes core/hdfs/etc -->
<artifactId>spark-sql_2.12</artifactId>
```

- What does scope 'provided' mean?
 - Hint:* need to change this for local development

Hello World

```
public class HelloSpark {  
    public static void main(String[] args) {  
  
        //Start Spark  
        SparkSession session = SparkSession.builder()  
            .appName("Hello Spark")  
            .master("local")  
            .getOrCreate();  
        JavaSparkContext spark = new JavaSparkContext(session.sparkContext());  
  
        List<String> words = Arrays.asList("Hello", "World");  
  
        //Process some data  
        List<String> processed = spark.parallelize(words)  
            .map(String::toUpperCase)  
            .collect();  
  
        processed.forEach(System.out::println);  
  
        //Stop Spark  
        spark.stop();  
    }  
}
```

Start the Spark session

Java friendly session wrapper

This is where we do our work



Do Now!

- ◆ Start IntelliJ and open the `demo` project
- ◆ Examine the code in the `demo.HelloSpark` class file
 - Click to the left of the main method to run the program
 - Examine the output

What Just Happened?

- ◆ The Hello World application does the following:
 1. Connects to a Spark instance
 2. The list of Strings was parallelized across the nodes
 - a. Spark creates a directed acyclic graph (DAG) to model the tasks required of the workers
 3. Each of the workers was asked to perform some work
 - a. Converts the string to uppercase, as a function passed to `.map(...)`
 4. The results are collected into a List and printed to the local console
 5. The Spark instance is shut down
- ◆ What was the run mode: Cluster/Client/Local?



Do Now!

- ◆ Return to the code in the `demos.HelloSpark` class file
 - Comment out this line:
 - ◆ `//System.setProperty("spark.master", "local");`
 - Open the **Maven** tool window
 - ◆ Under lifecycle, click package to build a jar for this project
 - Open the **Project** tool window
 - ◆ Check that `demos-1.0.jar` has been created in the target folder
 - Right-click the target folder and select **Open in Terminal**
 - In the terminal window, type the command: `spark-master`
 - ◆ In Chrome, visit: `localhost:8080` and note the **Spark Master URL**
 - ◆ `spark://xxx.xxx.xxx.xxx:7077`
 - In notepad++ edit the following file and set the spark URL to the one noted above
 - ◆ `C:\Java\spark-2.4.3-bin-hadoop2.7\bin\spark-worker.bat`
 - In the terminal window, type the command: **spark-worker**
 - ◆ Refresh the Spark web page in Chrome and note the worker process in the Workers section
 - In the terminal window, submit your jar to the cluster:
`spark-submit --class demos.HelloSpark --deploy-mode cluster --master <your spark master url> cs-demos-1.0.jar`
 - Can you find the output for the driver process in the web console?

Chapter Concepts

Overview of Spark

Running Spark Applications

Java Development on Spark

➤ **Resilient Distributed Datasets**

Spark SQL

Spark Streaming

Chapter Summary

Resilient Distributed Dataset – RDD

- ◆ Resilient Data Set
 - The building blocks of Spark
 - Used to split datasets into smaller chunks
 - Supports parallel operations
 - Fault tolerant
- ◆ Two ways to create an RDD
 1. Convert data in memory
 2. Load from external resource
 - Local file system (shared drives better)
 - HDFS
 - See Hadoop's `InputFormat` interface

Create an RDD from Java Collection

- ◆ Create an RDD in memory
 - Parallelizing an existing collection in your driver program
 - For example:
 - ◆ `JavaRDD<String> lines = spark.parallelize(words)`

Create an RDD from External File

- When working on 'big data' load from external resource(s):
 - Will be partitioned across the cluster
 - Supports wildcards and compressed format files

```
// load a String for each line of the file (128MB chunks from HDFS)
JavaRDD<String> lines = sc.textFile("data.txt");

// combines all lines from all files
JavaRDD<String> allLines = sc.textFile("data-1.txt", "data-2.txt");

// filename -> content (zipped)
JavaPairRDD<String, String> files =
    spark.wholeTextFiles("hdfs://localhost:19000/trades/*.gz");

// load binary data
JavaPairRDD<String, PortableDataStream> images =
    spark.binaryFiles("file:///c:/images/*.gif");
```

Operating on RDDs

- ◆ Pass functions to the RDD methods

```
List<Integer> data = Arrays.asList(1, 2, 3, 4, 5);  
  
//optionally specify number of partitions  
JavaRDD<Integer> distData = sc.parallelize(data, 10);  
  
// function is serialized and executed in parallel  
distData.reduce((a, b) -> a + b)
```

Writing Functions

- ◆ Can pass anonymous inner classes

```
JavaRDD<String> lines = sc.textFile("data.txt");

JavaRDD<Integer> lineLengths = lines.map(new Function<String, Integer>() {
    public Integer call(String s) { return s.length(); }
});

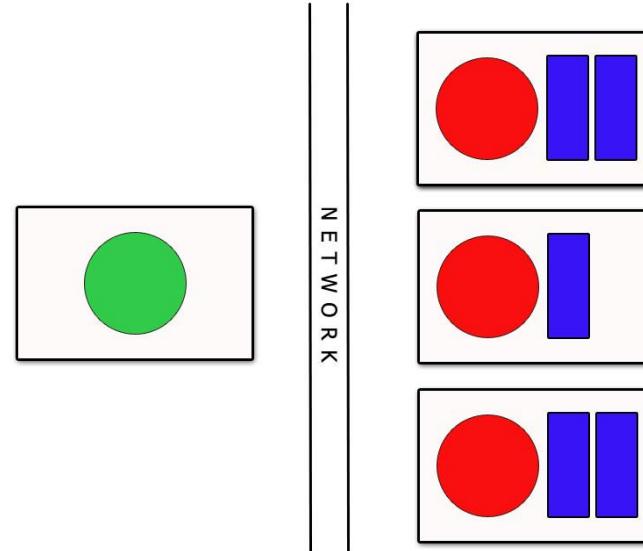
JavaRDD<Integer> numChars = lines.reduce(new Function<Integer, Integer>() {
    public Integer call(Integer a, Integer b) { return a + b; }
});
```

- ◆ Much more concise to use Java 8 lambda syntax

```
JavaRDD<String> lines = sc.textFile("data.txt");
long numChars = lines.map(String::length).reduce(Integer::sum).longValue();
```

Operating on RDDs

- Functions are serialized and sent to the executor nodes
 - Used to work on chunks of data in parallel
 - Enables **massive** data throughput



What Drivers Do

- ◆ **Drivers** translate the logic of the application into a set of **Tasks** which are processed on the **Executors**
 - The translation process starts with the creation of a directed acyclic graph which is a **logical model** of the application steps
 - The logical model is converted into a **physical execution plan**
 - The execution plan is optimized and partitioned into a set of **stages**
 - The stages are broken down into Tasks
 - Tasks are executed on the Executors
- ◆ The execution of the plan requires:
 - Data to be loaded into RDD partitions
 - Your data operation functions to be sent across the network
 - ◆ Used to **transform** the RDDs

Types of RDD Operation

◆ **Transform** – produces a new RDD

- Example: `incrementedRdd = inputRdd.map(i -> i + 1)`
- Evaluated lazily
 - ↳ Keeps a note of the mapping function
 - ↳ Only computed when an action requires the result

◆ **Actions** – return a result to the driver program

- Requires results from all transformations
- Examples: `rdd.count()`, `rdd.reduce()`

◆ **Do now!** Identify the transform and the action functions:

```
JavaRDD<String> lines = sc.textFile("data.txt");
JavaRDD<Integer> lineLengths = lines.map(s -> s.length());
int totalLength = lineLengths.reduce((a, b) -> a + b);
```

◆ See RDD programming guide for details of all transformations and actions (<https://spark.apache.org/docs/latest/rdd-programming-guide.html>)

RDD Caching

- ◆ By default all RDDs are recomputed each time an action is performed
- ◆ Can cache results for better performance too
 - Either to disk
 - Or to memory
- ◆ Example, to be able to reuse the line lengths later:

```
JavaRDD<String> lines = sc.textFile("data.txt");
JavaRDD<Integer> lineLengths = lines.map(s -> s.length());
int totalLength = lineLengths.reduce((a, b) -> a + b);
lineLengths.persist(StorageLevel.MEMORY_ONLY());
```

Distributed Programming Gotcha – Q1

- ◆ Remember, you are working in a distributed environment
 - Spark makes it much simpler
 - Easy to make mistakes
- ◆ What is the problem with this code?

```
// print data.txt to the console
JavaRDD<String> lines = sc.textFile("data.txt");
rdd.foreach(i -> {
    System.out.println(i);
});
```

Distributed Programming Gotcha – A1

- ◆ Executed locally with a single thread the code will work just fine
 - Print out each line of the file
- ◆ In a cluster, multiple processes will work on the file
 1. The lines are printed on the worker nodes
 2. There is no guarantee of ordering when printing
- ◆ Need to do something like this:

```
// the collect action brings the results into the driver
JavaRDD<String> lines = sc.textFile("data.txt");
rdd.collect().foreach(i -> {
    System.out.println(i);
});
```

- ◆ Or, if there are too many results:

```
rdd.take(1000).foreach(i -> { System.out.println(i); });
```

Distributed Programming Gotcha – Q2

- ❖ What is the problem with this code?

```
// calculate the total of the numbers in the RDD  
AtomicInteger counter = new AtomicInteger(0);  
rdd.foreach(x -> counter.getAndAdd(x));  
System.out.println("Counter value: " + counter);
```

Distributed Programming Gotcha – A2

- ◆ The lambda function will be serialized and sent to worker nodes
 - Any scoped variables are **copied** and sent too
 - Executors will update their copy of the counter only
 - ◆ No changes made in the driver program
 - ◆ In fact, Spark documentation states that the result is *undefined*
- ◆ Remember to use an action to get the results back to the driver
- ◆ Better to do this:

```
Long counter = rdd.reduce((a, b) -> a + b).longValue();  
System.out.println(counter);
```

Shared Variables – Accumulators

- ◆ It is possible to pass a variable for nodes to update
- ◆ The shared counter example could make use of an Accumulator:

```
LongAccumulator accum = spark.sc().longAccumulator();  
rdd.foreach(accum::add);  
long accumChars = accum.value();
```

- ◆ Only supports additive operations
- ◆ Results from all executors are gathered locally

Shared Variables – Broadcast

- ◆ Broadcast Variables can be sent to all executors
 - Read-only
 - Useful for sending large chunks of data to the executors
 - Saves overhead of repeatedly sending local state in lambdas
 - More efficient serialization
 - Happens once only

Potentially large chunk
of data here

```
//the phrase is sent once to all nodes
Broadcast<String> phrase = spark.broadcast("to be or not to be");

//can be read in the executors here
lines.filter(line -> line.contains(phrase.getValue())).count();
```

Key/Value Pairs

- ◆ Many mapping operations require a key/value or 'tuple' of data
- ◆ Scala provides a Tuple class for this
 - `scala.Tuple`
 - Exposed by Spark's Java API
 - *Note:* Scala is a JMV language compatible with Java
- ◆ Example: Can you see what this map reduce operation calculates?
 - Name the transformation
 - Name the action

```
JavaRDD<String> lines = sc.textFile("data.txt");

JavaPairRDD<String, Integer> pairs = lines.mapToPair(s -> new Tuple2(s, 1));

JavaPairRDD<String, Integer> counts = pairs.reduceByKey((a, b) -> a + b);
```

Data Shuffling

- ◆ Spark must sometimes *shuffle* data across nodes
 - To keep load balanced
 - As a result of operations that work on the whole dataset
- ◆ Shuffling is expensive
 - Requires moving data around between partitions and machines
 - ◆ In memory, disc, network
- ◆ `reduceByKey` creates new RDDs with String keys and line counts
 - Not all values for a single key will reside on the same partition
 - To guarantee new 'counts' RDD is distributed fairly, Spark will:
 - ◆ Read all key values from all nodes
 - ◆ Gather results from each node and bring together for new RDD
 - This is the shuffle
- ◆ Which operations cause shuffling?
 - `xxxByKey` (not counting), `repartition`, `coalesce`, `cogroup`, `join`

Chapter Concepts

Overview of Spark

Running Spark Applications

Java Development on Spark

Resilient Distributed Datasets

➤ **Spark SQL**

Spark Streaming

Chapter Summary

Spark SQL

- ◆ Provides a SQL table abstraction over RDDs
 - Uses column names to identify data fields
 - Doesn't replace RDD
 - Preserves RDD characteristics
 - ◆ Immutable
 - ◆ Distributed
- ◆ Imposes structure onto a distributed collection of data

Input Data Formats

- ◆ Data can be loaded from many formats:
 - JSON, Hive, JDBC, CSV
- ◆ We will use JSON to demonstrate
- ◆ Spark makes some changes to standard JSON
 - One record per line
 - ↳ Can be parsed/loaded in parallel
 - ↳ No need to load complete JSON document
 - ↳ Can be written by line too
 - ↳ Don't format your JSON!

```
// Regular JSON
[ { "name" : "Steve" } ,
  { "name" : "Andy" , "age":30} ,
  { "name" : "Rick" , "age":19} ]
```

```
// Spark JSON
{ "name" : "Steve" }
{ "name" : "Andy" , "age":30}
{ "name" : "Rick" , "age":19}
```

Loading Data and the DSL

- ◆ Data is loaded via the SparkSession object
 - Stored as **Dataset<Row>**
 - Manipulated using domain specific language (DSL)

```
//load the people file
Dataset<Row> ds = session
    .read()
    .json("src/main/resources/people.json");

//show the table
ds.show();

// Select everybody, but increment the age by 1
ds.select(col("name"), col("age").plus(1)).show();
```

age	name
null	Steve
30	Andy
19	Rick

age	name
null	Steve
31	Andy
20	Rick

Running SQL Queries

- Spark SQL provides Structured Query Language interface
 - Convert the Dataset internally as a table
 - Run queries
 - Most SQL query features available, including joins

```
Dataset<Row> ds = session
    .read()
    .json("src/main/resources/people.json");

//register the dataset as a table
ds.createOrReplaceTempView("people");

//run SQL queries
session.sql("SELECT name, age from people where age is
not null sort by age ASC").show();
```

name	age
Rick	19
Andy	30

DataFrames

- ❖ Create Dataset of POJO for increased type safety
 - Compiler catches problems earlier

```
Dataset<Row> ds = session
    .read()
    .json("src/main/resources/people.json");

//convert to a DataFrame / AKA Dataset<Person>
Dataset<Person> pd = ds.as(Encoders.bean(Person.class));
```

Spark SQL vs. RDD

- ◆ Dataset interface provides higher-level abstraction for manipulating data
- ◆ RDDs provide low-level interface for working with data
- ◆ Can still access the RDD (or JavaRDD) when required

```
Dataset<Row> ds = session
    .read()
    .json("src/main/resources/people.json");

//convert to a DataFrame / AKA Dataset<Person>
Dataset<Person> pd = ds.as(Encoders.bean(Person.class));

//or us the RDD if you need to
JavaRDD<Person> rdd = pd.javaRDD();
List<Person> people = rdd.map(...).filter(...).collect();
```

Chapter Concepts

Overview of Spark

Running Spark Applications

Java Development on Spark

Resilient Distributed Datasets

Spark SQL

➤ **Spark Streaming**

Chapter Summary

Spark Streaming

- ◆ Streams are an important design pattern
 - Asynchronous message passing
 - Highly scalable
 - Fault tolerant
- ◆ Spark has a streaming API
 - Integrates with external systems
 - ↳ Files, network sockets, Kafka, etc.
 - Integrates with other Spark APIs
 - ↳ SQL, Machine Learning/MLib



Spark Streaming – Java Module

- ◆ Add a project dependency for Spark Streams

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-streaming_2.12</artifactId>
  <version>2.4.3</version>
  <scope>compile</scope>
</dependency>
```

- ◆ And any additional modules you need

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-streaming-kafka-0-10_2.12</artifactId>
  <version>2.4.0</version>
</dependency>
```

Spark Streaming – Java Code

- ◆ Create a `JavaStreamingContext` and connect to a stream of data
 - Application thread must wait for termination
 - When do you think the RDD data is processed?

```
SparkConf conf = new SparkConf().setAppName("StreamFromFile");

JavaStreamingContext jssc = new JavaStreamingContext(conf, Durations.seconds(5));

JavaDStream<String> lines = jssc.textFileStream("src/main/resources/streaming");

JavaDStream<String> words = lines
    .flatMap(x -> Arrays.asList(x.split(" ")).iterator());

words.print();

jssc.start();
jssc.awaitTermination();
```

Streaming Lines from Files

- ◆ To stream from a file:
 - Use the directory or a wildcard URL to identify the files of interest
 - Load big data files directly from HDFS
 - As new files are added their lines will be passed to the `DStream`
 - ◆ Modifications to the files are **NOT** loaded

```
//from previous slide...

JavaDStream<String> lines = jssc.textFileStream("hdfs://namenode:8040/logs/2017/*");

JavaDStream<String> words = lines
    .flatMap(x -> Arrays.asList(x.split(" ")).iterator());

words.print();
```



Do Now! File Streams

- ◆ Start IntelliJ
 - Stop any Java processes running in IntelliJ
 - Open the `demo` project
 - If the following folder does not exist, create it:
 - ◆ `src/main/resources/streaming`
- ◆ Examine the code in the `demos.StreamFromFile` file
 - Click to the left of the `main` method to run the program
 - Examine the output
 - Copy a text file to the directory being monitored and test the behavior
 - ◆ What happens if you modify the file?



Do Now! Socket Streams

- ◆ Start IntelliJ
 - Stop any Java processes running in IntelliJ
 - Open the demo project
- ◆ Examine the code in the `utils.ChatterboxServer` file
 - Click to the left of the main method to run the program
 - This code runs a simple chat server on port 9000
- ◆ Examine the code in the `demos.StreamFromSocket` file
 - Click to the left of the main method to run the program in the **debugger**
 - Once started, right-click the debug tab and move to **Bottom Right**
- ◆ From the console window for the chat application:
 - Type a message and press **Enter**
 - Check that the message has been received in the stream
 - Is there much of a delay?
 - Experiment stopping/starting these processes
 - ◆ Do they recover gracefully?

Processing Streams

- Spark translates the streams into batches of RDDs for processing
 - AKA DStreams
- Batches are evaluated according to a timeout you specify
 - This can be tuned to meet your requirements

```
JavaStreamingContext jssc = new JavaStreamingContext(conf, Durations.seconds(5));
```



Chapter Concepts

Overview of Spark

Running Spark Applications

Java Development on Spark

Resilient Distributed Datasets

Spark SQL

Spark Streaming

 **Chapter Summary**

Chapter Summary

In this chapter, we have:

- ◆ Presented an overview of Apache Spark
- ◆ Investigated the ways in which Spark applications are executed
- ◆ Built powerful data processing applications with Spark
- ◆ Made incremental changes to data using RDDs
- ◆ Simplified complex operations with SQL
- ◆ Reacted to new data using Streams



ROI TRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Module 2: Global Technical Analyst Program

CHAPTER 8: INTRODUCTION TO MACHINE LEARNING

Chapter Summary

In this chapter, we will:

- ◆ Give a high-level overview of machine learning
- ◆ Identify the categories of algorithms available
- ◆ Give real-world examples of where Machine Learning (ML) can be applied
- ◆ Present an example using Spark MLlib

Chapter Concepts



Overview of Machine Learning

Supervised Learning

Unsupervised Learning

Example: Movie Recommender

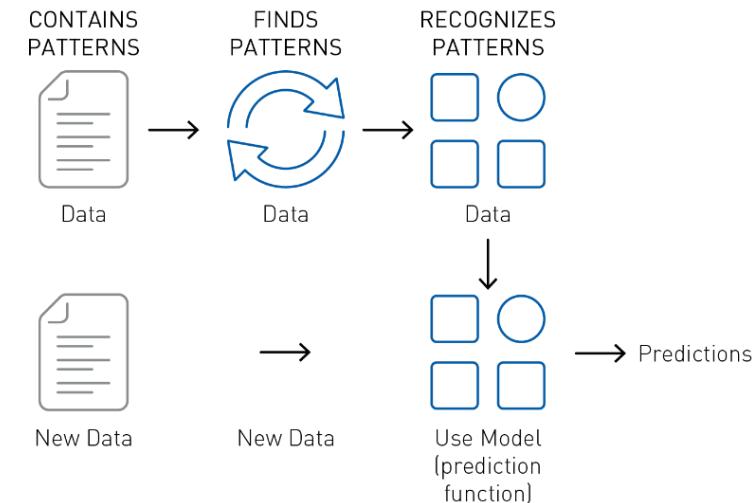
Chapter Summary

What Is Machine Learning?

- ❖ What is Machine learning?
 - Algorithms and statistical models which can perform a given task
 - No need to write programs to perform specific tasks
 - Relies on identifying patterns and inferring results from big data sets
 - ❖ An area of Artificial Intelligence (AI)
 - A computer system that mimics natural intelligence
 - ❖ Relies on statistical modeling of pre-existing data
 - Detects patterns
 - Makes deductions given further inputs
 - ❖ What applications have you used that employ ML?
-

Machine Learning Phases

- ◆ Phases in machine learning:
 - **Data Discovery:** analysis on historical data and model selection
 - **Training:** uses sample data to 'train' the model
 - **Using the model:** apply the model to new data
- ◆ Models should be continuously monitored and updated with new data



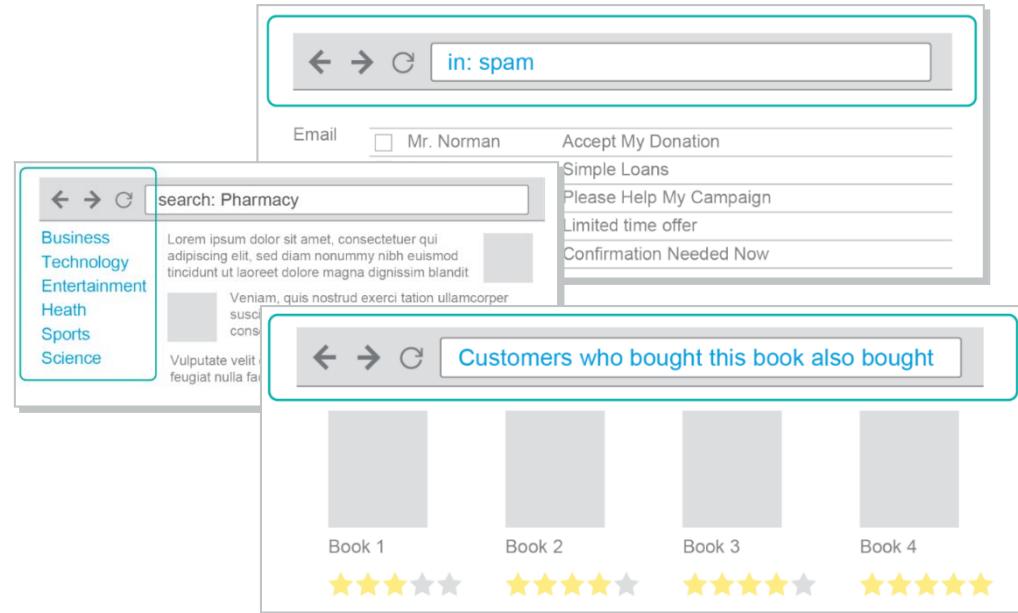
Choosing an Algorithm 1 – Your Intent

- ◆ There are many different ML algorithms
 - Rely on complex statistical analysis and modeling techniques
 - Challenge can be in choosing the best algorithm for your use case

- ◆ Classification

- ◆ Clustering

- ◆ Prediction



Choosing an Algorithm 2 – Your Data

- ◆ There are (generally) two types of learning model

- **Supervised**

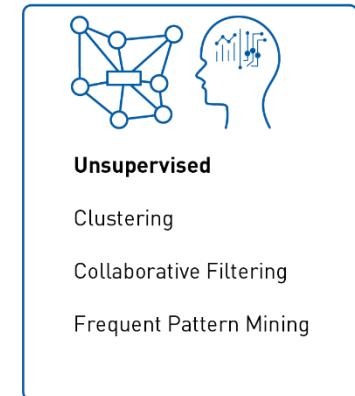
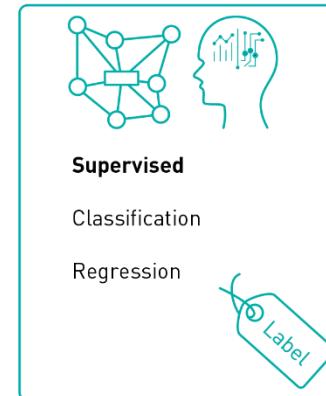
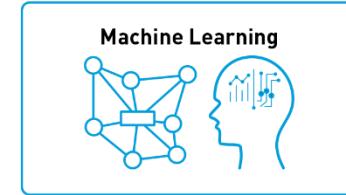
- ◆ Use metadata to label the data
 - Star ratings by users
 - Marked as spam
 - Handwriting recognition

- **Unsupervised**

- ◆ Identifies patterns in data
 - Predict forest fires
 - Shopping trends

- Types can be combined

- ◆ **Reinforcement** can be used to provide feedback on results that improve the model



Chapter Concepts

Overview of Machine Learning

➤ **Supervised Learning**

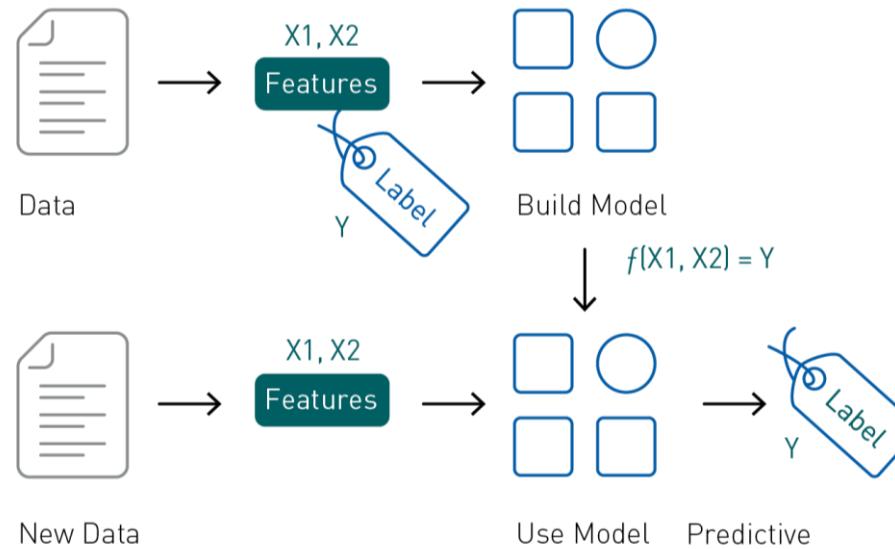
Unsupervised Learning

Example: Movie Recommender

Chapter Summary

Supervised Learning

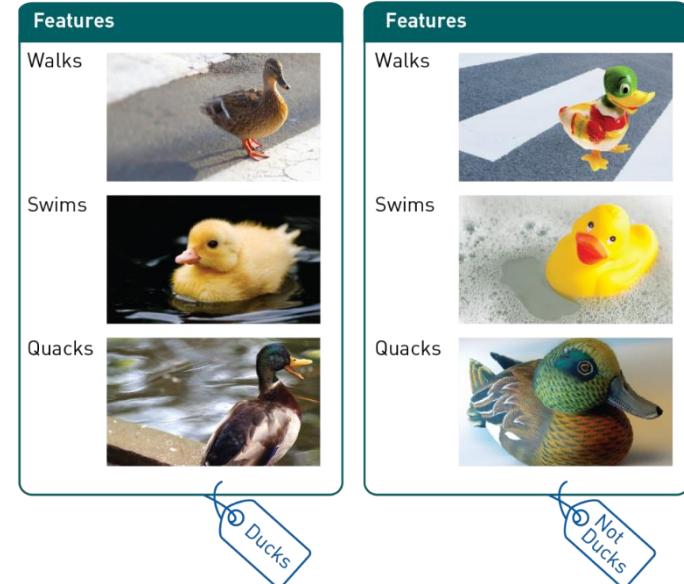
- Supervised algorithms use labeled data in which both the input and target outcome, or label, are provided to the algorithm



Classification

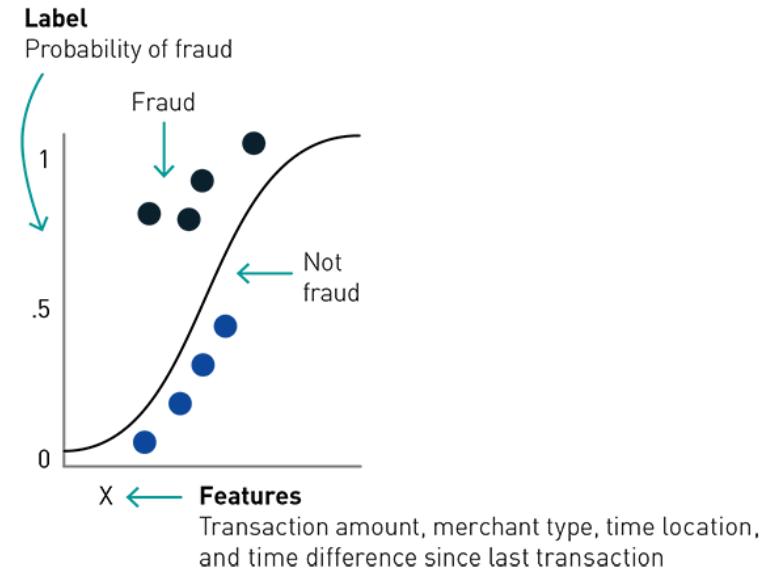
- ◆ Used to classify input data
 - Identify characteristics in your data that can train the model
 - Look for the “if’s”
 - ↳ If it walks like a duck...
 - ↳ If it swims like a duck...
 - ↳ If it quacks like a duck?
 - Label the input data and identify an output/classification “label”
 - ↳ It's a duck!
 - ↳ Image recognition
 - ↳ Handwriting
- ◆ Spark classification algorithms:
 - Logistic regression
 - Decision tree classifier
 - Random forest classifier
 - More...

If it walks/swims/quacks like a duck ... then it must be a duck.



Regression

- ◆ Gives a confidence score for input data fitting the model
- ◆ Statistical regression model plots likelihood of fit to previous data
 - Can have any number of inputs



Chapter Concepts

Overview of Machine Learning

Supervised Learning

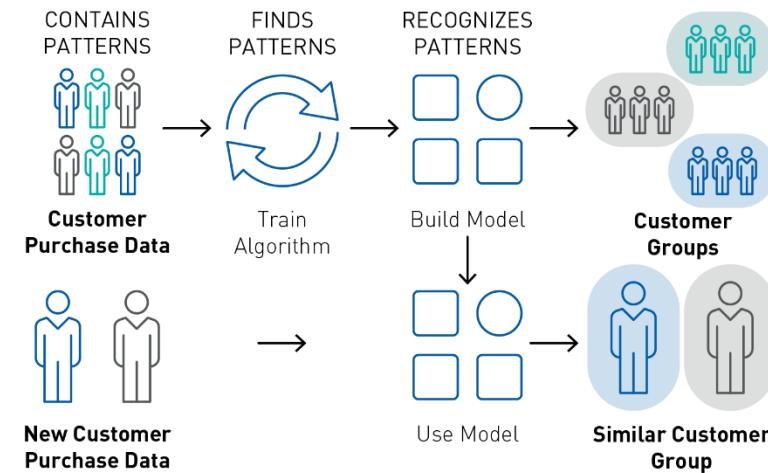
► **Unsupervised Learning**

Example: Movie Recommender

Chapter Summary

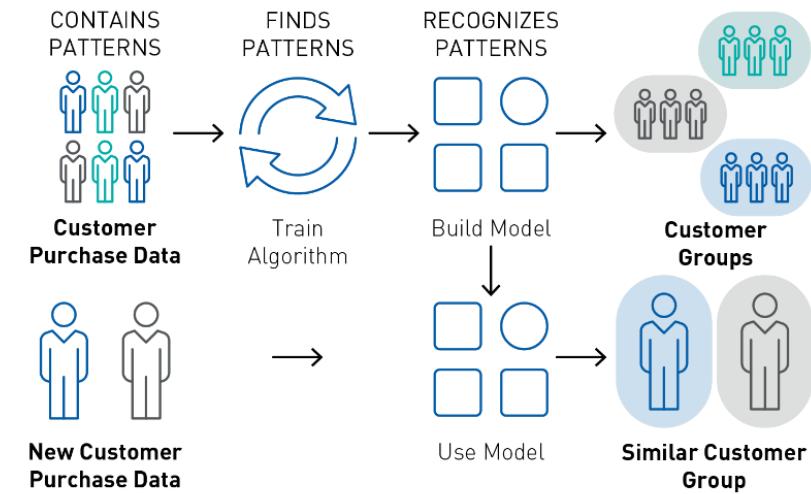
Unsupervised Learning

- ◆ Unsupervised learning algorithms
 - Discover similarities, or regularities, in the input data
- ◆ An example of unsupervised learning is grouping similar customers, based on purchase data
 - Online supermarket data
 - Amazon
 - Netflix
- ◆ Can be further classified as:
 - **Clustering**
 - **Filtering/Recommenders**
 - ↳ **Collaborative**
 - ↳ **Content-based**



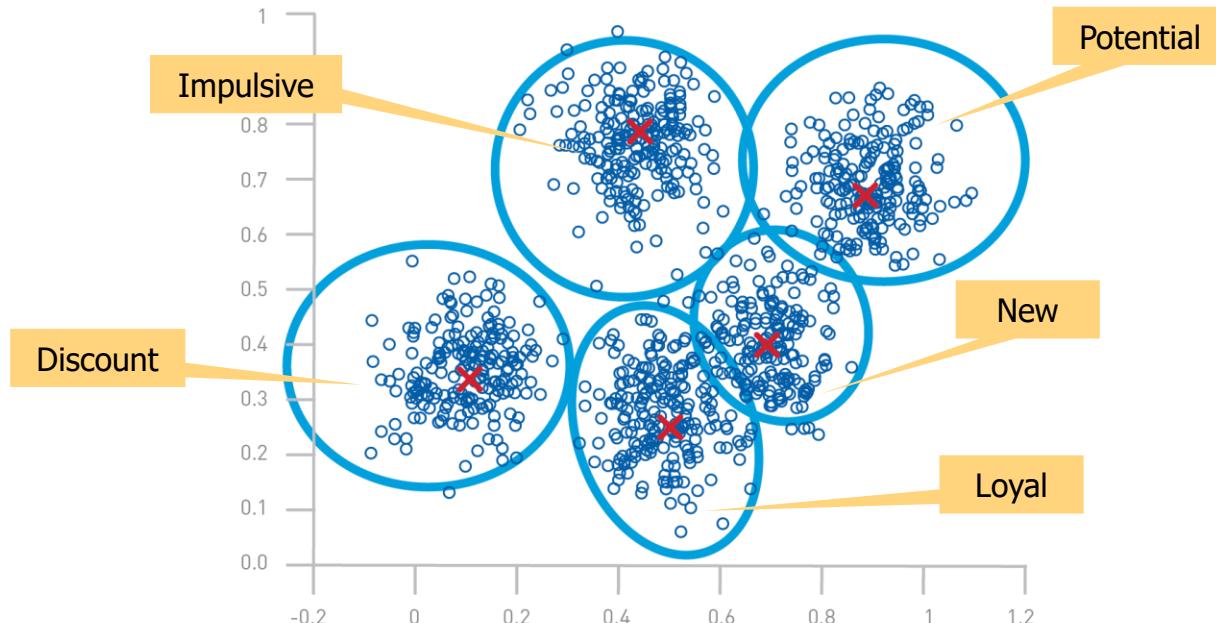
Clustering Algorithms

- Identify clusters of data given the input labels
 - E.g., types of customer for targeting advertisements
 - Potential customer
 - New customer
 - Impulsive customer
 - Discount customer
 - Loyal customer



Clustering with K-Means

- Spark provides a K-Means implementation
- Groups data into K clusters



Filtering/Recommender Algorithms

◆ Content-Based Filtering

- Utilize a series of discrete characteristics of an item in order to recommend additional items with similar properties
- Need to identify classifier for similarity
- No historical user data required
- Can be expensive to calculate prediction model

◆ Collaborative Filtering

- **Collaboration:** based upon information about *many users*
 - Identify users with similar preferences
- **Filtering:** priorities choices preferred by similar users
- Example use cases:
 - Suggest movies based upon your ratings
 - Suggest shopping items based upon similar purchases by other users

Chapter Concepts

Overview of Machine Learning

Supervised Learning

Unsupervised Learning

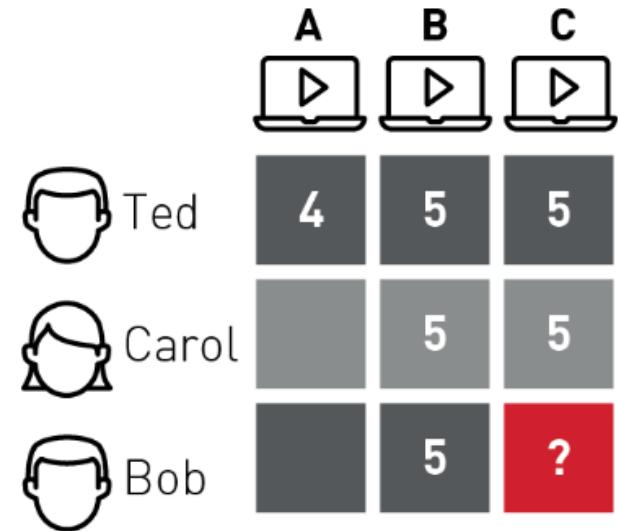


Example: Movie Recommender

Chapter Summary

Movie Recommender

- ◆ Our data contains information from **many users**
 - ◆ We want to **filter** the movies on offer to only offer most relevant
 - ◆ Collaborative Filtering!
 - ◆ Example: a movie recommender
 - Ted and Carol like movies B and C
 - Bob likes movie B
 - What other movie might Bob like?
-



Alternating Least Squares

- ◆ Netflix set a challenge to write a movie prediction algorithm
 - \$1M USD prize
- ◆ Training data provided
 - 100,480,507 ratings
 - 480,189 users
 - 17,770 movies
- ◆ Submitted algorithms ranked against the true ratings
 - I.e., Distance from actual preference data
- ◆ Winning algorithm: **Alternating Least Squares** ALS
 - Solves sparse matrix problems
 - Scales well to large data sets
- ◆ Uses multiple gradient descent algorithms internally
 - Optimized in Spark ML core libraries
- ◆ Implementation provided in Spark MLib

Do Now!



- ◆ Start IntelliJ and open the `demo` project
- ◆ Examine the code in the `demo.mllib.JavaALSExample` class file
 - Click to the left of the main method to run the program
 - Examine the output

Chapter Concepts

Overview of Machine Learning

Supervised Learning

Unsupervised Learning

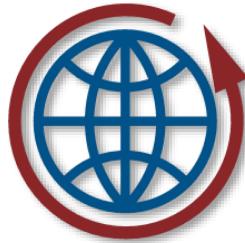
Example: Movie Recommender

Chapter Summary

Chapter Objectives

In this chapter, we have:

- ◆ Given a high-level overview of machine learning
- ◆ Identified the categories of algorithms available
- ◆ Given real-world examples of where ML can be applied
- ◆ Presented an example using Spark MLib



ROI TRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Module 2: Global Technical Analyst Program

CHAPTER 9: CLOUD COMPUTING

Chapter Objectives

In this chapter, we will:

- ◆ Define cloud computing
- ◆ Explore characteristics of clouds
- ◆ Identify cloud service and deployment models
- ◆ Explore cloud-based options for cluster computing

Chapter Concepts

► Defining Cloud Computing

Cloud Service Models

Cloud Deployment Models

Cluster Computing

Chapter Summary



What Is Cloud Computing?

- ◆ Cloud computing often means different things to different people
 - Depending on their experience/context
- ◆ What is cloud computing to you?



How do you define it?

What Is Cloud Computing? (continued)

- ◆ Many definitions exist for cloud computing:

“A style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service to external customers using Internet technologies.”

—Gartner Report

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

—www.nist.gov

What Is Cloud Computing? (continued)

- ◆ Main goal of cloud computing is to allow IT resources to be delivered and consumed as a service
 - The same way services like electricity, telephone, etc. are delivered and consumed
 - Allows organizations to focus on their core business
 - ◆ And not become experts in deploying, managing, maintaining IT infrastructures
- ◆ The simplest definition of cloud computing:



Cloud Computing = IT as a Utility

Key Cloud Characteristics

- ◆ Massive scalability and rapid elasticity
 - To the point that resources appear to be unlimited
 - Including compute, storage, and network capacity
- ◆ Measured service, or “pay as you go” only for what is needed
- ◆ Pooling of shared resources—networks, servers, storage, applications
 - Requires a **multi-tenancy** model
- ◆ On-demand, self-service
 - Increase or decrease resources as needed
 - Provision without requiring human interaction with service provider
- ◆ Capabilities accessed over the network
- ◆ Capital expenditure is often converted to operational expenditure

Benefits of Cloud Computing

- ◆ Potential benefits of cloud computing may include:
 - Performance
 - Scalability
 - Elasticity
 - Fault tolerance and high availability
 - Agility
 - Security
 - Financial

Chapter Concepts

Defining Cloud Computing

➤ **Cloud Service Models**

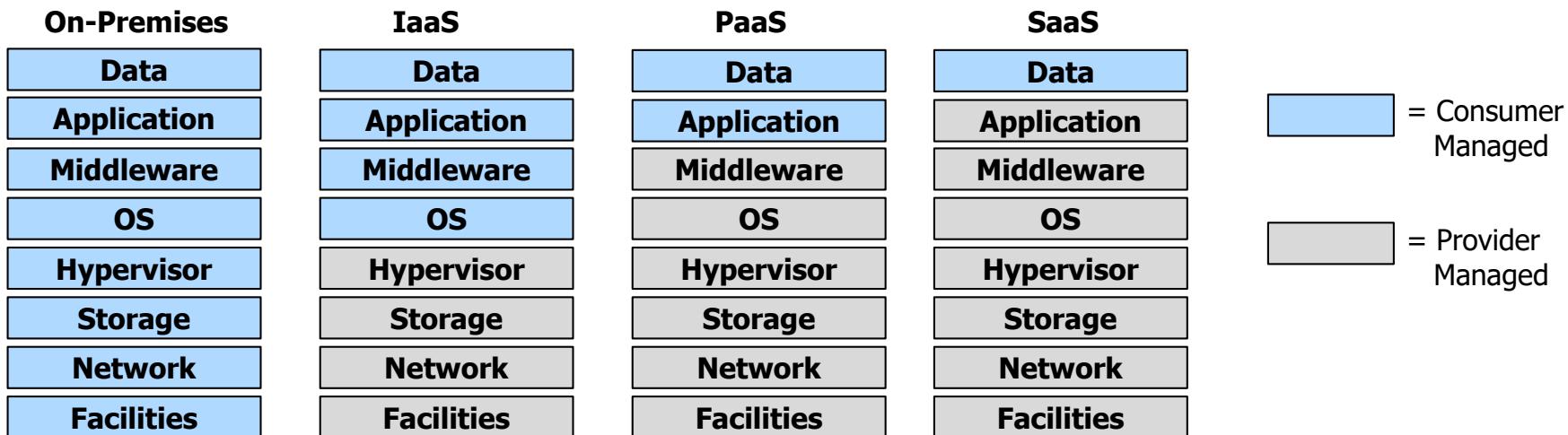
Cloud Deployment Models

Cluster Computing

Chapter Summary

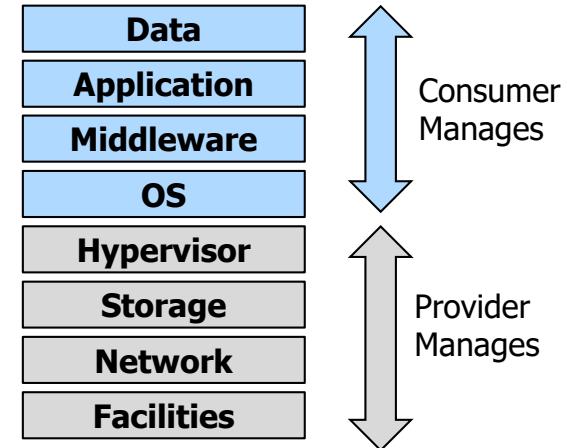
Cloud Service Models

- ◆ Cloud services are commonly divided into three categories
 - Infrastructure as a Service (IaaS)
 - Platform as a Service (PaaS)
 - Software as a Service (SaaS)



Infrastructure as a Service (IaaS)

- ◆ Delivers core infrastructure as a service
 - Networks, storage, servers, etc.
- ◆ IaaS provides the most flexible cloud solution
 - Build applications on top of this infrastructure
 - Can configure however you want
 - Can install any software you want
- ◆ But consumers are responsible for maintaining it
 - Cloud provider does not maintain, backup, patch, update the software
- ◆ Not much different from administering local servers

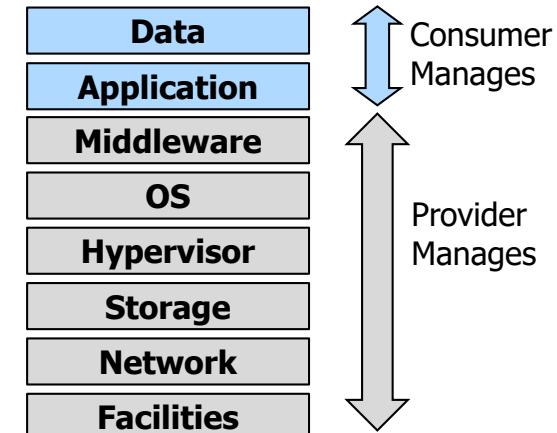


IaaS

- ◆ Often the easiest path onto the cloud
 - Migrate on-premises VMs to VMs hosted in the cloud
- ◆ Complete control over operating system, configuration, software
 - You are responsible for all maintenance other than hardware

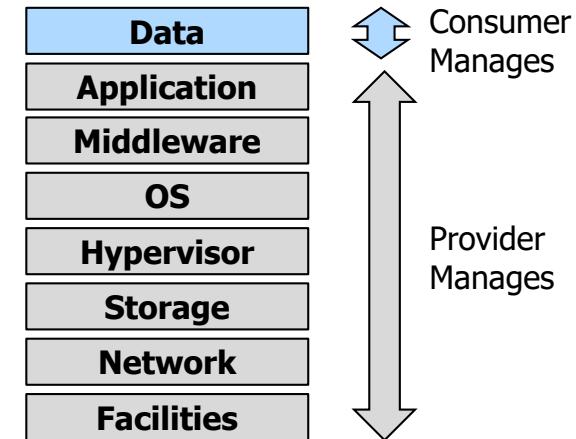
Platform as a Service (PaaS)

- ◆ Delivery of a hosted application environment as a service
 - Provides resources required to deploy and execute an application
 - Including auto-scaling, load balancing, health checks, etc.
- ◆ No need to buy or maintain resources required to execute the software
 - Simply deploy your application code into the environment
 - Enables organizations to concentrate on area of expertise
 - ↳ And not worry about managing IT infrastructure
 - Downside is code must conform to rules of the platform
 - ↳ Might require re-work for existing applications
- ◆ Many combinations of services by different PaaS providers



Software as a Service (SaaS)

- ◆ Model where provider licenses application use as a service on demand
 - No need to manually deploy or manage infrastructure in the cloud
 - No need to install software on the client computer
 - Provider supplies the entire infrastructure and software application
- ◆ Many traditional software applications have been rewritten as SaaS
 - Email, office suites, accounting software, productivity tools, CRM
 - Accessed using a web browser



SaaS – Serverless Computing

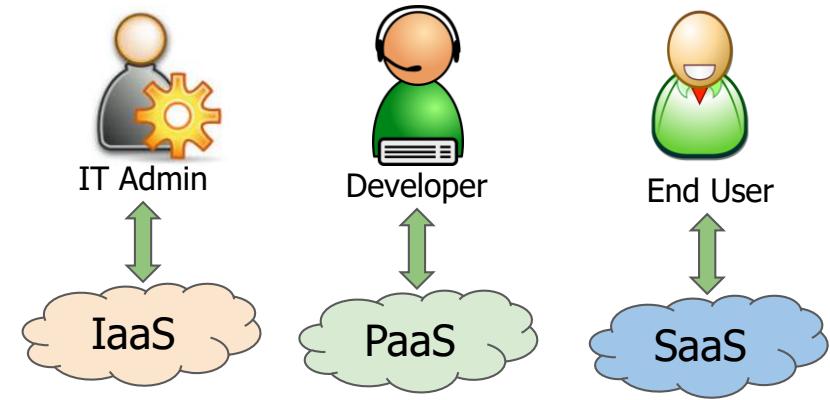
- ◆ SaaS can also be used to deliver cloud services
 - Some sophisticated cloud services are consumed as a SaaS no-ops solution
 - Sometimes called *serverless* computing
 - No need to manually provision or manage servers
 - There are still servers running, but the provider dynamically manages the allocation of machine resources
- ◆ Some examples of serverless cloud services:
 - Data warehouses
 - Data analytics
 - Machine Learning libraries

Cloud Service Models (continued)

IaaS		PaaS			SaaS	
IT Ops	Sys Ops	Dev Ops	Low Ops	Serverless/No Ops		
Managing resources: VMs, Storage, Software, Backups, Configuration, Maintenance		Managing applications: Application Development/ Deployment		Managing allocation: Capacity, Users and Identities, Subscriptions, Consumption		

Service Model Users

- ◆ Originally, the service models catered to specific roles
- ◆ The new “serverless” or no-ops services are changing that
 - Cloud providers are developing SaaS that target all job roles
 - AWS Lambda, Azure Functions, and GCP Cloud functions are SaaS for developers
 - BigQuery, Athena, and Data Lake Analytics are for data scientists/business analysts
 - These do not require an admin to provision or manage anything



Chapter Concepts

Defining Cloud Computing

Cloud Service Models

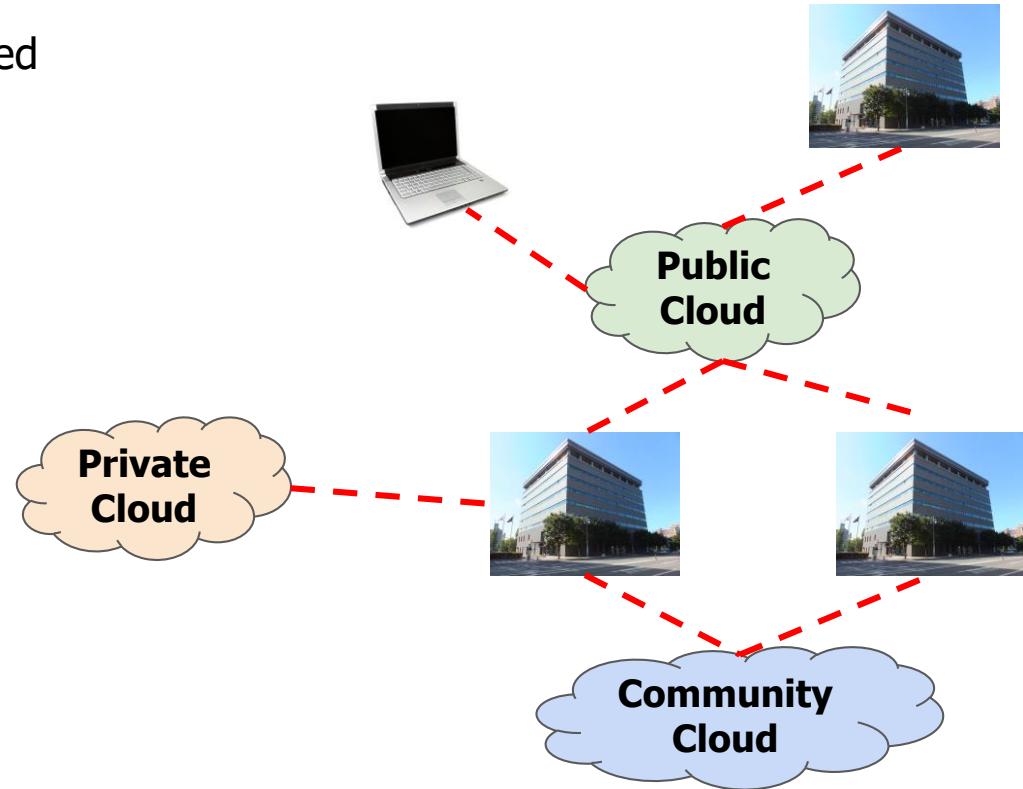
➤ **Cloud Deployment Models**

Cluster Computing

Chapter Summary

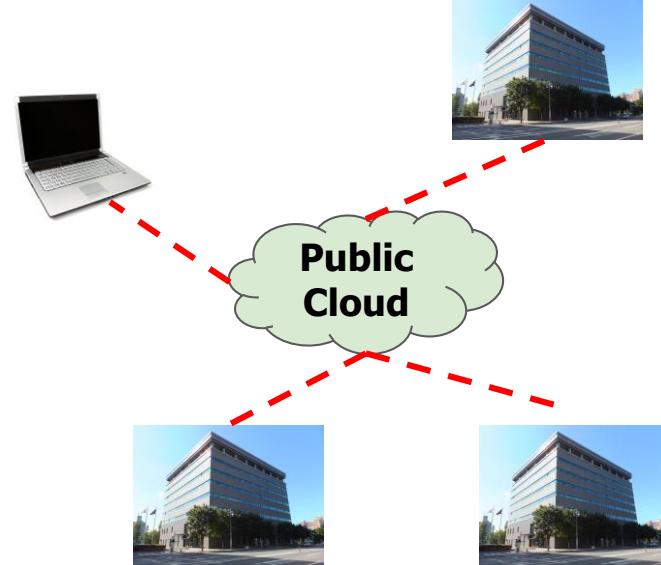
Deployment Models

- ◆ Cloud computing can be deployed in several different ways
 - Public
 - Private
 - ↳ On site
 - ↳ Hosted
 - Others
 - ↳ Community
 - ↳ Hybrid
 - ↳ Multi



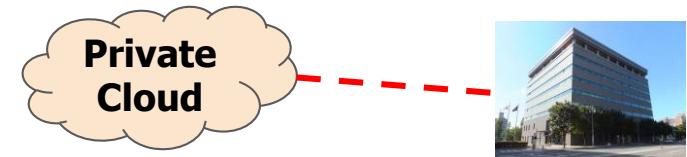
Public Cloud

- ◆ Cloud services hosted by a third-party organization
 - And made available to the general public
 - A multi-tenant environment
- ◆ Main public cloud providers:
 - Amazon Web Services (AWS)
 - Microsoft Azure
 - Google Cloud Platform (GCP)
 - Oracle
 - IBM
 - Alibaba
 - DigitalOcean



Private Cloud

- ◆ Cloud services dedicated to a single organization
 - Single-tenant
 - May be easier to comply with corporate security standards, policies, and regulatory requirements
- ◆ On-premises private cloud
 - Managed by organization or third party
- ◆ Hosted private cloud
 - Managed by third party
- ◆ Some cloud management software examples:
 - Openstack
 - VMWare VCloud
 - Microsoft System Center



Chapter Concepts

Defining Cloud Computing

Cloud Service Models

Cloud Deployment Models

► **Cluster Computing**

Chapter Summary

Big Data Analysis

- ◆ Amazon EMR
 - Managed service, supports many Big Data frameworks
 - Apache **Hadoop, Spark**, Presto, Hive, Pig, etc.
- ◆ Google DataProc
 - Managed service to run **Hadoop, Spark**, Hive, Pig, etc.
 - Can create clusters in 90 sec or less
- ◆ Azure HDInsight
 - Managed **Hadoop-based** service

Machine Learning (ML)

- ◆ Cloud providers all provide ML services to create intelligent models
 - Can create your own neural networks and train them in the cloud
 - Leverage the TensorFlow open-source machine learning framework
- ◆ Cloud providers also publish access to pre-trained ML models
 - GCP: Vision, Translation, Natural Language, Data Loss Prevention, etc.
 - AWS: Rekognition, Lex, Polly, etc.
 - Azure: Translator, Face, Anomaly Detection, etc.

Deployment Services

- ◆ Google Kubernetes Engine (GKE)
 - Deployments are managed using Kubernetes orchestration framework
 - ◆ Open source
 - ◆ Works on any platform
 - ◆ Supports **Docker**, as well as other container engines
- ◆ AWS
 - EC2 Container Service (ECS)
 - Amazon Elastic Container Service for Kubernetes (Amazon EKS)
- ◆ Azure Container Service
 - Supports using DC/OS, **Docker** Swarm, or Kubernetes



Demo (Optional)

- ◆ Your instructor will now demonstrate a cloud management console

Chapter Concepts

Defining Cloud Computing

Cloud Service Models

Cloud Deployment Models

Cluster Computing

Chapter Summary

Chapter Summary

In this chapter, we have:

- ◆ Defined cloud computing
- ◆ Explored characteristics of clouds
- ◆ Identified cloud service and deployment models
- ◆ Explored cloud based options for cluster computing



ROI TRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Module 2: Global Technical Analyst Program

COURSE SUMMARY

Course Summary

In this course, we have:

- ◆ Revised our knowledge of Java
- ◆ Highlighted successful software team practices
- ◆ Investigated the principles of blockchain algorithms
- ◆ Understood the concepts of MapReduce and Big Data
- ◆ Leveraged Hadoop as a reliable, scalable MapReduce framework
- ◆ Utilized the Hadoop distributed file system (HDFS) for storing big data files
- ◆ Developed MapReduce applications with Apache Spark
- ◆ Completed a substantial team project using the technologies discussed