DTS207TC Database Development and Design

Lecture 6
UML for CW1Q4

Chap 10 Big Data*

Di Zhang, Autumn 2025

*Page titles with * will not be assessed*

Outline

- UML
- Big Data
- Hadoop

CW1Q4

(a) Design UML schema for this application and draw UML class diagram for the schema. Specify key attributes of each entity type and structural constraints on each relationship type. Note any unspecified requirements, and make appropriate assumptions to make the specification complete. (10 marks)

(b) Translate the UML class diagram created in the above question **(a)** into a relational schema. In the relational schema, only use relation name and attribute, e.g., student (name, number, SSN,..etc.) and specify foreign key as fk. (10 marks)

This part belongs to object-oriented programming (C++, Python, etc.)

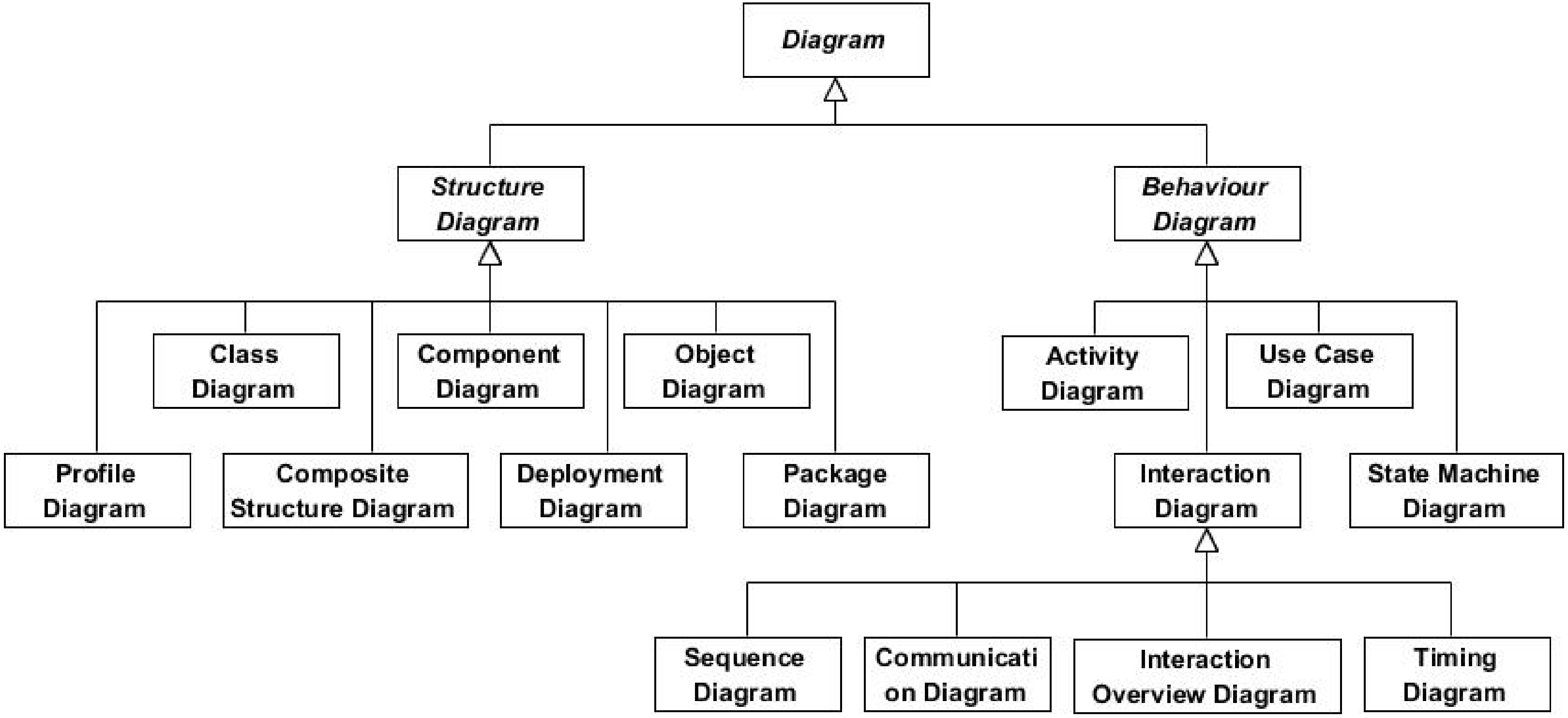
What is UML?

- Definition:
  - A standardized, powerful graphical modeling language.
  - Used to visualize, specify, construct, and document software-intensive systems.
- Core Purpose:
  - Communication: Provides a unified bridge for communication between development teams, clients, and architects.
  - Blueprint: Creates a blueprint for the system before coding, reducing complexity in understanding and design.
  - Documentation: Generates documentation crucial for understanding and maintaining the system.
- It is not a programming language, but a tool for designing software models.

Overview of UML Diagrams

- UML diagrams are primarily divided into Structure Diagrams and Behavior Diagrams.
- Structure Diagrams (Static Model) - Describe the static structure of a system.
  - **Class Diagram - Core diagram showing classes, interfaces, and their static relationships.**
  - Object Diagram - A snapshot of object instances and their relationships at a point in time.
  - Component Diagram - Describes the physical components and their dependencies.
  - Deployment Diagram - Shows how software is deployed on hardware.
  - Composite Structure Diagram - Describes the internal structure of a class.
  - Package Diagram - Used to group elements.
- Behavior Diagrams (Dynamic Model) - Describe the dynamic behavior of a system.
  - Use Case Diagram - Describes system functionality from a user's perspective.
  - Activity Diagram - Models workflows or business processes.
  - State Machine Diagram - Describes the state changes of an object during its life cycle.
  - Sequence Diagram - Emphasizes the time order of messages in an interaction.
  - Communication Diagram - Emphasizes the structural organization of objects in an interaction.

Overview of UML Diagrams

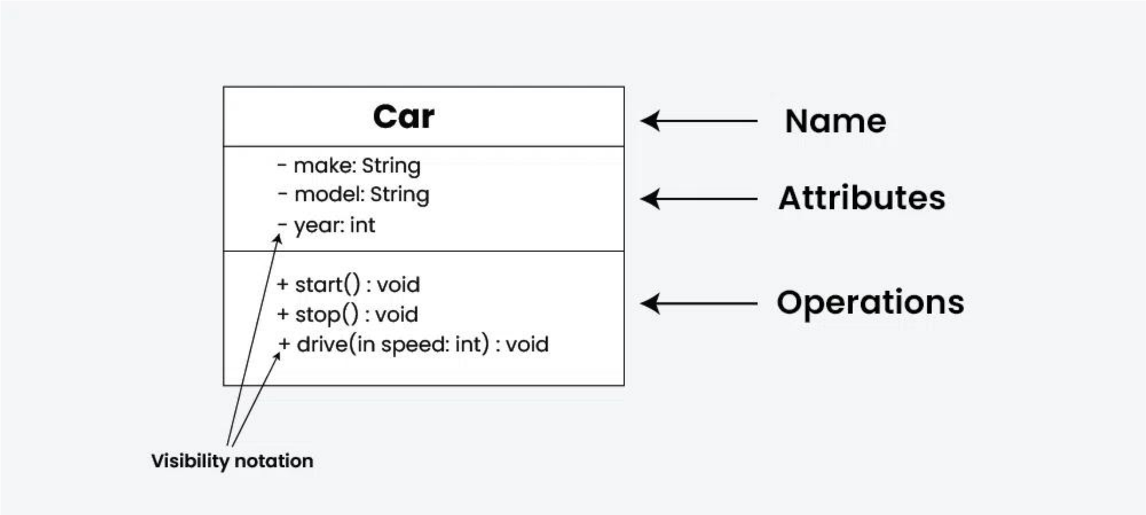


Class Diagrams - The Heart of UML

- What is a Class Diagram?
  - A diagram that describes the classes, interfaces, and their static relationships within a system.
  - The most commonly used and fundamental diagram for object-oriented system modeling.
- Main Purpose of Class Diagrams:
  - Provides a unified view of the system design for the development team.
  - Details the logical structure of the system.
  - Serves as the basis for Forward Engineering (generating code skeletons from class diagrams) and Reverse Engineering (generating class diagrams from code).
- Core Components: Classes, Attributes, Methods, Relationships.

The Basic Structure of a Class

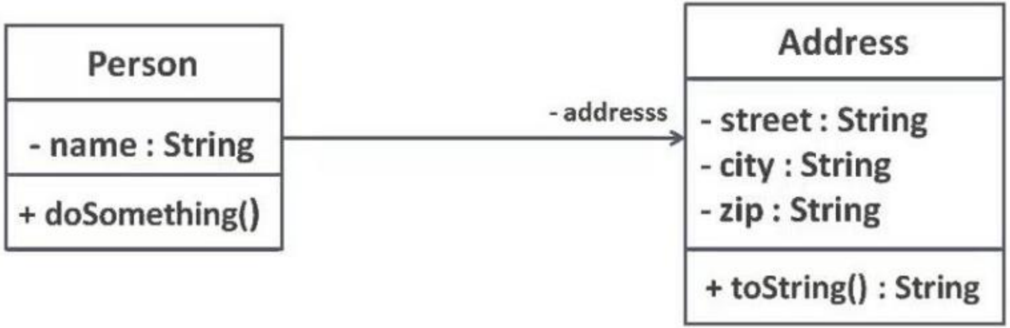
- A class is represented in UML as a rectangle divided into three compartments:
  - ClassName
  - attributes
  - operations



Class Relationships

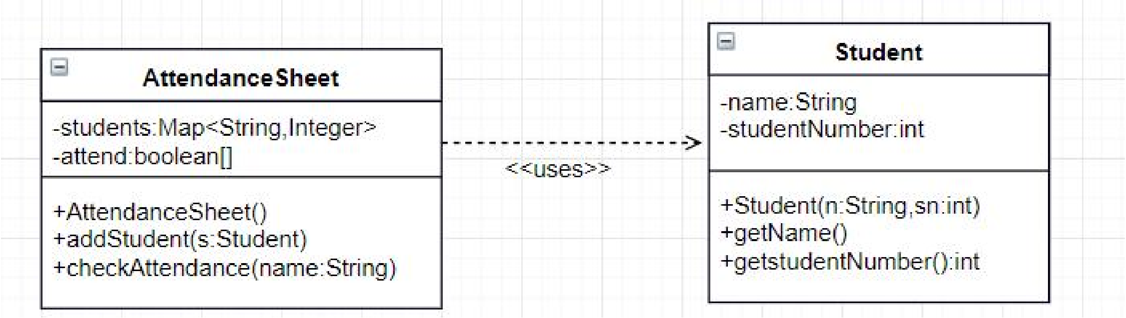
Xi'an Jiaotong-Liverpool University 西交利物浦大学

- 1. Association
- Description: A structural relationship between classes, indicating a long-lasting, "knows-a" connection between objects.
- Notation: A solid line.
- Navigability: Arrows indicate the direction that can be navigated. A bidirectional association may have no arrows or two arrows.
- Role Names: Can be added at each end of the association to describe the role the other class plays.
- Multiplicity: Indicates how many objects of one class relate to another class.
- 1 (exactly one), 0..1 (zero or one), * or 0..* (many), 1..* (one or more), n (a specific number)



Class Relationships

- 2. Dependency
- Description: A "using" relationship, temporary and transient. A change in one class may affect the dependent class.
- Notation: A dashed line with an open arrow ------>
- Common Scenarios: Method parameters, local variables, static method calls.



Example: Association vs. Dependency

Xi'an Jiaotong-Liverpool University 西交利物浦大学

```
class Person:
    """
    Association Example: Person HAS-A Car.
    This is a structural, long-term relationship.
    """
    def __init__(self, car: Car):
        # Association: Person "has" a Car.
        # 'my_car' is an instance variable (field).
        # The relationship persists as long as the Person object exists.
        self.my_car = car

    def drive_my_car(self):
        """Uses the car that the person owns."""
        self.my_car.start()
```

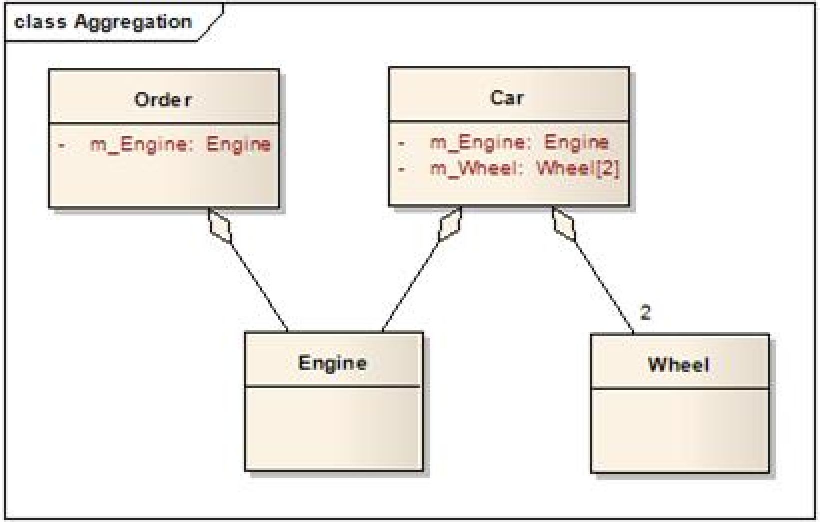
Example: Association vs. Dependency

```python
def drive_rental_car(self, rental_car: Car):
    """
    Dependency: Person "uses" a rental car temporarily.
    'rental_car' is a method parameter, a temporary input.
    """
    rental_car.start()
    # The relationship with this specific 'rental_car' object ends when the method finishes.

def go_to_test_drive(self):
    """
    Another Dependency example: creating a local variable inside a method.
    """
    test_car = Car()  # 'test_car' is a local variable
    test_car.start()
    # When the method ends, the 'test_car' reference goes out of scope.
```

Class Relationships

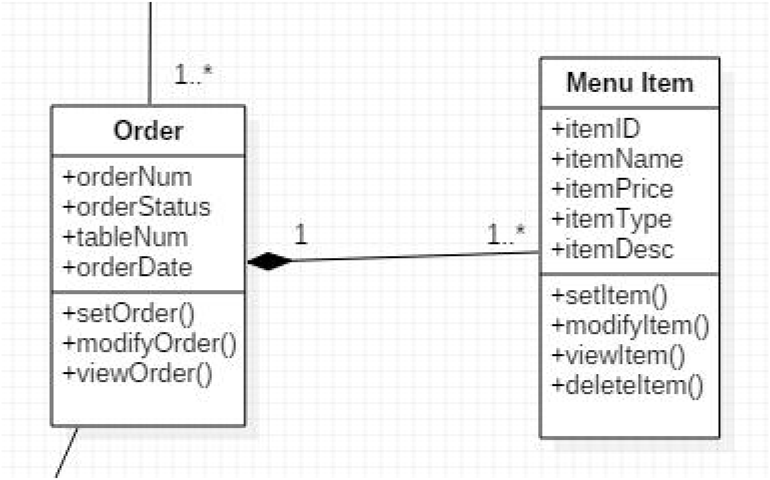
- 3. Aggregation
- Description: A special type of Association representing a "whole-part" relationship where the part can exist independently of the whole. It's a weak "has-a" relationship.
- Notation: A solid line with a hollow diamond, diamond connected to the whole.
- Whole ◊ —— Part



Class Relationships

Xi'an Jiaotong-Liverpool University 西交利物浦大学

- 4. Composition
- Description: A stronger form of Aggregation, representing a strict "whole-part" relationship where the part's lifecycle is dependent on the whole. It's a strong "has-a" relationship.
- Notation: A solid line with a filled diamond, diamond connected to the whole.
- Whole ◆ —— Part



Example: Aggregation

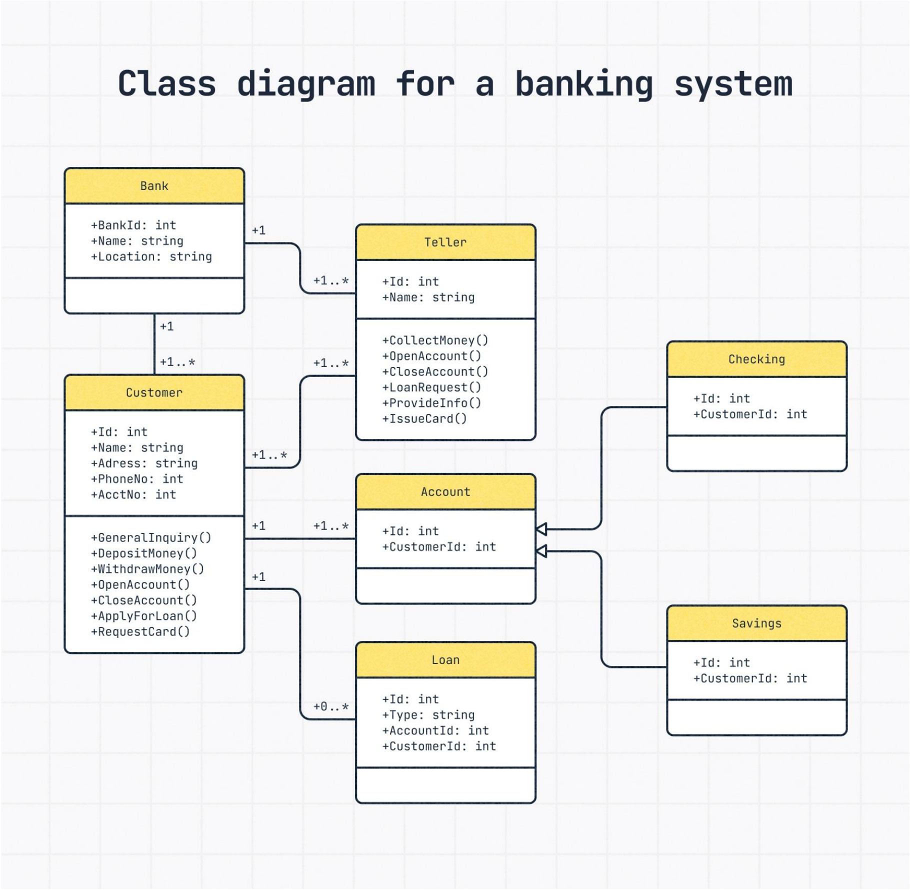
Xi'an Jiaotong-Liverpool University
西交利物浦大学

```python
class Department:
    """
    Aggregation Example: Department HAS-A Teacher, but weakly.
    The Teacher can exist independently of the Department.
    """
    def __init__(self, name: str):
        self.name = name
        # Aggregation: Department has a list of Teachers
        # The teachers are passed from outside and can exist without the department
        self.teachers = []  # This is a weak "has-a" relationship
```

Example: Composition

```python
class Car:
    """
    Composition Example: Car HAS-A Engine and Wheels, STRONGLY.
    The Engine and Wheels are created when the Car is created and cannot exist without it.
    """
    def __init__(self, model: str):
        self.model = model
        # Composition: Car creates its own Engine and Wheels
        # These parts are created WHEN the car is created and have no independent existence
        self.engine = Engine("V6")  # Strong "has-a" relationship
        self.wheels = [
            Wheel("Front Left"),    # Strong "has-a" relationship
            Wheel("Front Right"),   # Strong "has-a" relationship
            Wheel("Rear Left"),     # Strong "has-a" relationship
            Wheel("Rear Right")     # Strong "has-a" relationship
        ]
        print(f"Car {self.model} created with engine and 4 wheels")
```

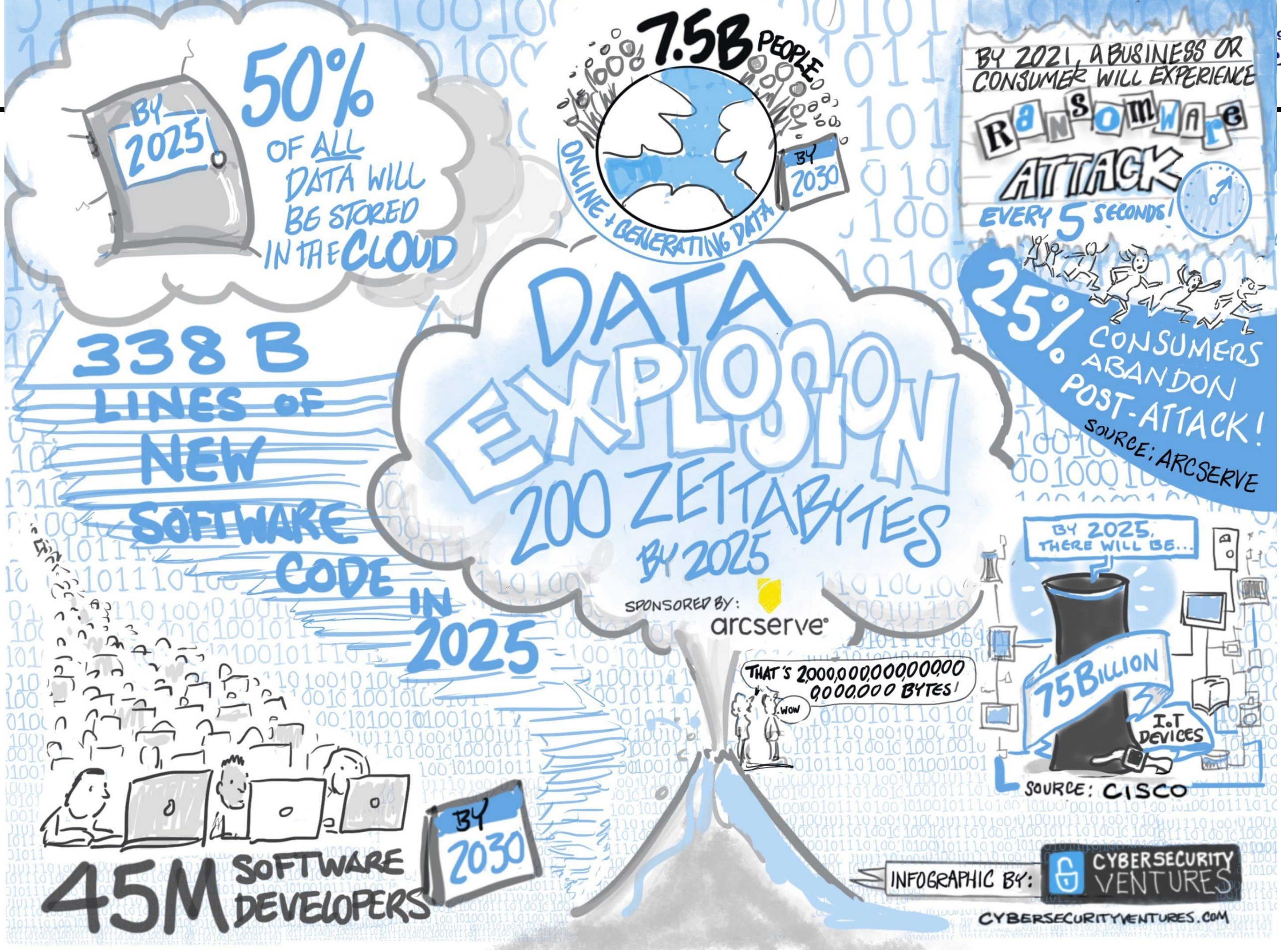
Example

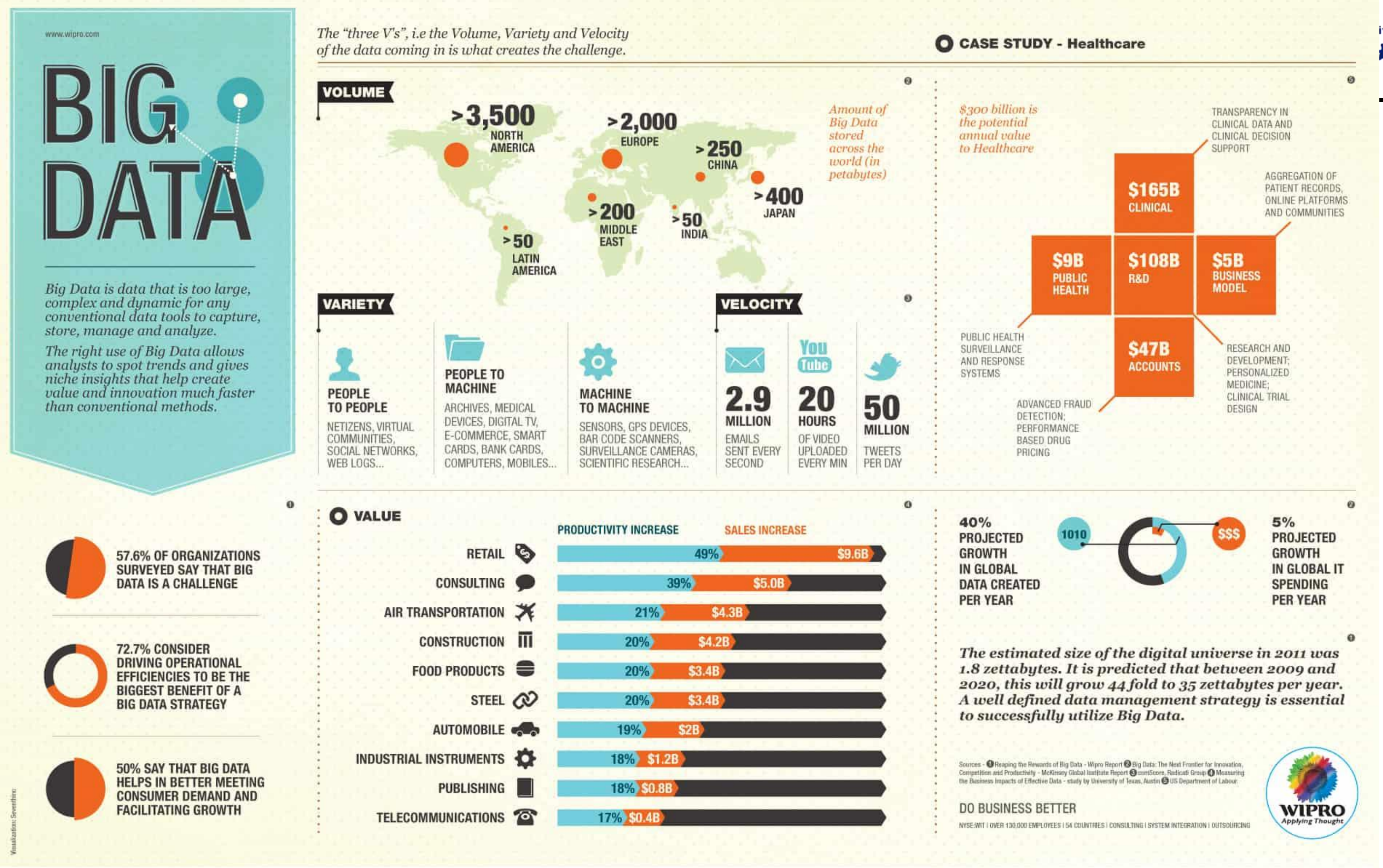


Class diagram for a banking system
Bank
+BankId: int
+Name: string
+Location: string
Teller
+Id: int
+Name: string
+CollectMoney()
+OpenAccount()
+CloseAccount()
+LoanRequest()
+ProvideInfo()
+IssueCard()
Customer
+Id: int
+Name: string
+Adress: string
+PhoneNo: int
+AcctNo: int
+GeneralInquiry()
+DepositMoney()
+WithdrawMoney()
+OpenAccount()
+CloseAccount()
+ApplyForLoan()
+RequestCard()
Account
+Id: int
+CustomerId: int
Loan
+Id: int
+Type: string
+AccountId: int
+CustomerId: int
Checking
+Id: int
+CustomerId: int
Savings
+Id: int
+CustomerId: int
+1
+1..*
+1
+1..*
+1..*
+1..*
+1
+1..*
+1
+0..*

Tool

- Any drawing tool is acceptable. Such as:
  - https://plantuml.com/
  - https://www.lucidchart.com/pages/examples/uml_diagram_tool
  - https://www.visual-paradigm.com/solution/freeumltool/
  - or whatever tool you choose


BY 2025
50% OF ALL DATA WILL BE STORED IN THE CLOUD
7.5B PEOPLE
ONLINE + GENERATING DATA
BY 2030
BY 2021, A BUSINESS OR CONSUMER WILL EXPERIENCE
RANSOMWARE ATTACK
EVERY 5 SECONDS!
25% CONSUMERS ABANDON POST-ATTACK!
SOURCE: ARCSERVE
338 B LINES OF NEW SOFTWARE CODE IN 2025
DATA EXPLOSION
200 ZETTABYTES BY 2025
SPONSORED BY:
arcserve
BY 2025, THERE WILL BE...
75 BILLION
I.T DEVICES
SOURCE: CISCO
THAT'S 2000,000,000,000,000,000,000 BYTES!
WOW
45M SOFTWARE DEVELOPERS
BY 2030
INFOGRAPHIC BY:
CYBERSECURITY VENTURES
CYBERSECURITYVENTURES.COM



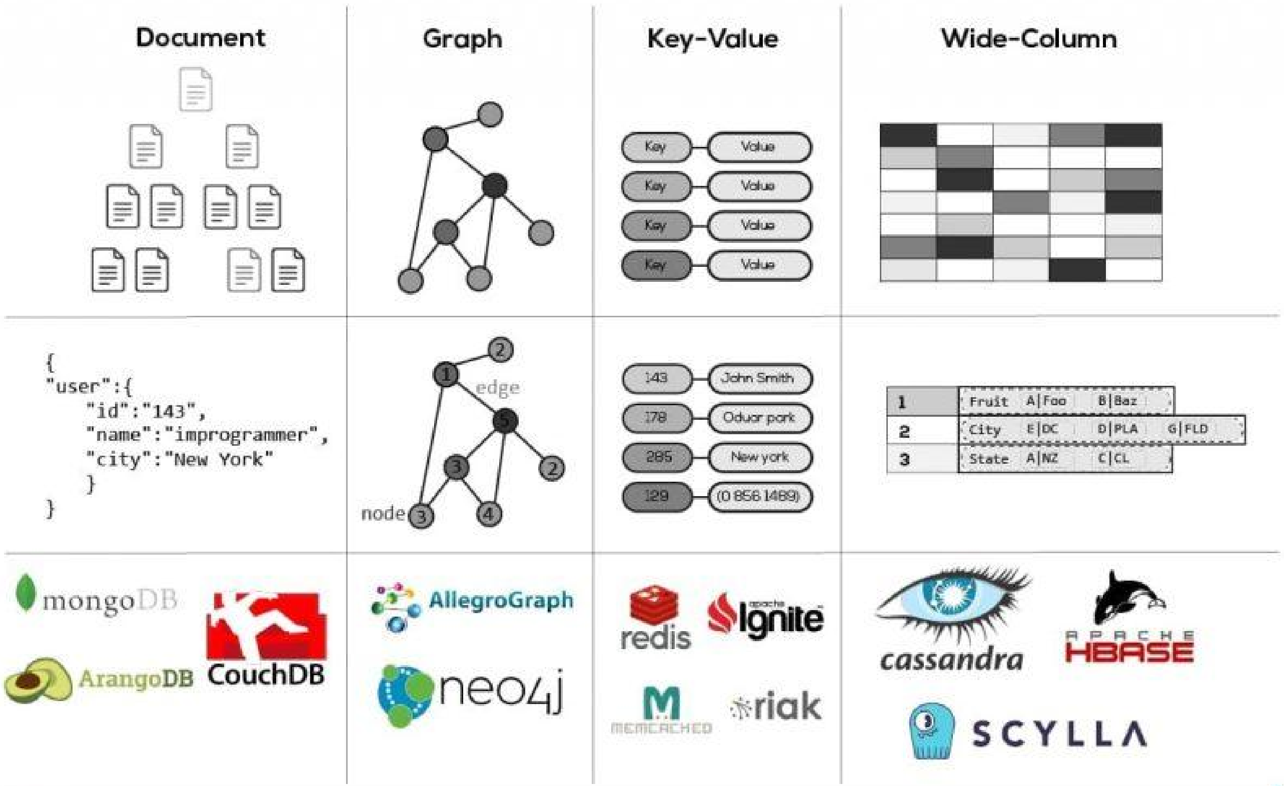
www.wipro.com
BIG DATA
Big Data is data that is too large, complex and dynamic for any conventional data tools to capture, store, manage and analyze.
The right use of Big Data allows analysts to spot trends and gives niche insights that help create value and innovation much faster than conventional methods.
The "three V's", i.e the Volume, Variety and Velocity of the data coming in is what creates the challenge.
VOLUME
>3,500 NORTH AMERICA
>2,000 EUROPE
>250 CHINA
>400 JAPAN
>200 MIDDLE EAST
>50 INDIA
>50 LATIN AMERICA
Amount of Big Data stored across the world (in petabytes)
VARIETY
PEOPLE TO PEOPLE
NETIZENS, VIRTUAL COMMUNITIES, SOCIAL NETWORKS, WEB LOGS...
PEOPLE TO MACHINE
ARCHIVES, MEDICAL DEVICES, DIGITAL TV, E-COMMERCE, SMART CARDS, BANK CARDS, COMPUTERS, MOBILES...
MACHINE TO MACHINE
SENSORS, GPS DEVICES, BAR CODE SCANNERS, SURVEILLANCE CAMERAS, SCIENTIFIC RESEARCH...
VELOCITY
2.9 MILLION EMAILS SENT EVERY SECOND
You Tube
20 HOURS OF VIDEO UPLOADED EVERY MIN
50 MILLION TWEETS PER DAY
CASE STUDY - Healthcare
$300 billion is the potential annual value to Healthcare
$165B CLINICAL
TRANSPARENCY IN CLINICAL DATA AND CLINICAL DECISION SUPPORT
$9B PUBLIC HEALTH
PUBLIC HEALTH SURVEILLANCE AND RESPONSE SYSTEMS
$108B R&D
RESEARCH AND DEVELOPMENT; PERSONALIZED MEDICINE; CLINICAL TRIAL DESIGN
$5B BUSINESS MODEL
AGGREGATION OF PATIENT RECORDS, ONLINE PLATFORMS AND COMMUNITIES
$47B ACCOUNTS
ADVANCED FRAUD DETECTION; PERFORMANCE BASED DRUG PRICING
57.6% OF ORGANIZATIONS SURVEYED SAY THAT BIG DATA IS A CHALLENGE
72.7% CONSIDER DRIVING OPERATIONAL EFFICIENCIES TO BE THE BIGGEST BENEFIT OF A BIG DATA STRATEGY
50% SAY THAT BIG DATA HELPS IN BETTER MEETING CONSUMER DEMAND AND FACILITATING GROWTH
VALUE
PRODUCTIVITY INCREASE
SALES INCREASE
RETAIL 49% $9.6B
CONSULTING 39% $5.0B
AIR TRANSPORTATION 21% $4.3B
CONSTRUCTION 20% $4.2B
FOOD PRODUCTS 20% $3.4B
STEEL 20% $3.4B
AUTOMOBILE 19% $2B
INDUSTRIAL INSTRUMENTS 18% $1.2B
PUBLISHING 18% $0.8B
TELECOMMUNICATIONS 17% $0.4B
40% PROJECTED GROWTH IN GLOBAL DATA CREATED PER YEAR
1010
$$$
5% PROJECTED GROWTH IN GLOBAL IT SPENDING PER YEAR
The estimated size of the digital universe in 2011 was 1.8 zettabytes. It is predicted that between 2009 and 2020, this will grow 44 fold to 35 zettabytes per year. A well defined data management strategy is essential to successfully utilize Big Data.
Sources - 1 Reaping the Rewards of Big Data - Wipro Report 2 Big Data: The Next Frontier for Innovation, Competition and Productivity - McKinsey Global Institute Report 3 comScore, Radicati Group 4 Measuring the Business Impacts of Effective Data - study by University of Texas, Austin 5 US Department of Labour
DO BUSINESS BETTER
NYSE:WIT | OVER 130,000 EMPLOYEES | 54 COUNTRIES | CONSULTING | SYSTEM INTEGRATION | OUTSOURCING
WIPRO Applying Thought


6
Steps
in
CRISP-DM
The Standard
Data Mining
Process
Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Data Presentation
Data

Challenge and Response

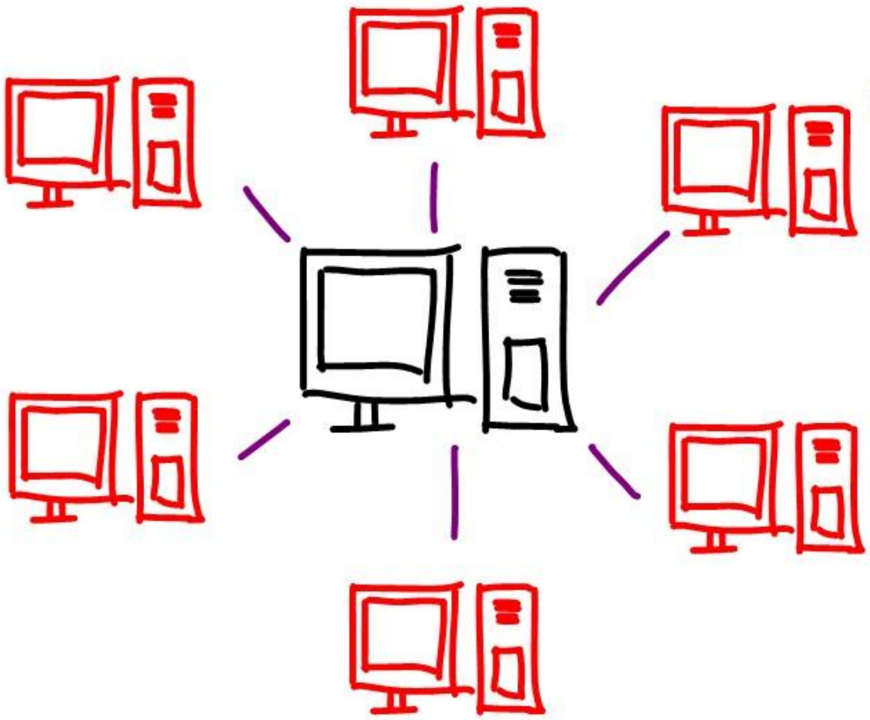
- Volume
  - Distributed Computing
- Velocity
  - ACID->BASE: NoSQL
- Variety
  - Unstructed Text

NoSQL DATABASE TYPES

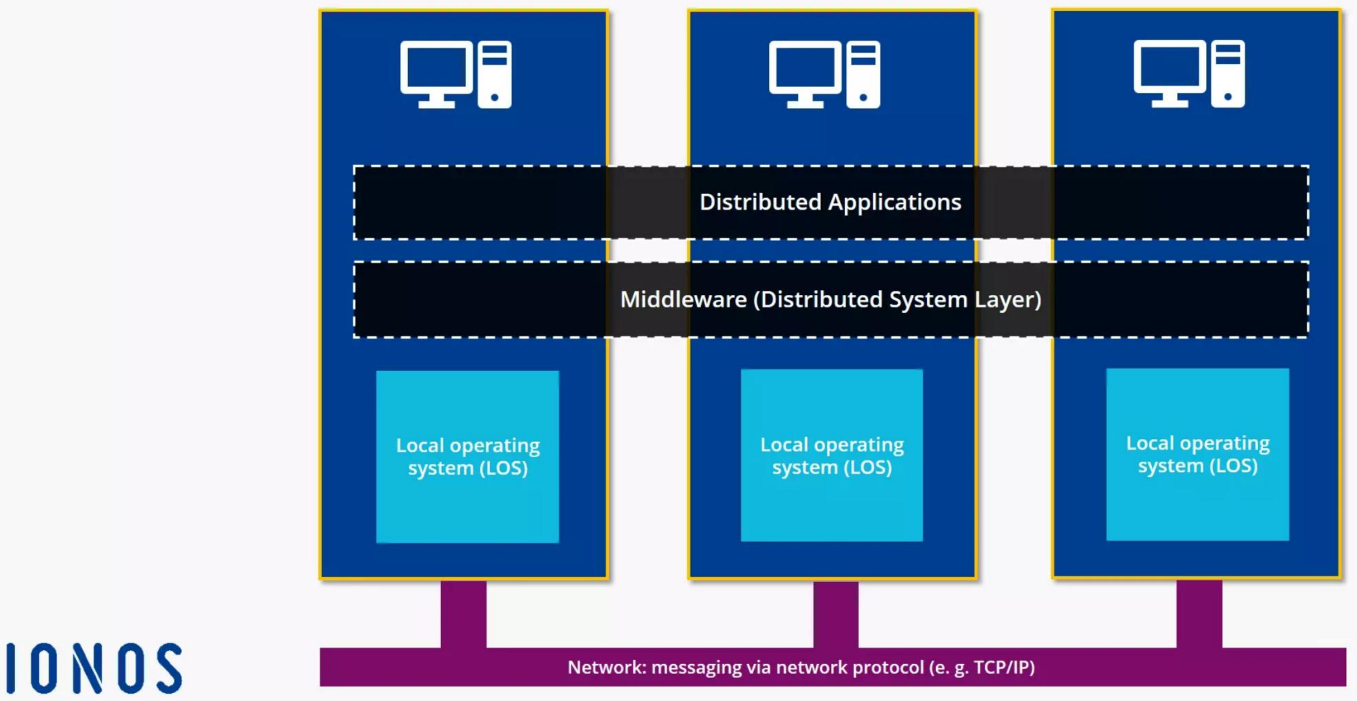


-

Distributed Computing



Distributed Computing



Distributed applications can solve problems across devices in a computer network. When used in conjunction with middleware, they can optimize operational interactions with locally accessible hardware and software.

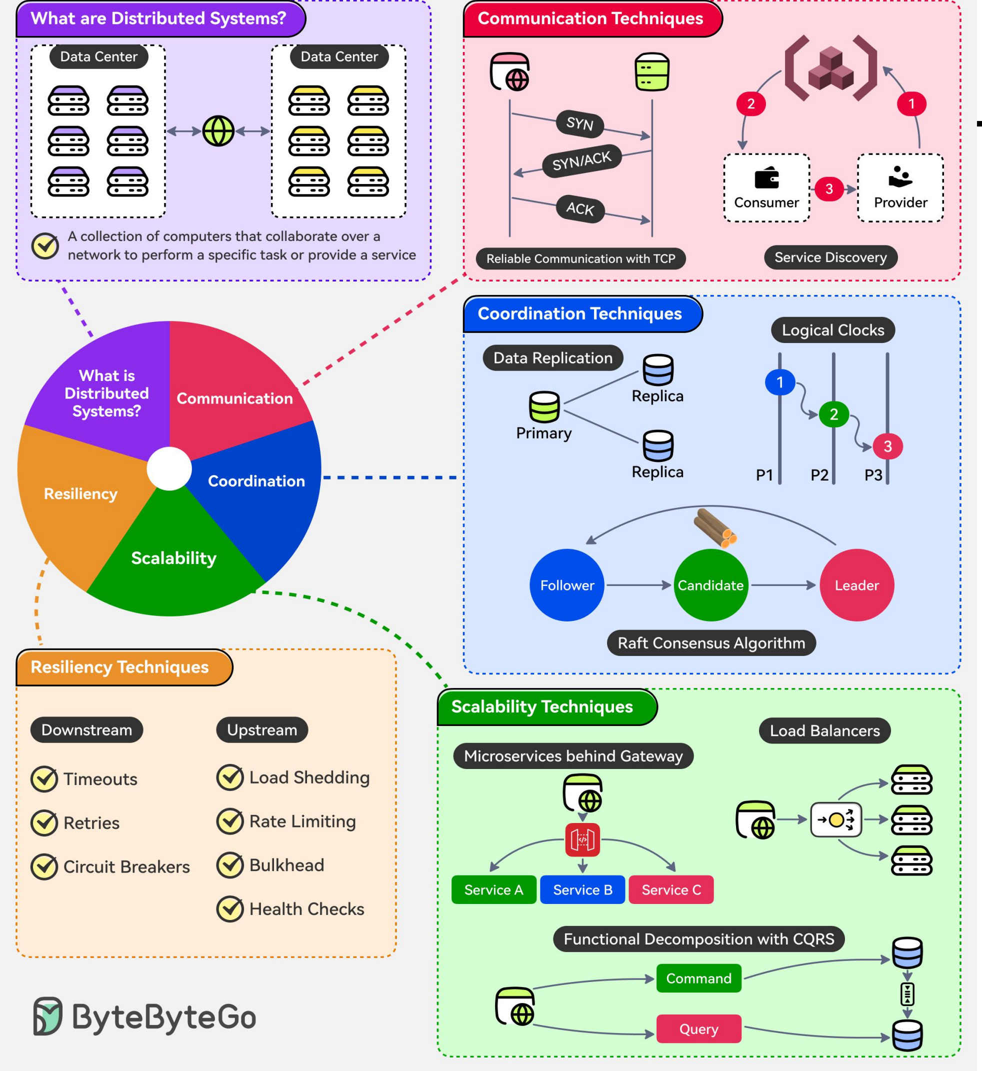
Distributed Computing

Advantages

- 🔒 Improved Security
- 📏 Flexibility and Scalability
- 🚀 Enhanced Performance
- ⚡ Low Latency
- 🔗 Increased Reliability
- 💾 Expanded Storage
- $ Cost effectiveness

Disdvantages

- 💰 Maintenance Fees
- 💔 Component Failure
- 🌐 Bandwidth Restrictions
- 🧩 Systems Complexity
- 🚧 Development Obstacles
- 🐢 Slow Network Transfers
- ❓ Issues with Standardization



What are Distributed Systems?
Data Center
Data Center
A collection of computers that collaborate over a network to perform a specific task or provide a service
Communication Techniques
SYN
SYN/ACK
ACK
Reliable Communication with TCP
1
2
3
Consumer
Provider
Service Discovery
What is Distributed Systems?
Communication
Coordination
Scalability
Resiliency
Coordination Techniques
Data Replication
Primary
Replica
Replica
Logical Clocks
1
2
3
P1
P2
P3
Follower
Candidate
Leader
Raft Consensus Algorithm
Resiliency Techniques
Downstream
Timeouts
Retries
Circuit Breakers
Upstream
Load Shedding
Rate Limiting
Bulkhead
Health Checks
Scalability Techniques
Microservices behind Gateway
Service A
Service B
Service C
Load Balancers
Functional Decomposition with CQRS
Command
Query
ByteByteGo

Distributed System

- A distributed system is a collection of independent computers that appear to users as a single coherent system
- Focuses on providing reliability, availability, & fault tolerance by distributing data & processes across multiple nodes
- To ensure system resilience, fault tolerance, & scalability by distributing services or data
- Involves systems such as distributed databases, message queues, or microservices

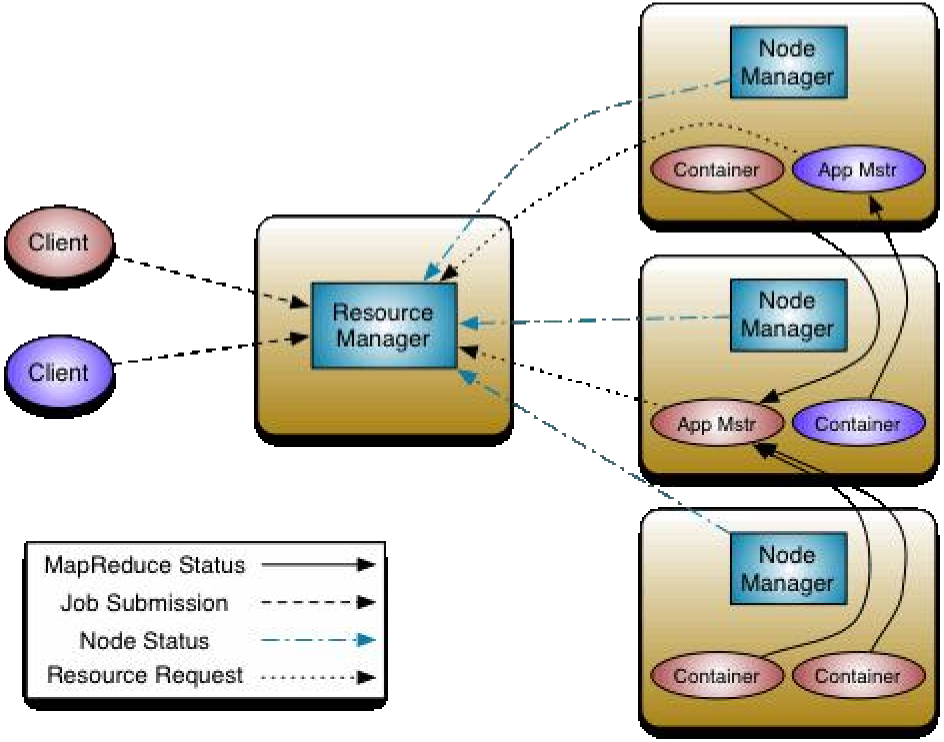
VS

Distributed Computing

- Distributed computing refers to using multiple computers to perform computational tasks collaboratively.
- Focuses on splitting large computational tasks into smaller chunks that can be executed concurrently.
- To solve complex computational problems by breaking them down and distributing tasks across multiple nodes.
- Involves dividing a large computational workload into smaller tasks

Aspect	Parallel Computing	Distributed Computing
Primary Goal	Increase speed, reduce execution time for a single, complex task.	Resource sharing, scalability, fault tolerance, solving large-scale problems.
System Architecture	Tightly-coupled. Processors connected via high-speed buses, shared memory, or specialized networks (InfiniBand).	Loosely-coupled. Computers (nodes) connected via standard networks (Ethernet, Internet).
Memory Model	Often Shared Memory. All processors access a common memory address space.	Often Distributed Memory. Each node has its own private memory; data is exchanged via message passing (e.g., MPI).
Communication & Coordination	Very low latency, very high bandwidth. Processes/threads require tight synchronization.	Higher latency, lower bandwidth. Nodes collaborate through asynchronous message passing.
User Perspective	Feels like using a single, powerful supercomputer.	Feels like using a single, coherent system (even though it's composed of many machines).
Scaling Method	Vertical Scaling (Scale-up). Adding more CPUs/cores within a single node. Limited by physical constraints.	Horizontal Scaling (Scale-out). Adding more commodity computer nodes. Theoretically unlimited.
Fault Tolerance	Typically weak. The failure of a single processor can cause the entire computation to fail.	Inherently strong. Designed with node failure in mind. Tasks can be restarted on other nodes if one fails.
Typical Applications	Scientific computing, 3D rendering, weather simulation, genomic sequence analysis.	Large web services (Google, Amazon), Big Data processing (Hadoop/Spark), blockchain, cloud platforms.
Programming Models/Tools	OpenMP (shared memory), MPI (distributed memory, for tight clusters), CUDA (GPU).	Hadoop, Spark, Akka, gRPC, and various cloud APIs (AWS, Azure).

Example: Hadoop



Why Hadoop belongs to distributed Computing, but MPI not?

- 1. Primary Goal: Scale vs. Speed
  - Extreme Speed vs. the Ability to Process Massive Data

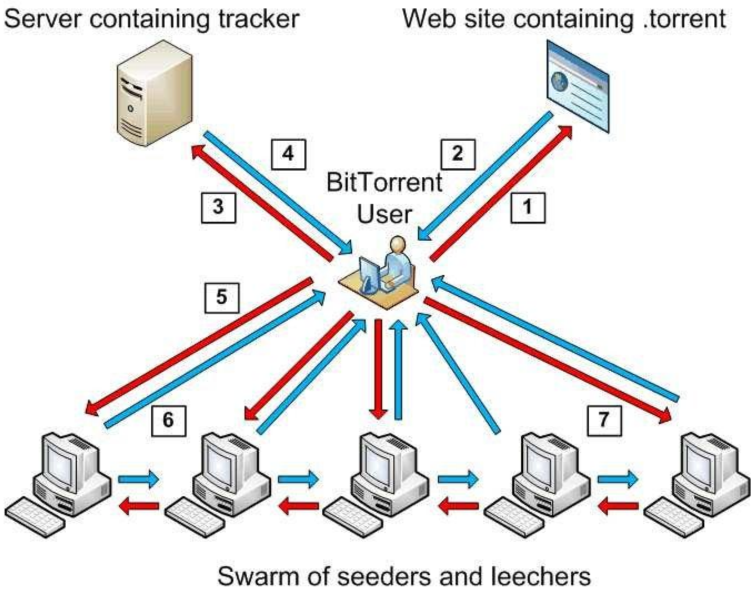
- 2. System Architecture: Fault Tolerance vs. Performance (the most critical difference)
  - Parallel Computing (MPI): Tightly coupled. After tasks are distributed to multiple nodes, frequent communication and synchronization between nodes are required.
  - Hadoop: Loosely coupled. Its core design philosophies are "mobile computing is cheaper than moving data" and "hardware failures are the norm, not the exception."

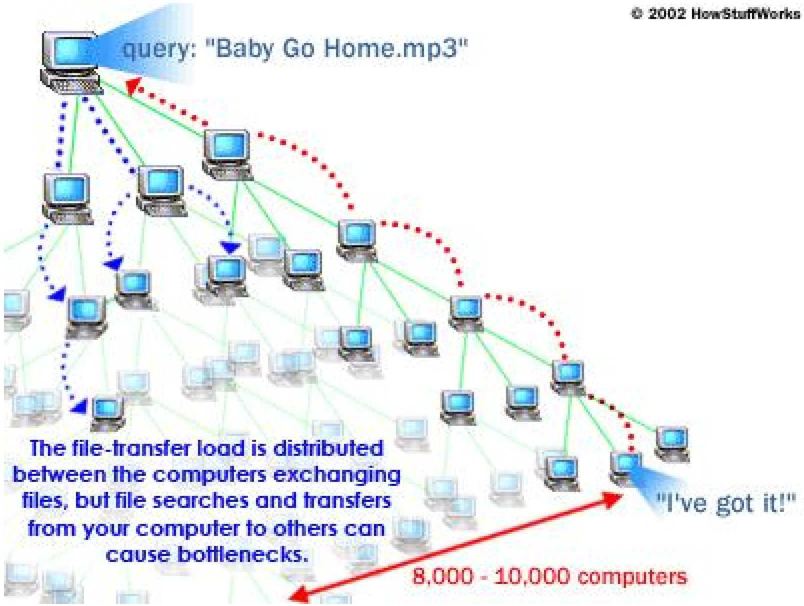
- 3. Communication Model: Message Passing vs. Data Sharing
  - Parallel Computing (MPI): Frequent, low-latency, bidirectional communication between processes is achieved through a message passing interface.
  - Hadoop MapReduce: There is almost no communication between processes. This "share nothing" architecture is a typical distributed system design.

- 4. Problem Domain: Single Complex Task vs. Decomposable Batch Tasks
  - Parallel Computing: Excels at solving a single, complex problem that requires a high degree of coordination (such as simulating an entire weather system). Hadoop MapReduce excels at batch processing problems that can be clearly decomposed into a large number of independent subtasks (such as counting the number of visits to each URL in a massive log). Each subtask processes a small portion of the data and does not depend on the intermediate results of other subtasks.

- An analogy: MPI is like a symphony orchestra. Hadoop is like a courier company's sorting center.

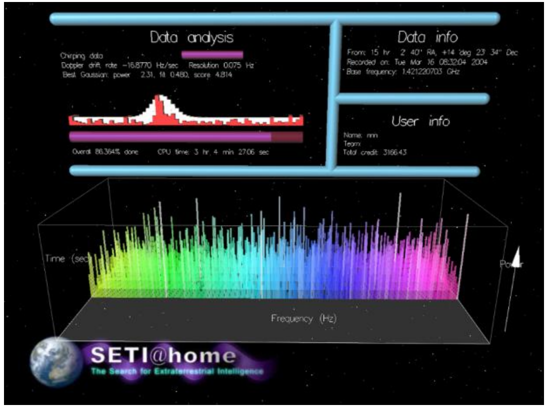
Example: P2P Download





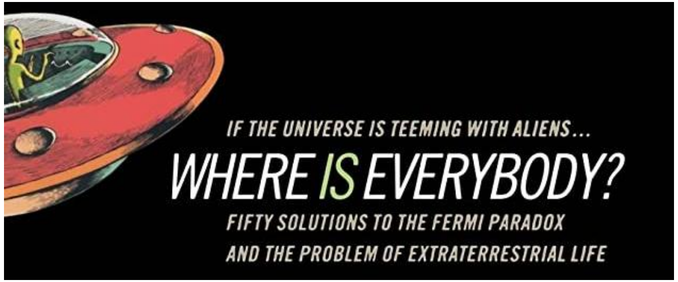
Example: the ‘Slowest’ Distributed System

Xi’an Jiaotong-Liverpool University 西交利物浦大学



Search Alien Civilization, using volunteer PCs on the internet

- Found nothing…
  - Fermi Paradox

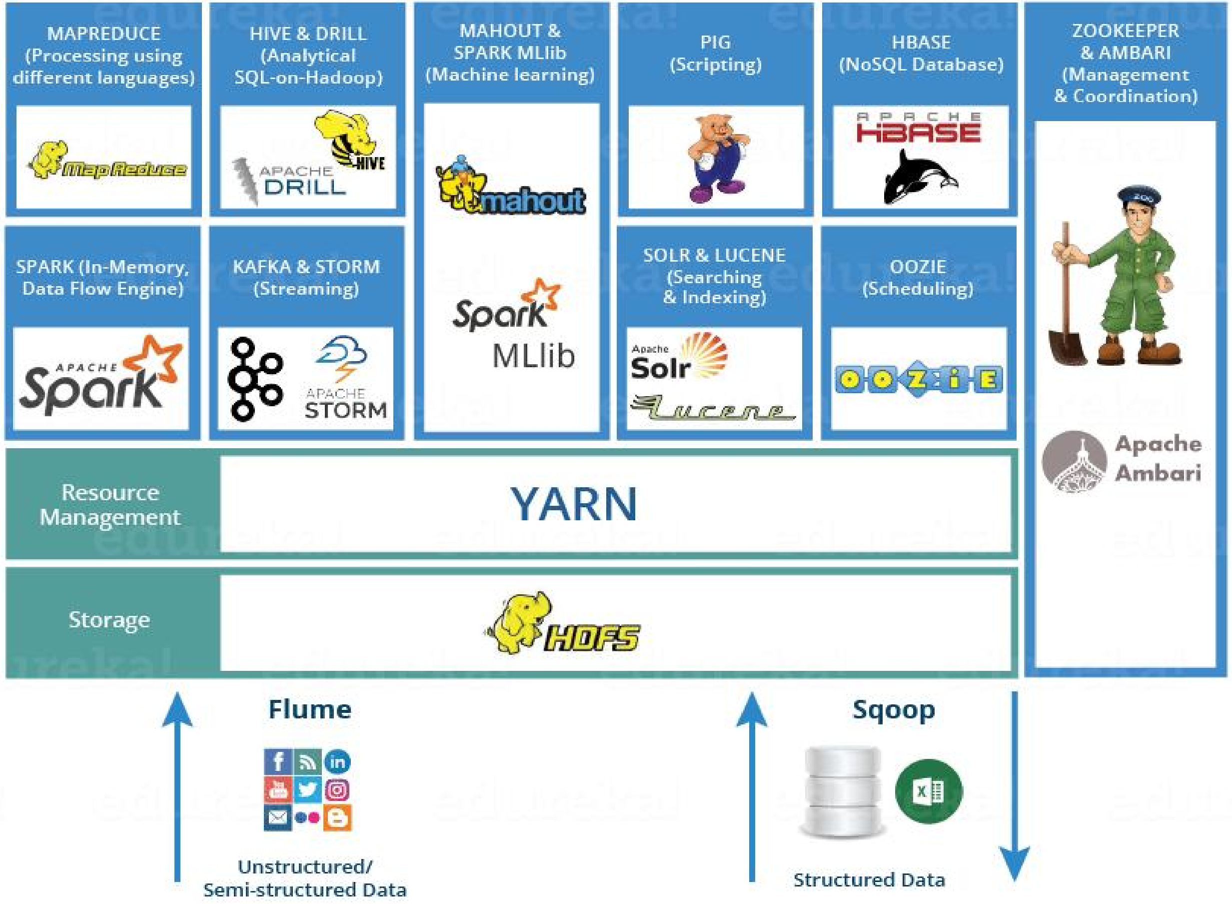


What is Hadoop?

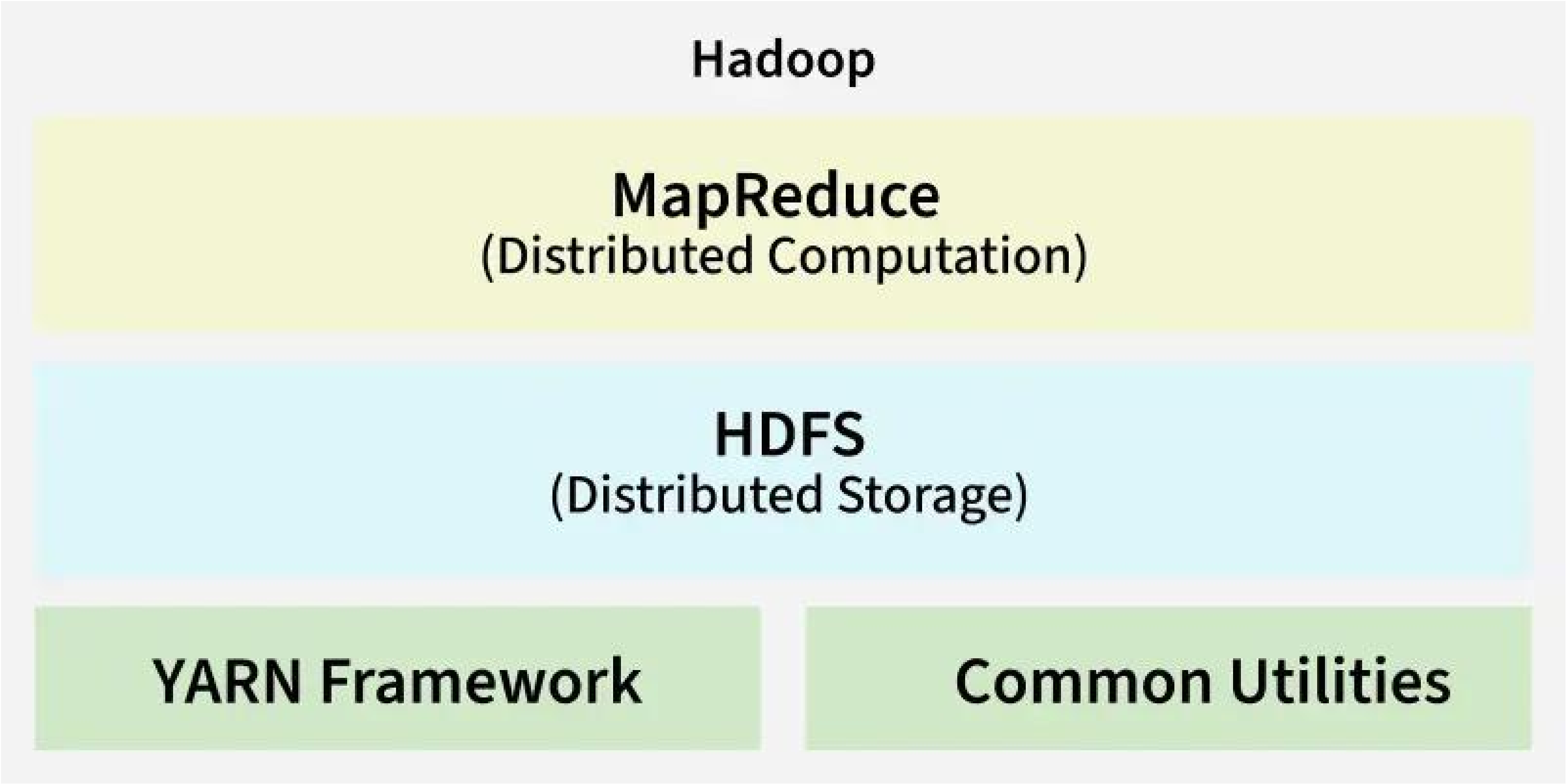
Xi'an Jiaotong-Liverpool University 西交利物浦大学

- An open-source, reliable, and scalable distributed computing framework developed by the Apache Foundation.
- Core ideas: Derived from Google's GFS and MapReduce papers.
- Key advantages: High fault tolerance, high throughput, and low cost (using commodity hardware).

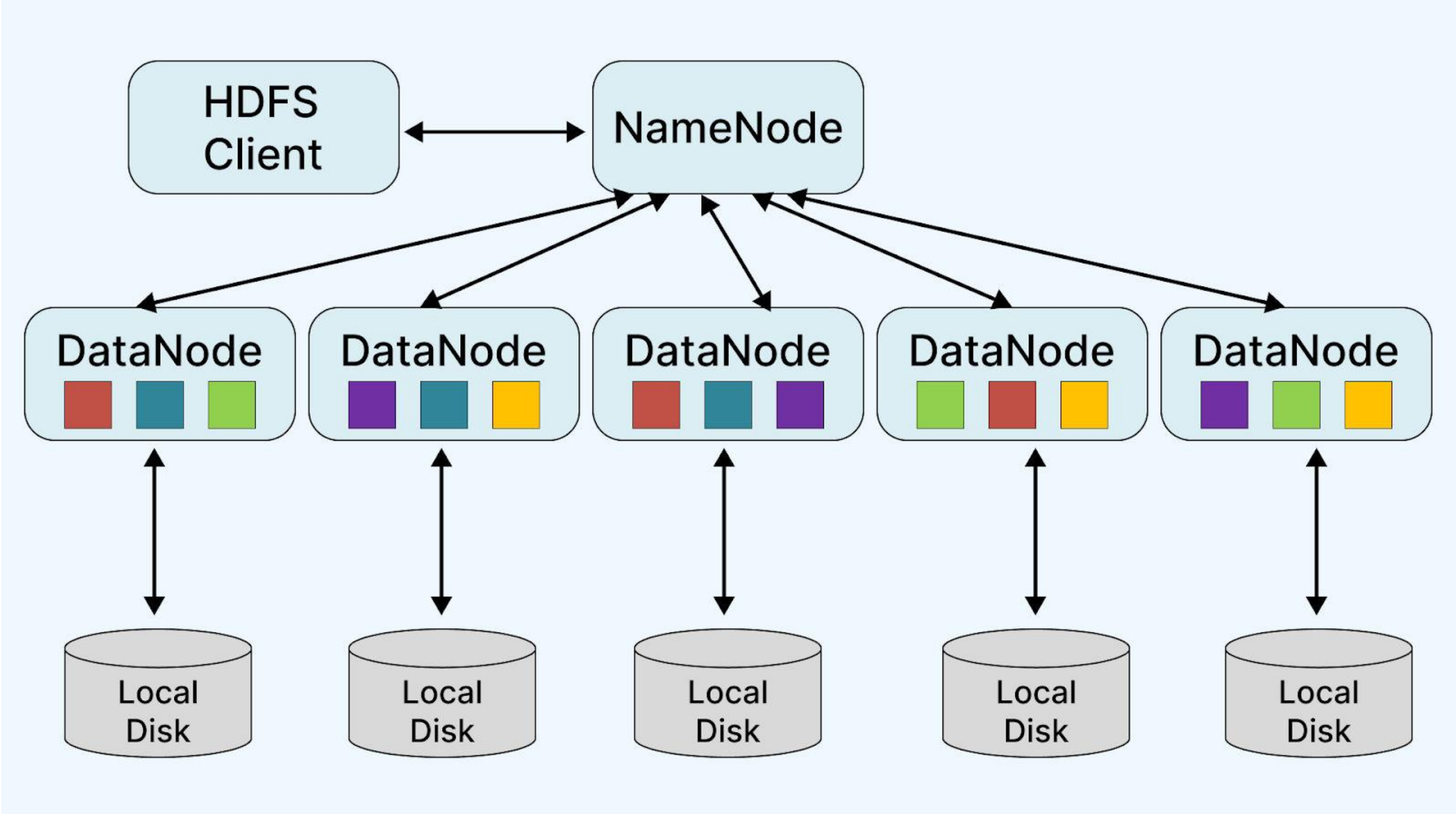
Hadoop Ecosystem



Hadoop Architecture

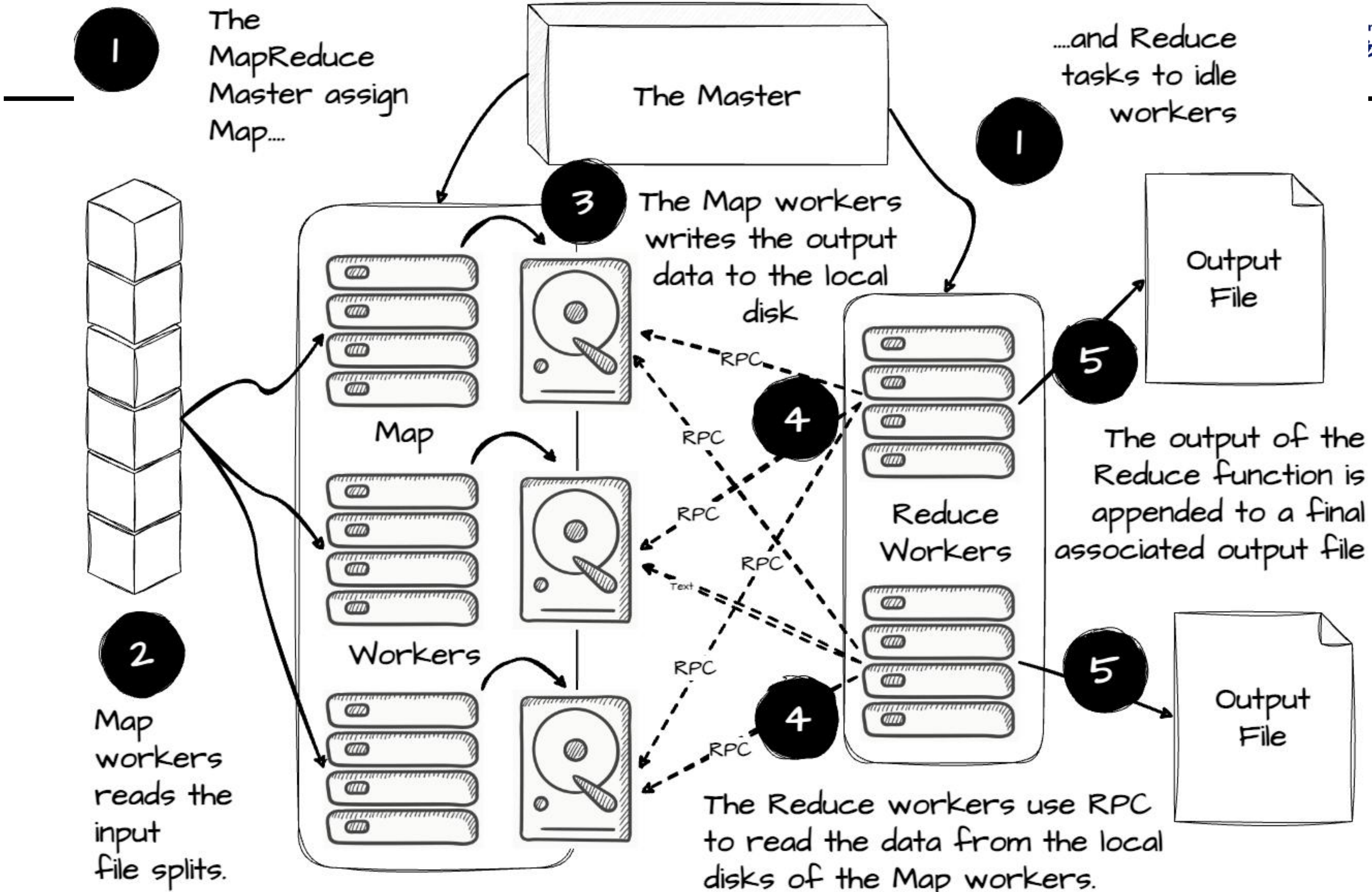


Hadoop Cluster



What is MapReduce?

- A programming model and computing framework for large-scale data processing.
  - The core concept is "divide and conquer."
  - Break down complex tasks into multiple smaller tasks, process them in parallel across a cluster, and then aggregate the results.

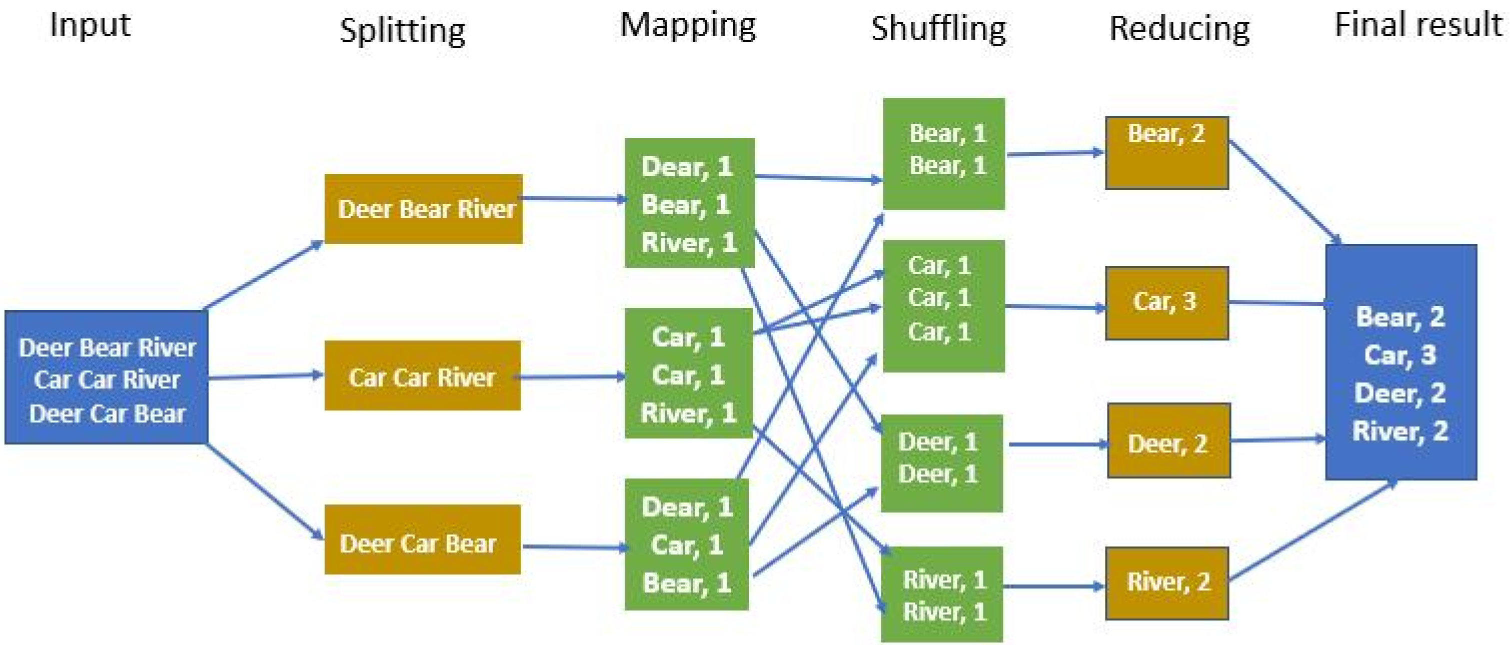


1 The MapReduce Master assign Map....
The Master
....and Reduce tasks to idle workers
1
3 The Map workers writes the output data to the local disk
Output File
Map
Workers
RPC
Text
4
5
The output of the Reduce function is appended to a final associated output file
Reduce Workers
2 Map workers reads the input file splits.
Output File
The Reduce workers use RPC to read the data from the local disks of the Map workers.

A vivid example: counting word frequencies in a library

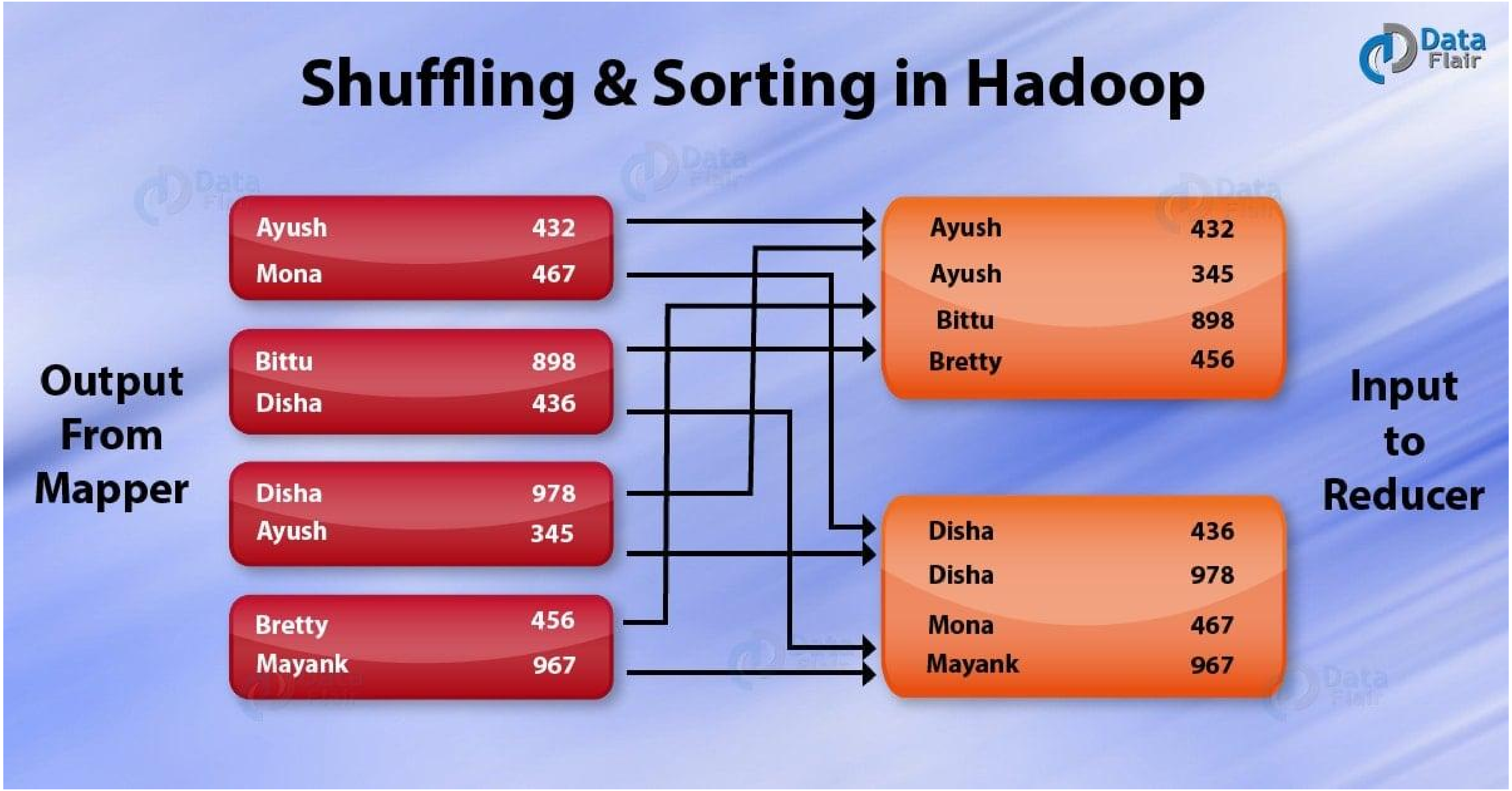
- Question: How can we quickly count the number of occurrences of each word in all the books in the library?
- Traditional method: One person counts each book one by one. Too slow!
- MapReduce Method:
  - Step 1: Map: Have multiple people (workers) each be assigned a few books, and each count the number of occurrences of a word in their own book, outputting an intermediate result such as (word, 1).
  - Step 2: Shuffle & Sort: Group and sort all intermediate results by word, so that all records for the same word are grouped together.
  - Step 3: Reduce: Have multiple people (workers) each be responsible for a subset of words, and sum up all the counts for each word to produce the final result (word, total counts).

The overall MapReduce word count process



Code

- https://www.geeksforgeeks.org/python/hadoop-streaming-using-python-word-count-problem/



- MapReduce = Map + Shuffle + Reduce