Introduction to Python and Spark

Agenda

- Python Basics
- Data Types, Variables, and Basic Operations
- Control Structures: if, for, while
- Functions and Modules
- Introduction to Apache Spark
- Overview of Big Data and Spark
- Spark’s Architecture and Components
- Setting up Spark with Python (PySpark)

Python Basics

- Python is a high-level, interpreted programming language known for its simplicity and readability.
- Used in web development, data analysis, machine learning, and more.
- Instagram uses Python to handle millions of users, allowing for rapid development and scalability.

Sample Code:

```
print('Hello, World!')
```

Data Types, Variables, and Basic Operations

- Data Types, Variables, and Basic Operations
- Understand how to store and manipulate data in Python
- Calculate monthly expenses for the company's finance department

Data Types, Variables, and Basic Operations: Cont..

Data Types

Python supports various data types such as integers, floats, strings, lists, tuples, and dictionaries.

Variables

Variables are used to store data values.

Basic Operations

Python can perform arithmetic operations like addition, subtraction, multiplication, and division, as well as comparison operations like equal to, not equal to, greater than, and less than.

Data Types, Variables, and Basic Operations

- Variables and operations used in budgeting tools
- Calculate expenses and savings

Data Types, Variables, and Basic Operations

Sample Code

```
Variables
    Age: 25
    Name: Alice
Basic Operations
    Sum: 10 + 5 = 15
    Product: 10 * 5 = 50
```

Print Results

```
    Age: 25
    Name: Alice
    Sum: 15
    Product: 50
```

Control Structures: if, for, while

- if: Generates reports based on specific conditions
- for: Iterates through lists of data
- while: Performs repetitive tasks until a condition is met

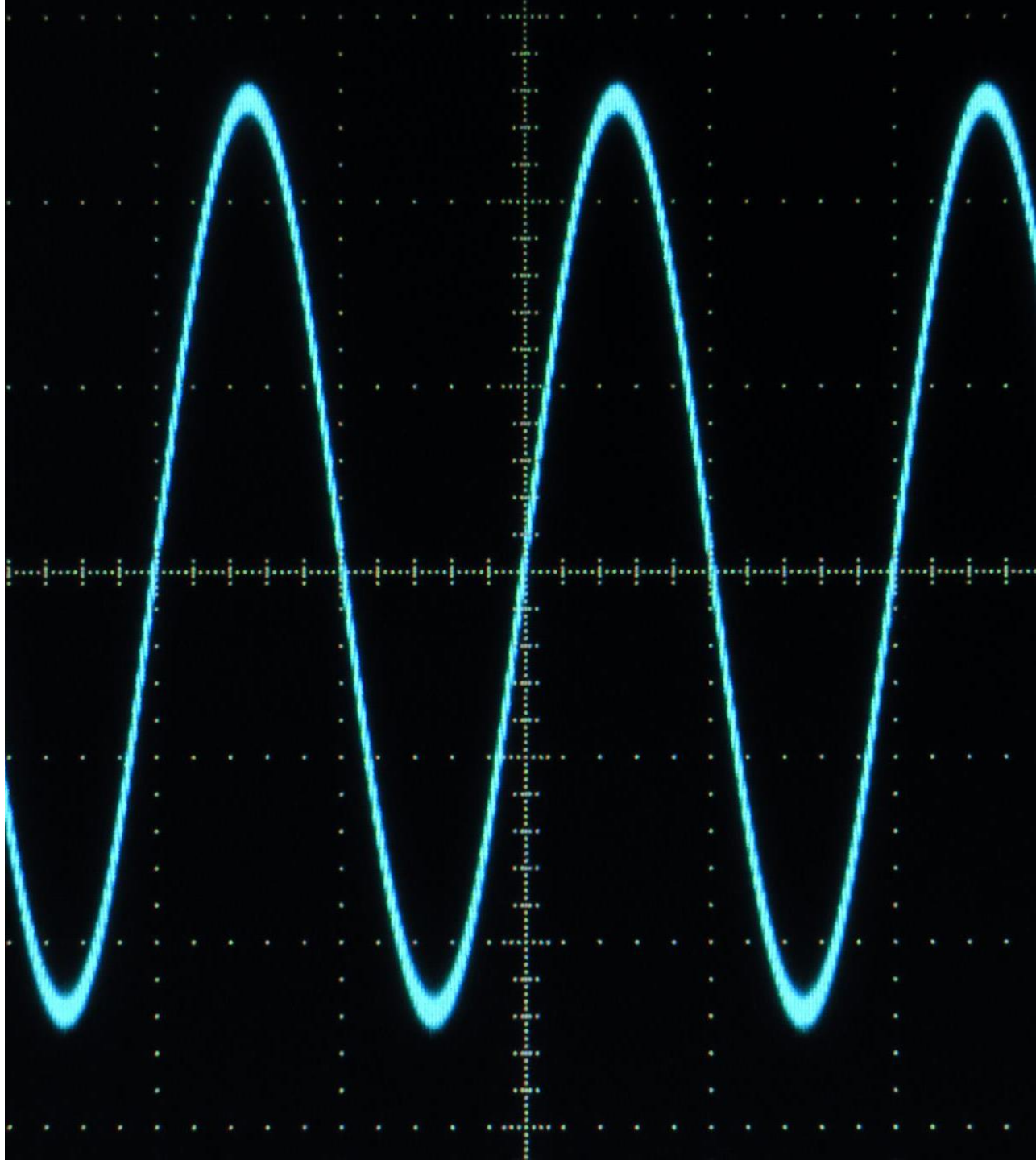
Control Structures: if, for, while

- if Statements
- Used for conditional execution
- for Loops
- Iterate over a sequence (e.g., list, tuple)
- while Loops
- Repeat a block of code while a condition is true

Control Structures: if, for, while

Control structures are used in automation scripts

Example: sending alerts based on sensor data



Control Structures: if, for, while

If Statement

Used to check a condition and execute code accordingly

For Loop

Iterates over a sequence and executes code for each element

While Loop

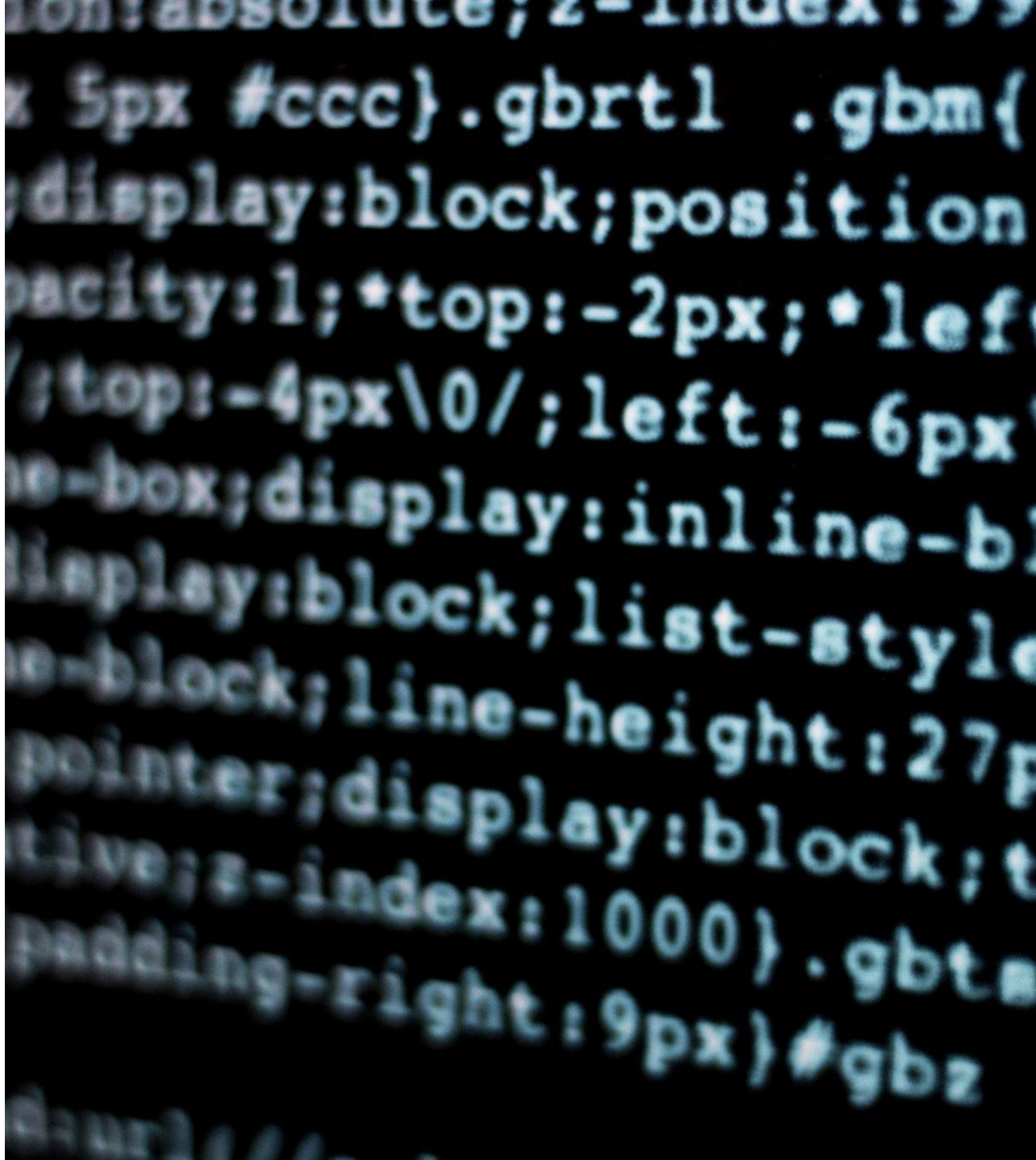
Executes code while a condition is true

Sample Code:

```
# if statement

age = 18

if age >= 18:

print("You are an adult.")

# for loop

numbers = [1, 2, 3, 4, 5]

for number in numbers:

print(number)

# while loop

count = 0

while count < 5:

print(count)

count += 1

This code snippet demonstrates the use of control

structures to control the flow of your program.
```

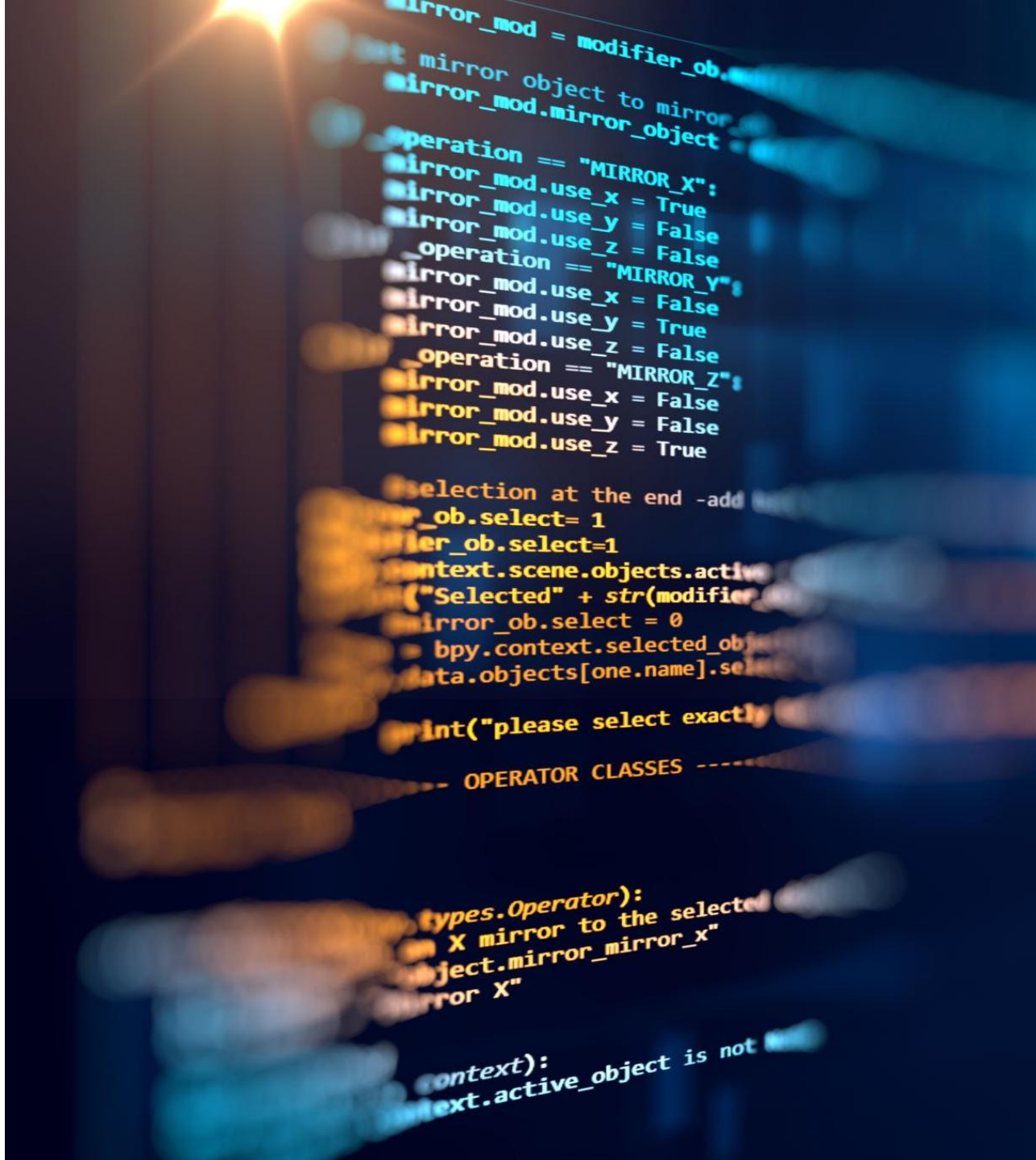
Functions and Modules

Organize code into reusable blocks
Keeps code clean and maintainable



Functions and Modules

- Functions
- Reusable blocks of code that perform specific tasks
- Modules
- Files containing Python code (functions, classes, variables) that can be imported and used in other programs

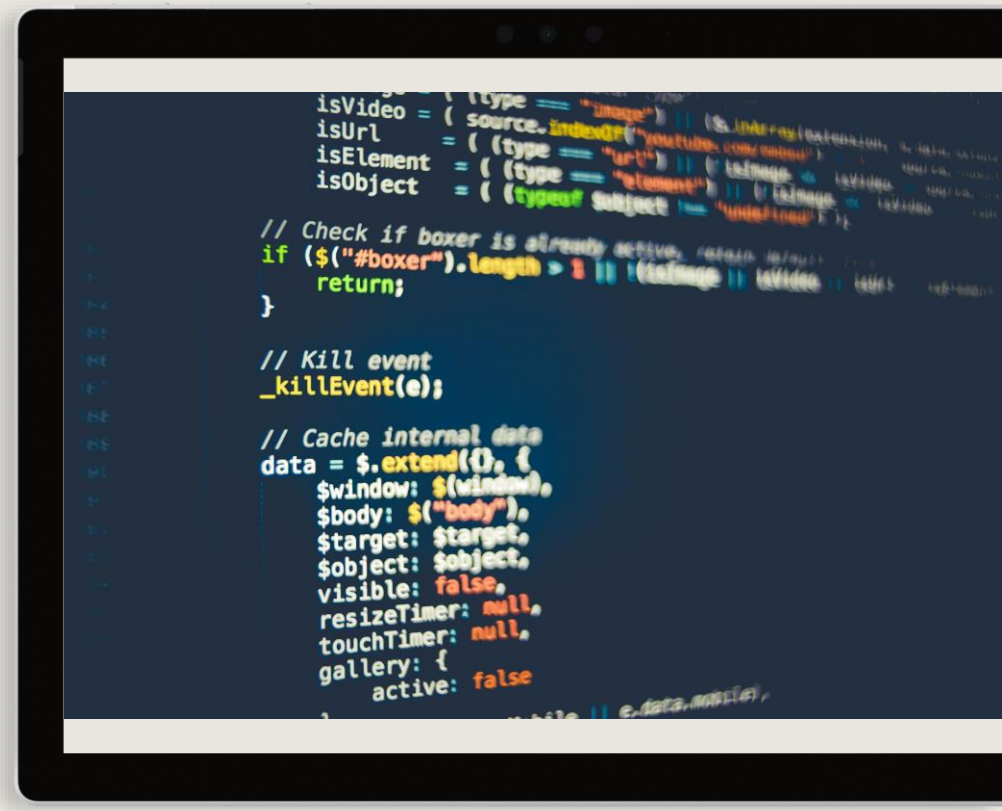


Functions and Modules

- Functions and modules are used to build modular and maintainable codebases
- Example: web applications

Functions and Modules

- Function Definition
- Defines a function to greet a person by name
- Module Example
- Defines a module with an add function
- Importing the module and using the add function



Sample Code:

```
# Function
def greet(name):
return f"Hello, {name}!"
print(greet("Alice"))
# Module example (save as my_module.py)
def add(a, b):
return a + b
# Importing the module
import my_module
print(my_module.add(3, 4))
```

This code snippet shows how to define a function and use a module to organize your code.

Introduction to Apache Spark

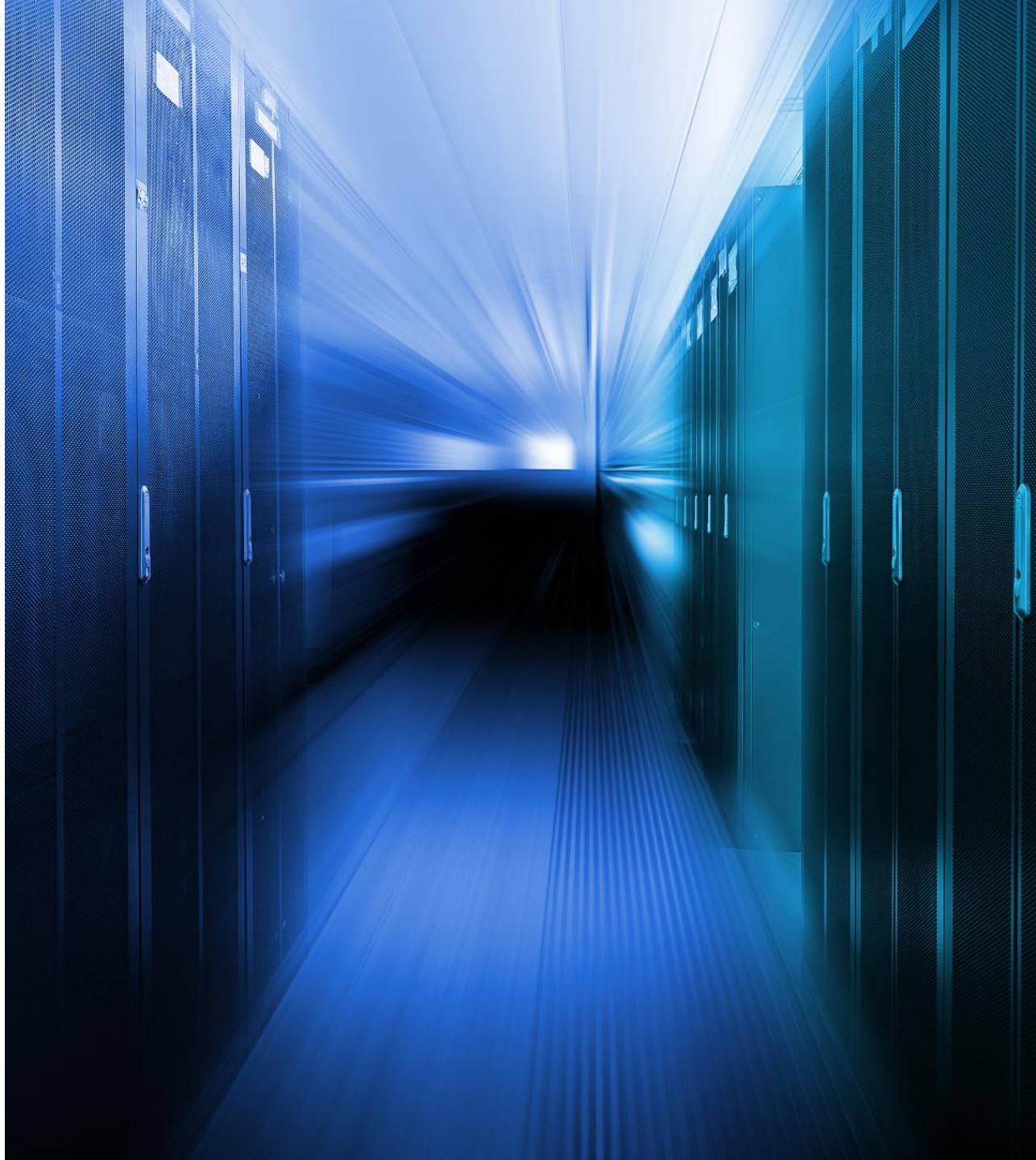
- Apache Spark is an open-source unified analytics engine for large-scale data processing.
- It provides an interface for programming entire clusters with implicit data parallelism and fault tolerance.
- Real-life Example: Netflix uses Spark to process petabytes of data for streaming recommendations and analytics.

Overview of Big Data and Spark

Understanding Big Data
Grasping the concept of Big Data
Handling Big Data with Spark
How Spark can handle Big Data

Overview of Big Data and Spark

- Big Data
- Refers to datasets that are too large or complex for traditional data-processing software.
- Spark's Role
- Efficiently processes big data through parallel computing.

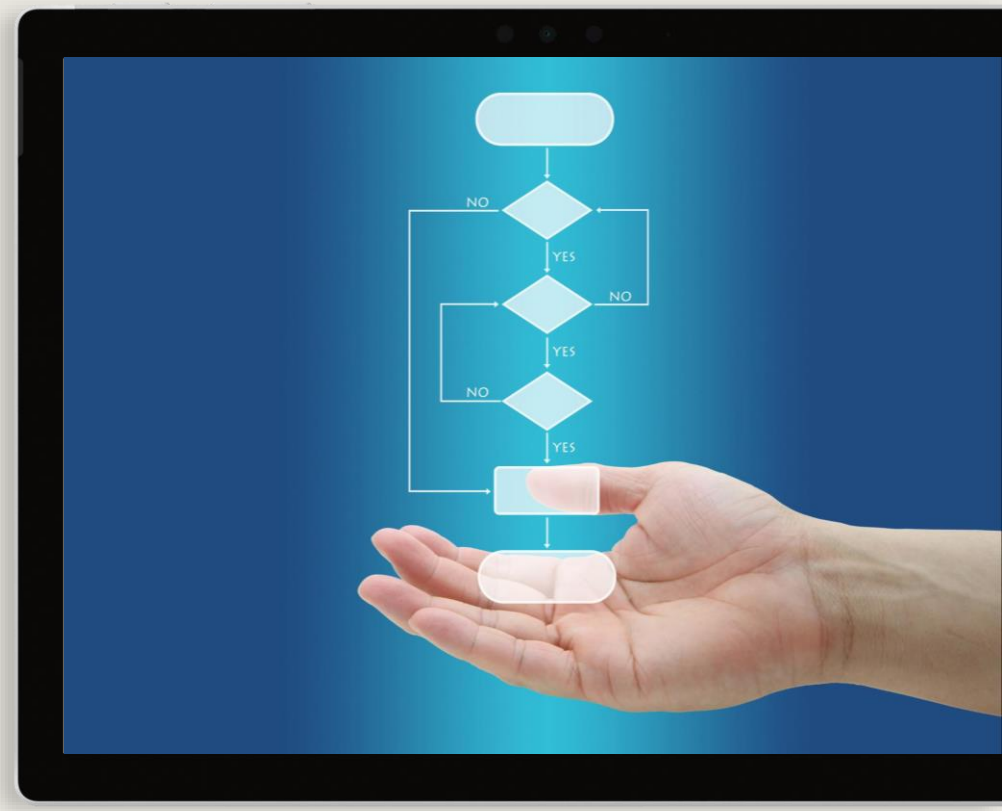


Overview of Big Data and Spark

- Real-life Example: Improving Product Recommendations
- Analysis of user behavior on e-commerce sites

Spark’s Architecture and Components

- Spark's Architecture and Components
- Enables efficient data processing



Spark’s Architecture and Components

- **Cluster Manager:**

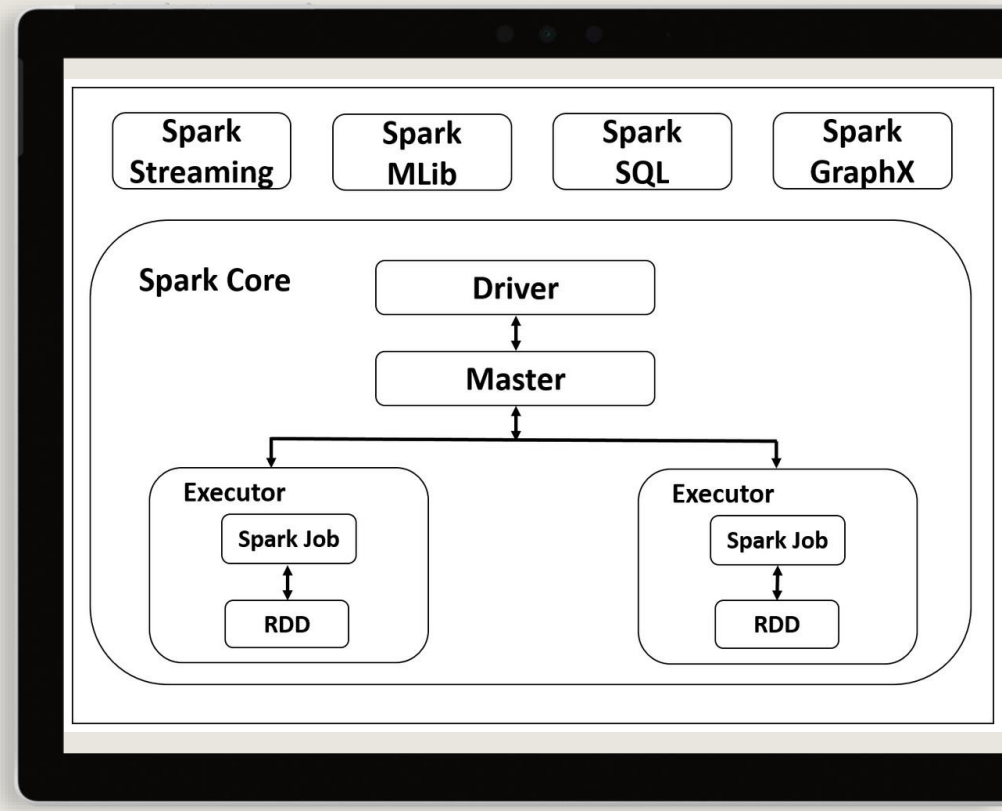
  Spark relies on a cluster manager to oversee the allocation of resources and coordinate tasks across the cluster. Analogous to a traffic cop managing the flow of vehicles at a busy intersection, the cluster manager ensures that each worker node gets its fair share of resources and assigns tasks efficiently.

- **Driver Program:**

  The driver program is the main coordinator, responsible for creating the SparkContext and orchestrating the overall execution of the Spark application. Picture the driver program as the director of a play, guiding the actors (worker nodes) to perform their roles according to the script (Spark application).

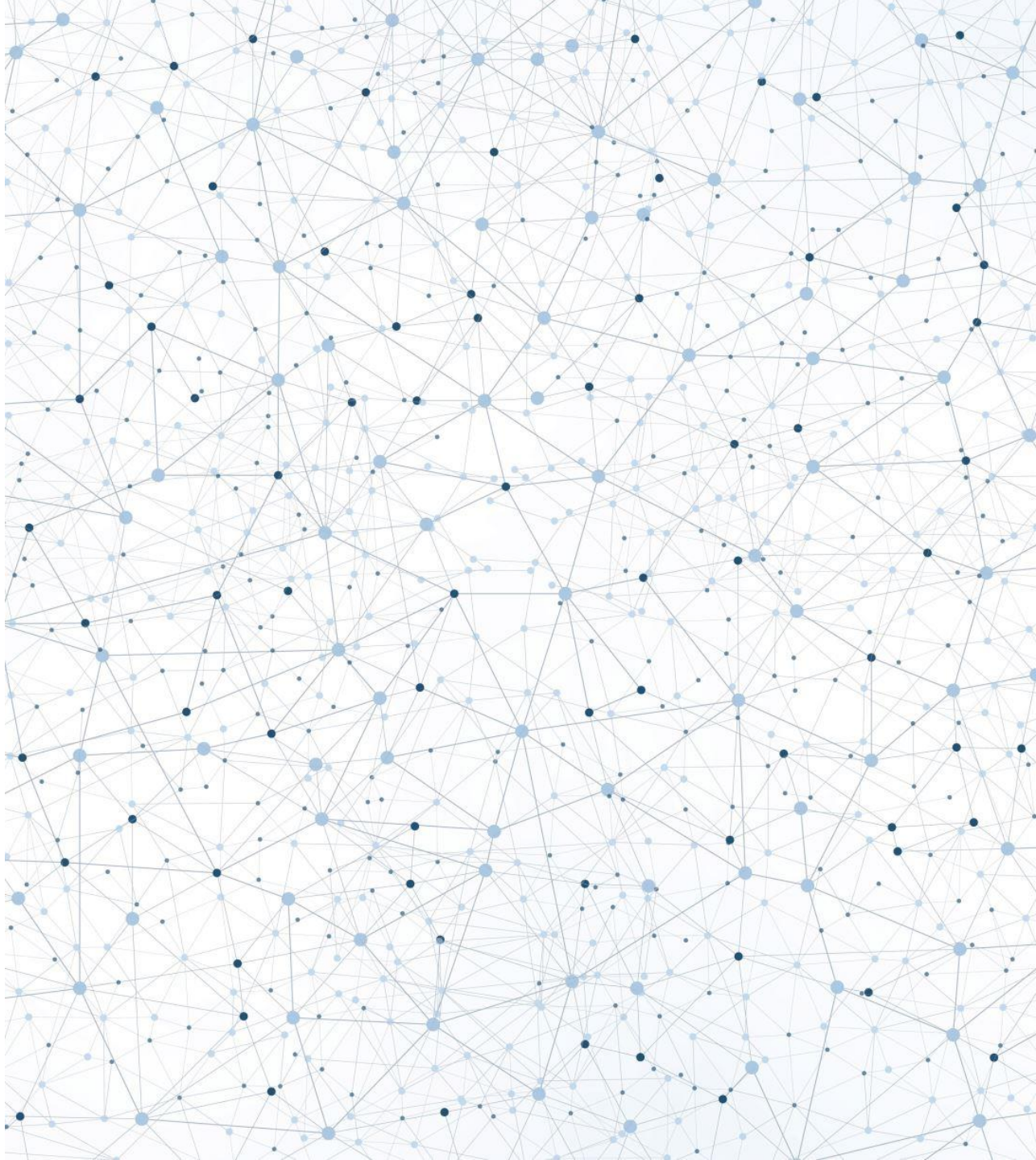
- **Executor Nodes:**

  Worker nodes, or executor nodes, are the actors in our analogy. They perform the actual computations on the data and report back to the driver program. Just like actors following the director's instructions, executor nodes execute tasks assigned by the Spark Context.



Spark’s Architecture and Components

- Scalable Data Processing
- Enables processing in cloud environments
- Used by Companies
- Uber analyzes real-time data for ride optimization



Spark Eco-System

- **Spark Core**

  Spark Core is the base engine for large-scale parallel and distributed data processing. Further, additional libraries which are built on the top of the core allows diverse workloads for streaming, SQL, and machine learning. It is responsible for memory management and fault recovery, scheduling, distributing and monitoring jobs on a cluster & interacting with storage systems.

- **Spark Streaming**

  Spark Streaming is the component of Spark which is used to process real-time streaming data. Thus, it is a useful addition to the core Spark API. It enables high-throughput and fault-tolerant stream processing of live data streams.

- **Spark SQL**

  Spark SQL is a new module in Spark which integrates relational processing with Spark's functional programming API. It supports querying data either via SQL or via the Hive Query Language. For those of you familiar with RDBMS, Spark SQL will be an easy transition from your earlier tools where you can extend the boundaries of traditional relational data processing.

- **GraphX**

  GraphX is the Spark API for graphs and graph-parallel computation. Thus, it extends the Spark RDD with a Resilient Distributed Property Graph. At a high-level, GraphX extends the Spark RDD abstraction by introducing the Resilient Distributed Property Graph (a directed multigraph with properties attached to each vertex and edge).

- **MLlib (Machine Learning)**

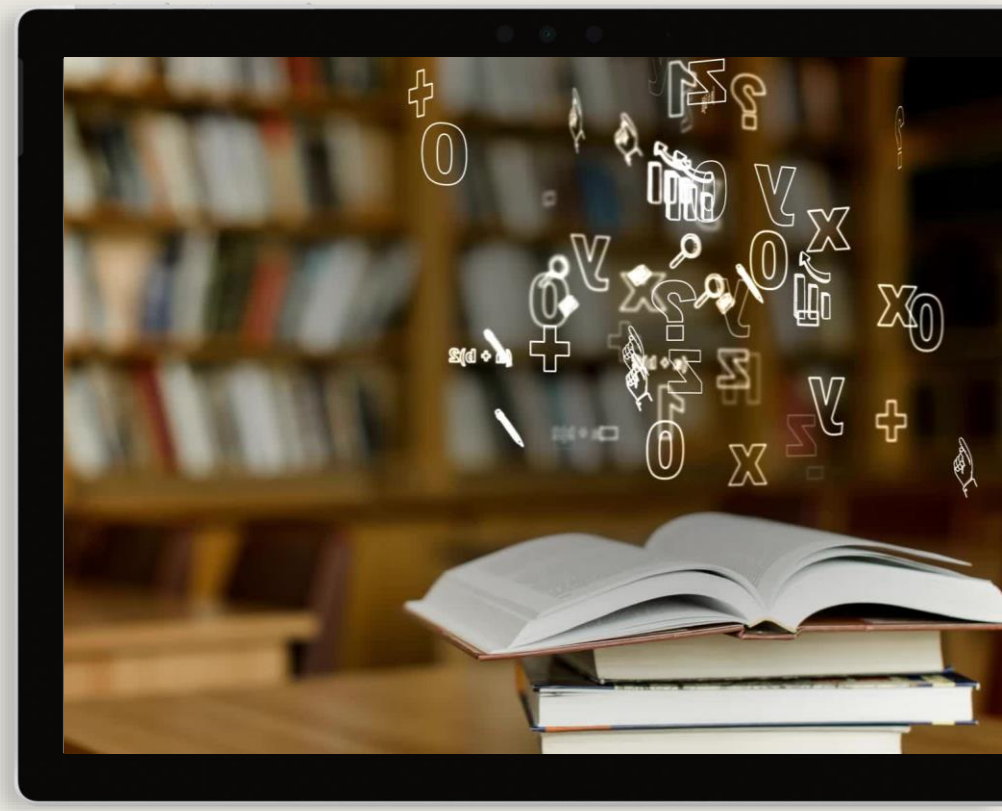
  MLlib stands for Machine Learning Library. Spark MLlib is used to perform machine learning in Apache Spark.

- **SparkR**

  It is an R package that provides a distributed data frame implementation. It also supports operations like selection, filtering, aggregation but on large data-sets.

Setting up Spark with Python (PySpark)

- Install necessary tools and libraries



Setting up Spark with Python (PySpark

- Install Java Development Kit (JDK)
- Download and Install Apache Spark
- Install PySpark using pip

Setting up Spark with Python (PySpark)

Code demonstrates how to set up a Spark session and create a DataFrame in PySpark

Uses SparkSession.builder.appName() to create Spark session

Creates DataFrame with sample data and specified columns

Displays DataFrame using df.show() method

```
from pyspark.sql import SparkSession
# Create Spark session
spark =
SparkSession.builder.appName("example").getOrC
reate()
# Create DataFrame
data = [("Alice", 34), ("Bob", 45), ("Cathy",
29)]
columns = ["Name", "Age"]
df = spark.createDataFrame(data, columns)
# Show DataFrame
df.show()
This code snippet demonstrates how to set up a
Spark session and create a DataFrame in
PySpark.
```

Setting up Spark with Python (PySpark)

- PySpark for Processing Large-Scale Data
- Log analysis
- Fraud detection