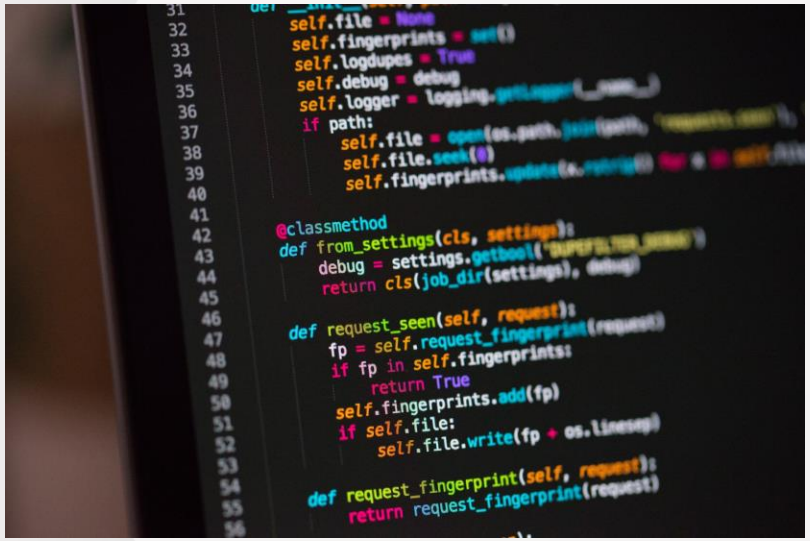
Level 5 Data Engineer
Module 3 Topic 7

Parallel Programming



L5 Data Engineer Higher Apprenticeship
Module 3 / 12 (“**Programming and Scripting Essentials**”)
Topic 7 / 9

The real-world value of parallel programming

How does parallel programming deliver value across industries?

Government departments

- Justice data quality enhanced via probabilistic record linkage
- Large datasets processed efficiently with parallel programming

The NHS

- Databricks used for real-time transactional data analysis, aiding in health risk identification

Finance

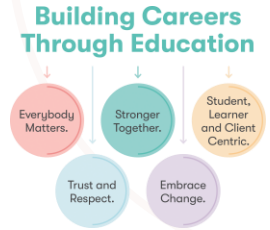
- Parallel programming used for real-time transactional data analysis and fraud detection
- Analytics used for risk assessment

Retail

- Databricks used for customer data analysis
- Machine learning algorithms predict customer preferences

BPP

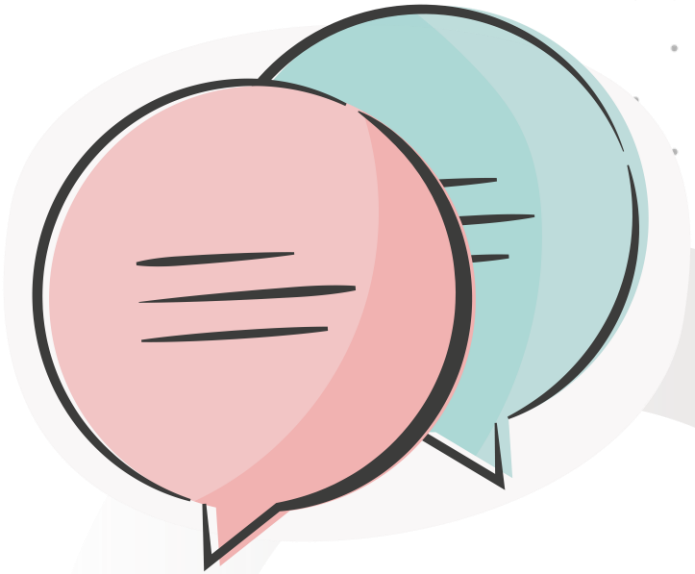
Spotlight on your experience



Share your thoughts on the following questions:

- Can you share the challenges you faced when implementing complex programs and how you overcame them?

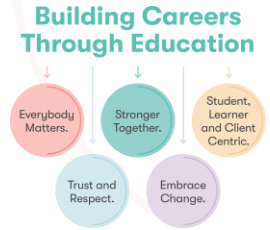
Submit any thoughts to the chat!



Discussion activity

BPP

Session aim and objectives



Completion of this topic supports the following outcomes:

- Identify and explain the key concepts of concurrency, parallelism, and distributed computing
- Explain how key parallelism concepts apply within the context of Python programming
- Report on the benefits of parallelisation
- Analyse performance and deployment considerations of parallel algorithms
- Report on the benefits of Spark for parallelism
- Evaluate platforms similar to Spark

BPP

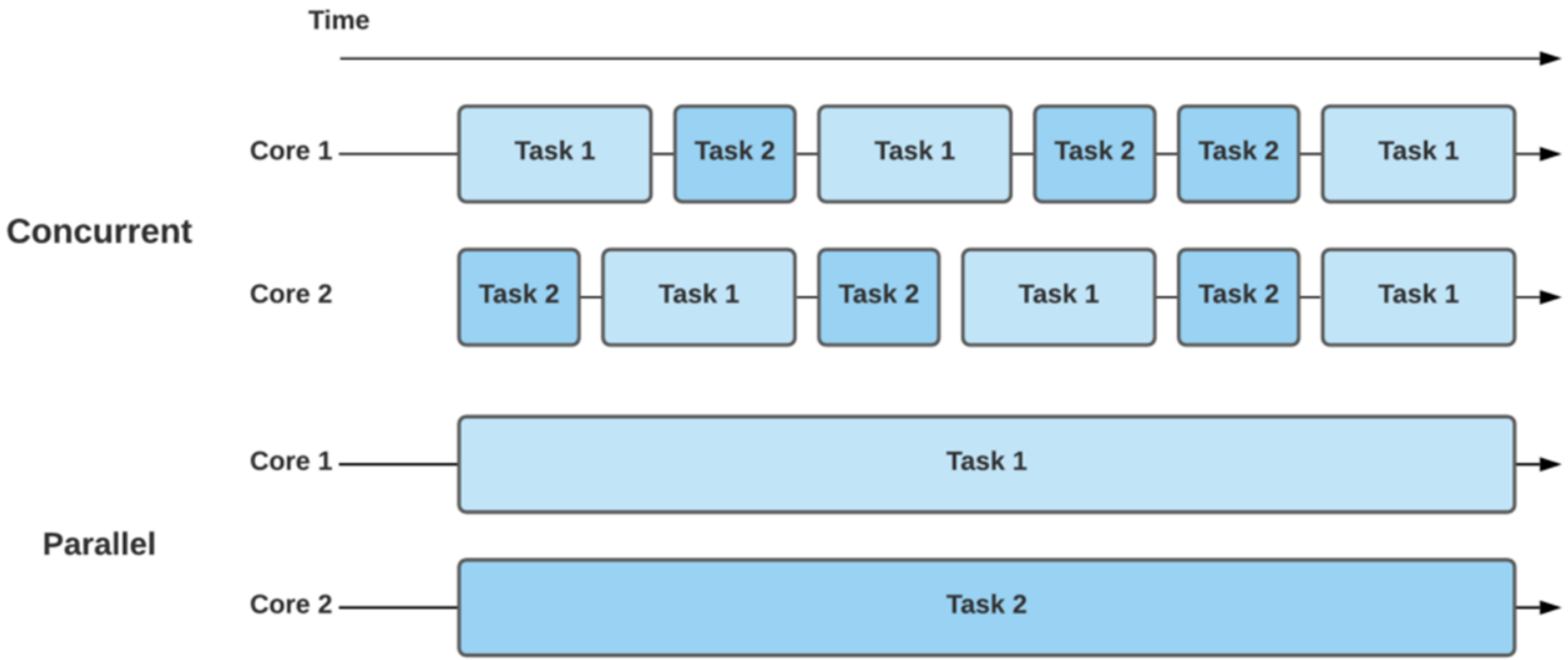
Webinar Agenda

This webinar will include the following:

- Parallelism, multi-processing and multi-threading in Python
- Concurrency fundamentals
- Parallel computing fundamentals
- Apache Spark
- Practical application

Concurrency vs parallelism

What's the difference?

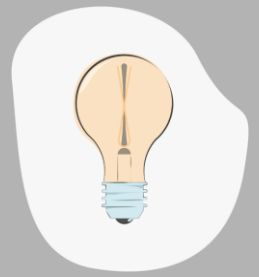


Comparing concurrency vs parallelism

An introduction to parallelism

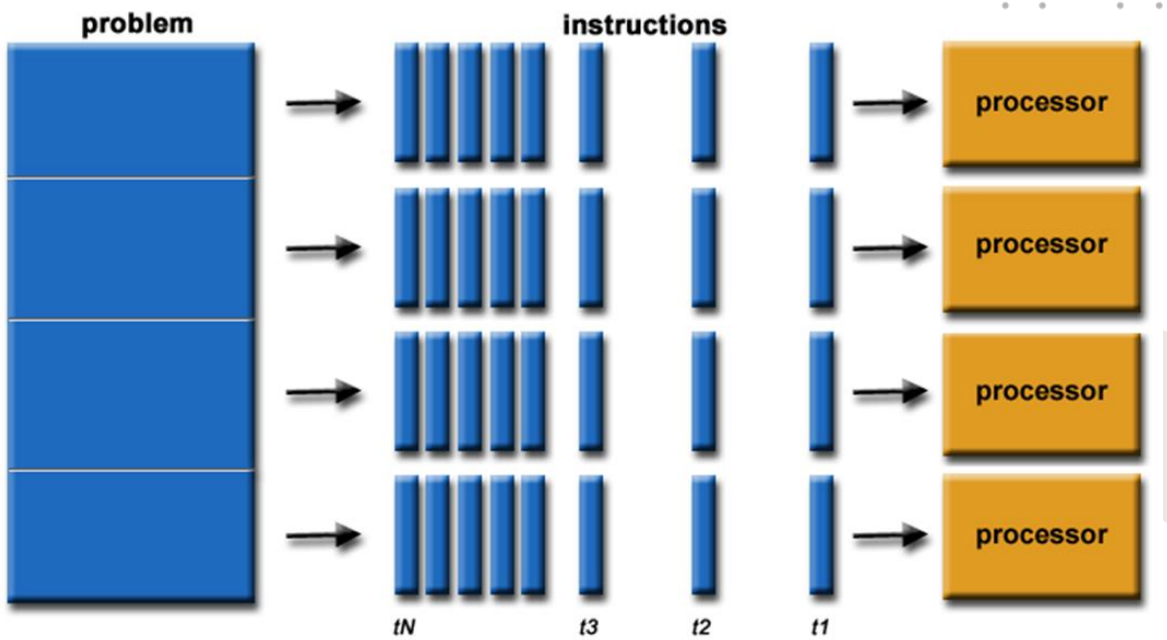
Solving things in parallel

- **Parallelism in programming:** Execution of multiple tasks simultaneously by breaking down a large problem
- **Efficiency:** Leverages modern multi-core and multi-processor systems for problem-solving
- **Design considerations:** Requires careful design for task management and coordination



Use case examples:

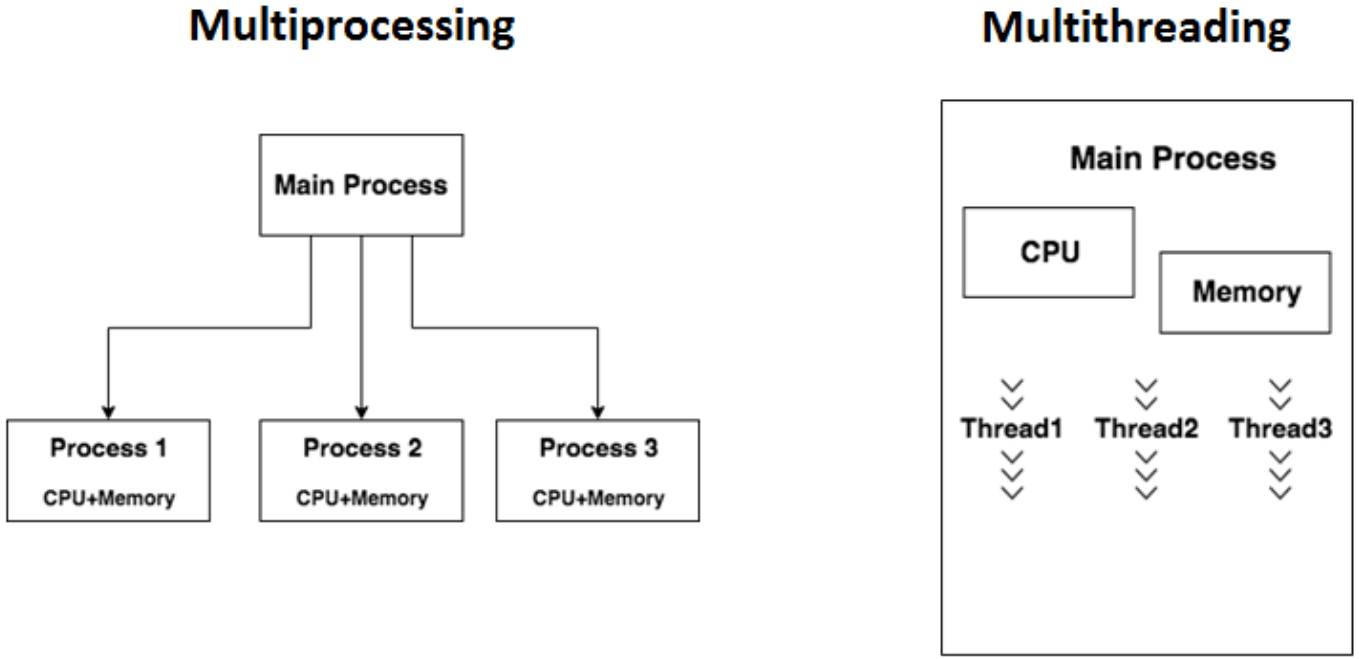
Dynamic simulation and modeling of real-world data, management of large datasets, and any other tasks that require speed and accuracy.



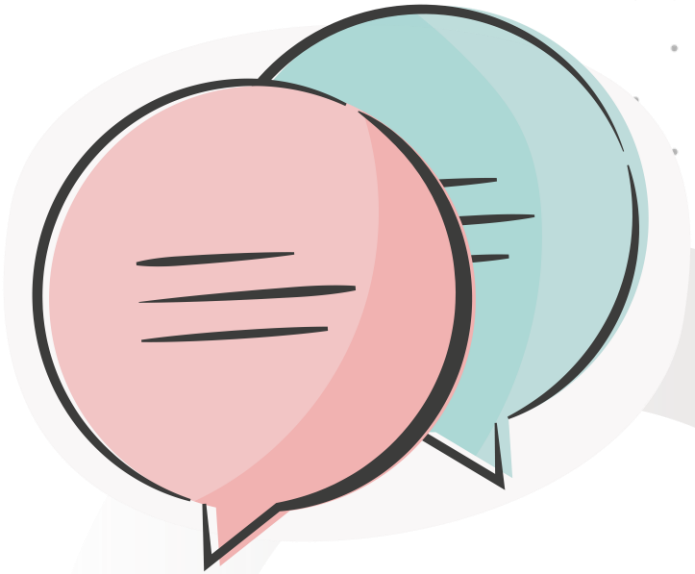
How parallelism works in practice

Multiprocessing and multithreading

Can you remember the difference between a process and a thread



Processes and threads



Discussion activity

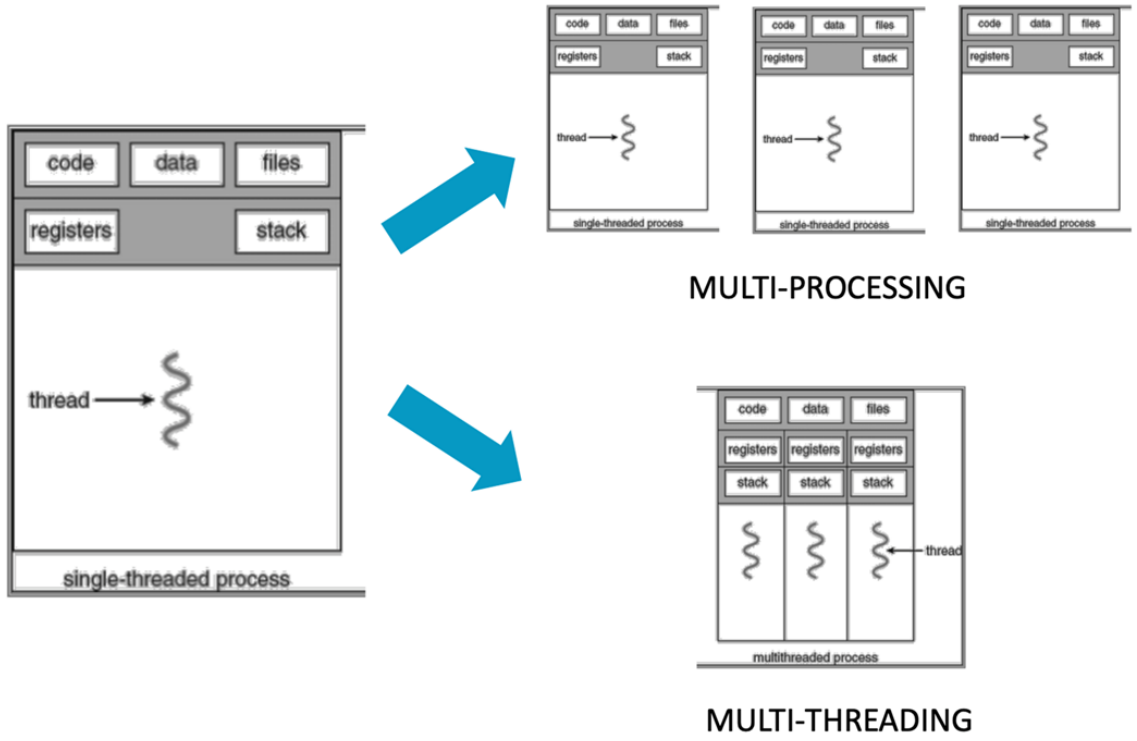
Remember:

- **Threads are share-everything -** The programmer has to lock things that will be shared
- **Processes are share-nothing** - Programmers have to explicitly share useful data/state

Multiprocessing and multi-threading

Two techniques to enhance computing power

- **Multiprocessing:** Multiple processors increase computing speed, ideal for CPU-bound tasks
- **Multithreading:** Processor executes multiple threads concurrently, ideal for IO-bound tasks
- **Address Space & Overhead:** Multiprocessing has separate address space and more overhead, multithreading shares address space and has less overhead
- **Concurrency & Parallelism: Multithreading achieves concurrency, multiprocessing achieves parallelism**



An illustration of multiprocessing and multi-threading

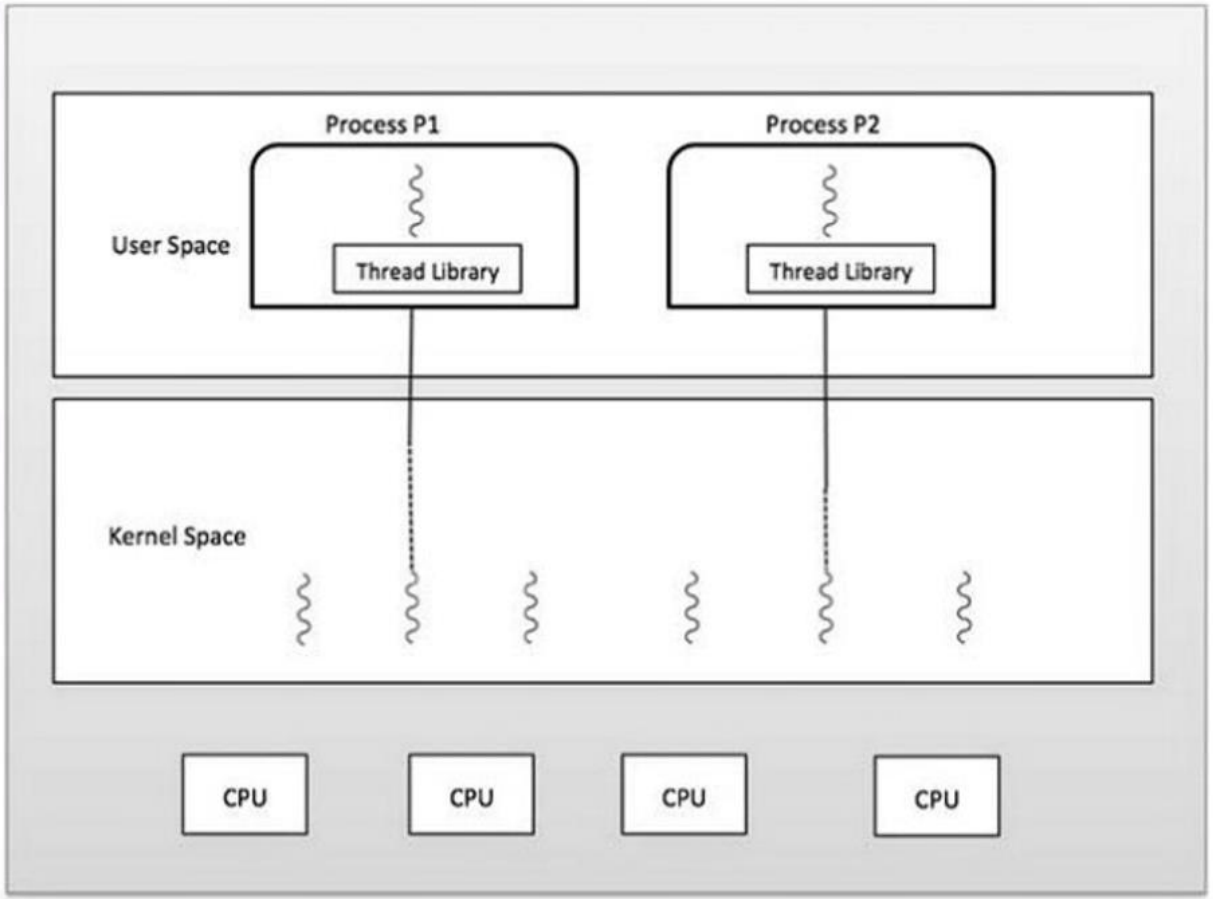
Multiprocessing, multithreading, and scheduling

Parallelism can still work on 1 machine, as modern computers have multiple cores

- A process is an instance of a computer program that is being executed

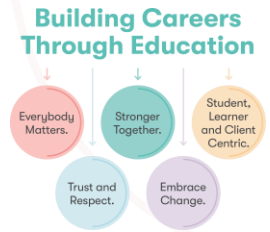
- A process can have 1 or several threads (1 in this hands-on)

- The kernel of the OS schedule threads to multiple cores



A diagram illustrating multiprocessing

Knowledge Check Poll



Which of the following statements is true about distributed computing and parallelism?

A. Distributed computing cannot be achieved with multiple processors

B. Parallelism in programming does not allow the execution of multiple tasks simultaneously

C. Multiprocessing is ideal for IO-bound tasks, while multithreading is ideal for CPU-bound tasks

D. Multithreading achieves concurrency, while multiprocessing achieves parallelism

Knowledge Check Poll

Building Careers Through Education

Everybody Matters. Stronger Together. Student, Learner and Client Centric. Trust and Respect. Embrace Change.

Which of the following statements is true about distributed computing and parallelism?

A. Distributed computing cannot be achieved with multiple processors

B. Parallelism in programming does not allow the execution of multiple tasks simultaneously

C. Multiprocessing is ideal for IO-bound tasks, while multithreading is ideal for CPU-bound tasks

D. Multithreading achieves concurrency, while multiprocessing achieves parallelism

Feedback

The correct statement is **D** – Multithreading allows multiple threads to be executed concurrently by a single processor, while multiprocessing uses multiple processors to achieve true parallelism.

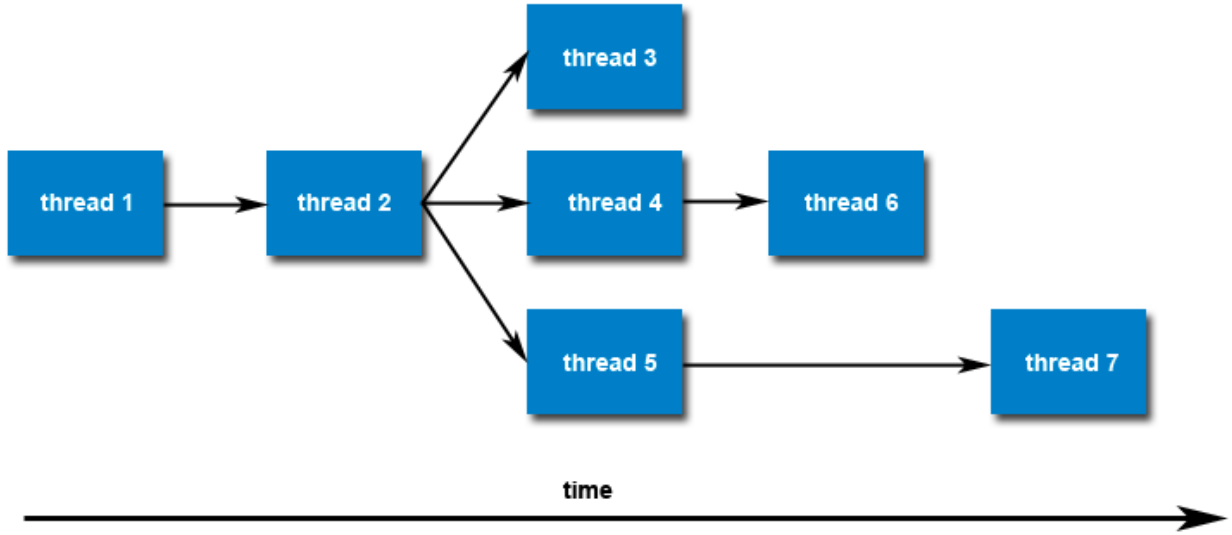
BPP

Parallelism, multi-processing and multi-threading in Python

BPP

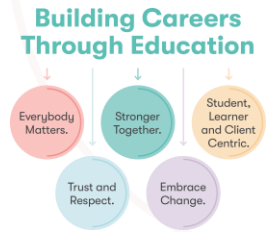
Python threads

- Python has real threads (created with help of the OS)
- Use `threading.Thread`
- Python only allows a single thread to be executing within the interpreter at once. This is concurrency rather than parallelism
- Enforced by the GIL (Global Interpreter Lock)
- Parallelism in Python can be achieved by instantiating multiple processes

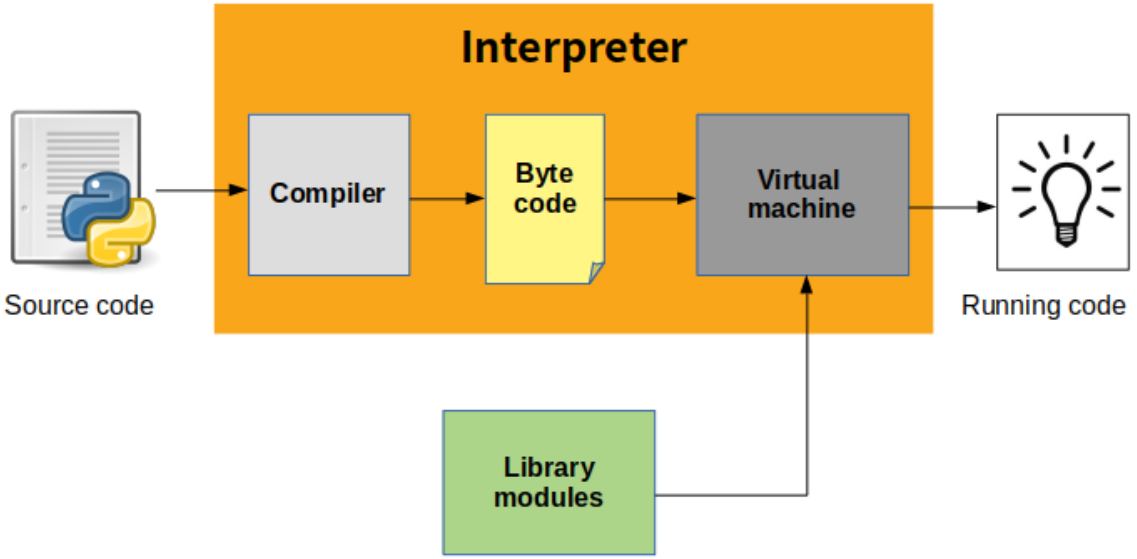


An illustration of Python threads in

Python multiprocessing



- Not bound by the GIL
- Allows data and memory sharing
- Beats the threading module in speed
    - Although process creation is slower
- Use `multiprocessing.Process`



An illustration of python compiler and interpreter

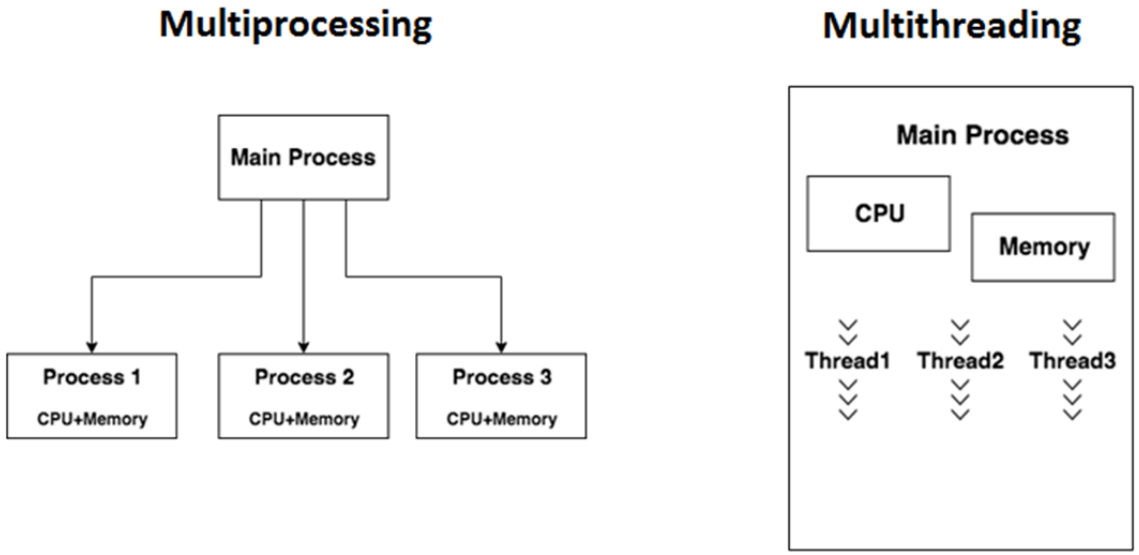
BPP

Creating processes / threads

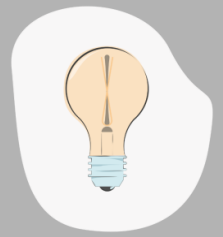
How do you do it?

Exactly like threading:

- Thread (target=func, args=(args,)).start()
- Process (target=func, args=(args,)).start()



Threads and processes in parallel programming



You can subclass multiprocessing.

Process exactly as you would with threading.

Knowledge Check Poll

Which of the following statements is true about Python threads and multiprocessing?

A. Python threads, also known as pthreads, allow multiple threads to execute in parallel within the interpreter

B. The Global Interpreter Lock (GIL) allows multiple threads to execute simultaneously in Python

C. Python's multiprocessing module is not bound by the GIL and allows data and memory sharing

D. In Python, creating processes is faster than creating threads

BPP

Knowledge Check Poll

Which of the following statements is true about Python threads and multiprocessing?

A. Python threads, also known as pthreads, allow multiple threads to execute in parallel within the interpreter

B. The Global Interpreter Lock (GIL) allows multiple threads to execute simultaneously in Python

C. Python's multiprocessing module is not bound by the GIL and allows data and memory sharing

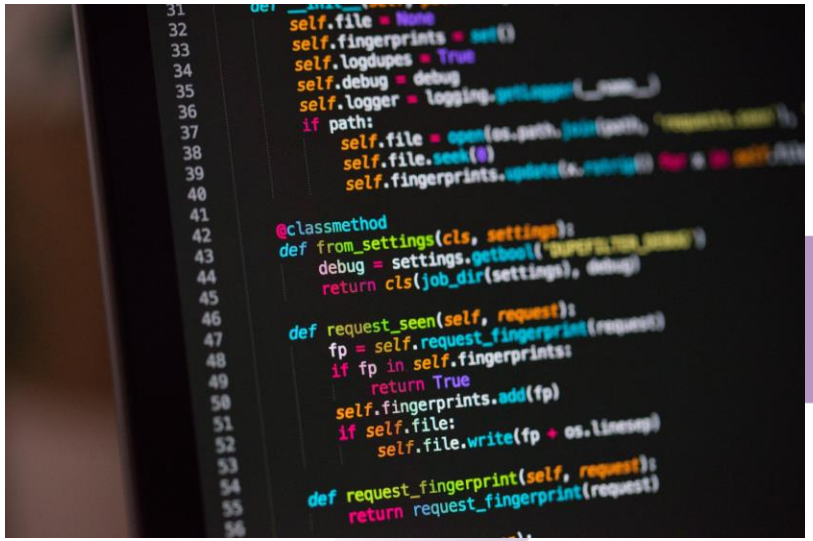
D. In Python, creating processes is faster than creating threads

Feedback

The correct statement is **C** – Python's multiprocessing module is not bound by the GIL and allows data and memory sharing.

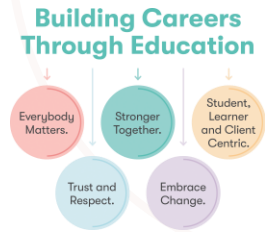
BPP

Concurrency



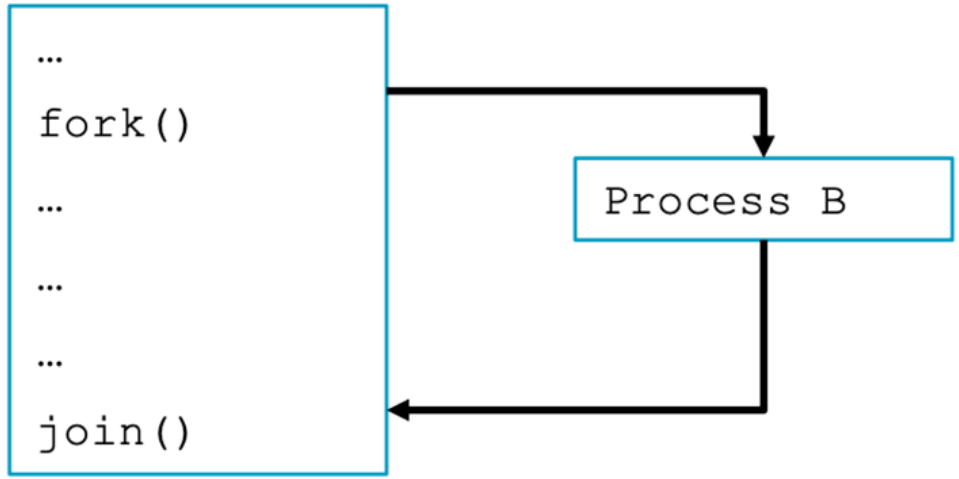
The Fork-Join model

What you need to know…



Fork/Join Execution Model

- Fundamental way of expressing parallelism within a computation
- Fork creates a new child process
- Parent continues after the Fork operation
- Child begins operation separate from the parent
- Parent waits until child joins (continues afterwards)

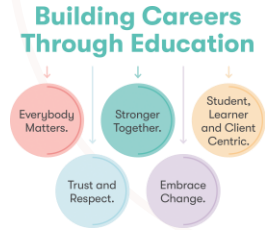


Note that "fork()" is pseudocode

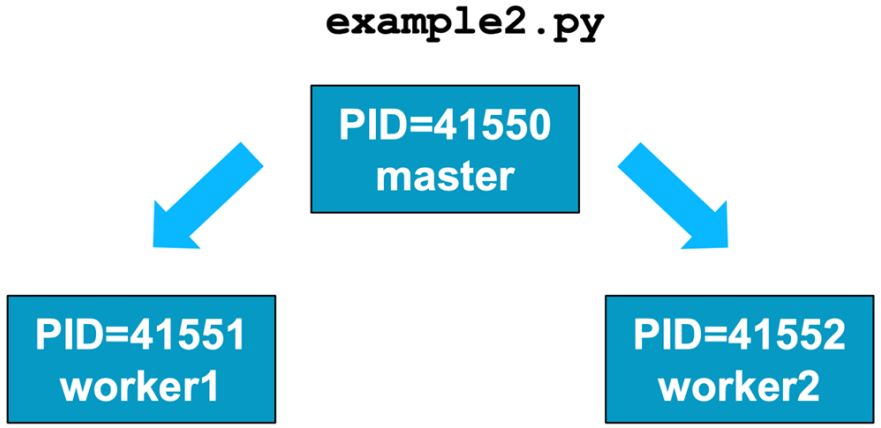
BPP

Processes or threads?

The fork-join model



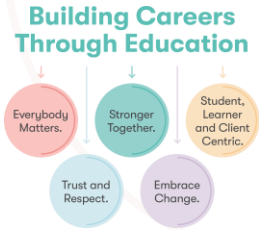
- **Divide and conquer:** It uses a parallel version of the divide and conquer paradigm
- Whether "fork-join" refers to **multiprocessing** or **multithreading** in Python depends on the application's needs for CPU utilization and the nature of the tasks (CPU-bound vs. I/O-bound).
- **Multiprocessing** allows you to utilize full CPU resources for compute-heavy tasks.
- **Threading** in this scenario is generally used for I/O-bound tasks where the main bottleneck is not CPU computation but waiting for external resources (like files or network services)



A simple diagram of the join-fork model

BPP

Python fork-join with Threads



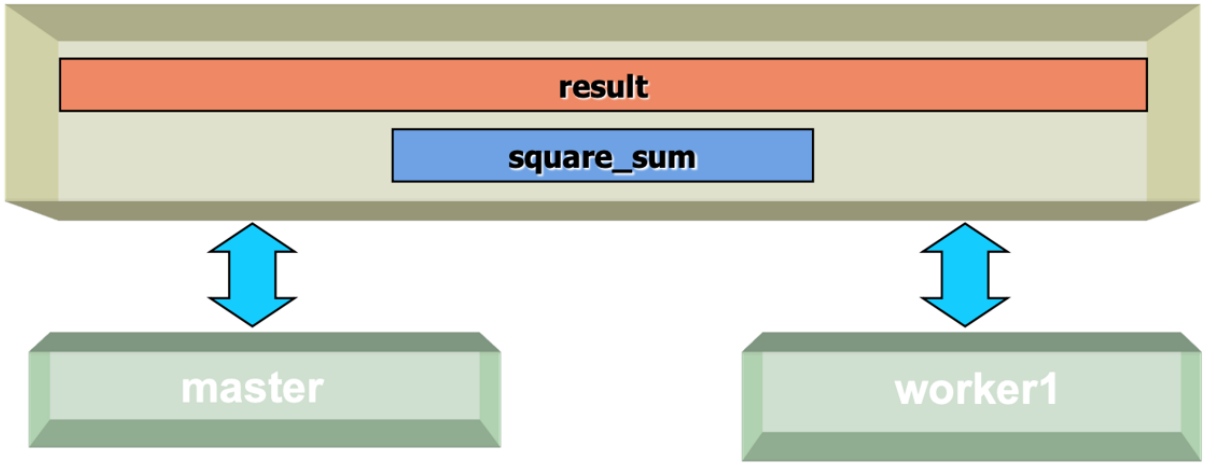

```
# Each job is put in a method which will run on a separate thread
def job1():
    # Job 1 code here

def job2():
    # Job 2 code here

# Create threads for each job
t1 = threading.Thread(target=job1)
t2 = threading.Thread(target=job2)

# Start the threads
t1.start()
t2.start()

# Wait for both threads to finish
t1.join()
t2.join()
```

BPP

Process communication

Shared memory

Most efficient way to share memory across processes

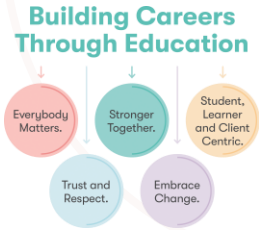


`multiprocessing` module provides `manager` class (**Advanced!**) that

- Share arbitrary object types like lists, dictionaries, Queue, Array, etc.
- A single manager can be shared by processes on different computers
- However, they are slower than using shared memory.

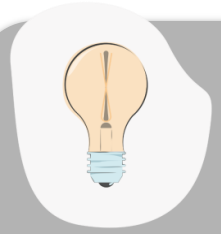
A diagram illustrating process communication and shared memory

Applying processes / threads to big data



Quick guidance

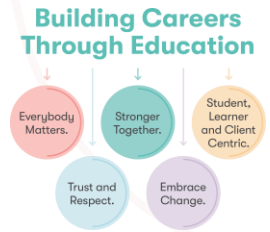
- We won't be applying processes/threads directly in most use cases
- Data Engineers use frameworks to enable multiprocessing at scale



We call this parallelism or parallel computing, and one key framework for parallel programming is called Spark

BPP

Knowledge Check Poll



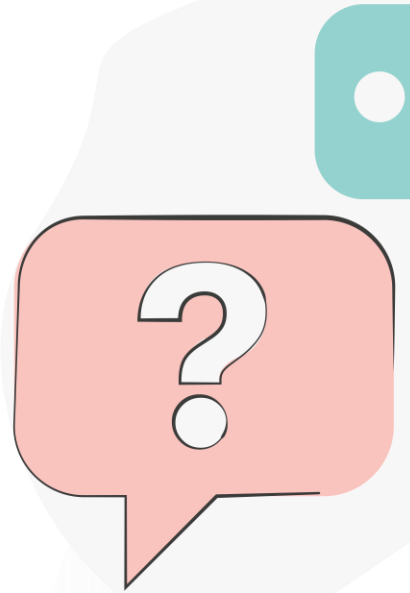
In the context of the Fork/Join Execution Model in concurrency, which of the following statements is correct?

A. The parent process waits until the child process joins before it continues

B. The child process waits until the parent process joins before it continues

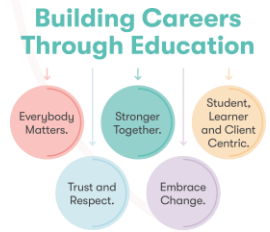
C. Both parent and child processes wait for each other to join before they continue

D. Neither the parent nor the child process waits for the other to join before they continue



BPP

Knowledge Check Poll



In the context of the Fork/Join Execution Model in concurrency, which of the following statements is correct?

A. The parent process waits until the child process joins before it continues

B. The child process waits until the parent process joins before it continues

C. Both parent and child processes wait for each other to join before they continue

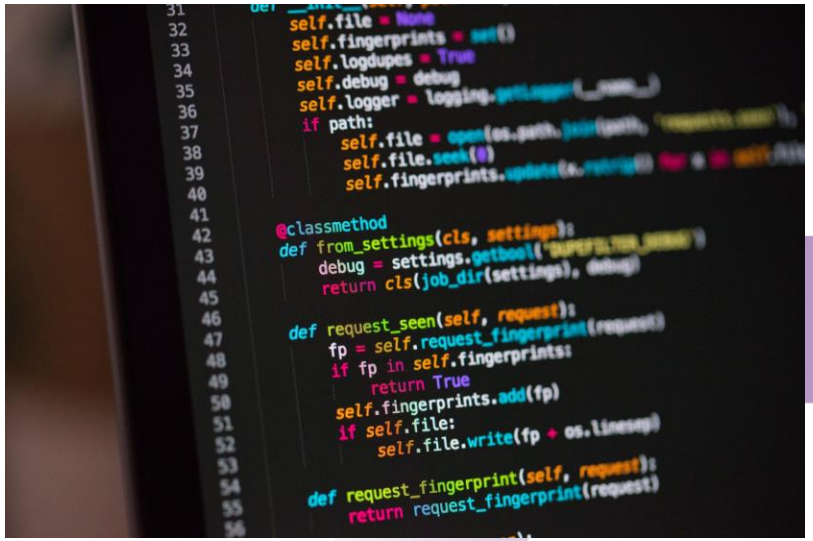
D. Neither the parent nor the child process waits for the other to join before they continue

Feedback

The correct statement is **A** – In the Fork/Join Execution Model, a fork operation creates a new child process.

BPP

Zooming in on: Parallel computing



Recap

What is parallel computing?

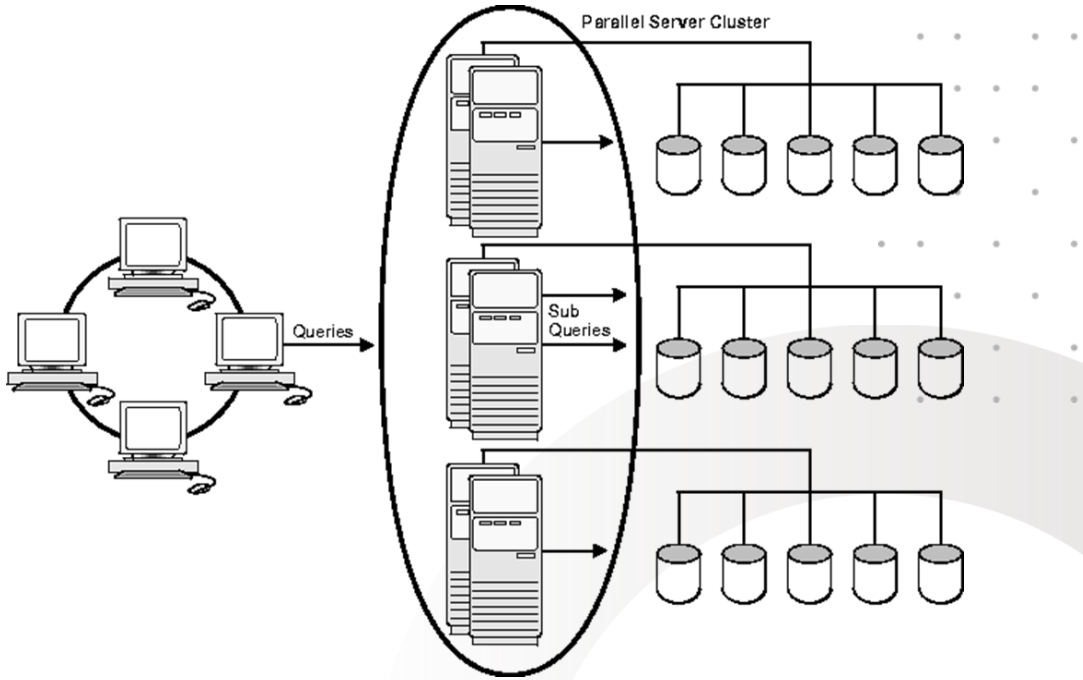
- Computations (processing) carried out simultaneously (at the same time)
- Can be done by CPU or GPU cores
- They can be on one or more machines
- Large problems can be divided into smaller tasks that can be solved in parallel

Parallel computing

Distributed computing

It doesn’t have to be a single machine

- **Distributed computing** uses multiple processors, possibly at different locations
- It enables **larger-scale computations** than one machine
- The model **boosts efficiency** and **cuts processing time**
- This principle is vital to **modern computing infrastructures**



An illustration of distributed computing

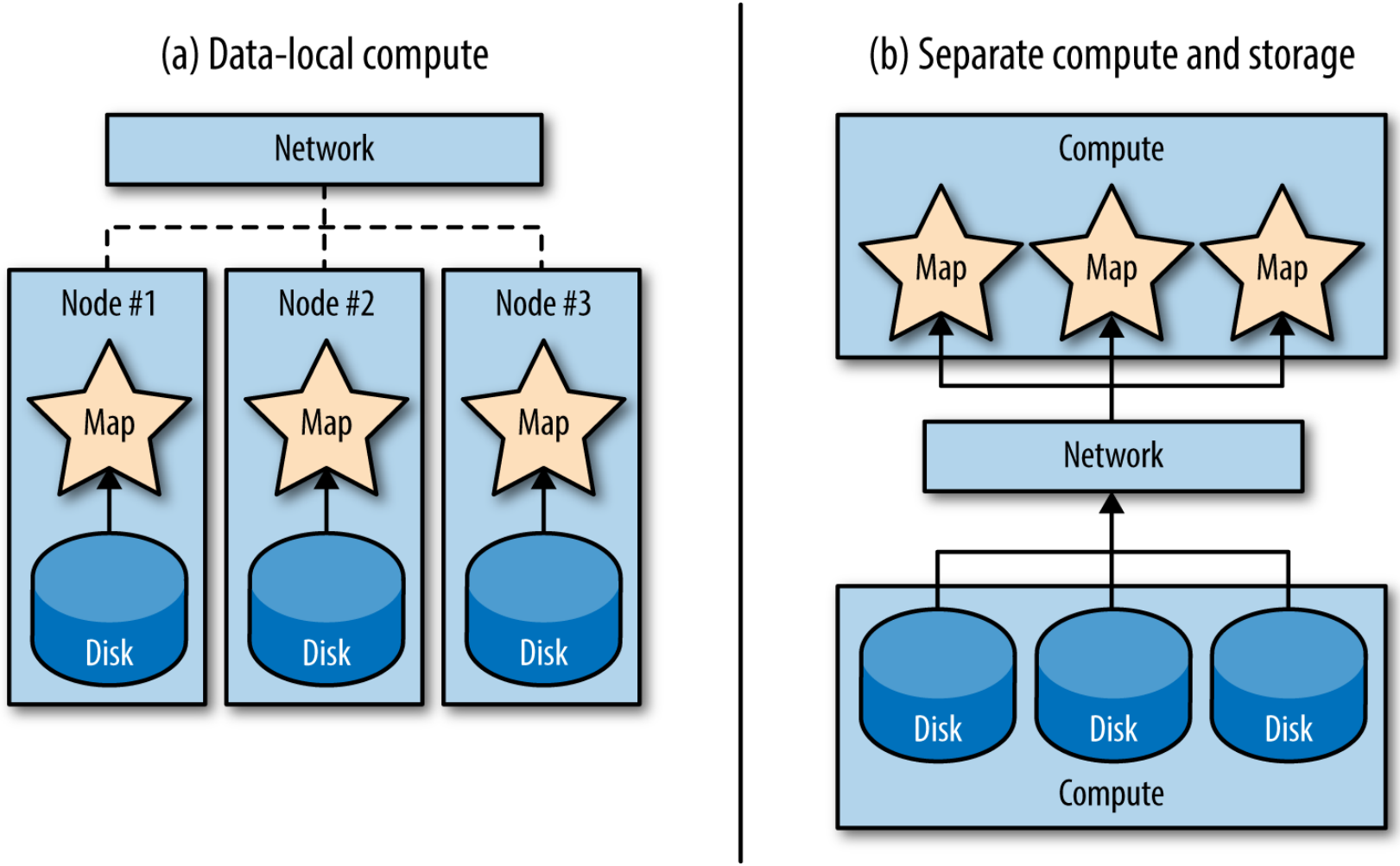
Task parallelism vs data parallelism

What's the difference?

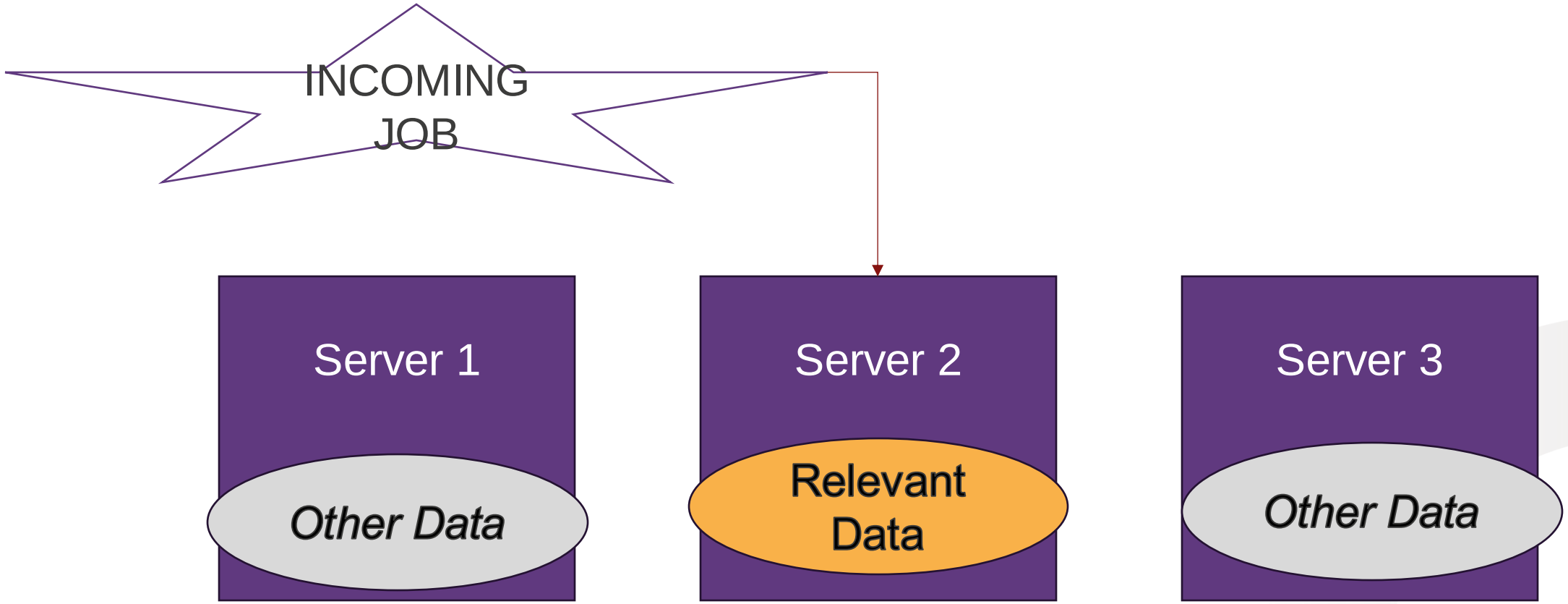
Data parallelism	Task parallelism
Source file is partitioned	Source files(s) are intact
Same operations are performed on different subsets of the same data	Different operations are performed on the same or different data
The number of threads / processes is normally dependent on data sizes	The number of threads / processes is normally decided by the developer
Tasks are similar to each other	Tasks can be very different from each other

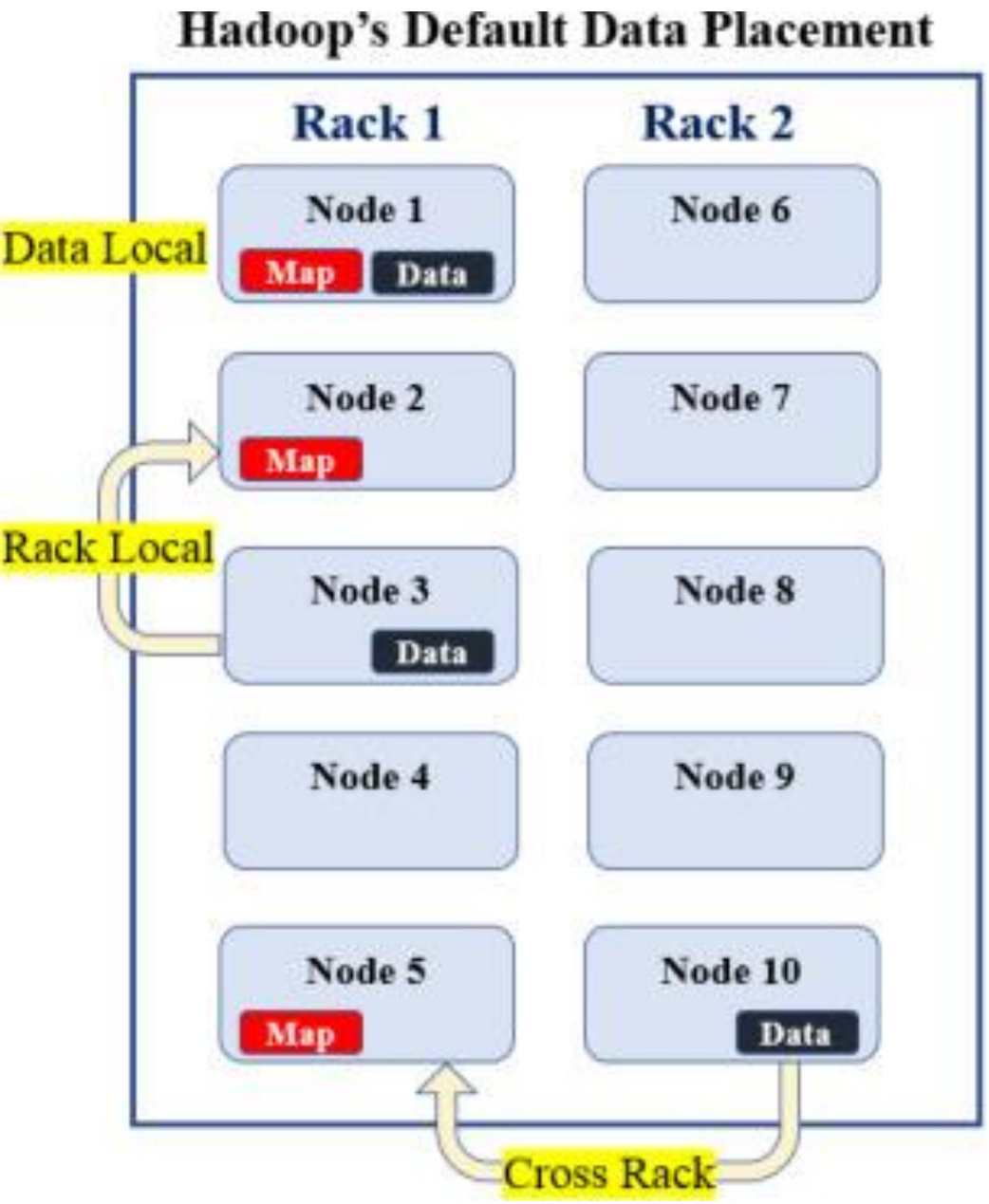
Comparing data and task parallelism

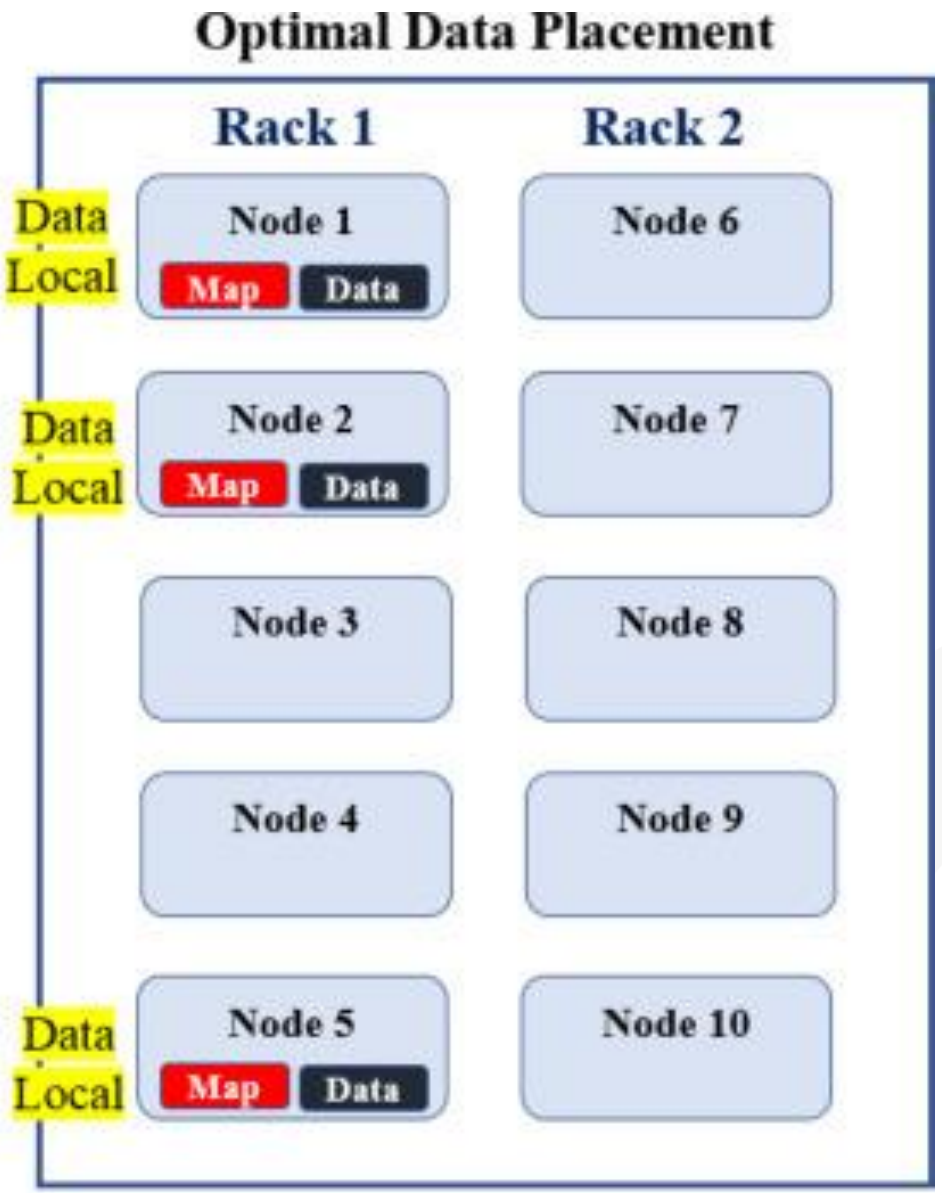
Implementing data parallelism



Bringing computation to the data







Knowledge Check Poll

What's the difference between data parallelism and task parallelism?

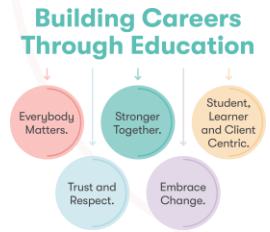
A. Task parallelism partitions data, data parallelism doesn't

B. Both partition data

C. Neither partitions data

D. Data parallelism partitions data, task parallelism doesn't

Knowledge Check Poll



What's the difference between data parallelism and task parallelism?

A. Task parallelism partitions data, data parallelism doesn't

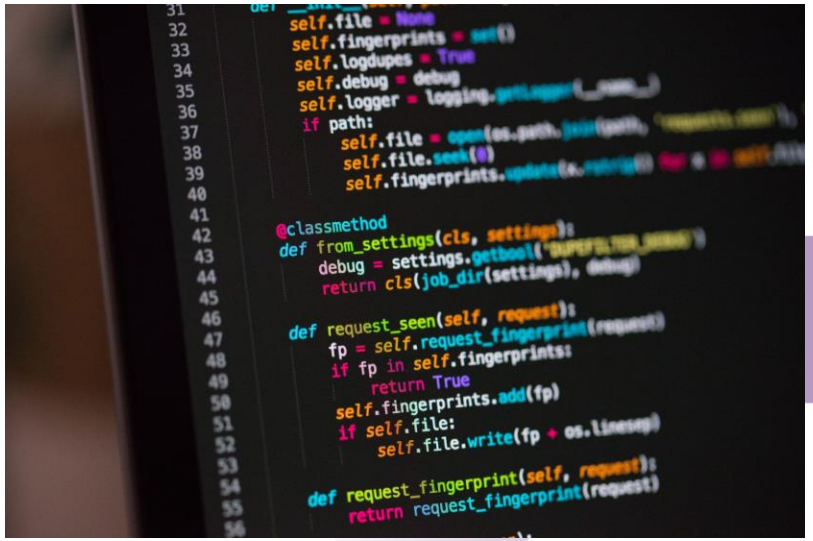
B. Both partition data

C. Neither partitions data

D. Data parallelism partitions data, task parallelism doesn't

Feedback
The correct statement is **D** – Data parallelism partitions data, task parallelism doesn't.

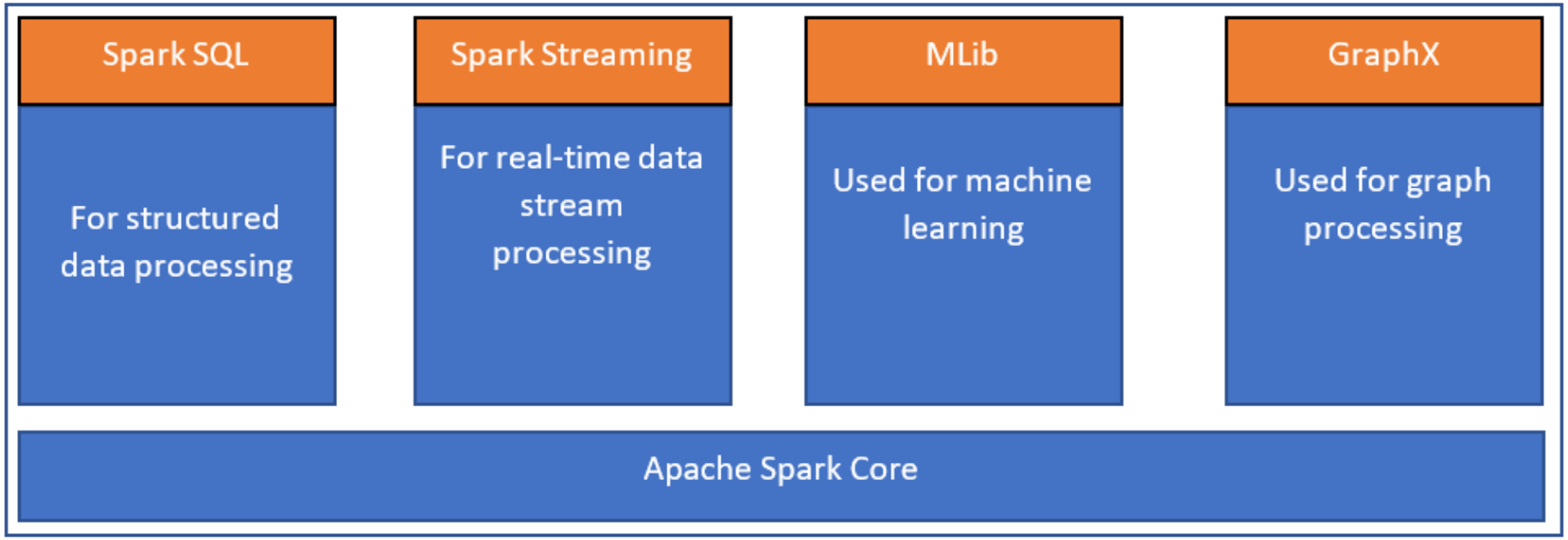
Apache Spark



Introduction

Let's understand more…

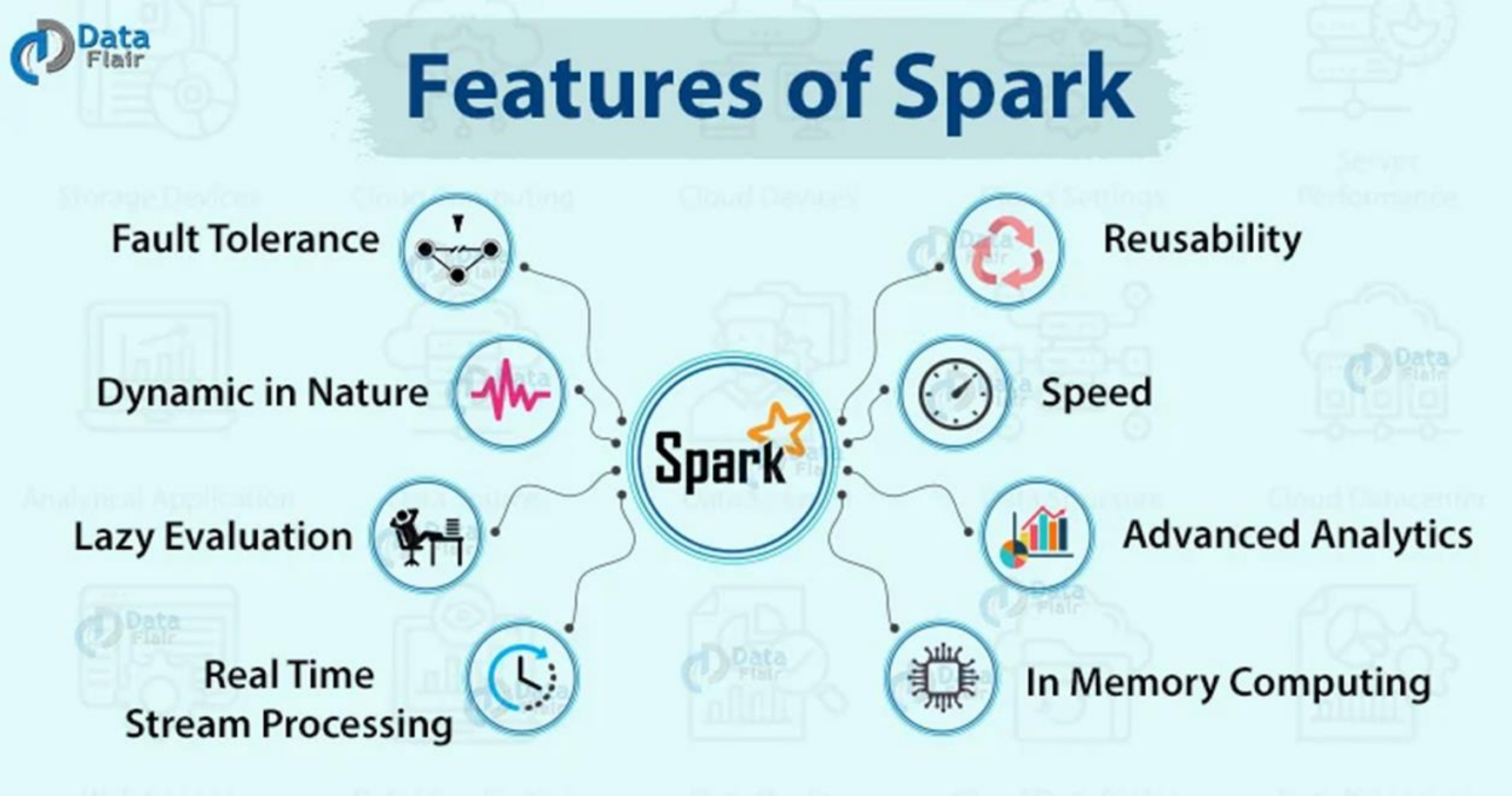
- Apache spark is a fast and general-purpose cluster computing system for large scale data processing
- High-level APIs in Java, Scala, Python and R



The Apache Spark ecosystem

Unpacking Apache Spark

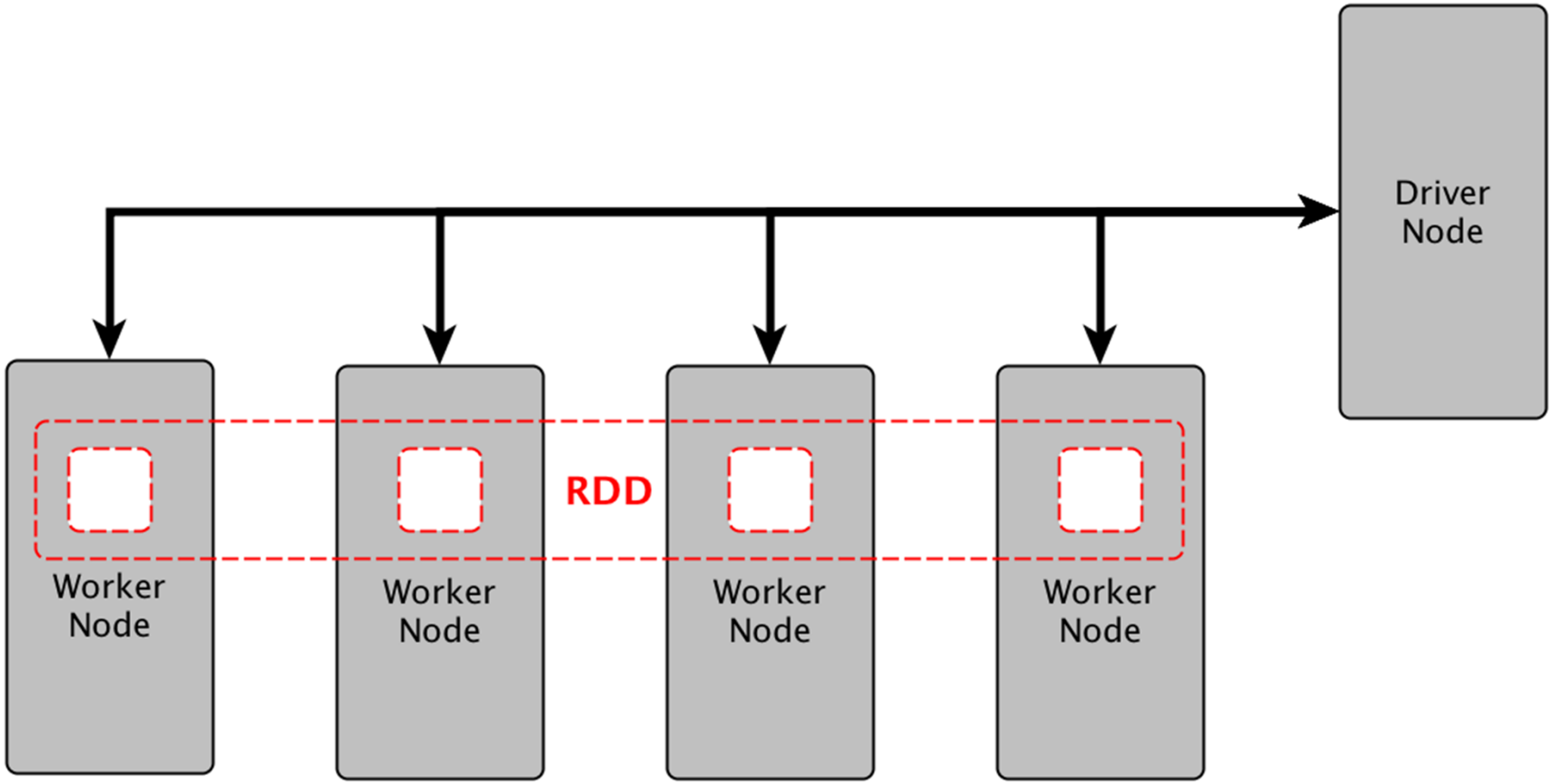
Key features…



Key features of Apache Spark

Key Apache Spark Concept: RDDs

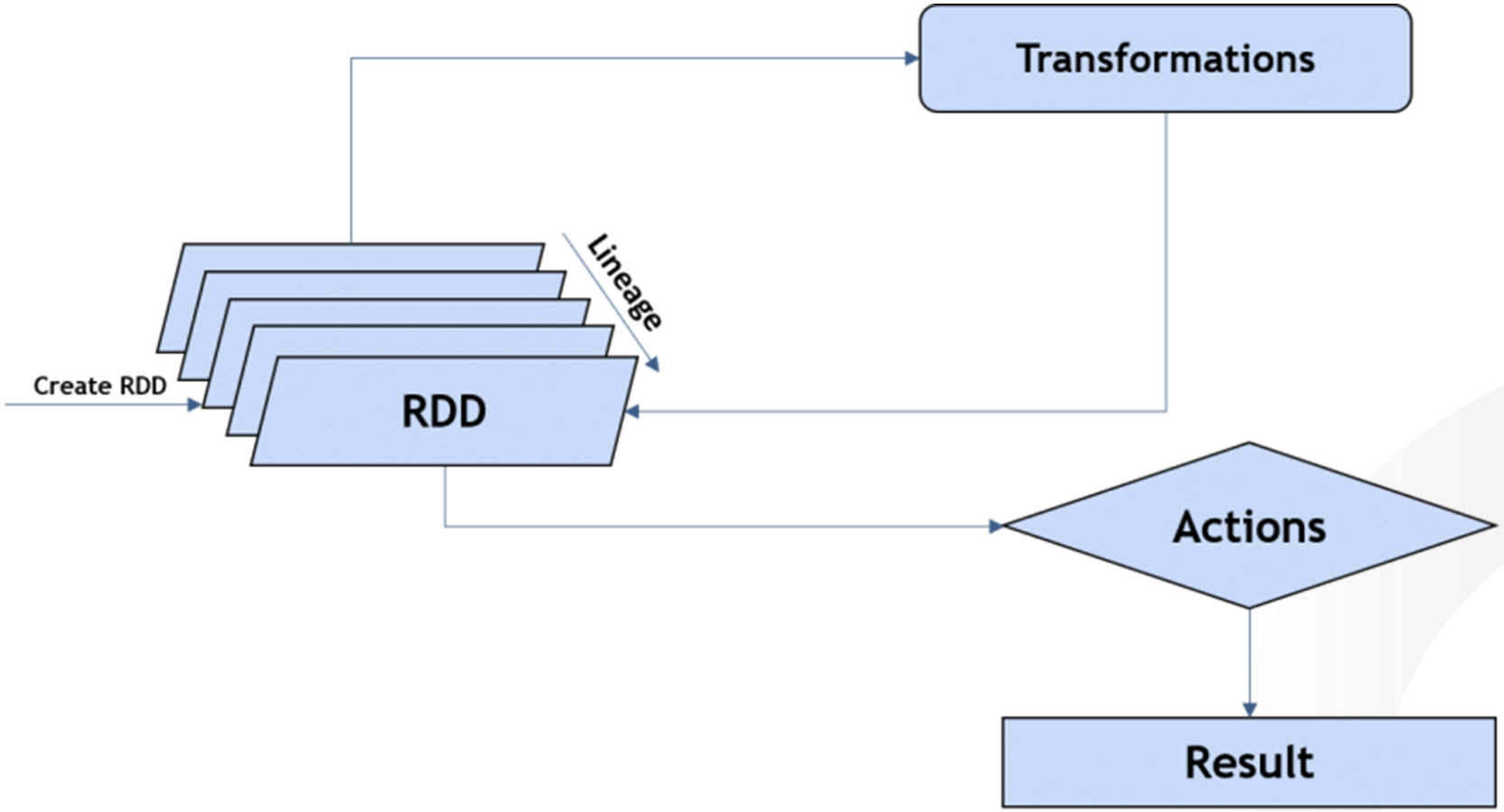
Writing programs in terms of operations on distributed datasets…



The architecture of Resilient Distributed Datasets (RDDs)

Key Apache Spark Concept: RDDs

Writing programs in terms of operations on distributed datasets…



Operations of Resilient Distributed Datasets (RDDs)

Example

Log mining

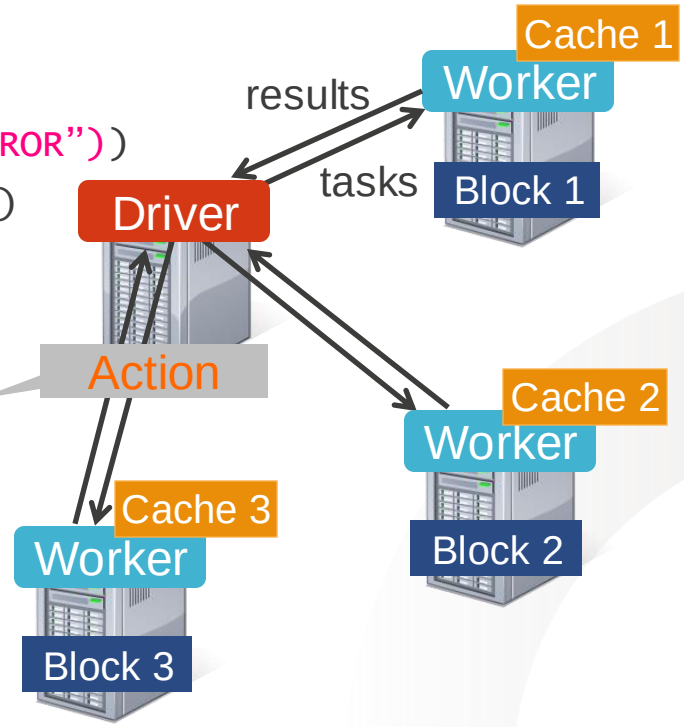
Base RDD

Transformed RDDs

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
```

```
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
. . .
```

Action



Full-text search of Wikipedia

- 60GB on 20 EC2 machine
- 0.5 sec vs. 20s for on-disk

Load error messages from a log into memory, then interactively search for various patterns…

RDDs and fault recovery

- RDDs track *lineage* information that can be used to efficiently re-compute lost data

```
msgs = textFile.filter(lambda s: s.startsWith("ERROR"))
               .map(lambda s: s.split("\t")[2])
```

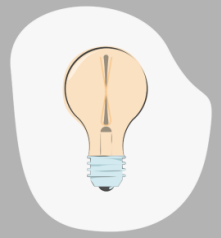
Operations of Resilient Distributed Datasets (RDDs)

BPP

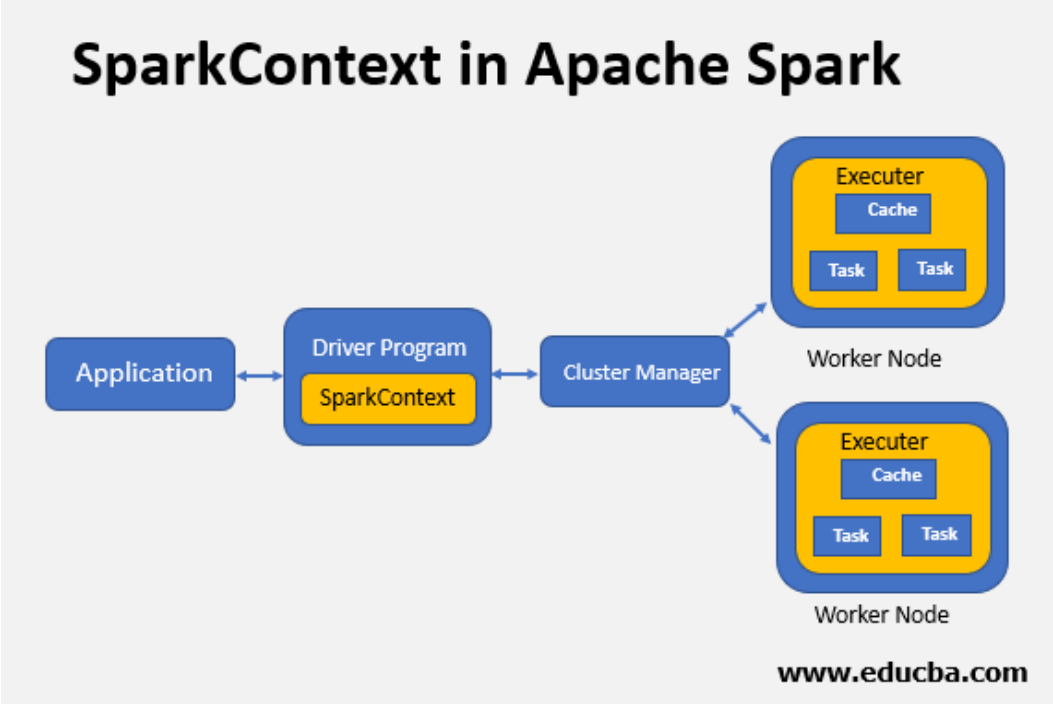
Spark context

What is parallel computing?

- Main entry point to Spark functionality
- Available in shell as variable sc



In standalone programs, you'd make your own (see later for details).

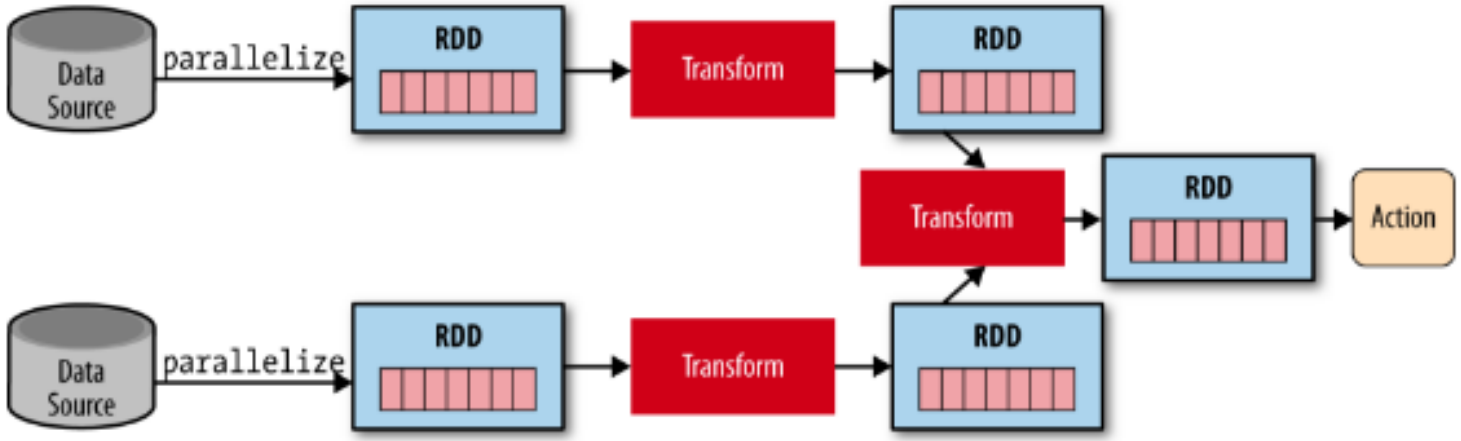


A diagram illustration of SparkContext

Creating RDDs using parallelize

An example

```
# Turn a Python collection into an RDD
> sc.parallelize([1, 2, 3])

# Load text file from local FS, HDFS, or S3
> sc.textFile("file.txt")
> sc.textFile("directory/*.txt")
> sc.textFile("hdfs://namenode:9000/path/file")

# Use existing Hadoop InputFormat (Java/Scala only)
> sc.hadoopFile(keyClass, valClass, inputFmt, conf)
```



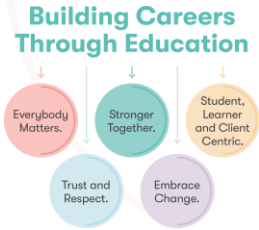
Methods used in Apache Spark for creating Resilient Distributed Datasets (RDDs)

Basic transformations


```
> nums = sc.parallelize([1, 2, 3])

# Pass each element through a function
> squares = nums.map(lambda x: x*x)   // {1, 4, 9}

# Keep elements passing a predicate
> even = squares.filter(lambda x: x % 2 == 0) // {4}

# Map each element to zero or more others
> nums.flatMap(lambda x: => range(x))
    > # => {0, 0, 1, 0, 1, 2}
```

BPP

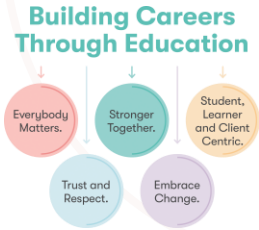
Basic actions




```
nums = sc.parallelize([1, 2, 3])

# Retrieve RDD contents as a local collection
nums.collect() # => [1, 2, 3]
# Return first K elements
nums.take(2) # => [1, 2]
# Count number of elements
nums.count() # => 3
# Merge elements with an associative function
nums.reduce(lambda x, y: x + y) # => 6
# Write elements to a text file
nums.saveAsTextFile("hdfs://file.txt")
```

BPP

Basic key-value operations

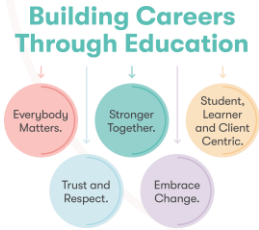



```
> pets = sc.parallelize(
    [("cat", 1), ("dog", 1), ("cat", 2)])
> pets.reduceByKey(lambda x, y: x + y)
                    # => {(cat, 3), (dog, 1)}
> pets.groupByKey() # => {(cat, [1, 2]), (dog, [1])}
> pets.sortByKey()  # => {(cat, 1), (cat, 2), (dog,
  1)}
```

`reduceByKey` also automatically implements combiners on the map side

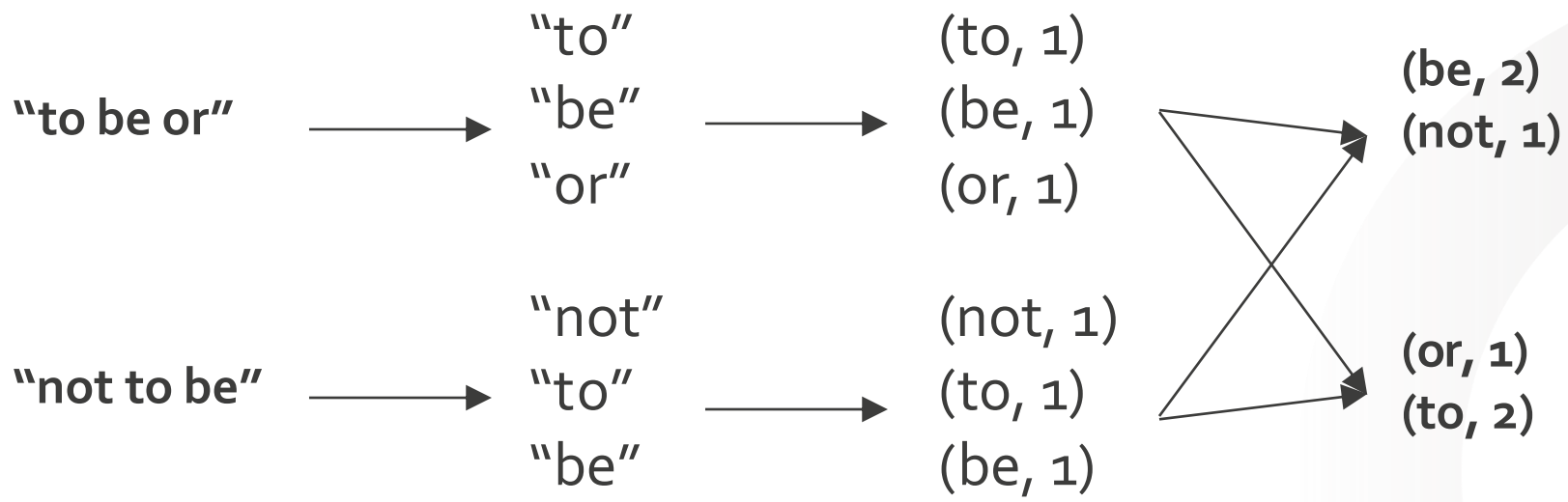
BPP

Basic key-value operations



An example: Wordcount

```
> lines = sc.textFile("hamlet.txt")
> counts = lines.flatMap(lambda line: line.split(" "))
                .map(lambda word => (word, 1))
                .reduceByKey(lambda x, y: x + y)
```



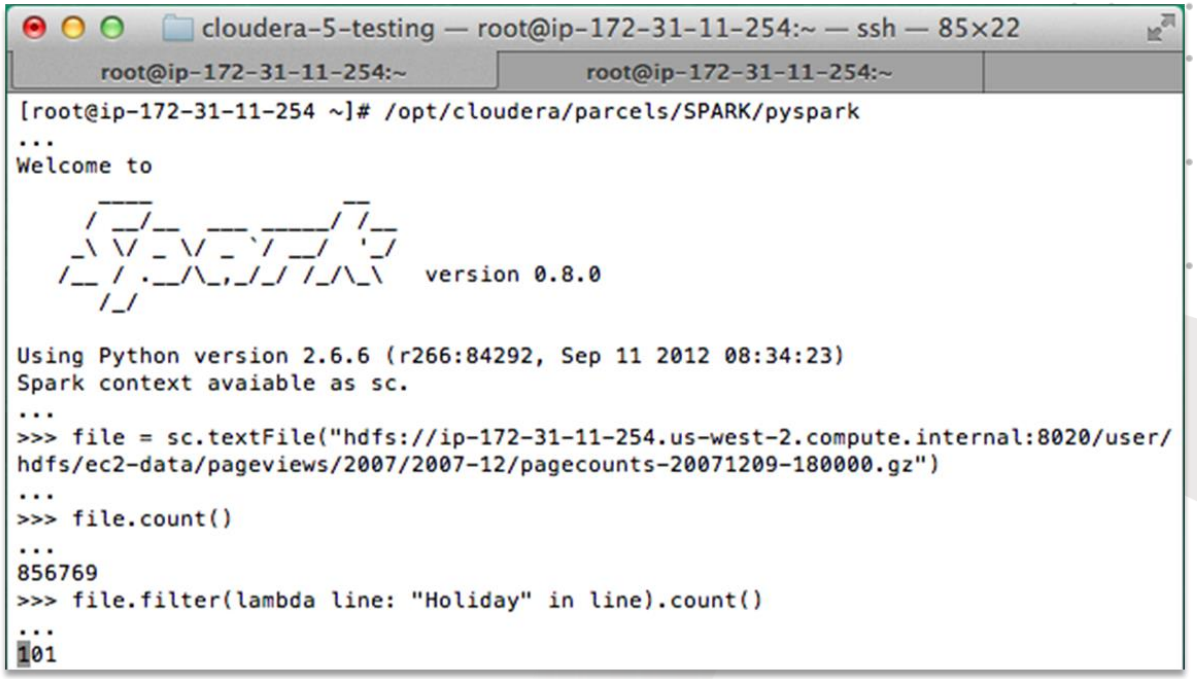
flatmap, map, and reduceByKey

BPP

Interactive shell

Here's what you need to know…

- The Fastest Way to Learn Spark
- Available in Python and Scala
- Runs as an application on an existing Spark Cluster…
- OR Can run locally

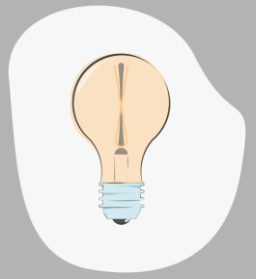


A screen grab for interactive shell

Spark standalone app (Python)

An example

```
import sys
from pyspark import SparkContext

if __name__ == "__main__":
    sc = SparkContext( “local”, “wordCount”, sys.argv[0],
None)
    lines = sc.textFile(sys.argv[1])

    counts = lines.flatMap(lambda s: s.split(“ ”)) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda x, y: x + y)

    counts.saveAsTextFile(sys.argv[2])
```



What can we learn from this code example?

BPP

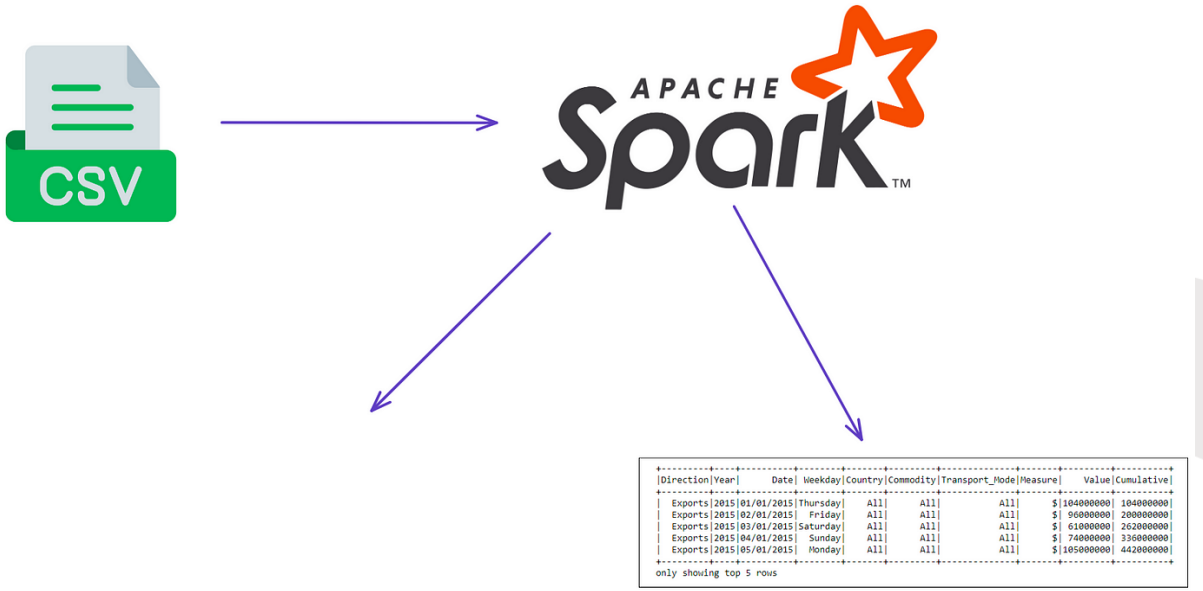
Parsing files

An example

- csv_lines = sc.textFile(“**example.csv**”)
- **def** csv_to_record(line):
  parts = line.split(",")
  record = {
  "**name**": parts[0],
  "**company**": parts[1],
  "**title**": parts[2]
  }
  return record

  # Apply the function to every record
  records = csv_lines.map(csv_to_record)

  # Inspect the first item in the dataset records.first()
- records.first()



An illustration of parsing files in ApacheSpark

What can we learn from this code example?

Practical application

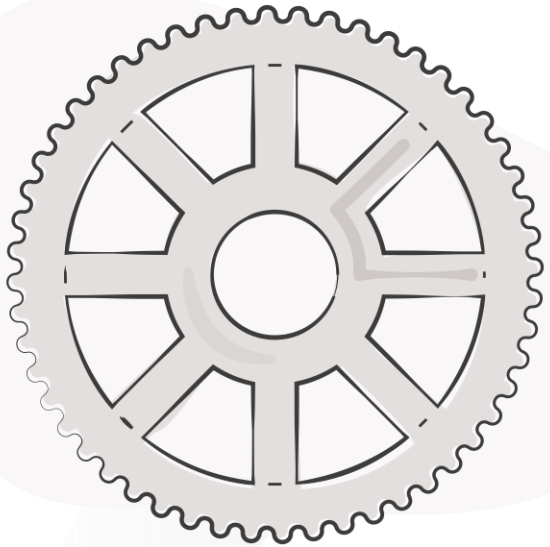
Tutorial walkthrough

For this walkthrough we complete the following steps:

1. Log in to Microsoft Azure (Github Student Pack)
2. Select Azure Databricks
3. Create a new deployment (Select free trial tier)
4. Create a new cluster
5. Create a new notebook
6. Follow the exercises in the worksheet

The link for the associated task briefing:

Briefing link



Practical application

Microsoft Azure

BPP

Thank you

Summary and prep for next week