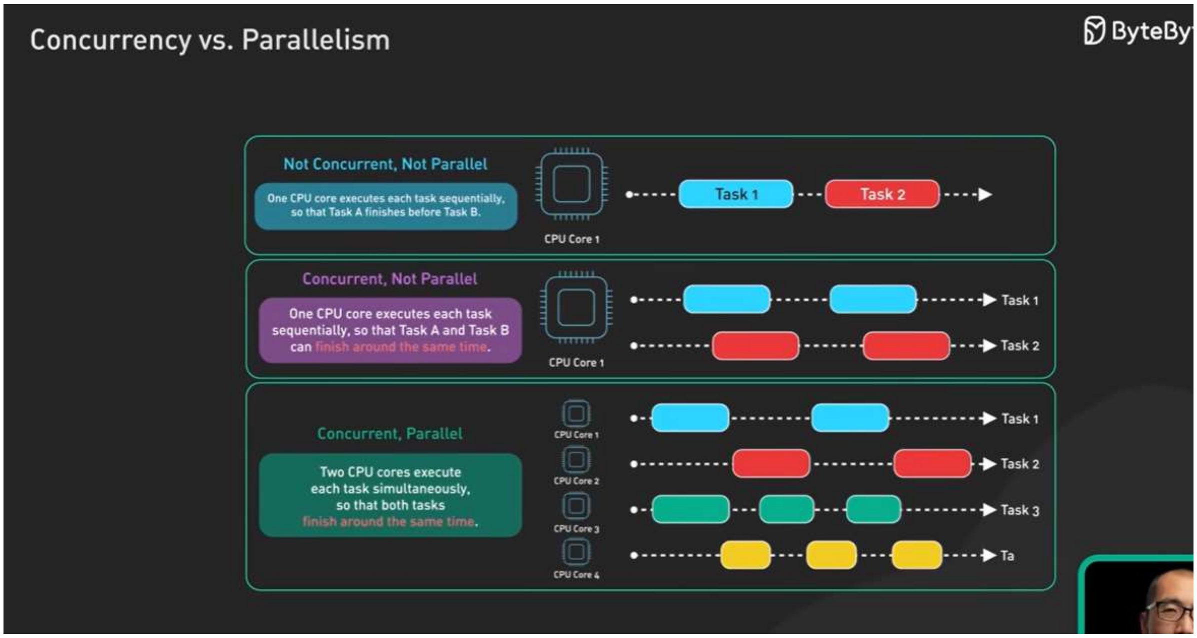
- 

Concurrency vs. Parallelism

Not Concurrent, Not Parallel

One CPU core executes each task sequentially, so that Task A finishes before Task B.

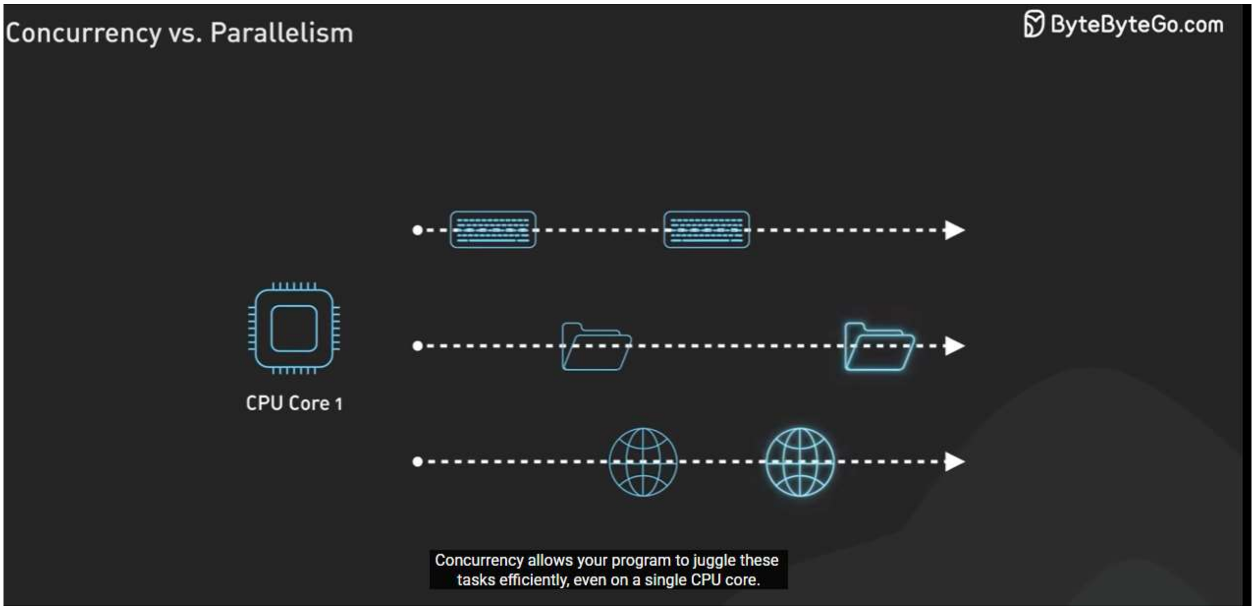
Concurrent, Not Parallel

One CPU core executes each task sequentially, so that Task A and Task B can finish around the same time.

Concurrent, Parallel

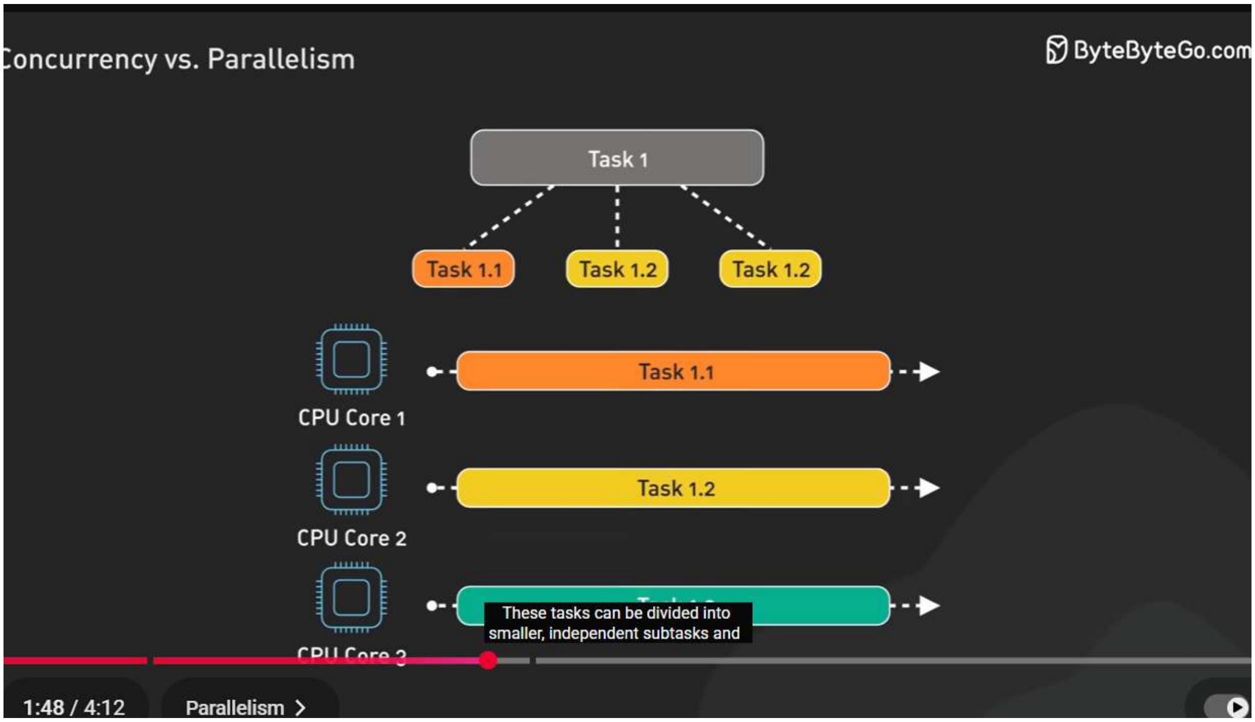
Two CPU cores execute each task simultaneously, so that both tasks finish around the same time.

m



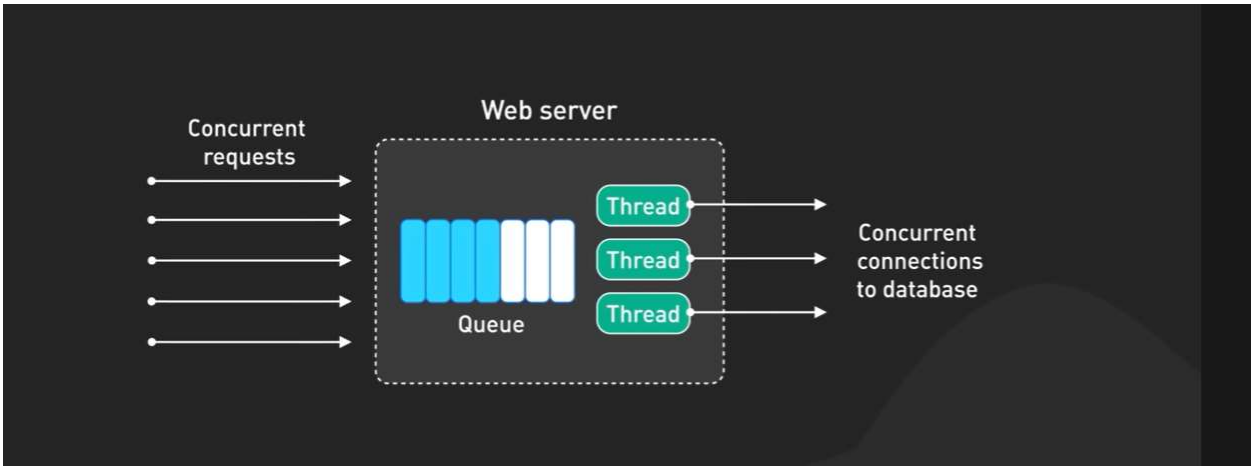
Concurrency vs. Parallelism

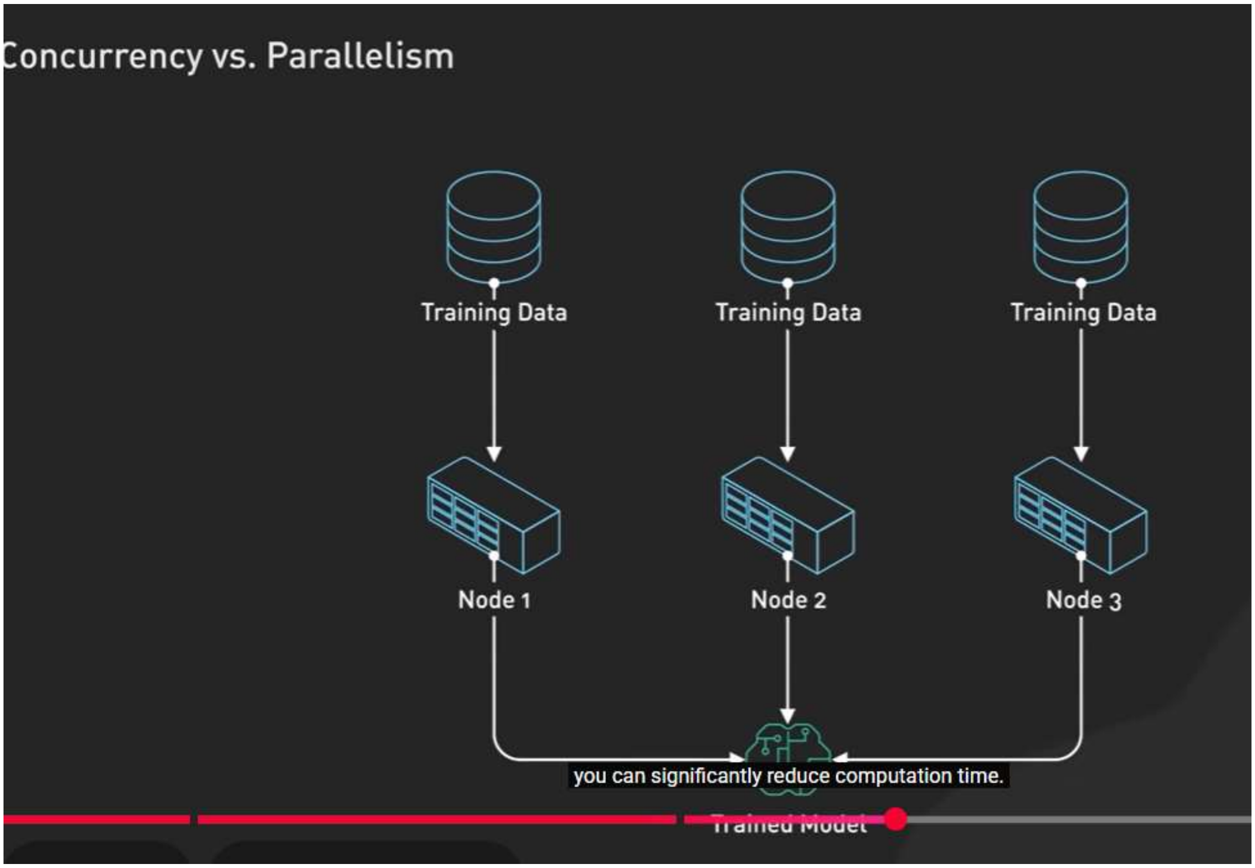
Concurrency allows your program to juggle these tasks efficiently, even on a single CPU core.



These tasks can be divided into smaller, independent subtasks and

m

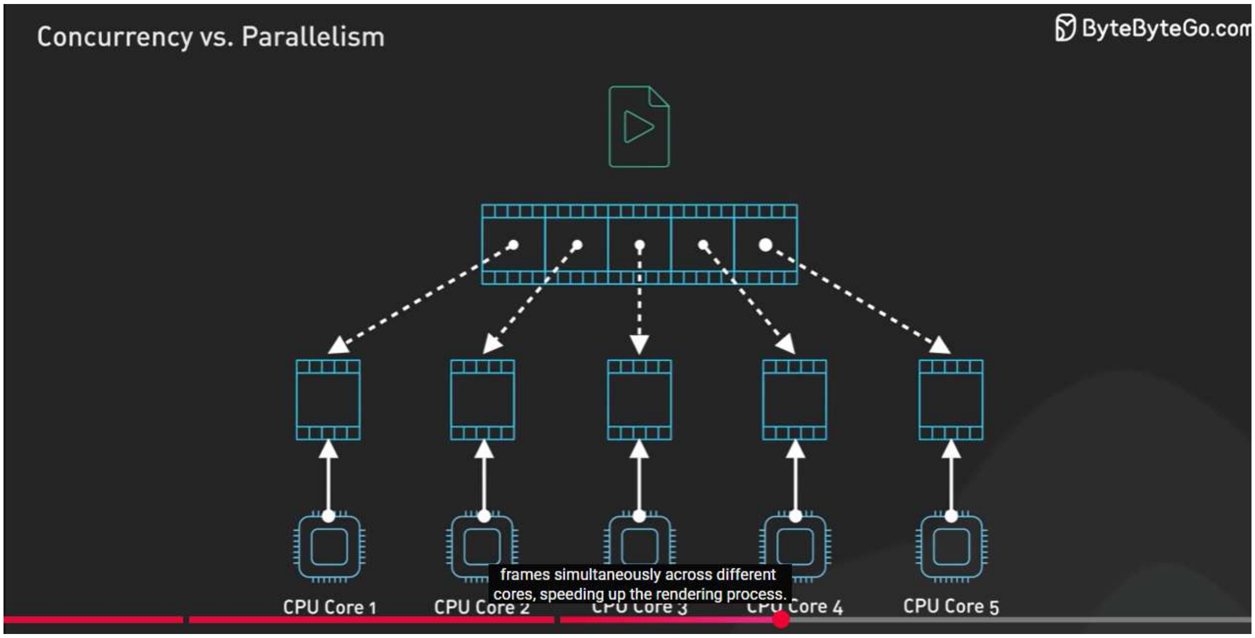




Concurrency vs. Parallelism

you can significantly reduce computation time.

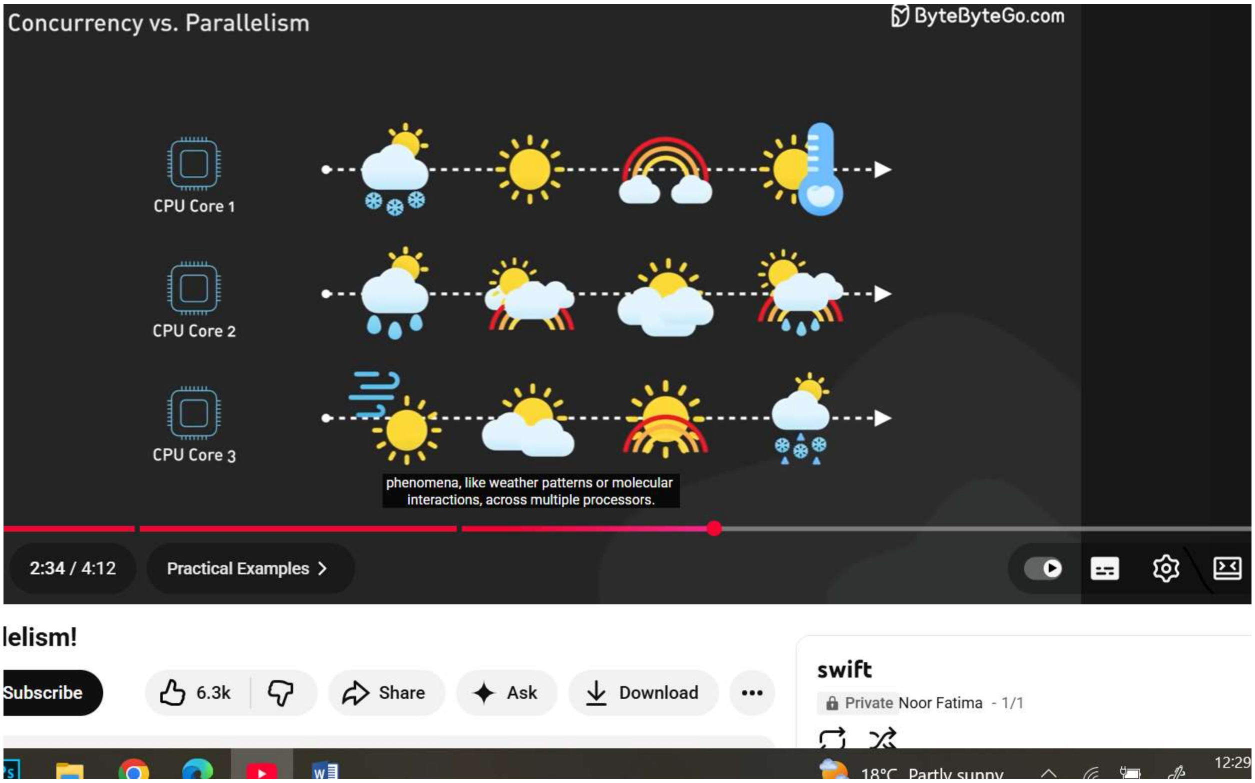
m



Concurrency vs. Parallelism

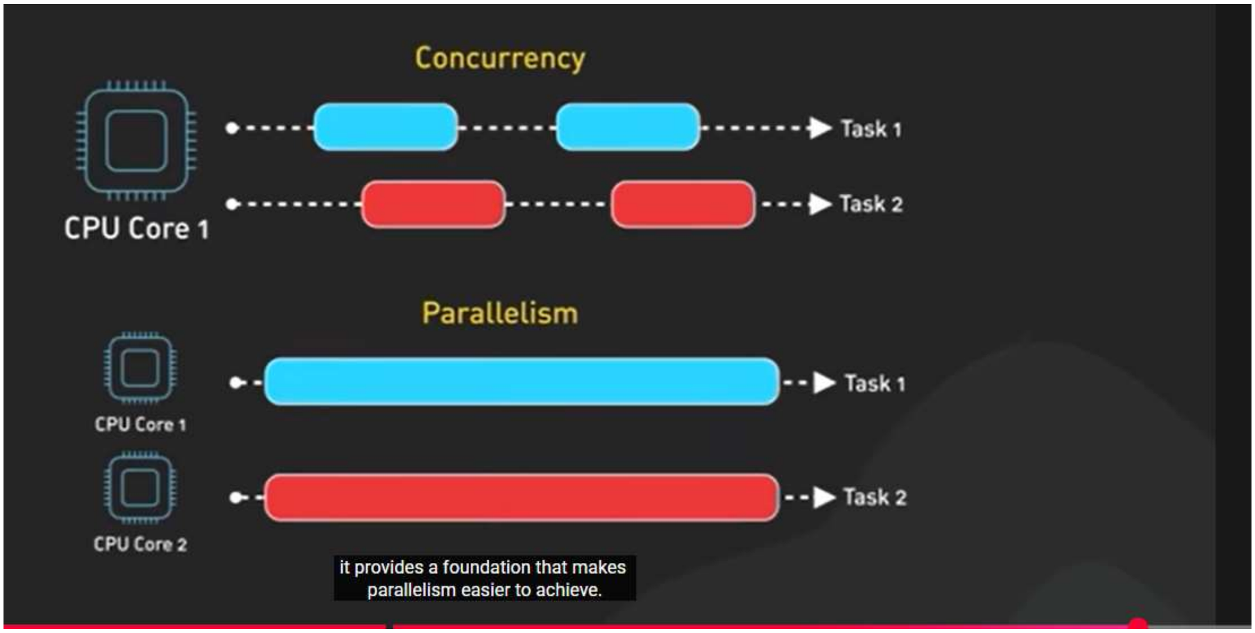
frames simultaneously across different cores, speeding up the rendering process.

Swift for Parallel & Distributed Computing (Full Topic)

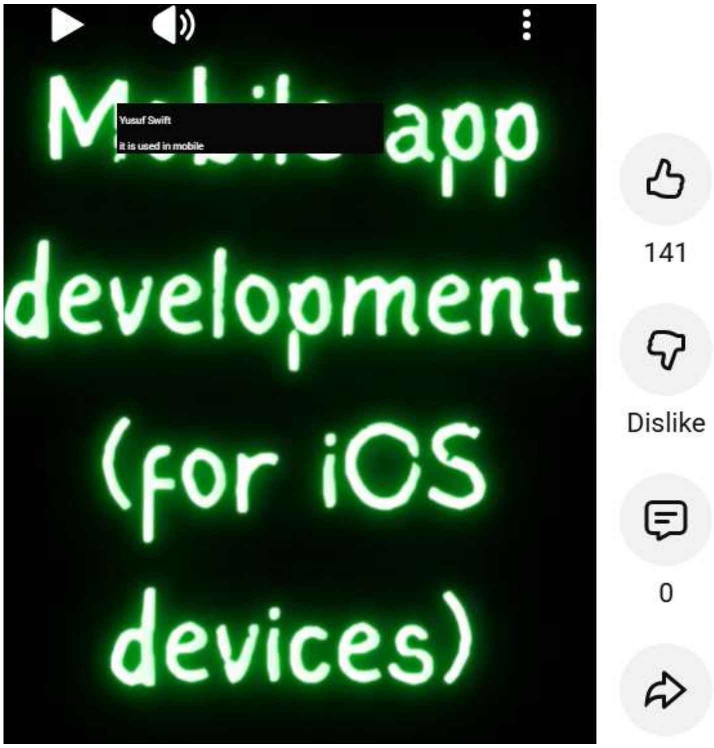


phenomena, like weather patterns or molecular interactions, across multiple processors.

m



it provides a foundation that makes parallelism easier to achieve.

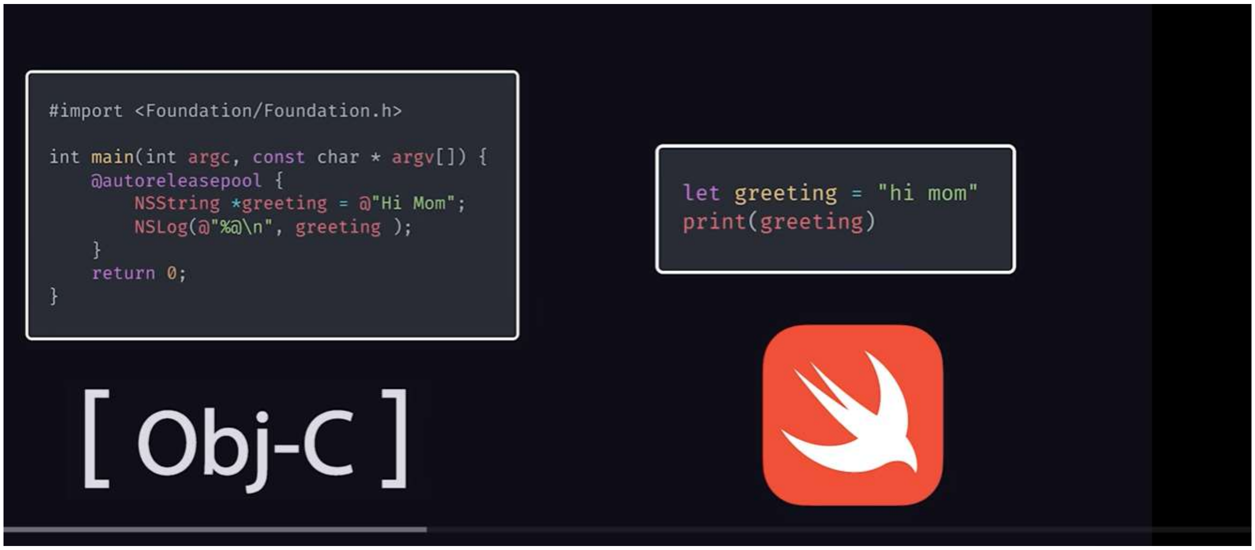


Mobile app development (for iOS devices)

Yusuf Swift

it is used in mobile

m

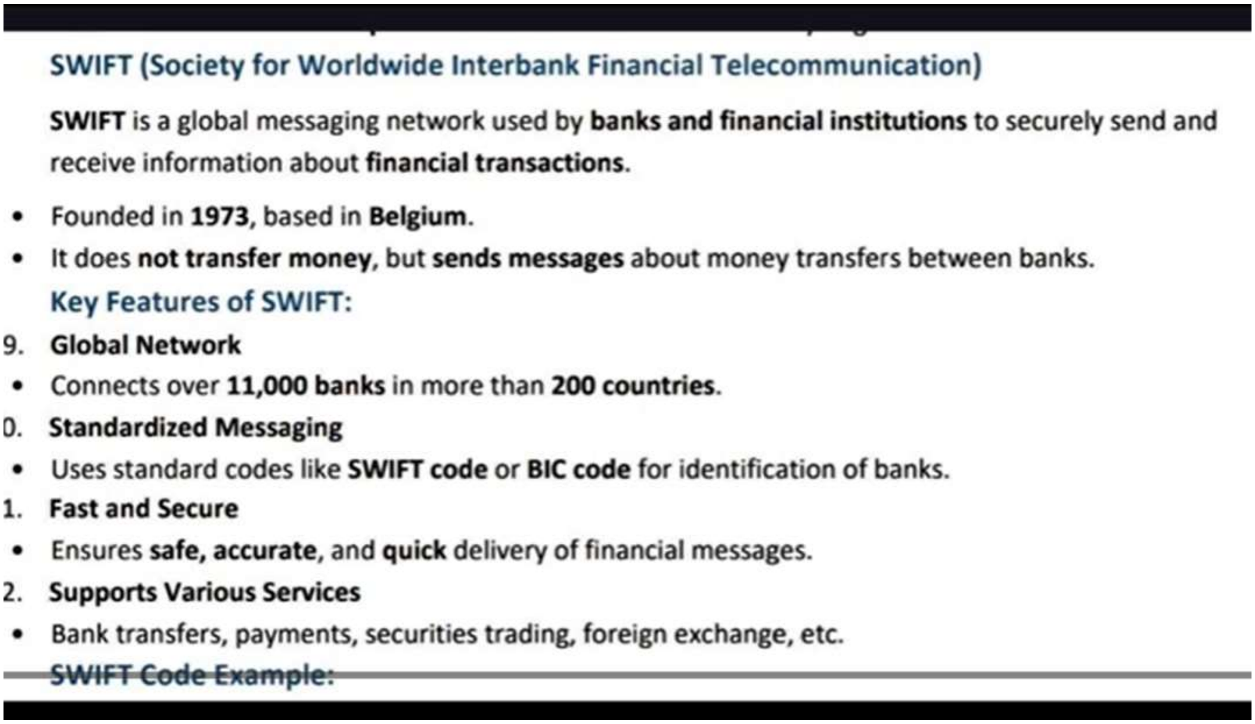


```
#import <Foundation/Foundation.h>

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSString *greeting = @"Hi Mom";
        NSLog(@"%@\n", greeting );
    }
    return 0;
}
```

```
let greeting = "hi mom"
print(greeting)
```

m



SWIFT (Society for Worldwide Interbank Financial Telecommunication)

SWIFT is a global messaging network used by **banks and financial institutions** to securely send and receive information about **financial transactions**.

- Founded in **1973**, based in **Belgium**.
- It does **not transfer money**, but **sends messages** about money transfers between banks.

Key Features of SWIFT:

9. **Global Network**

- Connects over **11,000 banks** in more than **200 countries**.

0. **Standardized Messaging**

- Uses standard codes like **SWIFT code** or **BIC code** for identification of banks.

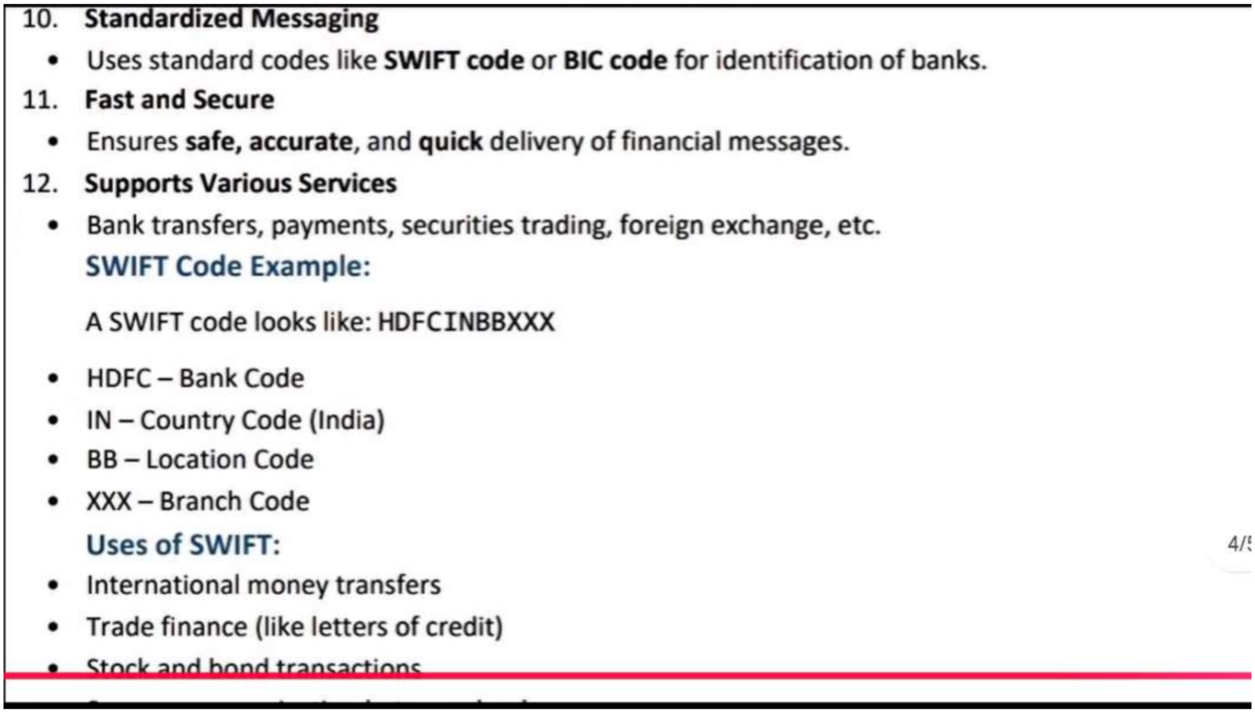
1. **Fast and Secure**

- Ensures **safe, accurate**, and **quick** delivery of financial messages.

2. **Supports Various Services**

- Bank transfers, payments, securities trading, foreign exchange, etc.

SWIFT Code Example:



10. **Standardized Messaging**
- Uses standard codes like **SWIFT code** or **BIC code** for identification of banks.

11. **Fast and Secure**
- Ensures **safe, accurate**, and **quick** delivery of financial messages.

12. **Supports Various Services**
- Bank transfers, payments, securities trading, foreign exchange, etc.

SWIFT Code Example:

A SWIFT code looks like: HDFCINBBXXX

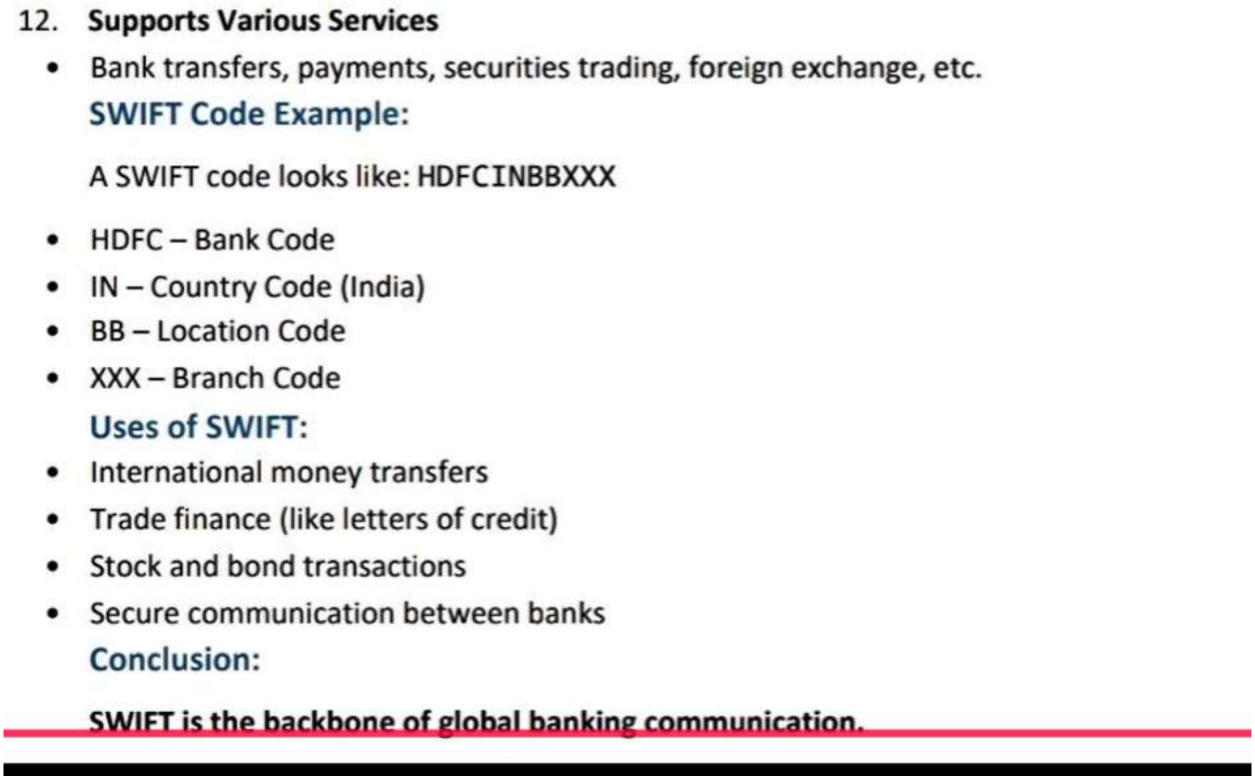
- HDFC – Bank Code
- IN – Country Code (India)
- BB – Location Code
- XXX – Branch Code

Uses of SWIFT:

4/

- International money transfers
- Trade finance (like letters of credit)
- Stock and bond transactions

m



12. **Supports Various Services**
- Bank transfers, payments, securities trading, foreign exchange, etc.

SWIFT Code Example:

A SWIFT code looks like: HDFCINBBXXX

- HDFC – Bank Code
- IN – Country Code (India)
- BB – Location Code
- XXX – Branch Code

Uses of SWIFT:

- International money transfers
- Trade finance (like letters of credit)
- Stock and bond transactions

- Secure communication between banks

Conclusion:

SWIFT is the backbone of global banking communication.

Swift is a modern programming language created by **Apple** mainly for building apps on Apple platforms like **iOS, macOS, watchOS, and tvOS**. It's designed to be powerful, fast, and easy to read. YouTube

📌 What the Video Covers

1. **What Swift Is**
   Swift is Apple’s successor to Objective-C — created to make development safer and more productive. It emphasizes simplicity, readability, and modern features. Outlecture
2. **How It Compares to Objective-C**
   Swift improves on older languages used for Apple development (like Objective-C) by having cleaner syntax, better safety, and modern language features. Outlecture
3. **Key Features of Swift**
   - **Modern & expressive syntax** that’s easier to learn. Outlecture
   - **Memory safety features** that help prevent bugs and crashes. Outlecture
   - Swift is also **open-source**, so it’s not limited strictly to Apple platforms — the language itself can be used outside Apple’s ecosystem. YouTube
4. **Typical Use Cases**
   - Building apps for **iPhones, iPads, Macs, Apple Watches, and Apple TVs** using tools like **Xcode** and frameworks like **SwiftUI** or **UIKit**. Outlecture
5. **Why Swift Is Popular**
   - It’s modern and approachable for new programmers. Outlecture
   - Supported directly by Apple with frequent updates and improvements.

1. What is Parallel & Distributed Computing?

Parallel Computing

- Multiple tasks run **at the same time**
- Usually on **one machine**
- Uses **multiple CPU cores / threads**

Example:

- Image processing
- Matrix multiplication
- Data analytics

Distributed Computing

- Tasks run on **multiple machines**
- Machines communicate over **network**
- Each machine has its own memory

Example:

- Microservices
- Cloud-based systems
- Online games

- Distributed databases

---

2. Why Swift for Parallel & Distributed Computing?

Swift is powerful because:

- Modern **concurrency model**
- Safe by design (avoids race conditions)
- High performance (compiled language)
- Native support for:
  - `async/await`
  - `Task`
  - `Actors`
  - `Distributed Actors` (Swift 5.7+)

Apple designed Swift concurrency to be:

- **Structured**
- **Predictable**
- **Thread-safe**

---

3. Swift Concurrency Model (Core Concepts)

Swift supports **structured concurrency**, not raw thread management.

Key Components

Concept	Purpose
async / await	Asynchronous programming
Task	Unit of concurrent work
TaskGroup	Parallel execution
Actor	Thread-safe state
Distributed Actor	State across machines

4. async / await (Foundation)

Problem Without async/await

Callbacks are messy and hard to read.

Solution

Swift uses `async` and `await` for clean concurrency.

Example

```
func fetchData() async -> String {
    return "Data loaded"
}

Task {
    let result = await fetchData()
    print(result)
}
```

Explanation

- `async` → function runs asynchronously
- `await` → waits without blocking the thread
- `Task` → runs concurrent work

5. Parallel Computing in Swift

Running Tasks in Parallel

Example: Two tasks at the same time

```
func task1() async -> Int {
    sleep(2)
    return 10
}

func task2() async -> Int {
    sleep(2)
    return 20
}

Task {
    async let a = task1()
    async let b = task2()

    let result = await a + b
    print(result)
}
```

What’s happening?

- `task1` and `task2` run **in parallel**
- Total time ≈ 2 seconds (not 4)
- CPU cores are used efficiently

6. TaskGroup (True Parallelism)

Used when:

- Number of tasks is dynamic
- You want maximum parallelism

Example: Parallel Sum

```
func parallelSum(_ numbers: [Int]) async -> Int {
    await withTaskGroup(of: Int.self) { group in
        for num in numbers {
            group.addTask {
                num * num
            }
        }

        var total = 0
        for await value in group {
            total += value
        }
        return total
    }
}
```

Why TaskGroup?

- Automatic task scheduling
- Error handling
- Structured lifecycle

---

7. Actors (Thread Safety)

Problem: Shared Memory

Multiple threads modifying same variable → **race condition**

Solution: Actors

Actors ensure **only one task accesses state at a time**.

Example

```
actor Counter {
    private var value = 0
    func increment() {
        value += 1
    }

    func getValue() -> Int {
        value
    }
}
```

Usage

```
 let counter = Counter()


Task {
    await counter.increment()
    print(await counter.getValue())
}
```

Key Rule

- Actor properties are **isolated**
- Must use `await` to access them

8. Swift Distributed Computing

What is Distributed Computing in Swift?

Swift introduces **Distributed Actors**, which:

- Work across **multiple machines**
- Communicate using **network**
- Hide networking complexity

9. Distributed Actors (Core Feature)

Definition

A distributed actor behaves like a local actor but runs on **remote nodes**.

Basic Syntax

```
distributed actor ChatServer {
    distributed func sendMessage(_ msg: String) {
        print("Message:", msg)
    }
}
```

Calling a Distributed Method

```
 let server = try await ChatServer.resolve(id: id, using: system)
```

```
try await server.sendMessage("Hello from client")
```

Behind the Scenes

- Message is serialized
- Sent over network
- Executed remotely

10. Distributed Actor System

You need:

- Transport layer (HTTP, gRPC, TCP)
- Actor system implementation

Swift provides:

- `DistributedActorSystem` protocol
- **Architecture**
- `Client → Network → Distributed Actor → Response`

11. Parallel vs Distributed (Swift Comparison)

Feature	Parallel	Distributed
Location	Same machine	Multiple machines
Memory	Shared	Separate
Latency	Low	High
Example	Image processing	Microservices

12. Numerical Example (Exam-Friendly)

Parallel Matrix Addition

```
func addMatrices(_ A: [[Int]], _ B: [[Int]]) async -> [[Int]] {
    let rows = A.count
    let cols = A[0].count
    var result = Array(repeating: Array(repeating: 0, count: cols), count:
rows)

    await withTaskGroup(of: (Int, Int, Int).self) { group in
        for i in 0..<rows {
            for j in 0..<cols {
                group.addTask {
                    (i, j, A[i][j] + B[i][j])
                }
            }
        }

        for await (i, j, value) in group {
            result[i][j] = value
        }
    }
    return result
}
```

Why Parallel?

- Each cell computed independently
- Runs on multiple cores

13. Real-World Use Cases

Parallel

- Image/video processing
- Machine learning pipelines
- Data analytics

Distributed

- Cloud services
- Chat applications
- Online booking systems
- Microservice architectures

14. Advantages of Swift Concurrency

- Memory safety
- No manual thread handling
- Compile-time checks
- High performance
- Scales from mobile to cloud

---

15. Summary (Exam Ready)

Keywords to Remember

- async / await
- Task

- TaskGroup
- Actor
- Distributed Actor
- Structured Concurrency

One-Line Definition

Swift enables parallel and distributed computing through structured concurrency, actors for thread safety, and distributed actors for network-based execution.

Simple Definition

Swift is a compiled, general-purpose programming language designed for performance, safety, and modern software development.

Who Developed Swift?

- Developed by **Apple Inc.**
- First released in **2014**
- Designed to replace **Objective-C**

Where is Swift Used?

Swift is mainly used for:

- **iOS applications**
- **macOS applications**
- **watchOS & tvOS apps**
- **Server-side applications**
- **Parallel & distributed systems**
- **High-performance computing**

Key Features of Swift

1. Fast Performance

- Compiled language

- Uses **LLVM compiler**
- Comparable to **C/C++ performance**

---

2. Safe by Design

- Prevents common errors
- No null pointer crashes (uses `Optional`)
- Strong type system

Example:

```
var name: String? = nil
print(name ?? "No name")
```

3. Modern Syntax

- Easy to read
- Less code, more clarity

```
let a = 10
let b = 20
print(a + b)
```

4. Supports Concurrency

- Built-in support for **parallel and distributed computing**
- Uses:
    - `async / await`
    - `Task`
    - `Actor`
    - `Distributed Actor`

5. Open Source

- Available on:
    - macOS
    - Linux
    - Windows (limited)
- Used in cloud and backend systems

Swift vs Other Languages

Feature	Swift	Java	C++
Speed	Very fast	Medium	Very fast
Safety	High	Medium	Low
Memory Management	Automatic	Automatic	Manual
Concurrency	Modern	Thread-based	Manual

Why Swift is Important?

- Reduces runtime crashes
- Improves developer productivity
- Ideal for **parallel & distributed systems**
- Used by Apple ecosystem and cloud services

One-Line Exam Answer

Swift is a modern, compiled, high-performance programming language developed by Apple for safe and efficient application development

No — **Swift tool nahi hai.**

Clear Answer:

Swift ek programming language hai.

Difference: Tool vs Language

◆ Tool

- Software jo kisi kaam ko perform kare
- Example:
    - Git
    - Docker
    - Postman
    - Xcode (IDE)

◆ Programming Language

- Jis se hum programs likhte hain
- Example:
  - Swift
  - C
  - Java
  - Python

Swift Kya Hai?

- Apple ki banayi hui **programming language**
- iOS, macOS, server-side, parallel & distributed systems ke liye use hoti hai

---

~~Swift Related Tools (Confusion Yahin Hoti Hai)~~

~~Name~~	~~Kya Hai~~
~~Swift~~	~~Programming Language~~
~~Xcode~~	~~IDE (tool)~~
~~Swift Compiler~~	~~Tool jo Swift code compile karta hai~~
~~Swift Package Manager~~	~~Dependency management tool~~

~~Exam-Friendly Line~~

~~**Swift ek programming language hai, tool nahi.**~~

~~Agar chaho to main:~~

- ~~**Swift vs OpenMP**~~
- ~~**Swift compiler explain**~~
- ~~**Swift ecosystem diagram**~~

~~bhi explain kar sakta hoon.~~