HIGH PERFORMANCE COMPUTING WITH

SWIFT

```swift
func magicEightBall() {
    let randomNum = Int.random(in: 
    switch randomNum {
    case 1:
        print("No.")
    case 2:
        print("Maybe.")
    case 3:
        print("Try asking again")
    case 4:
        print("Of course")
    default:
        print("Nah.")
    }
}
magicEightBall()
magicEightBall()
magicEightBall()
magicEightBall()
```

APPLE'S OPEN SOURCE

PROGRAMMING LANGUAGE

HISTORY ABOUT SWIFT

- Started development in 2010 by Chris Lattner (Apple Software Engineer known for LLVM and Clang).
- Version 1.0 released in 2014 with the Objective-C runtime library, allowing C, Objective-C, C++, and Swift code to run within one program.
- Compiler optimized for performance and language optimized for development, with a focus on error handling and fixing common programming errors.

CAN SWIFT BE USED FOR HPC?

- Features such as GCD (Grand Central Dispatch), LLVM, and Clang have allowed for new uses of Swift, including HPC.
- Although OpenMP and MPI bindings are not available yet for Swift, Swift is “parallel” similar to how C and C++ are without special tools.
- Solution to this? Wrapping! Swift allows you to create C++ wrappers which you can use to parallelized and optimized parts of your code for implementing in iOS, watchOS, and macOS applications easily.

FUNCTION WRAPPING

- (My_Header.h) Create a header file for your function with C++

```cpp
// My_Header.h
// Class that stores an Int and has a get function
class A {
    public:
    A(int);
    int getInt(); private:
    int m_Int;
};

```

- (My_Implementation.cpp) Create the implementation file

```cpp
#include "My_Header.h"

A::A(int _i) : m_Int(_i) {}

int A::getInt() {
    return m_Int;
}

```

FUNCTION WRAPPING

- Create the executable and library:
  - g++ -c My_Implementation.cpp
  - ar r libmycode.a My_Implementation.o
- Add the “libmycode.a”, “My_Header.h”, and “My_Implementation.cpp” to Xcode
- (Wrapper.cpp) Make a wrapper file in C++

```
#include <stdio.h>
#include "My_Header.h"
// extern "C" will cause the C++ compiler
// (remember, this is s.ll C++ code!) to
// compile the func.on in such a way that // it can be called from C
// (and Swi$).
extern "C" int getIntFromCPP(int myNumber)
{
// Create an instance of A, defined in // the library, and call getInt() on it:
    return A(myNumber).getInt();
}

```

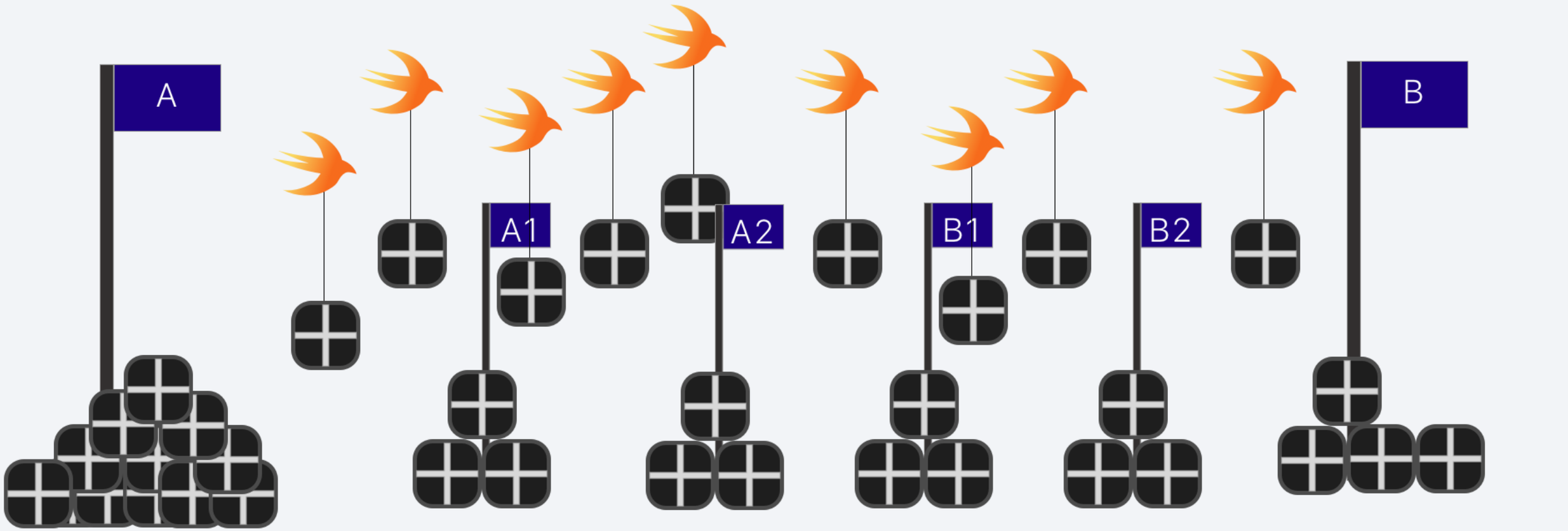
FUNCTION WRAPPING

- Now in the Swift Implementation, we can include the getIntFromCPP function, which will allow you to obtain a value using a C++ function in a Swift application. (Binding file required)

```
9  import Foundation
10
11 // Using Wrappers in Swift
12     print("Hello, World!")
13     func returnMyNumber() -> Int32 {
14       return 314159
15     }
16
17     var theNumber = returnMyNumber()
18     print("The value of PI without a decimal is \(getIntFromCPP(Int32(theNumber)))")
19
```

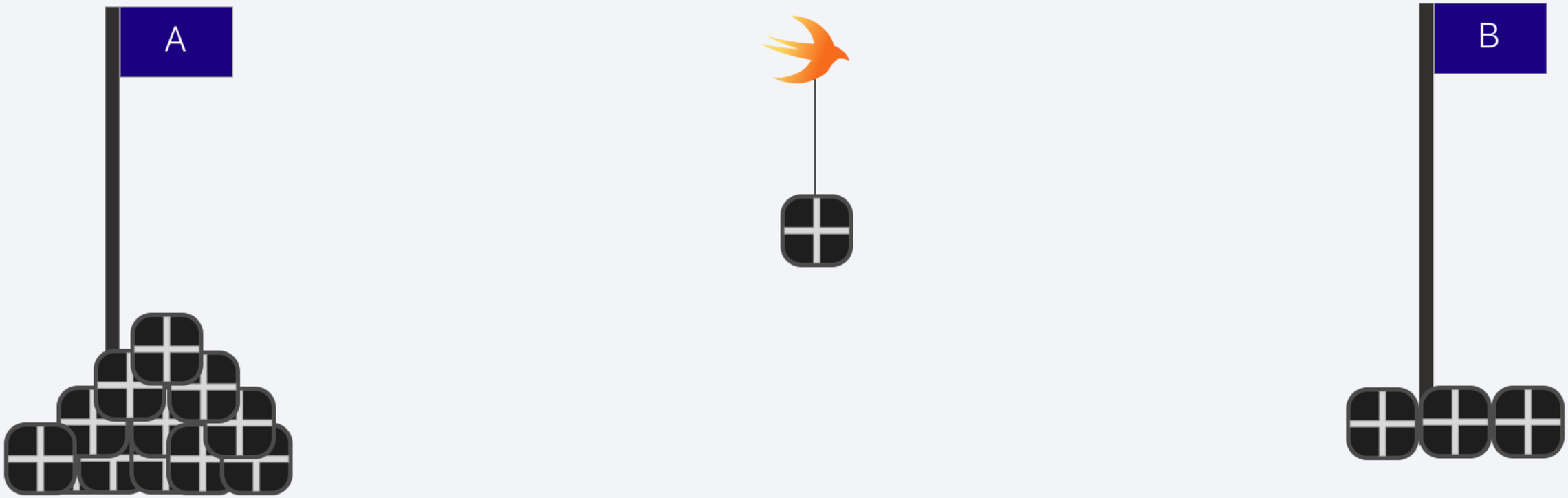
```
Hello, World!
The value of PI without a decimal is 314159
Program ended with exit code: 0
```

- Wanna try it out? Follow this guide: http://macadmins.psu.edu/wp-content/uploads/sites/24696/2016/06/psumac2016-17-swift_and_HPC.pdf

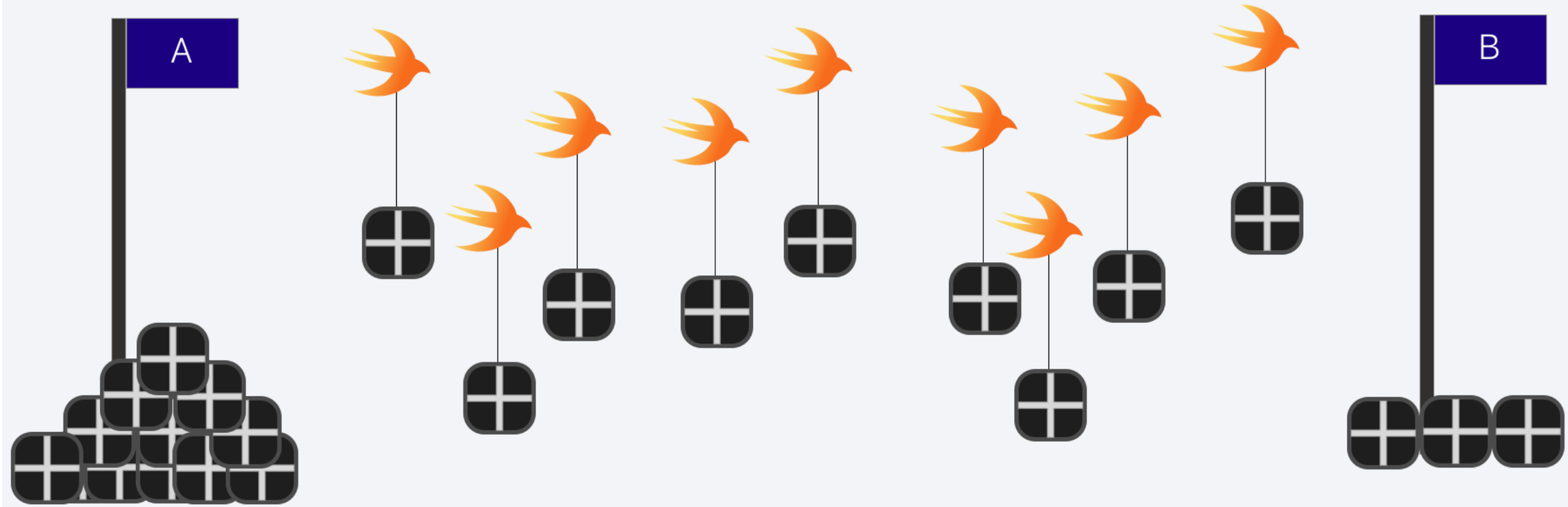


HOW TO DO MULTIPLE THINGS AT ONCE?

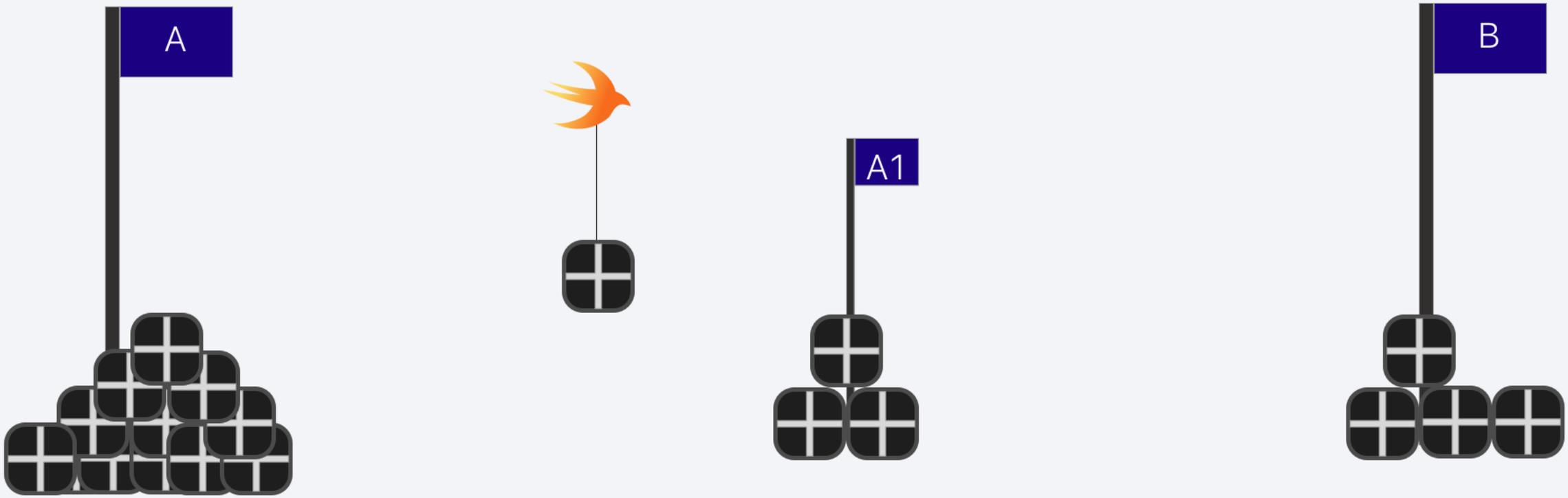
CONCURRENCY AND THREADS



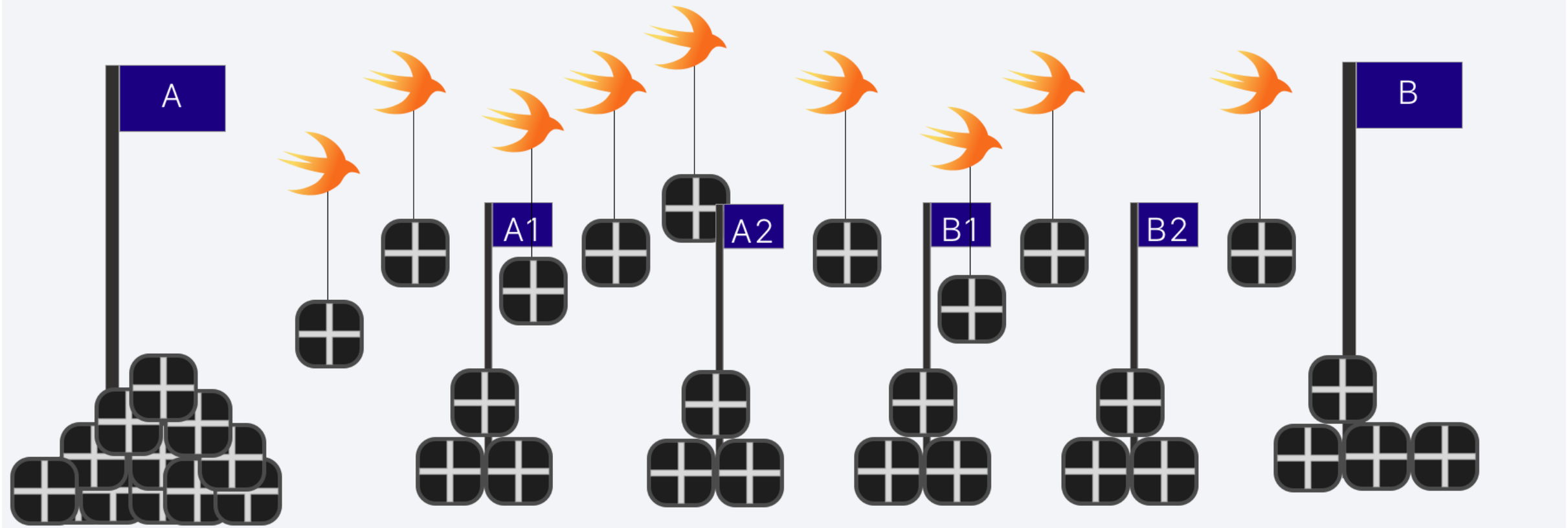
A WORKER WANTS TO TO MOVE SOMETHING FROM POSITION A TO POSITION B, WHAT'S FASTER?



IF WE HAVE MULTIPLE WORKERS, IT WOULD BE FASTER. BUT WHAT IF WE HAD JUST ONE?



THIS IS WHAT CONCURRENCY IS ALL ABOUT, ALLOWING YOU TO DIVIDE WORK INTO SMALLER TASKS



AND IF YOU HAVE MULTIPLE CORES, WE CAN MAKE IT EVEN BETTER!

THREADS

- Threads allow hardware to process different things during sequential timelines.

```
// Using Threads
    class CustomThread: Thread {
      override func main() {
        print("Starting new thread.")
      }
    }

    let customThread = CustomThread()
    customThread.start()
    print("Thread is done")
```

- Can problems occur with this implementation?
- Copy Paste the class available on github.com/BUHPC
- Try this at http://online.swiftplayground.run

DISPATCH QUEUES

- Important topic that will build skills in system resource optimization.

“Since we only have one process and threads are limited to 64, there have to be other options to run code concurrent. Apple’s solution is dispatch queues. You can add tasks to a dispatch queue and expect it to be executed at some point. There are different types of dispatch queues. One is the SerialQueue. In this type everything will be processed in the same order as it was added to the queue. The other is the ConcurrentQueue. As the name suggests, tasks can be executed concurrently within this queue.

This isn’t really concurrent yet, right? Especially when looking into SerialQueues, we didn’t win anything. And ConcurrentQueues don’t make anything easier. We do have threads, so what’s the point?

Let’s consider what happens if we have multiple queues. We could just run a queue on a thread and then whenever we schedule a task, add this into one of the queues. Adding some brain power, we could even distribute the incoming tasks for priority and current workload, thus optimizing our system resources.”

- https://medium.com/flawless-app-stories/basics-of-parallel-programming-with-swift-93fee8425287

DISPATCH QUEUES

```swift
// Using Concurrent Queue
    let concurrentQueue = DispatchQueue(label: "queuename", attributes: .concurrent)
    concurrentQueue.sync {
        print("Task 1")
    }
    print("Task 1 Done")

concurrentQueue.sync {
    print("Task 2")
}
print("Task 2 Done")
```

RESOURCES

- Swift Tutorial: https://docs.swift.org/swift-book/GuidedTour/GuidedTour.html
- HPC in Swift: http://macadmins.psu.edu/wp-content/uploads/sites/24696/2016/06/psumac2016-17-swift_and_HPC.pdf
- Parallel Programming with Swift: https://medium.com/flawless-app-stories/basics-of-parallel-programming-with-swift-93fee8425287
- Thread Programming with Swift: https://medium.com/better-programming/threading-in-swift-simply-explained-5c8dd680b9b2
- Online Swift Playground: http://online.swiftplayground.run