COCOAHEADS PARIS

JUNE 2020 (ONLINE EDITION)

WEBDAV SYNC WITH COMBINE

FRANK LEFEBVRE

DISCLAIMER

- Work in progress
- My first non-trivial project with Combine
- I'm still learning

AGENDA

- WebDAV Constraints
- Synchronization Algorithm
- Implementation with Combine
- Challenges & Pitfalls

WEBDAV

- What?
- Why?
- Constraints & Assumptions
  - Client-driven
  - Atomicity Limitations
  - Reliability

ARCHITECTURE & ALGORITHM

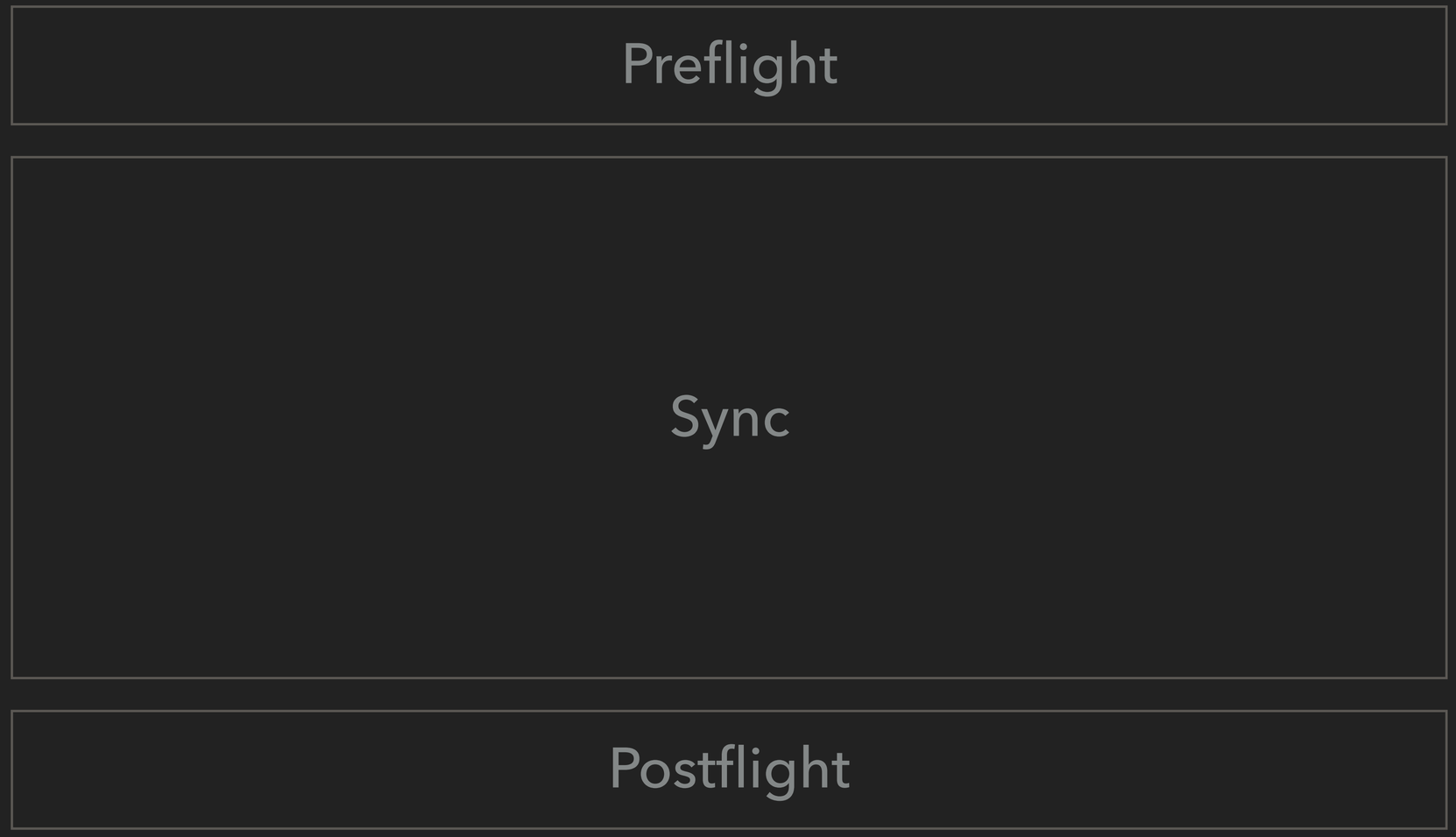
ARCHITECTURE & ALGORITHM (CLIENT REPRESENTATION)

- Synchronized objects
  - unique identifier
  - change counter
  - snapshot during sync

ARCHITECTURE & ALGORITHM (SERVER REPRESENTATION)

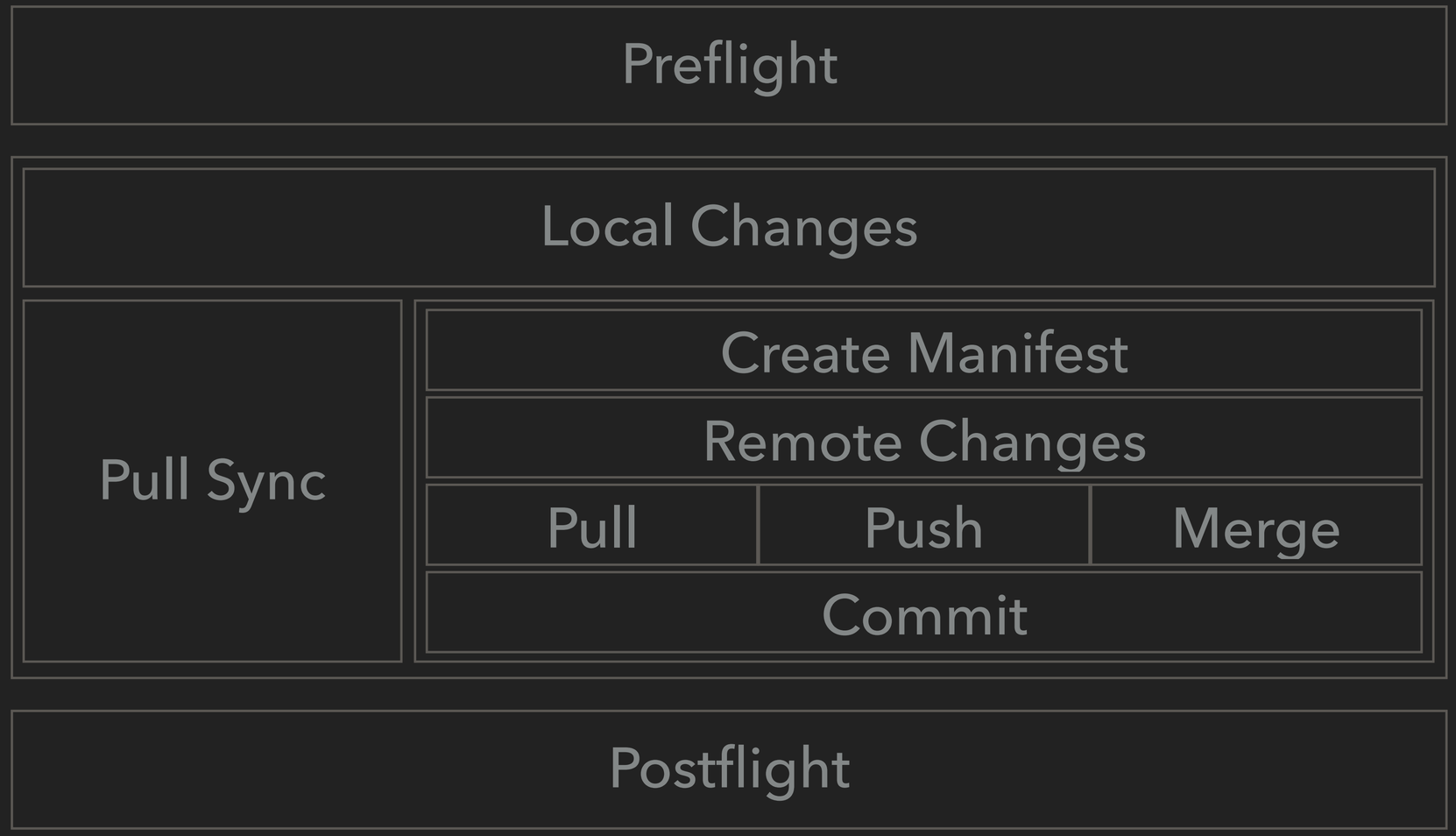
- clients/
  - (id)/
    - description
    - heartbeat
    - incoming/
- manifests/
- sessions/

ARCHITECTURE & ALGORITHM (EVENT SEQUENCE)



ARCHITECTURE & ALGORITHM (EVENT SEQUENCE)

ARCHITECTURE & ALGORITHM (EVENT SEQUENCE)



IMPLEMENTATION TIPS

ADAPTING CALLBACK-BASED ASYNCHRONOUS CODE

ADAPTING CALLBACK-BASED ASYNCHRONOUS CODE

- WebDAV Client: https://github.com/amosavian/FileProvider

ADAPTING CALLBACK-BASED ASYNCHRONOUS CODE

- WebDAV Client: https://github.com/amosavian/FileProvider

- ```
  func contents(path: String, completionHandler: @escaping
  (_contents: Data?, _ error: Error?) -> Void) -> Progress?
  ```
  ```

ADAPTING CALLBACK-BASED ASYNCHRONOUS CODE

- WebDAV Client: https://github.com/amosavian/FileProvider

- `func contents(path: String, completionHandler: @escaping (_contents: Data?, _ error: Error?) -> Void) -> Progress?`

- `func contents(path: String) -> AnyPublisher<Data, Error>`

ADAPTING CALLBACK-BASED ASYNCHRONOUS CODE

```
public func contents(path: String) -> AnyPublisher<Data, Error> {
    let subject = PassthroughSubject<Data, Error>()
    contents(path: path) { (data, error) in
        if let data = data {
            subject.send(data)
        }
        if let error = error {
            subject.send(completion: .failure(error))
        }
        else {
            subject.send(completion: .finished)
        }
    }
    return subject.eraseToAnyPublisher()
}
```

ADAPTING CALLBACK-BASED ASYNCHRONOUS CODE

```
private func publisher<T, E: Error>(action: (@escaping (T?, E?) -> ()) -> ()) -> AnyPublisher<T, E> {
    let subject = PassthroughSubject<T, E>()
    action() { (value, error) in
        if let value = value {
            subject.send(value)
        }
        // ...
    }
    return subject.eraseToAnyPublisher()
}
```

ADAPTING CALLBACK-BASED ASYNCHRONOUS CODE

```
private func publisher<T, E: Error>(action: (@escaping (T?, E?) -> ()) -> ()) -> AnyPublisher<T, E> {
    let subject = PassthroughSubject<T, E>()
    action() { (value, error) in
        if let value = value {
            subject.send(value)
        }
        // ...
    }
    return subject.eraseToAnyPublisher()
}

public func contents(path: String) -> AnyPublisher<Data, Error> {
    return publisher() { callback in
        self.contents(path: path) { callback($0, $1) }
    }
}
```

SYNCHRONIZATION ENGINE IMPLEMENTATION

- Dependency Injection
  - localStore, remoteStore, loader
- `func start() -> AnyPublisher<Void, Error>`
- Short methods
  - return `AnyPublisher<T, Error>`
  - `flatMap` everywhere

ITERATION THROUGH RECURSION

```
func createMissingSubfolders(upTo location: [String], currentIndex: Int = 0) ->
AnyPublisher<Void, Error> {

    guard currentIndex < location.count else {

        return Just(()).setFailureType(to: Error.self).eraseToAnyPublisher()

    }

    return self.createFolderIfMissing(location[...currentIndex])

        .flatMap {

            self.createMissingSubfolders(upTo: location, currentIndex: currentIndex + 1)

    }

    .eraseToAnyPublisher()

}
```

USING ZIP FOR CHECKPOINTS

```
let (toPush, toPull, toMerge) = self.mergeAndSplit(...)


let push = self.pushToRemote(objects: toPush)

let pull = self.loadFromRemote(objects: toPull)

let merge = self.resolveConflicts(objects: toMerge)


return Publishers.Zip3(push, pull, merge).map { _, _, _ in () }
```

USING REDUCE FOR CHECKPOINTS

```
func pushToRemote(objects: [ObjectIdentifier]) -> AnyPublisher<Void, Error> {
    guard !objects.isEmpty else {
        return Just(()).setFailureType(to: Error.self).eraseToAnyPublisher()
    }
    return objects.publisher
        .setFailureType(to: Error.self)
        .flatMap { objectIdentifier in
            self.loader.loadLocalObject(at: objectIdentifier)
                .flatMap { localObject in
                    self.loader.storeRemoteObject(localObject, at: objectIdentifier)
                }
        }
        .reduce(()) { _, _ in () }
        .eraseToAnyPublisher()
}
```

IGNORING FAILURES

```
func createFolderIfMissing(_ location: [String]) -> AnyPublisher<Void, Error> {
    guard let name = location.last else {
        fatalError("This can't be called with an empty path.")
    }
    let parent = location.dropLast().joined(separator: "/")
    return fileProvider.create(folder: name, at: parent)
        .tryCatch { error in
            // ... check error ...
            return Just(()).setFailureType(to: Error.self)
    }
    .eraseToAnyPublisher()
}
```

UNIT TESTS: SYNCHRONOUS EXECUTION

```
func wait<T, E: Swift.Error>(_ publisher: AnyPublisher<T, E>, timeout seconds: Int = 5) throws -> T {
    var subscriptions = Set<AnyCancellable>()
    let sema = DispatchSemaphore(value: 0)
    var result: Result<T, E>?
    publisher
        .sink(receiveCompletion: { completion in
            if case let .failure(error) = completion {
                result = .failure(error)
            }
            sema.signal()
        }, receiveValue: { value in
            result = .success(value)
        })
        .store(in: &subscriptions)
    guard case .success = sema.wait(timeout: .now() + .seconds(seconds)) else {
        throw Error.timeout
    }
    guard let receivedResult = result else {
        throw Error.missingValue
    }
    subscriptions.removeAll()
    return try receivedResult.get()
}
```

INTEGRATION TESTS: DOCKER

```
mkdir ~/Public/dav


docker run -v ~/Public/dav:/var/lib/dav \
    -e AUTH_TYPE=Digest \
    -e USERNAME=test -e PASSWORD=correcthorsebatterystaple \
    --publish 80:80 --name webdav \
    -e LOCATION=/webdav -d bytemark/webdav
```

CHALLENGES, PITFALLS

IDIOMATIC COMBINE?

- `eraseToAnyPublisher()`: cost?
  - time
  - energy
  - space

- `AnyPublisher<Void, Error>`
  - `replaceEmpty(with: ())`
  - `return Just(()).setFailureType(to: Error.self).eraseToAnyPublisher()`
  - `Future<Void, Error>`

MAP VS MAP: READABILITY

```
func objectIdentifiers(...) ->
AnyPublisher<[ObjectIdentifier], Error> {

    return fileProvider.contentsOfDirectory(path: ...)

        .map { $0.map { $0.name } }

        .eraseToAnyPublisher()

}
```

CONCLUSION

- Debugging async code is still hard
- Lifecycle Tools
  - .print()
  - .handleEvents()
  - Timelane + Instruments

QUESTIONS

@franklefebvre