Server-side Swift with Kitura and Vapor

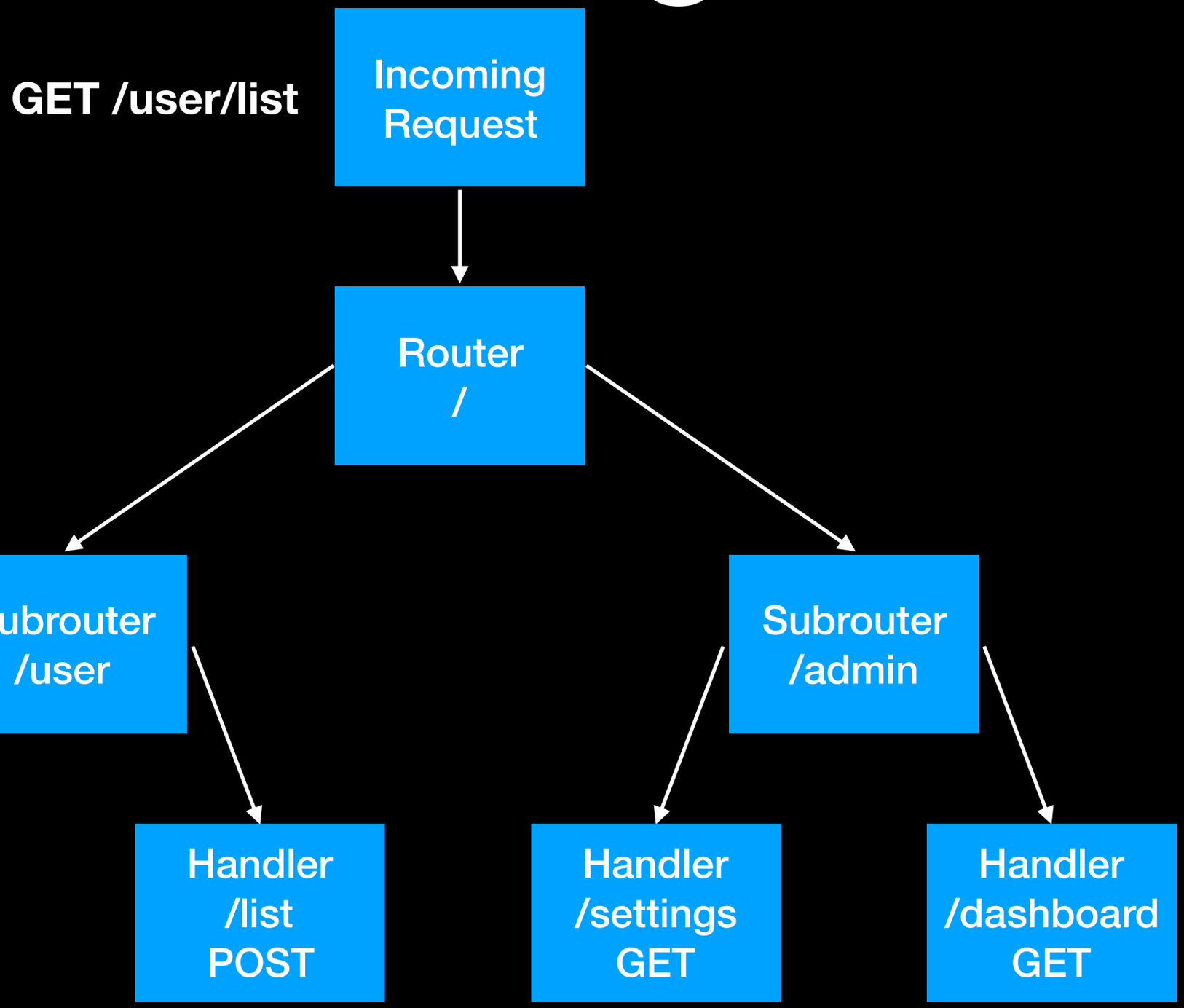
Cocoaheads Aachen, January 2020

Jonas Schlabertz

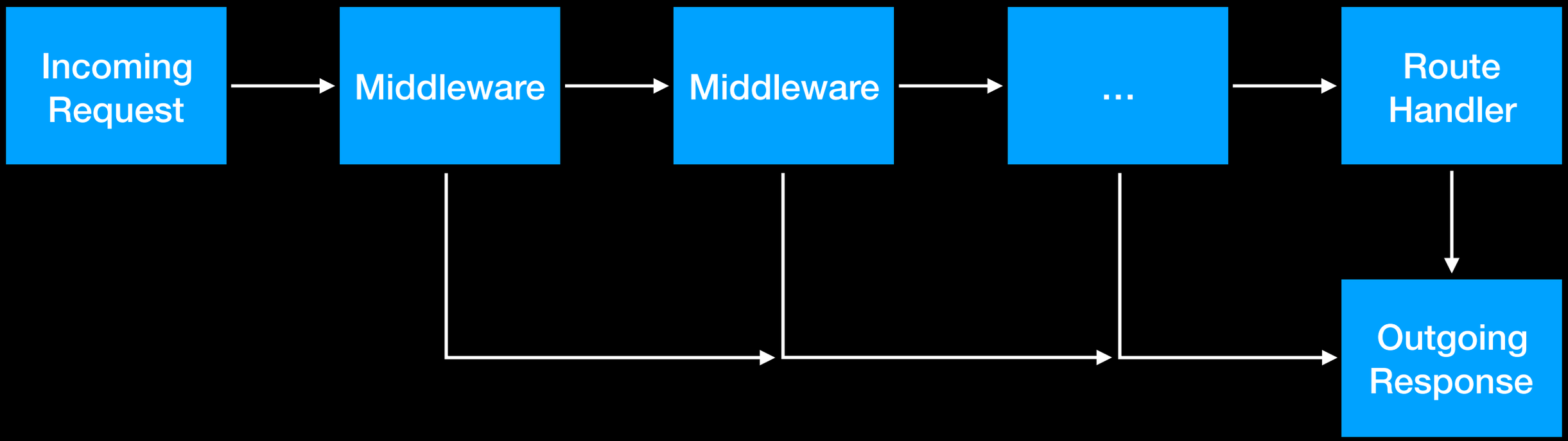
Why is this relevant?

- CloudKit is only good as data storage
- Firebase also cannot do everything
    - Also not having Google in your app is a plus
- Custom backend needed in some use-cases
- No need to learn an additional language beside Swift

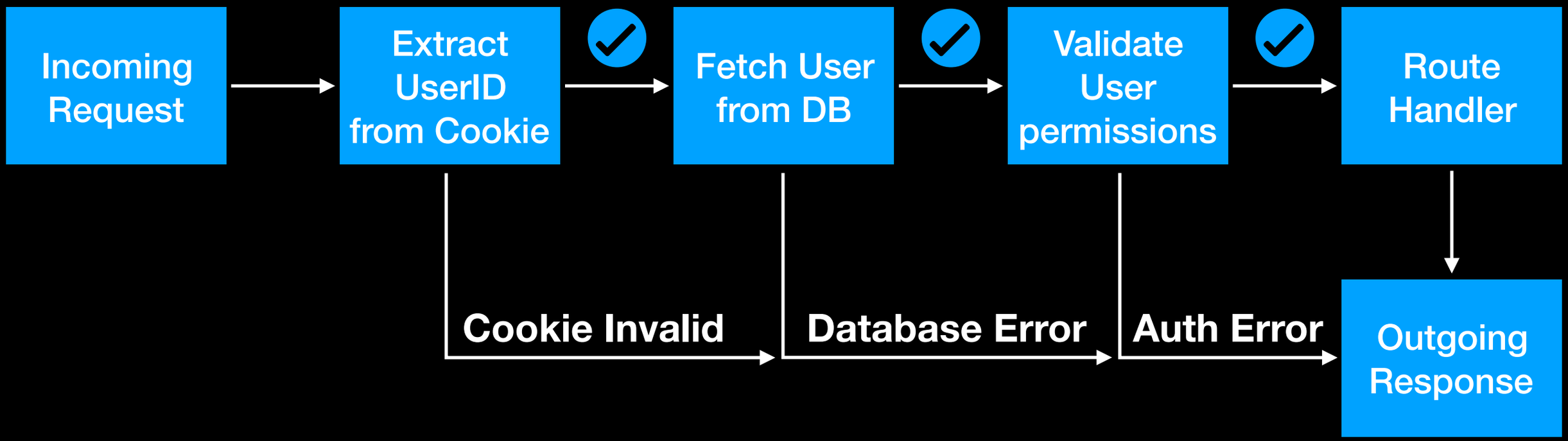
Routing


GET /user/list
Incoming Request
Router /
Subrouter /user
Subrouter /admin
Handler /list GET
Handler /list POST
Handler /settings GET
Handler /dashboard GET

Middlewares



Middlewares



Kitura

- Developed by IBM
- Runs on macOS and Linux
- API heavily inspired by Express.js
- API mostly compatible with older Kitura versions

Kitura - Getting Started

```
import Kitura

let router = Router()

router.get("/info") { request, response, next in
    response.send("Hey, is this thing on?")
}

Kitura.addHTTPServer(onPort: 8080, with: router)

Kitura.run()
```

Kitura - Middlewares

```
class DemoMiddleware: RouterMiddleware {
    func handle(request: RouterRequest,
                response: RouterResponse,
                next: @escaping () -> Void) throws {
        print(request.headers)
        print(request.body)
        print(request.cookies)

        next()
    }
}

router.all(middleware: DemoMiddleware())
```

Kitura - Databases

```
import SwiftKuery
import SwiftKueryORM
import SwiftKueryPostgreSQL

// Database Model
struct UserModel: Model {
    let name: String
    let age: Int
    let mail: String
}

// Codable Route - Body automatically parsed
router.post("/usermodel") { (user: UserModel,
                             completion: @escaping (UserModel?,
                                                    RequestError?) -> Void) in
    user.save(completion)
}
```

Kitura Live Demo

Vapor

- Built on top of SwiftNIO
  - Network application framework
    - Used to implement network protocols
- More “Swifty”-API
- Vapor API has shown significant changes over different versions

Vapor - Getting Started

```
import Vapor

var config = Config.default()

var services = Services.default()

// Register Services, Routers and Middlewares here
let router = EngineRouter.default()
router.get("/") { (request) -> String in
    return "Hello, is this thing on?"
}

services.register(router, as: Router.self)

let app = try Application(config: config,
                         environment: .detect(),
                         services: services)
try app.run()
```

Vapor - Middlewares

```
class DemoMiddleware: Middleware {
    func respond(to request: Request,
                 chainingTo next: Responder) throws
                 -> EventLoopFuture<Response> {

        // Do some stuff with the request here

        return try next.respond(to: request)
    }
}

var middlewares = MiddlewareConfig()
middlewares.use(DemoMiddleware())
services.register(middlewares)
```

Vapor - Databases

```
import Fluent
import FluentPostgreSQL

final class UserModel: PostgreSQLModel {
    var id: Int?
    var name: String
    var mail: String
}

extension UserModel: Migration { }

var migrations = MigrationConfig()
migrations.add(model: UserModel.self, database: .psql)

services.register(migrations)
```

Vapor - Database Migrations

```
final class UserModel: PostgreSQLModel {
    …
    // Now additionally store the user's age
    var age: Int
}

struct UserModelMigration: Migration {
    static func prepare(on conn: PostgreSQLConnection) -> EventLoopFuture<Void> {
        return PostgreSQLDatabase.update(UserModel.self, on: conn) { (updater) in
            let constraint = PostgreSQLColumnConstraint
                                .default(.literal(PostgreSQLLiteral(integerLiteral: -1)))
            updater.field(for: \.age, type: .int, constraint)
        }
    }

    static func revert(on conn: PostgreSQLConnection) -> EventLoopFuture<Void> {
        return PostgreSQLDatabase.update(UserModel.self, on: conn) { (updater) in
            // We don't need this... I hope
        }
    }
}

migrations.add(migration: UserModelMigration.self, database: .psql)
```

Vapor Live Demo

Production Ready?

- Docker supported by Kitura and Vapor
- Both frameworks provide libraries for Authentication, Sessions, Websockets, CORS and Templating
- Database Migrations very easy in Vapor
    - Little bit more complex in Kitura
- December 2019: IBM is no longer working on Swift
    - https://forums.swift.org/t/december-12th-2019/31735
    - Consequences for Kitura still unknown

Thank you!

Jonas Schlabertz
jonas@schlabertz.de