2004

The year I started web development

W3C XHTML 1.0 ✓

W3C CSS level 2 ✓

2004

W3C XHTML 1.0

W3C CSS level 2

php

2004

W3C XHTML 1.0

W3C CSS level 2

php

2004

Mess things around

2004

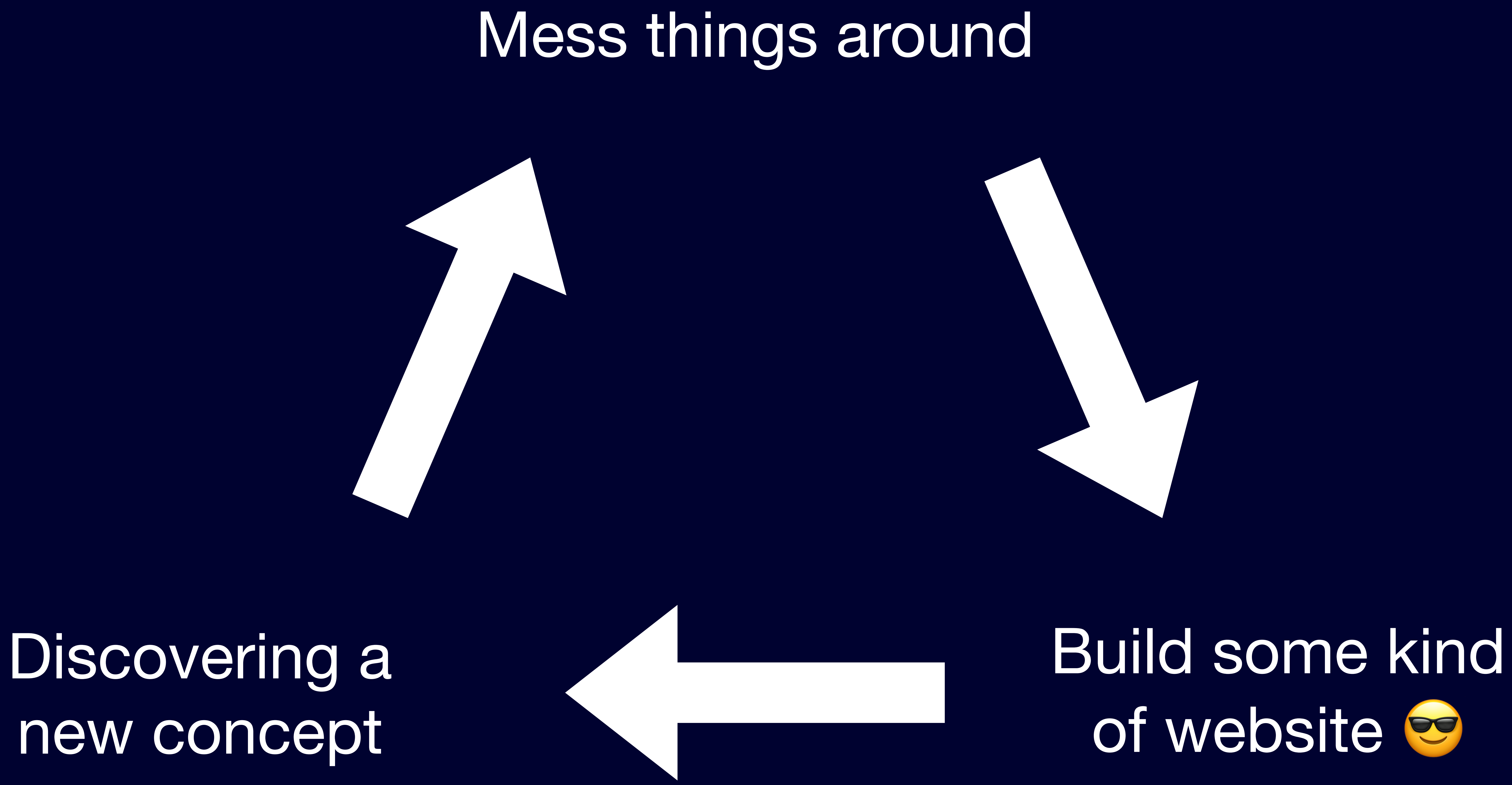
Mess things around

Build some kind of website 😎

2004

$$\begin{tikzcd}
 & \text{Mess things around} \arrow[dr] & \\
\text{Discovering a new concept} & & \text{Build some kind of website 😎} \arrow[ll]
\end{tikzcd}$$

2004


Mess things around
Build some kind of website 😎
Discovering a new concept

2004

2014

The year Swift was introduced

So many things I wish I knew back then…

- Like why I was getting crashes after renaming an @IBOutlet var?
- Why does everything have NS prefixes?
- How am I supposed to prevent my UIViewController from crossing the thousand-line threshold?
- Why are there optionals of optionals?

2004 2014

2017

My job converted to (almost) full backend

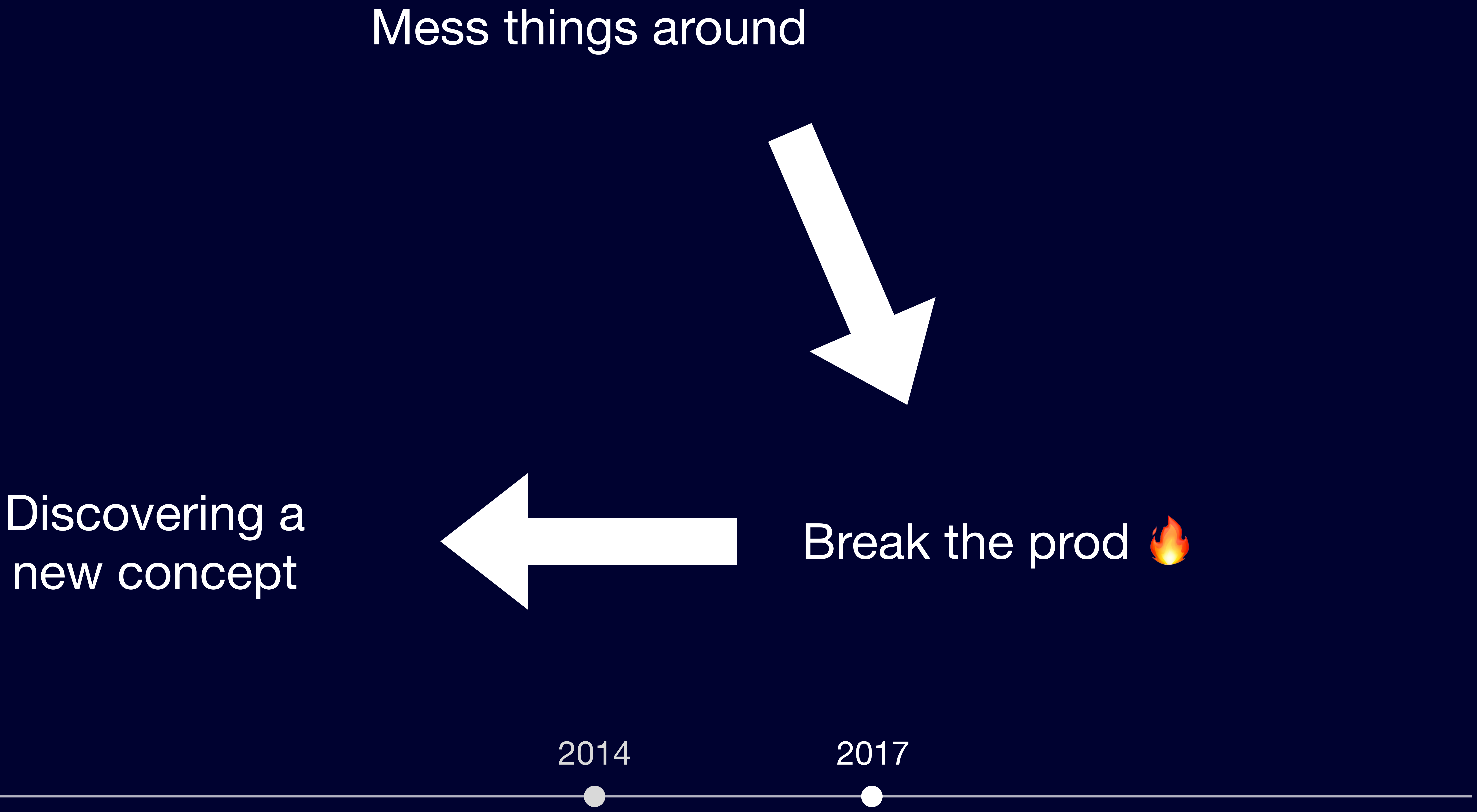
Mess things around

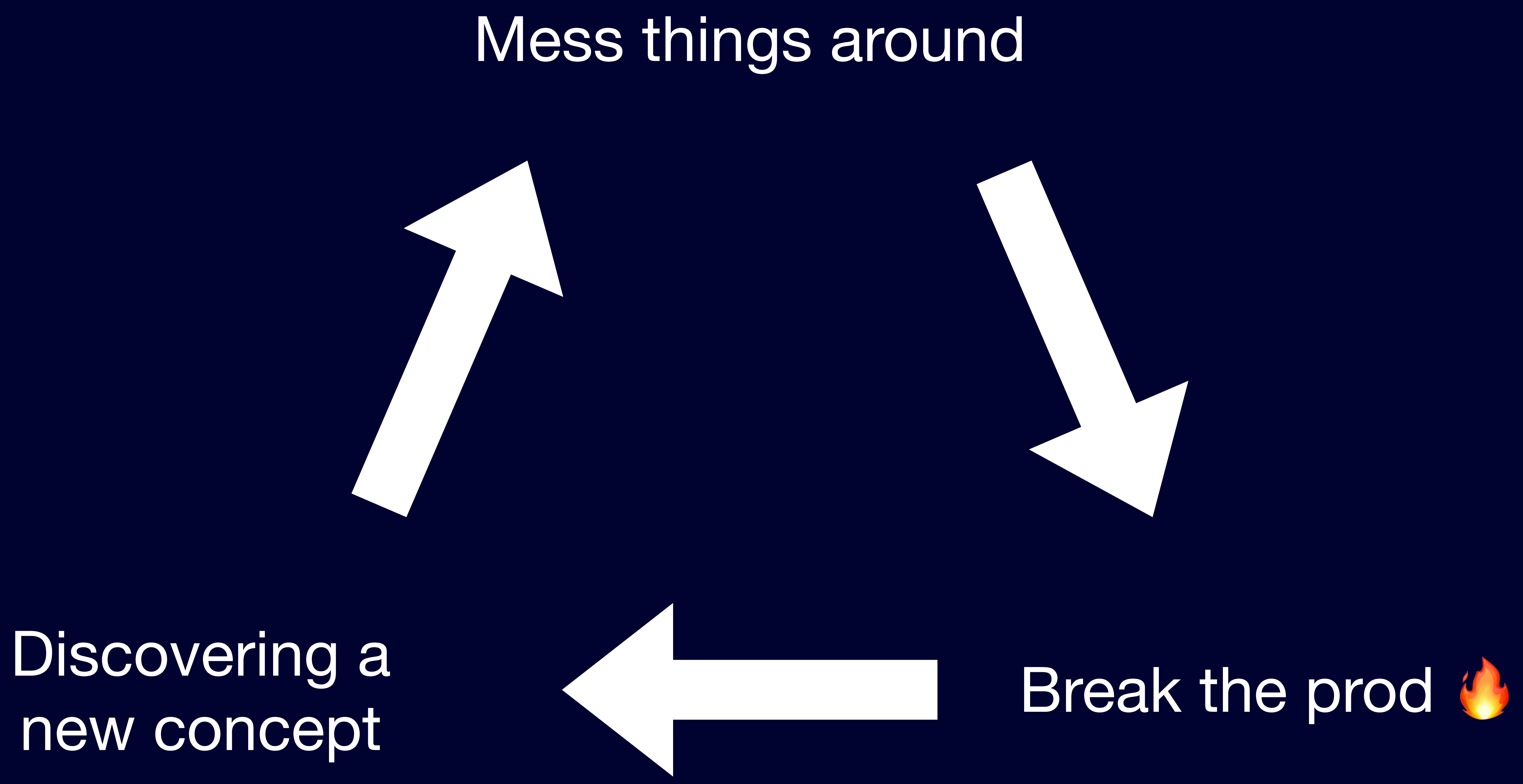
2004 2014 2017

Mess things around

Break the prod 🔥

2004 2014 2017


Mess things around
Break the prod 🔥
Discovering a new concept
2004
2014
2017


Mess things around
Break the prod 🔥
Discovering a new concept

2004
2014
2017

2021

Back at building iOS apps … for me

Ask me stickers!

PAD
LOK

Download on the
App Store

2004 2014 2017 2021

Today

at ServerSide.swift

« I know Swift, I’m learning backend with Swift »

Kickstart your journey into the backend world

Things I wish someone told me when I got started

ServerSide.swift 2024

Thomas Durand

Call me Dean

Backend architect at DiliTrust

iOS indie dev

Speaker, tech blog writer, enthusiast

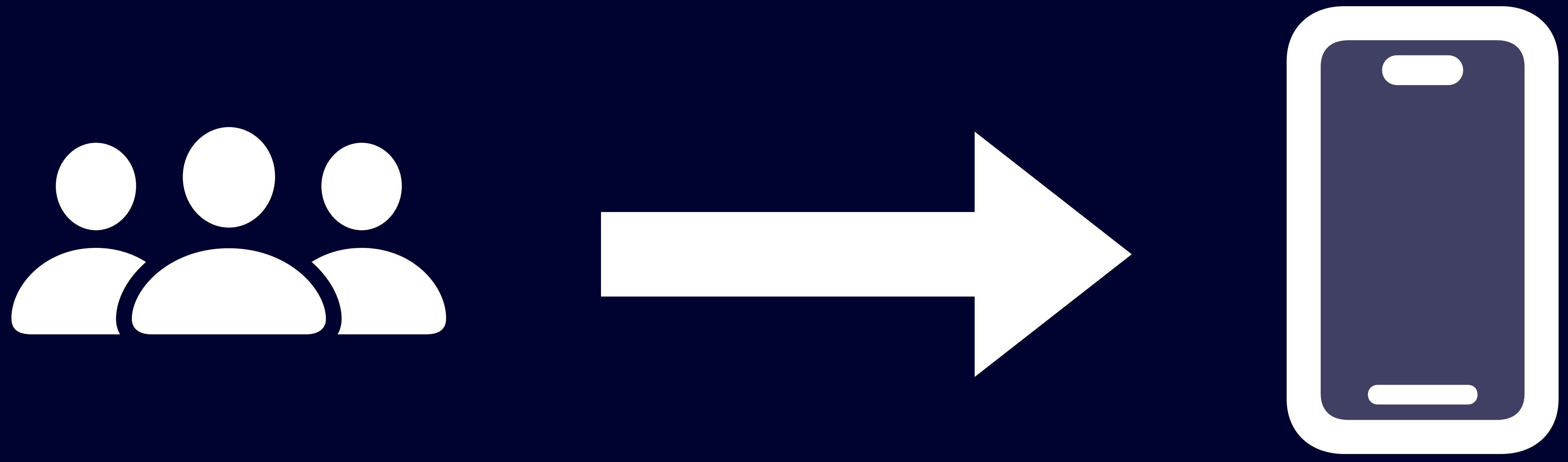
https://thomasdurand.fr

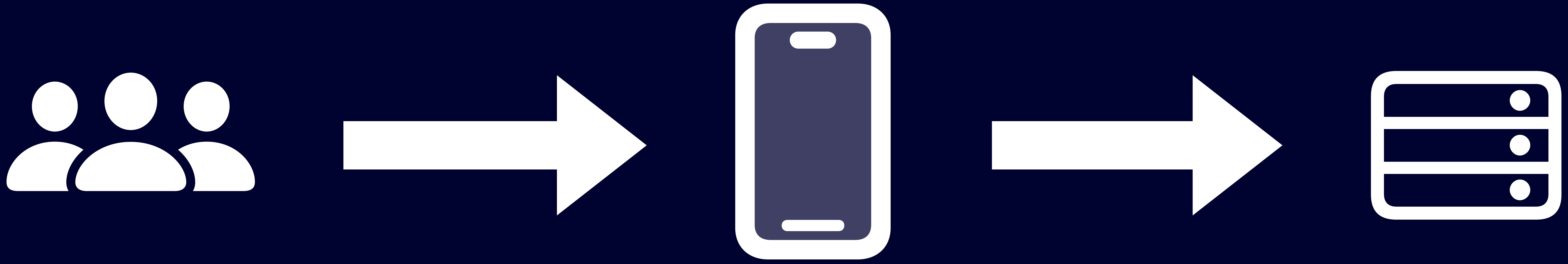
@deanatoire@mastodon.social

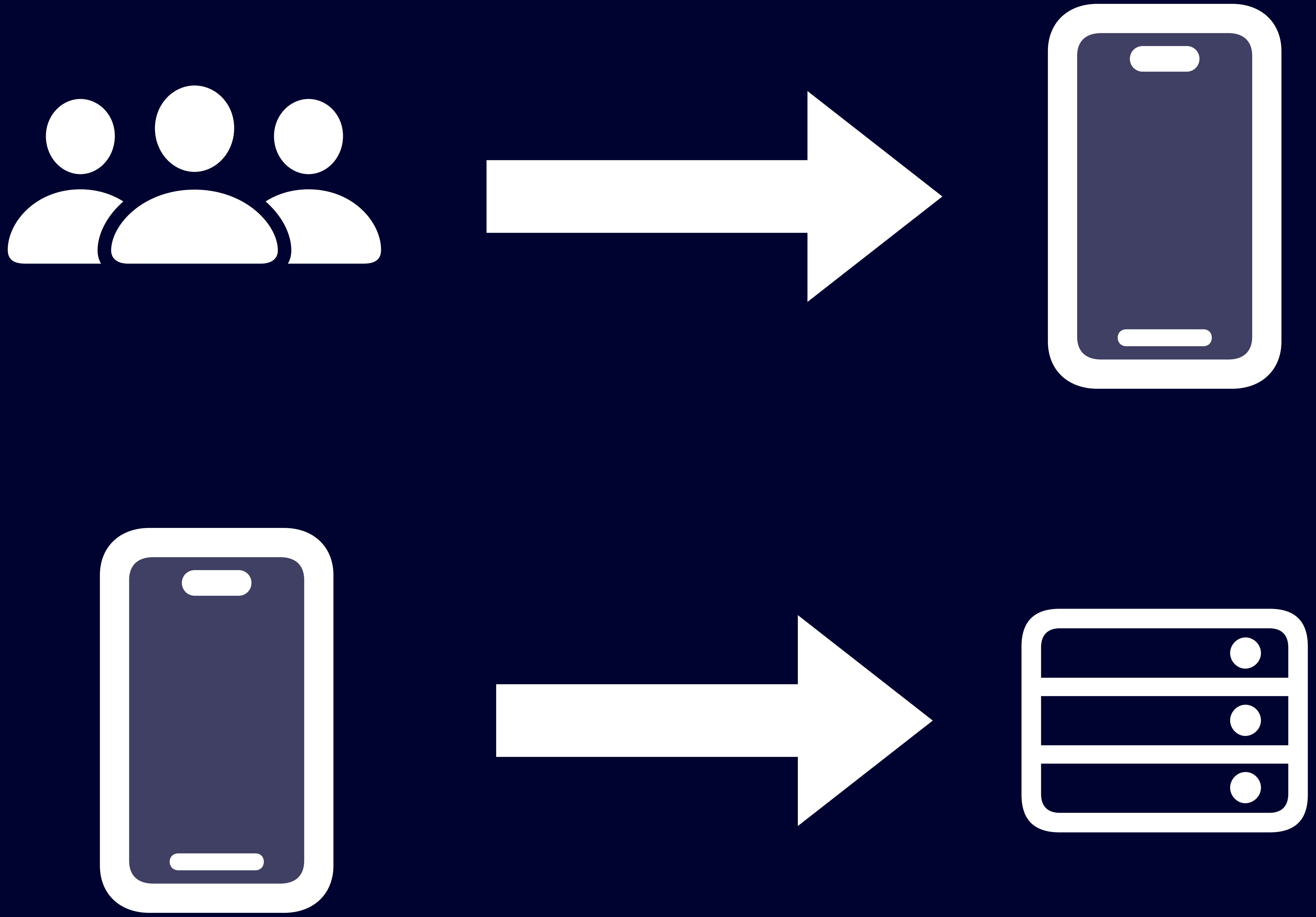
@deantoire@threads.net

@deanatoire@twitter.com

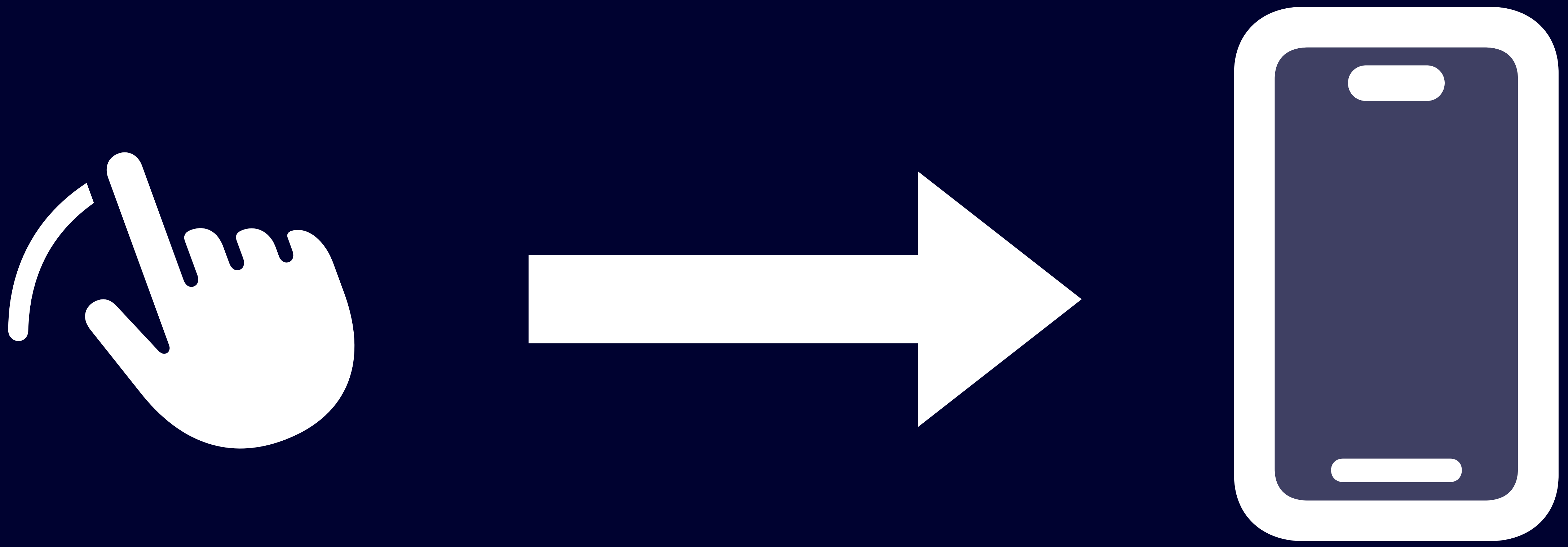
Welcome to the ~~jungle~~ backend!

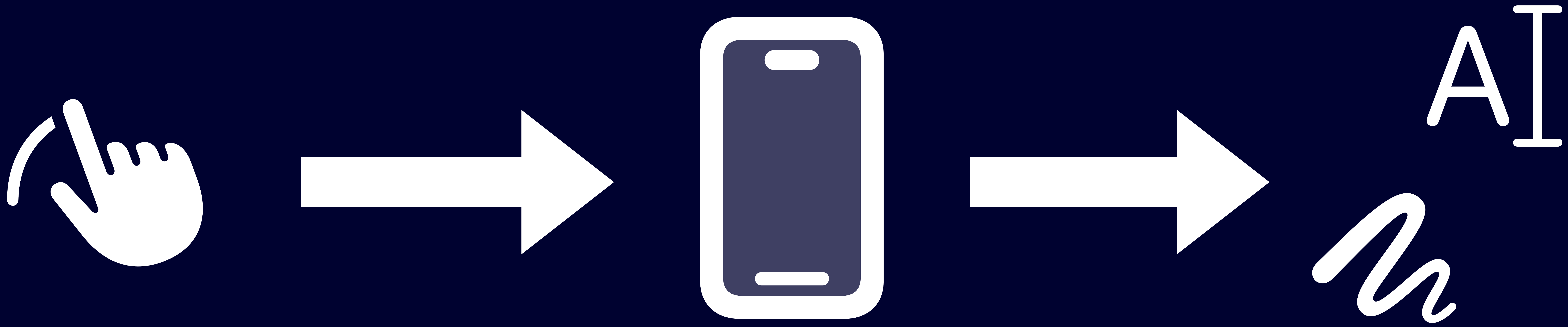


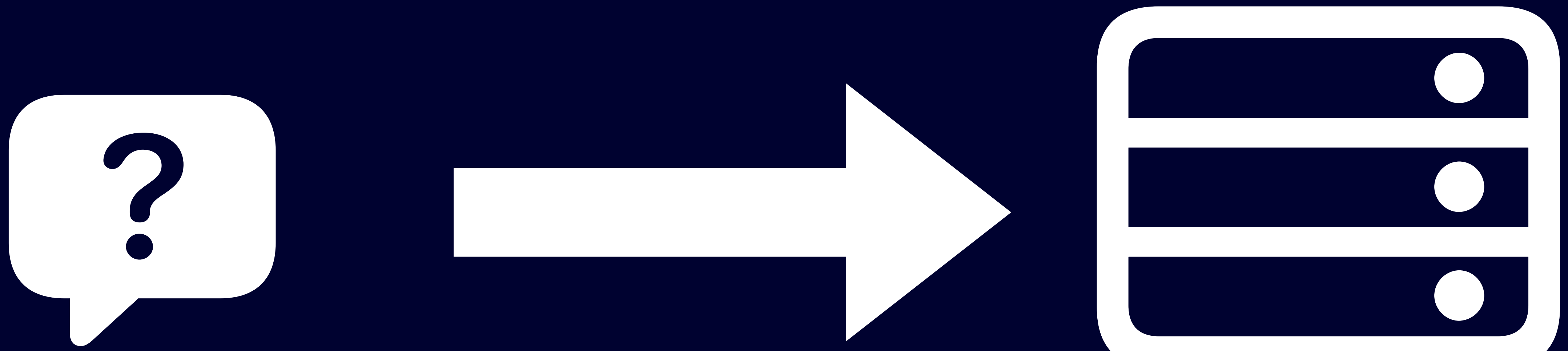


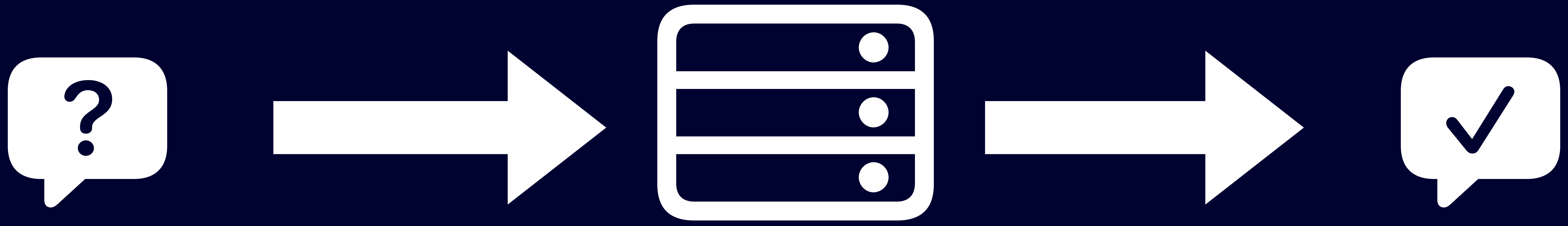


Your app is the user of your backend









Backend is simpler!

Backend is simpler!

is it?

In this session

In this session

Handle incoming requests

In this session

Handle incoming requests

Approach API evolution

In this session

Handle incoming requests

Approach API evolution

Security concerns

In this session

Handle incoming requests

Approach API evolution

Security concerns

Authenticate your requests

In this session

Handle incoming requests

Approach API evolution

Security concerns

Authenticate your requests

Where to go from there?

Handle incoming requests

What’s a request?

What’s a request?

```swift
import Foundation

let baseUrl = URL(string: "https://api.illumineering.fr").unsafelyUnwrapped
var request = URLRequest(url: baseUrl.appending(path: "apps"))
request.httpMethod = "GET"
request.setValue("application/json", forHTTPHeaderField: "Accept")
request.setValue("en-GB,en;q=0.9", forHTTPHeaderField: "Accept-Language")
```

What’s a request?

```swift
import HTTPTypes

var request = HTTPRequest(
  method: .get,
  scheme: "https",
  authority: "api.illumineering.com",
  path: "/apps"
)
request.headerFields[.accept] = "application/json"
request.headerFields[.acceptLanguage] = "en-GB,en;q=0.9"
```

What’s a request?

```swift
// https://github.com/apple/swift-http-types
import HTTPTypes

var request = HTTPRequest(
  method: .get,
  scheme: "https",
  authority: "api.illumineering.com",
  path: "/apps"
)
request.headerFields[.accept] = "application/json"
request.headerFields[.acceptLanguage] = "en-GB,en;q=0.9"
```

What’s a request?

```
    method: .get
```

What’s a request?

```
    path: "/apps"
```

What’s a request?

```
request.headerFields[.accept] = "application/json"
request.headerFields[.acceptLanguage] = "en-GB,en;q=0.9"
```

What’s a request?

```swift
import Foundation

import HTTPTypesFoundation







let (data, response) = try await URLSession.shared.data(for: request)
```

What’s a request?

```swift
import Foundation

import HTTPTypesFoundation


  method: .post,




request.headerFields[.contentType] = "application/json"
let (data, response) = try await URLSession.shared.upload(for: request, from: payload)
```

What’s a request?

```swift
import Foundation
import HTTPTypes
import HTTPTypesFoundation

var request = HTTPRequest(
  method: .post,
  scheme: "https",
  authority: "api.illumineering.com",
  path: "/apps"
)
request.headerFields[.accept] = "application/json"
request.headerFields[.acceptLanguage] = "en-GB,en;q=0.9"
request.headerFields[.contentType] = "application/json"
let (data, response) = try await URLSession.shared.upload(for: request, from: payload)
```

Route

Headers

Payload

What’s in the response?

```
(data, response)
```

What’s in the response?

```
_ = data // x,xxx bytes
```

What’s in the response?

```
_ = data // x,xxx bytes

_ = response.status // 200 OK
```

What’s in the response?

```
_ = data // x,xxx bytes

_ = response.status // 200 OK

_ = response.headerFields[.contentDisposition] // application/json; charset=utf-8
```

Build a response

Build a response

```swift
import Vapor

func routes(_ app: Application) throws {
    // ...
}
```

Build a response

```
import Vapor

func routes(_ app: Application) throws {
    app.get("apps") { req async -> ... in
        // ...
    }
}
```

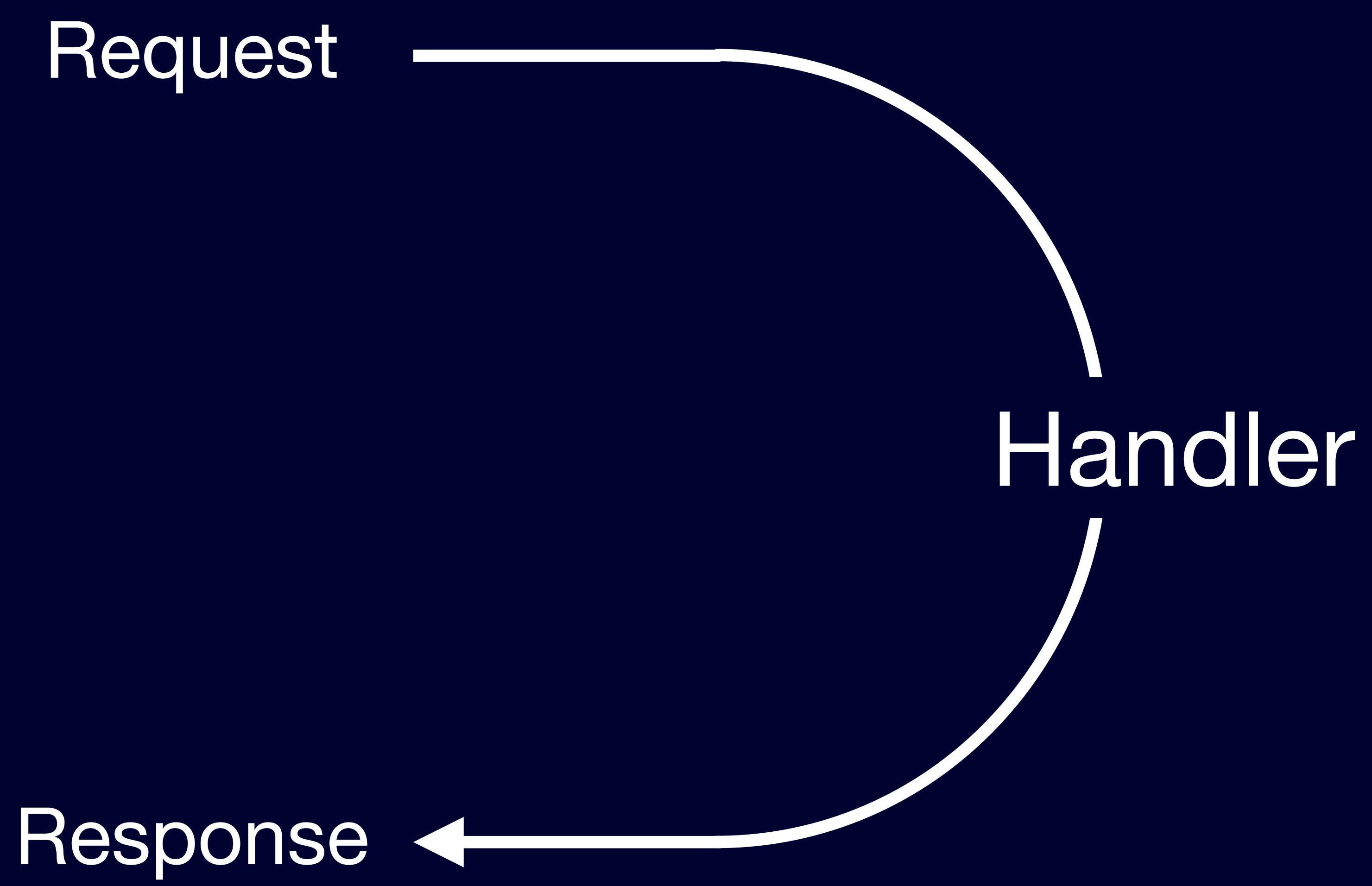
Build a response

```
get "apps"
```

Build a response

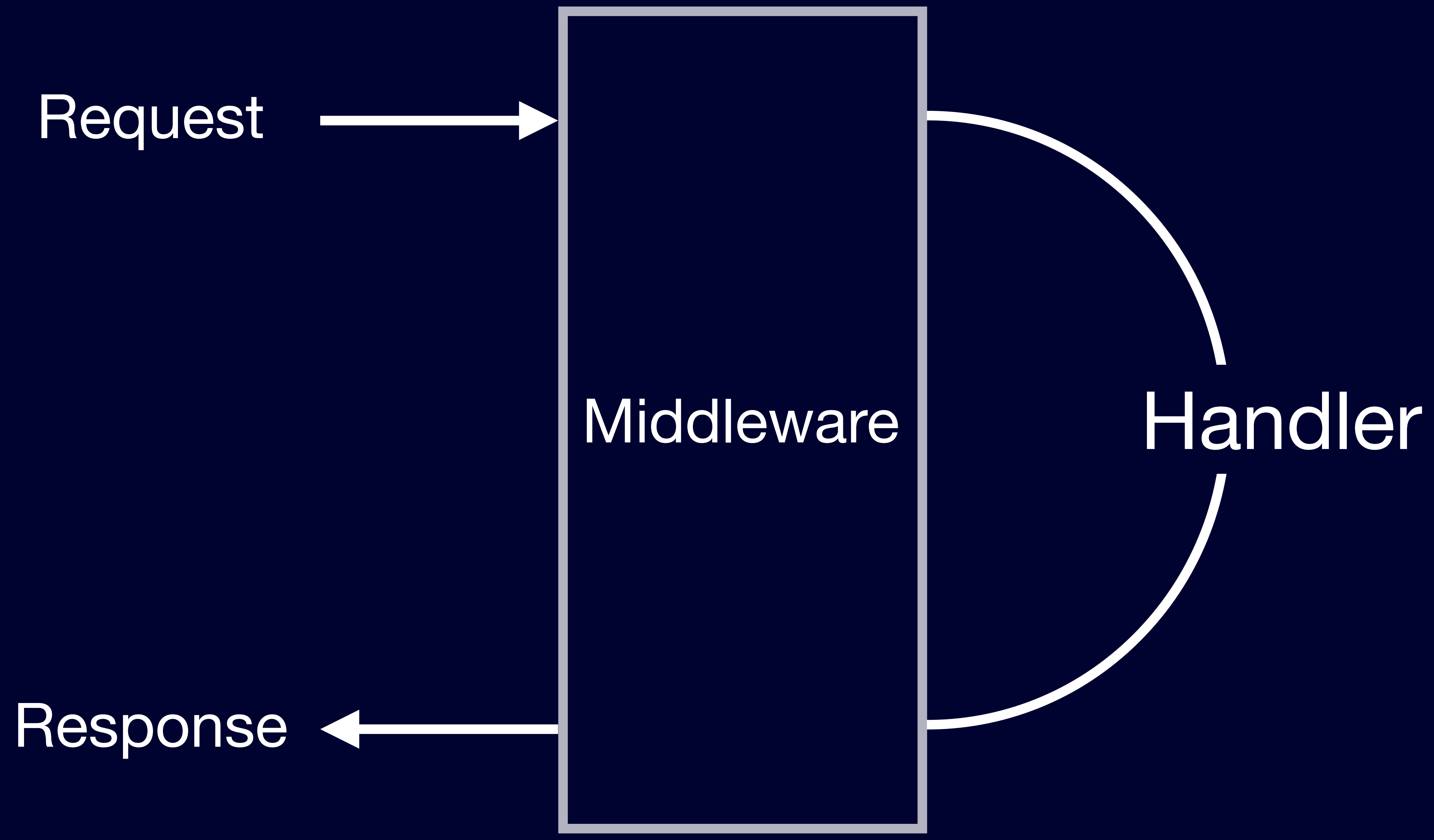
```swift
import Vapor

func routes(_ app: Application) throws {
    app.get("apps") { req async -> [App] in
        App.allCases
    }
}

struct App: Content {
    let id: String
    let name: String
    let weight: Int
}

extension App: CaseIterable {
    // ...
}
```

The responder chain



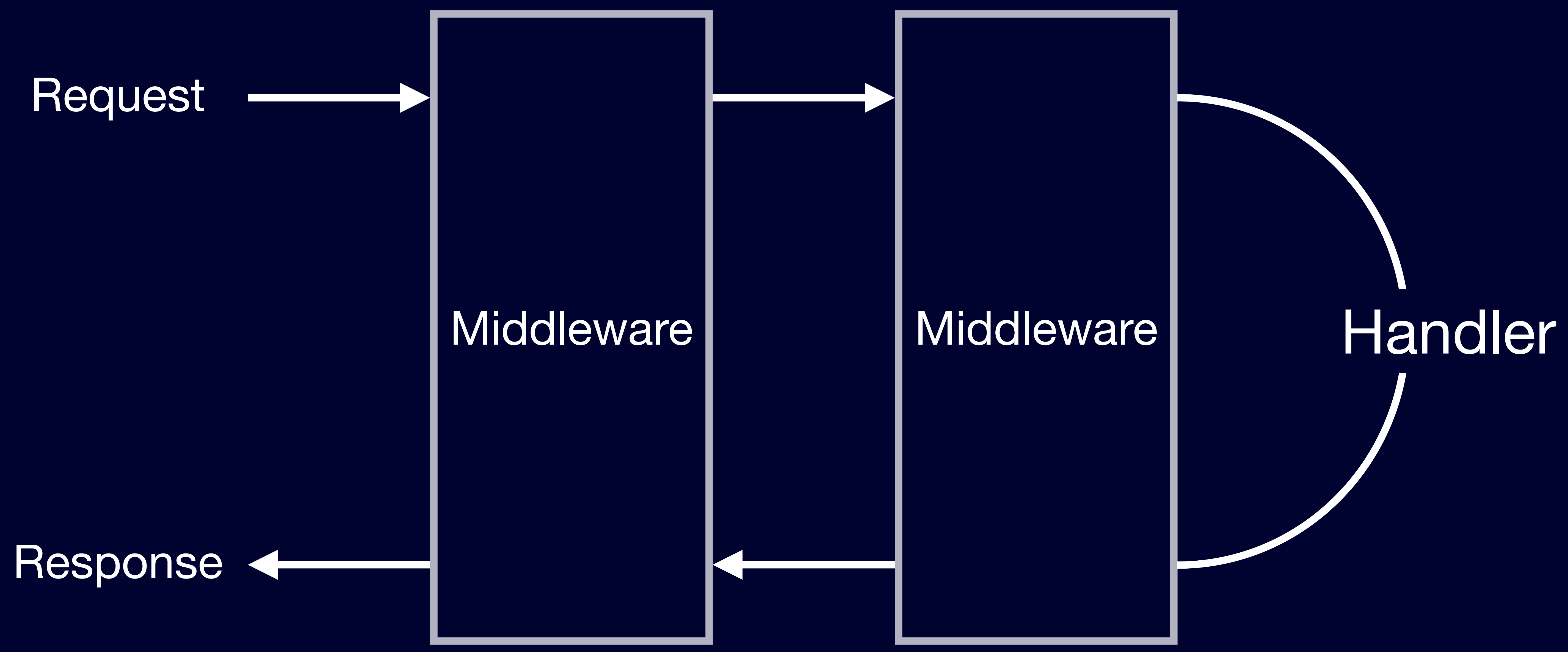
The responder chain

and the concept of middleware



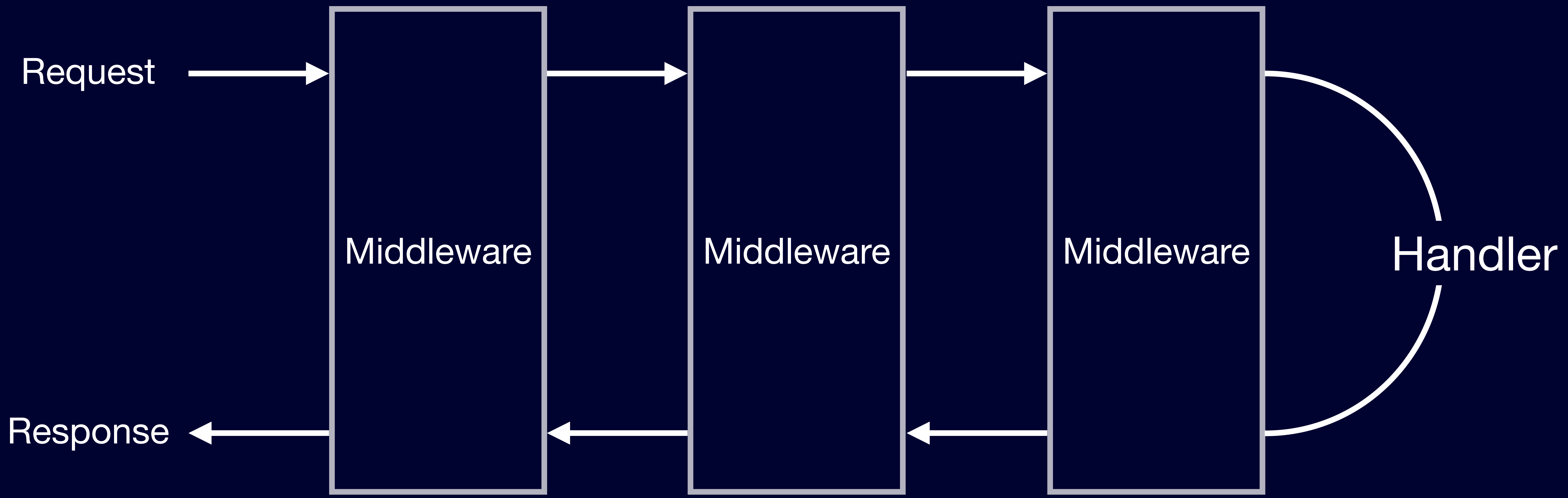
The responder chain

and the concept of middleware



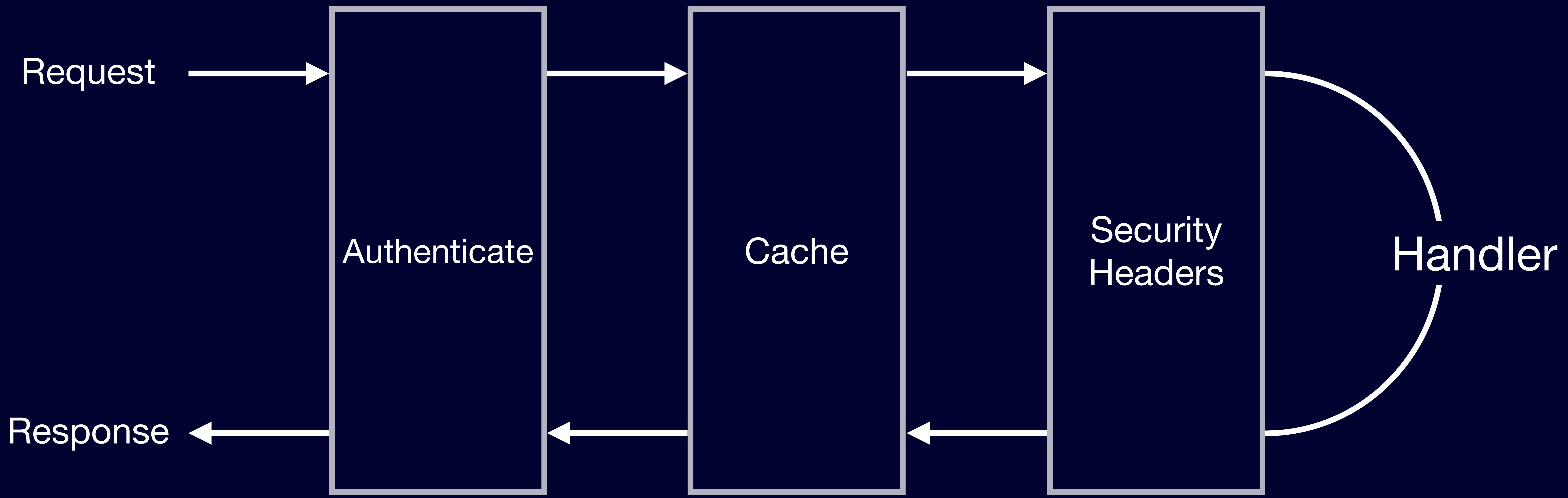
The responder chain

and the concept of middleware



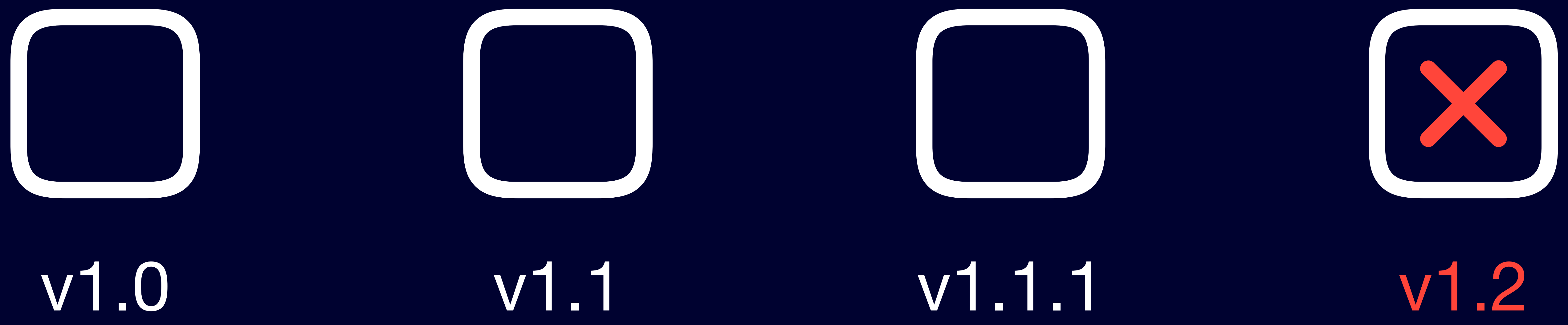
The responder chain

and the concept of middleware

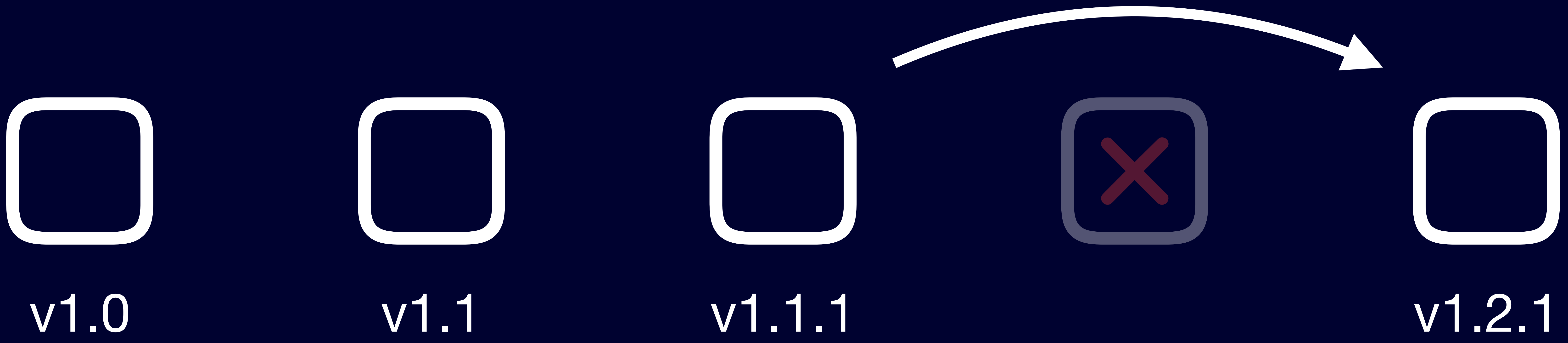


Middlewares bring separation of concerns & reusability

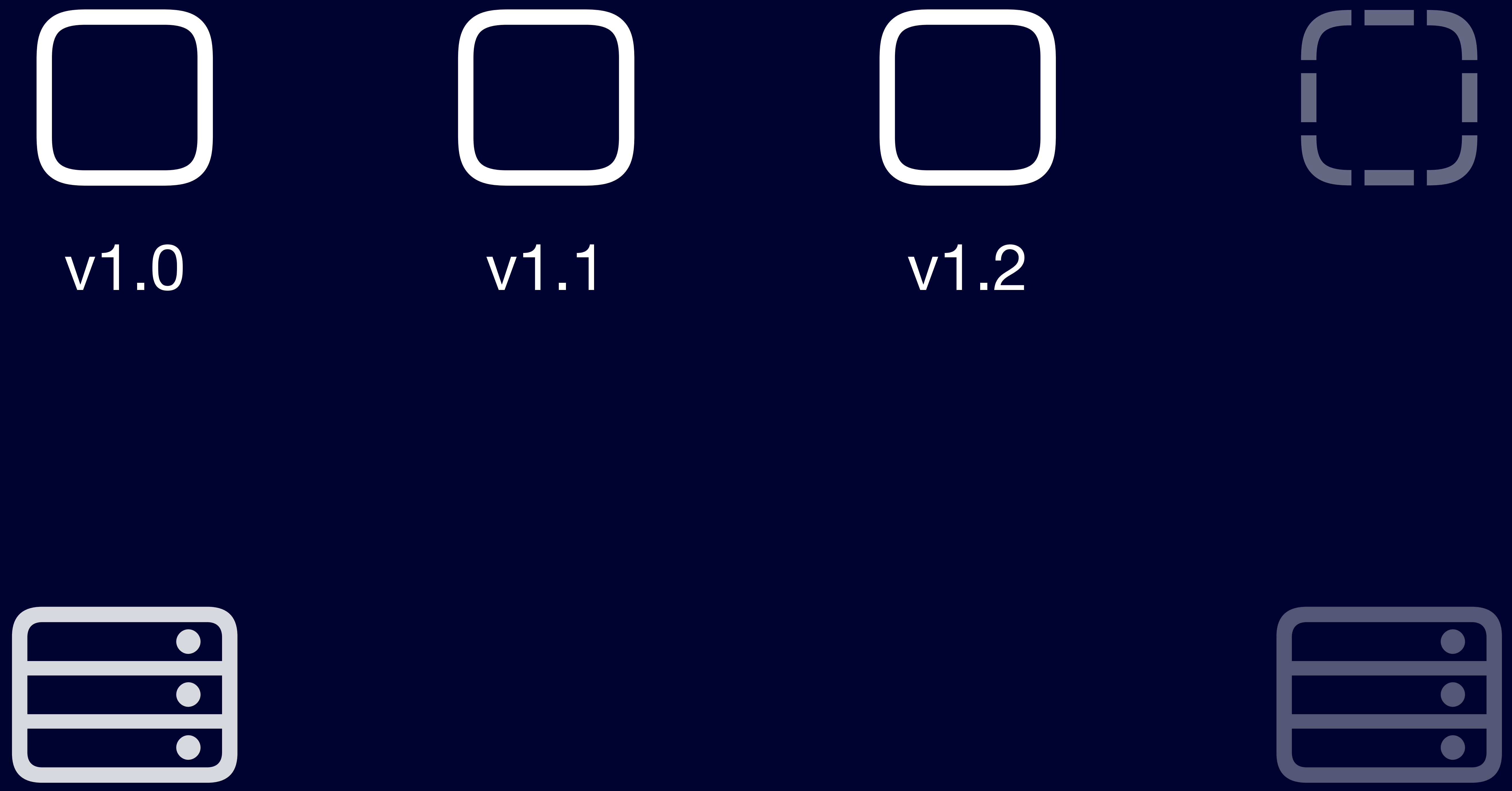
Approach API evolution

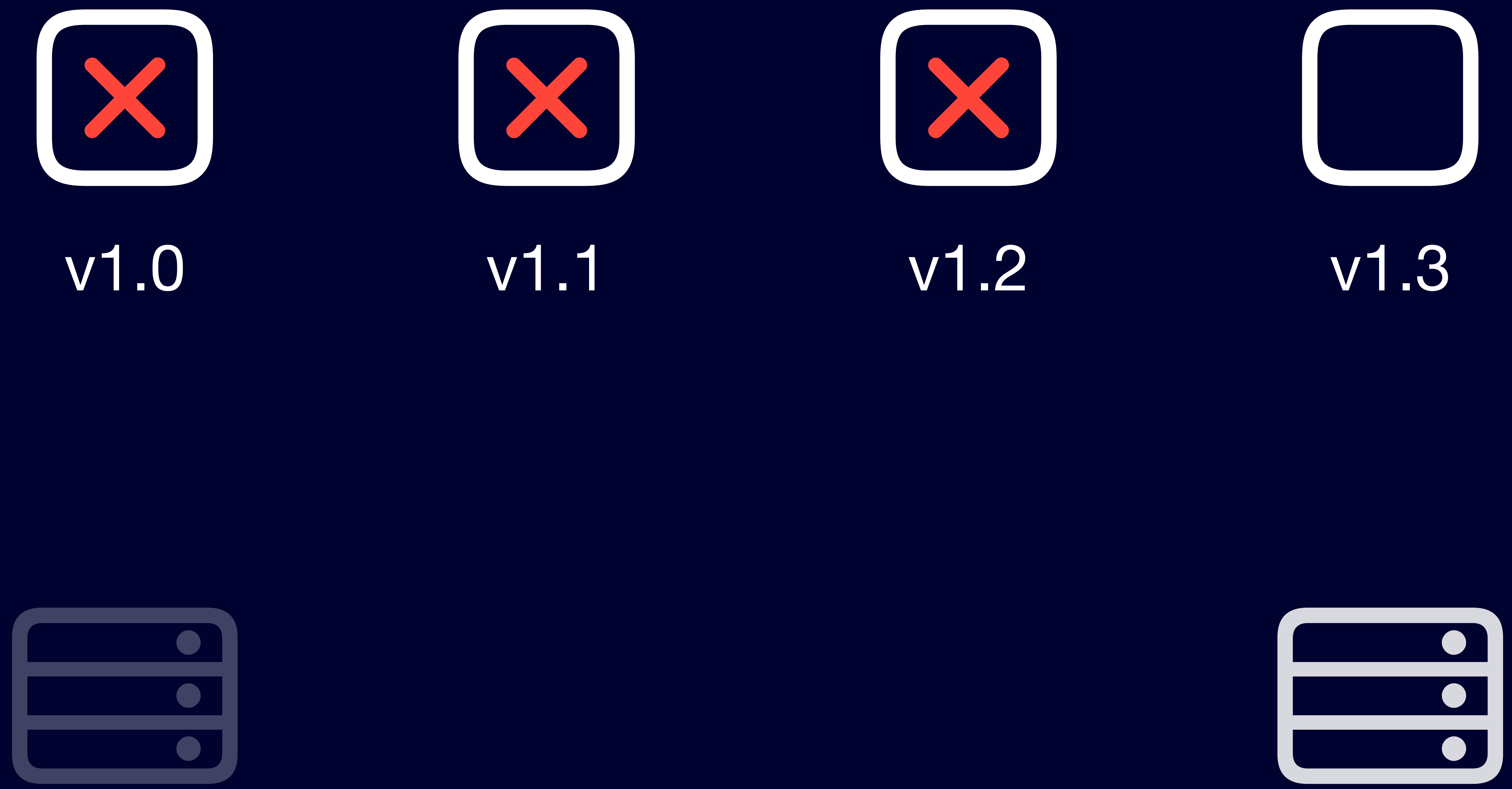

v1.0
v1.1
v1.1.1
v1.2

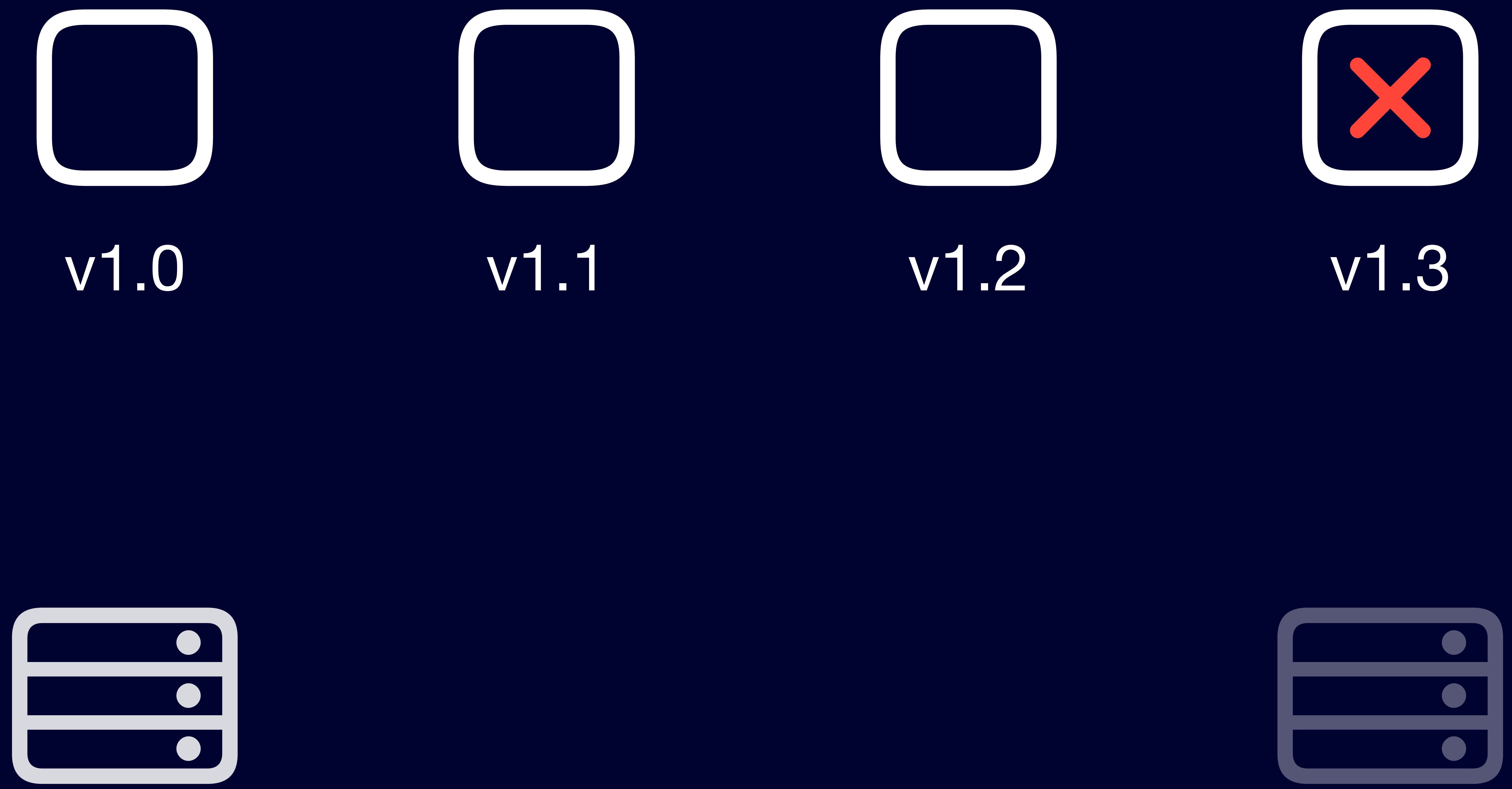
☐	☐	☐	☒	☐
v1.0	v1.1	v1.1.1	v1.2	v1.2.1

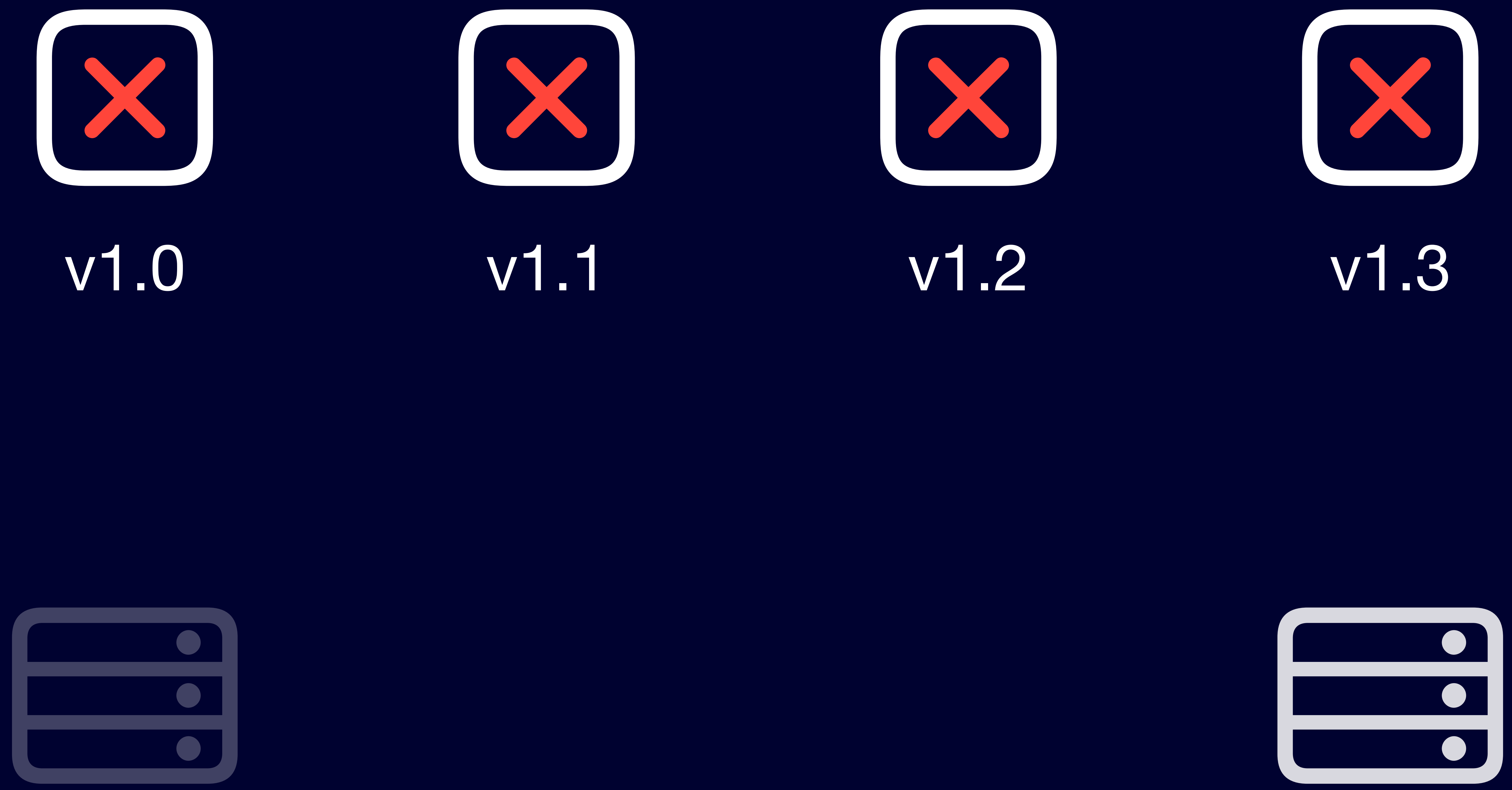

v1.0
v1.1
v1.1.1
v1.2.1

v1.0
v1.1
v1.2


v1.0
v1.1
v1.2


v1.0
v1.1
v1.2
v1.3


v1.0
v1.1
v1.2
v1.3


v1.0
v1.1
v1.2
v1.3

How to avoid that?

Write Tests !

Functional Tests

```swift
@testable import App

import Vapor
import XCTVapor

@Test("Checks apps are returned")
func appsRouteReturnAppsList() async throws {

        try await app.test(.GET, "apps") { res throws in
            try #require(res.status == .ok)
            let json = try res.content.decode([App].self)
            if json.isEmpty {
                throw AppTestError.noApps
            }
            guard let padlok = json.first(where: { $0.id == "padlok" }) else {
                throw AppTestError.missingApp(id: "padlok")
            }
            #expect(padlok.name == "Padlok")
            #expect(padlok.weight == 100)
        }

}
```

Functional Tests

```
.GET  "apps"
```

Functional Tests

```
[App].self
```

Functional Tests

```swift
struct App: Content {
    let id: String
    let name: String
    let weight: Int
}
```

```swift
[App].self
```

This tests that the server works

This tests that the server works

but does not test against your app expectations

Test against your app expectations

```swift
/// The app struct as v1.0 knows it!
private struct AppV1_0: Decodable {
    let id: String
    let name: String
}

@Test("Checks apps are compatible with v1.0")
func appsRouteReturnAppsListForV1_0() async throws {

        try await app.test(.GET, "apps") { res throws in
            try #require(res.status == .ok)
            _ = try res.content.decode([AppV1_0].self)
        }

}
```

Add tests for new app versions!

```swift
/// The app struct as v1.2 knows it!
private struct AppV1_2: Decodable {
    let id: String
    let name: String
    let weight: Int
}

@Test("Checks apps are compatible with v1.2")
func appsRouteReturnAppsListForV1_2() async throws {

        try await app.test(.GET, "apps") { res throws in
            try #require(res.status == .ok)
            _ = try res.content.decode([AppV1_2].self)
        }

}
```

Tests bring confidence in deployments

Plan ahead!

Introduce "backward compatible" changes

Plan ahead!

Introduce "backward compatible" changes

```
GET https://api.illumineering.fr/apps
```

```
{
  "apps": [...]
}
```

Plan ahead!

Introduce "backward compatible" changes

```
GET https://api.illumineering.fr/apps?type=utility
```

```
{
  "apps": [...]
}
```

Plan ahead!

Introduce "backward compatible" changes

```
GET https://api.illumineering.fr/apps?type=utility
```

```
{
  "apps": [...],
  "featured": [...]
}
```

Plan ahead!

Introduce "backward compatible" changes

```
POST https://api.illumineering.fr/feedback
```

```
{
  "message": "This app is awesome!"
}
```

Plan ahead!

Introduce "backward compatible" changes

```
POST https://api.illumineering.fr/feedback
```

```
{
  "message": "This app is awesome!",
  "email": "an.optional.email@thomasdurand.fr"
}
```

Plan ahead!

Introduce "backward compatible" changes

Add new routes!

Plan ahead!

Response format matters!

```
[
    {
        "id": "padlok",
        "name:": "Padlok",
        "weight": 100
    },
    {
        "id": "sharepal",
        "name:": "SharePal",
        "weight": 200
    }
]
```

Returning an array is common

Plan ahead!

Response format matters!

```
{
    "items": [
        {
            "id": "padlok",
            "name:": "Padlok",
            "weight": 100
        },
        {
            "id": "sharepal",
            "name:": "SharePal",
            "weight": 200
        }
    ]
}
```

Returning an object is better

Plan ahead!

Response format matters!

```
{
    "items": [
        {
            "id": "padlok",
            "name:": "Padlok",
            "weight": 100
        },
        {
            "id": "sharepal",
            "name:": "SharePal",
            "weight": 200
        }
    ],
    "count": 2
}
```

Adding new properties allows more versatility

Introduce a container!

```
struct List<T: Content>: Content {
    let items: [T]
}
```

Introduce a container!

```swift
struct List<T: Content>: Content {
    let items: [T]
}

extension List: RandomAccessCollection {
    // ...
}

extension List: ExpressibleByArrayLiteral {
    // ...
}
```

Add your app version in Request Header

App side

```
request.setValue("1.0.0", forHTTPHeaderField: "App-Version")
```

Add your app version in Request Header

App side

```
request.setValue("1.0.0", forHTTPHeaderField: "App-Version")
```

Server side

```
if req.headers.appVersion >= "1.2.0" {
    // Something new
} else {
    // Backward-compatible something
}
```

Add your app version in Request Header

App side

```
request.setValue("1.0.0", forHTTPHeaderField: "App-Version")
```

Server side

```
if req.headers.appVersion >= "1.2.0" {
    // Something new
} else {
    // Backward-compatible something
}

struct SemanticVersion: Comparable, Equatable, ExpressibleByStringLiteral { /* ... */ }
extension HTTPHeaders {
    var appVersion: SemanticVersion? { /* ... */ }
}
```

New story!

Backend APIs: Constant evolution but tethered to the past

Dean's blog

Sharing insights and opinions on software development and computer science.

https://blog.thomasdurand.fr/story/2024-05-12-backend-api-tethered-to-the-past/

Security concerns

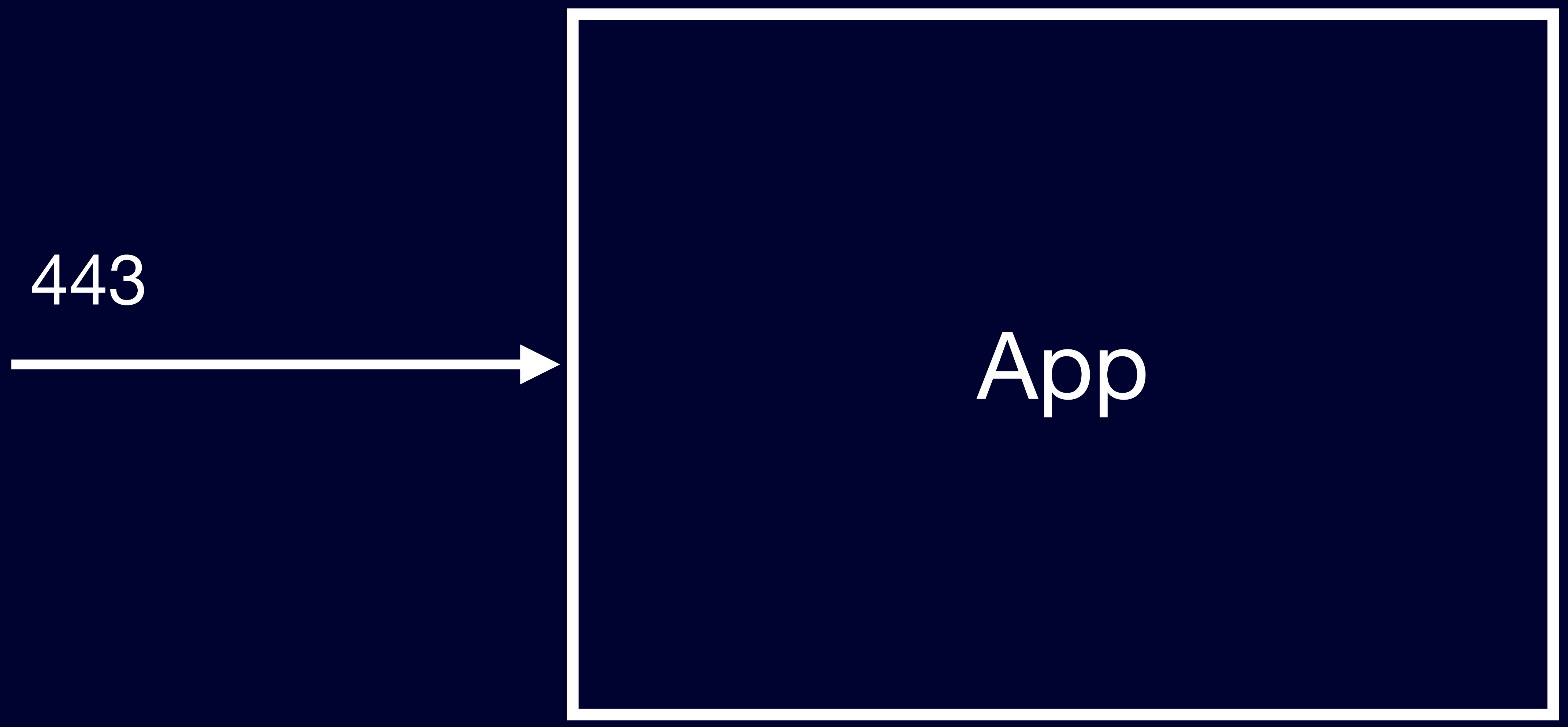
Use TLS!

Use Transport Layer Security!

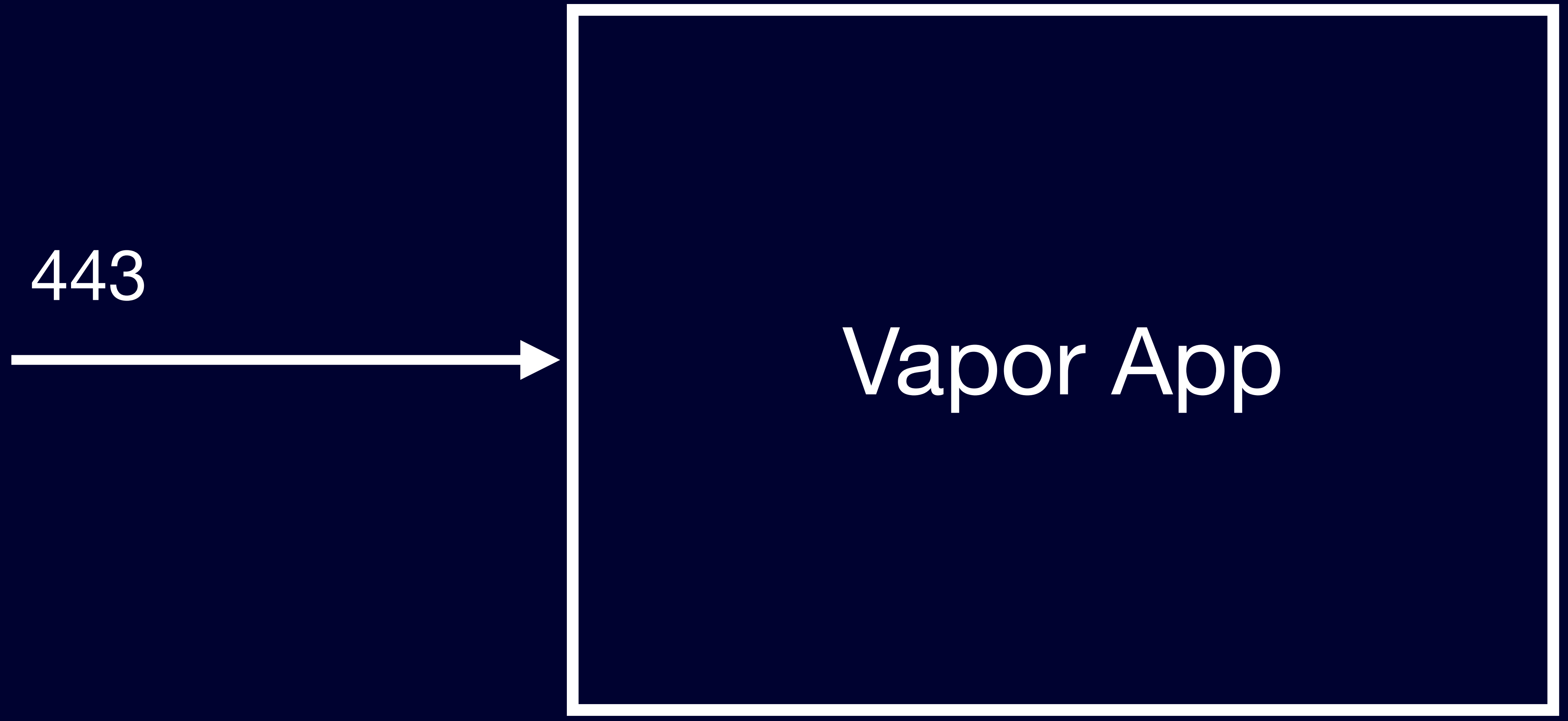
Use Transport Layer Security!

https://

TLS uses 443 port by default



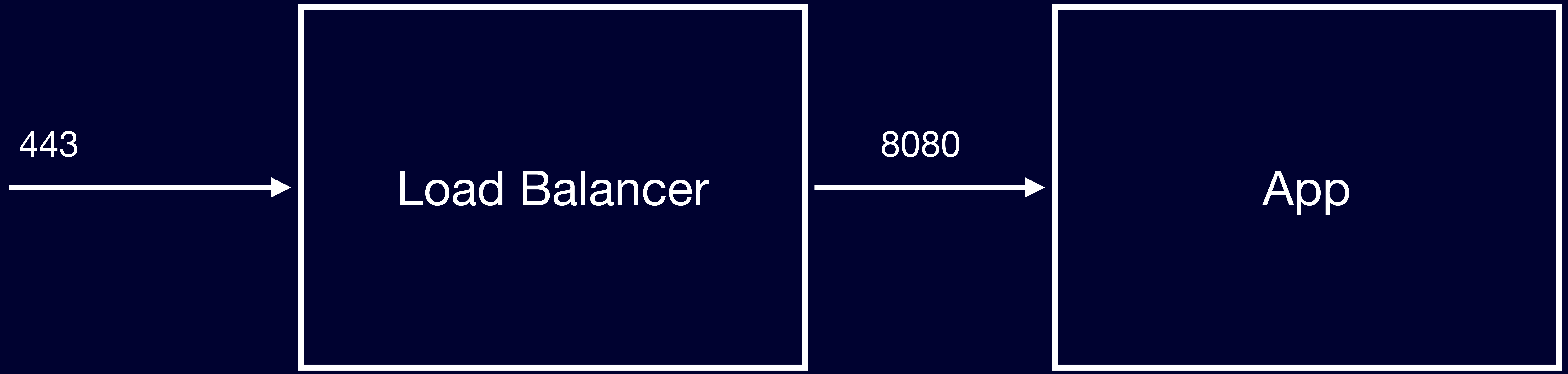
Vapor supports TLS



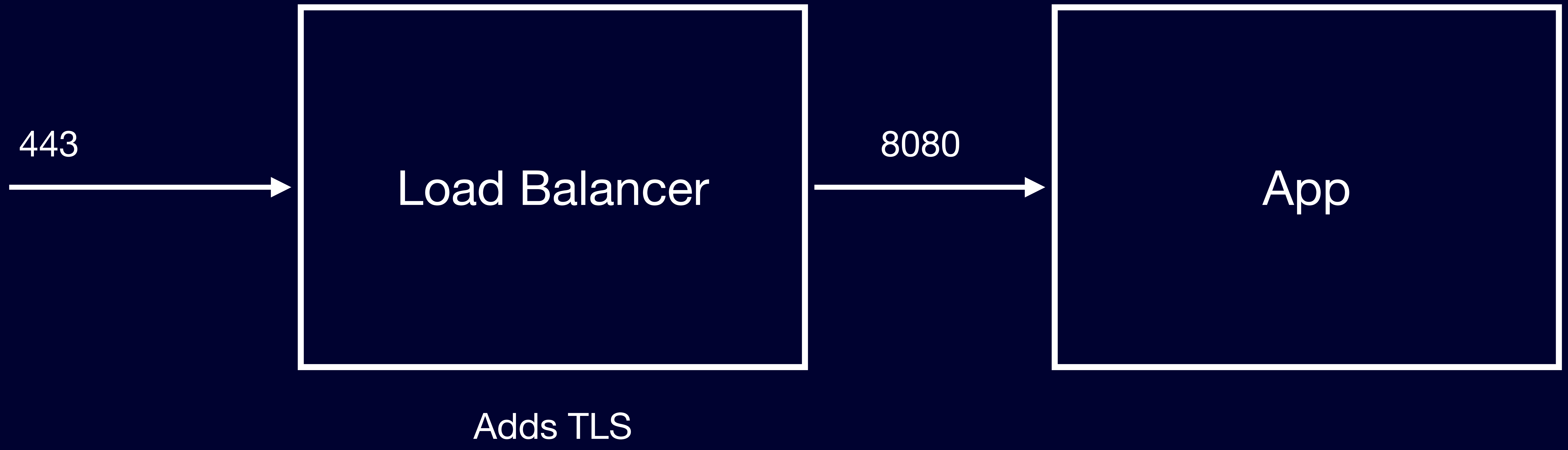
```
// Enable TLS over 443
app.http.server.configuration.port = 443
app.http.server.configuration.tlsConfiguration = .makeServerConfiguration(
    certificateChain: try NIOSSLCertificate.fromPEMFile("/path/to/cert.pem").map { .certificate($0) },
    privateKey: .file("/path/to/key.pem")
)
```

https://docs.vapor.codes/advanced/server/#tls

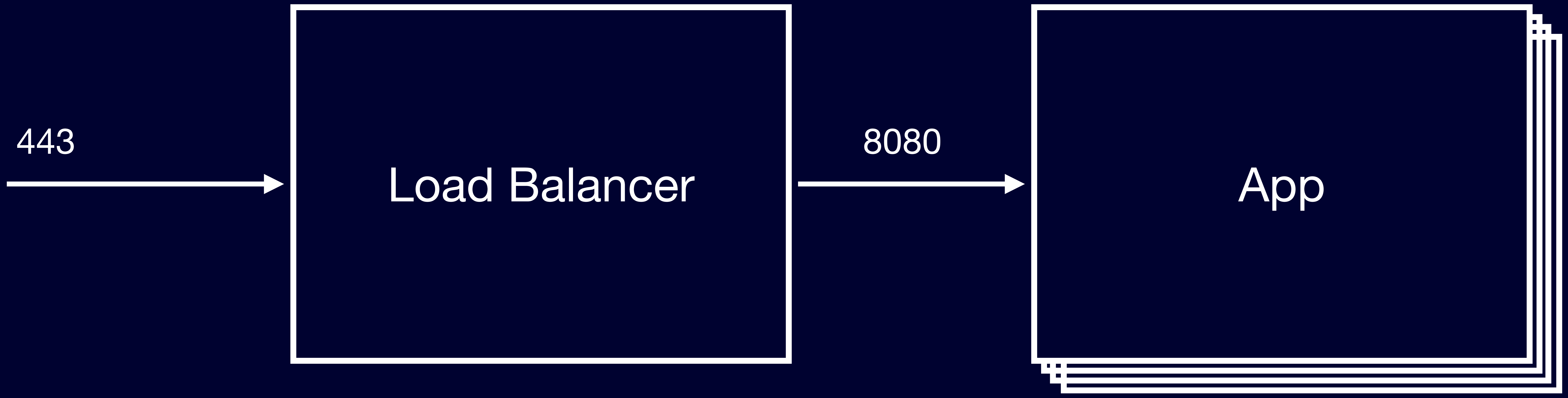
Delegate TLS outside is preferred!



Delegate TLS outside is preferred!



Delegate TLS outside is preferred!



Adds TLS

Get your response headers right!

```swift
// Add API security headers
let securityHeadersFactory = SecurityHeadersFactory.api()

app.middleware.use(securityHeadersFactory.build())
```

https://github.com/brokenhandsio/VaporSecurityHeaders.git

Get your response headers right!

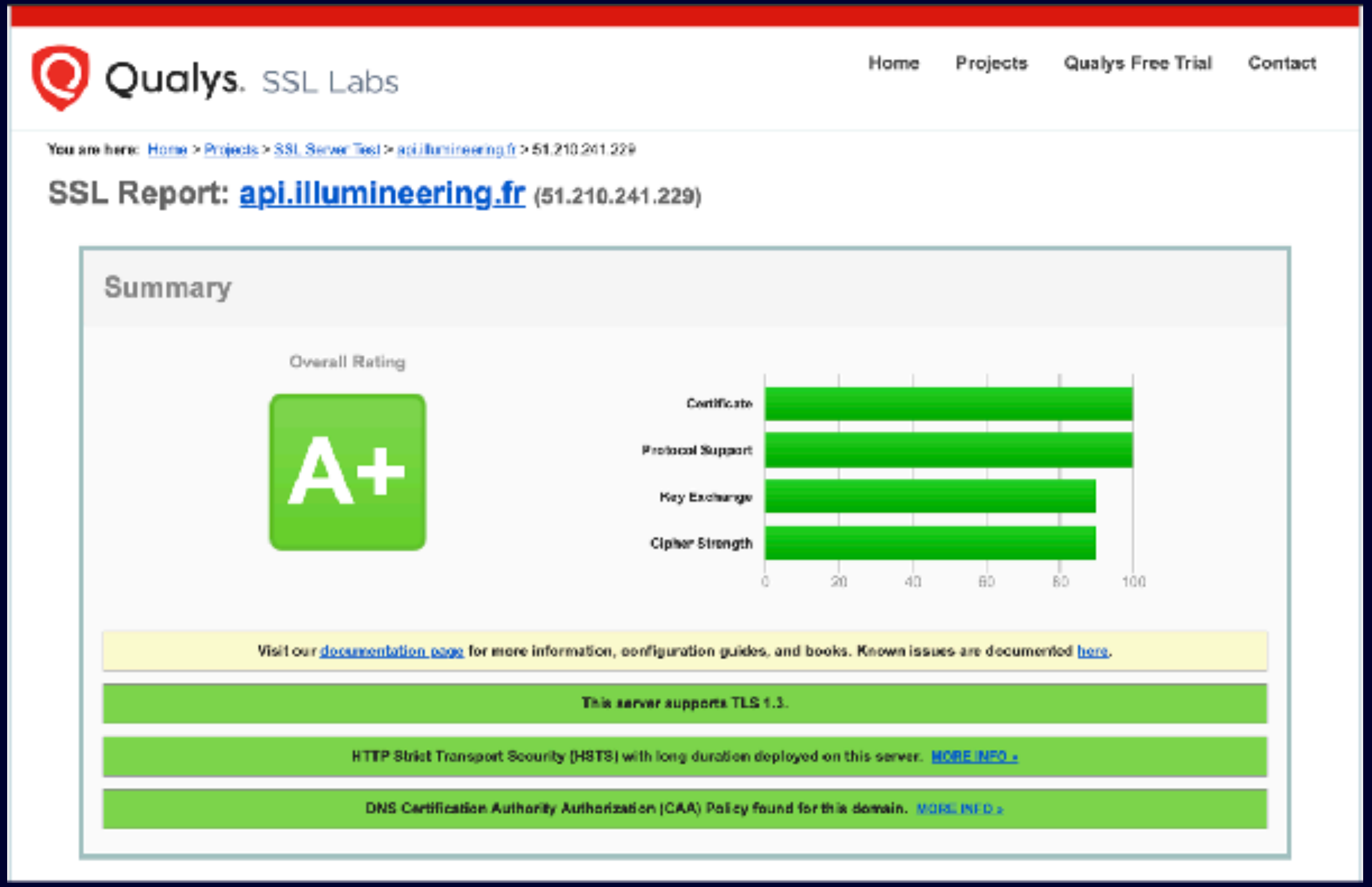
```
// Add HSTS if Vapor is responsible for TLS
securityHeadersFactory.with(strictTransportSecurity:
StrictTransportSecurityConfiguration())
```

https://github.com/brokenhandsio/VaporSecurityHeaders.git

Test TLS and headers!

https://www.ssllabs.com/ssltest/

https://securityheaders.com/



You are here: Home > Projects > SSL Server Test > api.illumineering.fr > 51.210.241.229

SSL Report: api.illumineering.fr (51.210.241.229)

Summary

Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

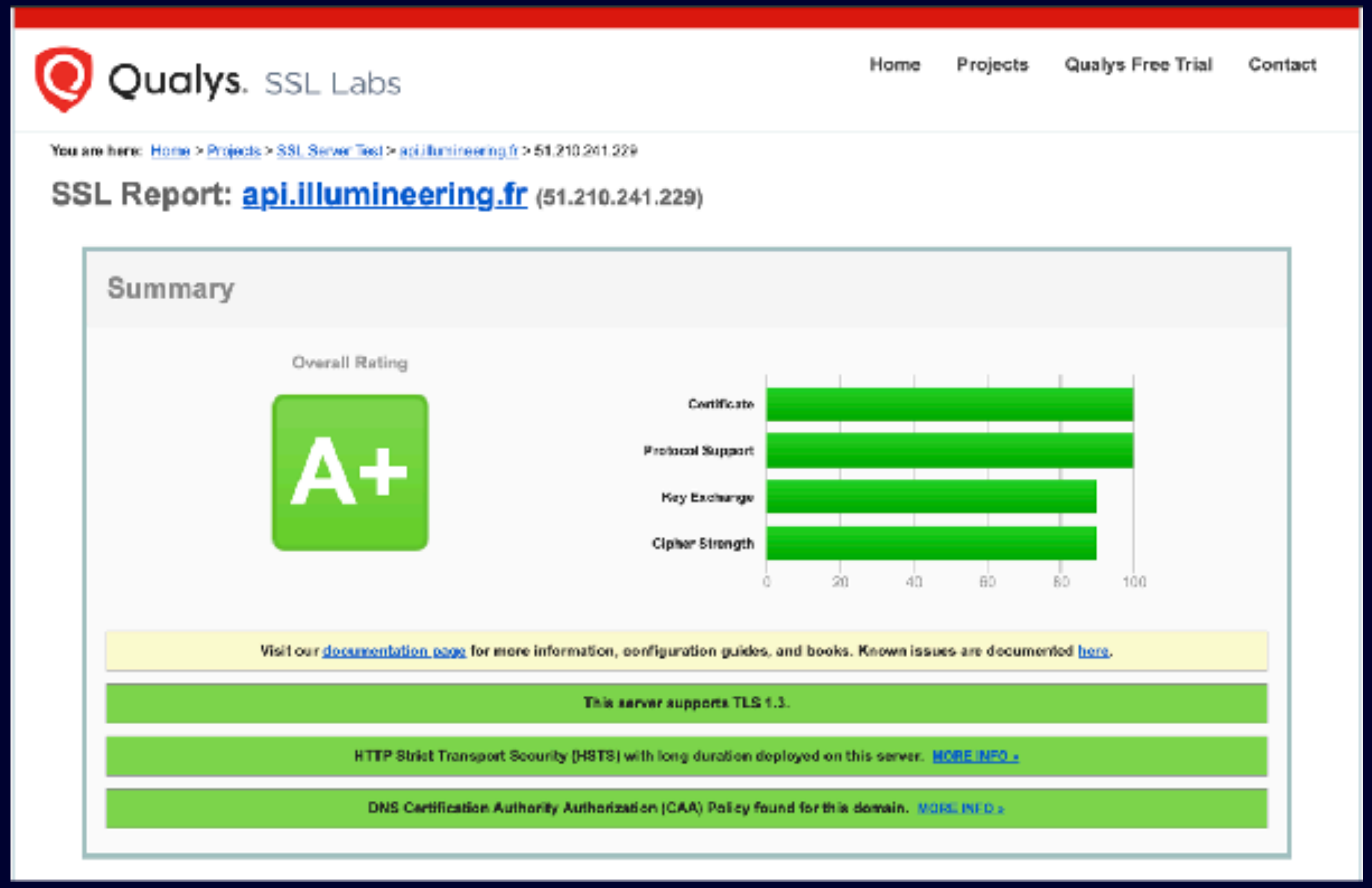
This server supports TLS 1.3.

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. MORE INFO »

DNS Certification Authority Authorization (CAA) Policy found for this domain. MORE INFO »

Test TLS and headers!

https://www.ssllabs.com/ssltest/


Qualys. SSL Labs

Home Projects Qualys Free Trial Contact

You are here: Home > Projects > SSL Server Test > api.illumineering.fr > 51.210.241.229

SSL Report: api.illumineering.fr (51.210.241.229)

Summary

Overall Rating

A+

Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

This server supports TLS 1.3.

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. MORE INFO »

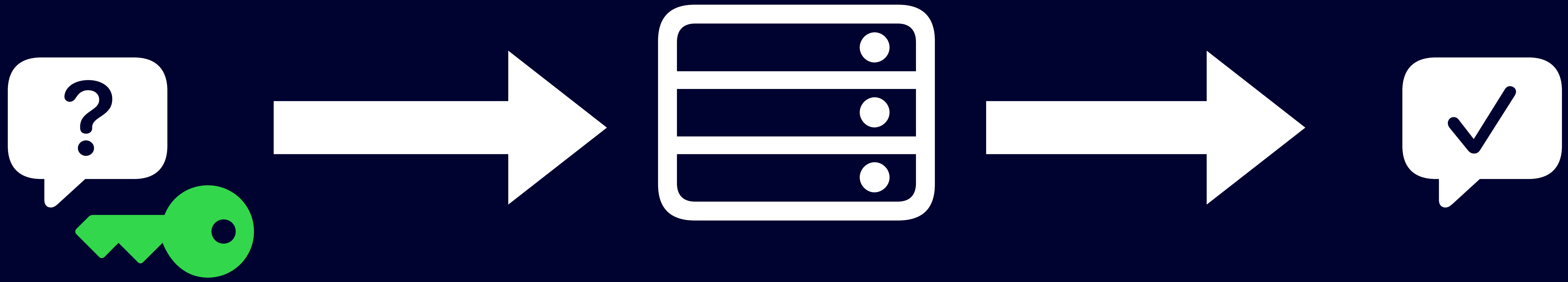
DNS Certification Authority Authorization (CAA) Policy found for this domain. MORE INFO »

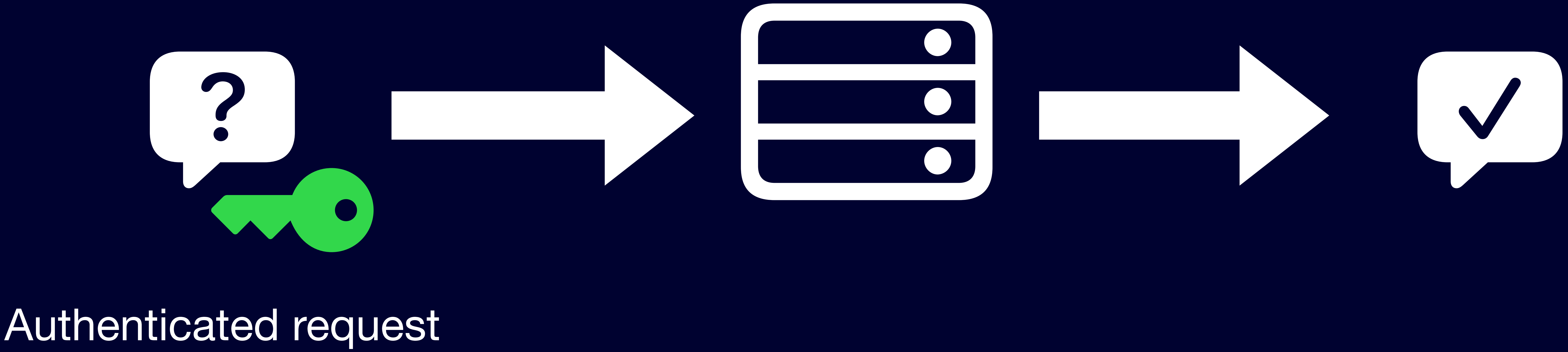
Not all headers matters for an API

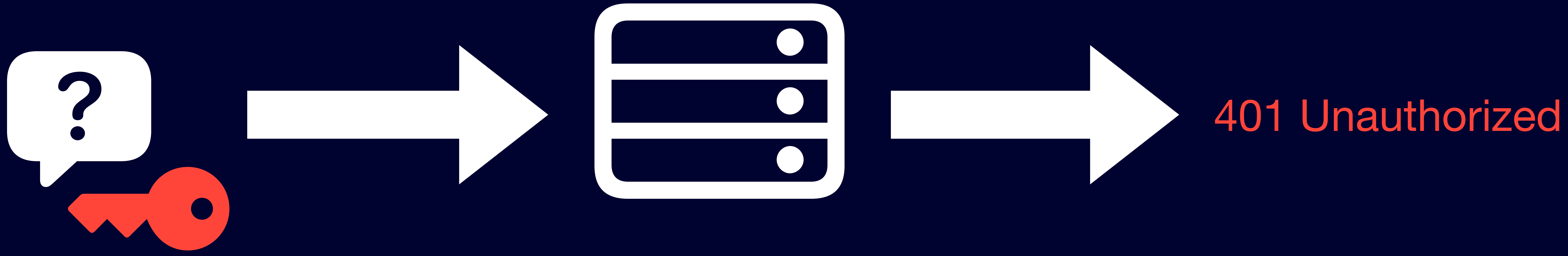
Validate user input!

https://docs.vapor.codes/basics/validation/

Authenticate your requests




Authenticated request



Unauthenticated request

How to get that secret?

How to get that secret?

Authenticate!

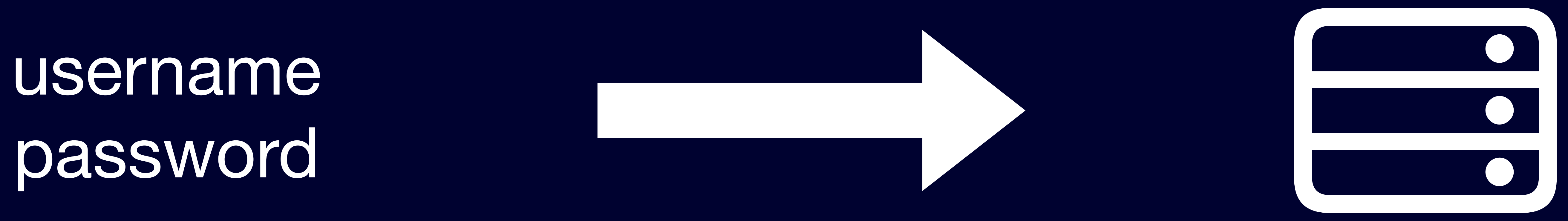
What are you authenticating?

User

App

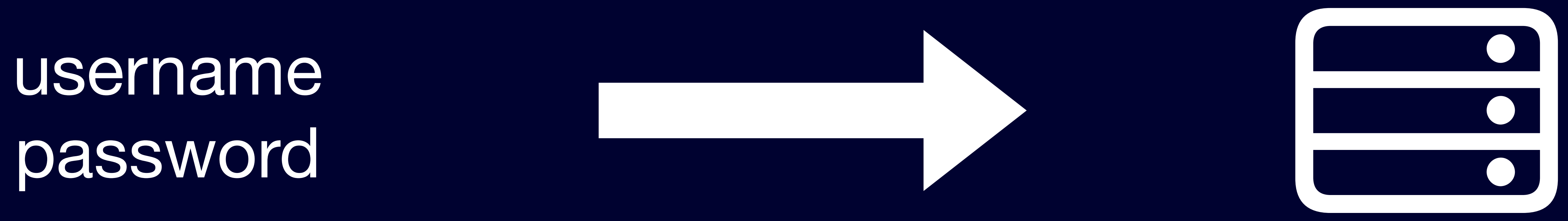
Authenticate as a user

Old school password



Authenticate as a user

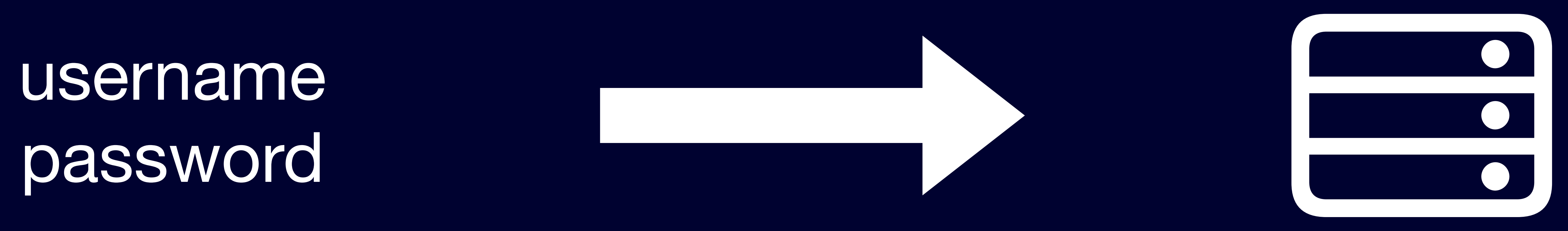
Old school password



Do not store password in plain text (please?)

Authenticate as a user

Old school password



Do not store password in plain text (please?)

https://docs.vapor.codes/security/passwords/

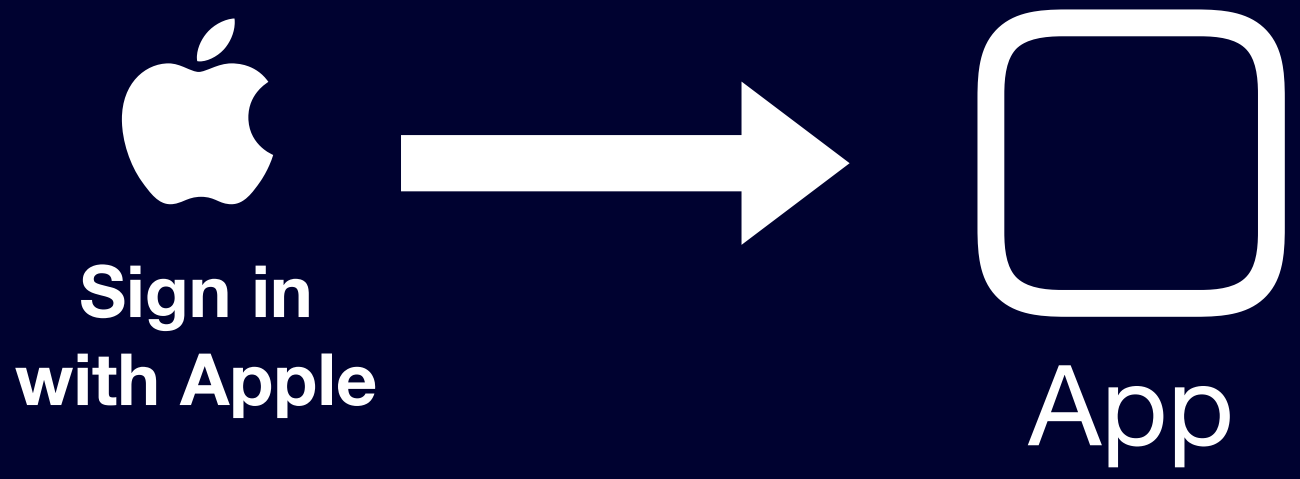
Authenticate as a user

Sign in with Apple

Sign in
with Apple

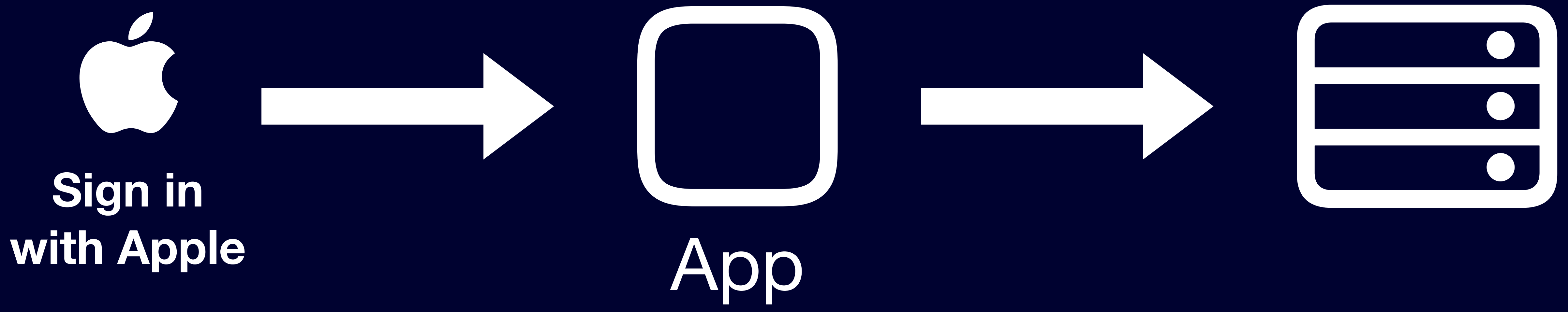
Authenticate as a user

Sign in with Apple



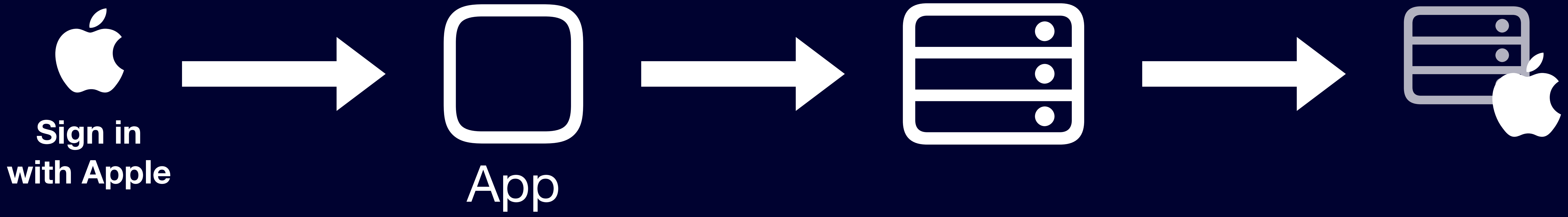
Authenticate as a user

Sign in with Apple



Authenticate as a user

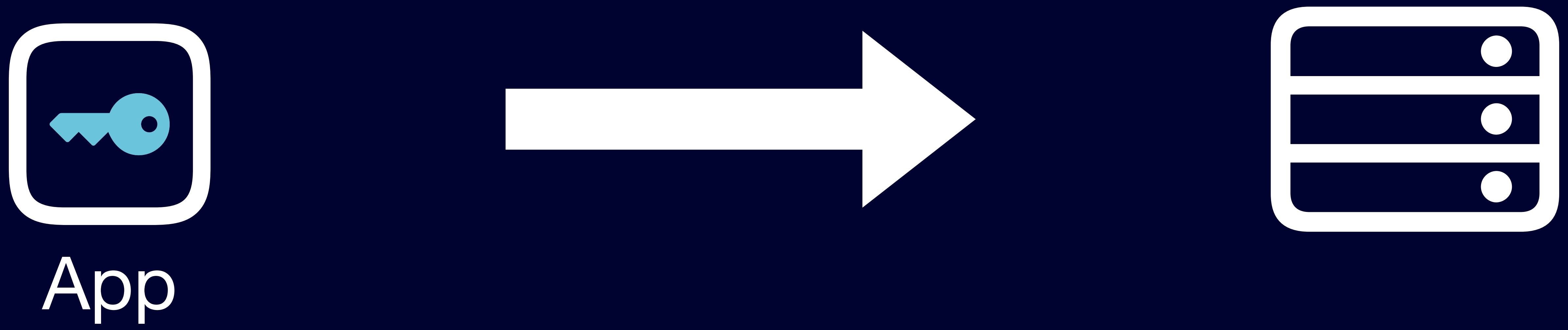
Sign in with Apple



Authenticate as an app

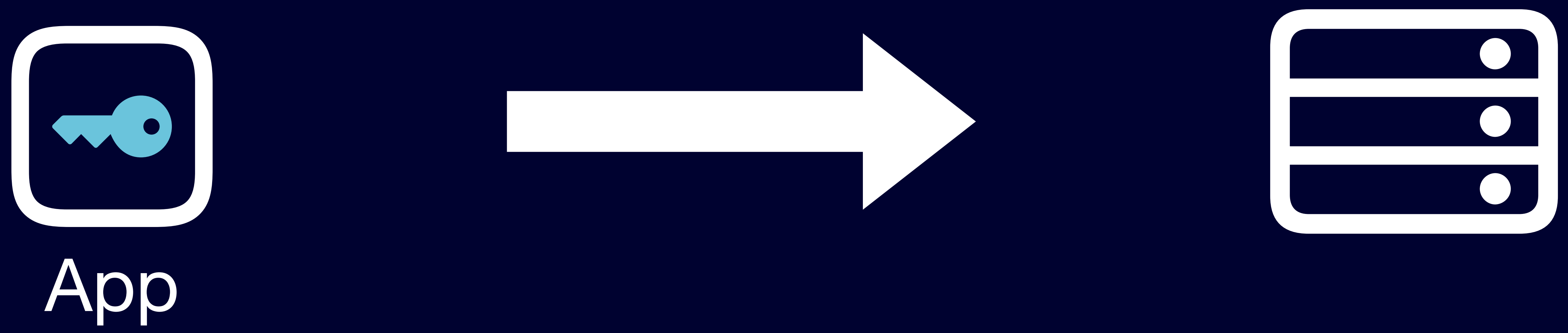
Authenticate as an app

Embed the key within your app?



Authenticate as an app

Embed the key within your app?



API Key

No.

No.

Api keys are NOT for client side

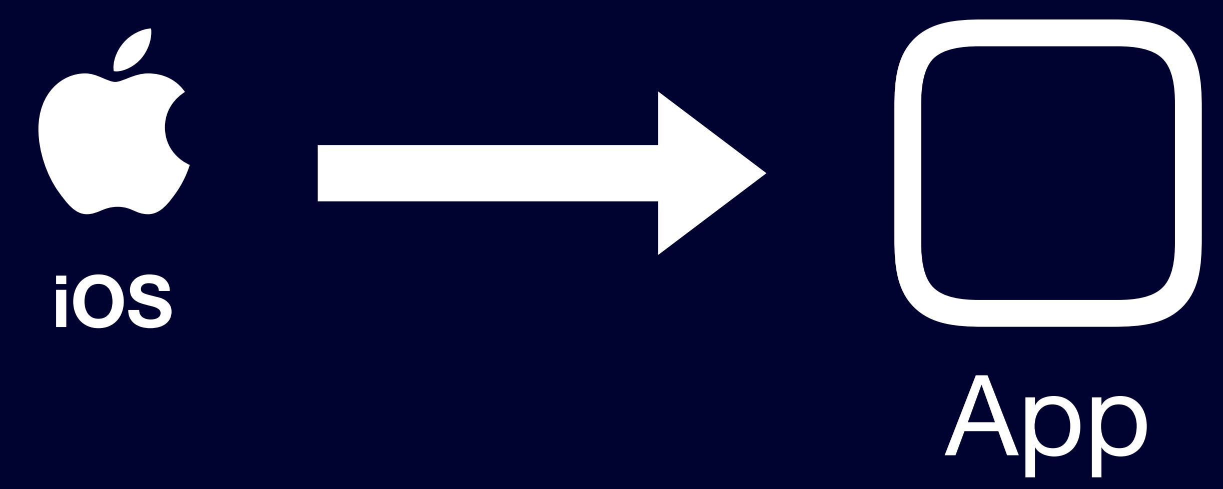
Authenticate your app

Use App Attest!

iOS

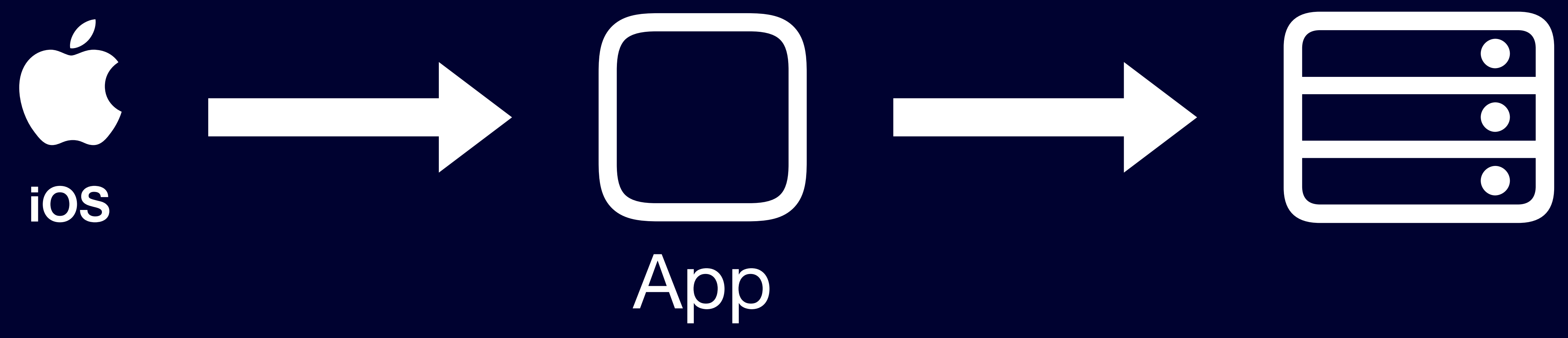
Authenticate your app

Use App Attest!



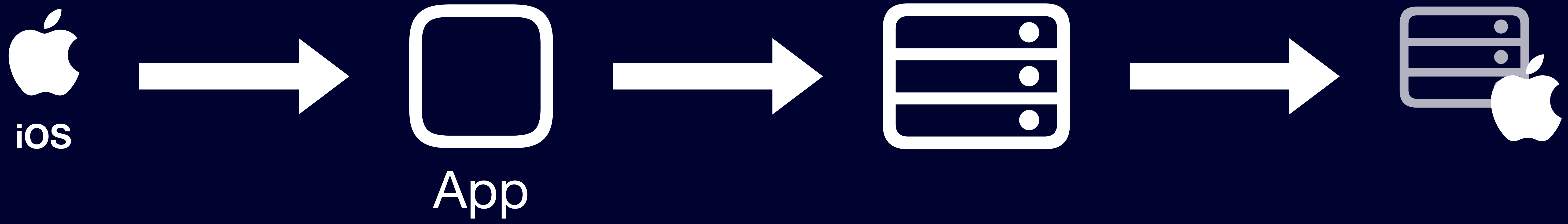
Authenticate your app

Use App Attest!



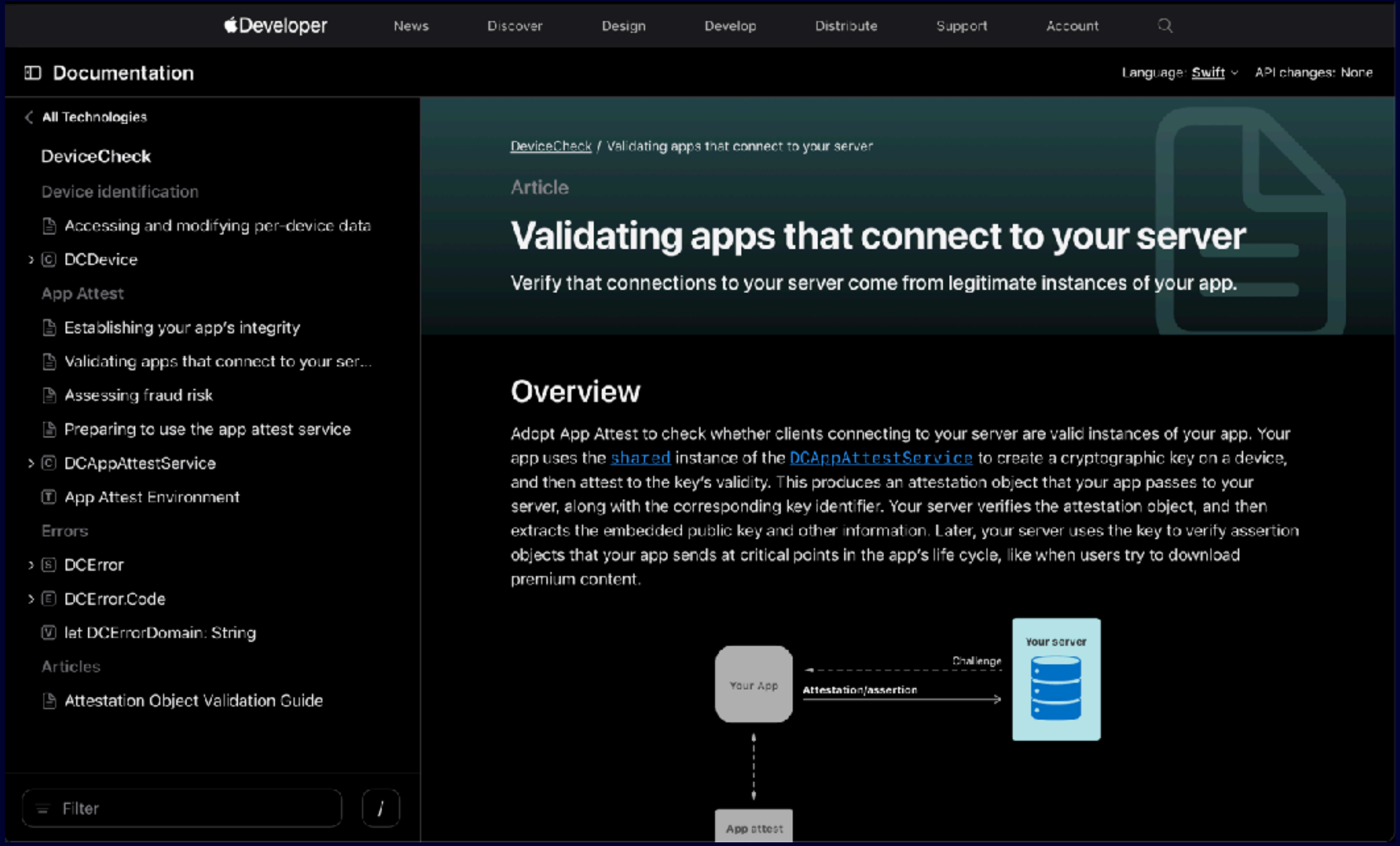
Authenticate your app

Use App Attest!



Authenticate your app

Use App Attest!



Developer News Discover Design Develop Distribute Support Account

Documentation

Language: Swift API changes: None

All Technologies

DeviceCheck

Device identification

Accessing and modifying per-device data

DCDevice

App Attest

Establishing your app's integrity

Validating apps that connect to your ser...

Assessing fraud risk

Preparing to use the app attest service

DCAppAttestService

App Attest Environment

Errors

DCError

DCError.Code

let DCErrorDomain: String

Articles

Attestation Object Validation Guide

Filter

DeviceCheck / Validating apps that connect to your server

Article

Validating apps that connect to your server

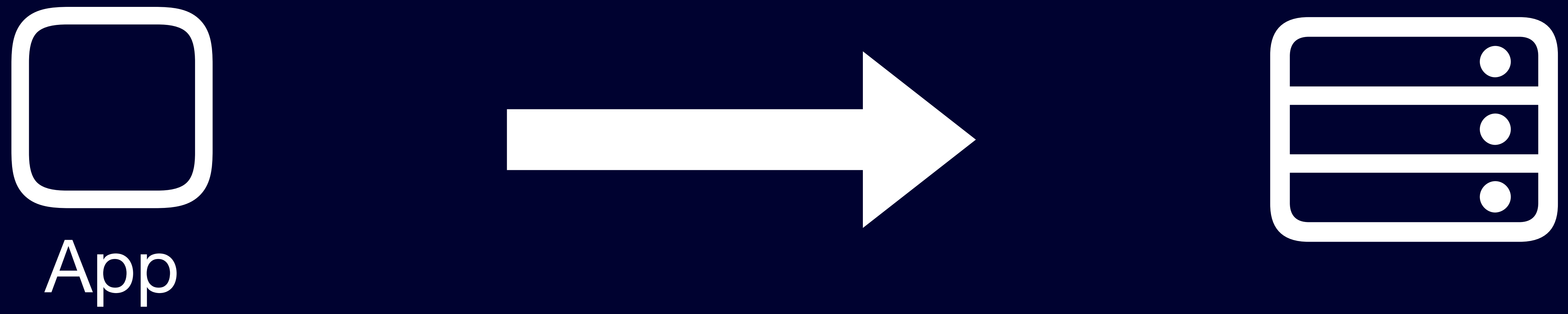
Verify that connections to your server come from legitimate instances of your app.

Overview

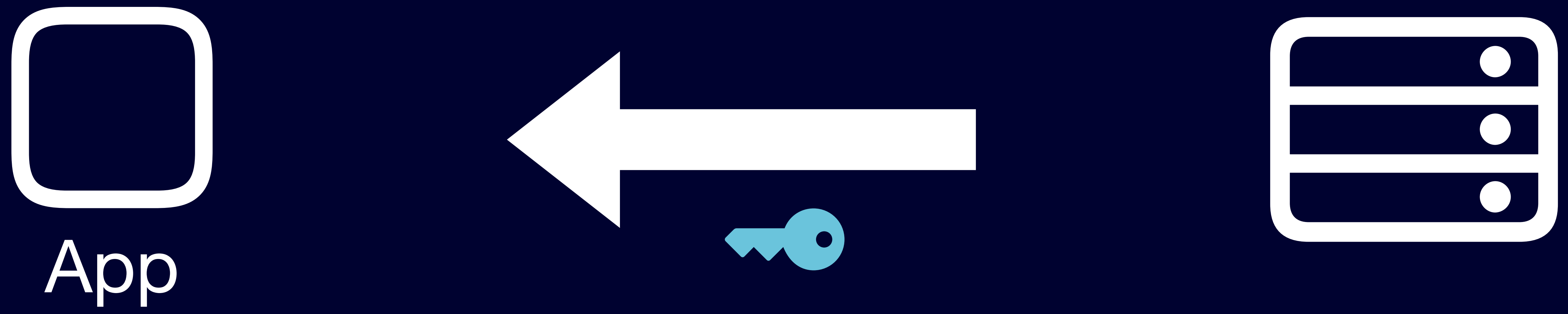
Adopt App Attest to check whether clients connecting to your server are valid instances of your app. Your app uses the shared instance of the DCAppAttestService to create a cryptographic key on a device, and then attest to the key's validity. This produces an attestation object that your app passes to your server, along with the corresponding key identifier. Your server verifies the attestation object, and then extracts the embedded public key and other information. Later, your server uses the key to verify assertion objects that your app sends at critical points in the app's life cycle, like when users try to download premium content.

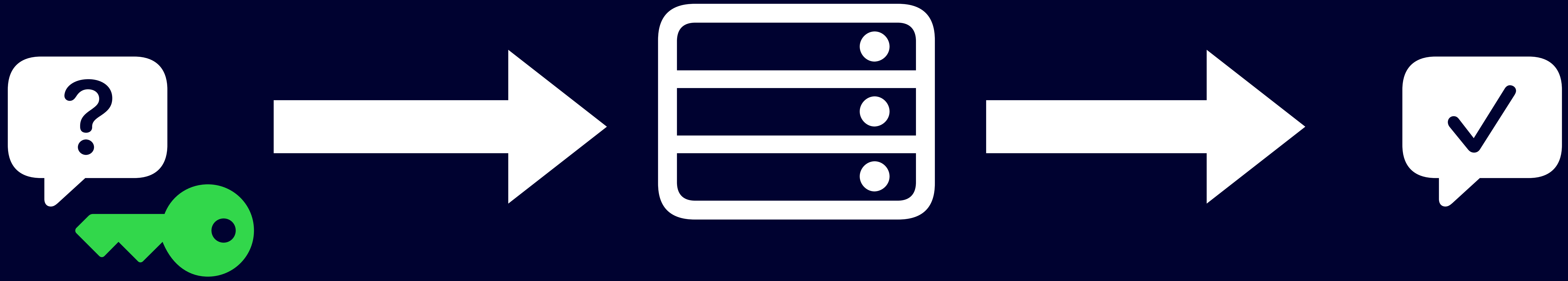
https://developer.apple.com/documentation/devicecheck/validating-apps-that-connect-to-your-server

Authentication ceremony



Authentication ceremony





Authenticated request

What kind of secret do you send back?

What kind of secret do you send back?

Anything random enough

Store a token in DB

Generate random token

```
PuTcER23gNYDjx5iLdriawUAAApxPzcF
```

Store a token in DB

Generate random token

`PuTcER23gNYDjx5iLdriawUAAApxPzcF`

Store it

Database

Store a token in DB

Generate random token

`PuTcER23gNYDjx5iLdriawUAAApxPzcF`

Store it

Database

Send it back

Send the token with the request

App side

```
request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
```

Check a token from DB

```swift
struct AuthenticationMiddleware: Middleware {
    func respond(to request: Request, chainingTo next: any Responder) async throws -> Response {
        // Do anything with the request here
        let response = try await next.respond(to: request)
        // Do anything with the response here
        return response
    }
}
```

Check a token from DB

```swift
struct AuthenticationMiddleware: Middleware {
    func respond(to request: Request, chainingTo next: any Responder) async throws -> Response {
        // Do anything with the request here
        let response = try await next.respond(to: request)
        // Do anything with the response here
        return response
    }
}
```

```swift
app.middleware.use(AuthenticationMiddleware())
```

Check a token from DB

```swift
struct AuthenticationMiddleware: Middleware {
    func respond(to request: Request, chainingTo next: any Responder) async throws -> Response {
        // Do anything with the request here
        return try await next.respond(to: request)
    }
}
```

Check a token from DB

```swift
struct AuthenticationMiddleware: Middleware {
    func respond(to request: Request, chainingTo next: any Responder) async throws -> Response {
        guard let authorization = request.headers["Authorization"].first else {
            throw Abort(.unauthorized)
        }
        guard authorization.starts(with: "Bearer ") else {
            throw Abort(.unauthorized)
        }
        let token = String(authorization.dropFirst(7))

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

Check a token from DB

```swift
struct AuthenticationMiddleware: Middleware {
    func respond(to request: Request, chainingTo next: any Responder) async throws -> Response {
        guard let authorization = request.headers["Authorization"].first else {
            throw Abort(.unauthorized)
        }
        guard authorization.starts(with: "Bearer ") else {
            throw Abort(.unauthorized)
        }
        let token = String(authorization.dropFirst(7))
        try await check(token: token, for: request)
        return try await next.respond(to: request)
    }

    private func check(token: String, for request: Request) async throws {
        // Check token validity in database here!
    }
}
```

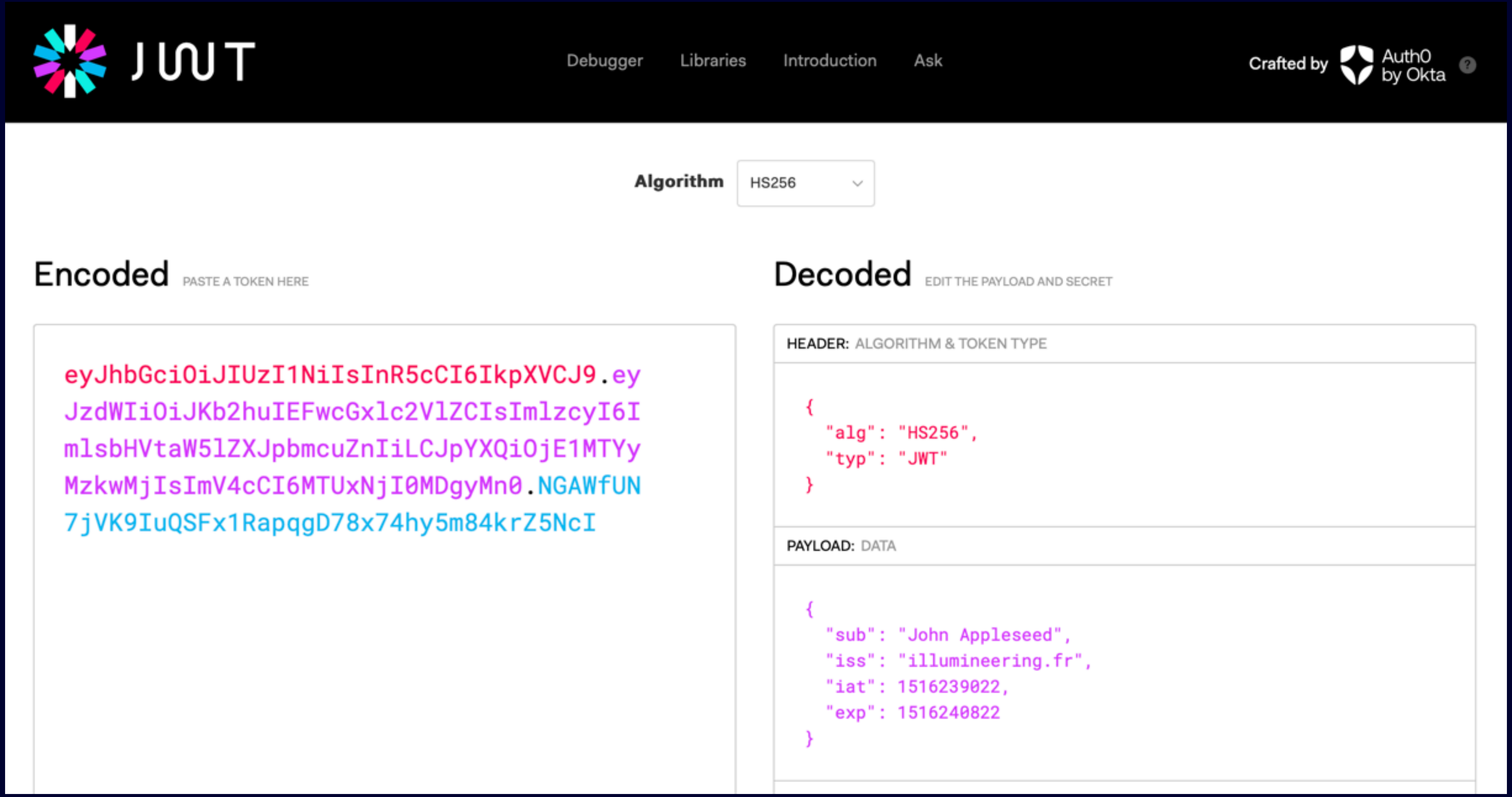
Issue a signed token

The specific case of JWT

Have user info in a token, along with a cryptographic signature

Issue a signed token

The specific case of JWT



Algorithm HS256

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiJKb2huIEFwcGxlc2VlZCIsImlzcyI6I
mlsbHVtaW5lZXJpbmcuZnIiLCJpYXQiOjE1MTYy
MzkwMjIsImV4cCI6MTUxNjI0MDgyMn0.NGAWfUN
7jVK9IuQSFx1RapqgD78x74hy5m84krZ5NcI
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "John Appleseed",
  "iss": "illumineering.fr",
  "iat": 1516239022,
  "exp": 1516240822
}
```

https://jwt.io/

Issue a signed token

```
await app.jwt.keys.add(hmac: "secret", digestAlgorithm: .sha256)
```

Issue a signed token

```swift
await app.jwt.keys.add(hmac: "secret", digestAlgorithm: .sha256)

struct AuthenticatedPayload: JWTPayload {
    let sub: SubjectClaim
    let exp: ExpirationClaim
    let iss: IssuerClaim
    let iat: IssuedAtClaim

    init(userID: SubjectClaim) {
        self.sub = userID
        self.exp = .init(value: .now.addingTimeInterval(1800))
        self.iss = "illumineering.fr"
        self.iat = .init(value: .now)
    }

    func verify(using algorithm: some JWTKit.JWTAlgorithm) async throws {
        try self.exp.verifyNotExpired()
    }
}
```

Issue a signed token

```swift
await app.jwt.keys.add(hmac: "secret", digestAlgorithm: .sha256)

struct AuthenticatedPayload: JWTPayload {
    let sub: SubjectClaim
    let exp: ExpirationClaim
    let iss: IssuerClaim
    let iat: IssuedAtClaim

    init(userID: SubjectClaim) {
        self.sub = userID
        self.exp = .init(value: .now.addingTimeInterval(1800))
        self.iss = "illumineering.fr"
        self.iat = .init(value: .now)
    }

    func verify(using algorithm: some JWTKit.JWTAlgorithm) async throws {
        try self.exp.verifyNotExpired()
    }
}

let payload = AuthenticatedPayload(userID: "John Appleseed")
let token = try await request.jwt.sign(payload)
```

Check a signed token

```swift
struct AuthenticationMiddleware: Middleware {
    func respond(to request: Request, chainingTo next: any Responder) async throws -> Response {
        guard let authorization = request.headers["Authorization"].first else {
            throw Abort(.unauthorized)
        }
        guard authorization.starts(with: "Bearer ") else {
            throw Abort(.unauthorized)
        }
        let token = String(authorization.dropFirst(7))
        try await check(token: token, for: request)
        return try await next.respond(to: request)
    }

    private func check(token: String, for request: Request) async throws {
        // Check token validity in database here!
    }
}
```

Check a signed token

```swift
struct AuthenticationMiddleware: Middleware {
    func respond(to request: Request, chainingTo next: any Responder) async throws -> Response {
        try await request.jwt.verify(as: AuthenticatedPayload.self)
        return try await next.respond(to: request)
    }
}
```

https://docs.vapor.codes/security/jwt/

Where to go from there?

Where to go from there?

All the topics you might look into

Use a database

https://docs.vapor.codes/fluent/overview/

Where to go from there?

All the topics you might look into

Use a key/value cache

https://docs.vapor.codes/redis/overview/

Where to go from there?

All the topics you might look into

Deployment!

Heroku: https://developer.apple.com/wwdc22/110360

Fly: https://docs.vapor.codes/deploy/fly/

Leverage the community!

Share what you learn!

Thank you!

Slides:

https://thomasdurand.fr/serverside-swift-2024

https://thomasdurand.fr

@deanatoire@mastodon.social

@deantoire@threads.net

@deanatoire@twitter.com