# WELCOME TO VAPOR LONDON MARCH!

# WE'RE HIRING!
# TIM.CONDON@BBC.CO.UK

ServerSide.swift

# Server-Side Swift Conference

**12th-14th September 2018. Berlin, Germany**

The best way to keep posted about the conference is by signing up below!
Next newsletter will contain everything about sponsor levels, ticket prices, schedule and more!

Email

Notify Me

## The Conference

ServerSide.swift is a framework-independent conference, where we will learn and share on a number of different related topics. The conference is aimed at being a non-profit conference and solely run for the love of server-side Swift.

## Where and When

The conference will take place in awesome **Berlin, Germany** and last for 3 days from **12th-14th September 2018**!

## Speakers

If you are interested in speaking you can let us know here! We are planning a mix of talks and workshops both for beginner and advanced server-side Swift developers and want to hear from all of the frameworks!

## The Team

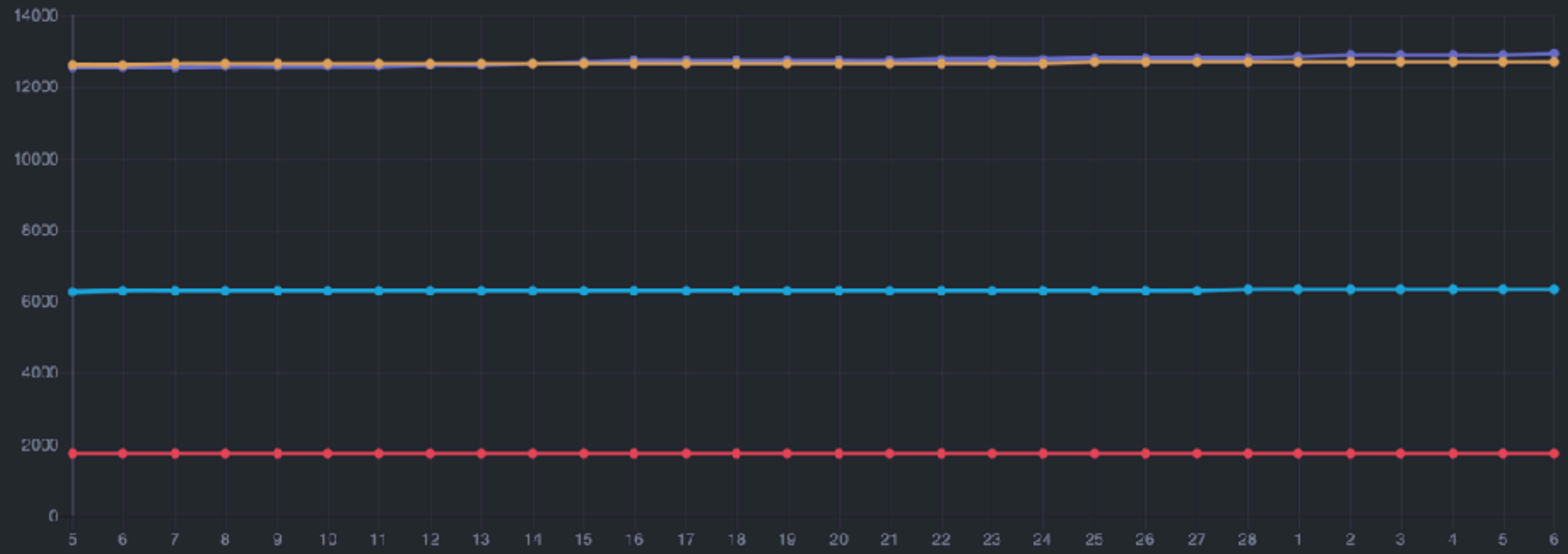
Collaboration between Martin J. Lasek (Berlin - DE), Tim Condon (Manchester - UK), Oliver Wang Hansen (Copenhagen - DK) and Steffen D. Sommer (Copenhagen - DK)

## Sponsors

## Want to help?

serversideswift.racing

# serversideswift.racing




**Feb. / Mar.**

STARS PER DAY

| 11.33 | 3.96 | 2.17 | 0.32 |
|---|---|---|---|
| vapor | perfect | kitura | zewo |

# WHAT'S NEW IN VAPOR 3


## TIM CONDON

**Broken Hands**
**BBC**

# INTRODUCTION

- @0xTim - Twitter/Slack/Github
- Varied background, currently at the BBC
- Created SteamPress, Vapor Security Headers, Vapor OAuth
- @brokenhandsio - Twitter/Github
- Instructor @ raywenderlich.com
- Currently writing Server Side Swift with Vapor

# VAPOR 3


KEEP UP WITH THE
VAPOR BETAS THEY SAID
IT WILL BE
FUN THEY SAID
imgflip.com

# VAPOR 3

- Frustrating, exciting, world-changing, head-banging-on-the-wall
- Going back to Vapor 2 is like writing Javascript after using a compiler....
- It's a complete redesign from the ground up

# VAPOR EVOLUTION

- Follows similar pattern to Swift
- Vapor 1 - new and exciting!
- Vapor 2 - dog fooding
- Vapor 3 - laying the foundation for the future
- Will be a painful but necessary transition!

# WHAT'S NEW

# SWIFT 4.1

- Introduces conditional conformance
- Vapor 3 ***requires*** Swift 4.1
- Affects the release schedule (more later)

# CODABLE

# CODABLE

- Vapor 2 has limited `Codable` support
- Vapor 3 embraces it...
- ....for everything!

# DEALING WITH JSON IN VAPOR 2

```
extension Post: JSONConvertible {
    convenience init(json: JSON) throws {
        self.init(
            content: try json.get(Post.Keys.content)
        )
    }

    func makeJSON() throws -> JSON {
        var json = JSON()
        try json.set(Post.Keys.id, id)
        try json.set(Post.Keys.content, content)
        return json
    }
}

extension Request {
    func post() throws -> Post {
        guard let json = json else { throw Abort.badRequest }
        return try Post(json: json)
    }
}

func returnPost(_ req: Request) throws -> ResponseRepresentable {
  let post = try req.post()
  return post
}
```

# DEALING WITH JSON IN VAPOR 3

```
extension Post: Content {}

func returnPost(_ req: Request) throws -> Future<Post> {
  let post = try req.content.decode(Post.self)
  return post
}
```

# CODABLE

- Same for Node
- Same for Row
- Used with queries, forms, file uploads
- Leaf interacts with `Codable`
- RIP Node 2016-2018 👻 🎉

# MODELS

# MODELS

- No more `Storage`
- Much easier to conform classes to `Model`
- Even supports `structs`!
- Easy support for UUID IDs and different ID names
- Can be as simple as:

```
extension Post: PostgreSQLModel {}
```

# STEAMPRESS MODEL VAPOR 2

```swift
public final class BlogPost: Model {

    public struct Properties {
        public static let blogPostID = "id"
        public static let title = "title"
        public static let contents = "contents"
        public static let slugUrl = "slug_url"
        public static let published = "published"
        public static let created = "created"
        public static let lastEdited = "last_edited"
        public static let authorName = "author_name"
        public static let authorUsername = "author_username"
        public static let createdDate = "created_date"
        public static let createdDateIso8601 = "created_date_iso8601"
        public static let lastEditedDate = "last_edited_date"
        public static let lastEditedDateIso8601 = "last_edited_date_iso8601"
        public static let shortSnippet = "short_snippet"
        public static let longSnippet = "long_snippet"
        public static let tags = "tags"
    }

    static var postsPerPage = 10

    public let storage = Storage()

    public var title: String
    public var contents: String
    public var author: Identifier?
    public var created: Date
    public var lastEdited: Date?
    public var slugUrl: String
    public var published: Bool

    public init(title: String, contents: String, author: BlogUser, creationDate: Date, slugUrl: String,
        published: Bool, logger: LogProtocol? = nil) {
        self.title = title
        self.contents = contents
        self.author = author.id
        self.created = creationDate
        self.slugUrl = BlogPost.generateUniqueSlugUrl(from: slugUrl, logger: logger)
        self.lastEdited = nil
        self.published = published
    }

    public required init(row: Row) throws {
        title = try row.get(Properties.title)
        contents = try row.get(Properties.contents)
        author = try row.get(BlogUser.foreignIdKey)
        slugUrl = try row.get(Properties.slugUrl)
        published = try row.get(Properties.published)
        let createdTime: Double = try row.get(Properties.created)
        let lastEditedTime: Double? = try? row.get(Properties.lastEdited)

        created = Date(timeIntervalSince1970: createdTime)

        if let lastEditedTime = lastEditedTime {
            lastEdited = Date(timeIntervalSince1970: lastEditedTime)
        }
    }

    public func makeRow() throws -> Row {
        let createdTime = created.timeIntervalSince1970

        var row = Row()
        try row.set(Properties.title, title)
        try row.set(Properties.contents, contents)
        try row.set(BlogUser.foreignIdKey, author)
        try row.set(Properties.created, createdTime)
        try row.set(Properties.slugUrl, slugUrl)
        try row.set(Properties.published, published)
        try row.set(Properties.lastEdited, lastEdited?.timeIntervalSince1970)
        return row
    }
}

extension BlogPost: Parameterizable {}

// MARK: - Node

public enum BlogPostContext: Context {
    case all
    case shortSnippet
    case longSnippet
}

extension BlogPost: NodeRepresentable {
    public func makeNode(in context: Context?) throws -> Node {
        let createdTime = created.timeIntervalSince1970

        var node = Node([:], in: context)
        try node.set(Properties.blogPostID, id)
        try node.set(Properties.title, title)
        try node.set(Properties.contents, contents)
        try node.set(BlogUser.foreignIdKey, author)
        try node.set(Properties.created, createdTime)
        try node.set(Properties.slugUrl, slugUrl)
        try node.set(Properties.published, published)

        if let lastEdited = lastEdited {
            try node.set(Properties.lastEdited, lastEdited.timeIntervalSince1970)
        }

        guard let providedContext = context else {
            return node
        }

        if type(of: providedContext) != BlogPostContext.self {
            return node
        }

        let dateFormatter = DateFormatter()
        dateFormatter.timeZone = TimeZone(abbreviation: "UTC")
        dateFormatter.dateStyle = .full
        dateFormatter.timeStyle = .none
        let createdDate = dateFormatter.string(from: created)

        try node.set(Properties.authorName, postAuthor.get()?.name)
        try node.set(Properties.authorUsername, postAuthor.get()?.username)
        try node.set(Properties.createdDate, createdDate)

        switch providedContext {
        case BlogPostContext.shortSnippet:
            try node.set(Properties.shortSnippet, shortSnippet())
            break
        case BlogPostContext.longSnippet:
            try node.set(Properties.longSnippet, longSnippet())

            let allTags = try tags.all()
            if !allTags.isEmpty {
                try node.set(Properties.tags, allTags)
            }
            break
        case BlogPostContext.all:
            let allTags = try tags.all()

            if !allTags.isEmpty {
                try node.set(Properties.tags, allTags)
            }

            let iso8601Formatter = DateFormatter()
            iso8601Formatter.dateFormat = "yyyy-MM-dd'T'HH:mm:ssZ"
            iso8601Formatter.locale = Locale(identifier: "en_US_POSIX")
            iso8601Formatter.timeZone = TimeZone(secondsFromGMT: 0)

            try node.set(Properties.createdDateIso8601, iso8601Formatter.string(from: created))
```

# STEAMPRESS MODEL IN VAPOR 3

```
public final class BlogPost<DatabaseType>: Model where DatabaseType: QuerySupporting & SchemaSupporting & JoinSupporting {

    public typealias ID = Int
    public static var idKey: ReferenceWritableKeyPath<BlogPost<DatabaseType>, Int?> {
        return \BlogPost.blogID
    }
    public typealias Database = DatabaseType

    public var blogID: Int?
    public var title: String
    public var contents: String
    public var author: BlogUser<DatabaseType>.ID
    public var created: Date
    public var lastEdited: Date?
    public var slugUrl: String
    public var published: Bool

    public init(title: String, contents: String, author: BlogUser<DatabaseType>, creationDate: Date, slugUrl: String, published: Bool) throws {
        self.title = title
        self.contents = contents
        self.author = try author.requireID()
        self.created = creationDate
        self.slugUrl = title
        self.lastEdited = nil
        self.published = published
    }
}

extension BlogPost: Migration {}
```

# ASYNC

# ASYNC AND FUTURES

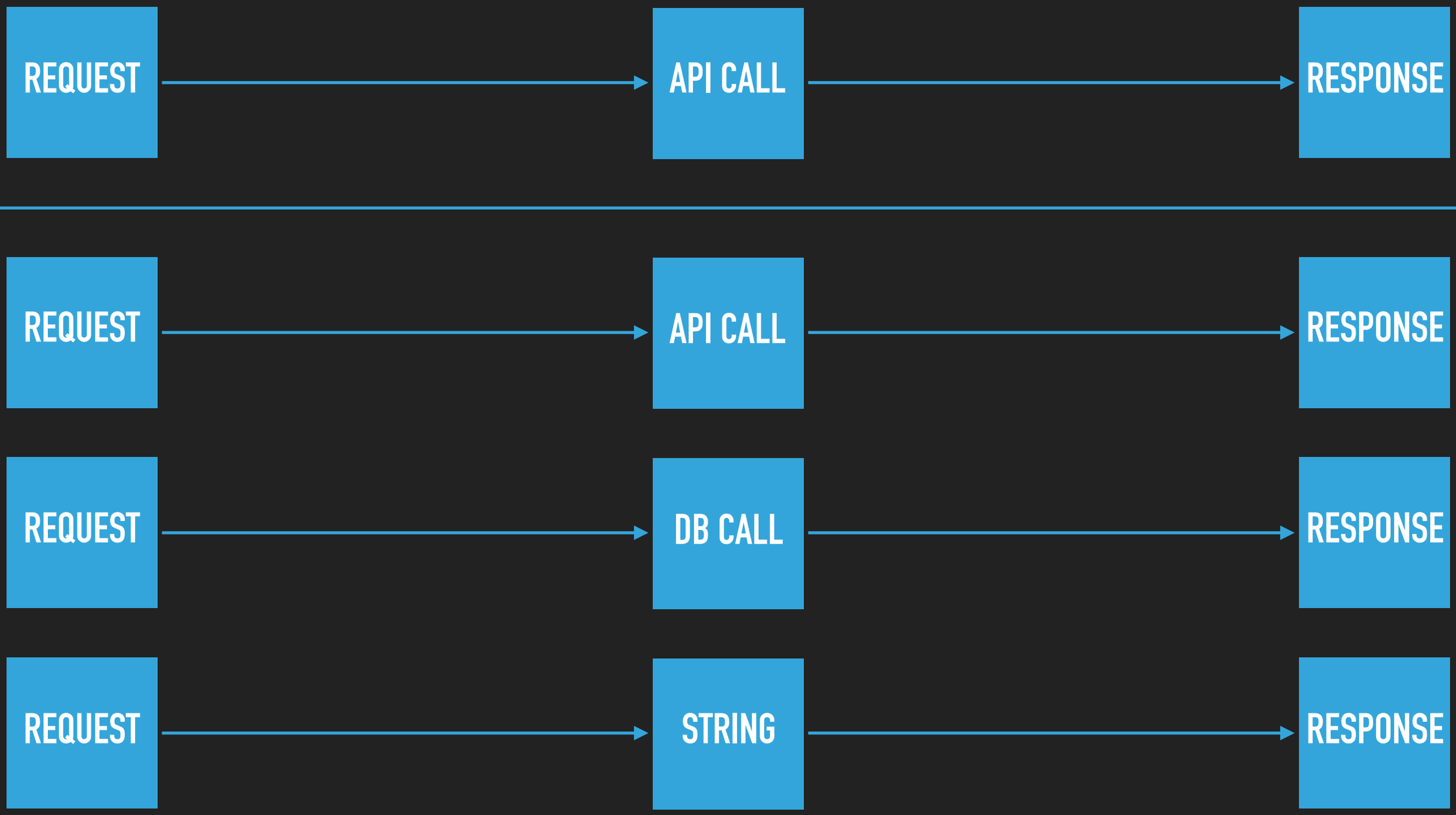
- Biggest change in Vapor 3
- Moves Vapor from blocking synchronous architecture to a non-blocking asynchronous architecture
- Must easier if you are coming from React/Angular instead of Vapor 2!

# ASYNC MOTIVATIONS

- Blocking is bad
- Limited resources even on the server
- Virtual thread context switching is expensive
- Thread safety is hard
- Provide a future with a promise to do something when the future object returns

# ASYNC MOTIVATIONS – PERFORMANCE

# ASYNC MOTIVATIONS – PERFORMANCE



# USING FUTURES – FLATMAP

- `flatMap` to unwrap future and return a future:

```
func updateFirstUser(_ req: Request) throws -> Future<User> {
  return User.query(on: req).all().flatMap(to: User.self) { users in
    let user = users[0]
    user.name = "Alice"
    return user.save(on: req)
  }
}
```

# USING FUTURES – MAP

- `map` to unwrap future and return a non-future:

```
func getFirstUser(_ req: Request) throws -> Future<User> {
  return User.query(on: req).all().map(to: User.self) { users in
    return users[0]
  }
}
```

# USING FUTURES – TRANSFORM

- `transform` to turn future into another type

```
func deleteFirstUser(_ req: Request) throws -> Future<HTTPStatus> {
  return User.query(on: req).all().flatMap(to: HTTPStatus.self) { users in
    return users[0].delete(on: req).transform(to: HTTPStatus.noContent)
  }
}
```

# FUTURES

- They are a learning curve
- Nice to use once used to them
- All nice and stable

# HOWEVER...

# SWIFT NIO

# SWIFTNIO

- Apple announced SwiftNIO at try! Swift Tokyo on Thursday
- Open source, non-blocking asynchronous network framework
- PRs ***and issues*** on Github 😍
- Effectively Netty for Swift
- Vapor team had an inclination it was coming
- Replaces a lot of the low-level Vapor modules

# SWIFTNIO

- Took < 48 hours to integrate with Vapor *and* Fluent
- Shouldn't have any major effect on the public API

# SERVICES

# SERVICES

- No guarantee of thread-safety
- Can't inject in `ViewRenderer` or `BCryptHasher`
- Stops you from using singletons 🙌
- Move to service locator pattern
- `Request` object is actually a container - use it to make services:

```
let client = try req.make(Client.self)
```

# DATABASE SUPPORT

# DATABASE SUPPORT

- Native, ***non-blocking***, drivers for SQLite, MySQL, PostgreSQL and Mongo
- No `libpsql-dev` etc requirement anymore!
- Support for native DB types (PostgreSQL JSON etc)
- Generic models are slightly harder
- Key path support for properties:

```
return Acronym.query(on: req).group(.or) { or in
  or.filter(\.short == searchTerm)
  or.filter(\.long == searchTerm)
}.all()
```

# LEAF

# LEAF

- Stream support - can pass Leaf futures
- Codable support
- Better syntax:
  - `#for(post in posts) { …`
  - `#if(post.title != “Vapor London”) { …`

# PERFORMANCE

# VAPOR 3 PERFORMANCE

- Still early days
- Vapor 3 ~130k RPS
- Gin (Go) ~99k RPS
- Node ~60k RPS
- Vapor 2 ~65k RPS
- Vapor 3 with NIO ~90k RPS
- Depends on your use-cases obviously! Tests are never accurate
- However NIO (and the non-blocking DB drivers) and going to help massively

# RELEASE SCHEDULE

# VAPOR 3 RELEASE SCHEDULE

- 9th Feb - first beta
- 23rd Feb - first RC
- There should be no more breaking changes…theoretically
- (I pushed one last night in Auth…)
- Vapor 3 will be released when Swift 4.1 is released
- …
- Assuming NIO integration goes well

# WHERE TO GO FROM HERE?

# VAPOR VIDEOS

# VAPOR BOOK


Up to date for iOS 11,
Xcode 9 & Swift 4
Server Side Swift with
Vapor
FIRST EDITION
Building Web APIs and Web Apps in Swift
By the raywenderlich.com Tutorial Team
Tim Condon, Tanner Nelson, Joannis Orlandos,
Jonas Schwartz & Logan Wright

# THE TEMPLATE

- `vapor new HelloVapor3 --branch=beta`

# QUESTIONS