# HAPI Philosophy

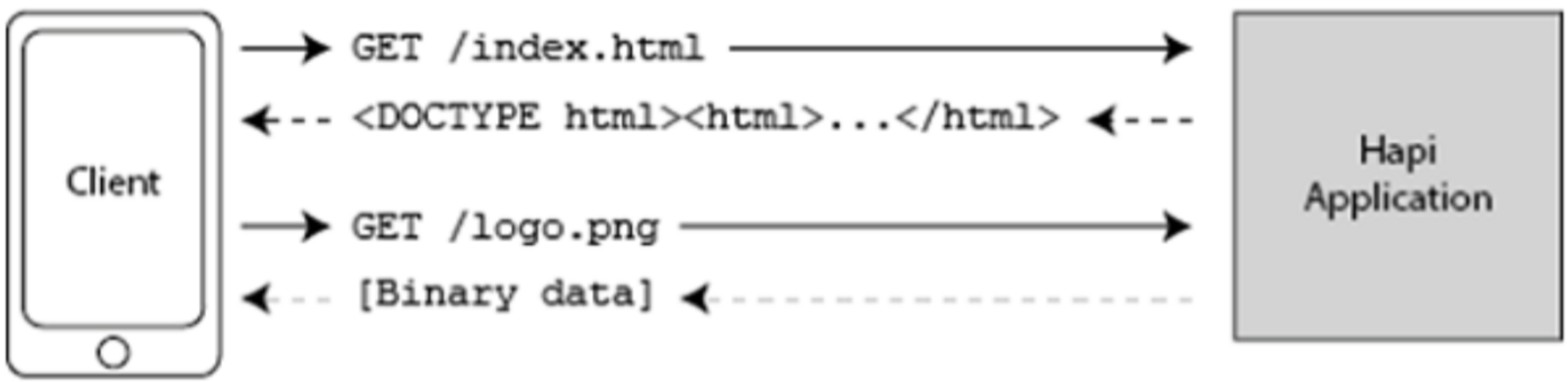
# What is HAPI?

- hapi.js is an open source framework for building web applications with Node.
- Can be used for building:
  - Web App
  - API Server


*“A rich framework for building applications and services hapi enables developers to focus on writing reusable application logic instead of spending time building infrastructure.”*

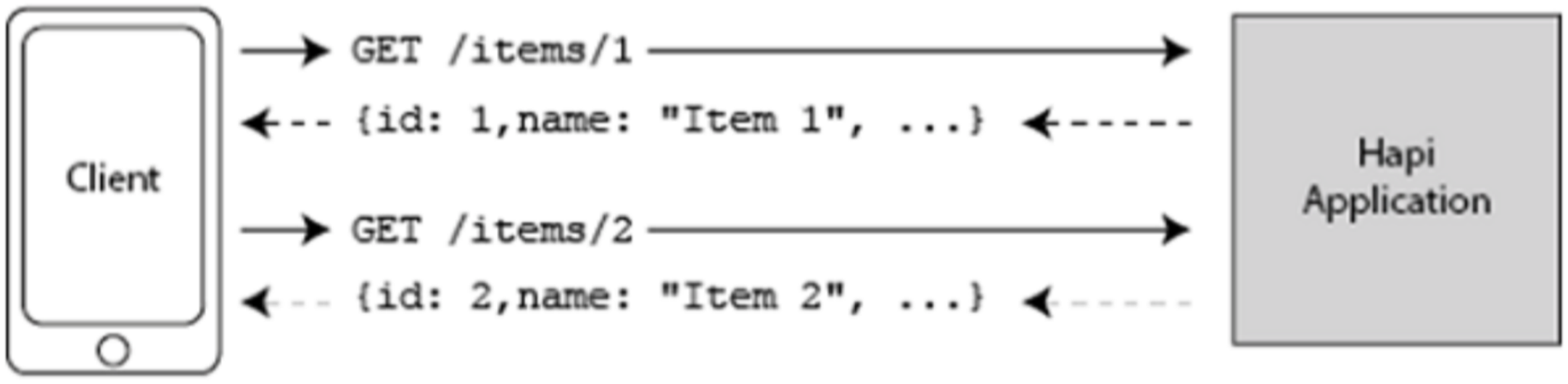
# Web Application

- Application delivers an Conventional Web Application
- All data conveyed in HTML format
- Client is a Web Browser

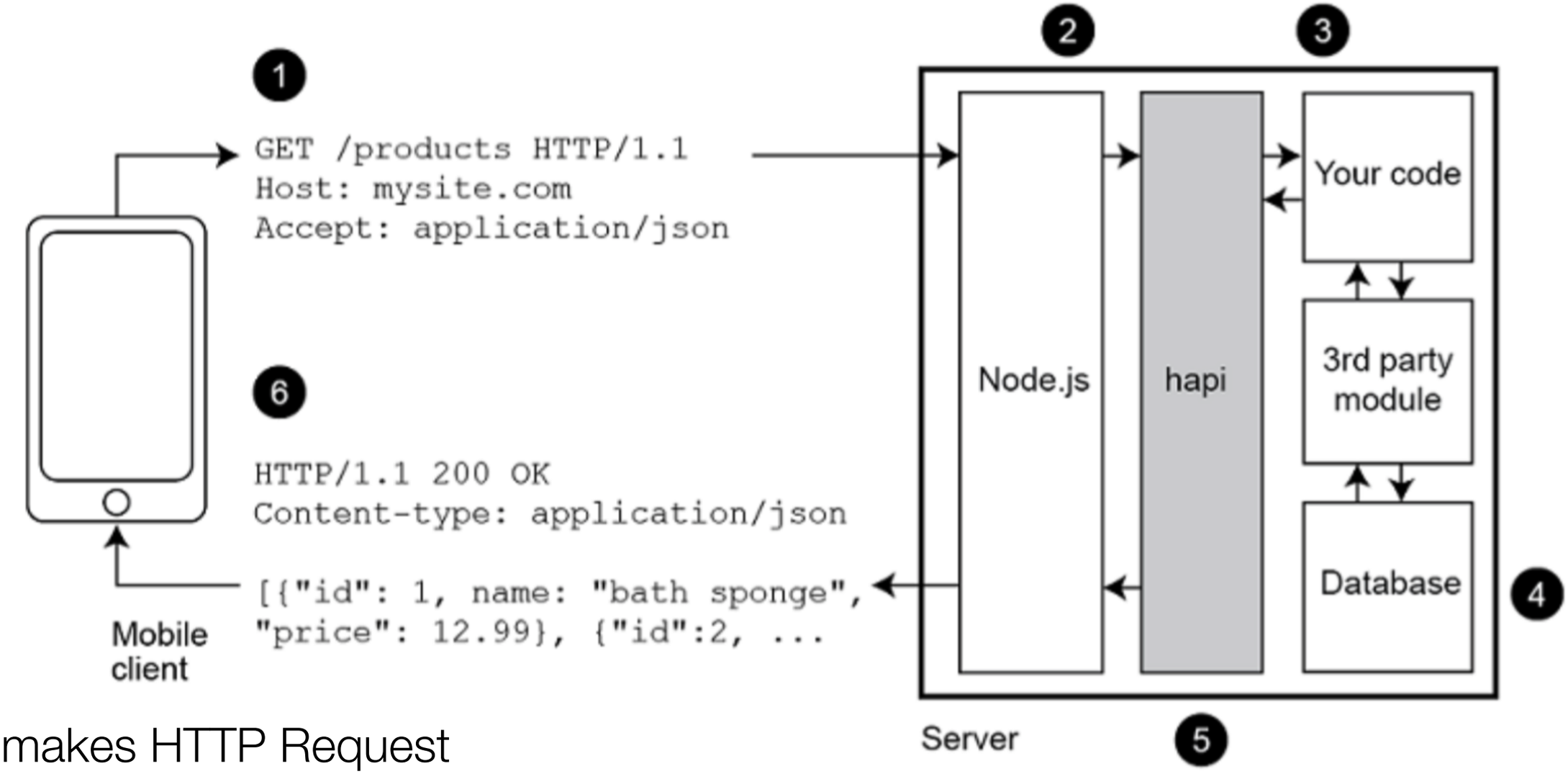


# API Server

- Application delivers an Application Programming Interface
- All data conveyed in JSON format
- Client are other programs : mobile, test clients, js client apps



# HAPI & Node

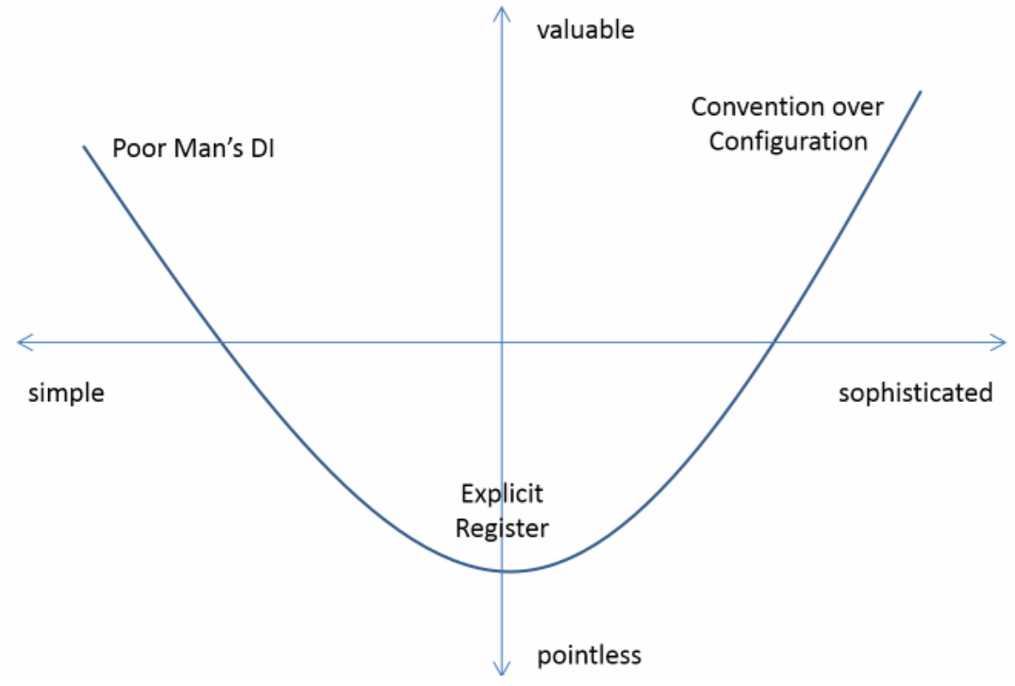



1. Client makes HTTP Request
2. Request received by Node and forwarded to api
3. Hapi authenticates user and routes request to correct function
4. Application logic executes, retrieves data from database
5. Data passed to Hapi reply function. Hapi validates, caches data.
6. Data transmitted over HTTP by node to client

# Why Choose Hapi?




- Its Node
- Its Modular
- It favours Convention over Configuration (or Code)



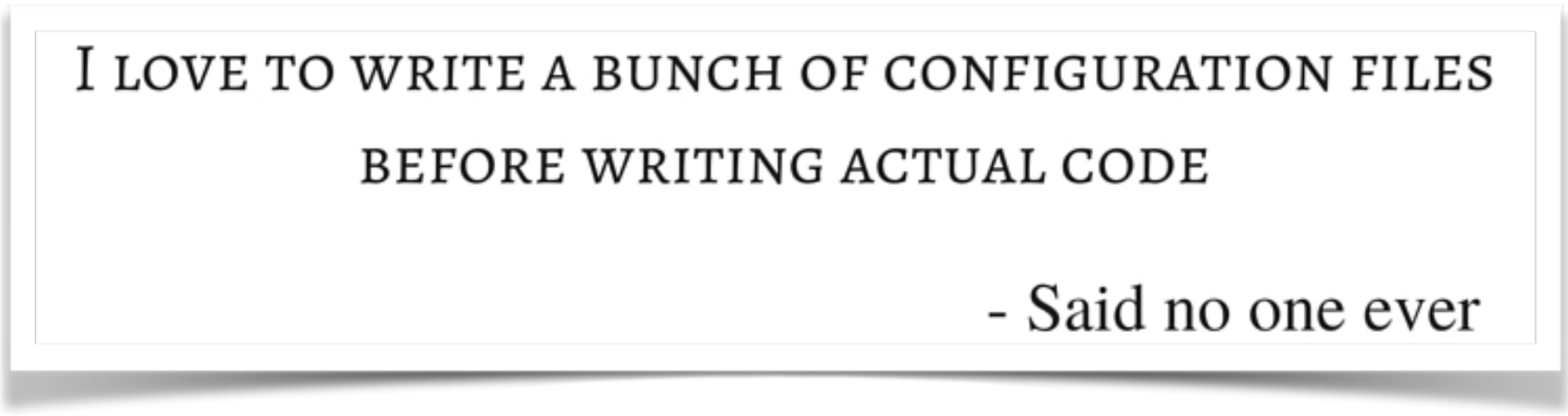
# Why Hapi? - its Node

- Node is strong for building APIs.
- JSON has become the de facto standard encoding for transferring data over the web.
- Working with JSON in JavaScript is a natural choice.
- The low- level implementation details of Node’s runtime let you scale your API to thousands of concurrent users without using expensive hardware.

# Why Hapi? - Modularity

- Hapi plugin system lets you join together isolated chunks of applications like Lego and have them run as a single application.
- These individual chunks or plugins can be developed, tested and distributed (as npm packages) totally independently, maybe by different developers or teams in a large organisation
- Plugins also let developers create functionality to share with the entire open- source community.

# Why Hapi? - Convention over Configuration

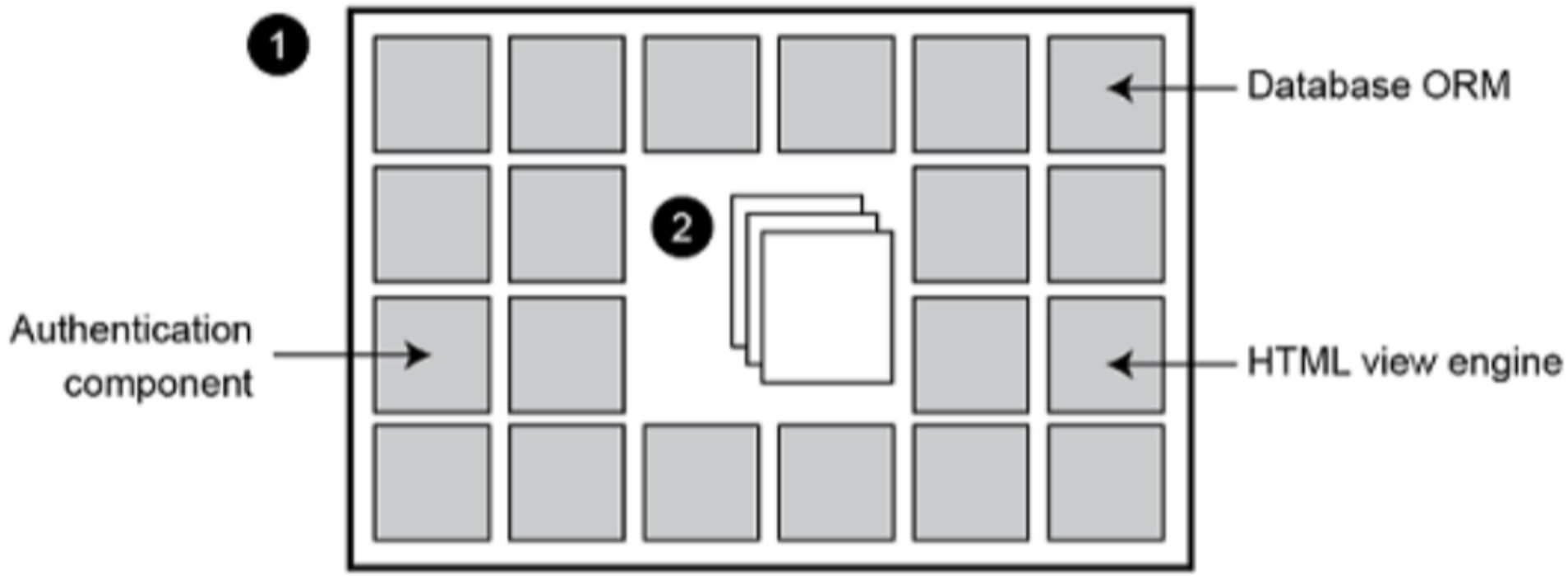



- Configuration-over-code means that there aren't lots of methods to remember to perform commonly required tasks

- Instead complex behaviours are wrapped up into simple configuration-driven APIs.

- You don't need to start learning all these configuration options until you really need them because sensible defaults are always chosen for you by the framework

# Types of Framework: Monolithic




- All Encompassing - Highly Opinionated

1. Large Application Library with Many Components

2. Application is tightly bound to the framework and may be challenging to use external software

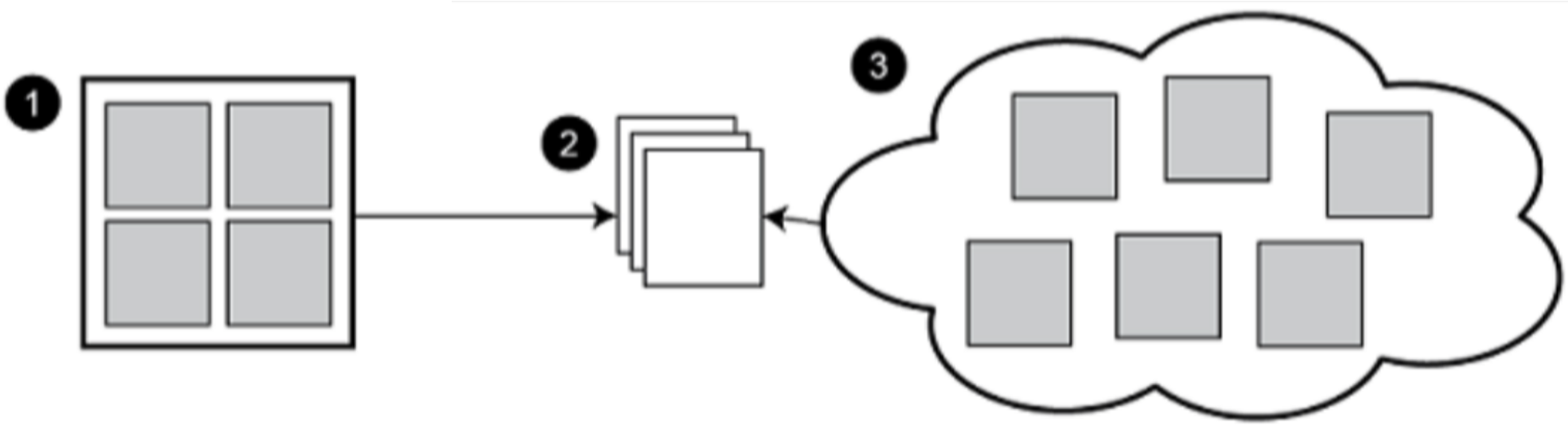
# Types of Framework: MicroFramework

- Lightweight, thin wrappers.

1. Small framework library with few components

2. Application is independent of framework

3. Application relies on many 3rd party libraries



# Framework Spectrum

All Encompassing -
Highly Opinionated

- Highly Opinionated frameworks require you to do things in a predictable and consistent way

*e.g. Rails, Sails*



- MicroFrameworks are often thin wrappers around some native capability of the platform to offer convenient APIs for common tasks


*e.g.Sinatra, Express*

Micro Frameworks
- Lightweight

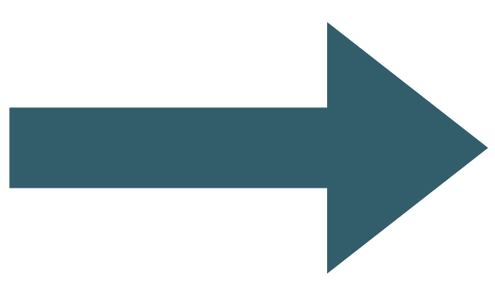
# Hapi Philosophy

All Encompassing -
Highly Opinionated

*e.g. Rails, Sails*

- Hapi threads a middle line between offering rich functionality out of the box while staying unimposing.

- The core library of hapi provides only the essential features that you will need when creating almost any modern web application.

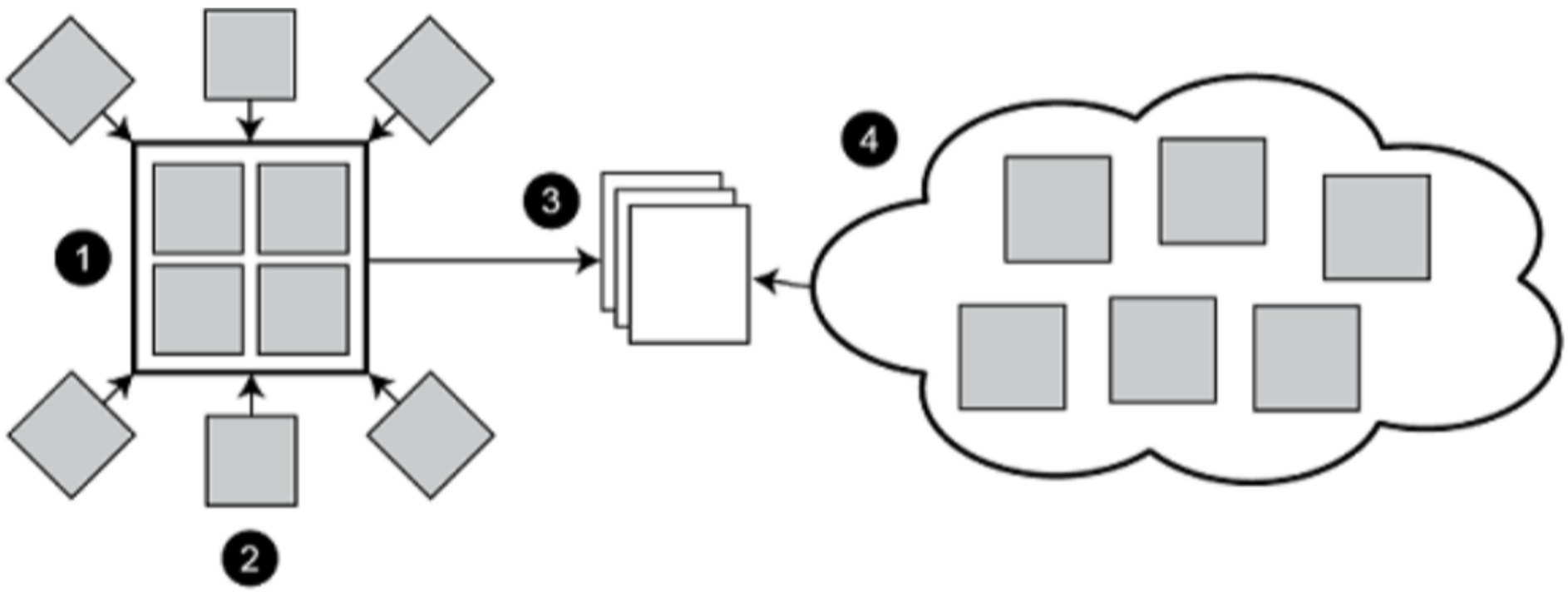




*e.g.Sinatra, Express*

Micro Frameworks
- Lightweight

# Hapi Approach




1. Small framework with few components

2. Frameworks core functionality extended with configurable official plugins

3. Application is independent of framework

4. Application relies on 3rd party libraries

# Example Hapi Application Structure