# Golang For DevOps And Cloud Engineers

# What is Go

- From the official Go homepage (https://go.dev/):
- Build **fast, reliable, and efficient software at scale**
  - Go is an **open source programming language** supported by Google
  - **Easy to learn** and get started with
  - **Built-in concurrency** and a **robust standard library**
  - Growing ecosystem of partners, communities, and tools

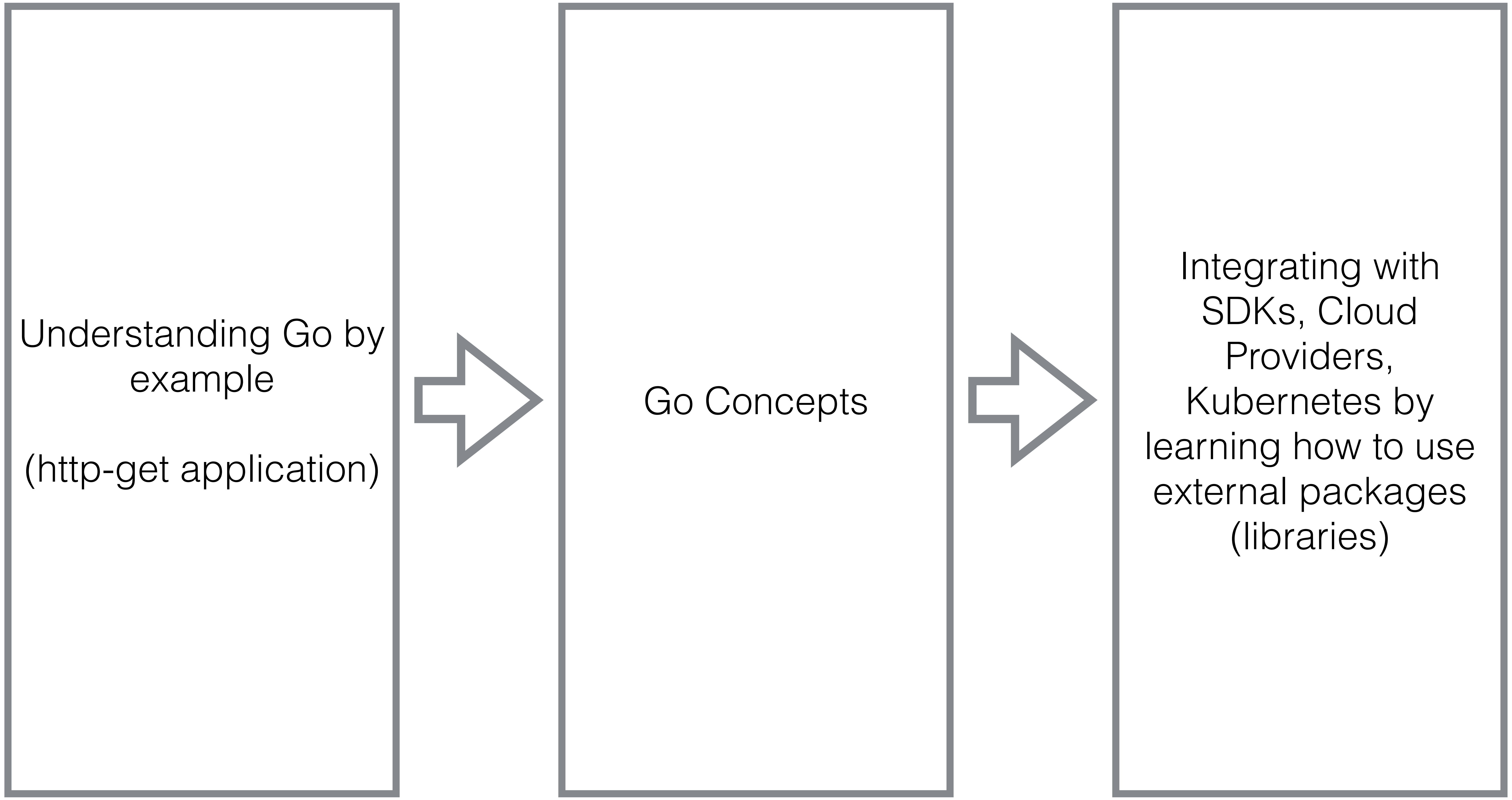
# Who am I

- My name is Edward Viaene
- I’m a **DevOps & Cloud specialist** and **Training Instructor**
- I started publishing on Udemy in 2015 and have now more than **250,000 students enrolled** in one of my DevOps / Cloud courses
- Since 2017 I have been using Go extensively, as it became more popular in the DevOps / Cloud space
- After years of writing Go code, I feel comfortable now to create this course, and to teach you all the Go tips & tricks I discovered over the years

# Course Objectives

- To be able to read, understand and write Go code
- To be able to write enterprise ready applications
- To be able to write applications that integrate REST APIs
- To be able to write applications that integrate with a cloud provider
- To be able to write applications that integrate with Kubernetes
- To be able to write applications that integrate with any custom integration that has a Go SDK available

# Course Layout



Understanding Go by example

(http-get application)

Go Concepts

Integrating with SDKs, Cloud Providers, Kubernetes by learning how to use external packages (libraries)

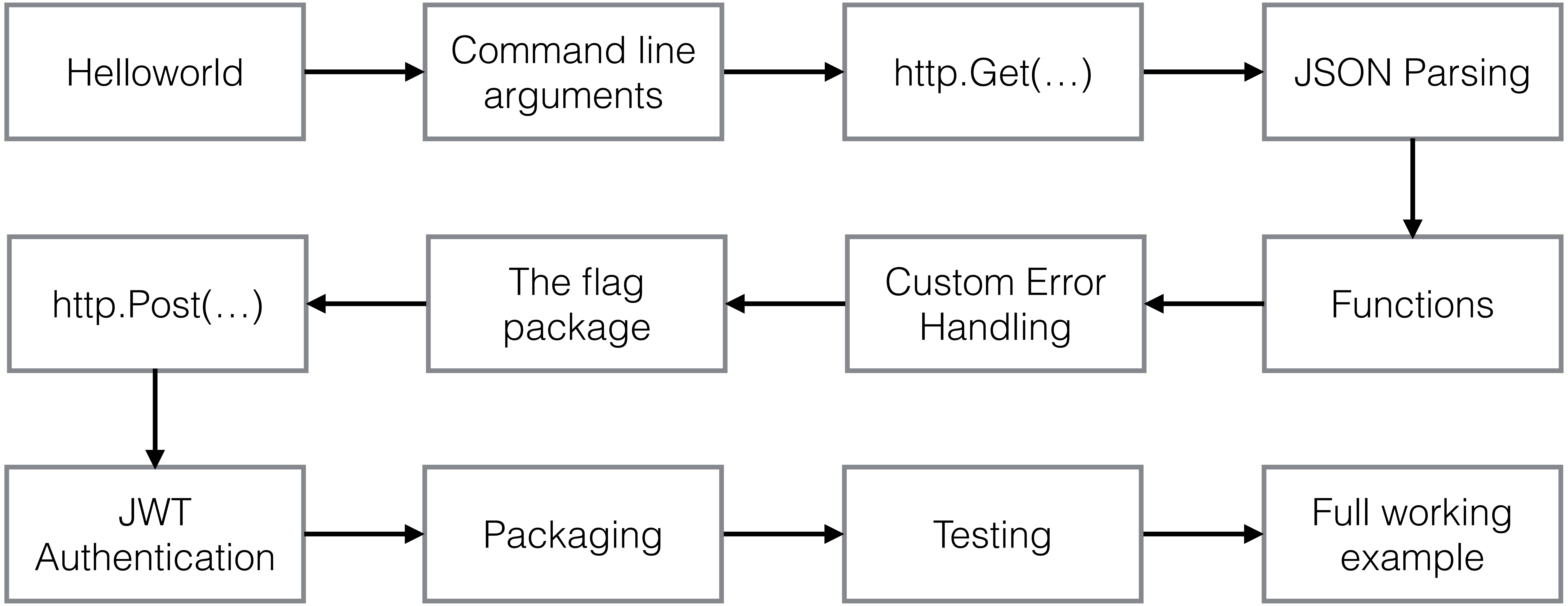
# Course Files

- A link to all course files can be found in the next lecture: Source files and useful information

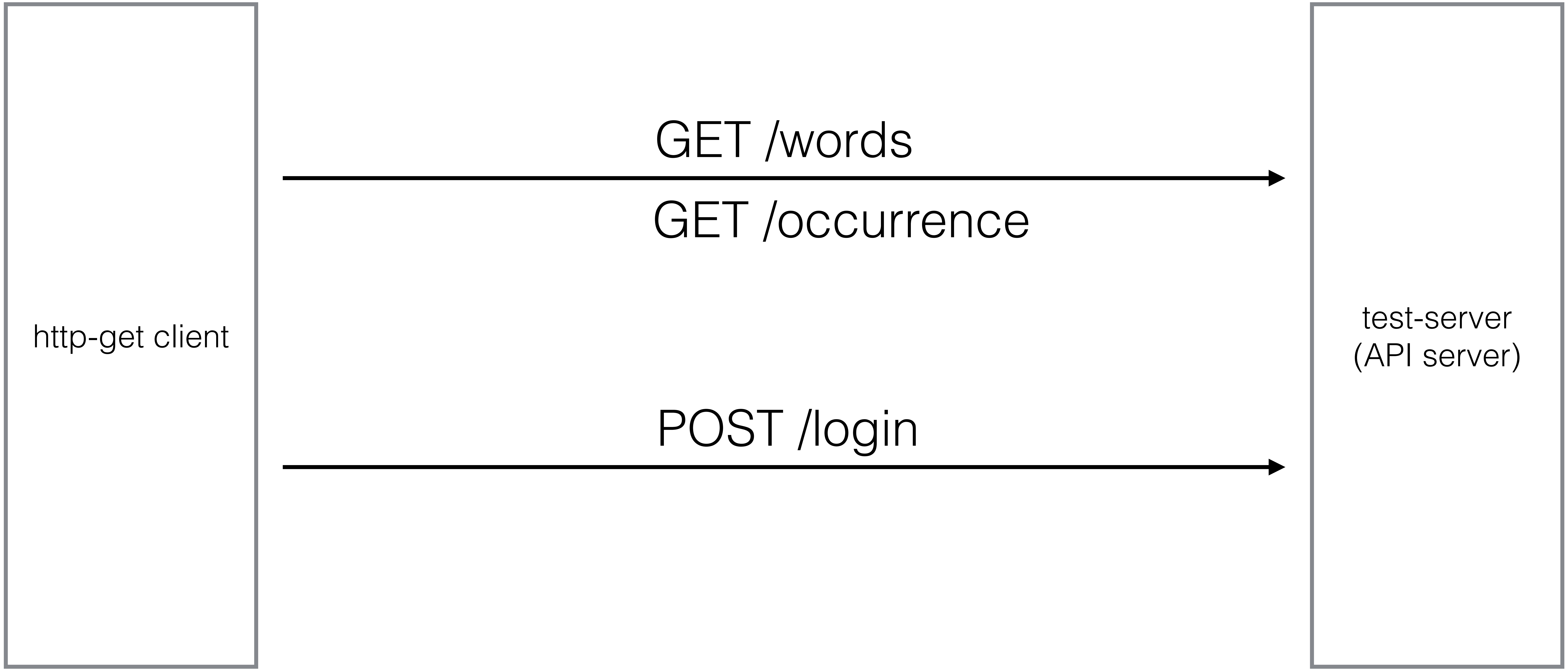
# Visual Studio Code Installation

# First Go application

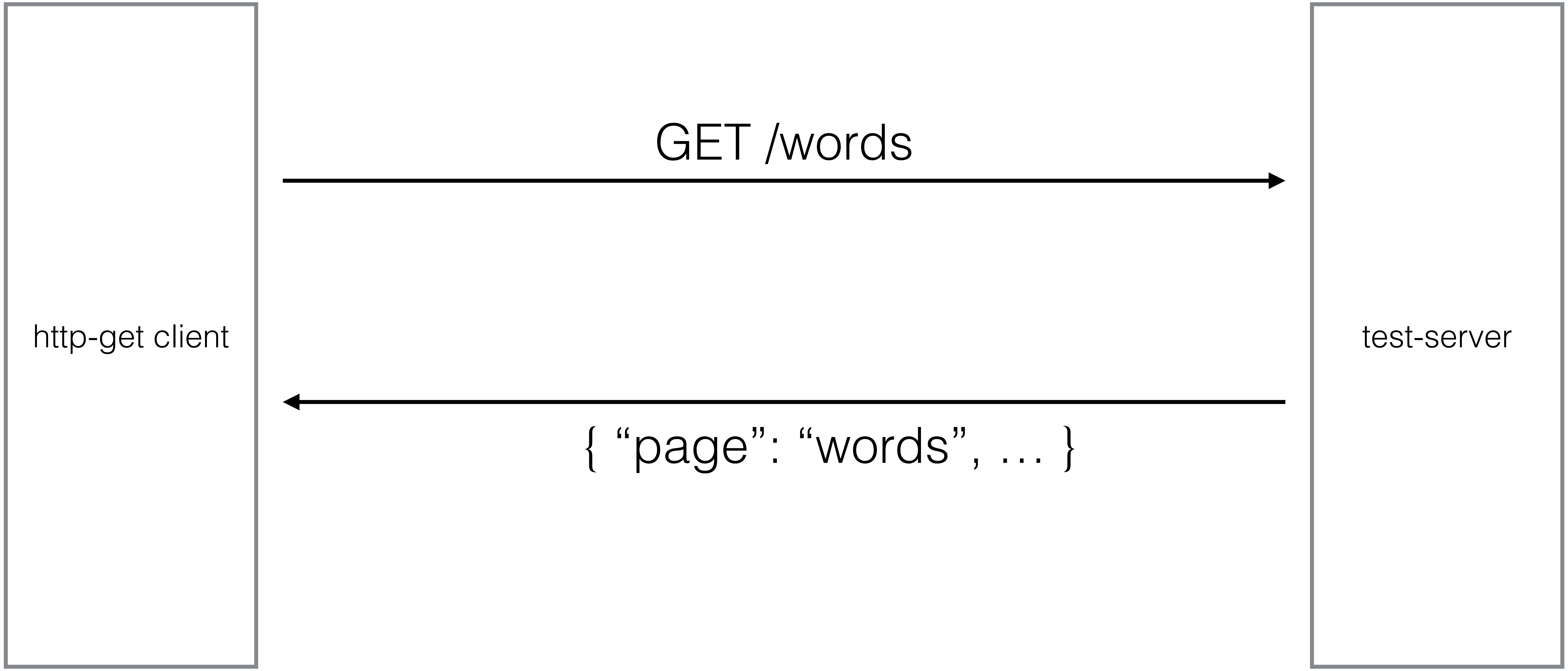
# First Go Application



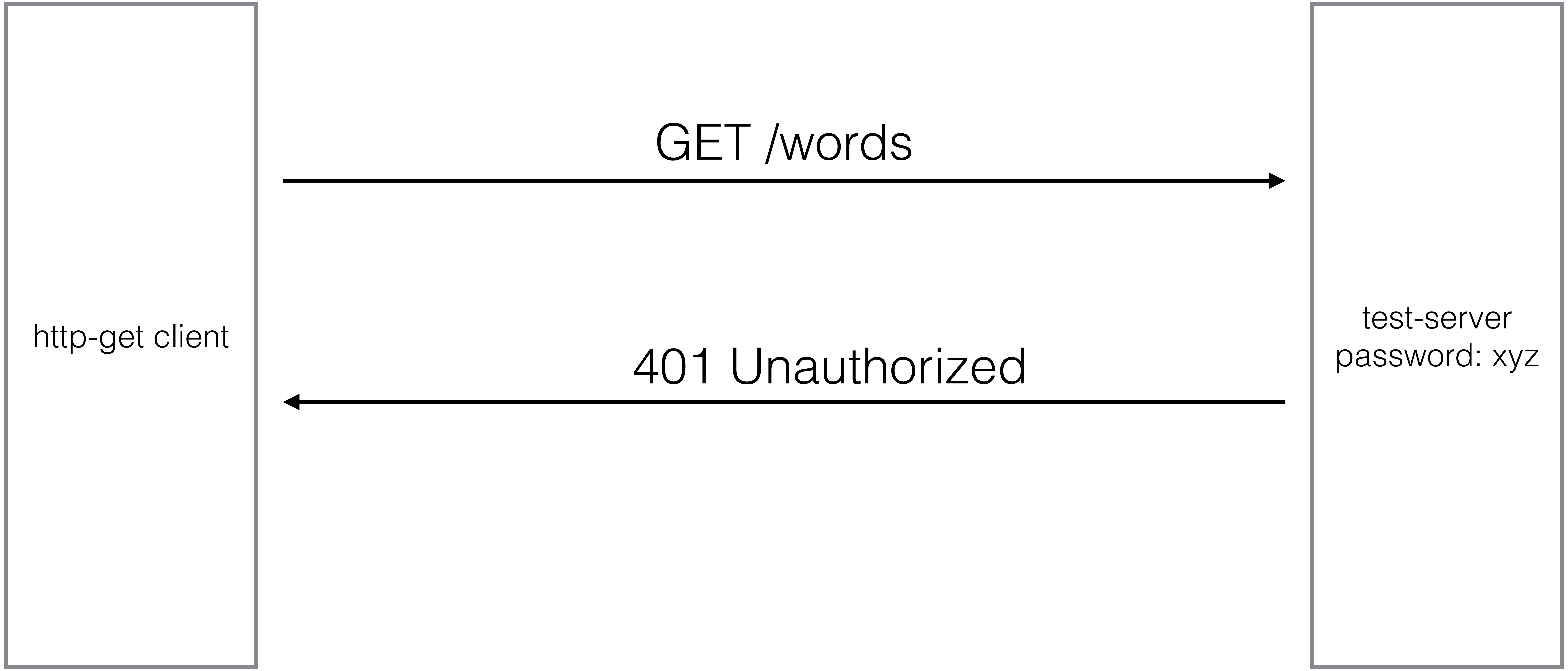
# http-get client



# JWT Auth



# JWT Auth



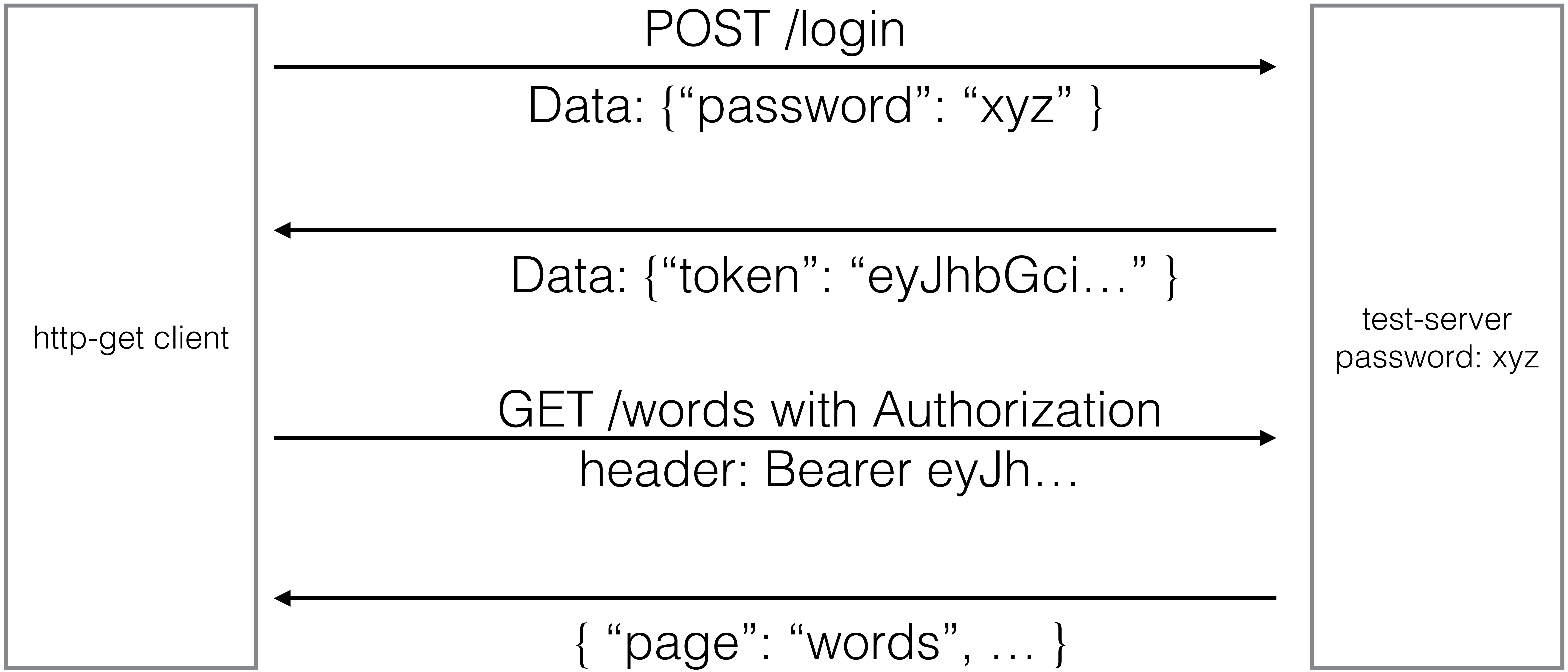
http-get client

GET /words

401 Unauthorized

test-server
password: xyz

# JWT Auth



http-get client

POST /login

Data: {“password”: “xyz” }

Data: {“token”: “eyJhbGci…” }

GET /words with Authorization header: Bearer eyJh…

{ “page”: “words”, … }

test-server
password: xyz

# Arrays and Slices

# Arrays and Slices

- Arrays have a fixed length whereas slices are dynamic

Array:

```
var buffer [7]byte
```

Slice:

```
var buffer []byte
```

# Arrays and Slices

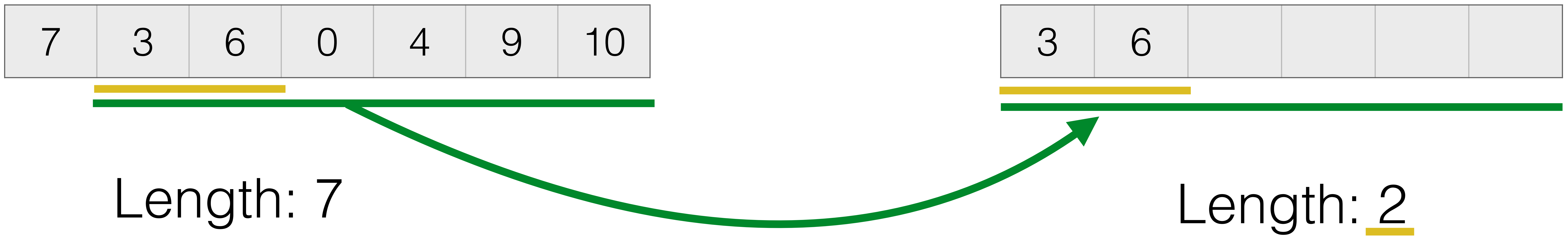
- Arrays are the building block of slices

Array:

```
var arr1 [7]int = [7]int{7,3,6,0,4,9,10}
```

Slice:

```
var arr2 []int = arr1[1:3]
```



| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | 3 | 6 | 0 | 4 | 9 | 10 |

| | | | | | |
|---|---|---|---|---|---|
| 3 | 6 | | | | |

Length: 7
Capacity: 7

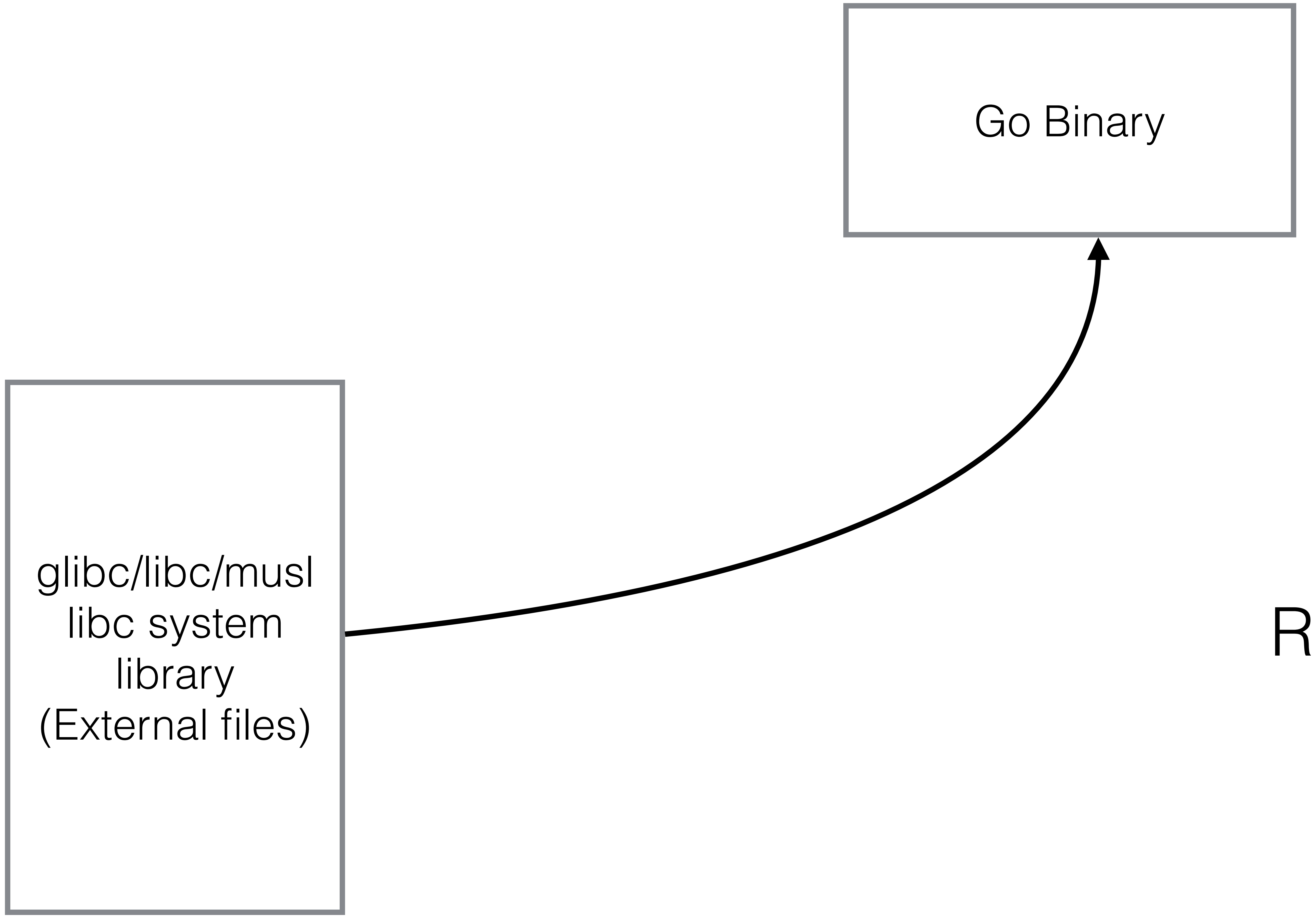
Length: 2
Capacity: 6
Element 0: arr1[1]

# (Cross)-Compiling and cgo

# Building Go Applications

- Go can cross-compile to any supported **OS** and **Architecture**
  - You need to supply **GOOS** and **GOARCH** during “go build”
  - **“go tool dist list**” shows you supported combinations
- When not cross compiling, **cgo** will be enabled, when cross-compiling it’ll be disabled
  - **cgo** allows you to run C code within Go
    - This is relevant even if you’re not using this feature yourself, because standard Go packages like “net” can use **cgo** (for example for DNS resolving)
  - **cgo** will link your binary to the current C library available on your operating system, but it’ll not work on an OS with a different C library

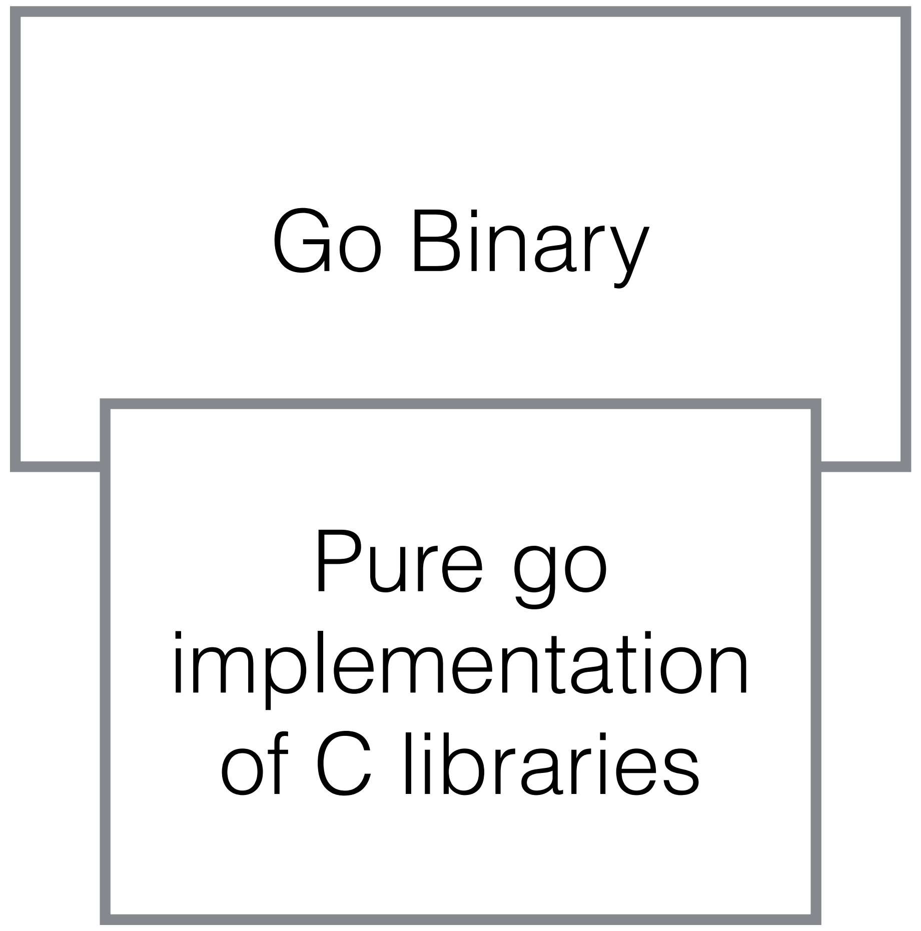
# Building Go Applications



CGO_ENABLED=1 go build

Result is a dynamically linked binary

# Building Go Applications



CGO_ENABLED=0 go build

Result is a statically linked binary

# Building Go Applications

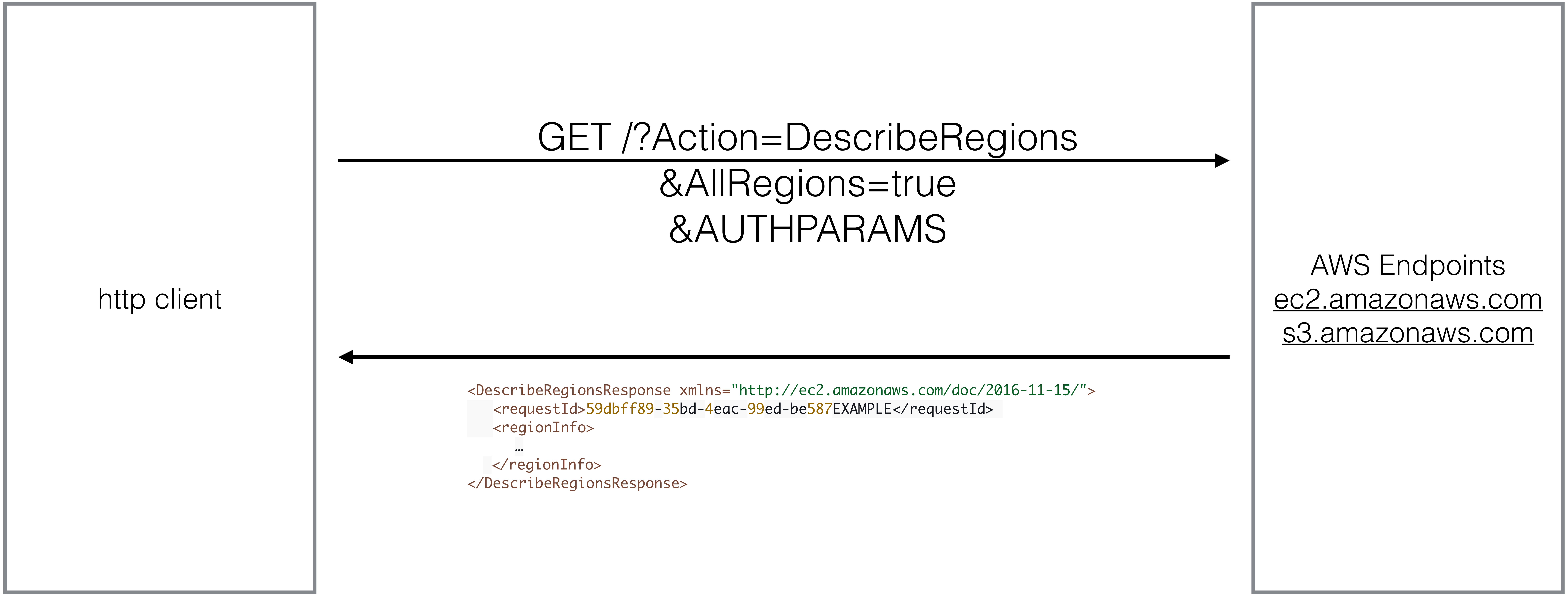
- Enabling CGO (when you’re not cross compiling)
  - May lead to a **binary smaller in size** (as C bindings for DNS Resolver, networking will be in libc/glibc/…)
    - You already have the **C libraries bundled with your OS**, there’s no need to have them included again in every binary
  - Will lead to **faster builds**

# Building Go Applications

- Disabling CGO
  - Is necessary when cross-compiling
  - Is also necessary if your C library on the destination system is different (for example you compile on Ubuntu Linux but run on Alpine)
    - Ubuntu is using GNU Libc and Alpine musl libc

# aws-sdk-go

# Amazon Web Services



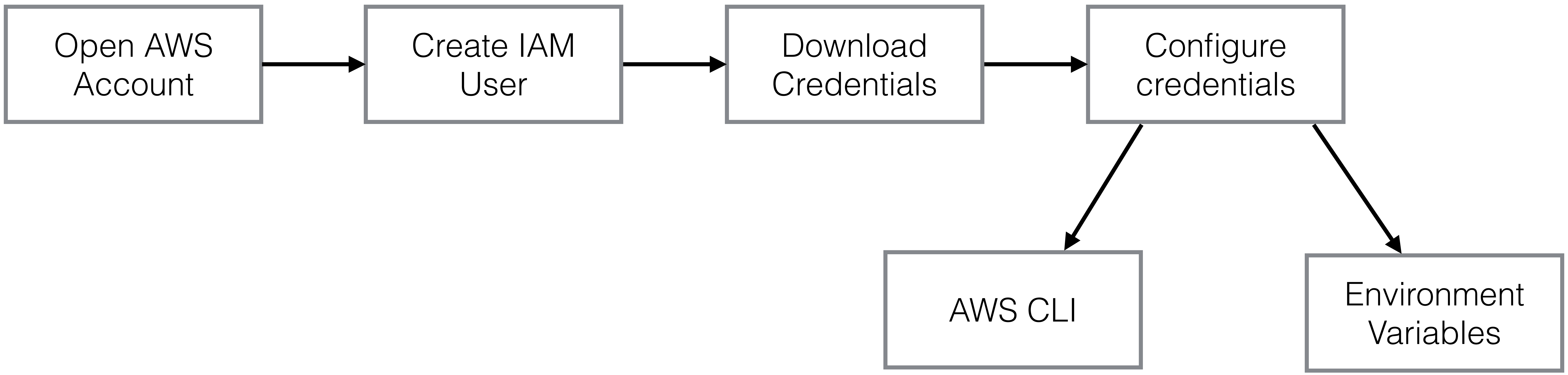
http client

GET /?Action=DescribeRegions
&AllRegions=true
&AUTHPARAMS

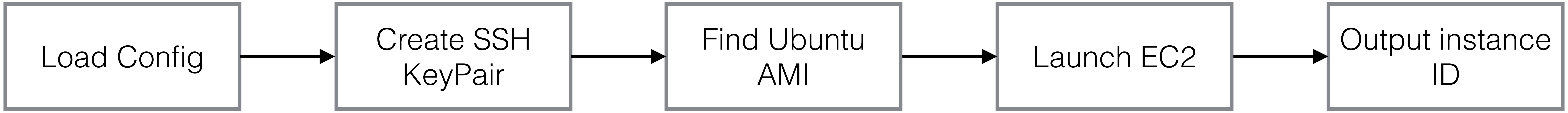
AWS Endpoints
ec2.amazonaws.com
s3.amazonaws.com

```xml
<DescribeRegionsResponse xmlns="http://ec2.amazonaws.com/doc/2016-11-15/">
   <requestId>59dbff89-35bd-4eac-99ed-be587EXAMPLE</requestId>
   <regionInfo>
      …
   </regionInfo>
</DescribeRegionsResponse>
```

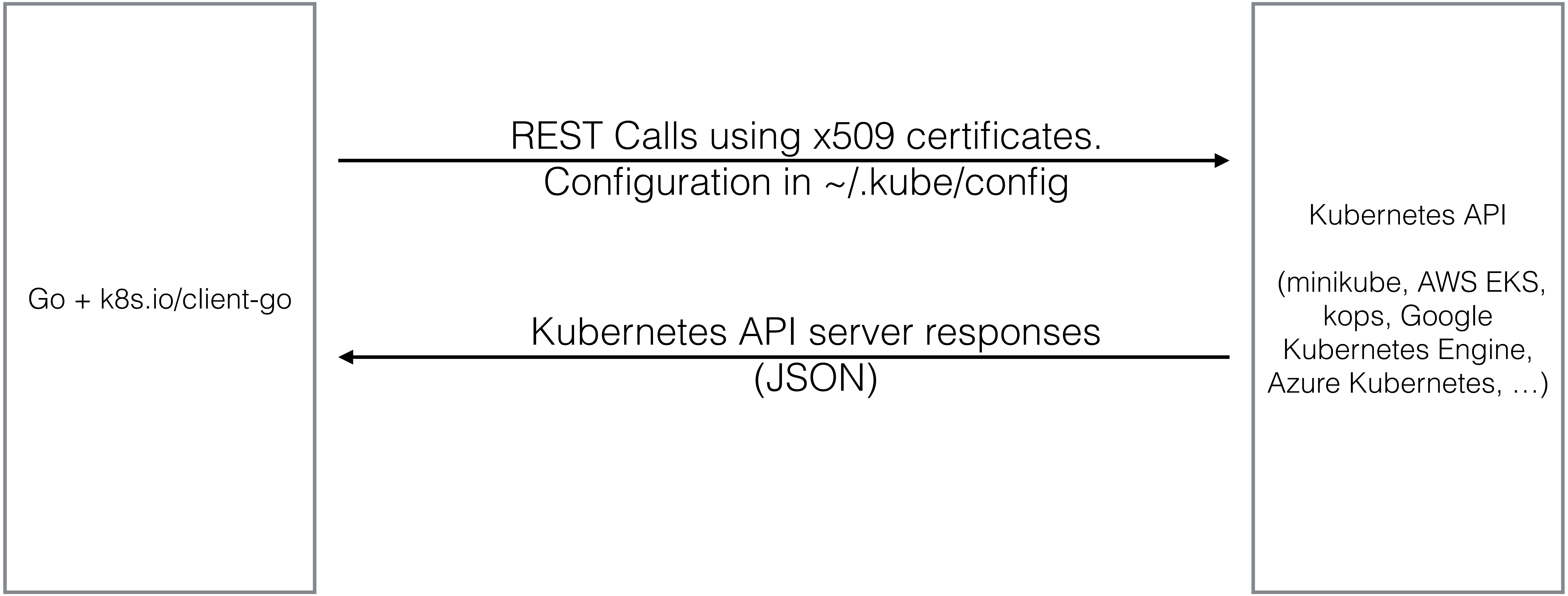
# AWS SDK


Open AWS Account
Create IAM User
Download Credentials
Configure credentials
AWS CLI
Environment Variables

# AWS SDK


Load Config
Create SSH KeyPair
Find Ubuntu AMI
Launch EC2
Output instance ID

# Kubernetes



Go + k8s.io/client-go

REST Calls using x509 certificates.
Configuration in ~/.kube/config

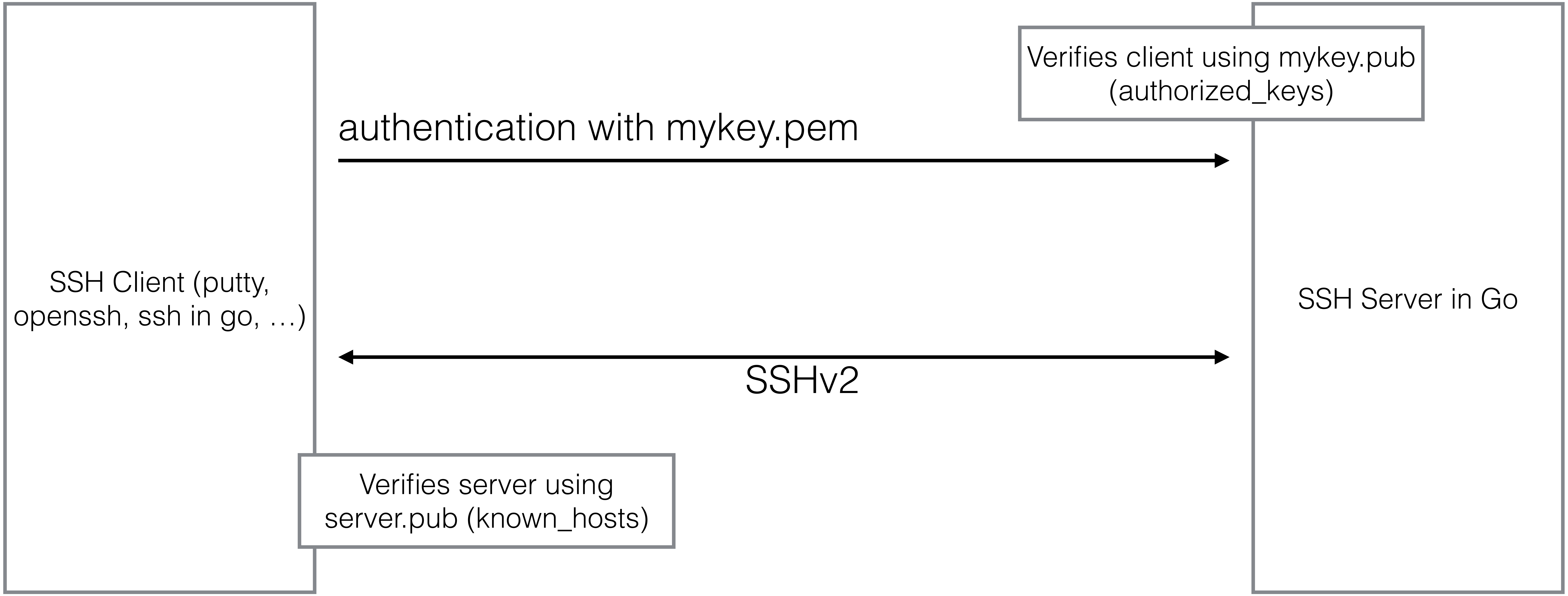
Kubernetes API server responses
(JSON)

Kubernetes API

(minikube, AWS EKS, kops, Google Kubernetes Engine, Azure Kubernetes, …)

# SSH in go

# SSH Server



SSH Client (putty, openssh, ssh in go, …)

authentication with mykey.pem

Verifies client using mykey.pub (authorized_keys)

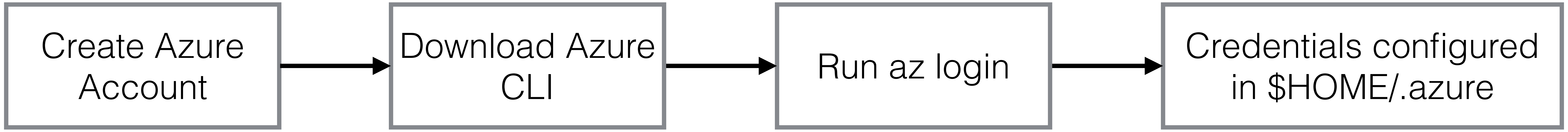
SSH Server in Go

SSHv2

Verifies server using server.pub (known_hosts)

# Microsoft Azure

# Azure Go SDK



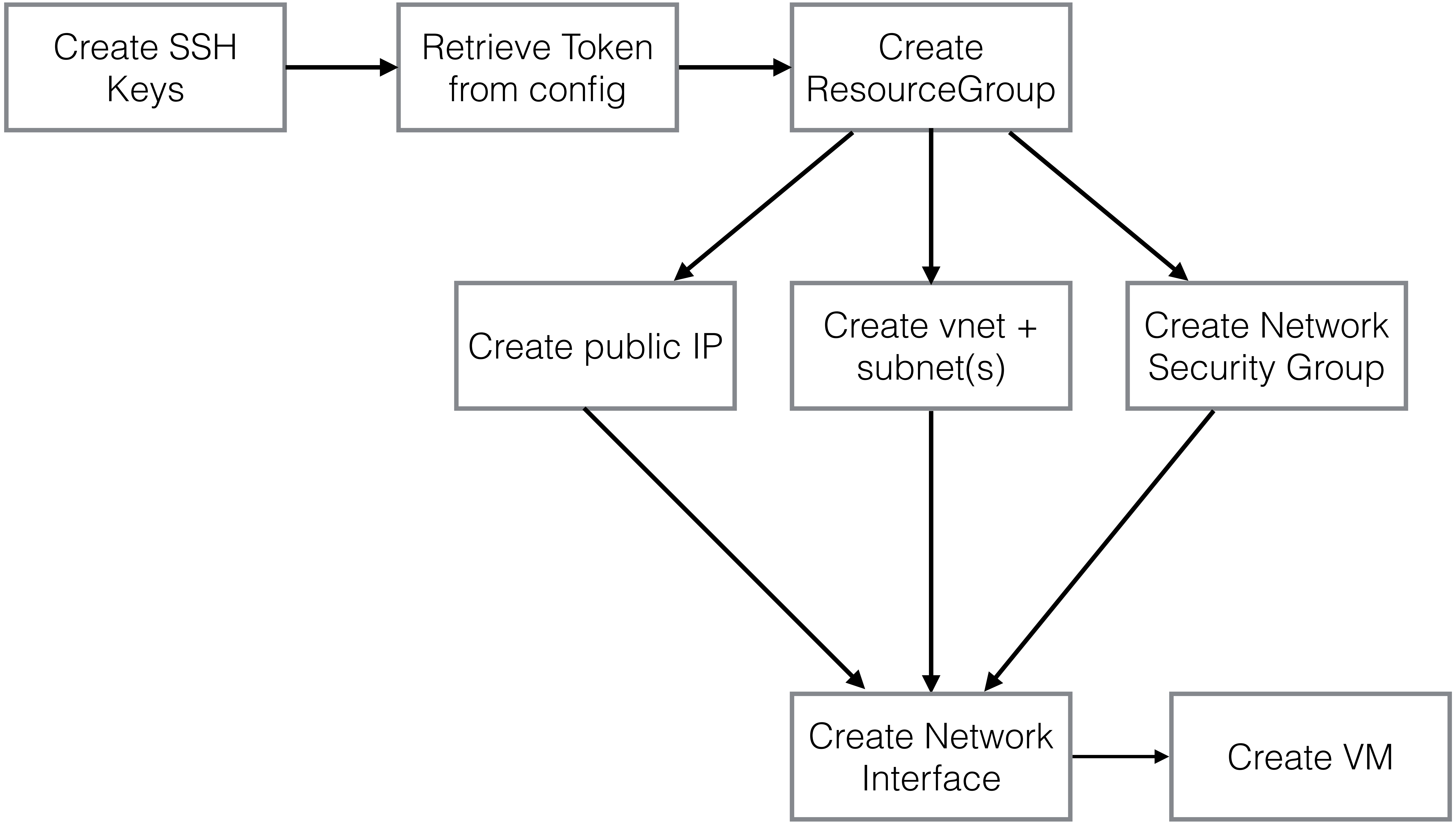
Create Azure Account

Download Azure CLI

Run az login

Credentials configured in $HOME/.azure

# Azure Go SDK



Create SSH Keys

Retrieve Token from config

Create ResourceGroup

Create public IP

Create vnet + subnet(s)

Create Network Security Group

Create Network Interface

Create VM

# Identity Providers

OpenID Connect

# What is an IdP

- Simply put, an Identity Provider (IdP) **manages and maintains identity data** for users

- It’s often used in conjunction with **Single Sign On (SSO)**

  - It gives a user a **single login & password (and optional MFA capability)**

  - It can be used for multiple applications and websites

  - While very convenient for the end-user, it’s also more secure

- 2 often used **implementations for authentication** within an Identity Providers setup are **OIDC and SAML**

# What is an IdP

- **OIDC** (OpenID Connect) and **SAML** (Security Assertion Markup Language) are authentication mechanisms, they don’t store login/password information themselves

- You’d still need to validate the login, password, and potentially MFA token with a separate mechanism

    - Users can be in a database, in LDAP, Microsoft Active Directory, or others

- **SAML 2.0** was released in 2005, while OpenID Connect (OIDC) was launched in 2015

- SAML is still used a lot, but **OpenID Connect** is more **lightweight** and **much easier to implement**

# Why implement OIDC

- It’s a great **learning experience**
  - Exposure to a lot of technologies: REST, OAuth, JWT, JWK
- You’re **often exposed to an IdP**, and it’s worth understanding the inner workings
- You can **build your own** IdP authorization server, client, or application
  - Understanding how the how flow works **will help you** when you need to build one of these components

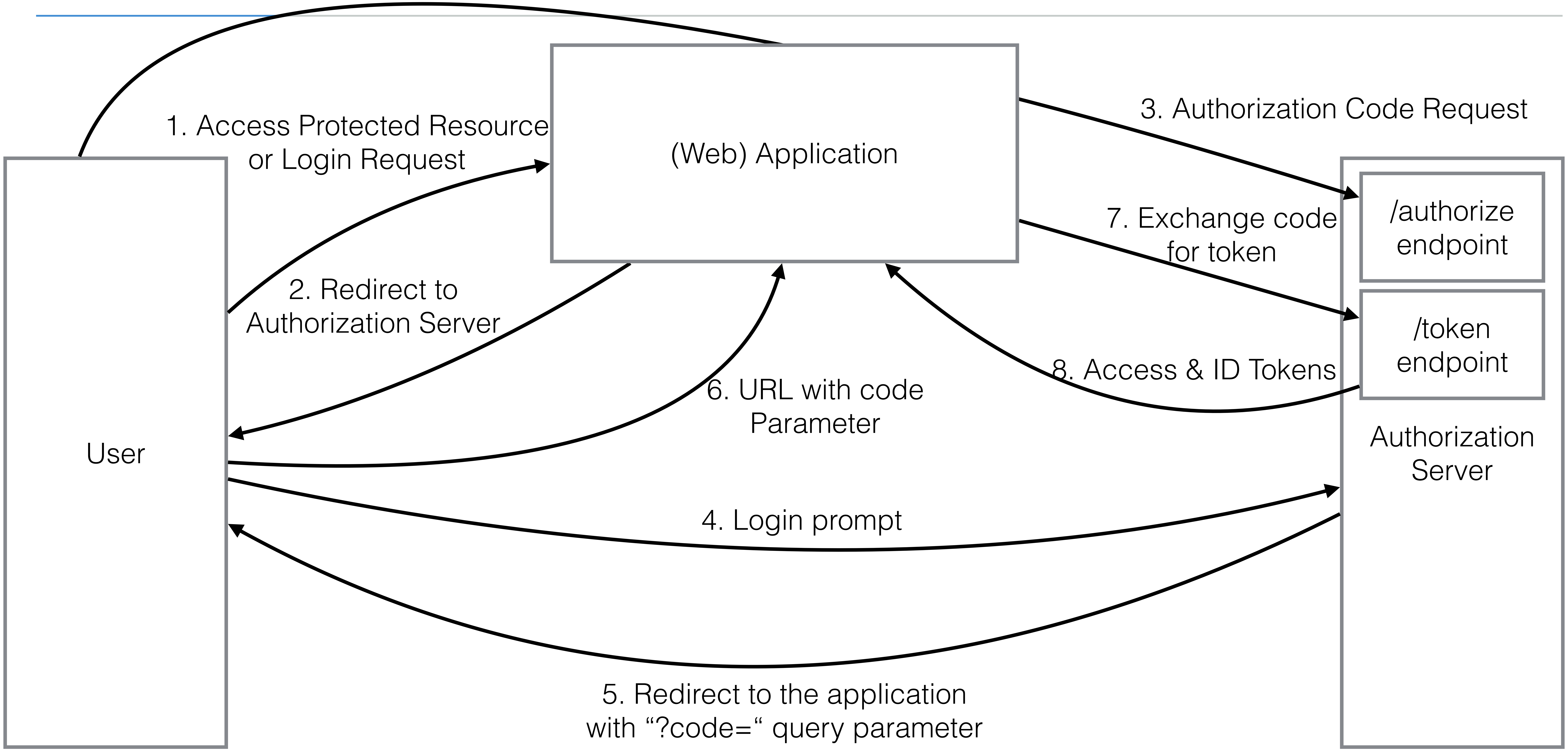
# What is OIDC

- **OIDC** is short for **OpenID Connect**
- It’s a simple **identity layer on top of the OAuth 2.0 protocol**
- OIDC can **verify the identity** of a user using an **Authorization Server**
- OIDC uses **REST endpoints**, so it’s **easily implemented**
- OIDC uses **JSON** and JSON Web Tokens (**JWT**)

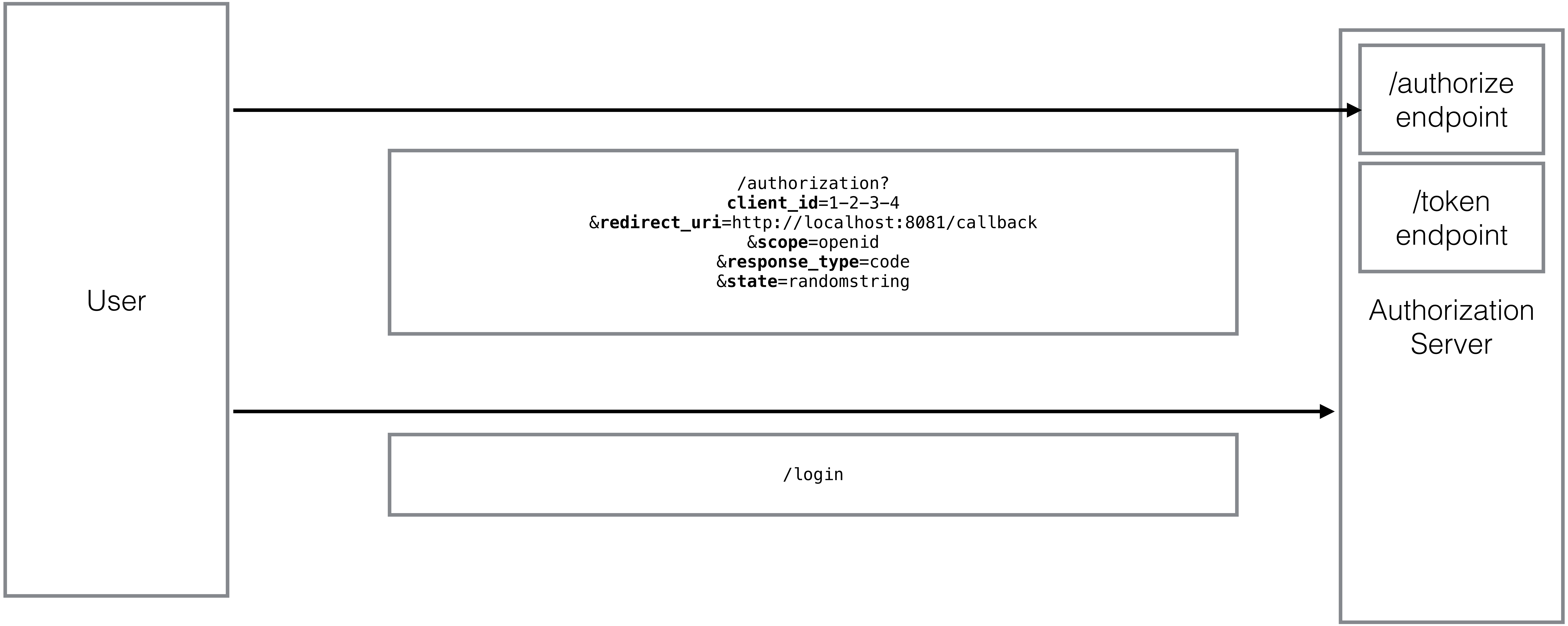
# OIDC Flows

- Authorization Code Flow
    - For web applications that can store a client_secret
    - This is the flow we're going to implement
- Implicit flow
    - For frontends / mobile apps that can't store a client_secret
- Hybrid flow
    - Combines above flows
    - Immediate access to an ID token

# OIDC


(Web) Application
User
/authorize endpoint
/token endpoint
Authorization Server
1. Access Protected Resource or Login Request
2. Redirect to Authorization Server
3. Authorization Code Request
4. Login prompt
5. Redirect to the application with “?code=“ query parameter
6. URL with code Parameter
7. Exchange code for token
8. Access & ID Tokens

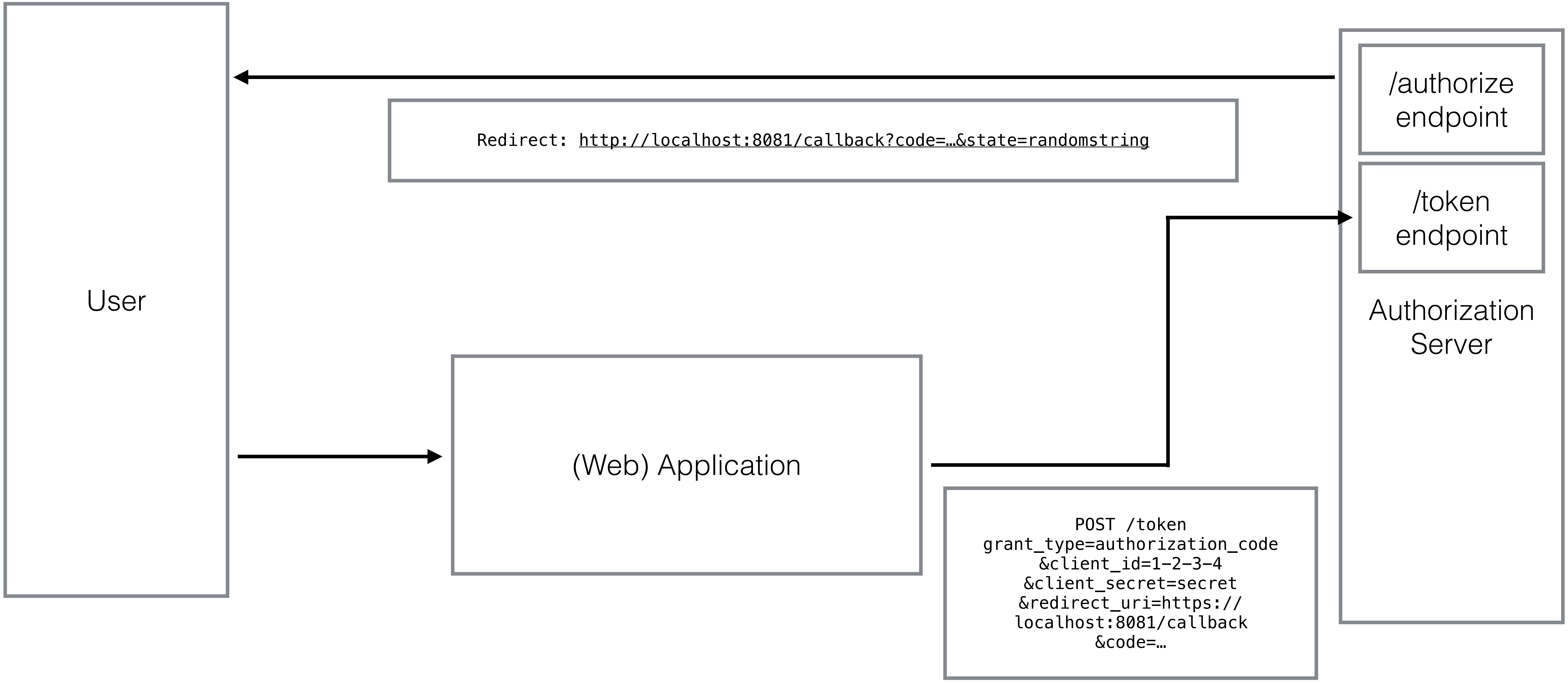
# OIDC



/authorization?  
**client_id**=1-2-3-4  
&**redirect_uri**=http://localhost:8081/callback  
&**scope**=openid  
&**response_type**=code  
&**state**=randomstring

/login

# OIDC



User

/authorize endpoint

```
Redirect: http://localhost:8081/callback?code=…&state=randomstring
```

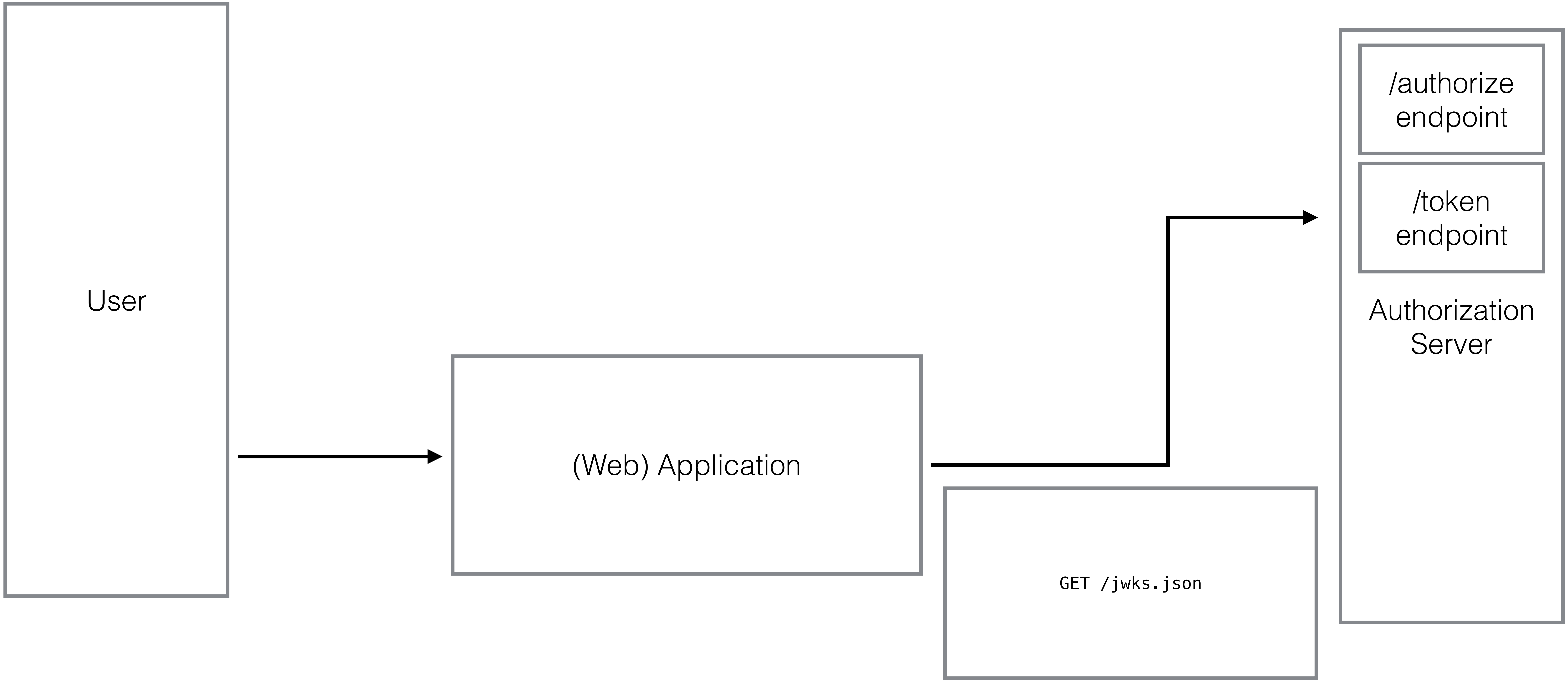
/token endpoint

Authorization Server

(Web) Application

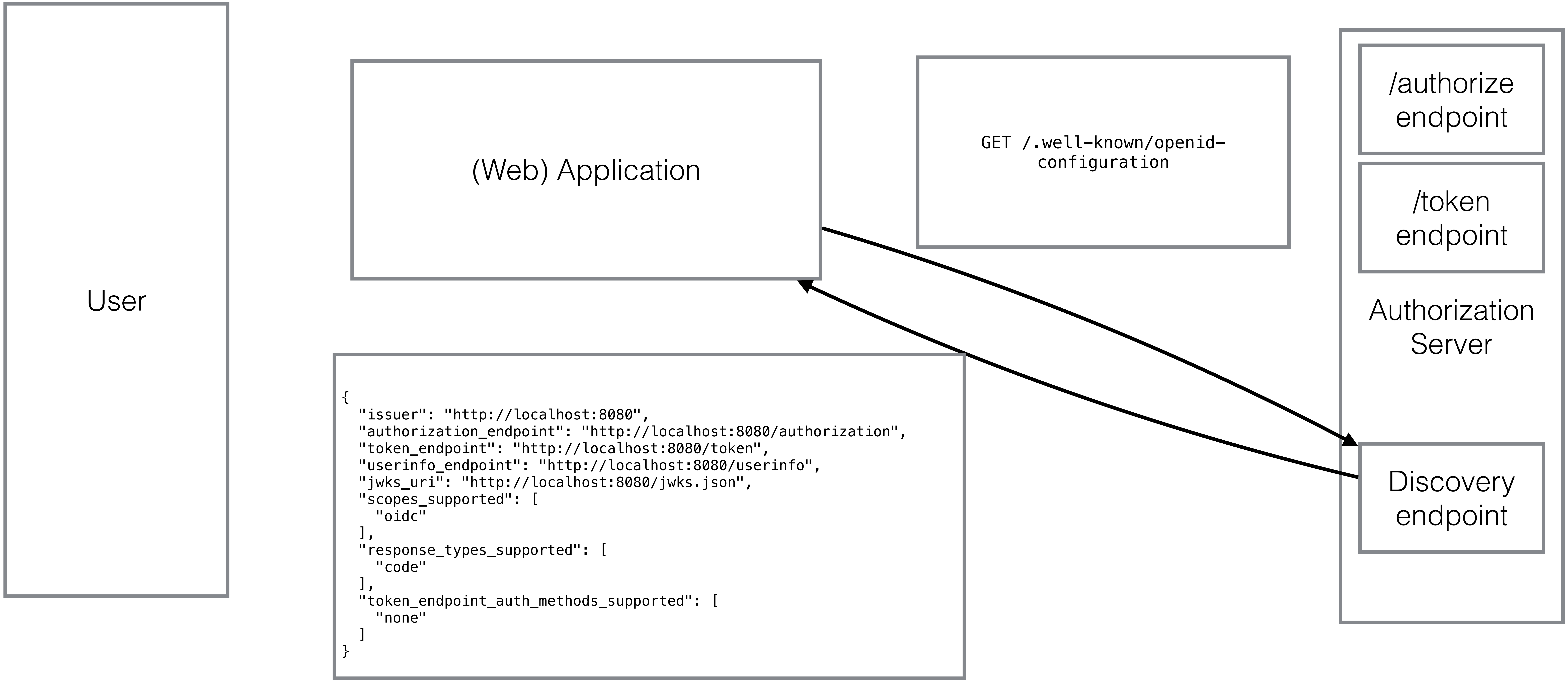
```
POST /token
grant_type=authorization_code
&client_id=1-2-3-4
&client_secret=secret
&redirect_uri=https://
localhost:8081/callback
&code=…
```

# OIDC



```
GET /jwks.json
```

# OIDC Discovery



User

(Web) Application

```
GET /.well-known/openid-
configuration
```

```
{
  "issuer": "http://localhost:8080",
  "authorization_endpoint": "http://localhost:8080/authorization",
  "token_endpoint": "http://localhost:8080/token",
  "userinfo_endpoint": "http://localhost:8080/userinfo",
  "jwks_uri": "http://localhost:8080/jwks.json",
  "scopes_supported": [
    "oidc"
  ],
  "response_types_supported": [
    "code"
  ],
  "token_endpoint_auth_methods_supported": [
    "none"
  ]
}
```

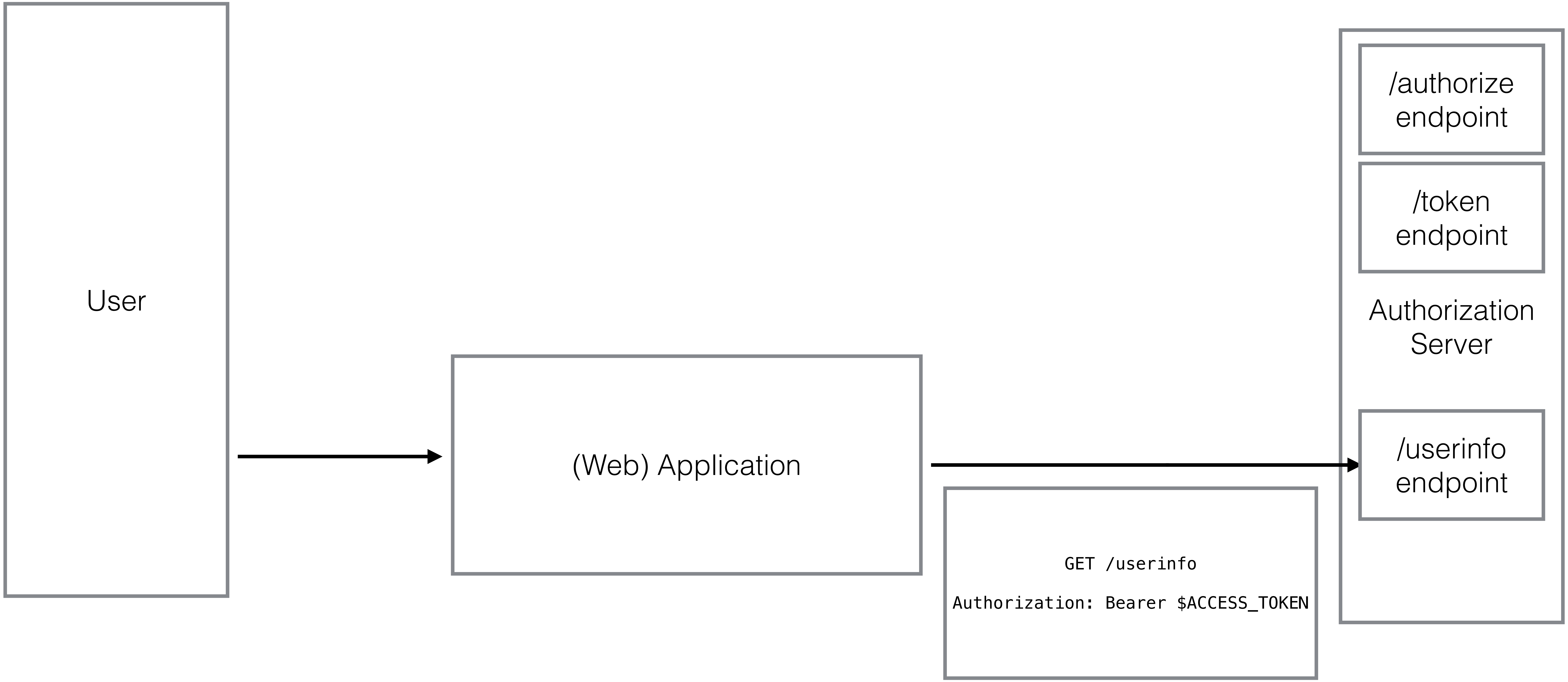
/authorize endpoint

/token endpoint

Authorization Server

Discovery endpoint

# OIDC



```
GET /userinfo

Authorization: Bearer $ACCESS_TOKEN
```

# Challenge

- If you'd like, you can **write** the **OIDC implementation yourself** first
- I'll include instructions to step-by-step write the implementation
- The start project contains already the function signatures, and I have written unit tests for those function to be able to test the validity of your code
- The start project is located in GitHub under oidc-start (https://github.com/wardviaene/golang-for-devops-course)
- The solution code is in the folder oidc-demo

# Using OIDC

- Now that we have an OIDC compatible authorization server we can **start adding applications that support OIDC**

- There are a lot of (SaaS) applications that support OIDC (often next to SAML)

- There's also companies that can act as an OIDC Provider itself, like Google, Apple, Facebook (social media logins have OIDC capabilities)

  - You could either use **their authorization server** and **trust their token**, or write an integration to validate a successful social login, and issue your own token with your own server

- Often plug-ins are available to existing tools and software to implement OIDC

# Using OIDC

- In the next lectures, I’ll show the OIDC integration with:
    - **Jenkins**, a popular CI/CD tool
    - IAM Federation with **OIDC in AWS**
        - We’ll make AWS trust our IDToken to issue access keys to our users

# TLS

# TLS

- TLS stands for **Transport Layer Security** and is used for **data encryption**
- Web encryption typically uses TLS to **encrypt communication between client and server**
- **TLS is the successor of SSL (Secure Socket Layer)**
- The default port for **unencrypted http** traffic is **80**, the default port for **encrypted (https)** traffic is **443**
- TLS itself is not an encryption algorithm, but a **protocol to negotiate and agree on a common set of encryption and hashing algorithms** (called the cipher suite)

# TLS

- With TLS enabled, the http server offers the client an **X.509 certificate**, which can be validated by the client to **ensure the server can be trusted**
    - The **hostname** of the server will be included in the server certificate
    - The server certificate will be **signed** by a Certificate Authority (CA)
    - If the client can **validate** the server certificate, we can trust the server
    - To be able to validate the certificates, we'll need to always have the **certificates of the Certificate Authorities** that can sign the certificates (also called **the root certificates**)
        - Browsers typically have this list **built-in**, and within Go, it'll also look for those files in **hardcoded system paths** to be able to validate certificates

# TLS

- This is all **client-server** communication where the server offers the client a certificate that can be validated
    - This is called **1-way TLS**
- You can also setup **2-way TLS** or **mutual TLS** (mTLS)
- In this scenario communication can only be established when the **client** also has an **X.509 certificate**
    - This is used often in **server-to-server communication**, for example to **secure communication between microservices**

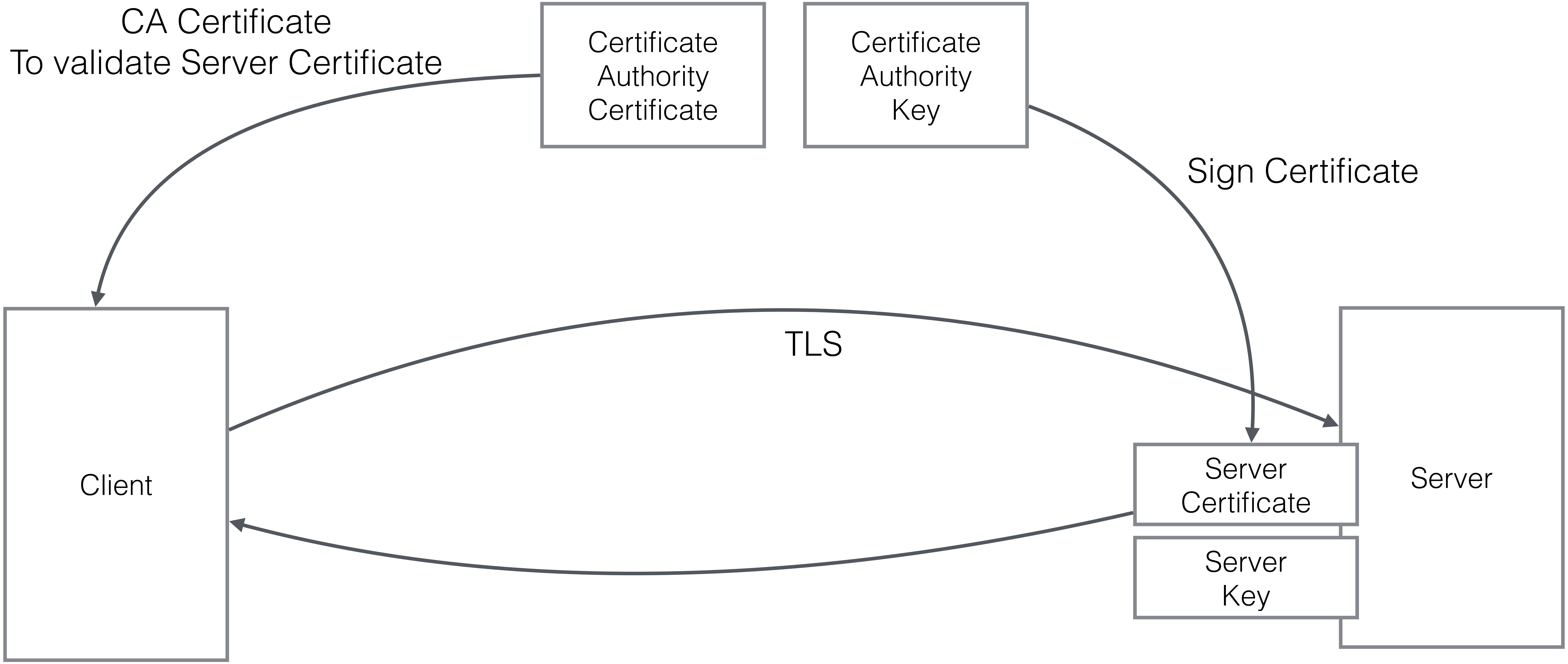
# TLS

- In the following lectures, I’m going to **add TLS support** to a simple Go http server
- There are **multiple strategies** to implement TLS:
    - Using a **self-signed certificate** (we will issue the Certificate Authority certificate ourselves, so only someone who has this specific CA certificate will be able to validate our server certificate)
    - Using a “real” certificate **issued by a company** that can **sign with a root certificate** (DigiCert, GeoTrust, RSA, GlobalSign, …)
    - Using **Let’s Encrypt**, a nonprofit Certificate Authority
- All these approaches can be used for 1-way TLS. For 2-way TLS a self-signed CA is common, but the other approaches would also work

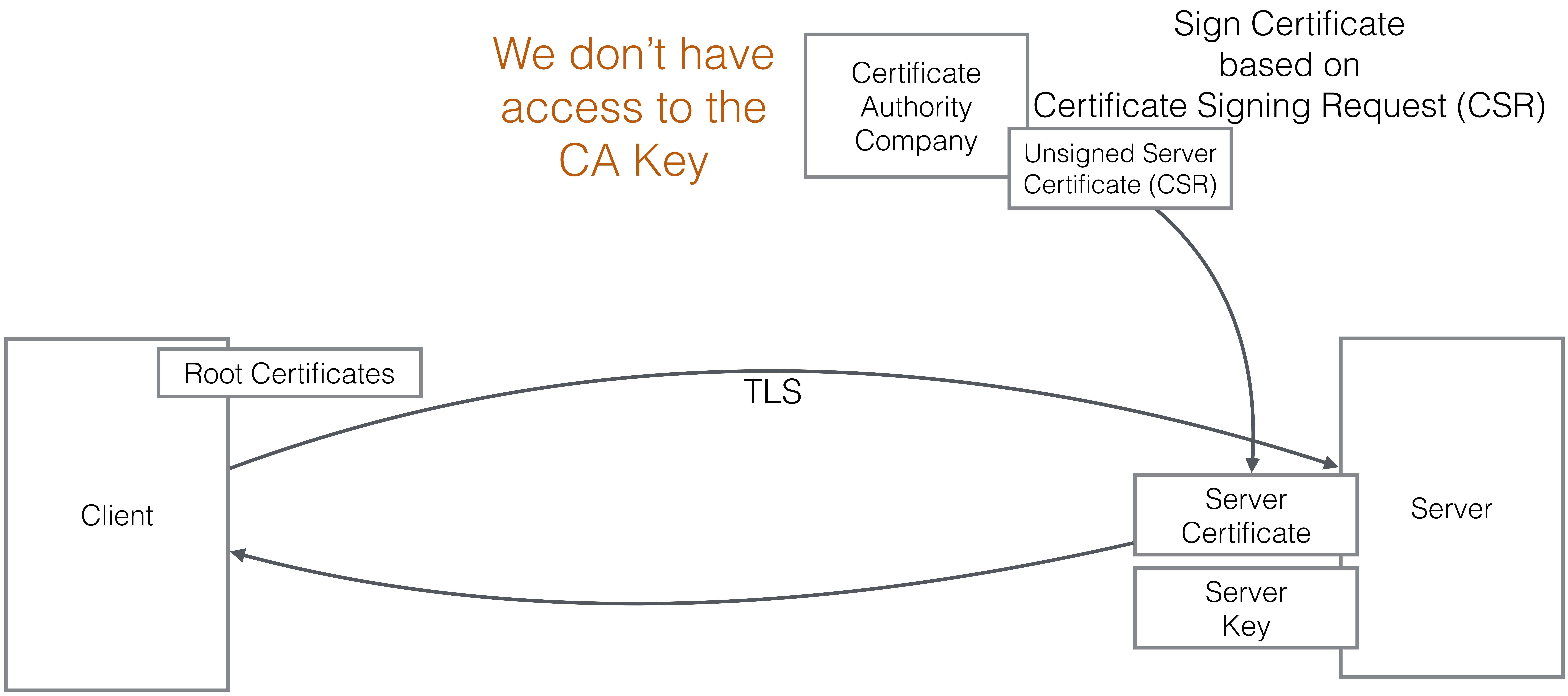
# TLS

Strategies (self-signed, signed by root CA, Let’s Encrypt)

# Self Signed CA


CA Certificate
To validate Server Certificate
Certificate Authority Certificate
Certificate Authority Key
Sign Certificate
TLS
Client
Server Certificate
Server Key
Server

# Root Signed CA



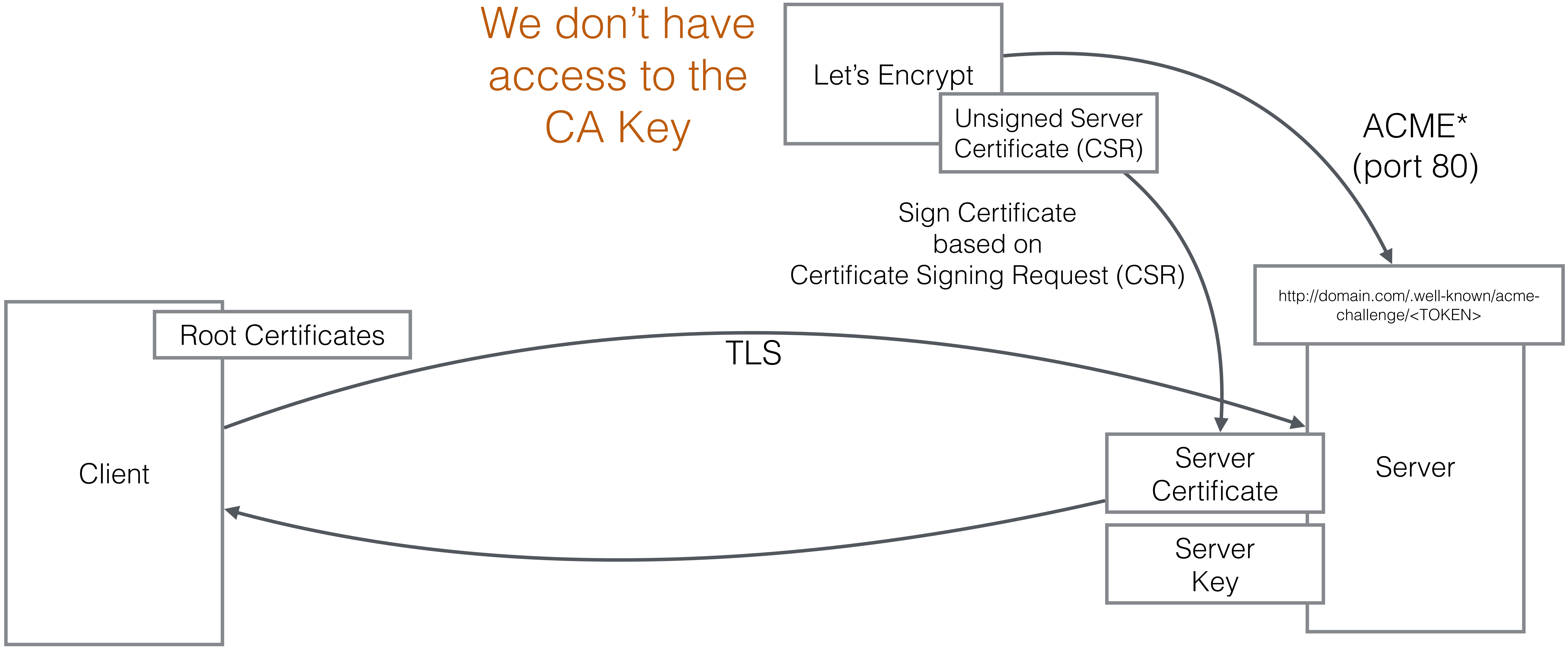
We don't have access to the CA Key

Sign Certificate based on Certificate Signing Request (CSR)

Certificate Authority Company
Unsigned Server Certificate (CSR)
Root Certificates
TLS
Client
Server Certificate
Server Key
Server

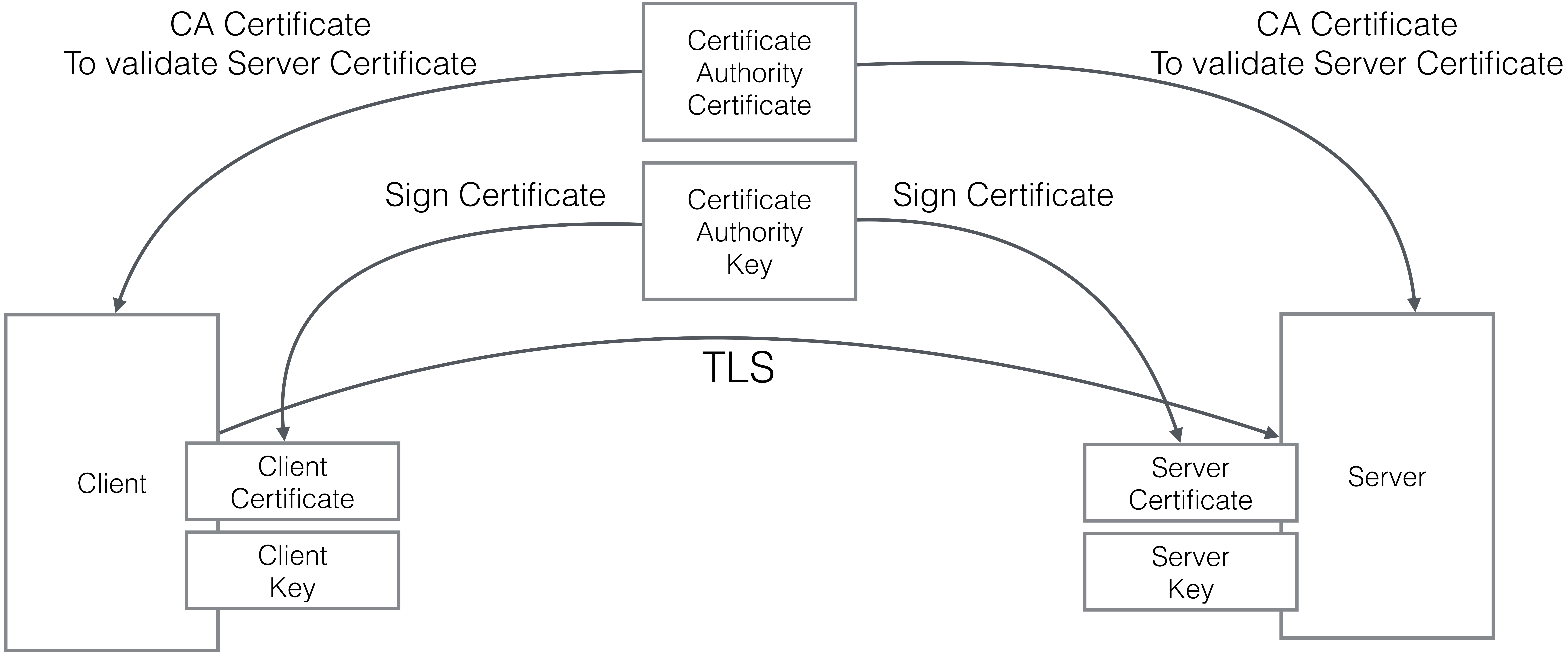
# Let’s encrypt

We don’t have access to the CA Key


Let’s Encrypt
Unsigned Server Certificate (CSR)
ACME* (port 80)
Sign Certificate based on Certificate Signing Request (CSR)
http://domain.com/.well-known/acme-challenge/<TOKEN>
Root Certificates
TLS
Client
Server Certificate
Server Key
Server

*Automatic Certificate Management Environment

# Mutual TLS



CA Certificate
To validate Server Certificate

Certificate Authority Certificate

CA Certificate
To validate Server Certificate

Sign Certificate

Certificate Authority Key

Sign Certificate

TLS

Client

Client Certificate

Client Key

Server Certificate

Server Key

Server