Lecture 4
Back-End Development

Web Servers,

Back-end Abstractions,

Building APIs

Web Servers

Hardware and server software

Serving static and dynamic resources

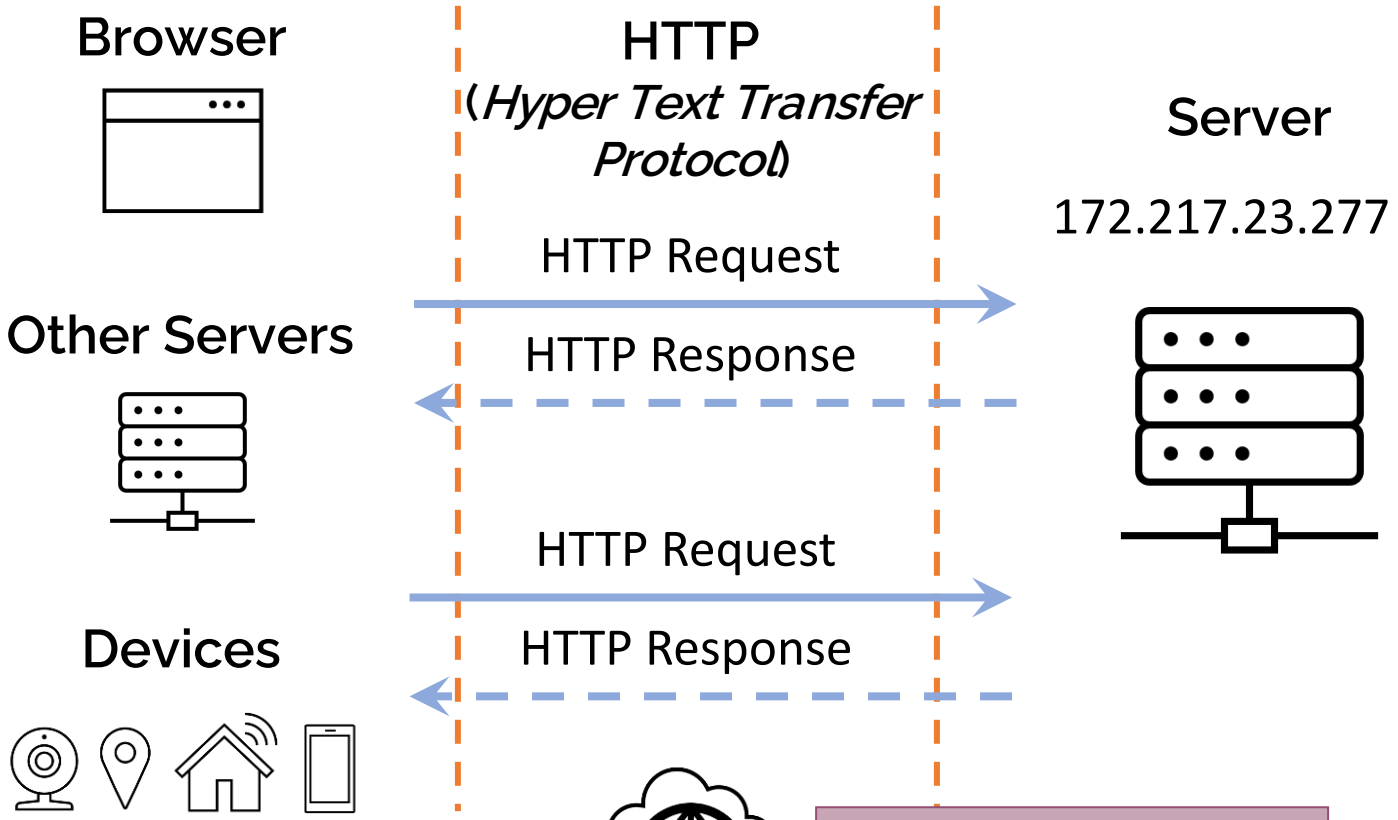
High Level Web Overview

- Servers wait for requests
- They serve web resources

Client https://www.google.com

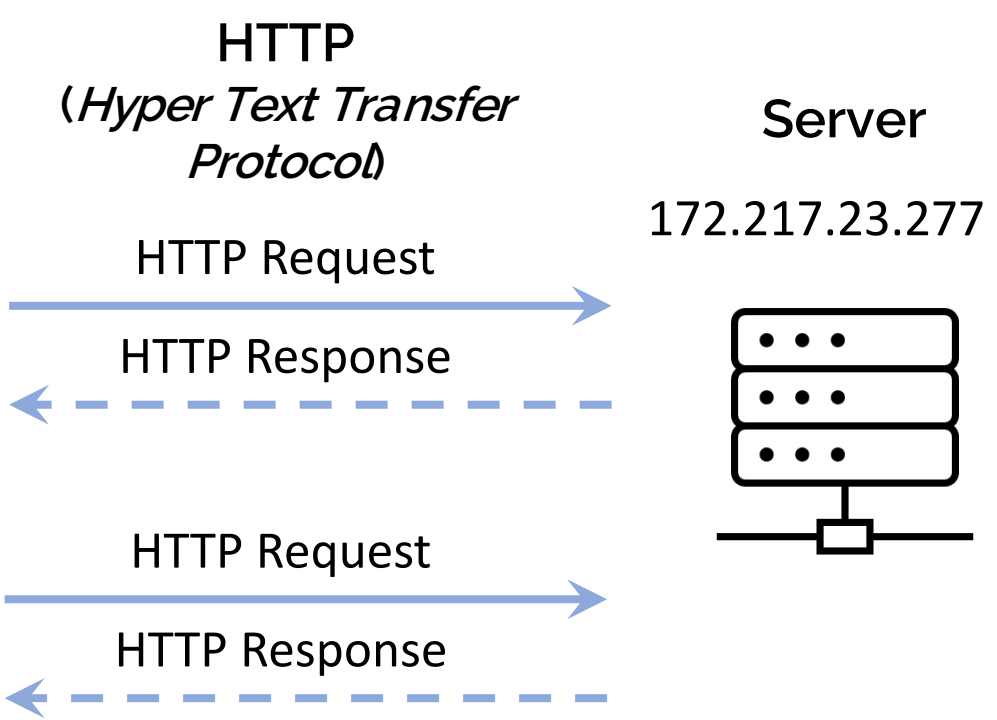
http://www.google.com ? 172.217.23.277

- **Domain Name System (DNS)**
  - translating hostname to IP address



- Multiple layers and proxies on the Internet

Web Server

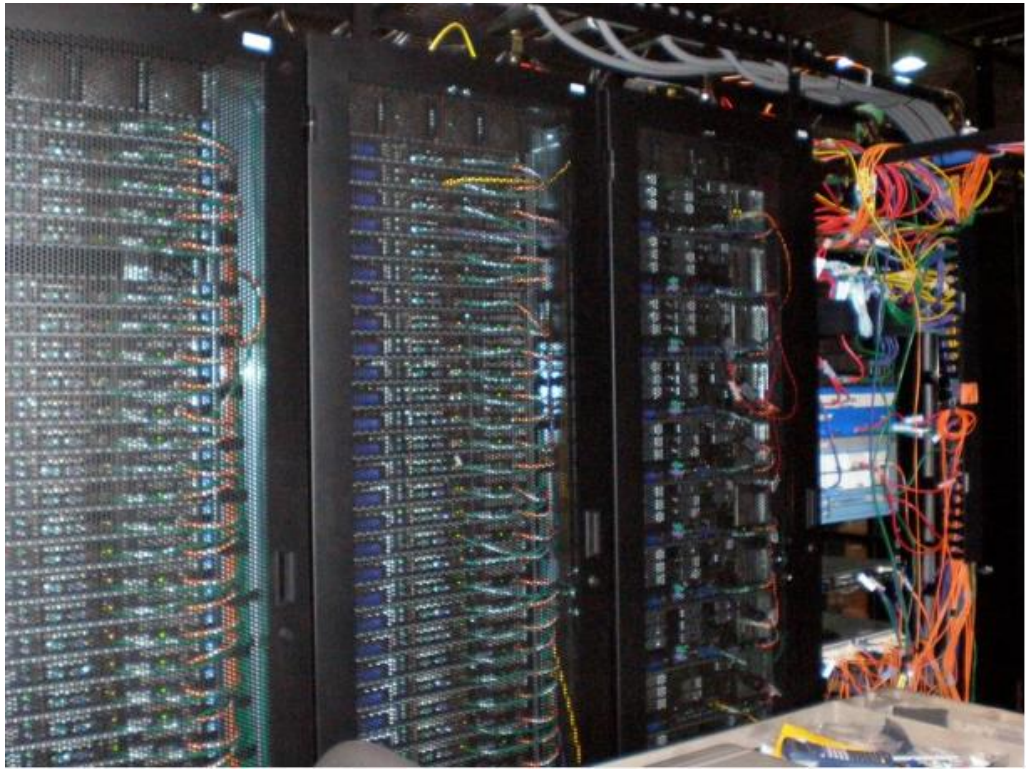


"Web Server" is an ambiguous term:

- **Hardware**: A computer ("server") connected to the internet (or any network)
- **Software**: A program running on a computer/server that accepts HTTP requests over a specific port and answers with HTTP responses

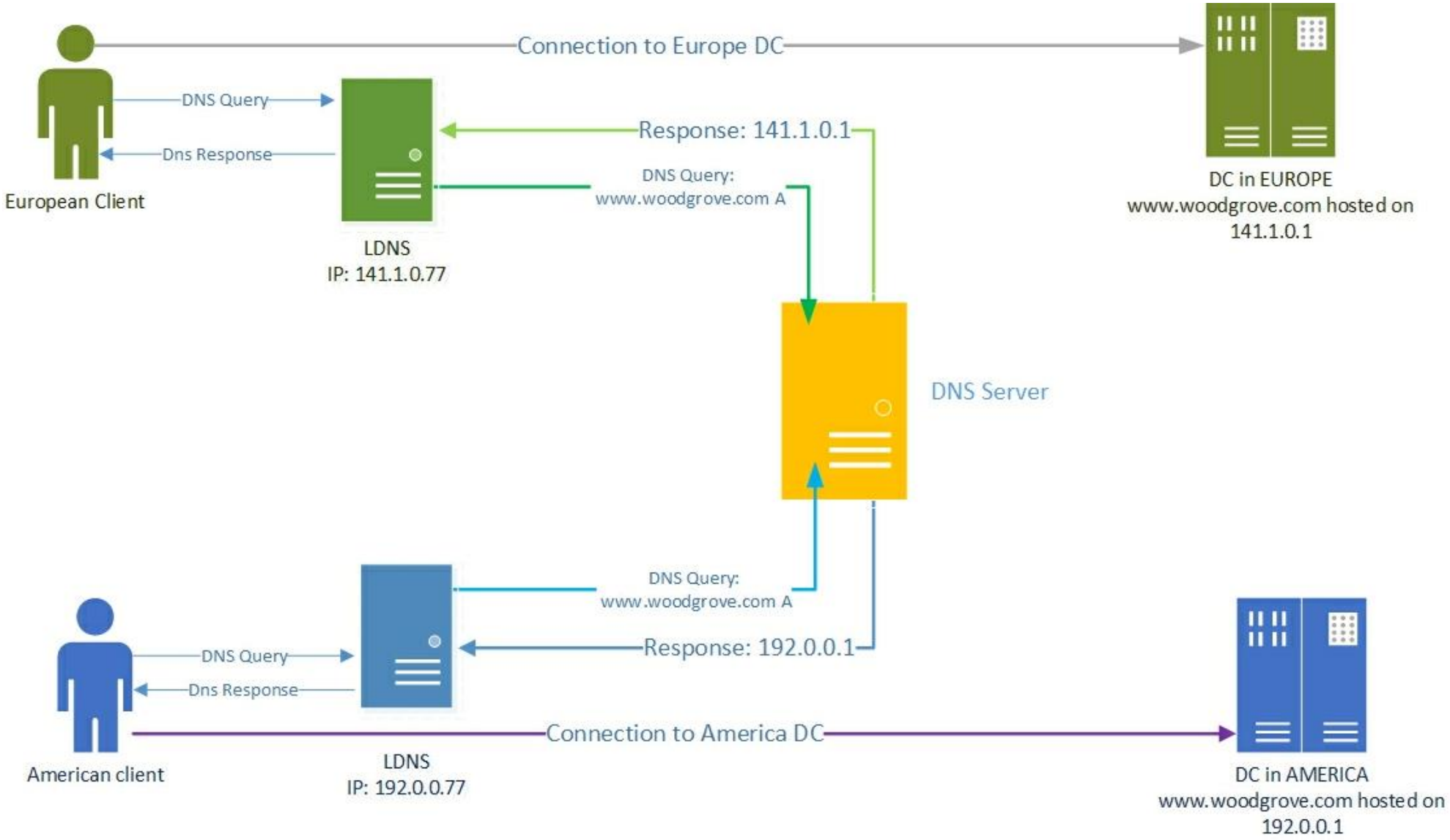
Web Server - Hardware

- **Hardware**: A computer ("server") connected to the internet (or any network)
- Properties of contemporary web servers
  - Part of large datacenters
  - Latency is geographically dependent, so web servers are often geographically distributed (works through, e.g., DNS)
  - **Virtual servers**: Physical servers can host many virtualized (web) servers
- Can also be your own computer (localhost)



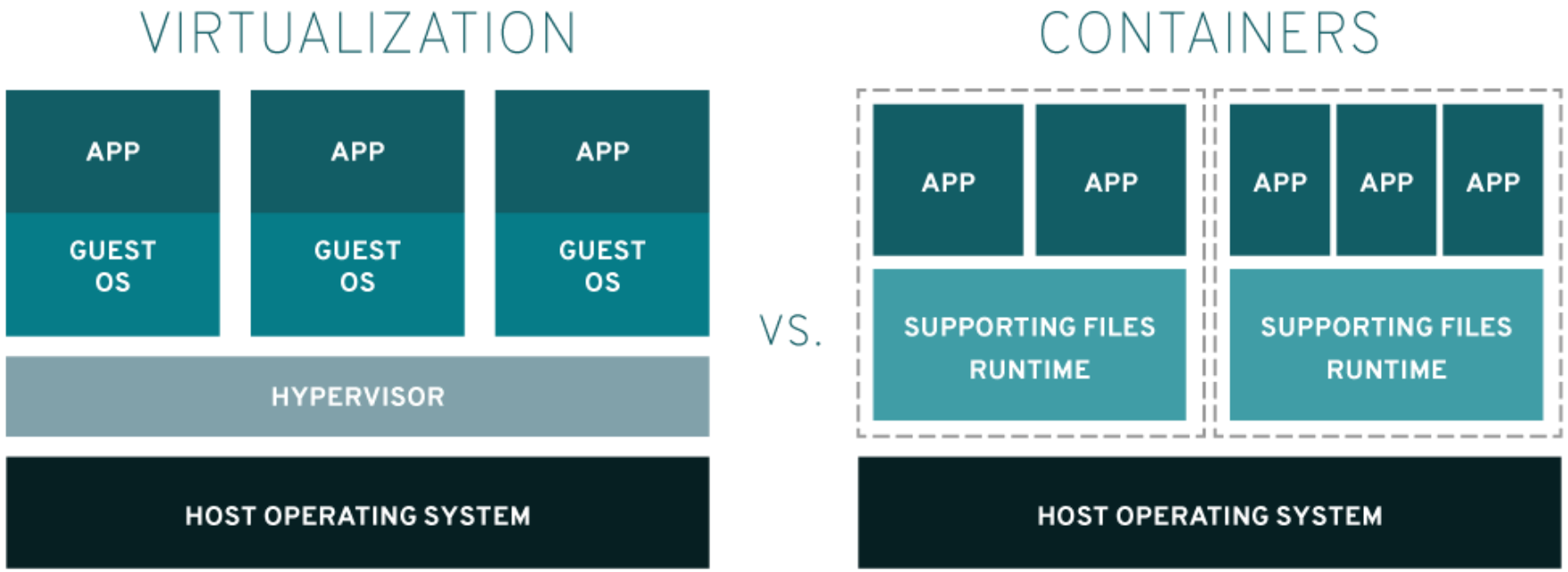
Web Server - Geo DNS

- Geo-location based Serving through DNS – proximity & policies



Web Server - Virtual Servers and Containers

- One physical server can host multiple **virtual servers** and/or **containers**

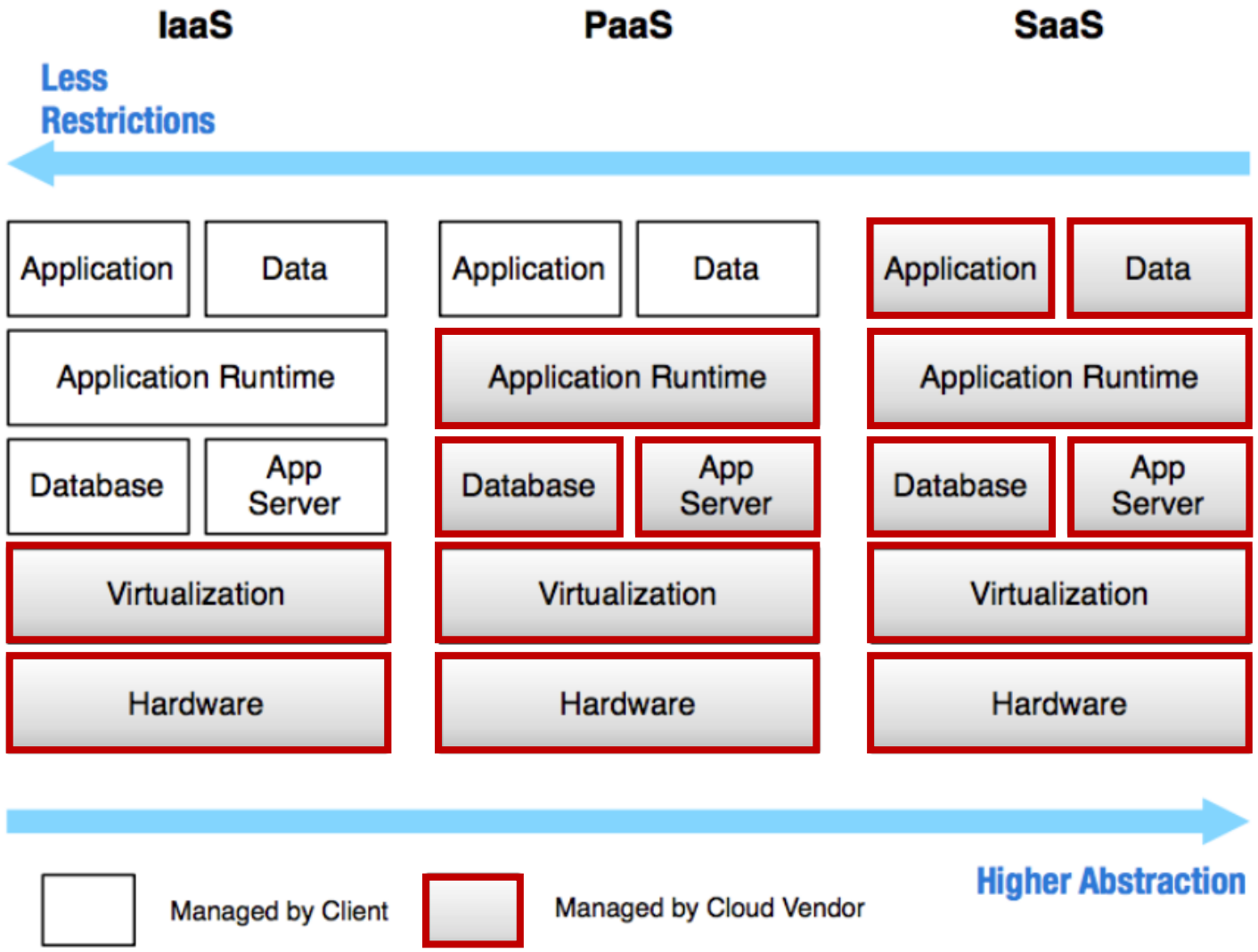


- Enables multiple virtual instances of different OSs to run in isolation through "hypervisor"
- Hypervisors divide physical resources so that virtual servers can use them and "translates" kernel operations

- "feel" like virtual machines, but are not virtualized
- provide lightweight process isolation (through cgroups) but share the Host OS kernel
- do not offer the same security boundaries

Web Server - Cloud

- The “cloud” enables provisioning of computational resources over an API



Web Server - Cloud

- Infrastructure as a service (IaaS) [***E.g.*** AWS EC2, Google Compute Engine]
  - API-driven infrastructure (web servers) at scale
  - Provides the ability to write a script that automates retrieving new (virtual) server capacity
- Platform as a Service (PaaS) [***E.g.*** Heroku, CloudFoundry, App Engine]
  - Managed application runtimes (e.g., web servers) that are built on top of IaaS for scalability
  - Underlying infrastructure (server) is abstracted away, configuration can provide directives
    - Deploy web applications by providing directives on process to start or providing container
- Software as a Service (SaaS) [***E.g.*** Salesforce, Office 365, Adobe Creative Cloud]
  - Delivers software applications over the internet, eliminating the need for users to install, maintain, and manage the software locally.
  - Typically offered on a subscription basis, allowing users to pay for the software on a recurring basis
- Function as a Service (FaaS) [***E.g.*** AWS Lambda, Azure Functions, Google Cloud Functions]
  - **aka Serverless computing** - allows developers to execute individual functions or pieces of code in response to events without managing the underlying infrastructure
  - The cloud provider automatically scales and allocates resources as needed

Web Server - Cloud

- Container as a Service (CaaS) [***E.g.*** Amzon: EKS/ ECS, GCP: GKE/ Run, Azure: AKS, ACI]
  - Enables developers to deploy, manage, and scale containerized applications without managing the underlying infrastructure
- Database as a Service (DBaaS) [***E.g.*** Amazon RDS, Google Cloud SQL, Azure Database]
  - Provides managed database services without the need for users to install, configure, and manage databases
- Backend as a Service (BaaS) [***E.g.*** Firebase, AWS Amplify, Kinvey]
  - Offers pre-built backend services for mobile and web applications
  - Includes features like user authentication, databases, and cloud storage
- Desktop as a Service (DaaS) [***E.g.*** Amazon WorkSpaces, Azure Virtual Desktop, VMware Horizon Cloud]
  - Delivers virtual desktops hosted on a cloud infrastructure, enabling remote access to desktop environments
  - Provides flexibility and accessibility for users, reducing the need for physical desktop hardware
- Identity as a Service (IDaaS) [***E.g.*** Okta, Azure Active Directory, Auth0]
  - Offers identity and access management services, providing authentication & authorization services
  - Enables organizations to manage user identities securely and ensure appropriate access controls

Web Server - Software

- A program running on a computer/server that accepts HTTP requests over a specific port and answers with HTTP responses
- Web Server/HTTP Server
  - Makes resources accessible over a URL and HTTP/S
    - (standard ports 80 and 433)
  - Starting a web server on local computer makes it accessible over
    - http://localhost
    - http://127.0.0.1
  - Maps path component of URL to
    - static asset on the file server
    - dynamically rendered resources
  - Often incorporates some functionality for caching and session handling

https://localhost:3000/**members/rackets?year=2020**

Path component + query parameters

Web Server - Static Assets

- Serving static assets from the file system
  - Web server automatically wraps static files with HTTP Response Headers
  - Static assets directly map URL path to relative part of the file system
    - They cannot react to other part of the request (e.g., query parameter)
  - MIME-Type is inferred through heuristics (e.g., file endings)
  - Example of common static files in web servers
    - HTML, CSS
    - JavaScript (for use in browser)
    - Media (Images, Video, Audio, etc.)

Example:

- Static assets made available at path **/var/www/public_html** on the server
- If we determine [this is configurable] **http://localhost/static/js/search.js** to be a request for static assets we could return **/var/www/public_html/js/search.js**

Web Server - Dynamic Resources

- Dynamic Resources
  - Executing programs in a server side programming language on the server
  - Dynamic resources can react to complete HTTP request (including header information)
    - Path and Query Parameters
    - HTTP Method (GET, POST, PUT, …)
    - Content Negotiation (Accept: application/json)
    - …
  - System output is treated as the complete HTTP response (including headers)
  - However, many programming languages offer library support for basic HTTP related functions and provide abstractions (e.g., for dealing with response headers)

Web Server - Examples

- Apache/httpd with CGI (Common Gateway Interface)
  - One of the earliest methods of providing dynamic scripting
- nginx
  - Reverse proxy and web server
- Node.js Web Server


```
const http = require('http');

const requestListener = function (req, res) {
  res.writeHead(200);
  res.end('Hello, World!');
}

const server = http.createServer(requestListener);
server.listen(8080);
```

Backend Abstractions

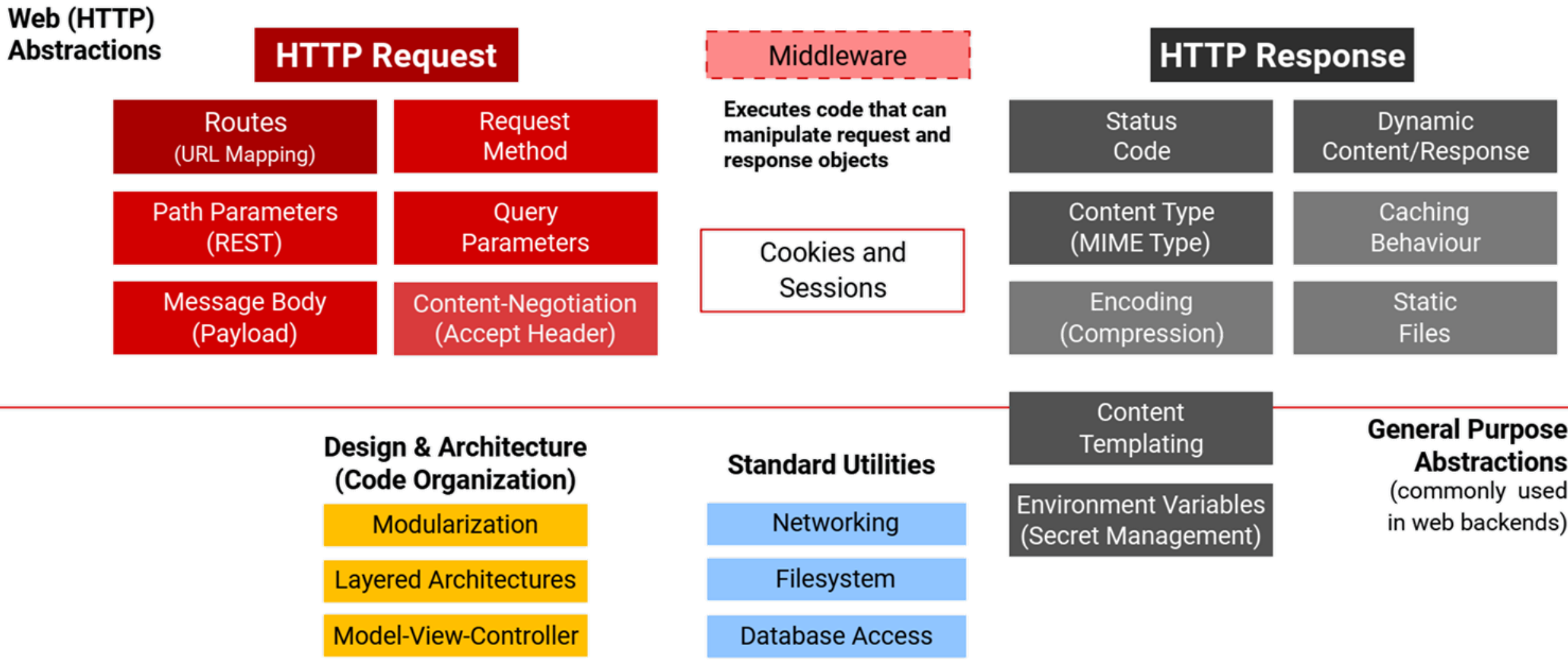
Building backend services

Case studies in Node.js

Node/Express

- Node.js
  - a JavaScript runtime environment that runs Chrome's V8 engine outside of the browser
  - is event driven (listening for requests) and provides facilities for synchronous and asynchronous computation
- NPM (Node Package Manager)
  - manages dependencies of external JavaScript packages, hosted in a package repository called npm registry
- Express.js
  - a web framework for Node.js that provides backend abstractions

Concepts/ Abstractions for Web Service Backends



Routes (URL Mapping)

```
const express = require('express');
const app = express();

app.get('/hello', function(req, res) {
    res.send('Hello World!');
});

const port = 3000;

app.listen(port, function() {
    console.log(`Waiting for requests on Port ${port}!`);
});
```

Request and Response Objects

```
app.put('/recipes/:id', (req, res) => {
    const recipeId = req.params.id;
    const hasImage = req.query.hasImage == 'true';

    const recipe = Recipes.find(recipeId, hasImage);

    if(!recipe) {
        return res.sendStatus(404);
    }

    const payload = req.body;

    recipe.update(payload);
    res.send({updateSuccess : recipeId});
});
```

Message Body as structured object (key-value pairs)

JavaScript objects are automatically serialized as JSON when sending the response. Could also use `res.json(obj)`

Middleware

- Middleware functions can manipulate request and response objects for every request-response cycle
- They are also provided a next() function that invokes the next middleware function in the chain (order matters)

```
// for parsing application/json
app.use(express.json());
// for parsing HTML form data
// application/x-www-form-urlencoded
app.use(express.urlencoded({ extended: true }));
...
const payload = req.body;
...
```

Message Body as structured object (key-value pairs)

Message body only possible because middleware intercepted the request, classified and parsed the message, and then set **`req.body`**

Cookies and Sessions

- Cookies are the consequence of the stateless nature of the HTTP protocol paired with the desire of still establishing some notion of association between client and server
- Cookies can be set by both client and server as part of HTTP headers and are transmitted with every request/response cycle

Sessions use cookies to store a unique identifier. The associated session data is stored on the server (either in-memory or in persistent storage)

```
routes.get('/', async (req, res) => {
  let sessionId = req.cookies.sessionId;
  ...
  if(!sessionId) {
    ...
    res.cookie('sessionId', sessionId);
  }
  ...
}
```

Cookies parsed through middleware from request header

Cookie written into response header

Environment Variables & Secret Management

- Environment variables provide a standard way for configurability and provide a strict way of separating configuration from code
- There are also several other ways to pass configuration to the program (pass parameters, read from configuration file, etc.)

```
const port = process.argv.length >= 3 ? +process.argv[2] : 3000;
```

Pass as parameter to process

```
const db = require('db')
db.connect({
  host: process.env.DB_HOST,
  username: process.env.DB_USER,
  password: process.env.DB_PASS
})
```

Parameters come from environment provided by the operating system

Templating

- Templates (aka views) provide separation between program logic and output.
- Template engines replace variables in static template files and control structures (conditionals and loops) with values passed from the program

```
app.set('view engine', 'pug')
...
routes.get('/', async (req, res) => {
  res.render('users', { title: 'Users',
    heading: 'List of users', users: getUsers() });
}
```

Enabled by middleware concept

PUG Template - users.pug

```
html
  head
    title= title
  body
    h1= heading
    div#container
      - for user in users
        div.user= user.email
```

Output for rendered response

```
<html>
    <head>
        <title>Users</title>
    </head>
    <h1>List of users</h1>
    <div id="container">
        <div class="user">
          jane.doe@tuwien.ac.at
        </div>
        <div class="user">
          jack.bauer@tuwien.ac.at
        </div>
    </div>
</html>
```

Templating – *Jinja*

```python
@app.route('/')
def index():
    # Data to be passed to the template
    title = 'Jinja Template Example'
    welcome_msg = 'Welcome to our website!'
    show_posts = True
    posts = [
        {'title': 'Post 1', 'content': 'Content 1'},
        {'title': 'Post 2', 'content': 'Content 2'},
        {'title': 'Post 3', 'content': 'Content 3'},
    ]
    # Render the template with data
    return render_template('example_template.html',
                           title=title,
                           welcome_msg=welcome_msg,
                           show_posts=show_posts,
                           posts=posts)
```

```jinja
<!DOCTYPE html>
<html>
<head>
    <title>{{ title }}</title>
</head>
<body>

<h1>{{ welcome_message }}</h1>

{% if show_posts %}
    <h2>Recent Posts:</h2>
    <ul>
        {% for post in posts %}
            <li><strong>{{ post.title }}</strong>: {{ post.content }}</li>
        {% endfor %}
    </ul>
{% else %}
    <p>No posts to display.</p>
{% endif %}

</body>
</html>
```

Networking (HTTP)

- Almost every programming language has multiple libraries of dealing with network and HTTP requests
- Node also has node-fetch, that has the same functionality and familiar contract as the one in the browser API

```
const fetch = require('node-fetch');
...
const response = await fetch(objectRequestUrl(objectID));
if(response.status !== 200) {
    console.log('Could not find object with id' + objectID);
    return false;
}
const object = await response.json();
```

Persistent Storage (Files)

```
const fs = require('fs');
const path = require('path');

const destinations = JSON.parse(fs.readFileSync(path.join(__dirname, '../res/data.json')));
```

Filesystem utilities have synchronous and asynchronous API in Node

- *Beware when deploying to the cloud*: Writing to the local filesystem on a server can lead to data loss if the server is ephemeral. The same goes for "local" databases.
- Use so-called backing services for attached resources you can access from an API for persistent storage

Modules

- Modules in Node.js are not the same as ES6 Modules for JavaScript in the browser
- But in similar ways, it enables code organization through file-based separation and encapsulation

routes/artworks.js

Relative Path

```
const met = require('../utils/met.js');

const artworks = met.search('van gogh');
```

Everything in module files not in `module.exports` is private/implementation detail

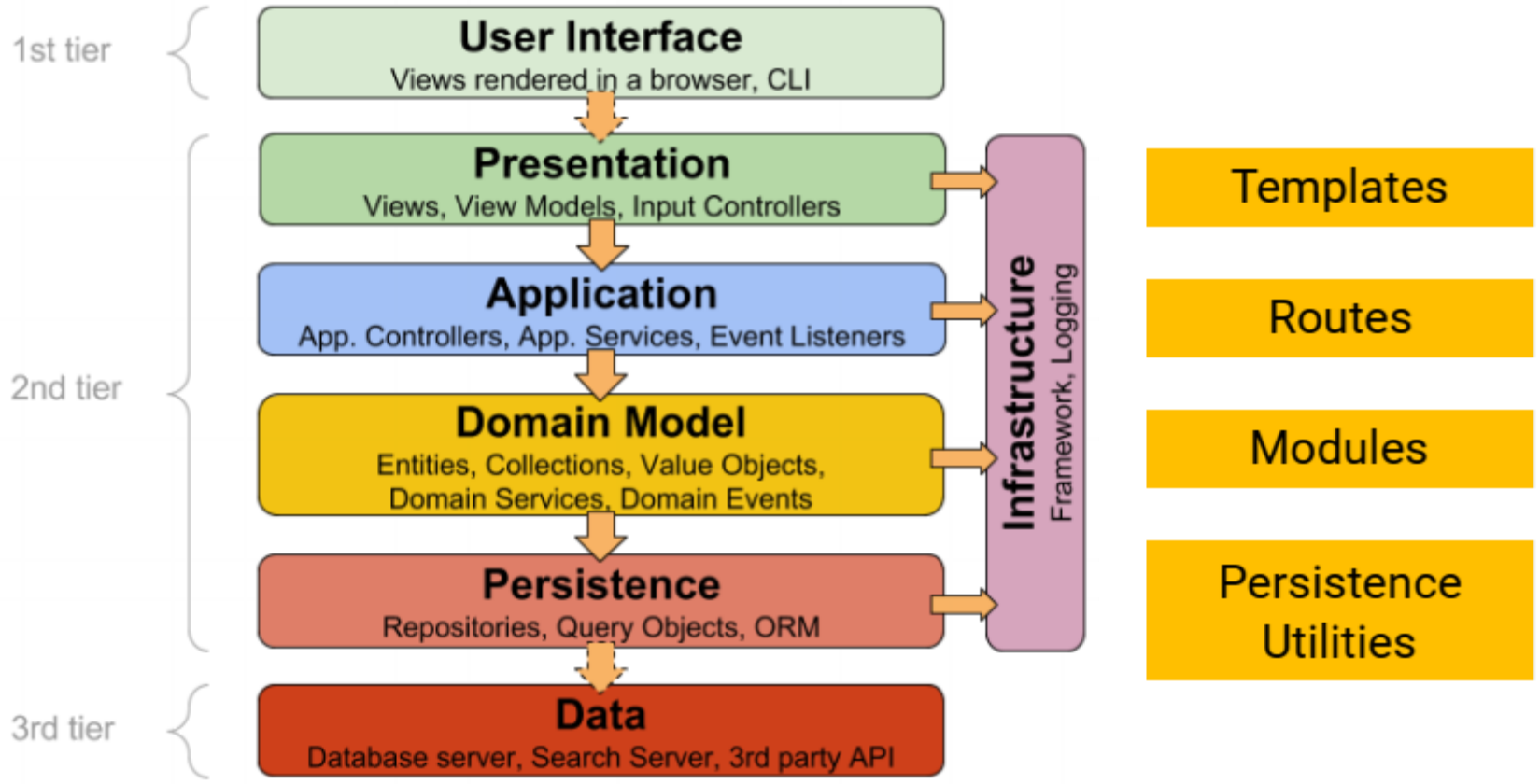
util/met.js

```
const search = async (term, max=100) => { ... }
...
module.exports.search = search;
```

Elements of `module.exports` become part of `met` object

Layered Architectures

- Layering in web service backends can be facilitated through existing abstractions



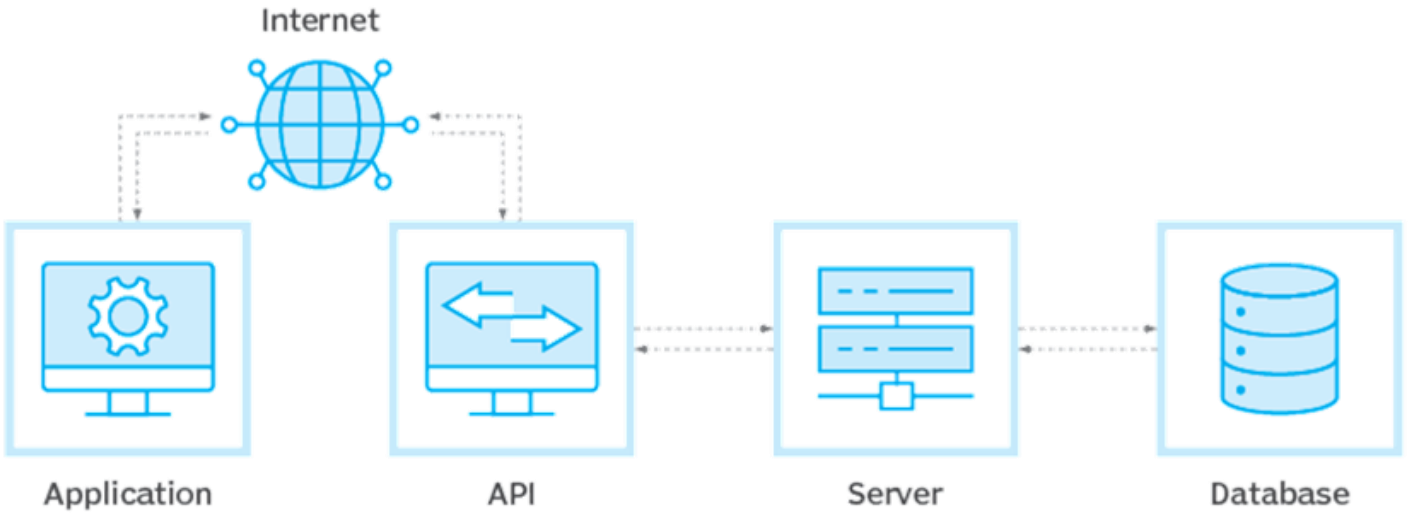
Building APIs

REST APIs

Django, Laravel, Express

What is an API?

- An **application programming interface (API)** defines
  - the rules that you must follow to communicate with other software systems
- Developers expose or create APIs so that other applications can communicate with their applications programmatically
  - E.g., the timesheet application exposes an API that asks for an employee's full name and a range of dates.
    - When it receives this information, it internally processes the employee's timesheet and returns the number of hours worked in that date range.



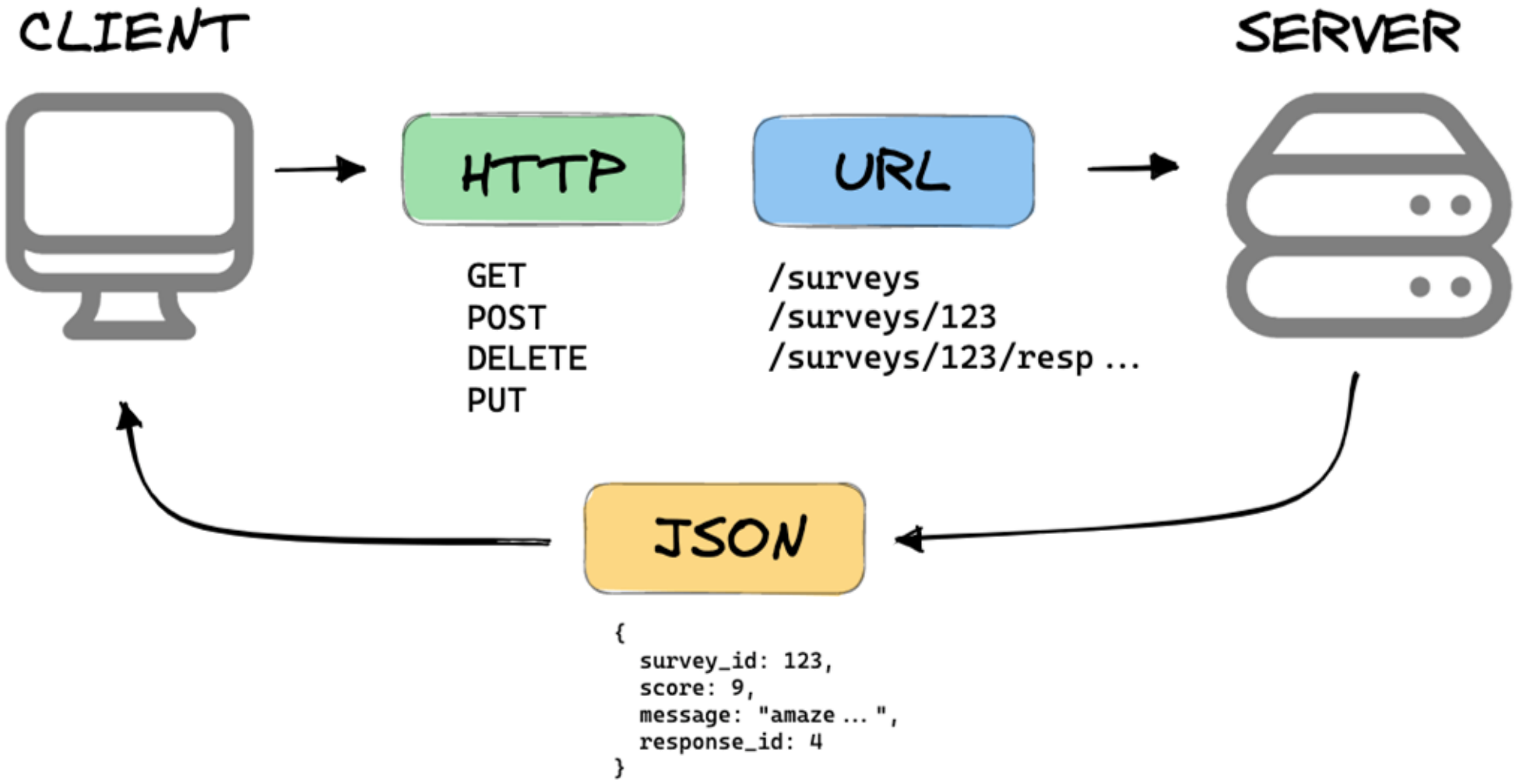
What is an API?

- A web API can be thought as a gateway between clients and resources on the web
  - **Clients** are users who want to access information from the web
    - The client can be a person or a software system that uses the API
      - ***E.g.***, developers can write programs that access weather data from a weather system
      - **Or**, you can access the same data from your browser when you visit the weather website directly
  - **Resources** are the information that different applications provide to their clients
    - Resources can be images, videos, text, numbers, or any type of data
    - The machine that gives the resource to the client is also called the server
    - Organizations use APIs to share resources and provide web services while maintaining security, control, and authentication
    - In addition, APIs help them to determine which clients get access to specific internal resources

REST

- **Representational State Transfer (REST)** is a software architecture that imposes conditions on how an API should work
- Principles of REST architectural style:
  - Uniform interface
    - Server transfers information in a standard format
  - Statelessness
    - Server completes every client request independently of all previous requests
  - Layered system
    - Clients can connect to authorized intermediaries and still receive response
    - Server can pass on requests to other servers
  - Cacheability
    - Support for storing responses on client /intermediary to improve server response time
  - Code on demand
    - Servers can temporarily extend or customize client functionality by transferring software programming code to the client

REST APIs


CLIENT
SERVER
HTTP
URL
GET
POST
DELETE
PUT
/surveys
/surveys/123
/surveys/123/resp ...
JSON
{
survey_id: 123,
score: 9,
message: "amaze ... ",
response_id: 4
}

Rest APIs

Resource	POST create	GET read	PUT update	DELETE delete
/dogs	**create a new dog**	**list dogs**	bulk update dogs	delete all dogs
/dogs/bo	error	**Show Bo**	**if exists update Bo If not error**	**Delete Bo**

REST APIs

- Associations
  - `GET /owners/bob/dogs`
  - `POST /owners/bob/dogs`
- Complex variations
  - `/dogs?color=red&state=running&location=park`
- Pagination
  - `/dogs?limit=25&offset=50`
- Search
  - `/search?q=fluffy+dog`
  - `/owners/bob/dogs/search?q=fluffy`
- Return Data
  - `/dogs?fields=name,color,location`

REST APIs

- Scalability
  - Optimizes client-server interactions
  - Statelessness removes server load
  - Well-managed caching partially/completely eliminates some client-server interactions
- Flexibility
  - Support total client-server separation
  - Simplify & decouple various server components so that each part can evolve independently
  - Platform or technology changes at the server app do not affect the client app
- Independence
  - Independent of the technology used
  - Can write both client and server applications in various programming language
  - Can change the underlying technology on either side without affecting the communications

Express (Node.js/JavaScript)

```
|-- controllers/
|-- models/
|-- public/
|   |-- css/
|   |-- js/
|   |-- images/
|-- routes/
|-- views/
|-- app.js
|-- package.json
|-- .env
```

- **controllers/**: Controllers handling HTTP requests and business logic
- **models/**: Data models and database interactions
- **public/**: Publicly accessible assets
- **css/, js/, images/**: CSS files, JavaScript files, and images
- **routes/**: Route definitions
- **views/**: Views (templates) for rendering HTML
- **app.js**: Entry point for the Express application
- **package.json**: Node.js package configuration
- **.env**: Environment configuration

Django (Python)

```
project_name/
|-- app_name/
|   |-- migrations/
|   |-- templates/
|   |-- static/
|   |-- __init__.py
|   |-- admin.py
|   |-- apps.py
|   |-- models.py
|   |-- tests.py
|   |-- views.py
|-- project_name/
|   |-- __init__.py
|   |-- settings.py
|   |-- urls.py
|   |-- asgi.py
|   |-- wsgi.py
|-- manage.py
```

- **app_name/**: Contains the application-specific components
- **migrations/**: Database migration files
- **templates/**: HTML templates
- **static/**: Static files (CSS, JavaScript, images)
- **__init__.py**: Python package initializer
- **admin.py**: Admin configuration for Django Admin
- **apps.py**: Application-specific configuration
- **models.py**: Database models
- **tests.py**: Unit tests
- **views.py**: Views handling HTTP requests
- **project_name/:** Contains the project-level components
- **__init__.py**: Python package initializer
- **settings.py**: Project settings and configurations
- **urls.py**: URL patterns for the project
- **asgi.py** and **wsgi.py**: Entry points for ASGI and WSGI
- **manage.py**: CLI utility for managing Django projects

Laravel (PHP)

```
app/
|-- Console/
|-- Exceptions/
|-- Http/
|   |-- Controllers/
|   |-- Middleware/
|-- Models/
|-- Providers/
config/
database/
|-- factories/
|-- migrations/
|-- seeders/
public/
resources/
|-- lang/
routes/
|-- web.php
tests/
|-- Feature/
|-- Unit/
|-- CreatesApplication.php
|-- TestCase.php
.env
artisan
```

- **app/**: Contains the application-specific components
- **Console/**: Artisan commands
- **Exceptions/**: Exception handling
- **Http/**: Controllers and middleware
- **Models/**: Eloquent models
- **Providers/**: Service providers
- **config/**: Configuration files
- **database/**: Database-related files
- **factories/**: Factory files for generating model instances
- **migrations/**: Database migration files
- **seeders/**: Database seeder files
- **public/**: Publicly accessible assets (CSS, JavaScript, images)
- **resources/**: Views and language files
- **lang/**: Language files
- **routes/**: Route definitions
- **web.php**: Web route definitions
- **tests/**: PHPUnit test files
- **Feature/**: Feature tests
- **Unit/**: Unit tests
- **CreatesApplication.php** and **TestCase.php**: Test-related setup
- **.env**: Environment configuration
- **artisan**: Artisan CLI for managing Laravel applications