Django pt. 1: Intro, Setup, and Views

Kianoosh Abbasi

CSC309 Winter 2022

So far

- How web works

  Client/server – request/response - HTTP

- HTML

  Tags: headers, inputs, etc.

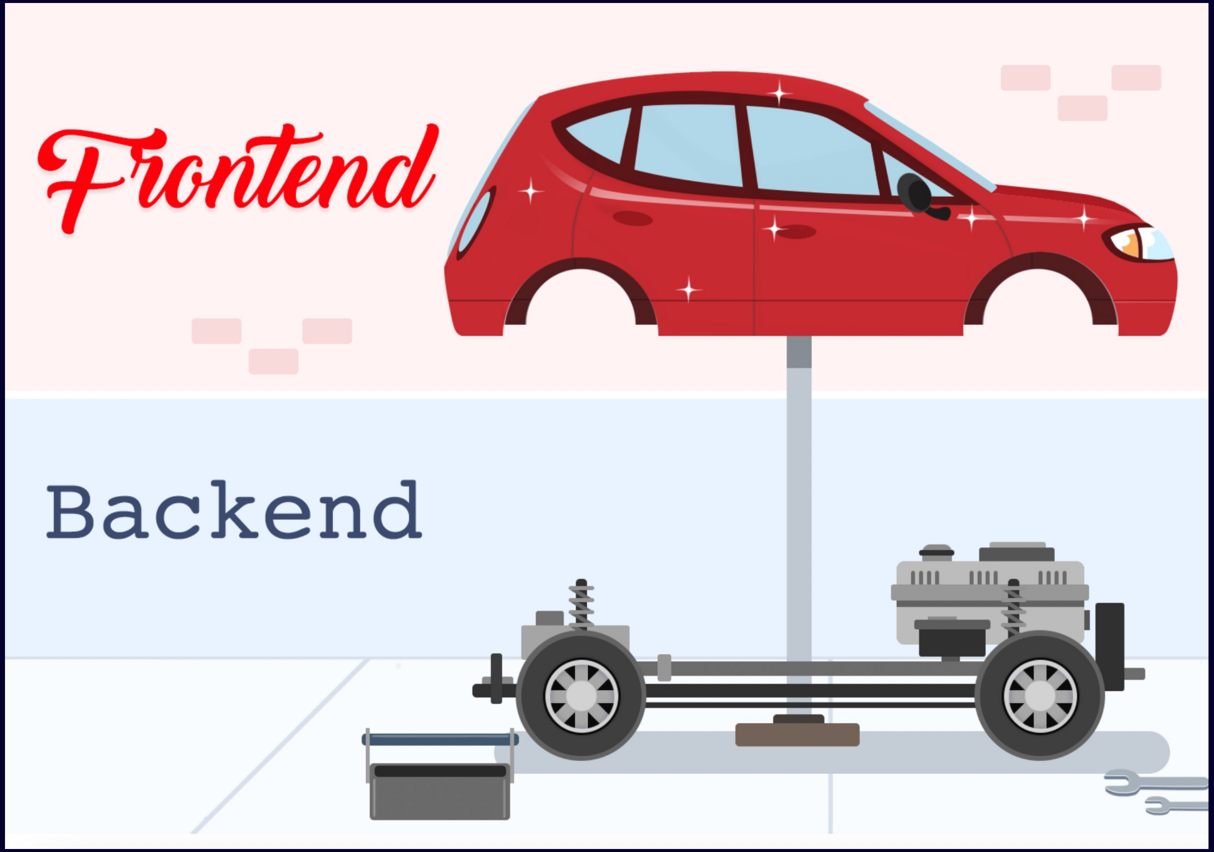
- CSS Styles

  Selectors, spacing, layout

This session

- Back-end development & frameworks
- Python projects
  Virtual environment & pip
- Django
  Setup, simple views, forms, templates

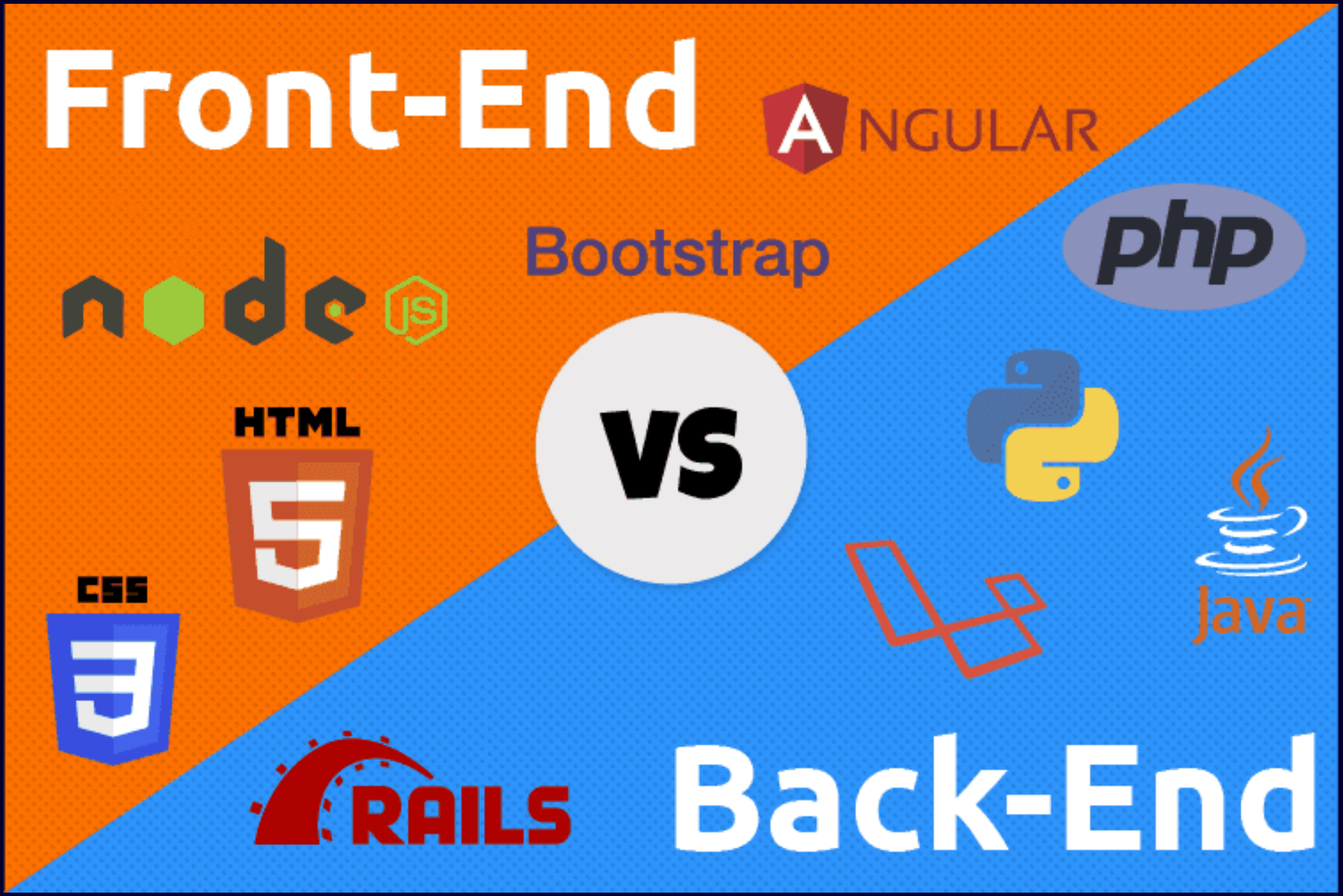
Web development



Source: blog.back4app.com

Source: https://www.reddit.com/r/ProgrammerHumor/comments/m187c4/backend_vs_frontend/

Web development



Source: nimapinfotech.com

Front-end development

- What user can see
  - User interface (UI)
  - User experience (UX)

- What is run on the client-side
  - HTML/CSS rendering
  - Javascript codes

Back-end development

- What user can’t see

  What does it even mean?

- All logic and processes that happen behind the scene

- At the server-side!

- Processing the requests, creating responses, data management

Web server

- Listens on specified port(s)

- Handles incoming connections
  - Generates a response
  - Fetches a file
  - Forwards them to corresponding applications

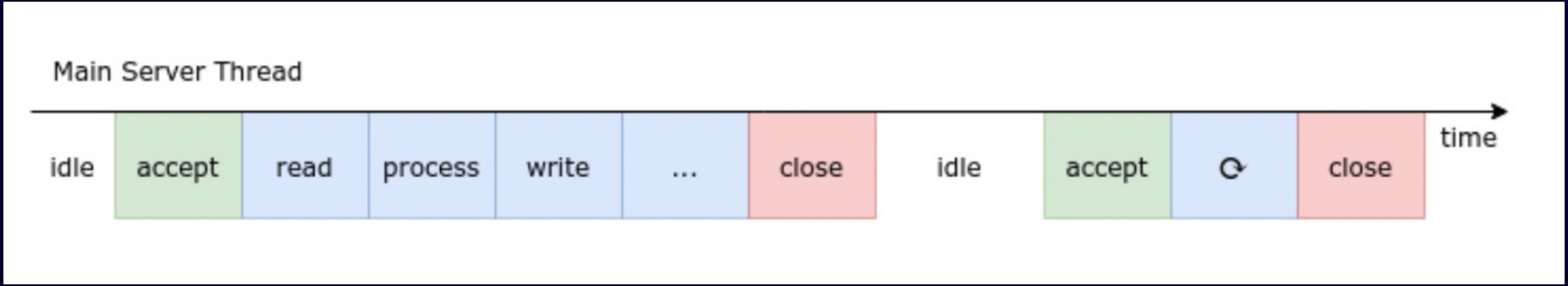
- Load balancing, security, file serving, etc

- Examples: Apache, Nginx

Web server architecture

Visit https://levelup.gitconnected.com/event-driven-servers-a-intuitive-study-6d1677818d2a

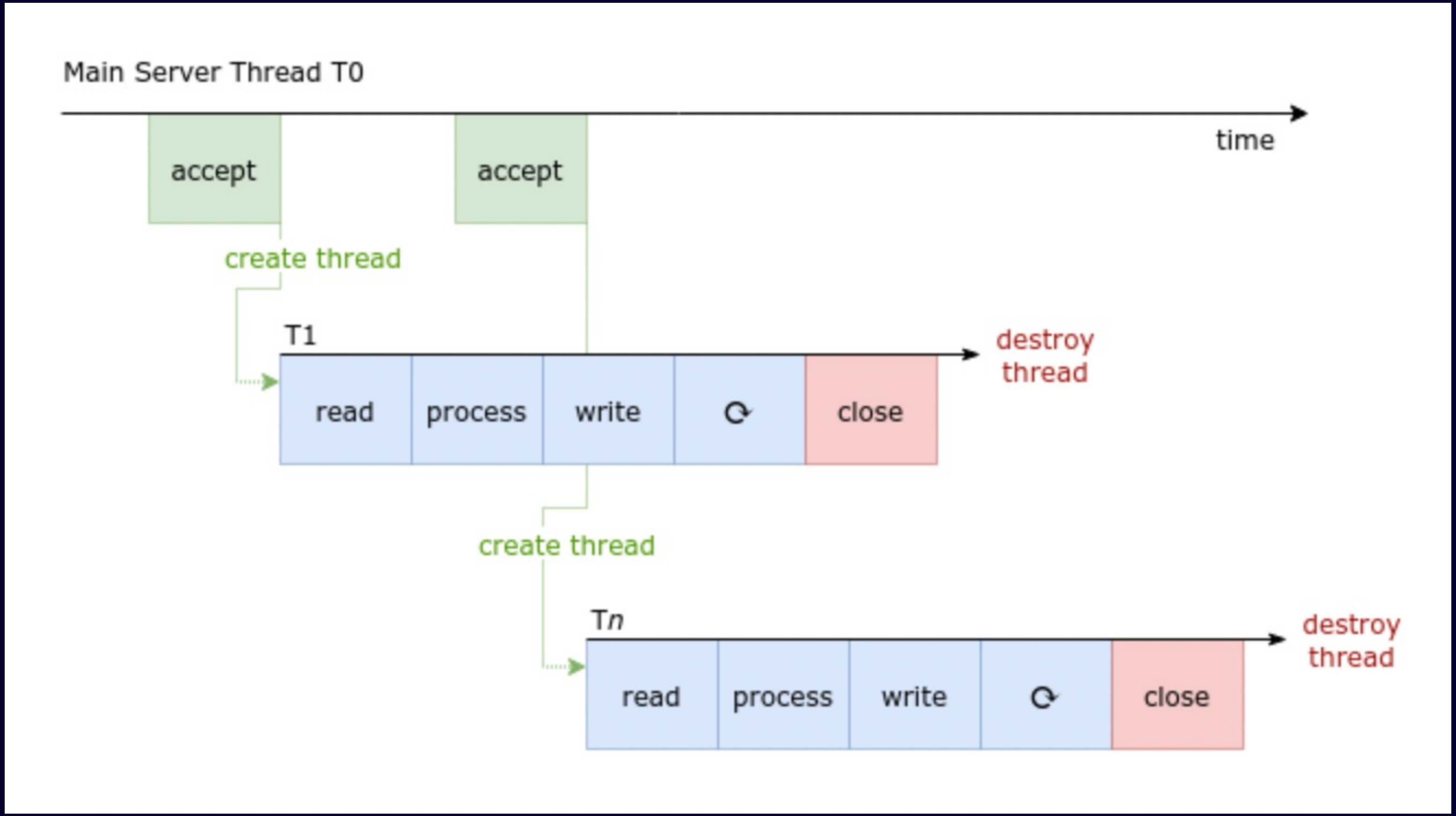
Single-threaded server



Caveat: Processes only one open connection at a time!

Web server architecture

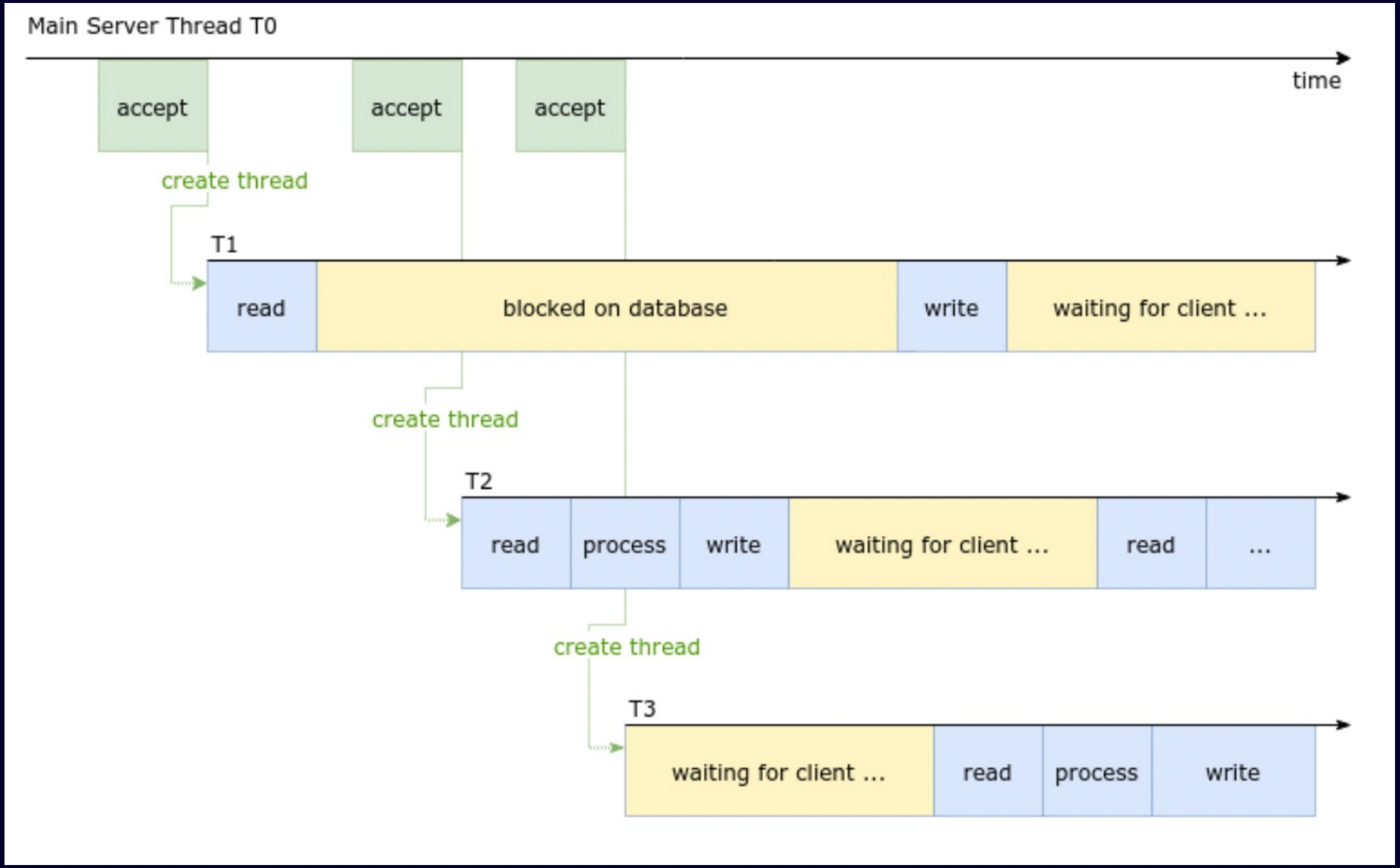
Multi-threaded server



Caveat: 1000 concurrent connections -> 1000 threads!

Web server architecture

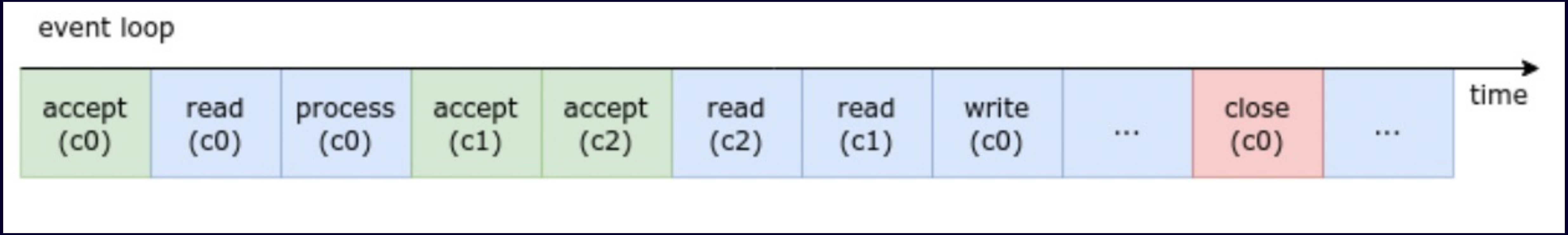
Multi-threaded server



Another caveat: threads may just be idle!

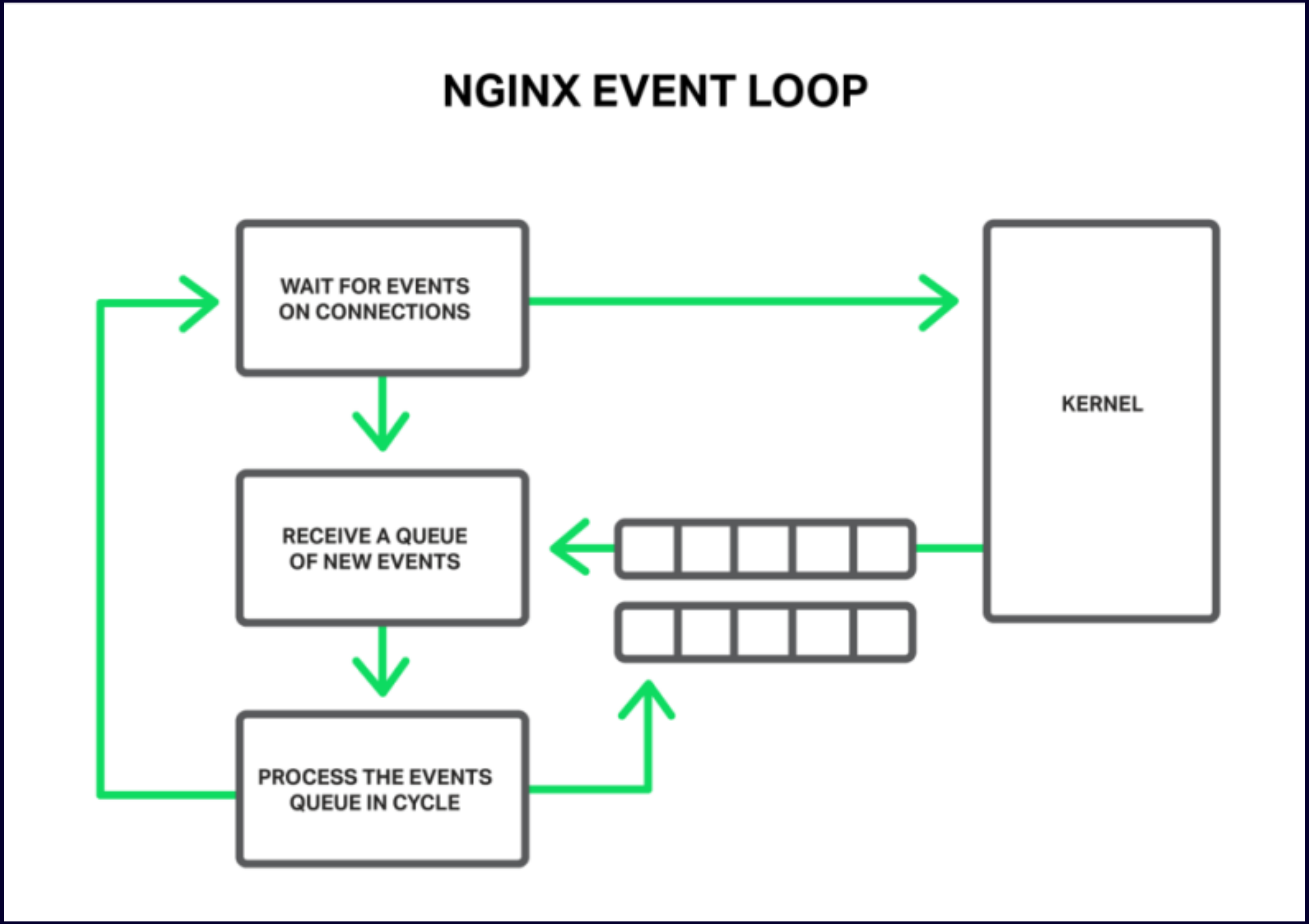
Web server architecture

Event-driven server



Events are queued, handled by the main thread, and sent to the corresponding processes/threads

Web server architecture


NGINX EVENT LOOP
WAIT FOR EVENTS ON CONNECTIONS
RECEIVE A QUEUE OF NEW EVENTS
PROCESS THE EVENTS QUEUE IN CYCLE
KERNEL

Backend frameworks

- Doing everything from scratch?

  Listen on a port, process http requests (path, method, headers, body), retrieve data from storage, process data, create the response

- Not really a good idea!

- A lot of frameworks are out there!

  A lot of things are pre-implemented

Backend frameworks

- PHP: Laravel, CodeIgniter
- Python: Django, Flask, FastAPI
- Javascript: ExpressJS, Spring
- Ruby: Ruby on Rails

Concept is more important than framework!

Django: a backend framework with Python

Source: www.geeksforgeeks.org

Python projects

- A big project needs several different packages!
- Python’s package manager: pip
- Command: `pip install Django`
- At the global scope, use pip3 instead of pip

Python projects

- Packages should NOT be shared between projects
- Even Python itself should NOT
- Reason: package versions and dependencies conflict
- Each project must have an isolated environment

Virtual Environment

- A package called virtualenv
  `pip3 install virtualenv`

- Creates a directory with its own Python, pip, and packages

- Command: `virtualenv -p /usr/bin/python3.9 venv`
  or use `` `which python3.9` `` instead of the full path

Virtual Environment

- Activate the environment

  `source venv/bin/activate`

- To test, type `which python` or `which pip`

- Packages will not be installed globally

- Easy to reset: just delete the entire venv folder

  Have a requirements.txt file to list all needed packages

Creating a Django project

- Create the folder, environment, and install Django
- Command: `django-admin startproject <name> .`
- Creates the skeleton for your work
- https://docs.djangoproject.com/en/3.2/intro/tutorial01

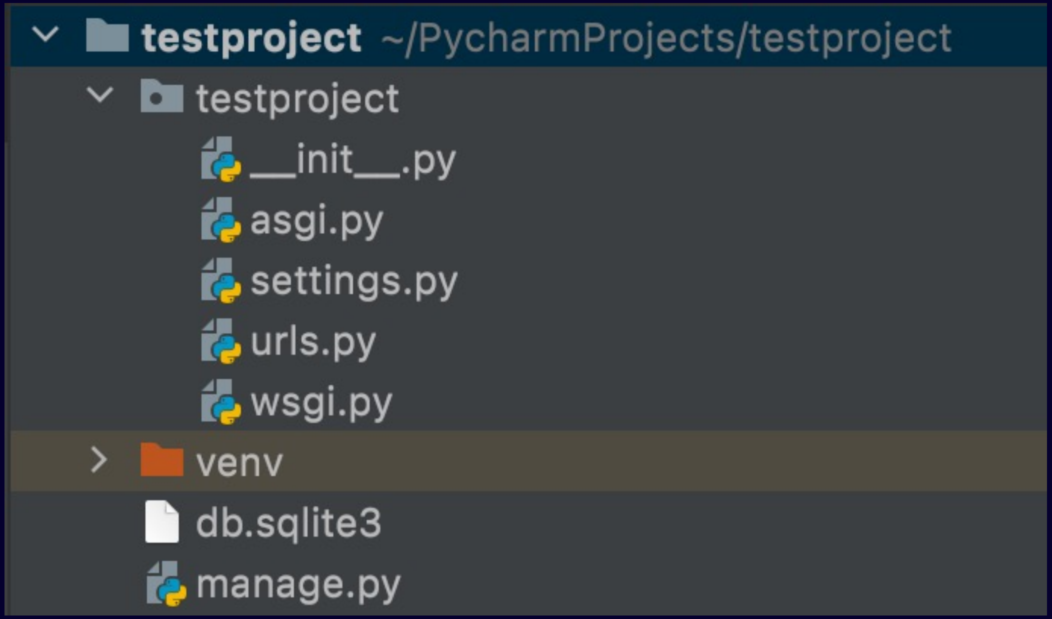
Project structure

- Run the project

  `python manage.py runserver`

- Access the website from http://localhost:8000

- Django has a small development server



```
testproject ~/PycharmProjects/testproject
  testproject
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
  venv
  db.sqlite3
  manage.py
```

Taking requests

- View: a piece of code that runs upon a request to a specific endpoint (URL)

- Can be a function or a class

- How to create a new view?

  First, you need to create an app

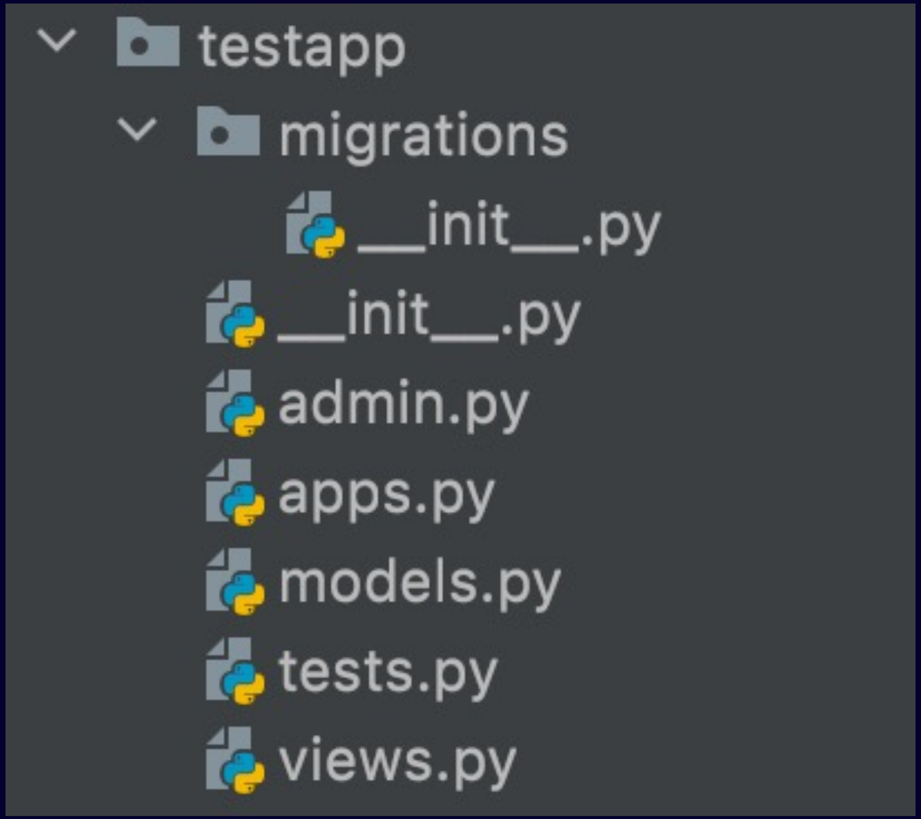
Django apps

- Django is intended for big projects
  Where tens or hundreds of views could exist
- Project's logic is organized by apps
- Each app takes care of a set of related views, urls, or models
- Example: one app for accounts, one for transactions, one for products, etc.
- Create a new app: `./manage.py startapp <name>`

App structure

- models.py, migrations, admin.py: next session

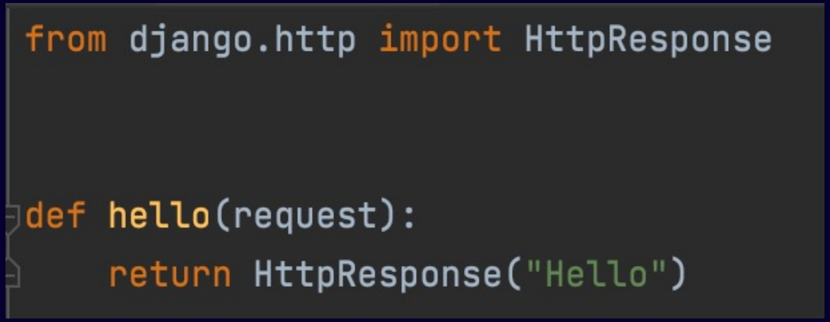
- ALWAYS add the app name to the end of INSTALLED_APPS in project's settings.py



```
testapp
    migrations
        __init__.py
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
```

Create a new view

- Just write a function in `views.py` that takes an argument: request
- Return an HttpResponse instance



```python
from django.http import HttpResponse


def hello(request):
    return HttpResponse("Hello")
```

Map a URL to the view

- Add a path to `urlpatterns`

  ```
  path('your/path', hello)
  ```

- Defining all urls in a single file is a terrible idea

  Makes the urls so messy and disorganized

- Solution: hierarchical urls based on apps

Hierarchical URL system

- Create a urls.py for each app
- Make a namespace for each app
- Main urls.py:
```
path('accounts/', include('accounts.urls'))
```
- App's urls.py:
```
path('', hello)
```
- Now access the page through http://localhost:8000/accounts/

More sophisticated views

- Receive arguments through the URL
  ```
  path('hello/<str:name>', hello)
  ```

- At the view function
  ```
  def hello(request, name):
  ```

- Extract request data
  ```
  request.method, request.GET, request.POST, request.headers
  ```

Exercise: Create a simple signup form

Form validation

- Email should be valid
- Password must be at least 8 characters
- Username must consist of lowercase letters and digits

- Can be checked at the front-end (a good UX)

- But it must always be checked at the backend as well

- User can always bypass front-end restrictions
  - Inspect element
  - Manual request

Form validation

- If data is invalid, an error can be returned
- Error 400: `HttpResponseBadRequest`
- Error 403: `HttpResponseForbidden`
- Error 404: `HttpResponseNotFound`

Form success

- On success, a redirect is often returned
  - Redirect to profile page or index page after log in
- Use `HttpResponseRedirect`
- Putting raw URLs is a very bad practice
- Django offers URL names

URL names

- Django separates the URLs users see from the URLs developers use

- Development URLs (aka named URLs) should be telling about project structure

- User URLs might change a lot

  Hard-coding them is a bad idea

URL names

- Add the name or namespace attribute to the paths

```
path('accounts/', include('accounts.urls',
namespace='accounts'))

path('', hello, name='hello')
```

- Add app_name to app's urls.py

- Redirect to `reverse('accounts:hello')`

  Can have args or kwargs

HTML Response

- Create a templates folder inside the app's directory
- Add an html file there: hello.html
- At view, return `TemplateResponse(request, 'hello.html')`
- Django standard: create a subdirectory with the same name as the app and put html files there
  - Template address would be `'<appname>/hello.html'`

Exercise: Serve the signup form from the Django server

Flow of forms

- The form and submission share the same endpoint
- If request's method is GET, the form itself is returned
- It it's POST, the submission is validated
  Don't forget to add `{% csrf_token %}` to the form
- If form is valid, a redirect is returned
- Otherwise, the form with errors will be returned!
  Not just a simple 400 error, which is a bad UX

Form errors

- Django templates are so dynamic!
  Data can be passed from the view to the template

- The `context` argument of `TemplateResponse`
  `context={'error': 'form is invalid’}`

- Access the variables at the template (html file)
  `<h3> {{ error }} </h3>`

- More on that next session!

This session

- Back-end development & frameworks
- Python projects
  Virtual environment & pip
- Django
  Setup, simple views, forms, templates

Next session

- MVC Design patterns
- Working with a database
  ORM and models
- Authentication
  User model, sessions, login
- Admin panel

Final notes

- Assignment 1 deadline is this Friday!
- Must have already formed project teams
- Phase1 deadline in two weeks
- Register for the first mentor session for phase1
- Go over Django tutorial at home
  https://docs.djangoproject.com/en/3.2/intro/tutorial01