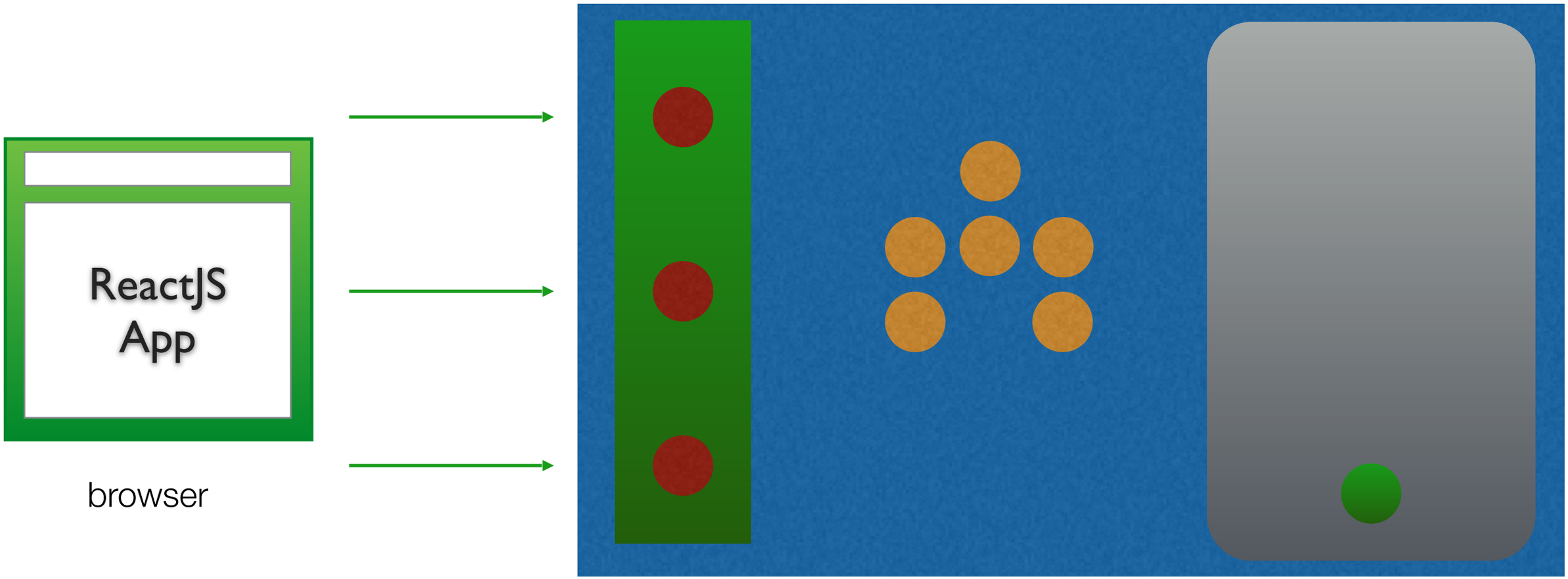
KAZAKH KBTU BRITISH
TECHNICAL
UNIVERSITY

WEB DEVELOPMENT

Lesson 10

Can you remember first lecture?

Web development terminology

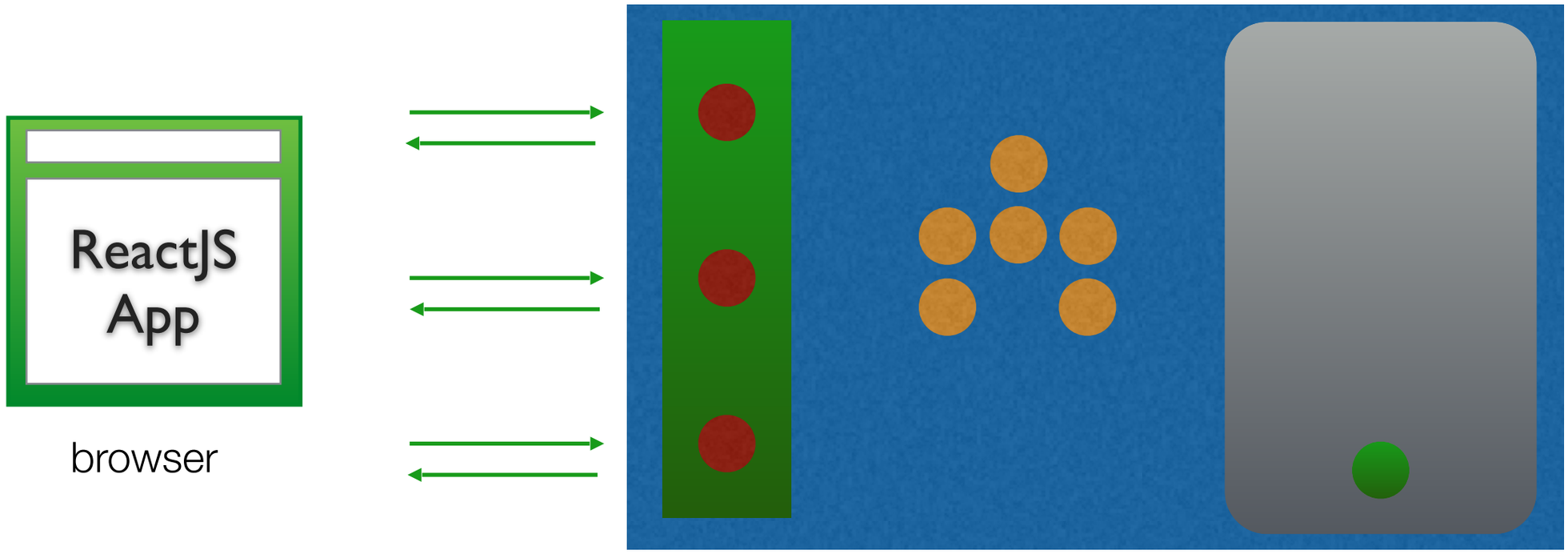


HTML
CSS
JavaScript

NGINX
Apache
IIS

Django
RoR
Laravel
MVC

Postgres
MongoDB
Oracle



Web server

NGINX
Apache
IIS
Unicorn
Thin
Puma

Server framework

Python
Django, Flask

Ruby
Rails, Sinatra

PHP
Laravel, Yii

Java
Spring, Play

C#
ASP.NET

Database

MySQL
Postgres
Microsoft SQL
SQLite
MongoDB
Oracle
Redis
Memcached
Cassandra

HTML
CSS
JavaScript

HAML
LESS SASS
CoffeScript
TypeScript

jQuery
React
Angular
Ember
Vue

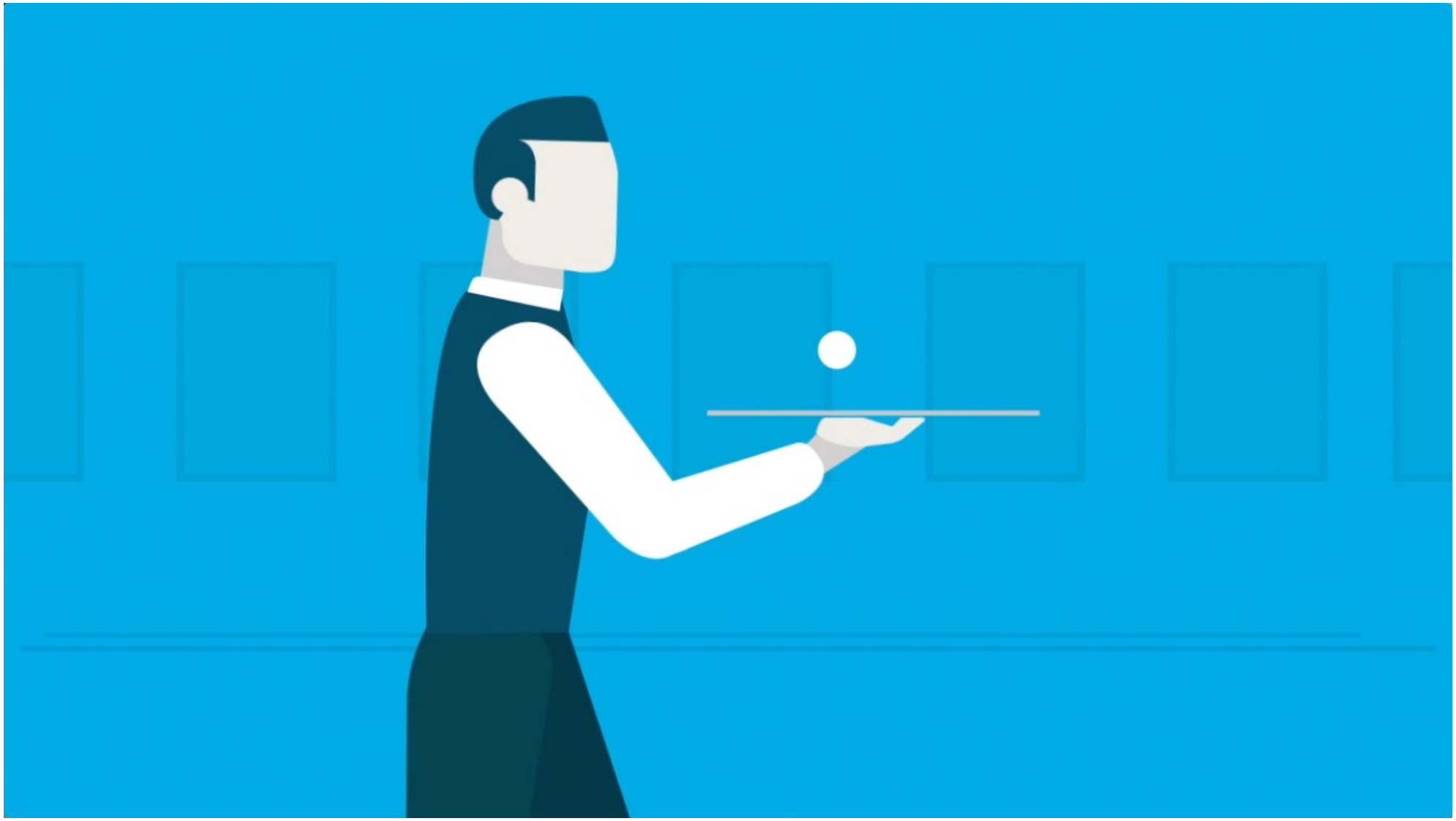
Hosting

AWS
Microsoft Azure
Heroku
ps.kz
hoster.kz

Back end frameworks

Django	Rails
Python	Ruby
MVT	MVC
Explicit is better than implicit	Convention over Configuration
beginners	seasoned professionals

What is API?



API — is like an artist performing on stage, and its users are the audience

RESTful API

1. REST (REpresentational State Transfer) — is an architectural style for developing web services

2. API (Application Program Interface) — is code that allows two software programs to communicate with each other

API endpoint for Companies

```
1. /getAllCompanies
2. /addNewCompany
3. /showCompanyDetail?id=23
4. /deleteCompany?id=23
```

The URL is a sentence, where resources are nouns and HTTP methods are verbs.

```
1. /companies          (GET)
2. /companies          (POST)
3. /companies/23       (GET)
4. /companies/23       (DELETE)
```

Data formats

- JSON

```
{
 "root": {
  "age": "18",
  "isStudent": "true",
  "name": "Nick"
 }
}
```

- XML

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <age>18</age>
   <isStudent>true</isStudent>
   <name>Nick</name>
</root>
```

- CSV

```
name,age,isStudent
Nick,18,true
```

Protocols

- TCP/IP — Transmission Control Protocol / Internet Protocol
  - communication among computers on Internet

    - HTTP — Hyper Text Transfer Protocol
      - Communicates with browsers to send web page packets

- HTTPS — Hyper Text Transfer Protocol Secure
  - HTTP with Secure Sockets Layer (SSL)

- FTP — File Transer Protocol
  - Used by FTP Clients to transfer file packets

HTTP response status codes

- 2xx — Success category
  - 200 Ok
  - 201 Created

- 3xx — Redirection Category
  - 304 Not Modified

- 4xx — Client Error Category
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found

- 5xx — Server Error Category
  - 500 Internal Server Error
  - 503 Service Unavailable

Python

Python is...

- Dynamic
- Interpreted
- Object-Oriented
- Exceptional
- Comfortable
- Readable
- Community

Interactive Shell

```
$ python
>>> print "Hello, world!"
Hello, world!
>>>
```

```
$ python3
>>> print("Hello, world!")
Hello, world!
>>>
```

Comments

```
# Best. Comment. Ever.
```

Booleans and Null

```
True

False

None
```

Strings

- `'Hello, world!'`
- `"Hello, world!"`
- `"""Hello,
  world!"""`
- `u"Hëllö, wörld!"`

String Operations

```
"foo" + "bar"

"foo"[0]

"foo"[:1]

"foo".upper()

"{0}: {1}".format("foo", "bar")

"{foo}: {bar}".format(foo=42, bar=11)

len(“foo”)
```

String Operations

```
"foo" + "bar" ==> "foobar"

"foo"[0] ==> "f"

"foo"[:1] ==> "f"

"foo".upper() ==> "FOO"

"{0}: {1}".format("foo", "bar") ==> "foo: bar"

"{foo}: {bar}".format(foo=42, bar=11) ==> "42: 11"

len(“foo") ==> "3"
```

Sequence Operation

```
[...][0]

[...][-1]

[...][:1]  # same as [...][0:1]
```

```
[...].append(7)

[...].pop()

len([...])
```

Dictionaries

```
{'key1': 'value1', 'key2': 'value2'}
```

Dictionary Operations

```
{...}['key1']

{...}.get('key2')

{...}.keys()

{...}.values()

{...}.items()

len({...})
```

Assignment & Comparison

```
foo = ‘bar’

foo == ‘baz'

foo != ‘baz'
```

```
foo is None

foo is not None
```

Flow Control

```
if expression:
      ...
elif expression:
      ...
else:
      ...
```

Flow Control

```
for item in sequence:

    if expression:
        continue
    else:
        break
```

Functions

```
def foo():
    return 42


def foo(bar):
    return bar


def foo(bar, baz=“fit”):
    return (bar, baz)
```

Classes

```
class Foo(object):
    def __init__(self, bar):
        self.bar = bar
```

Docstrings

```
"Modules can have docstrings."

class Foo(object):

    "Classes can have docstrings too."

    def __init__(self, bar):
        self.bar = bar
```

Exceptions

```
try:
    raise Exception("OH NOES!”)

except:
    log_error()
    raise

else:
    do_something_more()

finally:
    clean_up()
```

Namespaces

```
import logging

from datetime import timedelta

from decimal import Decimal as D


from models import Product
```

Style: PEP-8

- Four-space indents
- lower_case_methods
- CamelCaseClasses
- Line breaks around
  78-79 chars

Installing Packages

- easy_install: easy_install package
- pip: pip install package

Installing Packages

- Installed packages go into a site-packages directory in your Python lib

- But different programs may need different versions of packages…

- So we have virtual environments!

Virtual Environments

- virtualenv
- Creates an isolated Python environment with its own site-packages
- Install whatever you want without fouling anything else up

Activate the Virtual Environment

```
# Mac/Linux/etc...

$ virtualenv myenv
$ source myenv/bin/activate
```

```
# Windows

> python virtualenv myenv
> myenv/Scripts/activate.bat
```

What is Django?

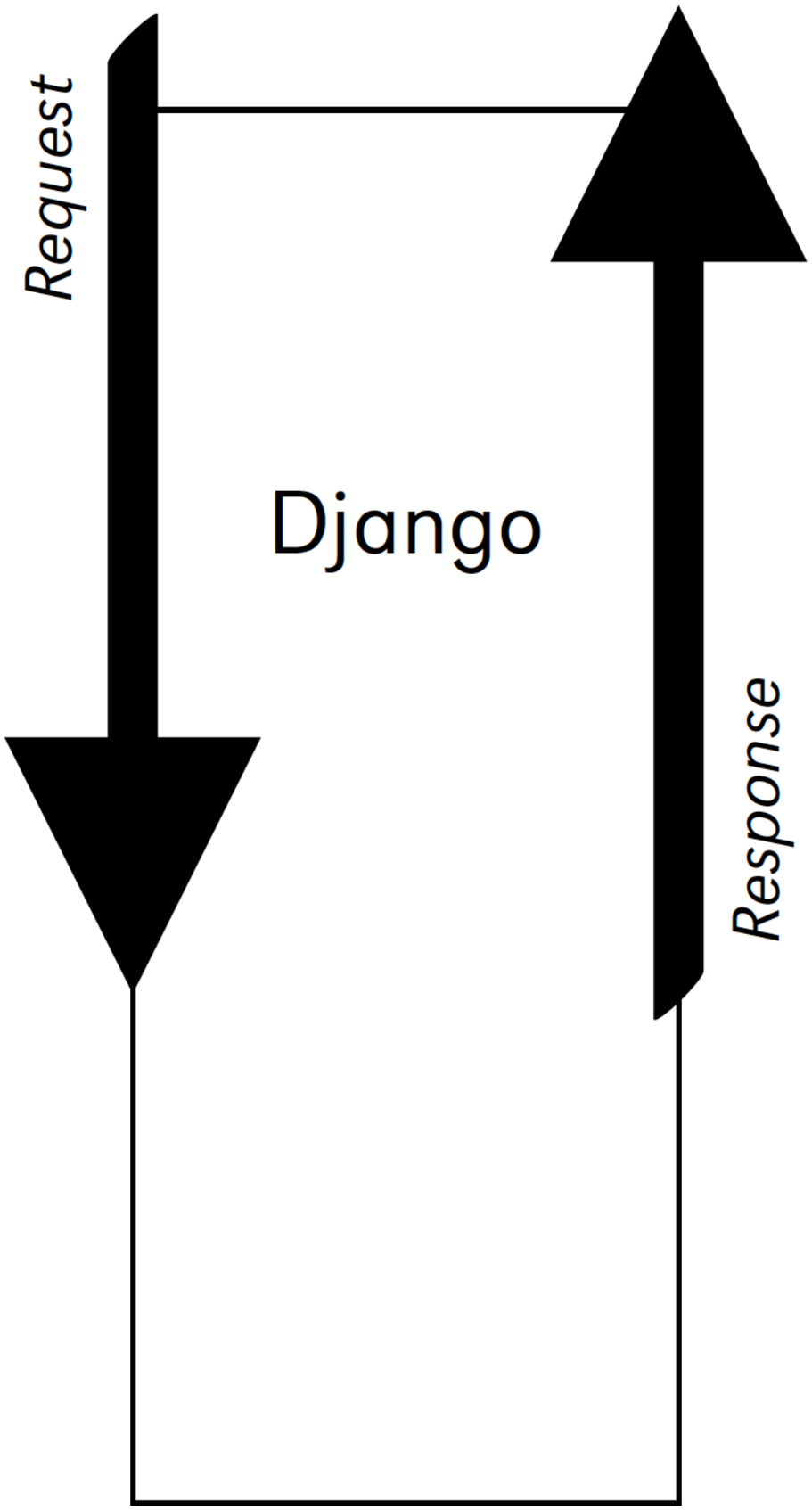
Django?

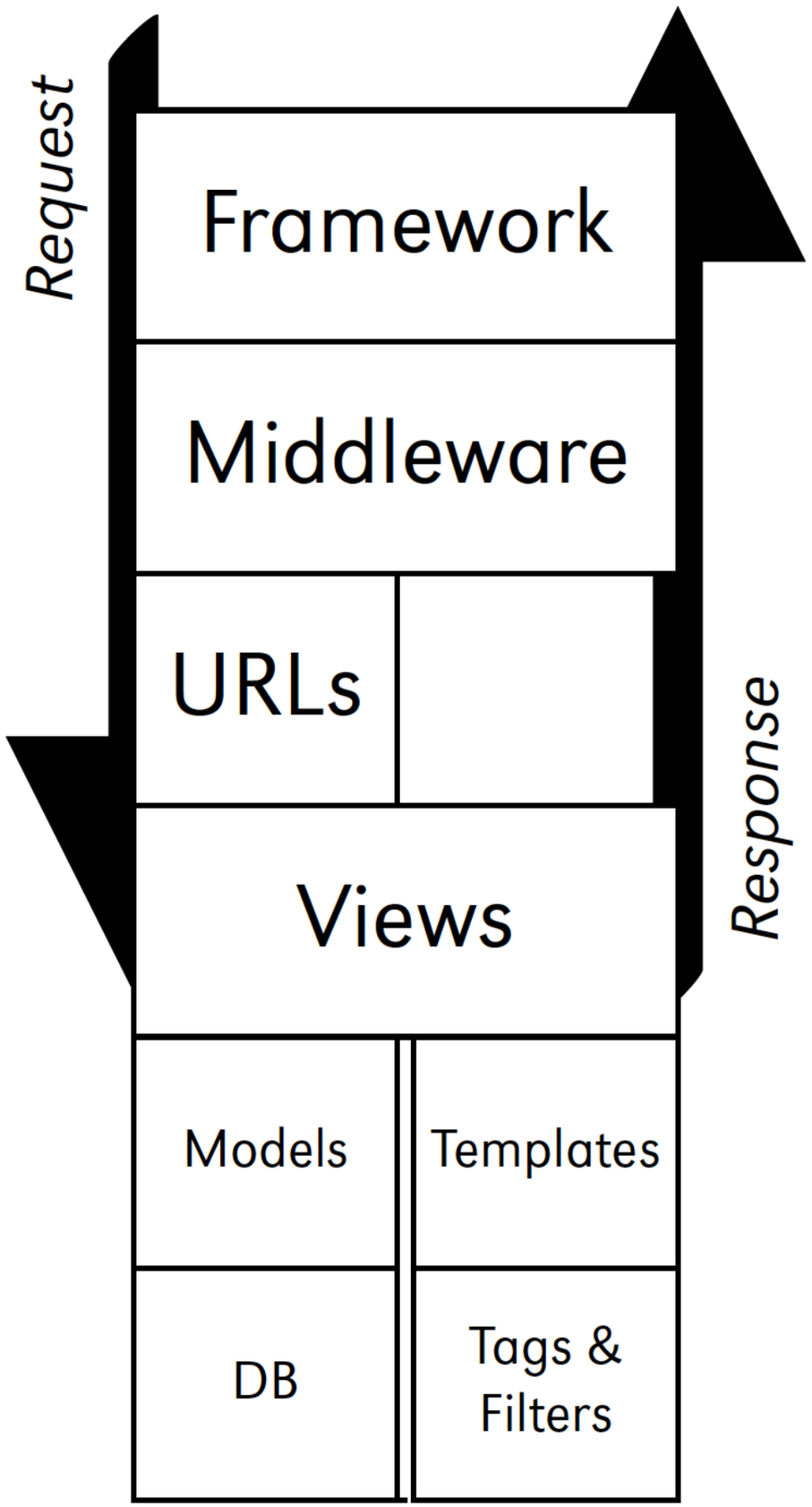
What is Django?

- High-level framework for rapid web development
- Complete stack of tools
- Data modelled with Python classes
- Production-ready data admin interface, generated dynamically
- Elegant system for mapping URLs to Python code
- Generic views' to handle common requests

Django Components

- Think MTV instead of MVC
- Models - Django ORM
- Templates - Django Template Engine
- Views - Python function, Request in Response out
- URL Patterns - Regular expression based


Request
Django
Response


Request
Framework
Middleware
URLs
Views
Models
Templates
DB
Tags &
Filters
Response

Defining Requirements

- requirements.txt

```
# Create requirements.txt for current env
$ pip freeze > requirements.txt
```

```
# Install all modules from requirements.txt file recursive
$ pip install -r requirements.txt
```

Starting a Project

```
# Mac/Linux/etc...

$ pip install django
$ django-admin startproject demo
$ cd demo
$ python manage.py migrate
$ python manage.py runserver

# Windows

> pip install django
> python Scripts/django-admin.py startproject demo
> cd demo
> python manage.py migrate
> python manage.py runserver
```

URLs

- Map URLs in requests to code that can be executed
- Regular expressions!
- Subsections of your site can have their own urls.py modules

Views

- Code that handles requests
- Other frameworks often call these “controllers”
- Basically a function that:
  - gets a request passed to it
  - returns text or a response

Questions?