~~Django REST Framework Is a Powerful and Flexible Toolkit for Building Web APIs.~~

iam<Python>
MEET. LEARN. GROW

Written by RAJA MAHENDRA

@rmreddy20

iampython.com

Architecture

API vs REST

Basic REST Concepts

Installation of DRF

Serializers

Apply Serialization

DRF Flow Diagram

Types of Serializers

View

View Types

DRF CRUD operations

Mixins

Generic Views

Viewsets

Types of Viewsets

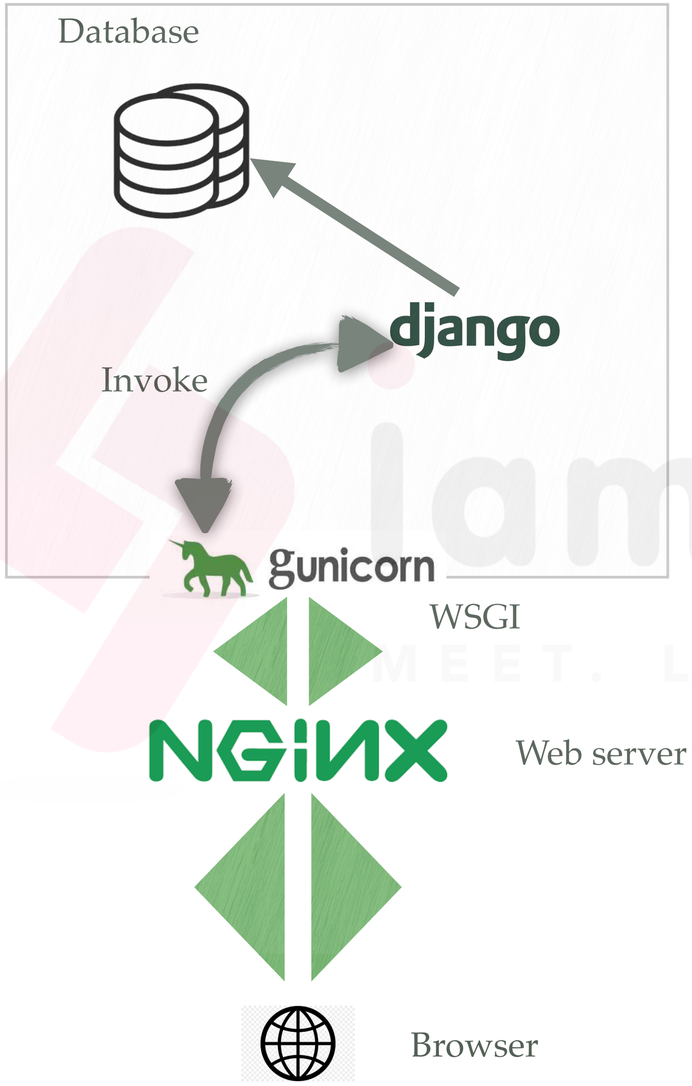
Routers

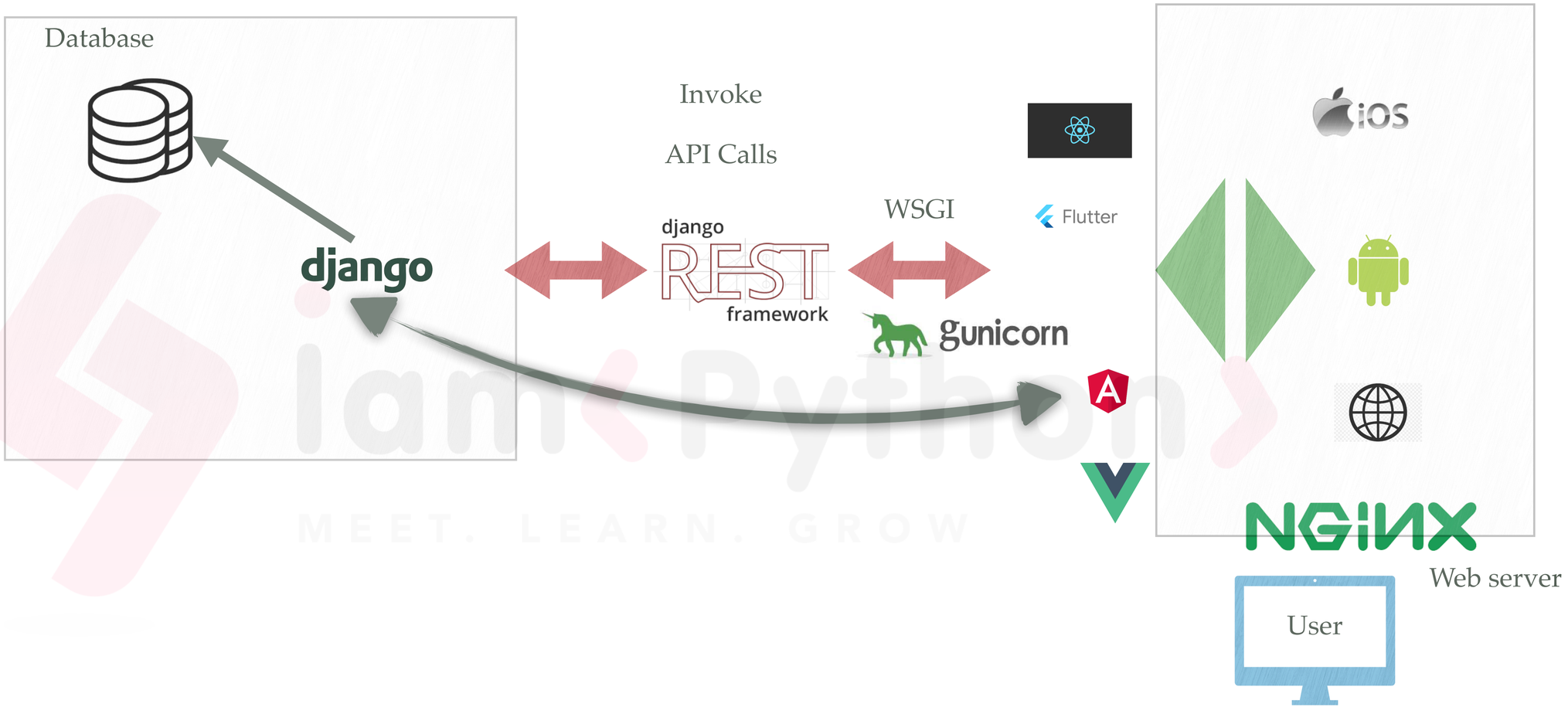
Types of Routers

Pagination

in progress ...

Django Core Project - Web





Django API Project - Web + REST

START

Why we need?

Web browsable API

Restful web-services

Authentication policies

Serialization

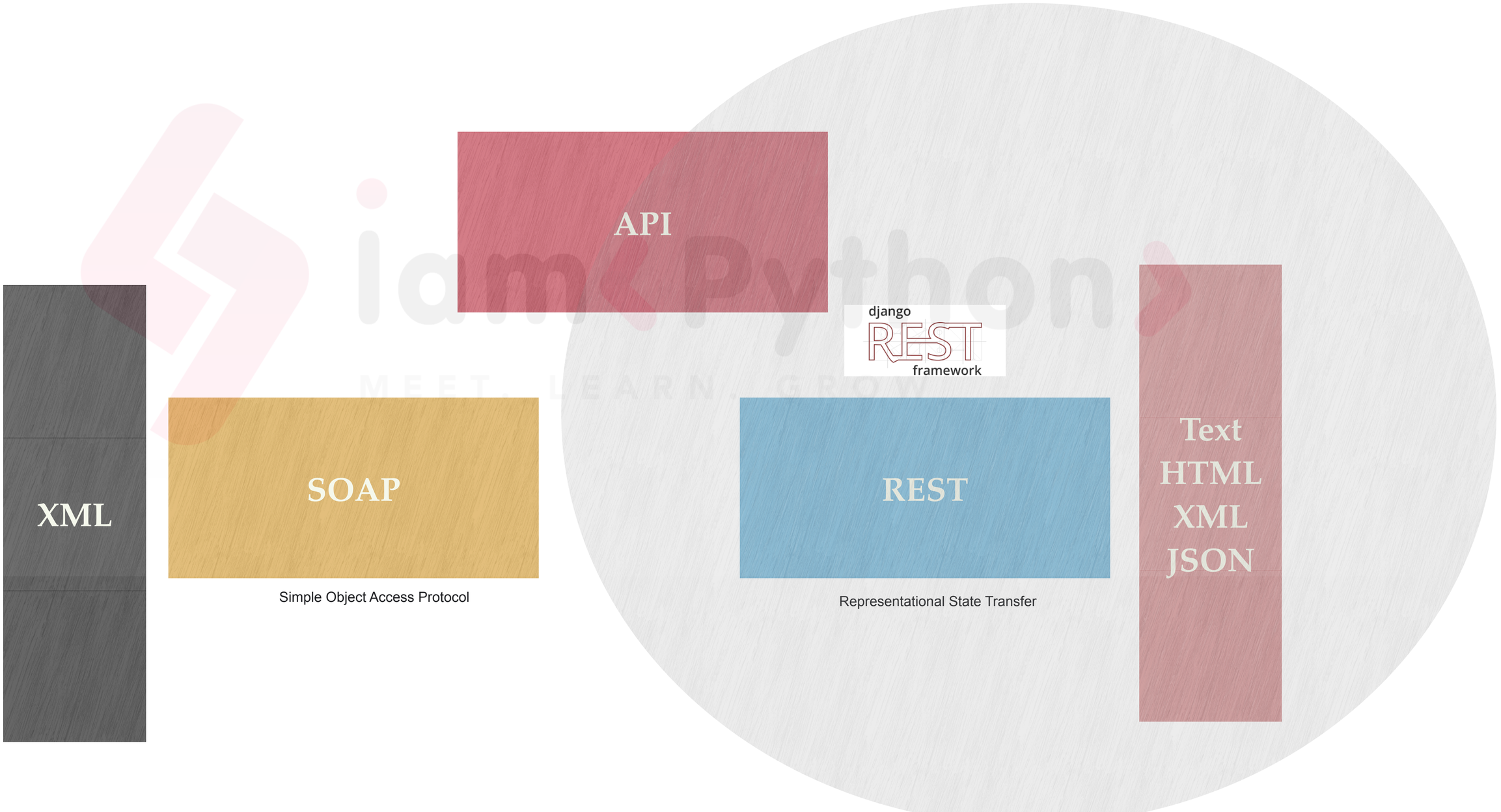
What you need to focus ?

API

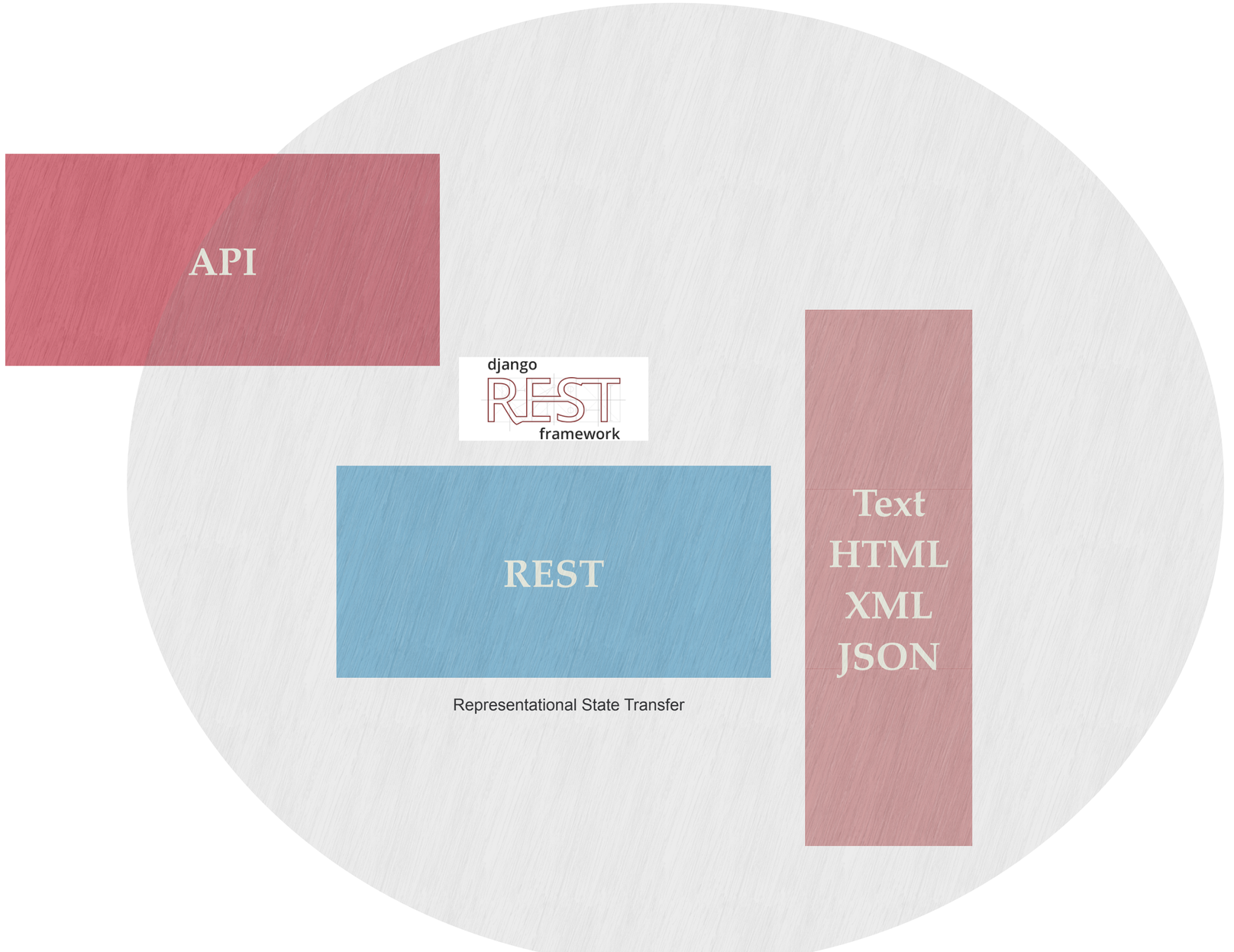
REST framework

JSON

API vs REST


API
XML
SOAP
Simple Object Access Protocol
django
REST
framework
REST
Representational State Transfer
Text
HTML
XML
JSON

API vs REST



Basic Concepts

API -

- Application programming interface

REST framework

- REpresentational State Transfer
- No WSDL required
- Every URL denotes the representation of object
- GET, POST, PUT, DELETE , PATCH
- HTTP
- Limit of length of the data

HTTP METHODS

GET - get resource
POST - create resource
DELETE - delete resource

PUT - update resource

PATCH - partial update resource

JSON

- Javascript object Notation
- Light weight, high performance and human readable message

Pre-Requisites & Install

Python above 3.5

Django 1.1 above (I recommend 3)

Install

---

pip install djangorestframework

```
INSTALLED_APPS = [
    ...
    'rest_framework',
]
```

Let’s have a look into DRF official Website

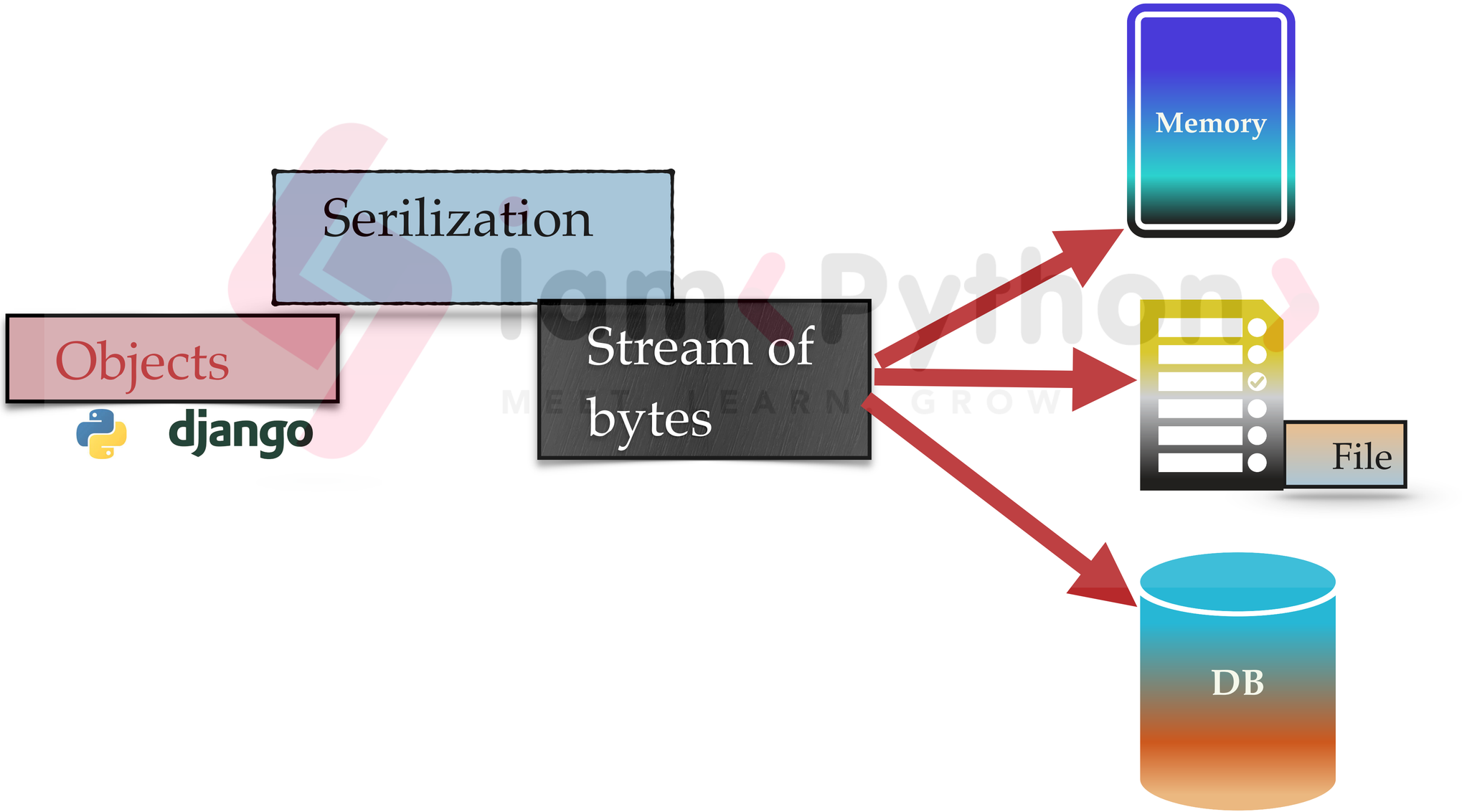
https://www.django-rest-framework.org

Serializers

Allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types.

Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data.

The serializers in REST framework work very similarly to Django's Form and ModelForm classes.



Serializer Class (serialzers.py)

```
from rest_framework import serializers

class ClassnameSerializer(serializers.Serializer):

    #  code goes here
```

Apply serialization (serialzers.py)

1. Class existence
2. Declaring the serializer
3. Serializing objects
4. De-Serializing
5. Saving Instances

6. Validations
7. Accessing the initial data and instance
8. Partial updates
9. Dealing with nested objects
10. Dealing with multiple objects

1. Class existence

2. Declaring the serializer

3. Serializing objects

4. De-Serializing

5. Saving Instances

6. Validations

7.Accessing the initial data and instance

8. Partial updates

9. Dealing with nested objects

10. Dealing with multiple objects

1. Class existence

2. Declaring the serializer

3. Serializing objects

4. De-Serializing

5. Saving Instances

(serialzers.py)
Serializing objects

```
serializer = ClassnameSerializer(Pythonobject)

serializer.data
```

```
from rest_framework.renderers import JSONRenderer

json = JSONRenderer().render(serializer.data)

json
```

1. Class existence

2. Declaring the serializer

3. Serializing objects

4. De-Serializing

5. Saving Instances

(serialzers.py)
De-Serializing objects


```
import io

from rest_framework.parsers import JSONParser



stream = io.BytesIO(json)

data = JSONParser().parse(stream)
```

1. Class existence

2. Declaring the serializer

3. Serializing objects

4. De-Serializing

5. Saving Instances

(serialzers.py) Saving Instances

.save() will create a new instance

.save() will update the existing 'object' instance

6. Validations

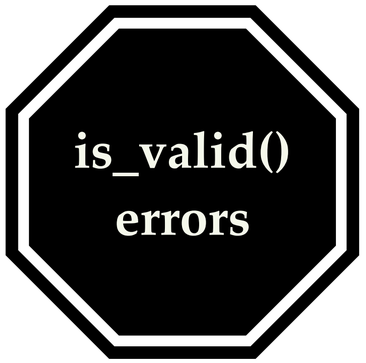
7.Accessing the initial data and instance

8. Partial updates

9. Dealing with nested objects

10. Dealing with multiple objects

(serialzers.py) Validations



serializer.is_valid() is validating the data

serializer.errors is validating displaying errors

Field-level validation

Object-level validation

6. Validations

7.Accessing the initial data and instance

8. Partial updates

9. Dealing with nested objects

10. Dealing with multiple objects

(serialzers.py)
Initial Data and Instance

When passing an initial object or queryset to a serializer instance, the object will be made available as `.instance`. If no initial object is passed then the `.instance` attribute will be `None`.

6. Validations

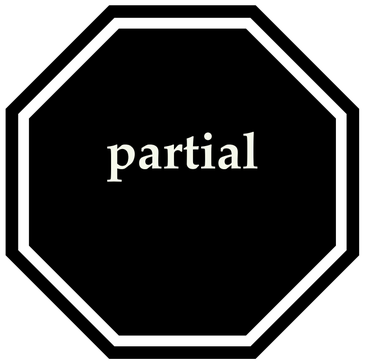
7.Accessing the initial data and instance

8. Partial updates

9. Dealing with nested objects

10. Dealing with multiple objects

(serialzers.py) Partial Updates



By default, serializers must be passed values for all required fields or they will raise validation errors. You can use the `partial` argument in order to allow partial updates.

```
serializer = SomeSerializer(object, data={'content': 'foo bar'}, partial=True)
```

6. Validations

7.Accessing the initial data and instance

8. Partial updates

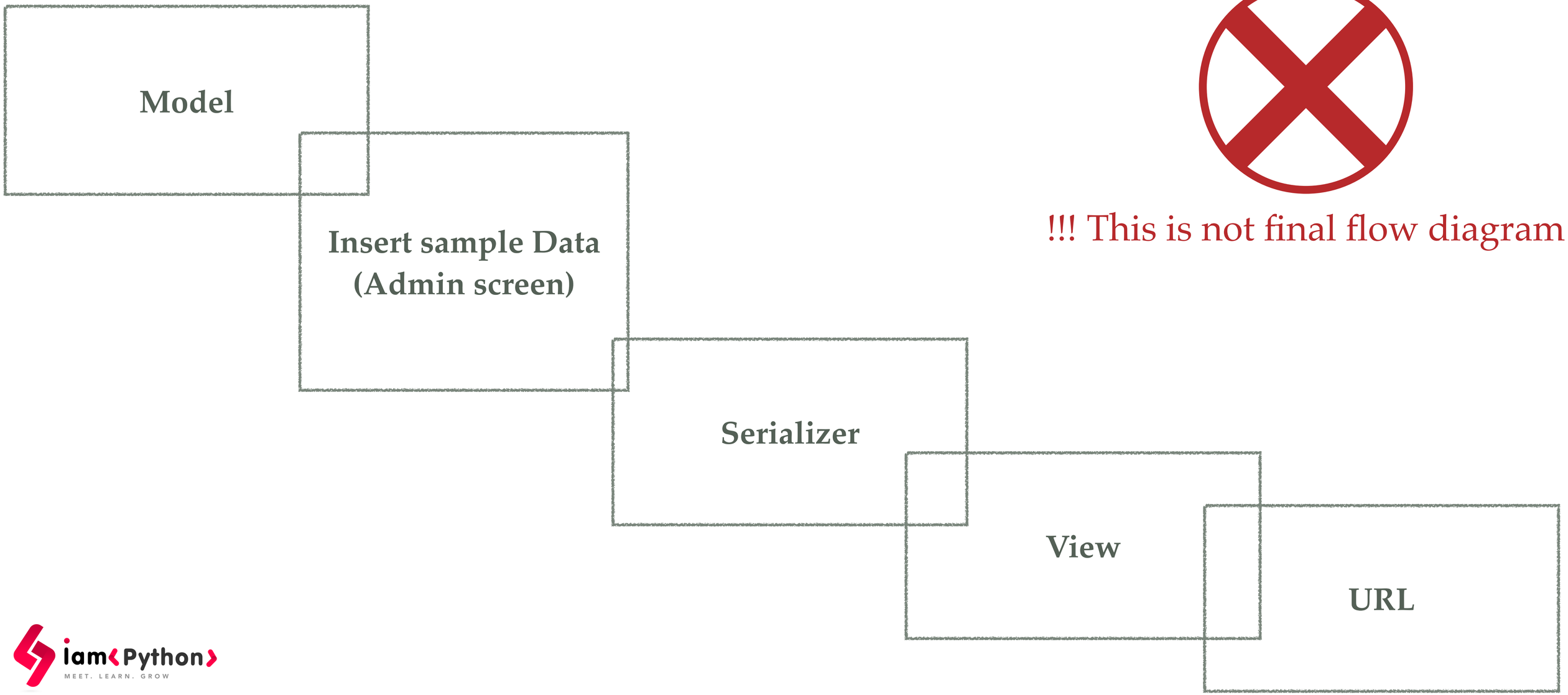
9. Dealing with nested objects

10. Dealing with multiple objects

(serialzers.py)
These 9 and 10 :

Will see in programming …… I will highlight …..

Flow Diagram



!!! This is not final flow diagram

Simple Serializers

Difference

Model Serializers

I already discussed about model serializer in previous video

Types

~~Simple Serializers~~

~~Model Serializers~~

HyperlinkedModelSerializer

List Serializers

Base Serializers

Views

Function Based

Generic Views

View Types

Class Based

ViewSets

Mixins

iam Python
MEET. LEARN. GROW

Function Based

```
@api_view(['GET', 'POST'])
```

View Types

Class Based

```
class ClassName(APIView):
```

Generic Views

```
class ClassName(generics.ListCreateAPIView):

class ClassName(generics.CreateAPIView):

class ClassName(generics.DestroyAPIView):
```

View Types

ViewSets

```
class UserViewSet(viewsets.ViewSet):
```

Django Rest Framework CRUD operations

C - Create

INSERT INTO table_name VALUES(data1, data2, ...)

R - READ

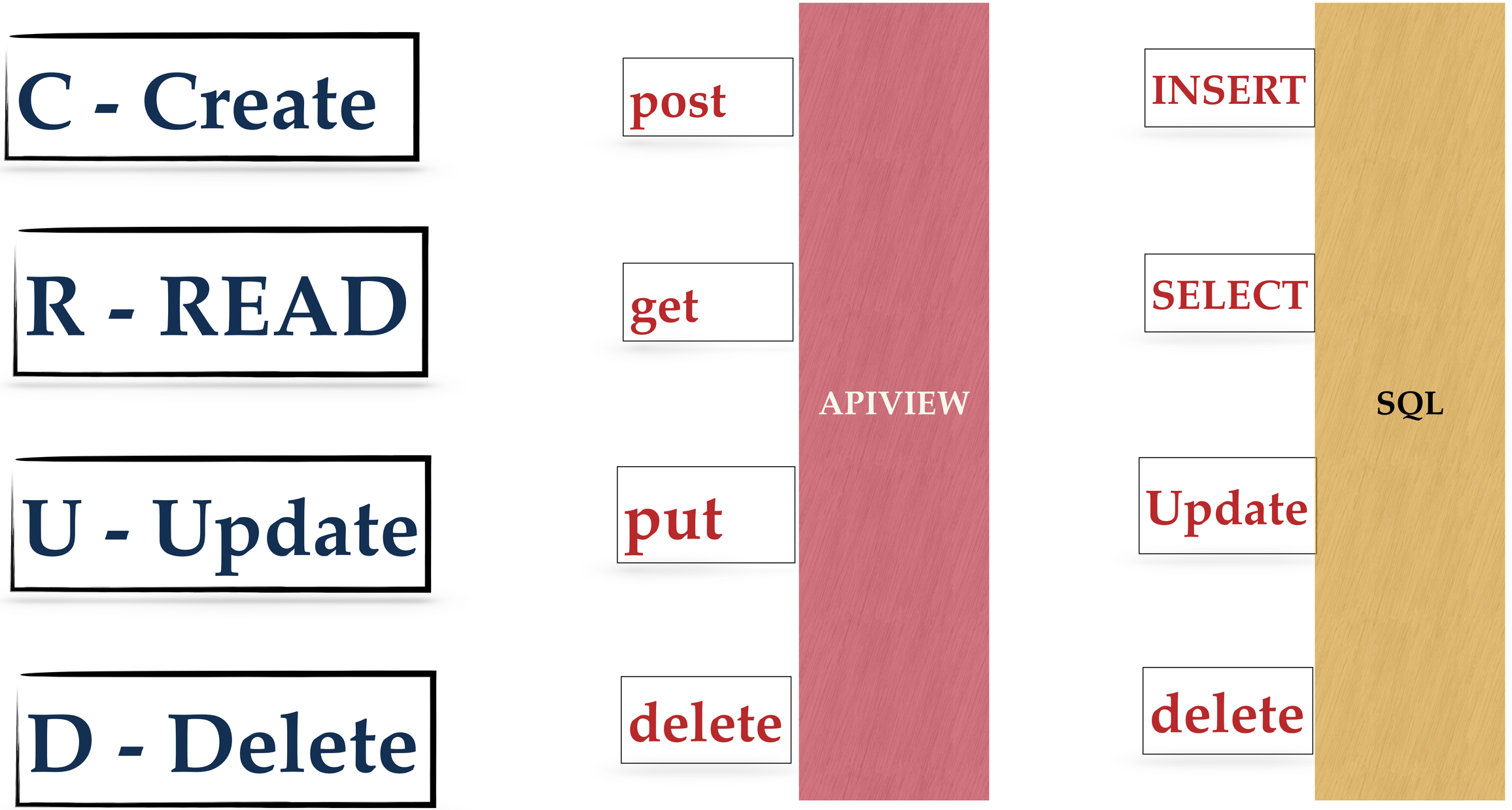
Select Columns from table where condition

U - Update

UPDATE table_name SET column1 = value1, column2 = value2 where condition

D - Delete

DELETE FROM table_name WHERE condition



Django Rest Framework Mixins

Mixins

or

Multiple Inheritance

Interfaces are ideal for defining mixins

Mixins are sometimes described as being "included" rather than "inherited".

iam<Python>
MEET. LEARN. GROW

Logos used for educational purpose only

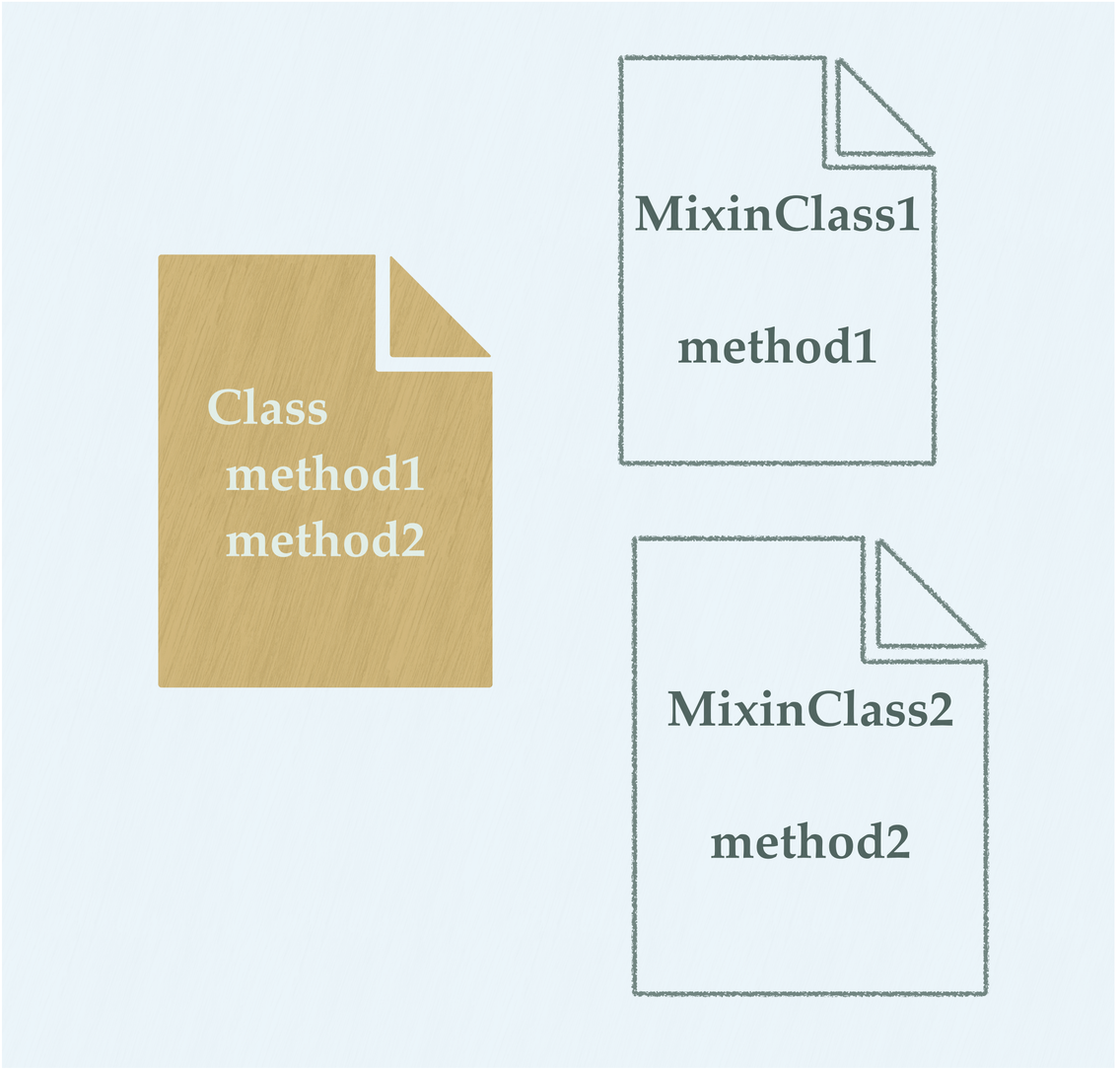
- Ada (by extending an existing tagged record with arbitrary operations in a generic)
- Cobra
- ColdFusion (Class based using includes and Object based by assigning methods from one object to another at runtime)
- Curl (with Curl RTE)
- D (called "template mixins"; D also includes a "mixin" statement that compiles strings as code.)
- Dart
- Factor[11]
- Groovy
- JavaScript Delegation - Functions as Roles (Traits and Mixins)
- Kotlin
- Less
- OCaml
- Perl (through roles in the Moose extension of the Perl 5 object system)
- PHP's "traits"
- Magik
- MATLAB[12]
- Python
- Racket (mixins documentation)
- Raku
- Ruby
- Scala[13]
- XOTcl/TclOO (object systems builtin to Tcl)[14]
- Sass (A stylesheet language)
- Smalltalk
- Vala
- Swift
- SystemVerilog
- TypeScript (mixins documentation)

Source : wikipedia

What is definition of Mixin (Mix in)?

A **class** that contains methods for use by other classes without having to be the parent class of those other classes

Source : wikipedia



Mixin

OOPS

Class

Function Based

@api_view

Classs Based

APIView

GenericAPIView

Django Rest Framework Generic Views

To quickly write common views of the data without having to repeat yourself.

Difference between Mixins and Generics

Mixins class provide the actions rather than providing the handler methods

We don’t override any method when we use only generics but we do in mixins

Mixins comes with combination of GenericAPIView that implements the Multiple inheritance

ViewSets

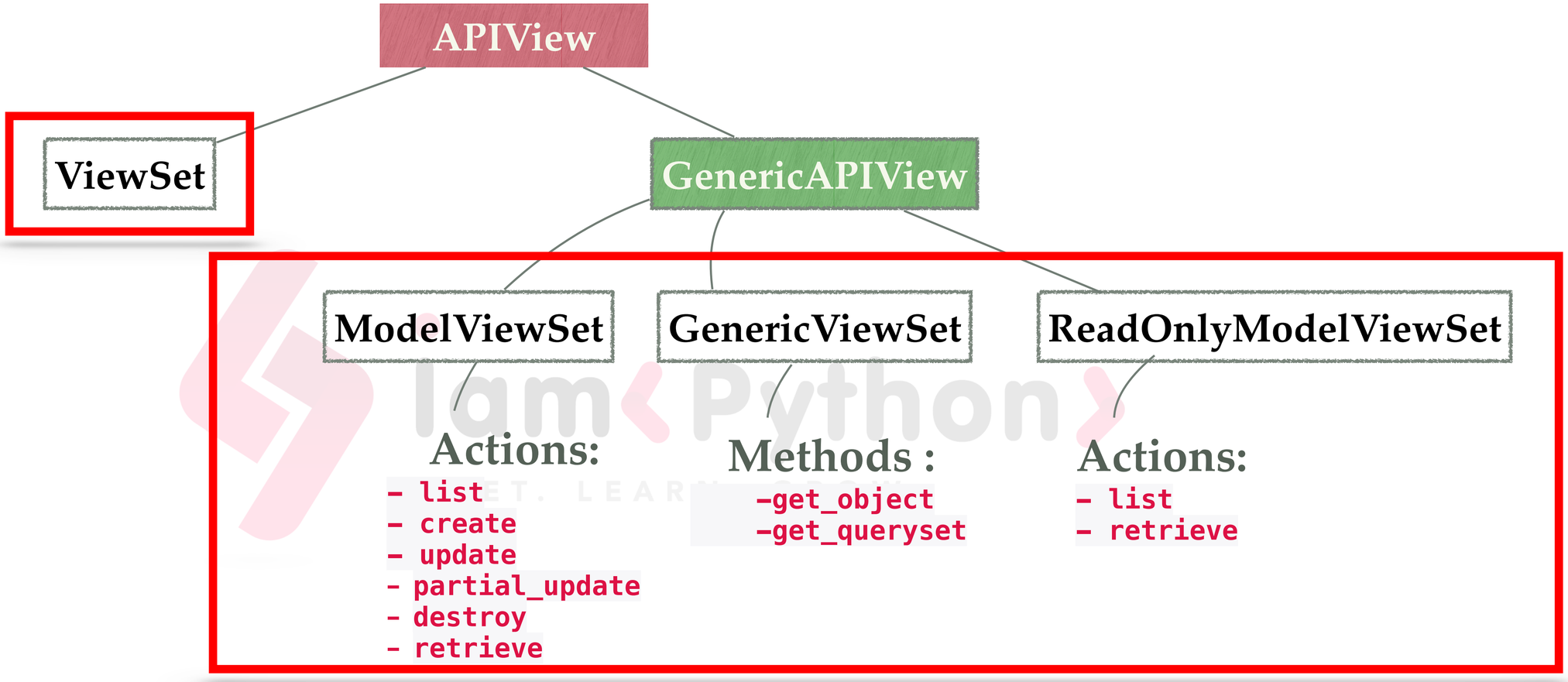
Combine the logic for a set of related views in a single class

~~Not provide any method handlers such as `.get()` or `.post()`~~

Provides actions such as `.list()` and `.create()`

django REST framework

Types of ViewSets

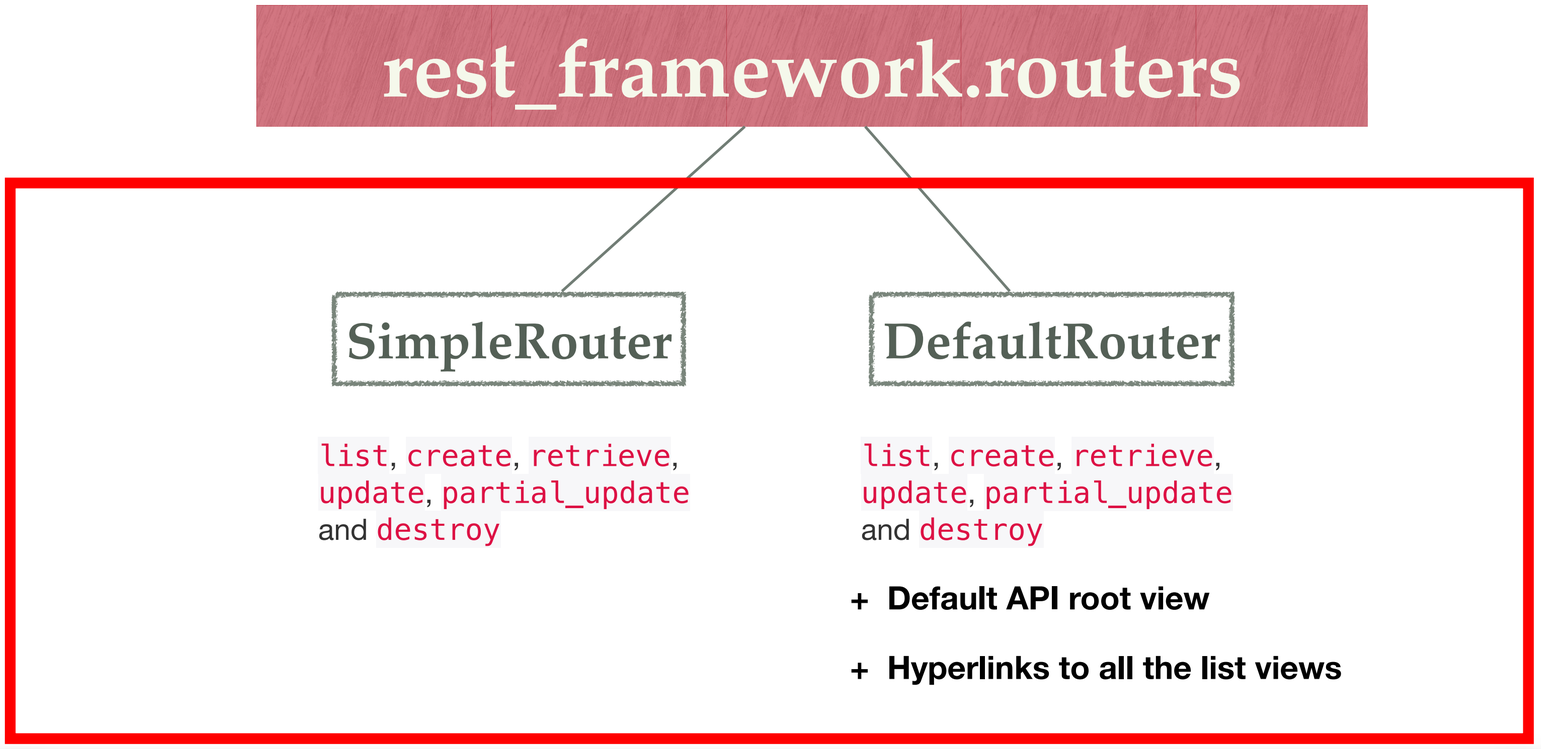


```
@action(detail=True, methods=['post'])
```

Routers

REST framework adds support for automatic URL routing to Django

Types of Routers



Code Now

Pagination

Setting Pagination Style

Global Level Settings

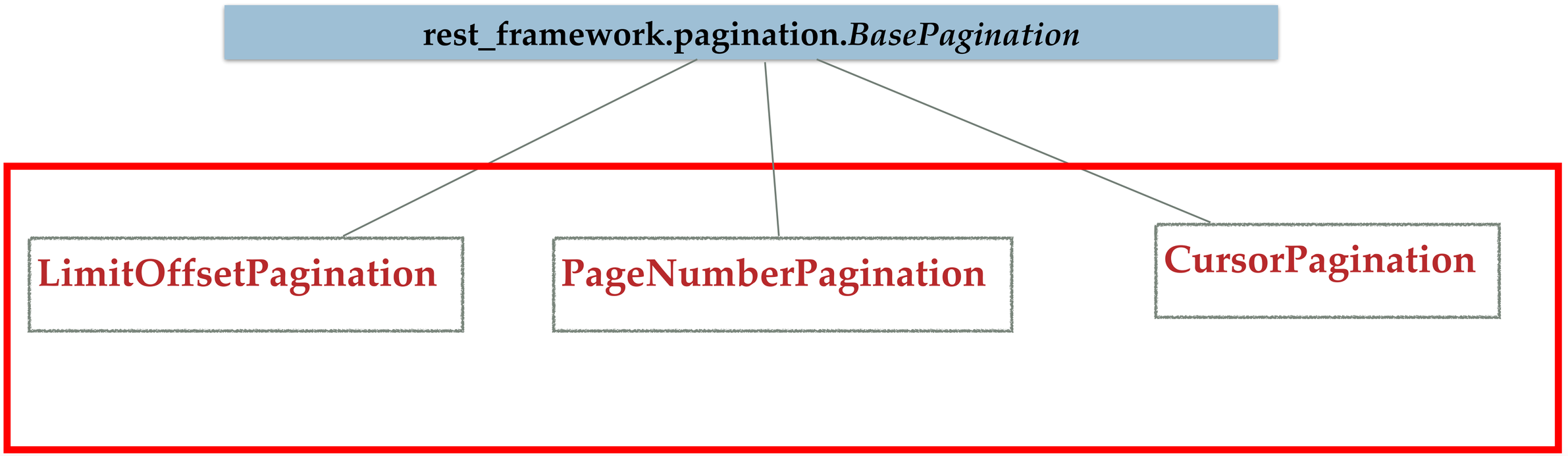
```
REST_FRAMEWORK = {

   'DEFAULT_PAGINATION_CLASS' :  'rest_framework.pagination.paginationtype'

   'PAGE_SIZE' : 2

}
```

iamPython
MEET. LEARN. GROW

View Level Settings

```
pagination_class = ' '
```

Types of Pagination Classes



Code Now