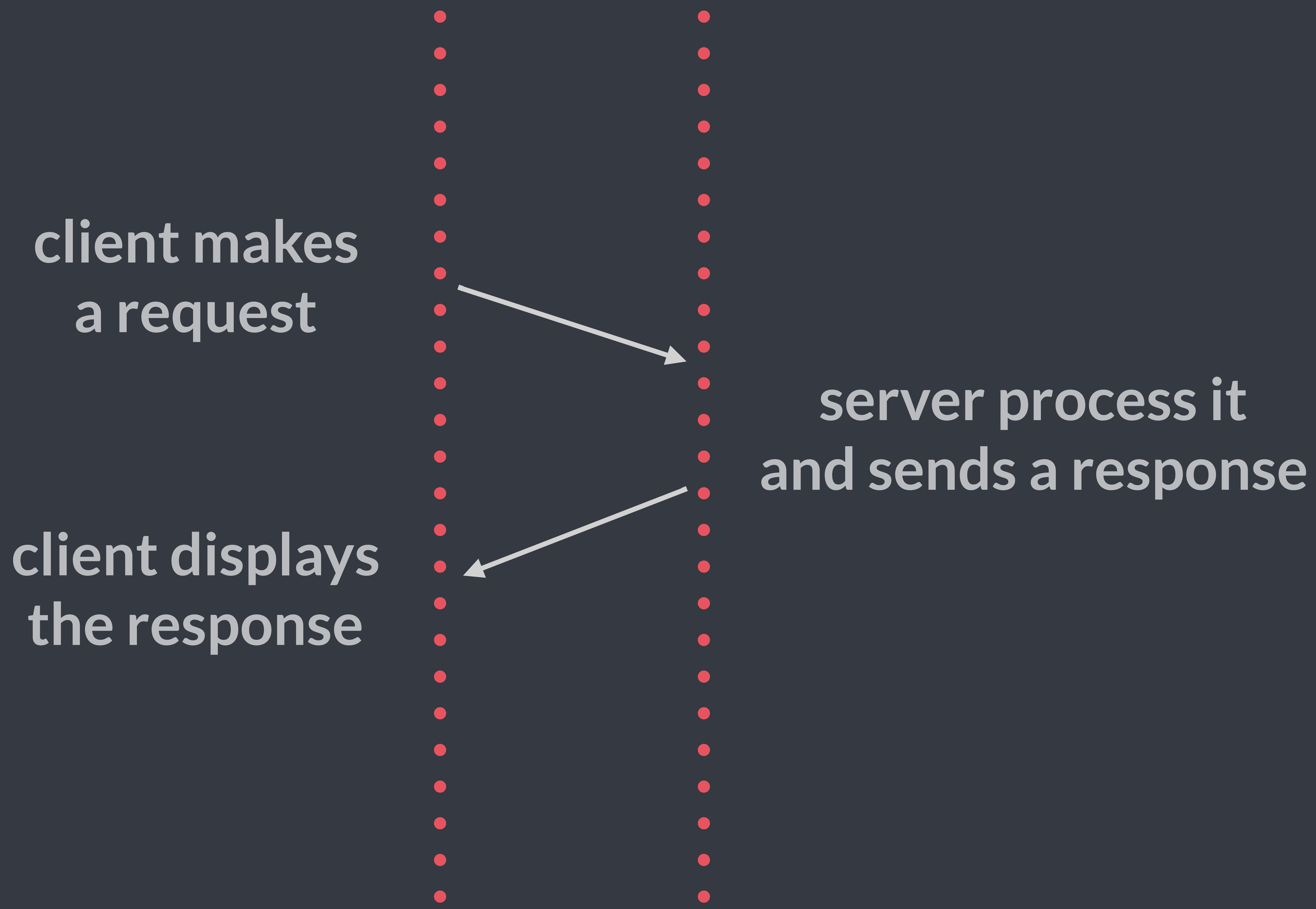
DJANGO CHANNELED

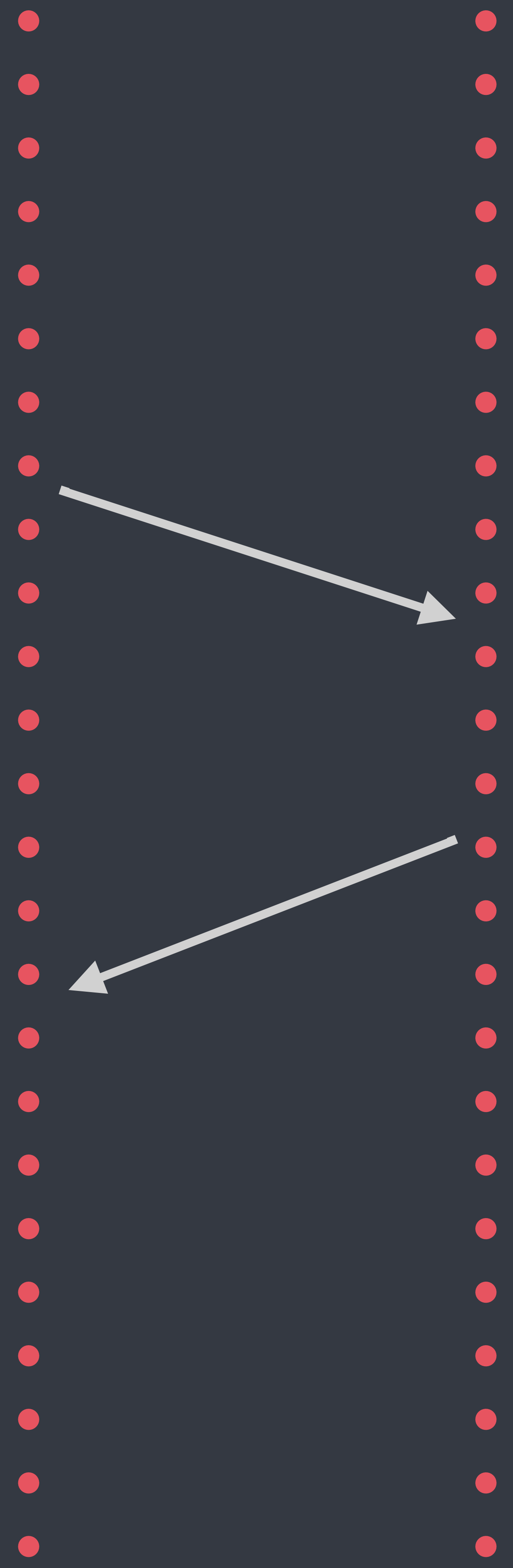
@jonatasbaldin

“jojo”

DJANGO IS AN OLD FRAMEWORK SOLVING OLD PROBLEMS


client makes a request
server process it and sends a response
client displays the response

client makes
a request



Django receives an HTTP Request
routes it to a View
which returns an HTTP Response

client displays
the response

In 2017, web apps need to know
how to display a feed in your timeline,
from thousands of people around the world,
as soon as they publish new content,
in less than one second

Real-Time
Web Applications

SSE
WebRTC
Streaming

WebSockets

WebSocket is a bidirectional and message-oriented transport layer, allowing clients and servers to exchange data using a persistent connection

WebSocket is a bidirectional and message-oriented transport layer, allowing clients and servers to exchange data using a persistent connection

WebSocket is a bidirectional and message-oriented transport layer, allowing clients and servers to exchange data using a persistent connection

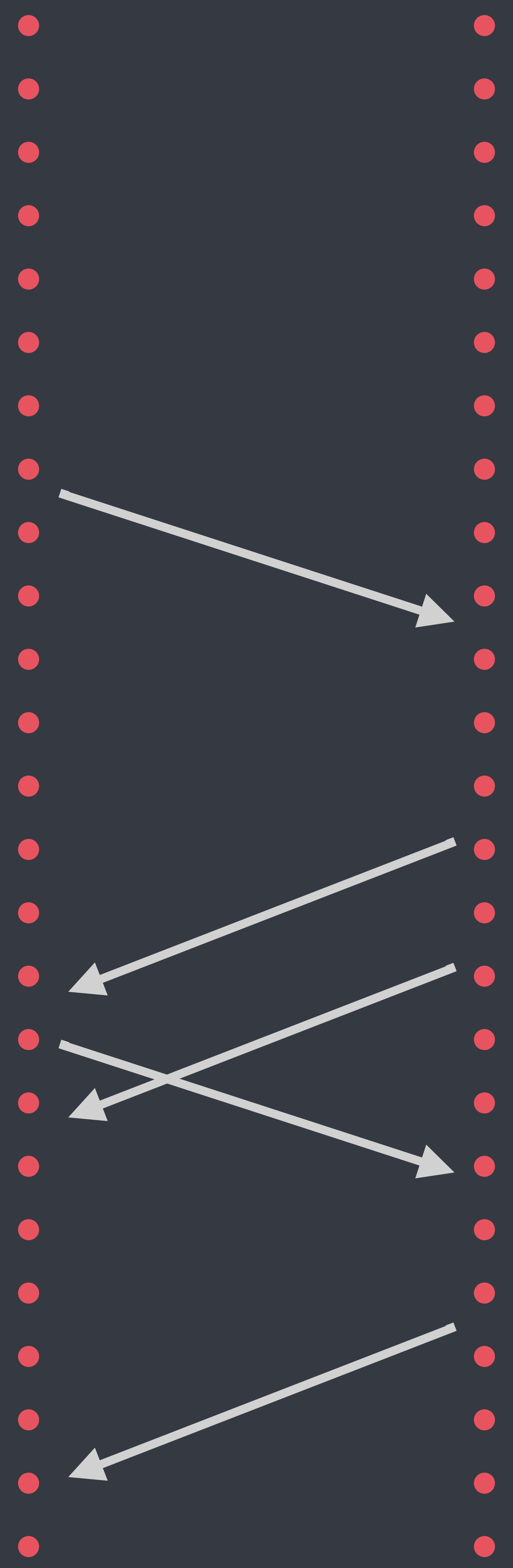
WebSocket is a bidirectional and message-oriented transport layer, allowing clients and servers to exchange data using a **persistent connection**

It has a simple JavaScript API

.onopen()
.onmessage()
.send()
.onerror()
.onclose()

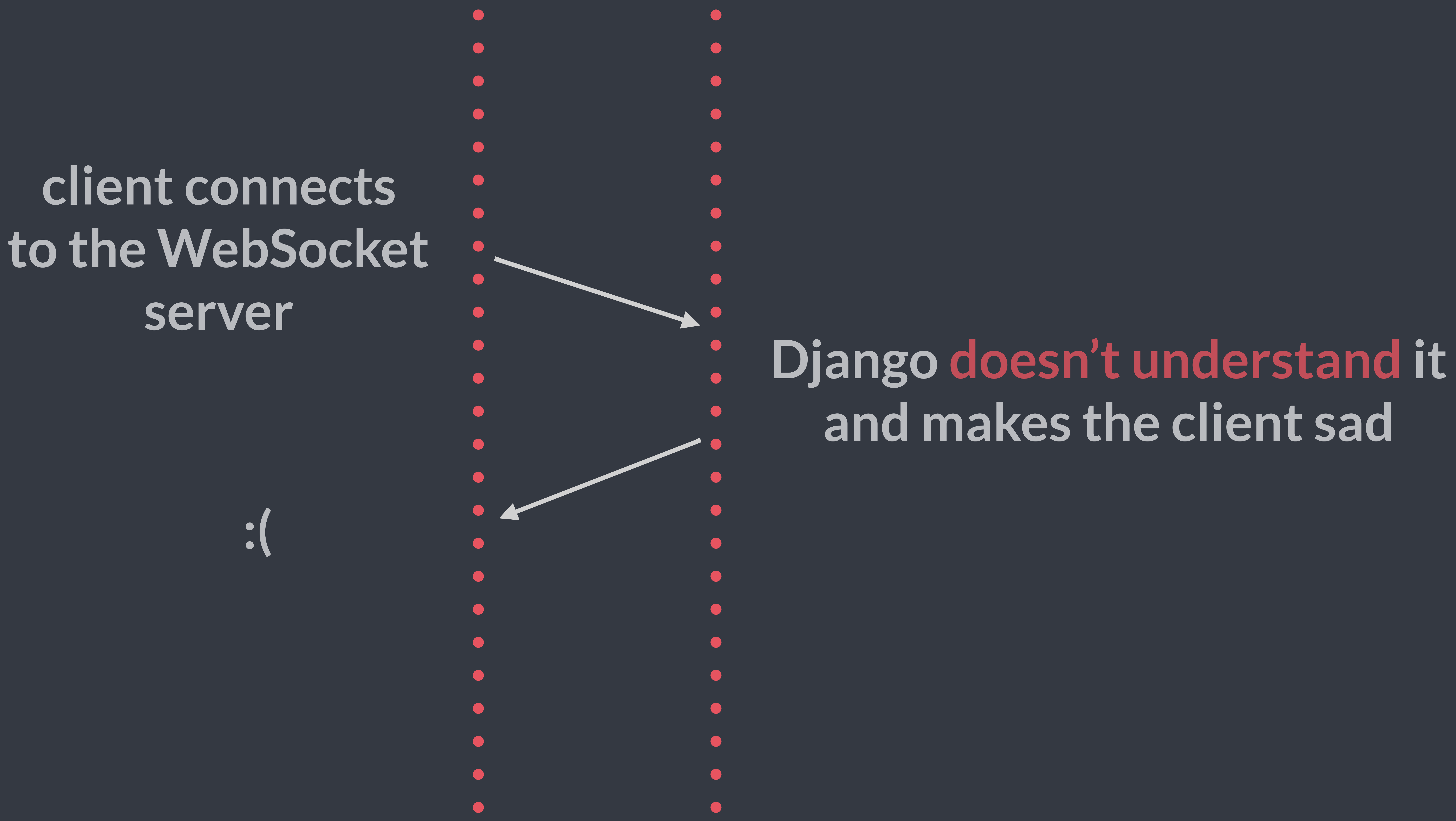
WebSocket is not HTTP!

client connects
to the WebSocket
server



server establishes
the connection

and the data *flows*


client connects
to the WebSocket
server
Django doesn’t understand it
and makes the client sad
:(

DJANGO IS AN OLD FRAMEWORK SOLVING OLD PROBLEMS

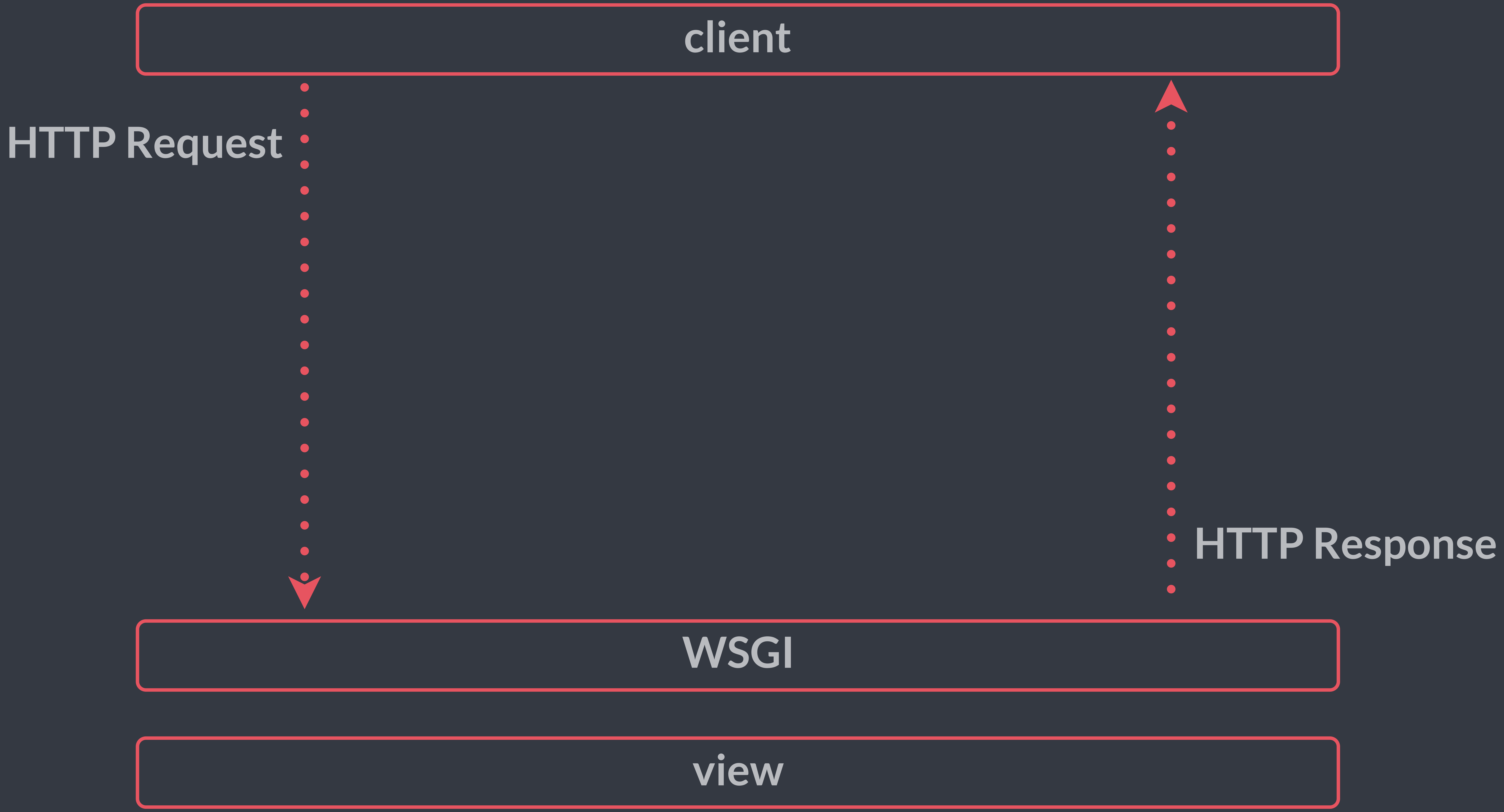
DJANGO IS AN OLD FRAMEWORK
SOLVING OLD PROBLEMS

BUT IT CAN BE EXTENDED
TO SOLVE NEW ONES

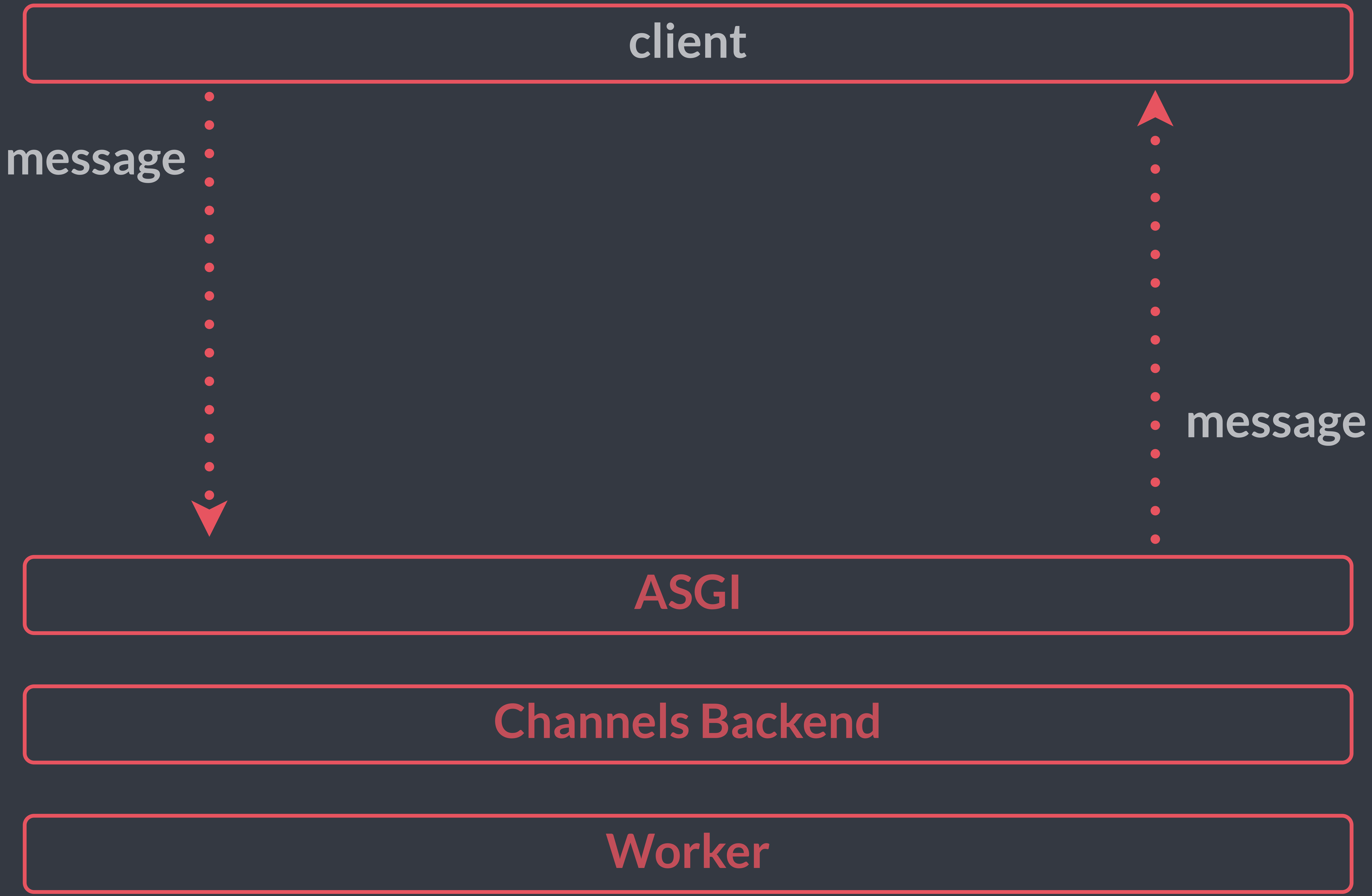
Channels

Enables Django to handle WebSockets and other asynchronous tasks using familiar Django design patterns

Before Channels


client
HTTP Request
HTTP Response
WSGI
view

After Channels


client
message
message
ASGI
Channels Backend
Worker

Let’s break it piece by piece

As WSGI doesn’t understand WebSockets, a new spec was created – ASGI – which basically replaces WSGI

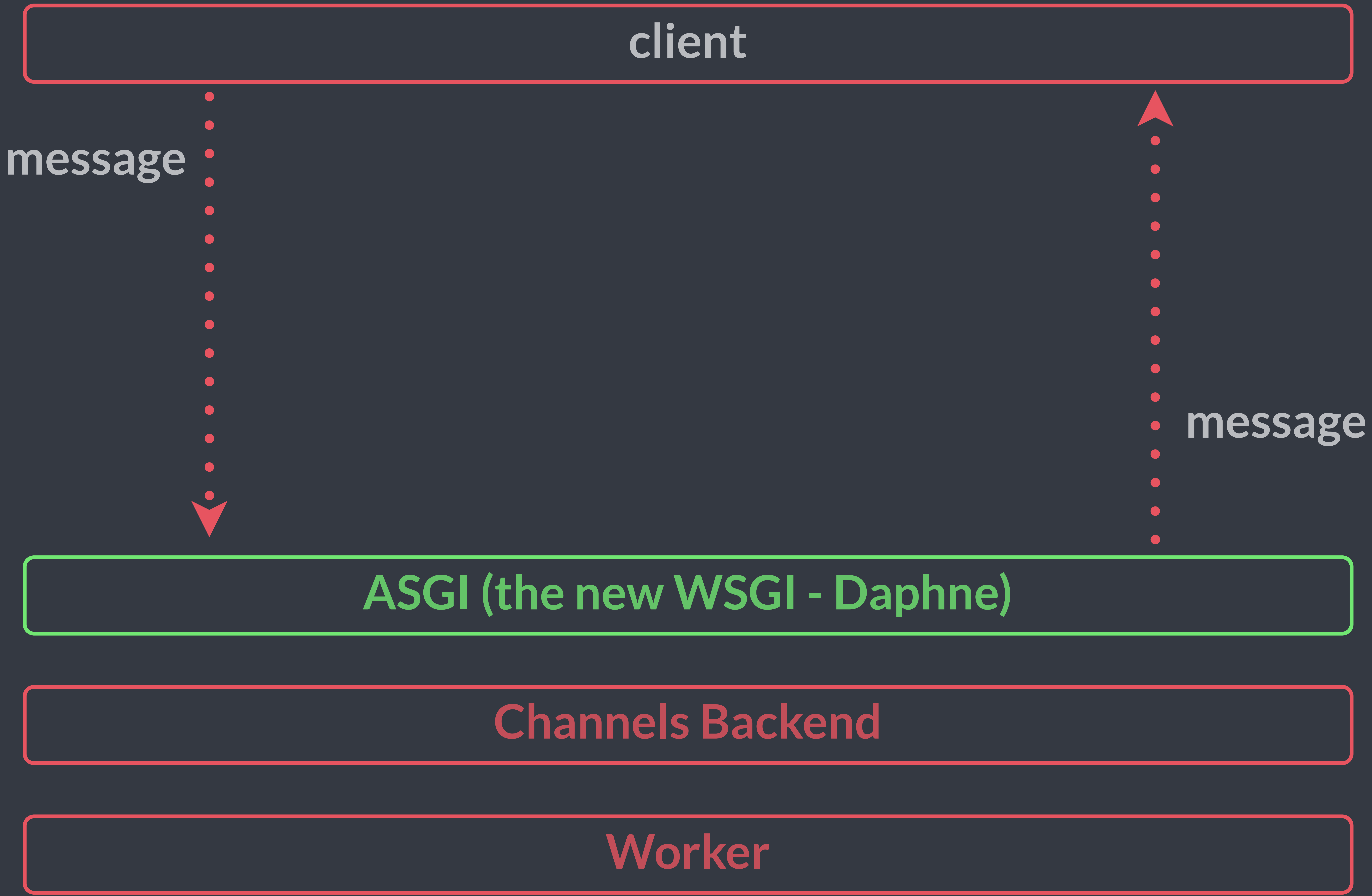
Channels ships with an ASGI implementation server, called Daphne

```python
# asgi.py
import os

from channels.asgi import get_channel_layer


os.environ.setdefault(
    'DJANGO_SETTINGS_MODULE',
    ‘wsquiz.settings'
)
channel_layer = get_channel_layer()
```

Daphne is executed like any other WSGI server, just run the command

```
$ daphne project.asgi:channel_layer
```

Note that Daphne also speaks HTTP, so you can completely remove your WSGI server


client
message
message
ASGI (the new WSGI - Daphne)
Channels Backend
Worker

A Channel is basically a **named task queue** used to store and process messages

It’s a FIFO queue with message expiry and at-most-once delivery

It’s a FIFO queue with message expiry and at-most-once delivery

It’s a FIFO queue with message expiry and at-most-once delivery

It’s a FIFO queue with message expiry and at-most-once delivery

Each message has a unique reply_channel that is used to send a response to the client

```python
# consumers.py
def ws_connect(message):
    message.reply_channel.send({'accept': True})

def ws_message(message):
    message.reply_channel.send(message[‘text’])


# routing.py
channel_routing = [
    route('websocket.connect', ws_connect),
    route('websocket.receive', ws_message),
]
```

We can also assign the reply_channel to a **Channel Group**, allowing the broadcast of messages

```python
# consumers.py
def ws_connect(message):
    Group('tweets').add(message.reply_channel)


# models.py
class Tweet(models.Model):
    text = models.CharField(max_length=140)

    def save(self, *args, **kwargs):
        result = super().save(*args, **kwargs)
        Group('tweets').send({'text': self.text})
        return result


# routing.py
channel_routing = [
    route('websocket.connect', ws_connect),
]
```

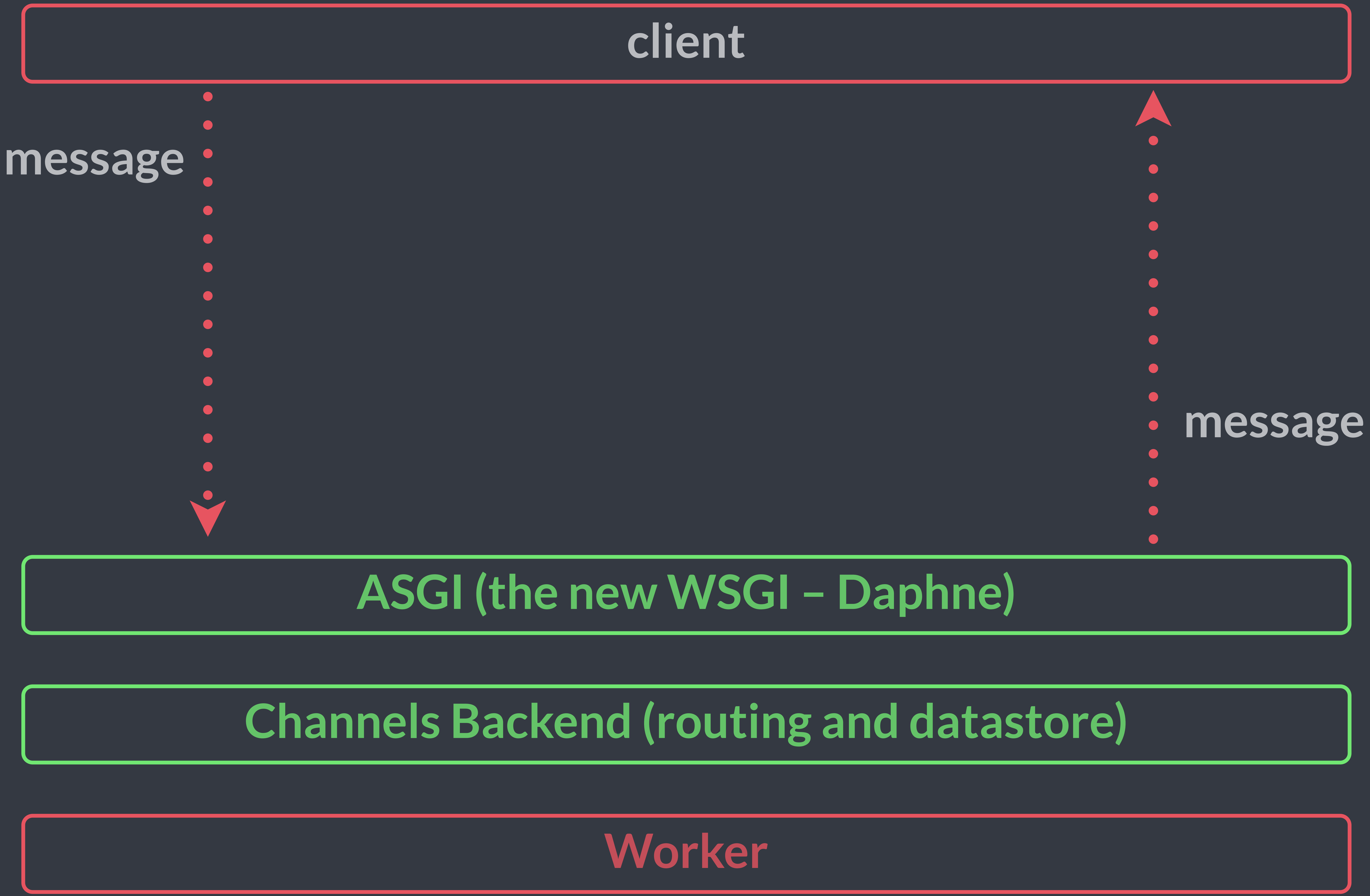
These messages can be stored in different ways

In-Memory
testing and single-process

POSIX IPC
single-machine

REDIS/RabbitMQ
network layer

```python
# settings.py
CHANNEL_LAYERS = {
    'default': {
        'BACKEND': 'asgi_redis.RedisChannelLayer',
        'CONFIG': {
            'hosts': [(REDIS_HOST, 6379)],
        },
        'ROUTING': 'wsquiz.routing.channel_routing',
    }
}
```

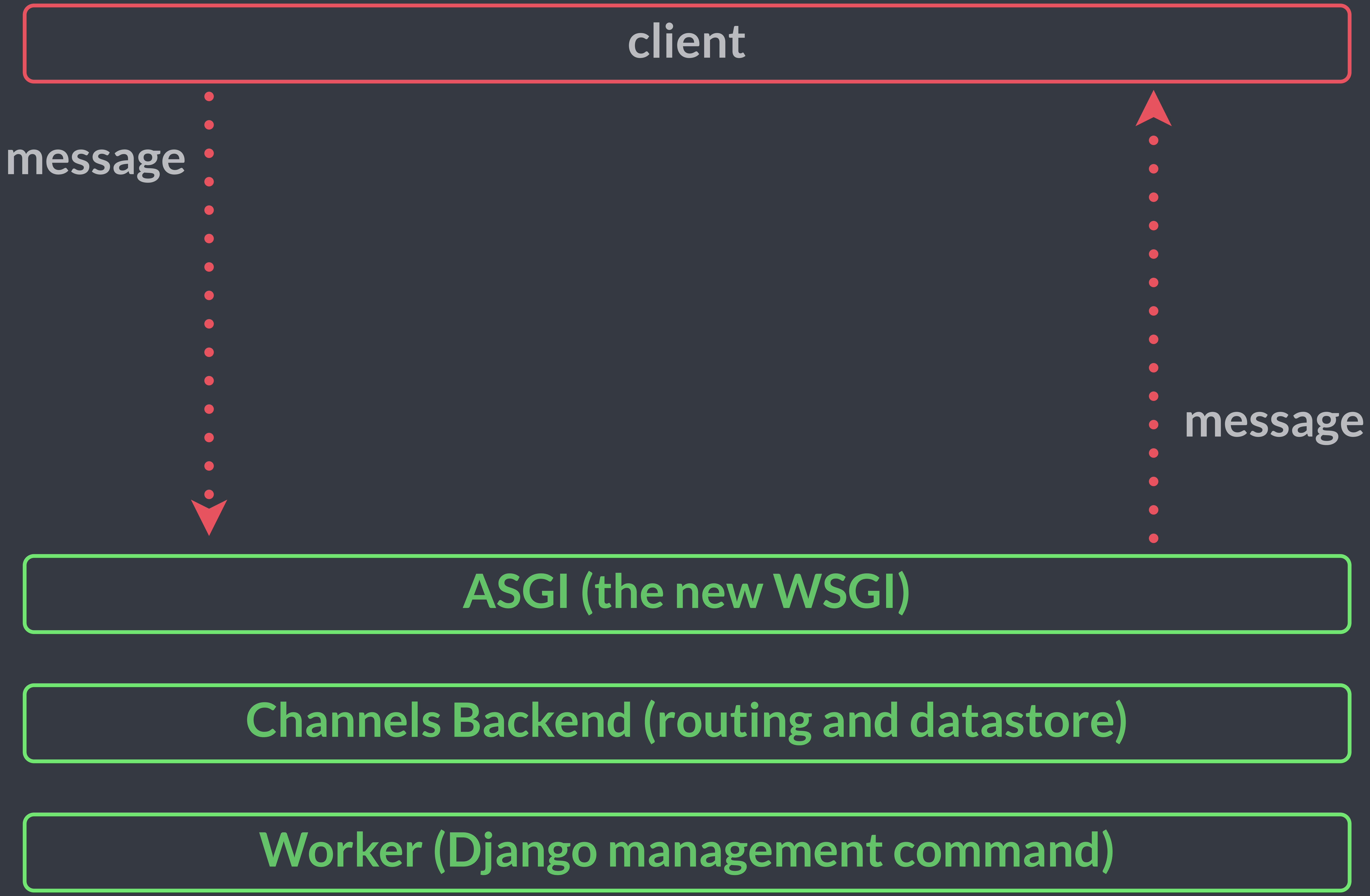

client
message
message
ASGI (the new WSGI – Daphne)
Channels Backend (routing and datastore)
Worker

The worker is responsible to listen to Channels and consume messages once they are ready

```python
# consumers.py
def ws_message(message):
    message.reply_channel.send(message['text'])


# routing.py
channel_routing = [
    route('websocket.receive', ws_message),
]
```

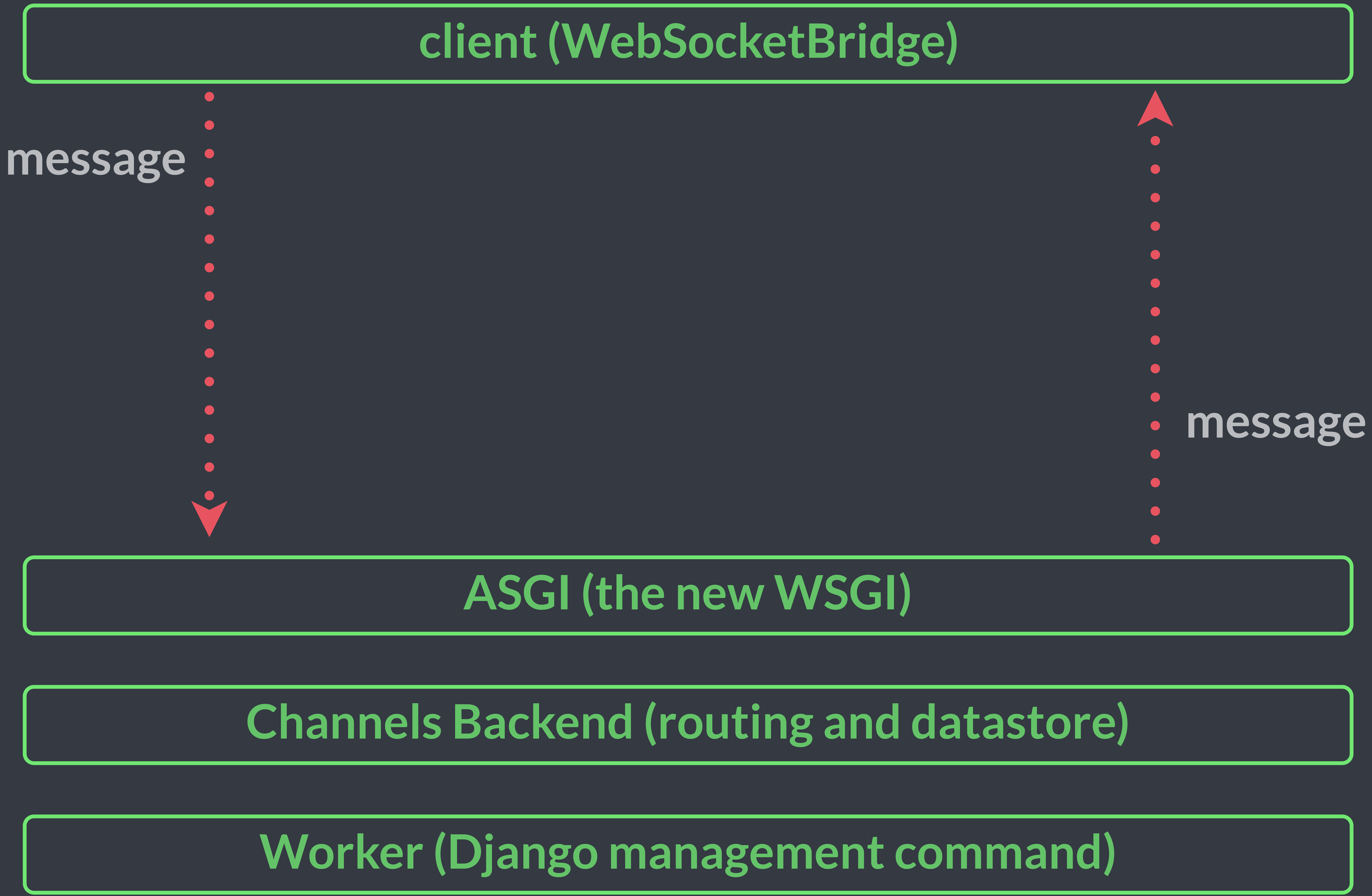
Channels comes with a Django management command for running workers!

```
$ python manage.py runworker
```

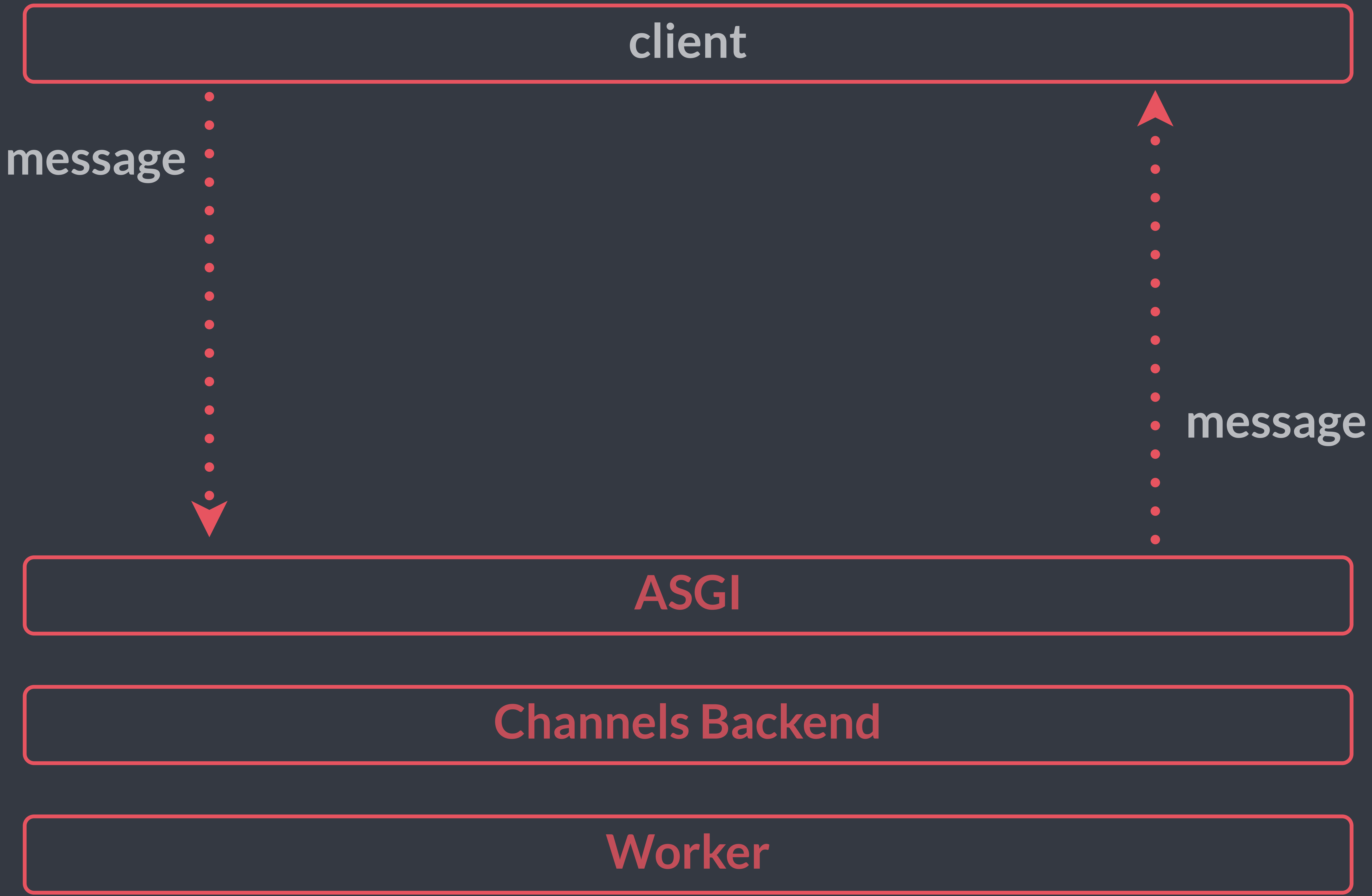

client
message
message
ASGI (the new WSGI)
Channels Backend (routing and datastore)
Worker (Django management command)

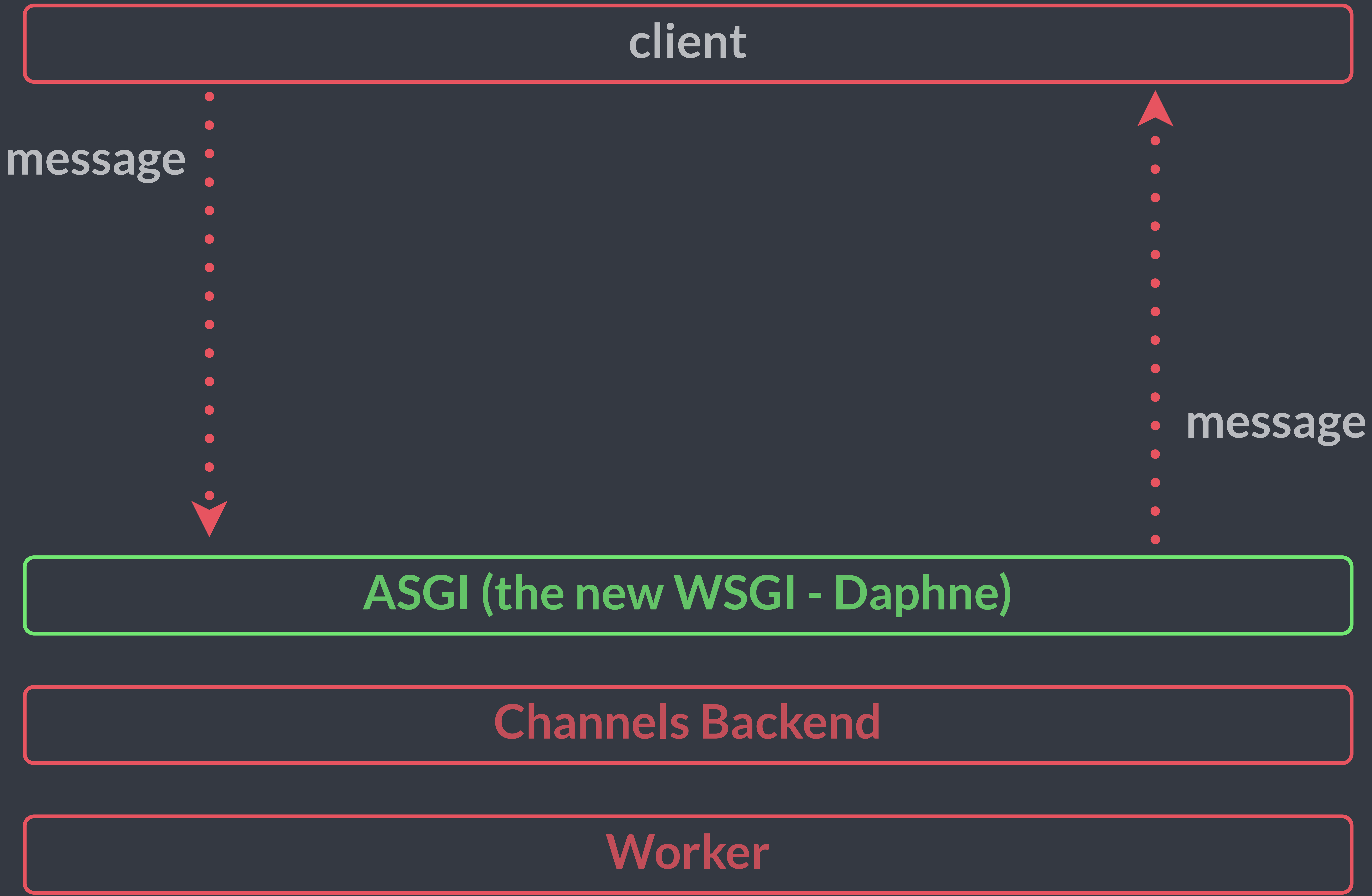
To develop the client
– like a JavaScript application –
Channels comes with a library called
WebSocketBridge

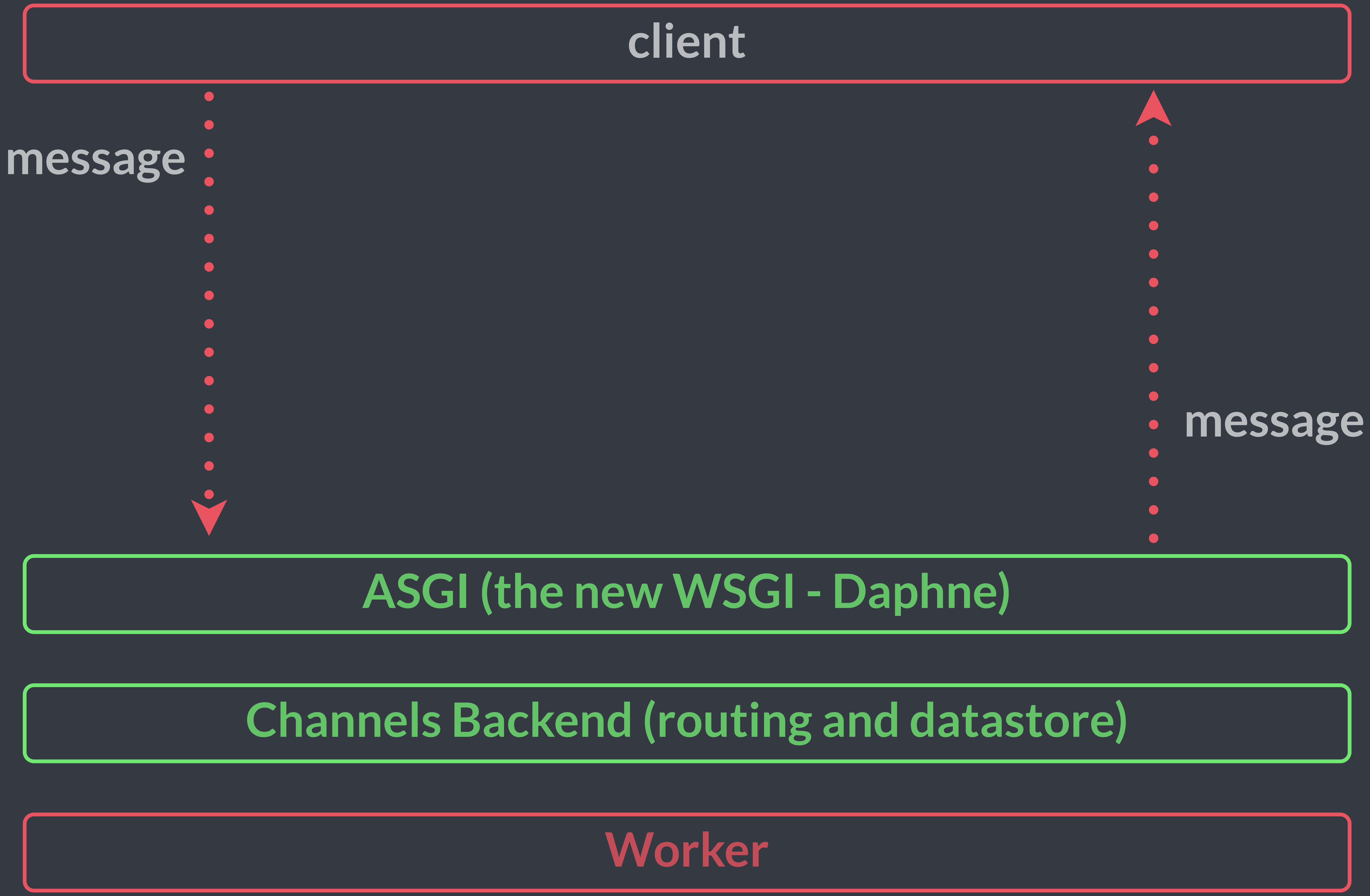
```javascript
const webSocketBridge = new channels.WebSocketBridge();

webSocketBridge.connect('/ws/');

webSocketBridge.listen(function(data) {
  console.log(data);
});
```

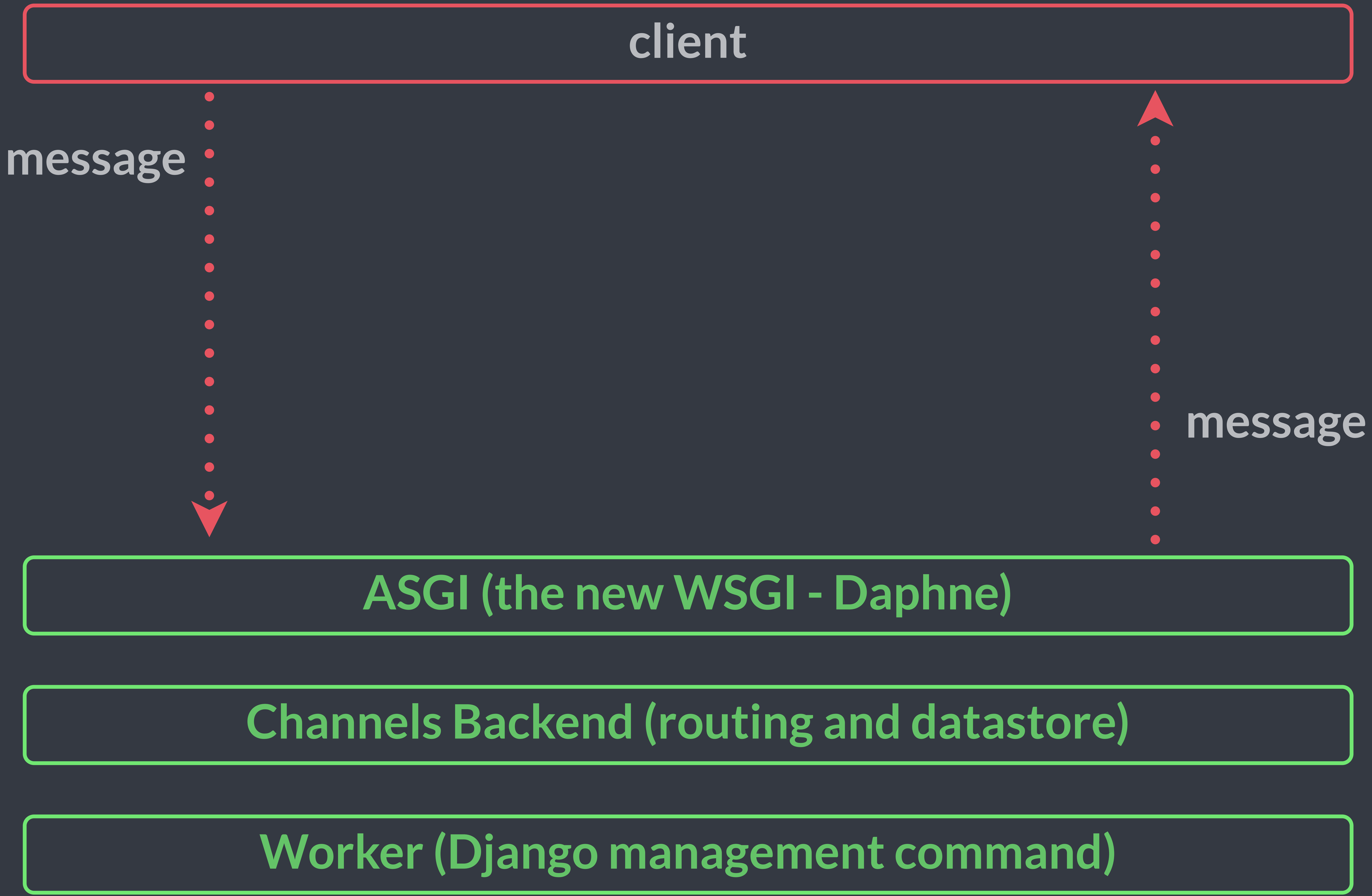

client (WebSocketBridge)
message
message
ASGI (the new WSGI)
Channels Backend (routing and datastore)
Worker (Django management command)

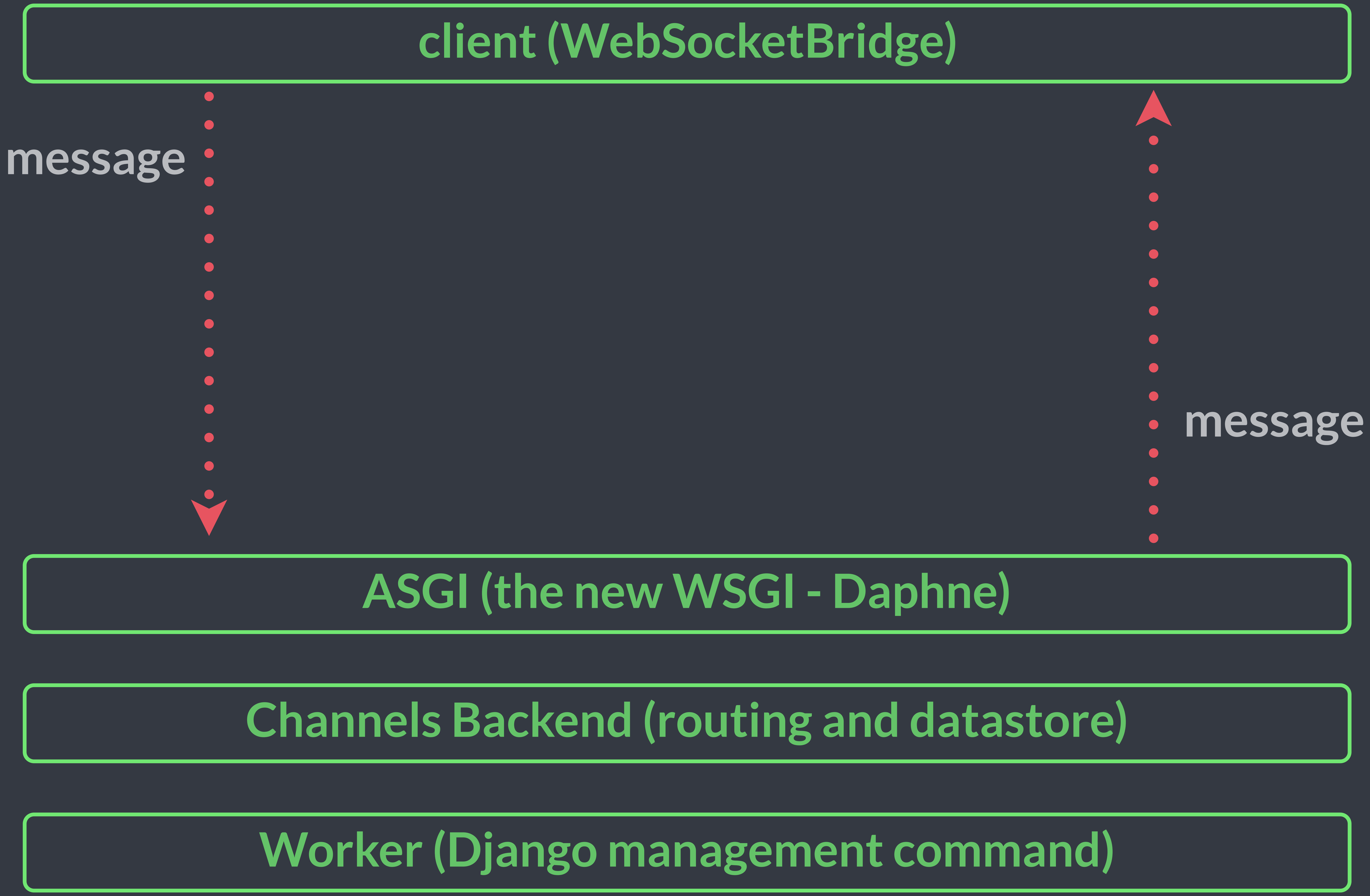
Summing up…


client
message
message
ASGI
Channels Backend
Worker

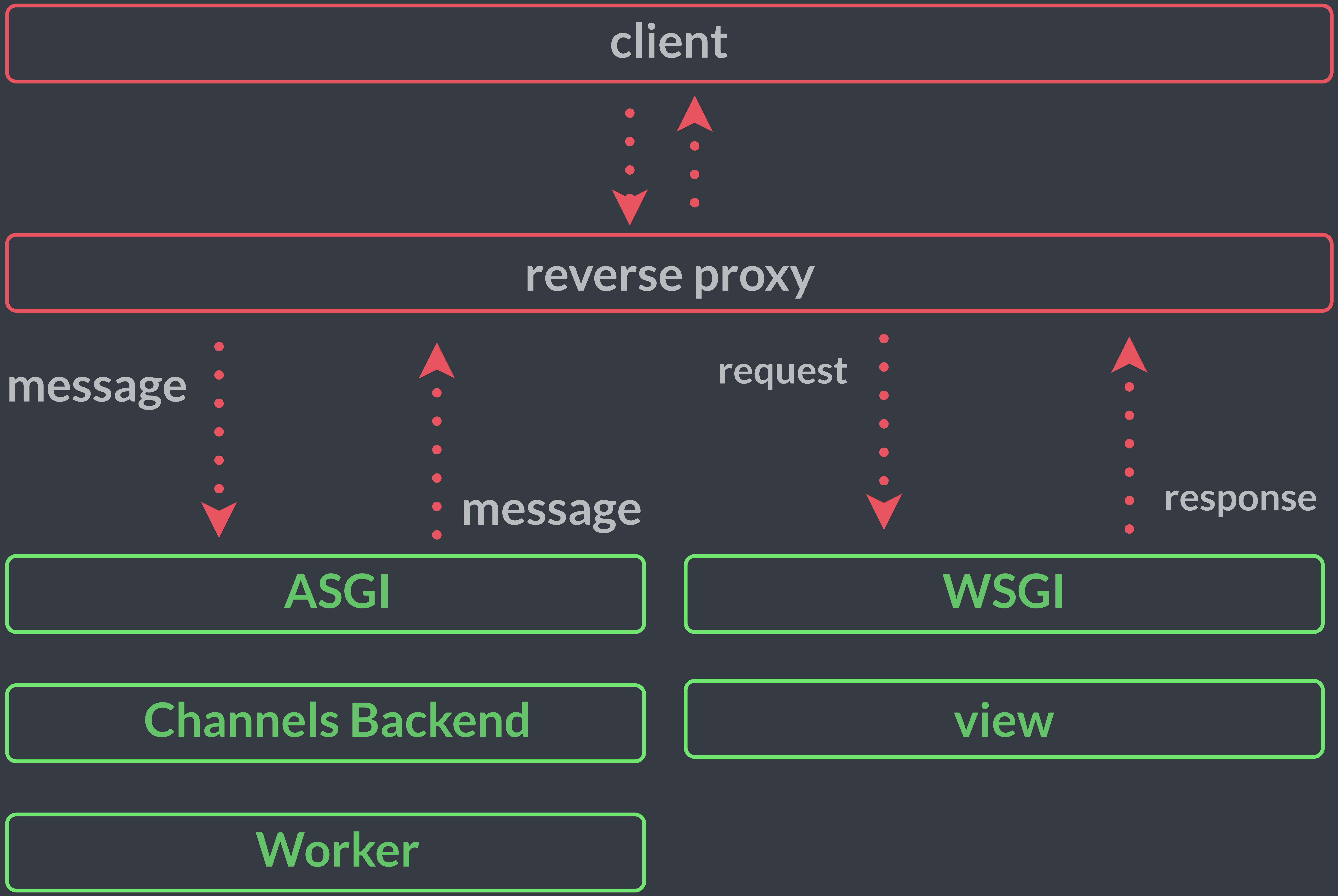

client
message
message
ASGI (the new WSGI - Daphne)
Channels Backend
Worker


client
message
message
ASGI (the new WSGI - Daphne)
Channels Backend (routing and datastore)
Worker


client
message
message
ASGI (the new WSGI - Daphne)
Channels Backend (routing and datastore)
Worker (Django management command)


client (WebSocketBridge)
message
message
ASGI (the new WSGI - Daphne)
Channels Backend (routing and datastore)
Worker (Django management command)

“But I still need to run *normal* Django”


client
reverse proxy
message
message
request
response
ASGI
WSGI
Channels Backend
view
Worker

Testing is simple!

```
$ python manage.py runserver
```

Testing is simple!

```
$ python manage.py runserver
```

not recommended for production

Channels changes the way Django runs to be event-oriented

The WSQuiz

https://wsquiz.herokuapp.com/

Things to take into consideration

WebSocket has TLS, please use it!

WebSocket and Subprotocols

Tests, documentation and monitoring

Today, all of the major browsers support WebSockets, but write fallbacks for critical core business

Channels is a young project,
but the first Django official app

Let’s study!

channels.readthedocs.io

github.com/andrewgodwin/channels-examples

github.com/jonatasbaldin/wsquiz

DJANGO

C H A N N E L E D

@jonatasbaldin

DJANGO
C H A N N E L E D

the D is silent

@jonatasbaldin