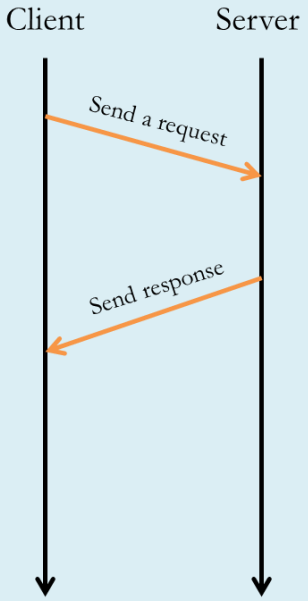
# Asynchronous APIs and architectures

- **Synchronous vs. Asynchronous architectures**
- **Synchronous vs. Asynchronous APIs**
  - *From the client perspective…*
- **Implementing  async APIs**

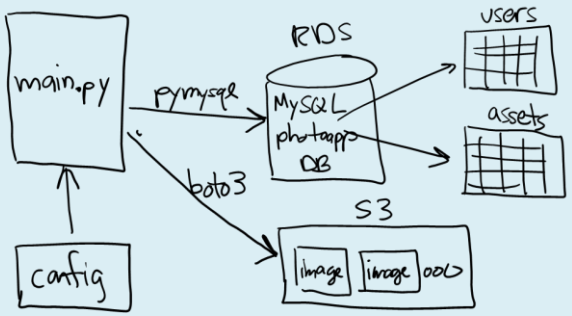
# Synchronous architectures

- So far, everything we have discussed has been a **synchronous** architecture
- Client (*web browser, mobile app, desktop app*) makes a request, and the Server responds
- The client starts the interaction, and waits for some sort of response – client and server are synchronized with one another

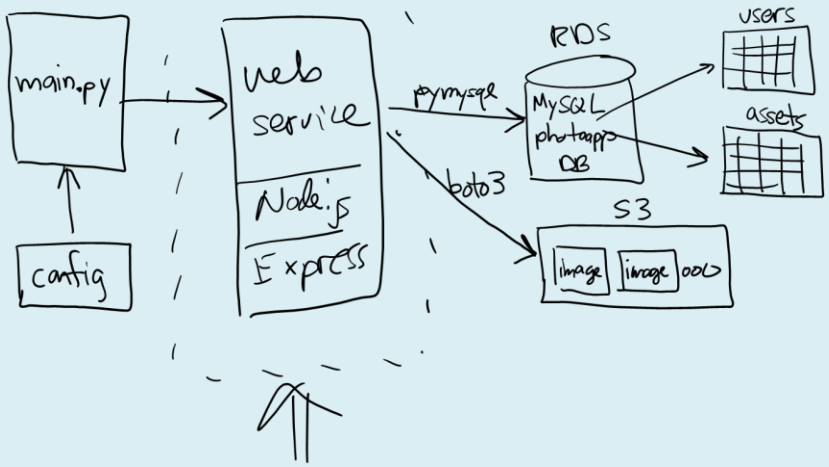


# Projects --- Sync architectures

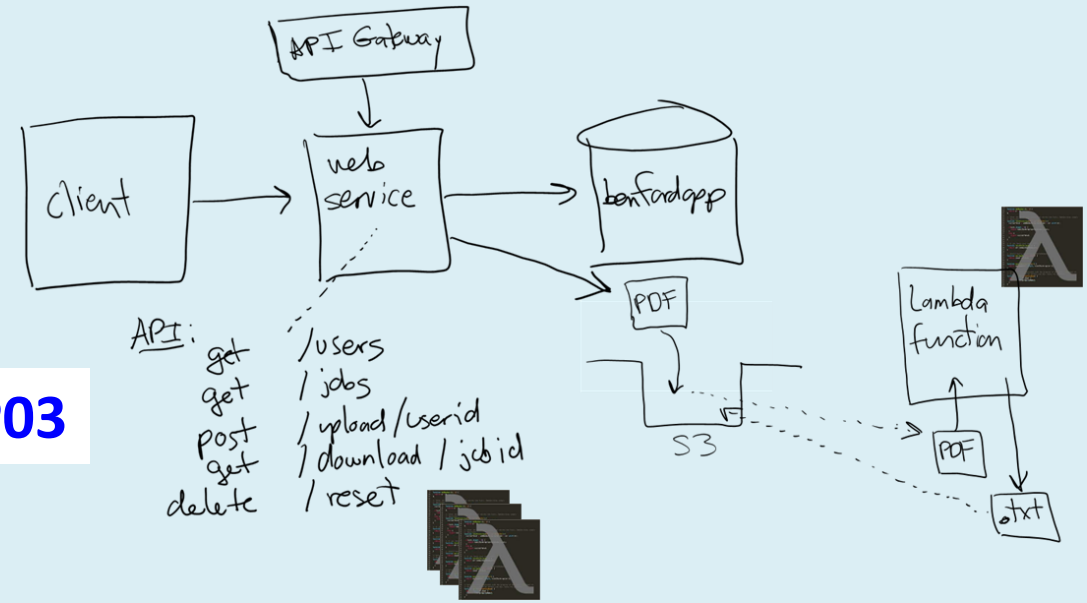
**P01**




**P02**

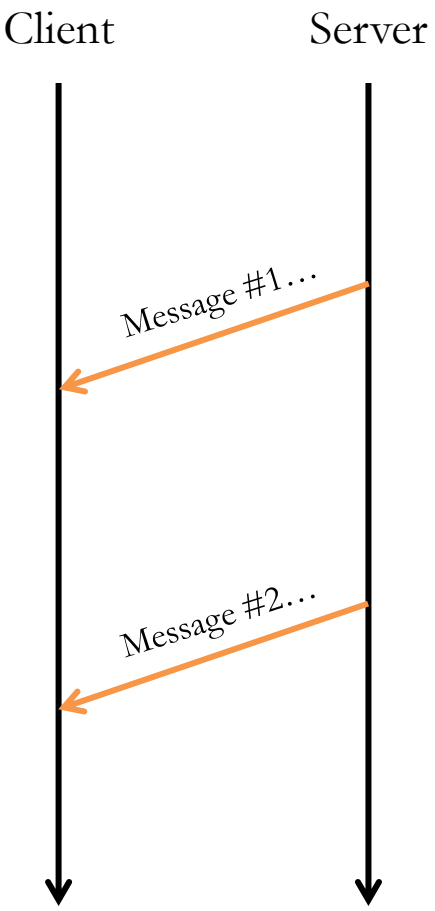



**P03**



# Asynchronous architectures

- **What if the server wants to initiate communication?**
  - *Notifications on your phone…*
  - *Uber: your car has arrived*
  - *GrubHub: your food has arrived*
  - *Amazon: your package has shipped*
- **Common use cases:**
  - *Texting*
  - *Messaging apps*
  - *Notification systems*
  - *Workflow systems (documents, food delivery)*
  - *Streaming services (music, audiobooks, videos)*

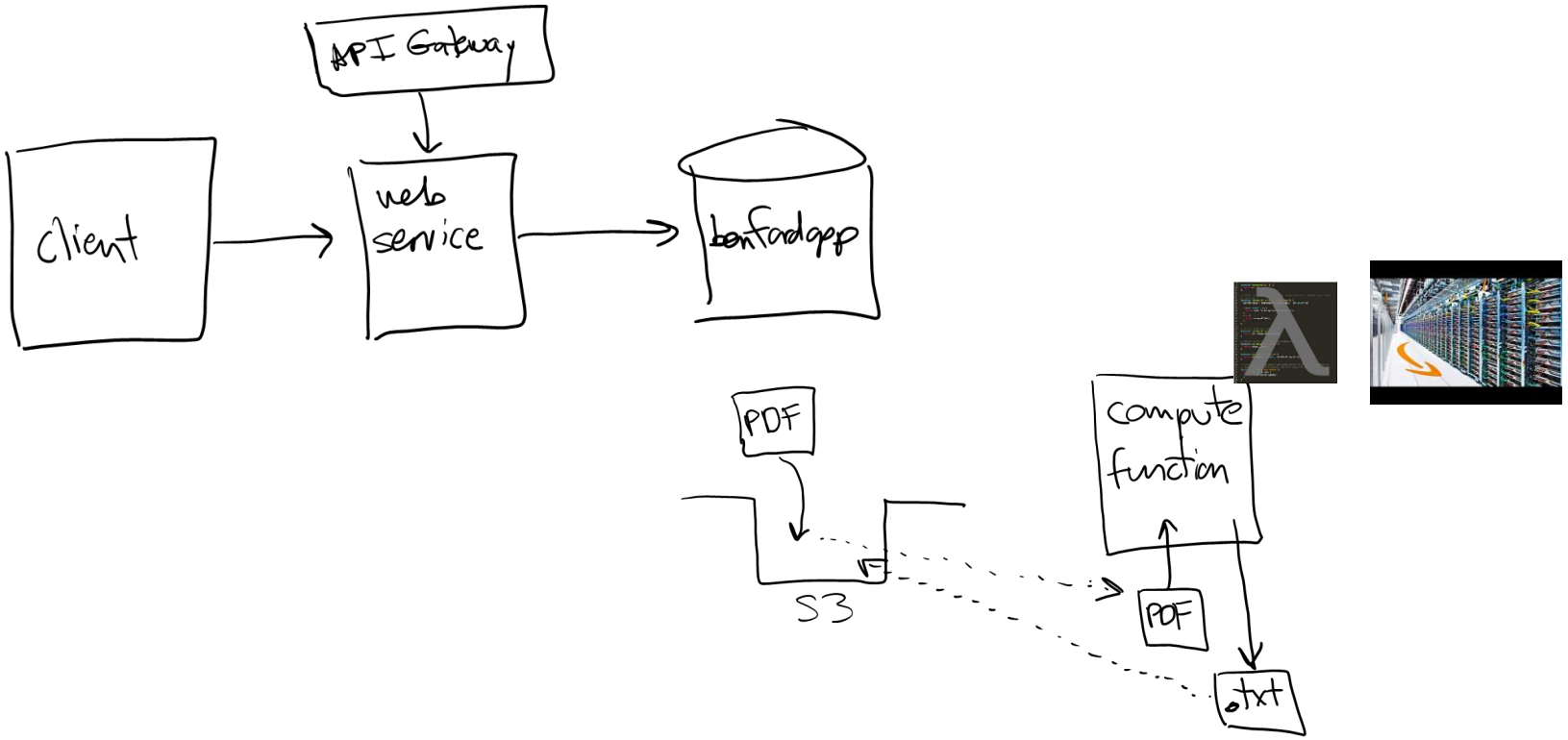


# Project 03…

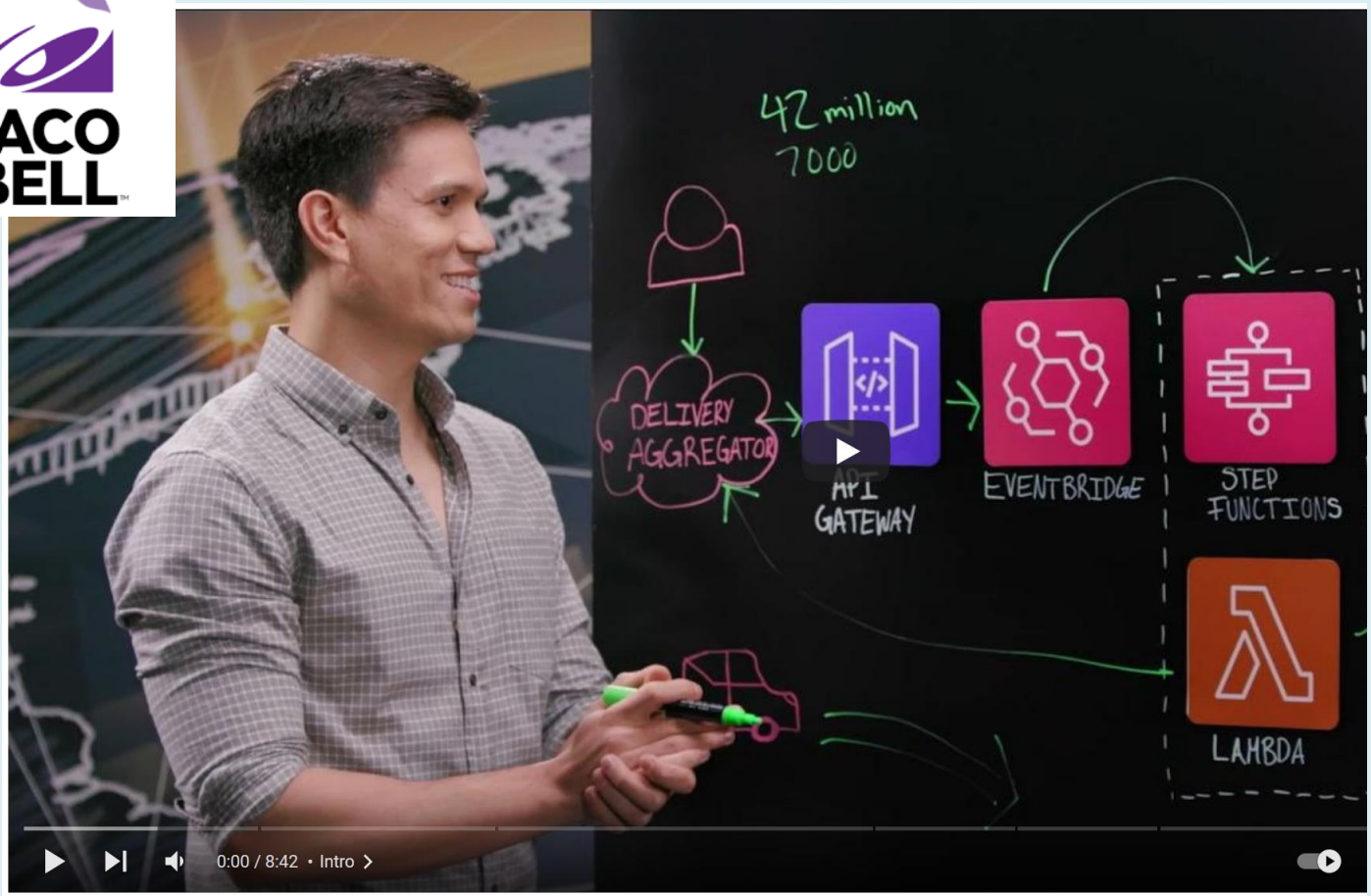
- **Project 03 introduced asynchronous aspects**
  - *Event-driven PDF analysis*
  - *Client-side polling to obtain results*

# Event-driven

- **Events are a component of asynchronous architectures…**

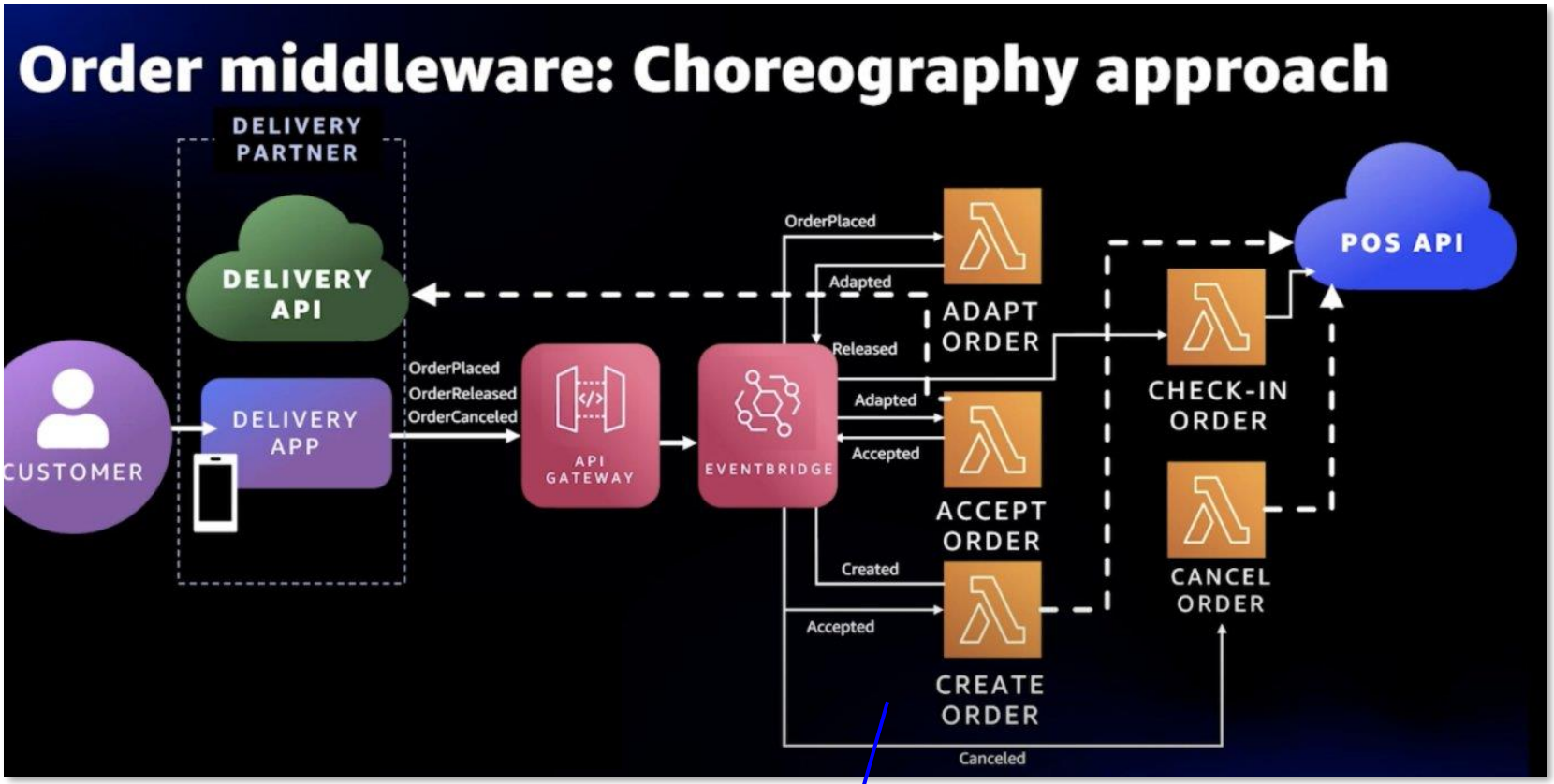


# Example




[Taco Bell: Order Middleware - Enabling Delivery Orders at Massive Scale - YouTube]

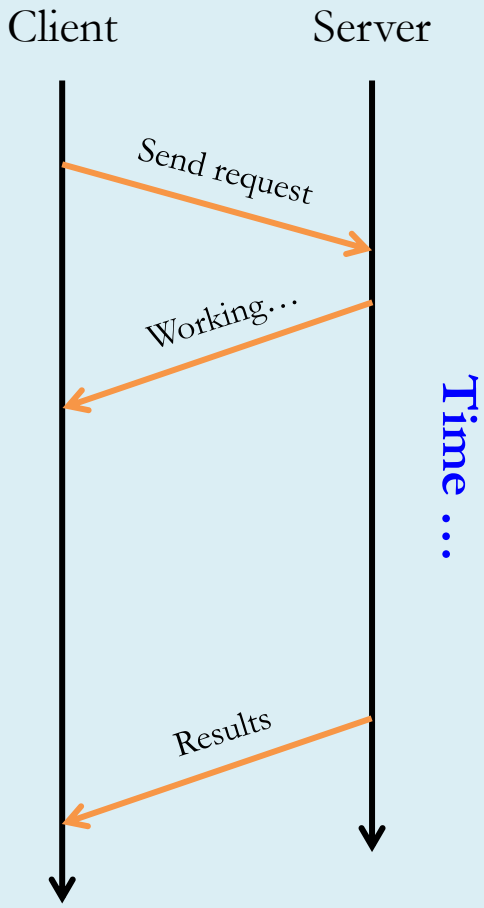
# Event-driven




**Finite state machine…**

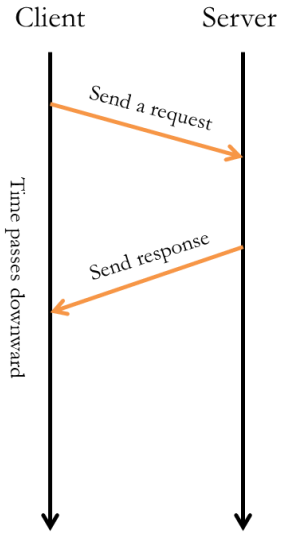
# Client-side polling

- **Project 03 also incorporates an asynchronous API**



# Synchronous API

- *Request:* upload a PDF

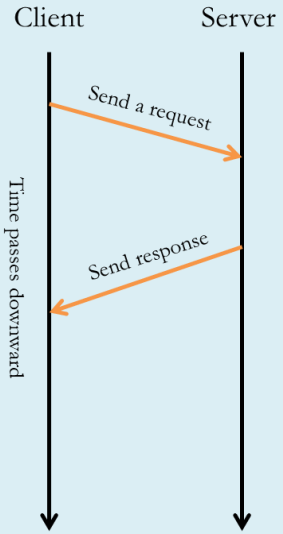



- *Response:* Here are the analysis results

**Synchronous APIs are simpler to think about, design, and build**

# Asynchronous API

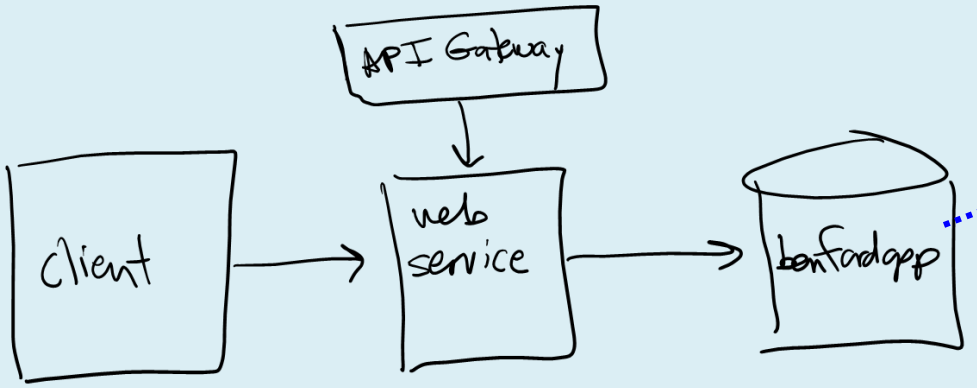
- *Request:* upload a PDF



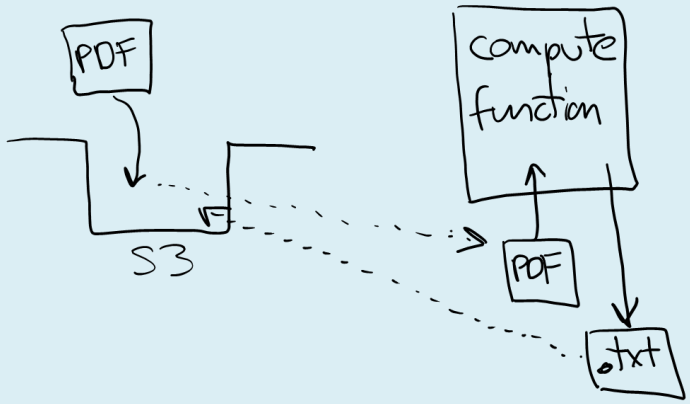

- *Response:* Here's the jobid, results are 'pending'

**Asynchronous APIs allow for longer / unpredictable processing times on the server. But this complicates the design, coding, and error handling.**

# Demo: project 03



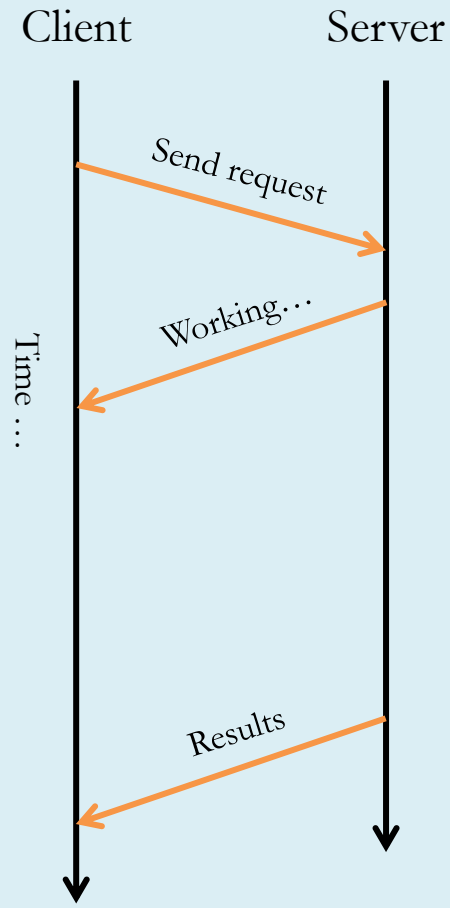

- **Job states:**
  1. *Uploaded*
  2. *Processing*
  3. *Completed*
  4. *Error*



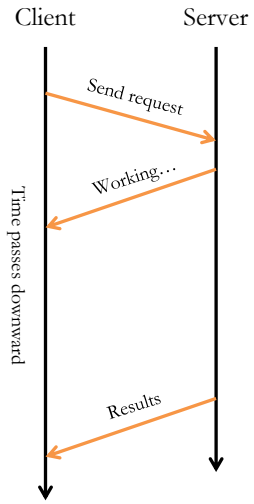
# How does client obtain results?

- **Options for the client?**
  (in order of complexity to config / implement)

  1. ***Client-side polling***
  2. ***Client callback***
  3. ***Websockets***



# Implementation #1: **Client-side polling**

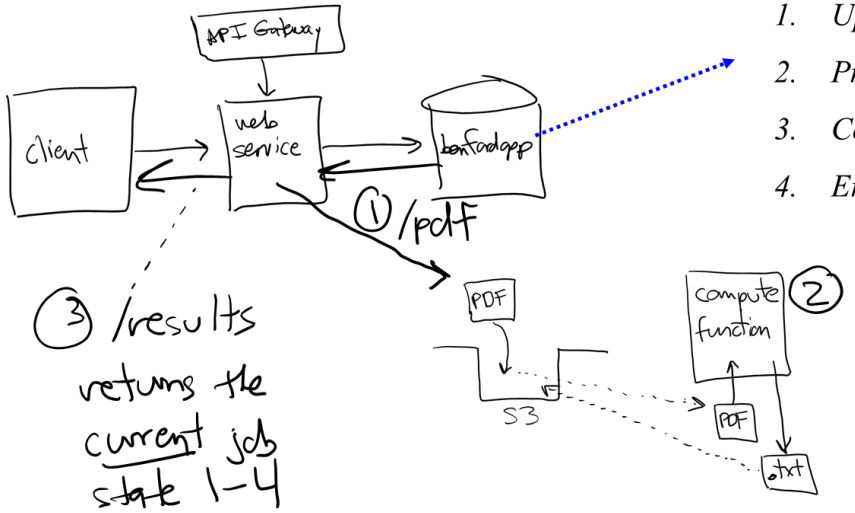



- Server stores *request record* in DB & returns a unique id
- When done, the server updates the record in the DB
- Client checks on results using the id, repeating as needed until results are available
- Easy to implement, least efficient

- **Example: Project 03**

```
jobid  = requests.post('/pdf/userid')

//
// polling loop:
//
while True:
  resp = requests.get('/results/jobid')
  if resp denotes completed or error:
    break
  else:
    display current status
    sleep
```



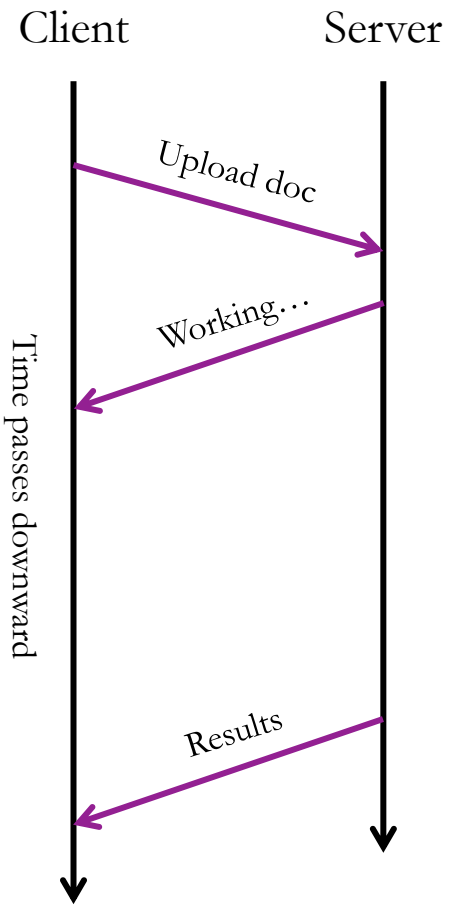

- **Job states:**
  1. *Uploaded*
  2. *Processing*
  3. *Completed*
  4. *Error*

# Implementation #2: **Client callback**

- Client provides a callback function (**webhook**) where it expects to receive a response
- Server uses callback to inform client when done

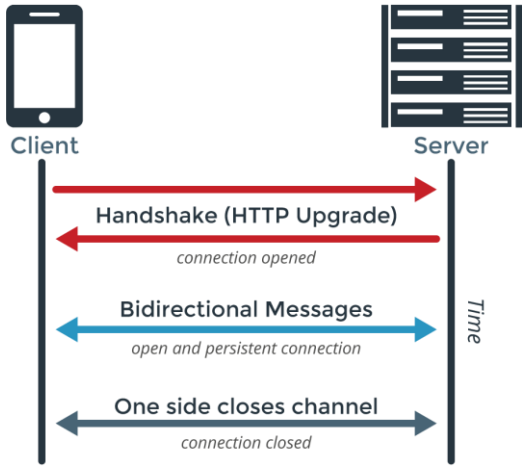
- Request → Response example:
  - *Client sends:* POST /uploadAttempt
    {"callback": "http://3.3.3.3:80/analysisComplete"}
  - Server sends: POST /analysisComplete

*This inplies **client** is running some sort of web server-like software…*



# Implementation #3: **Websockets**

- A Websocket is a **long-lived**, **bi-directional** network connection
  - *Similar to a TCP socket*
- Client creates websocket connection to server
  - *Client's connection ID is saved in a database*
- Client sends API requests through the websocket
- Server-side functions lookup connection id and push data back to client through websocket




*Requires fast, reliable internet connection, potentially more client-side config*

Details at:
https://hackernoon.com/websockets-api-gateway-9d4aca493d39

# That's it, thank you!