

# FastAPI 비동기 작업 지연으로 만난 상황

# uvicorn async

- pod probe체크를 실패하여 pod재실행
- 비동기함수에서 동기함수를 사용해서 지연되는 걸로 파악

```
app = FastAPI()

@app.get("/hello")
async def hello():
    time.sleep(5)
    return {"hello": "world"}
```

[비동기함수에서 동기함수 호출]

VS

```
async def hello():
    await asyncio.sleep(5)
    return {"hello": "world"}

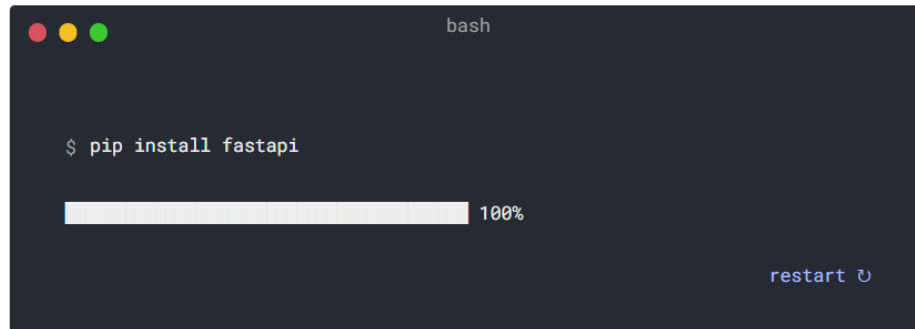
@app.get("/hello")
async def hello_endpoint():
    return await hello()
```

[비동기에서 비동기함수 호출]

# uvicorn async

- 테스트하기 전 FastAPI 비동기 처리 과정을 살짝 이해가 필요할 듯!
- FastAPI는 ASGI Server가 실행하는 웹 애플리케이션

## Installation

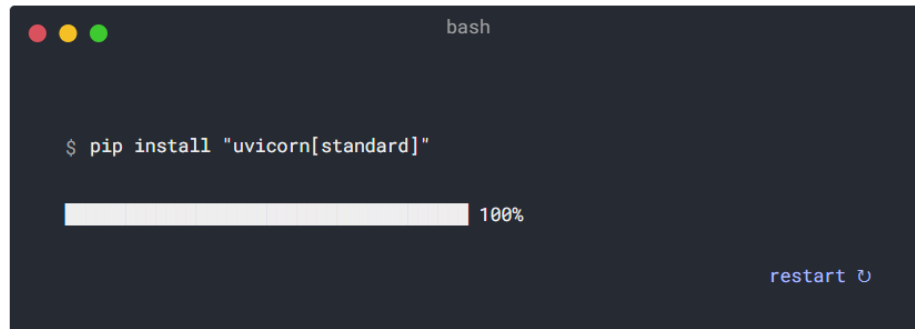


```
bash

$ pip install fastapi
```

A progress bar indicates 100% completion. A "restart" button with a refresh icon is visible in the bottom right corner.

You will also need an **ASGI** server, for production such as [Uvicorn](#) or [Hypercorn](#).



```
bash

$ pip install "uvicorn[standard]"
```

A progress bar indicates 100% completion. A "restart" button with a refresh icon is visible in the bottom right corner.

# uvicorn async

- 그리고 Starlette(ASGI Framework) sub class로 구현?되어 있어 Starlette특성을 가지고 있음

## Starlette features 📌

**FastAPI** is fully compatible with (and based on) **Starlette** [↔]. So, any additional Starlette code you have, will also work.

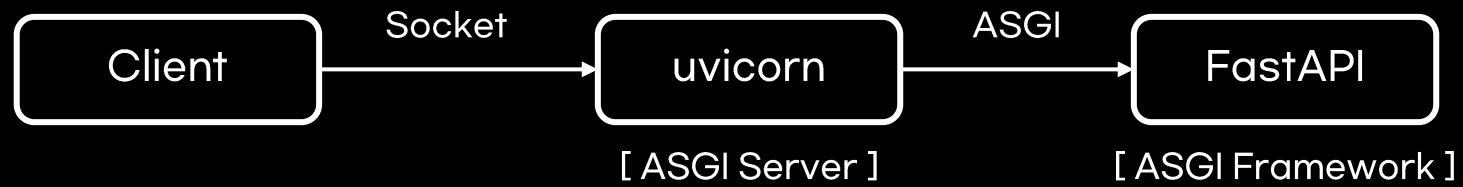
**FastAPI** is actually a sub-class of **Starlette**. So, if you already know or use Starlette, most of the functionality will work the same way.

With **FastAPI** you get all of **Starlette**'s features (as FastAPI is just Starlette on steroids):

- Seriously impressive performance. It is **one of the fastest Python frameworks available**, on par with **NodeJS** and **Go** [↔].
- **WebSocket** support.

# uvicorn async

## - 전체 흐름 요약

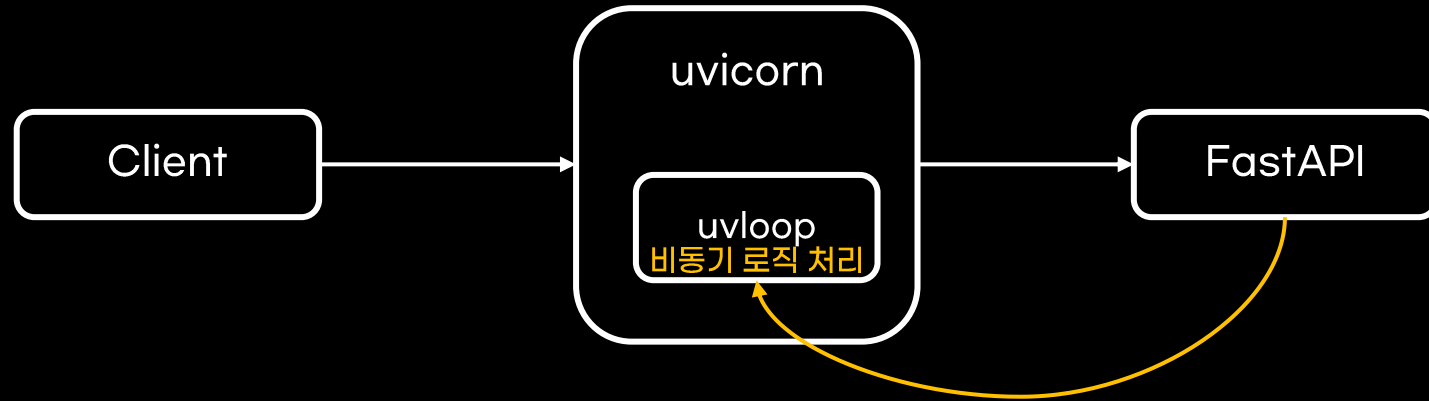


유사



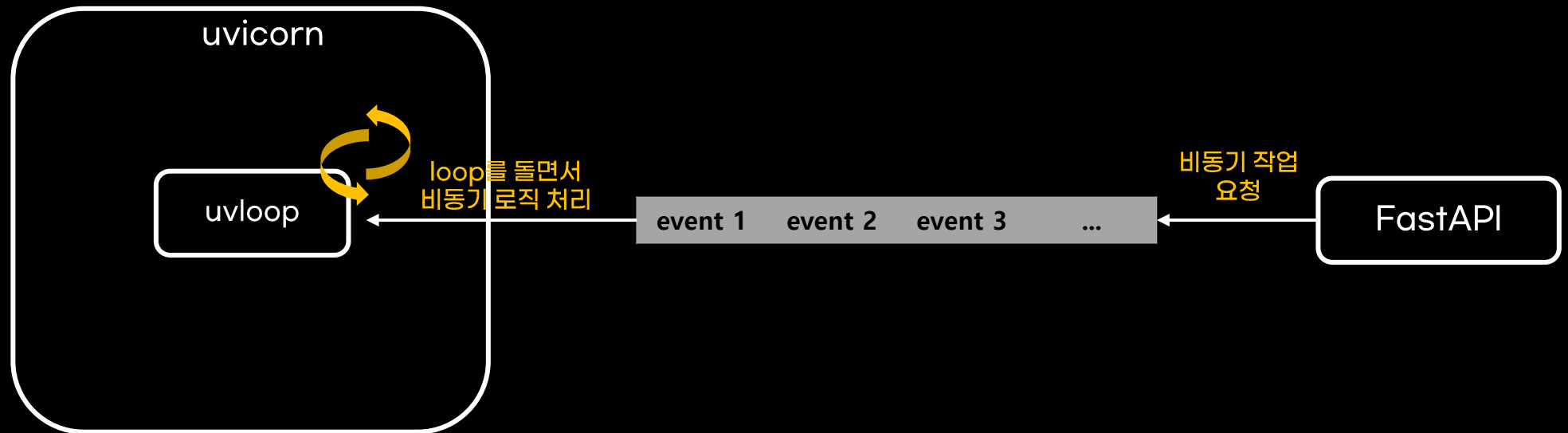
# uvicorn async

- 비동기는 uvloop에서 처리



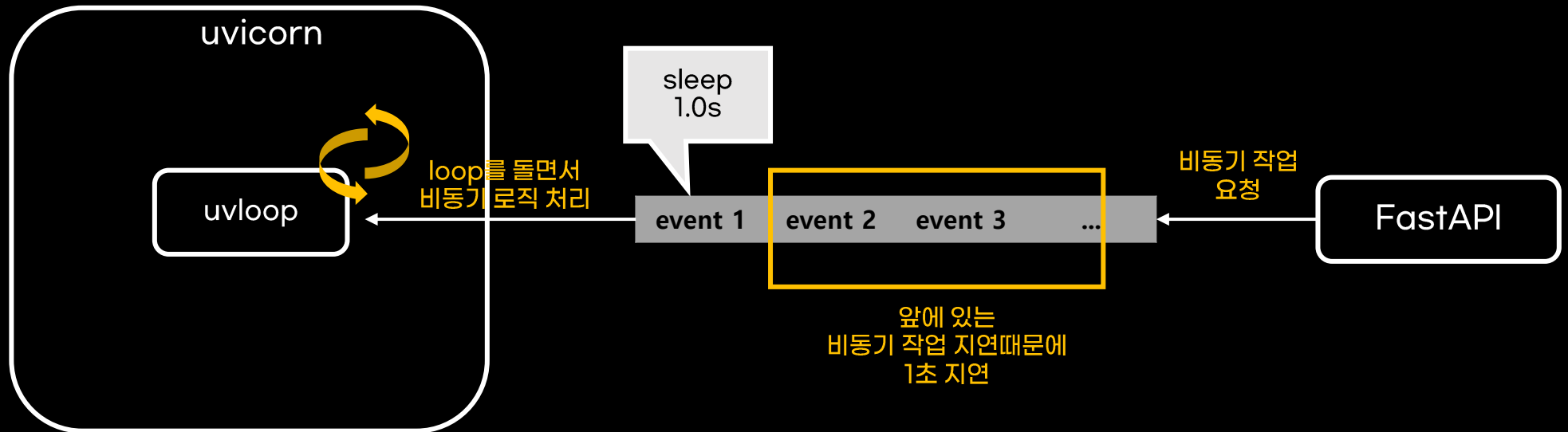
# uvicorn async

- 아마도... 비동기 event queue가 있고  
uvloop가 loop를 실행하면서 차례대로 실행



# uvicorn async

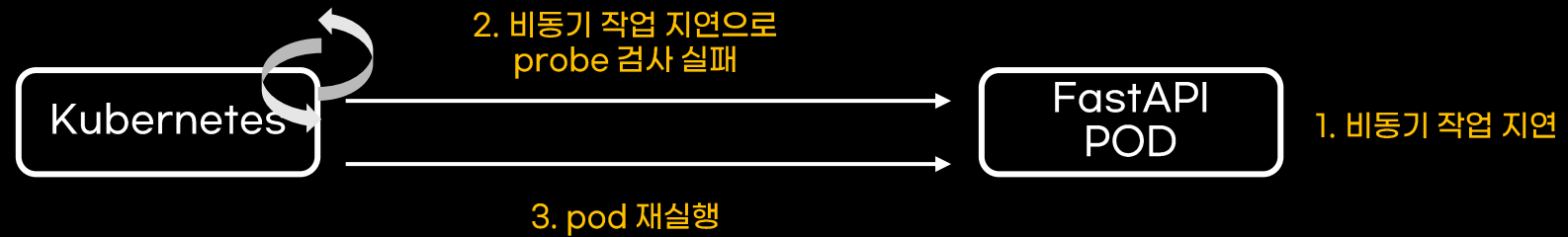
- 비동기 작업 한 개가 지연(예: 동기 request)이 생기면  
나머지 비동기 작업도 그만큼 지연





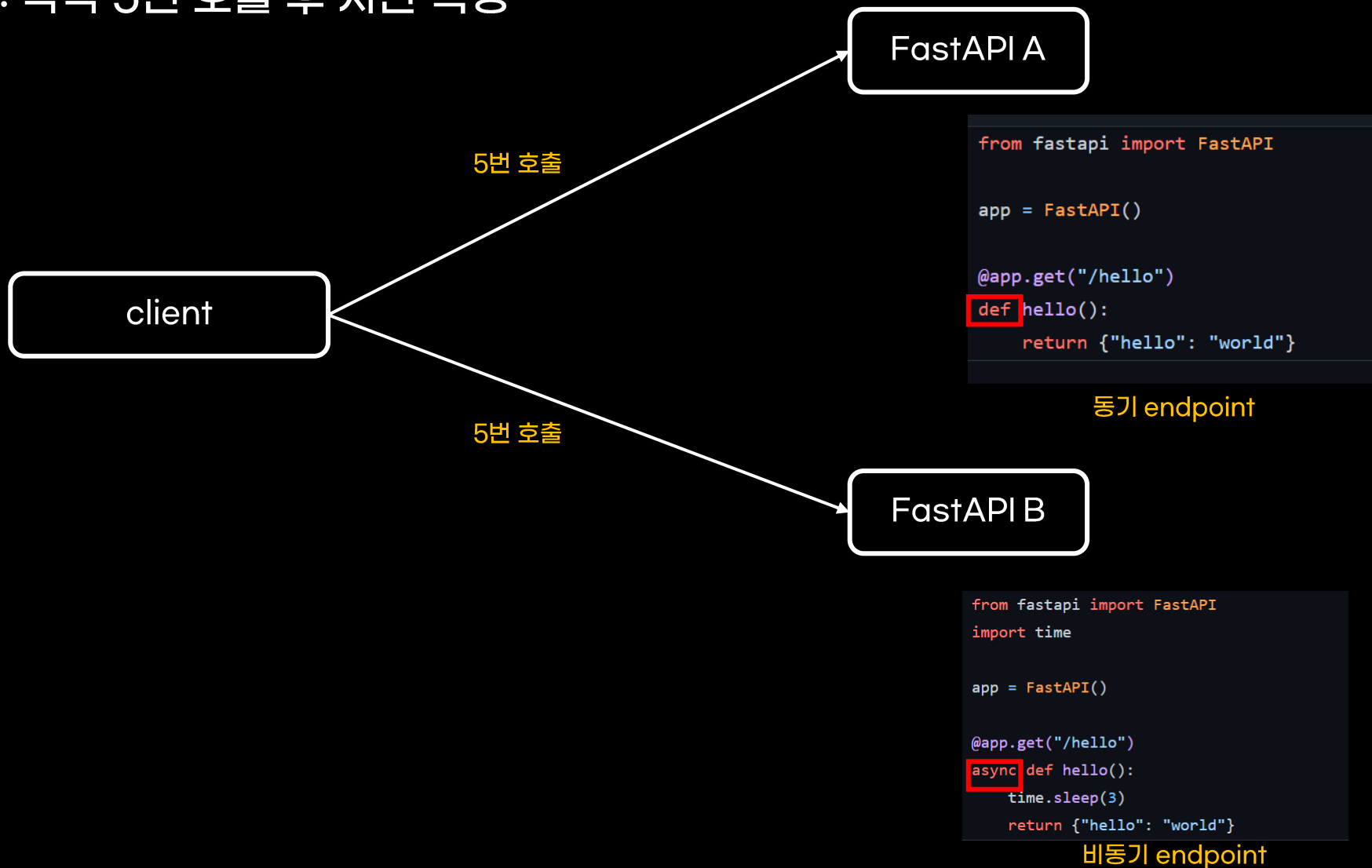
# uvicorn async

- 비동기 작업 지연때문에 liveness probe검사에 영향
- 결국, pod가 자동 재실행됨



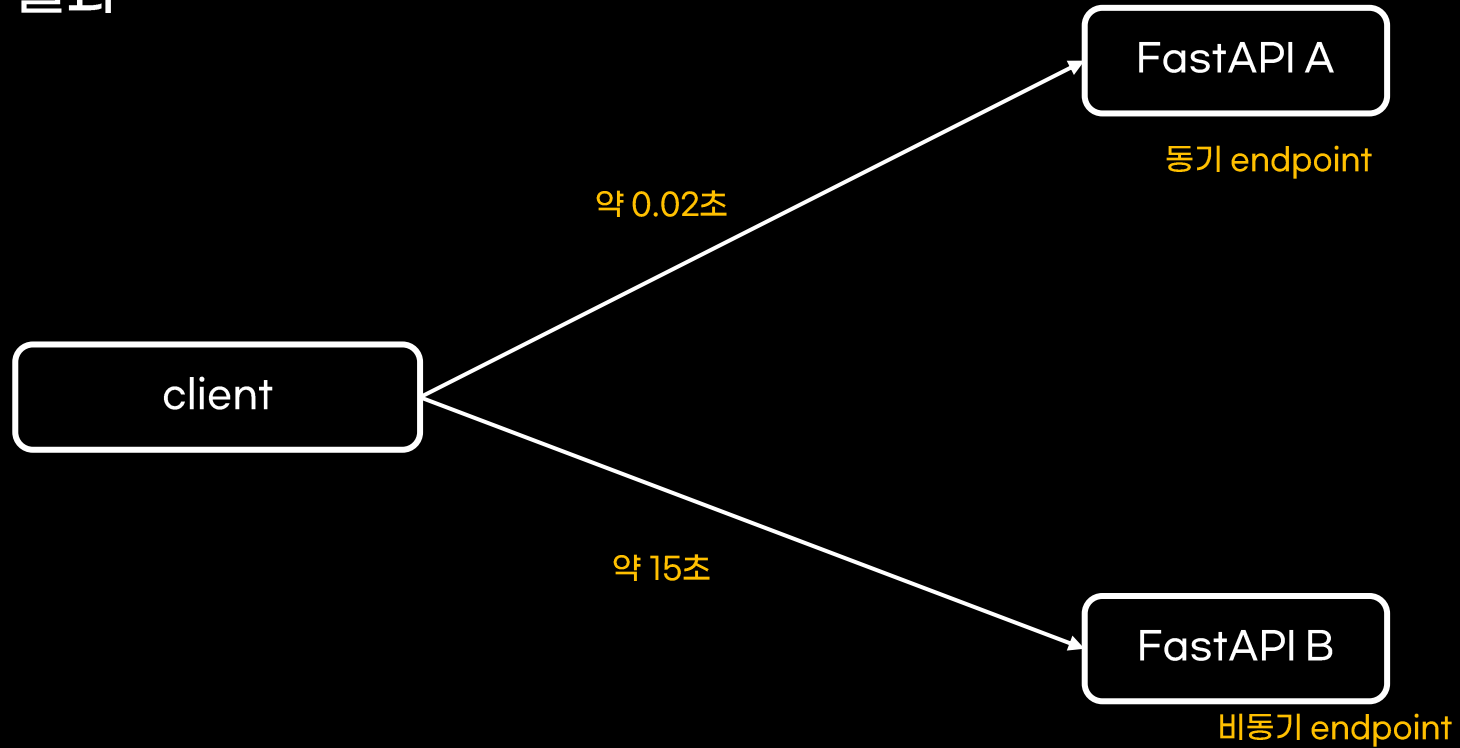
# uvicorn async

- 예제1: 각각 5번 호출 후 시간 측정



# uvicorn async

## - 예제1 결과



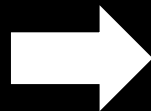
# uvicorn async

- 예제2 동기 지연을 비동기 지연으로 수정

```
from fastapi import FastAPI
import time

app = FastAPI()

@app.get("/hello")
async def hello():
    time.sleep(3)
    return {"hello": "world"}
```



```
import asyncio
from fastapi import FastAPI

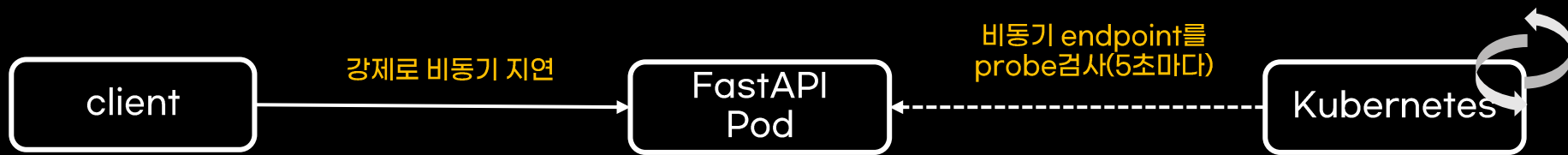
app = FastAPI()

async def hello():
    await asyncio.sleep(3)
    return {"hello": "world"}

@app.get("/hello")
async def hello_endpoint():
    return await hello()
```

# uvicorn async

- 예제3: Kubernetes probe
  - 비동기 지연으로 pod가 재실행되는지 확인



```
from fastapi import FastAPI
import time

app = FastAPI()

@app.get("/healthcheck")
async def healthcheck():
    return {"health": "check"}

@app.get("/hello")
async def hello():
    time.sleep(120)
    return {"hello": "world"}
```

```
livenessProbe:
  httpGet:
    path: /healthcheck
    port: 80
  initialDelaySeconds: 10
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 2
  periodSeconds: 5
```