Concurrency

vs

Parallelism

A bit about me

Naren

Backend/Product Engineer

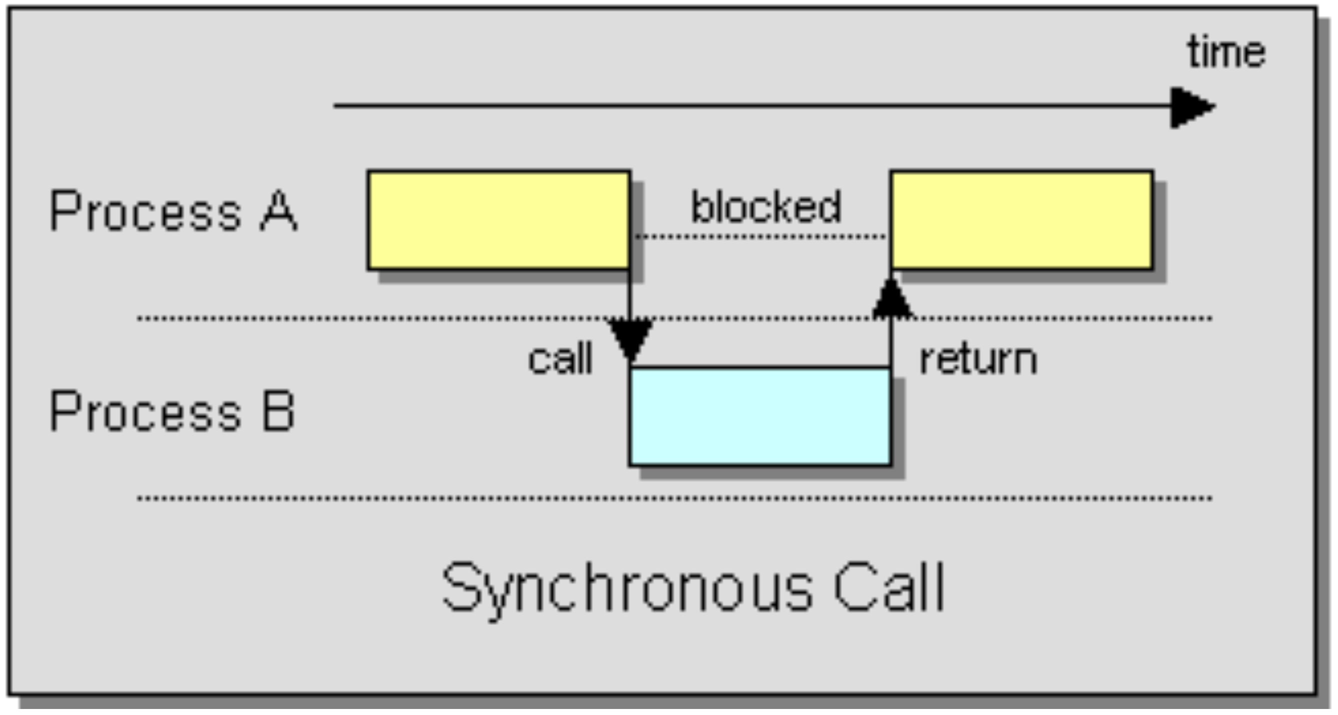
Scaling A.I to millions
@ MadStreetDen

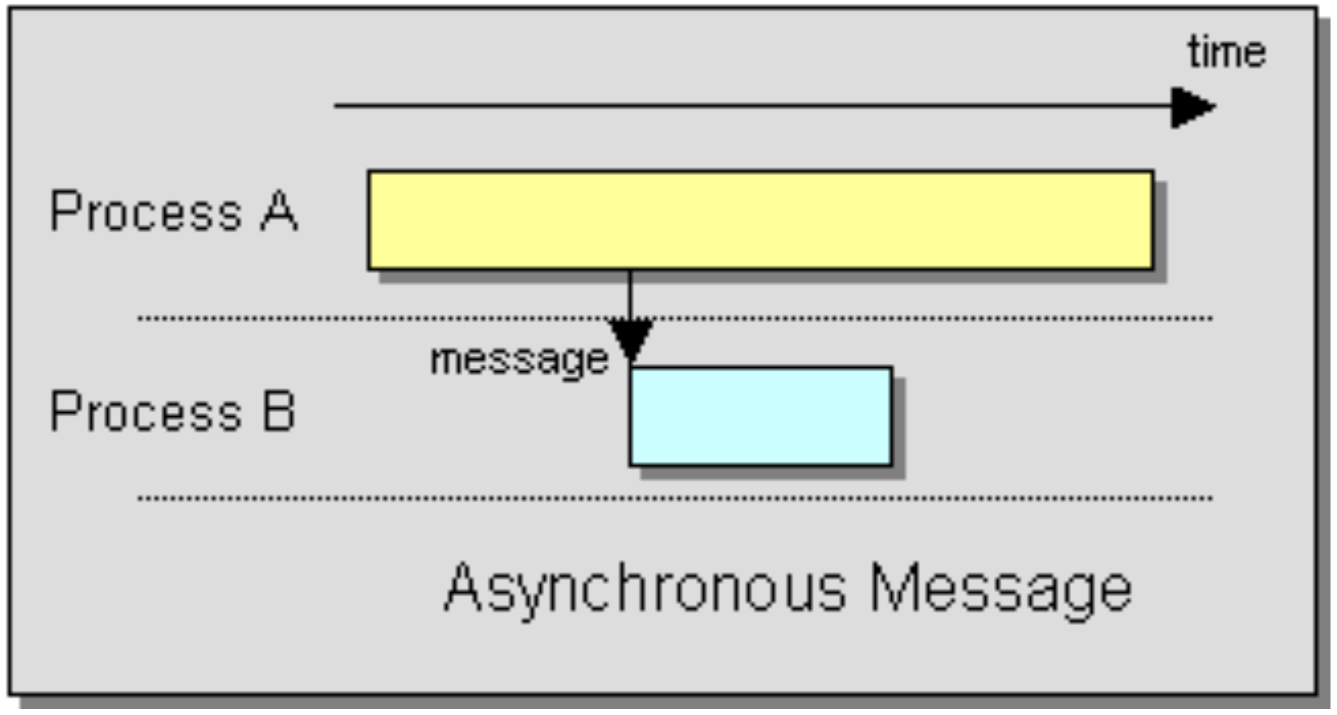
python, golang, FOSS, cycling, travel

twitter :
@DudeWhoCode

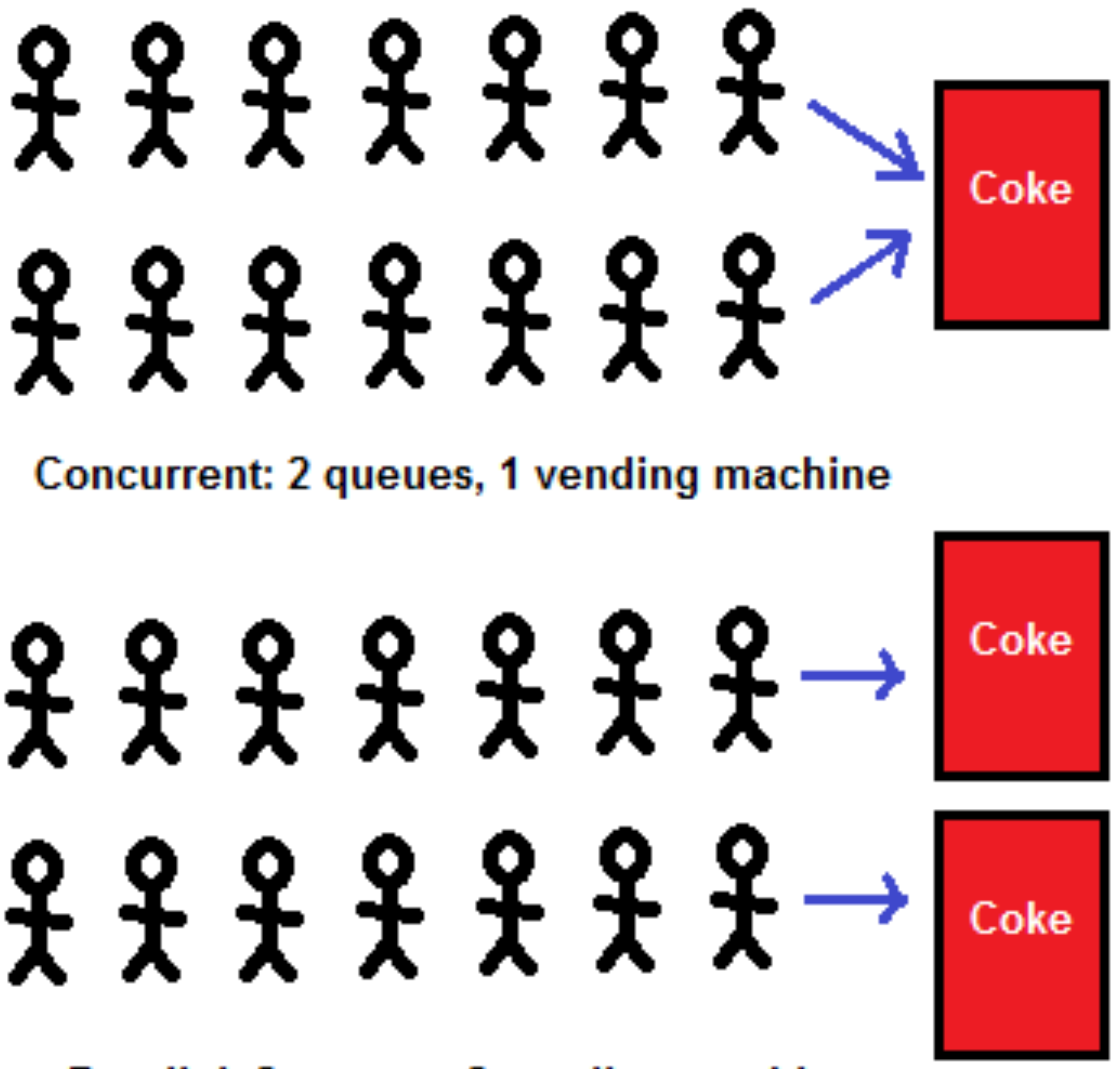
www.dudewho.codes

Sync vs Async





Concurrent vs Parallel



Concurrent: 2 queues, 1 vending machine

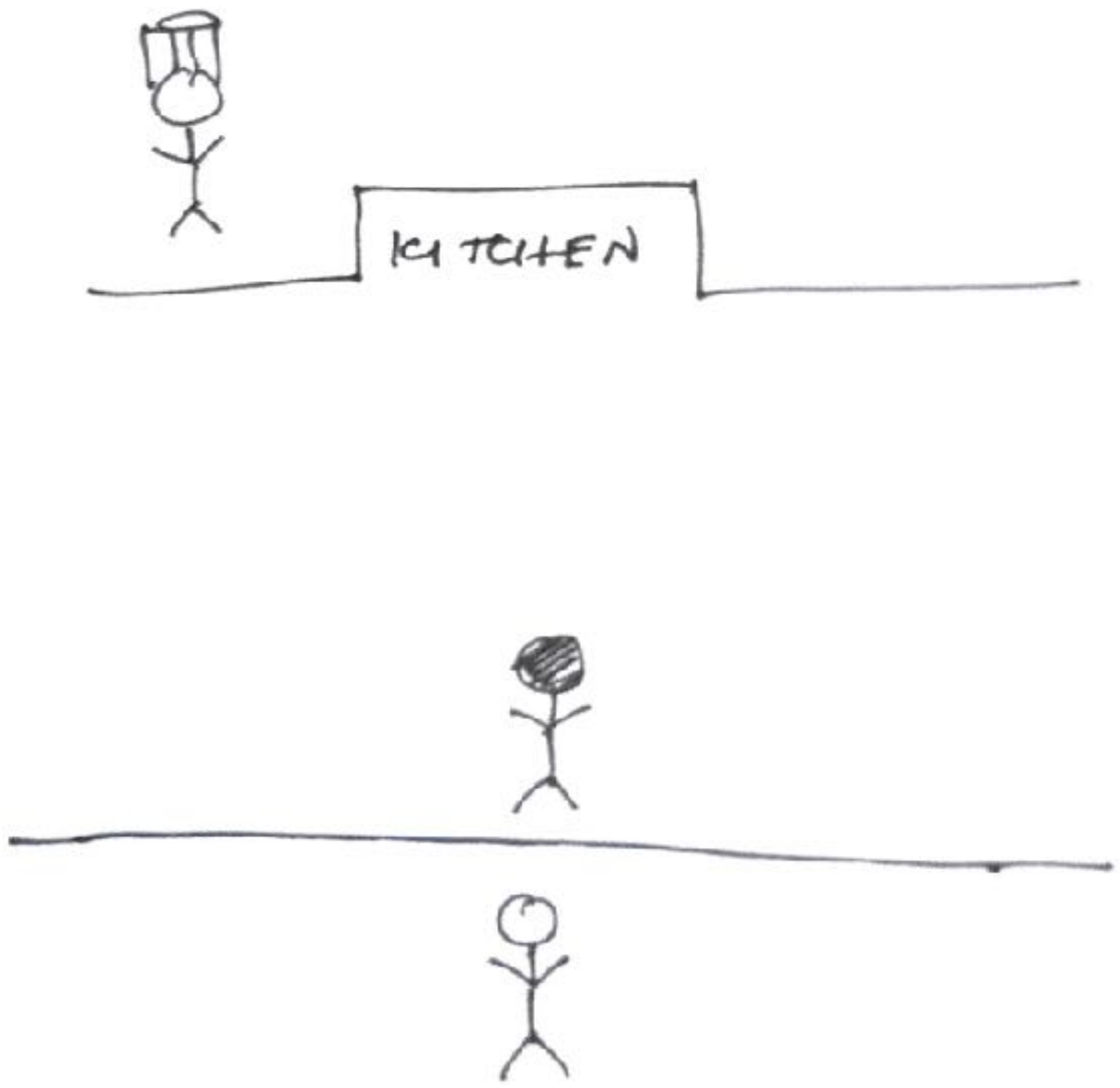
Parallel: 2 queues, 2 vending machines

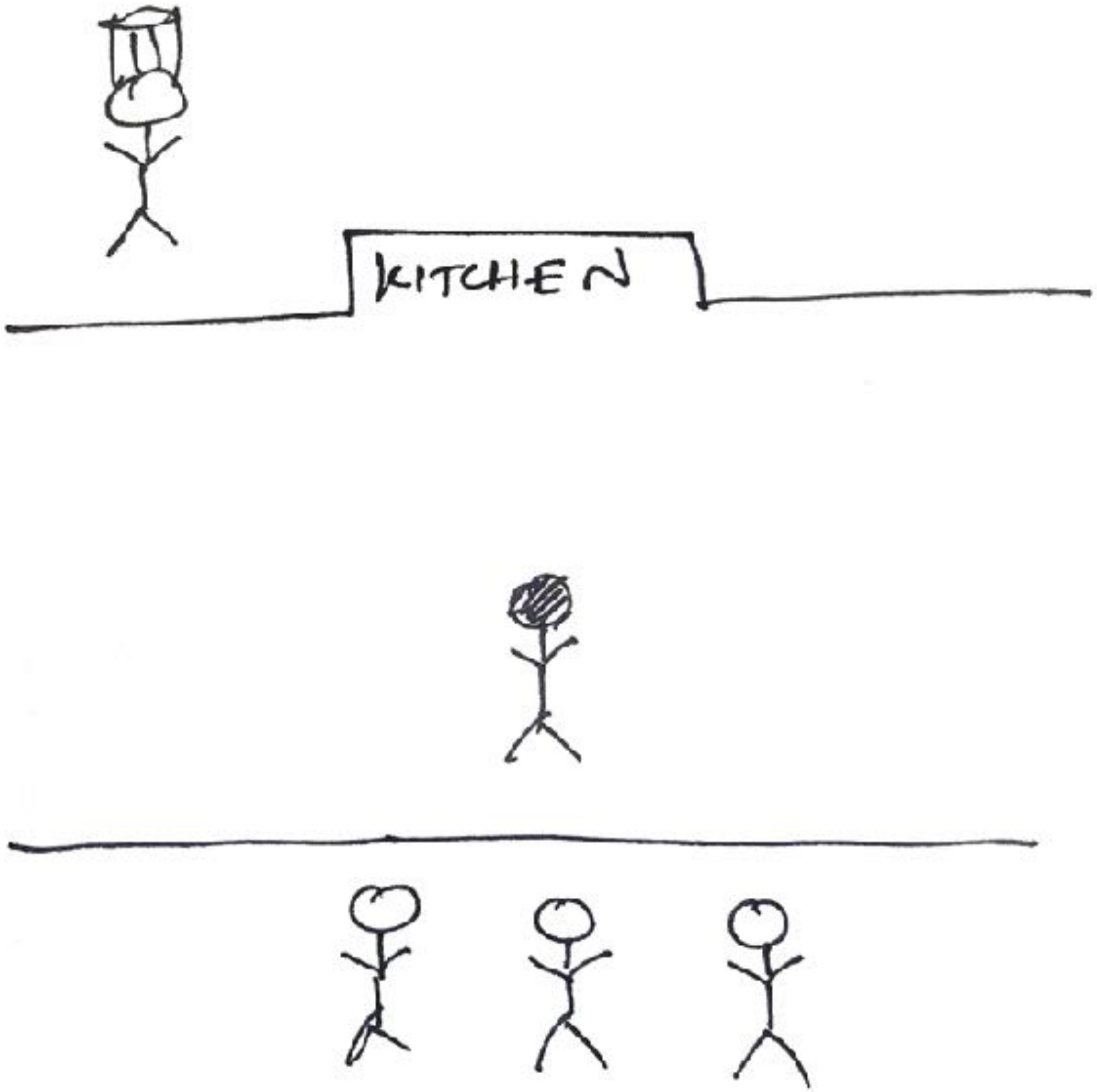
Recap

- **Sync:** Blocking operations.
- **Async:** Non blocking operations.
- **Concurrency:** Making progress together.
- **Parallelism:** Making progress in parallel.

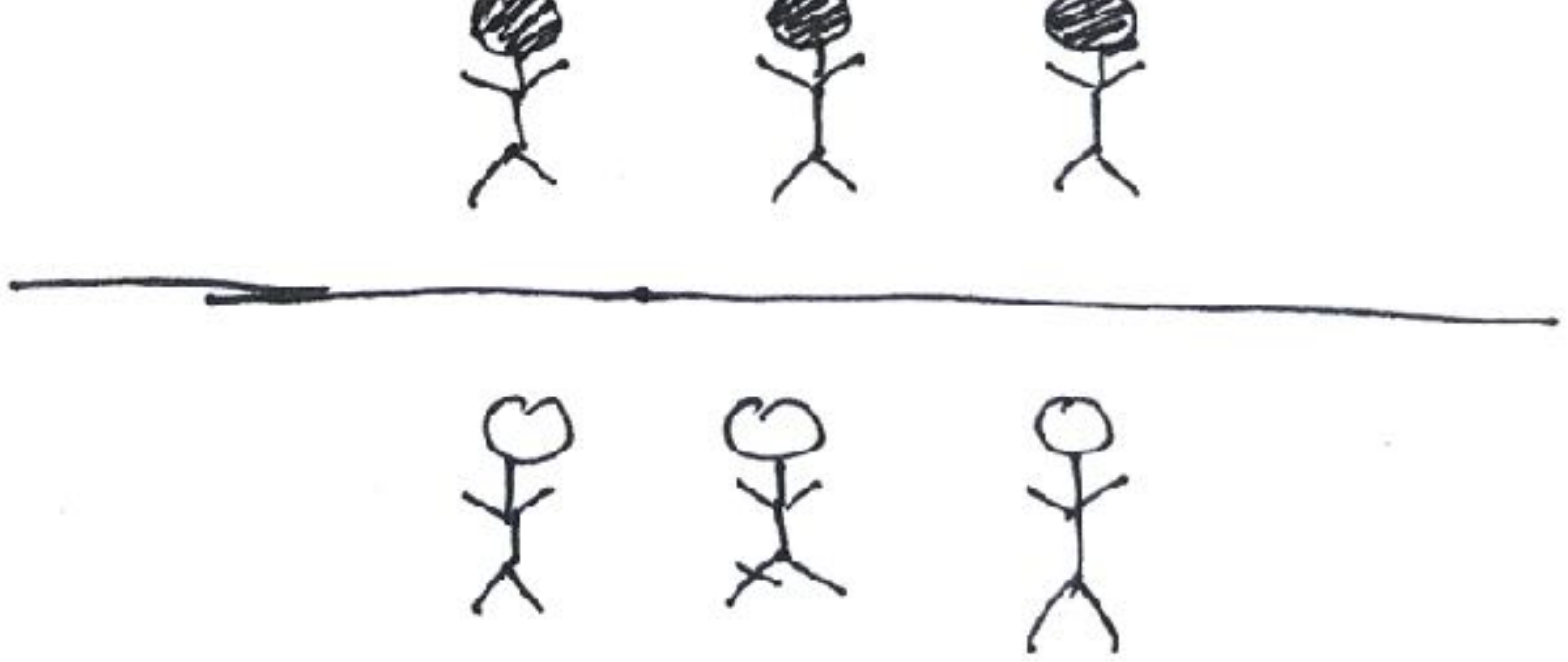
Concurrency is not Parallelism

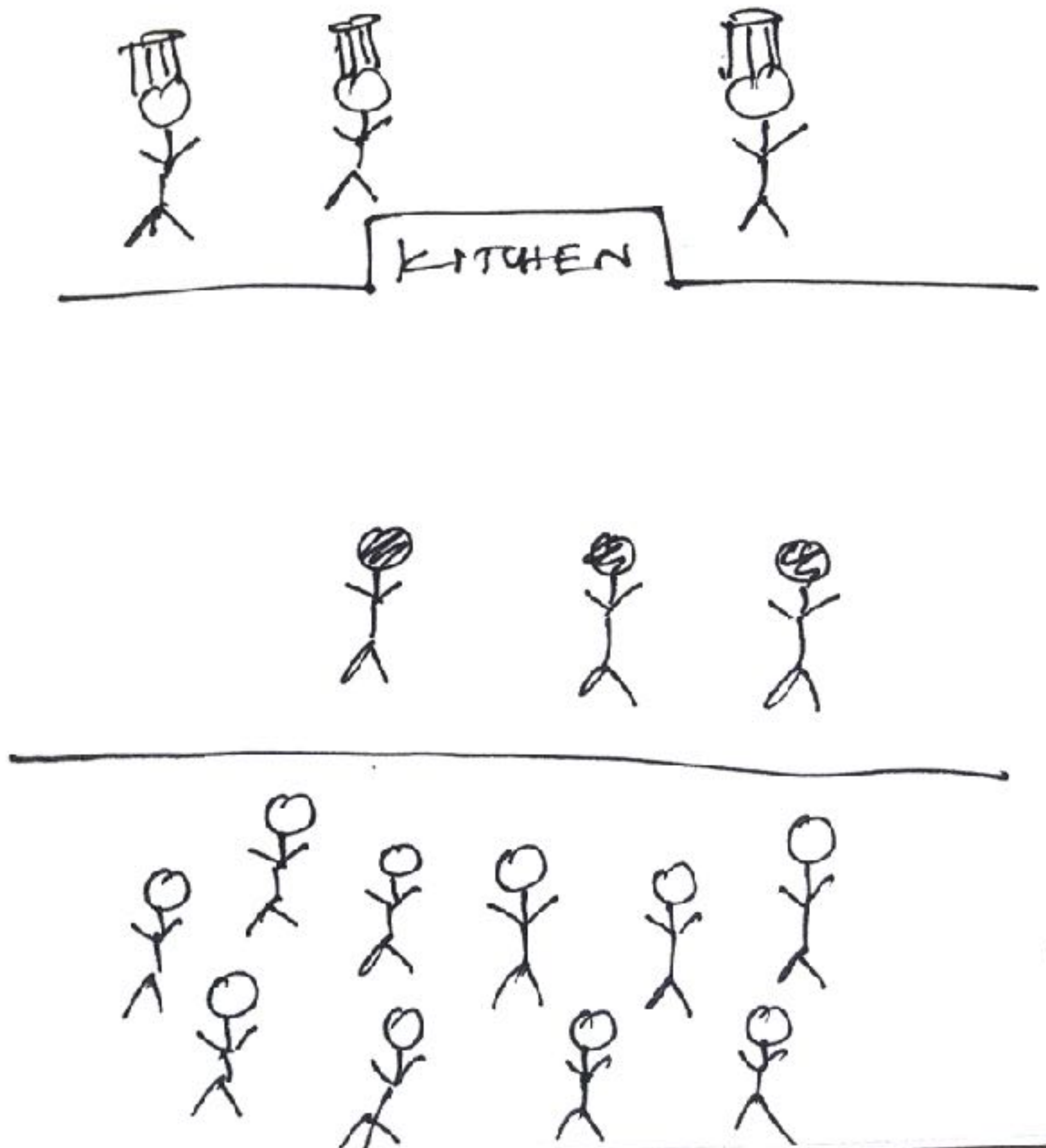
Concurrency is not Parallelism

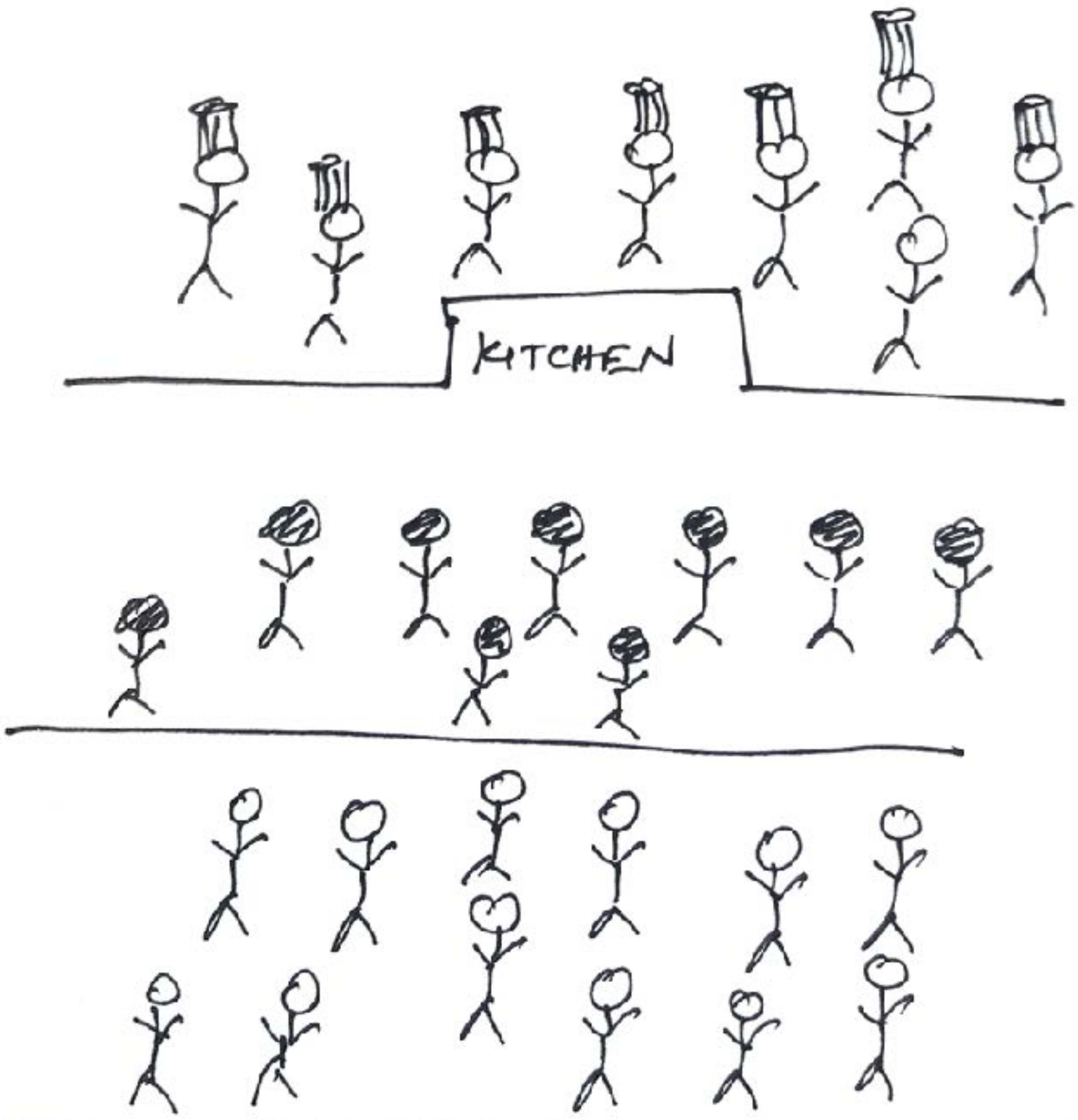

KITCHEN

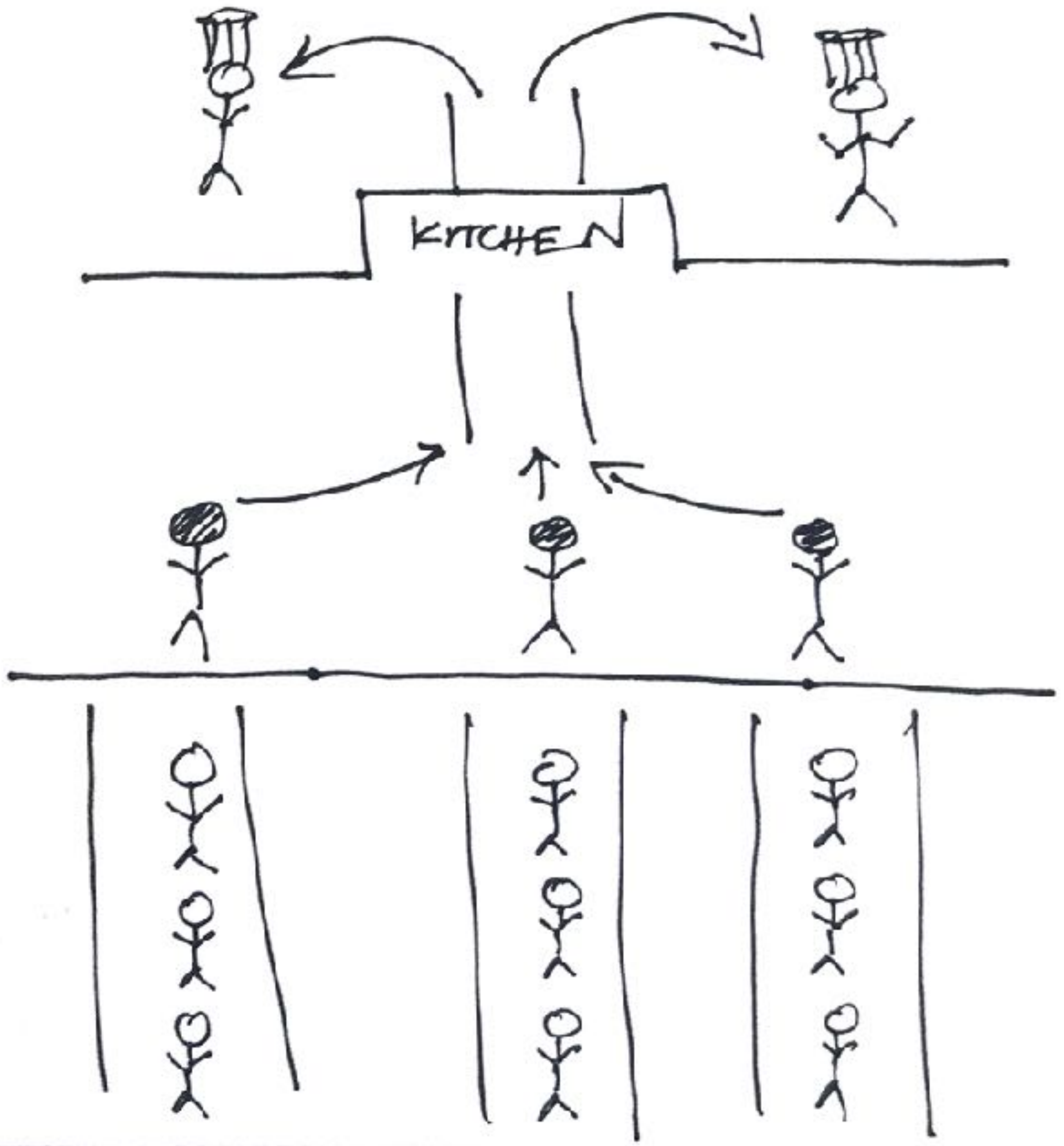

KITCHEN

KITCHEN




KITCHEN


KITCHEN


KITCHEN

asyncio

- Python 3 library to write/execute your code asynchronously.

Event Loop

- Manages and distributes the execution of different tasks.

- Responsible for registering the tasks and distributing flow of control between them

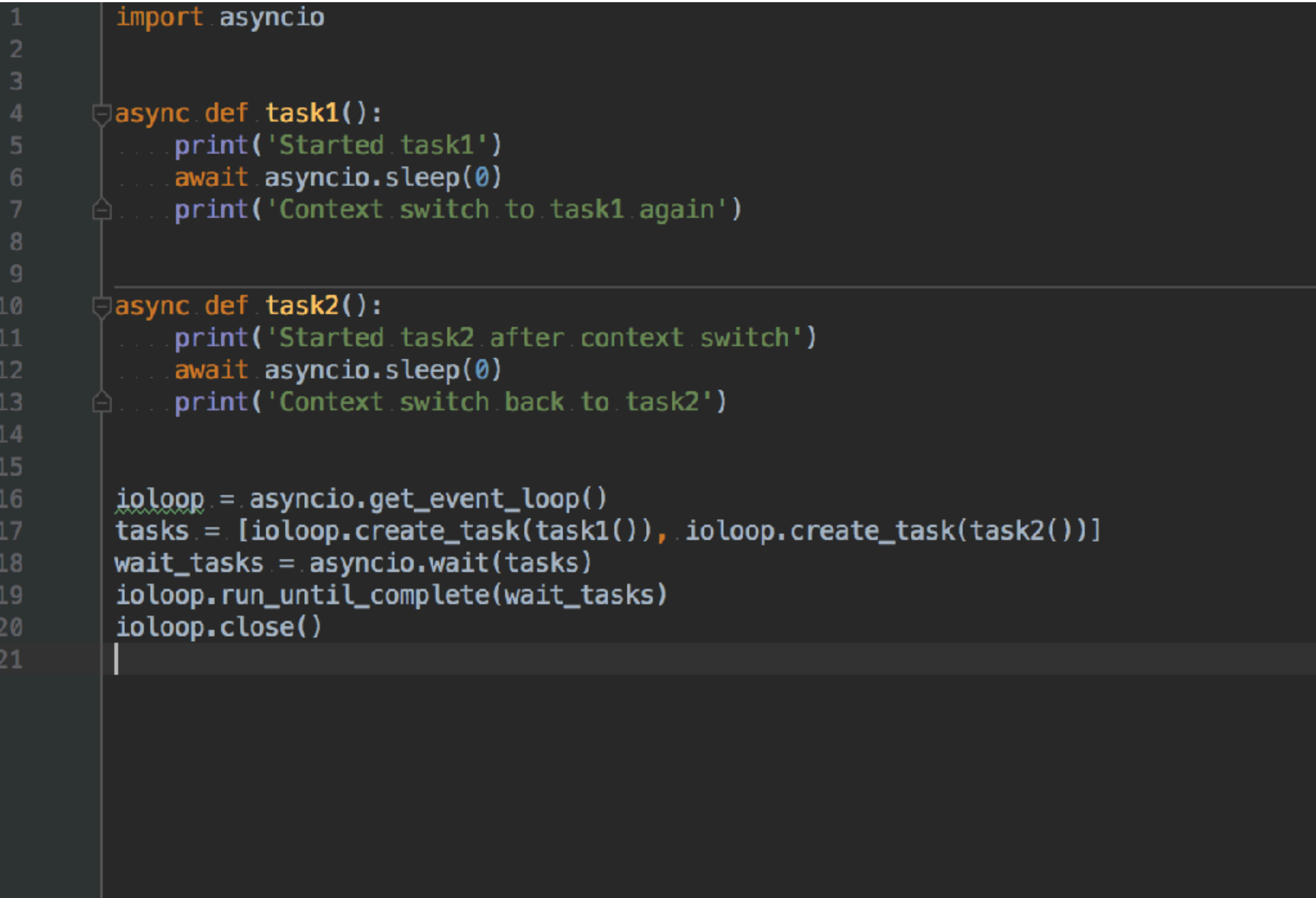
Coroutines

- Special functions that on **await** they release the flow of control back to the event loop.
- Similar to python generators
- A coroutine is scheduled using an event loop

Futures

- Objects that represent the result of a task.
- The task can be completed or unfinished.
- Object may be exceptions too.

Context Switch



```python
import asyncio


async def task1():
    print('Started task1')
    await asyncio.sleep(0)
    print('Context switch to task1 again')


async def task2():
    print('Started task2 after context switch')
    await asyncio.sleep(0)
    print('Context switch back to task2')


ioloop = asyncio.get_event_loop()
tasks = [ioloop.create_task(task1()), ioloop.create_task(task2())]
wait_tasks = asyncio.wait(tasks)
ioloop.run_until_complete(wait_tasks)
ioloop.close()
```

Demo

Threading vs Async

- **Async :** You decide when a piece of code can take back control using **await**
- **Threading** : Python scheduler takes care of this and it may lose control anytime.

Summary

- **Sync:** Blocking operations.
- **Async:** Non blocking operations.
- **Concurrency:** Making progress together.
- **Parallelism:** Making progress in parallel.

Summary

- Python 3, asyncio, aiohttp, aiofiles
- Eventloops, co-routines, futures

Where to go next?

- Python Documentation : www.bit.ly/asyncio-docs
- Detailed tutorial on asyncio : http://bit.ly/asyncio-tutorial

slides :
dudewho.codes/
talks

Thank you
@DudeWhoCode