ThoughtWorks®

Writing Robust and Maintainable Concurrent Program

Wilson Tjhi

@wilsontjhi

Github: wilsontjhi/pycon-asia-2018

What lessons can we learn from food seller about concurrency?

Making of kue lapis sagu

Making of kue lapis sagu

```
def build_layer(tray):
    start_mixer()
    pour_colour_mixture(colorama.Back.MAGENTA, tray)
    pour_colour_mixture(colorama.Back.GREEN, tray)
    pour_colour_mixture(colorama.Back.WHITE, tray)
```

Making of kue lapis sagu

```
def build_layer(tray):
    start_mixer()
    pour_colour_mixture(colorama.Back.MAGENTA, tray)
    pour_colour_mixture(colorama.Back.GREEN, tray)
    pour_colour_mixture(colorama.Back.WHITE, tray)


def start_mixer():
    perform_operation(1.0)


def pour_colour_mixture(color, tray):
    perform_operation(0.005)
    tray.append(color + '                    ')
```

Making of kue lapis sagu

```
def main():
    tray = []
    for _ in range(4):
        build_layer(tray)
```

Multithreading

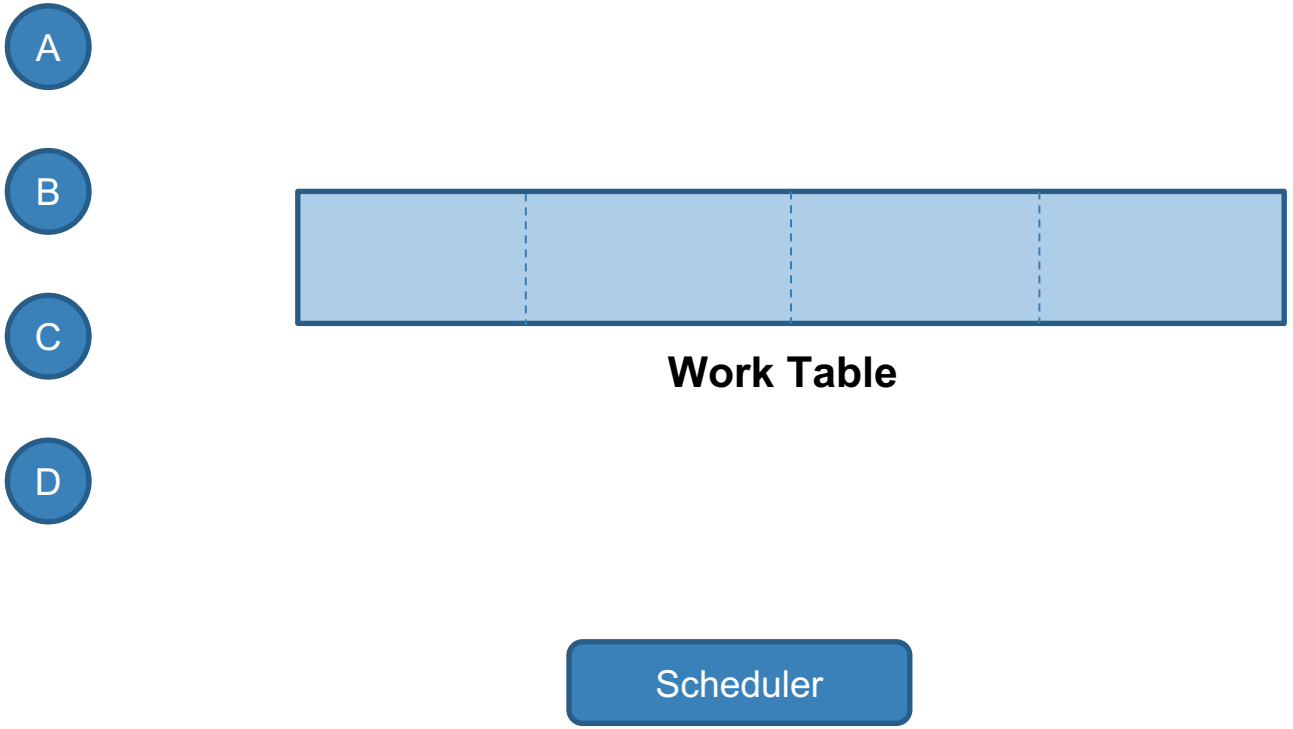
```
from threading import Thread

def main():
    tray = []
    for _ in range(4):
        t = Thread(target=build_layer, args=(tray,))
        t.start()
```

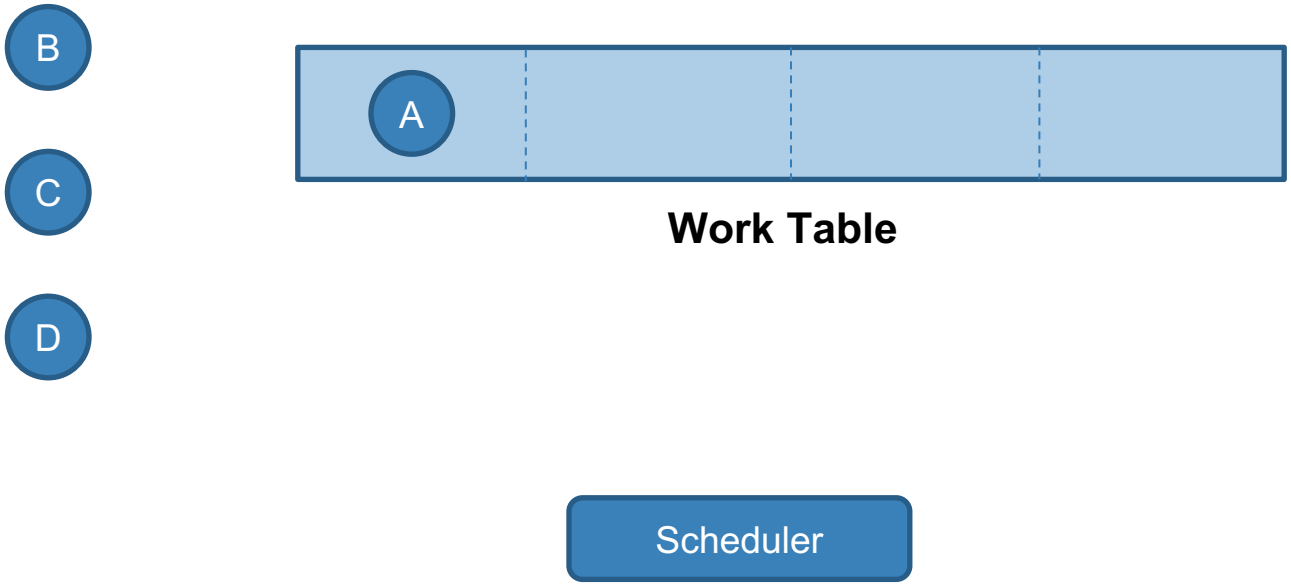
Multithreading

- Python interpreter is not thread safe
- GIL (Global Interpreter Lock) ensures that only 1 thread can run at a time

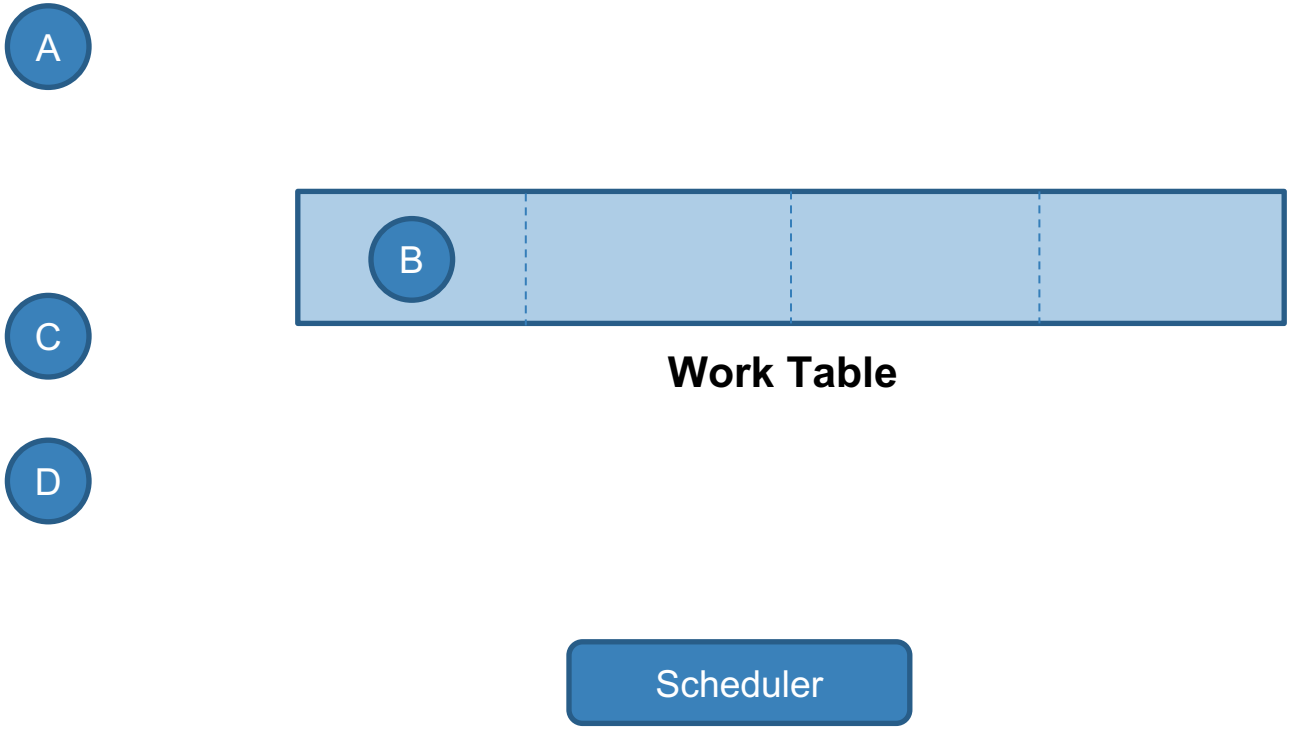
Multithreading



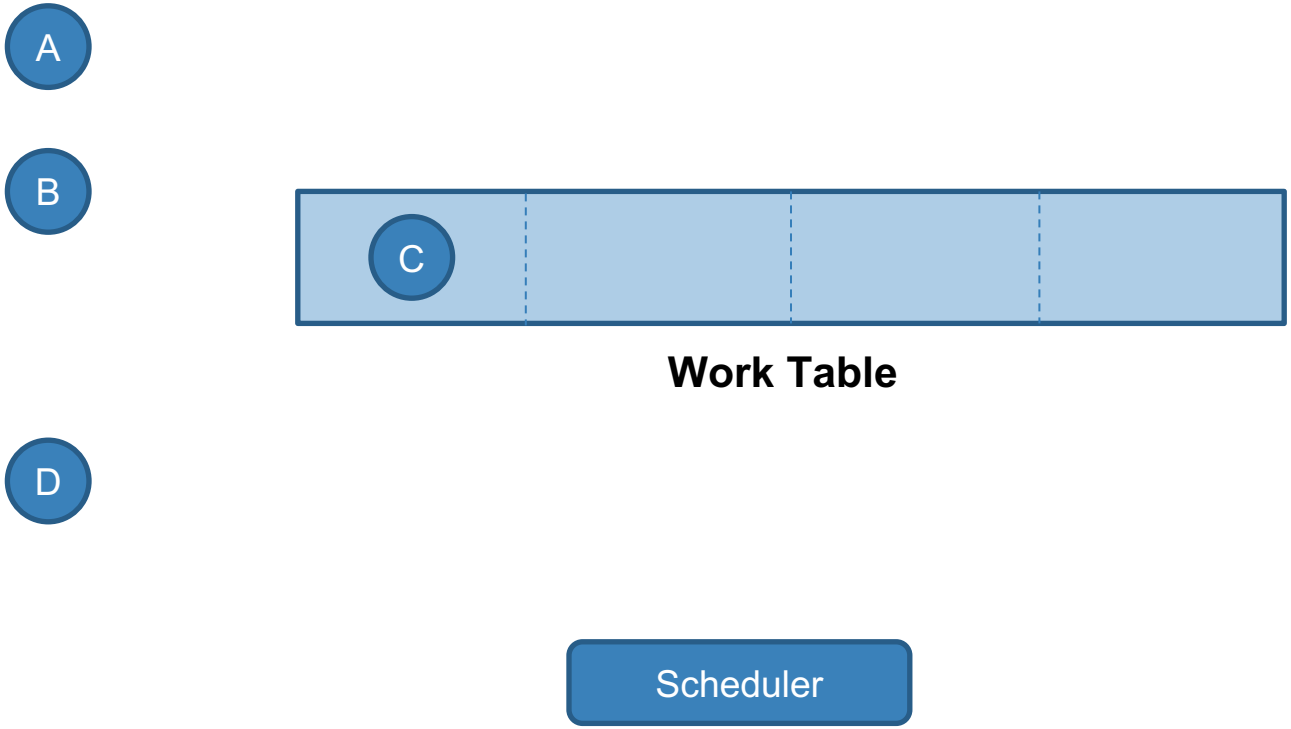
Multithreading



Multithreading



Multithreading



What is the purpose of threading then?

Multithreading

I/O Operation vs Compute Operation

- Compute operation occurs in processor
- I/O operation occurs outside the processor

Examples

- Disk operation
- Network operation
- Http call or web api
- Sleep

Multithreading

```
def build_layer(tray):
    start_mixer()
    pour_colour_mixture(colorama.Back.MAGENTA, tray)
    pour_colour_mixture(colorama.Back.GREEN, tray)
    pour_colour_mixture(colorama.Back.WHITE, tray)


def start_mixer():
    perform_io_operation(1.0)
```

Multithreading

```
def build_layer(tray):
    start_mixer()
    pour_colour_mixture(colorama.Back.MAGENTA, tray)
    pour_colour_mixture(colorama.Back.GREEN, tray)
    pour_colour_mixture(colorama.Back.WHITE, tray)

def perform_io_operation(in_seconds=1):
    time.sleep(in_seconds)
```

Tip 1: Use multithreading for I/O operation

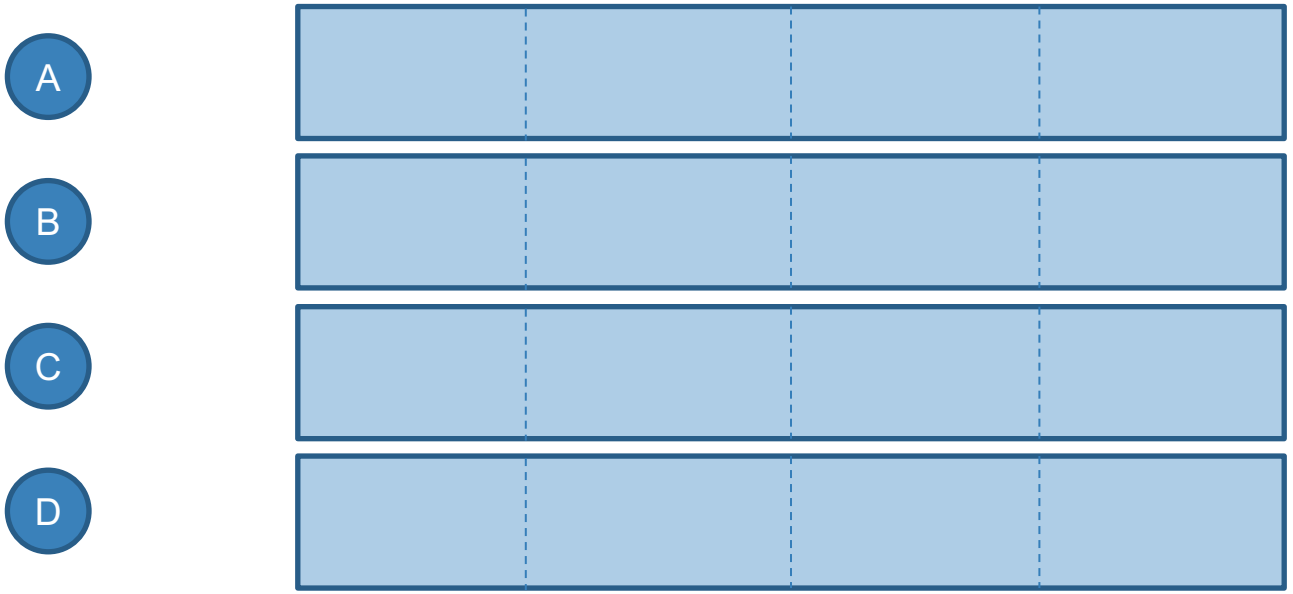
Watch out for synchronization problem due to preemption

What if I want to achieve parallelism?

Multiprocessing

```
from multiprocessing import Process, Queue
def main():
    tray = []
    result = Queue()
    for _ in range(4):
        p = Process(target=build_layer, args=(tray, result))
        p.start()
```

Multiprocessing



Tip 2: Use multiprocessing to achieve parallelism in compute operation

Watch out for the need for serialization

The synchronization problem

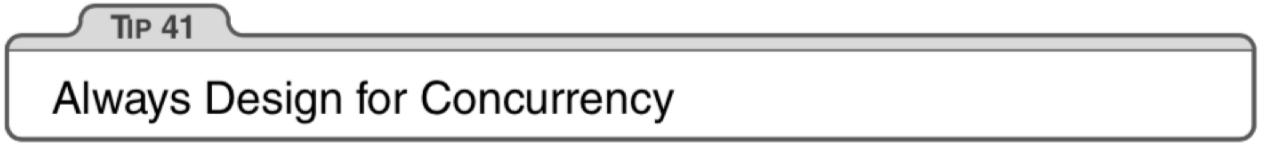
Solution 1: Use lock?

```
lock = Lock()

def build_layer(tray):
    start_mixer()
    with lock:
        pour_colour_mixture(colorama.Back.MAGENTA, tray)
        pour_colour_mixture(colorama.Back.GREEN, tray)
        pour_colour_mixture(colorama.Back.WHITE, tray)
```

Solution 2: Always design for concurrency

The Pragmatic Programmer by Andrew Hunt and David Thomas



- **To Prevent:** Programming by coincidence

No shared state & No side effect

```
def main():
    tray = []
    for _ in range(4):
        t = Thread(target=build_layer, args=(tray,))
        t.start()
```

No shared state

```
def main():
    result = []
    threads = []
    for _ in range(4):
        tray = []
        t = Thread(target=build_layer, args=(tray,))
        t.start()
        threads.append(t)
        result.append(tray)
    for t in threads:
        t.join()

    final_tray = reduce(lambda acc, tray: acc + tray, result, [])
```

No shared state

```
from concurrent.futures import ThreadPoolExecutor

def main_pool():
    with ThreadPoolExecutor() as pool:
        result = pool.map(build_layer, range(4))
    final_tray = reduce(lambda acc, tray: acc + tray, result, [])
```

Idempotent state

Meaning

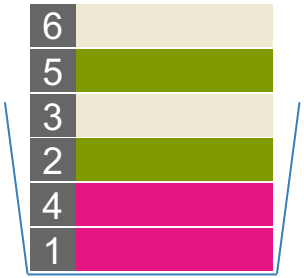
Idempotent state means the meaning of the state does not change even when updates to the state do not happen in the same sequence.

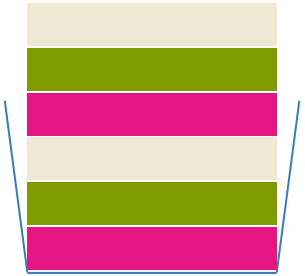
Example

Distributed inventory management system.

Store transaction record instead of updating the remaining items.

Idempotent





Each build_layer appends layer colour with its' position

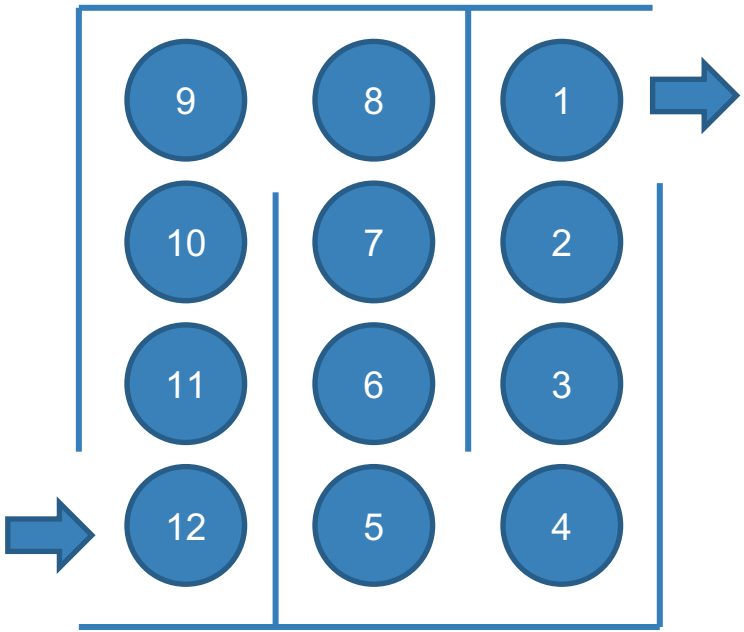
Finally, the layer can be repositioned based on the position

Design for concurrency

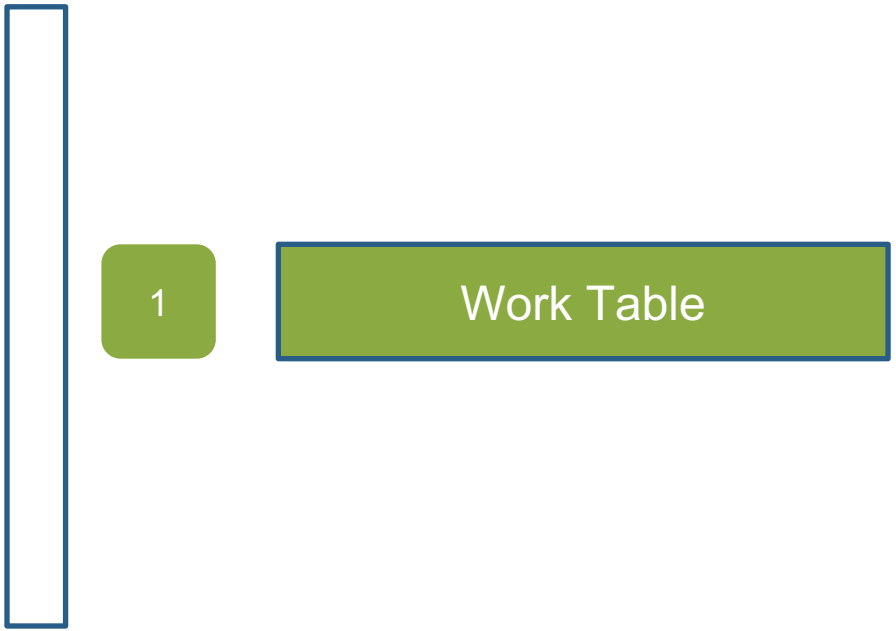
Tip 3: No shared state & no side effect
Tip 4: Use pool.map from concurrent.futures
Tip 5: Have an idempotent state

Once we have designed code for concurrency, we can apply different concurrent model easily.

Restaurant queue



12 Customers in the queue

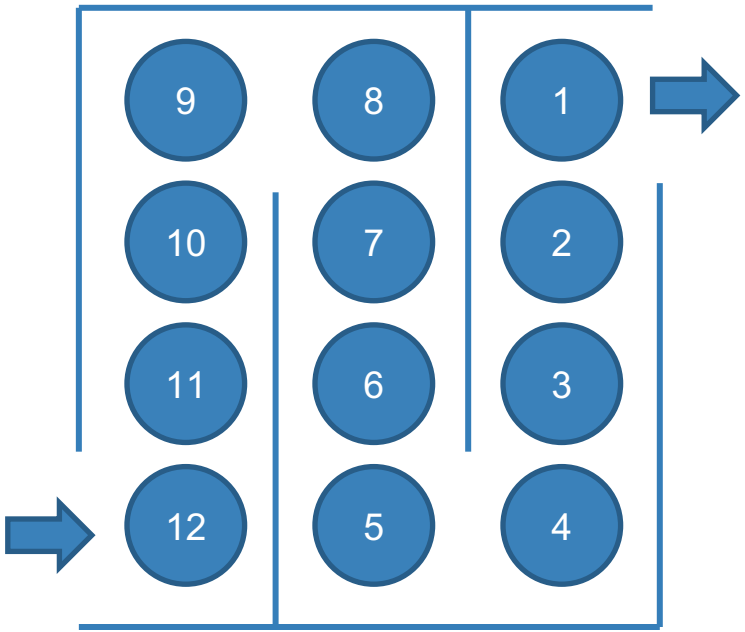


Restaurant queue

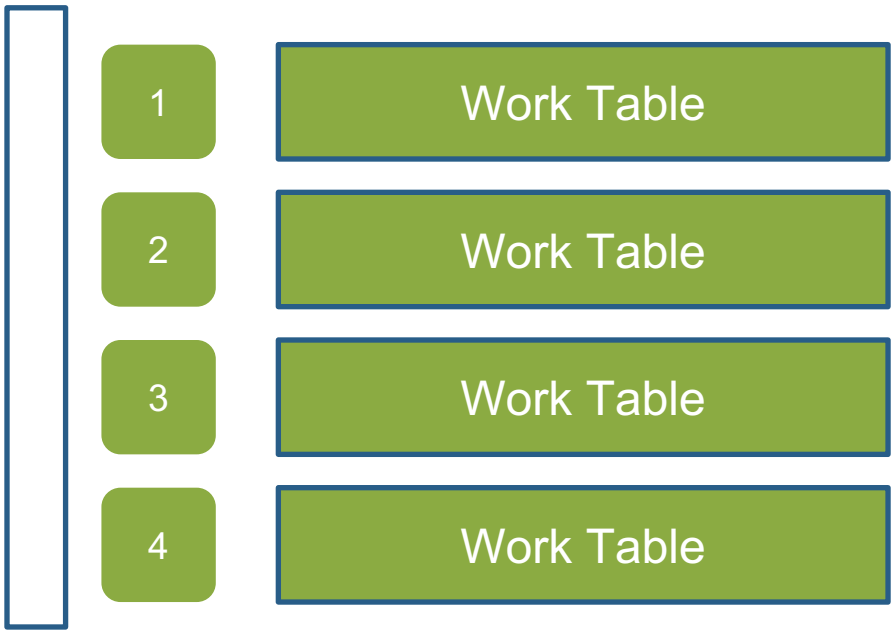
```
def prepare_order(_):
    take_order(_)
    prepare_sandwich()
    prepare_kue_lapis()
    prepare_ice_cream()
    prepare_milk_shake()
```

Try embarrassingly parallel

Embarrassingly Parallel



12 Customers in the queue



Embarrassingly Parallel

```
def main_multi_process(num_of_customer):
    with ProcessPoolExecutor(4) as pool:
        pool.map(prepare_order, range(num_of_customer))


def prepare_order(_):
    take_order(_)
    prepare_sandwich()
    prepare_kue_lapis()
    prepare_ice_cream()
    prepare_milk_shake()
```

Look for I/O operations,

and implement fan-out fan-in model

Fan-out fan-in

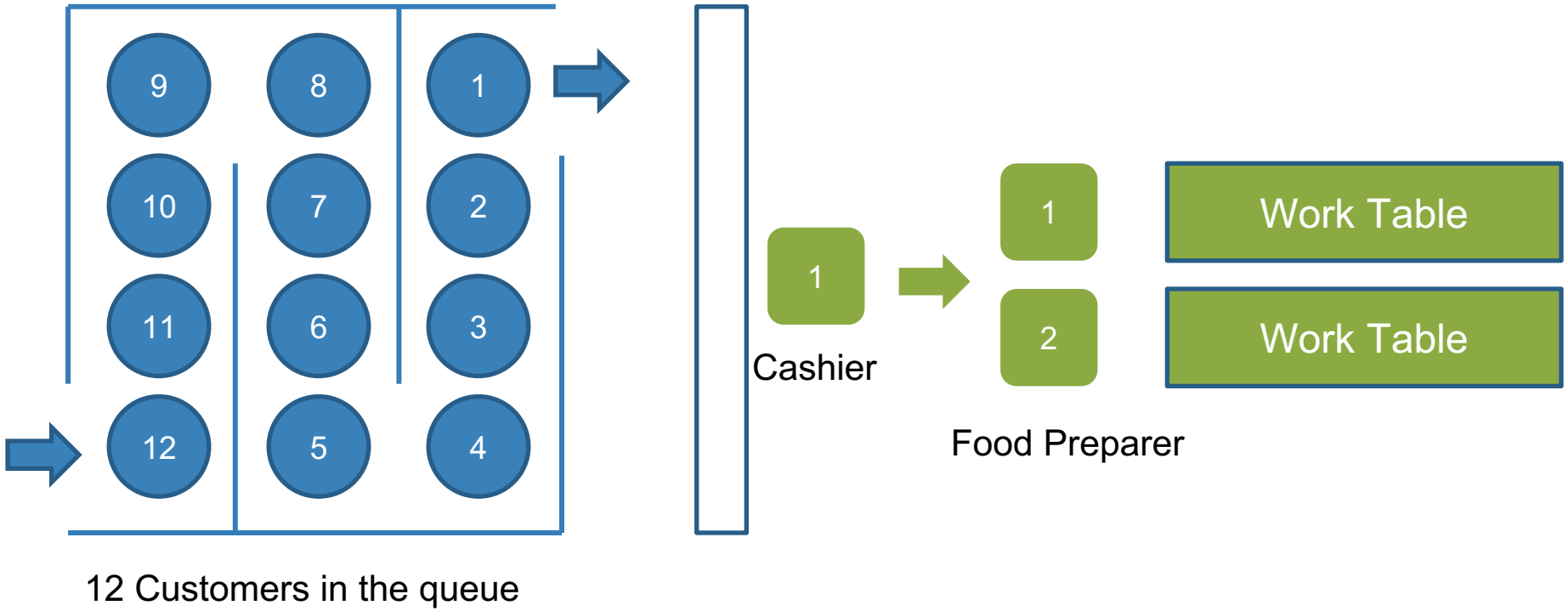
Pycon 2014 by Brett Slatkin on fan-out and fan-in: https://youtu.be/CWmq-jtkemY

Fan-out fan-in (using asyncio)

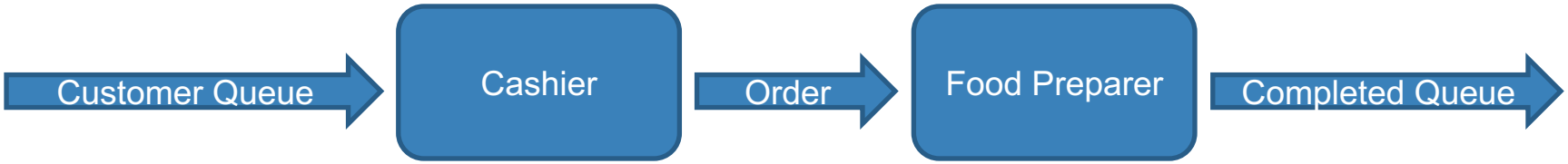
```
async def prepare_order_async(_):
    awaiting_kue_lapis = prepare_kue_lapis()
    awaiting_milk_shake = prepare_milk_shake()
    prepare_sandwich()
    prepare_ice_cream()
    await asyncio.wait([awaiting_kue_lapis, awaiting_milk_shake])
```

Try pipelining

Producer Consumer Queue



Producer Consumer Queue



Producer Consumer Queue

```
def main(num_of_user):
    customer_queue = multiprocessing.JoinableQueue()
    order_queue = multiprocessing.JoinableQueue()
    completed_queue = multiprocessing.Queue()

    cashier = Actor(customer_queue, order_queue, take_order)
    preparer1 = Actor(order_queue, completed_queue, prepare_order)
    preparer2 = Actor(order_queue, completed_queue, prepare_order)

    cashier.start()
    preparer1.start()
    preparer2.start()

    for user_no in range(num_of_user):
        customer_queue.put(user_no + 1)
```

Summary

- Always (and I mean always) design for concurrency
- Understand the basic building block of concurrency in Python
- Make use of the appropriate concurrency model.

THANK YOU

For questions or suggestions:

Wilson Tjhi (@wilsontjhi)
wtjhi@thoughtworks.com

Github: wilsontjhi/pycon-asia-2018

ThoughtWorks®