

# Better Simulator, Better Software Testing


Xiao Shiliang (肖世良)

2017.10, Hangzhou

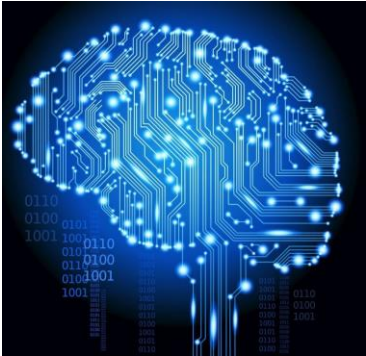
**NOKIA** 上海贝尔




# What are people using Python for?




Scientific Research



Artificial Intelligence



Web Development



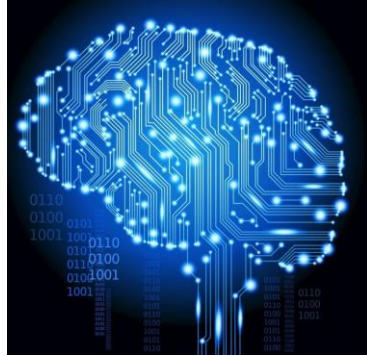
Software Testing

.....

SW testers write python code to make their daily manual & automated activities more efficient and effective.



## Scientific Research



# Artificial Intelligence



# Web Development



# Software Testing

SW testers write python code to make their daily manual & automated activities more efficient and effective.

# Basic Law of Software Testing



## The Google Testing Law (谷歌测试定律)

*“As **multi-level** software testing proceeds (Unit Test -> Module Test -> Integration Test -> System Test), the **cost** of fixing a discovered software bug increases at an **exponential** scale”.*

More Readings: (1) [The Google Testing Law](#)

(2) [Just Say No to More End-to-End Tests](#) (译:对更多的「端到端测试」说不)



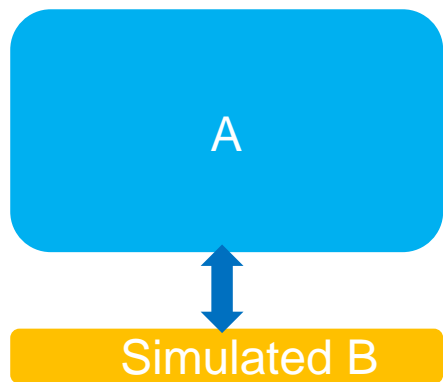
Make SW Testing **Left-shifted**  
(测试左移)



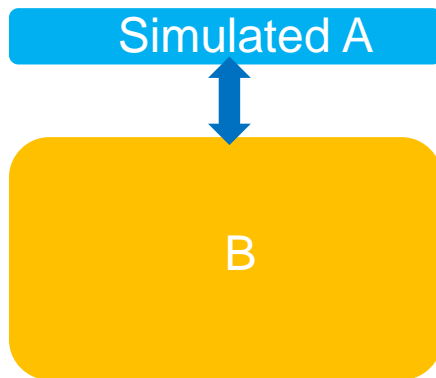
Simulator(or Mocking): **Enabler** of  
SW Testing Left-shift

# Simulator: Enabler of Test Left-shift

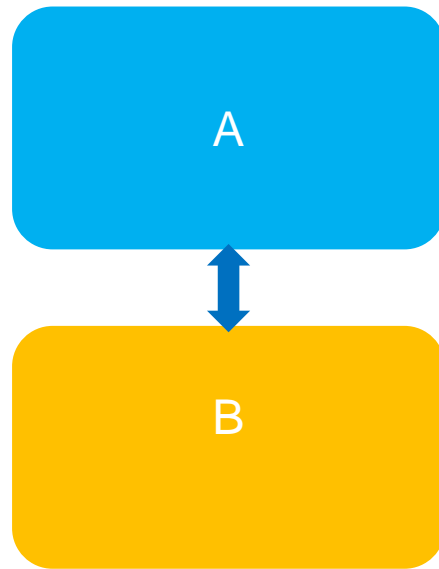
- Left-shift Testing from the perspective of *test kickoff time*: **begin test as early as possible**
- Left-shift Testing from the perspective of *test coverage*: **invest more on low-level tests**



(1)



(2)





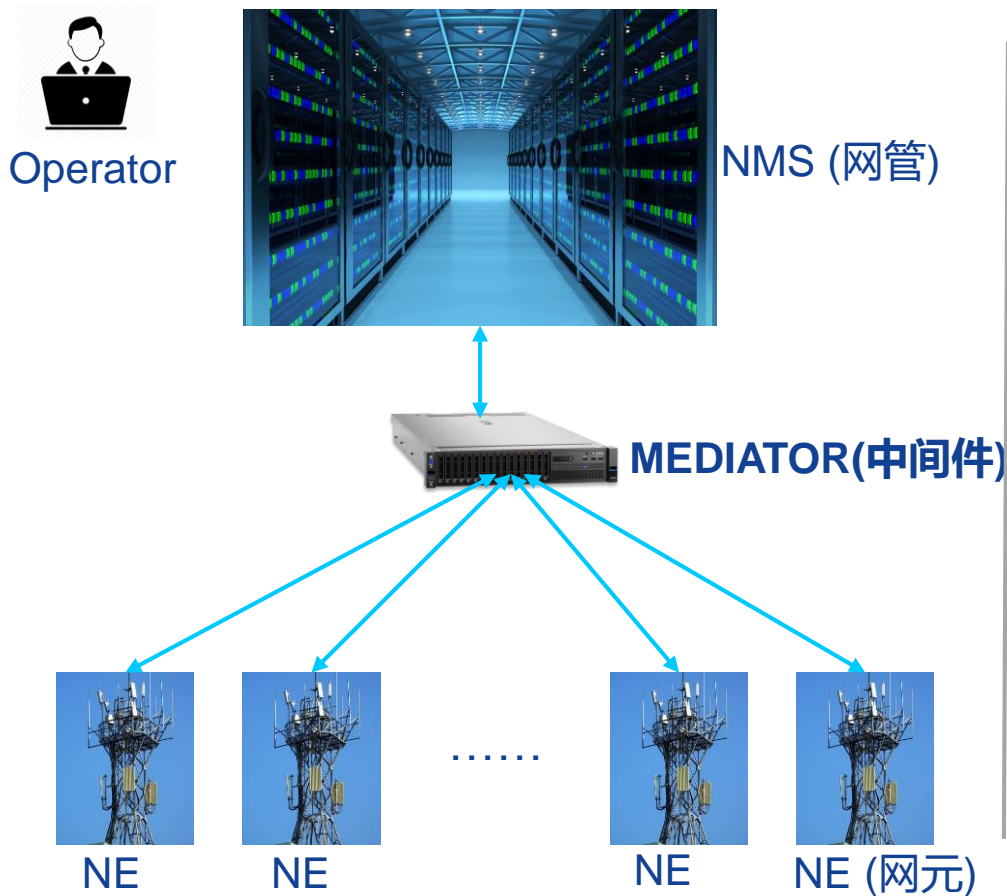
**MUST  
HAVE**

Simulator is even **mandatory** in scenarios where using real objects is almost impossible.

## Good Practices from The NBS (Newbie Simulator) Project



# NBS: Simulating NMS & NE for Network Mediator Testing



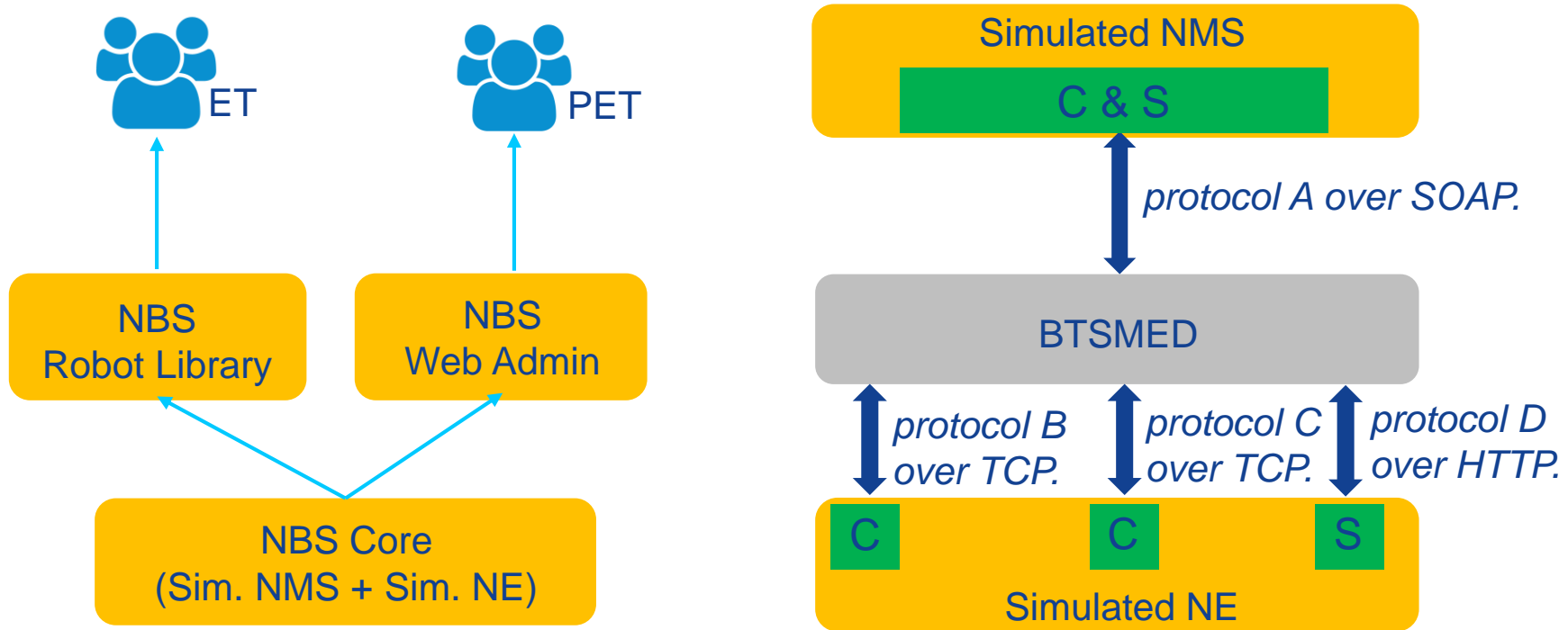
## Mediator Entity Testing (ET)

- NBS simulates NMS & NE
- Provide an Robot Framework compatible library for **automated** ET cases
- Enabler of **344** automated cases

## Mediator Performance Testing (PET)

- NBS simulates NMS & NE (or only NE)
- Needs to simulate as much as **12000** NEs
- Enabler of **133** manual cases

# NBS: Overview



NBS Core = A collection of python modules that perform various **data processing & data transceiving** tasks following several private Client/Server communication protocols.

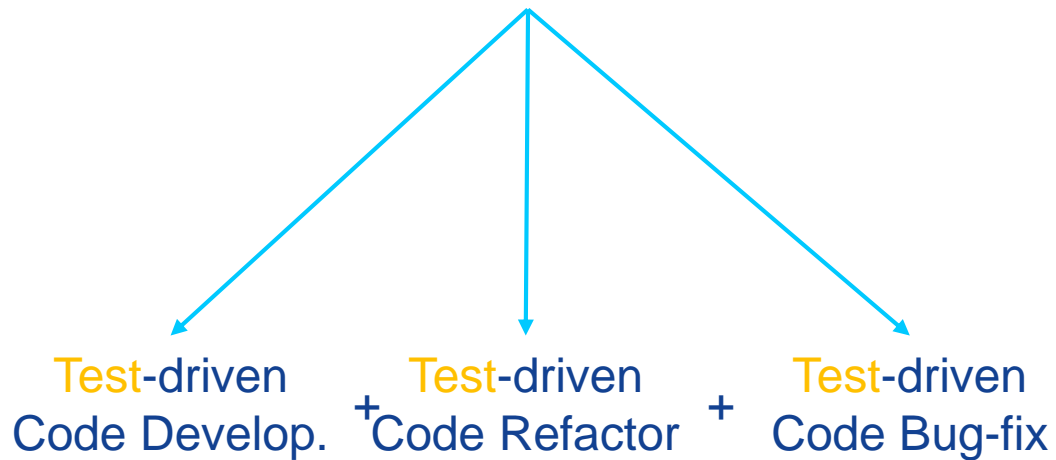

$$E = mc^2$$

Error = more code<sup>2</sup>

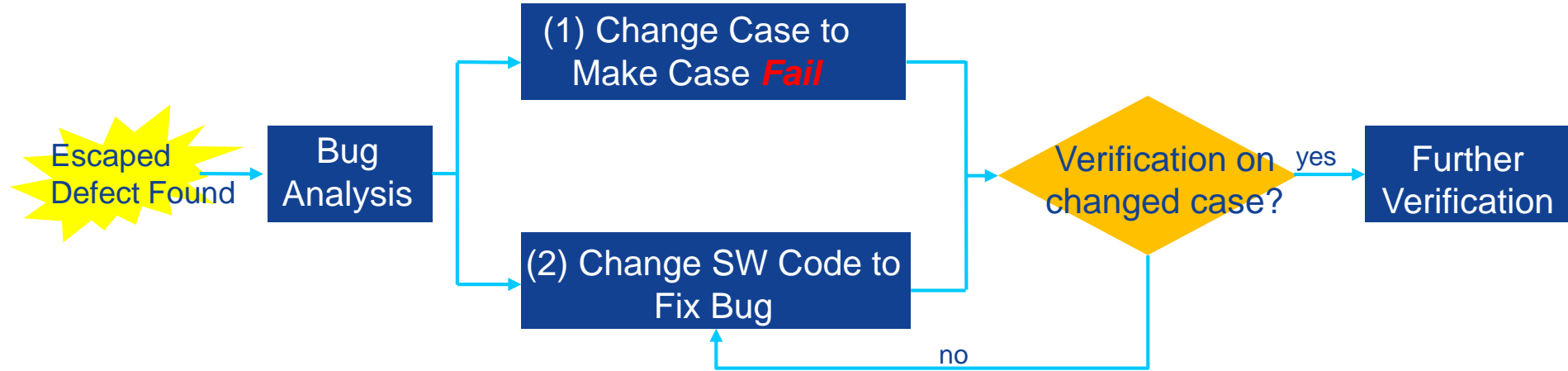
Make **minimal** simulator to implement test cases' requirement



## TDCC: **Test**-driven Code Change



# Test Driven Bug-fixing



- (1) Quality of test case is improved
- (2) Verification of bug-fixing is accelerated

# Mock MEDIATOR in NBS Testing

>>1,000,000



NMS & NE

>50,000



Mediator

~7,000



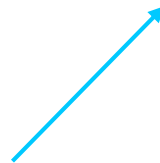
NBS(core)

~300



Mocked Mediator

Foundation of ~260 unit  
test cases that can run  
in ~15 seconds.





Build Continuous Integration(CI)  
infrastructure from the very beginning

**300+**

merged branches

**~2800**

commits

**~2800**

runs of testing

**~15<sub>s</sub>**

verification time

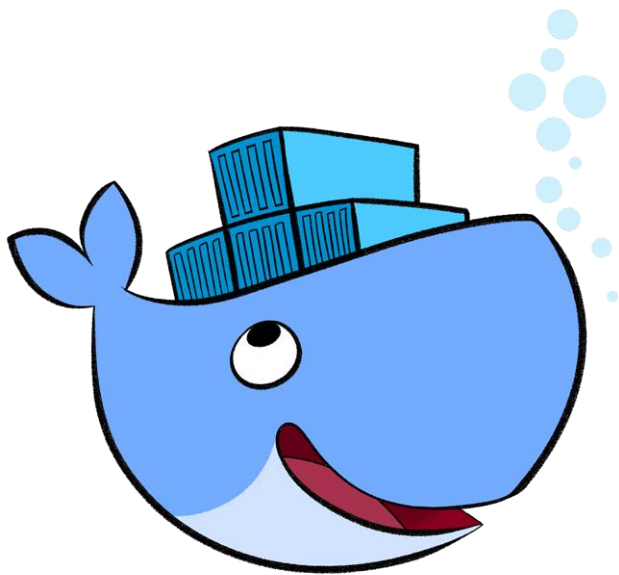
**~4<sub>min</sub>**

releasing time

**300+**

releases





**Docker**-based simulator deployment to reduce environment issues and save maintenance effort



Be very careful when you need to squeeze high  
**concurrency performance** from Python-based simulators

# CPU-bound Task vs. IO-bound Task

CPU密集型

IO密集型

## An example

1. Retrieve file from a remote server

**IO-BOUND**

2. Parse file to get desired information

**CPU-BOUND**

**How to accelerate the above tasks when there're large numbers of files?**

# Solving the Problem with Different Concurrent Approaches

1: Sequential Approach (no concurrency)

2: Multi Threading Approach

3: Coroutine(协程) Approach

4: Multi Processing Approach

```
1 from utils import do_work, TASK_LIST
2
3 for t in TASK_LIST:
4     do_task(t)
```

1

```
1 from utils import do_work, TASK_QUEUE
2 import threading
3
4 NUM_THREAD = 10
5
6 def thread_worker:
7     while True:
8         t = TASK_QUEUE.get_nowait()
9         do_work(t)
10        TASK_QUEUE.task_done()
11
12 for i in range(NUM_THREAD):
13     t = threading.Thread(target=thread_worker)
14     t.daemon = True
15     t.start()
16
17 TASK_QUEUE.join()
```

2

```
1 from gevent import monkey
2 monkey.patch_all()
3
4 from utils import do_work, TASK_LIST
5 from gevent.pool import Pool
6
7 POOL_LIMIT = 10
8
9 Pool(POOL_LIMIT).map(do_worker, TASK_LIST)
```

3

```
1 from utils import do_work, TASK_LIST
2 from multiprocessing import Pool
3
4 NUM_PROCESS = 10
5
6 Pool(NUM_PROCESS).map(do_work, TASK_LIST)
```

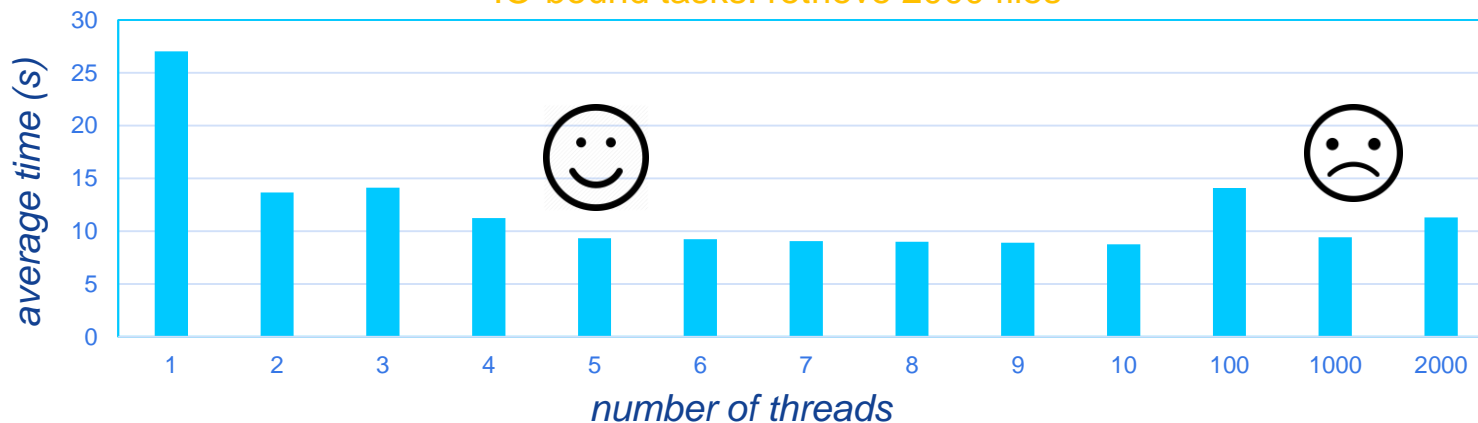
4

# Evaluation of Multi Threading Approach

CPU-bound tasks: parsing 2000 files

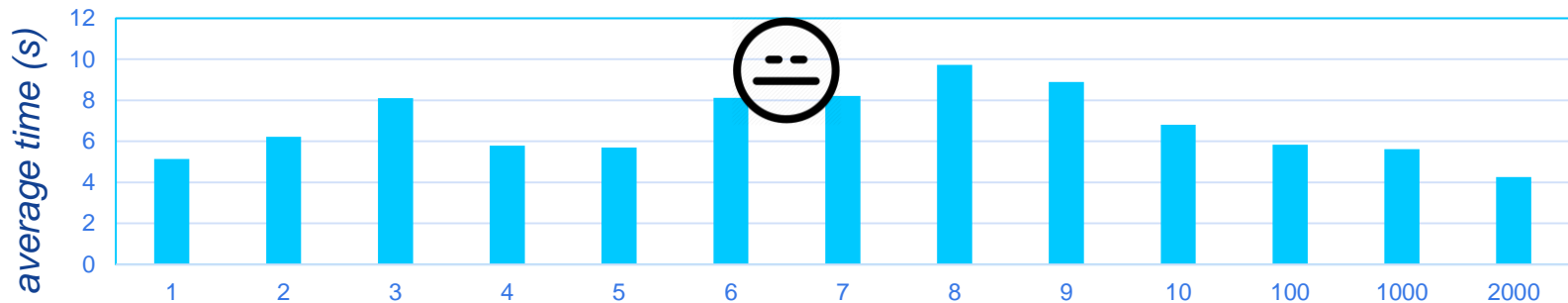


IO-bound tasks: retrieve 2000 files

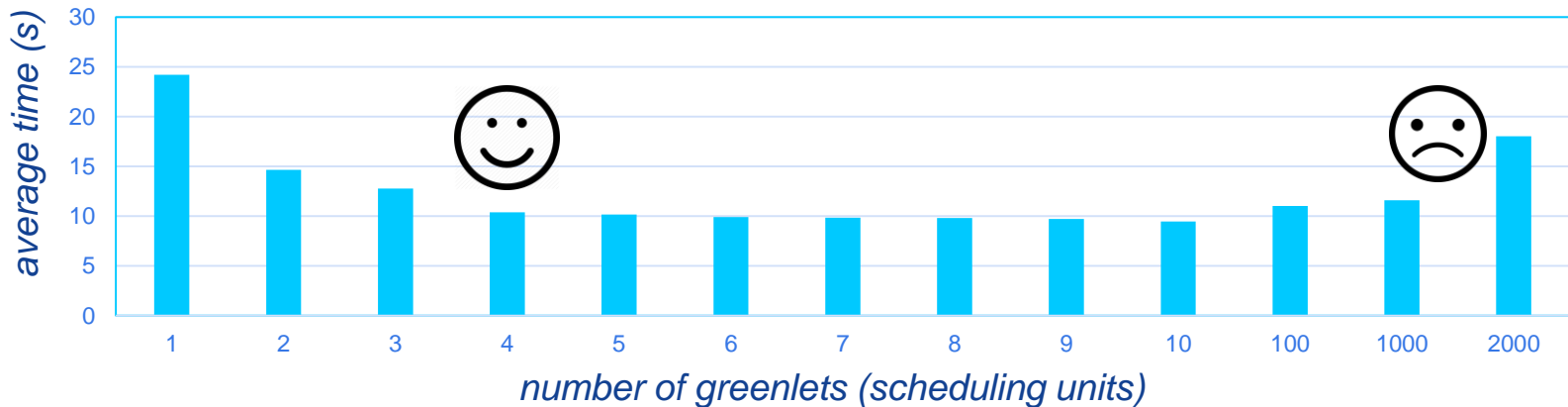


# Evaluation of Coroutine Approach

CPU-bound tasks: parse 2000 files

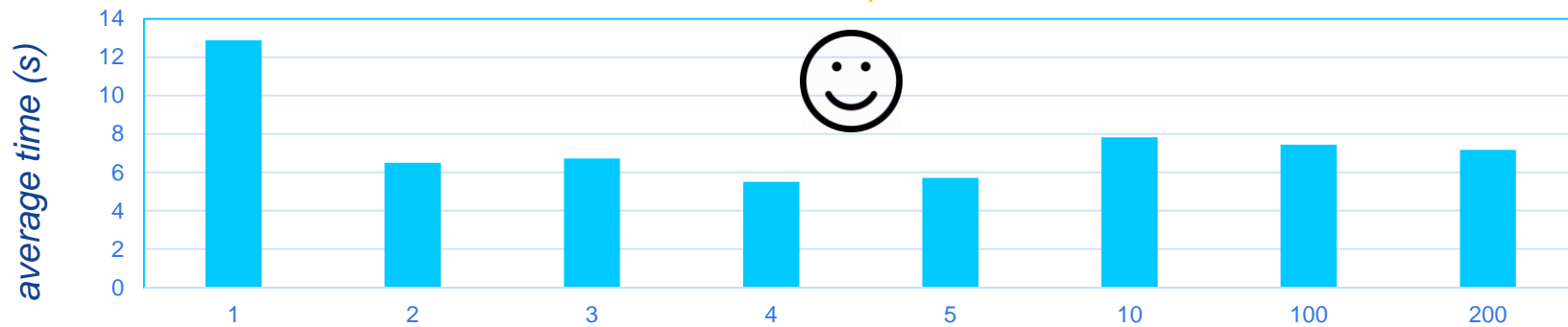


IO-bound tasks: retrieve 2000 files

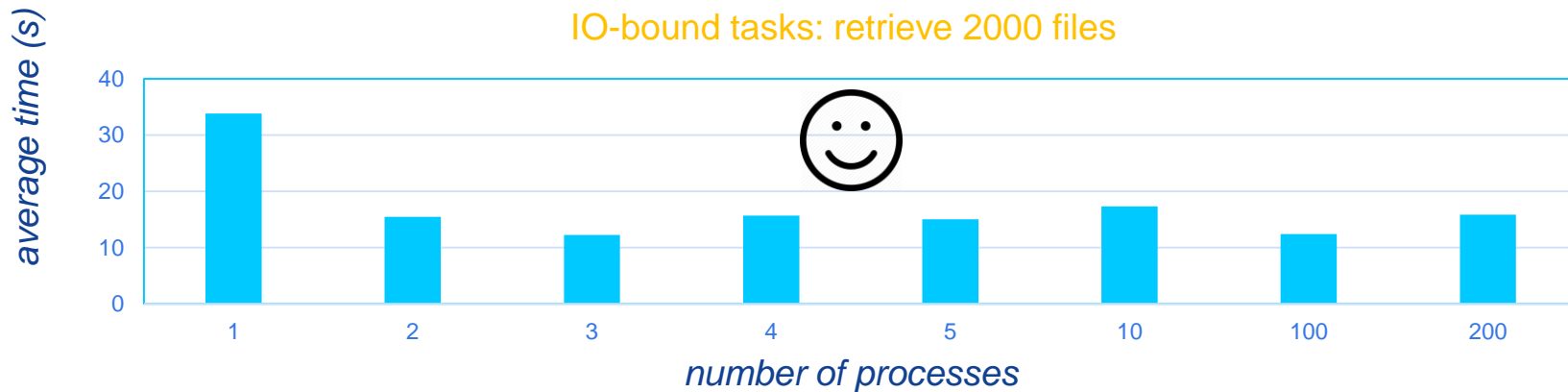


# Evaluation of Multi Processing Approach

CPU-bound tasks: parse 2000 files

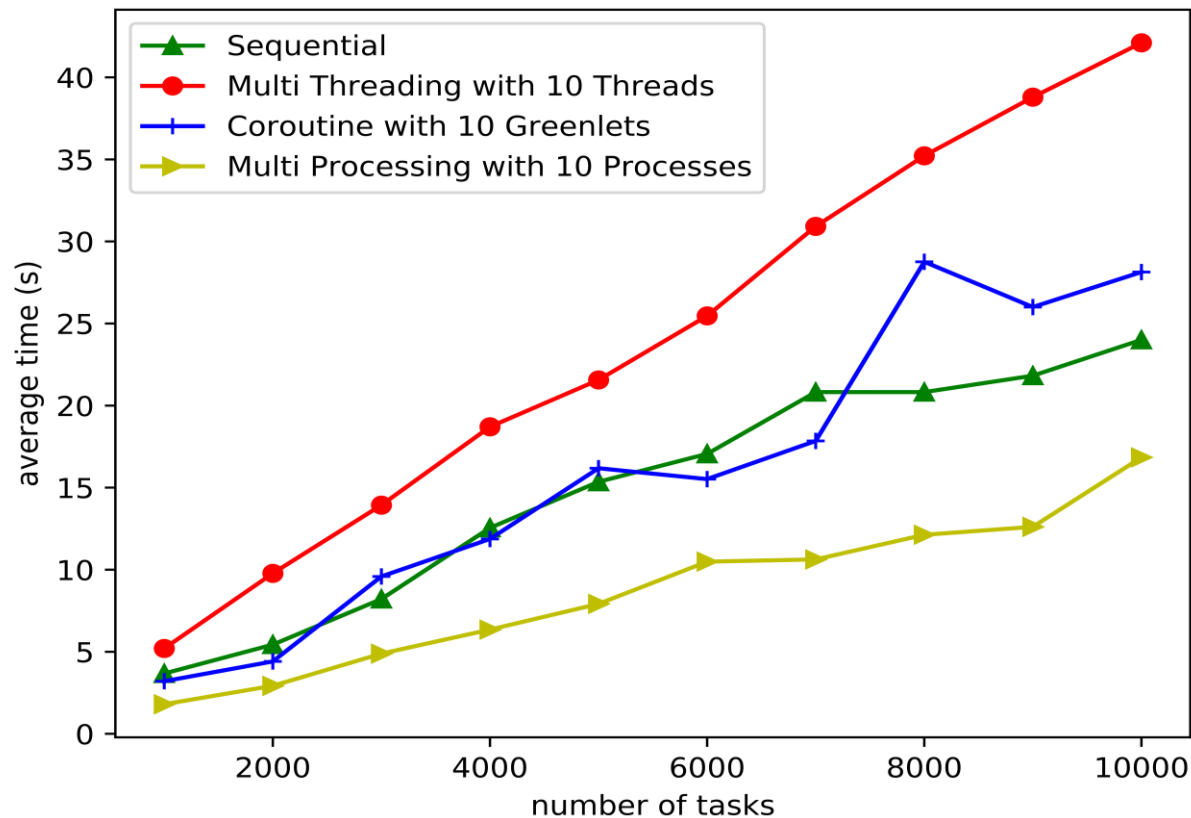


IO-bound tasks: retrieve 2000 files

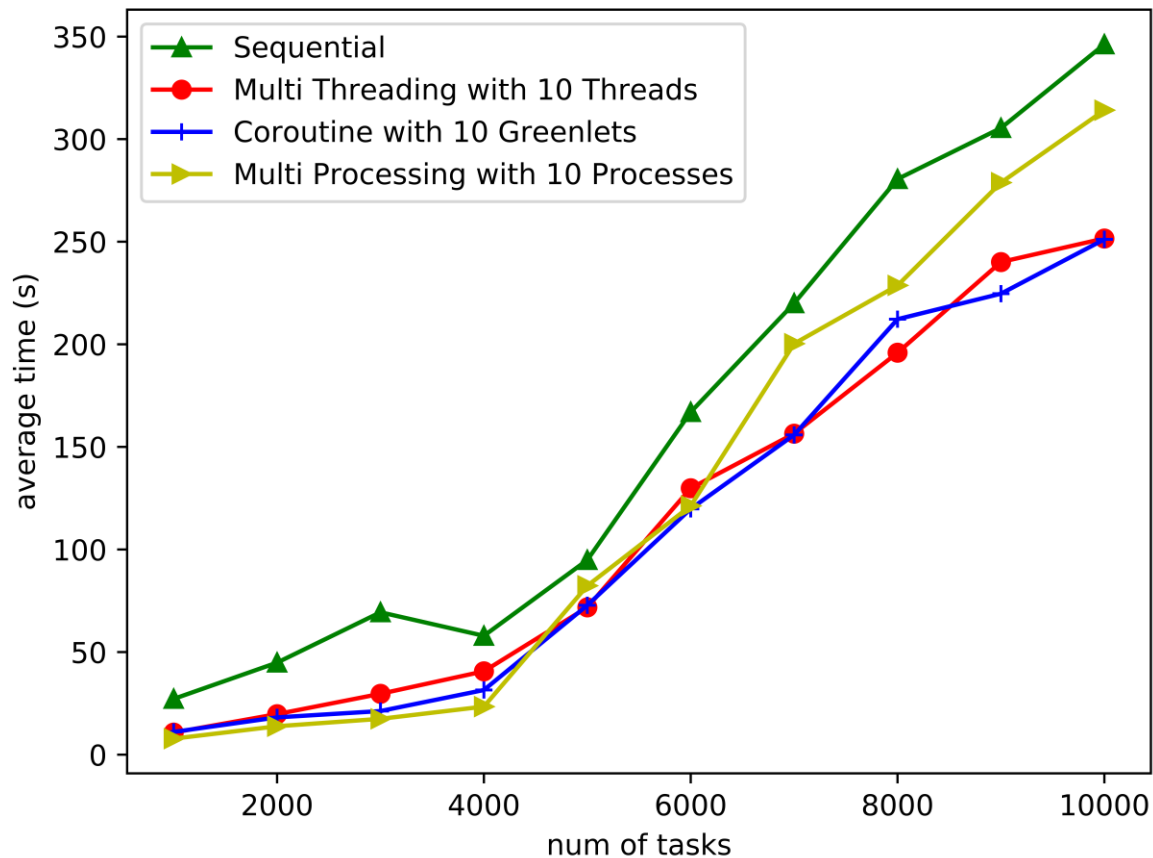




# Comparison of Different Approaches — CPU-bound Task



# Comparison of Different Approaches — IO-bound Task



## Conclusion on Python's Concurrency?

I prefer **NOT** to conclude when it comes to concurrency of Python.

# Conclusion on this Talk?

Here are the take-away messages:

1. Google Testing Law & Software Test **Left-shifting** & Simulator
2. Make minimal simulator to implement test cases' requirement
3. TDCC: **Test Driven Code Change** (test-driven development + test-driven refactor & test-driven bug-fixing)
4. Build Continuous Integration(CI) **infrastructure** from the very beginning
5. Docker-based deployment to reduce environ. issues & save maintain. efforts
6. Be very careful when you need to squeeze high **concurrency performance** from Python-based simulators



[learn more](#)

# Q & PA

Questions & Possible Answers.