Low-Level Python

May 9, 2023

Final Project

~~Final Project Proposal~~	~~End of Week 5~~
Meetings with Parth/Tara	Weeks 7/8
Presentations	Week 10 (May 31)

Announcements

1. a2: augment.py (final artifact) due in one week — feedback on augmentation sketch coming soon

   general themes: wi-fi on micro:bit, occasionally a bit too ambitious, think about connections with the final project, **super cool!**

2. Bring your augmented artifact to class next week for show and tell!
3. Transition to working on the final project — possibly an extension of assignment 2
4. On the horizon: final project pizza party 🥹

Learning Objectives

After today's class, you'll be able to...

1. Write Python code to take advantage of the operating system's thread scheduling paradigm
2. Explain what the global interpreter lock (GIL) is, describe why it was created, and evaluate whether it's needed
3. Describe how Python's functionality can be extended with C code and evaluate places where that might be useful

Agenda

1. Multithreading
2. Global interpreter lock
   - Reference counting
   - I/O bound vs. CPU bound threads
3. Extending Python with C or C++

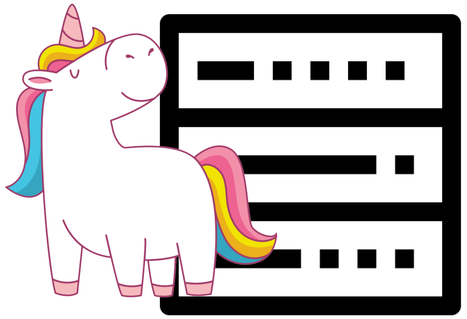
Agenda

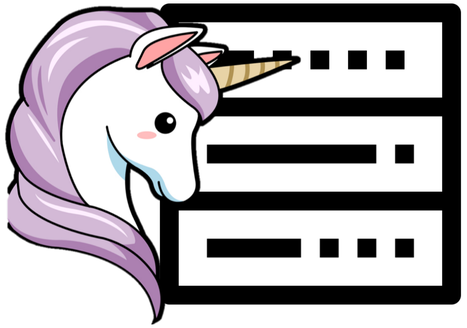
1. Multithreading
2. Global interpreter lock
   - Reference counting
   - I/O bound vs. CPU bound threads
3. Extending Python with C or C++

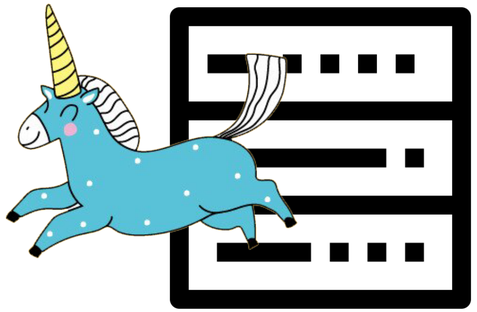
getting lower level

Multithreading

parth's computer

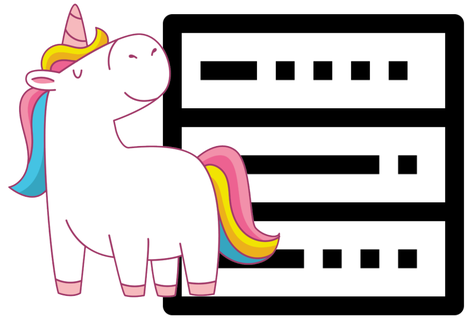


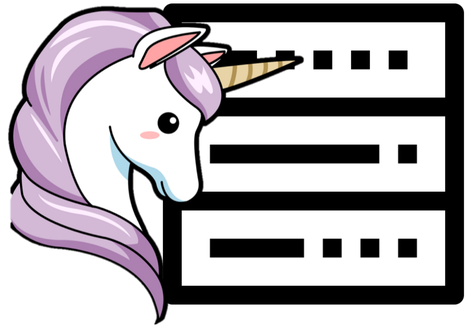


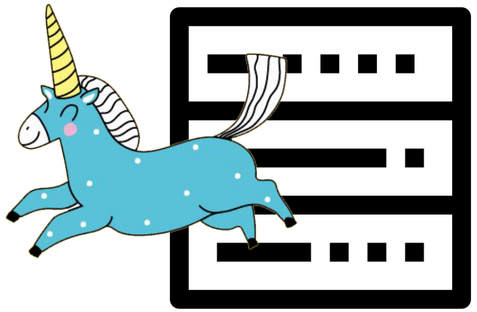


single-threaded downloading

parth's computer

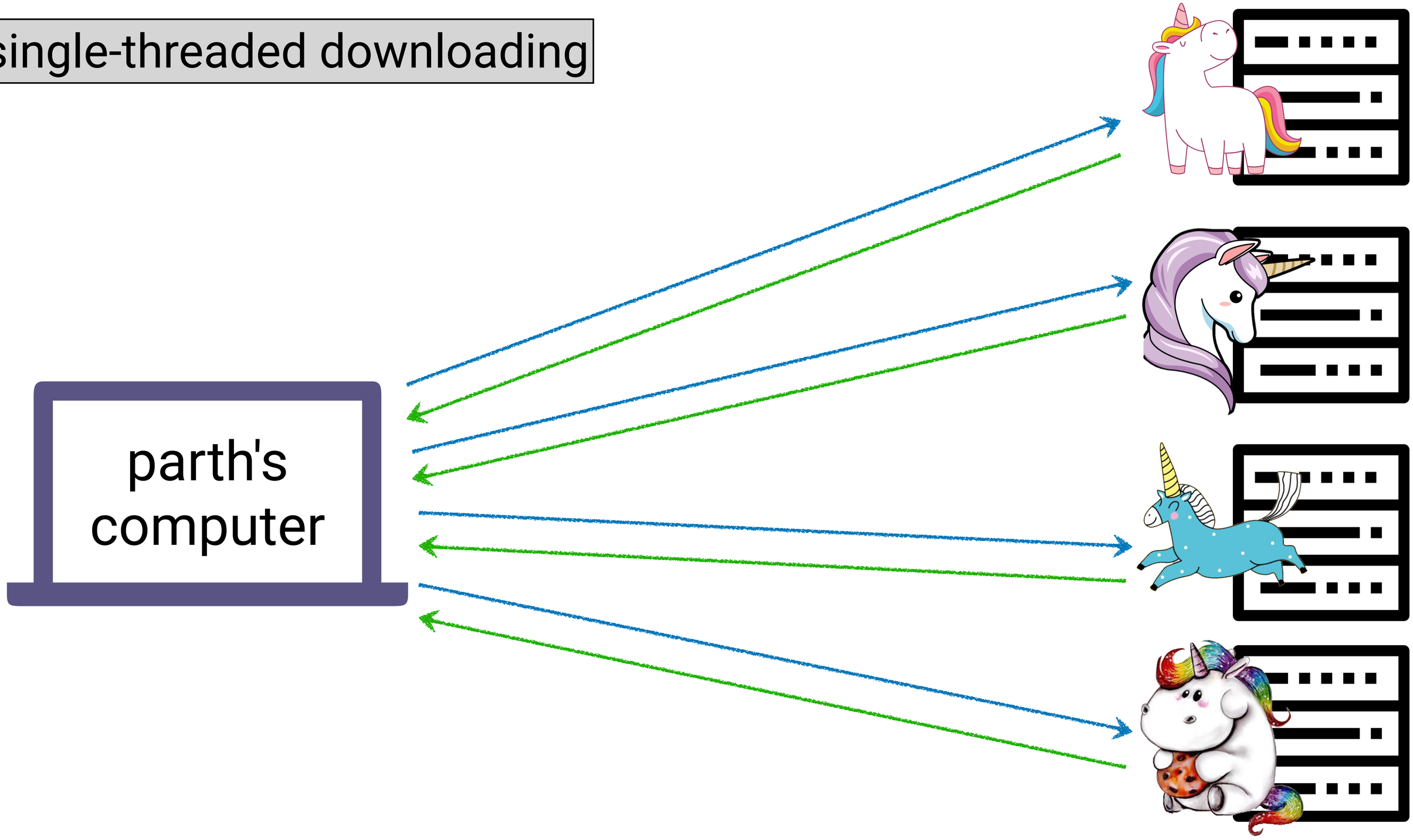






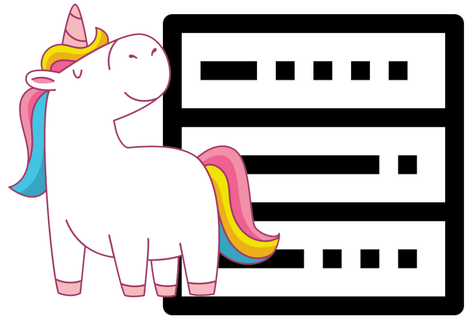
single-threaded downloading

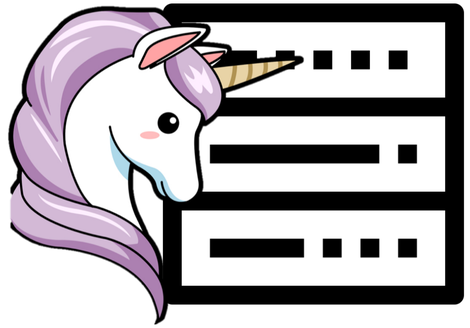
parth's computer



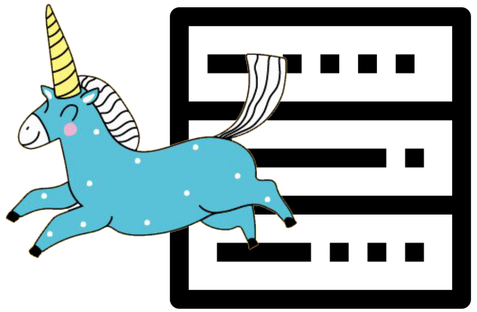
Key idea: While the computer is waiting to receive data, it can do other things

multi-threaded downloading



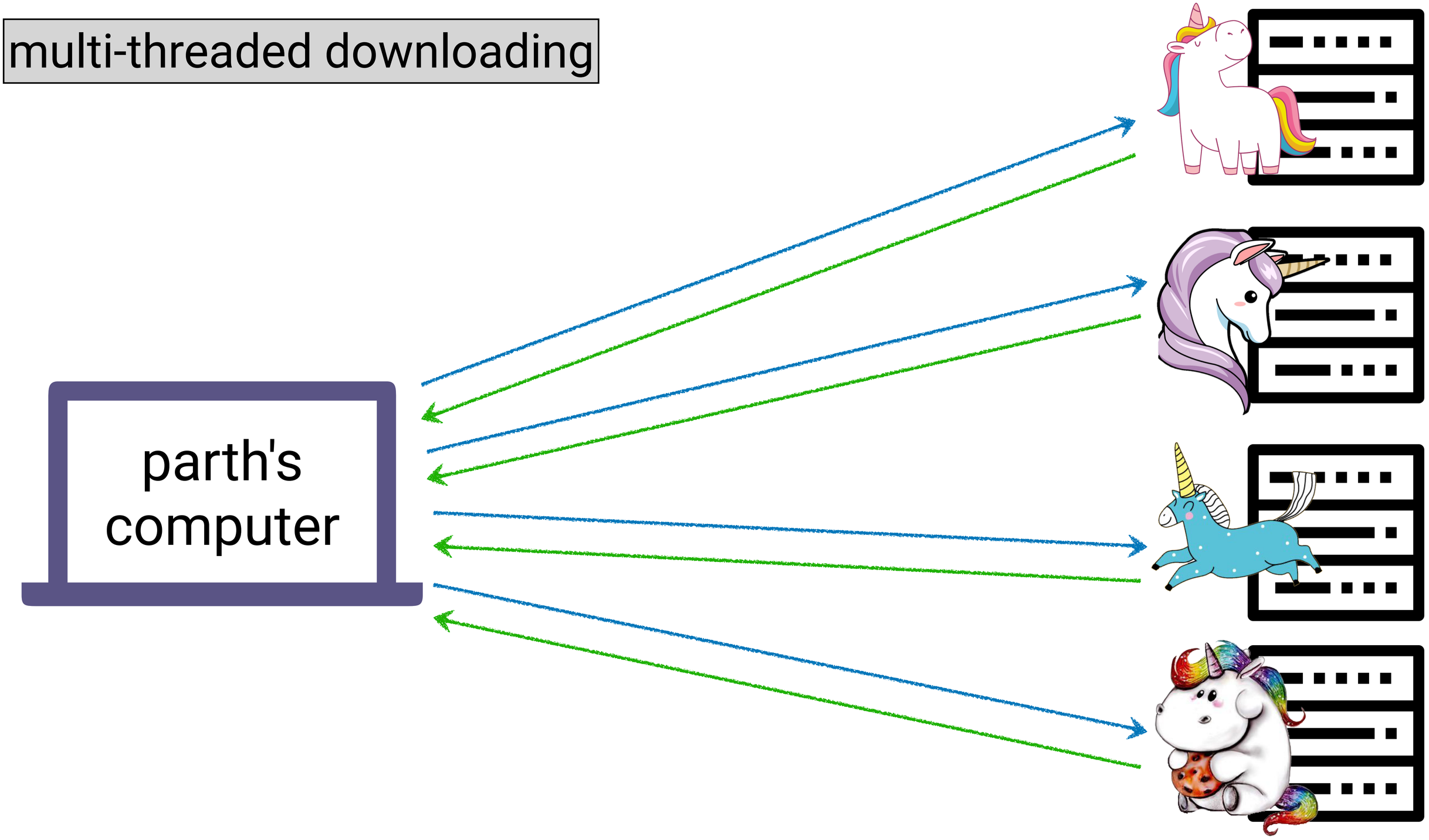


parth's computer



multi-threaded downloading

parth's computer



```
requests.get(url)
```

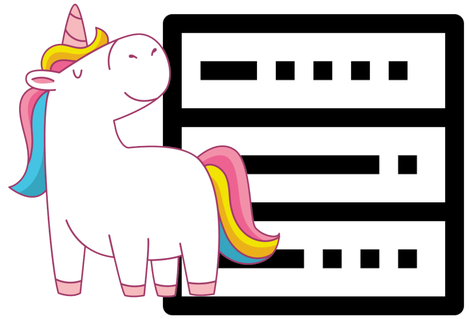
...is really...

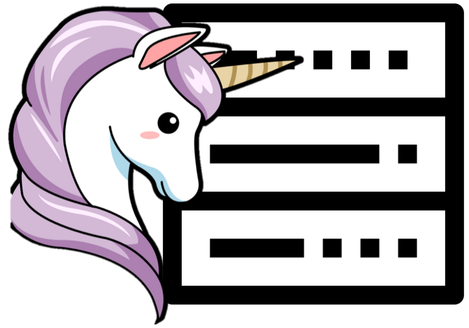
```
send a request to url
wait for the response
return the data
```

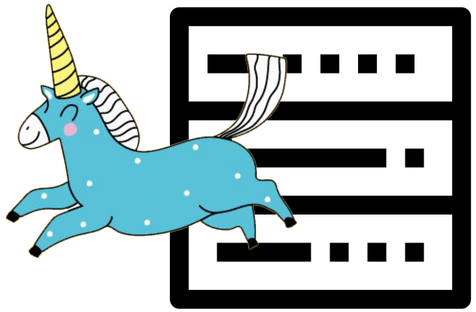
```
requests.get(url1)
requests.get(url2)
requests.get(url3)
requests.get(url4)
```

...is really...

```
send a request to url1
wait for the response
return the data
send a request to url2
wait for the response
return the data
send a request to url3
wait for the response
return the data
send a request to url4
wait for the response
return the data
```







send a request to url1

wait for the response

return the data

send a request to url2

wait for the response

return the data

send a request to url3

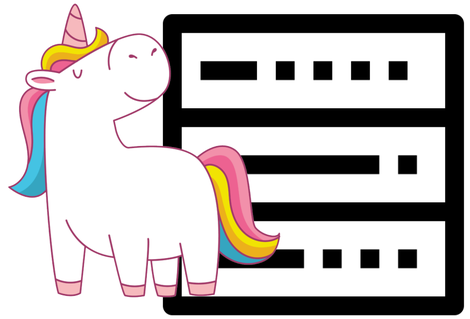
wait for the response

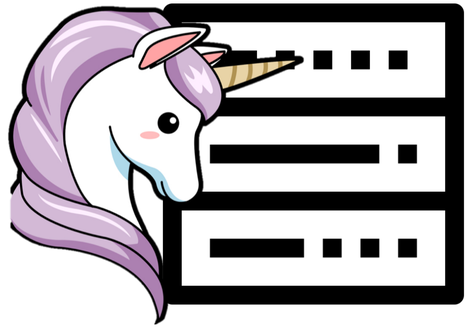
return the data

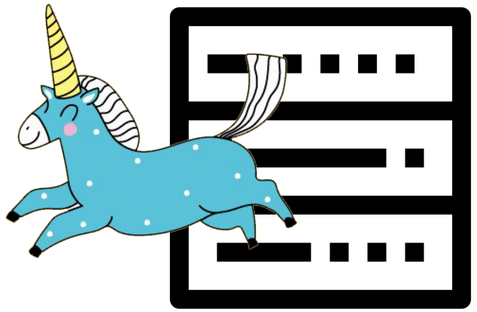
send a request to url4

wait for the response

return the data

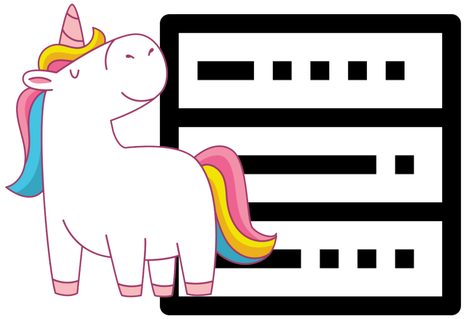


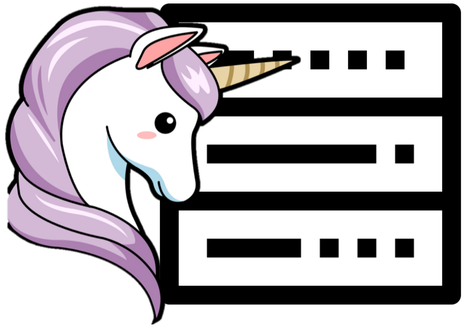


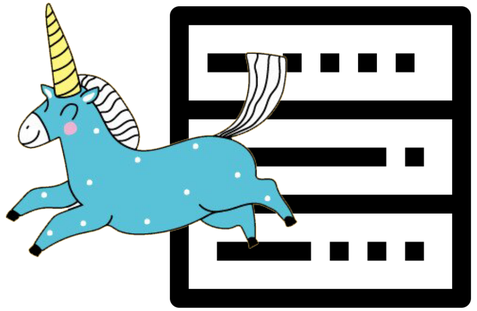


starts at 0, takes 0.5 seconds

```
send a request to url1
wait for the response
return the data
send a request to url2
wait for the response
return the data
send a request to url3
wait for the response
return the data
send a request to url4
wait for the response
return the data
```



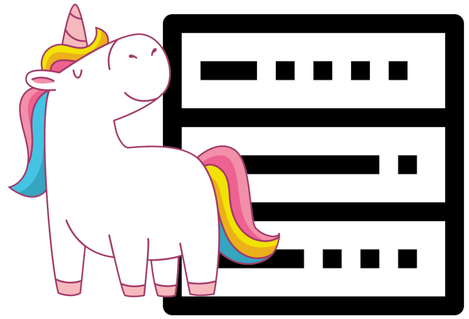


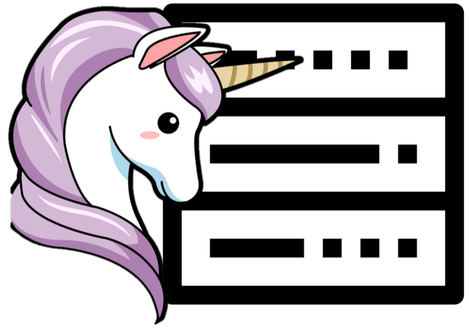


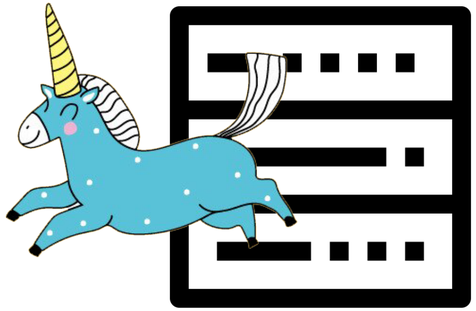
starts at 0, takes 0.5 seconds

starts at 0.5, takes 10 seconds

```
send a request to url1
wait for the response
return the data
send a request to url2
wait for the response
return the data
send a request to url3
wait for the response
return the data
send a request to url4
wait for the response
return the data
```





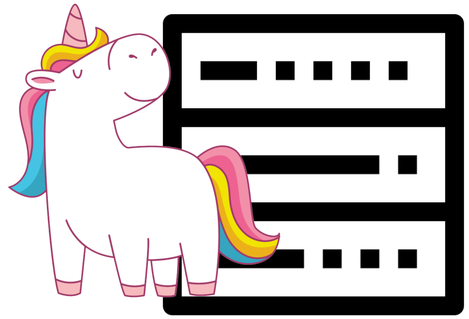


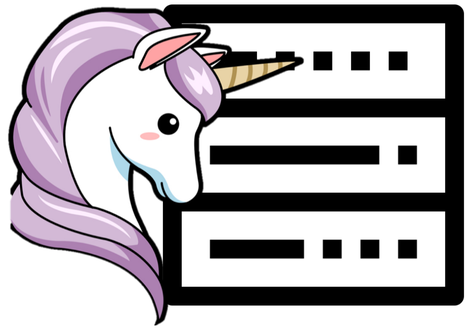
starts at 0, takes 0.5 seconds

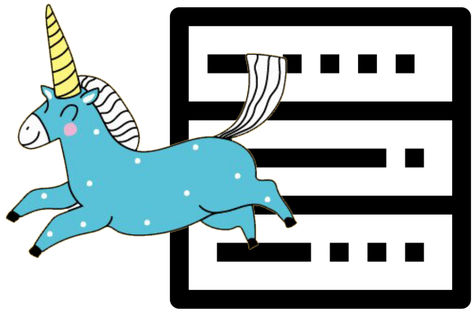
starts at 0.5, takes 10 seconds

starts at 10.5, takes 0 seconds

```
send a request to url1
wait for the response
return the data
send a request to url2
wait for the response
return the data
send a request to url3
wait for the response
return the data
send a request to url4
wait for the response
return the data
```







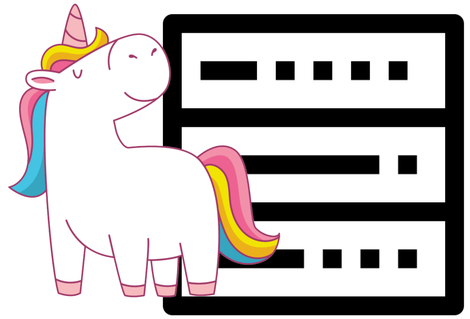
starts at 0, takes 0.5 seconds

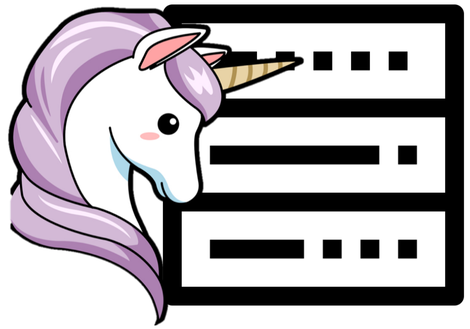
starts at 0.5, takes 10 seconds

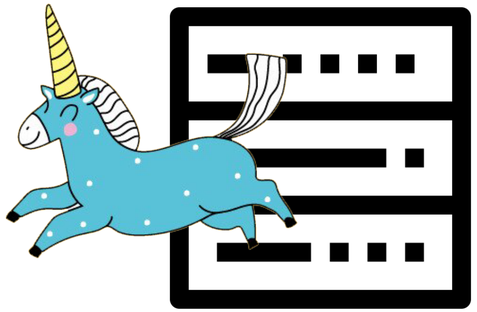
starts at 10.5, takes 0 seconds

starts at 10.5, takes 0.5 seconds

```
send a request to url1
wait for the response
return the data
send a request to url2
wait for the response
return the data
send a request to url3
wait for the response
return the data
send a request to url4
wait for the response
return the data
```







starts at 0, takes 0.5 seconds

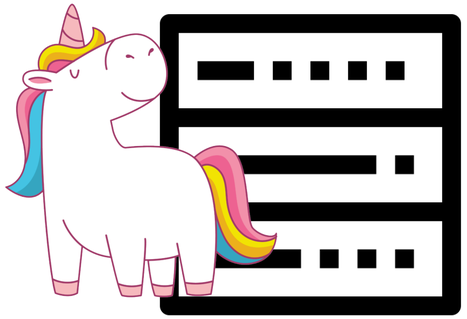
starts at 0.5, takes 10 seconds

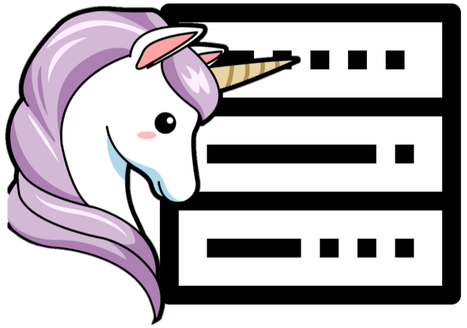
starts at 10.5, takes 0 seconds

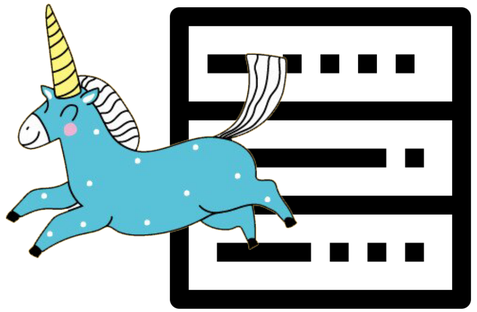
starts at 10.5, takes 0.5 seconds

starts at 11, takes 10 seconds

```
send a request to url1
wait for the response
return the data
send a request to url2
wait for the response
return the data
send a request to url3
wait for the response
return the data
send a request to url4
wait for the response
return the data
```







starts at 0, takes 0.5 seconds

starts at 0.5, takes 10 seconds

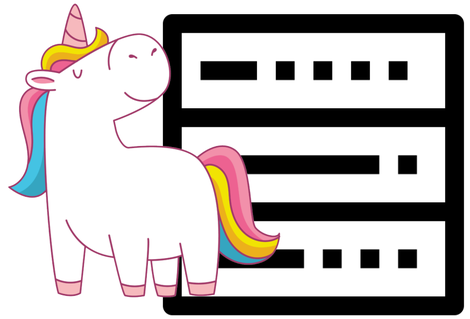
starts at 10.5, takes 0 seconds

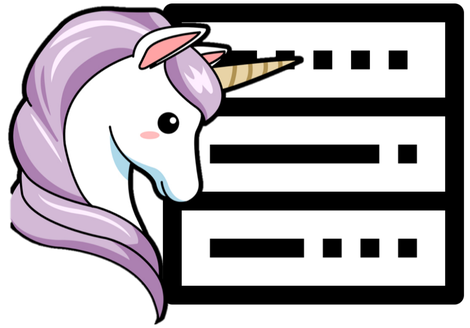
starts at 10.5, takes 0.5 seconds

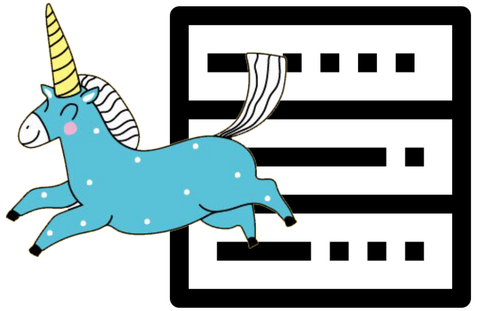
starts at 11, takes 10 seconds

starts at 21, takes 0 seconds

```
send a request to url1
wait for the response
return the data
send a request to url2
wait for the response
return the data
send a request to url3
wait for the response
return the data
send a request to url4
wait for the response
return the data
```







starts at 0, takes 0.5 seconds

starts at 0.5, takes 10 seconds

starts at 10.5, takes 0 seconds

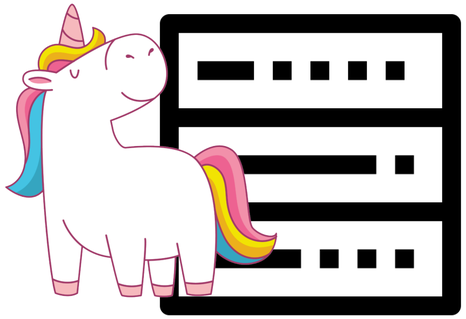
starts at 10.5, takes 0.5 seconds

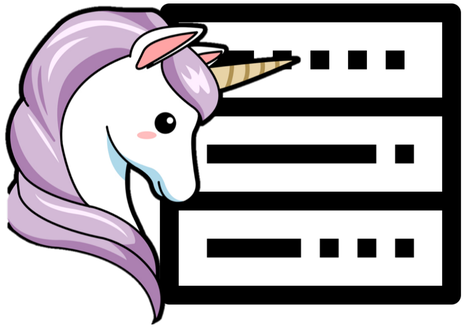
starts at 11, takes 10 seconds

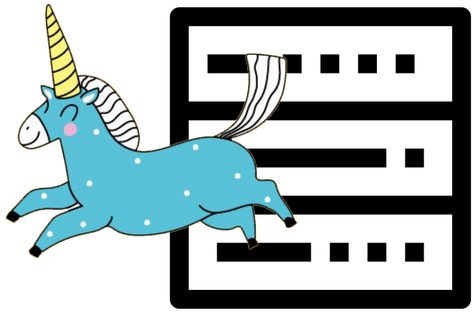
starts at 21, takes 0 seconds

starts at 21, takes 0.5 seconds

```
send a request to url1
wait for the response
return the data
send a request to url2
wait for the response
return the data
send a request to url3
wait for the response
return the data
send a request to url4
wait for the response
return the data
```







starts at 0, takes 0.5 seconds

starts at 0.5, takes 10 seconds

starts at 10.5, takes 0 seconds

starts at 10.5, takes 0.5 seconds

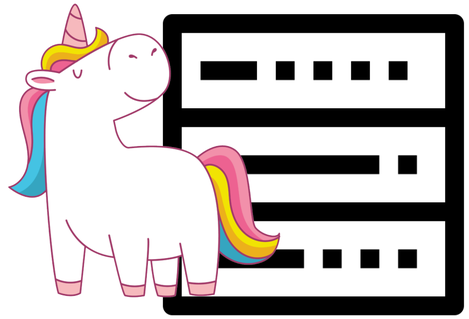
starts at 11, takes 10 seconds

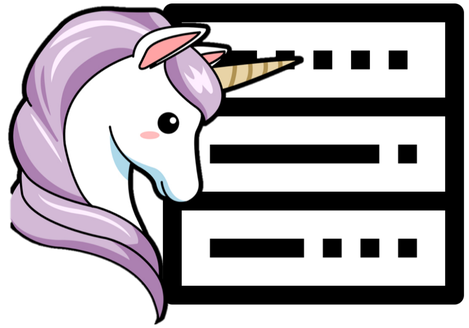
starts at 21, takes 0 seconds

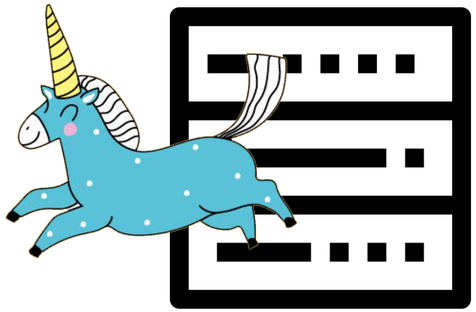
starts at 21, takes 0.5 seconds

starts at 21.5, takes 10 seconds

```
send a request to url1
wait for the response
return the data
send a request to url2
wait for the response
return the data
send a request to url3
wait for the response
return the data
send a request to url4
wait for the response
return the data
```







starts at 0, takes 0.5 seconds

starts at 0.5, takes 10 seconds

starts at 10.5, takes 0 seconds

starts at 10.5, takes 0.5 seconds

starts at 11, takes 10 seconds

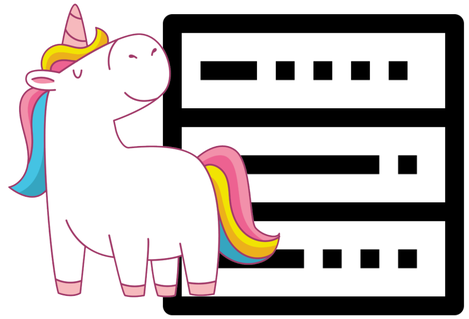
starts at 21, takes 0 seconds

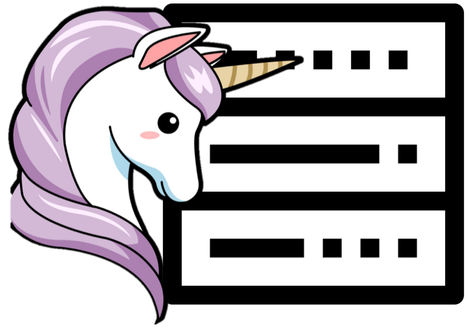
starts at 21, takes 0.5 seconds

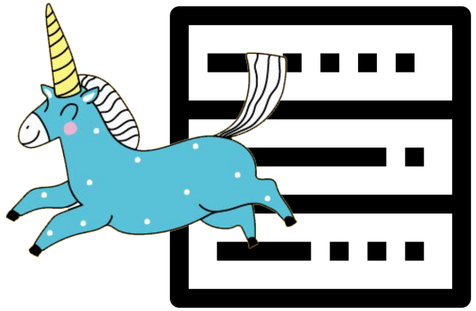
starts at 21.5, takes 10 seconds

starts at 31.5, takes 0 seconds

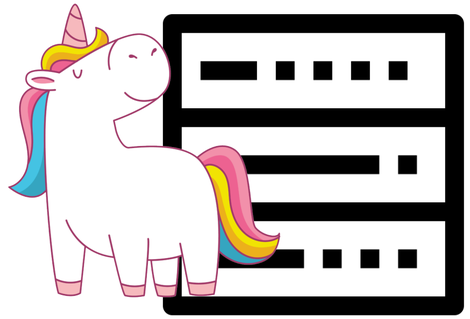
```
send a request to url1
wait for the response
return the data
send a request to url2
wait for the response
return the data
send a request to url3
wait for the response
return the data
send a request to url4
wait for the response
return the data
```

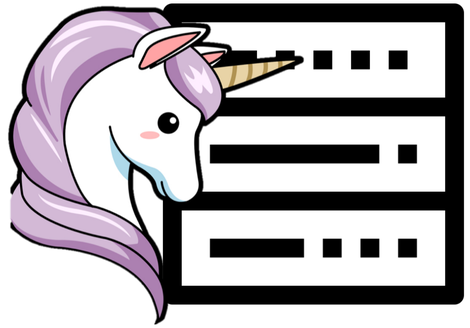


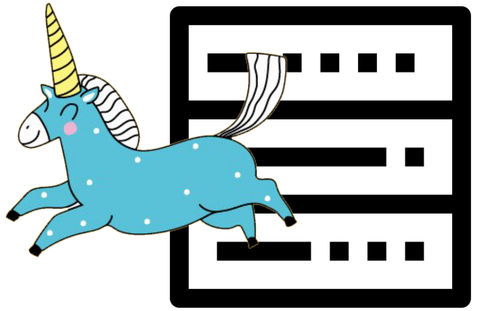




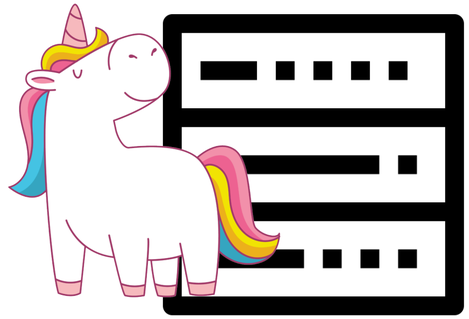
starts at 0, takes 0.5 seconds	send a request to url1
starts at 0.5, takes 10 seconds	wait for the response
starts at 10.5, takes 0 seconds	return the data
starts at 10.5, takes 0.5 seconds	send a request to url2
starts at 11, takes 10 seconds	wait for the response
starts at 21, takes 0 seconds	return the data
starts at 21, takes 0.5 seconds	send a request to url3
starts at 21.5, takes 10 seconds	wait for the response
starts at 31.5, takes 0 seconds	return the data
starts at 31.5, takes 0.5 seconds	send a request to url4
	wait for the response
	return the data

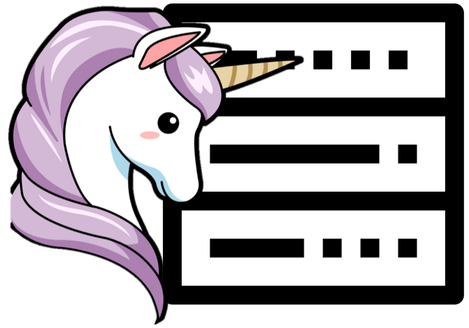


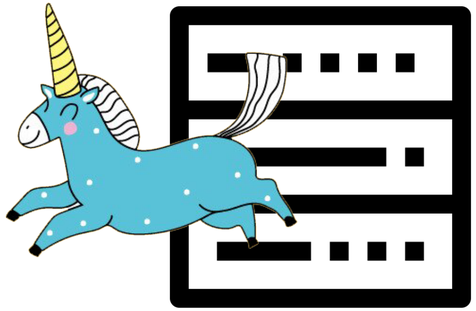




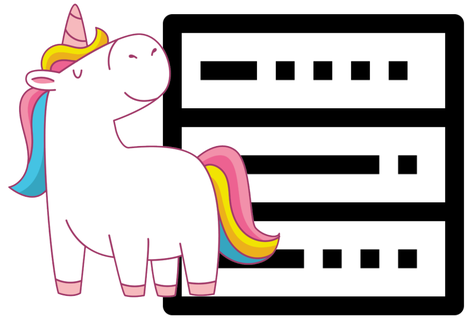
starts at 0, takes 0.5 seconds	`send a request to url1`
starts at 0.5, takes 10 seconds	`wait for the response`
starts at 10.5, takes 0 seconds	`return the data`
starts at 10.5, takes 0.5 seconds	`send a request to url2`
starts at 11, takes 10 seconds	`wait for the response`
starts at 21, takes 0 seconds	`return the data`
starts at 21, takes 0.5 seconds	`send a request to url3`
starts at 21.5, takes 10 seconds	`wait for the response`
starts at 31.5, takes 0 seconds	`return the data`
starts at 31.5, takes 0.5 seconds	`send a request to url4`
starts at 32, takes 10 seconds	`wait for the response`
	`return the data`

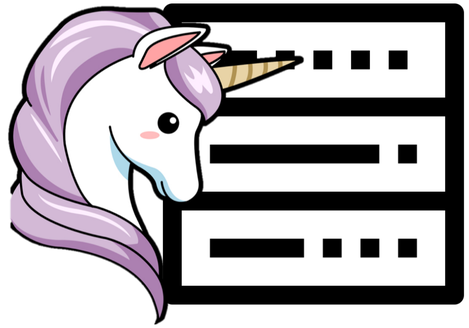


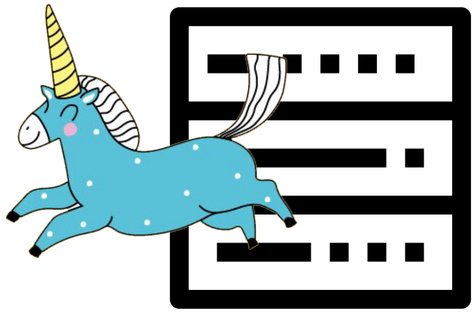




starts at 0, takes 0.5 seconds	`send a request to url1`
starts at 0.5, takes 10 seconds	`wait for the response`
starts at 10.5, takes 0 seconds	`return the data`
starts at 10.5, takes 0.5 seconds	`send a request to url2`
starts at 11, takes 10 seconds	`wait for the response`
starts at 21, takes 0 seconds	`return the data`
starts at 21, takes 0.5 seconds	`send a request to url3`
starts at 21.5, takes 10 seconds	`wait for the response`
starts at 31.5, takes 0 seconds	`return the data`
starts at 31.5, takes 0.5 seconds	`send a request to url4`
starts at 32, takes 10 seconds	`wait for the response`
starts at 42, takes 0 seconds	`return the data`







```
send a request to url1
send a request to url2
send a request to url3
send a request to url4
wait for a response from url1
return the data
wait for a response from url2
return the data
wait for a response from url3
return the data
wait for a response from url4
return the data
```

starts at 0, takes 0.5 seconds

```
send a request to url1
send a request to url2
send a request to url3
send a request to url4
wait for a response from url1
return the data
wait for a response from url2
return the data
wait for a response from url3
return the data
wait for a response from url4
return the data
```

(will arrive at 10)

starts at 0, takes 0.5 seconds

starts at 0.5, takes 0.5 seconds

```
send a request to url1          (will arrive at 10)
send a request to url2          (will arrive at 10.5)
send a request to url3
send a request to url4
wait for a response from url1
return the data
wait for a response from url2
return the data
wait for a response from url3
return the data
wait for a response from url4
return the data
```

starts at 0, takes 0.5 seconds

starts at 0.5, takes 0.5 seconds

starts at 1, takes 0.5 seconds

```
send a request to url1          (will arrive at 10)
send a request to url2          (will arrive at 10.5)
send a request to url3          (will arrive at 11)
send a request to url4
wait for a response from url1
return the data
wait for a response from url2
return the data
wait for a response from url3
return the data
wait for a response from url4
return the data
```

starts at 0, takes 0.5 seconds

starts at 0.5, takes 0.5 seconds

starts at 1, takes 0.5 seconds

starts at 1.5, takes 0.5 seconds

```
send a request to url1          (will arrive at 10)
send a request to url2          (will arrive at 10.5)
send a request to url3          (will arrive at 11)
send a request to url4          (will arrive at 11.5)
wait for a response from url1
return the data
wait for a response from url2
return the data
wait for a response from url3
return the data
wait for a response from url4
return the data
```

starts at 0, takes 0.5 seconds

starts at 0.5, takes 0.5 seconds

starts at 1, takes 0.5 seconds

starts at 1.5, takes 0.5 seconds

starts at 2, takes 8 seconds

```
send a request to url1
send a request to url2
send a request to url3
send a request to url4
wait for a response from url1
return the data
wait for a response from url2
return the data
wait for a response from url3
return the data
wait for a response from url4
return the data
```

(will arrive at 10)

(will arrive at 10.5)

(will arrive at 11)

(will arrive at 11.5)

starts at 0, takes 0.5 seconds

starts at 0.5, takes 0.5 seconds

starts at 1, takes 0.5 seconds

starts at 1.5, takes 0.5 seconds

starts at 2, takes 8 seconds

starts at 10, takes 0 seconds

```
send a request to url1
send a request to url2
send a request to url3
send a request to url4
wait for a response from url1
return the data
wait for a response from url2
return the data
wait for a response from url3
return the data
wait for a response from url4
return the data
```

(will arrive at 10)

(will arrive at 10.5)

(will arrive at 11)

(will arrive at 11.5)

starts at 0, takes 0.5 seconds

starts at 0.5, takes 0.5 seconds

starts at 1, takes 0.5 seconds

starts at 1.5, takes 0.5 seconds

starts at 2, takes 8 seconds

starts at 10, takes 0 seconds

starts at 10, takes 0.5 seconds

```
send a request to url1
send a request to url2
send a request to url3
send a request to url4
wait for a response from url1
return the data
wait for a response from url2
return the data
wait for a response from url3
return the data
wait for a response from url4
return the data
```

(will arrive at 10)

(will arrive at 10.5)

(will arrive at 11)

(will arrive at 11.5)

starts at 0, takes 0.5 seconds

starts at 0.5, takes 0.5 seconds

starts at 1, takes 0.5 seconds

starts at 1.5, takes 0.5 seconds

starts at 2, takes 8 seconds

starts at 10, takes 0 seconds

starts at 10, takes 0.5 seconds

starts at 10.5, takes 0 seconds

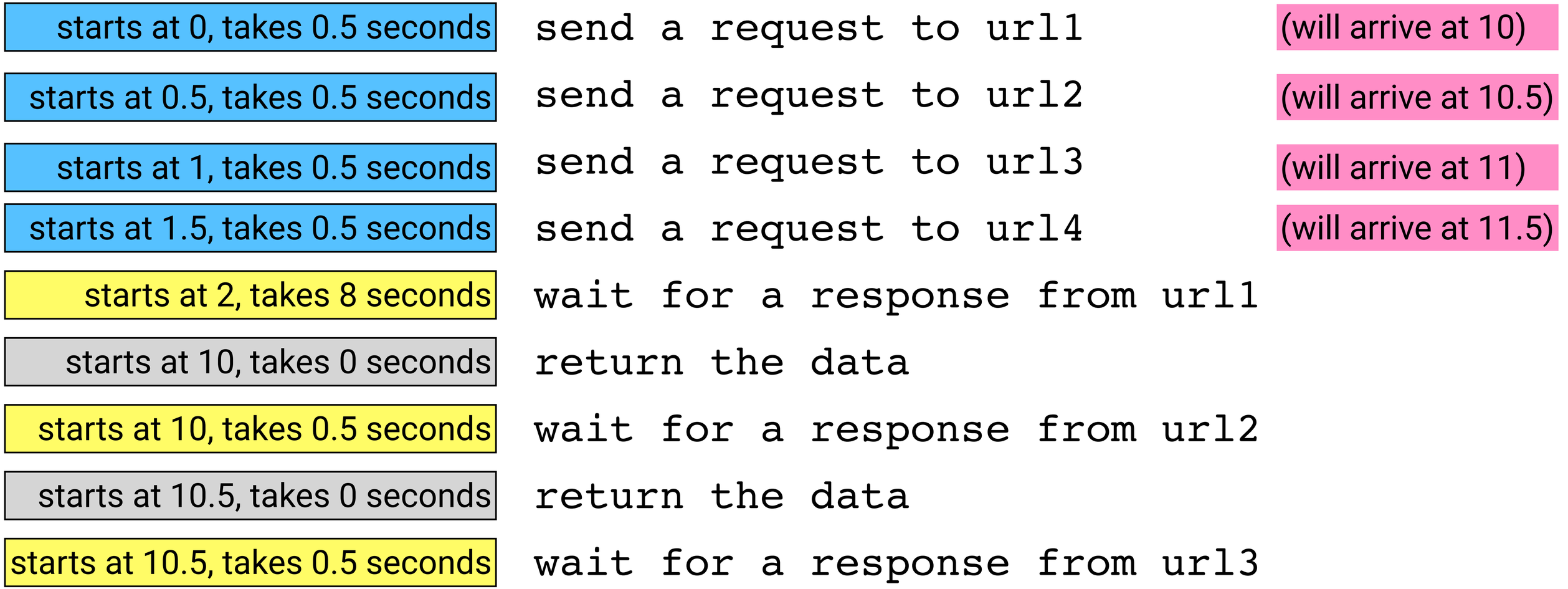
```
send a request to url1
send a request to url2
send a request to url3
send a request to url4
wait for a response from url1
return the data
wait for a response from url2
return the data
wait for a response from url3
return the data
wait for a response from url4
return the data
```

(will arrive at 10)

(will arrive at 10.5)

(will arrive at 11)

(will arrive at 11.5)


starts at 0, takes 0.5 seconds
send a request to url1
(will arrive at 10)
starts at 0.5, takes 0.5 seconds
send a request to url2
(will arrive at 10.5)
starts at 1, takes 0.5 seconds
send a request to url3
(will arrive at 11)
starts at 1.5, takes 0.5 seconds
send a request to url4
(will arrive at 11.5)
starts at 2, takes 8 seconds
wait for a response from url1
starts at 10, takes 0 seconds
return the data
starts at 10, takes 0.5 seconds
wait for a response from url2
starts at 10.5, takes 0 seconds
return the data
starts at 10.5, takes 0.5 seconds
wait for a response from url3
return the data
wait for a response from url4
return the data

starts at 0, takes 0.5 seconds

starts at 0.5, takes 0.5 seconds

starts at 1, takes 0.5 seconds

starts at 1.5, takes 0.5 seconds

starts at 2, takes 8 seconds

starts at 10, takes 0 seconds

starts at 10, takes 0.5 seconds

starts at 10.5, takes 0 seconds

starts at 10.5, takes 0.5 seconds

starts at 11, takes 0 seconds

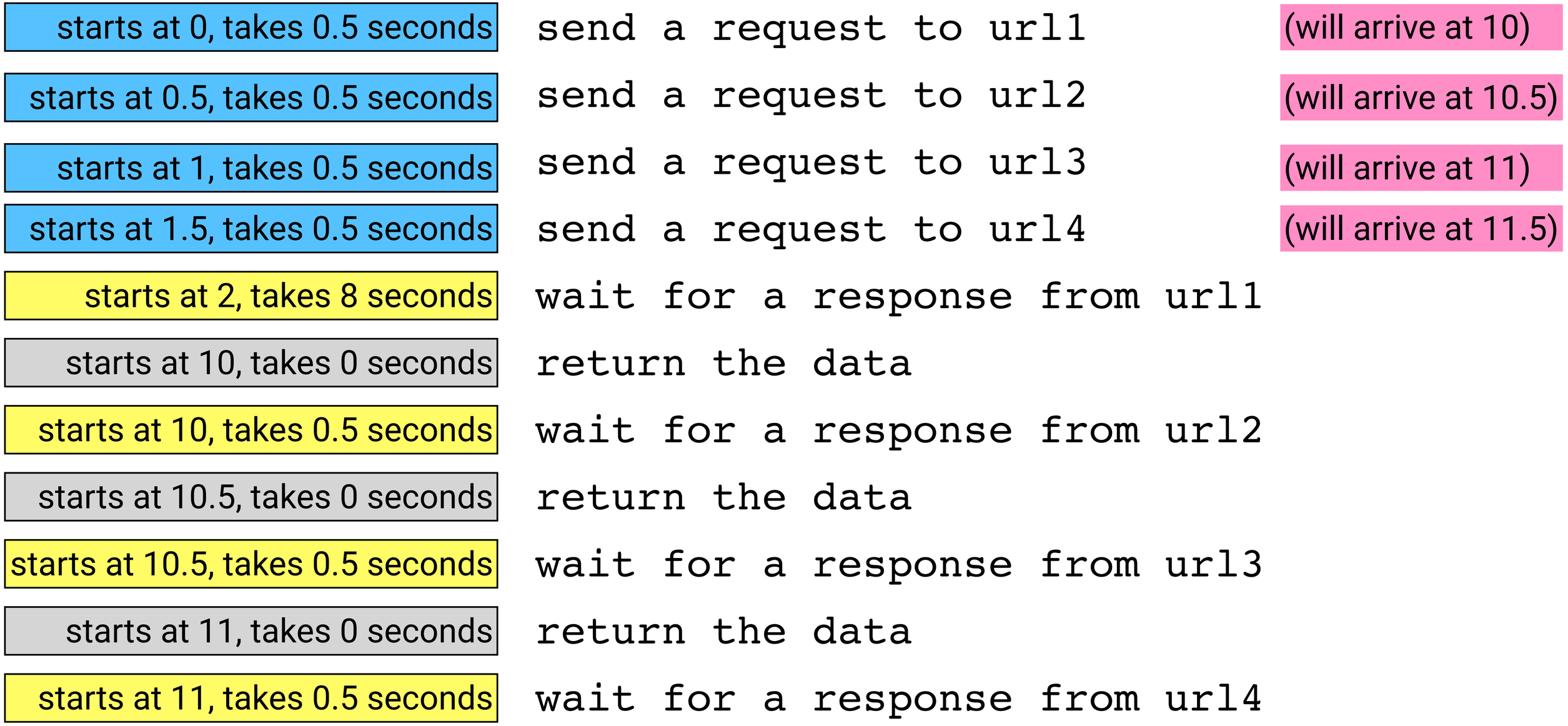
```
send a request to url1
send a request to url2
send a request to url3
send a request to url4
wait for a response from url1
return the data
wait for a response from url2
return the data
wait for a response from url3
return the data
wait for a response from url4
return the data
```

(will arrive at 10)

(will arrive at 10.5)

(will arrive at 11)

(will arrive at 11.5)


starts at 0, takes 0.5 seconds
send a request to url1
(will arrive at 10)
starts at 0.5, takes 0.5 seconds
send a request to url2
(will arrive at 10.5)
starts at 1, takes 0.5 seconds
send a request to url3
(will arrive at 11)
starts at 1.5, takes 0.5 seconds
send a request to url4
(will arrive at 11.5)
starts at 2, takes 8 seconds
wait for a response from url1
starts at 10, takes 0 seconds
return the data
starts at 10, takes 0.5 seconds
wait for a response from url2
starts at 10.5, takes 0 seconds
return the data
starts at 10.5, takes 0.5 seconds
wait for a response from url3
starts at 11, takes 0 seconds
return the data
starts at 11, takes 0.5 seconds
wait for a response from url4
return the data

starts at 0, takes 0.5 seconds	`send a request to url1`	(will arrive at 10)
starts at 0.5, takes 0.5 seconds	`send a request to url2`	(will arrive at 10.5)
starts at 1, takes 0.5 seconds	`send a request to url3`	(will arrive at 11)
starts at 1.5, takes 0.5 seconds	`send a request to url4`	(will arrive at 11.5)
starts at 2, takes 8 seconds	`wait for a response from url1`	
starts at 10, takes 0 seconds	`return the data`	
starts at 10, takes 0.5 seconds	`wait for a response from url2`	
starts at 10.5, takes 0 seconds	`return the data`	
starts at 10.5, takes 0.5 seconds	`wait for a response from url3`	
starts at 11, takes 0 seconds	`return the data`	
starts at 11, takes 0.5 seconds	`wait for a response from url4`	
starts at 11.5, takes 0 seconds	`return the data`	

Small problem: How do we break the function up and move the pieces around?

```
requests.get(url1)
```

...is really...

```
send a request to url1
wait for the response
return the data
```

```
requests.get(url2)
```

...is really...

```
send a request to url2
wait for the response
return the data
```

Small problem: How do we break the function up and move the pieces around?

```
requests.get(url1)
```

...is really...

```
send a request to url1
wait for the response
return the data
```

```
requests.get(url2)
```

...is really...

```
send a request to url2
wait for the response
return the data
```

Answer: Rely on the operating system

Threading and the OS

Threading and the OS

- Operating systems manage "threads" and "processes" on the computer

Threading and the OS

- Operating systems manage "threads" and "processes" on the computer
  - a process is a program that you're running

Threading and the OS

- Operating systems manage "threads" and "processes" on the computer
  - a process is a program that you're running
  - a thread is a sequence of instructions within the process

Threading and the OS

- Operating systems manage "threads" and "processes" on the computer
  - a process is a program that you're running
  - a thread is a sequence of instructions within the process
- The OS knows when a thread is waiting for something to happen (e.g., waiting for a file to be downloaded)

Threading and the OS

- Operating systems manage "threads" and "processes" on the computer
  - a process is a program that you're running
  - a thread is a sequence of instructions within the process
- The OS knows when a thread is waiting for something to happen (e.g., waiting for a file to be downloaded)
- In that situation, it can switch to another stream of execution

Threading and the OS

- Operating systems manage "threads" and "processes" on the computer
  - a process is a program that you're running
  - a thread is a sequence of instructions within the process
- The OS knows when a thread is waiting for something to happen (e.g., waiting for a file to be downloaded)
- In that situation, it can switch to another stream of execution
  - For example, if one program is waiting for a file to download and you play a YouTube video in the meantime, your computer can switch to focus on the browser while the file is downloading

Threading and the OS

- Operating systems manage "threads" and "processes" on the computer
  - a process is a program that you're running
  - a thread is a sequence of instructions within the process
- The OS knows when a thread is waiting for something to happen (e.g., waiting for a file to be downloaded)
- In that situation, it can switch to another stream of execution
  - For example, if one program is waiting for a file to download and you play a YouTube video in the meantime, your computer can switch to focus on the browser while the file is downloading
  - In reality: context switches happen frequently; take CS 111!

single-threaded downloading

```
import requests

def download_picture(url, filename):
    r = requests.get(url)
    open(filename, 'wb').write(r.content)

url = 'https://cataas.com/cat/says/chase%20is%20the%20bomb!'
filename = 'chase.jpg'

download_picture(url, filename)
```

multi-threaded downloading

```
import requests
from threading import Thread

def download_picture(url, filename):
    r = requests.get(url)
    open(filename, 'wb').write(r.content)

url = 'https://cataas.com/cat/says/chase%20is%20the%20bomb!'
filename = 'chase.jpg'

t = Thread(target=download_picture, args=(url, filename))
t.start()
t.join()
```

multi-threaded downloading

```
t1 = Thread(target=download_picture, args=(url1, filename1))
t2 = Thread(target=download_picture, args=(url2, filename2))

t1.start()
t2.start()

t1.join()
t2.join()
```

Demo 1: Multithreading

The Global Interpreter Lock

Multithreading the `sum_upto` function

Demo 2: GIL

Multi-threading the `sum_upto` function

Turn and Talk

- Why does the program speed up when we are downloading pictures, but not when we're competing `sum_upto`?
- Why does the `sum_upto` program get faster using C++ threads but not with Python threads?

The Global Interpreter Lock

- Python uses reference counting...

```
import sys

lst = []
extra_ref = lst
sys.getrefcount(lst) # => 3
```

- When the reference count for an object hits zero, that object is deleted off of the system
- If an object is being accessed by multiple threads at the same time, the reference count can get messed up...
- The interpreter has a "lock" which means that **only one thread can execute Python code at a time**

click here to read more!

Downloading images vs. summing

Downloading images vs. summing

- In the download image example, the CPU switches when a thread is ***waiting*** on the internet, and only one piece of Python code is running at a time
  - This is called "I/O-bound"

Downloading images vs. summing

- In the download image example, the CPU switches when a thread is ***waiting*** on the internet, and only one piece of Python code is running at a time
  - This is called "I/O-bound"

```
requests.get(url)
```

```
send a request to url
wait for the response
return the data
```

Downloading images vs. summing

- In the download image example, the CPU switches when a thread is ***waiting*** on the internet, and only one piece of Python code is running at a time
  - This is called "I/O-bound"
- In the sum example, the code is never ***waiting*** on an external source — the CPU is always executing code, so it can't switch to another thread
  - This is called "CPU-bound"

```
requests.get(url)

send a request to url

wait for the response

return the data
```

Downloading images vs. summing

- In the download image example, the CPU switches when a thread is ***waiting*** on the internet, and only one piece of Python code is running at a time
  - This is called "I/O-bound"
- In the sum example, the code is never ***waiting*** on an external source — the CPU is always executing code, so it can't switch to another thread
  - This is called "CPU-bound"

```
requests.get(url)

send a request to url

wait for the response

return the data
```

```
def sum_upto(n):
    output = 0
    for i in range(1, n+1):
        output += i
    return output
```

C++ vs. Python

C++ vs. Python

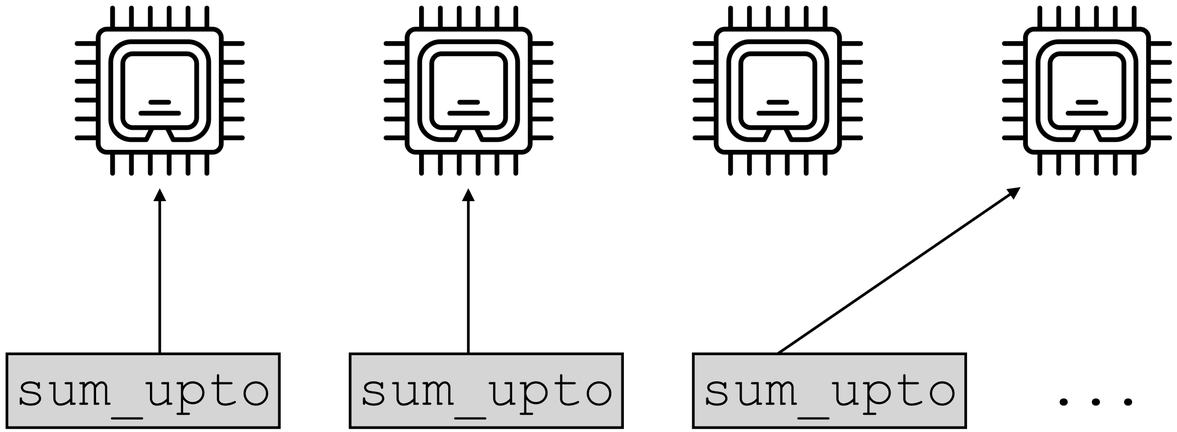
- C++ doesn't have a global interpreter lock, so CPU-bound tasks can execute at the same time

C++ vs. Python

- C++ doesn't have a global interpreter lock, so CPU-bound tasks can execute at the same time
- My computer has 10 CPU cores in it, which means it's capable of executing up to 10 instructions at the same time

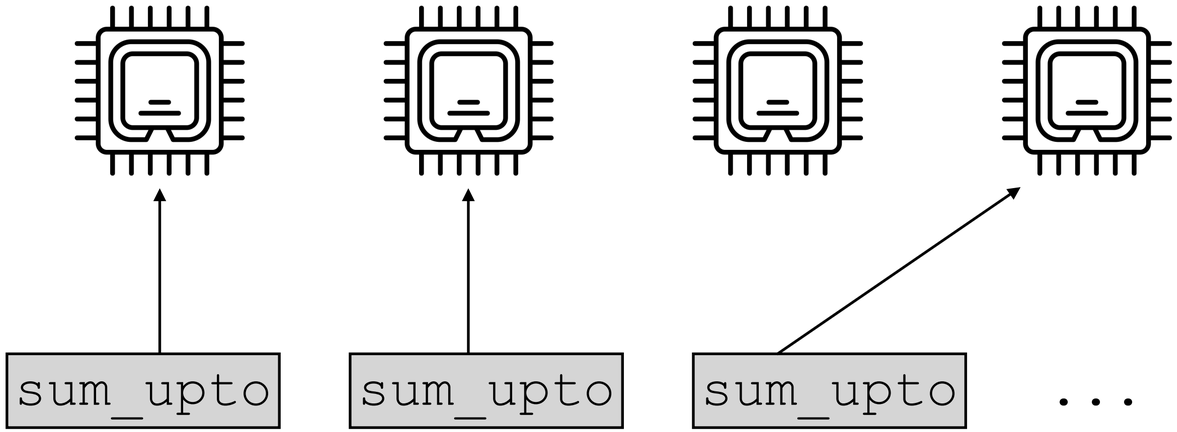
C++ vs. Python

- C++ doesn't have a global interpreter lock, so CPU-bound tasks can execute at the same time
- My computer has 10 CPU cores in it, which means it's capable of executing up to 10 instructions at the same time



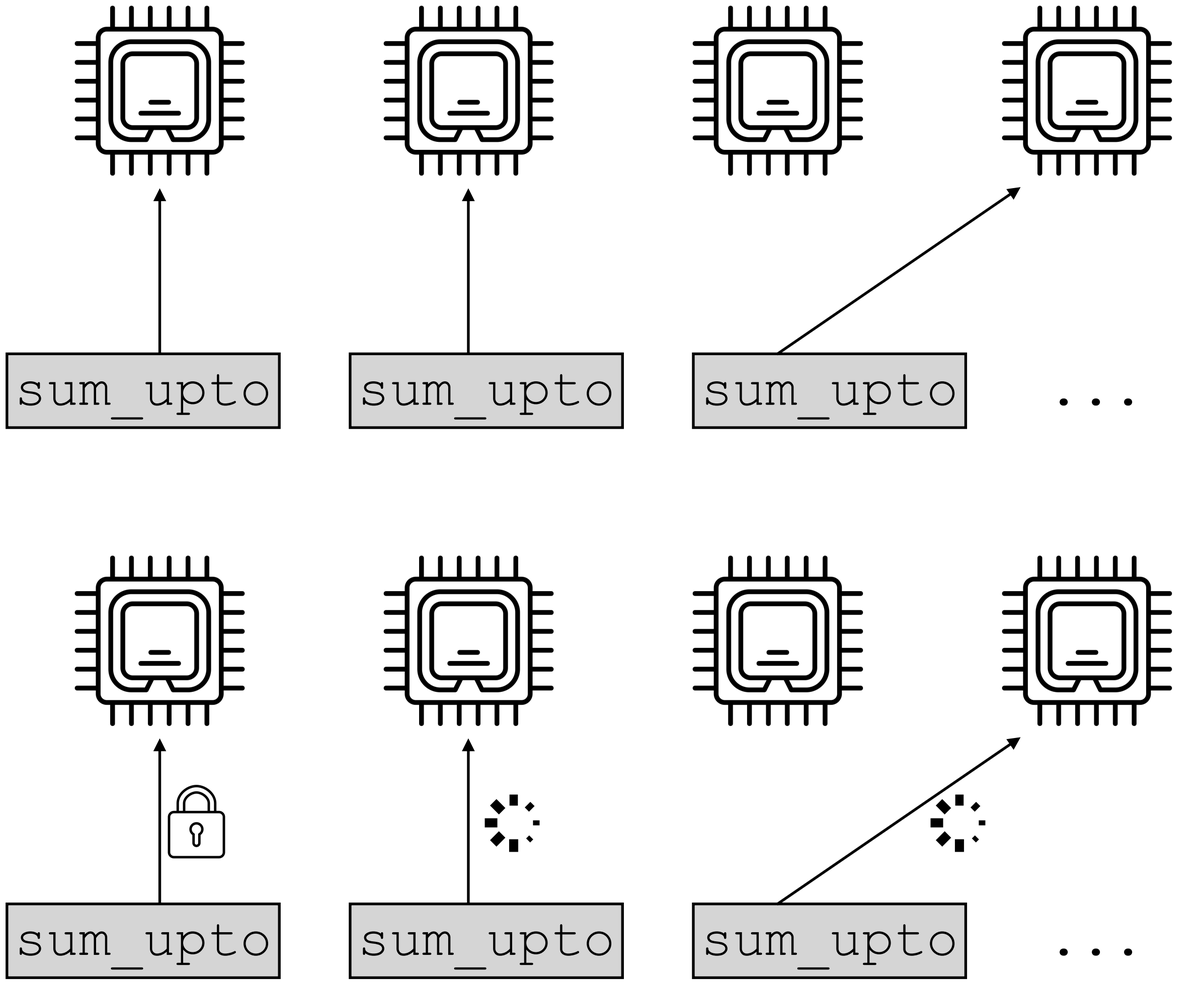
C++ vs. Python

- C++ doesn't have a global interpreter lock, so CPU-bound tasks can execute at the same time
- My computer has 10 CPU cores in it, which means it's capable of executing up to 10 instructions at the same time
- Python can't take advantage of this with threading, but other languages can



C++ vs. Python

- C++ doesn't have a global interpreter lock, so CPU-bound tasks can execute at the same time
- My computer has 10 CPU cores in it, which means it's capable of executing up to 10 instructions at the same time
- Python can't take advantage of this with threading, but other languages can



Draw a Diagram

Include:

- A hypothetical (reasonable) ordering of events
- Which parts might be executed in parallel
- Where the GIL prevents parallelization
- Cute pictures — bonus points if there are unicorns

```
def download_picture(url, filename):
    # Download the image
    r = requests.get(url)

    # Convert the image to PIL format
    img = Image.open(BytesIO(r.content))

    # Apply a filter to the image and save
    img = img.filter(ImageFilter.FIND_EDGES)
    img.save(filename)

def download_threads():
    threads = [
        Thread(
            target=download_picture,
            args=(url, filename)
        )
        for url, filename in FILES
    ]

    for thread in threads:
        thread.start()

    for thread in threads:
        thread.join()
```

Extending Python with C/C++

How does this work?

How does this work?

- You're (probably) working with CPython — Python, written in C — so the interpreter is written in C, which parses and executes your Python commands in realtime

How does this work?

- You're (probably) working with CPython — Python, written in C — so the interpreter is written in C, which parses and executes your Python commands in realtime
- It's possible to instruct the interpreter to directly run C code...

How does this work?

- You're (probably) working with CPython — Python, written in C — so the interpreter is written in C, which parses and executes your Python commands in realtime
- It's possible to instruct the interpreter to directly run C code...

```
import my_c_module

my_c_module
.method('unicorns', 1)
```

How does this work?

- You're (probably) working with CPython — Python, written in C — so the interpreter is written in C, which parses and executes your Python commands in realtime
- It's possible to instruct the interpreter to directly run C code...

```
import my_c_module

my_c_module
.method('unicorns', 1)
```

```
static PyObject*
method(PyObject *self, PyObject *args) {
    char *s;
    int n;

    if (!PyArg_ParseTuple(args, "si", &s, &n))
        return NULL;

    // do something with s, n
}
```

Demo 4: Extending Python