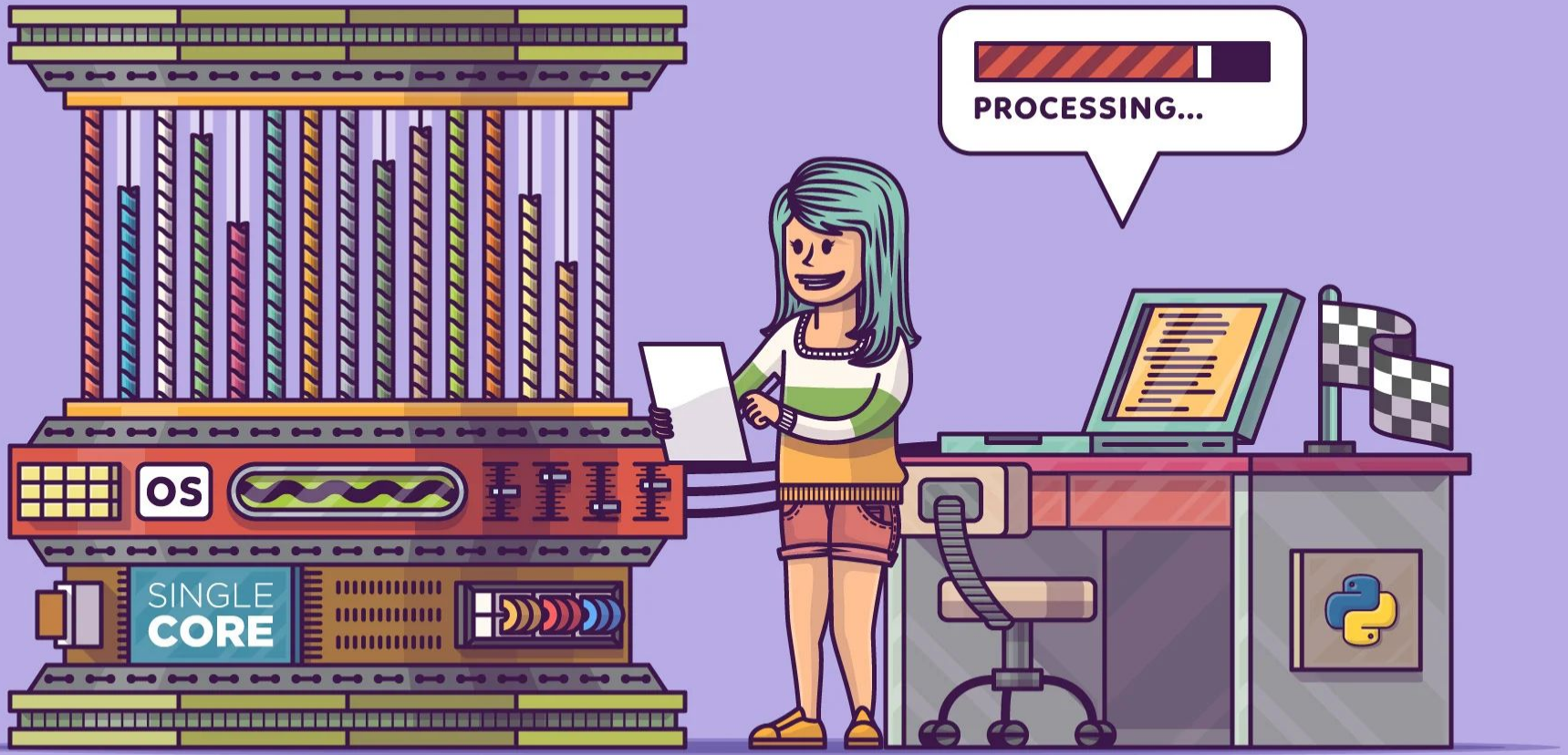
Gevent & Eventlet

Diferencia entre los dos pools de Celery.

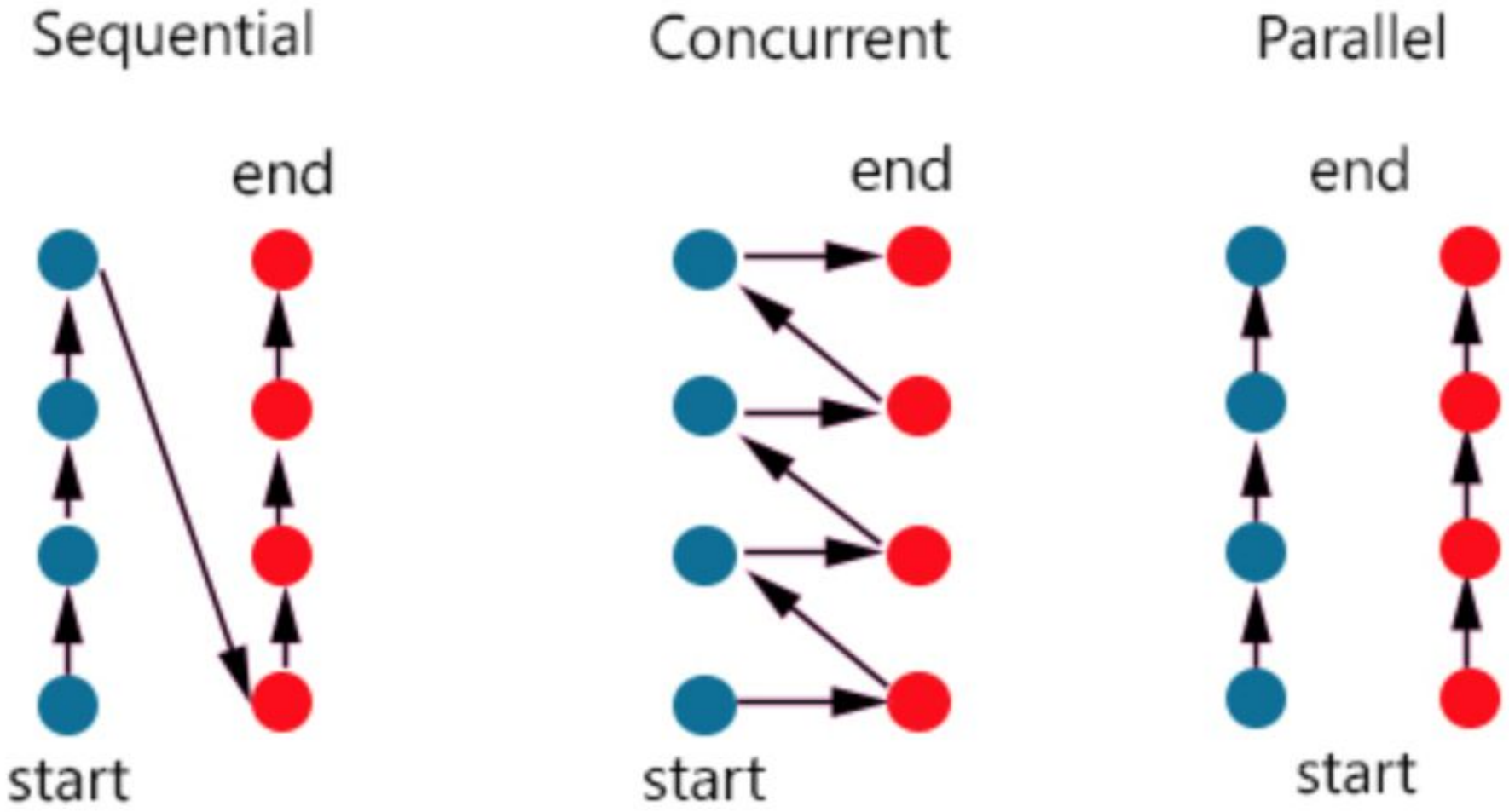
Presenta:
Fernando Pérez Gómez

Introducción

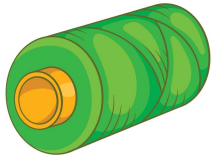
Real Python



Synchronous vs. Asynchronous


Sequential
end
start
Concurrent
end
start
Parallel
end
start

Greenlets



- Usados en la programación concurrente.

- Permiten la **ejecución de tareas de forma cooperativa** *(el programa/nosotros controla cuando se ejecuta)* y en **un solo hilo de ejecución** *(pero pueden existir varios greenlets)*.

Ejemplo de Greenlets

Notebook en Colap

```python
from greenlet import greenlet

def test1():
    print("[gr1] main  -> test1")
    gr2.switch()
    print("[gr1] test1 <- test2")
    return 'test1 done'

def test2():
    print("[gr2] test1 -> test2")
    gr1.switch()
    print("This is never printed.")


gr1 = greenlet(test1)
gr2 = greenlet(test2)
```

Greenlets vs Threads - Naturaleza

Threads –

Son **manejados por el sistema operativo** y comparten recursos con otros hilos del mismo proceso.

Greenlets –

Son implementaciones de hilos cooperativos, lo que significa que **son manejados por el propio programa** (nosotros) o un programa externo llamado “supervisor” (como un event loop) **y no por el sistema operativo.**

Greenlets vs Threads - Sincronización y Comunicación

Threads –

Comparten los mismos recursos dentro de un proceso. Lo que los hace **más sencillo de compartir datos y comunicarse entre sí.** Sin embargo, **también pueden llevar a problemas de concurrencia** (race conditions y bloqueos) **lo requiere una programación cuidadosa**.

Greenlets vs Threads - Sincronización y Comunicación

Greenlets –

Se ejecutan dentro de un único hilo del sistema y **NO comparten directamente los recursos** (memoria) como otros greenlets.

La comunicación y sincronización entre ellos se logra a través de métodos de paso de mensajes o mediante el uso de un supervisor que los coordine.

Greenlets vs Threads - Concurrencia y Multitarea

Threads –

Pueden aprovechar múltiples núcleos de CPU en sistemas multiprocesador/multinúcleo, lo que **permite una ejecución más rápida de tareas intensivas en CPU****. **Sin embargo, volvemos a los problemas de antes de compartir los recursos.**

Greenlets –

Al ser cooperativos y ejecutarse en un solo hilo del sistema operativo, **NO aprovechan directamente múltiples núcleos de CPU.**

Greenlets vs Threads - Complejidad

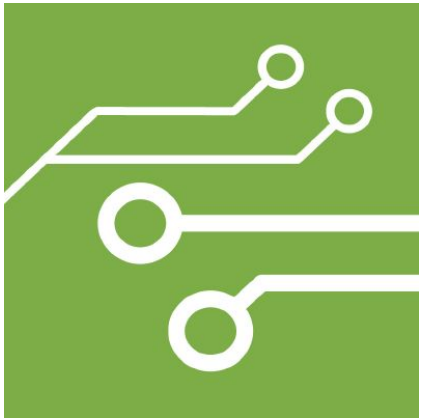
Threads –

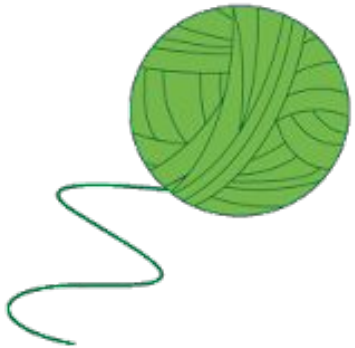
Más complejos de manejar debido a los problemas que se mencionaron anteriormente **de concurrencia y sincronización.**

Greenlet –

Al ser más ligeros y manejados por el programa, los greenlets **suelen ser más sencillos de utilizar y de depurar.** Sin embargo, **también pueden requerir más planificación y diseño por parte del programador para garantizar la cooperación adecuada entre ellos.**

Gevent y Eventlet



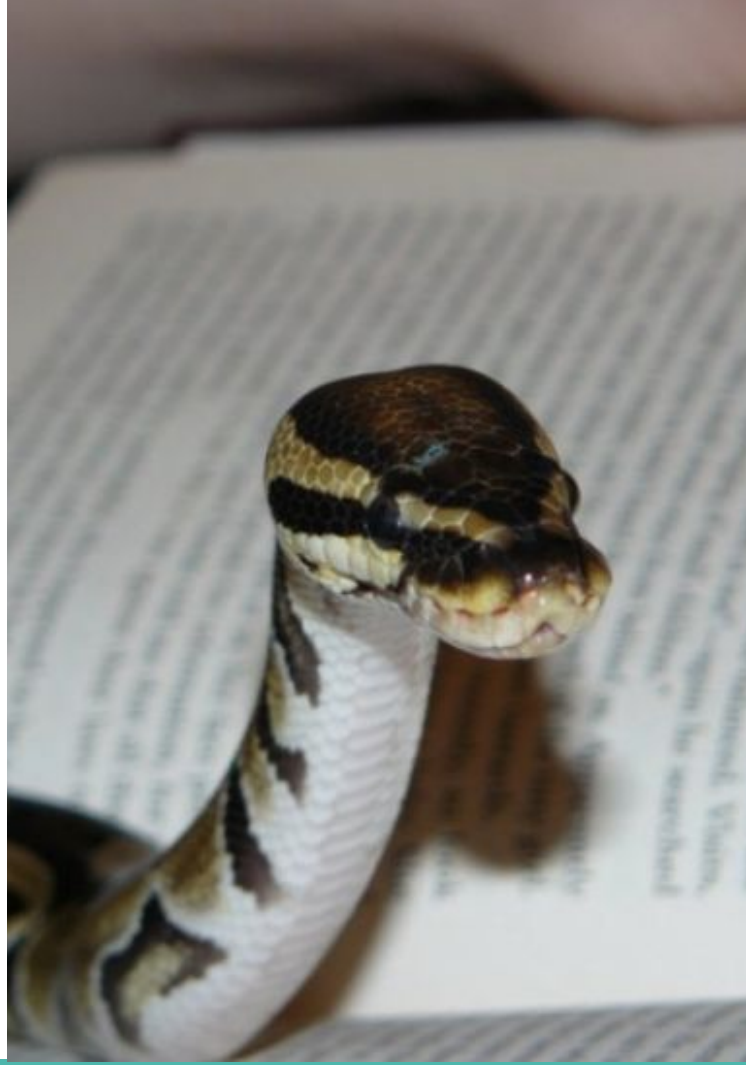


Historia - ¿Cómo surge Gevent?

- Eventlet estaba **por delante de otras opciones existentes** en cuestión de características y de facilidad de uso.

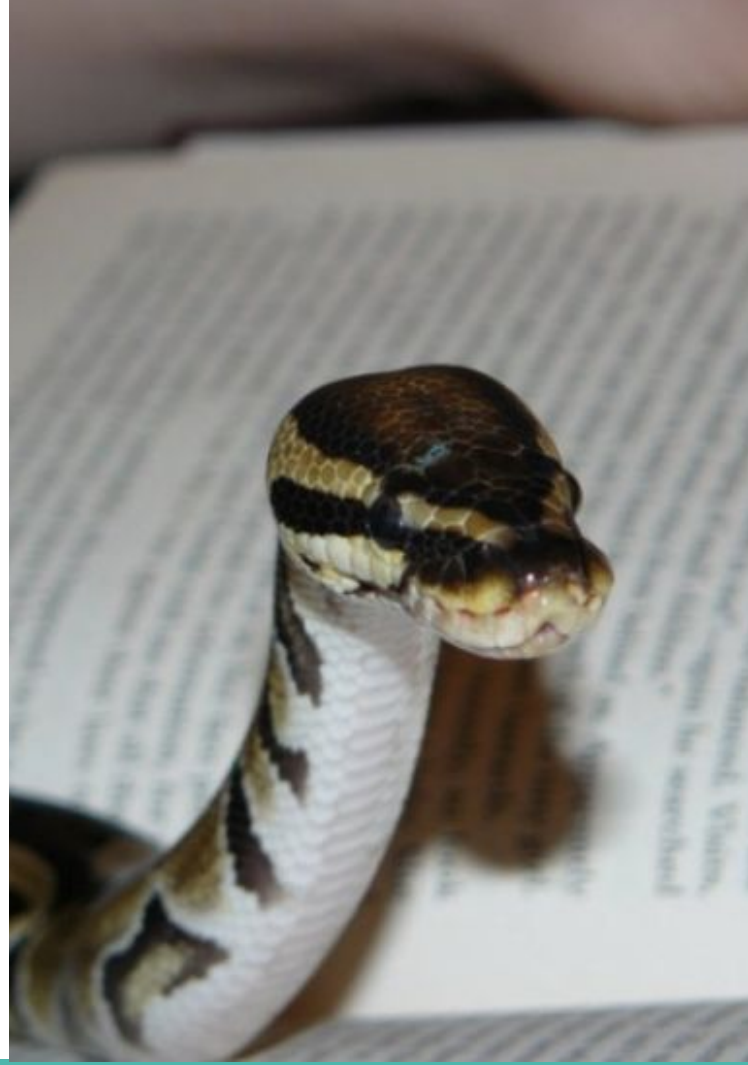
- Denis Bilenko había trabajado previamente con Eventlet y descubrió una serie de errores. Reescribió la mayor parte del núcleo y la corrección fue integrada en la versión 0.8.11

- Posteriormente, inicio otro proyecto que incorporaba Eventlet pero no cumplia un par de requisitos....



- Se requeria el **event-loop de libevent** por una librería en C que lo usaba y quería usarse juntos **en un único proceso.** En ese momento **NO tenía soporte** para libenv. **Ahora internamente lo usa en el pool pyevent.**

- **Necesitaba que el módulo socker funcionara perfectamente.** En ese momento **Eventlet tenia algunos errores que podrían causar una operación que colgara al socket.**



Diferencias entre Gevent y Eventlet

Diferencias - Convención

- A diferencia de Evenlet, la interfaz de **Gevent sigue las convenciones establecidas por la biblioteca estandar.**

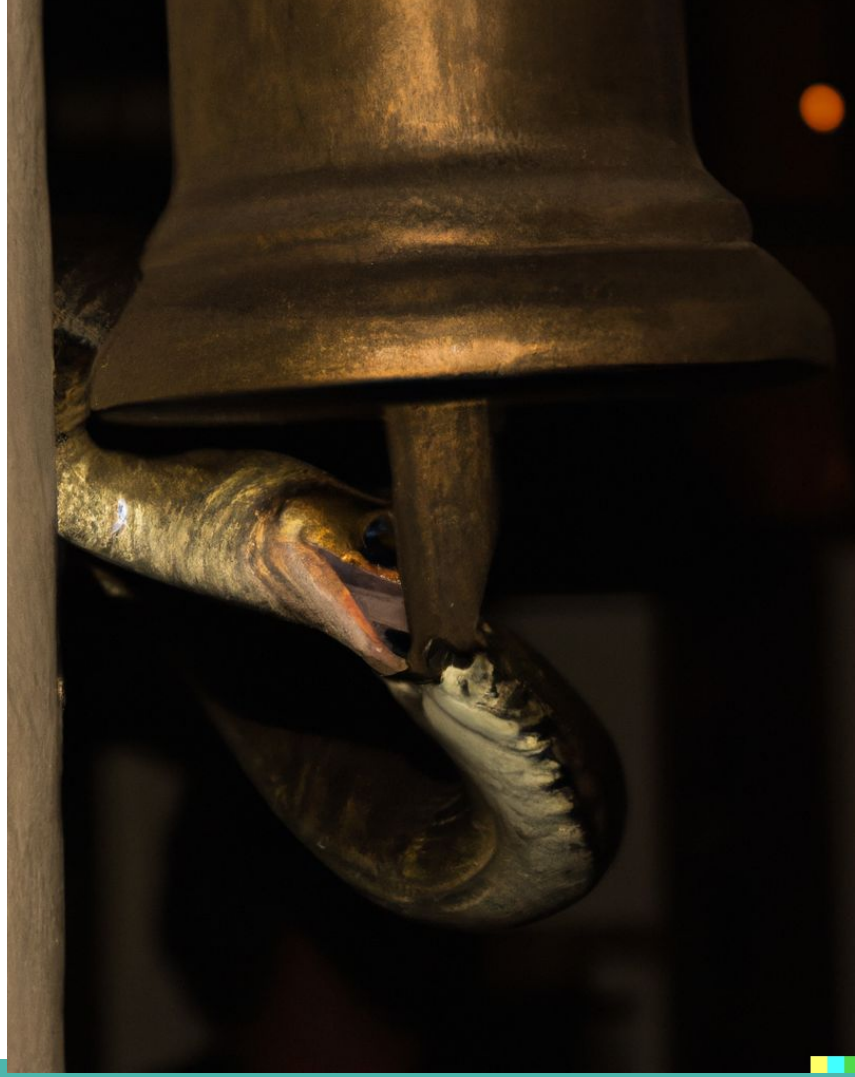
  Por ejemplo, gevent.event.Event tiene la misma interfaz y la misma semántica que threading.Event y multiprocessing.Event pero funciona a través de greenlets.

- **Disponer de una interfaz coherente mejora la velocidad a la que podemos leer y razonar sobre el código. Aprender la API también resulta más fácil.**
-
-

Diferencias - Event-loop (notificación de eventos)

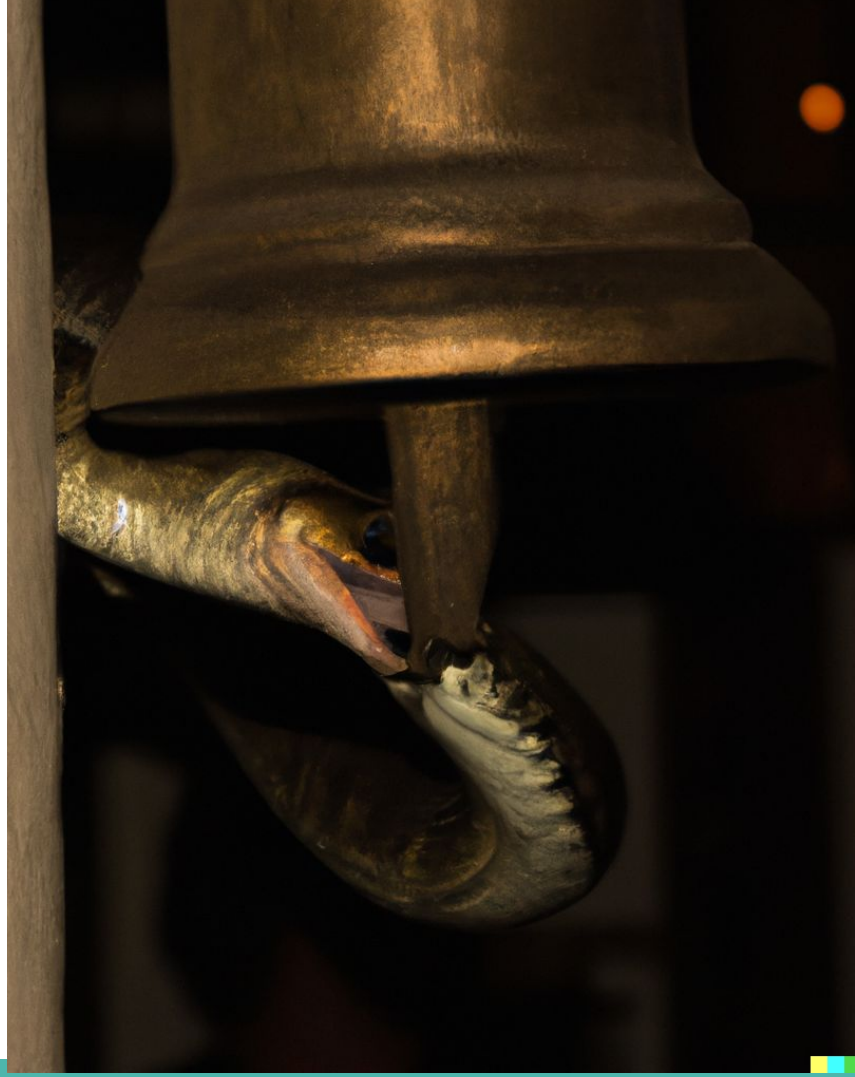
- Al principio Gevent se basaba en **libevent** ahora usa libev.
  El API de libevent ofrece un mecanismo para **ejecutar una función callback cuando un evento específico ocurre en un file-descriptor o después de un tiempo de espera.**

- Eventlet utiliza su **propio event-loop con python puro**.



Libevent ofrece:

- Tiene un **rendimiento superior**.
- La **gestión de señales** está integrada en el bucle de eventos.
- **Otras librerías basadas en libevent pueden integrarse con tu aplicación** a través de un único bucle de eventos.



Gevent ***pretende ser un núcleo pequeño y estable*** *en el que todos puedan confiar.*

Delega el trabajo a libevent siempre que sea posible *y proporciona abstracciones convenientes para la programación basada en coroutines.*

Está ***inspirado en Eventlet pero no es un fork*** *y presenta una nueva API.*

Referencias

- **“Celery Execution Pools: What is it all about?”**, 2018. Recuperado el 25 de julio de 2023 del siguiente enlace https://distributedpython.com/posts/celery-execution-pools-what-is-it-all-about/
- **“Greenlet: Lightweight concurrent programming”**, 2011, Armin Rigo y Christian Tismer. Recuperado el 25 de julio de 2023 del siguiente enlace https://greenlet.readthedocs.io/en/latest/
- **“Comparing gevent to eventlet”**, 2010, Denis. Recuperado el 25 de julio de 2023 del siguiente enlace https://blog.gevent.org/2010/02/27/why-gevent/
- **“Libev and libevent”**, 2011, Denis. Recuperado el 25 de julio de 2023 del siguiente enlace https://blog.gevent.org/2011/04/28/libev-and-libevent/
- **“Gevent: Official documentation”**. Recuperado el 25 de julio de 2023 del siguiente enlace https://blog.gevent.org/2011/04/28/libev-and-libevent/
-