# Anuncios

1. Recuerden contestar la ECA y Encuesta Medio Semestre.
2. Hoy: actividad formativa. No olviden el *feedback*.
3. Notas T00. ¡Recuerden la recorrección! NO hay más plazo.
4. El sábado termina el plazo para realizar la T01.
5. Programatón a las 17:00.

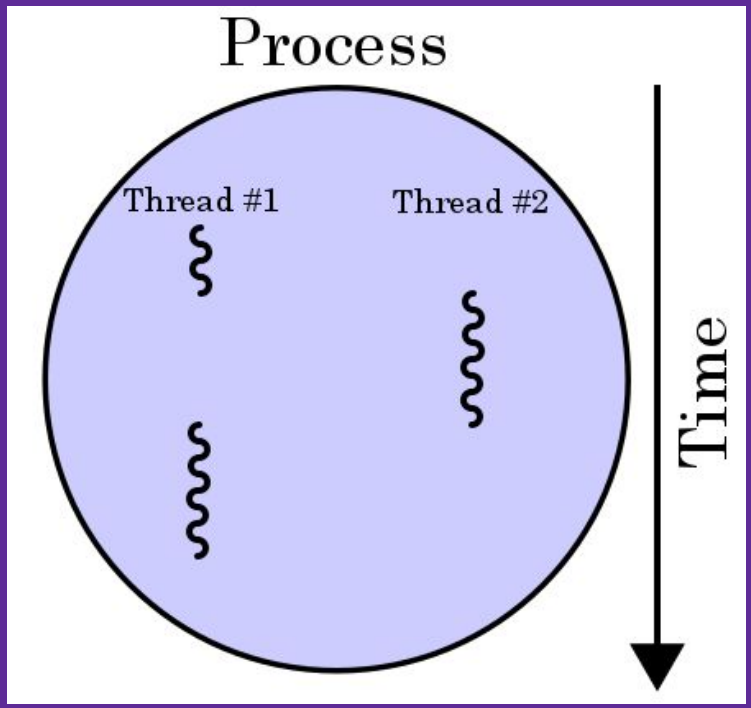
# *Threading*

*Semana 08 - Jueves 12 de septiembre de 2019*

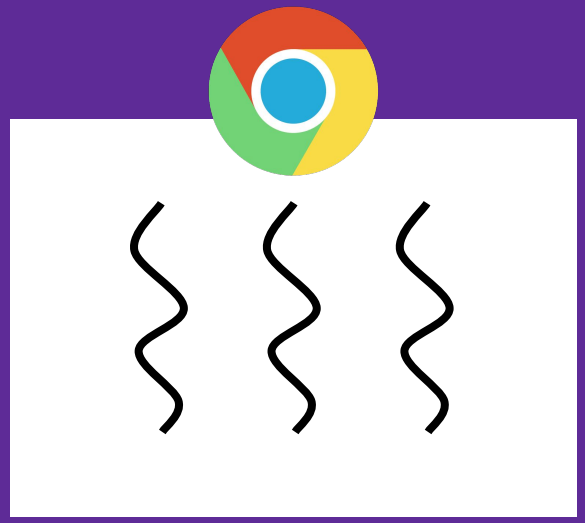
# Paralelismo

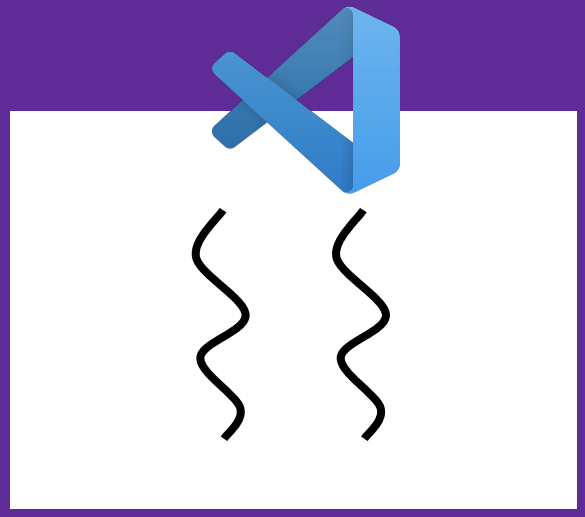
- ¿Sólo un procesador?
- *Scheduling*
- *Slicing*

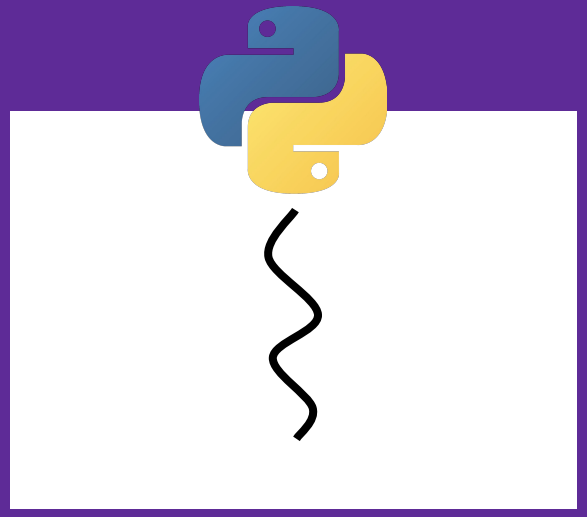
# Paralelismo



# Paralelismo: Procesos y *Threads*







# *Threads*

- `start()`
- `run()`

- *Thread* principal
- Otros *threads*

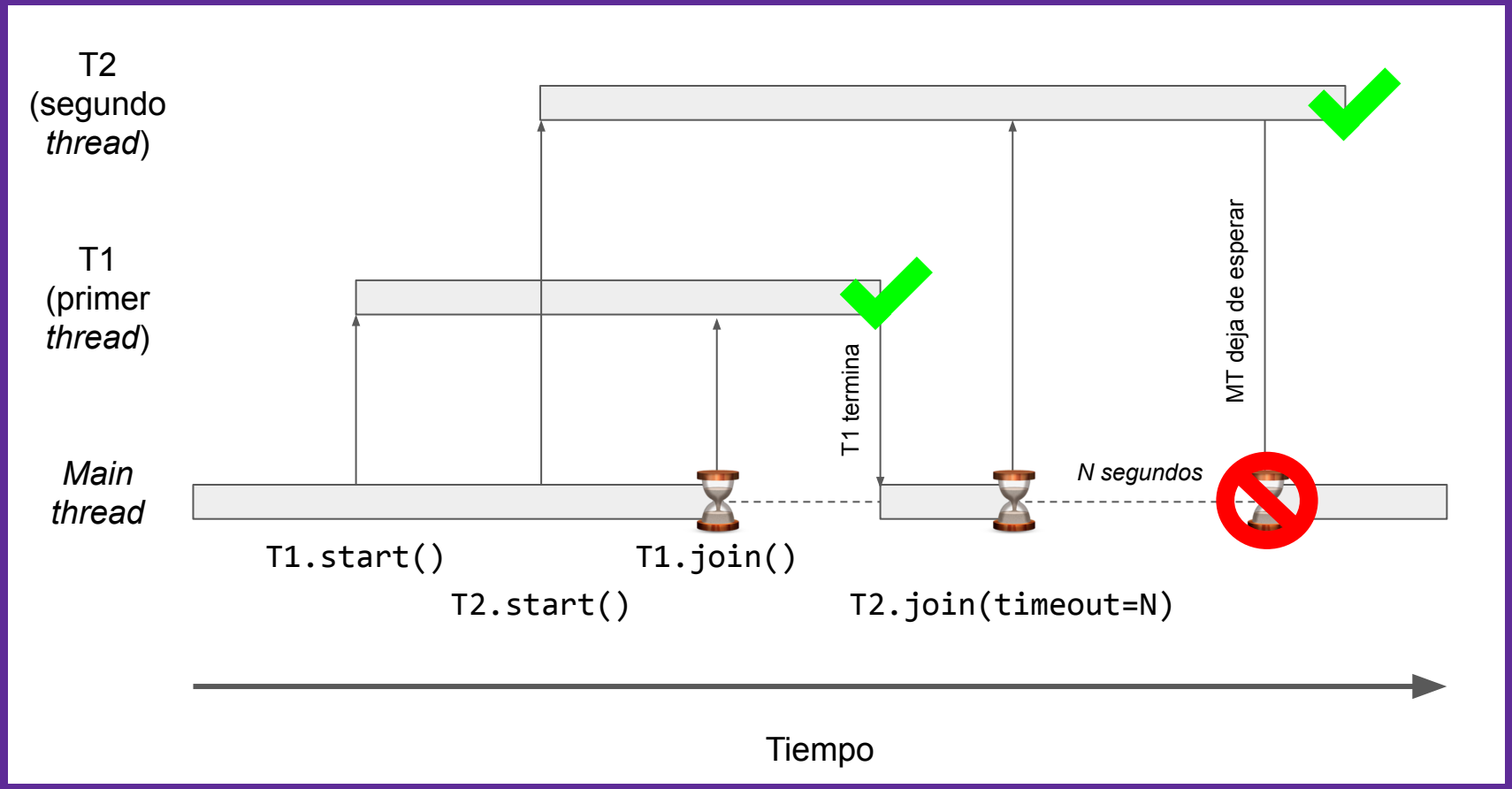
# Threads

```
from threading import Thread

def funcion():
    # secuencia de instrucciones
    ...


t = Thread(target = funcion)
t.start()
```

# Threads

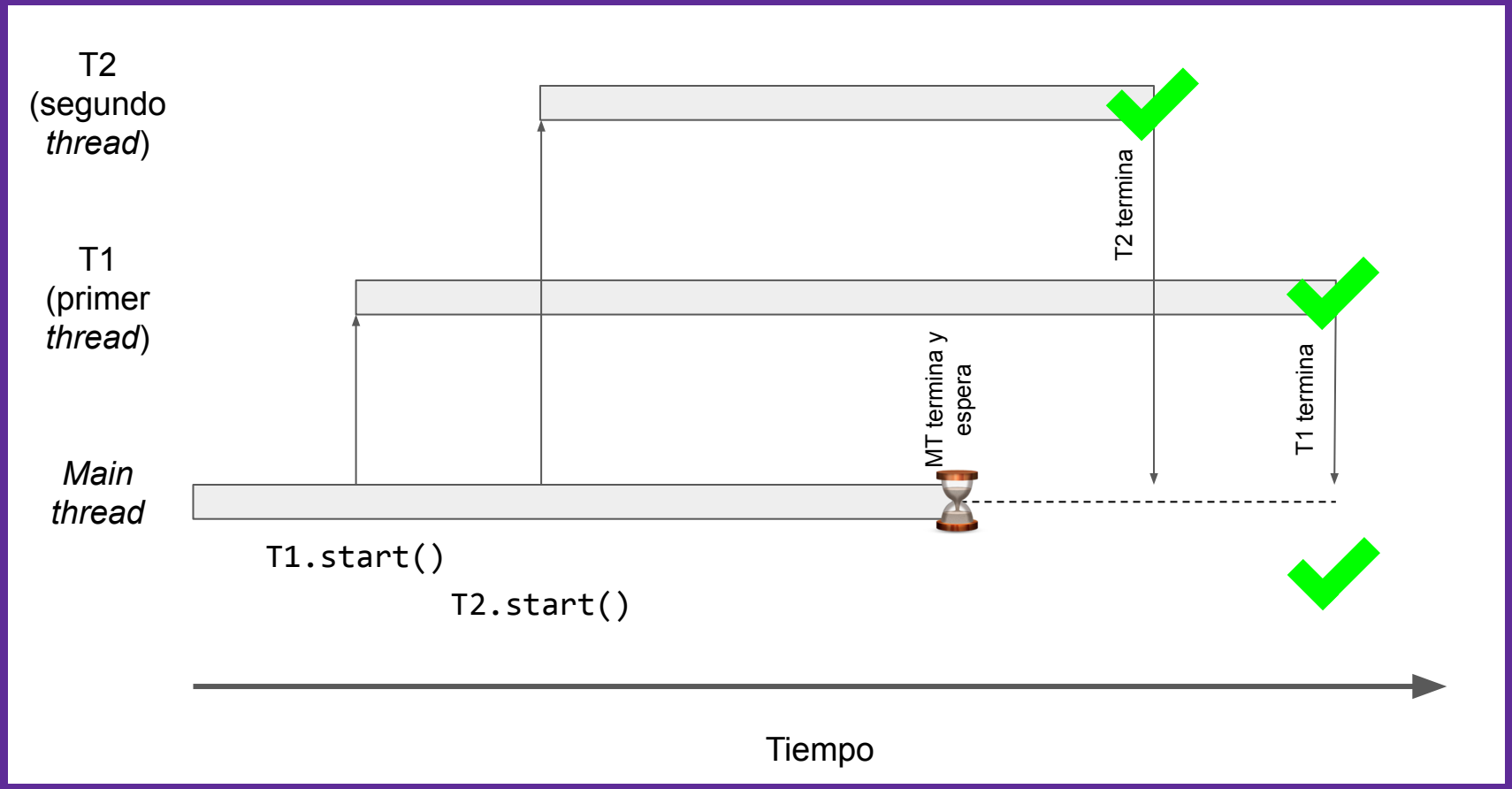
```
from threading import Thread

class MiThread(Thread):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs) # ¡Importante!

    def run(self):
        # Este metodo inicia el trabajo de este thread
        # cuando lo ejecutamos el metodo start()
        print(f"{self.name} partiendo...")

t = MiThread()
t.start()
```

# join()

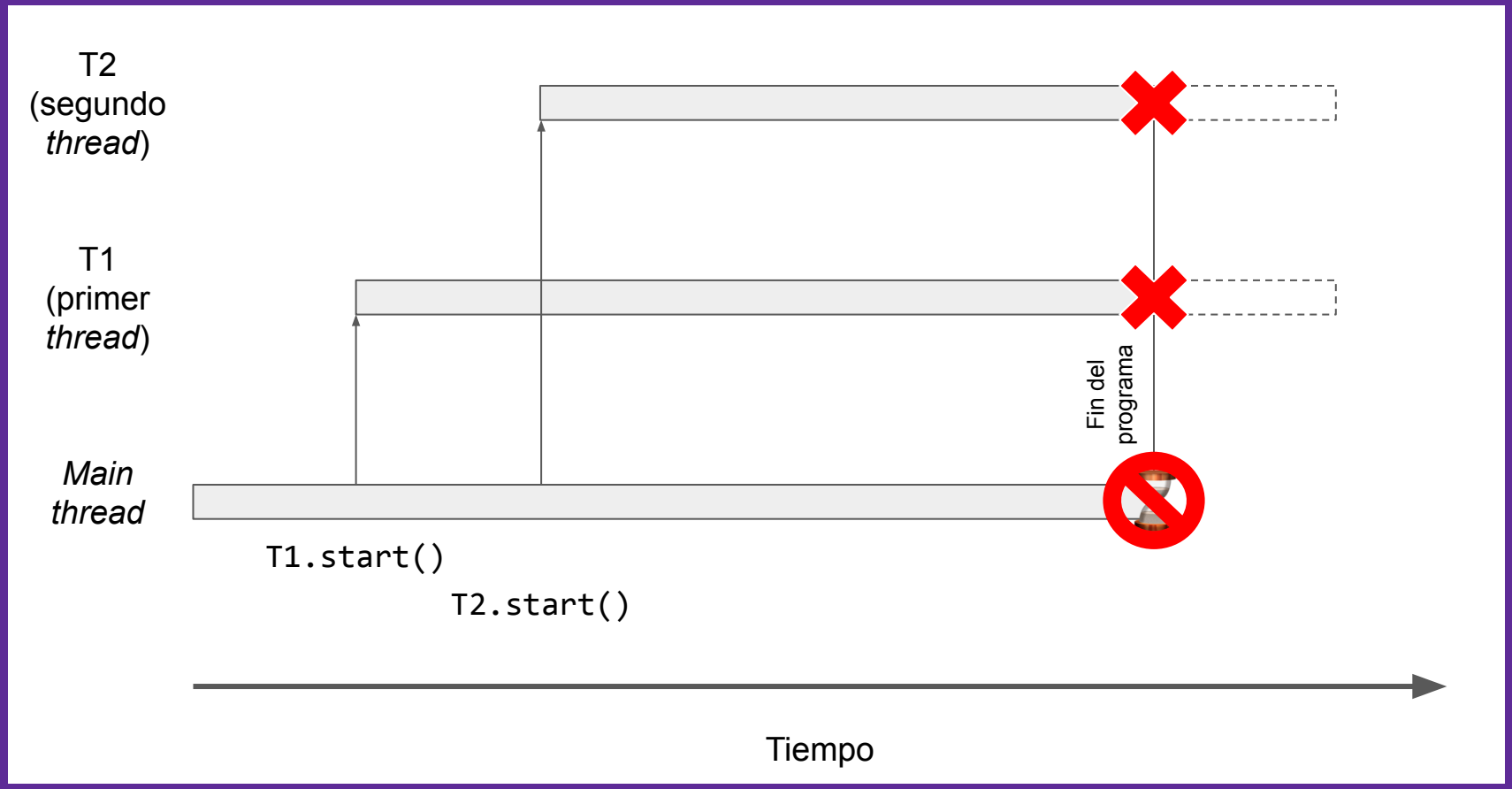

T2
(segundo
*thread*)
T1
(primer
*thread*)
*Main*
*thread*
T1 termina
MT deja de esperar
N segundos
T1.start()
T1.join()
T2.start()
T2.join(timeout=N)
Tiempo

# daemon=False

## Comportamiento por defecto



# daemon=True


T2
(segundo
*thread*)
T1
(primer
*thread*)
*Main*
*thread*
Fin del
programa
T1.start()
T2.start()
Tiempo

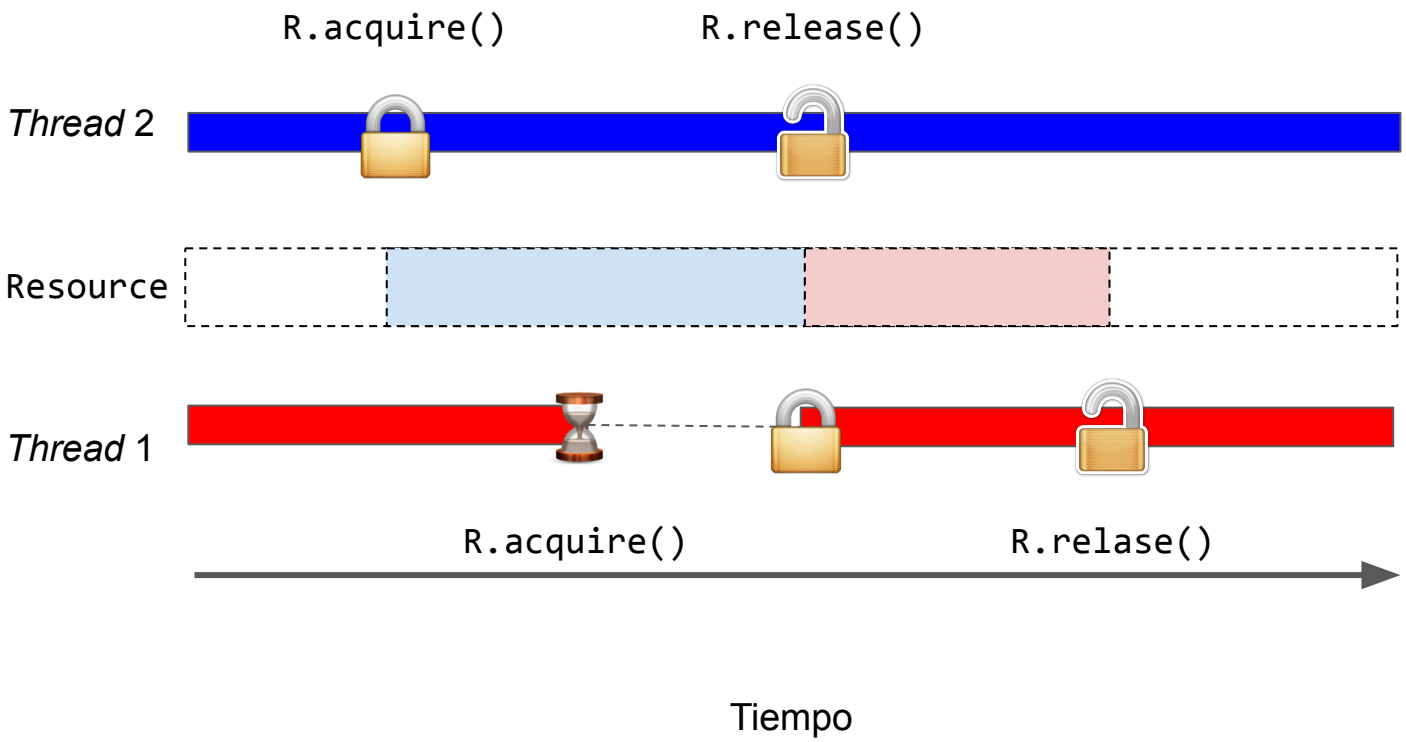
# Concurrencia

# Asignación de recursos

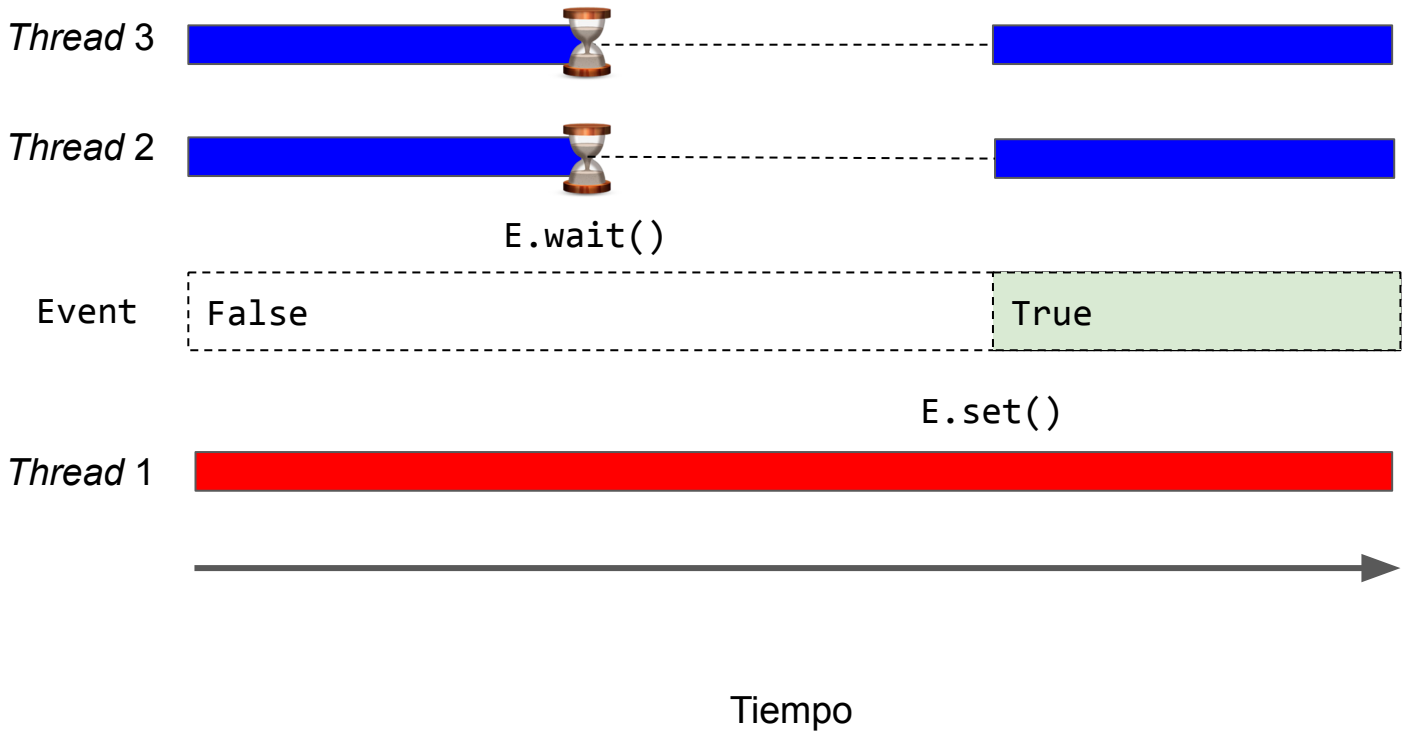
- `lock()`
- `set()`
- `wait()`

- Operación atómica

# Lock()


R.acquire()
R.release()
Thread 2
Resource
Thread 1
R.acquire()
R.relase()
Tiempo

# Event: set()  wait()

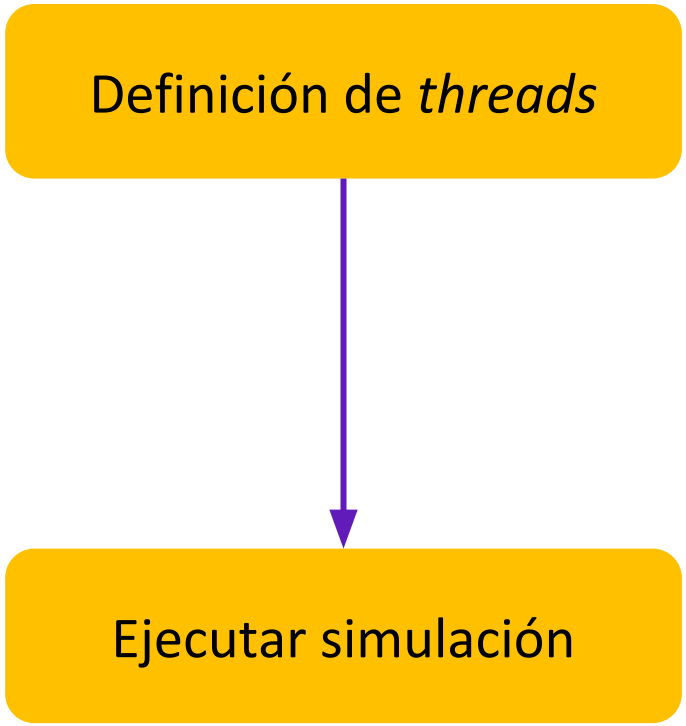


# Actividad

1. En el *syllabus*, vayan a la carpeta “Actividades” y descarguen el enunciado de la actividad 4 (AC04) https://github.com/IIC2233/syllabus
2. Trabajen **individualmente** hasta las **16:30**.
3. Recuerden hacer *commit* y *push* cada cierto tiempo.

# Cierre

# Diagrama de flujo de AC04




```
class Excavador(Thread)
class Imprimidor(Thread)
```

```
def comenzar()
def estadisticas()
```

# *Threading*

¿Se podría haber hecho la AC sin el uso de *threads*?

*Threads* proveen una interfaz y forma única de modelar situaciones de ejecución concurrente.

Simular este comportamiento con programas *single-threaded* es mucho más difícil.

# Threads

```
# main.py
...
for excavador in self.excavadores:
    excavador.start()
for imprimidor in self.imprimidores:
    imprimidor.start()
...
```

```
# excavadores.py
class Excavador(Thread):
    ...

    def run(self):
        # comportamiento
```

```
# imprimidores.py
class Imprimidor(Thread):
    ...

    def run(self):
        # comportamiento
```

# Threads

```
# excavadores.py
class Excavador(Thread):

    def __init__(self, nombre, berlin, tunel):
        super().__init__() # importante
        self.nombre = nombre
        self.berlin = berlin
        self.tunel = tunel

    def run(self):
        while self.tunel.metros_avanzados < self.tunel.largo:
            reloj(10) # demora 10 minutos
            self.avanzar(randint(50, 100))
            if uniform(0, 1) <= 0.1:
                self.problema_picota()
```

# Locks

```
# excavadores.py
class Excavador(Thread):

    def problema_picota(self):
        with self.berlin:
            print(“Berlin llegó a ayudarme”)
            reloj(5)
            print(“Berlin solucionó el problema”)

    excavar_lock = Lock()

    def avanzar(self, metros):
        with self.excavar_lock:
            self.tunel.metros_avanzados += metros
            print(f“Cavé {metros}”)
            ...
```

# Señales

```
# main.py
... # threads iniciados
self.tunel.tunel_listo.wait()
self.bolsa_de_dinero.dinero_listo.wait()
...
```

# Señales

```
# excavadores.py
def avanzar(self, metros):
    ...
    # Si se completa el tunel
    if self.tunel.metros_avanzados >= self.tunel.largo:
        self.tunel.tunel_listo.set()


# imprimidores.py
def imprimir_dinero(self, dinero):
    ...
    # Si se completa el monto
    if self.bolsa.dinero_acumulado >= self.bolsa.meta:
        self.bolsa.dinero_listo.set()
```