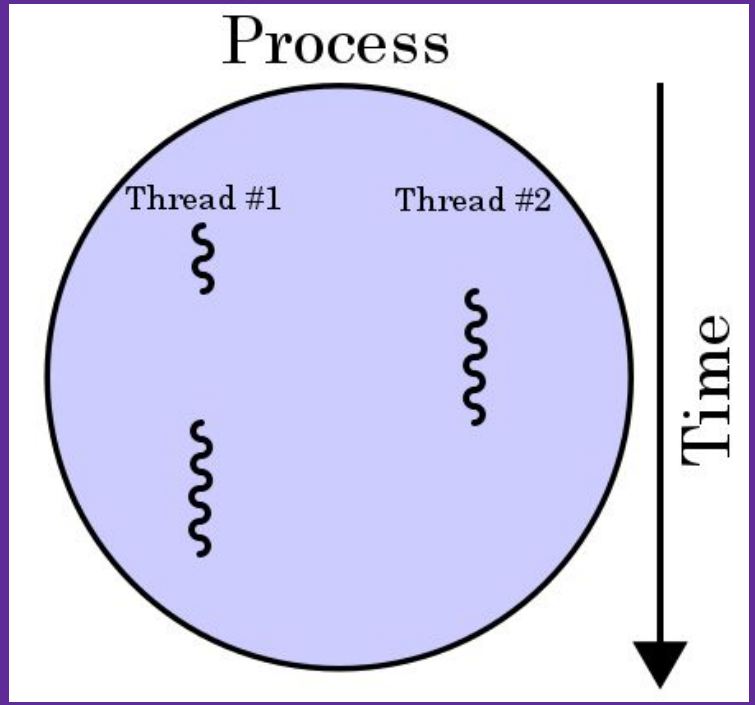
Programación Avanzada
IIC2233 2024-1

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Dante Pinto - Francisca Cattan

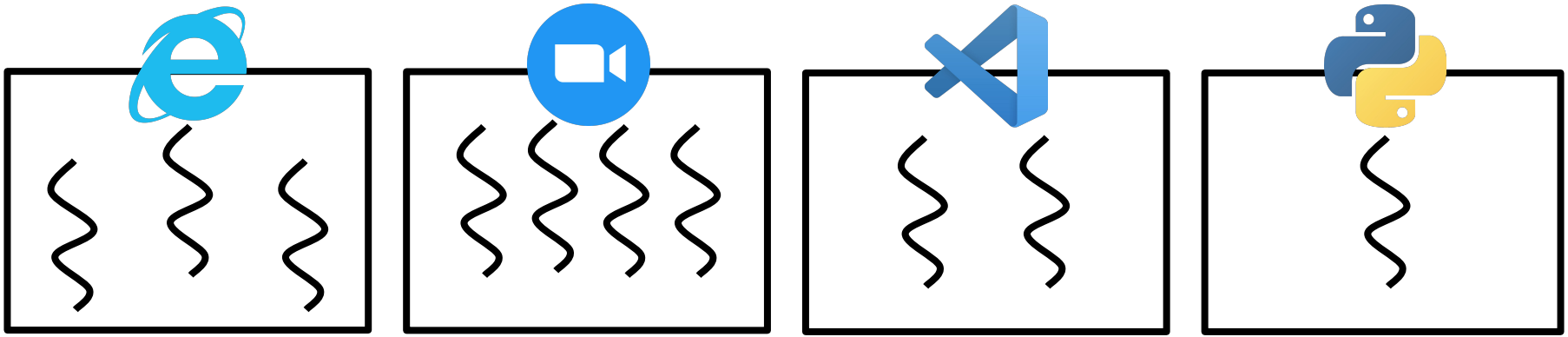
Anuncios

1. Recuerden responder la ECA. Pueden hacerlo desde el domingo hasta el martes.
2. Está disponible la ETC.
3. Hoy tenemos la segunda actividad.
4. Mañana es el *midterm*, empezará a las 17:30.

Paralelismo



Paralelismo: Procesos y *Threads*



DCCommits

Parte A

DCCommits (Parte A)

Necesitamos notificar los *commits* a corregir, pero solo podemos notificar de a uno a la vez.

¿Cómo podemos mejorar esto? Queremos que se publiquen *commits* de las 5 secciones simultáneamente.

```
class Commits:
    def __init__(self, commits):
        ...

    def notificar_commit(self, commit):
        self.subiendo = True
        time.sleep(10)
        Commits.publicados += int(1)
        self.subiendo = False

    def publicar(self):
        for commit in self.commits:
            self.notificar_commit(commit)

c = Commits(commits)
c.publicar()
```

DCCommits (Parte A)

Necesitamos notificar los *commits* a corregir, pero solo podemos notificar de a uno a la vez.

¿Cómo po
que se pub
secciones

¡Necesitamos *Threads*!

```
class Commits:
    def __init__(self, commits):
        ...

    def notificar_commit(self, commit):


                                    nt(1)


        for commit in self.commits:
            self.notificar_commit(commit)

c = Commits(commits)
c.publicar()
```

DCCommits (Parte A)

Necesitamos notificar los *commits* a
corregir, pero solo podemos n
uno a la vez.

¿Cómo po
que se pub
secciones

¡Ne ds!


```
class Commits:
    nit__(self, commits):

    ificar_commit(self, commit):

    nt(1)

    commit in self.commits:
    self.notificar_commit(commit)

    (commits)
c.publicar()
```

Threads

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

- *Thread* principal
- Otros *threads*

Threads

```python
from threading import Thread


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




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

```python
from threading import Thread

# ¡Importante heredar!
class MiThread(Thread):
    def __init__(self, *args, **kwargs):
        # ¡Importante el super!
        super().__init__(*args, **kwargs)

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


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

Threads

Con el uso de *threads* ahora podemos cambiar nuestro código de DCCommits por uno que permita la concurrencia.

Cada *Thread* se encargará de recopilar los *commits* de una sección y de aumentar en 1 el contador global "`Commits.publicados`". Así sabremos cuantos *commits* llevamos ya publicados en cada momento.

DCCommits

Parte B

DCCommits (Parte B)

Ahora cada *commit* se sube de manera independiente entre las 5 secciones.

Pero … **la cantidad total de *commits* publicados no calza con la cantidad real de *commits*** 😨

```
- C1 lee 0 de Commits.publicados
- C1 se pausa

…

- C2 lee 5 de Commits.publicados
- C2 suma 1 => 6
- C2 guarda 6 en Commits.publicados
- C2 se pausa


- C1 se reanuda
- C1 suma 1 => 1 (😨)
- C1 guarda 1 en Commits.publicados (😨)
```

DCCommits (Parte B)

Ahora cada *commit* se sube de manera independiente entre las 5 secciones.

Pero … **la**
publicado
de *commi*

¡Necesitamos sincronizar los accesos a este contador!

```
- C1 lee 0 de Commits.publicados
- C1 se pausa
                                    s
                                    ados
- C1 se reanuda
- C1 suma 1 => 1 (😨)
- C1 guarda 1 en Commits.publicados (😨)
```

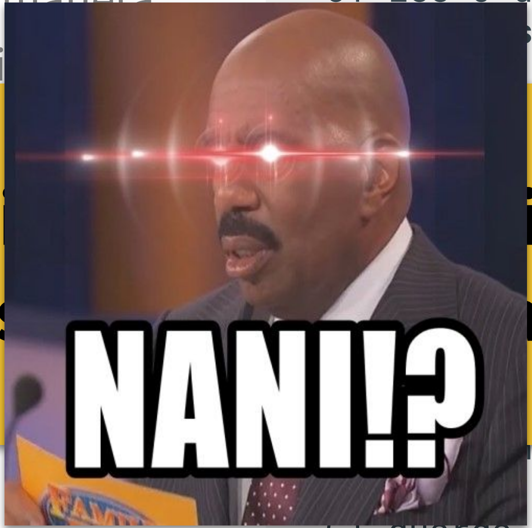
DCCommits (Parte B)

Ahora cada *commit* se sube de manera independiente entre las 5 secci

Pero … **la**
publicado
de *commit*

```
- C1 lee 0 de Commits.publicados
                a
                                   s
                                   ados
         uda
         => 1 (😨)
- C1 guarda 1 en Commits.publicados (😨)
```

¡Necesi izar los acces dor!

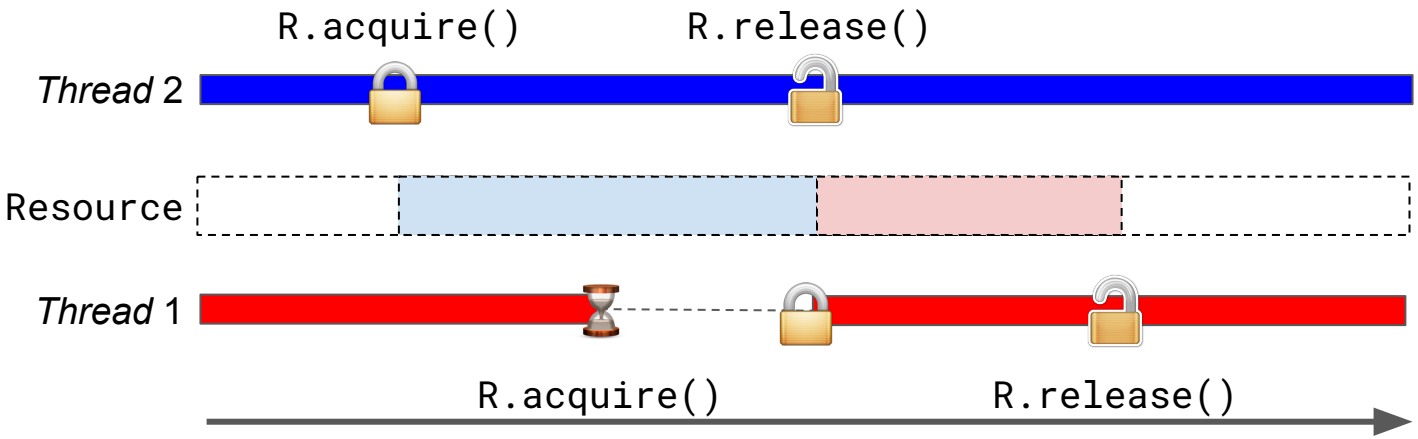


Sincronización de recursos

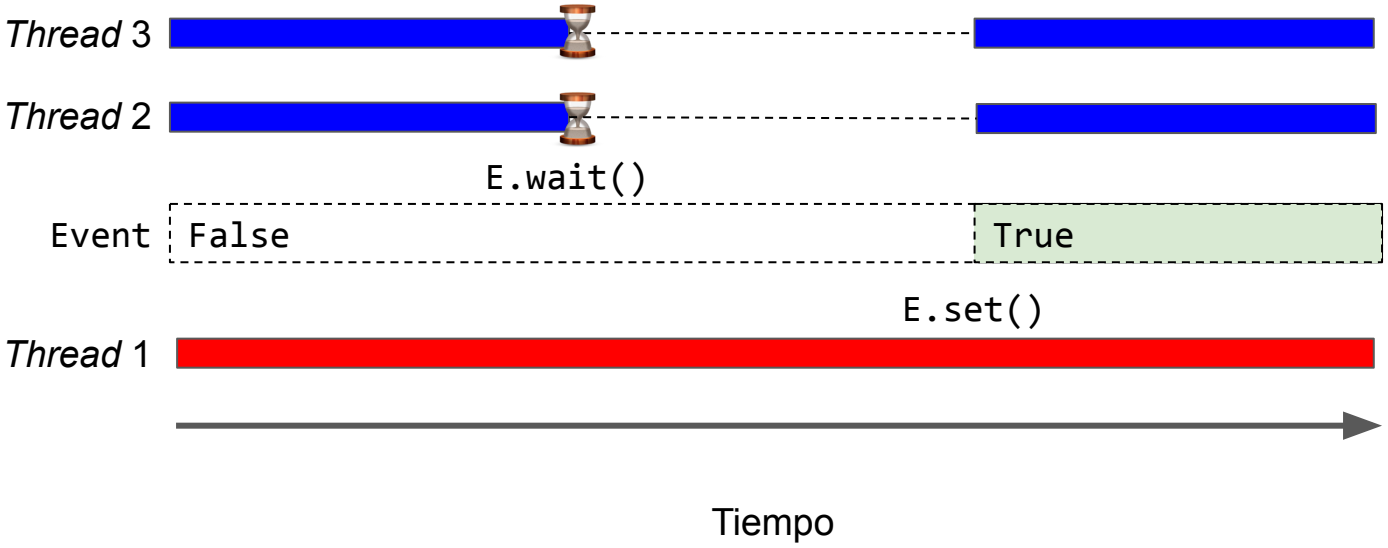
- `lock()`
- `set()`
- `wait()`

- Operación atómica

Lock()


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

Event: set() - wait()



Operaciones atómicas

También llamadas operaciones ***thread-safe*** son acciones en Python que no pueden ser interrumpidas a la mitad.

- Asignar valores
  - `x = 1`
- Obtener dato de una lista
  - `lista[2]`
- Agregar dato de una lista
  - `lista.append(2)`

Mientras que hay operaciones que no son ***thread-safe***.

- Realizar más de una acción al mismo tiempo:
  - `lista.append(lista[0])`
  - `x = x + 1`**

**Desde Python 3.10, esta operación fue optimizada, pero no en todos los lenguajes o versiones es así.

`Lock()`

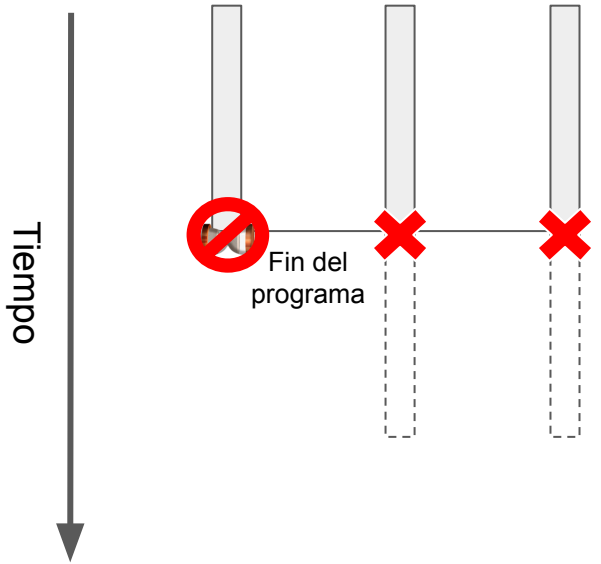
En este caso, con el uso de *locks* podemos volver a modificar nuestro código de `DCCommits` para asegurar que solo 1 *thread* a la vez modifique el contar global "`Commits.publicados`".

DCCommits

Parte C

DCCommits (Parte C)

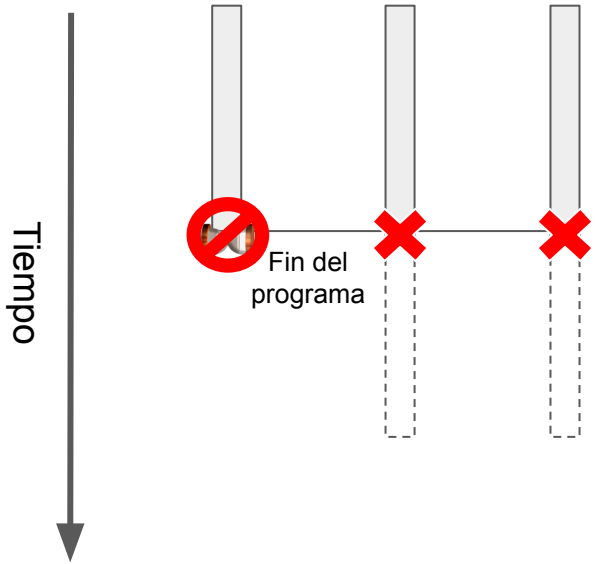
El sindicato nos pide trabajar 1 hora diaria, así que solo publicaremos *commits* durante ese tiempo y luego hay que bajar el sistema.



DCCommits (Parte C)

El sindicato nos pide trabajar 1 hora diaria, así que solo publicaremos *commits* durante ese tiempo y luego hay que bajar el sistema.

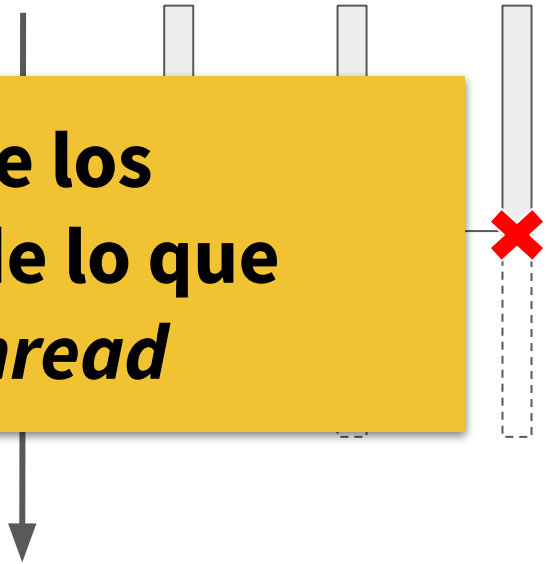
Necesitamos **un mecanismo para detener la publicación de *commits* que no alcanzaron a subirse luego de 1 hora.**



DCCommits (Parte C)

El sindicato nos pide trabajar 1 hora diaria, así que solo publicaremos *commits* durante es
el sistema.

Necesitam
la publica
alcanzaron a subirse luego de 1 hora.



Necesitamos que los *threads* dependan de lo que ocurre en otro *thread*

DCCommits (Parte C)

El sindicato nos pide trabajar 1 hora diaria,
así que solo publicaremos *comm*
durante es
el sistema.

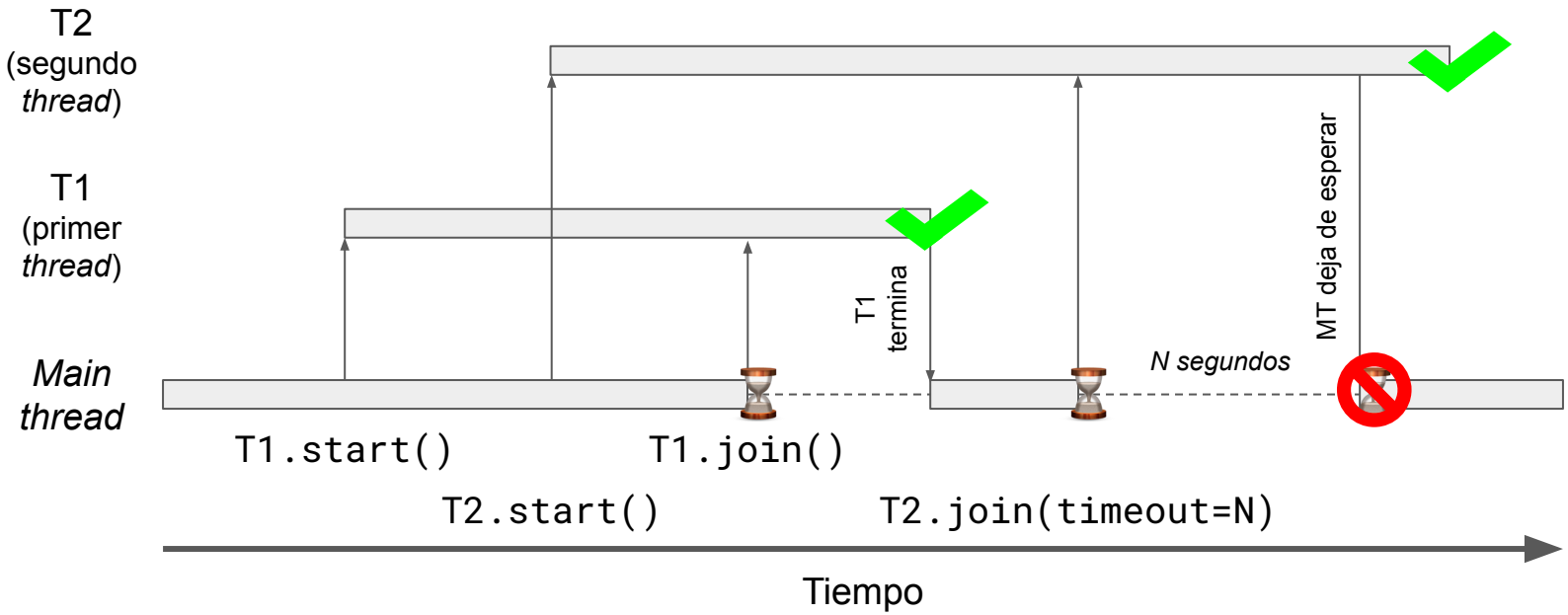
Necesitam
la publica
alcanzaron a subirse luego de 1

Nec **los**
threads **e lo que**
ocu ***read***

Dependencia entre *threads*

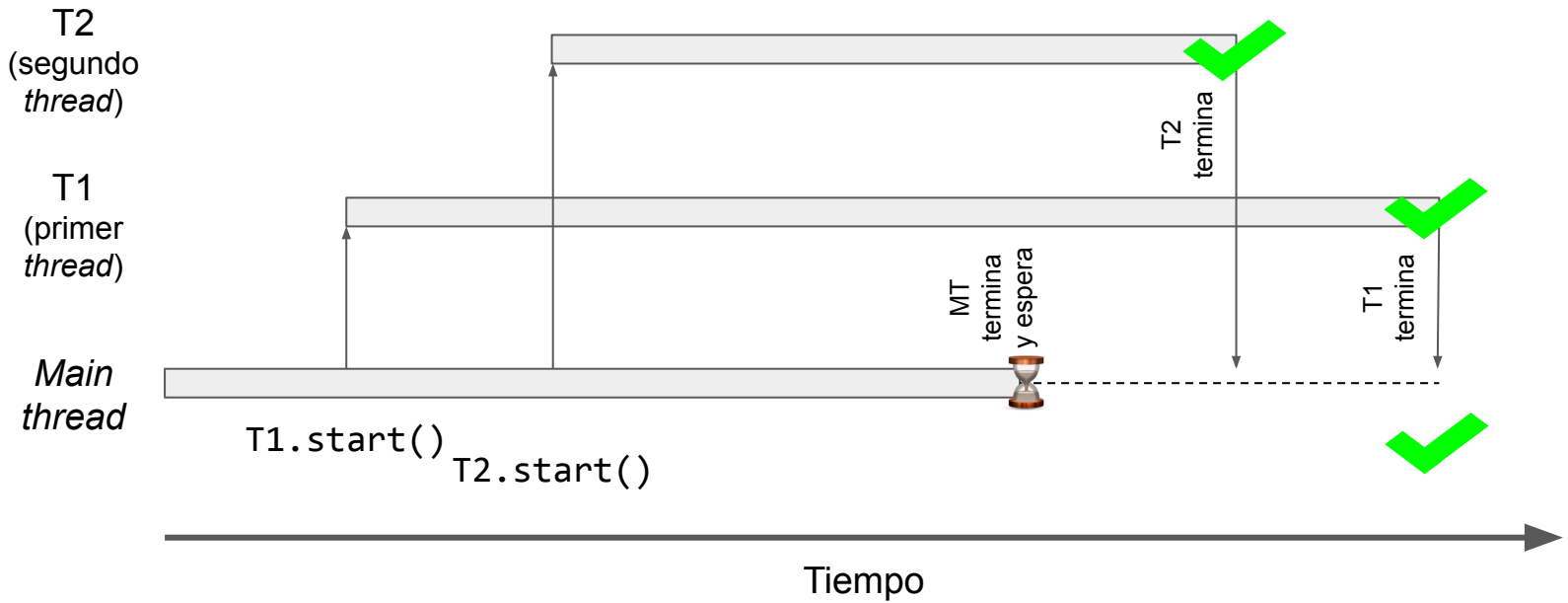
- `join()`
- `daemon`

join()

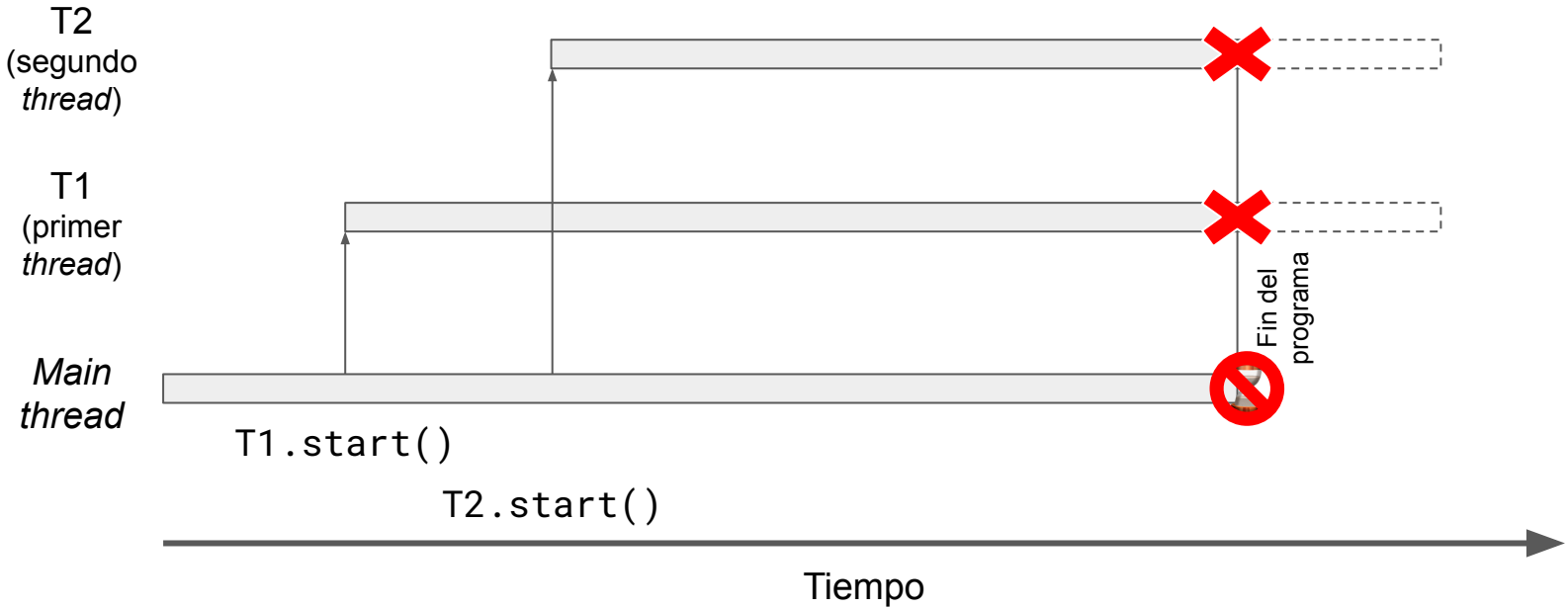

T2
(segundo
thread)
T1
(primer
thread)
Main
thread
T1
termina
MT deja de esperar
N segundos
T1.start()
T1.join()
T2.start()
T2.join(timeout=N)
Tiempo

daemon=False

Comportamiento por **defecto**



daemon=True


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

daemon

En este caso, con el uso de ***daemon=True*** podemos volver a modificar nuestro código de **DCCommits** y una vez que el programa principal espere 1 hora, se para y todos los *threads* dejan de ejecutarse.

Recapitulación

- Con *Thread* podemos asegurar concurrencia.
- Con *Lock* y eventos, podemos detener los *threads* para que esperen cierta acción o para asegurar que solo 1 *thread* ejecute a la vez cierto código.
- Con *join* y *daemon*, podemos permitir que la ejecución de un thread dependa de otro *thread* o del programa principal.

Programación Avanzada
IIC2233 2024-1

Hernán Valdivieso - Daniela Concha - - Dante Pinto - Francisca Cattan

Comentarios AC2