Python

Programación Concurrente

Rogelio Ferreira Escutia

Profesor / Investigador
Tecnológico Nacional de México
Campus Morelia

Técnicas para Programación Concurrente

Técnicas Clásicas

- **Serial.**

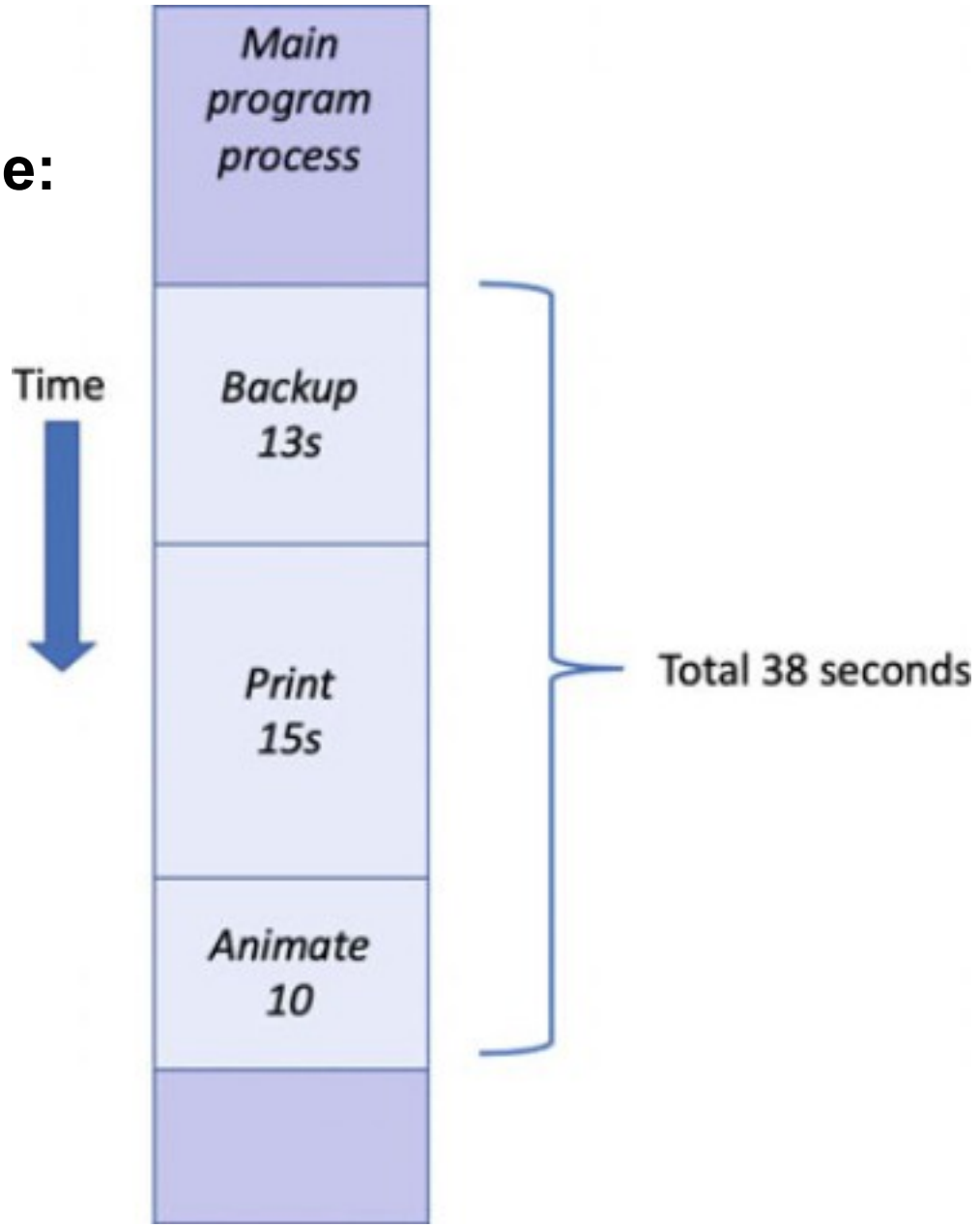
Técnicas Concurrentes

- **Hilos.**
- **Procesos.**
- **Paralelismo.**
- **Distribución.**

Programación Serial

Serial

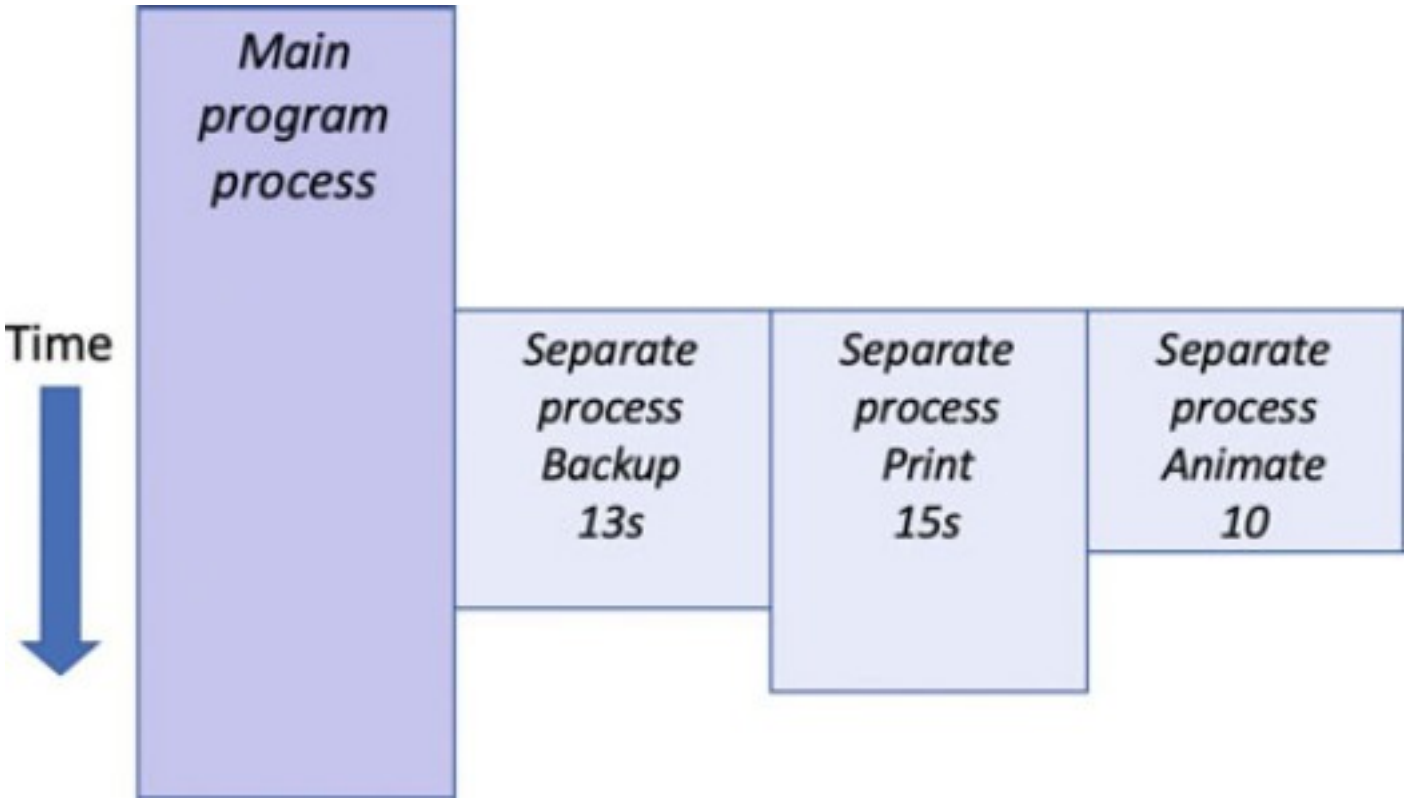
- **Ejecución de tareas en serie:**



Programación Concurrente

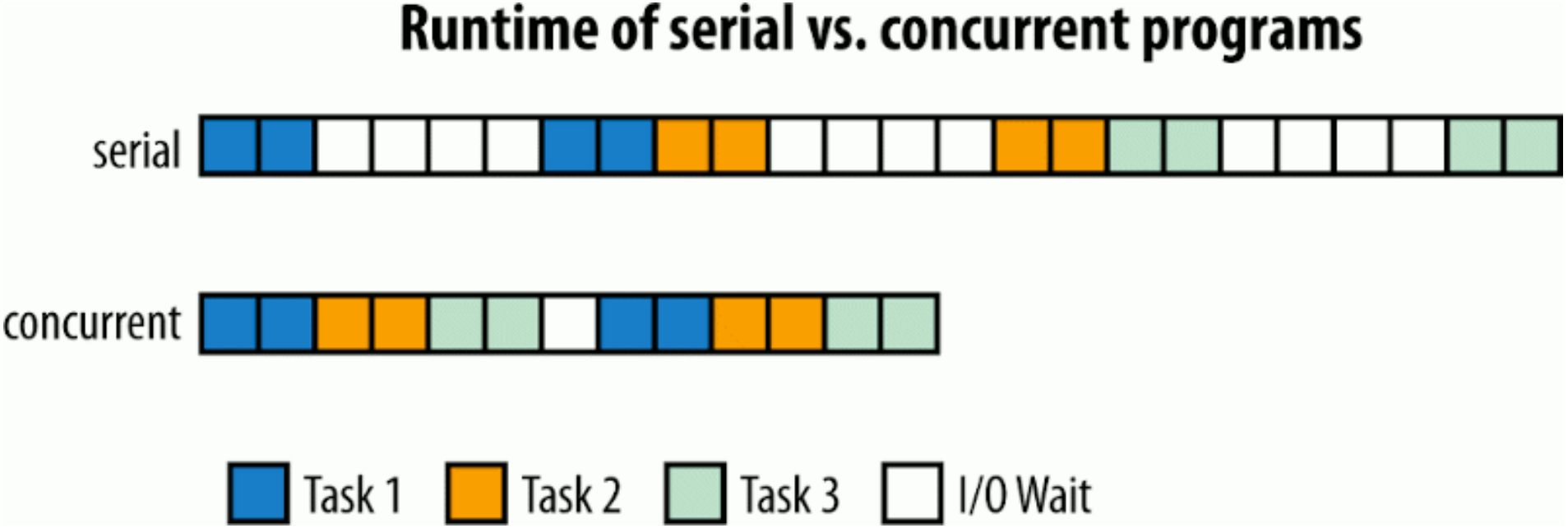
Concurrente

- **Ejecución de tareas de manera concurrente:**



Serial vs. Concurrente

- **Comparación en tiempo:**



Programación con Hilos

Hilos - características

- **Son parte de un proceso en ejecución.**
- **Se les conoce como procesos ligeros, ya que no tienen su propio direccionamiento de memoria.**
- **Pueden ser interrumpidos en su ejecución.**
- **Los hilos pueden compartir datos entre ellos de manera directa.**
- **La mayoría de los sistemas operativos no los considera como un entidad separada.**

Pasos para programar hilos

- **Importar biblioteca (thread).**
- **Definir el código que ejecutará el hilo (funciones)**
- **Crear nuestro hilo (instanciar objeto de tipo thread) y definir su código de ejecución “target”).**
- **Arrancar el hilo (start).**

Hilos

- **“Hola Mundo!” utilizando un hilo:**

```
from threading import Thread                    # Librería para hilos

def funcion_hilo():                             # Función asignada al hilo
    print("\nHola Crayola desde un hilo!!!")    # "Hola Mundo" desde el hilo

print("\nInicio del programa principal")
hilo_uno = Thread(target=funcion_hilo)          # Creación del hilo
hilo_uno.start()                                # Arranque del hilo
```

Hilos

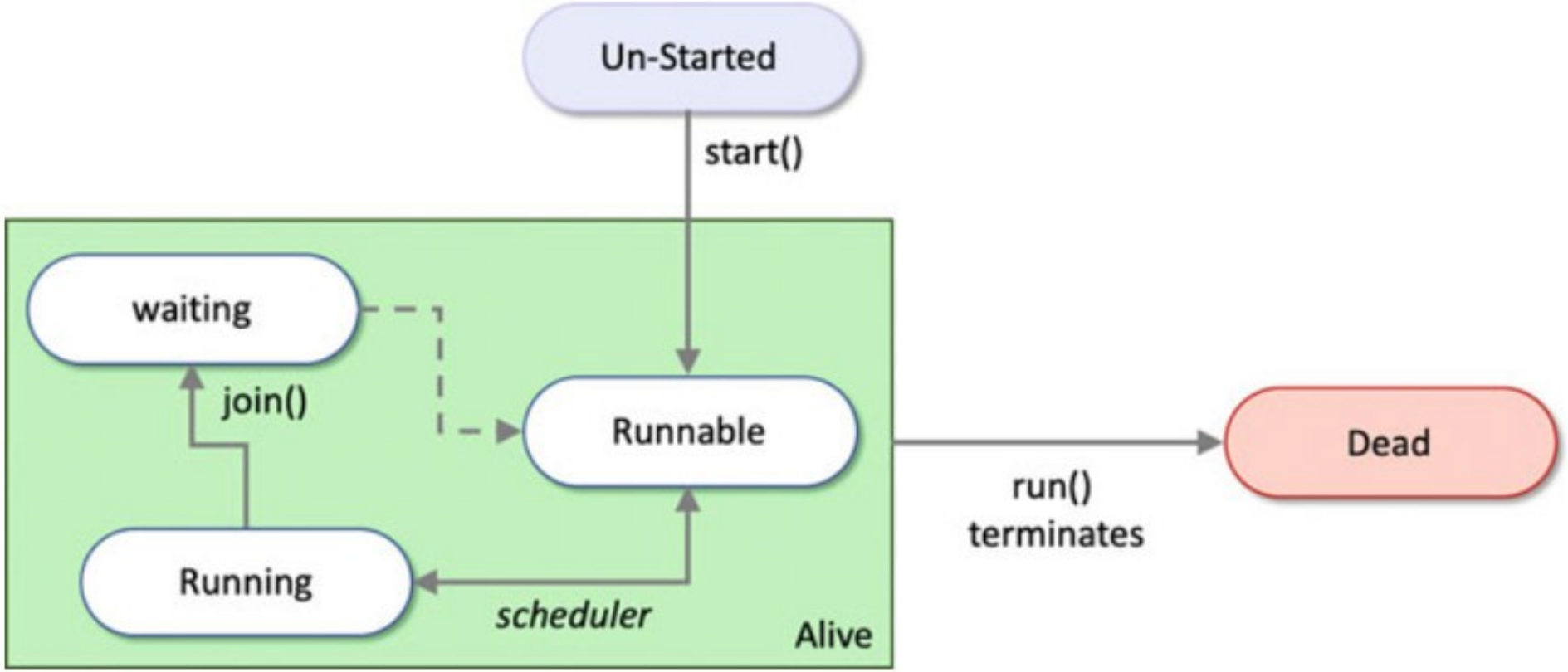
- **“Hola Mundo!” utilizando 3 hilos:**

```
from threading import Thread                    # Librería para hilos

def funcion_hilo(numero):                       # Función asignada a cada hilo
    print("Hola Crayola desde el hilo: " + str(numero))

print("\nInicio del programa principal\n")
hilo_uno = Thread(target=funcion_hilo(1))       # Crear el hilo 1
hilo_dos = Thread(target=funcion_hilo(2))       # Crear el hilo 2
hilo_tres = Thread(target=funcion_hilo(3))      # Crear el hilo 3
hilo_uno.start()                                # Arranque del hilo 1
hilo_dos.start()                                # Arranque del hilo 2
hilo_tres.start()                               # Arranque del hilo 3
```

Estados de un hilo

- **Un-Started: Ya está definido el hilo pero no está en ejecución.**
- **Runnable: Cuando se arrancó (start) y está en espera de ser llamado por el scheduler del sistema operativo.**
- **Running: Hilo en ejecución.**
- **Waiting: Cuando está en espera de que otro hilo termine su ejecución.**
- **Dead: Cuando la ejecución del hilo ha terminado.**

Estados de un hilo



Variables globales

- **Los variables que usa cada hilo, son variables locales, es decir, otro hilo no puede conocer esa variable ni su contenido.**

- **Para usar una variable como “global” es necesario declarar esa variable de manera normal pero dentro del programa principal, y al inicio de cada hilo que quiera accesar a la variable global tenemos que poner el prefijo “global” y luego el nombre de la variable.**

- **De esta manera un hilo puede accesar y modificar una variable global.**

Variables globales

```
from threading import Thread                    # Librería para hilos
suma = 0                                        # Variable que será usada como global

def funcion_hilo():                             # Función asignada al hilo
    print("\nIniciando hilo")                   # "Hola Mundo" desde el hilo
    global suma                                 # Manejar una variable global
    suma = suma + 5                             # Modificando la variable global
    print("Variable global modificada")

print("\nInicio del programa principal")
print("Valor Inicial: " + str(suma))
hilo_uno = Thread(target=funcion_hilo)          # Creación del hilo
hilo_uno.start()                                # Arranque del hilo
hilo_uno.join()                                 # Esperando a que termine el hilo
print("\nValor Final: " + str(suma))
```

Programación con Procesos

Procesos - características

- **Son ejecuciones de código independientes.**
- **Tienen su propio direccionamiento de memoria.**
- **Son independientes unos de otros.**
- **Los procesos no comparten datos entre ellos de manera directa.**

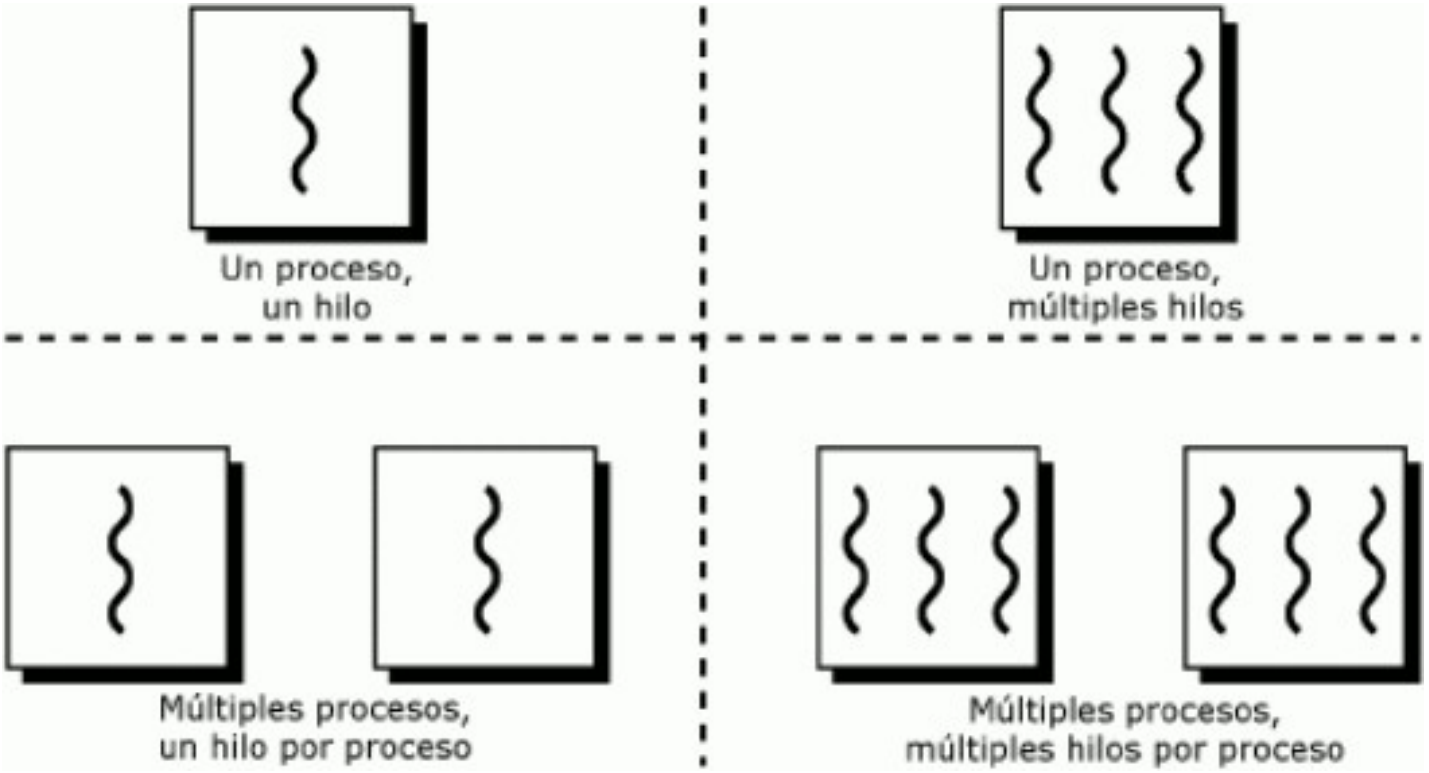
Procesamiento

- **Cola de de tareas en un procesador:**

"Programación concurrente", http://ferestrepoca.github.io/paradigmas-de-programacion/progconcurrente/concurrente_teoria/index.html, abril 2020

Hilos y Procesos

- **Diferentes formas de programar:**



Métodos usados en Procesos

- **start(): Arranca un proceso y sólo pueden ser invocados una vez por proceso.**

- **join([timeout]): Espera a que un proceso termine (si se ejecuta sin parámetros. Si en envía un “timeout” (opcional) entonces se espera la cantidad de segundos que se le indiquen.**

- **is_alive(): Sirve para indentificar si un proceso sigue en funcionamiento.**

Procesos – Hola Mundo (1)

- **Importar la librería para uso de procesos:**

```
from multiprocessing import Process
```

Procesos – Hola Mundo (2)

- **Definir el código (función) que ejecutará nuestro proceso quer recibirá como parámetro el número del proceso:**

```
def funcion_proceso(numero):
    print("\nHola Crayola!!!")
    print("Desde el proceso: " + str(numero))
```

Procesos – Hola Mundo (3)

- **Definir el inicio del código del programa principal.**
- **Esto es un requisito para el uso de procesos.**
- **Esto es importante si un programa va a ser llamado desde otro programa.**

```
if __name__ == '__main__':
```

Procesos – Hola Mundo (4)

- **Definir el código del programa principal.**
- **Crear el proceso.**
- **Arrancar el proceso.**

```
if __name__ == '__main__':                          # Definiendo del programa principal
    print("\nInicio del Programa Principal")
    numero = 1
    proceso_uno = Process(target=funcion_proceso, args=(numero,))   # Creación del proceso
    proceso_uno.start()                                             # Arranque del proceso
```

Procesos – Hola Mundo (5)

- **Programa completo:**

```
from multiprocessing import Process          # Librería para procesos

def funcion_proceso(numero):                 # Función asignada al proceso
    print("\nHola Crayola!!!")               # "Hola Mundo" desde el proceso
    print("Desde el proceso: " + str(numero))   # Imprime desde el proceso

if __name__ == '__main__':                   # Definiendo del programa principal
    print("\nInicio del Programa Principal")
    numero = 1
    proceso_uno = Process(target=funcion_proceso, args=(numero,))   # Creación del proceso
    proceso_uno.start()                                             # Arranque del proceso
```

Intercambio de datos en procesos

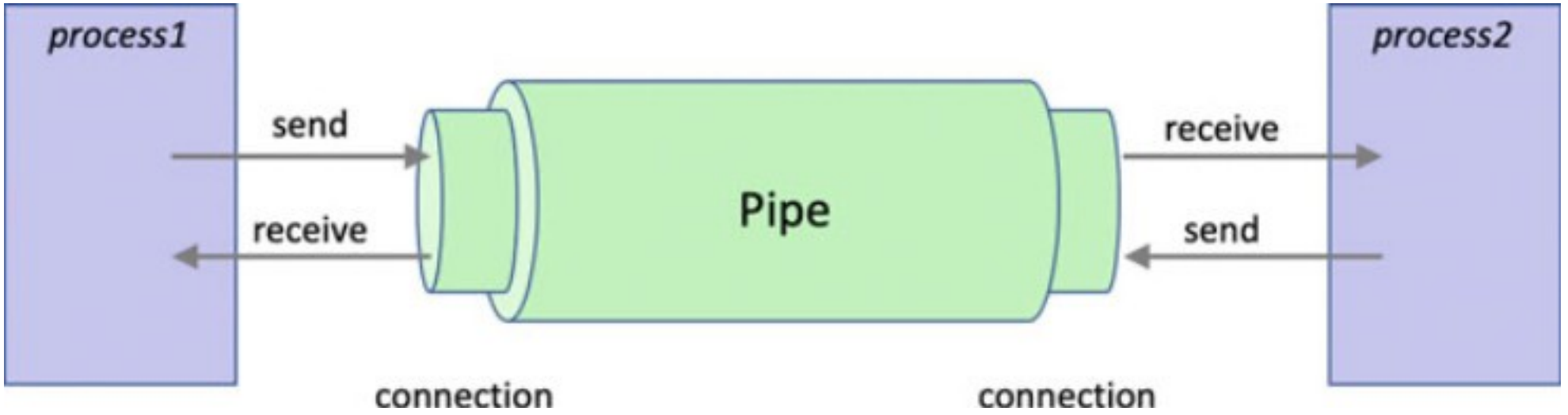
- **Los procesos son independientes entre sí y no comparten datos entre ellos de manera directa (no hay memoria compartida).**

- **Se requiere establecer una conexión entre procesos (pipe).**

- **Una vez establecida la conexión entre procesos, la comunicación es dúplex (bidireccional).**

Intercambio de datos en procesos

- **Conexión entre 2 procesos:**



Intercambio de datos en procesos

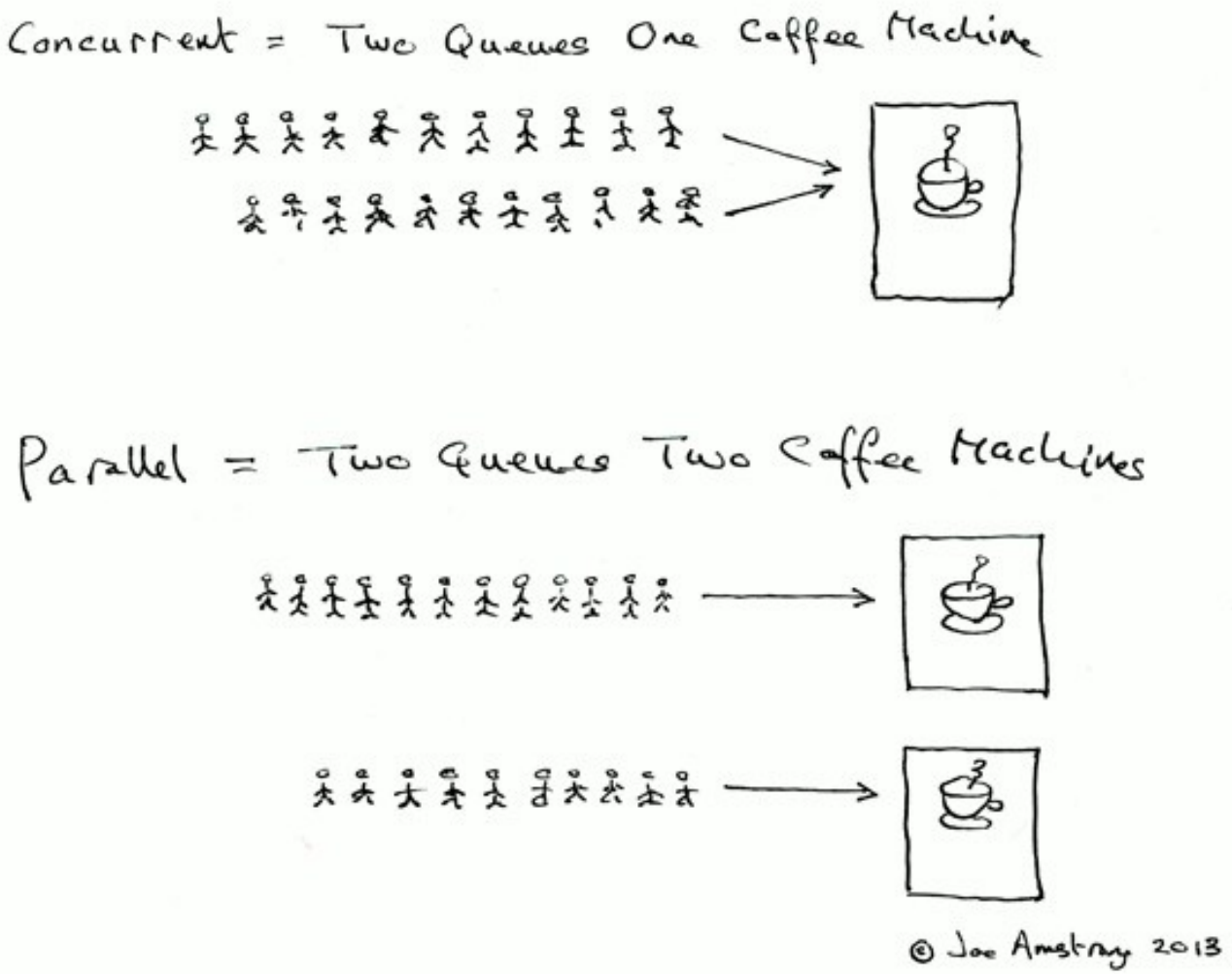
- **Conexión entre 2 procesos:**

```
from multiprocessing import Process, Pipe        # Libreria para procesos

def funcion_cliente(pipe_terminal, nombre):      # Funcion asignada al proceso
    print("\nProceso Cliente recibe: " + str(nombre))
    mensaje ="Bienvenido a la Matrix!"           # Mensaje a regresar
    pipe_terminal.send(mensaje)
    pipe_terminal.close()

if __name__ == '__main__':                       # Definiendo del programa principal
    print("\nInicio del Proceso Servidor")
    nombre = "Rogelio"                           # Mensaje a enviar
    pipe_servidor, pipe_cliente = Pipe()
    proceso = Process(target=funcion_cliente, args=(pipe_cliente, nombre))
    proceso.start()
    print("\nProceso Servidor recibe del Proceso Cliente: " + pipe_servidor.recv())
    proceso.join()
```

Programación Paralela

Comparación

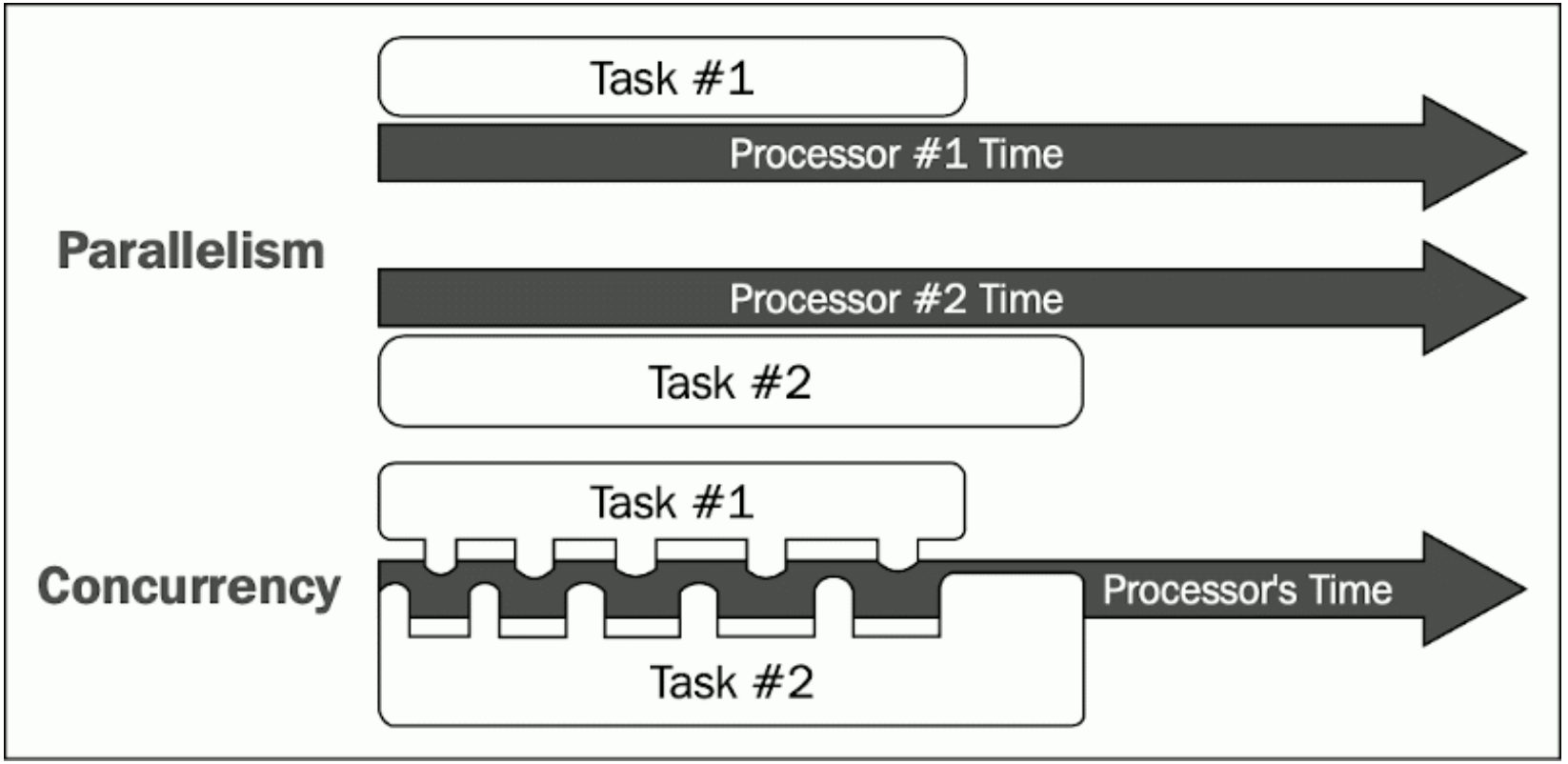
- **Concurrencia vs. Paralelismo:**



"Programación concurrente", http://ferestrepoca.github.io/paradigmas-de-programacion/progconcurrente/concurrente_teoria/index.html, abril 2020

Comparación

- **Concurrencia vs. Paralelismo:**

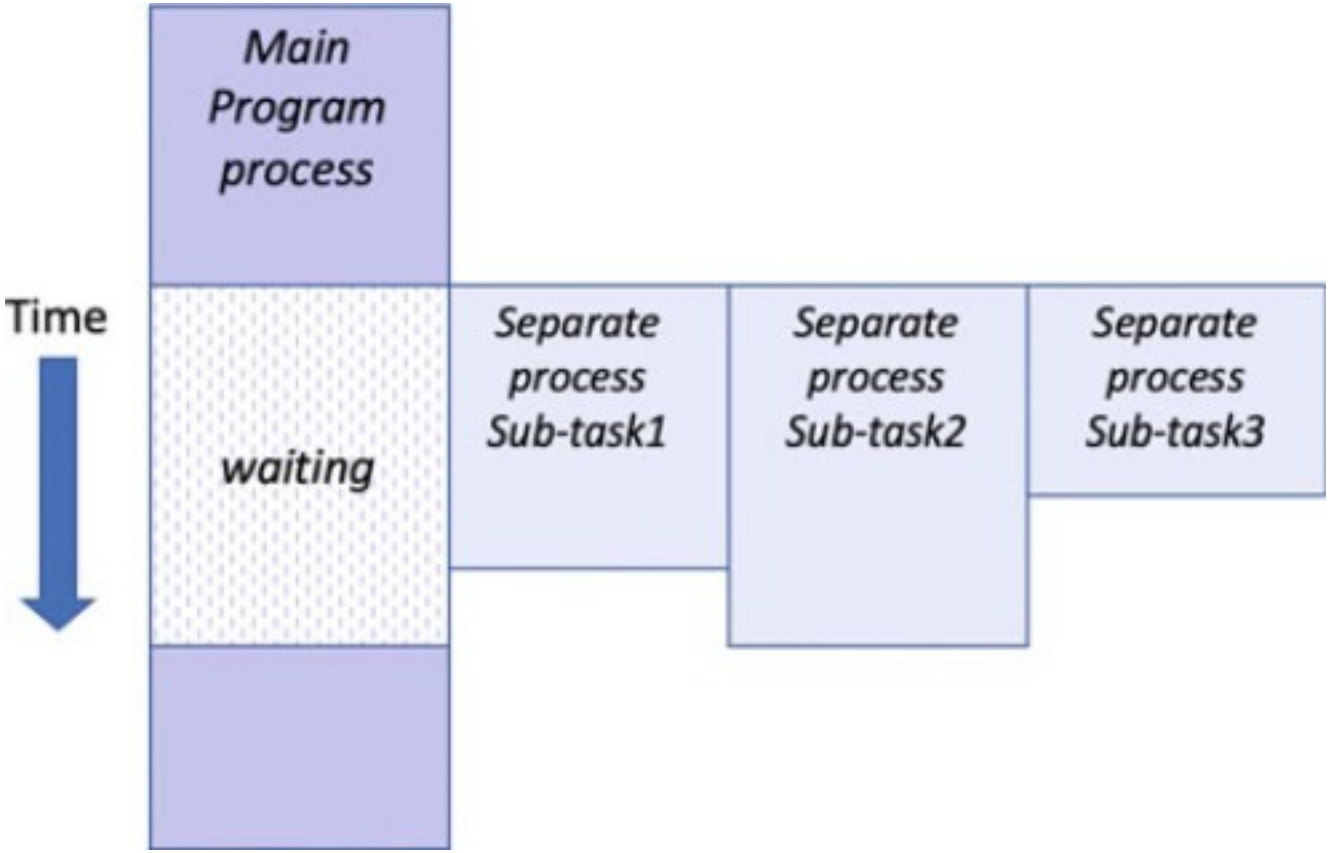


Tareas que se pueden paralelizar

- **Motores de búsqueda.**
- **Procesamiento de imágenes.**

Programación paralela

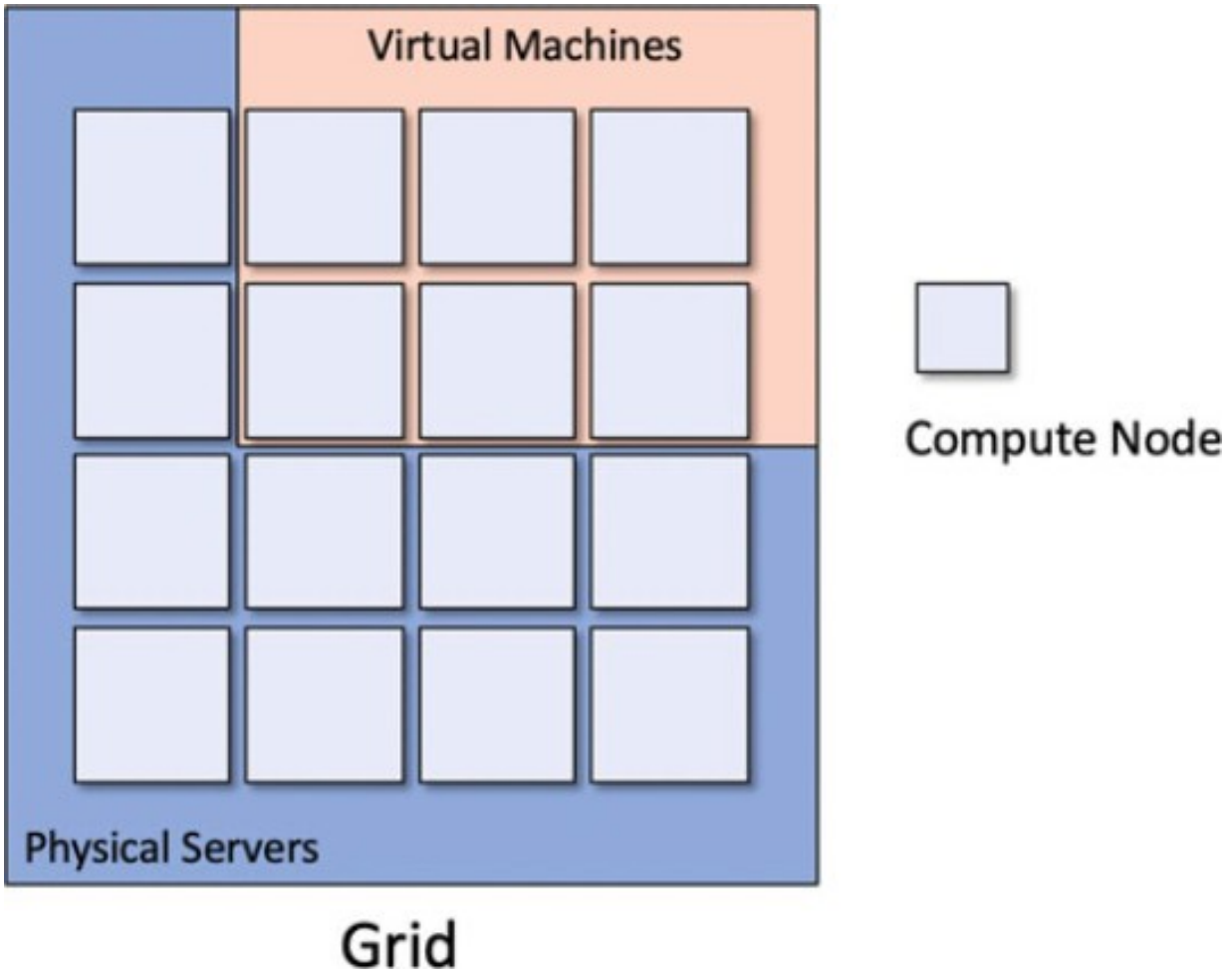
- **Una misma tarea dividida en pequeñas sub-tareas:**



Programación Distribuida

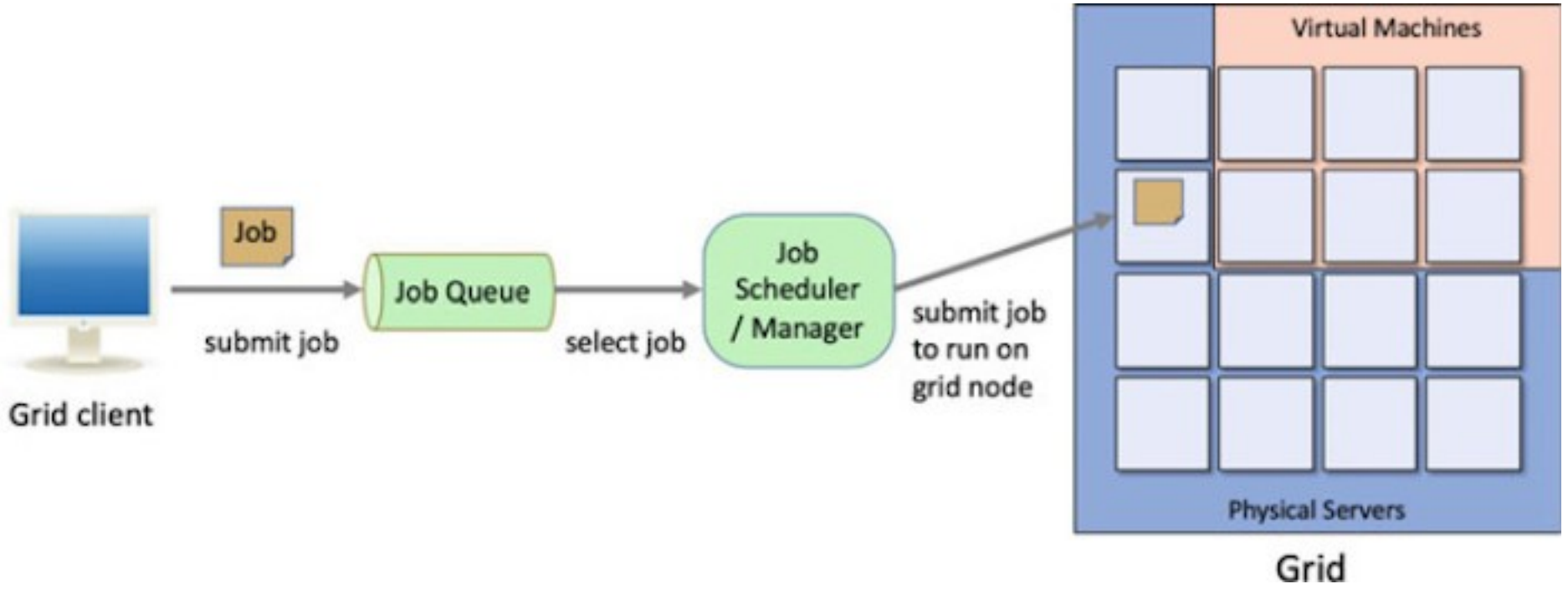
Programación Distribuida

- **Usando nodos reales y virtuales en una Grid Computing:**



Programación Distribuida

- **Un cliente envía una tarea a la Grid:**



Rogelio Ferreira Escutia

Profesor / Investigador
Tecnológico Nacional de México
Campus Morelia

rogelio.fe@morelia.tecnm.mx

rogeplus@gmail.com

xumarhu.net

@rogeplus

https://www.youtube.com/channel/UC0on88n3LwTKxJb8T09sGjg

rogelioferreiraescutia