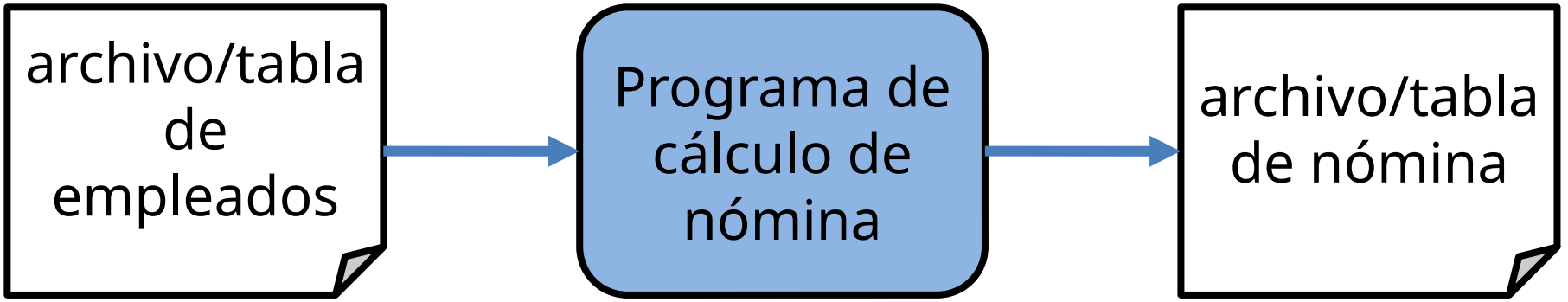
# Programación dirigida por eventos (Event Driven Programming)

Jaime A. Pavlich Mariscal

# Procesos batch

- Considere un programa “clásico” de cálculo de nómina



# Procesos batch

- Pseudocódigo

```
def calcular_nómina(empleado):
    ...


empleados_file = open("empleados", "r")
nómina_file = open("nómina", "w")

for empleado in empleados_file:
    nómina = calcular_nómina(empleado)
    nómina_file.write(empleado, nómina)

empleados_file.close()
nómina_file.close()
```

# Procesos batch

- El programador decide el flujo de control del programa
  - El orden en que se ejecutan los comandos
  - Cuándo se llama cada función
- Flujo de control
  - El programa lee las entradas
  - El programa procesa los datos
  - El programa genera una salida

# Procesos interactivos

- Flujo de control
  - El programa puede recibir datos en cualquier momento
    - Recibe los datos
    - Procesa los datos
    - Entrega un resultado
  - Los datos se reciben en un orden, a menudo, impredecible
- Ejemplo: Interfaz gráfica

# Procesos interactivos

- Pseudocódigo interfaz gráfica

```
while True:
    if usuario_hizo_click_en_botón_1():
        responda_a_click_en_botón_1()

    elif usuario_escribió_en_caja_de_texto_1():
        responda_a_cambio_en_caja_de_texto_1()

#   ...

    else:
        ignorar_o_lanzar_excepción()
```

# Procesos interactivos

- Pseudocódigo otro proceso interactivo

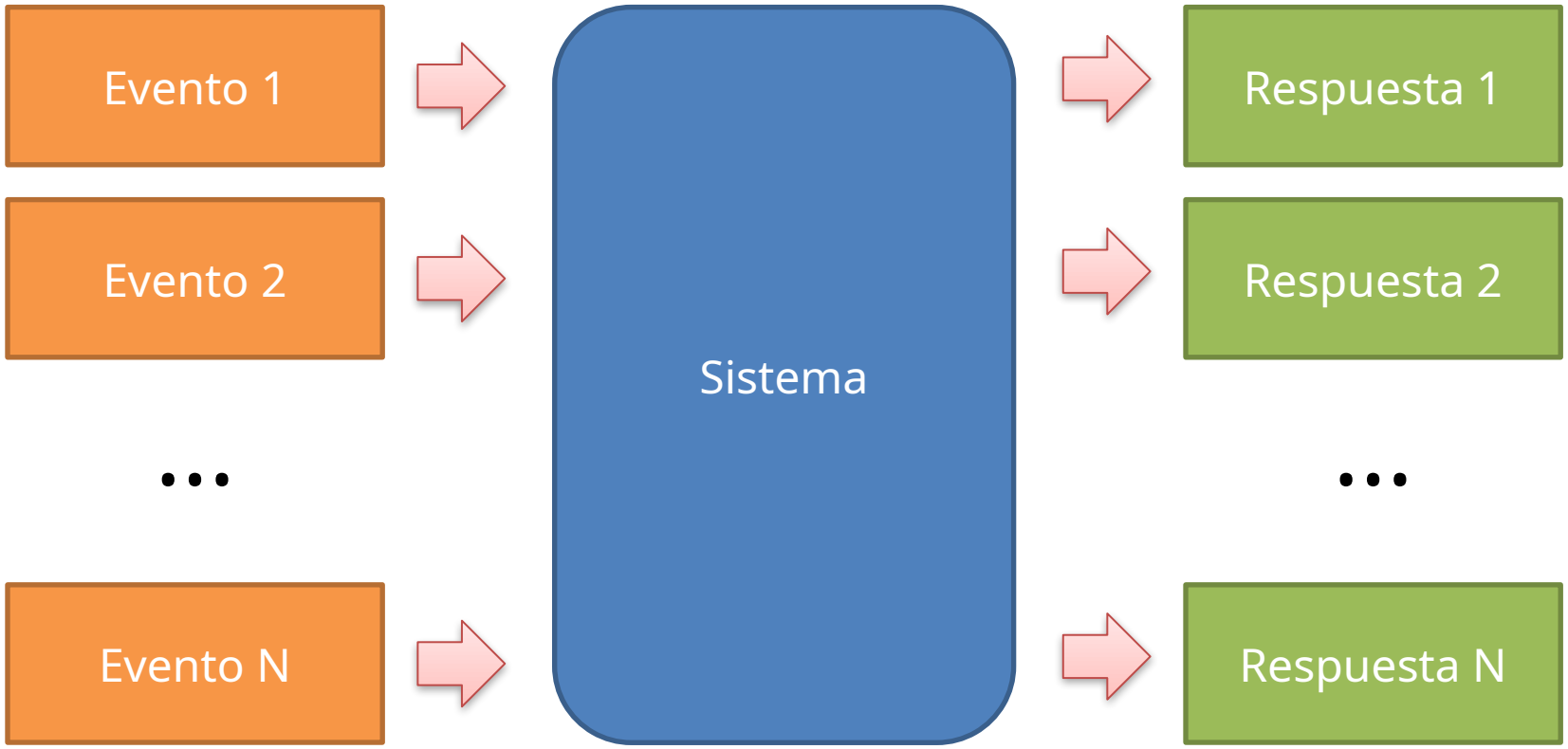
```
while True:
    if se_recibió__mensaje_desde_conexión_con_computador_1():
        responda_mensaje_recibido_de_computador_1()

    elif el_cliente_6_cumplió_2_años_de_afiliación_a_la_empresa():
        envie_felicitación_a_cliente()

    ...
```

# Procesos batch vs interactivos

- Entradas al principio, salidas al final
- El programador decide flujo de control
- El programador implementa programa paso a paso

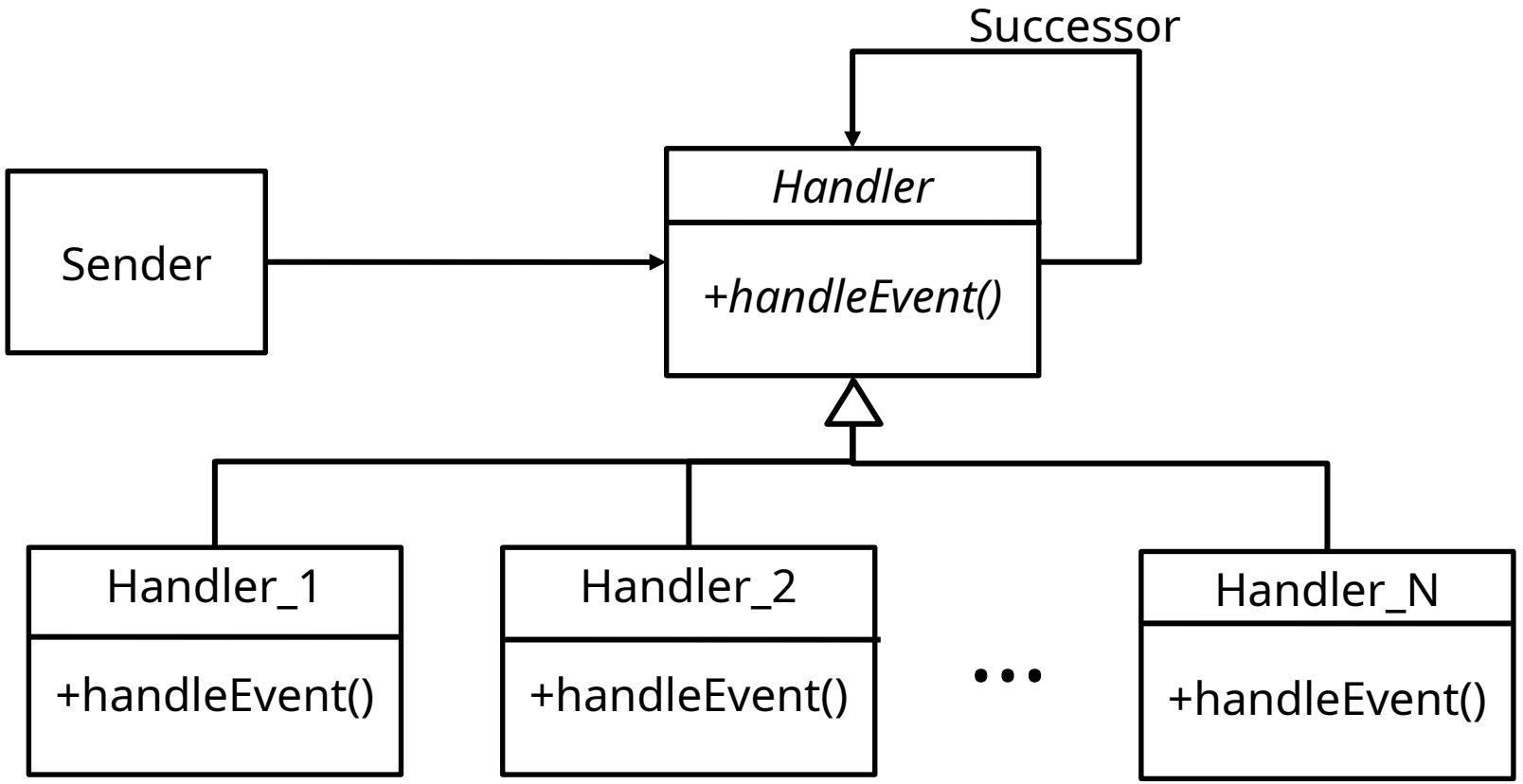
- Entradas y salidas en cualquier momento
- El programa decide flujo de control
- El programador implementa solo las respuestas a los datos recibidos

# Programación dirigida por eventos (Event Driven Programming)

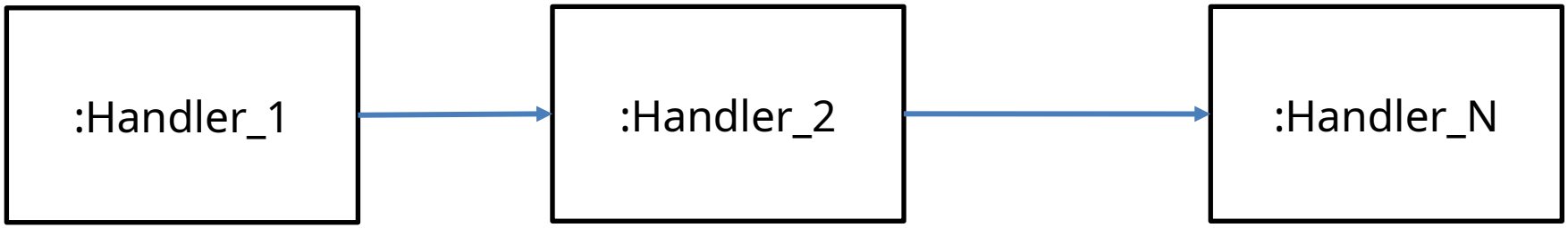


¿Cómo detectar los eventos?
¿Cómo responder a los eventos?
¿Cómo manejar el estado del sistema?

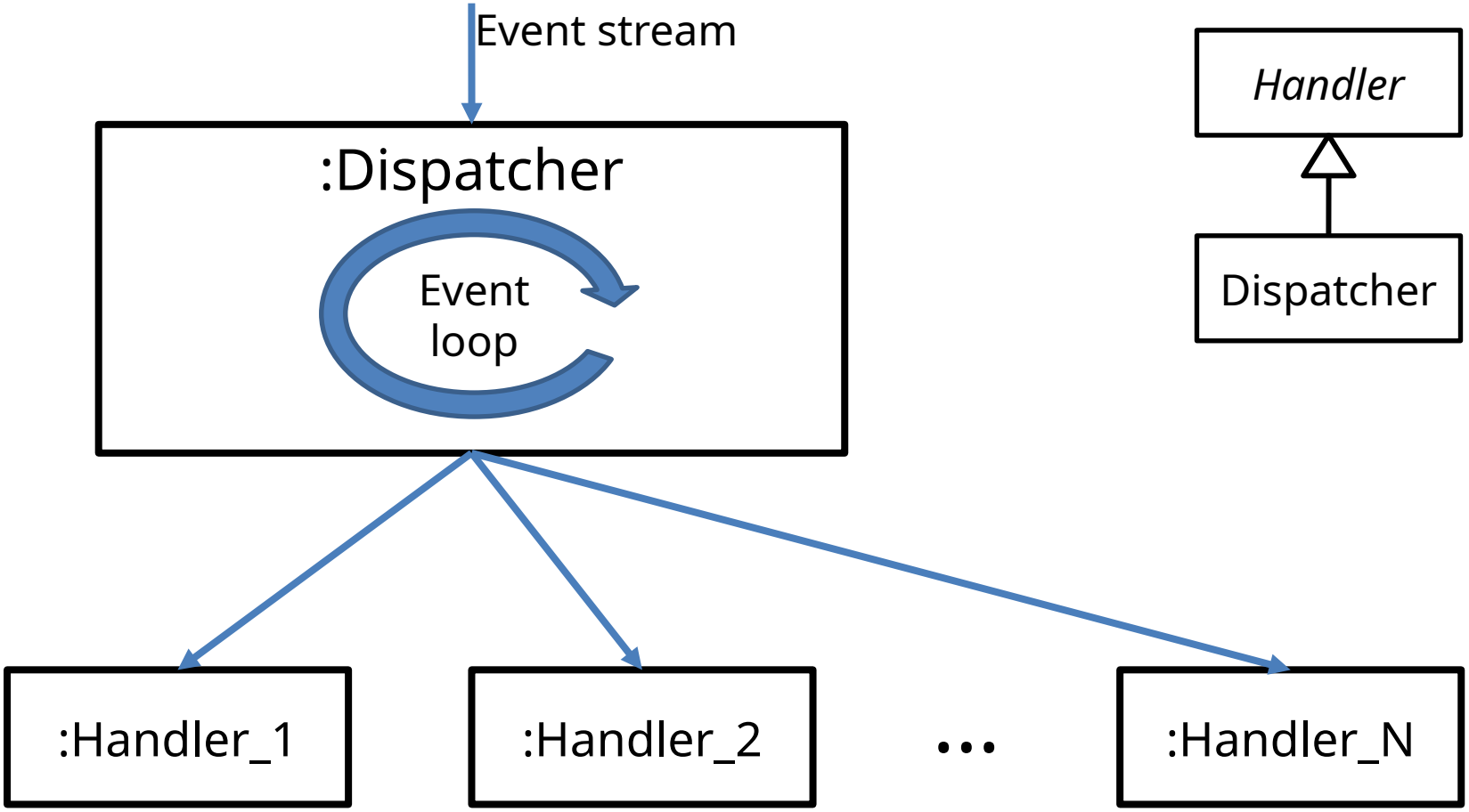
# Chain of Responsibility Pattern (CoR)



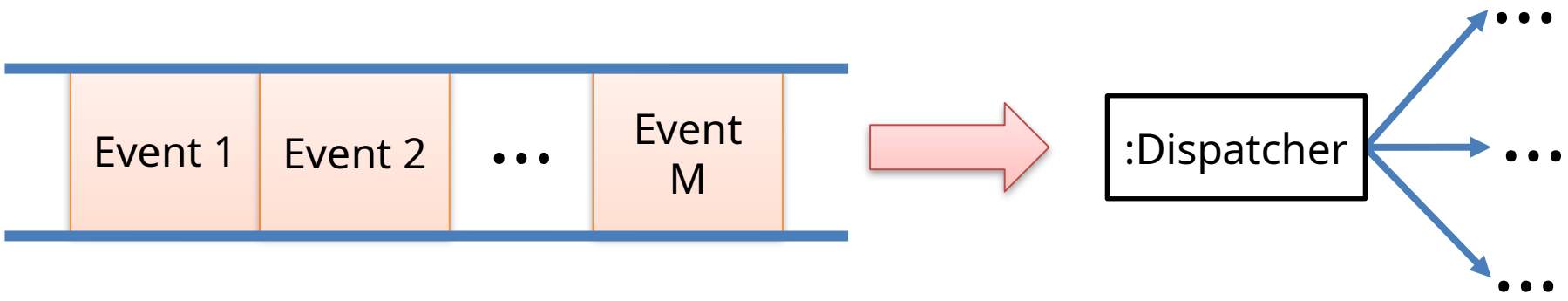
# CoR Alternative 1:
# Headless Event Handlers



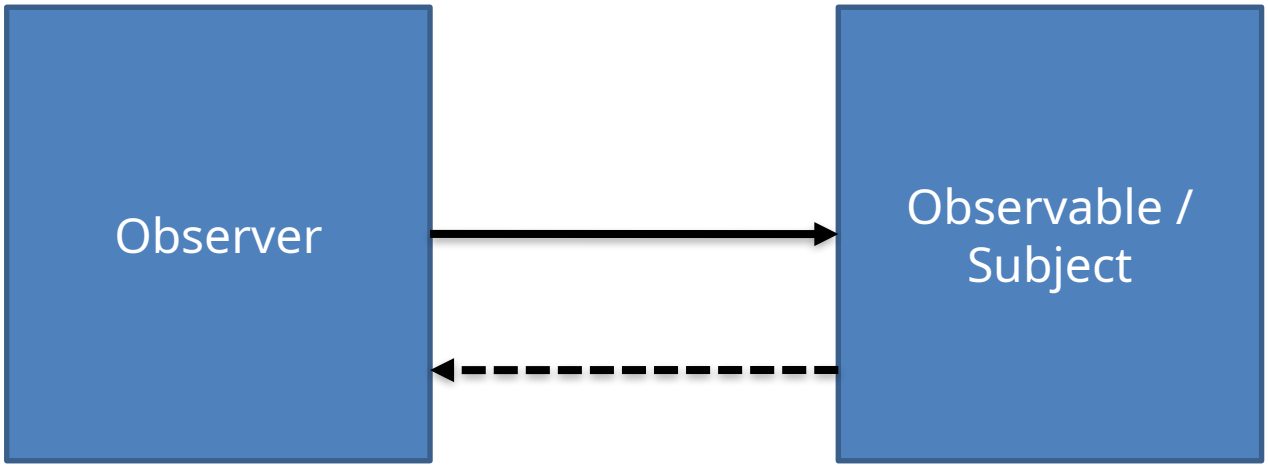
# CoR Alternative 1: Dispatcher and event loop



# Event queue



# Patrón Observador (Observer)



# Inversión de Control
# Inversion of Control (IoC)

# Caso “tradicional”

- Implementado por el programador

```
def calcular_nómina(empleado):
    ...


empleados_file = open("empleados", "r")
nómina_file = open("nómina", "w")

for empleado in empleados_file:
    nómina = calcular_nómina(empleado)
    nómina_file.write(empleado, nómina)

empleados_file.close()
nómina_file.close()
```

- Implementado por la librería

```
open()
write()
close()
```

# Inversion of Control (IoC)

- Implementado por el programador

```
def responda_a_cambio_en_caja_de_texto_1():
    ...

def responda_a_click_en_botón_1():
    ...
```

- Implementado por el framework

```
def usuario_hizo_click_en_botón_1():
    ...

def usuario_escribió_en_caja_de_texto_1():
    ...

def ignorar_o_lanzar_excepción():
    ...

while True:
    if usuario_hizo_click_en_botón_1():
        responda_a_click_en_botón_1()

    elif usuario_escribió_en_caja_de_texto_1():
        responda_a_cambio_en_caja_de_texto_1()

#   ...

    else:
        ignorar_o_lanzar_excepción()
```

# Máquinas de estados