IIC2523
Sistemas Distribuidos

Hernán F. Valdivieso López
(2025 - 2 / Clase 04)

Recordatorios importantes

1. El control 1 se entrega mañana a las 20:00. No olviden hacerlo.
    a. Si bien se elimina el peor, no intenten eliminar el primero 😓
2. No olviden responder la ECA, voy a hablar de ella al final de la clase.
3. Actualicé la página de Inicio con algo muy importante, recuerdenme hablar de eso finalizada la clase.

Comunicación entre nodos II
Comunicación indirecta e invocación remota

Temas de la clase

1. Comunicación indirecta
    a. Características generales
    b. Colas de mensaje
    c. Sistema *publish-subscribe*
2. Invocación remota
    a. Características generales
    b. *Remote Procedure Call* (RPC)
    c. RPC en Python
    d. gRPC y *protocol buffer*

Comunicación indirecta

Características generales

Colas

Modelo *publish-subscribe*

Comunicación Indirecta

Podemos resolver cualquier problema introduciendo un nivel adicional de indirección

Roger Needham, Maurice Wilkes and David Wheeler

Comunicación Indirecta

Podemos resolver cualquier problema introduciendo un nivel adicional de indirección

Roger Needham, Maurice Wilkes and David Wheeler

excepto el problema de tener muchas capas de indirección

Comunicación Indirecta - Características generales

La comunicación entre entidades de un sistema distribuido a través de un **intermediario** sin acoplamiento referencial entre el emisor y los receptores.

Comunicación Indirecta - Características generales

La comunicación entre entidades de un sistema distribuido a través de un **intermediario** sin acoplamiento referencial entre el emisor y los receptores.

Hay una variedad de mecanismos indirectos, lo más conocidos son:

- **Grupos**: el emisor manda el mensaje a un "proceso grupo". Este intermediario se encarga de enviar el mensaje a todos los procesos miembros.
- **Memoria Compartida**: se define un sector de la memoria donde todos los procesos pueden leer y escribir. No hay envío de mensaje, sino que se comparten variables.

Comunicación Indirecta - Características generales

La comunicación entre entidades de un sistema distribuido a través de un **intermediario** sin acoplamiento referencial entre el emisor y los receptores.

Hay una variedad de mecanismos indirectos, lo más conocidos son:

- **Grupos**: el emisor manda el mensaje a un "proceso grupo". Este intermediario se encarga de enviar el mensaje a todos los procesos miembros.
- **Memoria Compartida**: se define un sector de la memoria donde todos los procesos pueden leer y escribir. No hay envío de mensaje, sino que se comparten variables.
- **Colas de Mensajes:** el emisor envió el mensaje a una cola y el proceso receptor extrae el mensaje posteriormente.
- ***Publish-subscribe***: el emisor envía eventos (mensajes) a un sistema que los enruta a múltiples suscriptores según el interés de estos suscriptores.

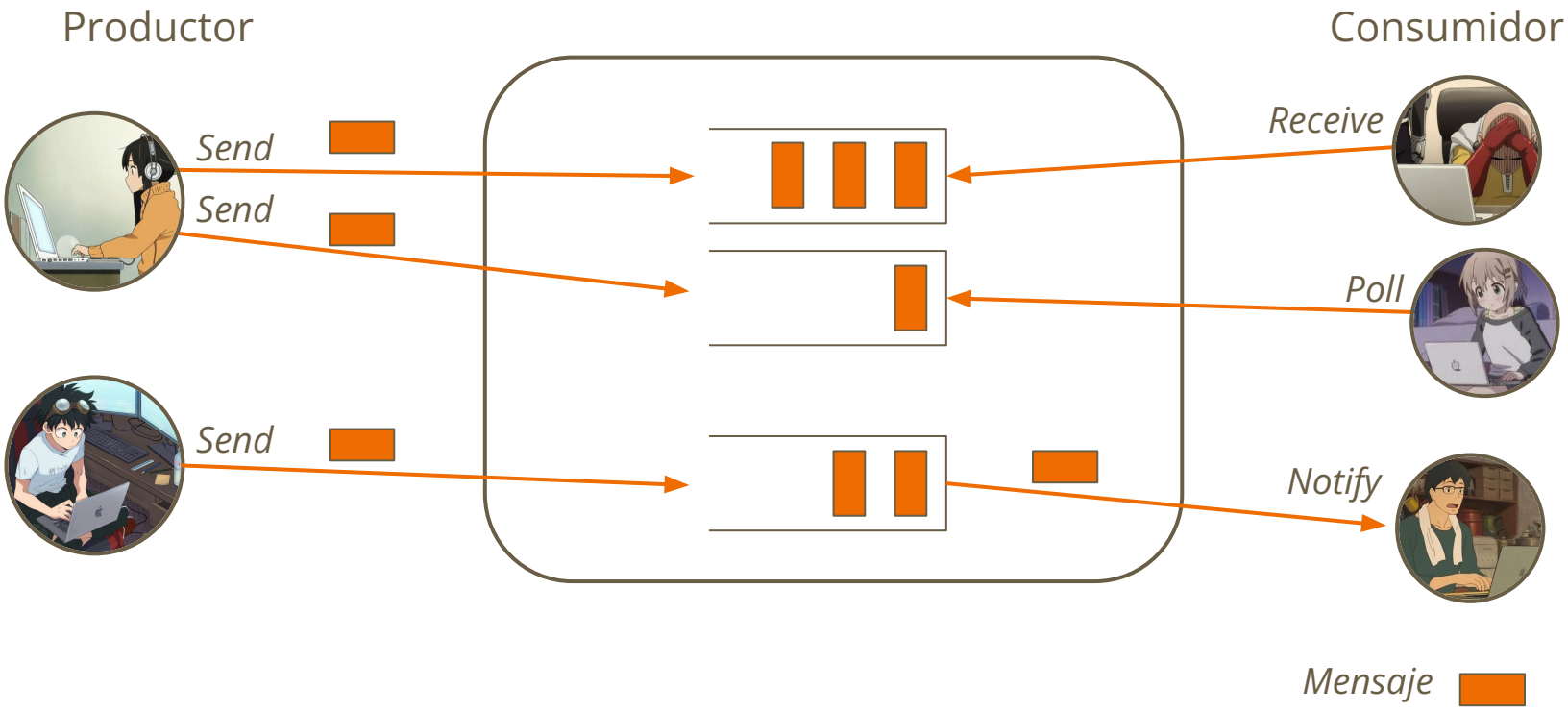
Comunicación Indirecta - Características generales

La comunicación entre entidades de un sistema distribuido a través de un **intermediario** sin acoplamiento referencial entre el emisor y los receptores.

Hay una variedad de mecanismos indirectos, lo más conocidos son:

- **Grupos**: el emisor manda el mensaje a un "proceso grupo". Este intermediario se encarga de enviar el mensaje a todos los procesos miembros.
- **Memoria Compartida**: se define un sector de la memoria donde todos los procesos pueden leer y escribir. No hay envío de mensaje, sino que se comparten variables.
- **Colas de Mensajes:** el emisor envió el mensaje a una cola y el proceso receptor extrae el mensaje posteriormente.
- ***Publish-subscribe***: el emisor envía eventos (mensajes) a un sistema que los enruta a múltiples suscriptores según el interés de estos suscriptores.

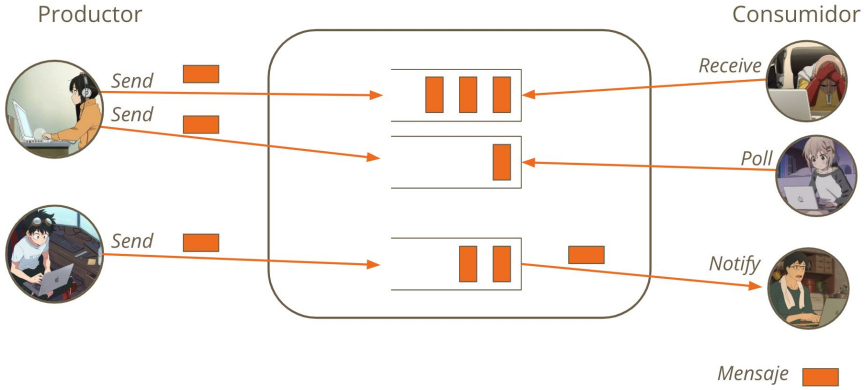
Comunicación Indirecta - Colas de Mensajes



Comunicación Indirecta - Colas de Mensajes

Funcionamiento

- El proceso "productor" envía mensajes a una cola.
- El proceso "consumidor" obtiene el mensaje de la cola.
- Una vez que un mensaje es extraído, generalmente es eliminado de la cola.
- Por defecto, este mecanismo asegura desacoplamiento temporal dado que la cola permite la persistencia de datos.

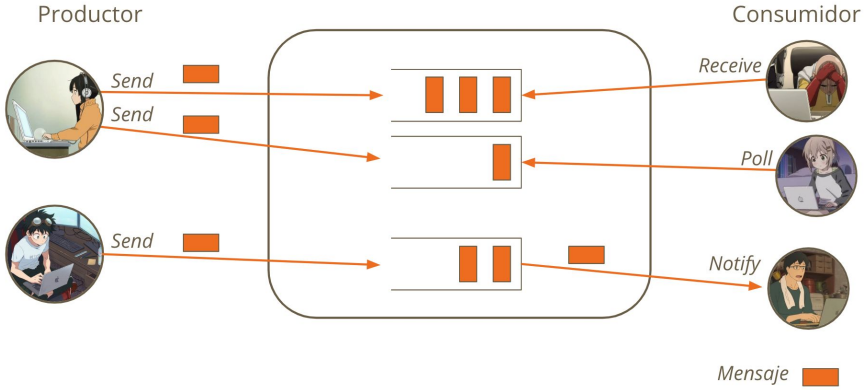


Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

Comunicación Indirecta - Colas de Mensajes

Operaciones básicas

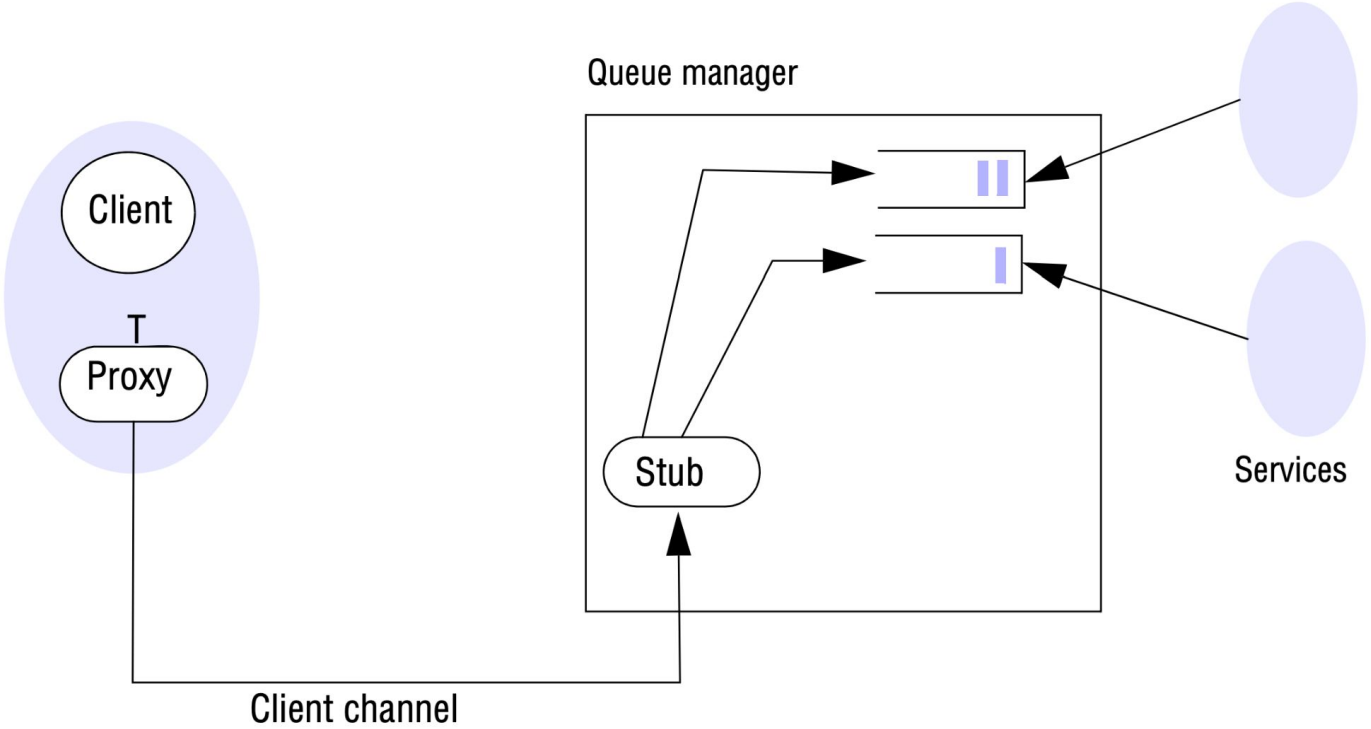
- **`Send`**: añadir mensaje a una cola.
- **`Receive`**: Esperar hasta que haya un mensaje en la cola.
- **`Poll`**: Verificar si hay mensaje, y si es que hay, obtenerlo.
- **`Notify`**: Instalar un controlador que se llama cuando un mensaje llega a la cola para avisar al consumidor.



Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

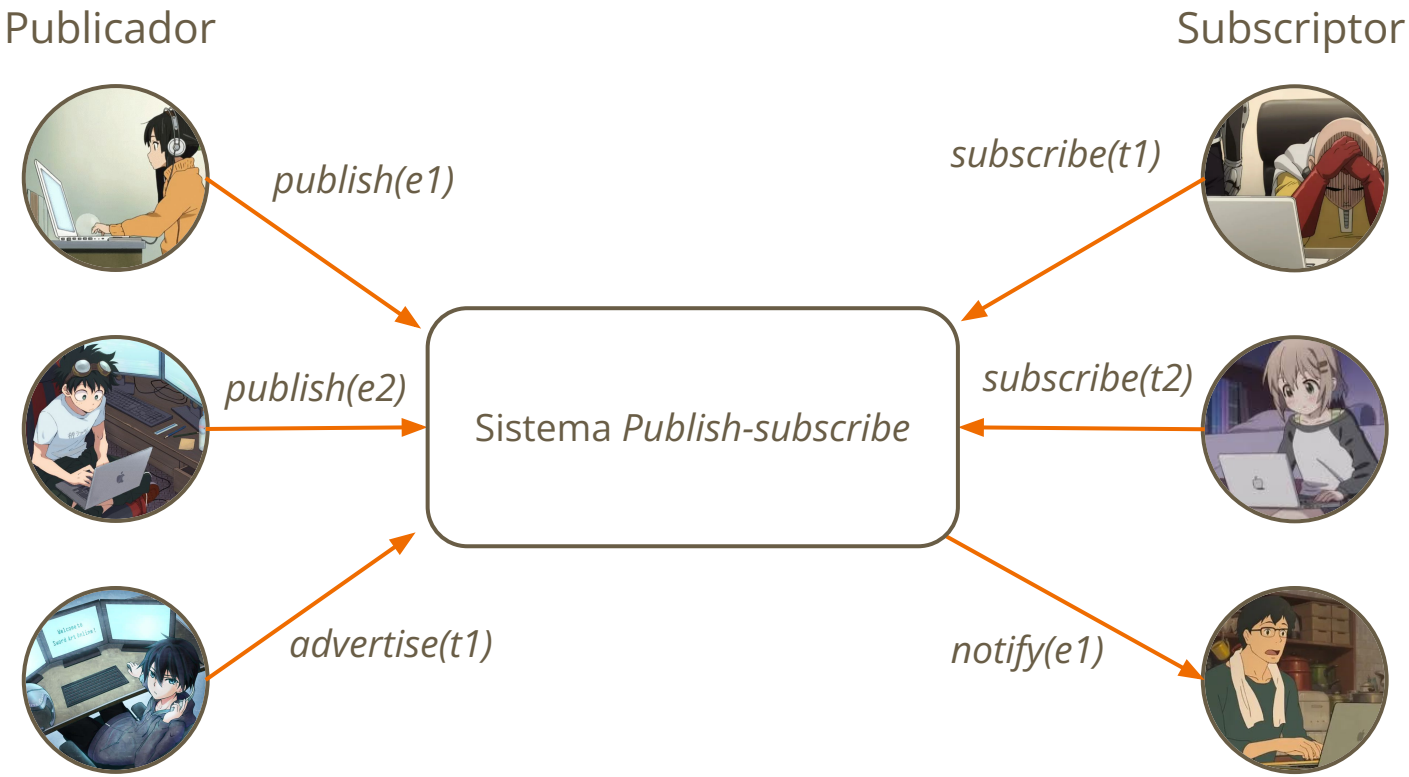
Comunicación Indirecta - Colas de Mensajes

Ejemplo WebSphere MQ desarrollado por IBM



Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

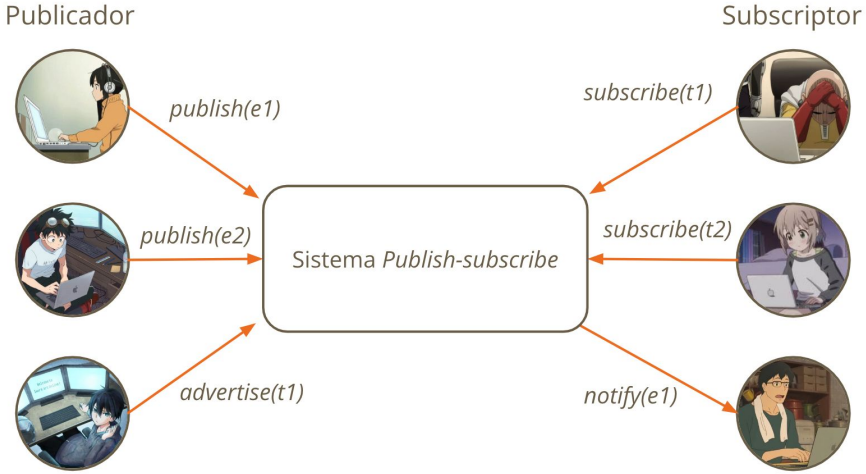
Comunicación Indirecta - Sistema *Publish-subscribe*



Comunicación Indirecta - Sistema *Publish-subscribe*

Funcionamiento

- Los publicadores envían eventos (mensajes) a un servicio intermediario.
- Los suscriptores expresan interés en eventos mediante filtros.
- El sistema compara los eventos con los filtros para enviar eventos a los suscriptores correspondientes.
- Según la implementación, puede garantizar desacoplamiento temporal si es que el sistema asegura persistencia de mensajes.

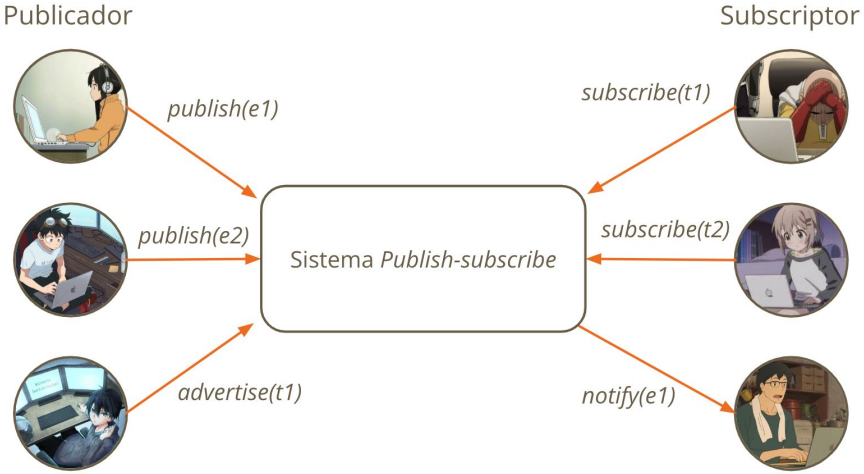


Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

Comunicación Indirecta - Sistema *Publish-subscribe*

Este sistema busca la **Heterogeneidad**

- Permite la interoperación entre componentes que no fueron diseñados para trabajar juntos, siempre que puedan publicar/suscribirse a eventos.

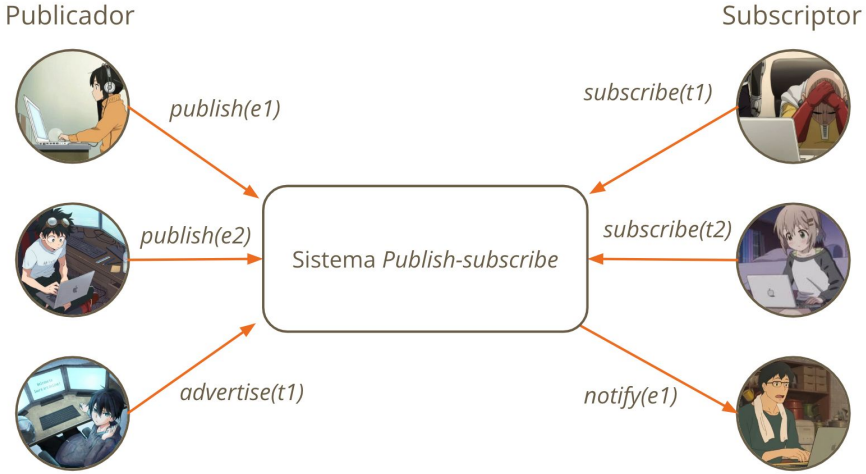


Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

Comunicación Indirecta - Sistema *Publish-subscribe*

Operaciones básicas

- **`publish(e)`**: enviar evento **`e`**.
- **`subscribe(f)`**: Expresar interés en eventos que coincidan con un filtro **`f`**.
- **`unsubscribe(f)`**: Revocar interés al filtro **`f`**.
- **`notify(e)`**: Entregar el evento **`e`**.
- **[opcional]** **`advertise(f)`**: Declaran el tipo de eventos (filtro **`f`**) que generarán en el futuro.



Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

Comunicación Indirecta - Sistema *Publish-subscribe*

Modelos de Suscripción (Filtros)

Las suscripciones a los eventos se pueden realizar según diferentes intereses:

- ***Channel-based***: Publicadores envían a canales nombrados. Suscriptores se suscriben a estos canales.
- ***Topic-based***: Los eventos tienen un campo de "tema/tópico". Las suscripciones se basan en este tema.
- ***Content-based***: Filtros más expresivos, que son consultas sobre los valores de los atributos del evento. Permiten coincidencias asociativas más complejas.

El filtro puede ser una mezcla de los diferentes tipos indicados anteriormente.

Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

Invocación remota

Características generales

***Remote Procedure Call* (RPC)**

RPC en Python

gRPC y *protocol buffer*

Comunicación remota (Respecto a Clase 03)

- Uso de componentes y/o protocolos que permiten el intercambio de datos entre procesos ubicados en diferentes máquinas conectadas en red.
  - Transmisión de información.
  - Envío de archivos.
  - Comunicarse con una API.

Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

Comunicación remota (Respecto a Clase 03)

- Uso de componentes y/o protocolos que permiten el intercambio de datos entre procesos ubicados en diferentes máquinas conectadas en red.
  - Transmisión de información.
  - Envío de archivos.
  - Comunicarse con una API.
- En este curso **estudiaremos específicamente el paradigma de Invocación Remota**, que es un caso particular de comunicación remota, pero con una característica clave: permite que el receptor ejecute una acción solicitada por el emisor.
  - Comunicarse con una API.

Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

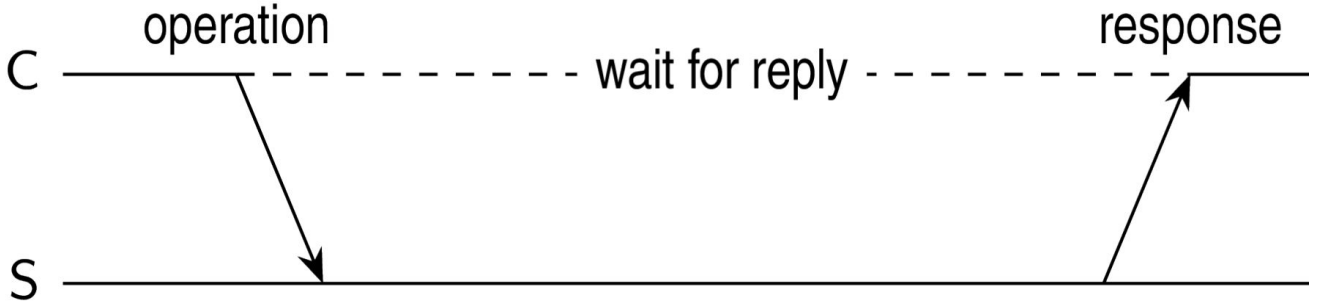
Invocación Remota - Características generales

- Paradigma de comunicación donde se invocan operaciones o métodos que se ejecutan en otra máquina/proceso.

Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

Invocación Remota - Características generales

- Paradigma de comunicación donde se invocan operaciones o métodos que se ejecutan en otra máquina/proceso.
- Por defecto, este paradigma utiliza un protocolo del tipo *request-reply*.
  - Enviar un mensaje y esperar respuesta.
  - Protocolo síncrono y acoplado temporalmente.
  - Es posible hacer este protocolo asíncrono con otras herramientas como *threads*, promesas, etc.).



Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

Invocación Remota - Características generales

- Existen 2 enfoques muy conocidos: REST y RPC.

Invocación Remota - REST

- Existen 2 enfoques muy conocidos: **REST** y RPC.
  - ***Re****presentational* ***S****tate* ***T****ransfer*.
  - Utiliza protocolo HTTP y se alínea a sus estándares para comunicarse.
  - Uso de URLs para identificar los distintos recursos/*endpoint* y las operaciones HTTP (GET, POST, etc.) para manipularlos.
  - Cada ejecución es ***Stateless*** → no tiene memoria de las ejecuciones anteriores.
  - Sus mensajes son *human readable*.

```
GET /anime/detective-conan HTTP/1.1
Host: www3.animeflv.net
Remote Address: 185.178.208.140:443
Accept: text/html
```

Invocación Remota - REST

- Existen 2 enfoques muy conocidos: **REST** y RPC.
  - ***Re****presentational* ***S****tate* ***T****ransfer*.
  - Utilizan protocolo HTTP para definir operaciones sobre recursos.
  - Sus mensajes son *human readable*.

Al momento de definir el concepto de "Invocación remota" (tipos años 70 👨‍🦳), no existía REST y por lo mismo la bibliografía no la incluye, pero sus características permiten que sea considerada dentro de este paradigma.

```
GET /anime/detective-conan HTTP/1.1
Host: www3.animeflv.net
Remote Address: 185.178.208.140:443
Accept: text/html
```

Invocación Remota - *Remote Procedure Call* (RPC)

- Enfoque donde se busca la **transparencia de acceso y de ubicación**
  - Se llaman a las funciones/procedimientos de una **forma idéntica a como se llamaría de forma local**.

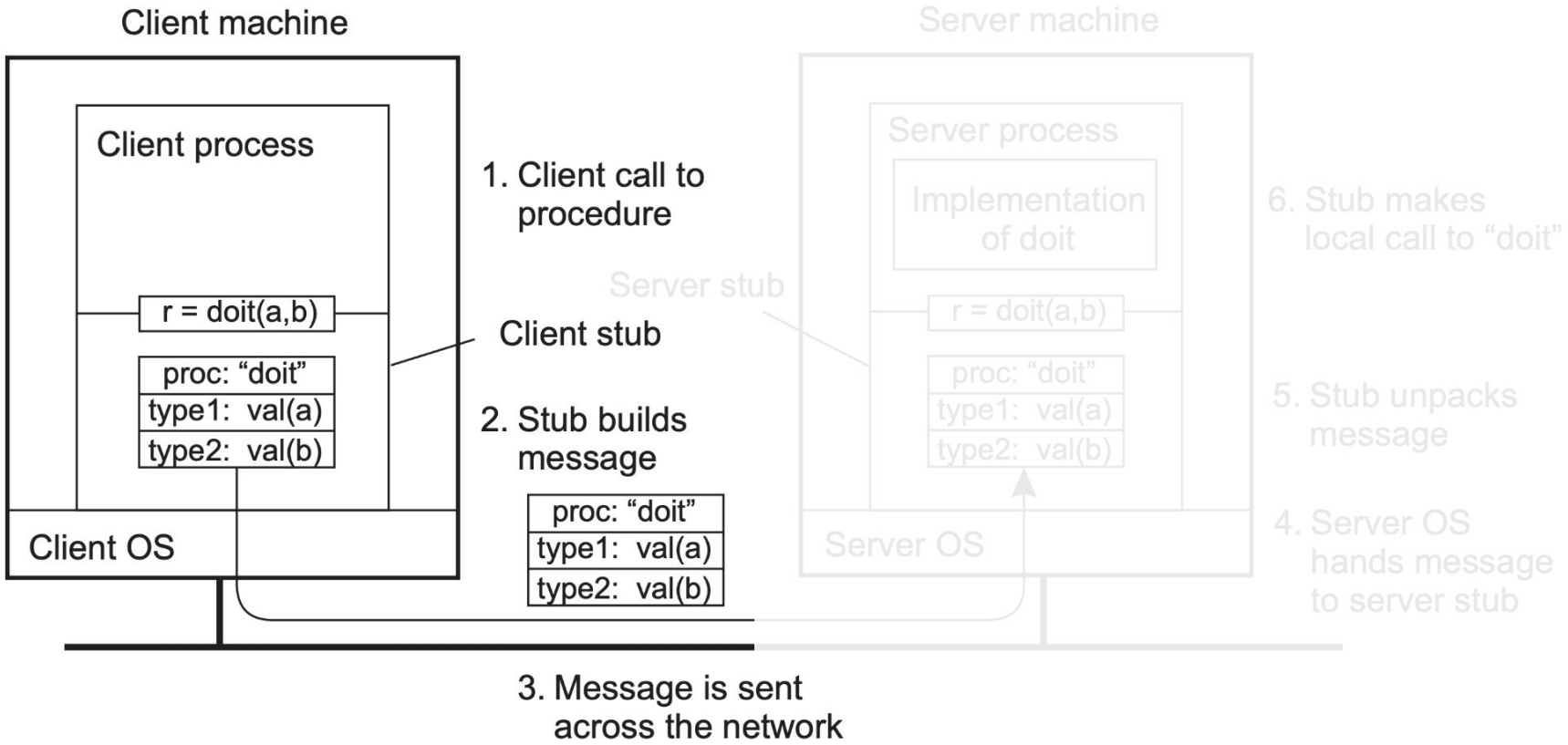
Invocación Remota - *Remote Procedure Call* (RPC)

- Enfoque donde se busca la **transparencia de acceso y de ubicación**
  - Se llaman a las funciones/procedimientos de una **forma idéntica a como se llamaría de forma local**.
- Se define un nuevo componente: *stub*
  - ***Client stub***: convierte los llamados del cliente y sus argumentos en mensajes para enviar al servidor. Luego interpreta el mensaje obtenido y la convierte en el resultado de la función ejecutada.
  - ***Server stub***: interpreta los mensajes del cliente para identificar qué función ejecutar y luego genera el mensaje con la respuesta de dicha ejecución.

Invocación Remota - *Remote Procedure Call* (RPC)

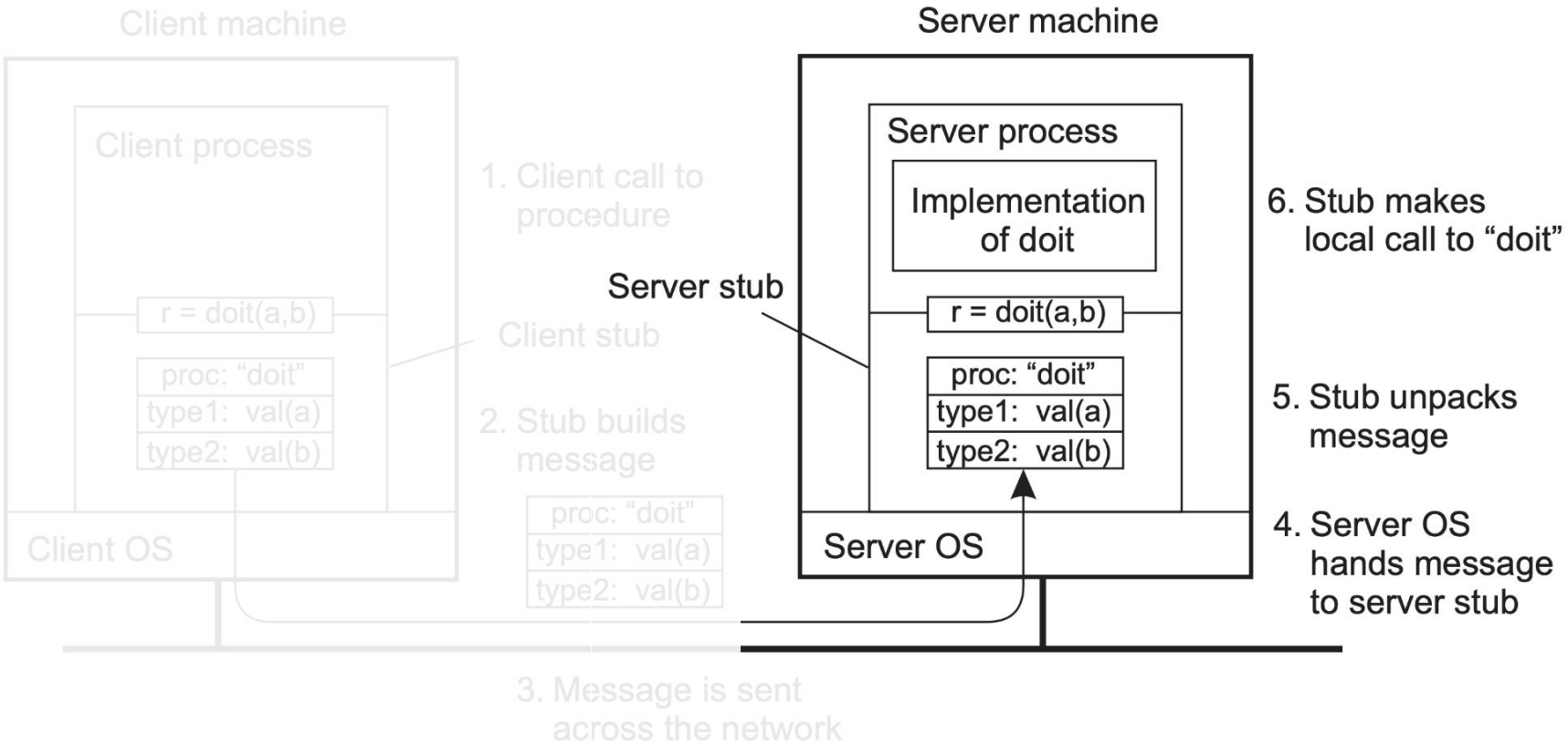
- Enfoque donde se busca la **transparencia de acceso y de ubicación**
  - Se llaman a las funciones/procedimientos de una **forma idéntica a como se llamaría de forma local**.
- Se define un nuevo componente: *stub*
  - ***Client stub***: convierte los llamados del cliente y sus argumentos en mensajes para enviar al servidor. Luego interpreta el mensaje obtenido y la convierte en el resultado de la función ejecutada.
  - ***Server stub***: interpreta los mensajes del cliente para identificar qué función ejecutar y luego genera el mensaje con la respuesta de dicha ejecución.
- La comunicación se realiza mediante serialización (frecuentemente binaria), aunque hoy en día existen implementaciones que usan formatos legibles por humanos.

Invocación Remota - *Remote Procedure Call* (RPC)



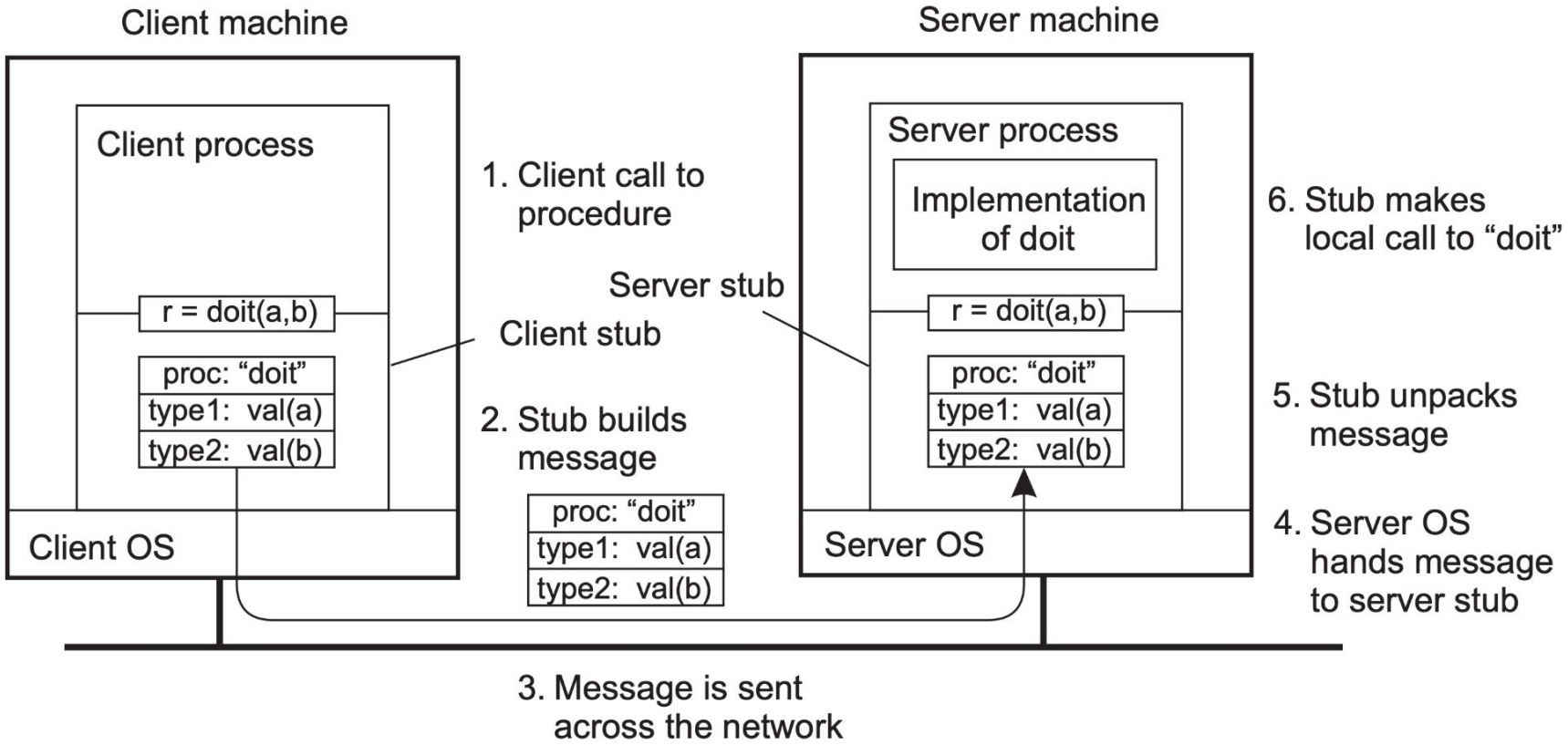
Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

Invocación Remota - *Remote Procedure Call* (RPC)



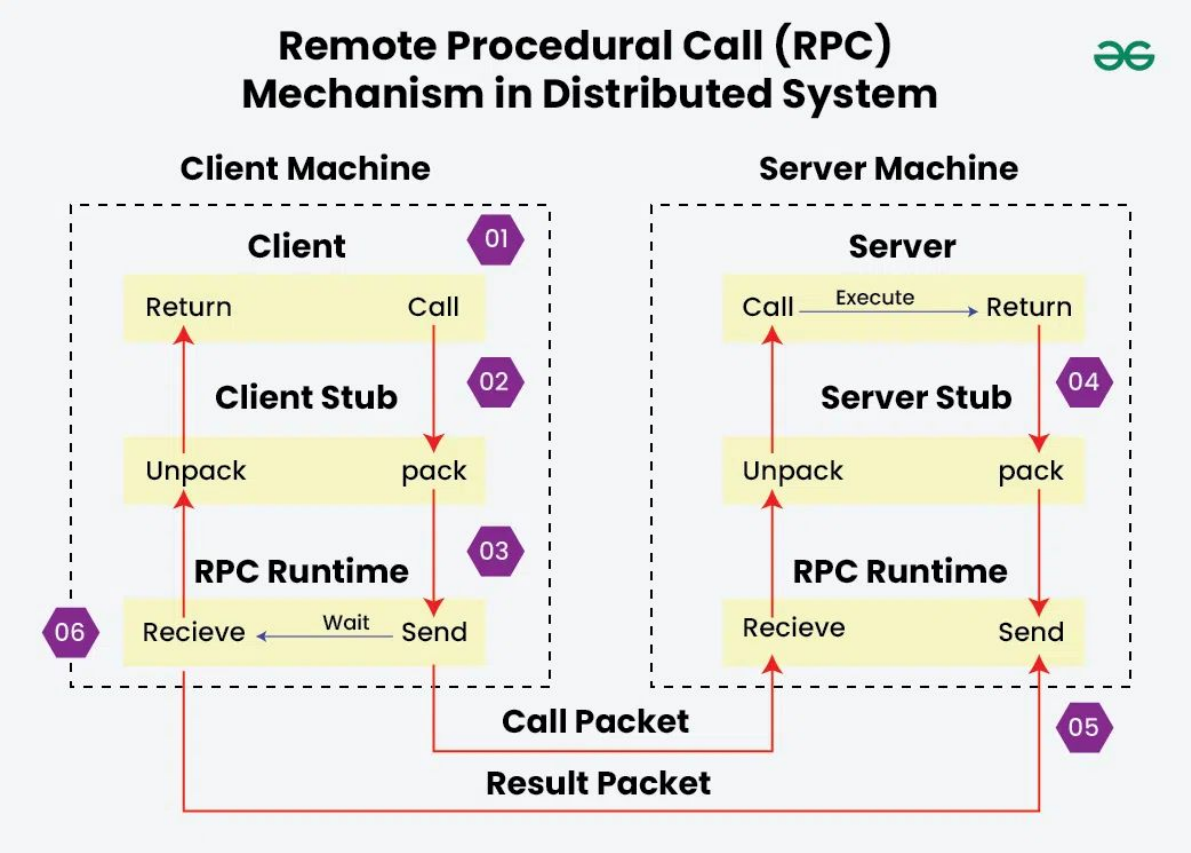
Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

Invocación Remota - *Remote Procedure Call* (RPC)



Fuente: Distributed Systems: Concepts and Design (5ª ed.), Coulouris, G., Dollimore, J., & Kindberg, T. (2005)

Invocación Remota - *Remote Procedure Call* (RPC)



Fuente: What is Remote Procedural Call (RPC) Mechanism in Distributed System?

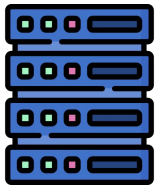
Invocación Remota - REST o RPC

GET /animes

```
[{"id": 1, "name": "Evangelion"},
 {"id": 2, "name": "Code Geass"} … ]
```

GET /animes/2

```
{"id": 2,
 "name": "Code Geass",
 "seasons": 2, … }
```



REST

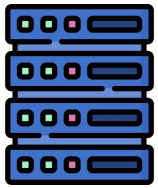
Invocación Remota - REST o RPC

GET /animes


```
[{"id": 1, …},
 {"id": 2, …} … ]
```


GET /animes/2


```
{"id": 2 … }
```

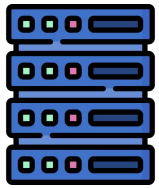
REST

s.animes()


```
[{"id": 1, …},
 {"id": 2, …} … ]
```


s.animes(id=2)


```
{"id": 2 … }
```

RPC

Invocación Remota - REST o RPC

- Al momento de comunicarse, los *stubs* pueden usar comunicación directa mediante *sockets* y sus propias normas, o emplear otros protocolos como HTTP. No obstante, no necesariamente respetan los estándares de HTTP:
  - El servidor RPC puede mantener estado y recordar ejecuciones pasadas.
  - Un GET puede modificar información.
  - Un POST puede ser solo para leer datos.
- Se puede incluir un intermediario (comunicación indirecta) para ofrecer funcionalidades adicionales.

Invocación Remota - RPC (servidor)

```
DATA = []

def f1(x, y):
    return x + y

def f2():
    return "Hello World"

def f3(x):
    DATA.append(x)
    return DATA[0]
```

Invocación Remota - RPC (servidor)

```python
DATA = []

def f1(x, y):
    return x + y

def f2():
    return "Hello World"

def f3(x):
    DATA.append(x)
    return DATA[0]
```

```python
from xmlrpc.server import SimpleXMLRPCServer


ip_port = ("localhost", 4444)
server = SimpleXMLRPCServer(ip_port, allow_none=True)
server.register_function(f1, "add")
server.register_function(f2, "hello_world")
server.register_function(f3, "append")
server.serve_forever()
```

Invocación Remota - RPC (cliente)

```
from xmlrpc.client import ServerProxy

servidor = ServerProxy("http://localhost:4444/", allow_none=True)
print("3 + 8 =", servidor.add(3, 8))
print("-->", servidor.hello_world())
print("DATA: ", servidor.append("Probando"))
print("DATA: ", servidor.append("4444"))
print("Concatenar string: ", servidor.add("AN", "YA"))
print("DATA: ", servidor.append(None))
```

Invocación Remota - *Remote Method Invocation* (RMI)

- Contexto: Java es un lenguaje que se basa fuertemente en la Programación Orientado a Objetos.
- RMI fue diseñado para implementar RPC en Java.
- RMI es un tipo específico de RPC donde se invocan **métodos** de un objeto dentro del paradigma de OOP.

Invocación Remota - RMI (servidor)

```
class RMIServer:
    def __init__(self):
        self.data = []

    def add(self, x, y):
        return x + y

    def hello_world(self):
        return "Hello World"

    def append(self, x):
        self.data.append(x)
        return self.data[0]
```

```
from xmlrpc.server import SimpleXMLRPCServer


rmi = RMIServer()
ip_port = ("localhost", 4444)
server = SimpleXMLRPCServer(ip_port,
                            allow_none=True)
server.register_instance(rmi)
server.serve_forever()
```

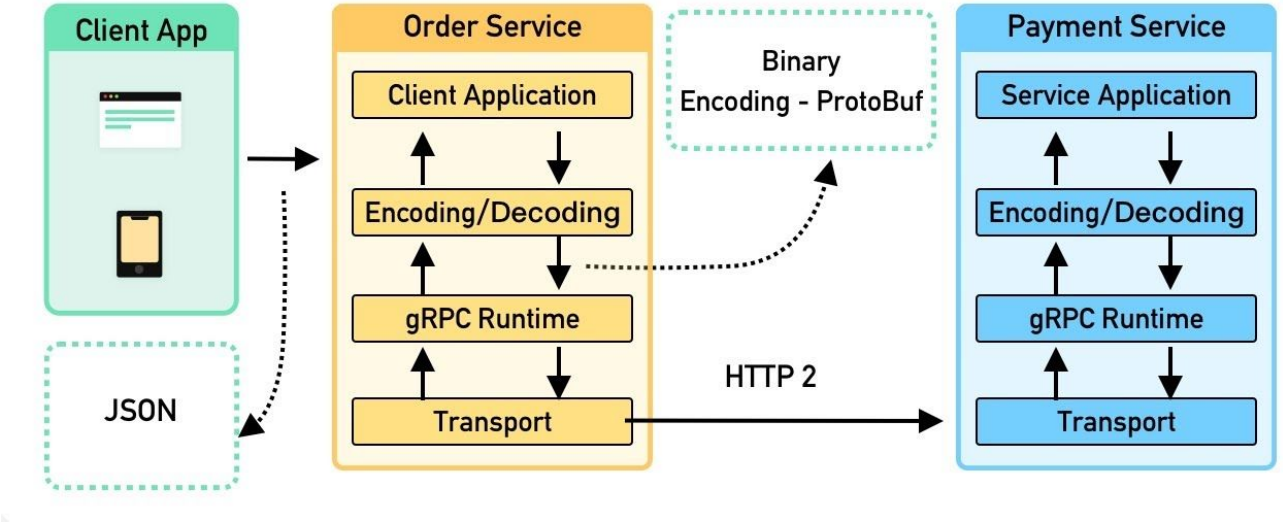
Invocación Remota - RMI (cliente)

```
# Se mantiene igual aunque el servidor use register_instance o register_function
from xmlrpc.client import ServerProxy

servidor = ServerProxy("http://localhost:4444/", allow_none=True)
print("3 + 8 =", servidor.add(3, 8))
print("-->", servidor.hello_world())
print("DATA: ", servidor.append("Probando"))
print("DATA: ", servidor.append("4444"))
print("Concatenar string: ", servidor.add("AN", "YA"))
print("DATA: ", servidor.append(None))
```

Invocación Remota - gRPC

- Implementación específica de RPC por Google el 2015.
- Utiliza un protocolo de comunicación más rápido y eficiente (HTTP/2).
- Admite más formatos de serialización como *Protocol Buffers*, JSON y XML.



Fuente: What is RPC? gRPC Introduction.

Invocación Remota - gRPC

- Implementación específica de RPC por Google el 2015.
- Utiliza un protocolo de comunicación más rápido y eficiente (HTTP/2).
- Admite más formatos de serialización como *Protocol Buffers*, JSON y XML.
- ***Protocol Buffers***

1. Se define la estructura de los datos a enviar en un archivo de esquema.
2. *Protocol Buffers* genera un código, en el lenguaje deseado, con la estructura lista para importar y utilizar.
3. Se instancian los datos usando dicha estructura.
4. Se serializar y deserializar esas instancias en binario para su transmisión.

Invocación Remota - gRPC

- *Protocol Buffers*

```
# Esquema
syntax = "proto3";

message Character {
   string username = "";
   string phase = "";
}
```

```
import character_pb3 # Este codigo creado por pb

character = character_pb3.Character()
character.username = "Anya"
character.phase = "Waku Waku"

with open("data.anya", "wb") as file:
    file.write(character.SerializeToString())


character_new = character_pb3.Character()
with open("data.anya", "rb") as file:
    character_new.ParseFromString(file.read())
```

Poniendo a prueba lo que hemos aprendido 👀

Un sistema de vigilancia está compuesto por múltiples cámaras que reportan datos en tiempo real a través de RPC hacia un servidor. Este servidor luego reenvía los datos a distintos clientes mediante un sistema de colas. ¿Qué **ventaja clave aporta** este diseño híbrido?

a. Aumentar la latencia en el traspaso de información.

b. Garantiza que los sensores puedan comunicarse incluso si el servidor está inactivo.

c. Asegura que todos los clientes reciban todos los datos al mismo tiempo.

d. Permitir que los clientes reciben los datos cuando ellos quieran.

e. Disminuir la latencia en el traspaso de información.

Poniendo a prueba lo que hemos aprendido 👀

Un sistema de vigilancia está compuesto por múltiples cámaras que reportan datos en tiempo real a través de RPC hacia un servidor. Este servidor luego reenvía los datos a distintos clientes mediante un sistema de colas. ¿Qué **ventaja clave aporta** este diseño híbrido?

a. Aumentar la latencia en el traspaso de información.

b. Garantiza que los sensores puedan comunicarse incluso si el servidor está inactivo.

c. Asegura que todos los clientes reciban todos los datos al mismo tiempo.

d. Permitir que los clientes reciben los datos cuando ellos quieran.

e. Disminuir la latencia en el traspaso de información.

Próximos eventos

Próxima clase

- **Nueva unidad**: Coordinación en un Sistema Distribuido.
- En particular, profundizaremos sobre sincronización en Sistemas distribuidos
  - ¿Qué pasa si un nodo es muy lento y otro muy rápido? ¿sus tiempos serán los mismos?
  - Estudiaremos *Relojes Lógicos de Lamport*. Último contenido para la Tarea 1.

Evaluación

- Control 2, se publica mañana y evalúa hasta esta clase.
- Tarea 1 *spoiler*: levantar más de un servidor TCP con *socket* que no se quede bloqueado ante múltiples clientes conectados.
- Tarea 1 *spoiler 2*: los servidores TCP tendrán que interactuar con un RCP 👀.

IIC2523
Sistemas Distribuidos

Hernán F. Valdivieso López
(2025 - 2 / Clase 04)

Créditos (animes utilizados)

K-On!


One Punch Man

Boku no Hero Academia

Sword Art Online