CS 7172
Parallel and Distributed Computation

Remote Procedure Calls

Kun Suo

Computer Science, Kennesaw State University

https://kevinsuo.github.io/

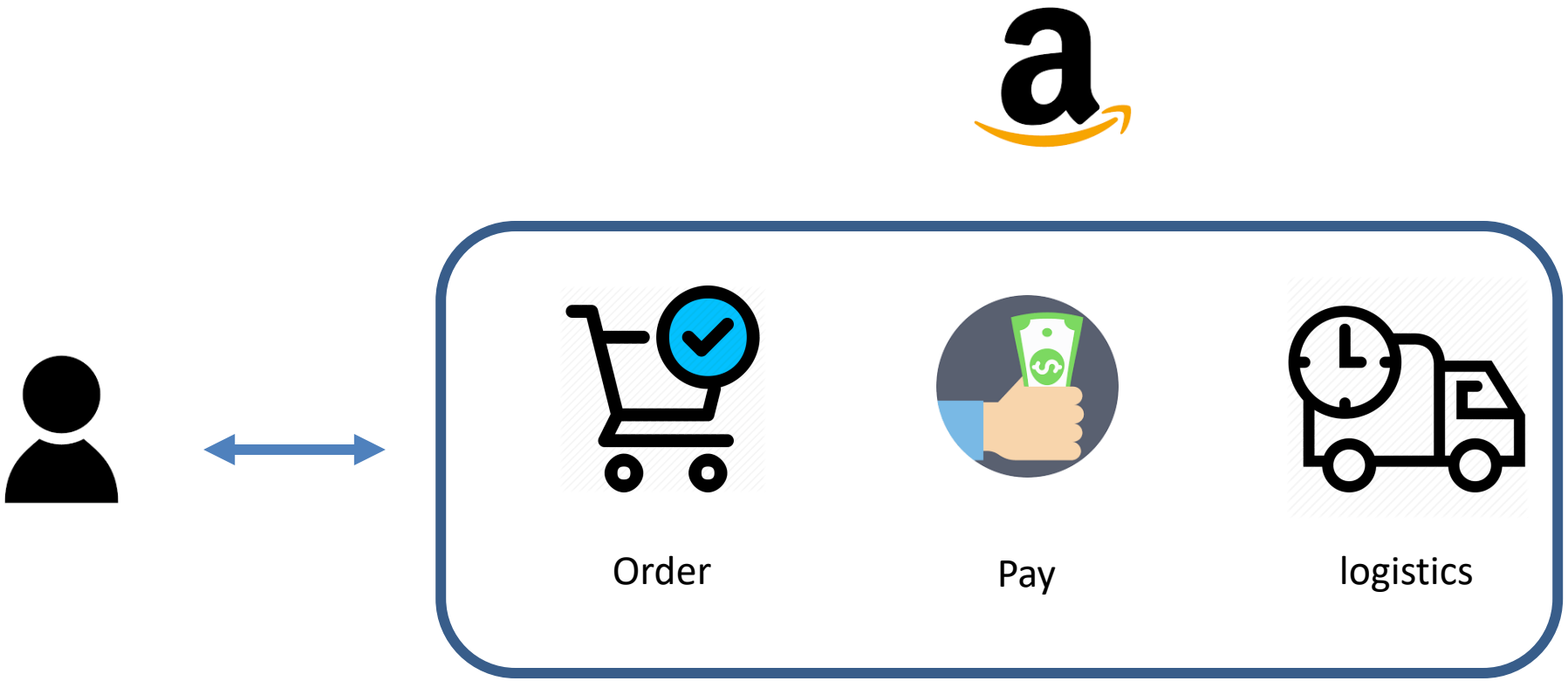
Outline

- What is RPC and why we need RPC?
- How RPC works?
- Implementation details of local call and RPC
- Examples of RPC
- Remote Method Invocation (RMI)
- Synchronous and asynchronous in RPC

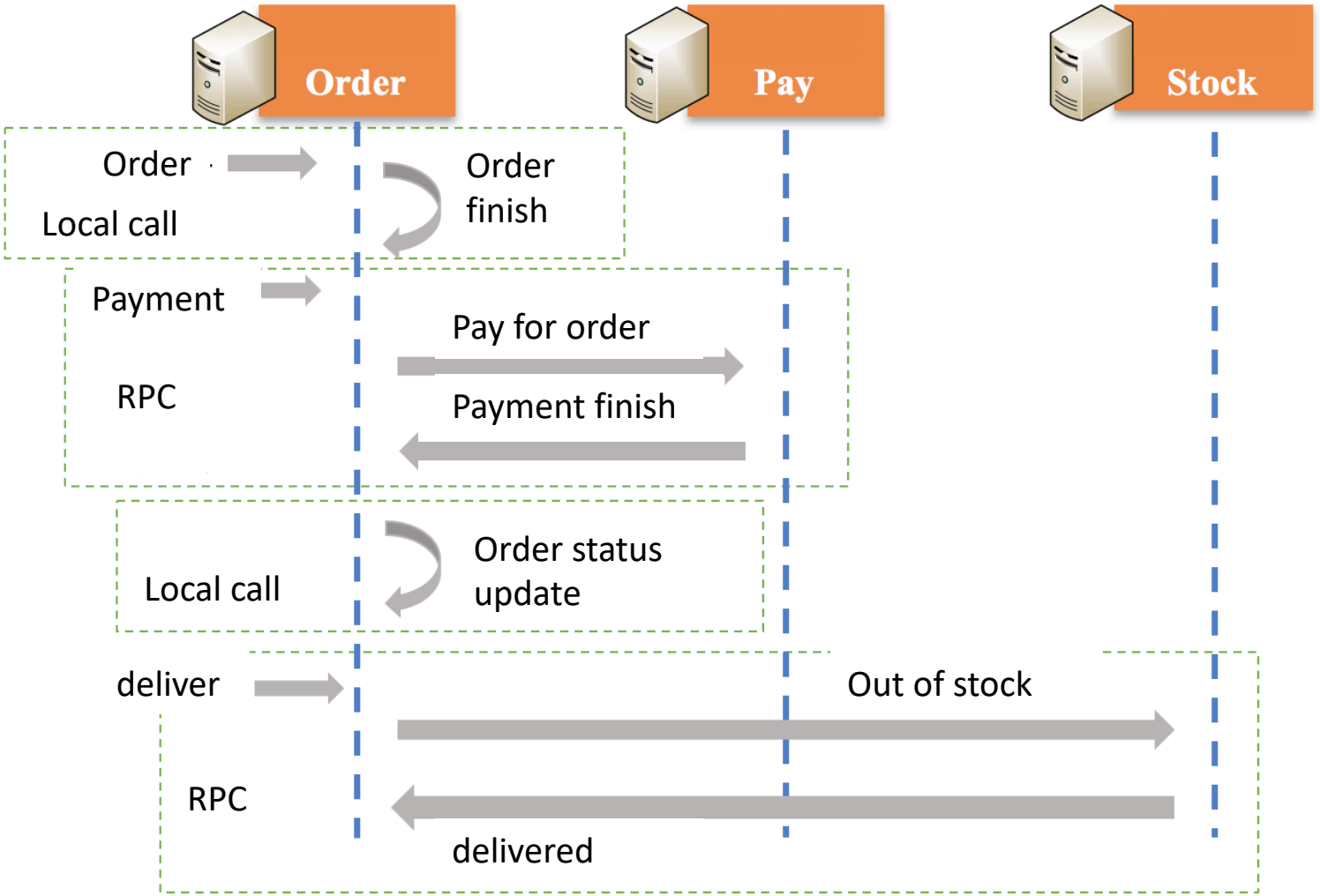
Why Remote Procedure Calls?

- The essence of distributed is multi-process/thread collaboration to complete tasks together
- In order to collaborate, communication is inevitable
- ***Remote Procedure Calls*** is one way for communication for processes in distributed systems

What is Remote Procedure Calls?



What is Remote Procedure Calls? Example



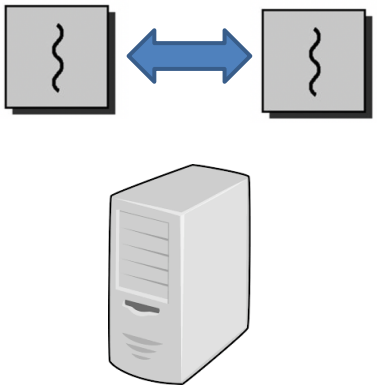
Local call and Remote call

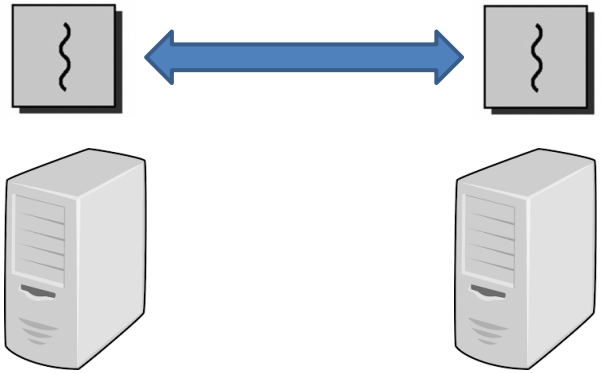
- Local call usually refers to the mutual call between functions within the process
- Remote call is the mutual call of functions between different processes
- Remote call is one way of Inter-Process Communication (IPC)

Two categories of Remote Call

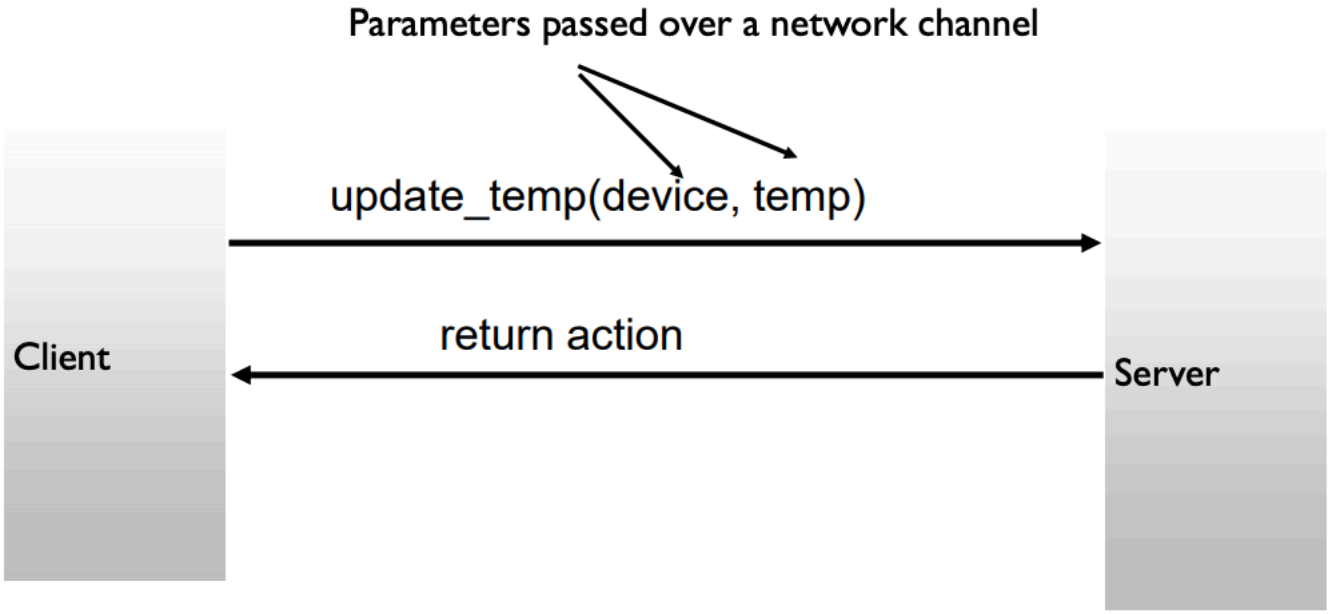
- Local Procedure Call (LPC): communication between processes running on the same machine

- Remote Procedure Call (RPC): communication between processes running on different machines ➔ Widely used in distributed systems





RPC Example


Parameters passed over a network channel
update_temp(device, temp)
return action
Client
Server

RPC Advantages

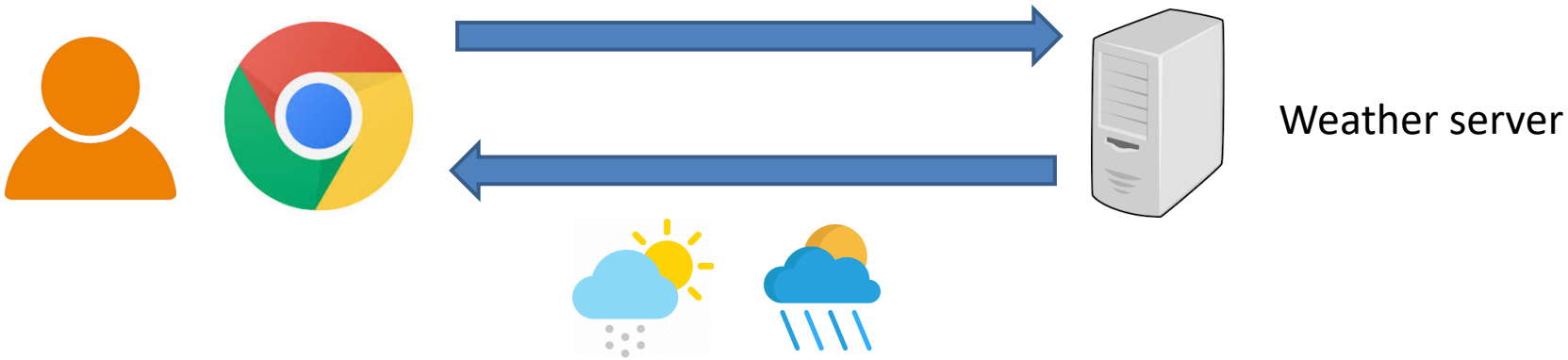
- Clean and simple to understand semantics like local procedure calls
- Generality: all languages have local procedure calls
  - RPC libraries augment the procedure call interface to make RPCs appear like local calls
- Abstraction for a common client/server communication pattern

```
push_temp(name) {
   t = get_current_temp();
         return update_temp (name, t);  //RPC
         }
```

B/S Architecture

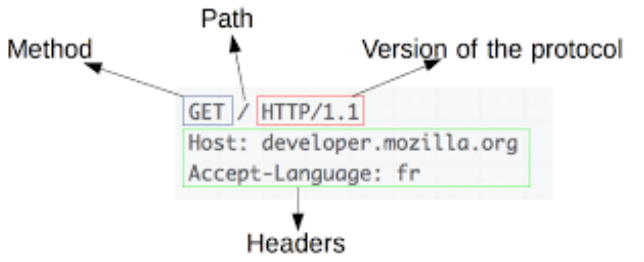
- Browser/Server architecture: runs an application directly from an internet browser

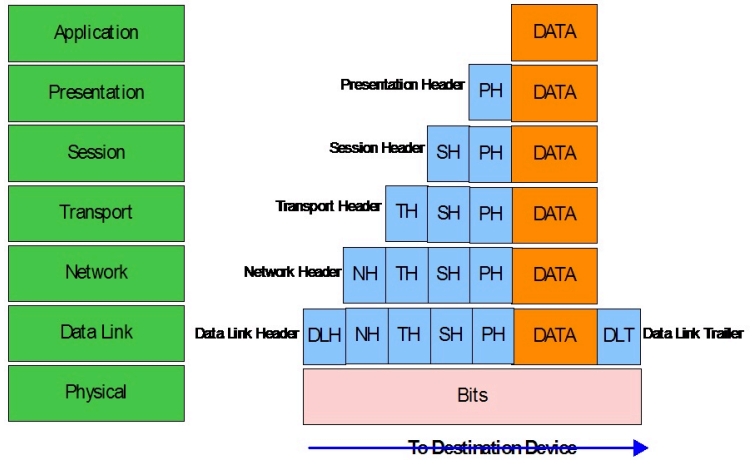
- The server side opens an interface and the client side uses browser to call the API to achieve remote call



B/S Architecture

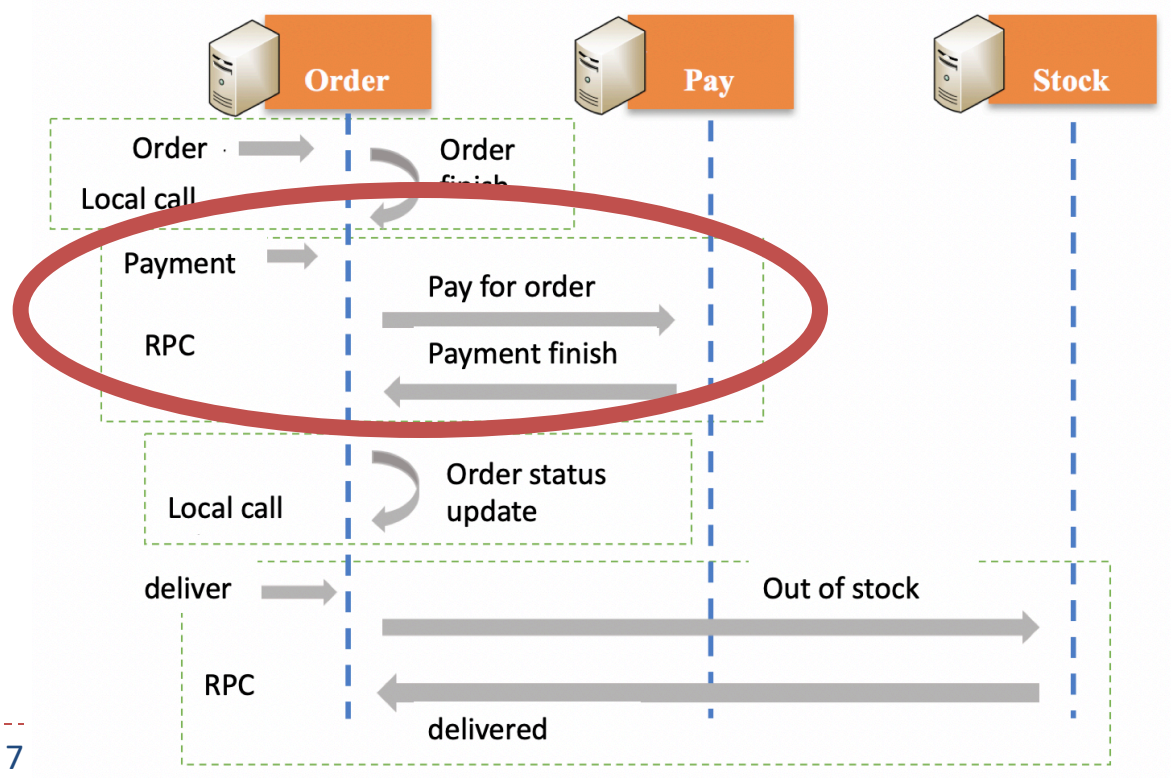
- B/S Architecture is implemented by HTTP protocol
  - Every time calling the APIs, client needs to send HTTP request
    - Complex protocol, close to user level
    - Time-consuming, not suitable to low latency request in large-scale distributed system



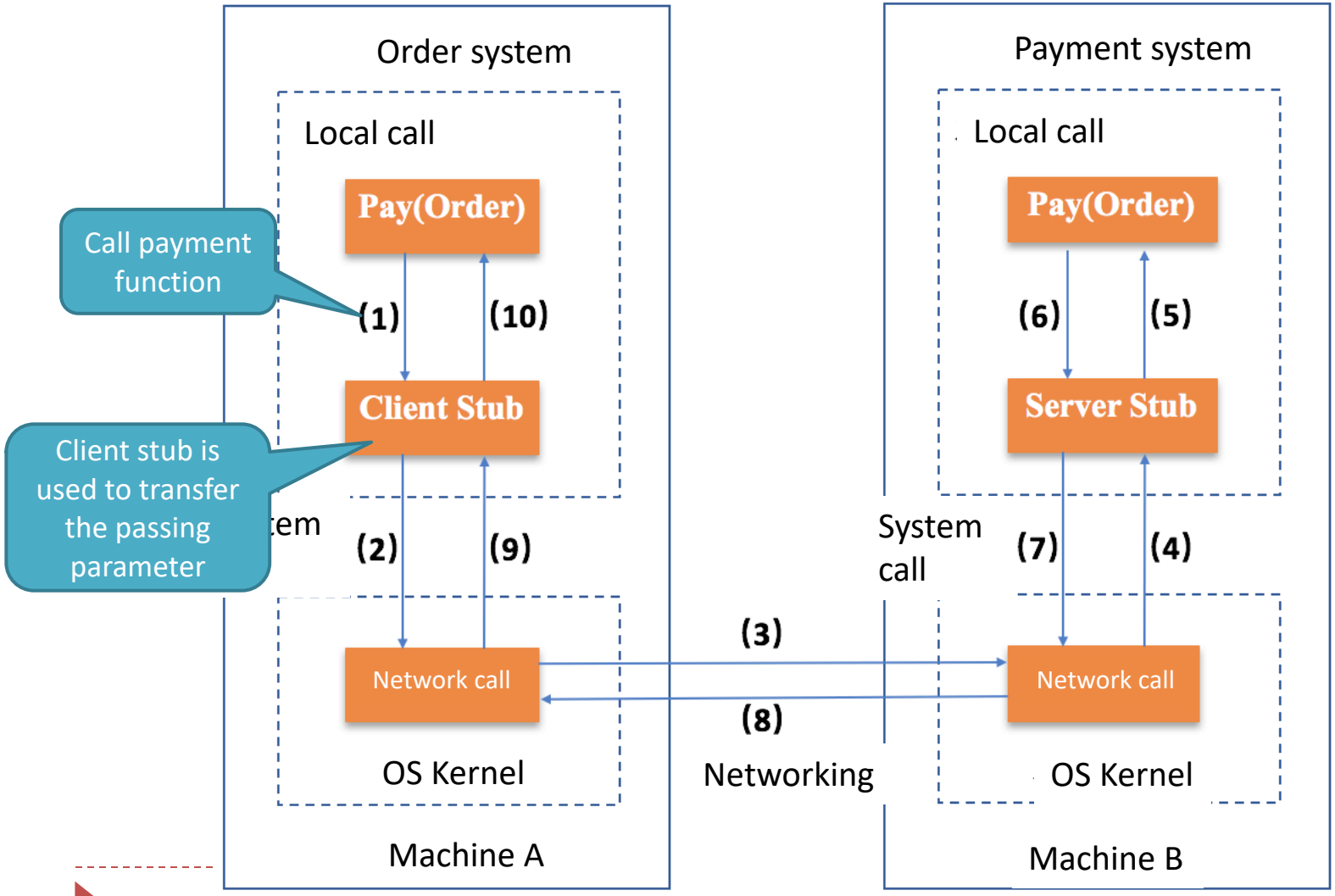


How RPC works?

- The caller uses the parameter passing method to call a function on the local machine to execute the function on the remote machine and receives the return results



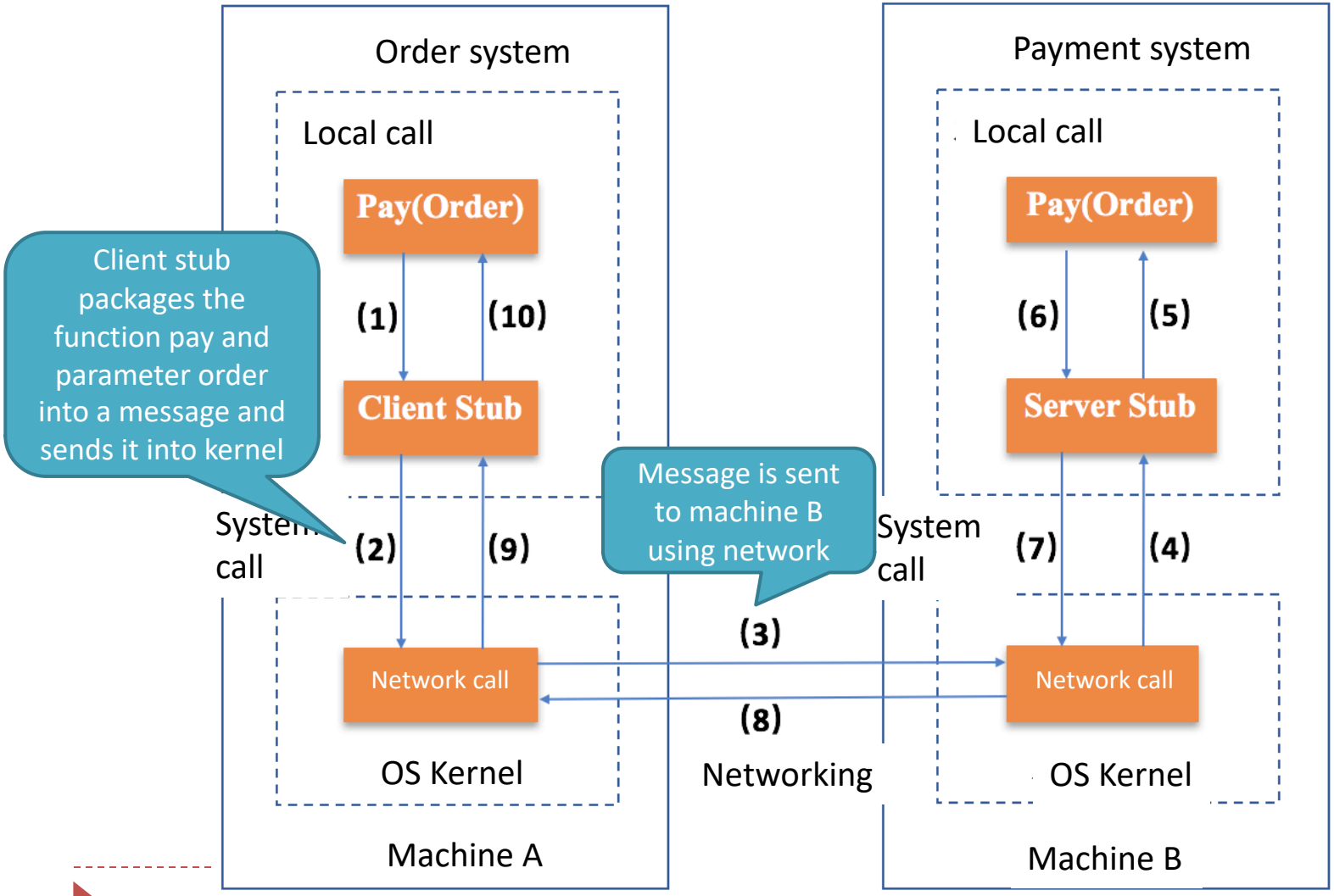
How RPC works?



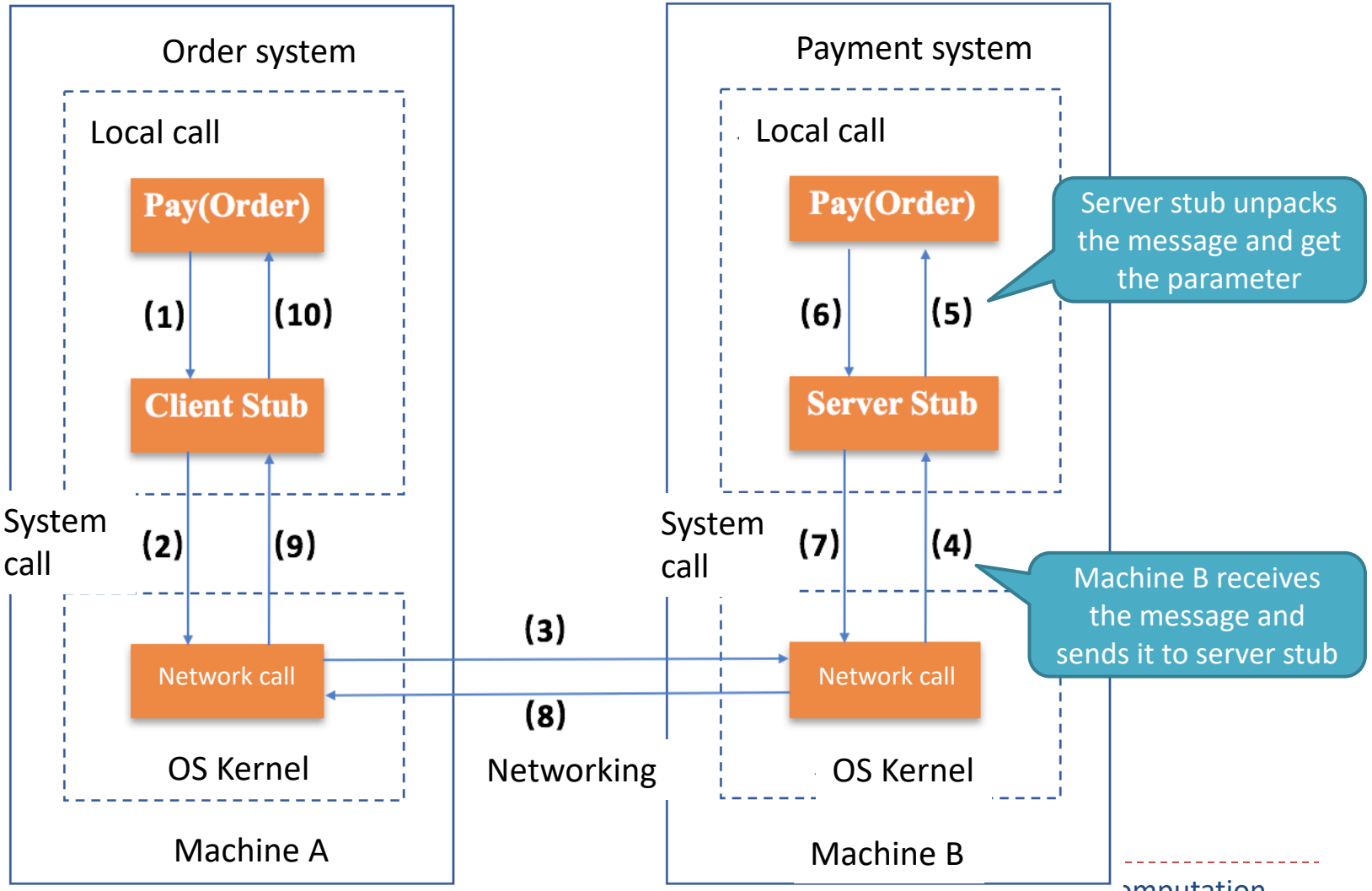
Client And Server Stubs

- Client makes procedure call (just like a local procedure call) to the client stub
- Server is written as a standard procedure
- Stubs take care of packaging arguments and sending messages
- Packaging parameters is called marshalling

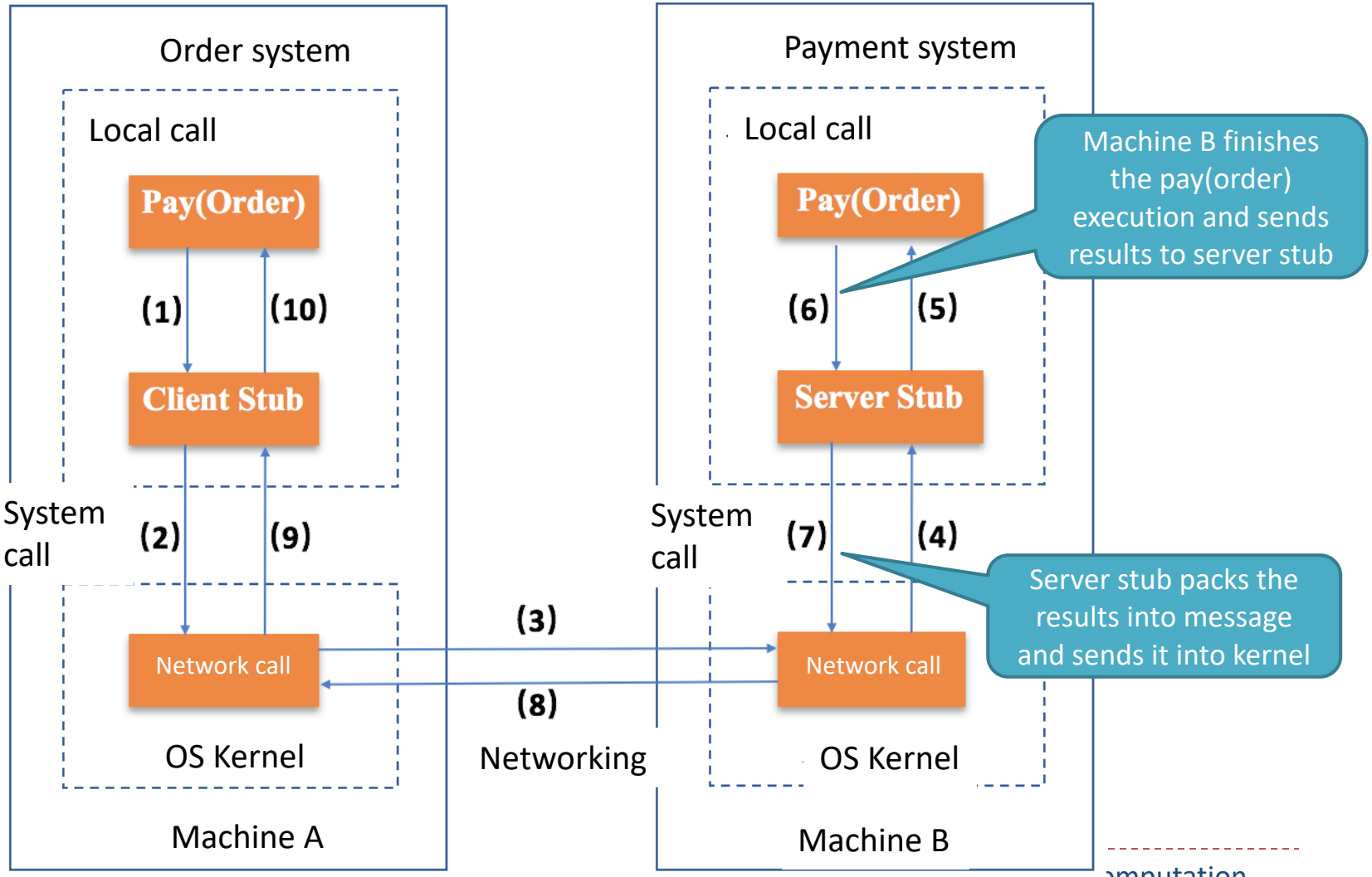
How RPC works?



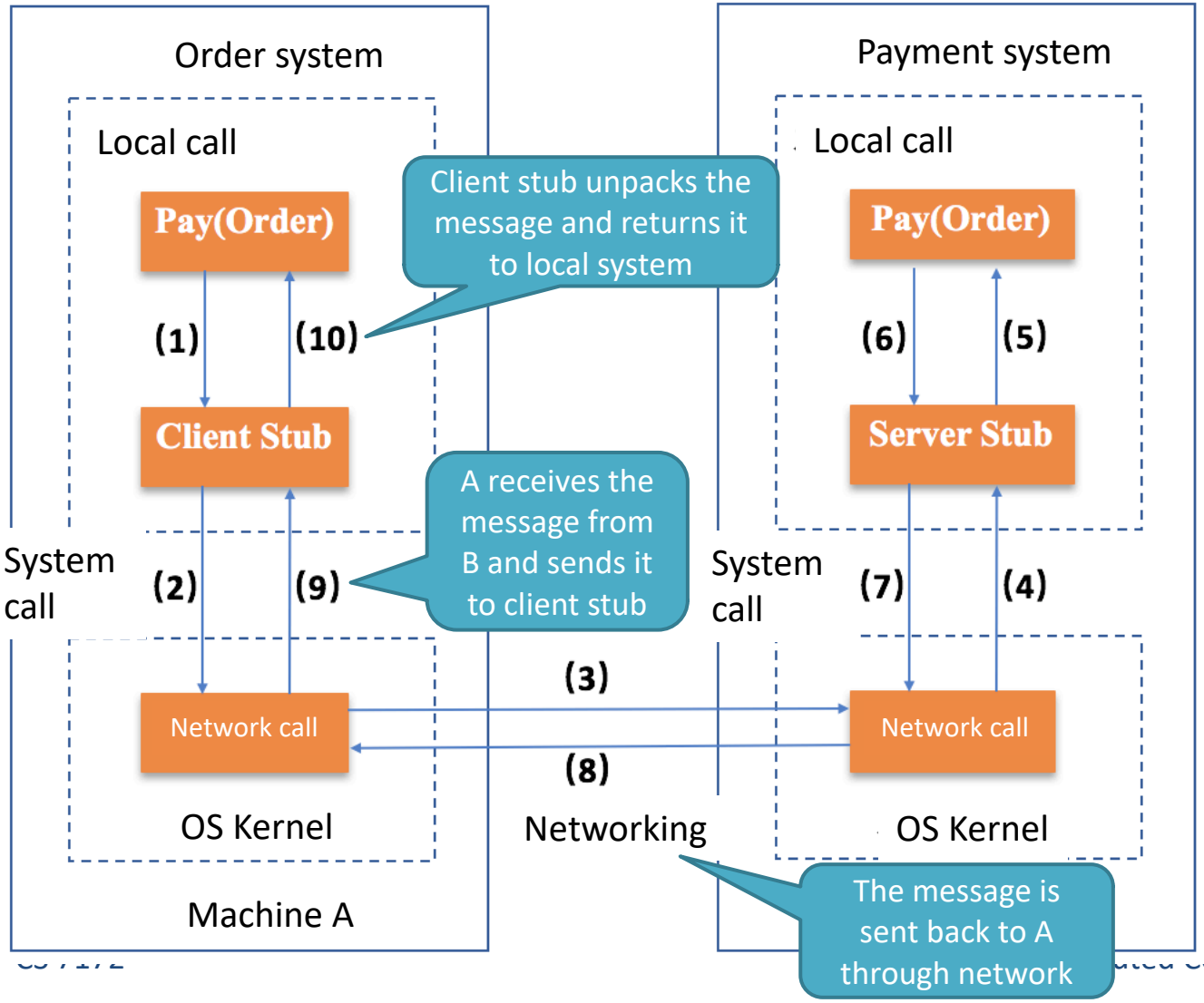
How RPC works?



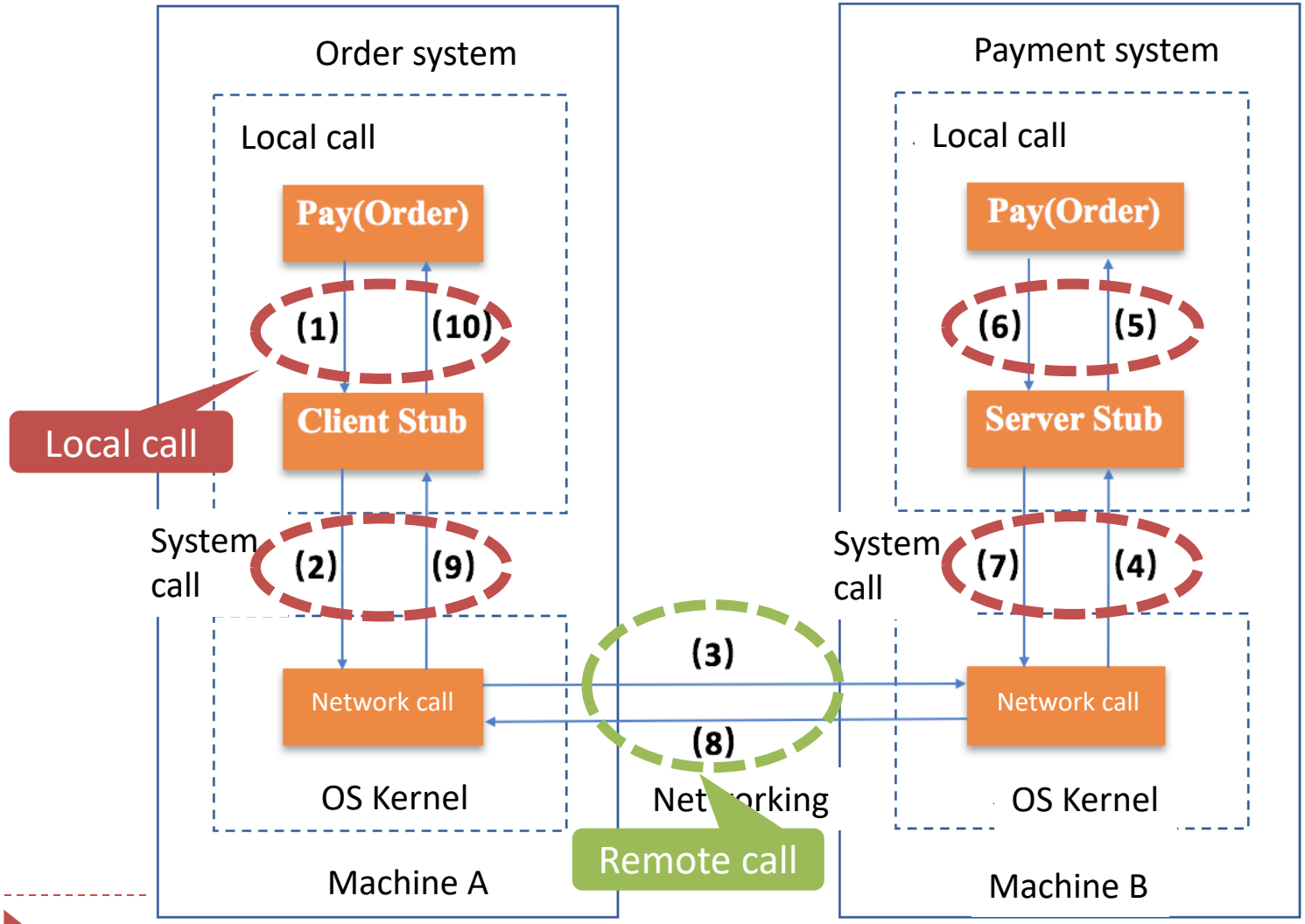
How RPC works?



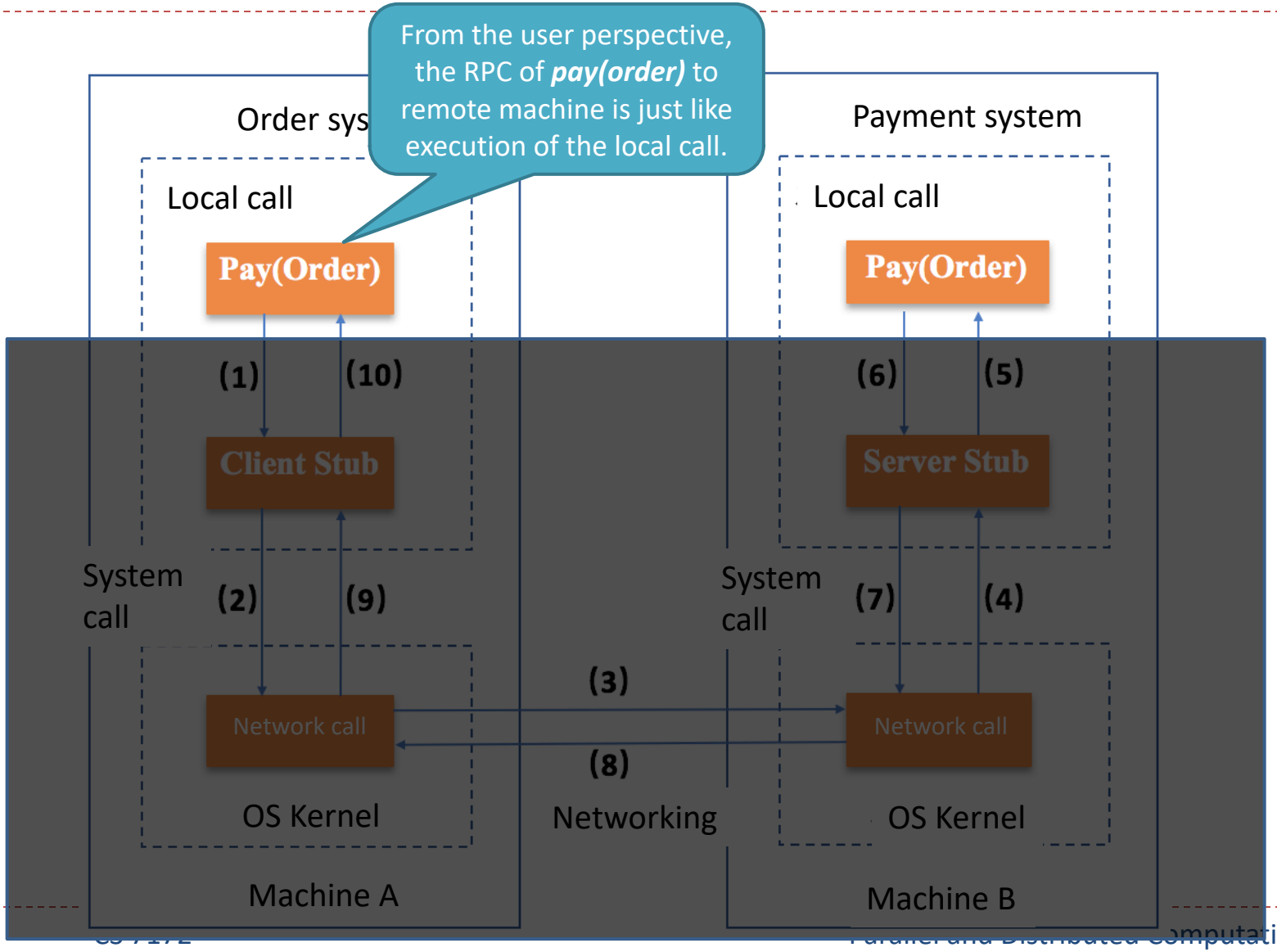
How RPC works?



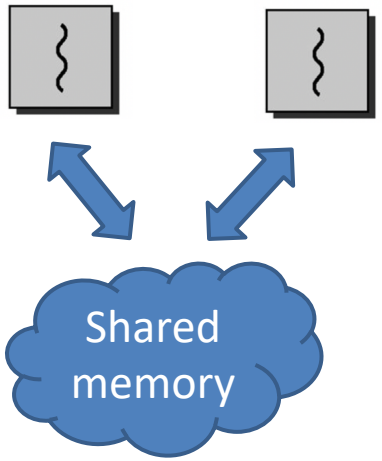
How RPC works?

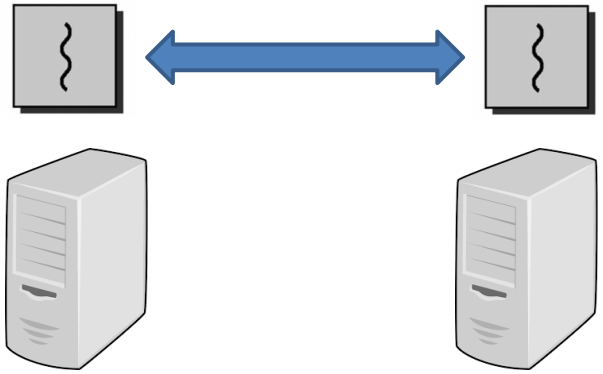


How RPC works?



Local call vs. Remote call implementation

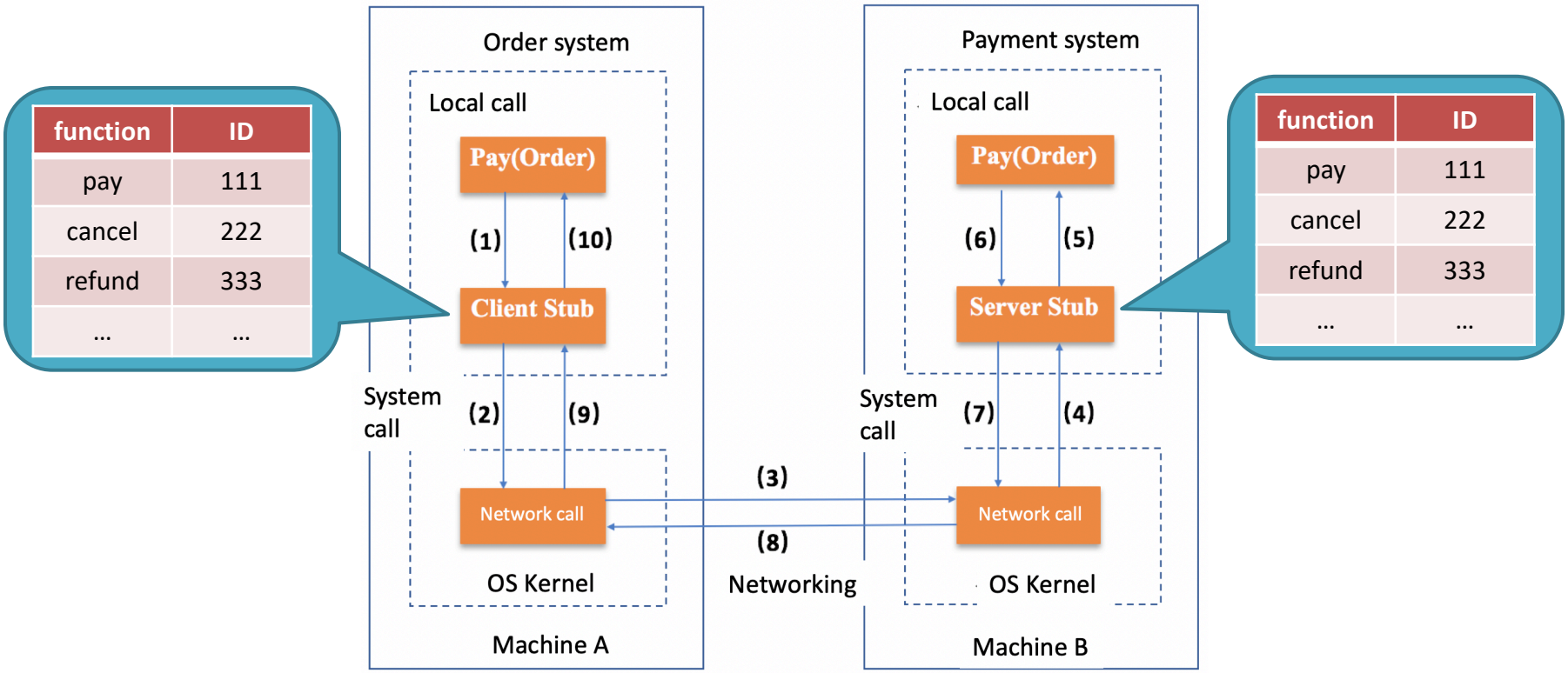




- Shared memory passing
- User can use ***{function + parameter}*** to do local function call
- Under the same memory address

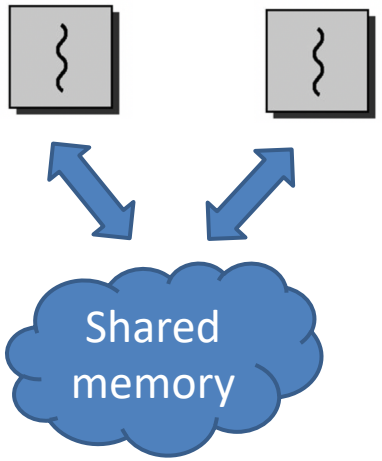
- Network passing
- User use ***{ID + function + parameter}*** to do remote function call due to different memory space
- Under the different memory address

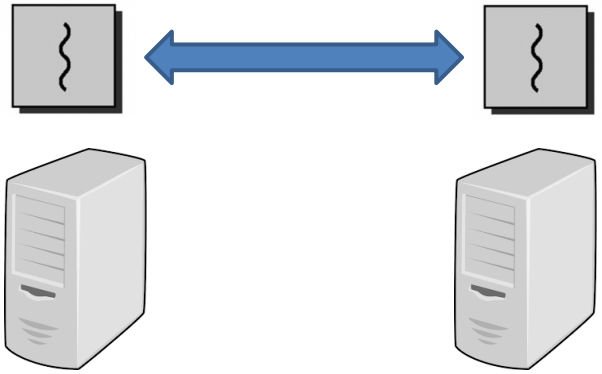
Local call vs. Remote call implementation



The ID is used to differentiate the function address under different memory addresses

Local call vs. Remote call implementation

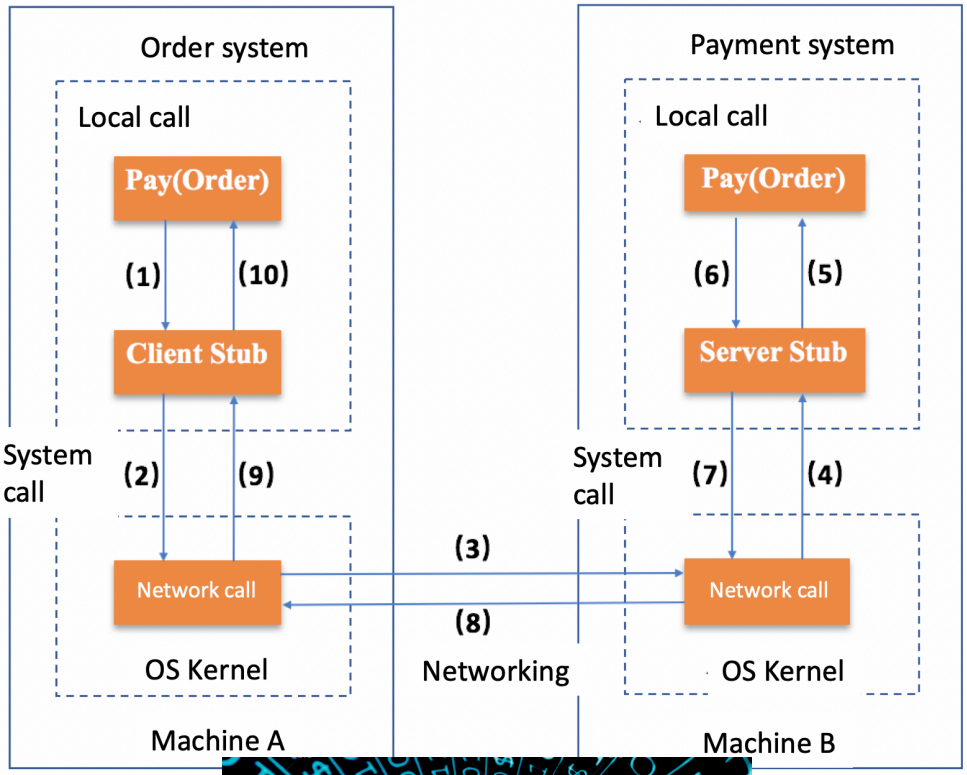




- Shared memory passing
- Memory can store different parameter types (e.g., int, double, string, object, …)

- Network passing (***binary stream***)
- Cannot pass parameter type directly
- The passing messages must be ***serialized and deserialized***

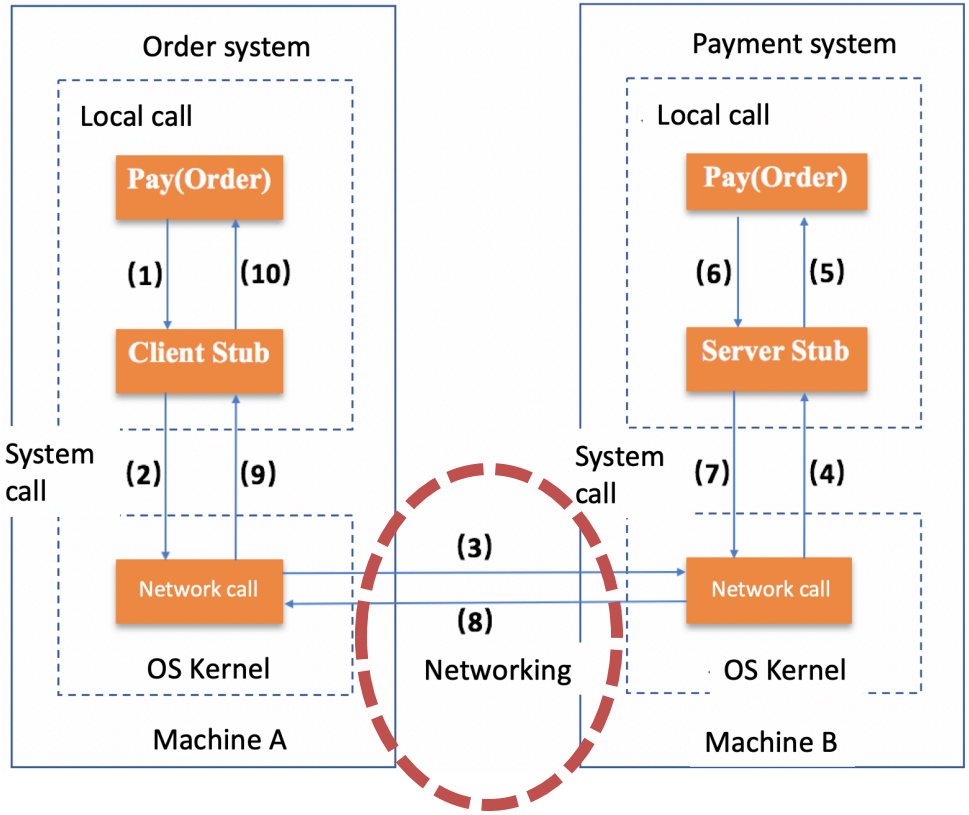
Local call vs. Remote call implementation



Compared to the local call, the RPC messages must be serialized and deserialized

Local call vs. Remote call implementation

- The RPC needs network transmission protocol for parameter passing and receiving
- Most of the RPC uses TCP for the protocol

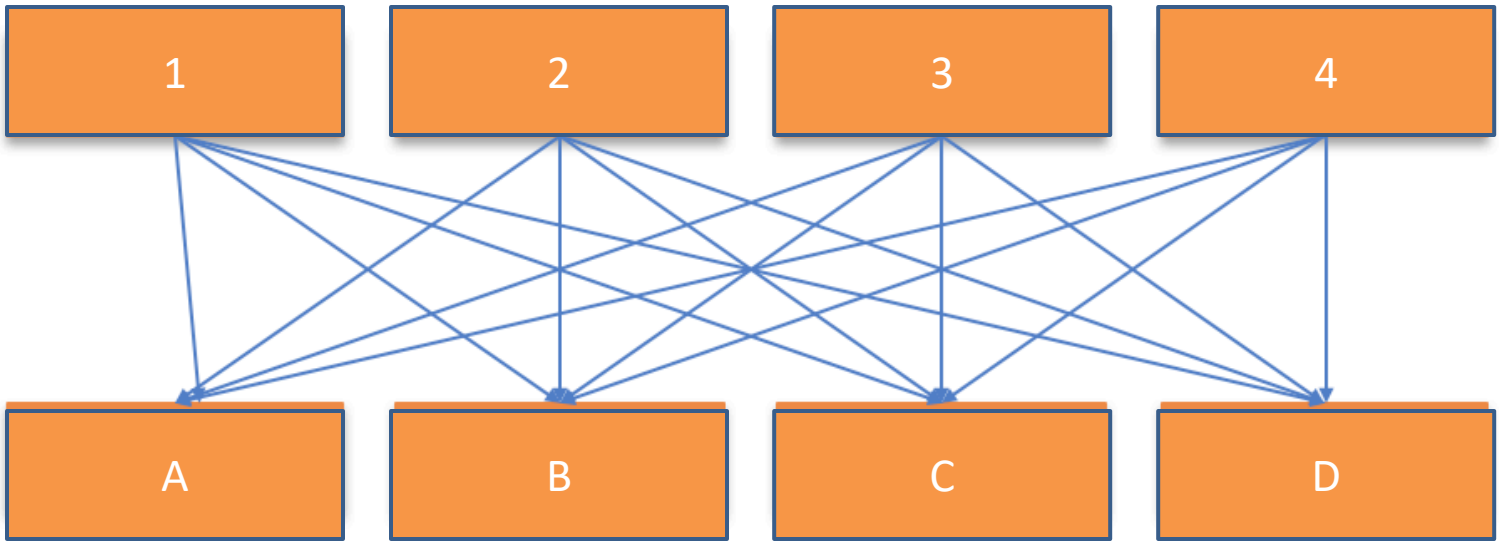


Network transmission protocol

Differences of Local call vs. Remote call implementation

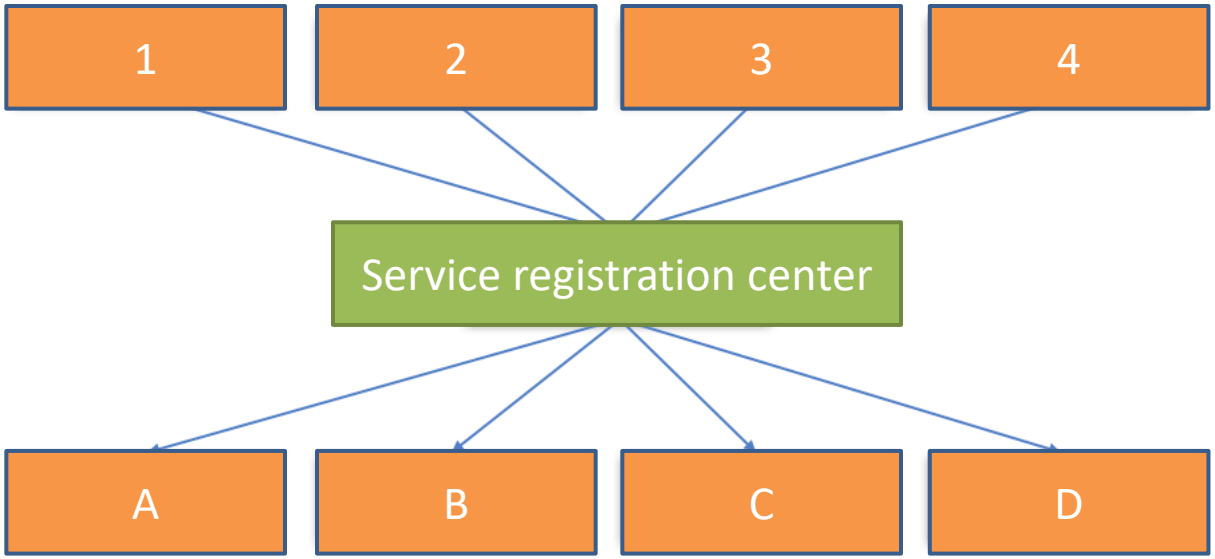
- Call ID and function mapping
- Serialization and deserialization
- Network transmission protocol

Potential Issues of RPC



- Suppose we have n service providers and m service callers
- The number of call relationship will be n*m, which makes the communication overhead too significant

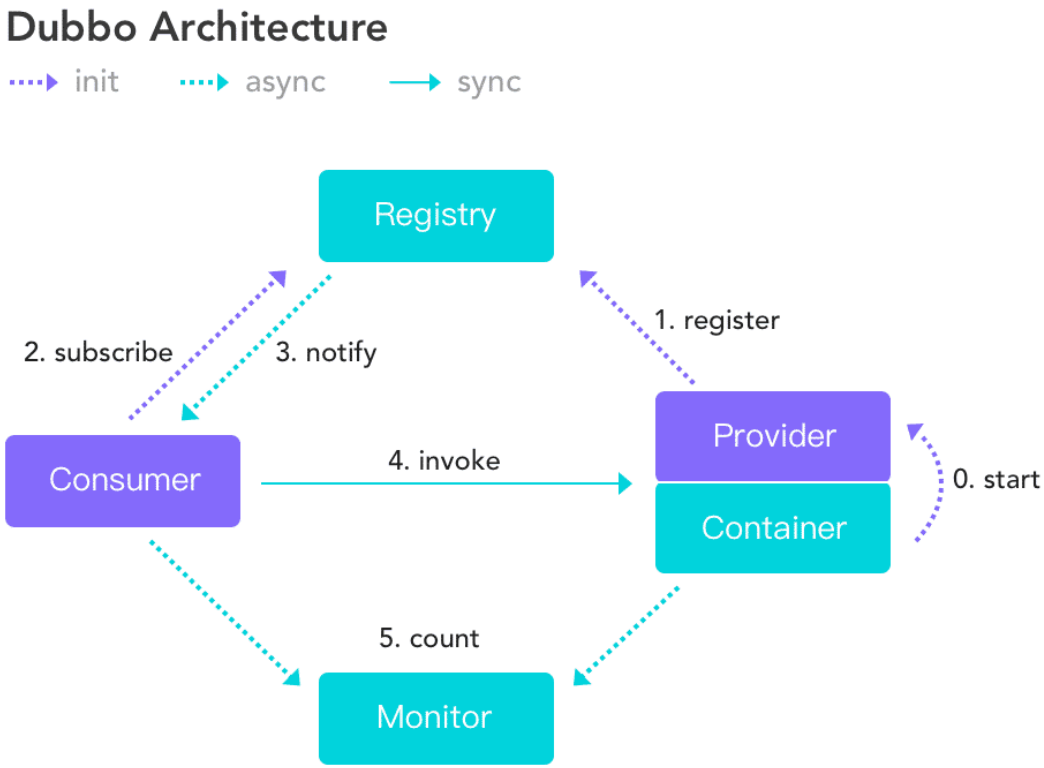
Service Registration Center for RPC



- Service providers register services on service registration center
- Service callers visit needed services from service registration center
- Such the design can reduce the communication volume significantly

Example: Apache Dubbo

- Apache Dubbo™ is a high-performance, java based open source RPC framework.

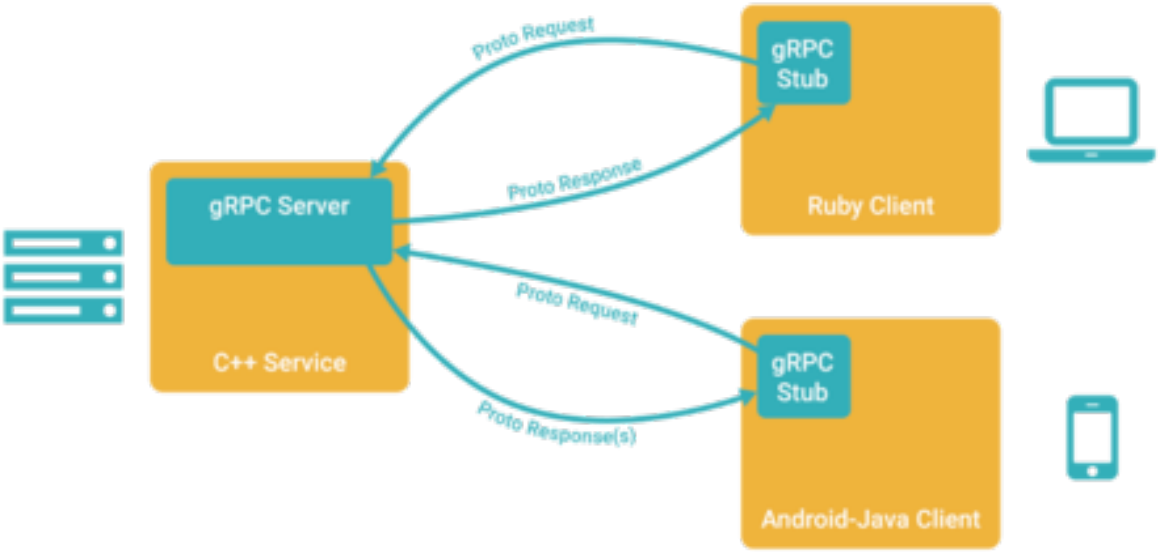


- http://dubbo.apache.org/

Example: gRPC

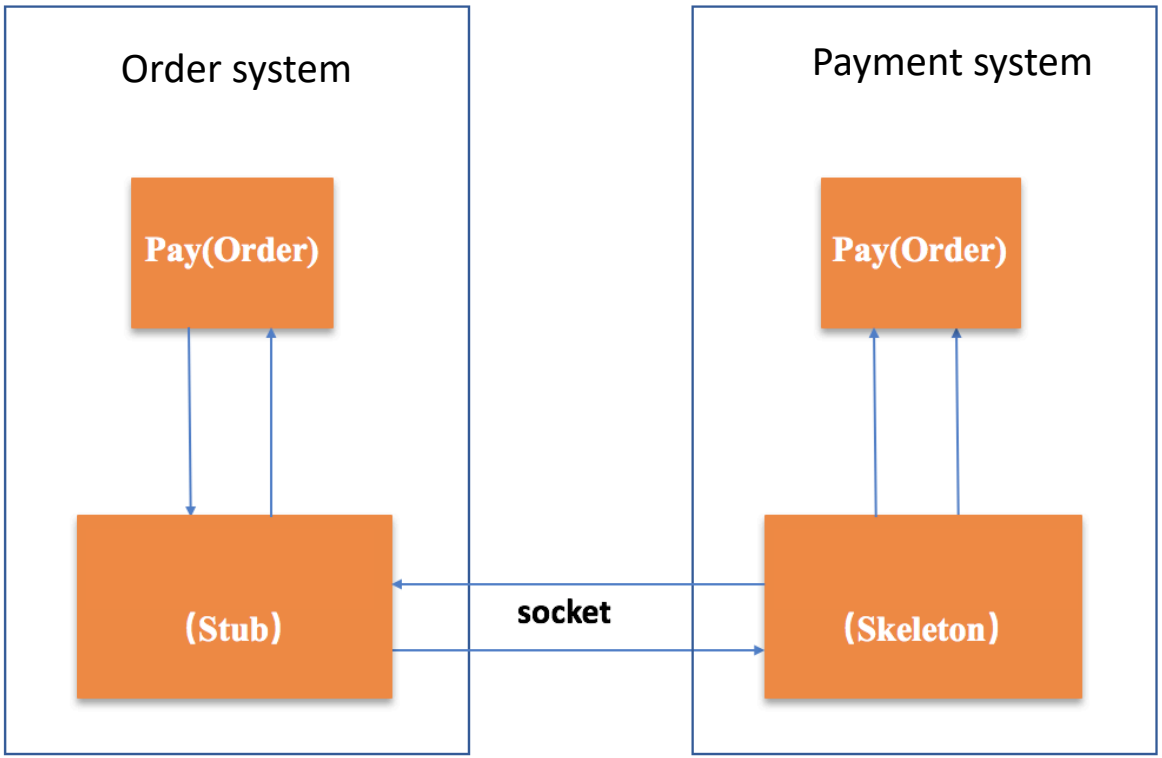
- gRPC is a modern open source high performance RPC framework that can run in any environment. It can efficiently connect services in and across data centers with pluggable support for load balancing, tracing, health checking and authentication.

- https://grpc.io/



A variation of RPC: RMI (Remote Method Invocation)

- RMI is a Java-based application programming interface that enables objects running on the local Java virtual machine to call objects on the remote Java virtual machine as if they were local objects.



An example of RMI

```java
package com.mkyong.rmiinterface;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RMIInterface extends Remote {

    public String helloTo(String name) throws RemoteException;

}
```

RMIInterface has only one method;

it receives a String parameter and returns String.

An example of RMI

```java
package com.mkyong.rmiserver;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

import com.mkyong.rmiinterface.RMIInterface;

public class ServerOperation extends UnicastRemoteObject implements RMIInterface{

    private static final long serialVersionUID = 1L;

    protected ServerOperation() throws RemoteException {

        super();

    }

    @Override
    public String helloTo(String name) throws RemoteException{

        System.err.println(name + " is trying to contact!");
        return "Server says hello to " + name;

    }

    public static void main(String[] args){

        try {

            Naming.rebind("//localhost/MyServer", new ServerOperation());
            System.err.println("Server ready");

        } catch (Exception e) {

            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();

        }

    }

}
```

Our Server
extends UnicastRemoteObject and implements the Interface we made before.

In the main method we bind the server on localhost with the name “MyServer”.

An example of RMI

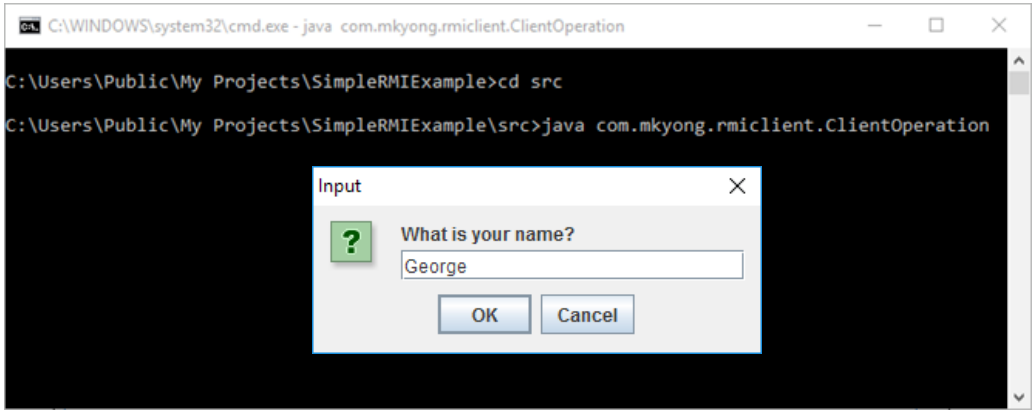
```
package com.mkyong.rmiclient;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

import javax.swing.JOptionPane;

import com.mkyong.rmiinterface.RMIInterface;

public class ClientOperation {

    private static RMIInterface look_up;

    public static void main(String[] args)
        throws MalformedURLException, RemoteException, NotBoundException {

        look_up = (RMIInterface) Naming.lookup("//localhost/MyServer");
        String txt = JOptionPane.showInputDialog("What is your name?");

        String response = look_up.helloTo(txt);
        JOptionPane.showMessageDialog(null, response);

    }

}
```

Remote call server side function helloTo()

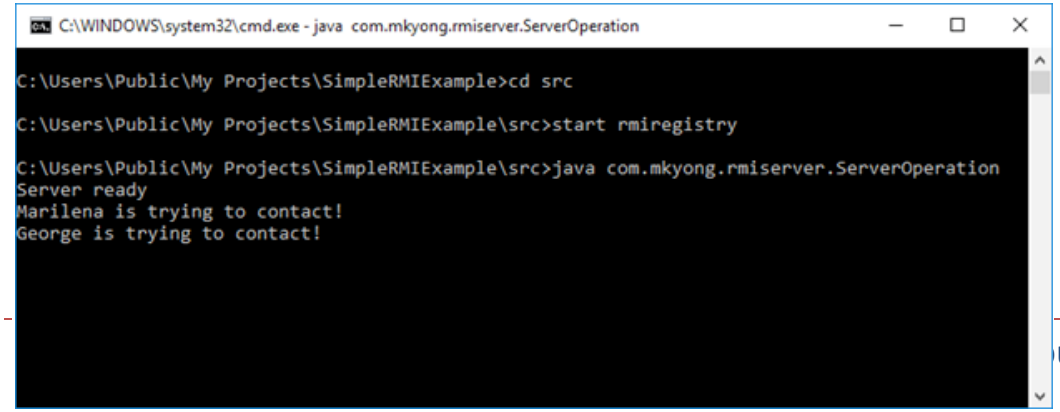
the Client uses an RMIInterface Object that “looks” for a reference for the remote object associated with the name we pass as parameter

An example of RMI

Client

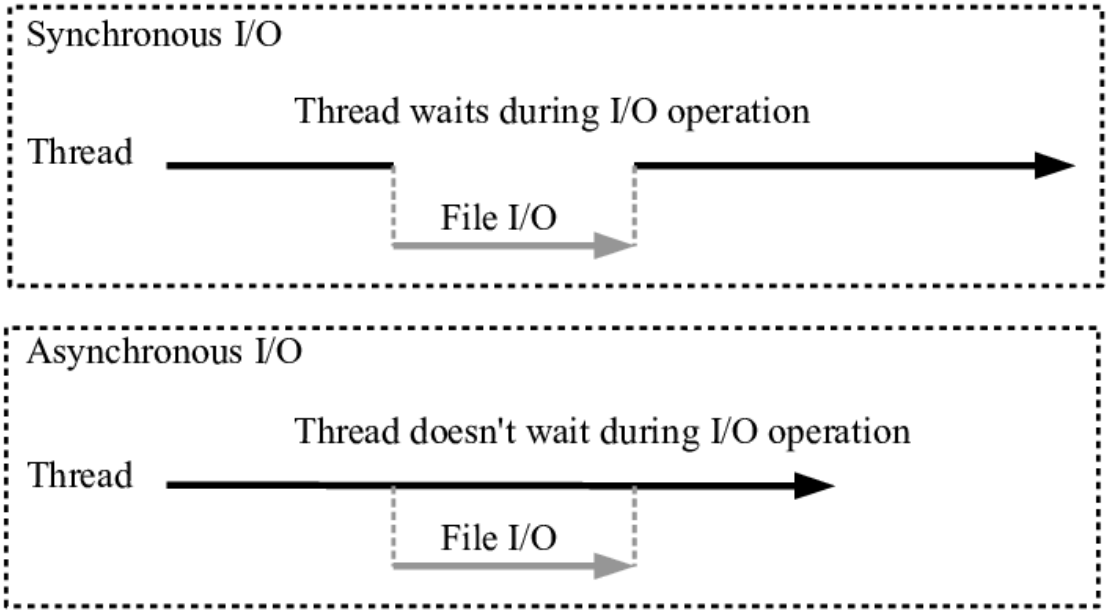


Server



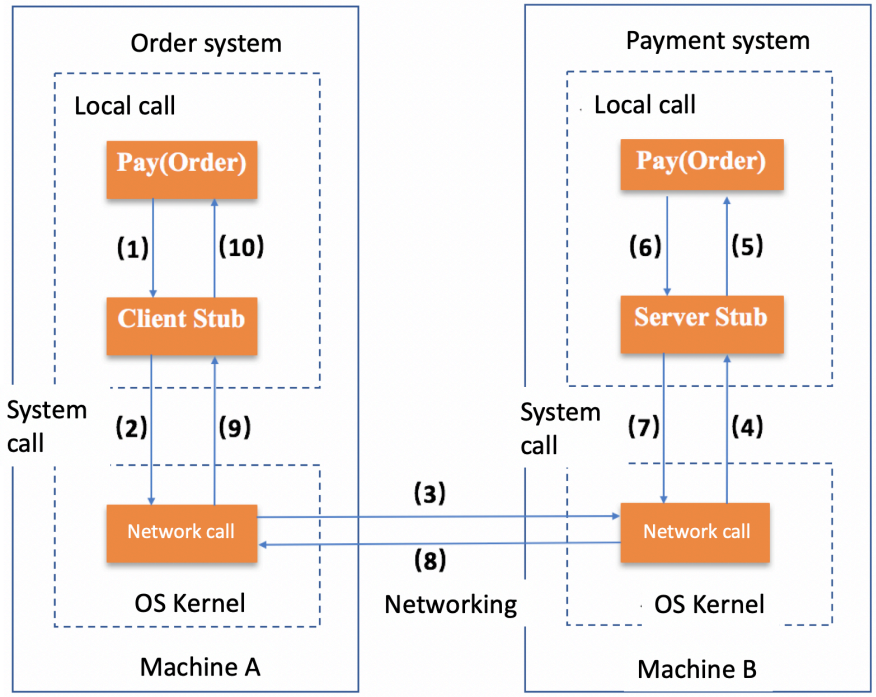
Synchronous vs. asynchronous call

- **Synchronous call:** application is blocked until the operation finishes
- **Asynchronous call:** application is not blocked and can does other tasks during the operation is processing



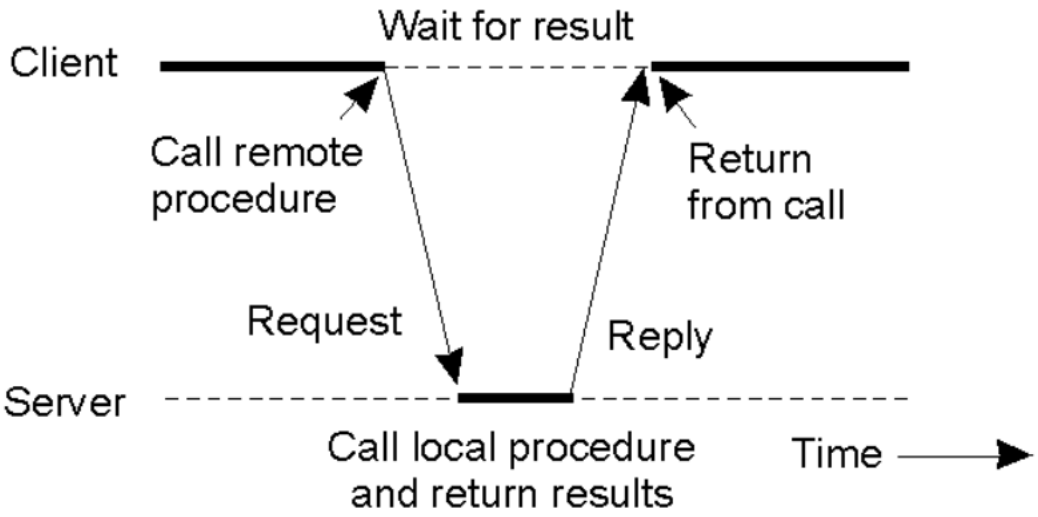
Synchronous vs. asynchronous call in RPC

- **Synchronous call:** is used for requiring exact results or returning the results directly
  - E.g., weather
- **Asynchronous call:** is used for high response efficiency but low exact return results
  - E.g., video decoding



Synchronous vs. asynchronous call in RPC

- Usually, RPCs are blocking
  - Thus, also useful for synchronization



Conclusion

- What is RPC and why we need RPC?
- How RPC works?
- Implementation details of local call and RPC
- Examples of RPC
- Remote Method Invocation (RMI)
- Synchronous and asynchronous in RPC