Remote Procedure Call

Malintha Adikari

Remote Procedure Call (RPC)

- **Remote procedure call** (**RPC**) is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction

- The programmer writes essentially the same code whether the subroutine is local to the executing program, or remote

- This is a form of client–server interaction (caller is client, executor is server)

- In the object-oriented programming paradigm, RPC calls are represented by remote method invocation (RMI)

Remote Procedure Call (RPC)

- Avoid explicit message exchange between process
- Basic idea is to allow a process on a machine to call procedure
- Make a remote procedure possibly look like a local one

What is Stub

- A **stub** is a piece of code that converts parameters passed between client and server during a remote procedure call (RPC).
- The client and server use different address spaces, so parameters used in a procedure call have to be converted, otherwise the values of those parameters could not be used, because pointers to parameters in one computer's memory would point to different data on the other computer.
- Stubs perform the conversion of the parameters, so a remote procedure call looks like a local function call for the remote computer.

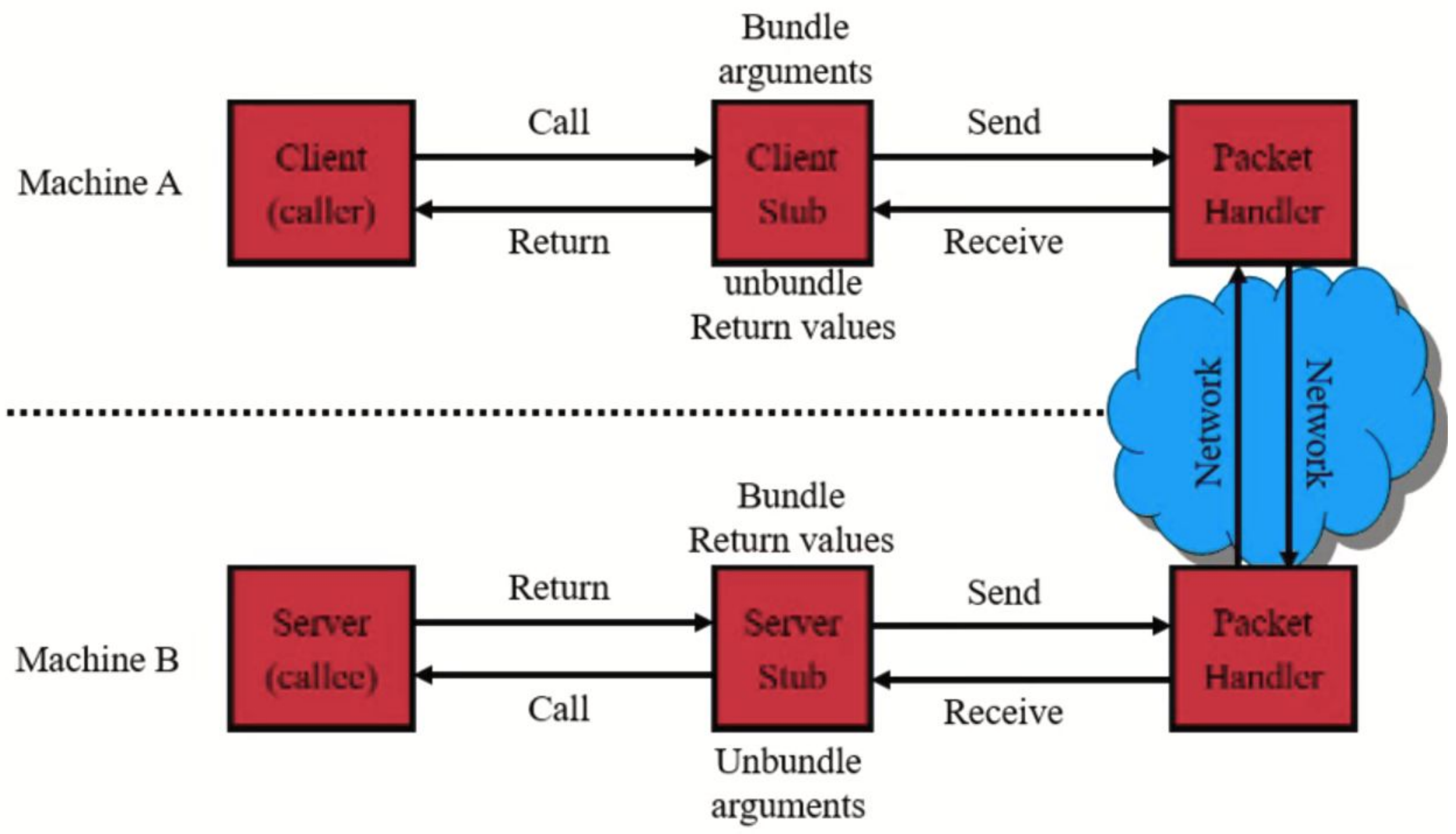
Client Stub

- The client stub module provides surrogate entry points on the client.
- When the client application makes a call to the remote procedure, its call first goes to the surrogate routine in the client stub file.
- The client stub routine performs the following functions:
  - Marshals arguments - The client stub packages input arguments into a form that can be transmitted to the server.
  - Calls the client run-time library to transmit arguments to the remote address space and invokes the remote procedure in the server address space.
  - Unmarshals output arguments - The client stub unpacks output arguments and returns to the caller.

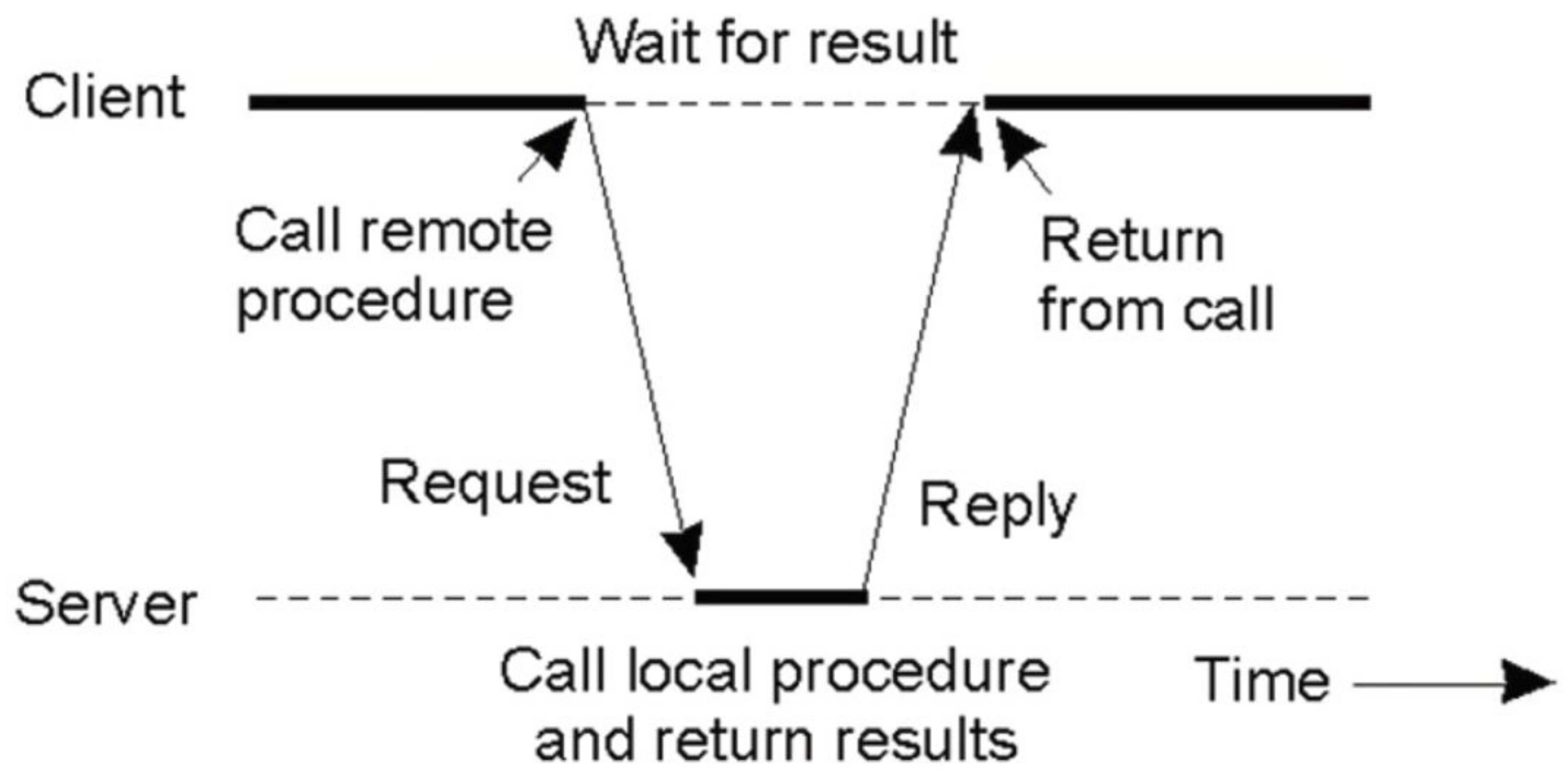
Server Stub

- The server stub provides surrogate entry points on the server.
- When a server stub routine is invoked by the RPC run-time library, it performs the following functions:
    - Unmarshals input arguments (unpacks the arguments from their transmitted formats).
    - Calls the actual implementation of the procedure on the server.
    - Marshals output arguments (packages the arguments into the transmitted forms)

Remote Procedure Call (RPC)

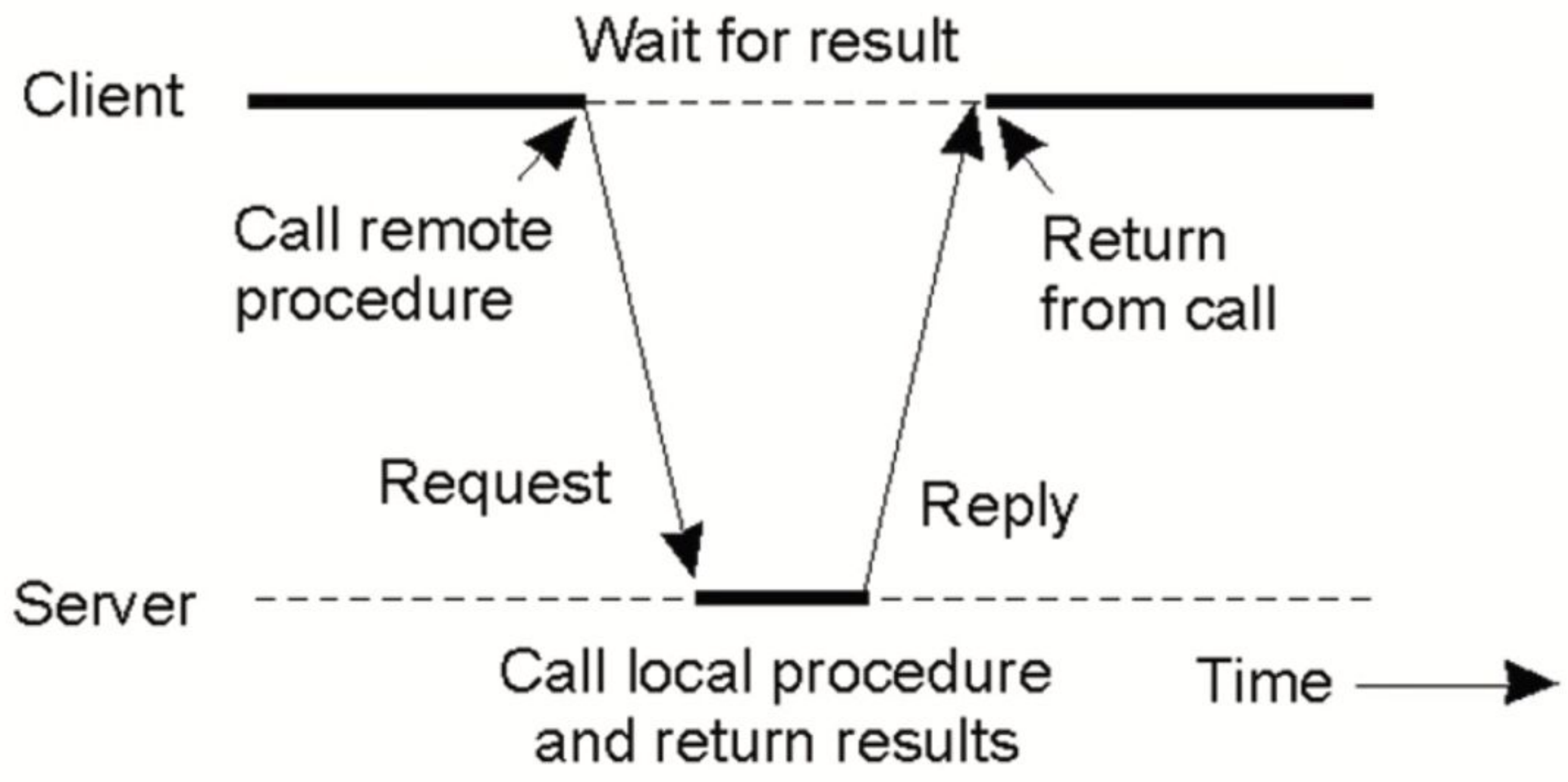


Remote Procedure Call (RPC)

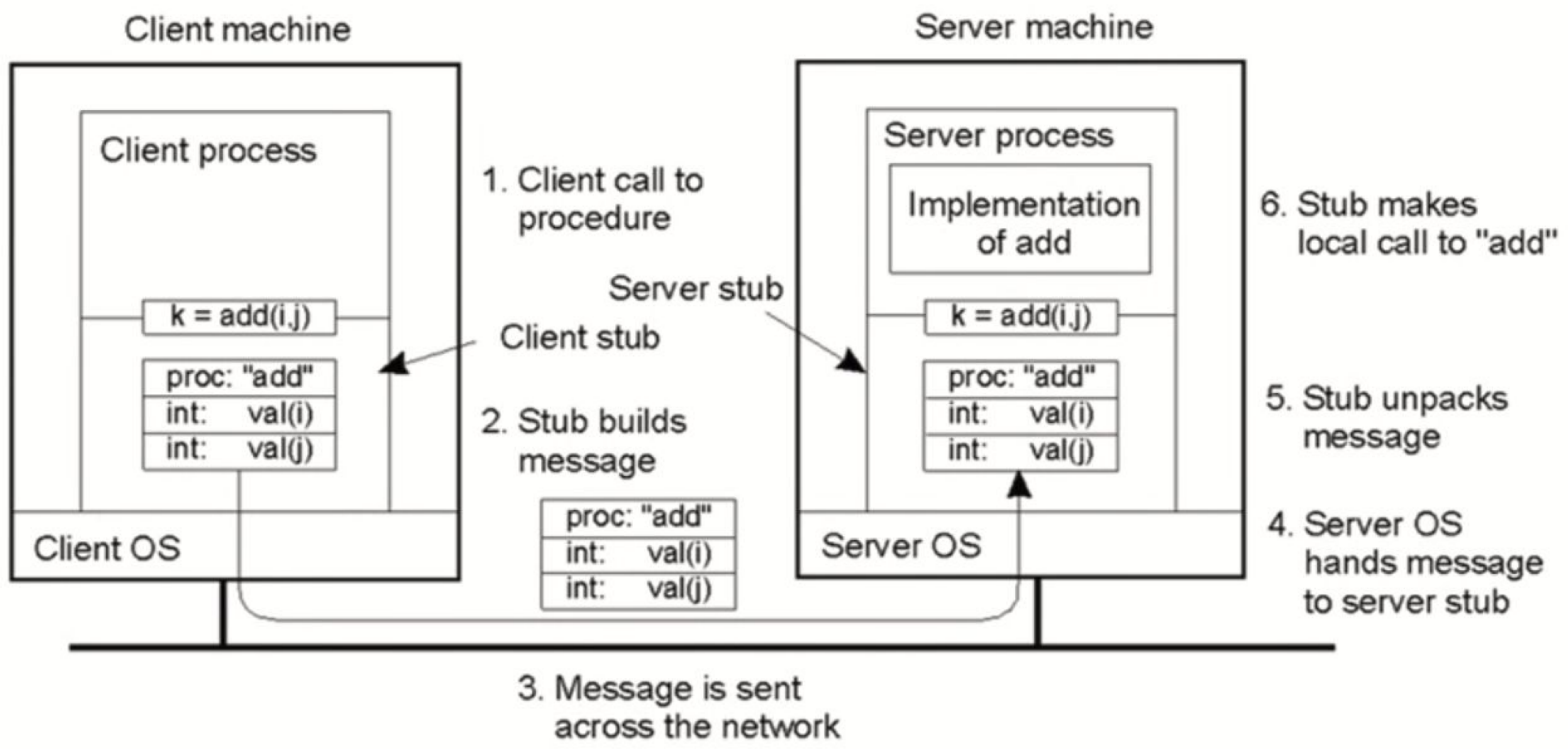


Remote Procedure Call Steps

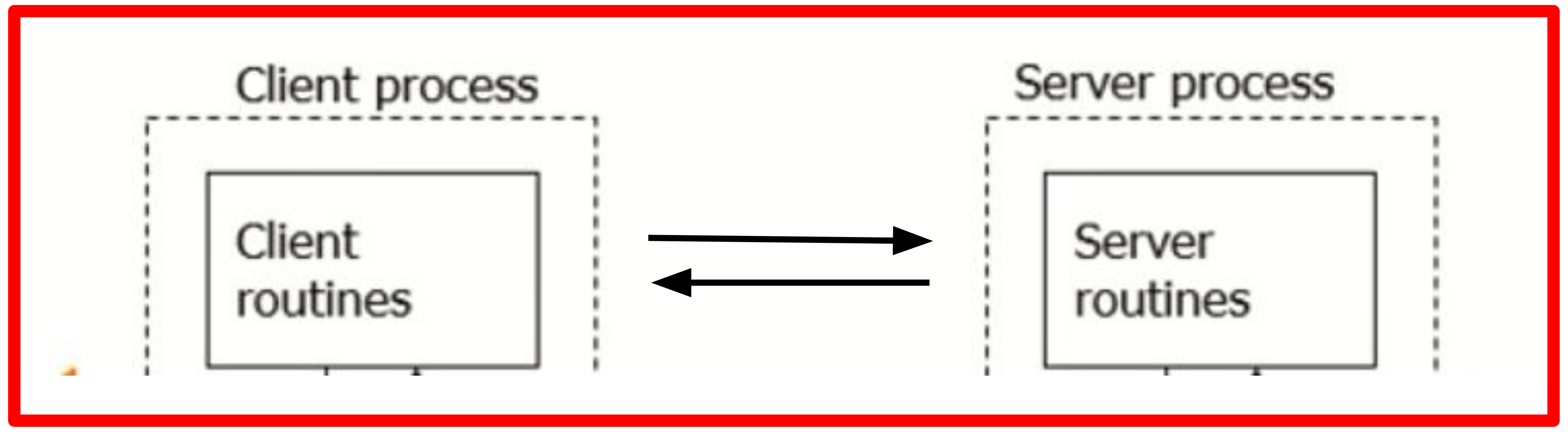
1. Client procedure calls client stub
2. Client stub build message, and calls local OS
3. Client OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, call server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS gives message to client stub
9. Stub unpacks result, return to client



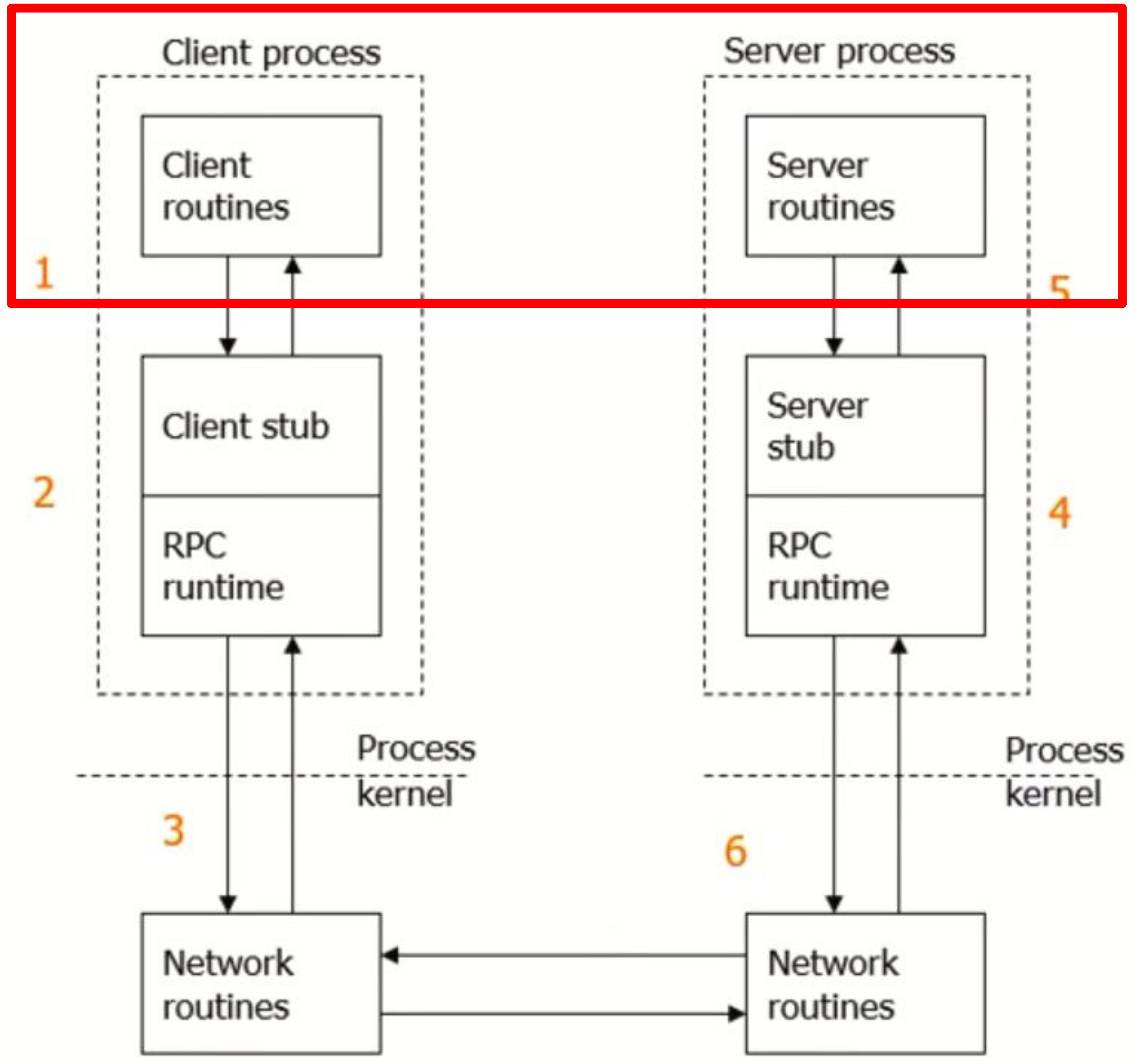
Remote Procedure Call



Remote Procedure Call



Remote Procedure Call



Remote Procedure Call

- Client calls local procedure on the client stub
- The client stub acts as a proxy and marshalls the call and the args
- The client stub send this to the remote system (via TCP/UDP)
- The server stub unmarshalls the call and args from the client
- The server stub calls the actual procedure on the server
- The server stub marshalls the reply and sends it back to the client.

Remote Procedure Call

- Client call s local procedure on the client stub
- The client stub acts as a proxy and marshalls the call and the args
- The client stub send this to the remote system (via TCP/UDP)
- The server stub unmarshalls the call and args from the client
- The server stub calls the actual procedure on the server
- The server stub marshalls the reply and sends it back to the client.

Transferring data structure used in remote procedure call from one address space to another

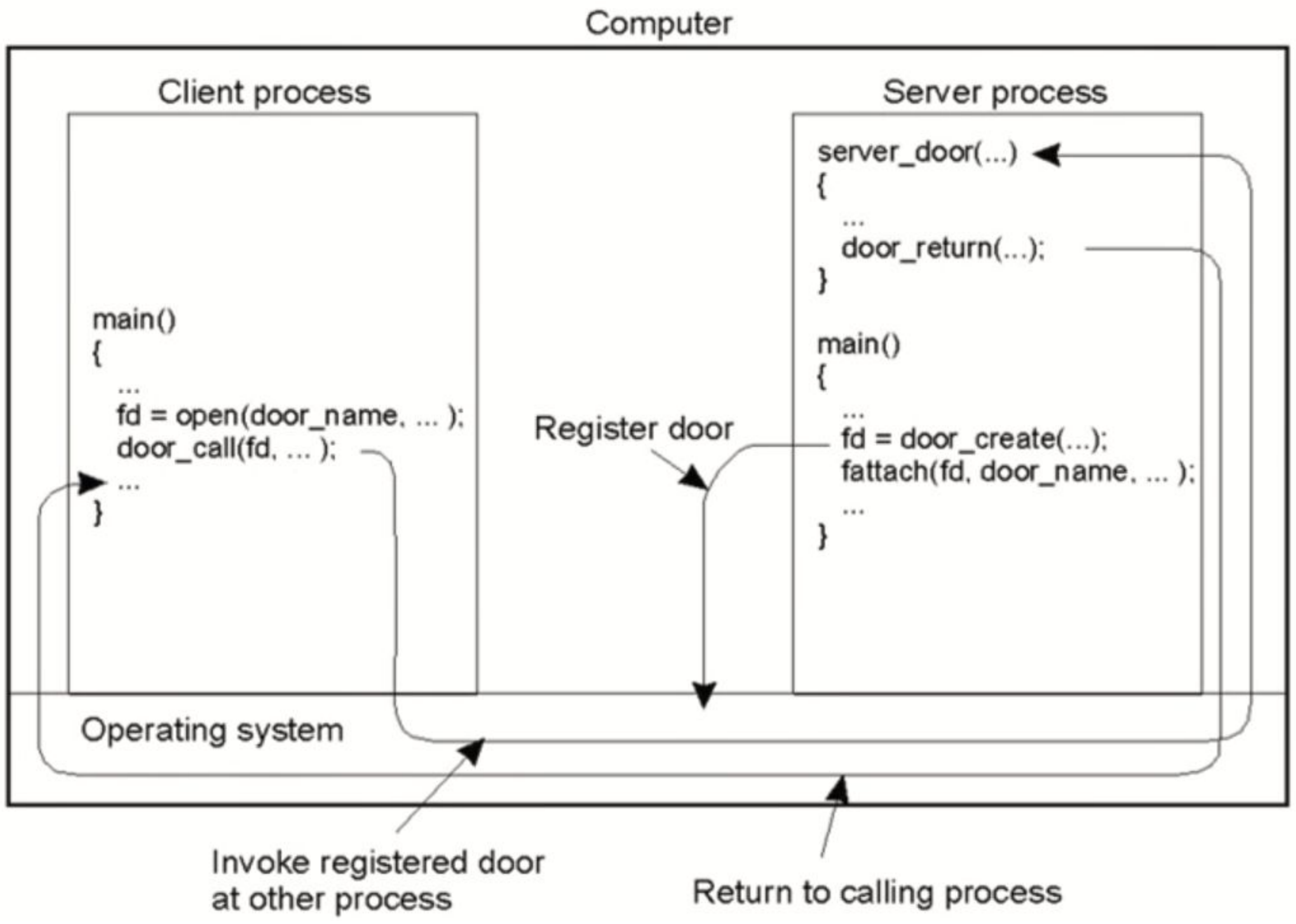
Lightweight RPCs

- Many RPCs occur between client and server on same machine
  - Need to optimize RPCs for this special case => use a lightweight RPC mechanism (LRPC)
- Server **S** exports interface to remote procedures
- Client **C** on same machine imports interface
- OS kernel creates data structure including an argument stack shared between S and C

Lightweight RPCs

- RPC Execution
  - Push argument onto stack
  - Trap to kernel
  - Kernel changes mem map of client to server address space
  - Client thread executes procedure (OS upcall)
  - Thread traps to kernel upon completion
  - Kernel changes the address space back and returns control to client

DOORs



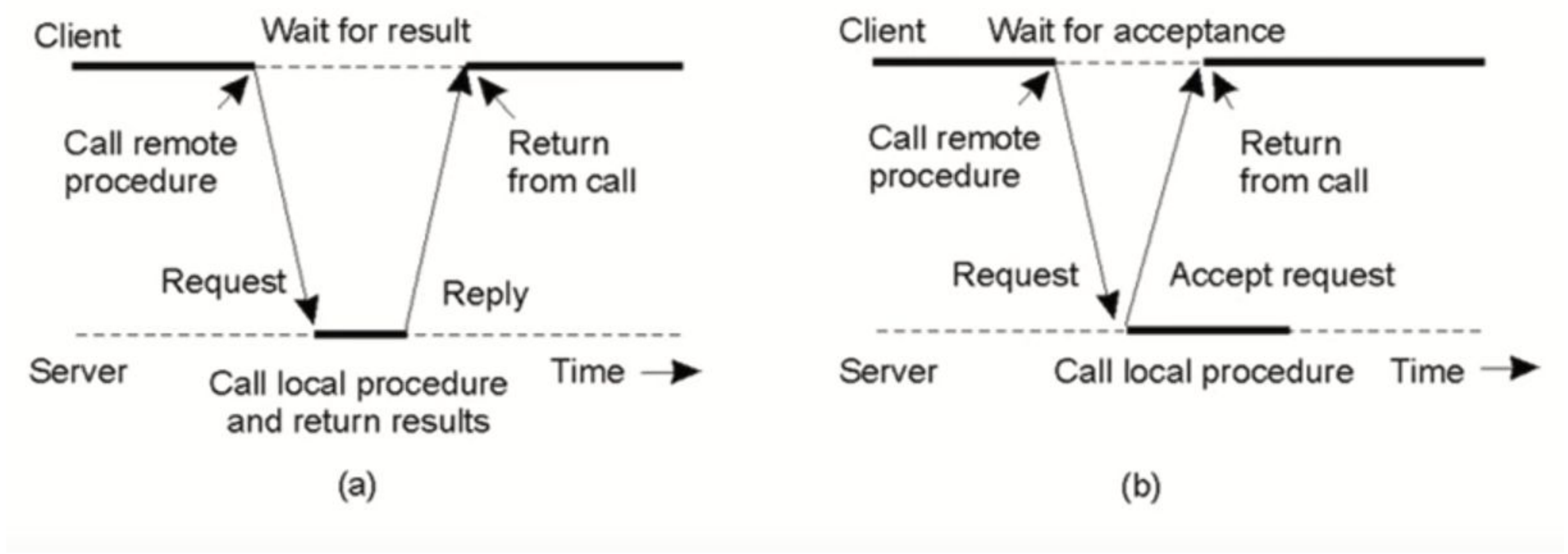
```
main()
{
  ...
  fd = open(door_name, ... );
  door_call(fd, ... );
  ...
}
```

```
server_door(...)
{
  ...
  door_return(...);
}

main()
{
  ...
  fd = door_create(...);
  fattach(fd, door_name, ... );
  ...
}
```

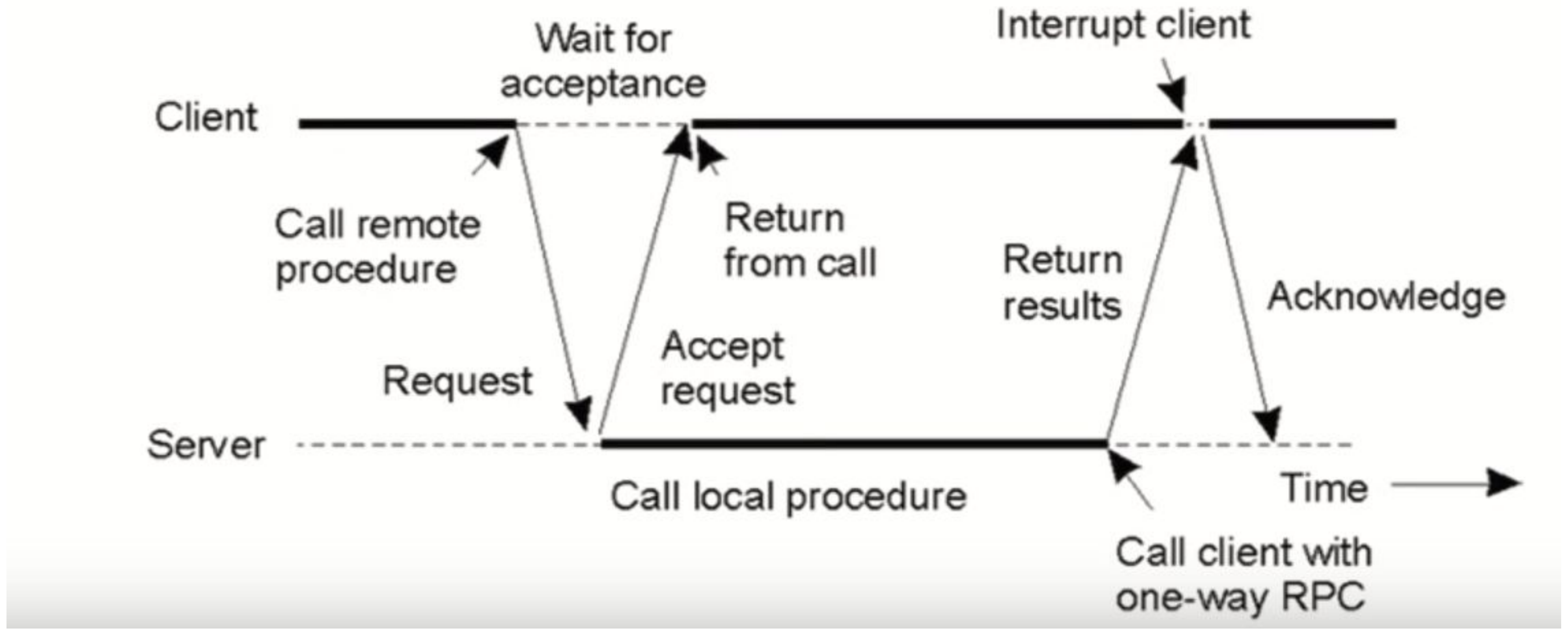
Asynchronous Remote Procedure Call

- A synchronous operation blocks a process till the operation completes.
- An asynchronous operations is non-blocking and only initiates the operation.



Asynchronous Remote Procedure Call

- Asynchronous message passing allows more parallelism
- Since a process does not block, it can do some computation while the message is in transit
- To receive, a process can express its interest in receiving message on multiple ports simultaneously



Thanks!

Any questions?