CSE 601: Distributed Systems

Toukir Ahammed

Communication

- Interprocess communication is at the heart of all distributed systems
- Communication in distributed systems
  - ways that processes on different machines can exchange information
- Two widely used models for communication:
  - Remote Procedure Call (RPC), and
  - Message-Oriented Middleware (MOM)

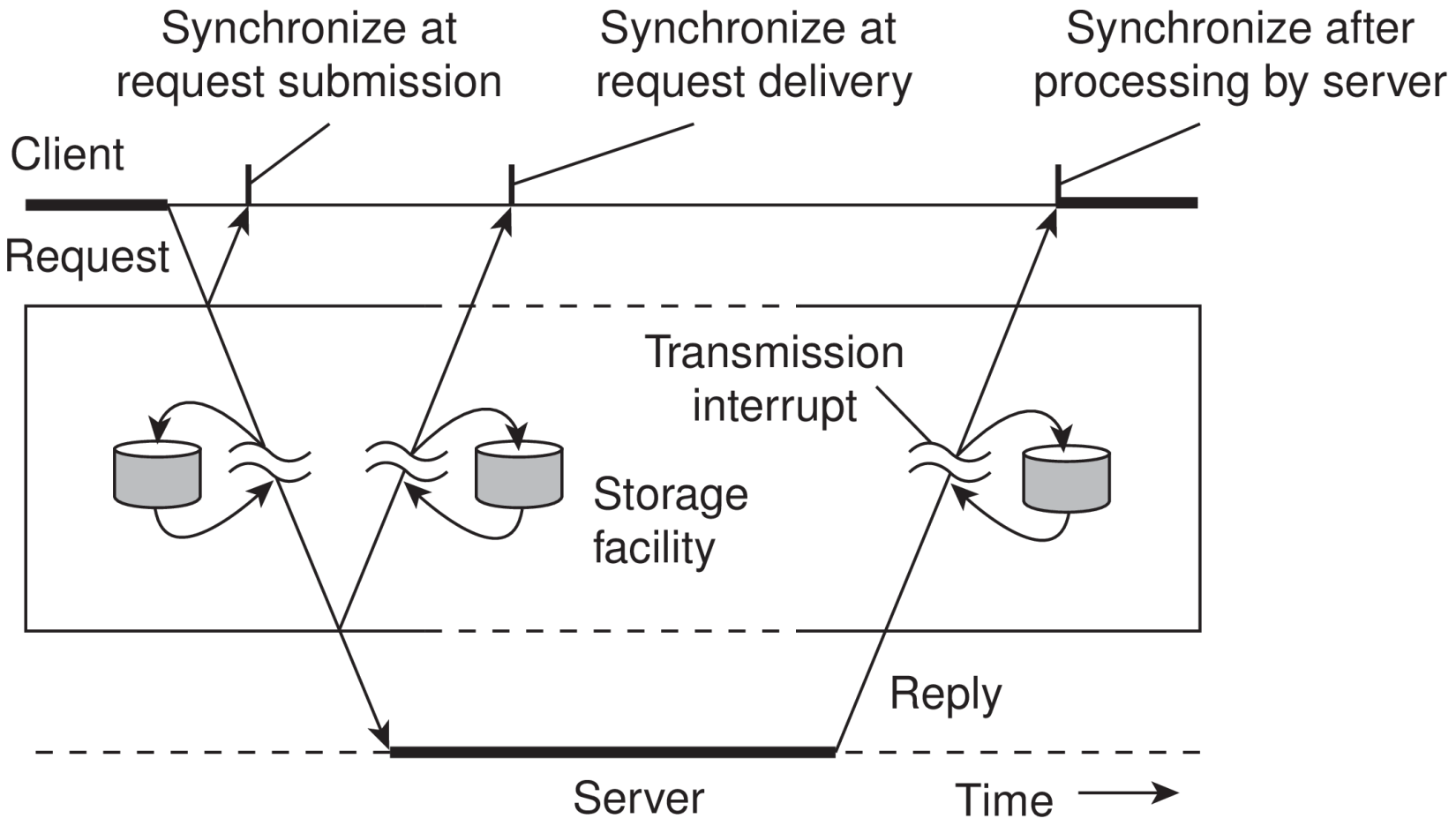
Middleware

- Middleware is invented to provide common services and protocols that can be used by many different applications
  - A rich set of communication protocols
  - (Un)marshaling of data, necessary for integrated systems
  - Naming protocols, to allow easy sharing of resources
  - Security protocols for secure communication
  - Scaling mechanisms, such as for replication and caching
- Middleware can offer various alternatives in communication to applications
- Middleware can be viewed as an additional service in client-server computing

Middleware

- Consider an e-mail system, the core of the mail delivery system can be seen as a middleware communication service.
- Each host runs a user agent allowing users to compose, send, and receive e-mail.
- A sending user agent passes such mail to the mail delivery system, expecting it, in turn, to eventually deliver the mail to the intended recipient.
- Likewise, the user agent at the receiver's side connects to the mail delivery system to see whether any mail has come in. If so, the messages are transferred to the user agent so that they can be read by the user.

Types of Communication



Transient
vs
Persistent

Asynchronous
vs
synchronous

Persistent Communication

- A message that has been submitted for transmission is stored by the communication middleware as long as it takes to deliver it to the receiver.
- The middleware will store the message at one or several of the storage facilities
- As a consequence, it is not necessary for the sending application to continue execution after submitting the message
- Likewise, the receiving application need not be executing when the message is submitted.
- An e-mail system is a typical example in which communication is persistent.

Transient Communication

- A message is stored by the communication system only as long as the sending and receiving application are executing
- If the middleware cannot deliver a message due to a transmission interrupt, or because the recipient is currently not active, it will simply be discarded
- All transport-level communication services offer only transient communication
- If a router cannot deliver a message to the next one or the destination host, it will simply drop the message.

Asynchronous Communication

- A sender continues immediately after it has submitted its message for transmission
- This means that the message is (temporarily) stored immediately by the middleware upon submission

Synchronous Communication

- A sender is blocked until its request is known to be accepted
- There are essentially three points where synchronization can take place
  - until the middleware notifies that it will take over transmission of the request
  - until its request has been delivered to the intended recipient
  - until its request has been fully processed, that is, up to the time that the recipient returns a response

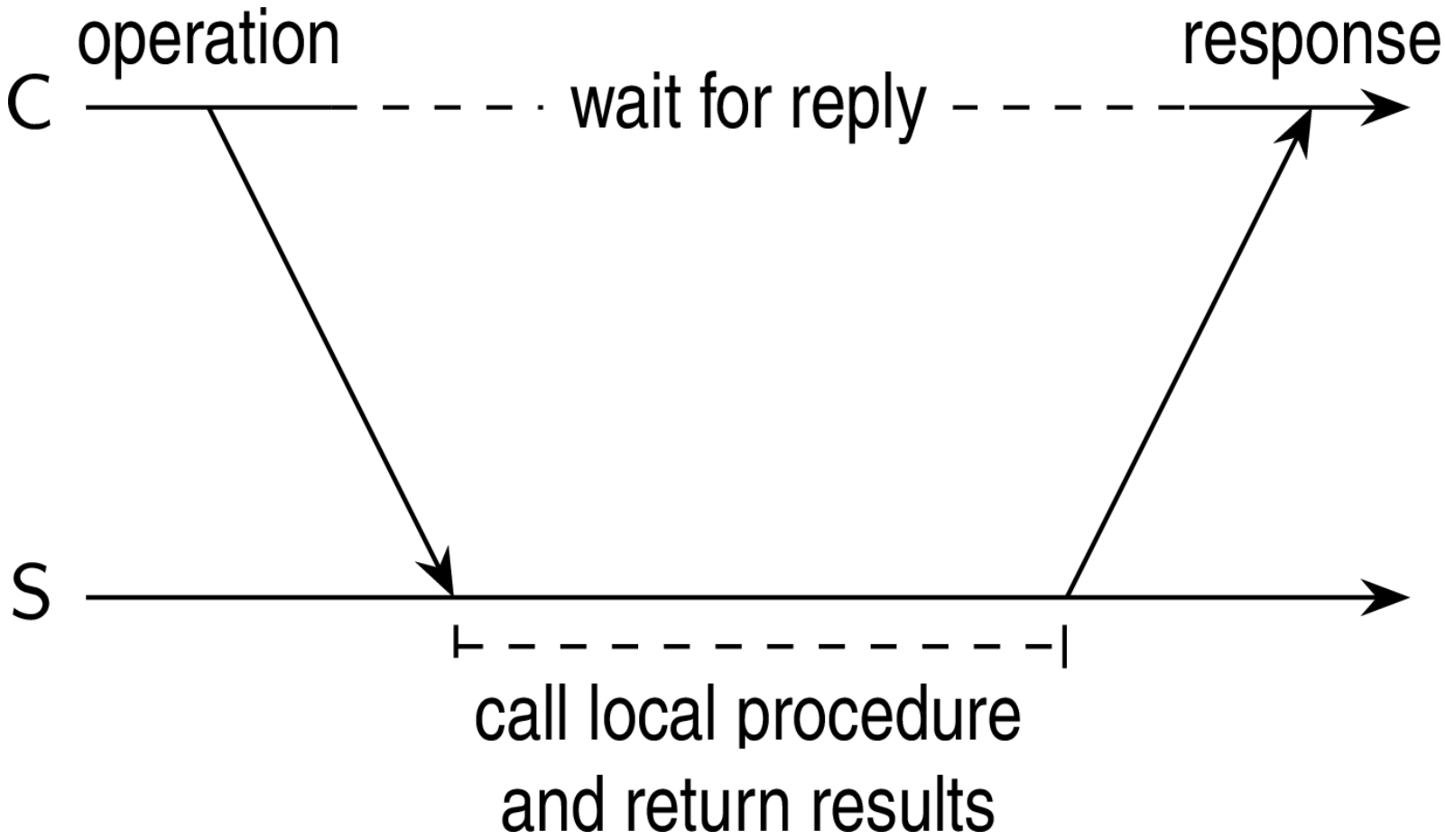
Types of Communication

- Various combinations of persistence and synchronization occur in practice.
- Popular ones are persistence in combination with synchronization at request submission, which is a common scheme for many message-queuing systems.
- Likewise, transient communication with synchronization after the request has been fully processed is also widely used. This scheme corresponds with remote procedure calls.

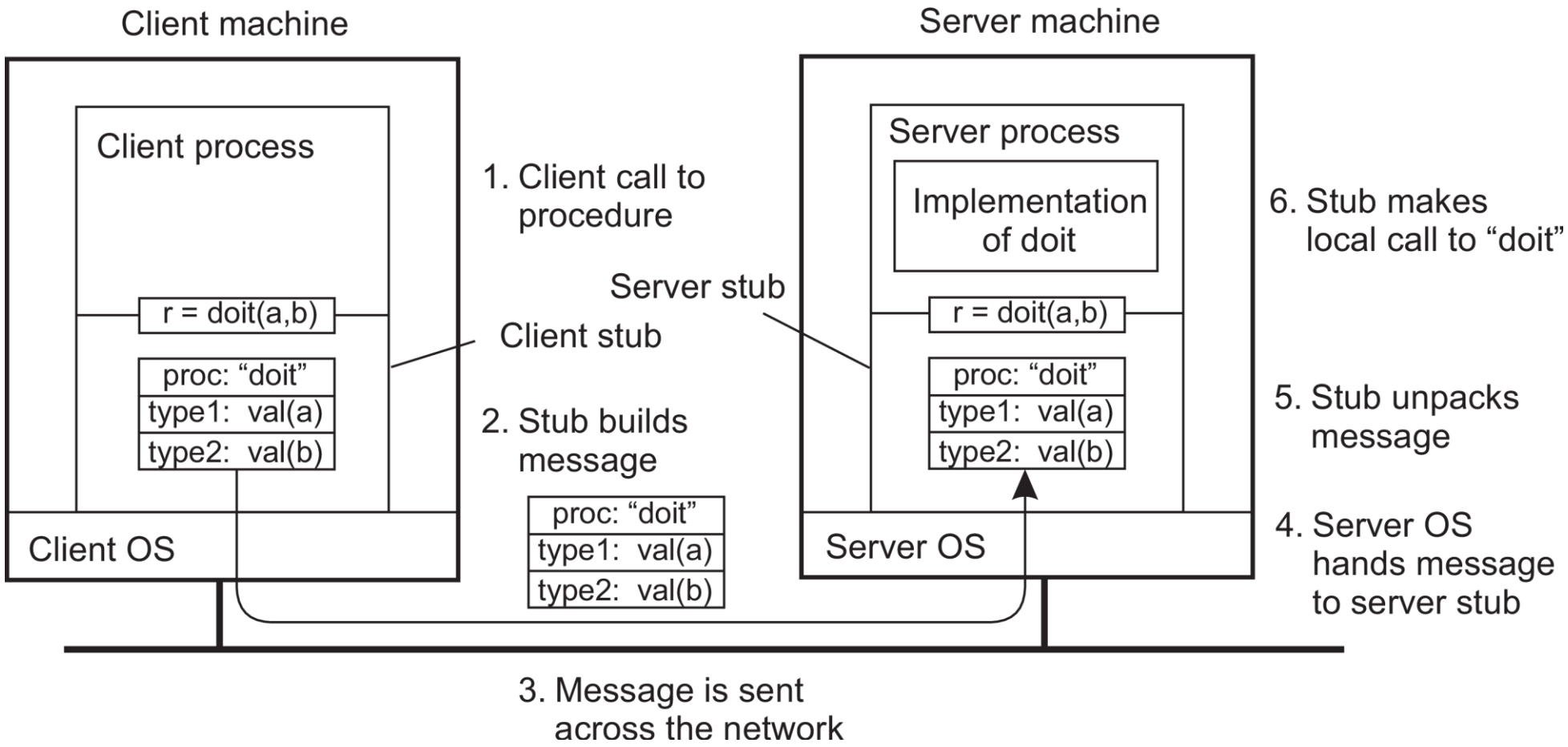
Remote Procedure Call

- When a process on a machine A calls a procedure on a machine B, the calling process on A is suspended, and execution of the called procedure takes place on B.
- Information can be transported from the caller to the callee in the parameters and can come back in the procedure result.
- No message passing at all is visible to the programmer. This method is known as **remote procedure call**, or often just **RPC**.

RPC between a client and server program.

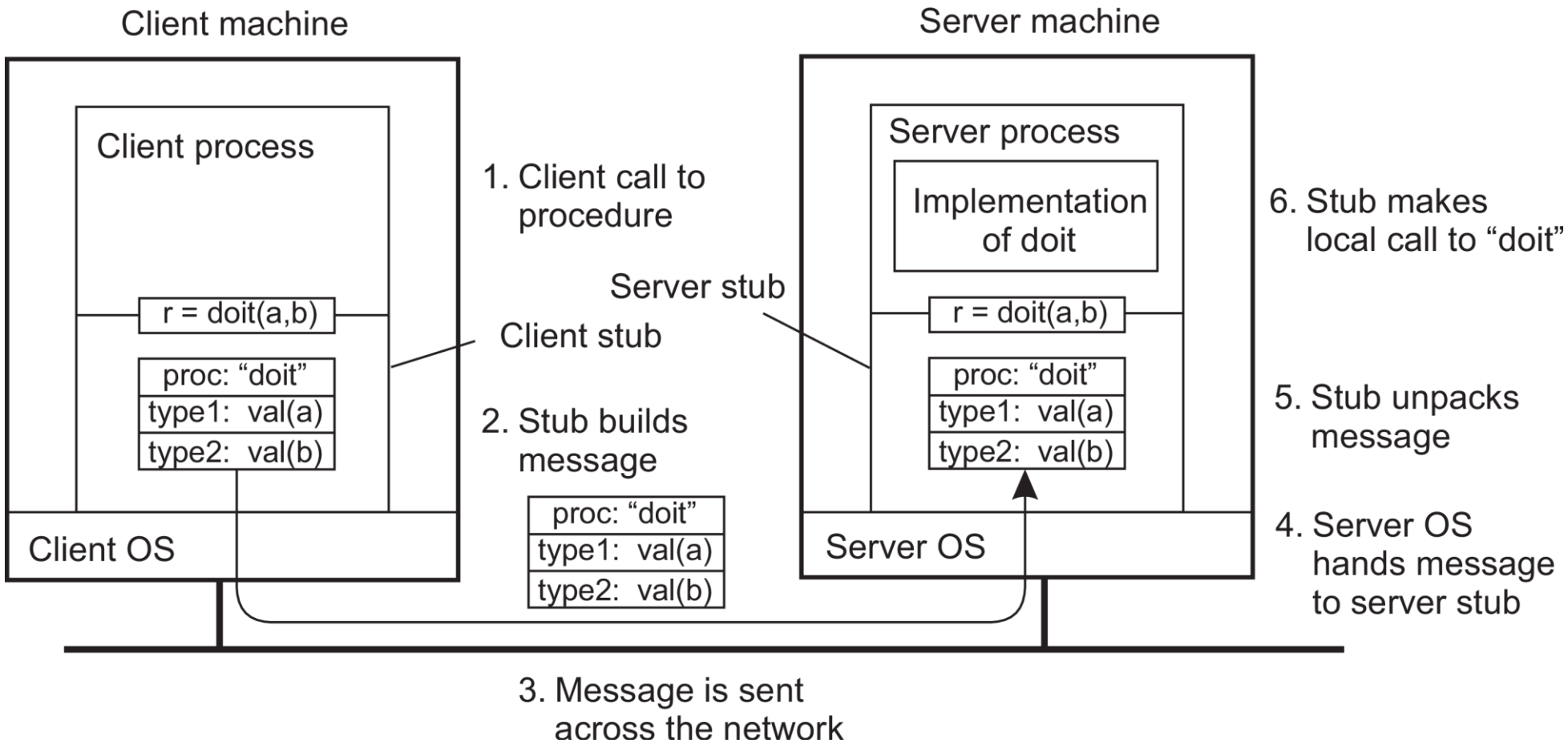


Basic RPC Operation



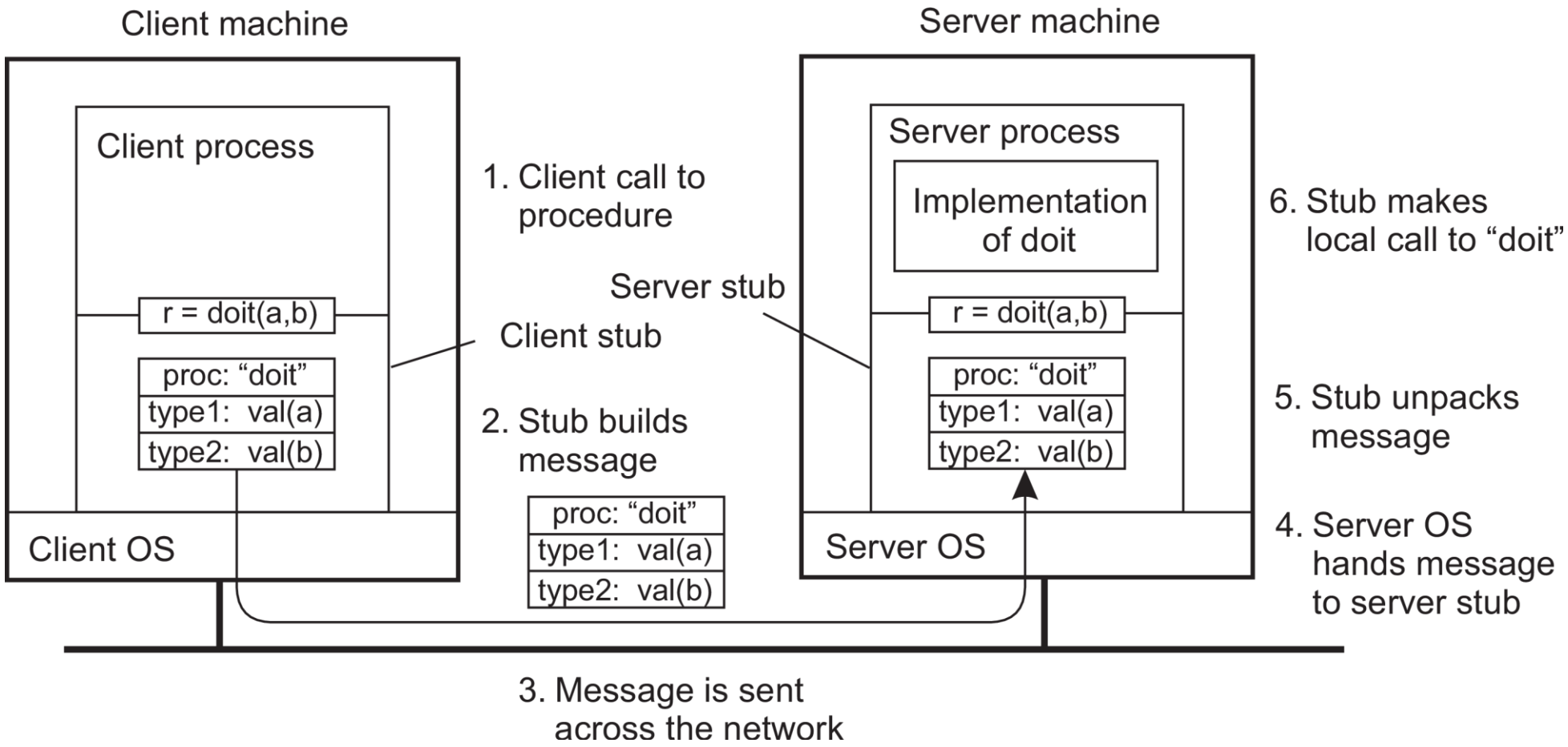
1. The client procedure calls the client stub in the normal way.

Basic RPC Operation



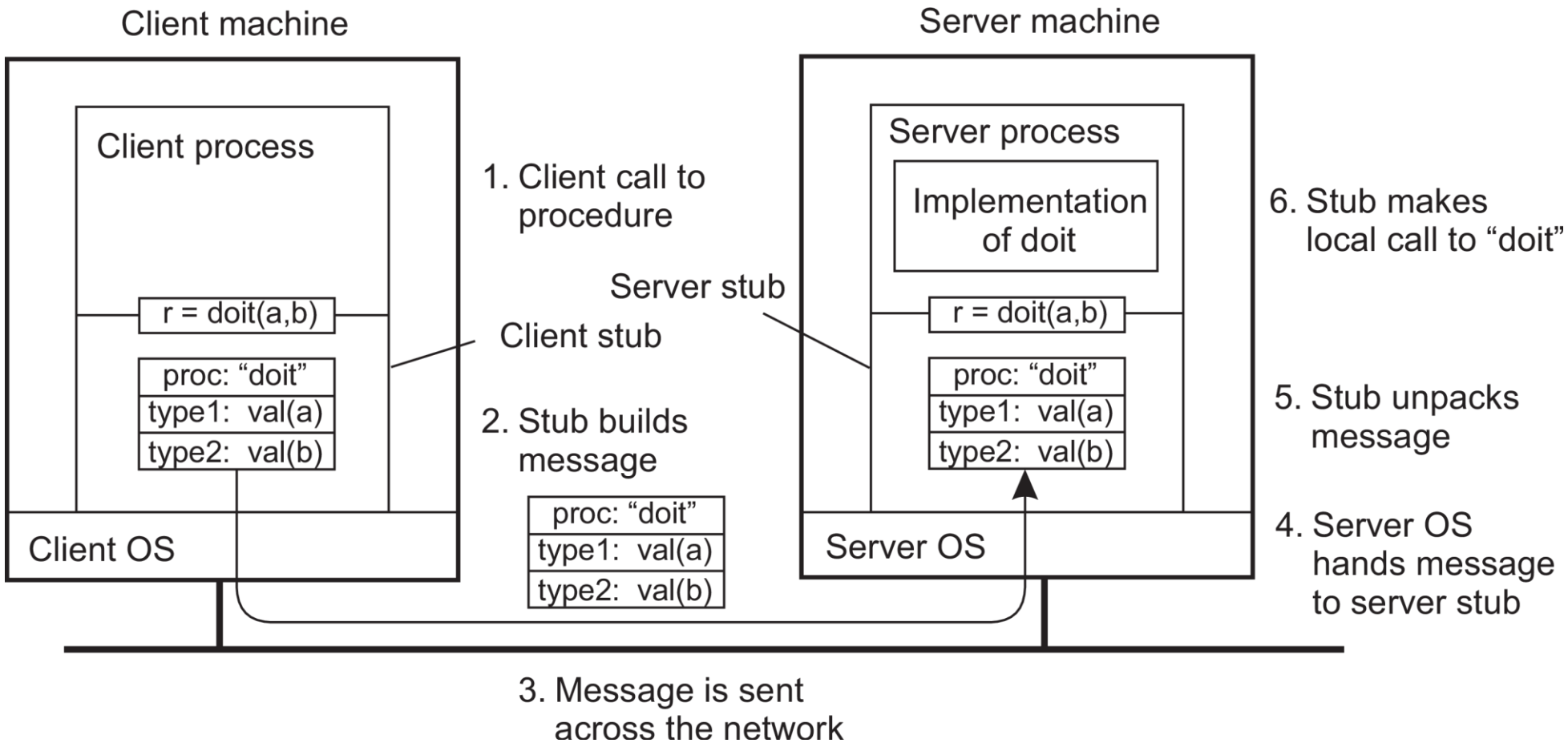
2. The client stub builds a message and calls the local operating system.

Basic RPC Operation



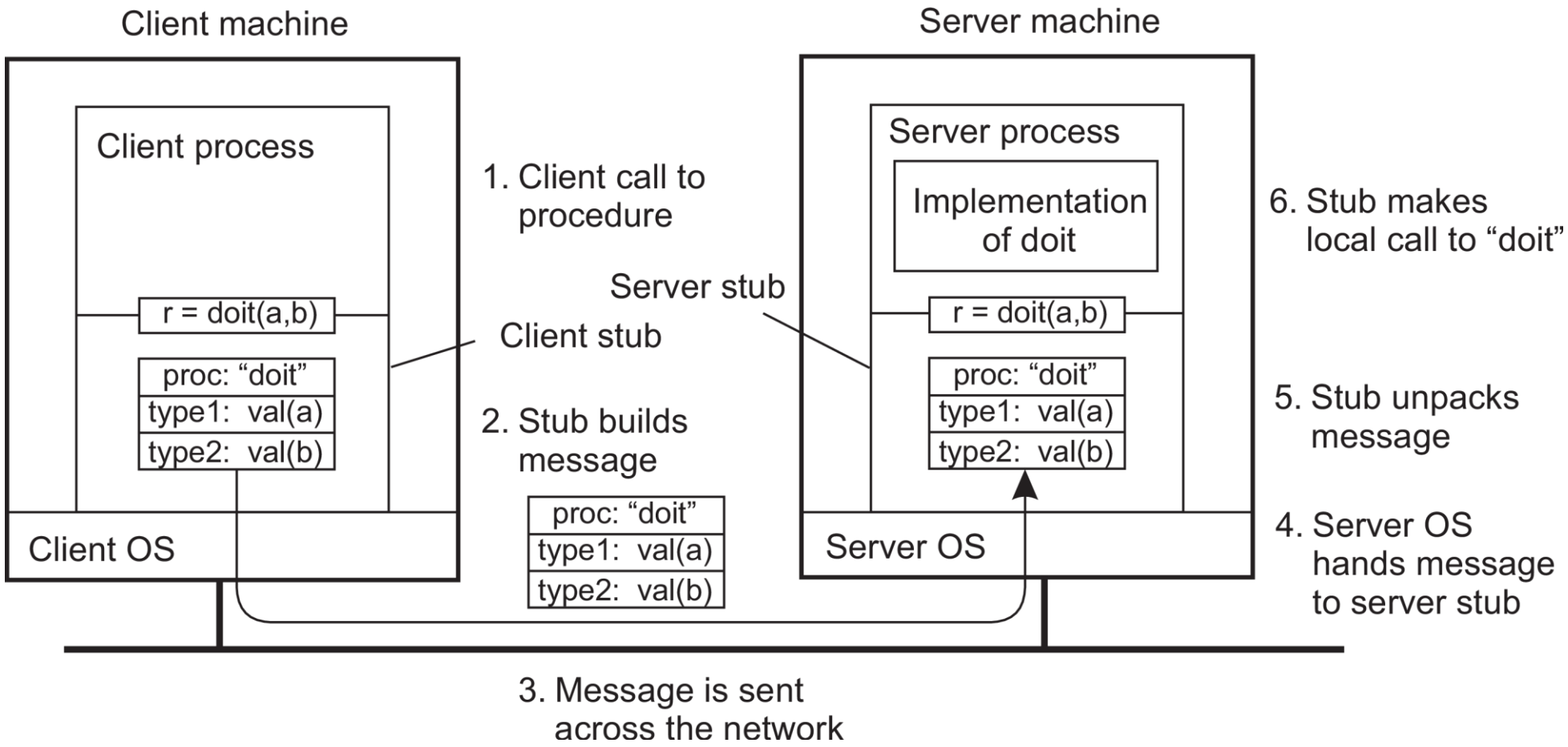
3. The client's OS sends the message to the remote OS.

Basic RPC Operation



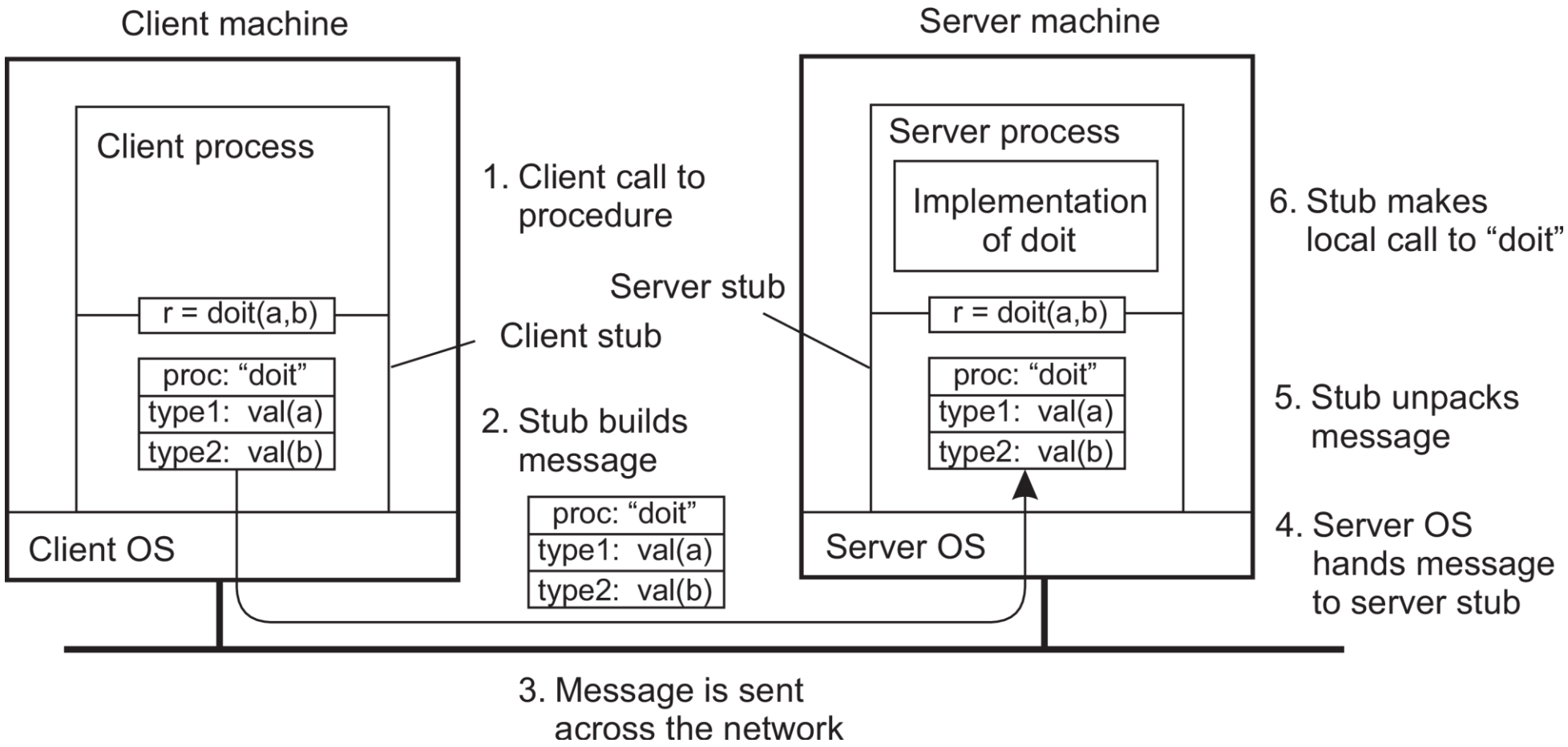
4. The remote OS gives the message to the server stub.

Basic RPC Operation



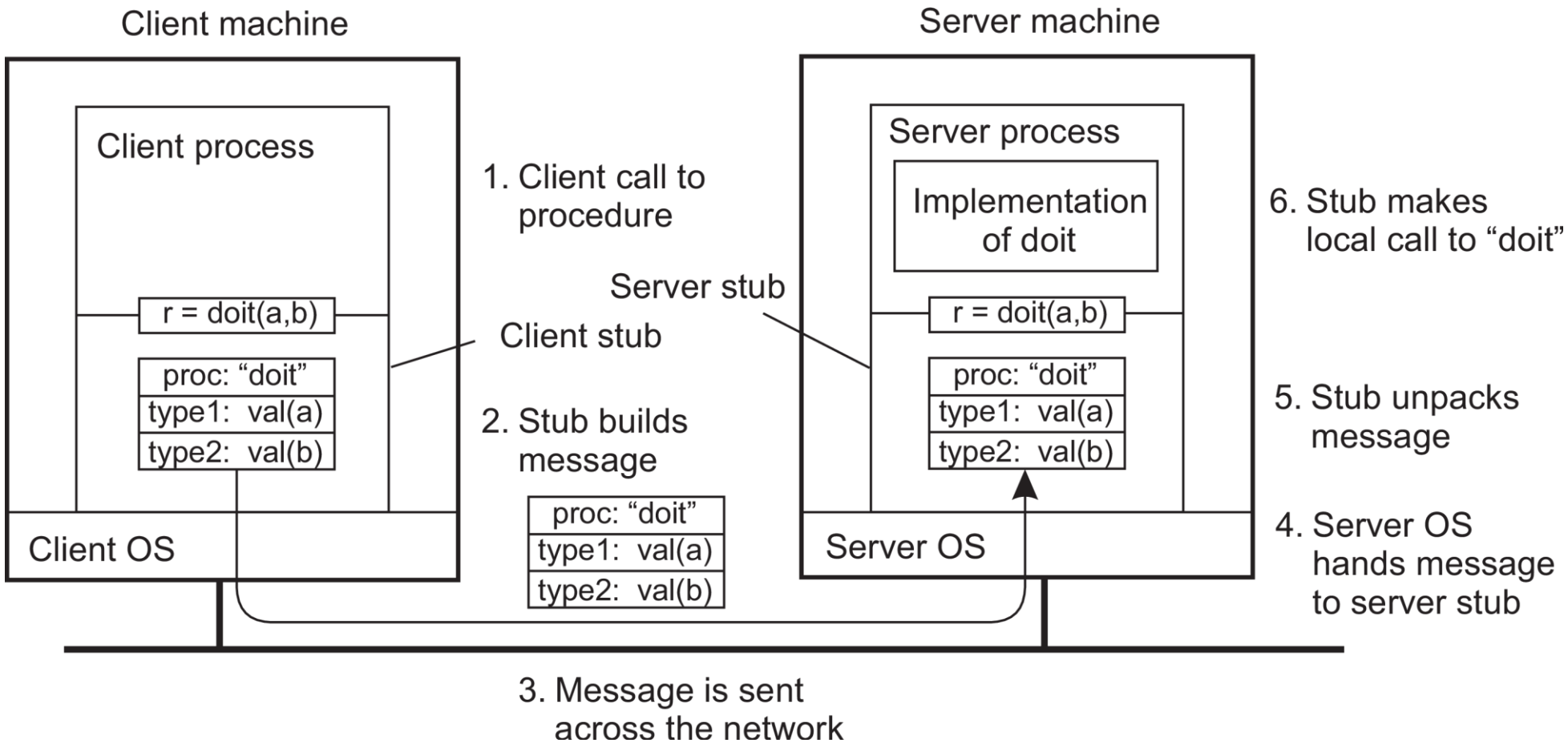
5. The server stub unpacks the parameter(s) and calls the server.

Basic RPC Operation



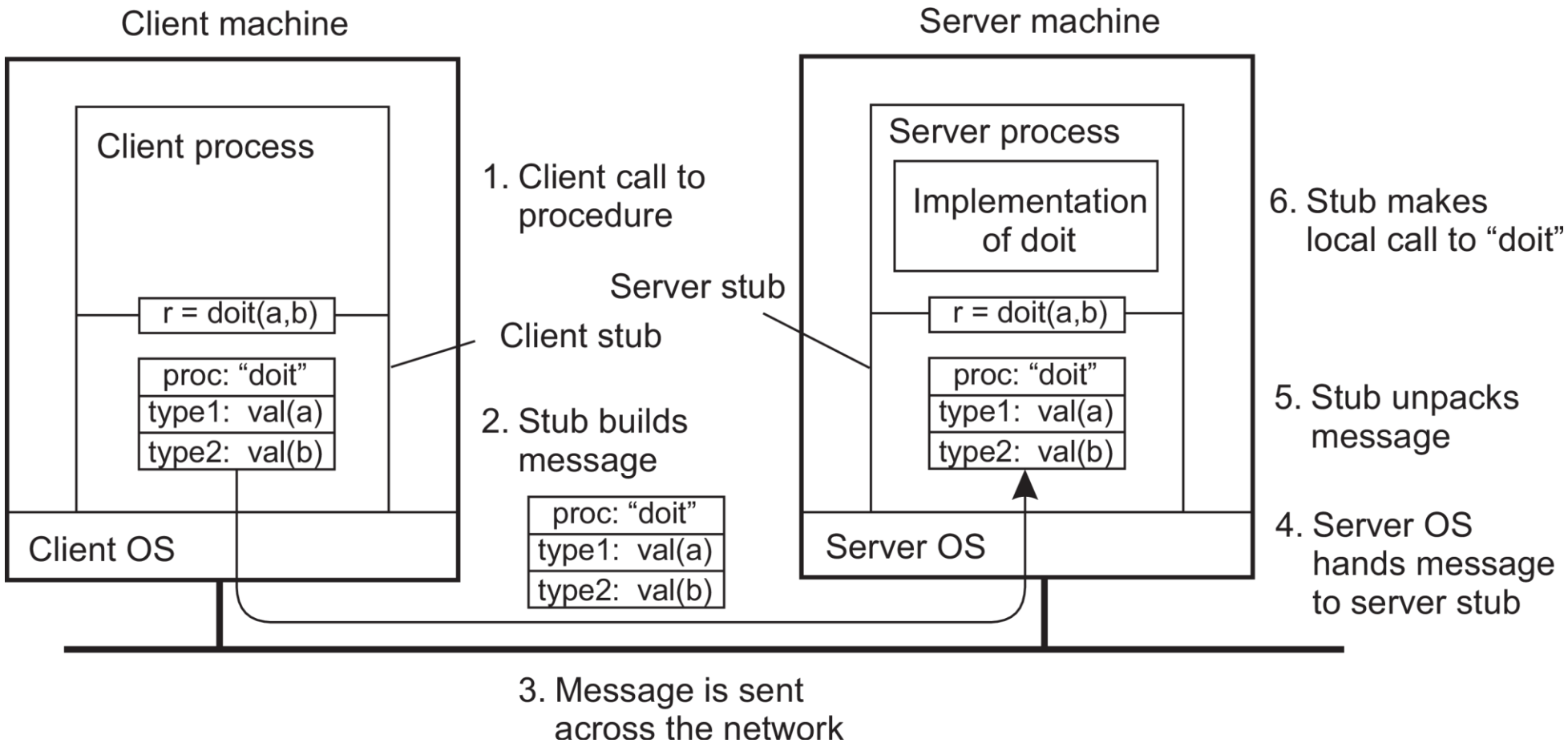
6. The server does the work and returns the result to the stub.

Basic RPC Operation



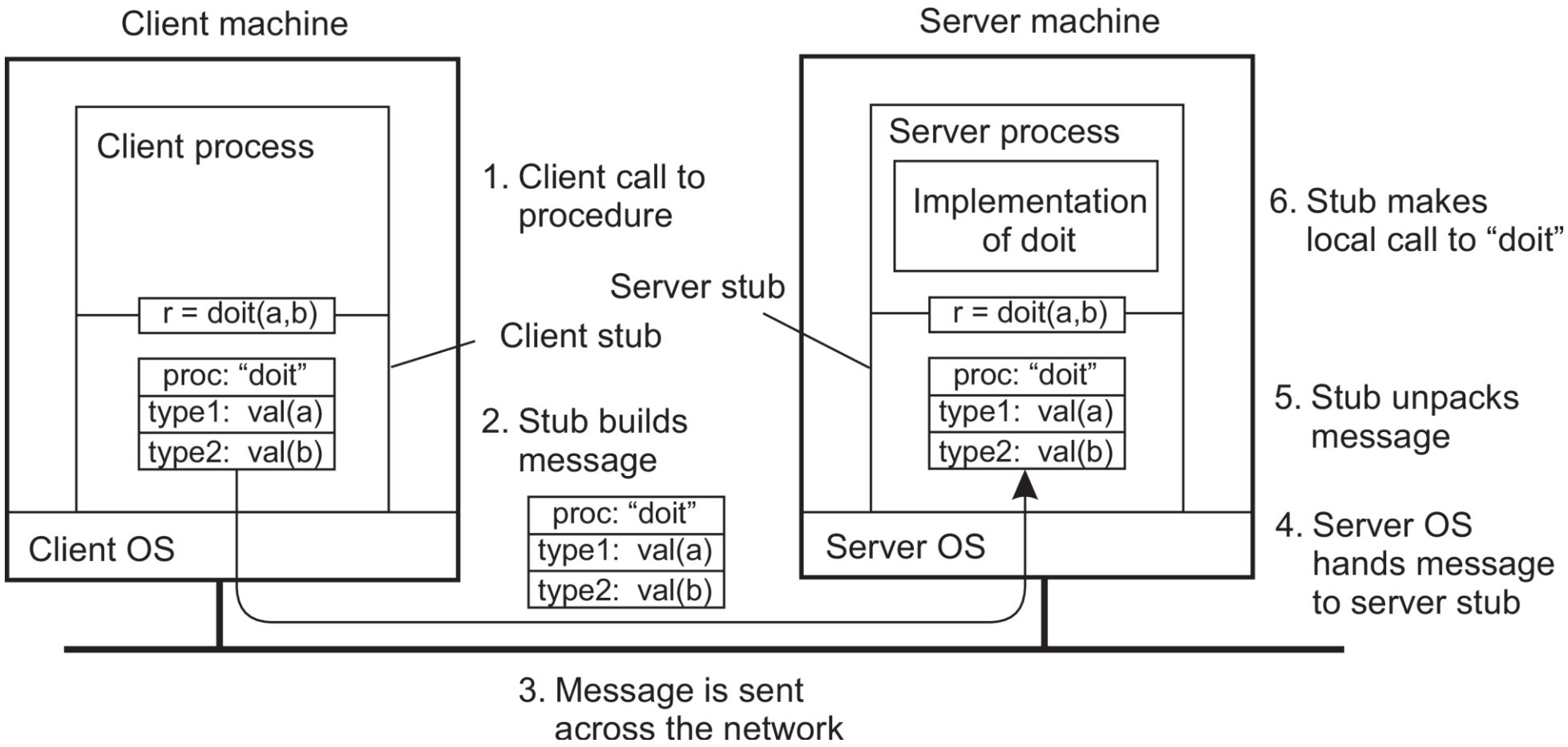
7. The server stub packs the result in a message and calls its local OS.

Basic RPC Operation



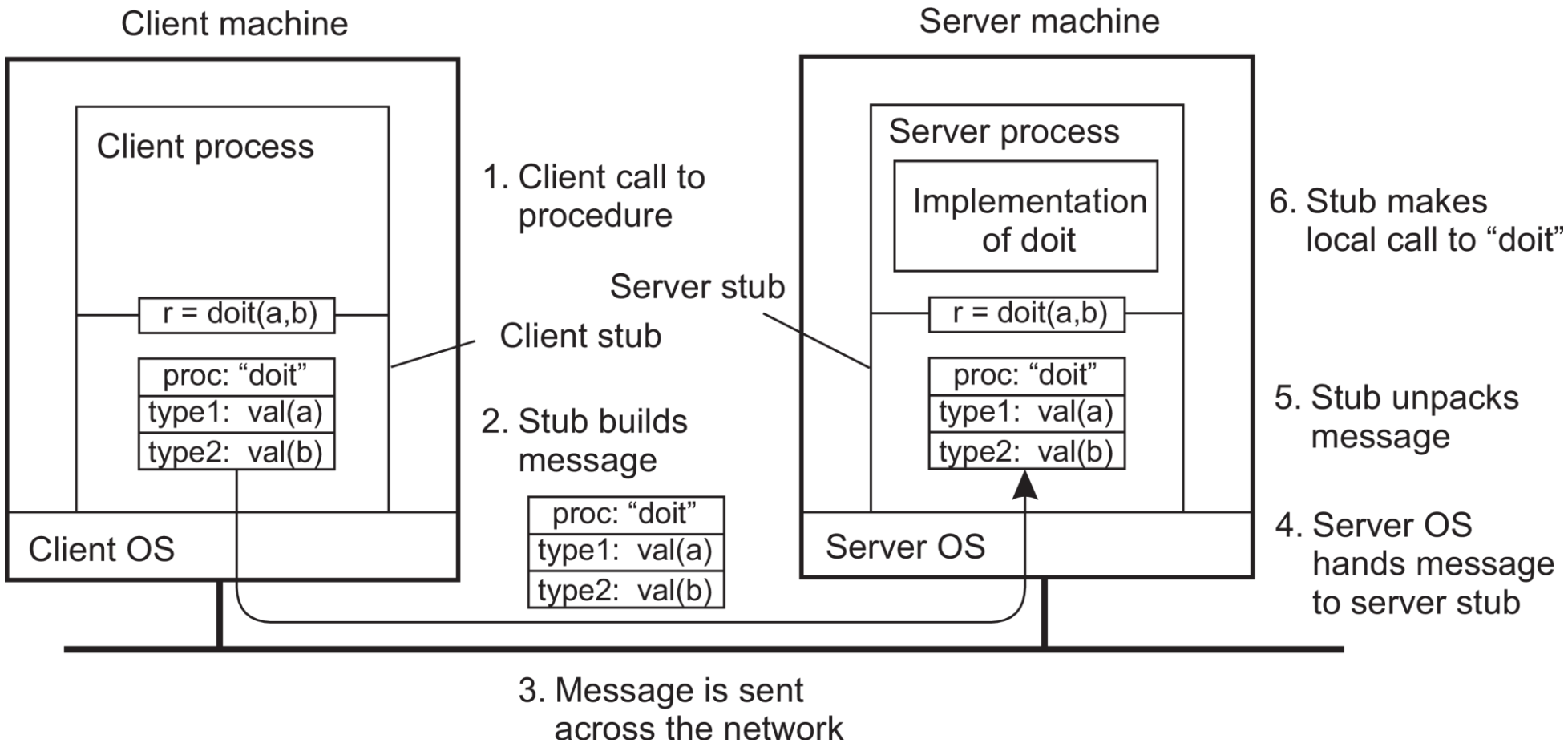
8. The server's OS sends the message to the client's OS.

Basic RPC Operation



9. The client's OS gives the message to the client stub.

Basic RPC Operation

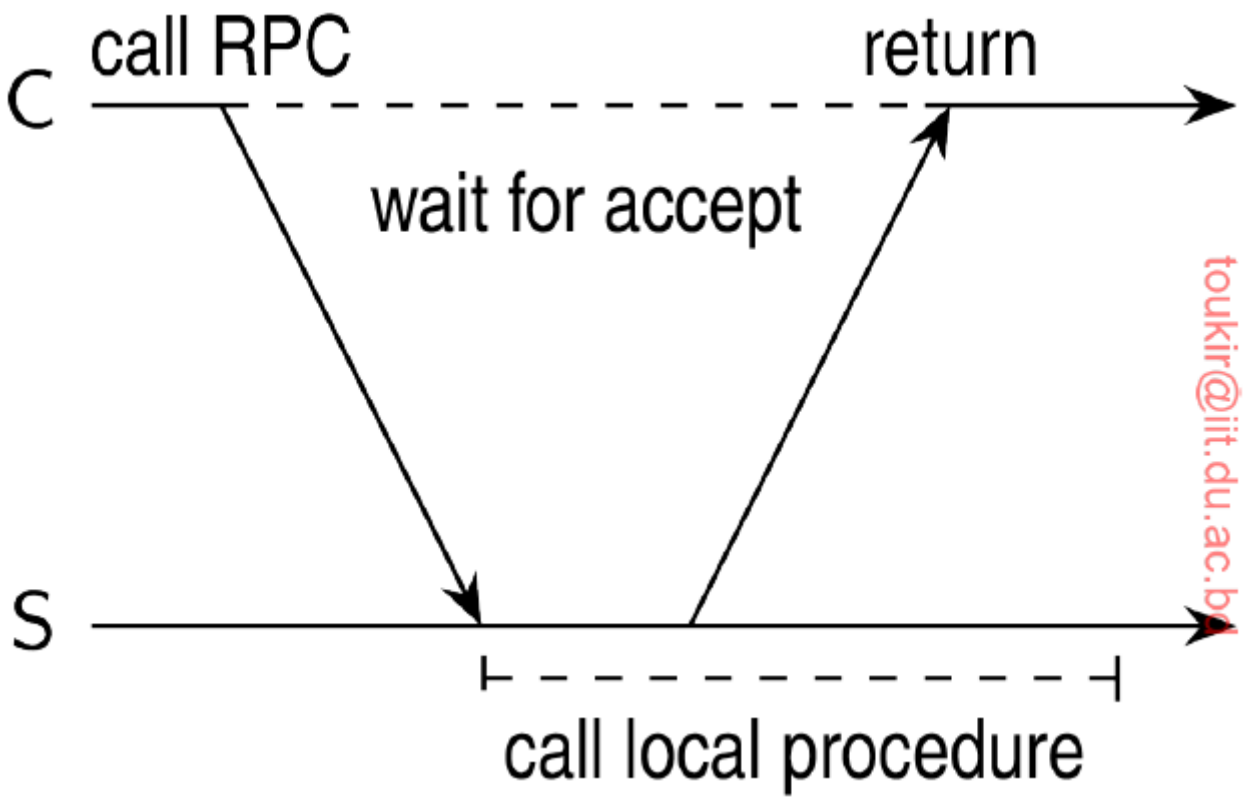


10. The stub unpacks the result and returns it to the client.

Variations on RPC

- As in conventional procedure calls, when a client calls a remote procedure, the client will block until a reply is returned.
- This strict request-reply behavior is unnecessary when there is no result to return, or may hinder efficiency when multiple RPCs need to be performed.

Asynchronous RPC



Deferred Synchronous RPC