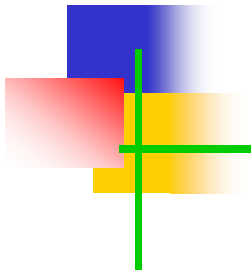


Uzaktan Çalıştırma: RPC ve Remote Invocation

http://java.sun.com/docs/books/tutorial/rmi/

Konu Başlıkları

- Remote Procedure Call (RPC) giriş
- Normal ve uzaktan procedure çağırma (RPC)
- RPC Mimarisi
- RPC Gerçekleştirimi
- RMI Programlama ve bir Örnek:
  - Sunucu taraflı (Server-Side) RMI programlama
  - İstemci taraflı (Client-Side) RMI programlama
- İleri Düzey RMI Kavramları
  - Güvenlik İlkeleri (Security Policies), İstisnalar (Exceptions), Dinamik Yükleme (Dynamic Loading)
- Daha İleri RMI uygulamaları
  - Dosya Sunucusu (File Server)

Distributed Computing-RPC
Dağıtık Bilgi İşleme - RPC

- Önceki dersten, client server ile soket bağlantısı üzerinden konuştuk. İki tarafın da bildiği bir protokol kullanıldı.
- Ancak, hem client hem de server, soket düzeyindeki detayları bilmek zorundaydı
- Amaç: bu detayların soyutlanması ve sunucuya olan isteğin istemci tarafından yerel procedure çağrısıymış gibi görünmesi
- Remote Procedure Call (RPC)'nin altında yatan fikir budur. Bu teknoloji, 1970'lerin sonlarında tanıtılmıştır
- İki RPC beyannamesi:
  - SUN's Open Network Computing (ONC) RPC
  - OSF's Distributed Computing Environment(DCE) RPC

Procedure Calls

- The procedure call (same as function call or subroutine call) is a well-known method for **transferring control** from one part of a process to another, with a return of control to the caller.

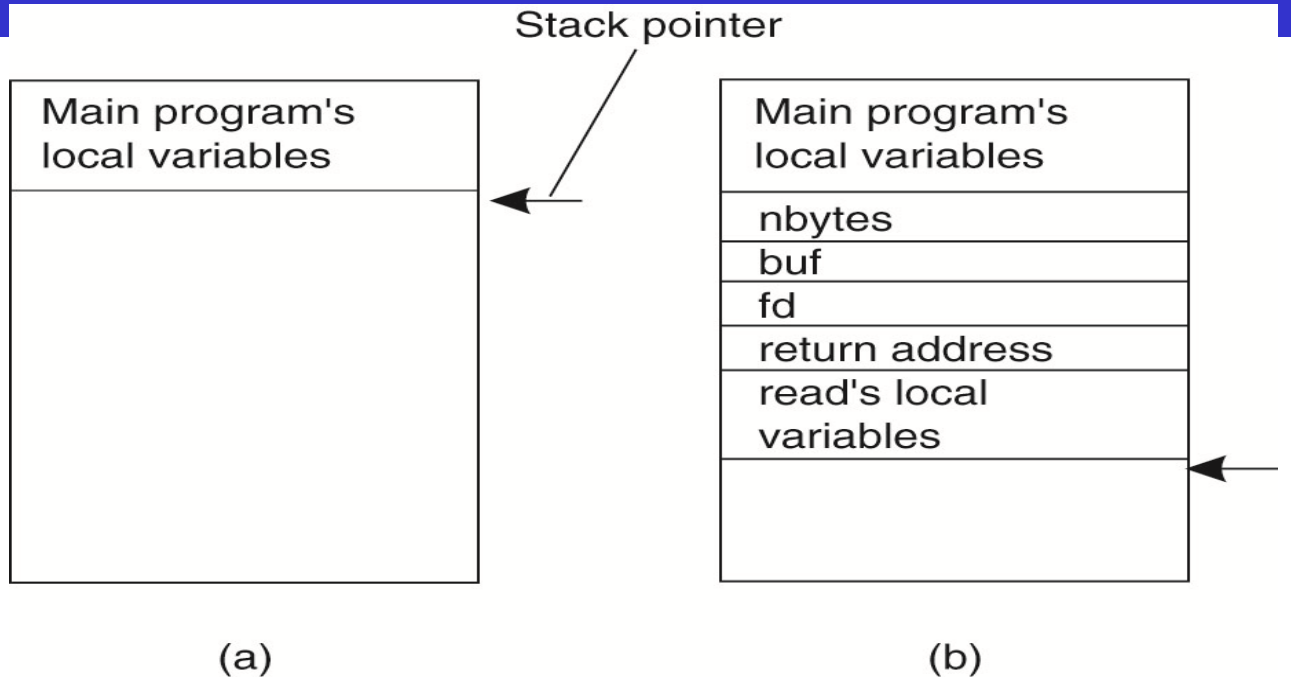
- Associated with the procedure call is the **passing of arguments** from the caller (the client) to the callee (the server).

- In most current systems the caller and the callee are **within a single process** on a given host system. This is what we call "**local procedure calls**".

Conventional Procedure Call

- local procedure call (on single machine) (in C)
  - *count = myread(fd, buf, nbytes)*
  - fd: an int indicating a file
  - buf: an array of chars into which data are read
  - nbytes: an int telling how many bytes to read
- Split the call into
  - A client part (machine A)
  - A server part (machine B)

Conventional Procedure Call



- (a) Parameter passing in a local procedure call: the stack before the main program calls to myread.
- (b) The stack while the called procedure is active.
- (c) after the read completes, ret val is put in a register, remove ret address and pass control to caller,
- d) caller removes parameters from stack (back to a)

Conventional Procedure Call

- Parameter passing in a procedure call:
  - in C: 2 ways
  - Call-by-value (fd, nbytes)
  - Call-by-reference
  - a ptr to variable, address of a var
  - arrays are always passed by reference
  - On the stack: address of char array, buf
  - If the called proc stores sthg into buf, it does modify the array in the caller
  - Difference b/w the two is very important for RPC

Remote Procedure Calls

- In a remote procedure call (RPC), a process on the local system invokes a procedure on a remote system. The reason we call this a “procedure call” is because the intent is to make it appear to the programmer that a normal procedure call is taking place.

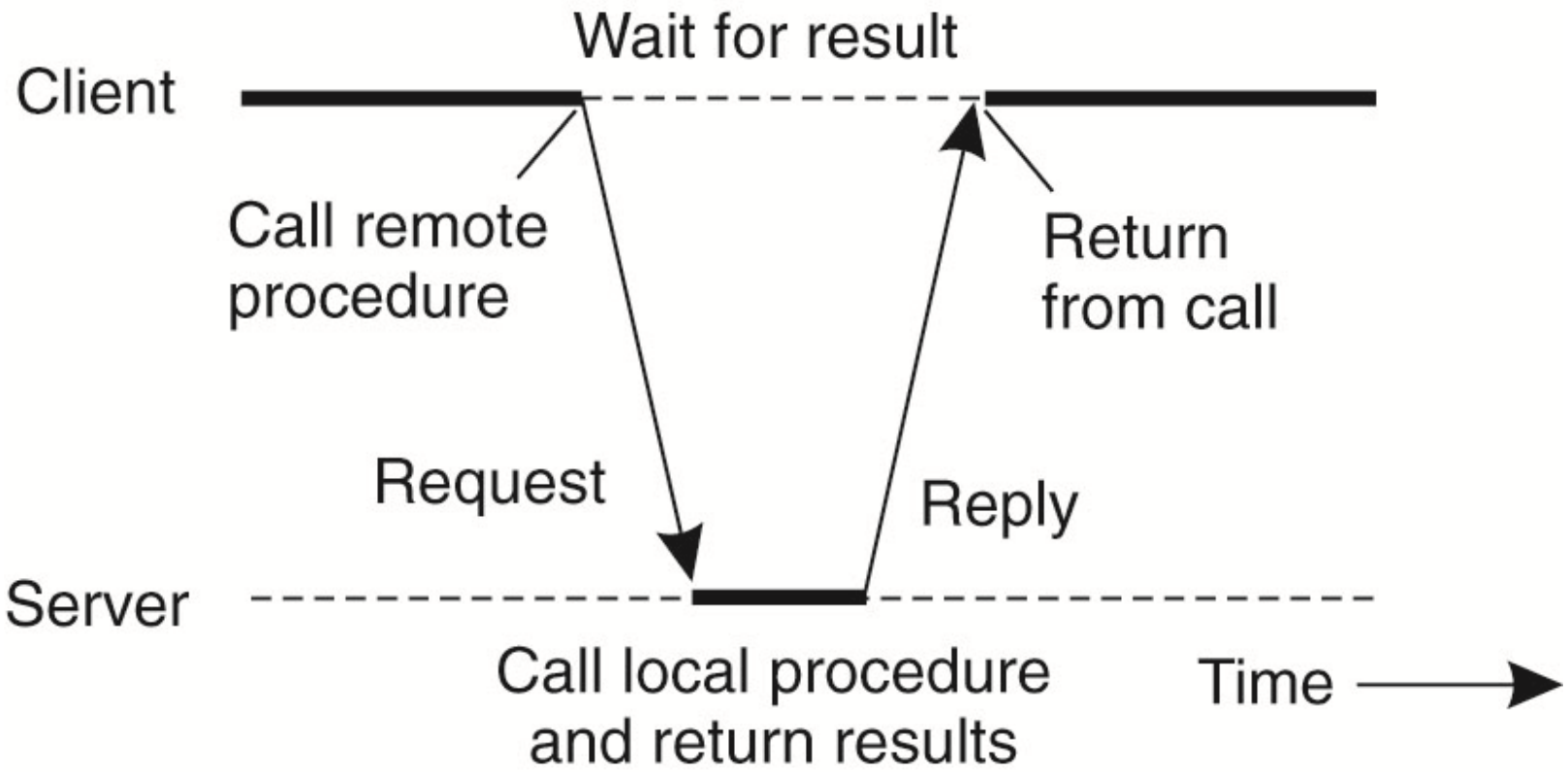
- We use the term “request” to refer to the client calling the remote procedure, and the term “response” to describe the remote procedure returning its result to the client.

- We use the SUN RPC as our remote procedure call example.

RPC Principle

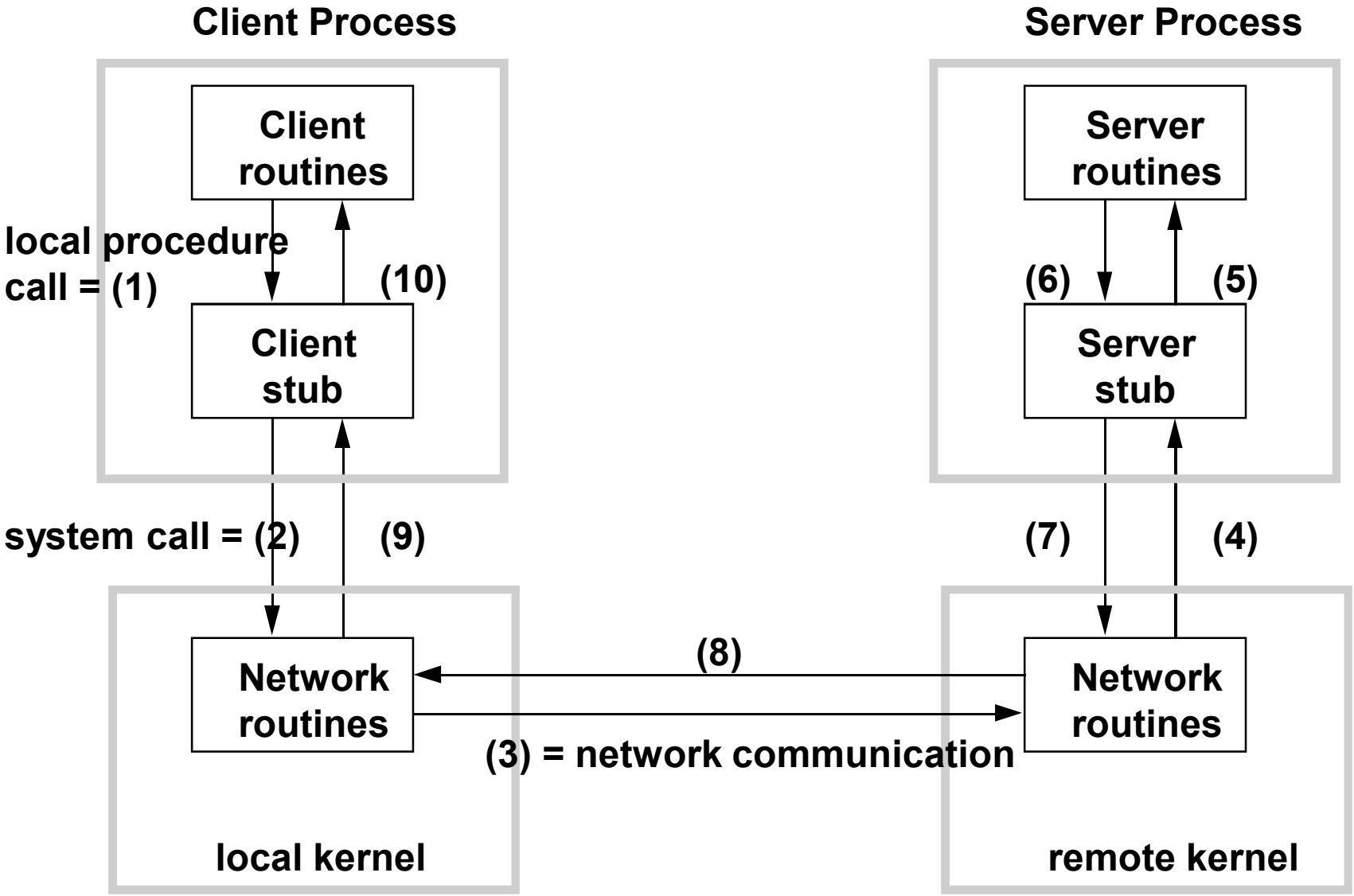
- Idea:
  - Make RPC look as much as possible like a local one
  - Make RPC transparent
- Client stub: A different version of read is used
  - Packs parameters into a msg
  - Request OS that msg to be sent to server
  - Following the **send**, it calls **receive** (blocks itself until a reply comes back)
- Server Stub
  - Calls **receive** and blocks
  - When msg arrives at server, OS there passes it to server stub
  - Transforms the msg into a local procedure call
  - Unpack parameters and calls the server procedure
  - It packs the result in a msg and calls **send** to return it to client

Client and Server Stubs



- Principle of RPC between a client and server program.

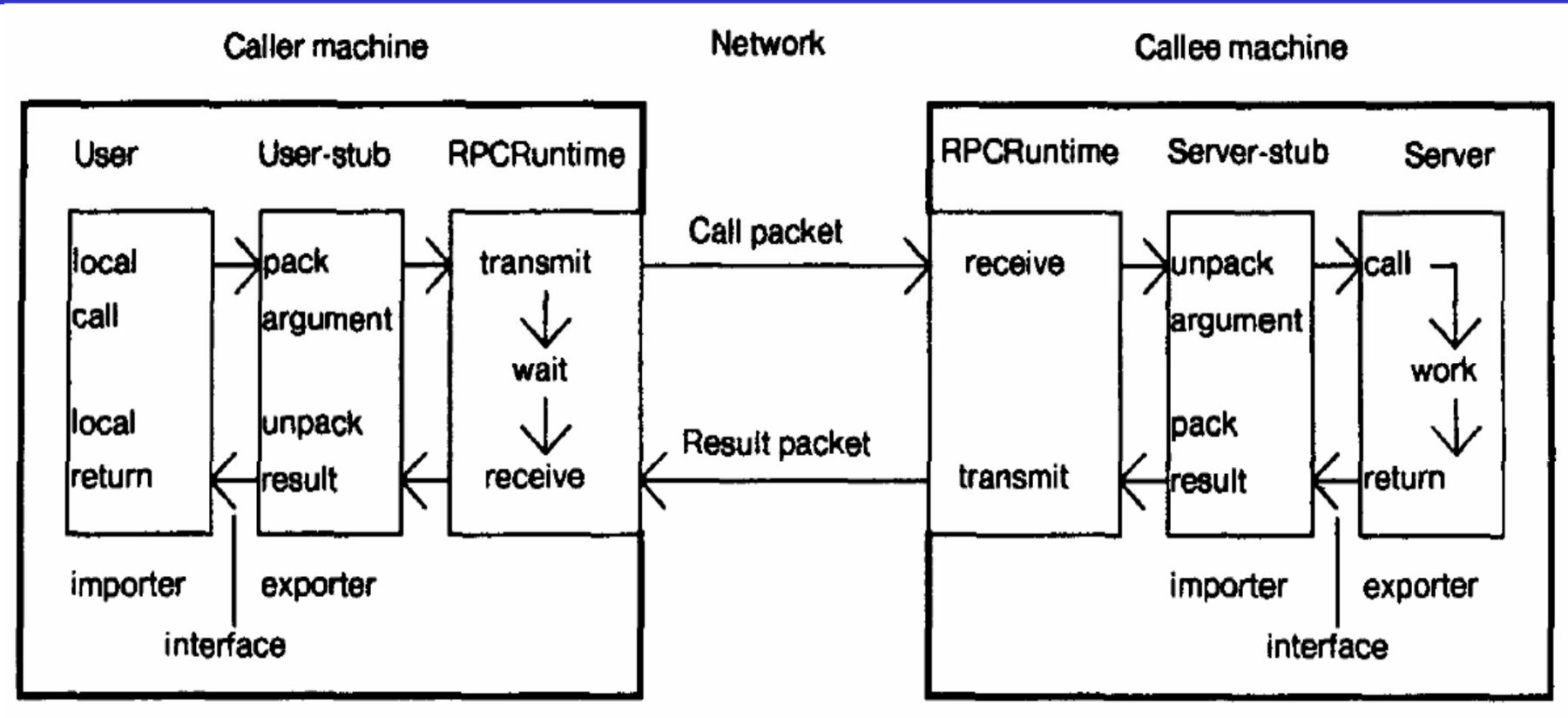
The 10 steps in a Remote Procedure Call (RPC)



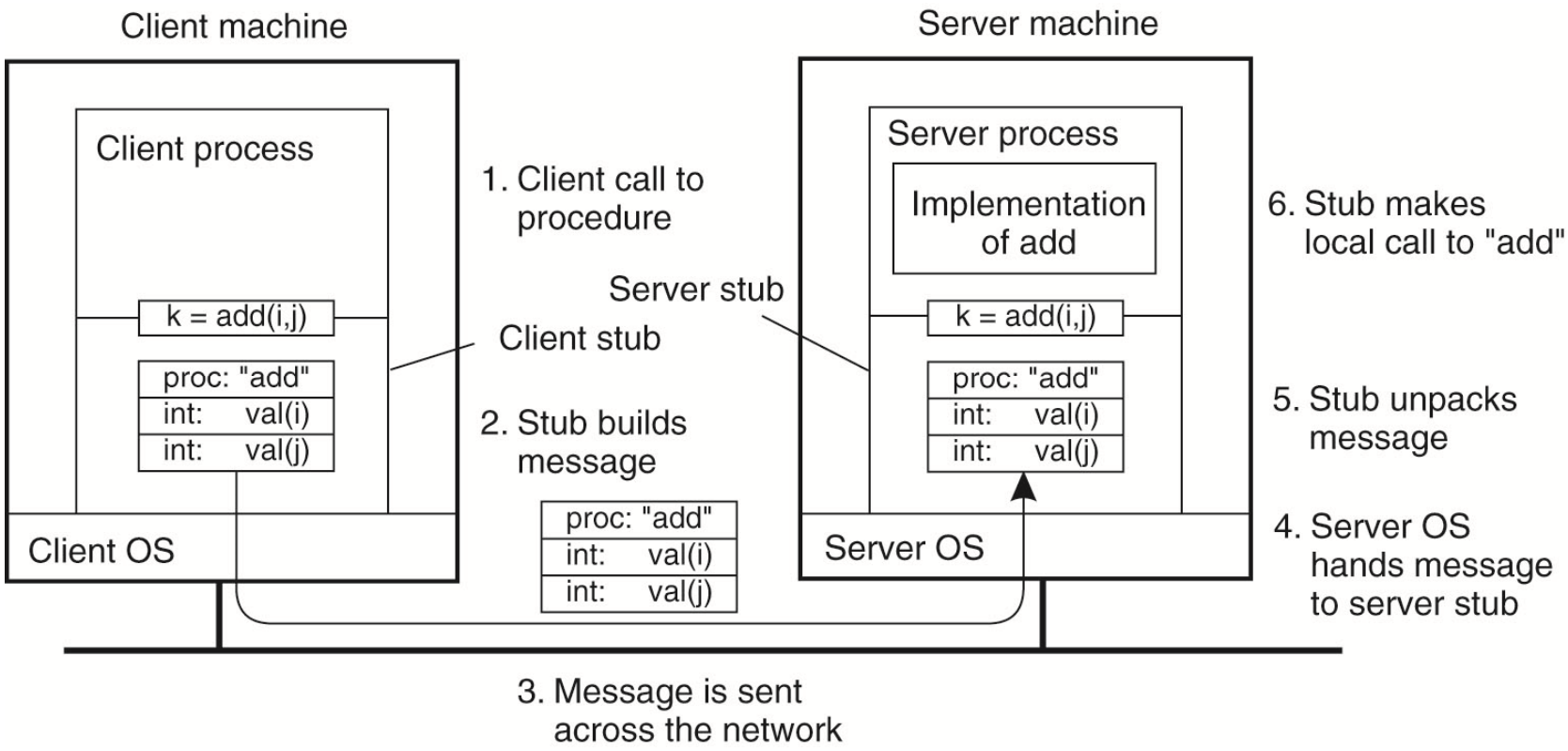
The 10 steps in a Remote Procedure Call (continue)

1. The client calls a local procedure, called the client stub. It appears to the client that the client stub is the actual server procedure that it wants to call. The purpose of the stub is to package the arguments for the remote procedure, possibly put them into some standard format and then build one or more network messages. The packaging of the client's arguments into a network message is termed **marshaling**.
2. These network messages are sent to the remote system by the client stub. This requires a system call to the local kernel.
3. The network messages are transferred to the remote system. Either a connection-oriented or a connection-less protocol is used. The client's OS sends the message to the remote OS.The remote OS gives the message to the server stub.
4. A server stub procedure is waiting on the remote system for the client's request. It **unmarshals** the arguments from the network message and possibly converts them.
5. The server stub executes a local procedure call to invoke the actual server function, passing it the arguments that it received in the network messages from the client stub.
6. When the server procedure is finished, it returns to the server stub with return values.
7. The server stub converts the return values, if necessary, and **marshals** them into one or more network messages to send back to the client stub.
8. The messages get transferred back across the network to the client stub.
9. The client stub reads the network messages from the local kernel and unmarshal it.
10. After possibly converting the return values, the client stub finally returns to the client function. This appears to be a normal procedure return to the client.

RPC Mimarisi (Architecture)

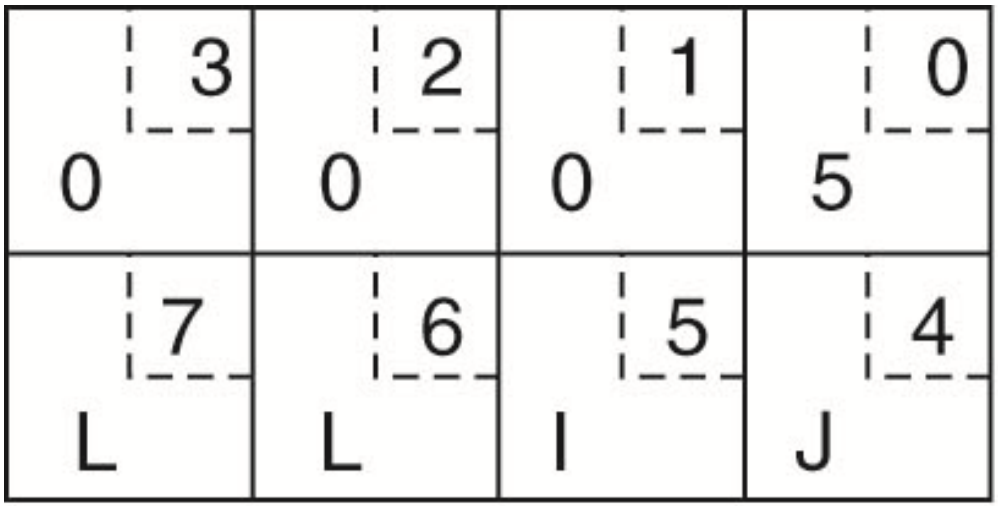


Passing Value Parameters (1)



- The steps involved in a doing a remote computation through RPC.

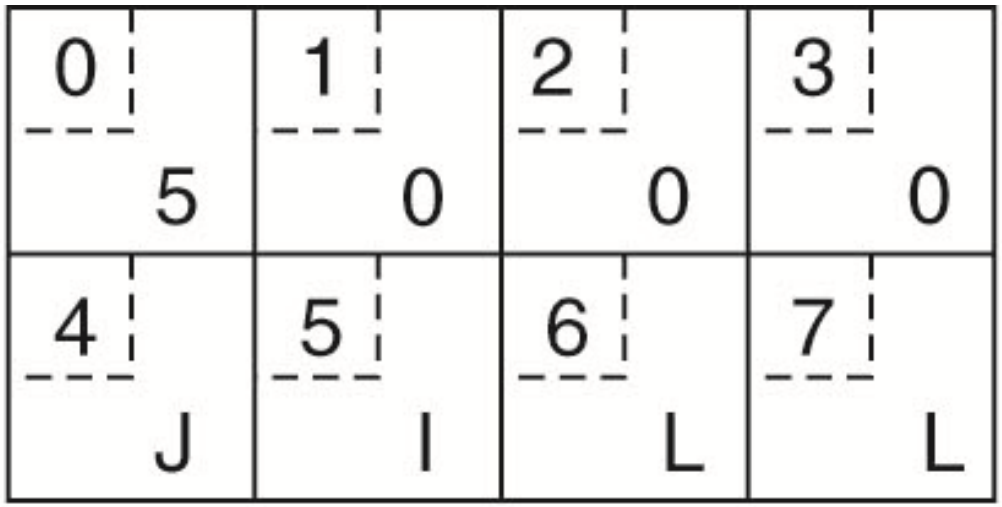
Passing Value Parameters (2)



(a)

- (a) The original message (client stub, 1st word int 5, 2nd string «JILL») on the Pentium, a little endian machine

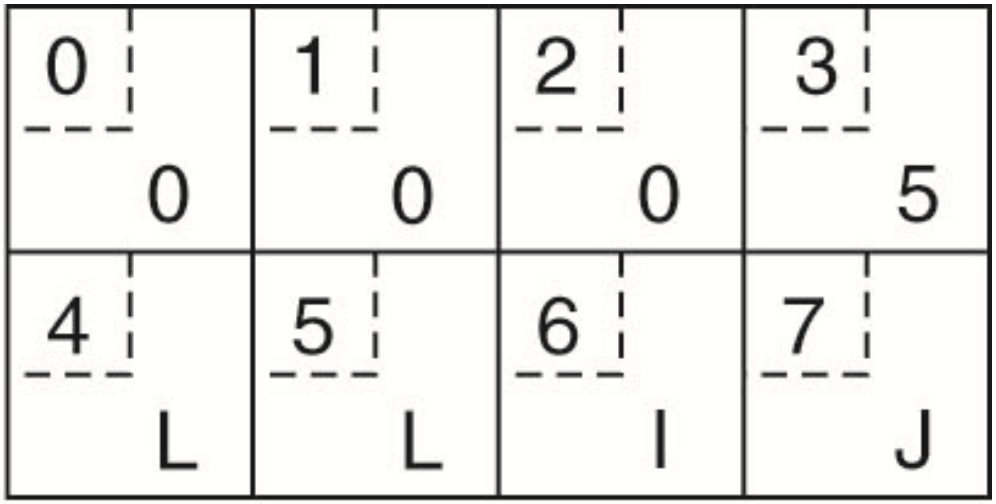
Passing Value Parameters (3)



(b)

- (a) The message (server stub, 1st word int 5x2^24, 2nd string «JILL») after receipt on SPARC, a big endian machine.

Passing Value Parameters (4)



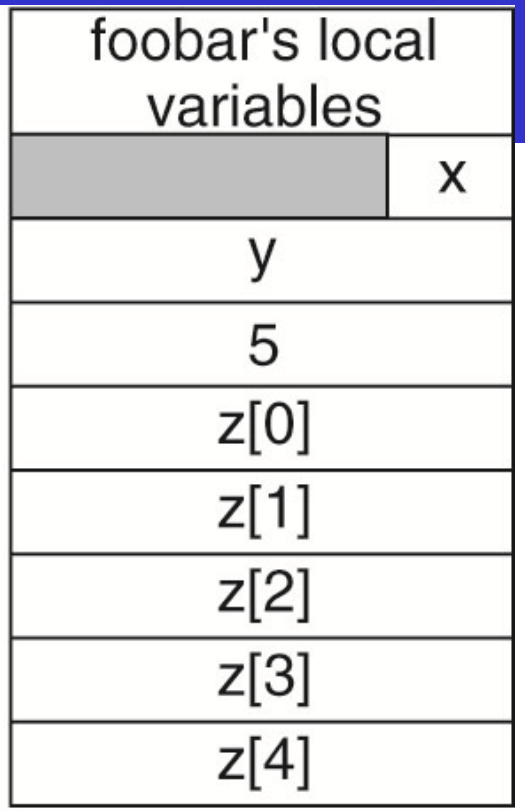
(c)

- (c) The message after being inverted (server stub, 1st word int 5, 2nd string «LLIJ»). The little numbers in boxes indicate the address of each byte.

Parameter Specification and Stub Generation

```
foobar( char x; float y; int z[5] )
{
   ....
}
```

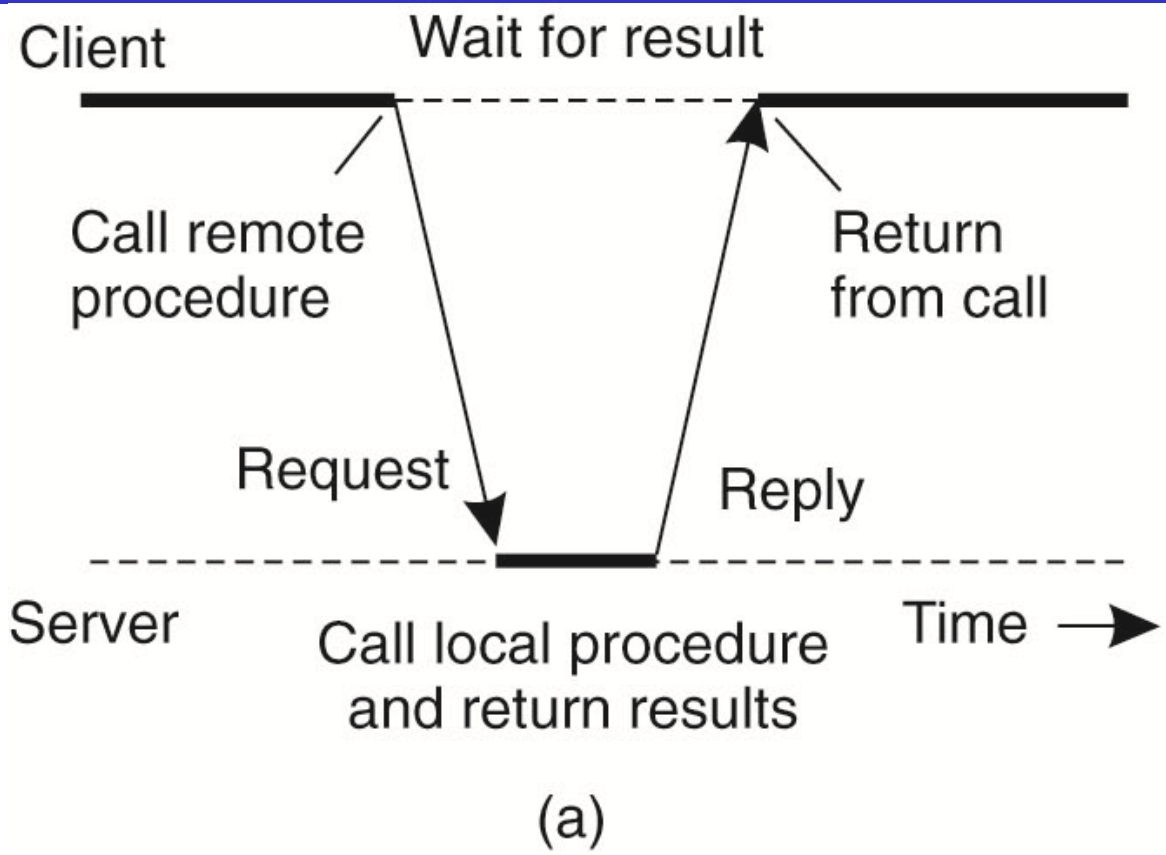
(a)



(b)

- (a) A procedure. (b) The corresponding message. Both client and server stubs agree on this format

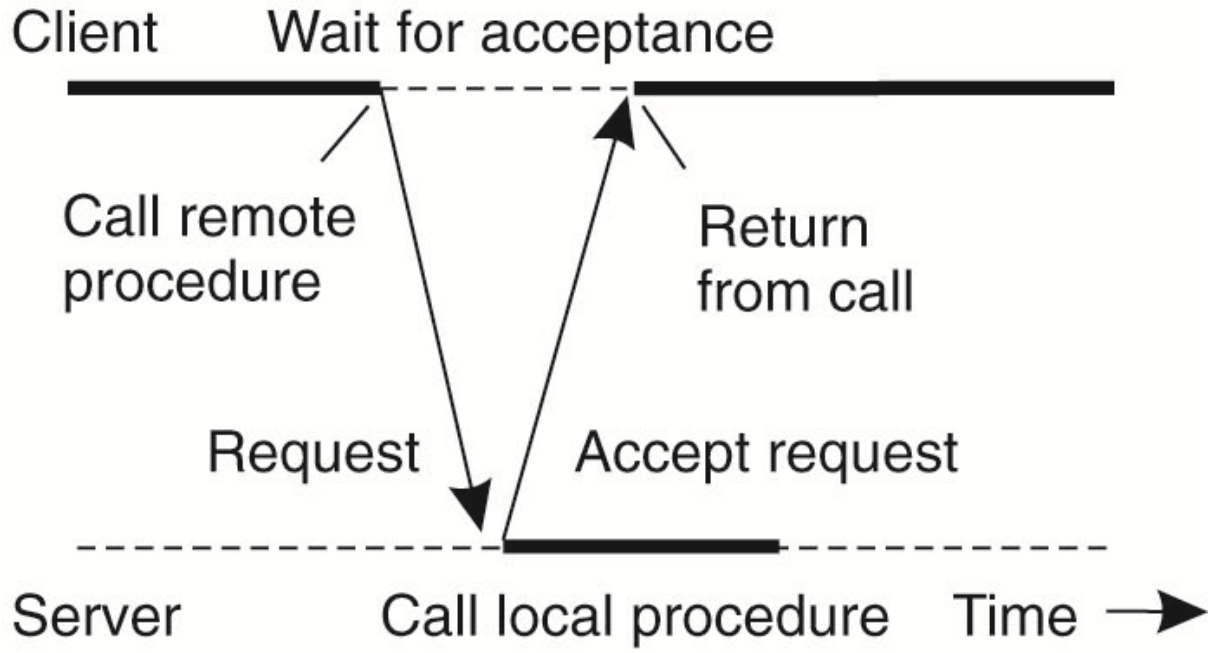
Asynchronous RPC (1)



(a)

- (a) The interaction between client and server in a traditional RPC. Wait for reply

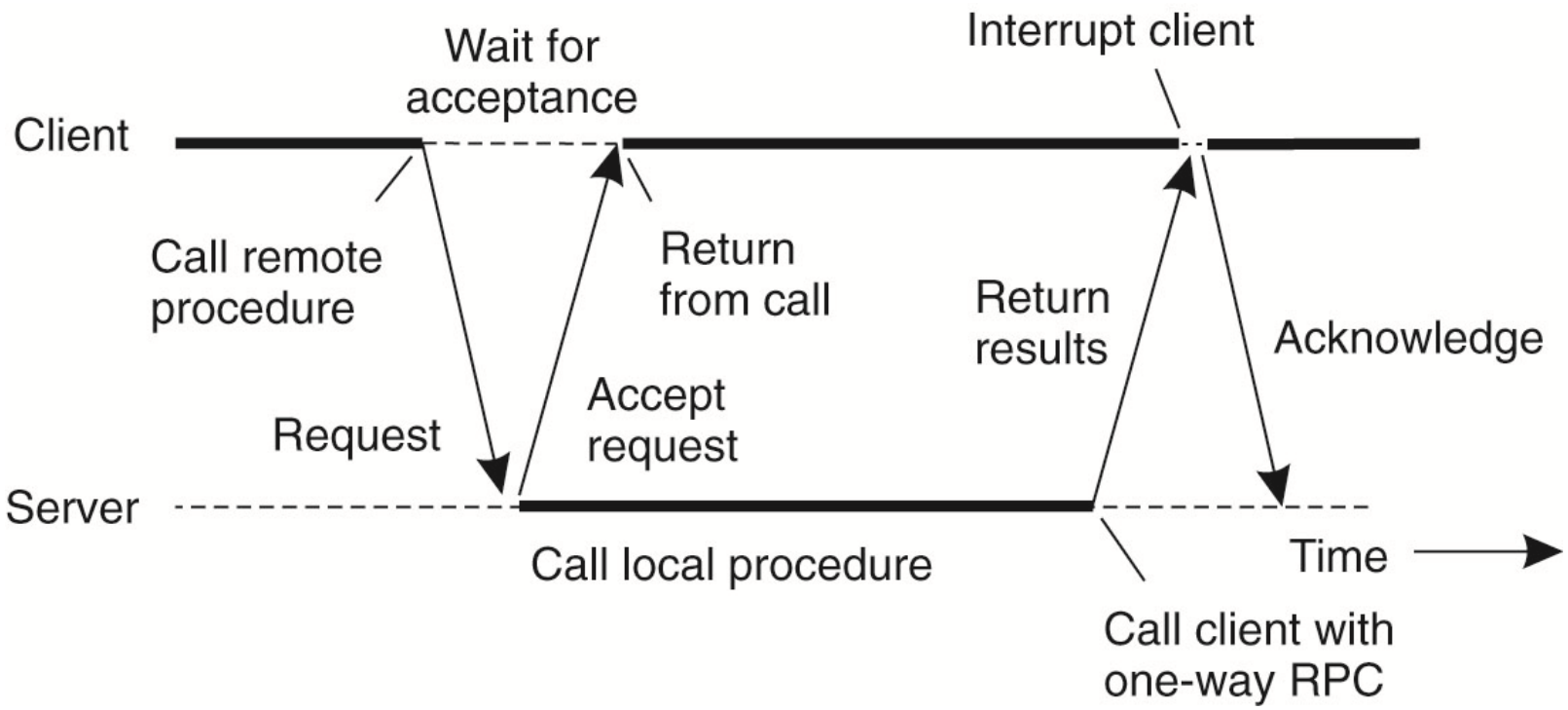
Asynchronous RPC (2)



(b)

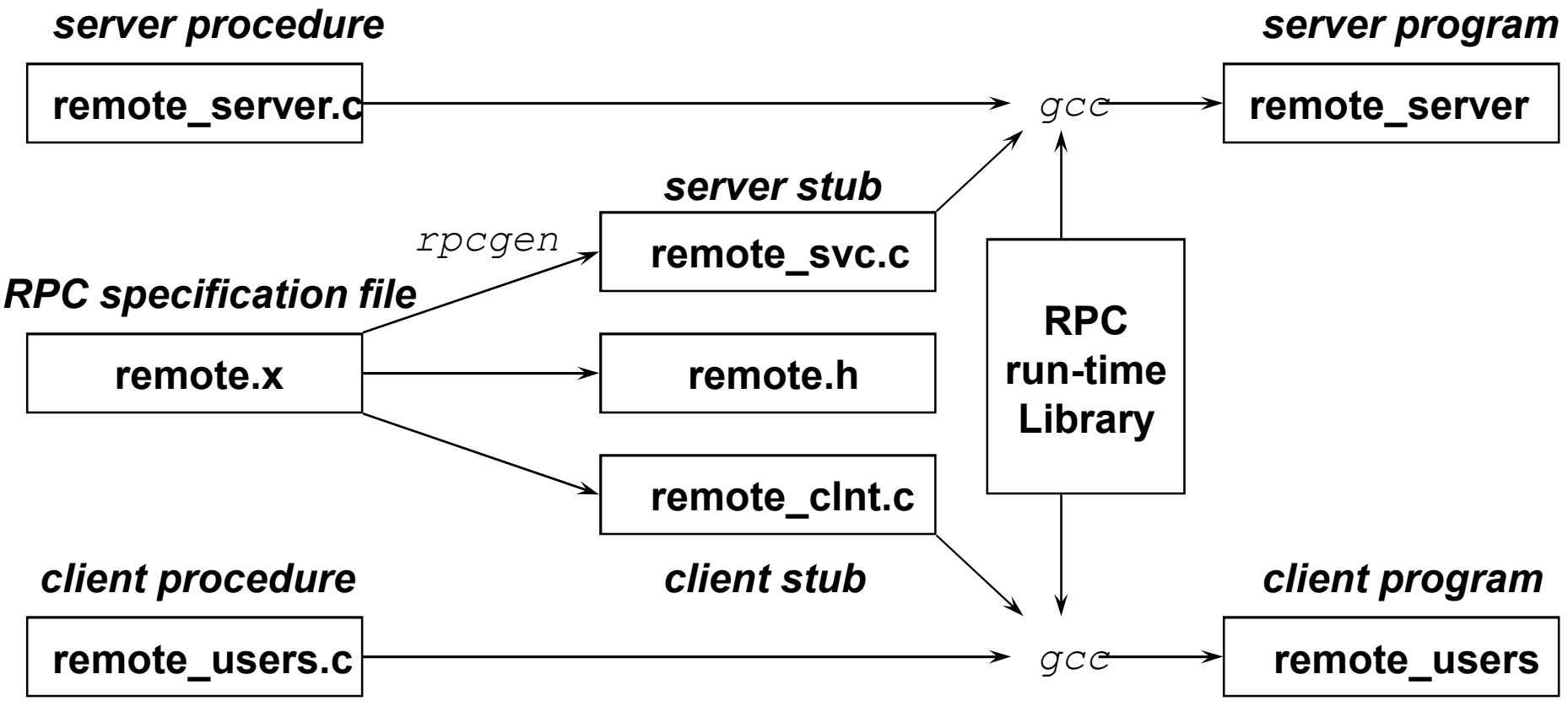
- (b) The interaction using asynchronous RPC. No need for a reply: money transfer, add entry to db

Asynchronous RPC (3)

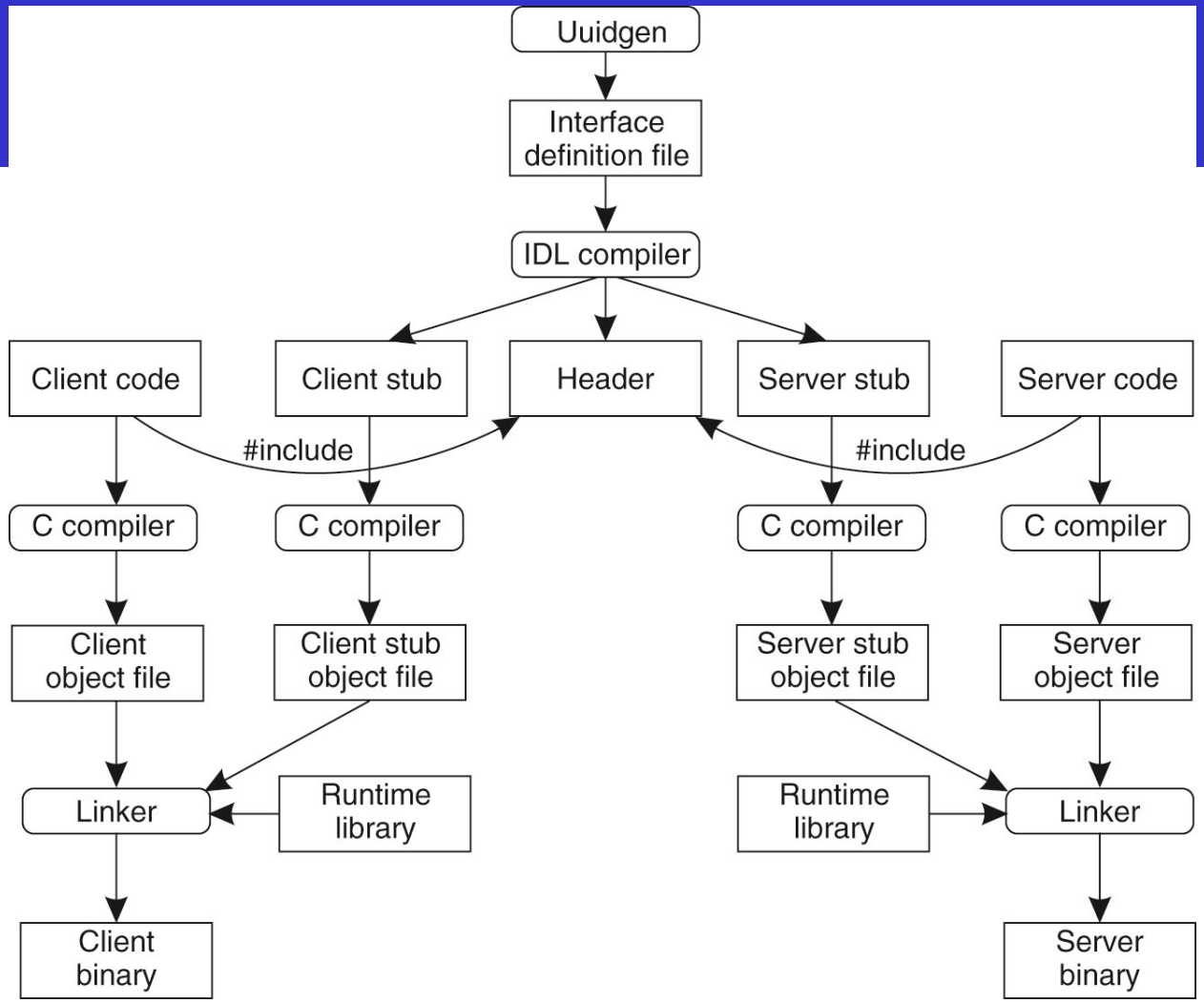


- A client and server interacting through two asynchronous RPCs. (name server: collecting addresses of a number of hosts

Client-Server Application using RPC



Writing a Client and a Server (1)



- The steps in writing a client and a server in DCE RPC.

Writing a Client and a Server (2)

- Three files output by the IDL compiler:

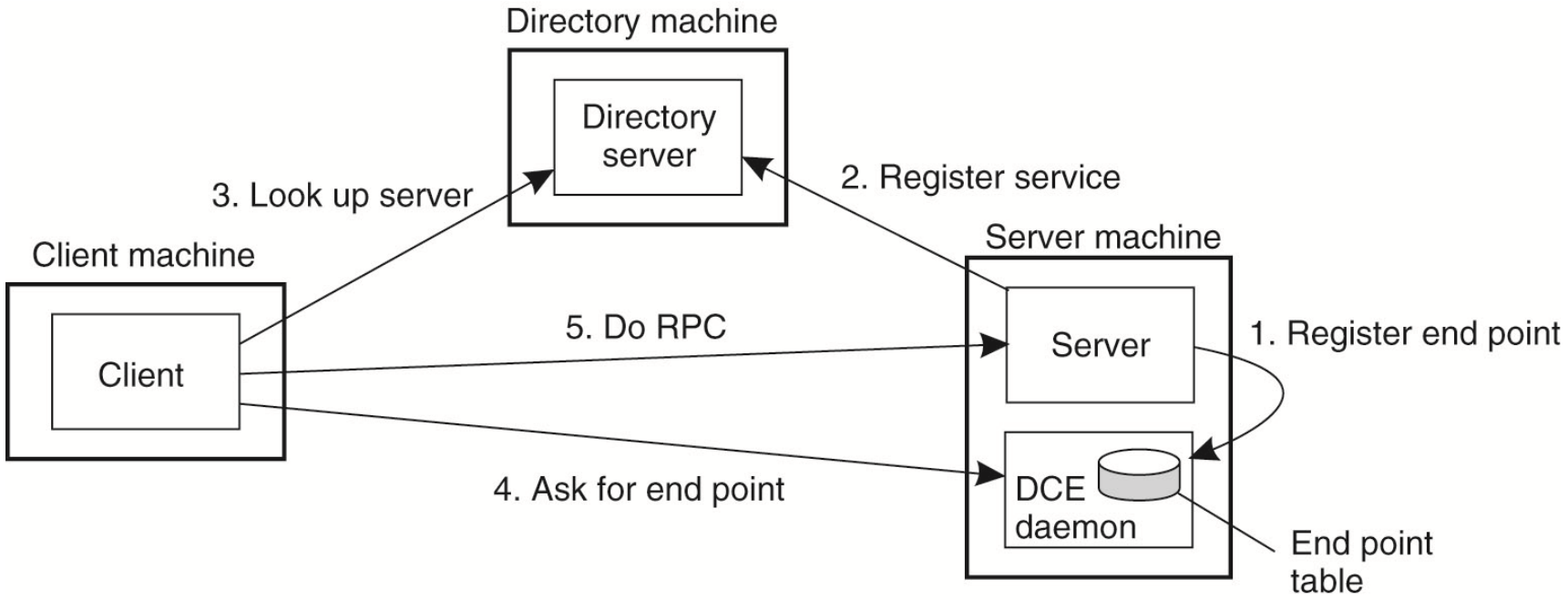
- A header file (e.g., interface.h, in C terms).
- The client stub.
- The server stub.

Binding a Client to a Server (1)

- Registration of a server makes it possible for a client to locate the server and bind to it.

- Locating server is done in two steps:
  1. Locate the server's machine.
  2. Locate the server process on that machine.

Binding a Client to a Server (2)



- Client-to-server binding in DCE.

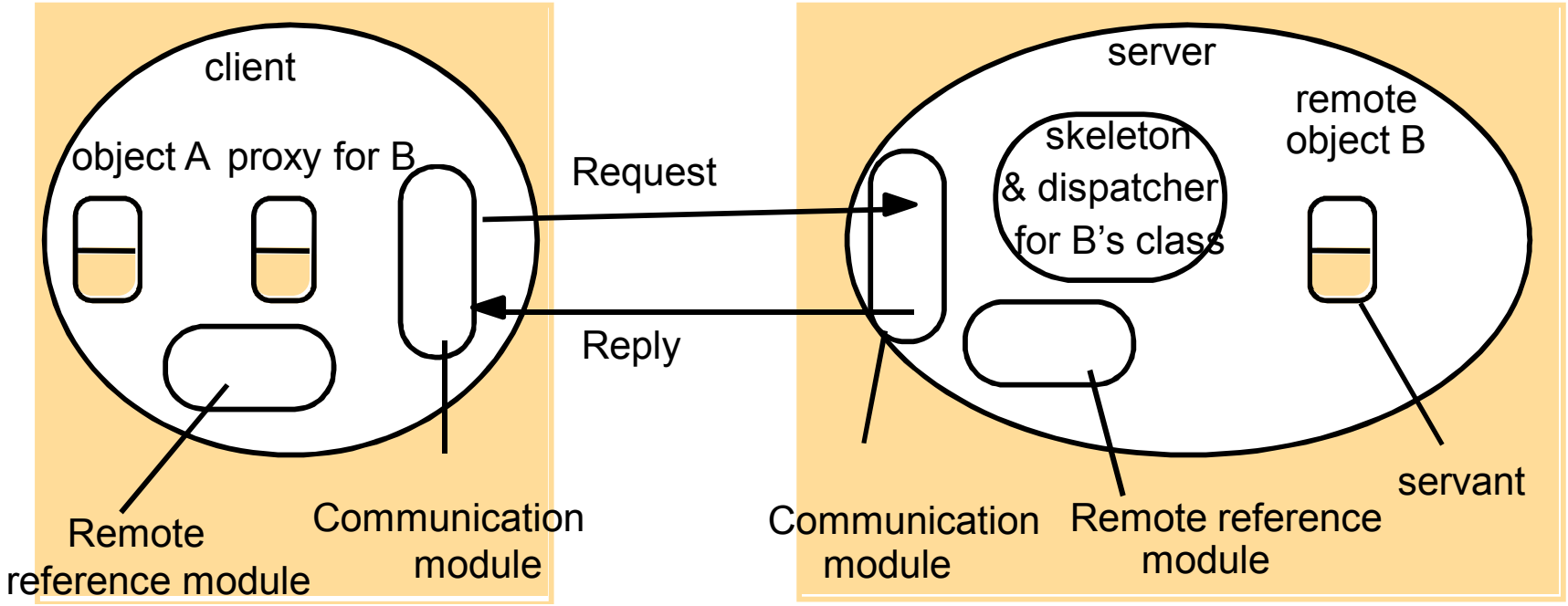
Java RMI nedir?

- Java Remote Method Invocation (Java RMI), programcının dağıtık Java teknolojisi tabanlı uygulamalar geliştirebilmesini sağlar. Farklı yerlerde bulunabilen Java sanal makinelerinden, uzak Java nesnelerinin metotları çalıştırılabilir.

- RMI nesne serileştirmesi (object serialization) kullanarak parametreleri kodlar (marshal) ve açar (unmarshal). Veri tiplerini koruyarak da gerçek object-oriented polymorphism'i destekler.

RMI Architecture & Components
RMI Mimarisi & Bileşenleri

- Remote reference module (Uzak referans modülü), proxy (stub) nesnesine adresleme sağlamakla sorumludur
- Proxy, stub oluşturmak ve istemciye saydamlık sağlamak için kullanılır. Doğrudan istemci tarafından çağrılır (uzak nesne proxynin kendisiymiş gibi), sonra da çağrıyı istek şeklinde gönderir
- Communication module (İlteişim modülü) ağ yönetiminden sorumludur
- Dispatcher (Yönlendirici) uygun iskeleti seçerek mesajı iletir
- Skeleton (İskelet) isteği açar ve uzak nesneyi çağırır



RMI Terimleri

- RMI, client ve remote object (uzak nesne) arasındaki bağlantıyı sağlamak için stub ve skeleton (iskelet) nesnelerini kullanır
- *Stub*, uzak nesne için bir *proxy*'dir. Client'tan asıl uzak nesne tanımının yer aldığı server'a metot çağrılarını iletir.
- Client'ın uzak bir nesneye referansı, yani aslında yerel stub'a referanstır. Client stub nesnesinin yerel bir kopyasını bulundurur.
- *Skeleton*, sunucu tarafında bulunan bir nesnedir. Çağrıları asıl uzak nesne tanımına yönlendiren bir metot bulundurur.
- Uzak bir nesne ile ilişkili yerel bir skeleton nesnesi vardır. Uzak çağrıları kendisine yönlendirir

Distributed Objects Technologies
Dağıtık Nesne Teknolojileri

- Ancak RPC object-oriented değildir. OO dünyasında, dağıtık nesneler ve uzak metot çağrıları kullanılır.
- Birçok Dağıtık Nesne Teknolojileri mevcut iken, bunların üçü oldukça yaygındır:
  - RMI
  - CORBA
  - SOAP
- Remote Method Invocation (RMI) (Uzak Metot Çalıştırma)
  - SUN tarafından geliştirilmiştir
  - Java API çekirdeğinin bir parçası olarak bulunur
  - Java-merkezli
  - Nesne interface'leri (arayüz) Java interface'leri olarak tanımlanmıştır
  - Object serialization kullanır (Nesne serileştirmesi)

Distributed Objects Technologies
Dağıtık Nesne Teknolojileri

- Common Object Request Broker Architecture (CORBA)
- Ortak Nesne İsteği Aracısı Mimarisi (CORBA)
  - Object Management Group (OMG) tarafından geliştirilmiştir
  - Dil ve platformdan bağımsızdır
  - Object interfaces defined in an Interface Definition Language (IDL)
    - Nesne arayüzleri bir Arayüz Tanımlama Dilinde (IDL) tanımlanır
  - Object Request Broker (ORB) client-server request/response işlemini kolaylaştırır
  - ORB'lar, Internet Inter-ORB Protocol (IIOP) denilen bir binary protokol aracılığı ile konuşurlar
- SOAP
  - Simple Object Access Protocol (Basit Nesne Erişimi Protokolü)
  - XML-Tabanlı
  - Önceden tanımlanan XML-RPC'den geliştirilmiştir.
  - W3C tarafından standartlaştırılmıştır
  - Birçok örneği mevcuttur

RMI

- Java uygulamaları için bir dağıtık nesne imkanı sağlar
- Bir Java metodunun uzak bir nesneye referans tutmasını ve uzak nesnenin metotlarını çağırmasını sağlar. Uzak nesne yerelmiş gibi kolayca erişilir.
- Uzak nesne, aynı bilgisayar üzerindeki başka bir JVM üzerinde, veya ağda bulunan diğer bilgisayarlar üzerinde bulunabilir
- Metot parametrelerinin paketlenip açılmasında nesne serileştirmesi (object serialization) kullanır.
- Gerekli class dosyalarının ağ üzerinden dinamik olarak indirilmesini destekler