Remote Procedure Call

Clients and Servers

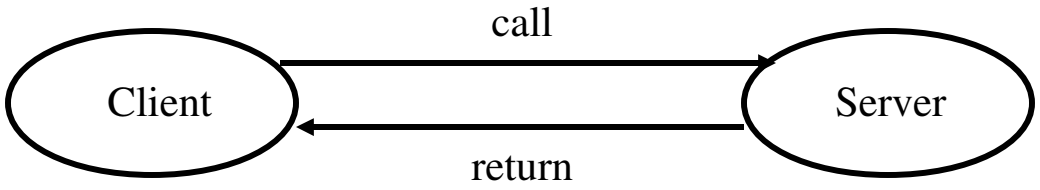
- A common model for structuring distributed computation is via the client/server paradigm
- A *server* is a program (or collection of programs) that provide some *service*, e.g., file service, name service, …
- The server may exist on one or more nodes.
- A *client* is a program that uses the service.
- A client first *binds* to the server, I.e., locates it in the network and establishes a connection.
- The client then sends *requests* to perform actions; this is done by sending messages that indicate which service is desired, along with params. The server returns a *response*.

The Problem with Messages

- While messages provide very flexible communication, they also have certain problems:
  - requires that programmer worry about message formats
  - messages must be packed and unpacked
  - messages have to be decoded by server to figure out what is requested
  - messages are often asynchronous
  - they may require special error handling functions
- Basically, messages are not a natural programming model for most programmers.

Procedure Call

- A more natural way to communicate is through procedure call:
  - every language supports it
  - semantics are well defined and understood
  - natural for programmers to use
- Basic idea: let's just define a server as a module that *exports* a set of procedures that can be called by client programs.
- To use the server, the client just does a procedure call, as if it were linked with the server



(Remote) Procedure Call

- So, we would like to use procedure call as a model for distributed communication.
- Lots of issues:
  - how do we make this invisible to the programmer?
  - what are the semantics of parameter passing?
  - how is binding done (locating the server)?
  - how do we support heterogeneity (OS, arch., language)
  - etc.

Remote Procedure Call

- The basic model for Remote Procedure Call (RPC) was described by Birrell and Nelson in 1980, based on work done at Xerox PARC.
- Goals was to make RPC look as much like local PC as possible.
- Used computer/language support.
- There are 3 components on each side:
  - a user program (client or server)
  - a set of *stub* procedures
  - RPC runtime support

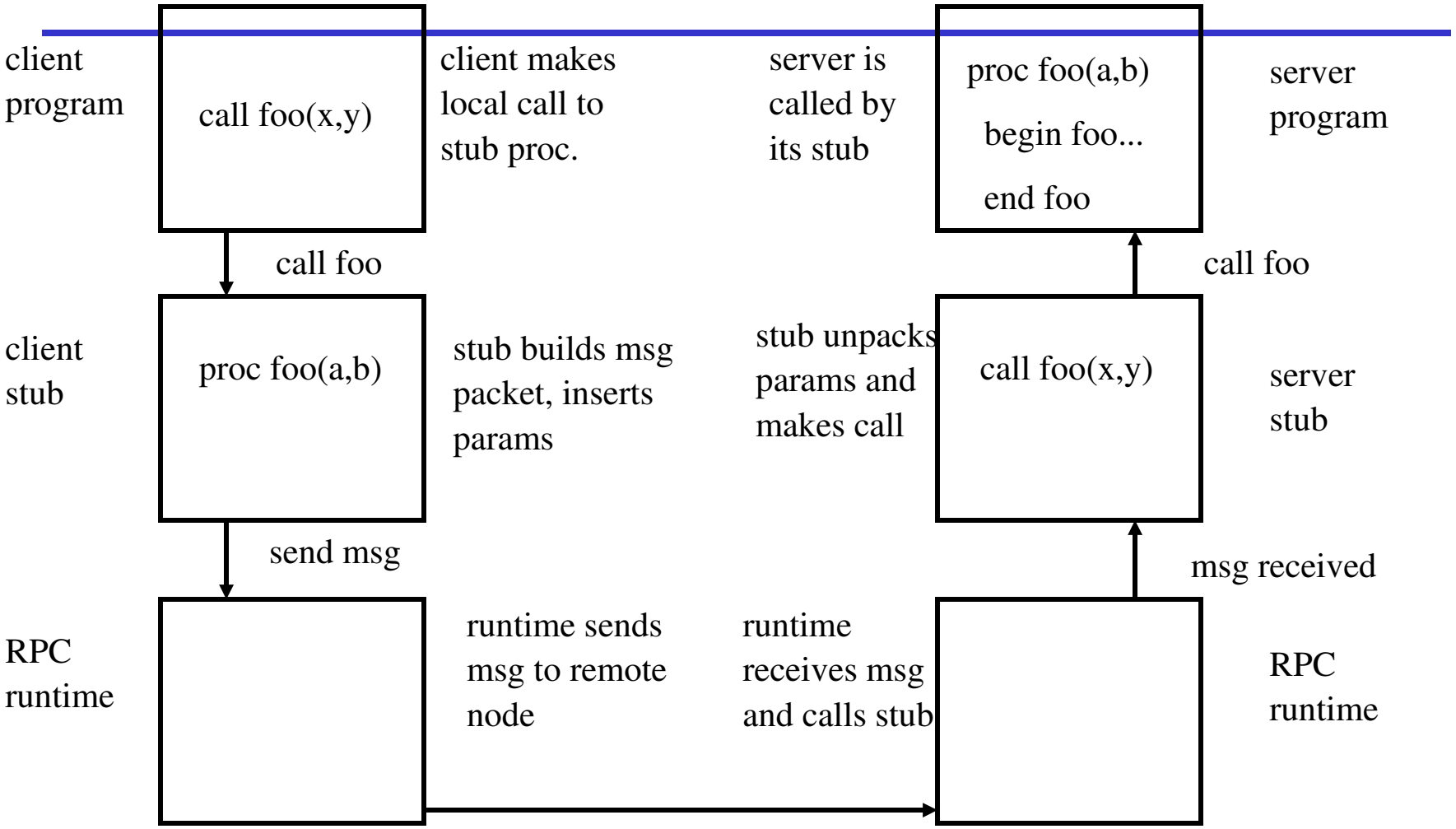
RPC

- Basic process for building a server:
  - Server program defines the server's interface using an *interface definition language* (IDL)
  - The IDL specifies the names, parameters, and types for all client-callable server procedures
  - A *stub compiler* reads the IDL and produces two stub procedures for each server procedure: a client-side stub and a server-side stub
  - The server writer writes the server and links it with the server-side stubs; the client writes her program and links it with the client-side stubs.
  - The stubs are responsible for managing all details of the remote communication between client and server.

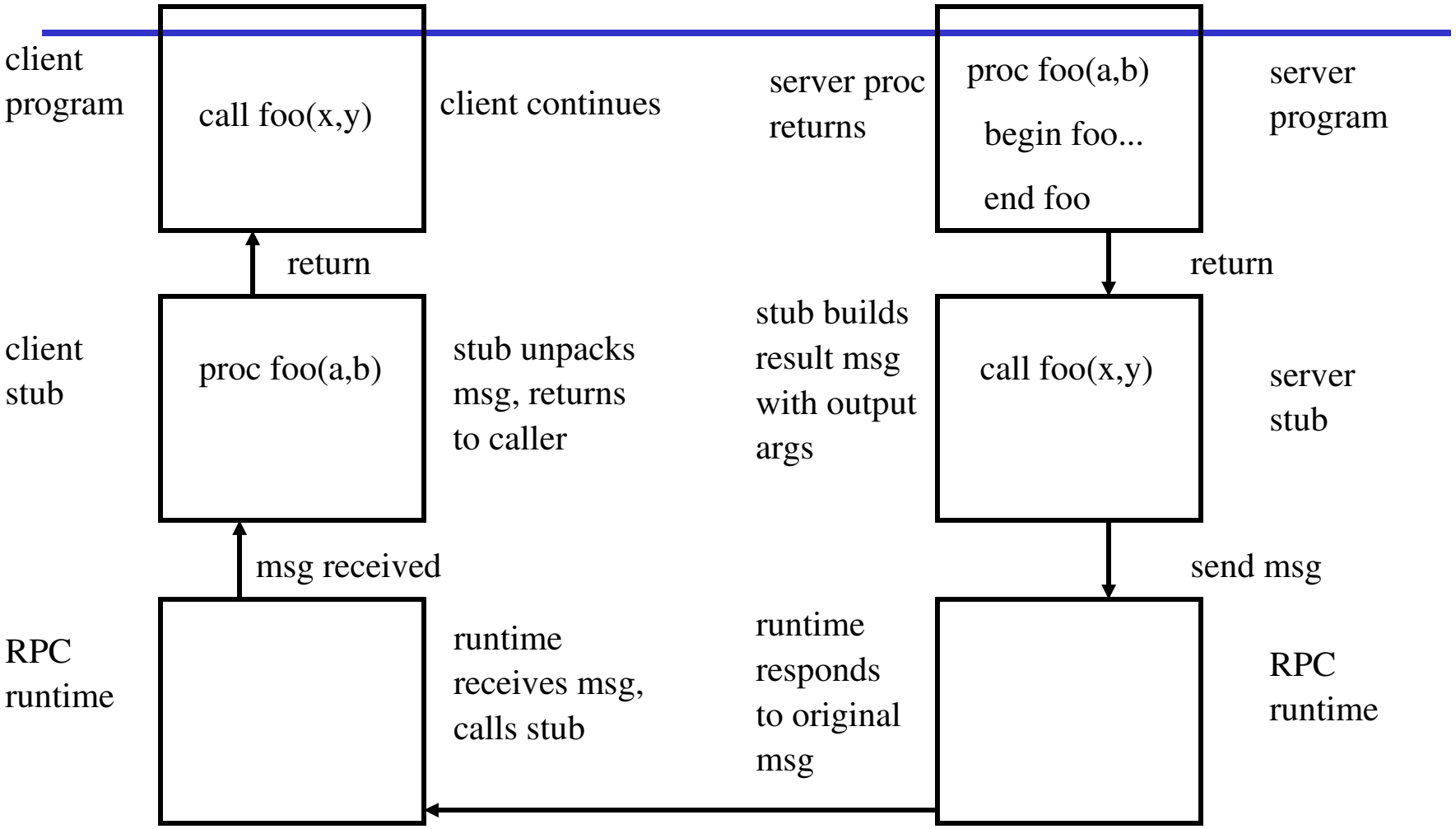
RPC Stubs

- Basically, a client-side stub is a procedure that looks to the client as if it were a callable server procedure.
- A server-side stub looks to the server as if it's a calling client.
- The client program thinks it is calling the server; in fact, it's calling the client stub.
- The server program thinks it's called by the client; in fact, it's called by the server stub.
- The stubs send messages to each other to make the RPC happen.

RPC Call Structure



RPC Return Structure



RPC Binding

- Binding is the process of connecting the client and server
- The server, when it starts up, *exports* its interface, identifying itself to a network name server and telling the local runtime its dispatcher address.
- The client, before issuing any calls, *imports* the server, which causes the RPC runtime to lookup the server through the name service and contact the requested server to setup a connection.
- The *import* and *export* are explicit calls in the code.

RPC Marshalling

- Marshalling is the packing of procedure parameters into a message packet.
- The RPC stubs call type-specific procedures to marshall (or unmarshall) all of the parameters to the call.
- On the client side, the client stub marshalls the parameters into the call packet; on the server side the server stub unmarshalls the parameters in order to call the server's procedure.
- On the return, the server stub marshalls return parameters into the return packet; the client stub unmarshalls return parameters and returns to the client.

RPC Final

- RPC is the most common model now for communications in distributed applications.
- RPC is essentially language support for distributed programming.
- RPC relies on a stub compiler to automatically produce client/server stubs from the IDL server description.
- RPC is commonly used, *even on a single node*, for communication between applications running in different address spaces. In fact, most RPCs are intra-node.