Remote Procedure Calls

Adapted from:
Paul Krzyzanowski
pxk@cs.rutgers.edu
ds@pk.org

Network Transfer Protocols

Connection Oriented

- often reliable, stream based
- analogous to making a direct phone call
- ***TCP (Transmission Control Protocol)*** is a connection-based protocol that provides a reliable flow of data between two computers

Connectionless

- unreliable, datagram based
- analogous to sending letters via the postal service
- ***UDP (User Datagram Protocol)*** is a protocol that sends independent packets of data, called ***datagrams***, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP.

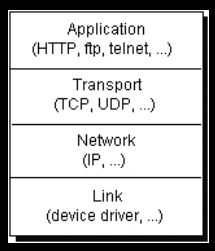
Internet Protocol and the TCP/IP Protocol Suite

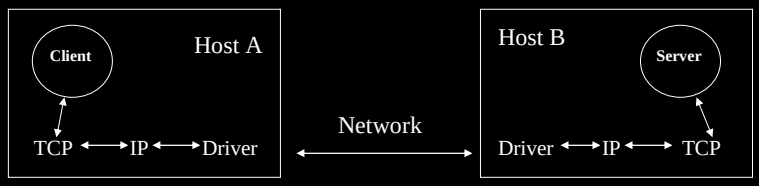
• ***IP (Internet Protocol)*** provides an unreliable, connectionless datagram delivery service.

• TCP/IP is a set of protocols created specifically to allow development of network and internetwork communications on a global scale.

• TCP/IP is the most commonly used protocols within the internet.

• TCP/IP is normally considered to be a four-***layer*** system





Sockets

• Sockets (Berkley sockets) are one of the most widely used communication APIs.

• A socket is an object from which messages and are sent and received.

• A socket is a network communication endpoint.

• In connection-based communication such as TCP, a server application binds a ***socket*** to a specific port number. This has the effect of registering the server with the system to receive all data destined for that port. A client can then rendezvous with the server at the server's port, as illustrated here:

• Data transfer operations on sockets work just like read and write operations on files. A socket is closed, just like a file, when communications is finished.

• Network communications are conducted through a pair of cooperating sockets, each known as the ***peer*** of the other.

• Processes connected by sockets can be on different computers (known as a ***heterogenous*** environment) that may use different data representations.

• Data is ***serialized*** into a sequence of bytes by the local sender and ***deserialized*** into a local data format at the receiving end.

Problems with sockets

Sockets interface is straightforward

- [connect]
- read/write
- [disconnect]

BUT … it forces read/write mechanism

- We usually use a procedure call

To make distributed computing look more like centralized:

- I/O is not the way to go

RPC

1984: Birrell & Nelson

- Mechanism to call procedures on other machines

Remote Procedure Call

Goal: it should appear to the programmer that a normal call is taking place

Regular procedure calls

Machine instructions for call & return but the compiler really makes the procedure call abstraction work:

- Parameter passing
- Local variables
- Return data

Regular procedure calls

You write:

```
x = f(a, "test", 5);
```

The compiler parses this and generates code to:

a. Push the value 5 on the stack
b. Push the address of the string "test" on the stack
c. Push the current value of a on the stack
d. Generate a call to the function f

In compiling f, the compiler generates code to:

a. Push registers that will be clobbered on the stack to save the values
b. Adjust the stack to make room for local and temporary variables
c. Before a return, unadjust the stack, put the return data in a register, and issue a return instruction

Implementing RPC

No architectural support for remote procedure calls

Simulate it with tools we have (local procedure calls)

Simulation makes RPC a
language-level construct
instead of an
operating system construct

Implementing RPC

The trick:

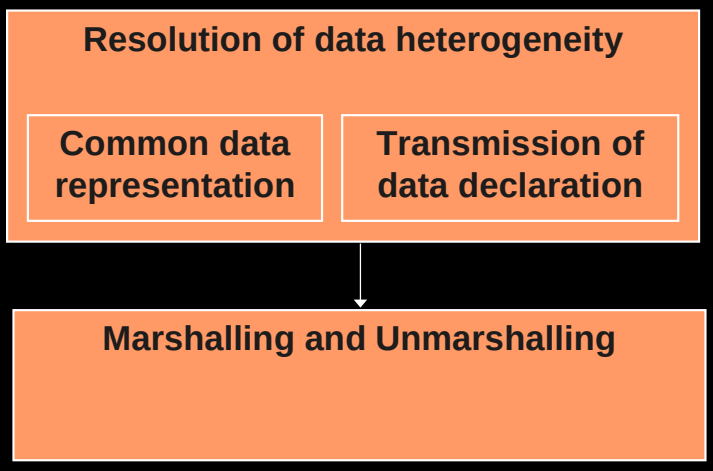
Create **stub functions** to make it appear to the user that the call is local

Stub function contains the function's interface

Remote Procedue Calls

- Enable procedure calls across host boundaries
- Call interfaces are defined using an Interface Definition Language (IDL)
- RPC compiler generates presentation and session layer implementation from IDL

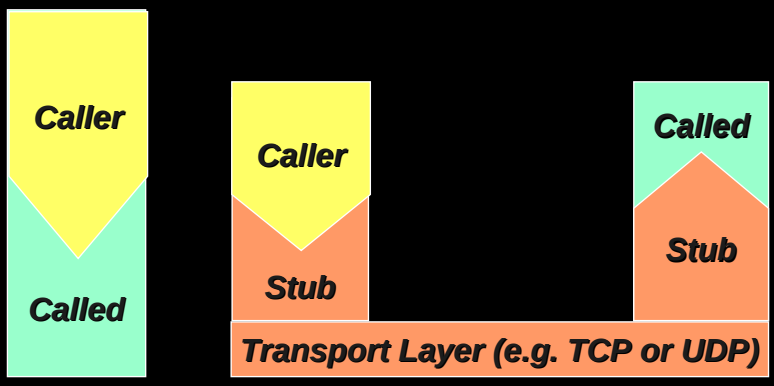
ISO/OSI Presentation Layer



Marshalling and Unmarshalling

- Marshalling: Disassemble (encode) data structures into transmittable form
- Unmarshalling: Reassemble (decode) transmitted form into original complex data structure.

Method Call vs. Object Request

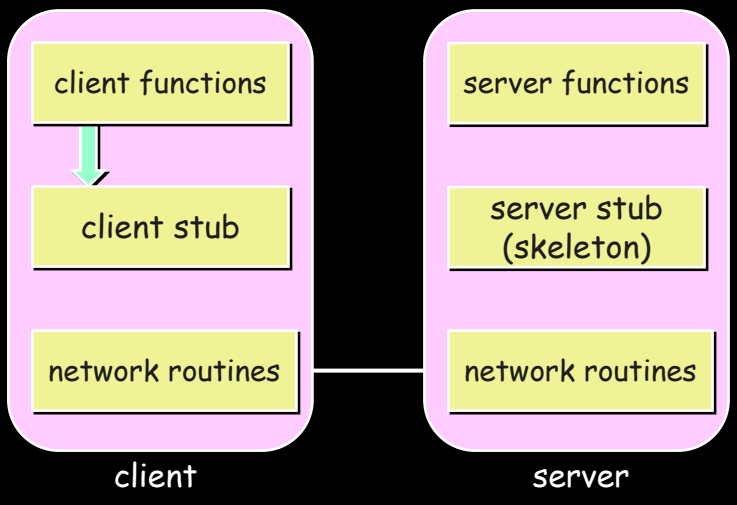


Stubs

- Creating code for marshalling and unmarshalling is tedious and error-prone.
- Code can be generated fully automatically from interface definition.
- Code is embedded in stubs for client and server.
- Client stub represents server for client, Server stub represents client for server.
- Stubs achieve type safety.
- Stubs also perform synchronization.

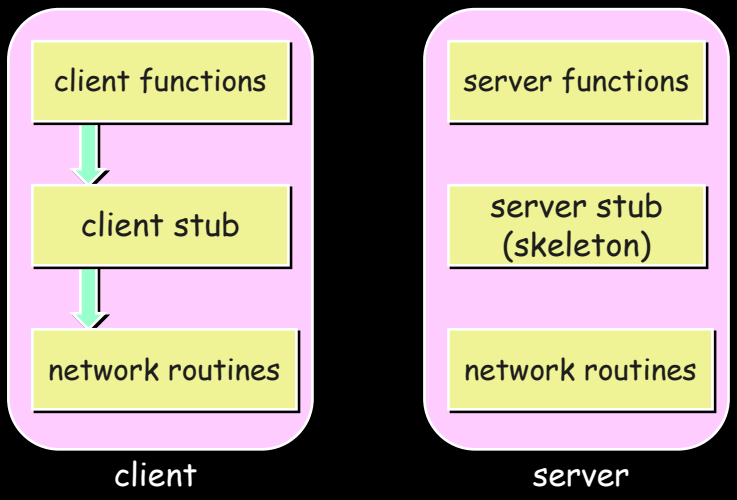
Stub functions

1. Client calls stub (params on stack)



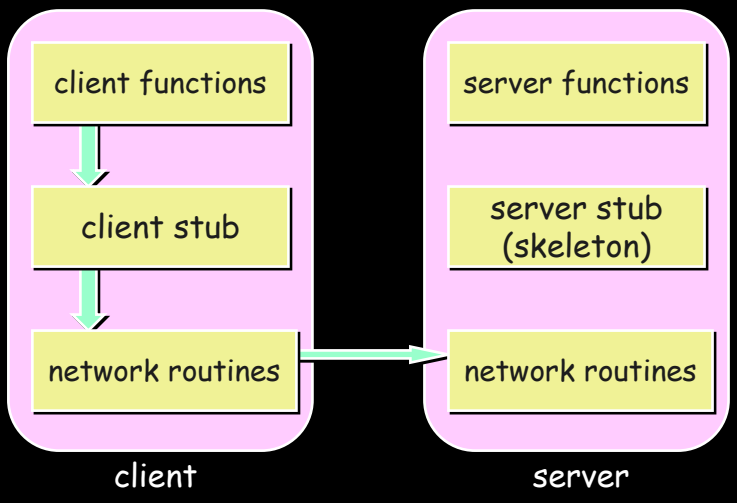
Stub functions

2. Stub marshals params to net message



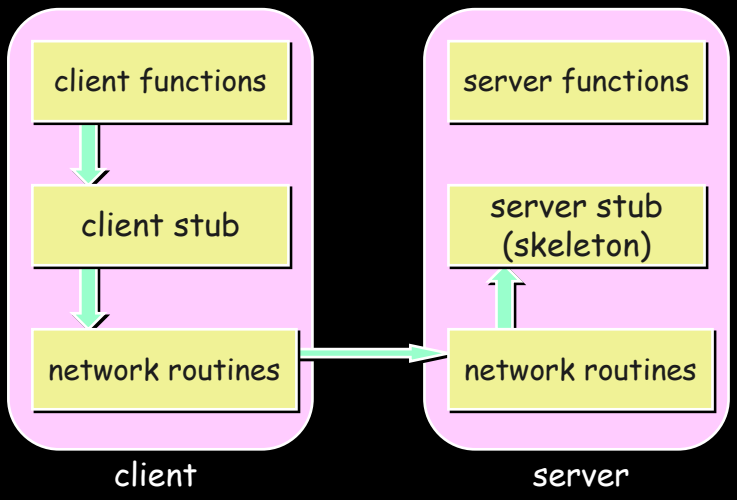
Stub functions

3. Network message sent to server



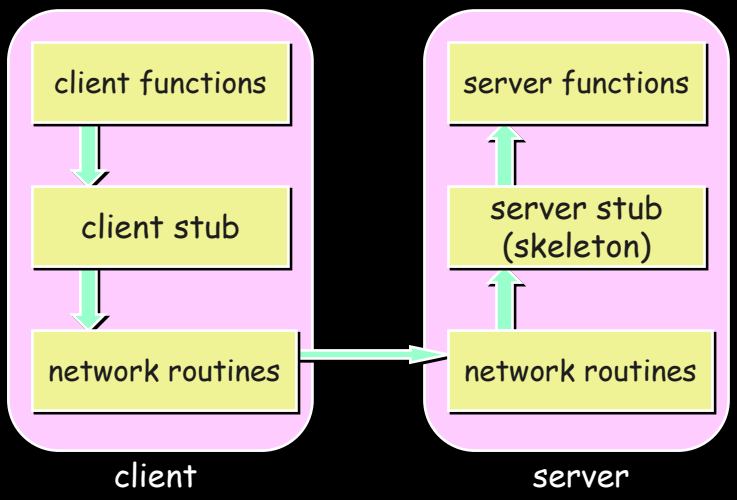
Stub functions

4. Receive message: send to stub



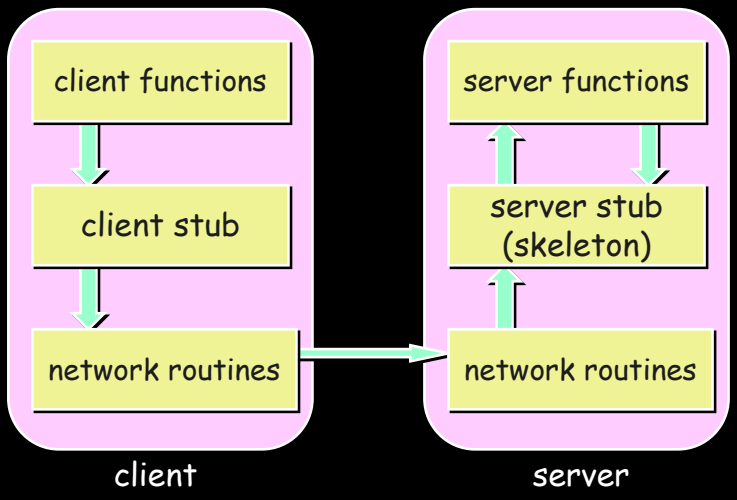
Stub functions

5. Unmarshal parameters, call server func



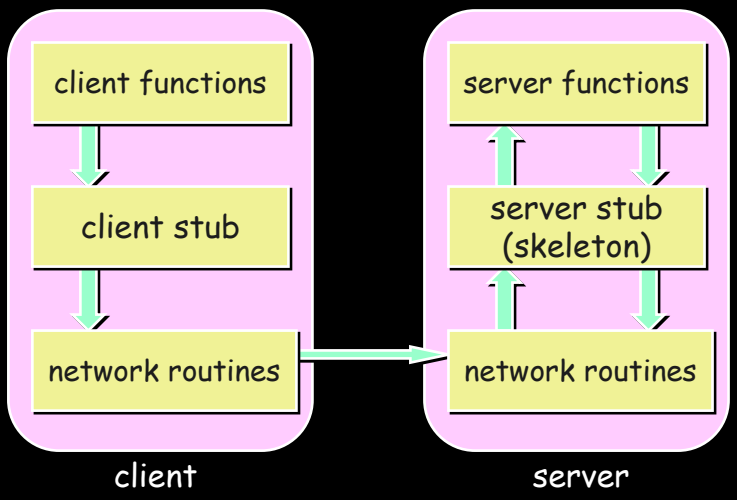
Stub functions

6. Return from server function



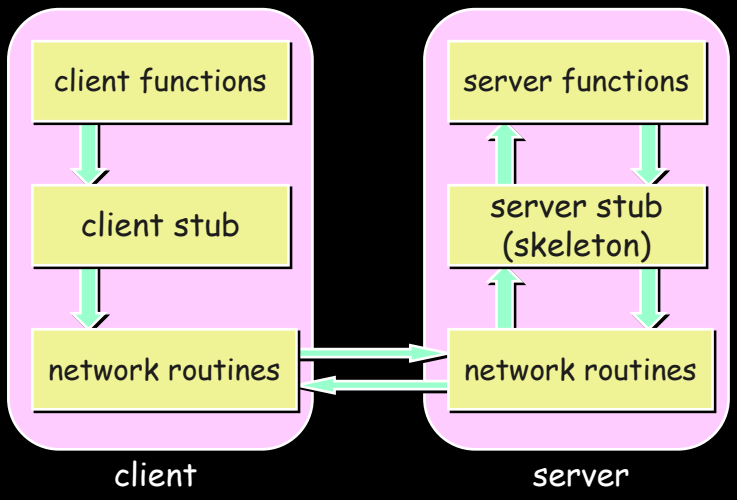
Stub functions

7. Marshal return value and send message



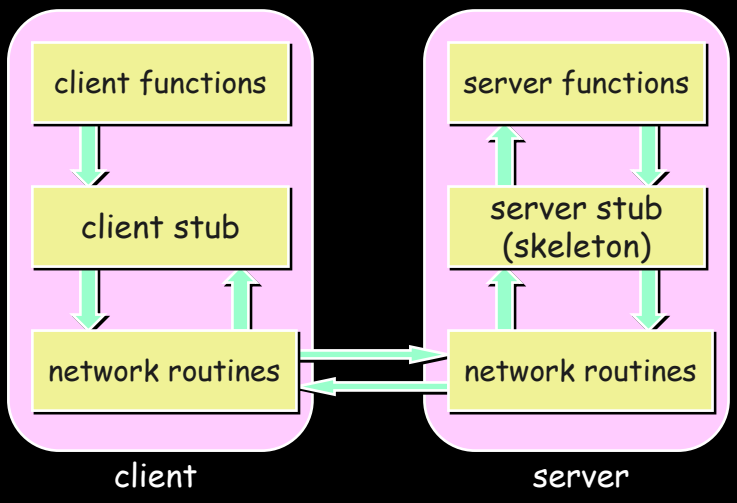
Stub functions

8. Transfer message over network



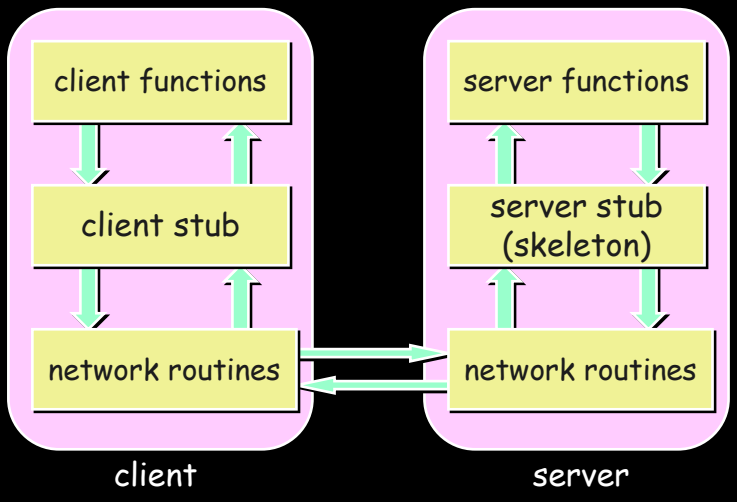
Stub functions

9. Receive message: direct to stub



Stub functions

10. Unmarshal return, return to client code



Benefits

- Procedure call interface
- Writing applications is simplified
  - RPC hides all network code into stub functions
  - Application programmers don't have to worry about details
    - Sockets, port numbers, byte ordering
- RPC: presentation layer in OSI model

RPC has issues

Parameter passing

Pass by value

- Easy: just copy data to network message

Pass by reference

- Makes no sense without shared memory

Pass by reference?

1. Copy items referenced to message buffer
2. Ship them over
3. Unmarshal data at server
4. Pass *local* pointer to server stub function
5. Send new values back

To support complex structures

- Copy structure into pointerless representation
- Transmit
- Reconstruct structure with local pointers on server

Representing data

No such thing as
incompatibility problems on local system

Remote machine may have:

- Different byte ordering
- Different sizes of integers and other types
- Different floating point representations
- Different character sets
- Alignment requirements

Where to bind?

Need to locate host and correct server process

1. Maintain centralized DB that can locate a host that provides a particular service *(Birrell & Nelson's 1984 proposal)*
2. A server on each host maintains a DB of *locally* provided services

Transport protocol

Which one?

- Some implementations may offer only one (e.g. TCP)
- Most support several
  - Allow programmer (or end user) to choose

When things go wrong

- Local procedure calls do not fail
  - If they core dump, entire process dies
- More opportunities for error with RPC:
- Transparency breaks here
  - Applications should be prepared to deal with RPC failure

When things go wrong

- Semantics of remote procedure calls
  - Local procedure call: ***exactly once***
- A remote procedure call may be called:
  - 0 times: server crashed or server process died before executing server code
  - 1 time: everything worked well
  - 1 or more: excess latency or lost reply from server and client retransmission

RPC semantics

- Most RPC systems will offer either:
  - ***at least once*** semantics
  - or ***at most once*** semantics

- Understand application:
  - **idempotent** functions: may be run any number of times without harm
  - **non-idempotent** functions: side-effects

More issues

Performance

- RPC is slower … a lot slower

Security

- messages visible over network
- Authenticate client
- Authenticate server

Programming with RPC

Language support

- Most programming languages (C, C++, Java, …) have no concept of remote procedure calls
- Language compilers will not generate client and server stubs

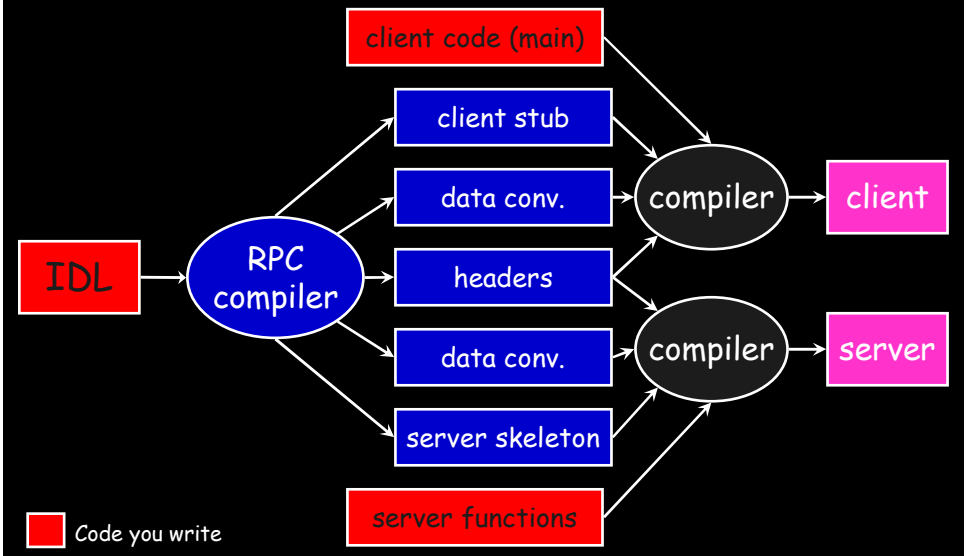
Common solution:

- Use a separate compiler to generate stubs (pre-compiler)

Interface Definition Language

- Allow programmer to specify remote procedure interfaces
  (names, parameters, return values)
- Pre-compiler can use this to generate client and server stubs:
  - Marshaling code
  - Unmarshaling code
  - Network transport routines
  - Conform to defined interface
- Similar to function prototypes

RPC compiler



Writing the program

Client code has to be modified

- Initialize RPC-related options
  - Transport type
  - Locate server/service
- Handle failure of remote procedure call

Server functions

- Generally need little or no modification

RPC API

What kind of services does an RPC system need?

- Name service operations
  - Export/lookup binding information (ports, machines)
  - Support dynamic ports
- Binding operations
  - Establish client/server communications using appropriate protocol (establish endpoints)
- Endpoint operations
  - Listen for requests, export endpoint to name server

RPC API

What kind of services does an RPC system need?

- Security operations
  - Authenticate client/server
- Internationalization operations
- Marshaling/data conversion operations
- Stub memory management
  - Dealing with "reference" data, temporary buffers
- Program ID operations
  - Allow applications to access IDs of RPC interfaces