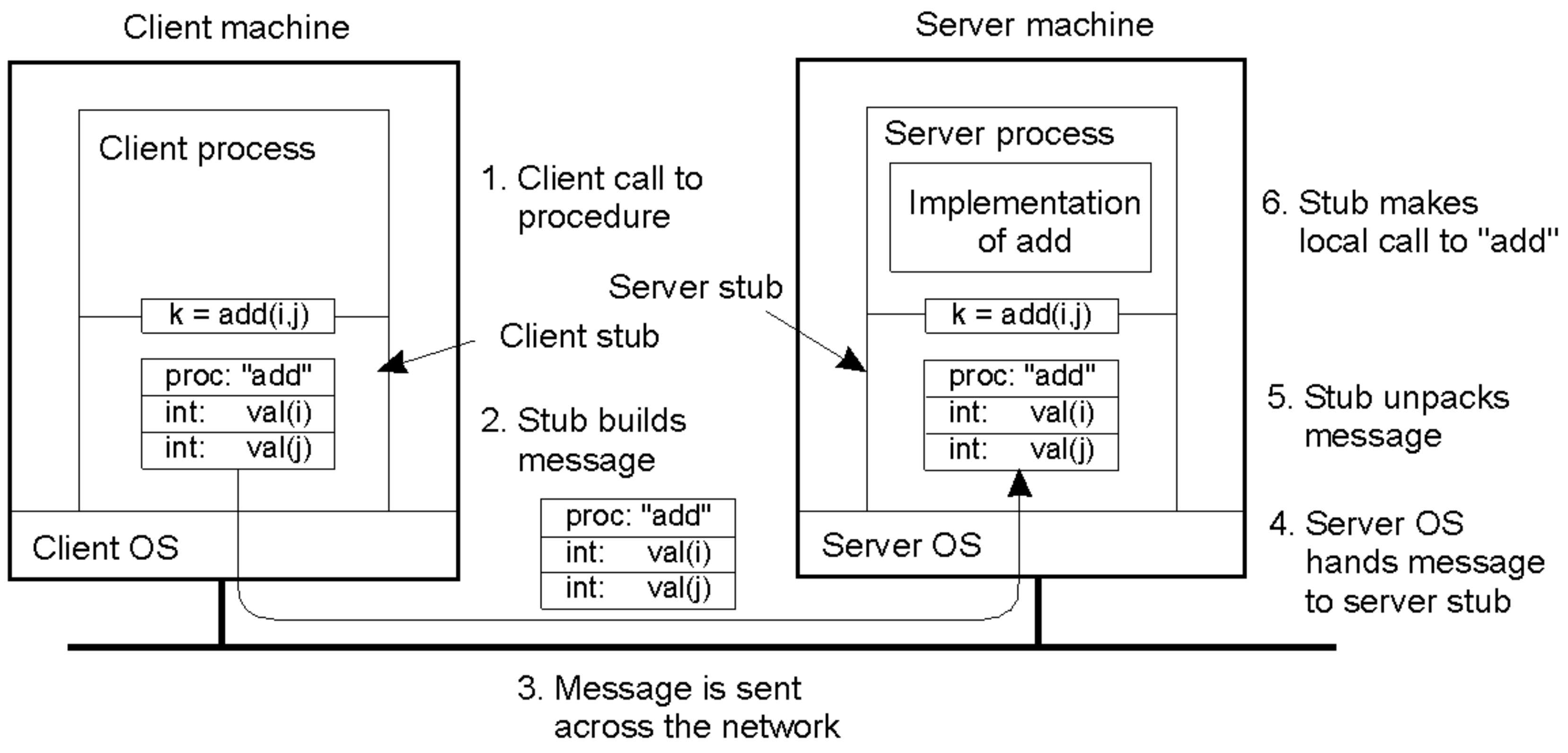
Lecture 5: RPC (exercises/questions)

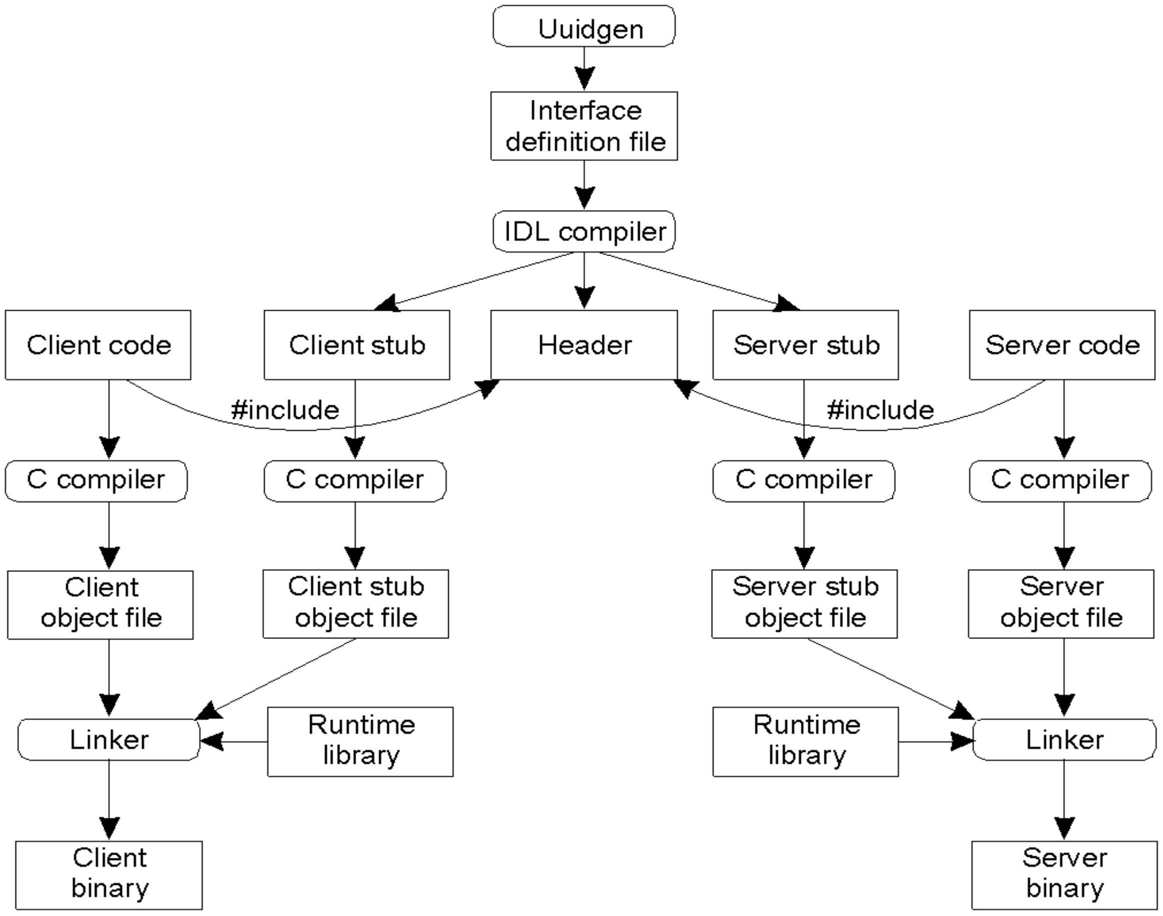
First Six Steps of RPC
TvS: Figure 4-7



Remaining Steps

- 7: **add** executes and returns answer to server stub
- 8: server stub composes reply message containing this answer
- 9: message is sent across the network
- 10: client OS passes reply to client stub
- 11: client stub unpacks and returns result

Producing a Client and a Server TVS Figure 4-12



Passing Reference Parameters

- In a language, such as C, procedure call parameters can be call-by-value or call-by-reference. In the first case, modifications by the procedure are valid only within a procedure. In the second case, a pointer is passed and modifications affect the memory location.
- Pointers are meaningful only within the address space of the process that uses them!
- Call by copy/restore can be used in RPC to replace call-by-reference. E.g.:
  - Client stub copies contents of a pointer to an array of characters to a message (assumes knowledge of size of the array)
  - Message sent to the server. Server stub provides pointer to the server to this array of characters. Modifications take place to this array of characters.
  - When the server finishes, the original message can be sent back to the client stub that copies it to the client.

(still cannot handle the general case of a pointer to an arbitrary data structure)

(see Tanenbaum, Sections 4.2.1, 4.2.2)

An Exercise

Given the following C code:

```
void doIt1 (int *p) {
  *p += 1 ;
  return ;
}
void doIt2 (int *a, int *b) {
  *a += 1 ;
  *b += 2 ;
  return ;
}

int main (int n, char **args) {
  int i = 1 ;
  doIt1 (&i) ;
  printf ("%i\n, i) ;
  doIt2 (&i, &i) ;
  printf ("%i\n", i) ;
}
```

What values would you expect to be printed for i normally?

What values would be printed if doIt1 and doIt2 were called using RPC instead of just being local functions?

Assume that the RPC knew, from the IDL, that the parameters were just "inout" references to ints so **copy/restore** is used.

Questions (1)

- Where does the need for <u>at-least-once</u> and <u>at-most-once</u> semantics come from? Why can't we have exactly-once semantics?
- Consider a client/server system based on RPC, and assume the server is replicated for performance. Sketch an RPC-based solution for hiding replication of the server from the client.
- Traditional RPC mechanisms cannot handle pointers. What is the problem and how can it be addressed?

Questions (2)

- Explain what is in the runtime library in the Figure of slide 4.
- Executing an RPC requires that a client can contact a server. How does it find the contact point for a server, and what does that contact point consist of?
- Explain the principal operation of a remote procedure call (RPC).
- An RPC to a replicated server can be made highly transparent to caller and callee with respect to access, replication, and failure transparency. How?

Questions (3)

- What is the major disadvantage of using RPCs in comparison to messaging as in message-queuing systems?
- Give two compelling arguments why an RPC can never provide the same semantics as a local procedure call.
- What is the main difference between a remote method invocation (RMI) and an RPC?
- Give a typical example of client-side software that is needed in a distributed system.

Answers

- An exercise: 2 and 5 on a single machine. Using copy/restore: 2 and either 3 or 4.
- The problem is how to deal with a suspected server crash. Exactly once is impossible because you don't know if the server has crashed before or after executing the requested operation.
- Simply take a client stub that replicates the call to the respective servers and execute these calls in parallel.
- Pointers refer to a memory location that is a local to the caller and meaningless to the recipient.
- The runtime library contains calls to the underlying transport-level interface or routines for converting data structures to host-independent representations.
- Contact a name server; the contact point will consist of an IP address and port number.
- RPC: you need to explain the figure in slide 2.
- Place the replicated call inside the client-side stub from where it can simply be executed in parallel to the servers.
- The main disadvantage is the synchronous and nonpersistent nature of RPCs – when a fault occurs it has to be dealt immediately.
- Access transparency cannot be achieved as it is impossible to pass pointers to local data structures. Also, masking failures (as we operate across a network) is a major issue.
- RMI provides a system-wide object reference mechanism, hence objects can be referenced from a remote machine.
- Any answer related to client-side stubs for implementing RPC, handling replication, fault-tolerence…