L7 Remote Procedure Call

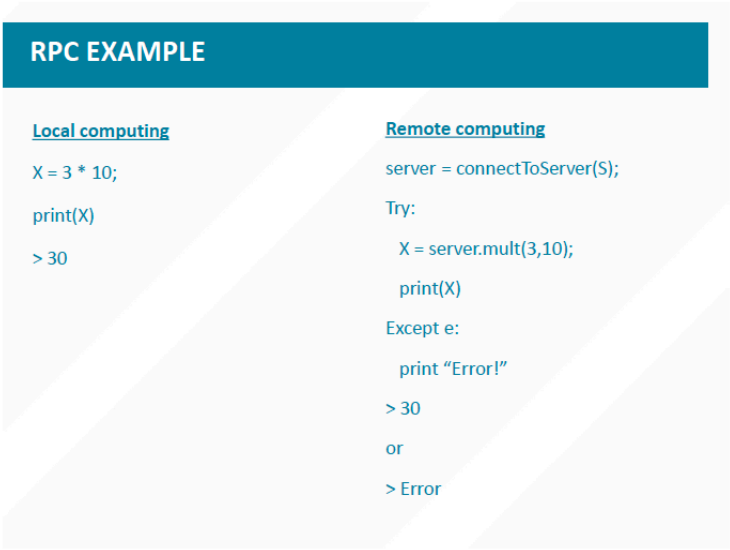
Why RPC?

Retain the feel of writing centralized code, easy to program network communication that makes client-server communication

Distributed programming is hard ==> abstraction

Goal of RPC

To make commnuication appear like a local procedure call : transparency for procedure calls



Difficulties

- Heterogeneity

  Client needs to communicate with server, but how to solve it when servers are of different types?

  1. They may have different data representations.
  - represent data using different sizes
  - different byte ordering

    X86-64: little endian some: big endian
  - represent floating numbers differently
  - have different data alignment
  2. Language support varies
  - Many programming language have no inbuilt concept of RPC(C, C++, eariler Java)
  - Some have: Python, Haskell, Go

- Failure

  Message dropped

  client, server, network fails
- Performace

  RPC in a datacenter is 1000 times slower than procedure calls

Solution

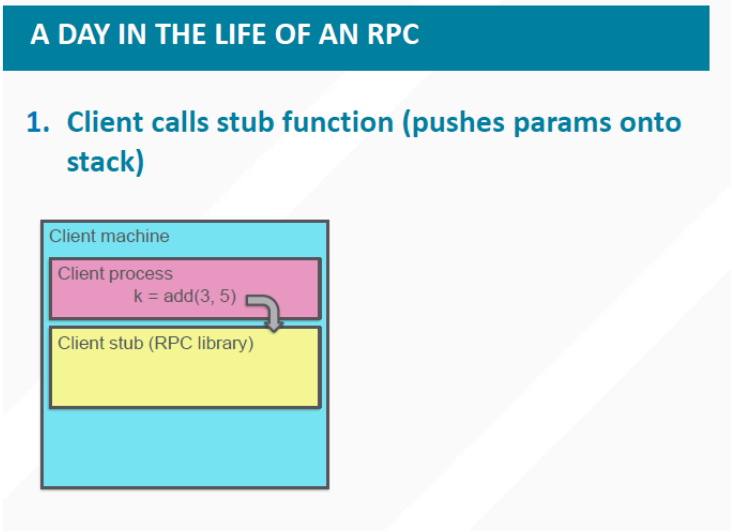
Write an interface description in the IDL

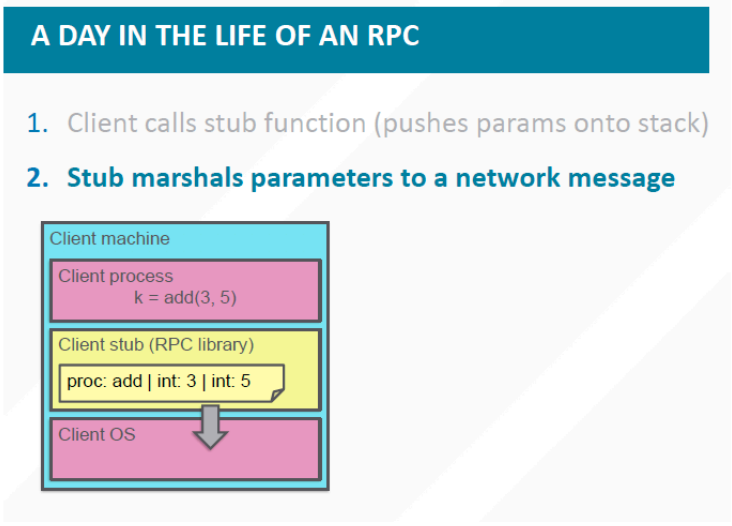
Use IDL Compiler to convert native data types with machine-independent byte streams

Client stub: forward procedure calls as a request to sever

Server stub: dispatch RPC to its implementation

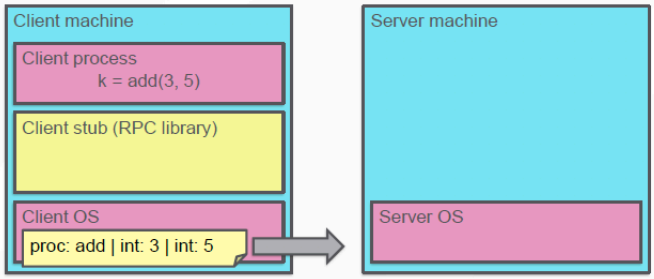
A Day of an RPC





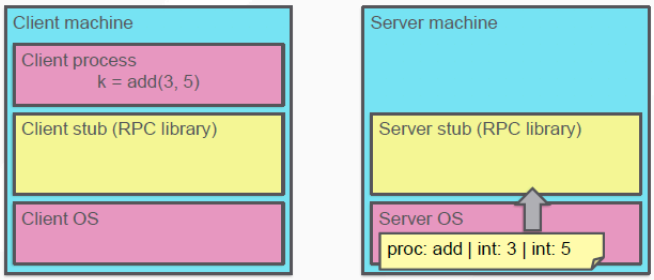
A DAY IN THE LIFE OF AN RPC

2. Stub marshals parameters to a network message
3. **OS sends a network message to the server**



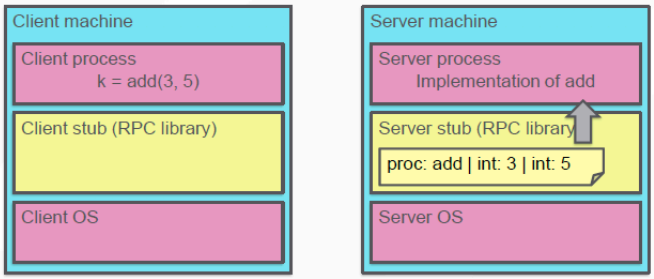
A DAY IN THE LIFE OF AN RPC

3. OS sends a network message to the server
4. **Server OS receives message, sends it up to stub**



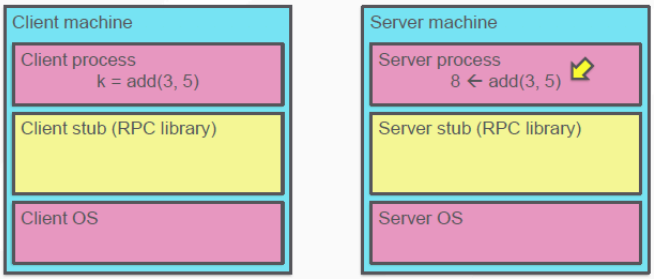
A DAY IN THE LIFE OF AN RPC

4. Server OS receives message, sends it up to stub
5. **Server stub unmarshals params, calls server function**



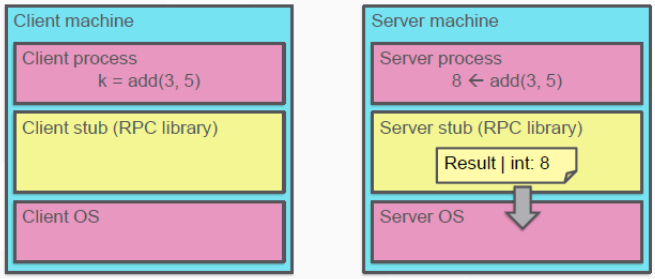
A DAY IN THE LIFE OF AN RPC

5. Server stub unmarshals params, calls server function
6. **Server function runs, returns a value**



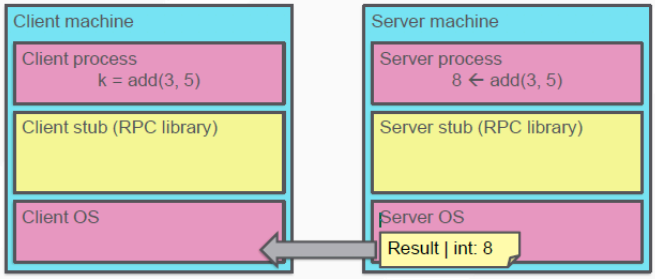
A DAY IN THE LIFE OF AN RPC

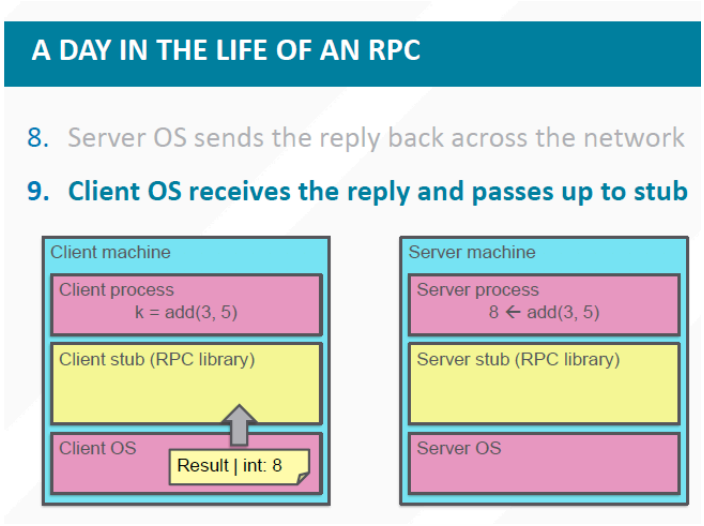
6. Server function runs, returns a value
7. **Server stub marshals the return value, sends msg**

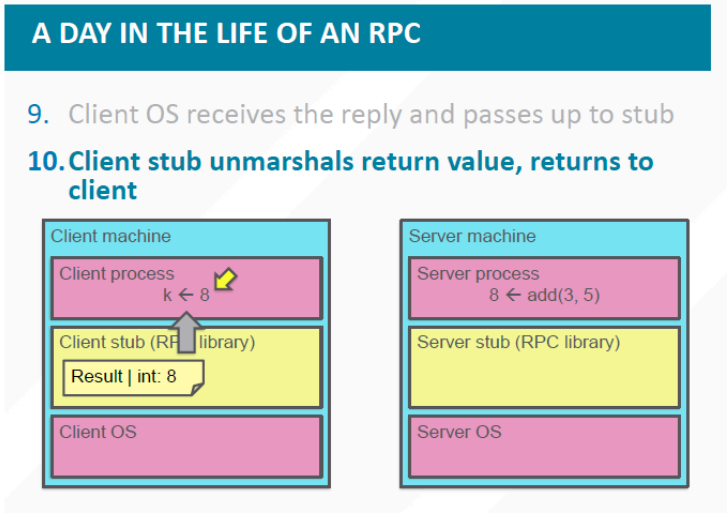


A DAY IN THE LIFE OF AN RPC

7. Server stub marshals the return value, sends msg
8. **Server OS sends the reply back across the network**







Server Stub

- Dispatcher

  Receives a client's RPC request, identifies appropriate server-side method to invoke
- Skeleton

  Unmarshals parameters to sever-native types

  Calls the local server procedure

  Marshals the response, sends it back to the dispatcher

Handling failure

Four kinds of failures may occur

- Client crash and reboot
- Packet dropped (packet loss, broken routing)
- Server crash and reboot
- network or server by very slow

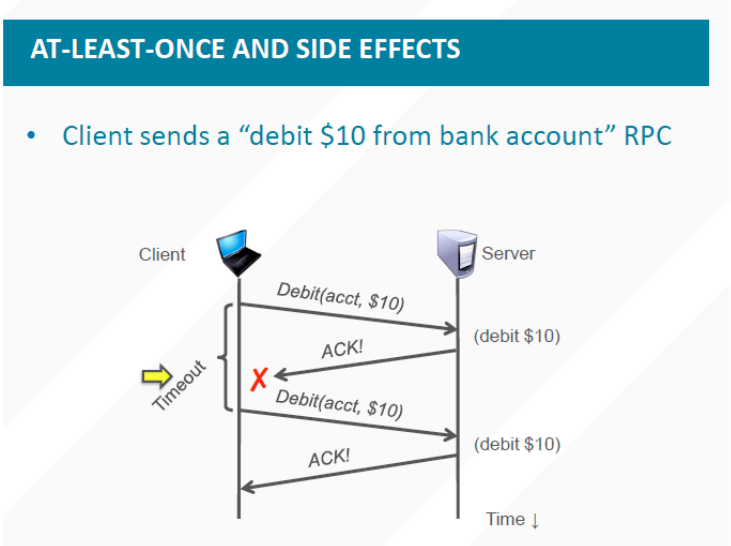
The latter three look the same to the client

It is hidden from the client.

Solution1: AT LEAST ONCE SCHEME

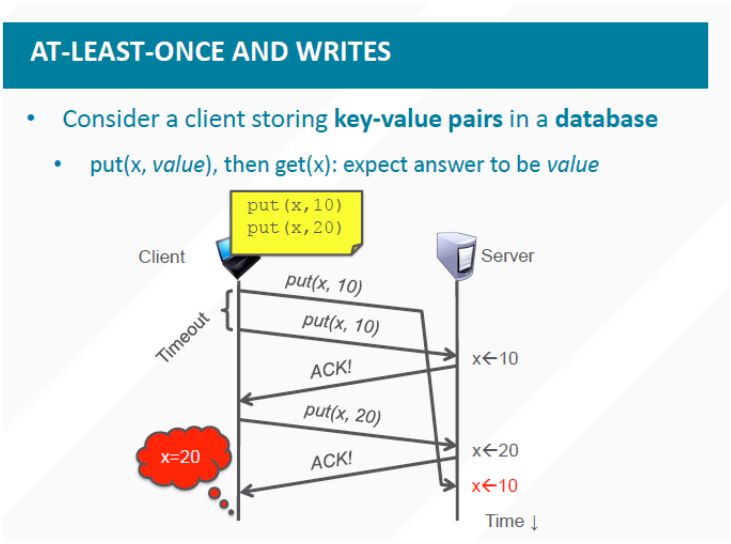
simplest idea for handling failures

1. Client stubs wait for a response(An acknowledge message from the server stub)
2. If no response arrives after a fixed timeout, the client stub resends the request.
3. If repeat the request for several times and still no response, return an error to the application



AT_LEAST_ONCE scheme is okay if there are only read-only operations with no side effects.

If the application has its own functionality to cope with duplication and reordering.



Solution2: AT-MOST-ONCE

Idea: Server RPC code detects duplicate requests

Returns previous reply instead of re running handler

How to detect a duplicate request?

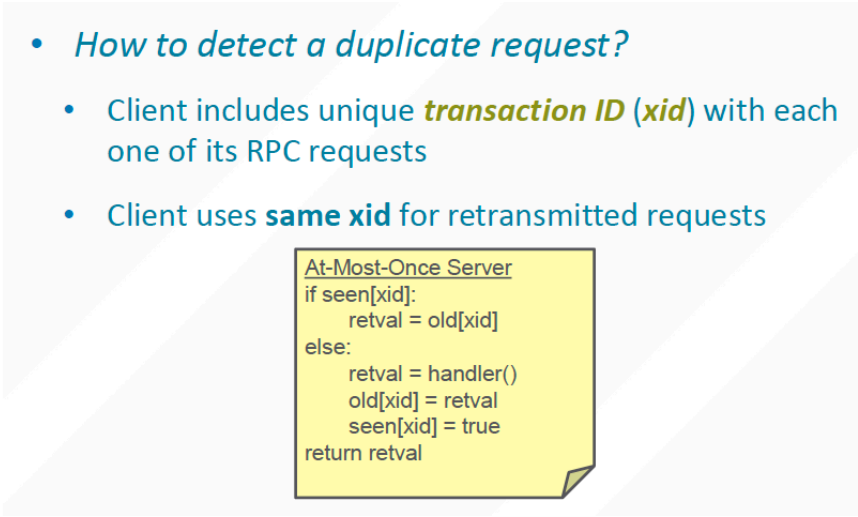
Server sees same function, same arguments twice ==> Is it a duplicate request?

No!!! Some applications legitimately submit the same function with same arguments, twice in a row.

Solution

Client includes a unique transaction ID(xid) with each of its RPC requests

Uses same xid for retransmitted requests



How to ensure xid unique?

1. Combine a unique client ID(IP address) with the current time of the day
2. Combine unique client ID with a sequence number
3. Big random number

Problem with AT-MOST-ONCE

seen and old arrays will grow without bound

Observation

By construction, when the client gets a response to a particular xid , it will never re-send it

Client should tell server "I'm done with xid x --delete it"

Have to tell server each and every retired xid -- paggyback

Significant overhead if many RPCs are in flight

Solution

Client includes “seen all replies ≤ X ” with every RPC

Each one of these is cumulative: later seen messages subsume earlier ones

Problem

How to handle a duplicate request while the original is still executing?

Idea

Add a pending flag per executing RPC Server waits for the procedure to finish, or ignores

Problem

Server may crash and restart

Solution

Write seen[], old[] to disk

Summary

RPC SEMANTICS

Delivery Guarantees			RPC Call Semantics
Retry Request	Duplicate Filtering	Retransmit Response	
No	NA	NA	*Maybe*
Yes	No	Re-execute Procedure	*At-least once*
Yes	Yes	Retransmit reply	*At-most once*

- RPC everywhere!
- Necessary issues surrounding machine heterogeneity
- Subtle issues around handling failures