direct communication

data representation

problem: heterogeneity in data structures between different systems

conversion approaches:

1. agreed form for transmission (implicit information)

   e.g. CORBA CDR, Sub XDR
2. full data description transmitted

   e.g. java serialized form

   primitive types: portable binary format
3. conversion to ASCII text

   e.g. XML

   markup language, data items tagged with "markup" strings

```
<person id=‘123456789”>
        <name>Smith</name>
        <place>London</place>
        <year>1934</year>
        <...>
</person>
```

marshalling - unmarshalling:

marshalling = data -> transmission form

unmarshalling = transmission form -> data items

message passing

- message destinations:

  process, port, mailbox

- reliability:

  may only allow a reasonable number of packets dropped / lost

  implementation: 多传/少传/乱传/错传

  1. avoid corruption (use checksum)
  2. avoid order mistakes / duplicates (use message identifier)
  3. avoid omission (use timer, acknowledgment, re-transmission)

- UDP (user datagram protocol, 用户数据报协议)

  相比于TCP，轻量、快速、不可靠

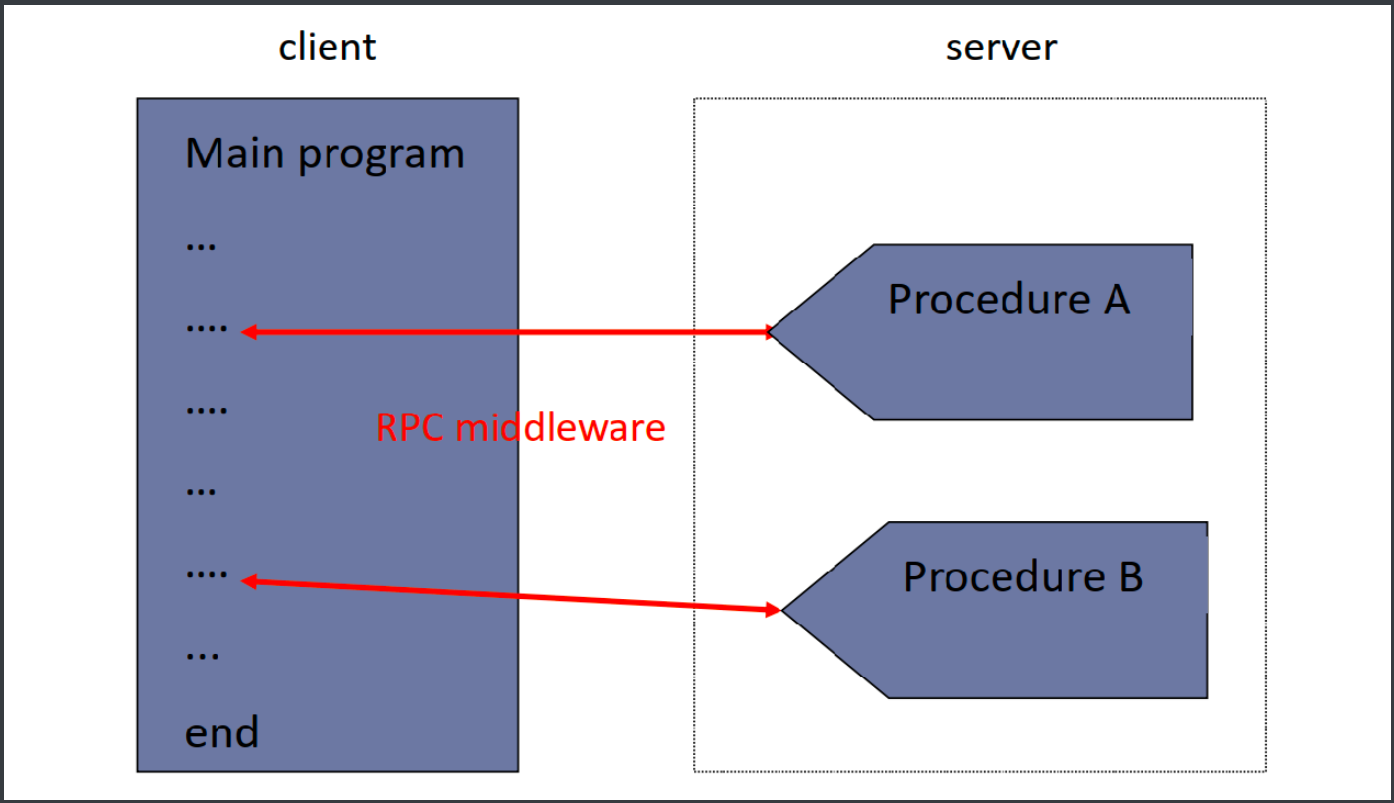
attribute	details
消息格式	restricted packet size, 超出截断，按bit传输无转换
同步	非同步发送，同步接收， 接收方可设超时限制
receive from any	可以接收来自任何源（IP地址+端口）的消息，也可以绑定到指定源
不可靠交付	只用checksum

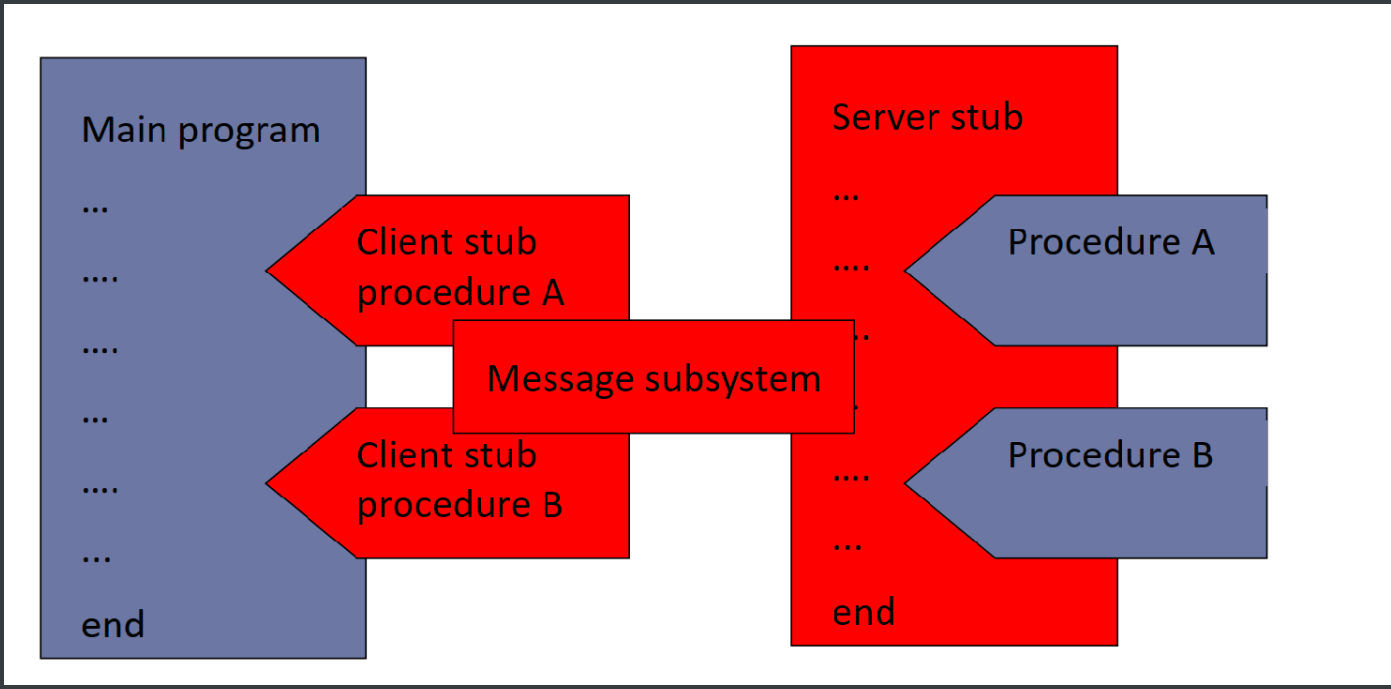
- TCP

  communication channel (sockets), 两端都有消息buffer，非同步发送同步接收（存在流量控制的时候有可能sender也block），可靠交付

request-reply protocols

remote procedure call





design issues

- classes of RPC systems

  rpc integrated within a particular programming language

  or

  rpc based on a special IDL

- interface definition language (IDL)

  describes operation signatures, stubs generated by compiler

  abstraction of heterogeneity

- exception handling

  client cannot distinguish network/server failure

  could use return codes etc.

- semantics of RPC

Delivery guarantees			RPC semantics
retry request	duplicate filtering	re-exec retrans reply	
no	not applic.	not.	Maybe
yes	no	re-exec	At-least-once
yes	yes	retrans reply	At-most-once

  retry request: 重试请求

  duplicate filtering：重复过滤（避免重试导致的副作用）

  re-exec / retransmission reply：调用失败时是否重新执行远程过程，或者简单地重新发送之前的回复

  Summary:

  maybe 和 **at-least-once** 适合快速、宽松的容错需求；

  at-most-once: 避免重复执行，适合需要防止副作用的场景；

  exactly-once: 经典难题，在有故障的分布式系统中几乎不可能；

- transparency

  possibility of failure should not be hidden

implementation aspects

- interface compiler:

  generate stubs, perform as a header for server procedure

- binding: linking client to server at execution time

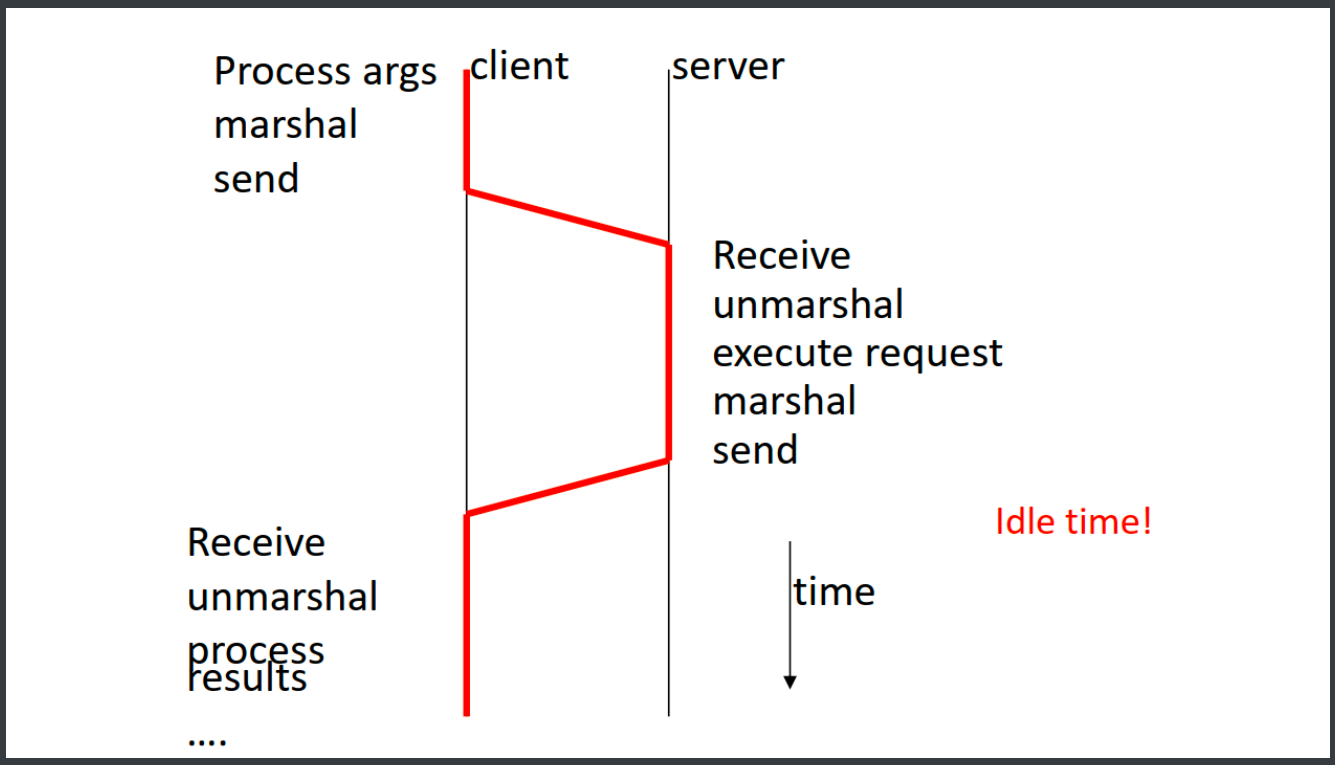
  server registers service at binder, clients lookup the service

  how to lookup?

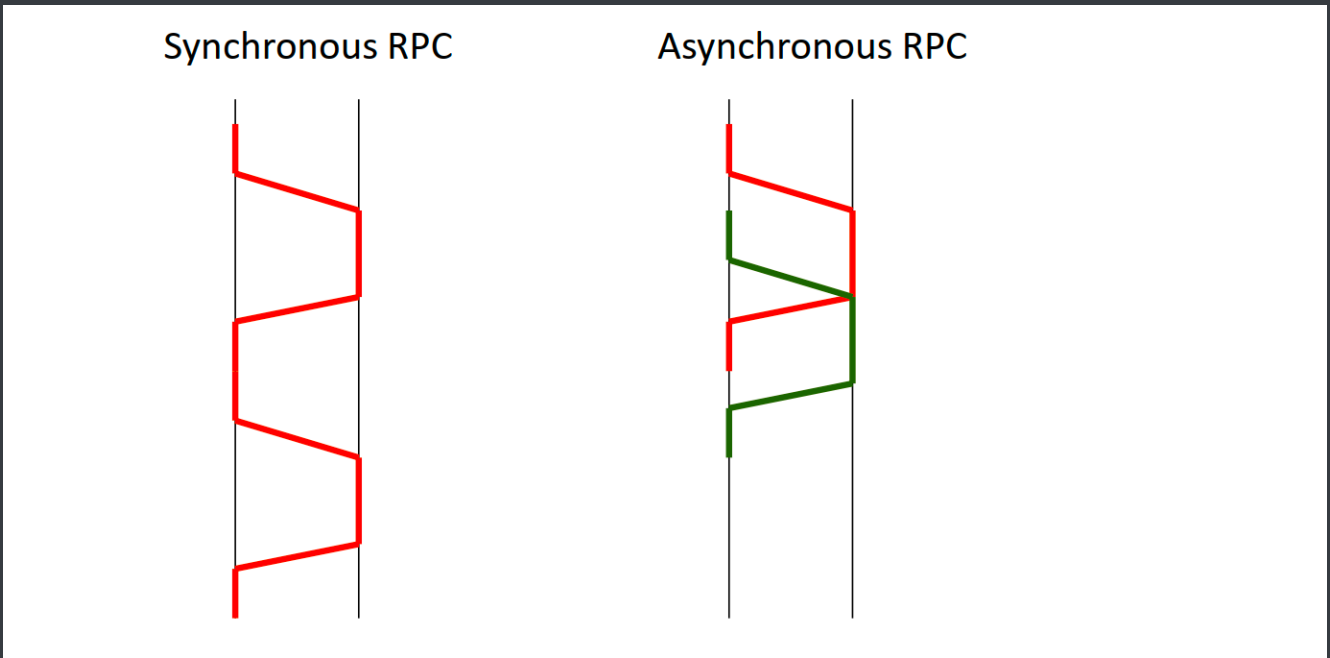
  well known host address, OS, or client broadcasting message

- variant: **asynchronous RPC**

  motivation:



  therefore,



scenario 1: many requests + small information + limited processing

seenario 2: parallel requests to several servers

extensions:

call streams 调用流

同时支持同步和异步调用，保留消息顺序，面向连接的，语义无法保证时断开连接

promise机制 （alias: future, ticket, continuation）

1. promise允许client执行其他任务，结果稍后通过promise检索
2. promise在发出异步调用时创建，充当一个占位符
3. 存储RPC调用的结果
4. 支持client通过特定借口获取结果，或检查结果是否可用

object request brokers 对象请求代理

- ORB可以被看作是基于RPC的一种更高级的、面向对象的扩展
- examples:

  CORBA (Common Object Request Broker Architecture)：
  - 最著名的ORB实现，由OMG（Object Management Group）定义。
  - CORBA标准化了分布式对象调用的协议和IDL语法。

  Java RMI (Remote Method Invocation)：

- Java语言中的一个类似ORB的机制，专注于Java对象之间的远程调用。

Microsoft DCOM (Distributed Component Object Model)：

- 微软的分布式对象模型实现，与ORB概念类似，但偏向于Windows生态。

.NET remoting

distributed object systems

- requirements
  1. synchronous invocation semantics

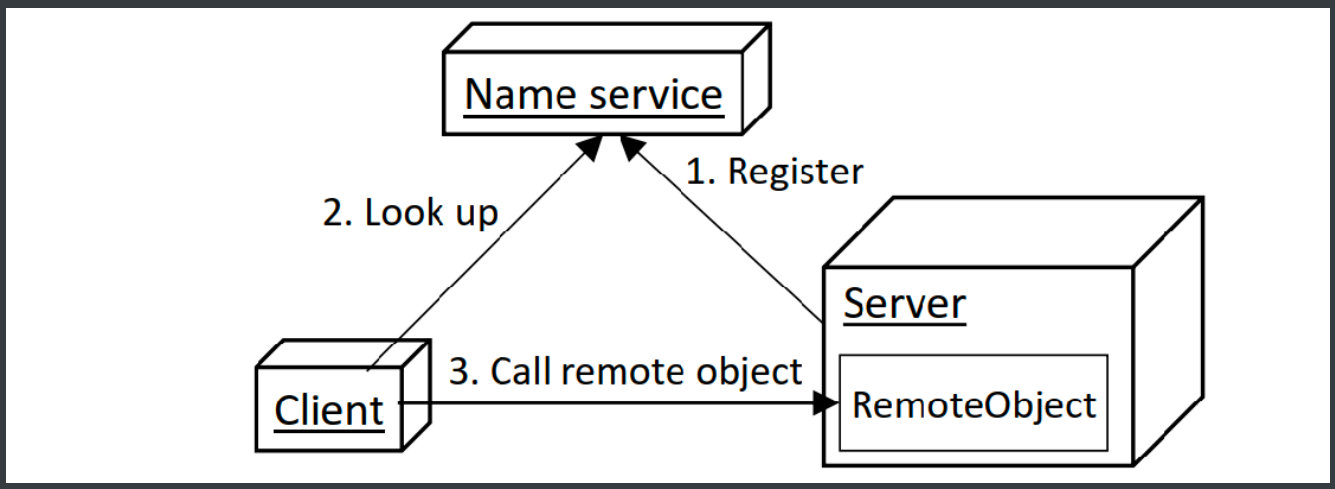
     location transparency: 本地对象和远程对象看起来相同

     access transparency: 本地和远程调用看起来相同
  2. inheritance
  3. polymorphism
- distirbuted object

  non-transparent aspects:
  1. invocation semantics, similar to RPC
  2. parameter passing:

     distributed objects are passed by reference,

     normal objects are passed by value
  3. distribution, concurrent access, latency
- typical architecture



name service: name => address (recall how references work)

- garbage collection

  JAVA的分布式垃圾回收方法 1：**reference count**

  如果reference count = 0说明没有任何client在使用这个对象

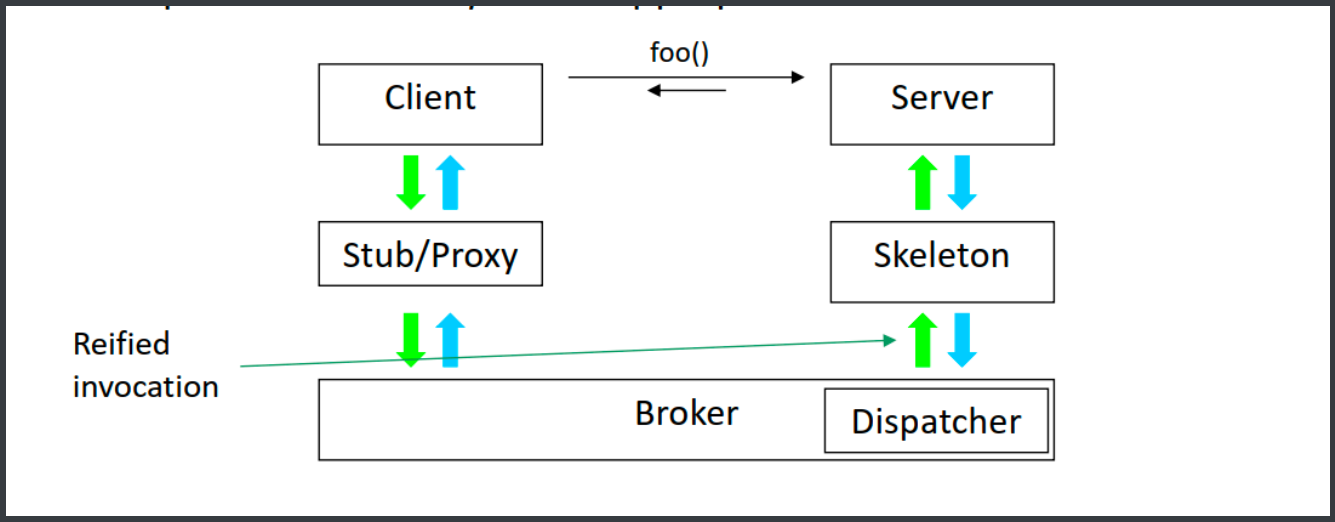
  但有问题：两个对象相互引用，但都没有被其他对象使用

  JAVA的分布式垃圾回收方法 2：**leasing-based**

  client obtains lease 租约 for a period of time

  容错：防止client crash / communication crash导致对象无法回收

- the object bus 对象总线



引用层的foo()是概念上的调用，object bus上发生实际调用；

case study: gRPC