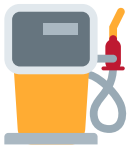


4.分布式通信管理

1.分层通信协议

开放式系统：可按照协议进行通信

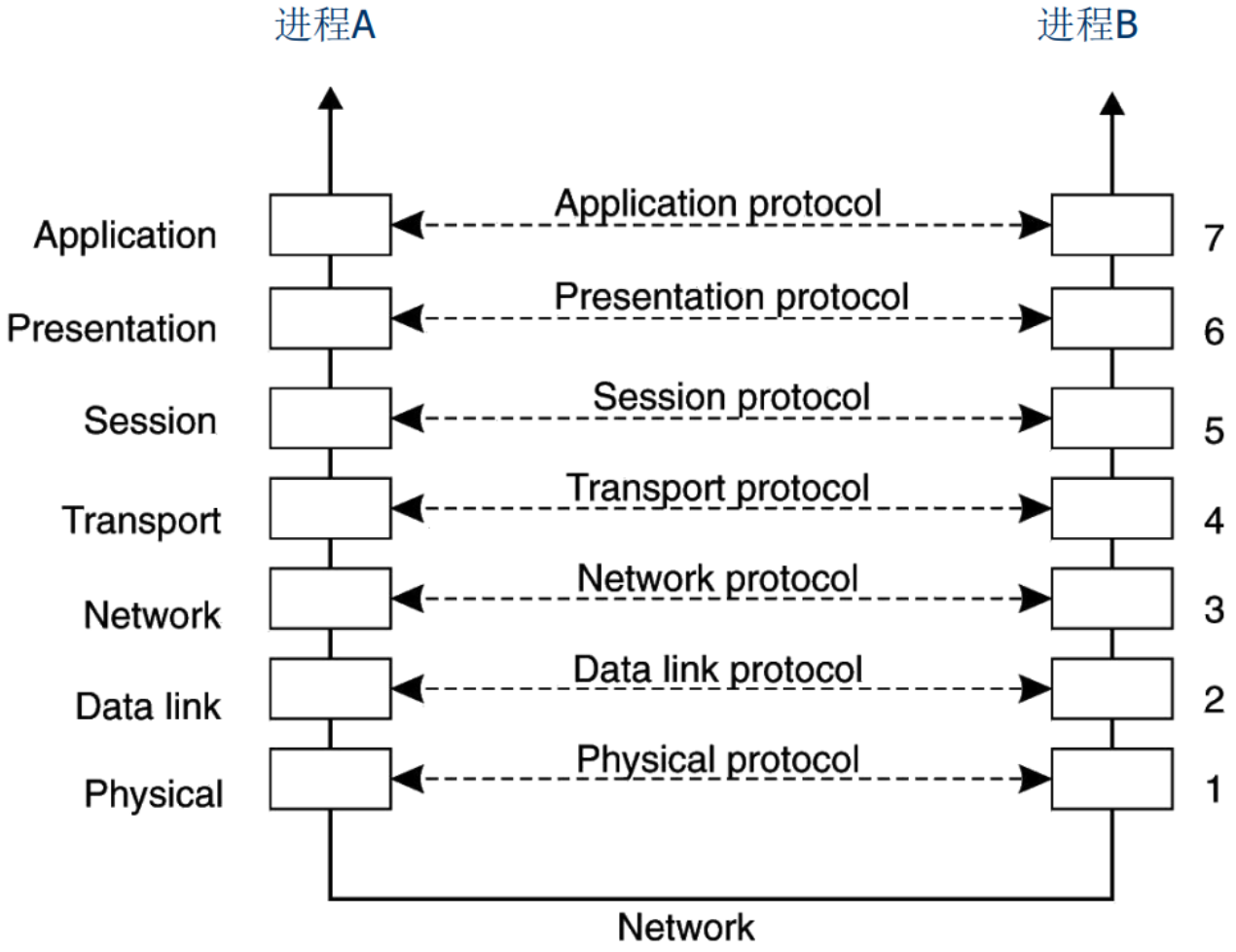
协议：消息的格式、内容和意义

协议的种类：

- 有连接（connection）：如，电话
- 无连接（connectionless）：如，短信

ISO OSI 参考模型分层协议（1983）

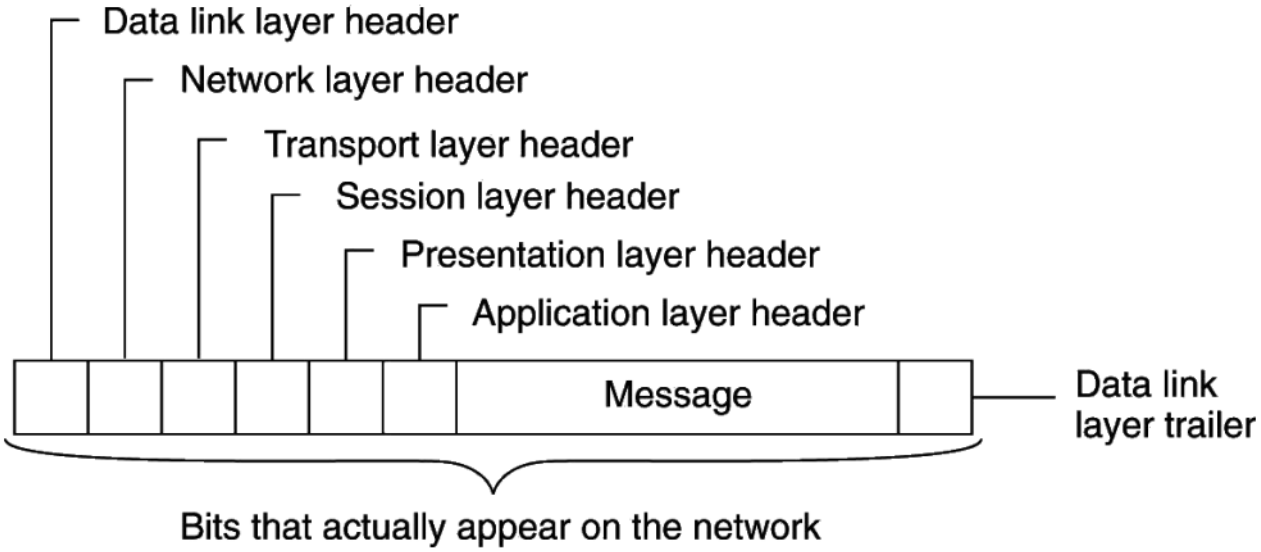
1.1 IOS OSI 参考模型分层协议



ISO OSI 参考模型的特点

- 分层：功能分解。独立性
- 接口（提供功能的操作集）：标准化
- 协议栈：有序性

报文（message）格式：



1.2 基本网络协议

物理层：

- 位（bit），字节（byte）
- 电气的、机械的、信号接口
- 例：RJ 双绞线、USB

数据链路层：

- 帧（frame）
- 帧头、帧尾
- 检错（如，checksum）

网络层：

- IP 包（IP packet）
- 网址
- 路由（routing）
- 无连接 IP 协议（最广泛的网络协议）

传输层

- 可靠的通信（不丢失，保证报序）
- TCP 协议：有连接的
- UDP 协议：无连接的

1.3 高层网络协议

会话层：

- 会话（session）控制
- 同步控制：如，检查点（checkpoint）

表示层：

- 数据的格式和含义：如，记录定义

应用层：

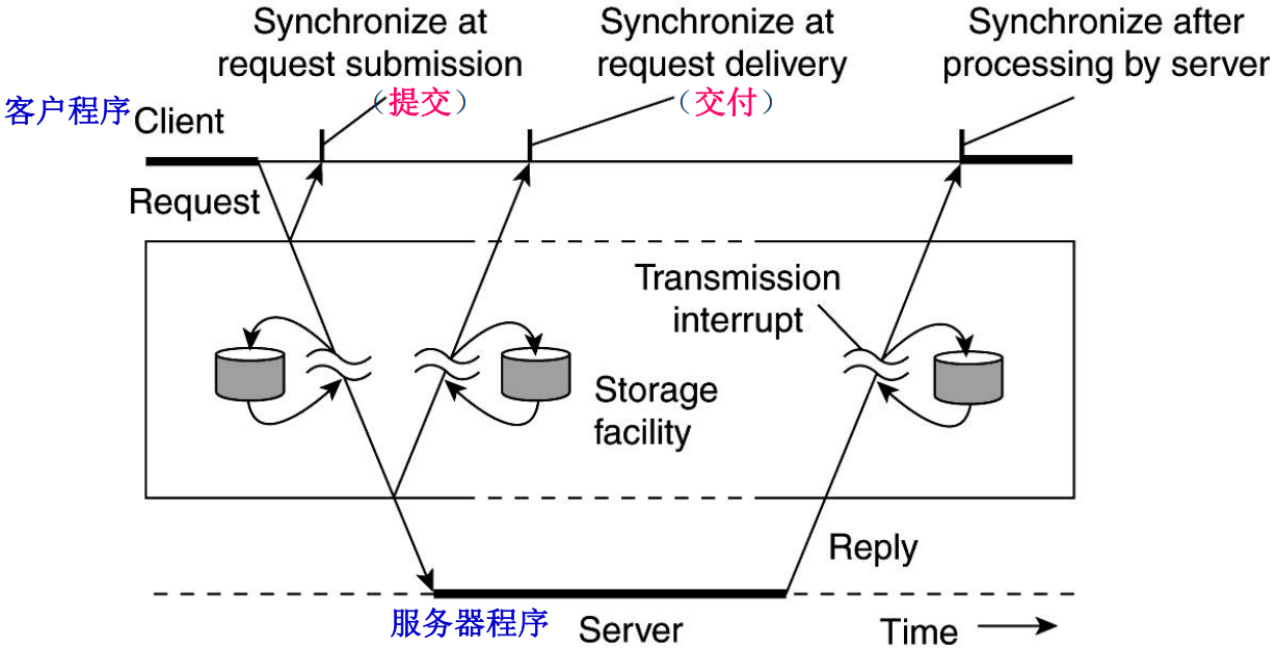
- 所有应用：电子邮件，文件传输，远程终端等
- 应用协议：具体应用协议；通用协议：FTP、HTTP

1.4 中间件协议

中间件层（Middleware）

- 独立于应用的协议，支持中间件服务
- 通信协议（RPC、RMI、CORBA、MPI）
- 安全性协议（认证、授权）
- 分布式事务处理协议（提交、封锁）

电子邮件：



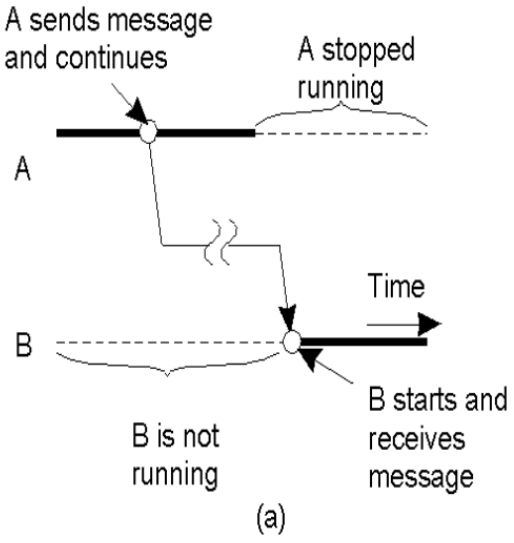
1.5 通信类型

保存方式

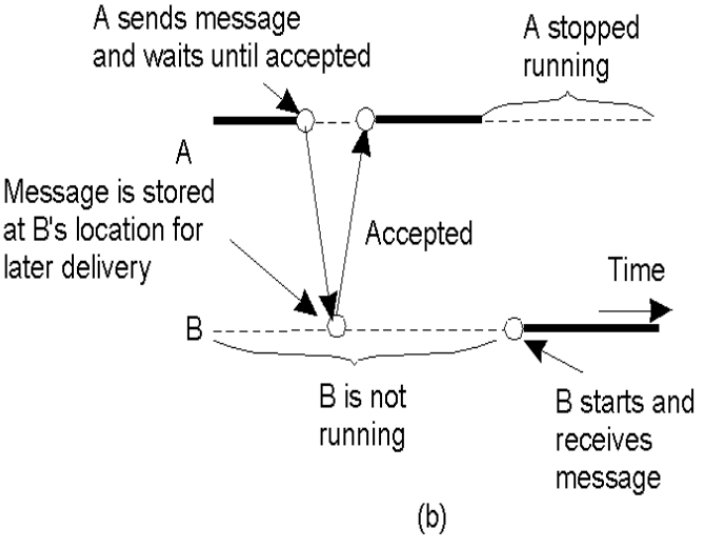
- 持久性通信（persistent）：发送方提交的消息一直被持久地存储，直到被交付给接收方
- 瞬时性通信（transient）：当传输中断或接收方不活动时，则丢弃消息

交互方式

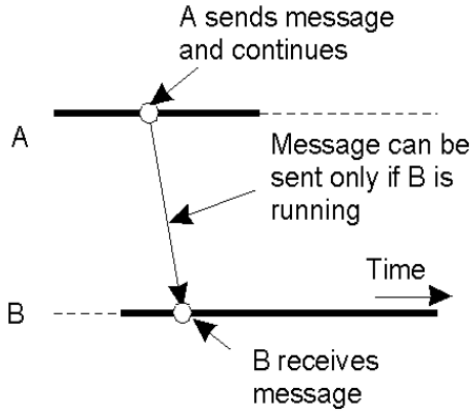
- 异步性同行（asynchronous）：发送方提交消息后，可立即执行
- 同步性同行（synchronous）：发送方被阻塞，直到得知其消息被接收后，方可继续执行



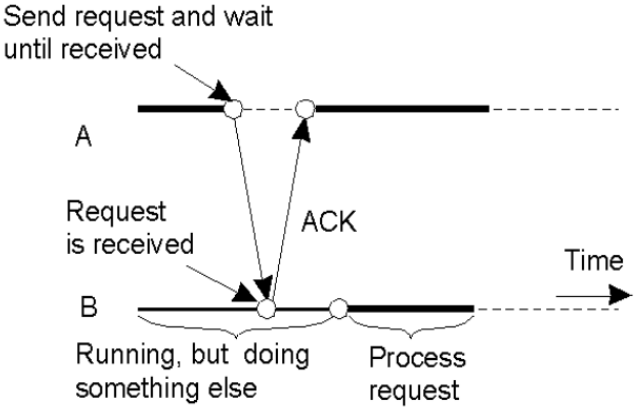
（a) 持久异步通信



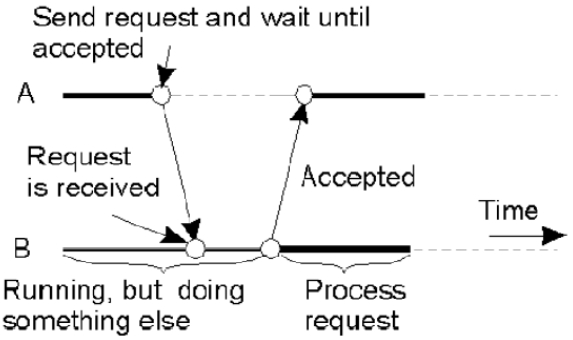
(b)持久同步通信



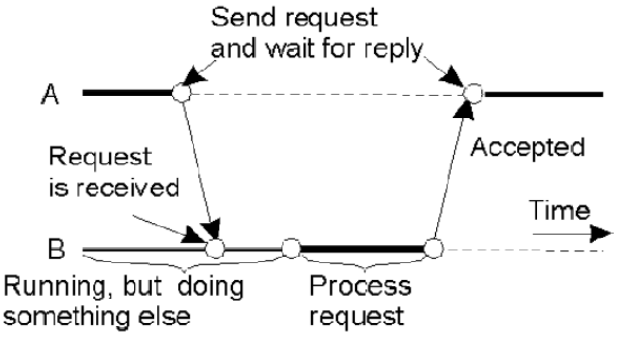
（c) 瞬时异步通信



(d) 基于接受的瞬时同步通信



（e)基于交付的
瞬时同步通信



(f)基于响应的
瞬时同步通信

2.远程过程调用（RPC）

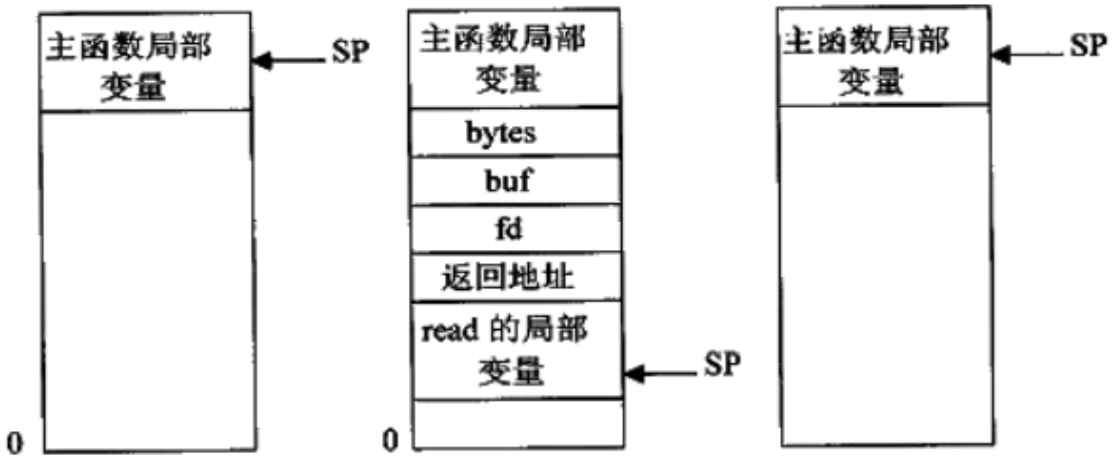
像本地子程序一样，调用远地子程序；调用者和被调用者都不用考虑消息通信

2.1 过程调用中的参数传递

```c
int read(int fd, char* buf, int nbytes);
count = read(fd, buf, nbytes);
```

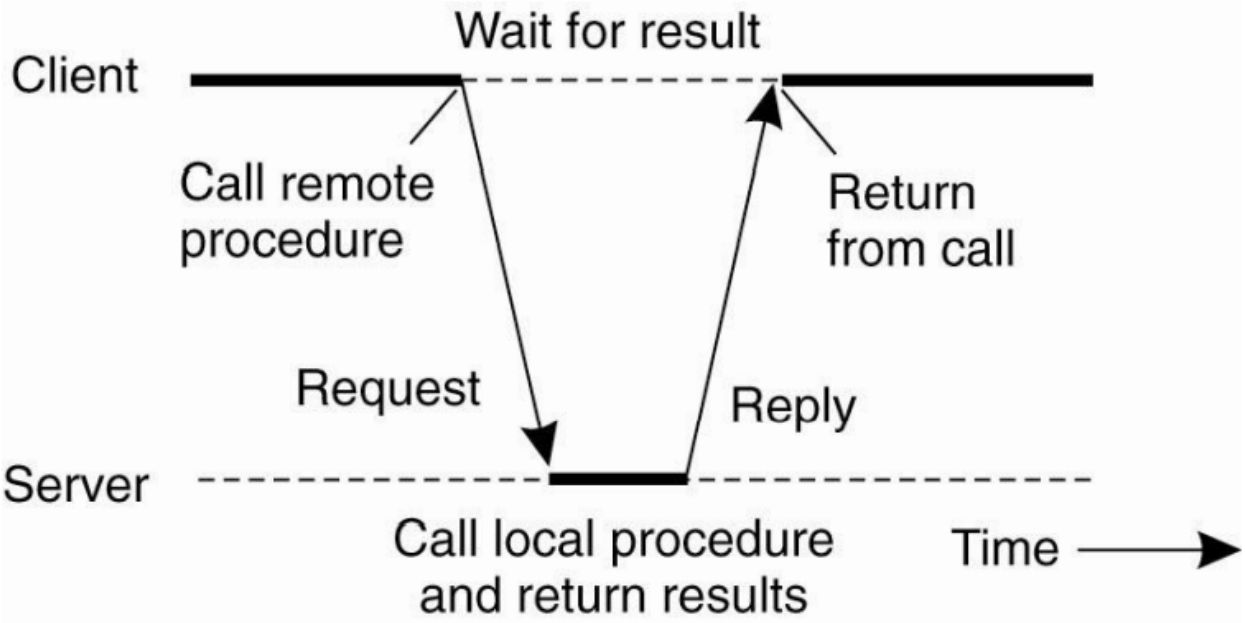
参数传递：

- call-by-value ： 例：fd, nbytes
- call-by-reference：例：*buf



栈的状态(a) 调用前　(b) 调用中　(c)调用后

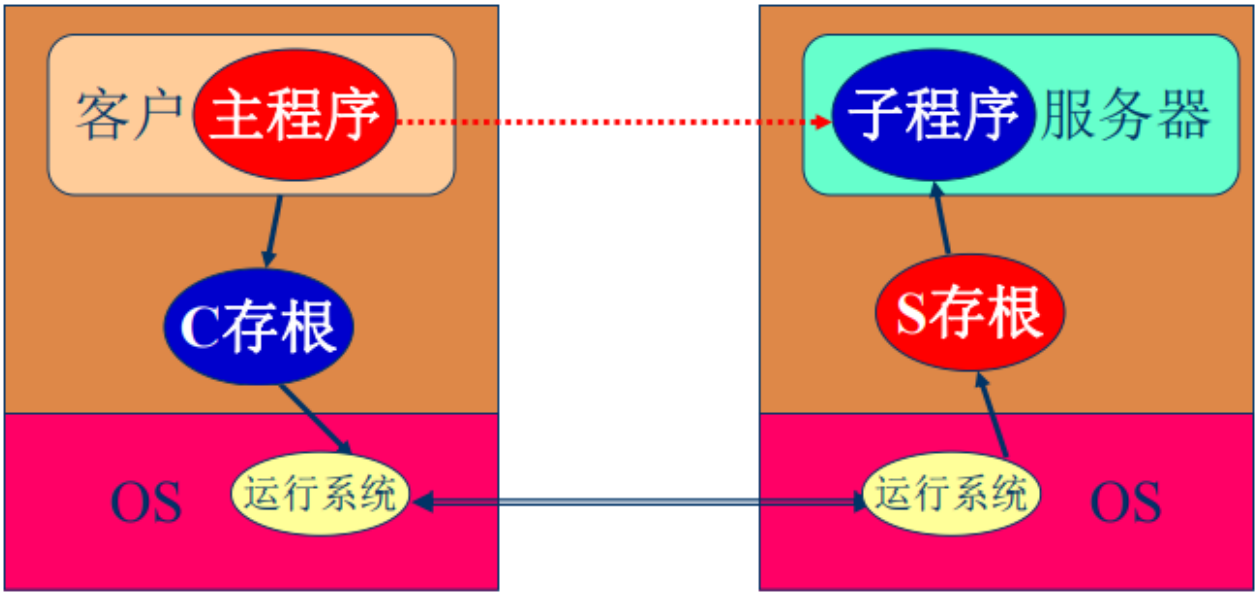
2.2 RPC 调用原理



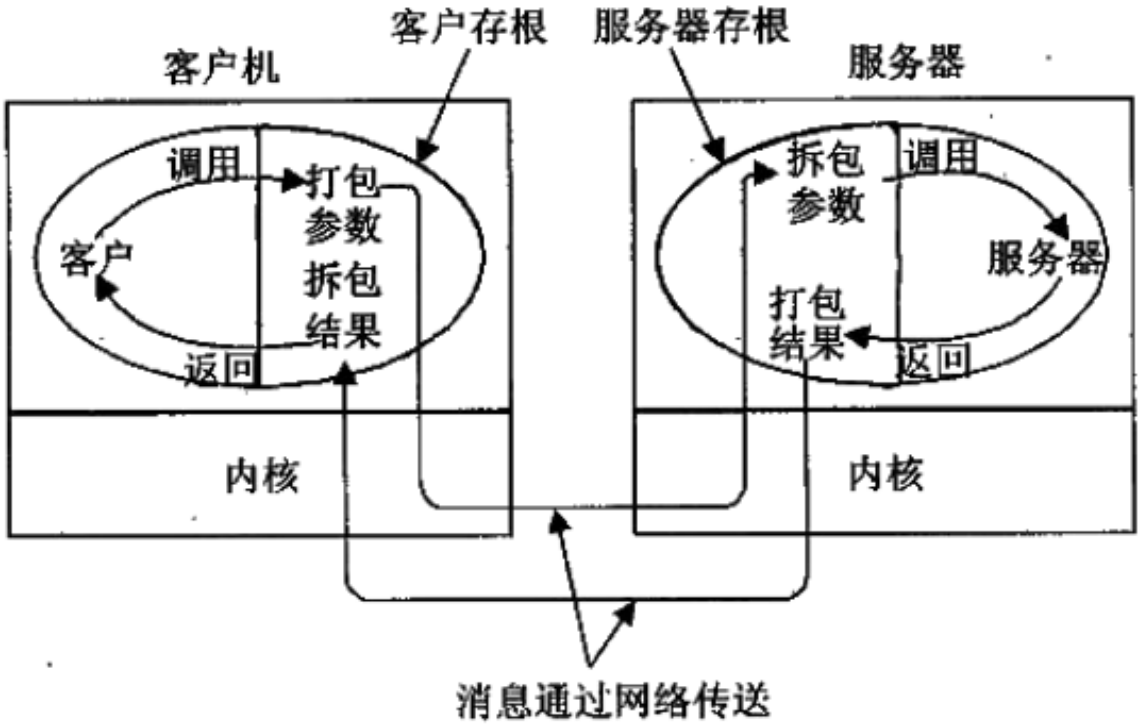
2.3 基本 RPC 操作

客户存根程序（client stub）：主程序端的子程序代理

服务器存根程序：子程序端的主程序代理



2.4 RPC 执行的主要步骤



1. 客户过程以普通方式调用相应的客户存根
2. 客户存根建立消息并激活内核陷阱
3. 内核将消息发送到远程内核
4. 远程内核将消息送到服务器存根
5. 服务器存根取出消息中的参数后调用服务器的过程
6. 服务器完成工作后将结果返回值服务器存根
7. 服务器存根将它打包并激活内核陷阱
8. 远程内核将消息发送至客户端内核
9. 客户端内核将消息交给客户存根
10. 客户存根从消息中取出结果返回给客户

效果：讲客户过程对客户存根发出的本地调用转换成对服务器过程的本地调用，而客户和服务器都不会意识到有中间步骤的存在。

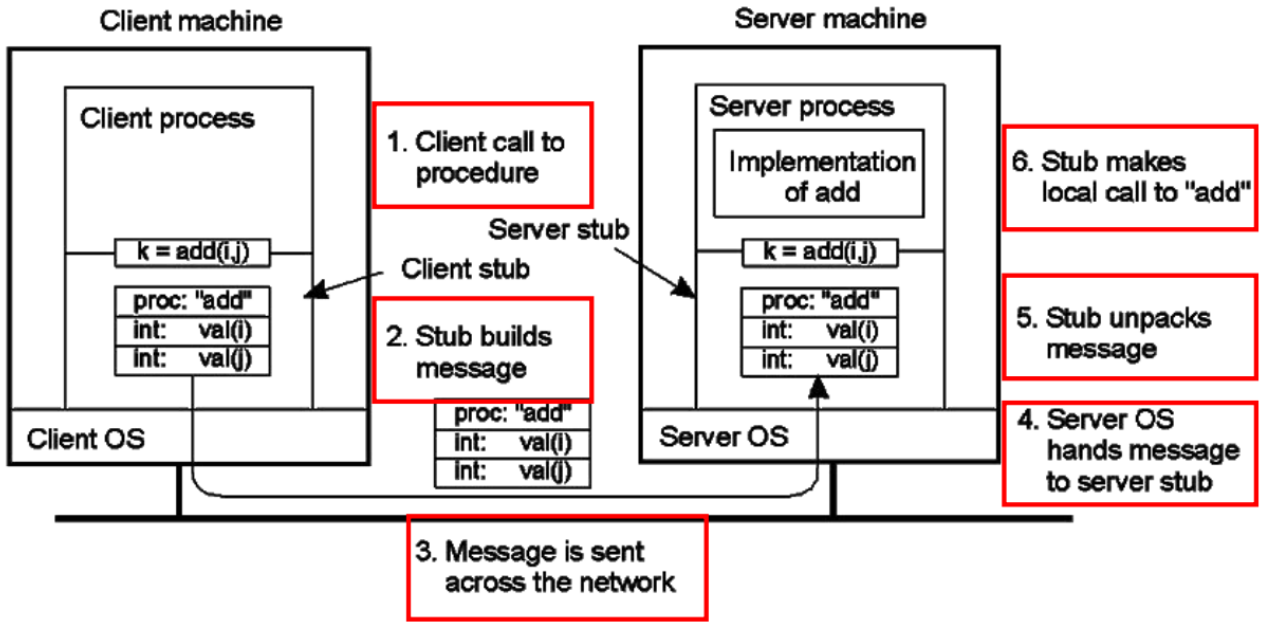
2.5 参数传递

参数编排（marshalling）：将参数装入消息

（1）远程过程

参数编排（marshalling）：将参数装入消息

例：远程过程 `add(i,j)`

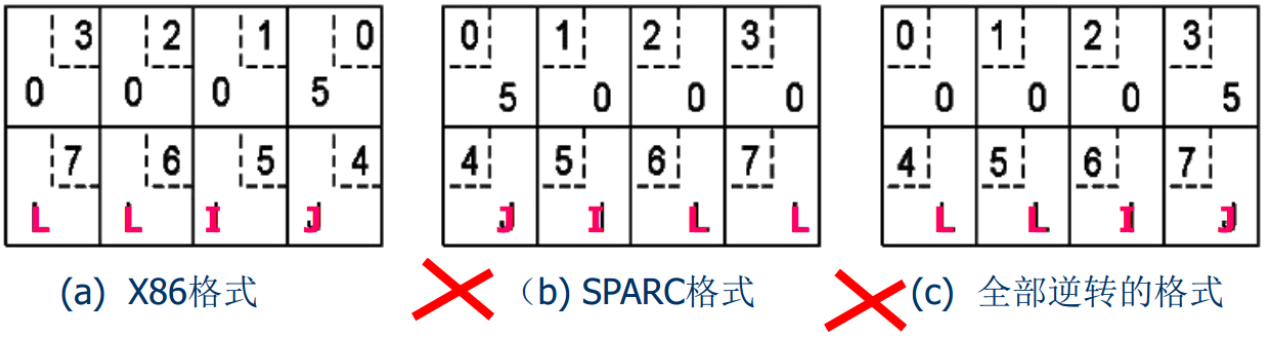


（2）参数传递问题

值的传递

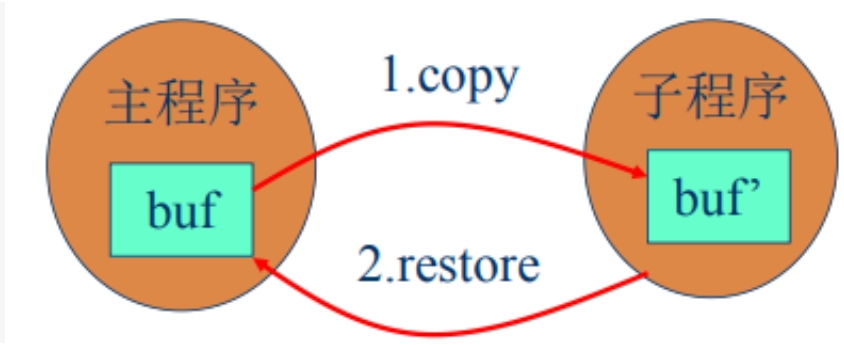
- 不同编码表示：ASCII/UNICODE
- 不同数字表示：小末端/大末端，补码/反码

例： x0005,'JILL'; x5000,'JILL'; x0005,'LLIJ'



指针（引用）的传递

- 禁止传递指针/引用
- 策略 1：call-by-copy/restore（复制-还原）

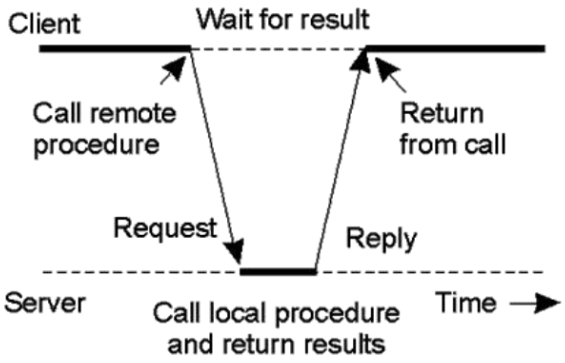


- 策略 2：请求被引用的数据

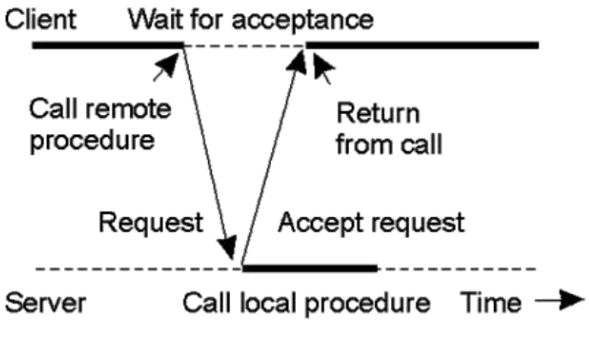
2.6 扩展 RPC 模型

（1）传统 RPC

（2）异步 RPC



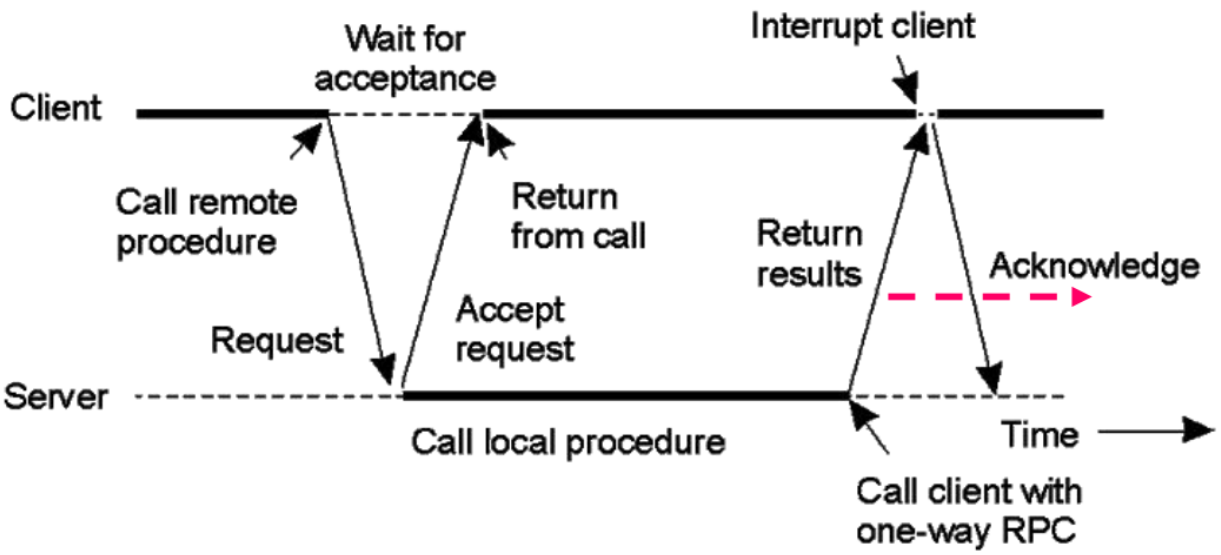
(a)传统的RPC



(b)异步RPC

（3）双异步 RPC

延期的同步 RPC(deferred synchronous RPC)



（4）单程 RPC

客户发出请求后， 不等待服务器应答， 立即继续执行

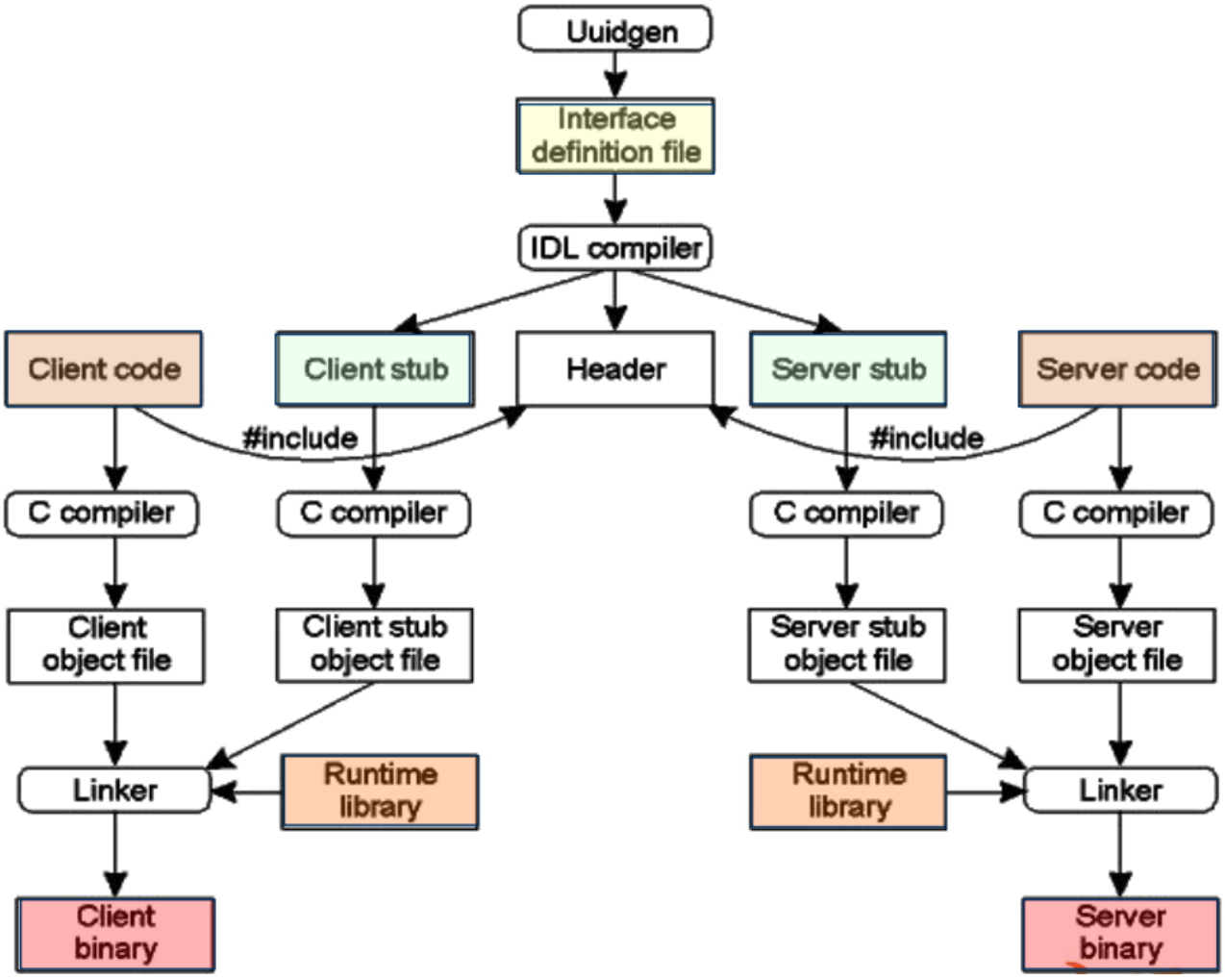
2.7 示例 DCE RPC

DCE(Distributed Computing Environment)

（1）DCE RPC 目标

- 客户透明地调用远程服务
- 支持客户与服务器的独立性
    - 不同的语言： C vs. Java
    - 不同的平台： windows, Unix
    - 不同的硬件

（2）DCE 客户/服务器构造



（3）客户/服务器程序编写

IDL 编译器输出

头文件 ： 如 c 语言形式的 interface.h

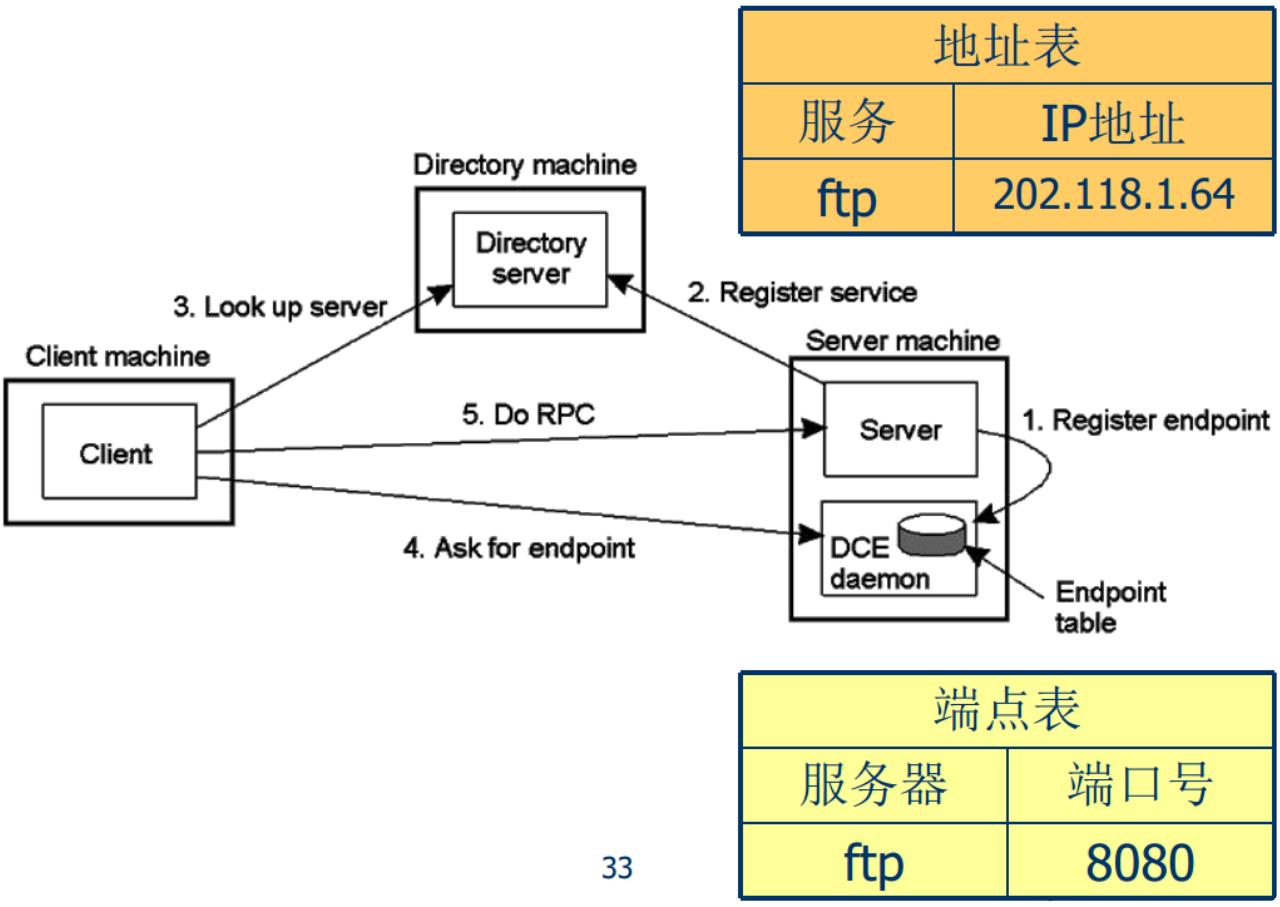
客户存根程序：由客户程序调用的过程

- 参数收集、 打包、 调用运行系统发送
- 提取应答、 返回给客户

服务器存根程序：由运行系统调用的过程

- 调用服务器过程以完成实际操作。

（4）客户进程于服务器进程的绑定



（5）执行过程

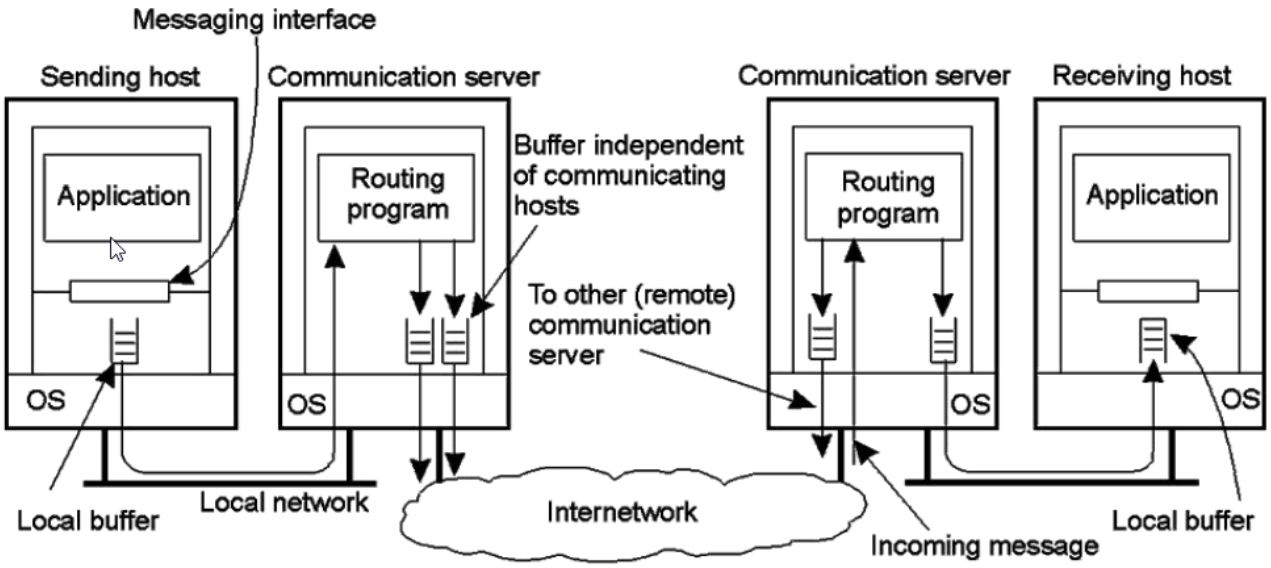
1. client 存根编排参数
2. client 机运行库按照绑定的协议， 向 server 机发送消息
3. server 机运行库按照消息中指定的“端点”， 转交消息给 server 存根
4. server 存根解编参数， 调用 server 程序
5. 应答过程与上面相反

（6）执行策略

最多一次（at-most-once） ： 不重复执行

幂等性（Idempotent） ： 可重复执行， 效果相同

3.面向消息的通信



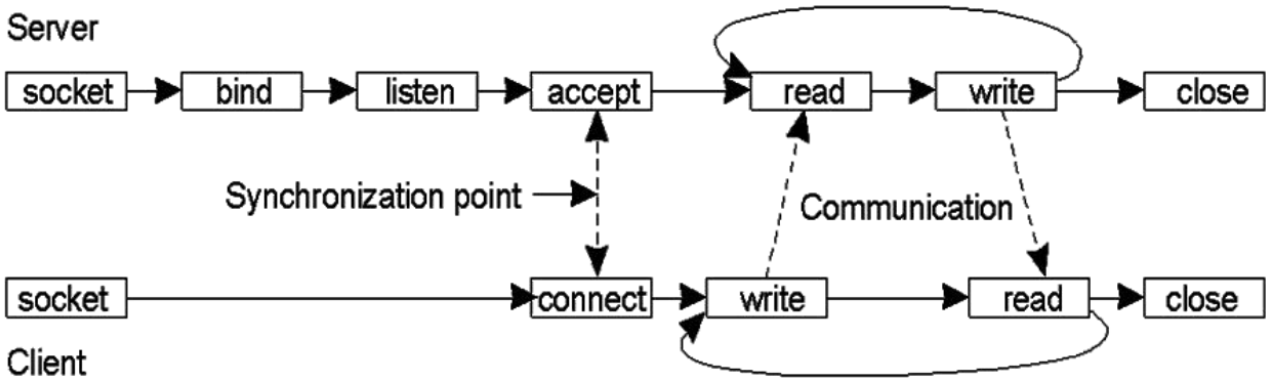
通信系统的一般结构

3.1 面向消息的瞬时通信

支持 TCP/IP 的 Berkeley Sockets 原语

原语	意义	
Socket	创建新的通信端点	客户、服务器调用
Bind	将本地地址附加(attach)到套接字上	服务器调用
Listen	宣布已准备好接受连接	
Accept	在准备好连接请求之前阻塞调用方	
Connect	主动尝试确立连接	客户调用
Send	通过连接发送数据	客户、服务器调用
Receive	通过连接接受数据	
Close	释放连接	

使用 Socket 的面向连接的通信模式



3.2 消息传递接口（MPI）

（1）特点

- 提供高层通信，支持并行计算，如大规模并行处理器 MPP，工作站集群 COW
- 可靠的底层网络
- 支持分组通信，地址（groupID， processID）
- 支持多种通信方式：
  - 瞬时异步 MPI_bsend
  - 基于收条的瞬时同步 MPI_send
  - 基于交付的瞬时同步 MPI_ssend
  - 基于响应的瞬时同步 MPI_sendrecv

（2）MPI 原语

原语	意义
MPI_bsend	将要送出的消息追加到本地发送缓冲区中
MPI_send	发送消息，并等待直到消息复制到本地或远程缓冲区中为止
MPI_ssend	发送消息，并等待直到对方开始接受为止
MPI_sendrecv	发送消息，并等待直到收到应答消息为止
MPI_isend	传送要送出消息的引用，随后继续执行
MPI_issend	传送要送出消息的引用，并等待直到对方开始接受为止
MPI_recv	接受消息，如果不存在等待的消息则阻塞
MPI_irecv	检查是否有输入的消息，但是无论有没有消息都不会阻塞

3.3 面向消息的持久性通信

消息队列系统（message queuing system）

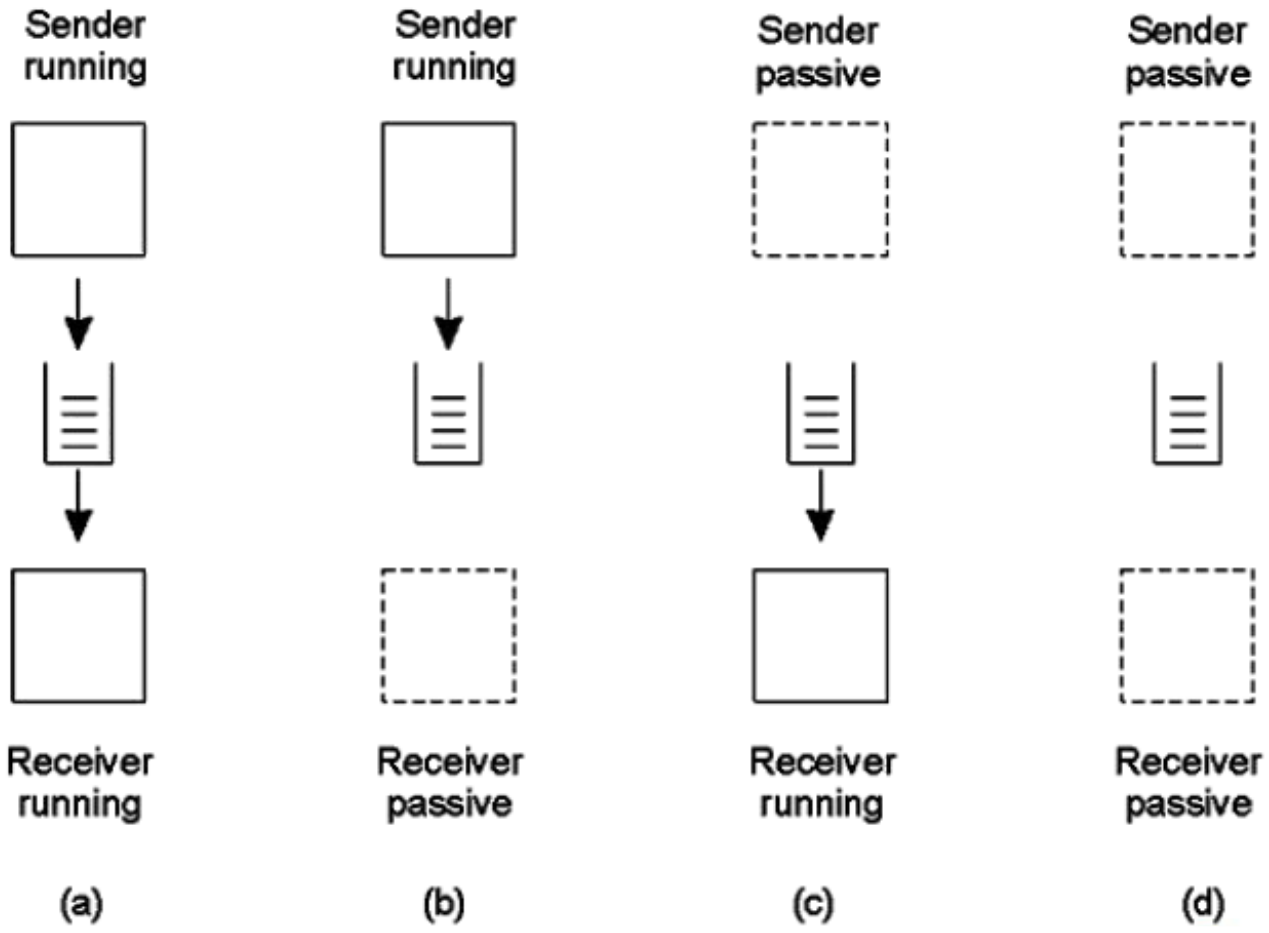
- 支持持久异步通信
- 适合于对传输时间要求宽松的场合，如几分钟。不保证消息到达接收方的时间，也不保证接受方一定读取消息
- 松散耦合的通信方式，使用方便

面向消息的中间件（MOM, message oriented middleware）

3.4 消息队列系统

（1）消息队列模型

使用消息队列的四种松散耦合通信



（2）消息队列系统基本接口

原语	意义
`Put`	将消息追加进指定队列
`Get`	调用进程阻塞，直到指定队列变为非空为止，然后取出队列中的第一个
`Poll`	查看指定队列中的消息，并且取出队列中的第一个消息。不阻塞调用进
`Notify`	注册一个处理程序，在有消息进入指定队列时调用该处理程序。

（3）消息队列系统的总体结构

队列分类：

- 源队列（source queue）：发送
- 目的队列（destination queue）：接受

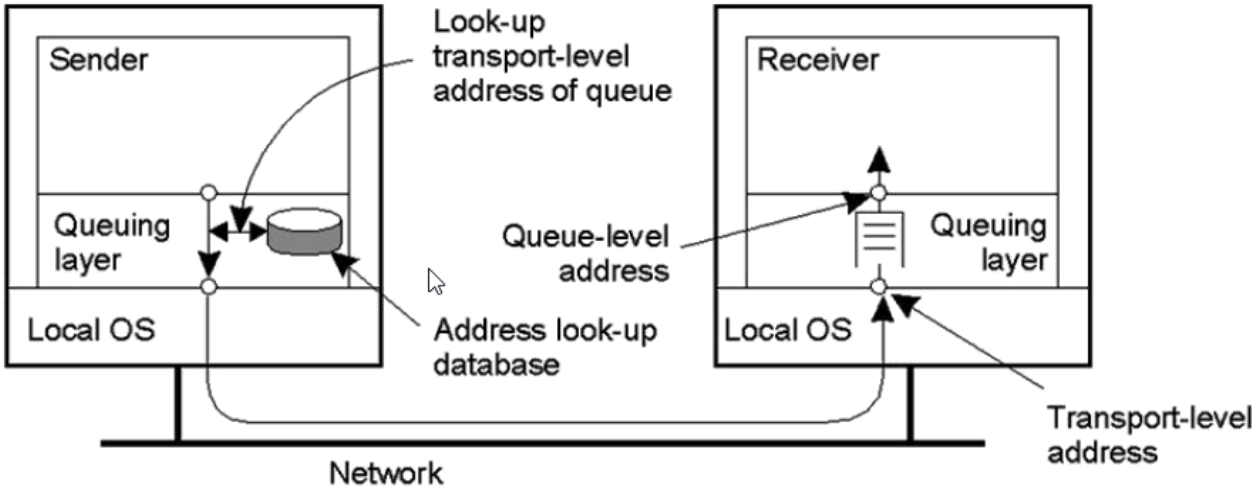
队列寻址

- 逻辑地址：队列名，如 steem@cs.vu.nl
- 网络地址：IP 地址

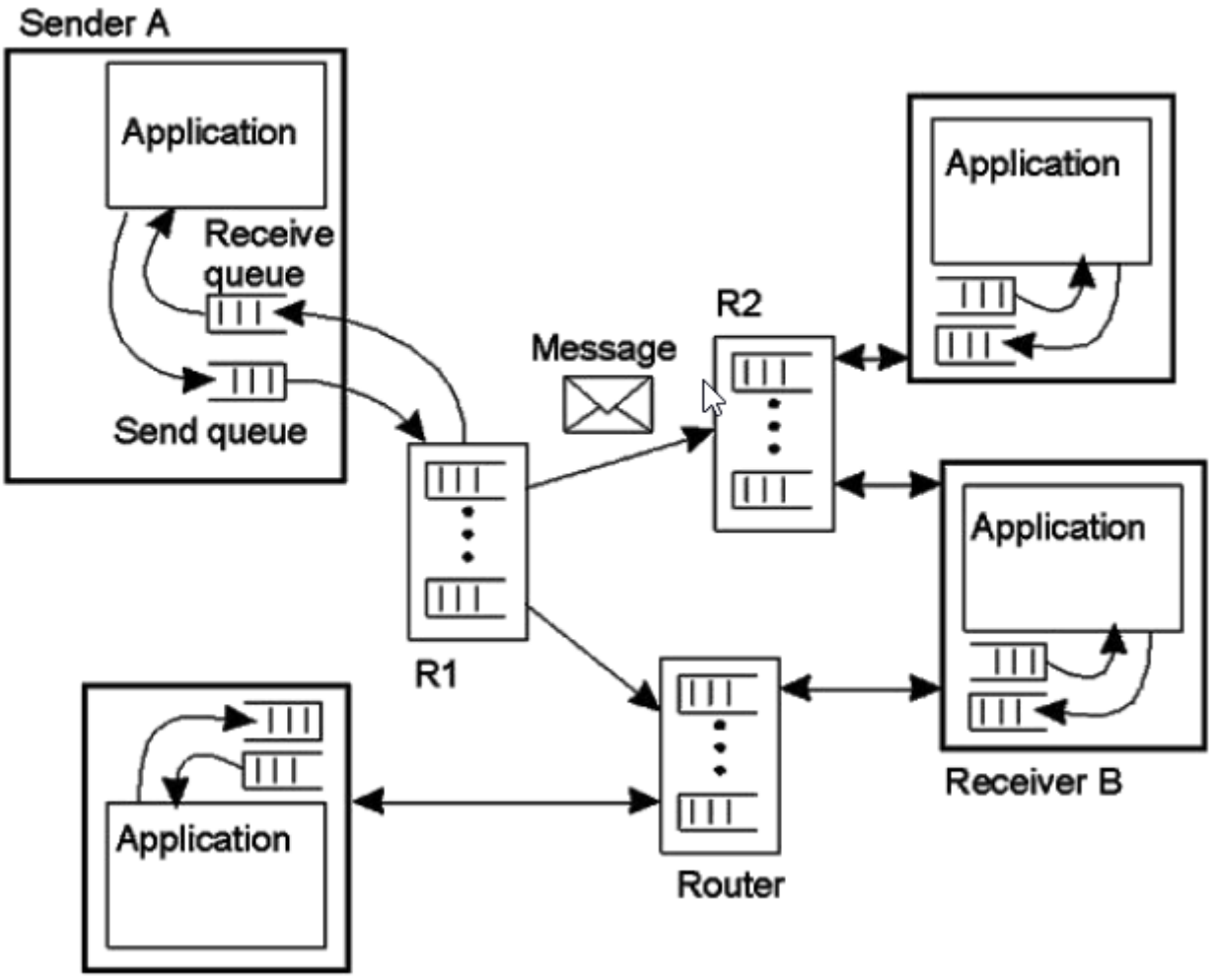
队列管理器

- 发送/接受
- 中转（relay）

队列层寻址和网络层寻址之间的联系



带有路由器的消息队列系统的结构



3.5 消息代理器（broker）

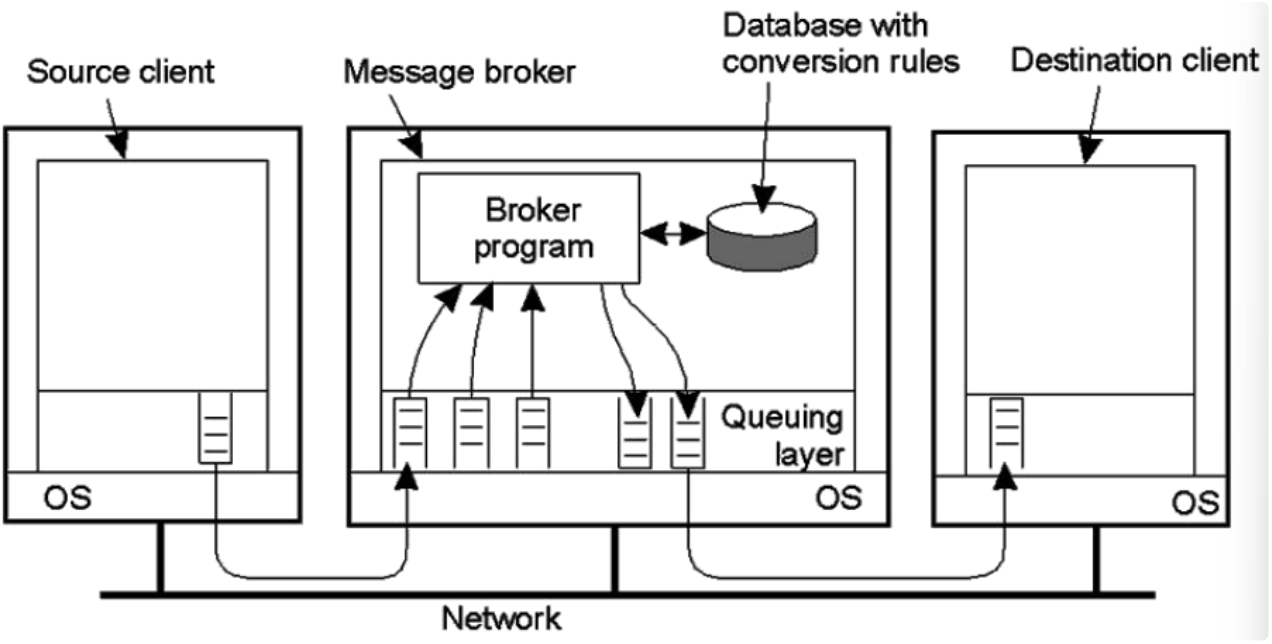
作用

- 不同消息系统之间格式转换
- 应用层网关（gateway）

实现

- 规则数据库
- 转换语言
- 规则开发工具

消息队列系统中消息代理器的通用结构



3.6 通道（Channel）

通道： 实现发送方和接收方 QM 之间的单向的、可靠的连接

消息通道代理（MCA）

- 发送 MCA
- 接收 MCA

通道代理的启动

- 应用程序激活
- 消息到达触发器
- 网络远程启动

通道代理的关闭

- 超时自动关闭

通道代理的属性：

属性	描述
Transport type	决定要采用的传输协议
FIFO delivery	表明消息将要按照发送的次序到达

Message length	单个消息的最大长度
Setup retry count	指定启动远程 MCA 的最大重试次数
Delivery retries	MCA 将收到的消息放入队列的最大重试次数

4.分组通信与多播通信

4.1 概念

组：由系统或用户确定的若干个进程的集合

通信方式：

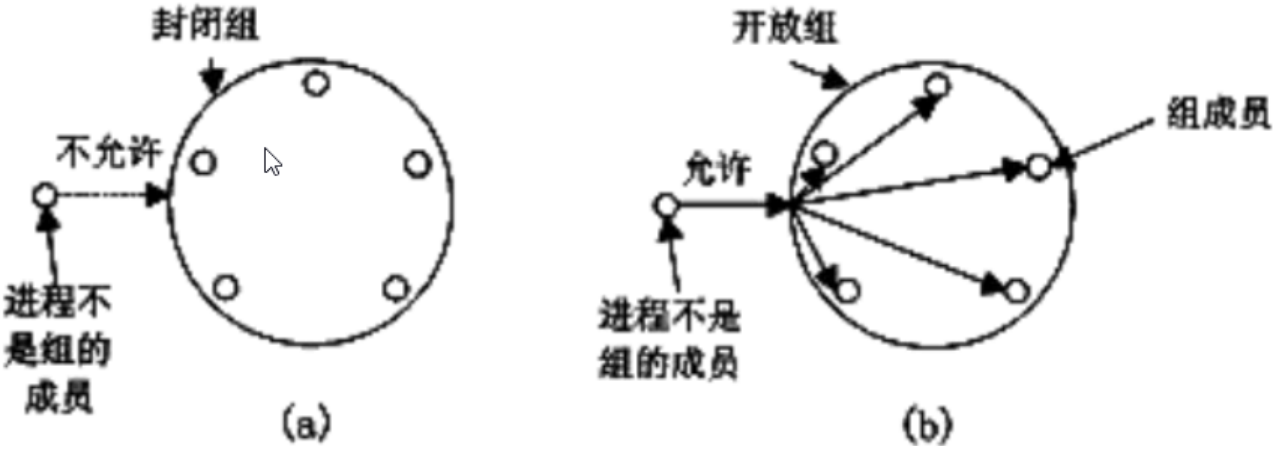
- 点到点通信（point-to-point）：单播（unicast）
- 一到多通信（one-to-many）：多播（multicast）、广播（broadcast）

4.2 组通信的设计

（1）封闭组与开放组

封闭组：只有组内成员之间可以广播通信

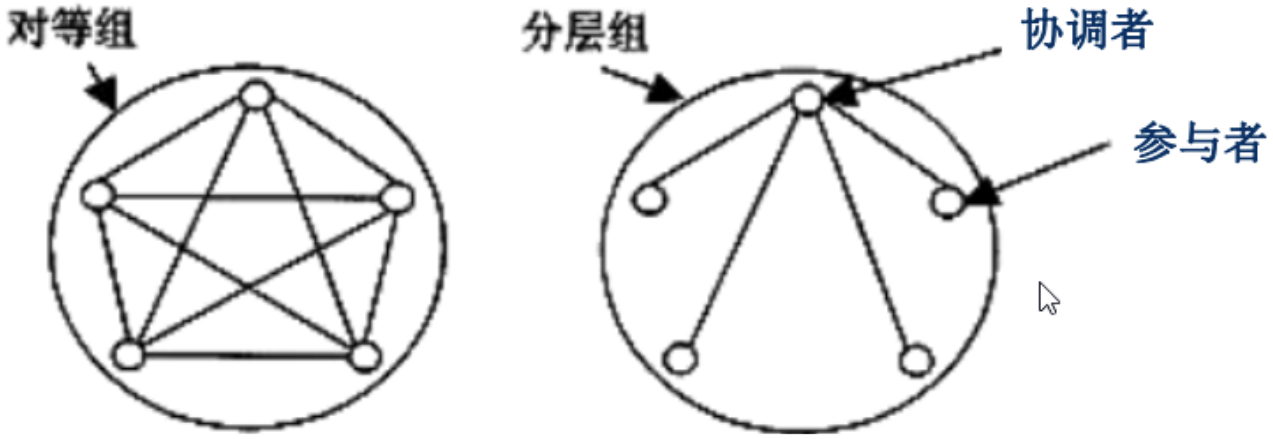
开放组：反之，组外进程也可以对组内进行广播



（2）对等组与分层组

对等组：成员之间的地位是平等的

分层组：反之，存在控制其他节点的超级节点，称为协调者



（3）组的成员管理/寻址

组的成员籍管理（membership）

- **集中式方法**：组服务器登记所有组成员
- **分布式方法**：所有节点自己登记同组成员

组的加入/退出

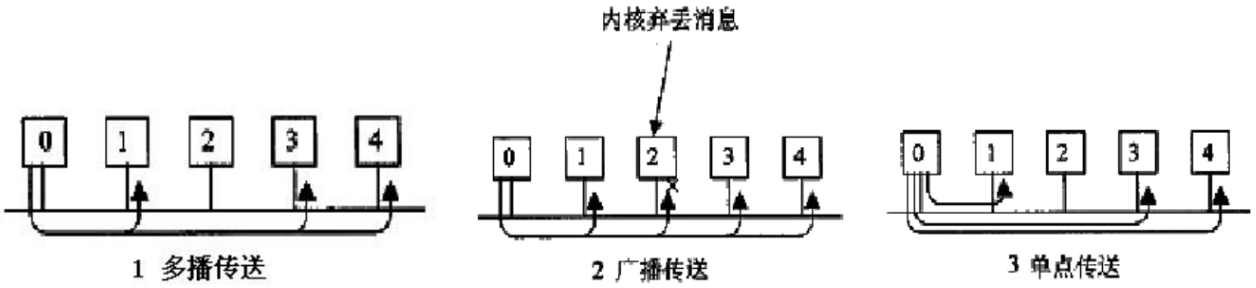
- **集中式方法**：通知组服务器
- **分布式方法**：通知所有节点

组的寻址

- **组地址**：唯一 ID 标识号
- **目的地址列表**：包含所有成员的地址，如 IP+ 端口号
- **谓词寻址**：关于成员地址的布尔表达式

（4）发送的实现方式

1. **多播**：利用网络的多播功能
2. **广播**：利用网络的广播功能
3. **单点**：利用进程间的点到点通信



（5）发送和接受原语

- `group_send`
- `group_receive`
- 单程（`one-way`）模型

（6）原子性

原子性广播：一个消息被所有成员接收到，或者，一个也没有接收到。

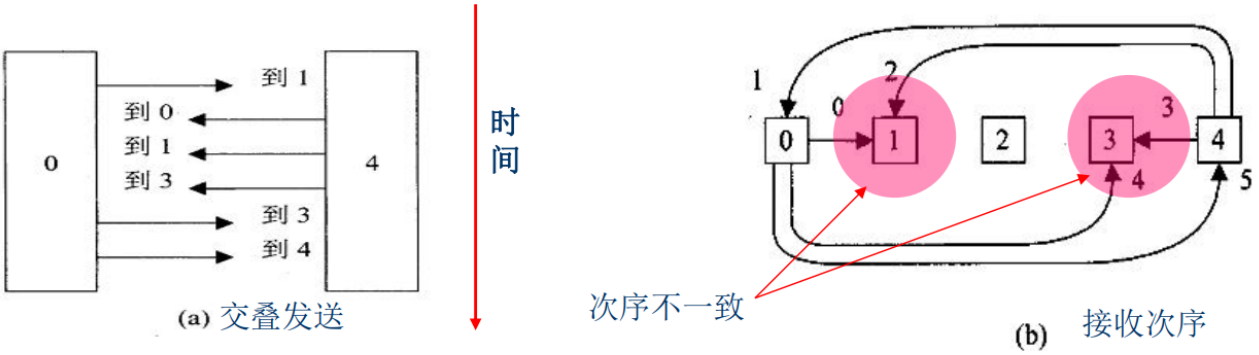
（7）消息的收发顺序

全局时间顺序

- `if ts(m1) < ts(m2) then G(m1) < G(m2)`?
- 绝对次序

一致性时间顺序

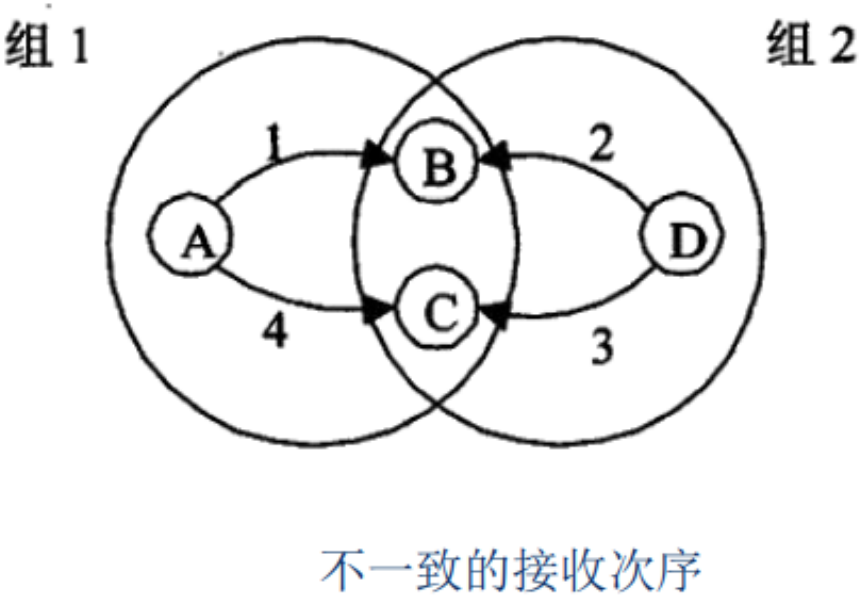
- `G(m1) < G(m2)` 或者 `G(m2) < G(m1)`
- 相对的次序



（8）重叠组

例：不一致性问题

在不同组之间的进行协调非常困难

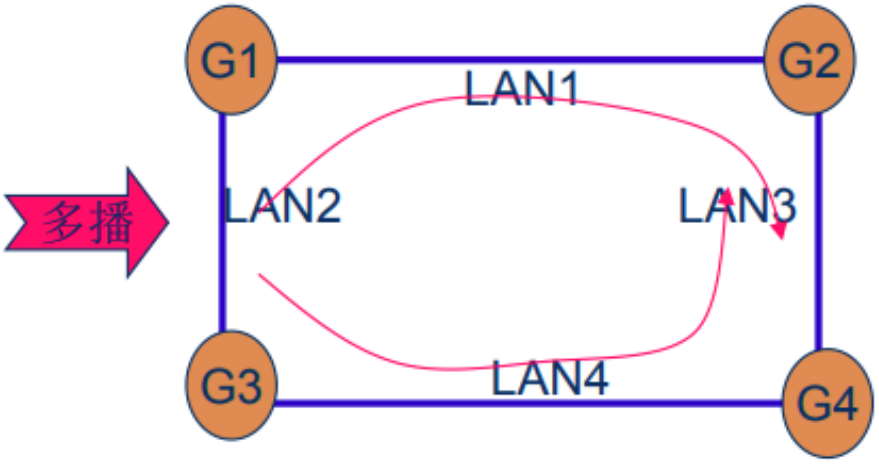


不一致的接收次序

（9）可伸缩性

例：在互联网络上，网关会在多个 LAN 上重复传送一个消息

很难在跨网络环境上实现



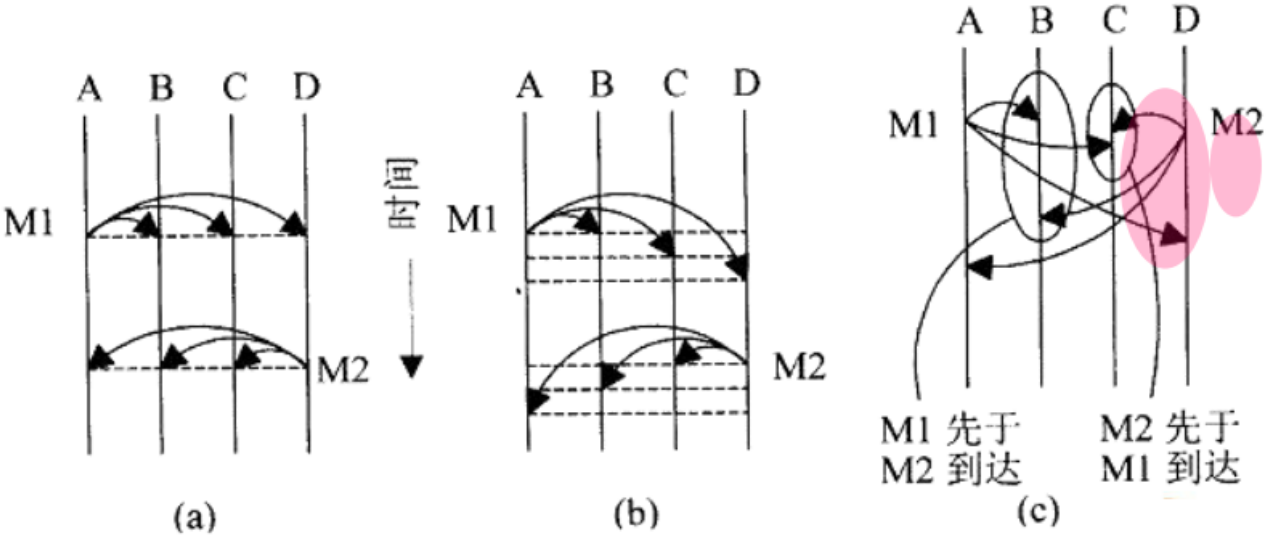
4.3 ISIS 中的通信

ISIS：康奈尔大学开发的工具集（1987）

（1）同步类型

1. 同步：同一事件在各处的发生严格地按照同一时间
2. 松散同步：事件发生需要一定的时间，但顺序一致

3. 虚拟同步：具有因果关系的事件，必须服从同一次序



（2）事件类型

因果相关的（causally-related）：如果事件 A 的发生可能影响事件 B 的发生，或反之

并发的（concurrent）：事件 A 的发生与事件 B 的发生没有关系

（3）ISIS 广播原语

- ABCAST：弱同步；两端提交协议
- CBCAST：虚拟同步

- GBCAST：管理组成员的 ABCAST 通信

（4）CBCAST 协议

时间戳向量：设组内有 n 个进程，每个进程设置一个 n 元向量 $V[n] = \{V[1], V[2], ..., V[n]\}$

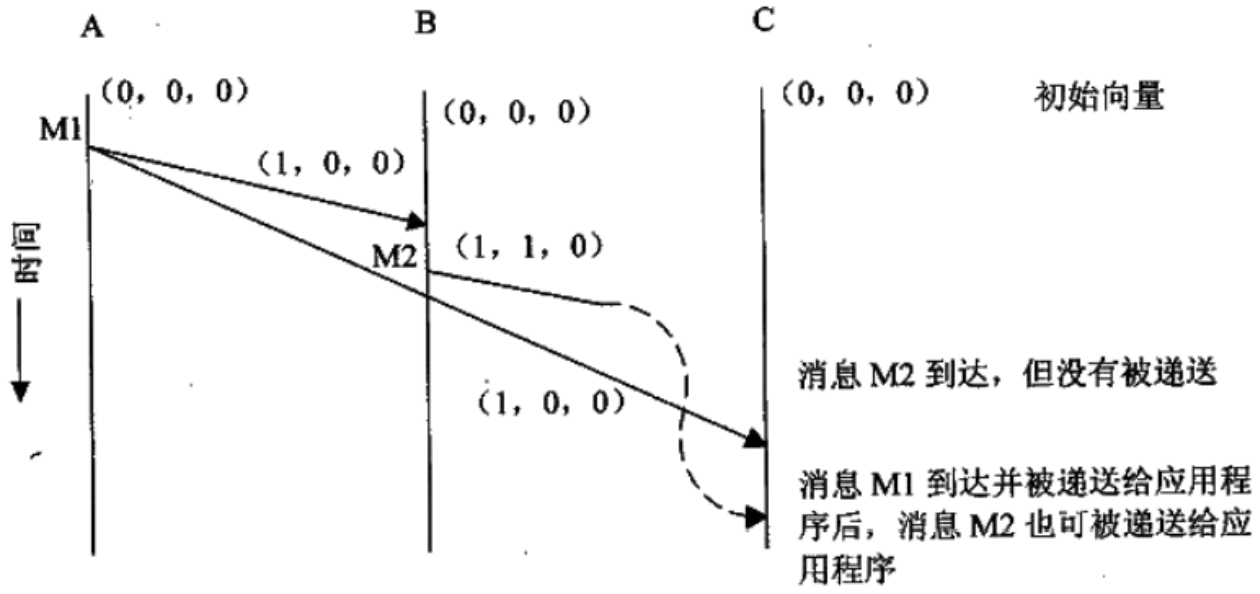
发送者：修改向量： $V[j] = V[j] + 1$; （第 j 个进程）

接收者：

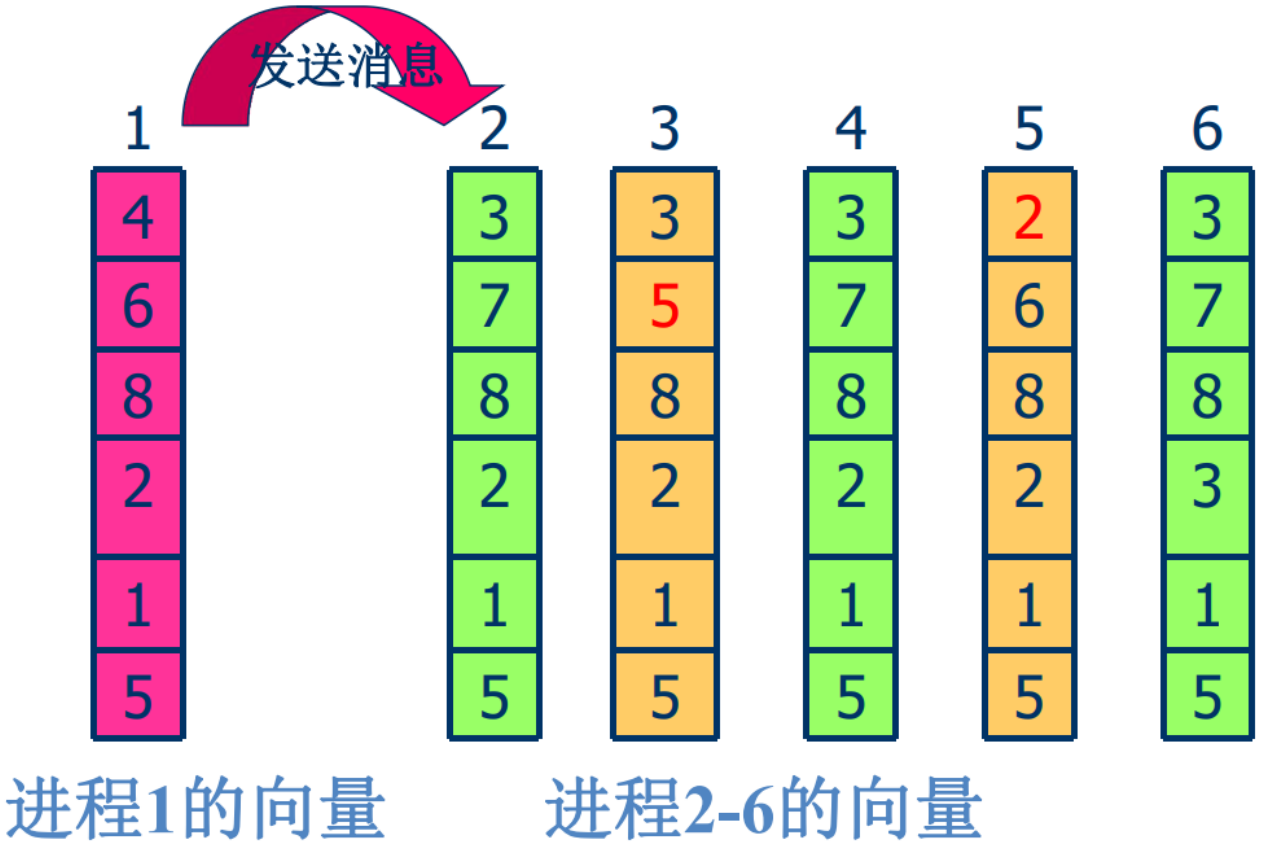
- 接受条件： (1) $L[j] = V[j] - 1$; (2) $L[i] \geq V[i](i \neq j)$?
- 修改向量： $L[j] = V[j]$ 如果可以接收

其中， V_i 是发送方向量种的第 i 位， L_i 是接收方向量中的第 i 位，假设消息由进程 j 发送。

示例：



举例 1：对于下列表格表示的进程，由进程 1 向其他的进程发送消息，此时刻，哪些进程需要等待，哪些进行可以接收消息。



对于此题，需要分别分析两个条件是否满足。

对于条件（1）：消息由进程 1 发送，则首先需要满足 $V_1 = L_1 + 1$

也就是对于每个进程的第一维数据（4，3，3，3，2，3）而言，4=3+1，所以可以看出进程 5 不能接收消息，需要等待，其余进程均满足条件（1）。

对于条件（2）：消息由进程 1 发送，则需要满足 $i \neq j$（即所有进程排除第一维数据），有 $V_i \leq L_i$

所以对于 1–6 这几个进程而言，排除第一维数据后，其余的数据都需要比进程 1 的各维数据相等或更大，所以可以看出对于进程 3 而言 5<6，所以进程 3 不能接收消息，需要等待。

综上两个条件，进程 3，5 需要等待，其余进程均可同时满足条件（1）（2），所以可以接收进程 1 发送的消息。

示例 2：下面表格中分别是进程 1 到进程 6 中消息的时间戳向量，在满足因果关系的前提下，试分析说明进程 2 中当前的消息 m 在哪些进程中能够递交，在哪些进程中暂时不能递交。

1		2		3		4		5		6

7		7		7		5		7	7
5		7		6		6		6	6
5		5		5		5		5	5
4		4		4		4		3	4
3		3		3		3		3	3
1		1		1		1		1	1

就是进程 1，4，5 需要等待；3，6 可以接收消息。

4.4 应用层多播

将节点组织成一个覆盖网络（overlay network），用来实现组通信

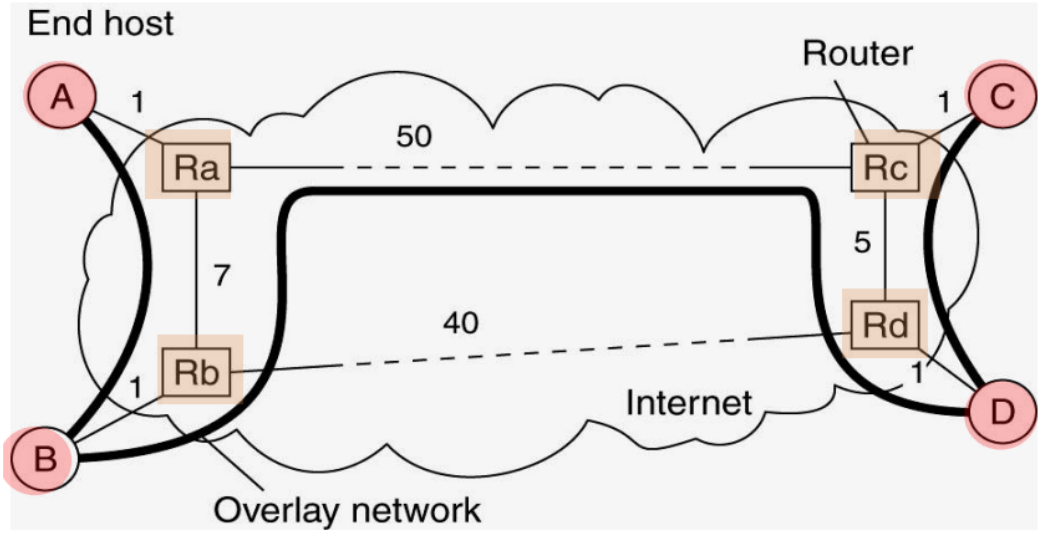
树型覆盖网络：节点之间只有唯一路径

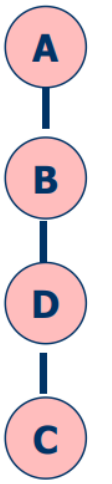
网状覆盖网络：节点之间由多条路径

（1）覆盖网络的性能

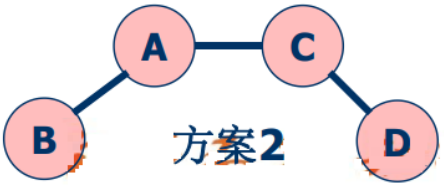
覆盖网络的链接与实际网络层中路由器的链接之间的关系

- 方案 1：若 A 多播消息，$< B, Rb >$，$< Ra, Rb >$ 等链路会重复传输
- 方案 2：消除了重复





方案1



方案2

（2）多播树的性能

链接树（link tree）：计算数据包通过每个物理链接的次数

延长（stretch）或相对延时惩罚（relative delay penalty）

- 两个节点之间的延时比

> Cost（B–Rb–Ra–Rc–C） =59
> Cost (B–Rb–Rd–Rc–C) =47
> stretch = 59/47 = 1.25

树开销

- 传播消息到所有节点的总时间开销
- 构造一棵开销最小的生成树

4.4 基于 gossip 的数据通信

传染协议(epidemic protocol)：用于使本地信息在大规模节点集合中快速地传播

消息传播模型：基于传染病理论

节点状态

- 感染的（infective）：已被更新，并愿意参加传播
- 疑似的（susceptible）：还未被更新
- 隔离的（removed）：不愿意参加传播

4.5 传染协议

（1）反熵（anti-entropy）传播模型

- 熵：系统混乱度的量度

（2）节点 P 随机地选择节点 Q，交换更新信息

- 基于 push 的方法，P 只将自己的更新推送给 Q （易感结点多）
- 基于 pull 的方法，P 只从 Q 中拉取新的更新 （感染结点多）
- 基于 push/pull 的方法，P 和 Q 互相发送更新
  - 传播一个更新信息到所有节点需要 `O(log(N))` 轮，N 为系统的节点数。
  - 每个节点作为发起者随机地选择一个节点进行了至少一次更新信息交换，定义为 1 轮（round）

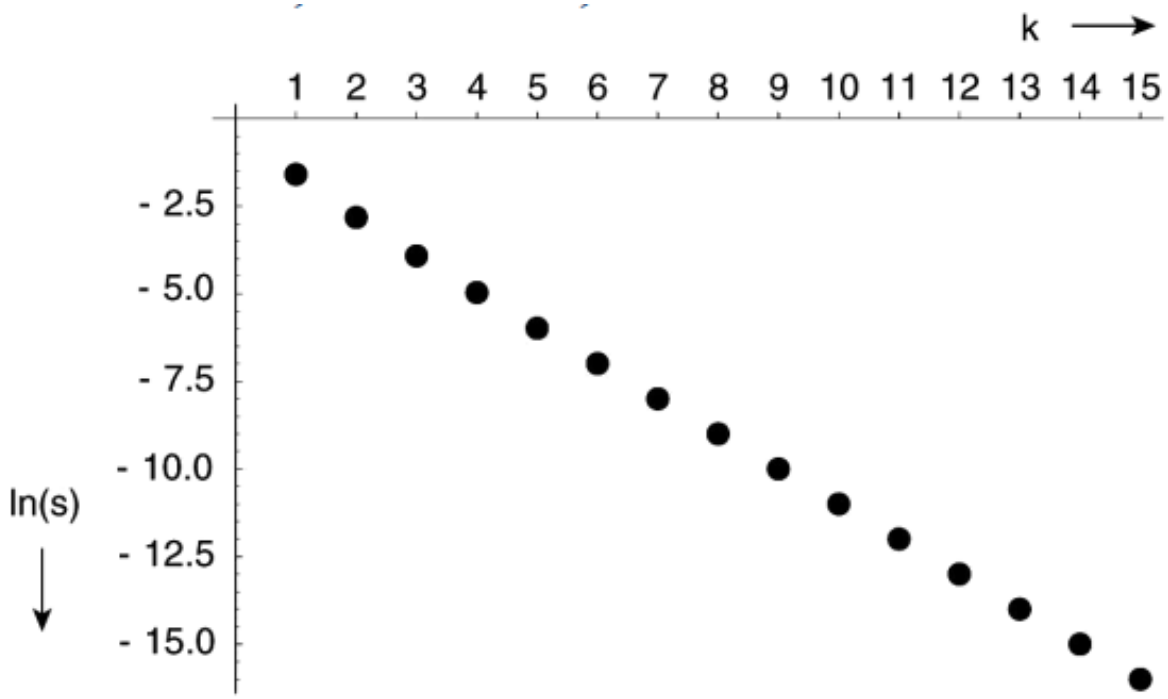
（3）传播方法

谣传(rumor spreading)或称为传言（gossiping）

仍为疑似的服务器的比例： $s = e^{-(k+1)(1-s)}$

有 $1/k$ 不再传播信息的概率

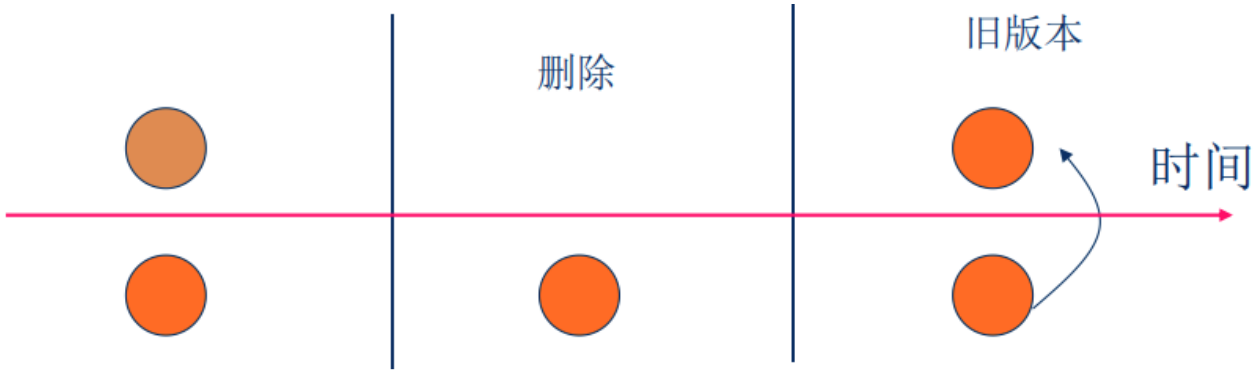
例如： $k = 4$，$s < 0.007$，即 0.7% 未被感染



未被更新的节点的系数s与参数k的关系

（4）删除数据

旧版本问题



（5）解决方案

死亡证