# 分布计算环境

北京邮电大学计算机学院

# Chapter 2

# 分布式系统的基本原理

# 主要内容

- 体系结构
- 进程
- 通信
- 命名
- 一致性和复制
- 容错
- 安全

北京郵電大學

# 通信主要内容

- 分层通信协议
- 远程过程调用和远程对象调用
- 面向消息的通信
- 面向流的通信
- 多播通信

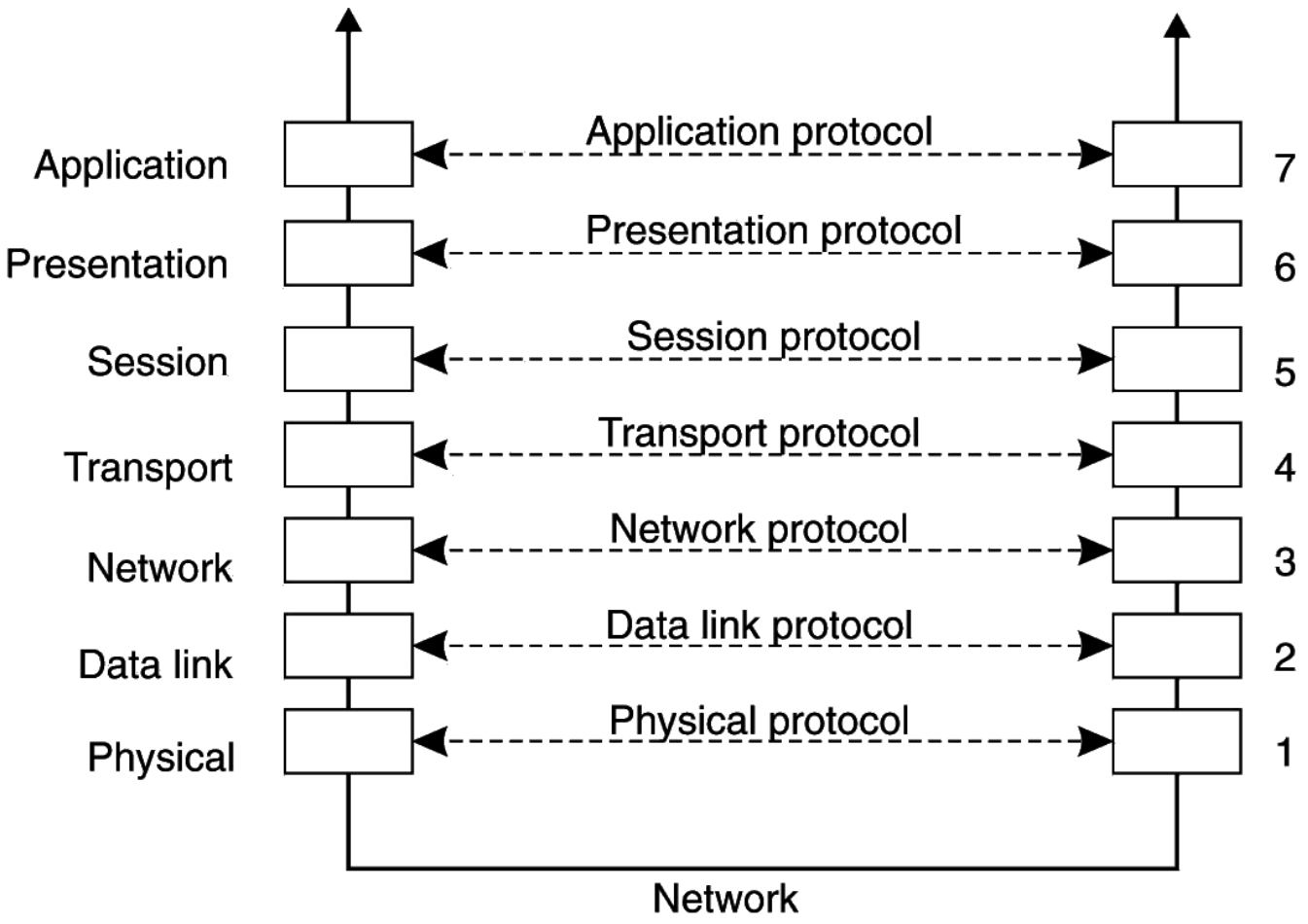
# 通信主要内容

- 分层通信协议
- 远程过程调用和远程对象调用
- 面向消息的通信
- 面向流的通信
- 多播通信

# OSI模型

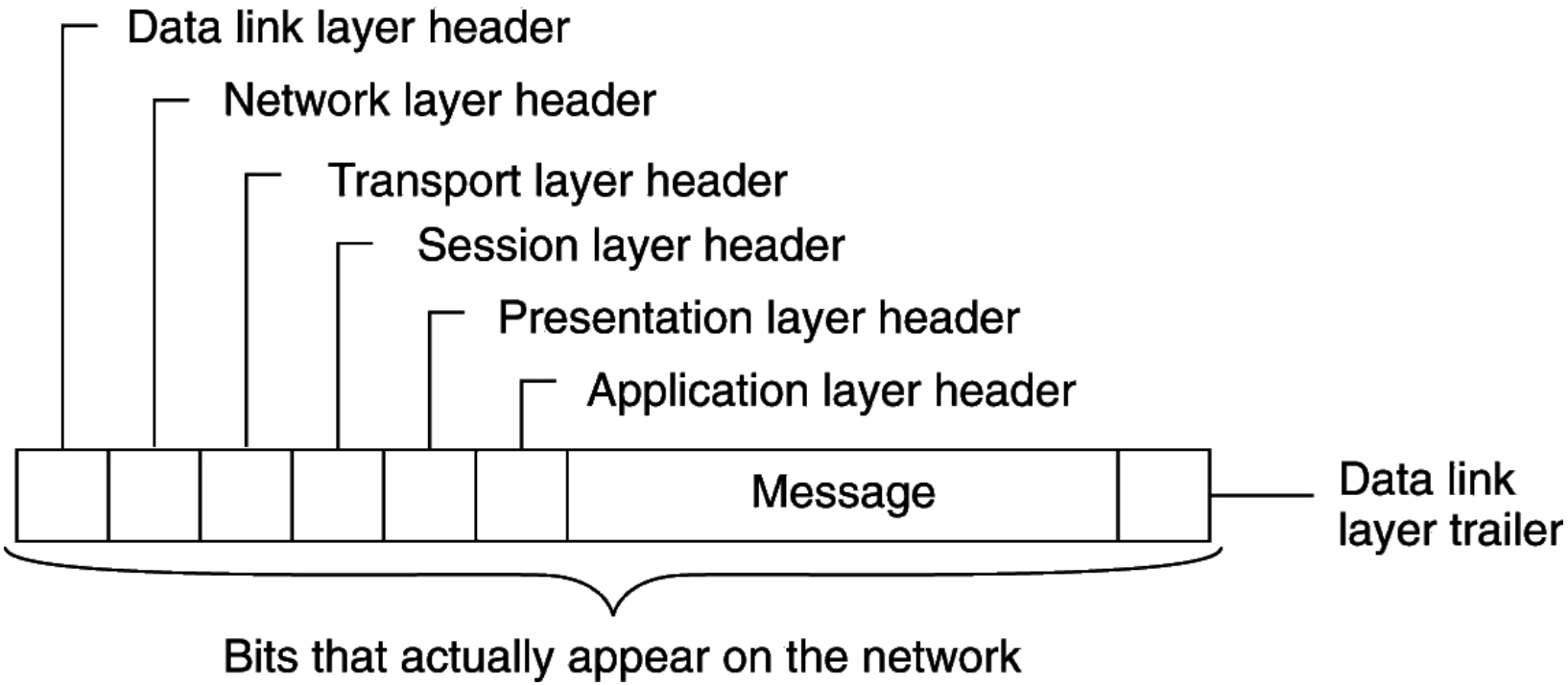
- 由于没有共享存储器，分布式系统中的所有通信都是基于（低层）消息交换的
- OSI模型（开放式系统互联参考模型）用来支持开放式系统间的通信
- 开放式系统是可以通过标准规则与其他开放式系统通信的系统，这些规则规定了发送和接收消息的格式、内容以及相应的含义

# OSI模型中的层、结构和协议




- 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

# 典型消息格式

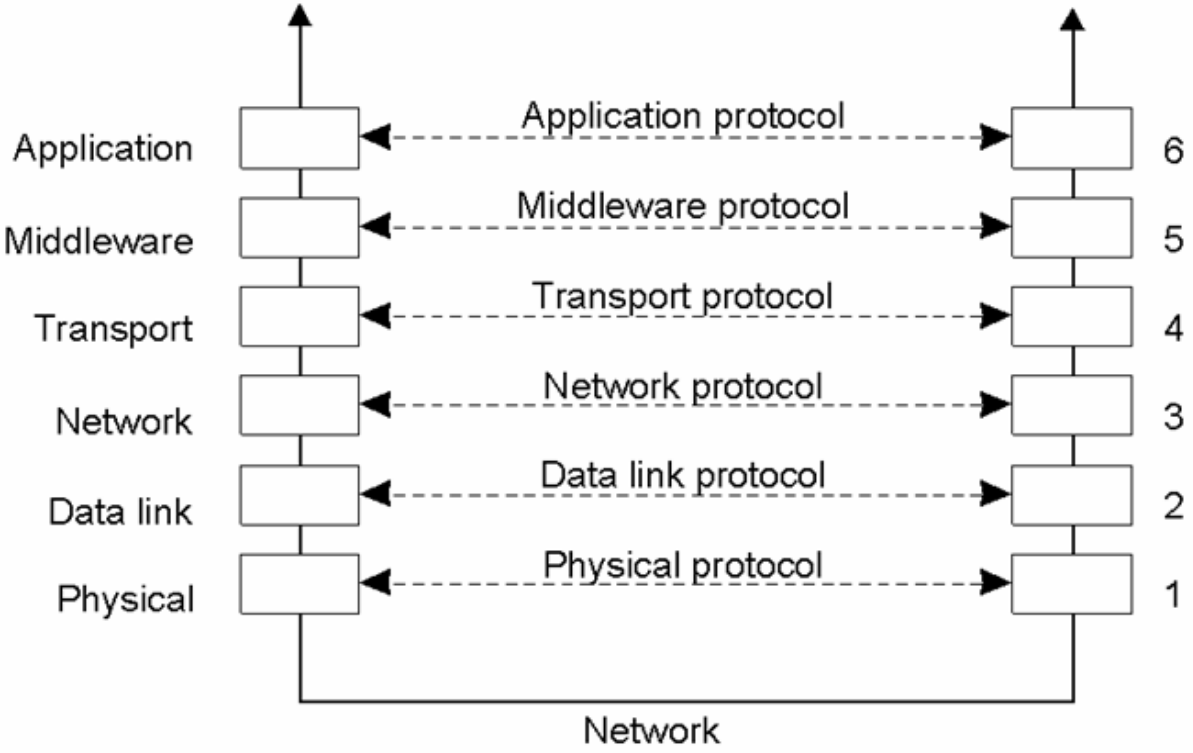

Data link layer header
Network layer header
Transport layer header
Session layer header
Presentation layer header
Application layer header
Message
Data link layer trailer
Bits that actually appear on the network

# 应用层协议

- 在OSI七层模型的传输层之上划分了三个层，在实践中，只用到了其中的应用层
- 在Internet协议簇中，传输层之上的所有内容都合并到了一起，称为应用层
  - 应用层成为所有由于各种原因不能归纳到某个较低层中去的应用程序和协议的容器
- 缺乏对应用程序、针对特定应用程序的协议以及通用协议的明确区分
  - ftp协议和ftp程序
  - HTTP协议

# 中间件协议

- 有的应用层协议，可用于支持多种应用程序的通信，因此可看做是对多种应用程序有用的通用协议
  - 但不能算作传输层协议，很多情况下归入**中间件协议**
  - 中间件协议：中间件使用的，用于建立各种中间件服务的协议，如支持通信、认证、事务、容错……



# 中间件协议（2）

- 中间件协议是中间件使用的用于支持各种中间件服务的协议，如支持通信、认证、事务、容错……
- 不同的中间件系统有不同的中间件协议
  - 支持远程过程调用的协议，如DCE中
  - 支持远程对象调用的协议，如CORBA中
  - 支持实时流数据传输并保持同步的协议
  - 可靠多播协议，用于支持可靠多播服务的中间件系统

# 中间件支持的通信类型

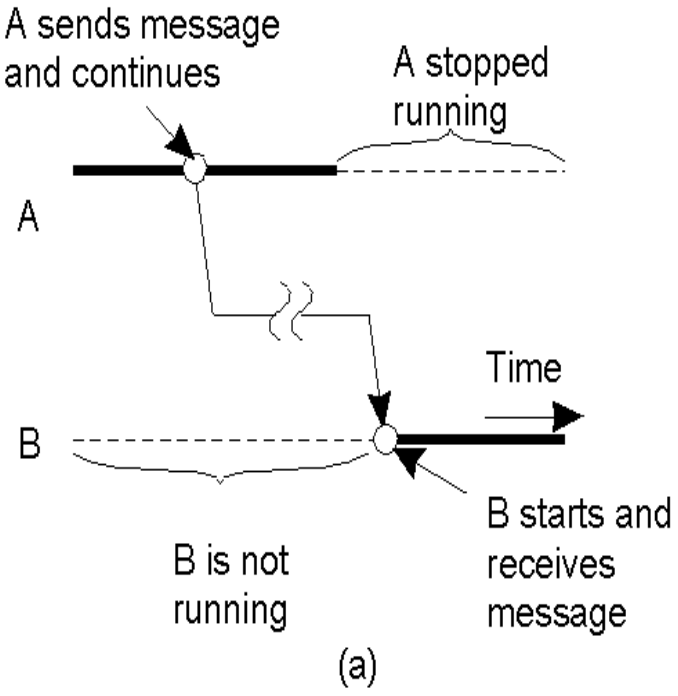
- 从通信持久性方面
  - 持久通信：传输的消息一直由通信中间件存储，直到该消息被传送给接收方，如电子邮件系统
  - 瞬时通信：通信中间件只在发送和接收应用程序正在运行的时候才存储消息，即由于传输中断或者接收方当前不在活动状态，中间件就不传输消息，而是丢弃消息

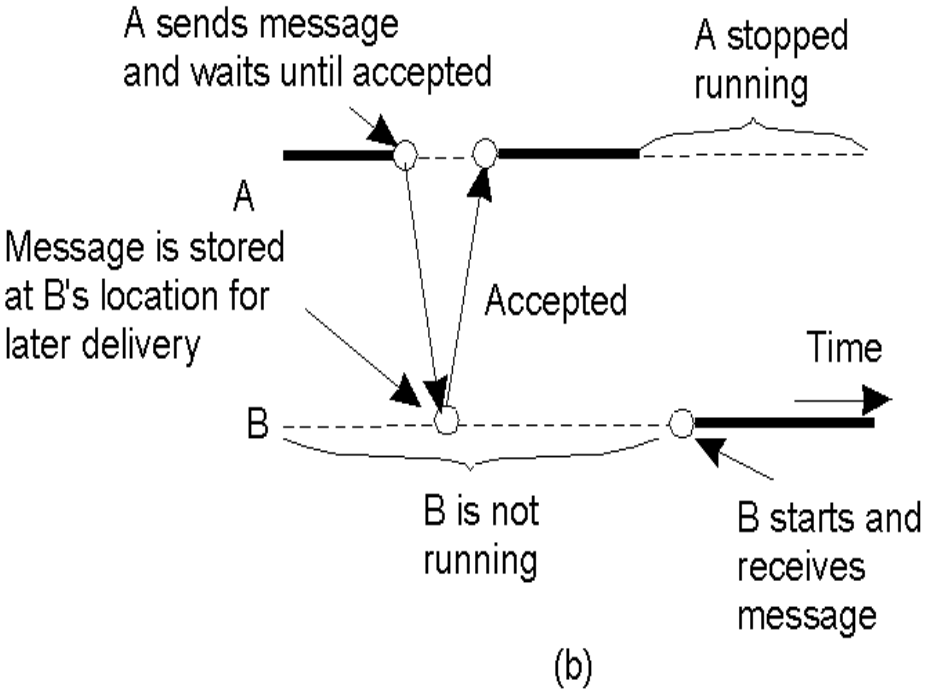
# 中间件支持的通信类型(2)

- 从通信同步性方面
  - 同步通信：发送方提交消息后将被阻塞，直到某个事件发生：发送方可有三类同步点
    - 基于发送：发送方中间件请求传输完成
    - 基于接收：请求被传送到目标接收方
    - 基于响应：接收方返回响应
  - 异步通信：发送方在提交要传输的消息后立刻接着后续的执行，不会阻塞
- 从通信连续性方面分：不连续通信和流通信

这些通信方式可以进行各种组合

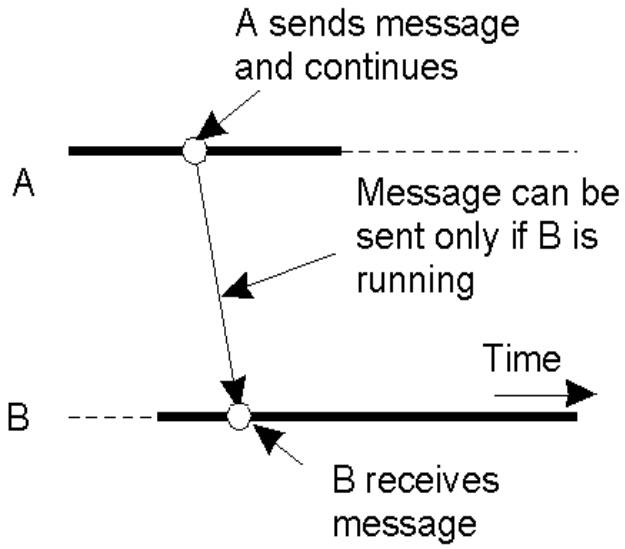
# 通信持久性和同步性的组合



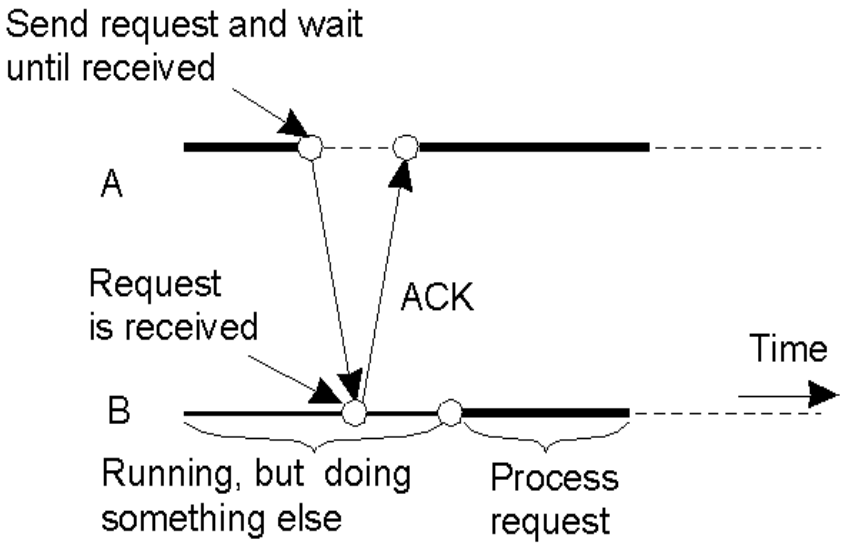




（a）持久异步通信　　(b)基于发送的持久同步通信

# 通信持久性和同步性的组合（2）




（c) 瞬时异步通信



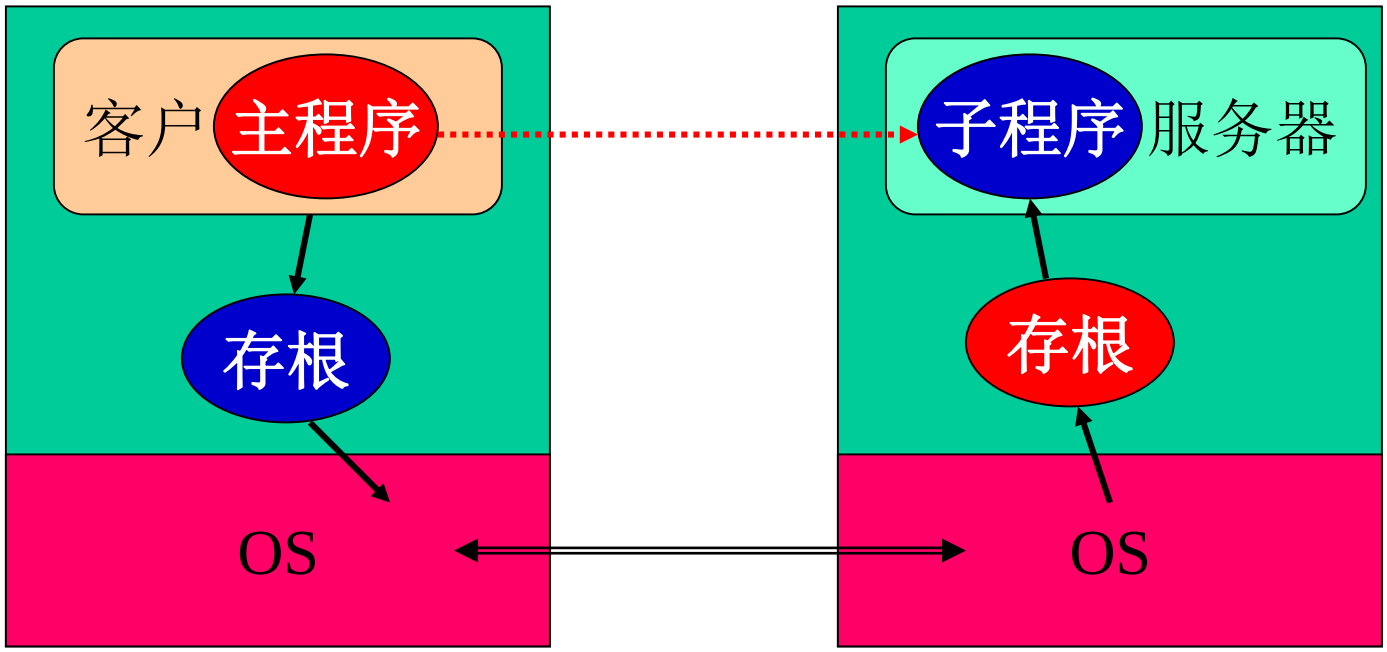

(d) 基于发送的瞬时同步通信

# 通信主要内容

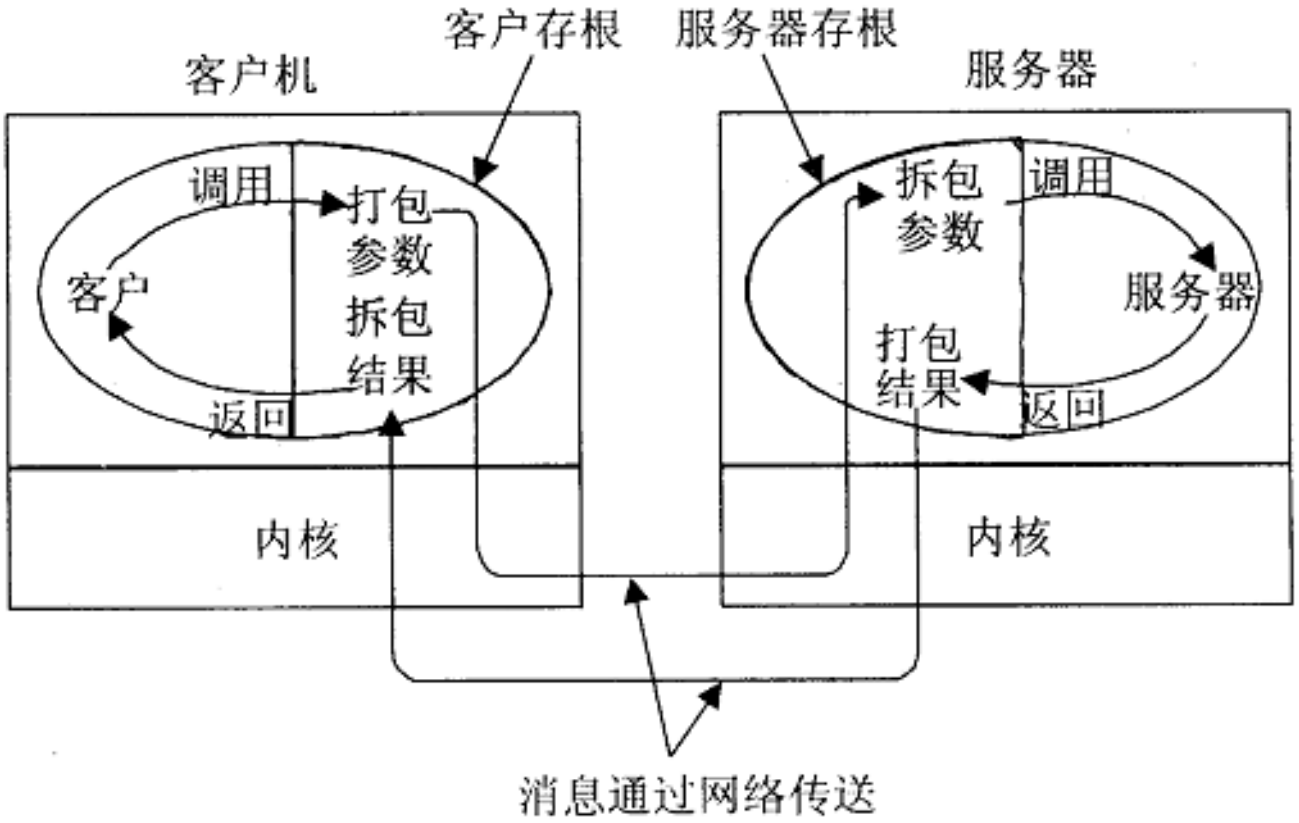
- 分层通信协议
- 远程过程调用和远程对象调用
- 面向消息的通信
- 面向流的通信
- 多播通信

# 远程过程调用RPC

- 像调用本地子程序一样，调用远程子程序
  - 调用者和被调者都不用考虑通信问题

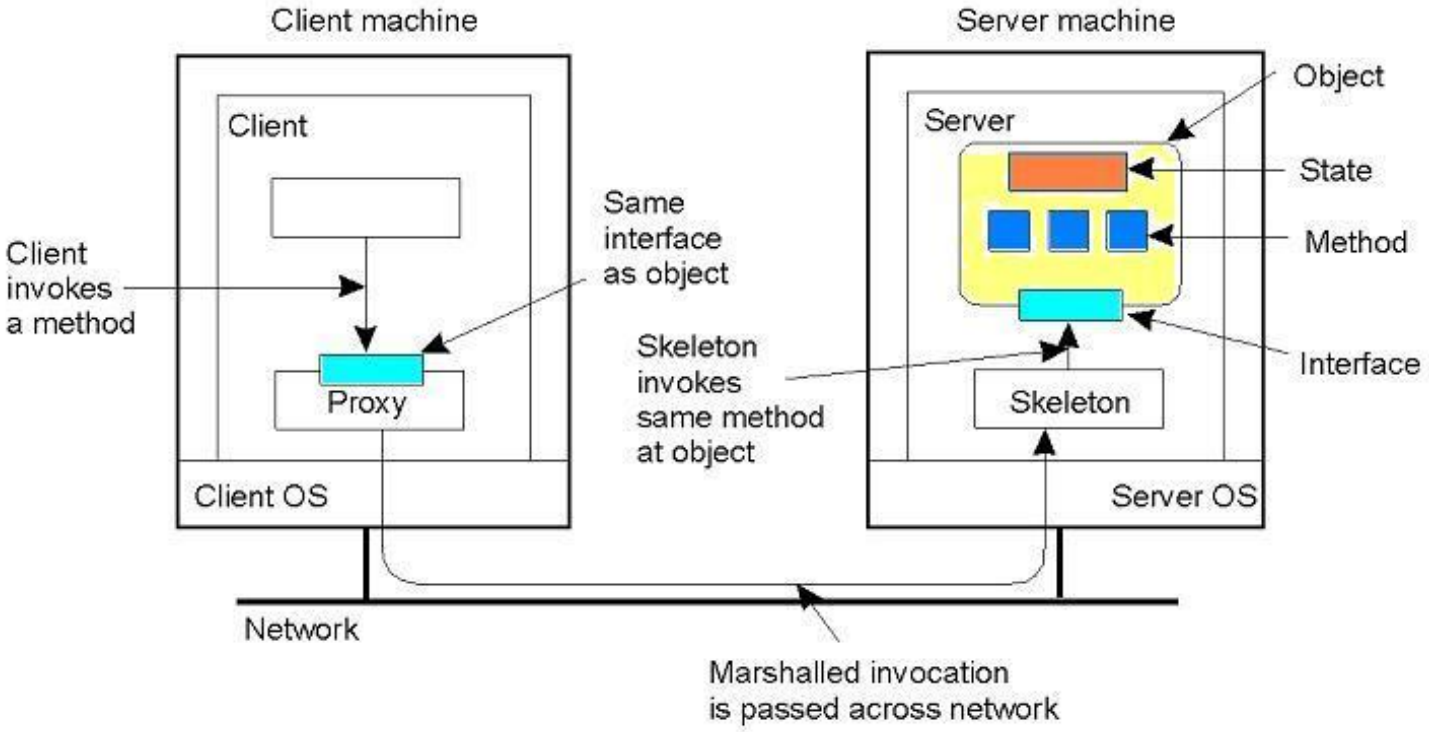


# RPC的基本处理过程


客户存根
服务器存根
客户机
服务器
调用
打包
参数
拆包
结果
客户
返回
拆包
参数
调用
服务器
打包
结果
返回
内核
内核
消息通过网络传送

# 远程方法调用RMI

- 像调用本地对象的方法一样，调用远程对象的方法
  - 调用者和被调者都不用考虑通信问题



# 通信主要内容

- 分层通信协议
- 远程过程调用和远程对象调用
- 面向消息的通信
- 面向流的通信
- 多播通信

# 面向消息的通信

- **远程过程调用和远程方法调用都有助于隐藏分布式系统中的通信复杂度**
- **但这两种机制并不总是适用的**
  - 如当无法保证发送请求时接收端正在执行的情况下
  - 如同步性会阻塞发送进程
- 需要其他通信机制的支持
  - 如面向消息的通信
    - ➔面向消息的瞬时通信
    - ➔面向消息的持久通信

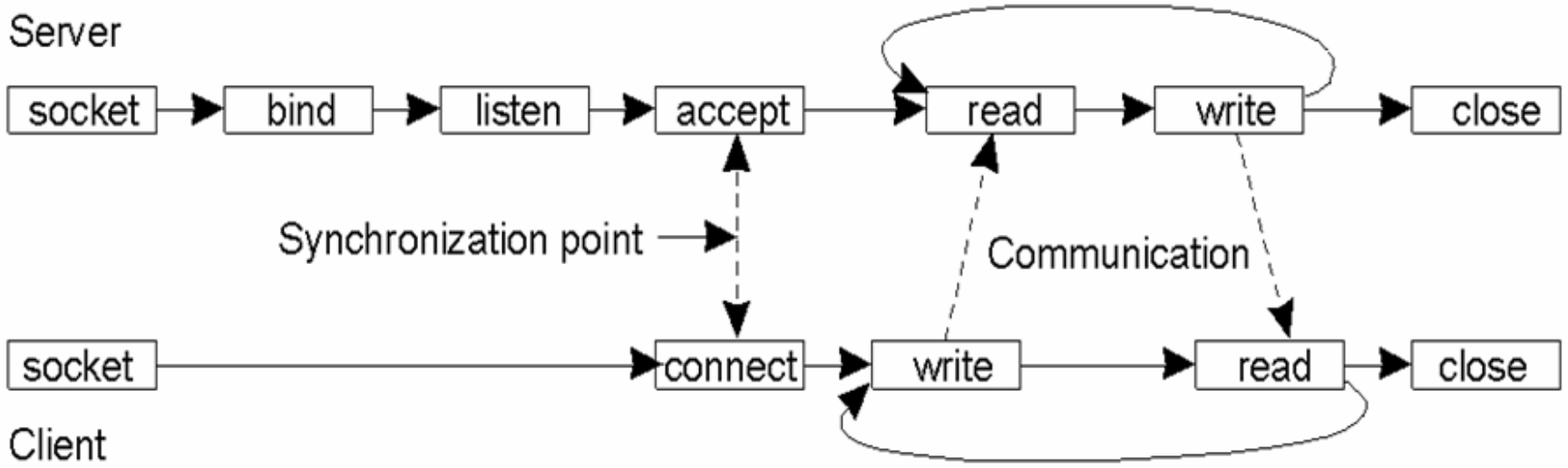
# 面向消息的瞬时通信：套接字

- 很多分布式系统和应用程序直接构建在传输层提供的简单的面向消息的模型之上
- 程序员通过一个简单的原语集就可以使用传输层提供的全部（消息传递）协议
  - 标准化的原语集接口使得应用程序在不同机器之间的移植变得容易
- 如支持TCP/IP的Berkeley套接字socket

# TCP/IP套接字原语

| 原语 | 意义 |
| --- | --- |
| Socket | 创建新的通信端点 |
| Bind | 将本地地址附加(attach)到套接字上 |
| 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 |
| Send | 通过连接发送数据 |
| Receive | 通过连接接受数据 |
| Close | 释放连接 |

# 使用套接字的面向连接通信模式



# 感受套接字编程（Java）

**Server 侧：**

```
ServerSocket server= new ServerSocket(10000);
While(true){
    Socket s= server.accept();
    new ServerThread(s).start();
}
```

**客户侧：**

```
Socket s= new Socket("192.168.2.1",10000);
InputStream is = s.getInputStream();
is.close();
s.close();
```

# 面向消息的瞬时通信：MPI

- 套接字抽象层在传输层，只支持简单的send和receive原语
- 套接字用于使用TCP/IP协议进行通信，不适用于为高速互联网开发的专用协议，比如不同的缓冲和同步方式
- 多数高性能计算机系统附带专用通信库，带来可移植性问题
- 消息传递方面的标准出台：消息传递接口MPI

# MPI的特点

- 为并行应用程序设计，因此是为瞬时通信而量身定做
- 支持分组通信
  - 地址（groupID, processID）可以唯一地确定消息的来源或者目的
- 支持多种瞬时通信方式
  - 瞬时异步 MPI_bsend
  - 基于发送的瞬时同步 MPI_send
  - 基于接收的瞬时同步 MPI_ssend
  - 基于响应的瞬时同步 MPI_sendrecv

# MPI原语例

| 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远程的**MPI**运行时系统为止 |
| **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 |

# 面向消息的持久通信

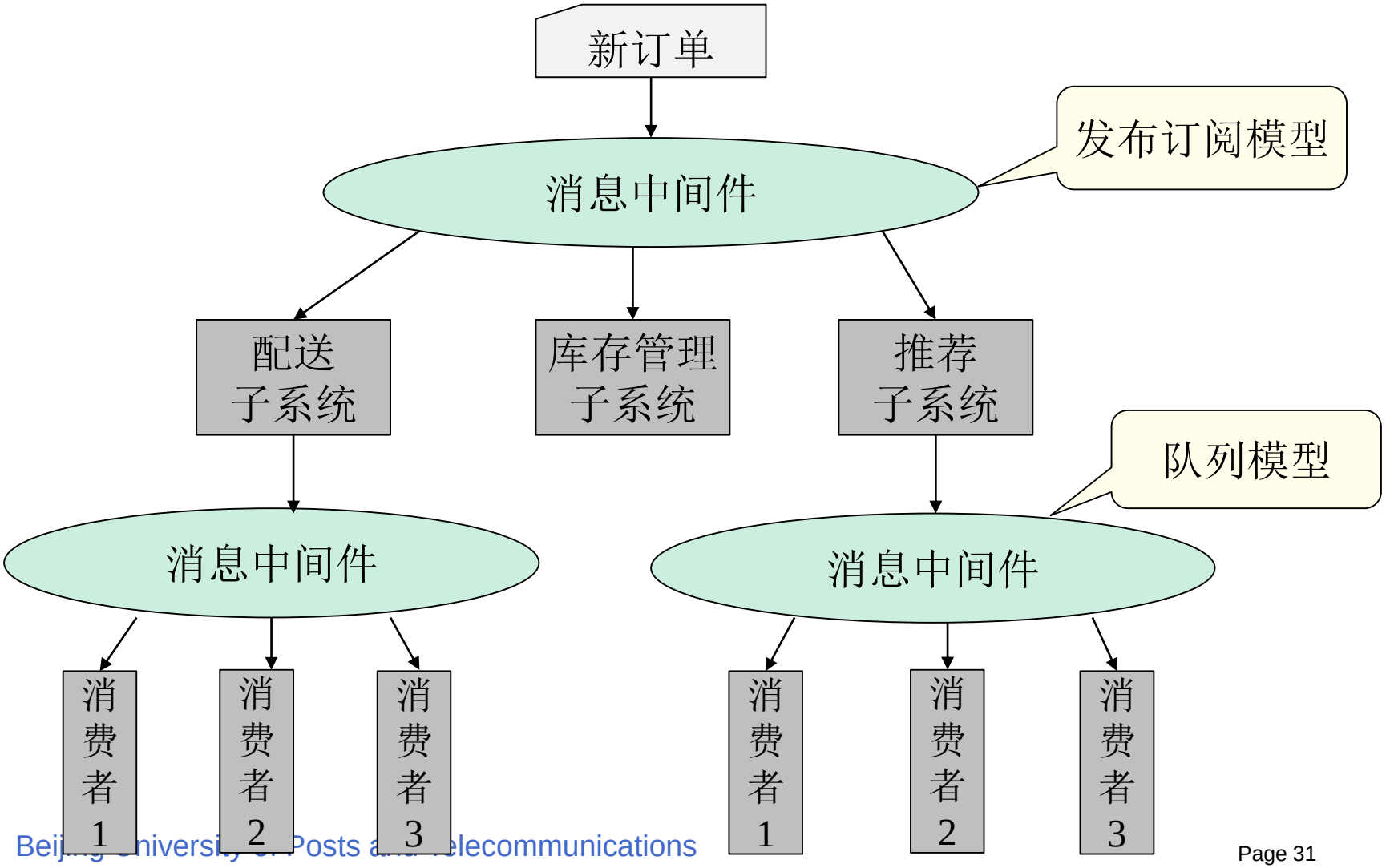
- 其支持系统一般称为消息队列，或者面向消息的中间件
- 支持持久异步和间接通信，通过第三方实体，将通信双方进行解耦
  - 消息发布者（发送者）不需要知道消息会发送给谁，消息消费者（接收者）不需要知道谁发送的消息：空间解耦
  - 发送者和接收者不需要同时存在：时间解耦
- 适合于对传输时间要求宽松的场合，如几分钟甚至更长时间。不保证消息到达接收方的时间，也不保证接收方一定读取消息

# 消息模型

- 队列模型：一组消费者和一组发布者通过一个队列（queue）联系起来，中间件保证有且只有一个消费者收到消息
  - 消费者可以进行负载均衡
- 发布订阅模型：一组消费者和一组发布者通过一个主题（topic）联系起来，发布者将消息发布到某个主题，订阅者订阅某个主题的消息。如果某个订阅者（消费者）订阅了一个主题，中间件保证它能收到该主题的所有消息
  - 消费者一侧没有负载均衡

# 消息模型的应用例

- 很多应用系统常常需要同时用到这两种模型，如



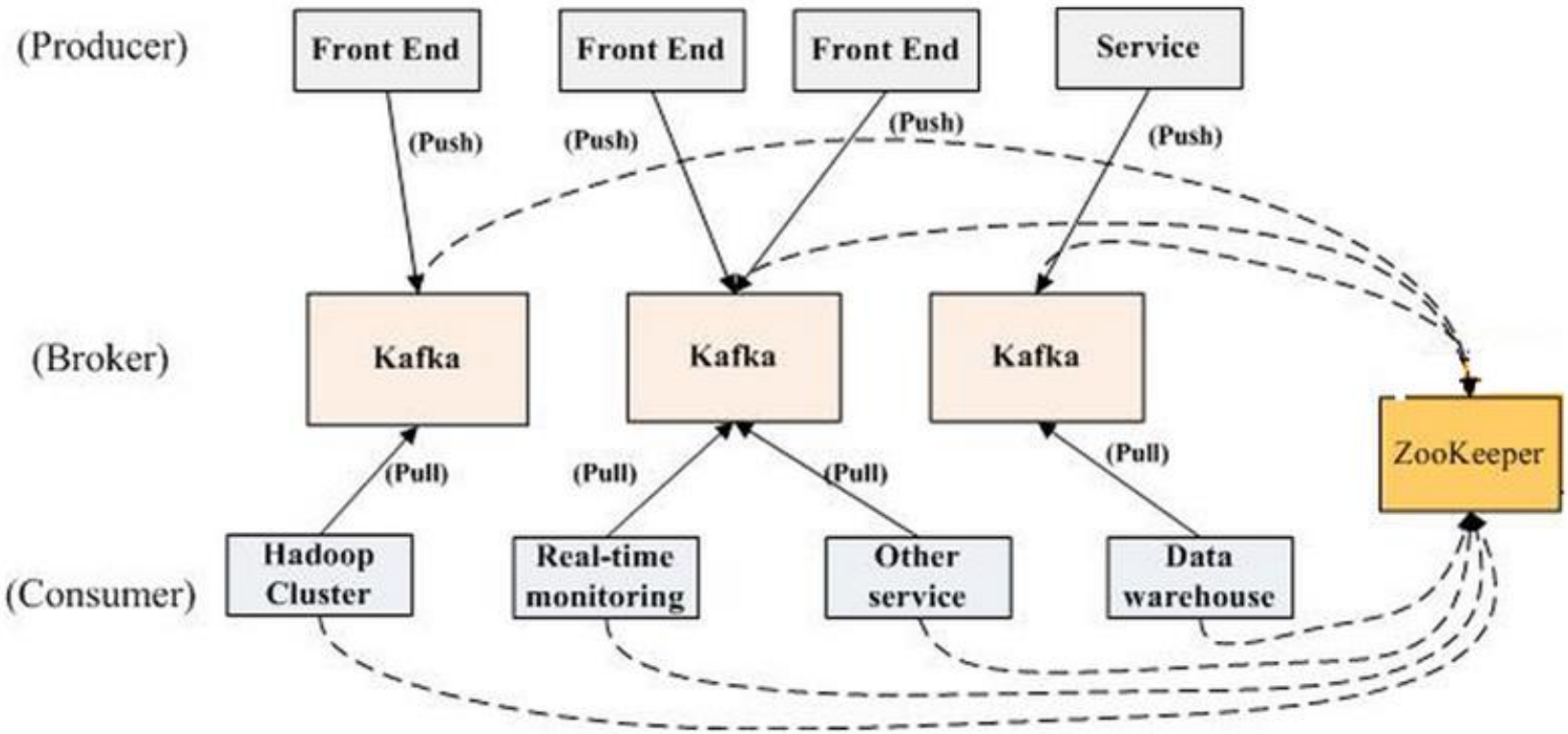
# 面向消息的中间件例：Kafka

- 是一个分布式的基于发布/订阅模式的消息中间件，主要应用于大数据实时处理领域
  - 最初由LinkedIn公司采用Scala语言开发，多分区、多副本并且基于ZooKeeper进行协调，于2010年贡献给了Apache基金会
  - 已定位为一个分布式流式处理平台，以 高吞吐、可持久化、可水平扩展、支持流处理等多种特性而被广泛应用
- 其它面向消息的中间件例：
  - **RabbitMQ**
  - **ActiveMQ**
  - **ZeroMQ**

# 主要设计目标

- 消息持久化：以时间复杂度为O(1)的方式提供消息持久化能力，即使对TB级以上的数据也能保证常数时间复杂度的访问性能
- 高吞吐：在廉价的商用机器上也能支持单机每秒10万条以上的吞吐量
- 分布式：支持消息分区以及分布式消费，并保证分区内的消息顺序
- 跨平台：支持不同技术平台的客户端（如Java、PHP、Python等）
- 实时性：支持实时数据处理和离线数据处理
- 伸缩性：支持水平扩展

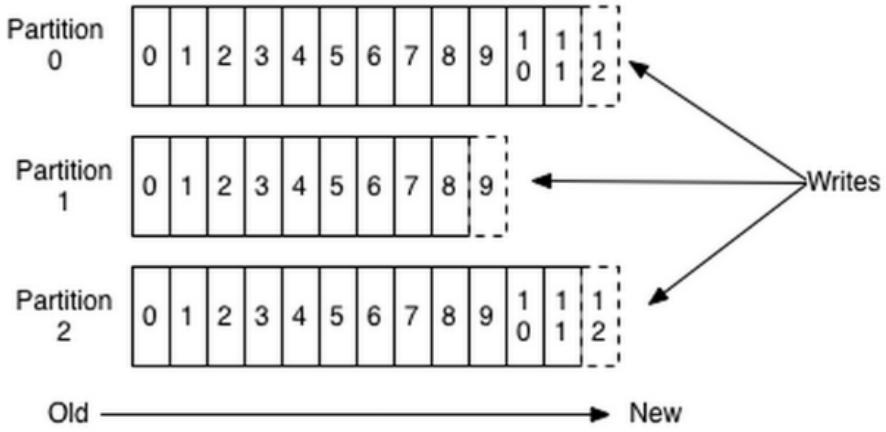
# 总体架构




- Broker：缓存代理，Kafka集群中的一台或多台服务器统称为Broker。Kafka 支持水平扩展，一般 broker 数量越多，集群吞吐率越高
- Producer：消息生产者，如 web 前端产生的 Page View、服务器日志等。使用 push 模式将消息发布到 broker
- Consumer：消息消费者， 使用 pull 模式从 broker 订阅并消费消息
- 通过 Zookeeper 管理集群配置，选举 leader，以及在 Consumer Group 发生变化时进行 rebalance

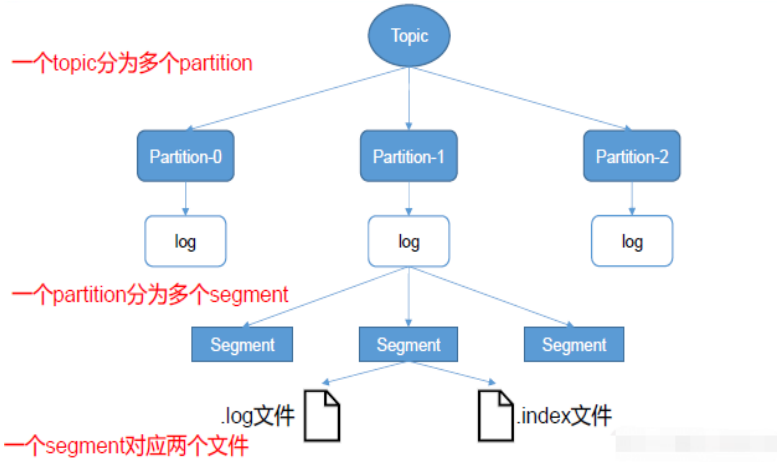
# 主题Topic和分区Partition

- 主题Topic：每条发布到Kafka集群的消息都有一个类别，这个类别被称为topic（相当于队列queue）
- 分区Parition：每个topic包含一个或多个partition，创建topic时可指定parition数量
  - 每个partition对应于一个文件夹，该文件夹下存储该partition的数据和索引文件
- 物理上把topic分成一个或多个partition，使得Kafka的吞吐率可以水平扩展

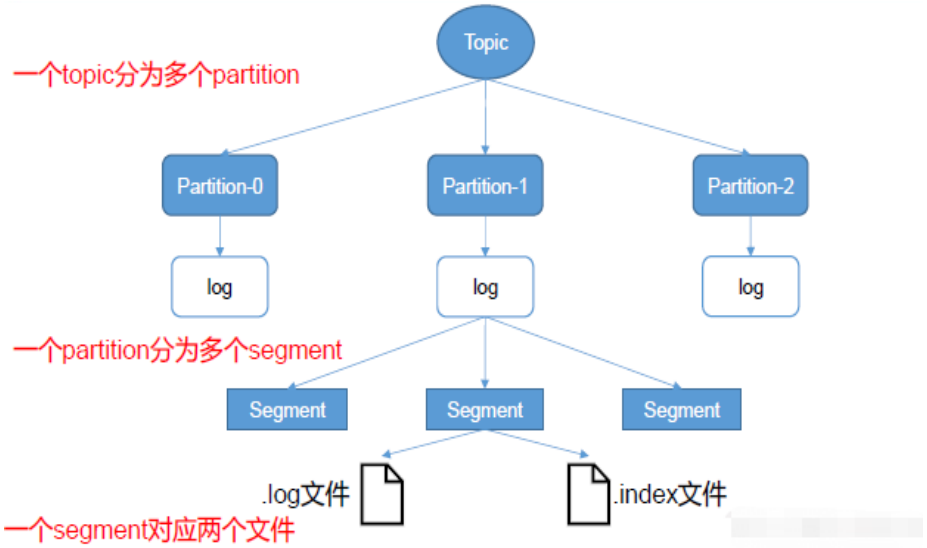


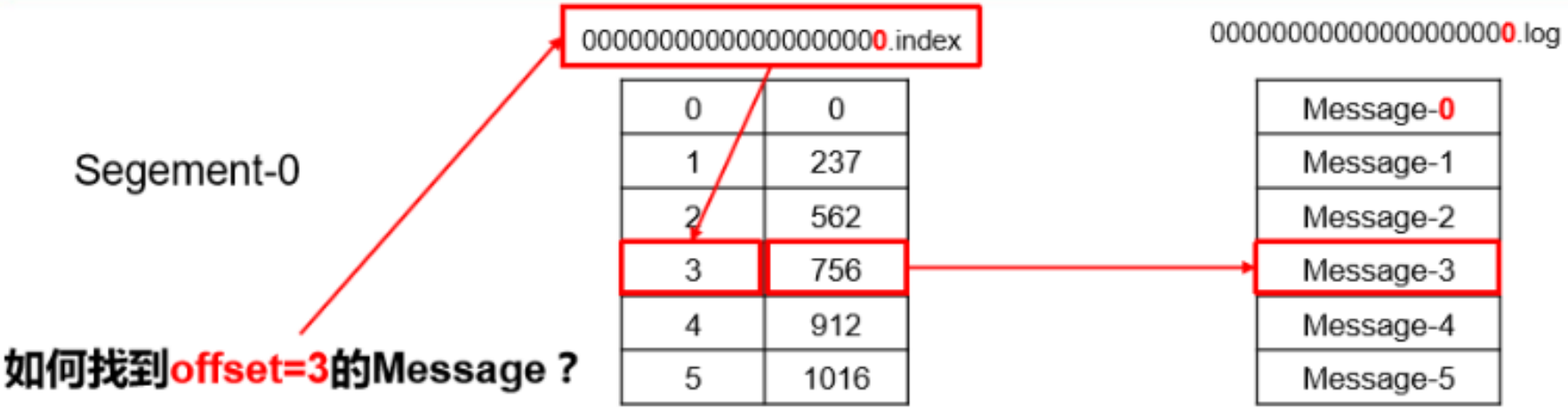
# 日志文件和消息

- Partition中的消息以日志文件的形式存储，日志文件中存放“log entries”序列
- 每个“log entries”主要包含如下信息：
  - 消息长度：4字节
  - 偏移量offset：相对、绝对
  - 消息体
- 日志文件分成多个segment进行存储
  - segment名称从0开始，之后的每一个segment名称为上一个segment文件最后一条消息的offset值
  - 每个 segment对应两个文件——“.index”文件和“.log”文件
  - 这些文件位于同一个文件夹下，该文件夹的命名规则为：topic 名称+分区序号
- index 和 log 文件消息严格按照提交顺序被添加到segment中，消息序列不可修改



# 日志文件和消息（2）



# 消息的发送和存储

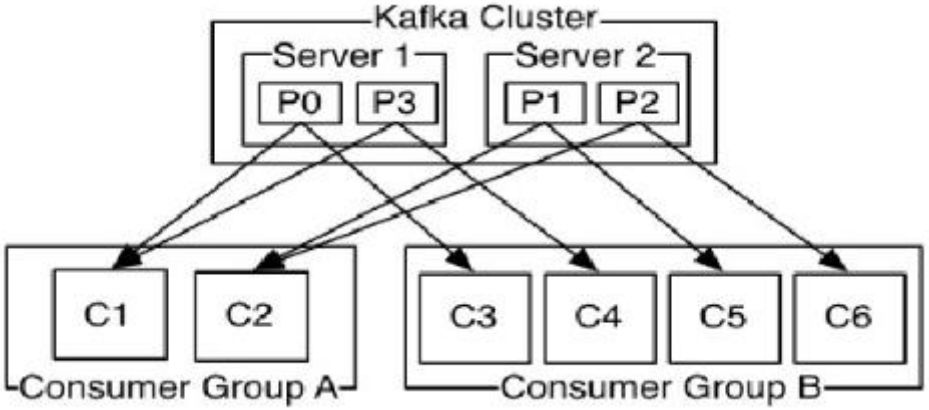
- Producer以某个topic发送消息，其自身负责决定将消息发布到该topic的哪个partition
  - 随机、轮盘式（round-robin）等
  - Producer会尝试在内存中收集足够数据，并在一个请求中一次性发送一批数据
- Broker收到发布的消息后，向对应partition的最后一个segment 上添加该消息
- 当segment上消息条数达到配置的阈值或消息发送时间超过阈值时，segment上的消息会被flush到磁盘。只有flush到磁盘上的消息，才可被订阅者获取到
- Segment达到一定大小后，将不会再向该segment写数据，broker会创建新的segment

# 分布与复制

- Topic中的分区将分布到kafka集群中的一些服务器上，每台服务器负责处理自己分区的读写请求
- 为了满足容错要求，每个分区的数据可根据配置要求被复制到集群的其它服务器上
- 每个分区有一个leader和0到多个follower服务器
  - Leader负责这个分区的所有读写请求
  - Follower被动复制leader
  - 如果leader宕机，其中一个follower会自动被选举为新leader
  - 有多少个partitions就意味着有多少“leader”，kafka会将“leader”均衡的分散在每个服务器上，来确保整体性能的稳定

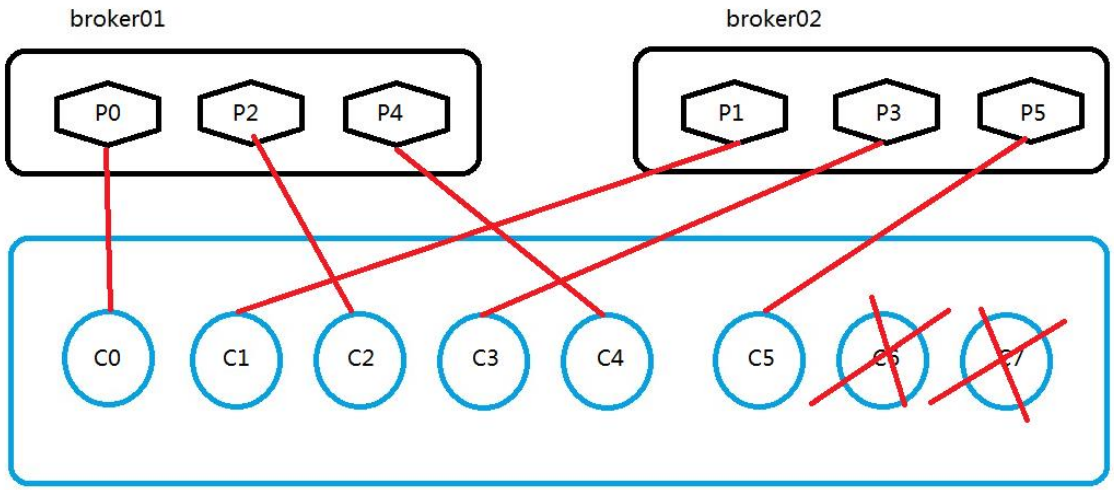
# 消费组

- Kafka中，每个consumer属于一个消费组consumer group；一个消费组中可以有一个或多个consumer
- 每个Topic可由多个消费组订阅
- 为了减小一个消费组中不同消费者之间的分布式协调开销，设定分区为最小的并行消费单位。发送到Topic的消息，只会被订阅此Topic的每个group中的一个consumer消费
  - 若所有消费者都属于同一组，则相当于队列模型，即一群消费者从一个队列中读取消息，同一消息只被一个消费者获取
  - 若消费者的组名各不相同，则相当于发布-订阅模型，即同一消息被广播到所有订阅的消费者



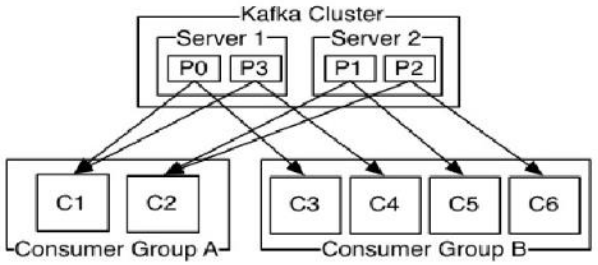
# 消费组成员个数与分区个数的关系

- 在一个消息组内，kafka确保每个分区只被分配到一个消费者上。这意味着，每个消费组中的有效消费者数量，一定小于或者等于主题所包含的分区数量



# 消息处理的顺序性和并发性

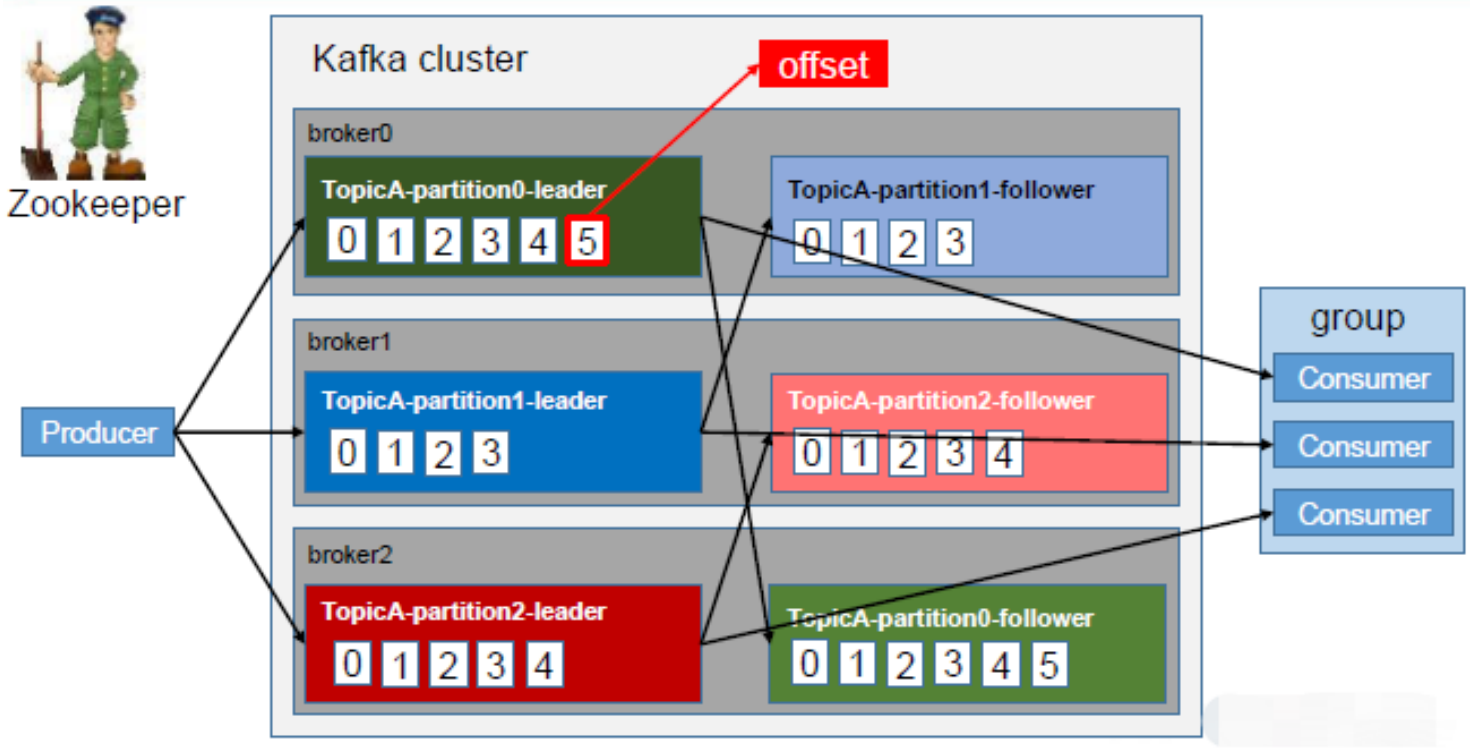
- 传统消息系统，服务器按照接收顺序存储消息，并按照同样的顺序向消费者分发消息。由于分发过程是异步的，消息抵达消费者的时间不可控
  - 当多个消费者进行并行处理时，消息的顺序性难以保证
  - 若只允许一个消费者消费，则没有了并发性
- Kafka更好地平衡了消息处理的顺序性和并发性
  - 在一个消费组内，kafka确保每个分区只被分配到一个消费者上，则该消费者是组内该分区的唯一读者，能够严格按照顺序获取消息
  - 同时，通过使用多分区，为同组多个消费者提供了并发处理能力
- kafka仅在同一分区内保证消息的顺序性，不保证跨分区的顺序性



# Broker的无状态性

- 消费者使用拉取pull方式从Broker获取数据，消费者会记录每个分区的消费进度（即偏移量）
- Broker不保存消息消费者的状态，由消费者自己保存
  - 大大简化了Broker的设计实现
- 无状态性使得Broker难以明确消息什么时候能够被删除，因此消息保留一定时间后才被删除。旧数据删除策略例：
  - 基于时间：log.retention.hours=168
  - 基于大小：log.retention.bytes=1073741824
- 消费者可以rewind back到任意位置重新进行消费
  - 当消费者故障时，可以选择最小的offset，重新获取数据

# kafka工作机制总结



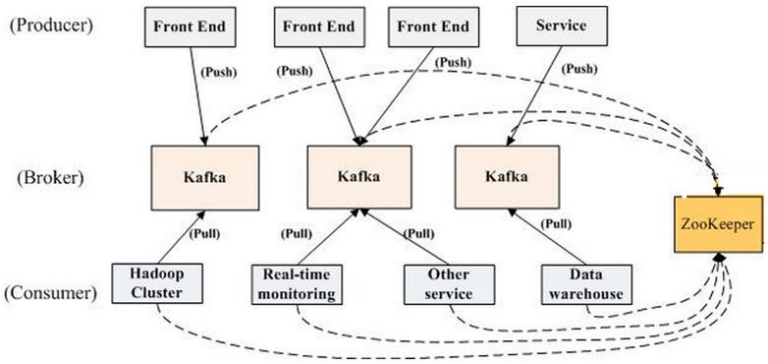

https://www.cnblogs.com/yxym2016/p/13413619.html

# 是否达到主要设计目标？

- 消息持久化：以时间复杂度为O(1)的方式提供消息持久化能力，即使对TB级以上的数据也能保证常数时间复杂度的访问性能
- 高吞吐：在廉价的商用机器上也能支持单机每秒10万条以上的吞吐量
- 分布式：支持消息分区以及分布式消费，并保证分区内的消息顺序
- 跨平台：支持不同技术平台的客户端（如Java、PHP、Python等）
- 实时性：支持实时数据处理和离线数据处理
- 伸缩性：支持水平扩展

# 基本使用方式

- 配置并启动Zookeeper集群
- 配置并启动Kafka集群
- 创建Topic
- 编写producer和consumer程序并运行
- 代码例：




```
ProducerConfig config = new ProducerConfig(props);
producer = new Producer<String, String>(config);
KeyedMessage<String, String> data =
     new KeyedMessage<String, String>(topic, msg);
producer.send(data);
```

# 四个核心API

- Producer API：用于应用程序发布消息到1个或多个topic
- Consumer API： 用于应用程序订阅一个或多个topic，并处理产生的消息
- Streams API ：应用程序可使用它构建流处理器，从1个或多个topic消费输入流，并生产一个输出流到1个或多个输出topic，有效地将输入流转换到输出流
- Connector API：主要用来与其它中间件系统建立流式通道，将topic连接到现有的应用程序或数据系统，如与QL数据库系统

# RPC or 消息？

- A进程使用B进程的服务
  - A需要基于B的实时响应结果进行下一步操作
    - ➔同步RPC：如登录
  - A不关心B的结果，或B的执行非常耗时
    - ➔异步RPC
    - ➔消息服务：如：登录后奖励积分
  - 松耦合A、B间关系
    - ➔消息服务：如浏览网页、搜索、点击等活动的信息被各个服务器（A）发布到kafka的topic中，然后消费者（B）通过订阅这些topic来做实时的监控分析
  - A系统的输出能力远远大于B系统的输入能力
    - ➔消息服务：如需要限流削峰时

# 通信主要内容

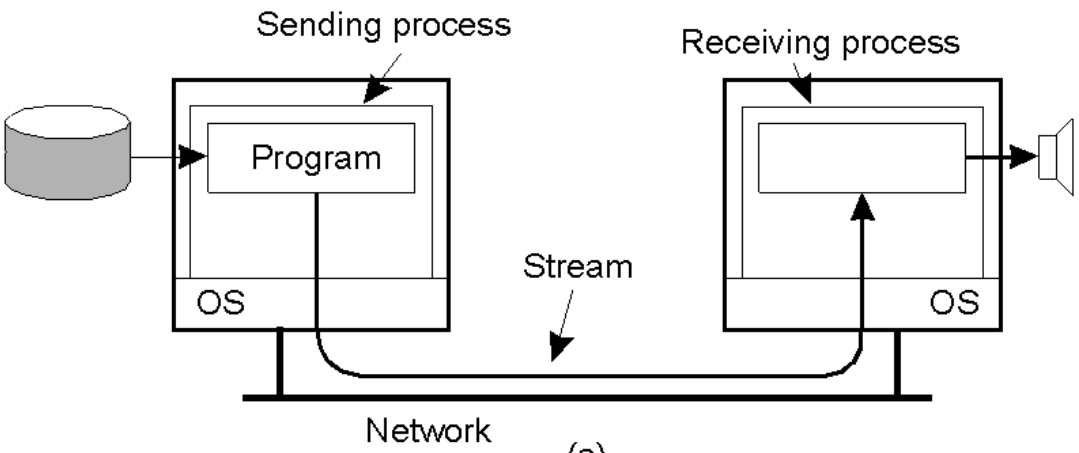
- 分层通信协议
- 远程过程调用和远程对象调用
- 面向消息的通信
- 面向流的通信
- 多播通信

# 面向流的通信

- 媒体（media）：
  - 媒体是指传播信息的媒介。它是指人借助用来传递信息与获取信息的工具、渠道、载体、中介物或技术手段
  - 例：图像格式:GIF、JPEG、TIF
- 连续型媒体(continuous representation media)
  - 在数据项之间存在时间关系
  - 例：CD:采样值/1/44100秒，VCD:祯/1/30秒
- 离散型媒体

# 数据流

- 数据流（stream）
  - 异步传输: delay∈(0, ∞) ; delay(传输延迟)
  - 同步传输: delay ∈ (0,max]
  - 等时(isochronous)传输: delay ∈ [min, max]
- 复杂数据流
  - 由多个子数据流（substream）组成
  - 例：DVD(（左声道，右声道），视频，字幕)



(a)

# 流的服务质量Qos

- 非功能需求一般称为服务质量QoS
- 流的QoS描述了低层分布式系统及网络在确保传输质量方面的需求
- QoS属性例：
  - 数据传输要求的比特率
  - 创建会话的最大延时
  - 端到端最大延时
  - 最大往返延时
  - 误码率
  - 丢包率

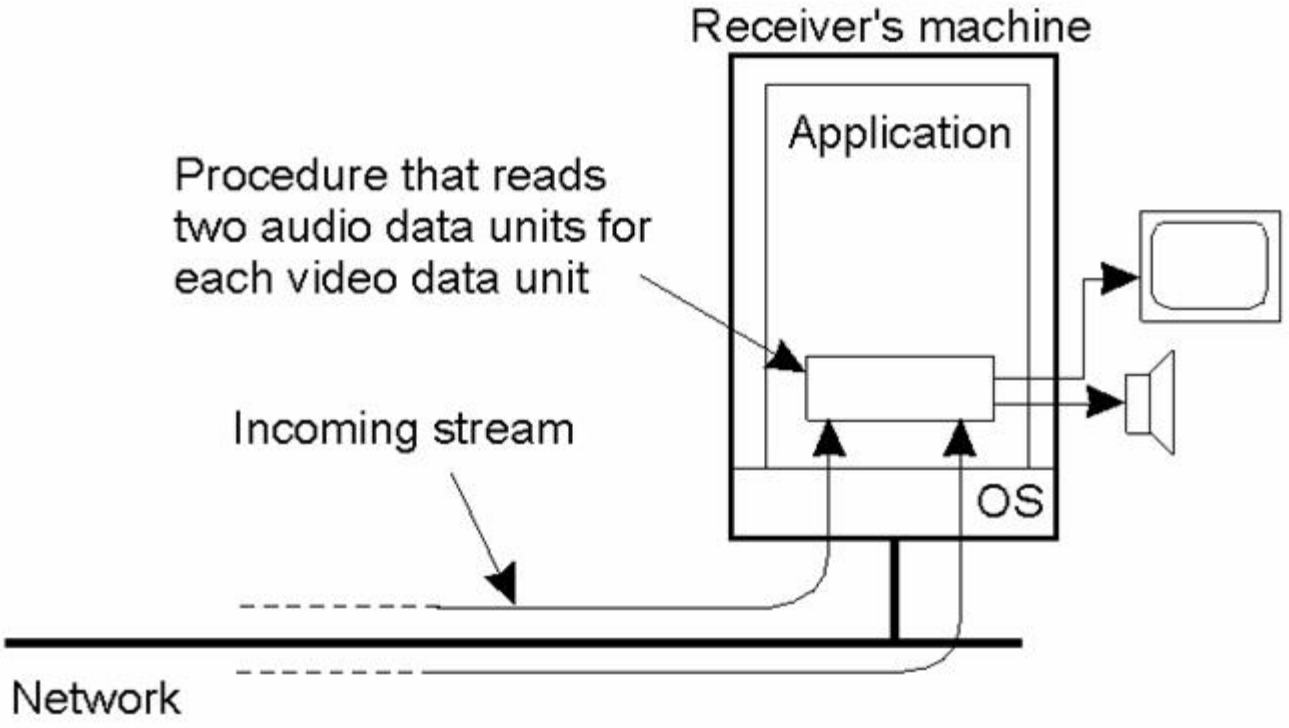
有很多支持QoS的方法

# 流同步

- 多媒体系统中，常常需要不同的流互相之间保持同步
- 如离散数据流与连续数据流之间保持同步
  - 幻灯片演示与音频
- 如连续流之间的同步
  - 放映影片时的视频流和音频流

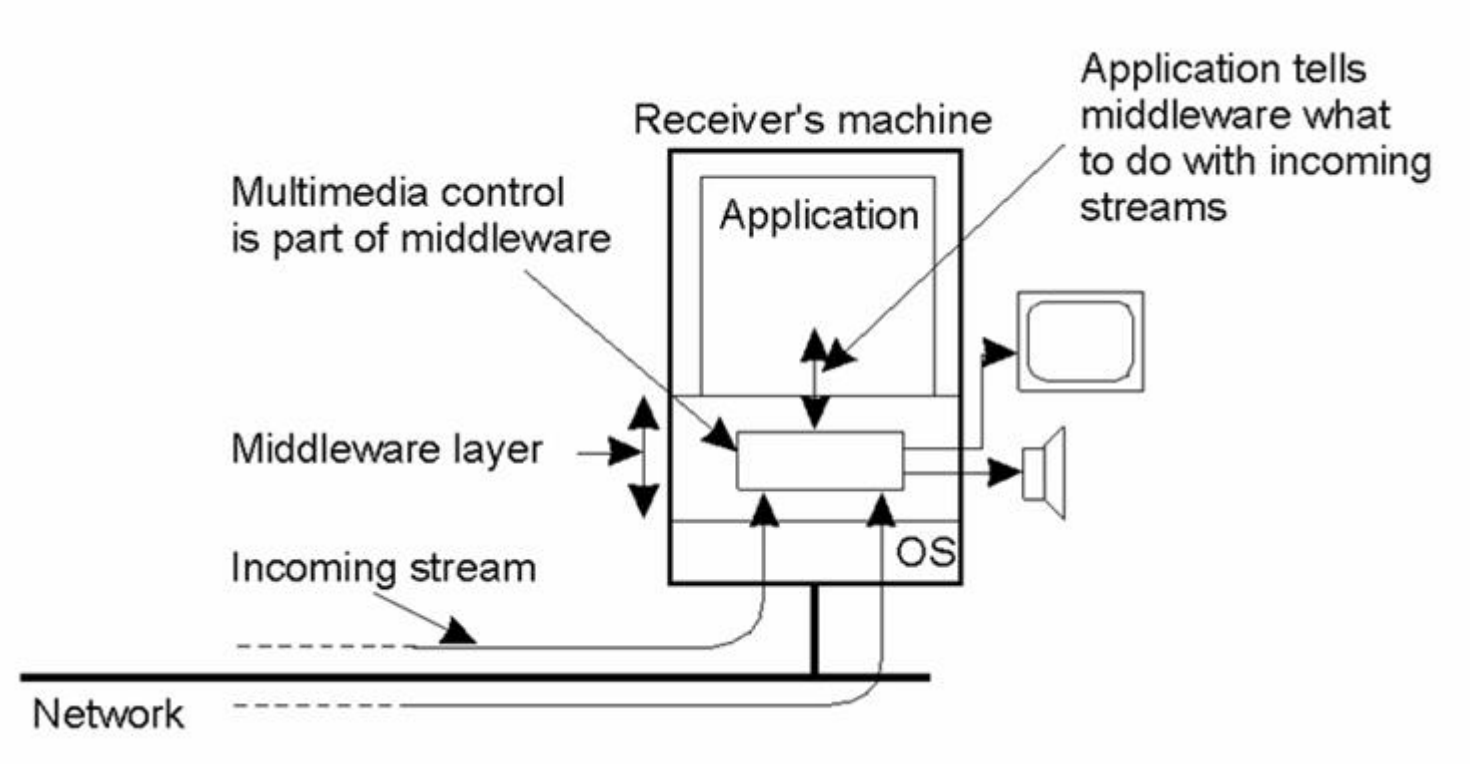
# 同步机制例1

- 在数据单元层上的显式同步
  - 读写进程：读流数据单元，同步写



# 同步机制例2

- 由高层接口支持的同步
  - 多媒体中间件提供控制接口
  - 用户自定义程序：检查同步，调整流速



# 通信主要内容

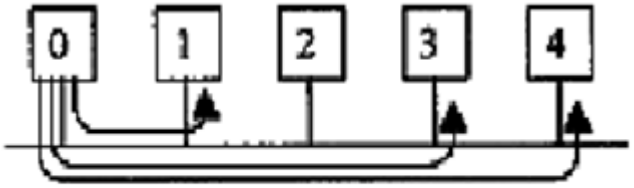
- 分层通信协议
- 远程过程调用和远程对象调用
- 面向消息的通信
- 面向流的通信
- 多播通信

# 多播通信

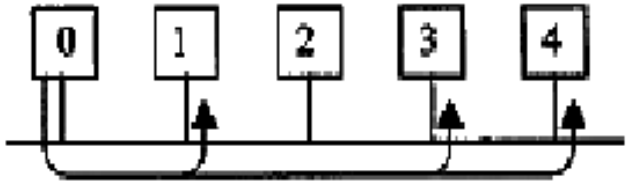
- 组：由系统或用户确定的若干个进程的集合
  - 组的成员籍（membership）
- 通信方式：
  - 点到点通信（point-to-point）:单播（unicast）
  - 一到多通信(one-to-many) : 多播（multicast）、广播（broadcast）


(a) (b)

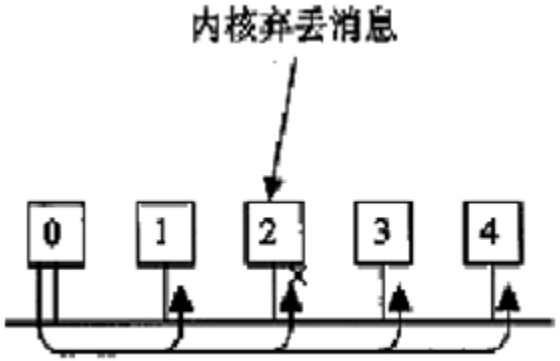
# 单点、多播和广播




3 单点传送




1 多播传送




2 广播传送

# “通信”总结

- 进程间通信功能是所有分布式系统所必须的
- 传统的分布式系统通过基于传输层提供的低层消息来进行通信，如TCP/IP 套接字
- 分布计算环境（中间件）提供更高层次的通信支持
  - RPC、RMI
  - 基于消息的通信
- RPC、RMI等主要提供瞬时同步通信功能
- 但在一些应用场合，面向消息的通信更方便一些
  - 瞬时和持久，同步和异步
  - 例：socket、MPI、kafka

# “通信”总结（2）

- 面向流的通信
  - 服务质量QoS
  - 流的同步
- 多播通信
  - 从发送方到多个接收方