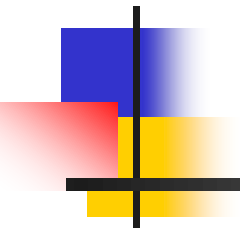
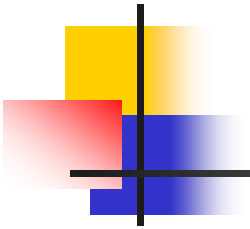
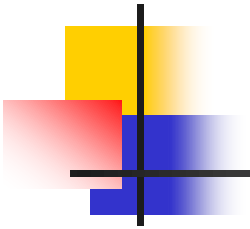


第2章　系统架构



内容提要

- 引言
- 物理模型
- 架构模型
- 架构风格
- 基础模型
- 总结



引言

- 不同类型的分布式系统表现出共同的基本特性及设计问题。

- 模型试图对分布式系统的相关方面给出抽象、简化但一致的描述。

引言

物理模型

- 考虑组成系统的计算机和设备的类型以及它们的互连
  - 不涉及特定的技术细节
- 物理模型是描述系统最显式的方法
  - 它从计算机和其他设备（例如移动电话）及其互联的网络方面考虑系统的硬件组成

引言

架构模型

- 从系统的计算元素执行的计算和通信方面来描述系统
  - 数据和计算任务在系统的物理节点间的分布
  - 构成系统各部分（**components, computers, procedures**)的位置、角色和它们之间的关系
    - 客户/服务器结构
    - 对等结构
    - 客户/服务器模型的变种
- 评估分布式系统的性能、可靠性、可扩展性以及其它的特性，估计可能存在的问题

引言

基础模型

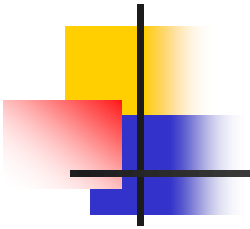
采用抽象的观点描述分布式系统的某个方面

- 交互模型
  - 处理消息发送的性能问题，解决在分布式系统中设置时间限制的难题。
- 故障模型
  - 试图给出进程和信道故障的一个精确的约定。
  - 它定义了什么是可靠通信和正确的进程。
- 安全模型
  - 讨论对进程的和信道的各种可能的威胁。
  - 引入了安全通道的概念，它可以保证在存在各种威胁的情况下通信的安全。

引言

分布式系统的困难和挑战

- 使用模式的多样性
- 系统环境的多样性
- 内部问题：
  - 非同步的时钟
  - 数据修改的不一致性
  - 系统中单个部件的软件硬件故障
- 外部问题
  - 数据在传输过程中存在着对私密性、完整性的攻击

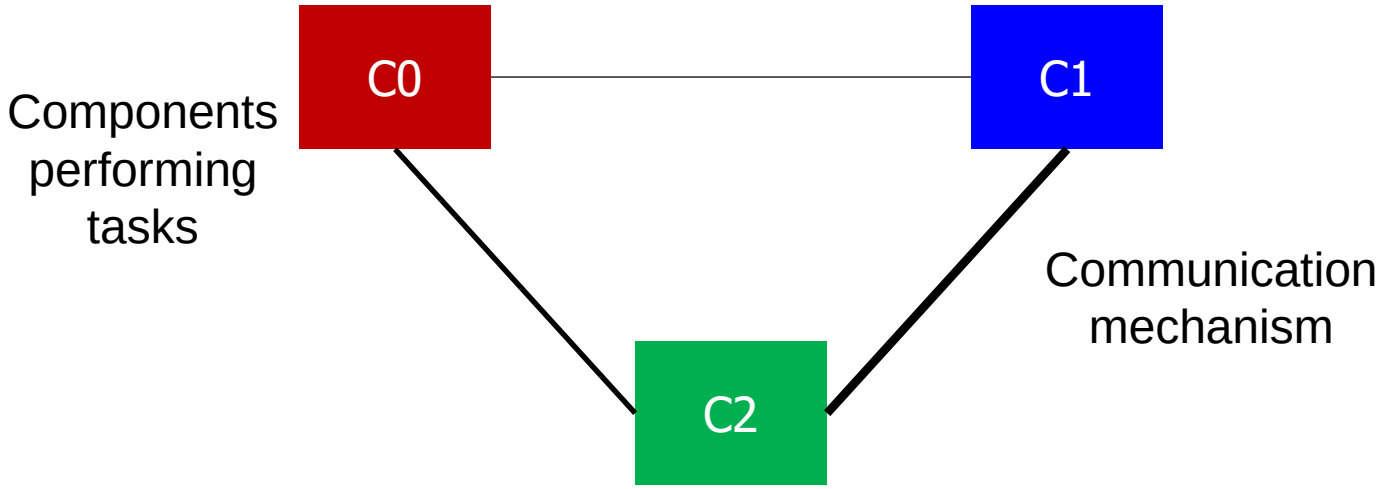


内容提要

- 引言
- 物理模型
  - 早期的分布式系统
  - 互联网规模的分布式系统
  - 当代的分布式系统
- 架构模型
- 架构风格
- 基础模型
- 总结

最小物理模型示意图

- A distributed system is simply a collection of hardware or software components that communicate to solve a complex problem
- Each component performs a "task"



物理模型

- 物理模型是从计算机和所用网络技术的特定细节中抽象出来的分布式系统底层硬件元素的表示
- 基线物理模型
  - 分布式系统被定义成其位于联网计算机上的硬件或软件组件，仅通过消息传递进行通信和协调动作的系统
- 最小物理模型
  - 最小物理模型是一组可扩展的计算机结点，这些结点通过计算机网络相互连接进行所需的消息传递
- 在这个基线模型之上，我们能有效地识别出三代分布式系统

早期的分布式系统

- 出现在20 世纪70 年代晚期和80 年代早期
  - 随着局域网技术如以太网的出现而出现
- 一般由通过局域网互联的10 -100个结点组成
  - 它们与互联网的连接有限并支持少量的服务(如共享的本地打印机和文件服务器以及电子邮件和互联网上的文件传输)
- 单个系统大部分是同构的
  - 开放性不是主要的问题
  - 服务质量提供还很少，是围绕这样的早期系统开展的很多研究中的一个焦点。

互联网规模的分布式系统

- 20 世纪90 年代更大规模的分布式系统开始出现
  - 互联网惊人的发展
  - 例如， Google 搜索引擎在1996 年第一次发布。在这样的系统中，底层物理基础设施由一个可扩展的结点集合的物理模型组成，这些结点通过一个网络相互连接。
- 这些系统利用了互联网提供的基础设施实现了全球化
  - 它们包含大量的结点并且为全球化组织提供分布式系统服务，也跨组织提供分布式系统服务
  - 在这样的系统中，从网络、计算机架构、操作系统、所采用的语言和所涉及的开发团队方面来说，异构性很突出
  - 这导致开放标准和相关的中间件技术(如CORBA 和最近的Web服务等)的重要性不断增加
  - 在这样的全球化系统中，采用了额外的服务来提供端到端的服务质量特性。

当代的分布式系统（1/2)

- 在上述系统中
  - 结点通常是台式机，因此是相对静态的(即在一段时间里停留在一个物理位置)
  - 分立的(没有嵌入到其他物理实体内)
  - 自治的(就物理基础设施而言，很大程度上独立于其他计算机)
- 移动计算的关键趋势促进了物理模型的进一步发展:
  - 移动计算的出现导致这样的物理模型
  - 在这种模型中，像笔记本电脑或智能手机这样的结点可以从一个位置移动到另一个位置
  - 它还导致了对诸如服务发现这样的新增功能的需要和对自发互操作的支持。

当代的分布式系统（2/2)

- 泛在计算的出现导致了体系结构从分立结点型转向计算机被嵌入到日常物品和周围环境中(例如，嵌入在洗衣机中或更一般地嵌入在智能家庭设备中)。
- 云计算特别是集群体系结构的出现导致了从自治结点完成给定任务转向→组结点一起提供一个给定的服务(例如，由 Google 提供的搜索服务)。

- 最终的结果是出现一个异构性有很大增加的物理体系结构
  - 例如，从泛在计算中使用的最小的嵌入式设备到网格计算中的复杂的计算元素。
  - 这些系统部署不断增加的不同的网络技术，并提供广泛的应用和服务。这样的系统可能涉及成百上千个结点。

系统之系统

- 超大规模(Ultra- Large- Scale ， ULS)的分布式系统[www.sei.cmu.eduJ报告收集了现代分布式系统的复杂性，把这样的(物理)体系结构叫做系统之系统
  - 反映了与将互联网看成网络的网络相同的观点

- 系统之系统可以被定义成一个复杂系统
  - 它由一系列子系统组成
  - 这些子系统本身也是系统
  - 它们一起完成一个或多个特定的任务。

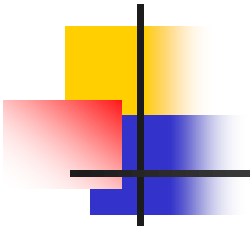
系统之系统的一个例子

- 考虑一个用于洪水预测的环境管理系统。
  - 部署了传感网来监视与河流、冲积平原、潮沙效应等相关的不同的环境参数的状态。
  - 这可以通过在集群计算机上运行模拟程序，与负责预测洪水可能性的系统藕合在一起。
  - 可以建立其他系统用于维护和分析历史数据或通过移动电话给关键的利益共享者提供早期报警。

三代分布式系统

分布式系统	早　期	互联网规模	当　代
规模	小	大	超大
异构性	有限（相对同构的配置）	从平台、语言和中间件方面来说都较大	维度增加，包括体系结构中完全不同的风格
开放性	不属于优先考虑的事	相当重要，引入一系列标准	重要的研究挑战，已有的标准不能包含复杂系统
服务质量	起步阶段	相当重要，引入一系列服务	重要的研究挑战，已有的服务不能包含复杂系统

云控车联网



内容提要

- 引言
- 物理模型
- 架构模型
  - 通信实体是什么?
  - 使用什么通信范型?
  - 扮演什么(可能改变的)角色，承担什么责任?
  - 怎样被映射到物理基础设施上(它们被放置在哪里)?
- 架构风格
- 基础模型
- 总结

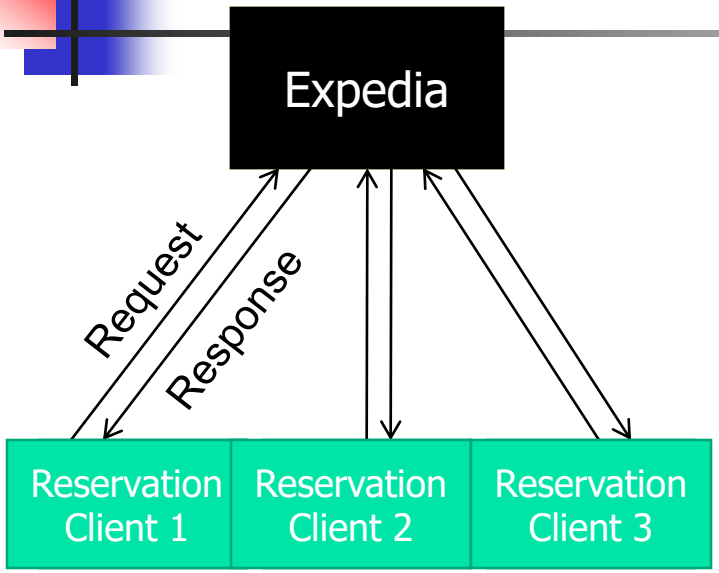
架构模型

- 一个系统的体系结构是用独立指定的组件以及这些组件之间的关系来表示的结构
  - 整体目标是确保结构能满足现在和将来可能的需求。主要关心的是系统可靠性、可管理性、适应性和性价比。
  - 建筑物的体系结构设计有类似的方面，不仅要决定它的外观，还决定其总体结构和体系结构风格(哥特式、新古典式、现代式)
  - 并为设计提供一个一致的参考框架。
- 分布式系统采用的几种主要的架构模型
  - 即分布式系统的体系结构风格
  - 客户-服务器模型、对等方法、分布式对象、分布式组件、分布式基于事件的系统以及这些风格之间的不同之处奠定基础

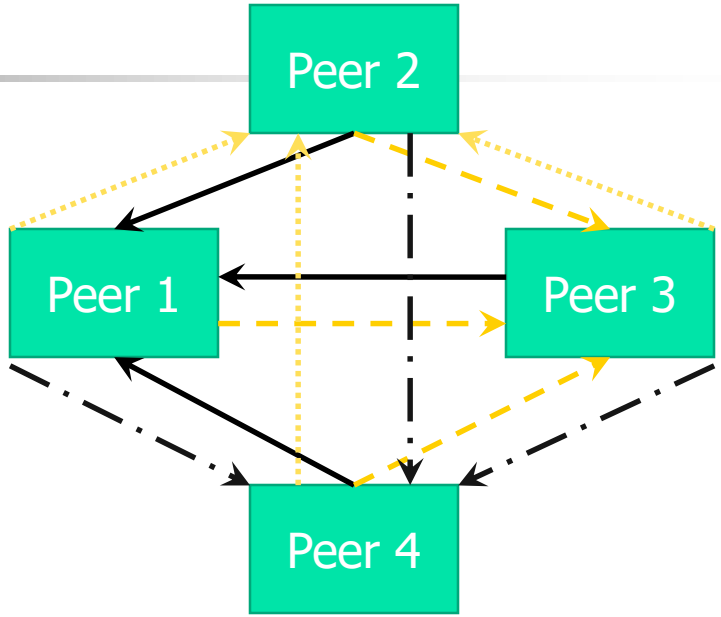
架构相关的四个关键问题

- 为了理解一个分布式系统的基础构建块，有必要考虑下面四个关键问题：
  - 在分布式系统中进行通信的实体是什么?
  - 它们如何通信，特别是使用什么通信范型?
  - 它们在整个体系结构中扮演什么(可能改变的)角色，承担什么责任?
  - 它们怎样被映射到物理分布式基础设施上(它们被放置在哪里)?

Bird's Eye View of Some Distributed Systems

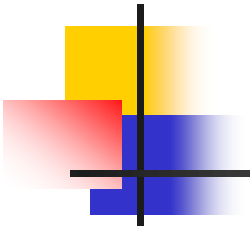


Google Search
Airline Booking



Bit-torrent
Skype

- How would one classify these distributed systems?

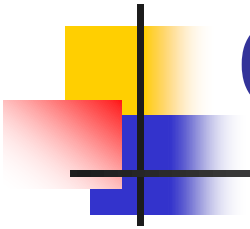


内容提要

- 引言
- 物理模型
- 架构模型
  - 通信实体是什么?
  - 使用什么通信范型?
  - 扮演什么(可能改变的)角色，承担什么责任?
  - 怎样被映射到物理基础设施上(它们被放置在哪里)?
- 架构风格
- 基础模型
- 总结

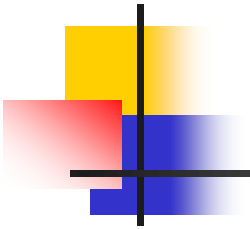
架构模型关键问题之一

- 为了理解一个分布式系统的基础构建块，有必要考虑下面四个关键问题：
  - 在分布式系统中进行通信的实体是什么?
  - 它们如何通信，特别是使用什么通信范型?
  - 它们在整个体系结构中扮演什么(可能改变的)角色，承担什么责任?
  - 它们怎样被映射到物理分布式基础设施上(它们被放置在哪里)?



Communicating Entities

- What entities are communicating in a DS?
  - System-oriented entities
    - Processes
    - Threads
    - Nodes
  - Problem-oriented entities
    - Objects (in *object-oriented programming* based approaches)
    - 对象、组件、服务



内容提要

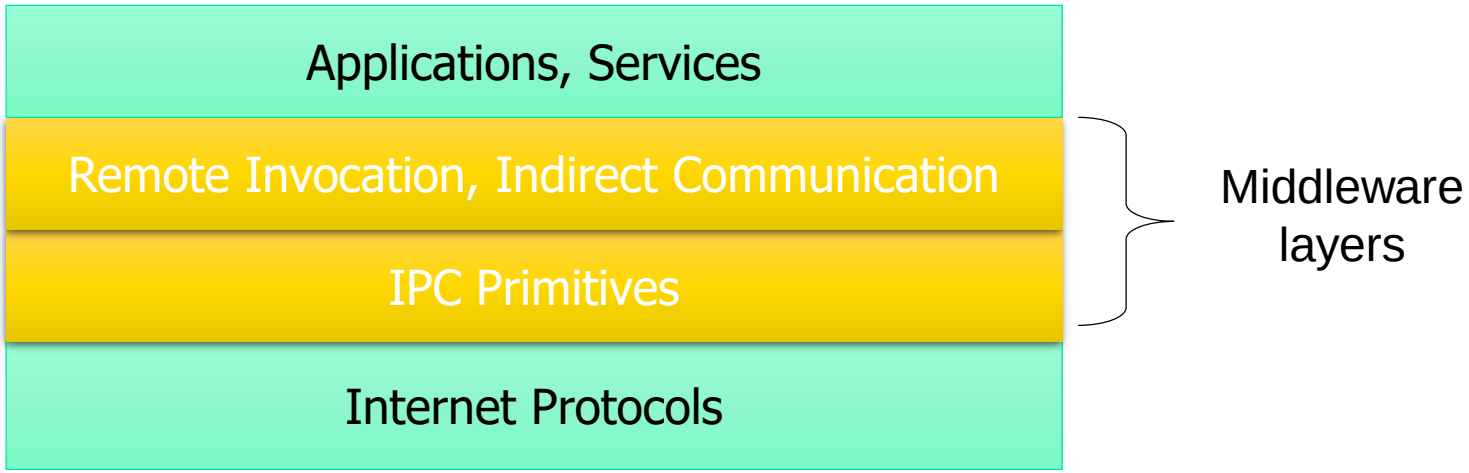
- 引言
- 物理模型
- 架构模型
  - 通信实体是什么?
  - 使用什么通信范型?
  - 扮演什么(可能改变的)角色，承担什么责任?
  - 怎样被映射到物理基础设施上(它们被放置在哪里)?
- 架构风格
- 基础模型
- 总结

架构模型关键问题之二

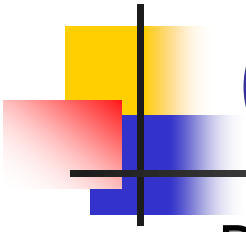
- 为了理解一个分布式系统的基础构建块，有必要考虑下面四个关键问题：
  - 在分布式系统中进行通信的实体是什么?
  - 它们如何通信，特别是使用什么通信范型?
  - 它们在整个体系结构中扮演什么(可能改变的)角色，承担什么责任?
  - 它们怎样被映射到物理分布式基础设施上(它们被放置在哪里)?

Communication Paradigms

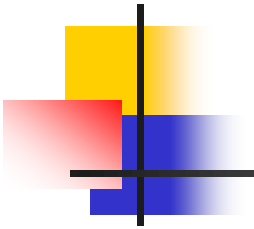
- Three types of communication paradigms
  - Inter-Process Communication (IPC)
  - Remote Invocation
  - Indirect Communication



Inter-Process Communication (IPC)



- Relatively low-level support for communication
  - e.g., Direct access to internet protocols (Socket API)
- Advantages
  - Enables seamless communication between processes on heterogeneous operating systems
    - Well-known and tested API adopted across multiple operating systems
- Disadvantages
  - Increased programming effort for application developers
    - Socket programming: Programmer has to explicitly write code for communication (in addition to program logic)
    - Space Coupling (Identity is known in advance): Sender should know receiver's ID (e.g., IP Address, port)
    - Time Coupling: Receiver should be explicitly listening to the communication from the sender



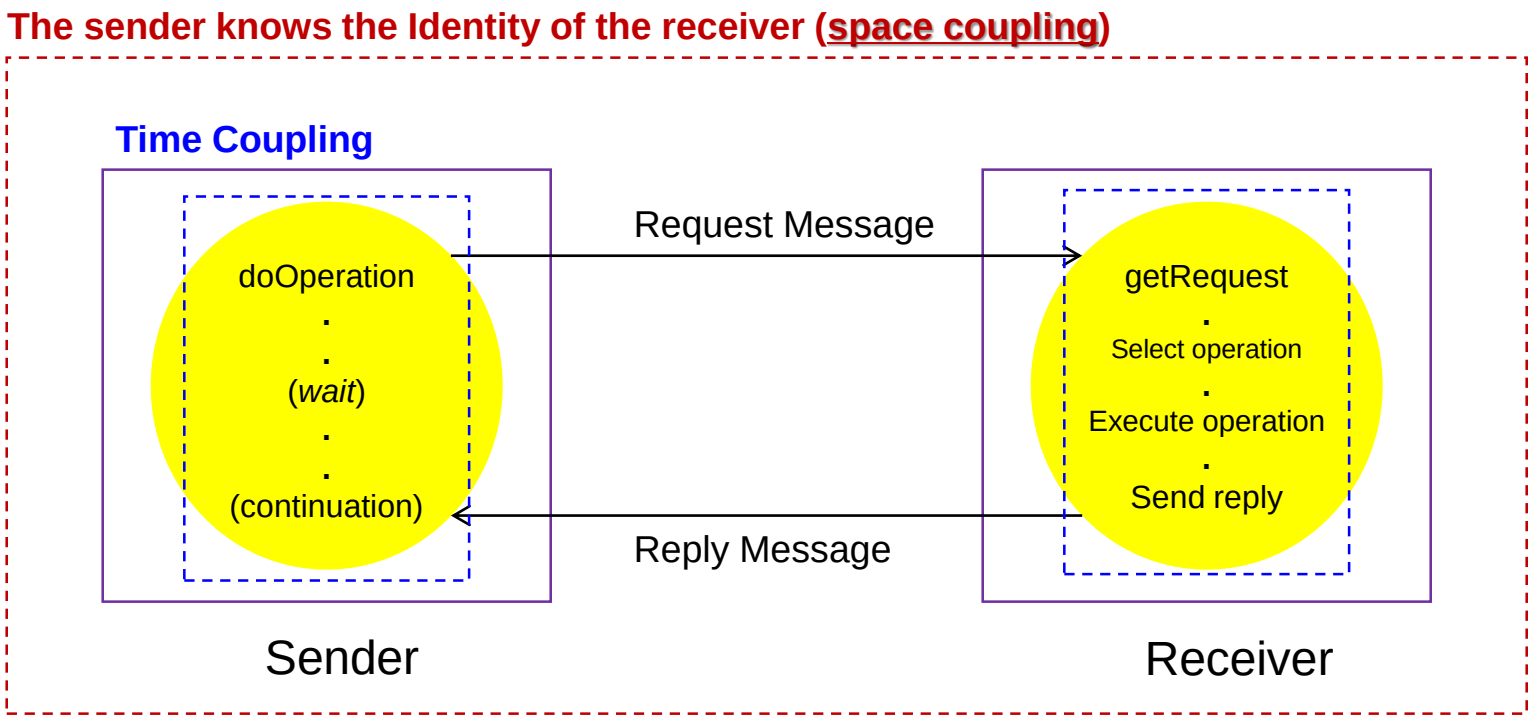
Remote Invocation

- An entity runs a procedure that typically executes on an another computer without the programmer explicitly coding the details for this remote interaction
  - A middleware layer will take care of the raw-communication
- Examples
  - Remote Procedure Call (RPC) – Sun's RPC (ONC RPC)
  - Remote Method Invocation (RMI) – Java RMI

Remote Invocation

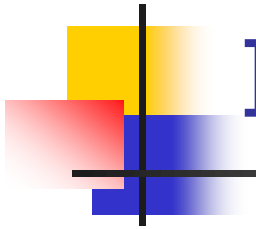
- Advantages:
  - Programmer does not have to write code for socket communication
- Disadvantages:
  - *Space Coupling*: Where the procedure resides should be known in advance
  - Time Coupling: On the receiver, a process should be explicitly waiting to accept requests for procedure calls

Space and Time Coupling in RPC and RMI



RMI strongly resembles RPC but in a world of distributed objects

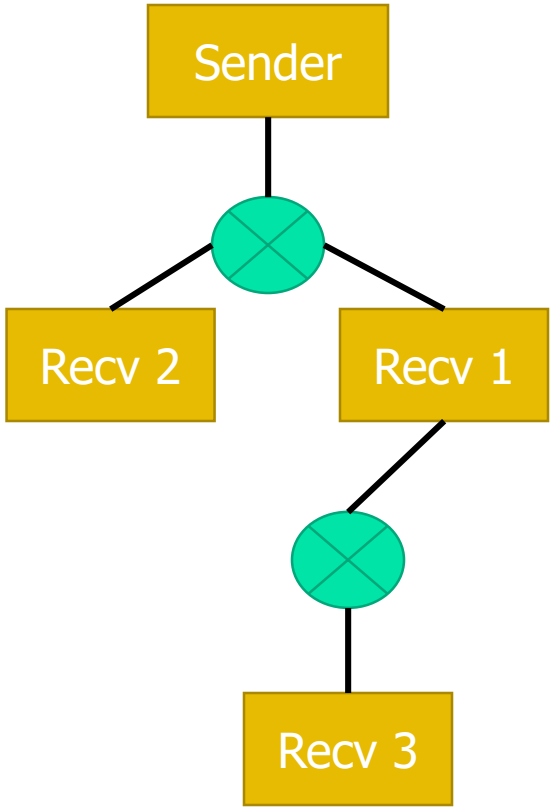
Indirect Communication Paradigm



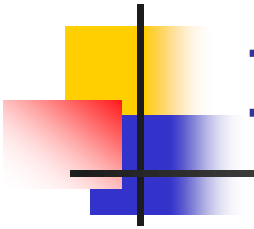
- Indirect communication uses middleware to:
  - Provide one-to-many communication
  - Some mechanisms eliminate space and time coupling
    - Sender and receiver do not need to know each other's identities
    - Sender and receiver need not be explicitly listening to communicate
- Approach used: Indirection
  - Sender → A middle-man → Receiver
- Types of indirect communication
  1. Group communication
  2. Publish-subscribe
  3. Message queues

1. Group Communication

- One-to-many communication
  - Multicast communication
- Abstraction of a group
  - Group is represented in the system by a *groupId*
  - Recipients join the group
  - A sender sends a message to the group which is received by all the recipients



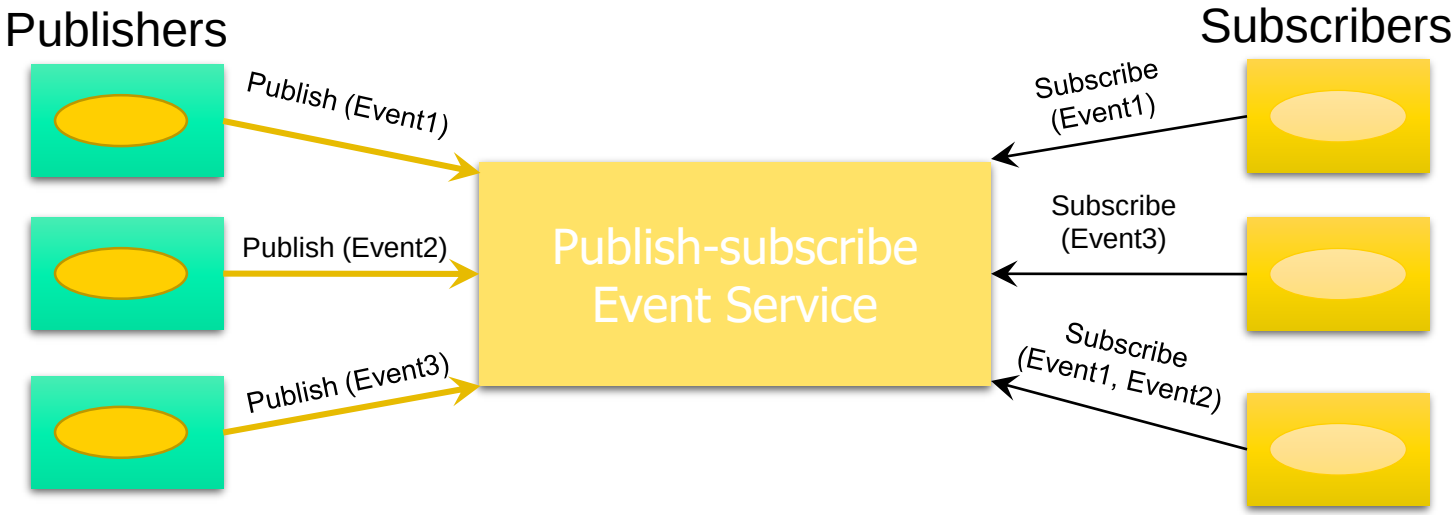
1. Group Communication (cont'd)



- Services provided by middleware
  - Group membership
  - Handling the failure of one or more group members
- Advantages
  - Enables one-to-many communication
  - Efficient use of bandwidth
  - Identity of the group members need not be available at all nodes
- Disadvantages
  - Time coupling

2. Publish-Subscribe

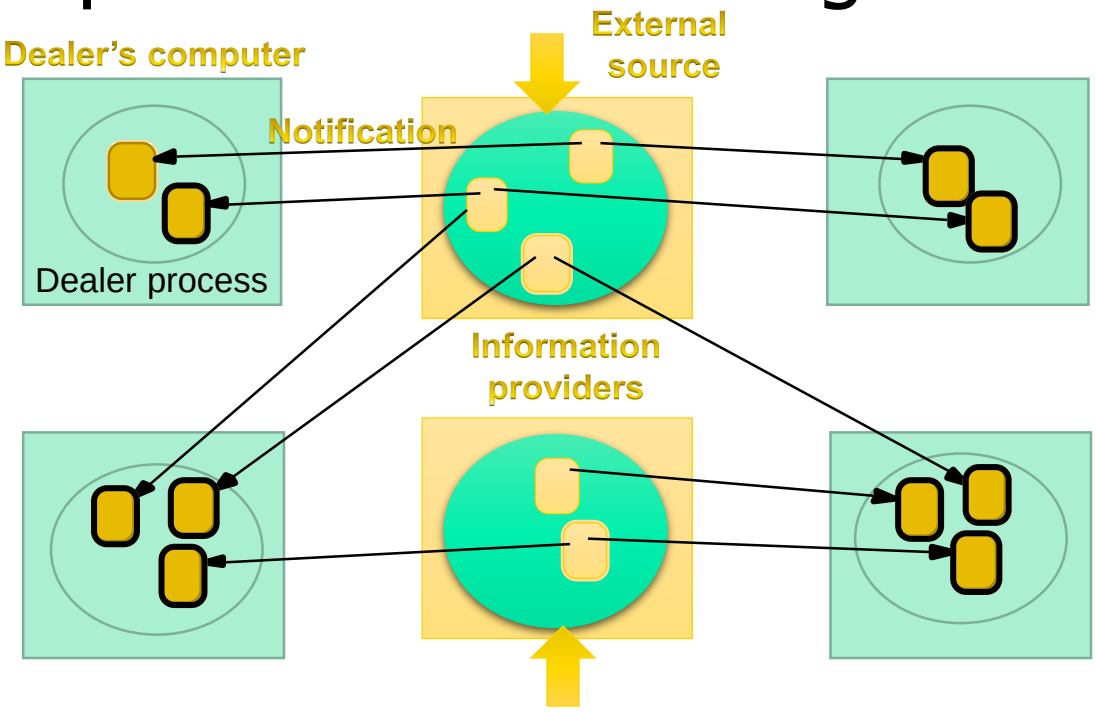
- An event-based communication mechanism
  - Publishers publish events to an event service
  - Subscribers express interest in particular events



- Large number of producers distribute information to large number of consumers

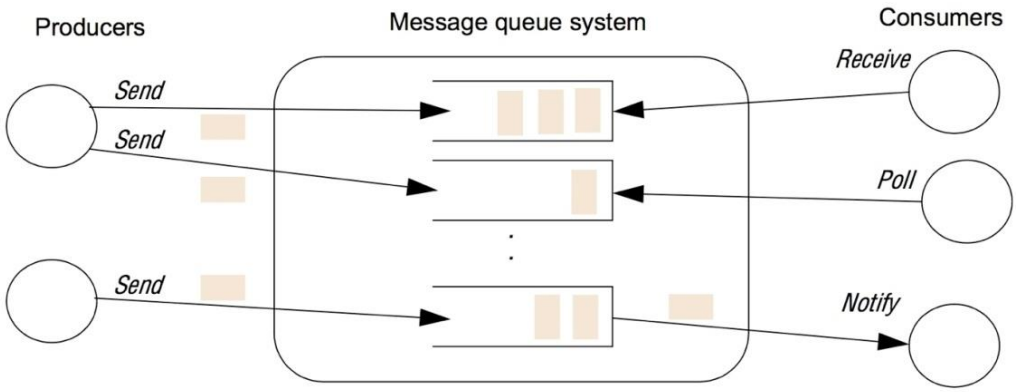
2. Publish-Subscribe (cont'd)

Example: Financial trading



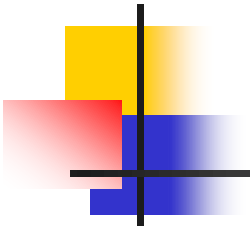
3. Message Queues

- A refinement of Publish-Subscribe where
  - Producers deposit the messages in a queue
  - Messages are delivered to consumers through different methods
  - Queue takes care of ensuring message delivery
- Advantages
  - Enables space decoupling
  - Enables time decoupling



Recap: Communication Entities and Paradigms

Communicating entities (what is communicating)		***Communication Paradigms (how they communicate)***		
System-oriented	*Problem-oriented*	*IPC*	*Remote Invocation*	*Indirect Communication*
• Nodes • Processes • Threads	• Objects	• Sockets	• RPC • RMI	• Group communication • Publish-subscribe • Message queues



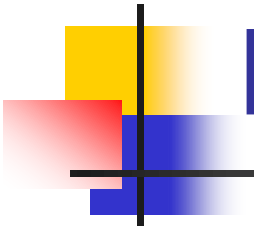
内容提要

- 引言
- 物理模型
- 架构模型
  - 通信实体是什么?
  - 使用什么通信范型?
  - 扮演什么(可能改变的)角色，承担什么责任?
  - 怎样被映射到物理基础设施上(它们被放置在哪里)?
- 架构风格
- 基础模型
- 总结

架构模型关键问题之三

- 为了理解一个分布式系统的基础构建块，有必要考虑下面四个关键问题：
  - 在分布式系统中进行通信的实体是什么?
  - 它们如何通信，特别是使用什么通信范型?
  - 它们在整个体系结构中扮演什么(可能改变的)角色，承担什么责任?
  - 它们怎样被映射到物理分布式基础设施上(它们被放置在哪里)?

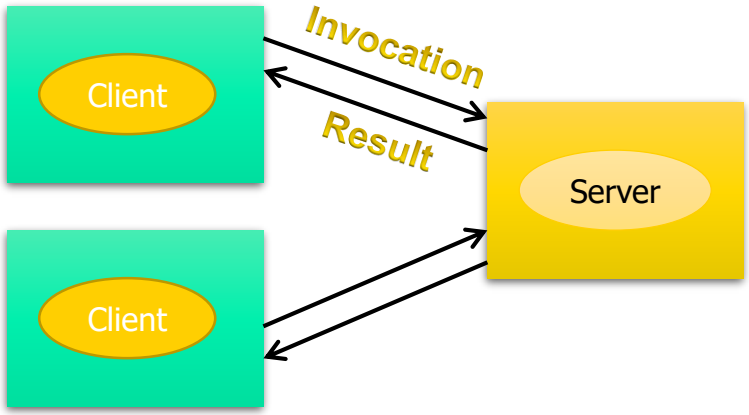
Roles and Responsibilities

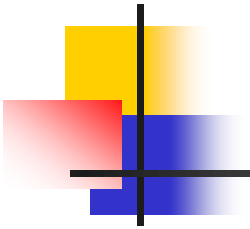


- In DS, communicating entities take on roles to perform tasks
- Roles are fundamental in establishing overall architecture
  - Question: Does your smart-phone perform the same role as Google Search Server?
- We classify DS architectures into two types based on the roles and responsibilities of the entities
  - Client-Server
  - Peer-to-Peer

Client-Server Architecture

- Approach:
  - Server provides a service that is needed by a client
  - Client requests to a server (invocation), the server serves (result)
- Widely used in many systems
  - e.g., DNS, Web-servers



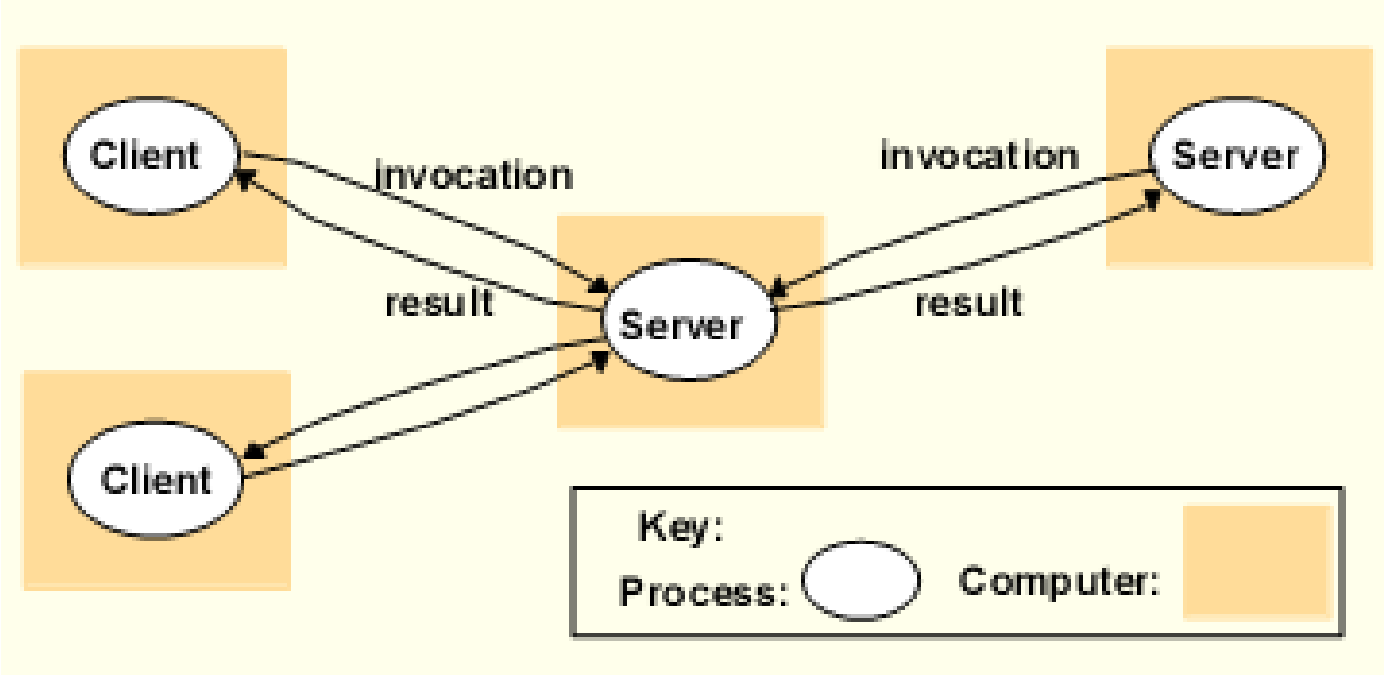


Client-Server Architecture: Pros and Cons

- Advantages:
  - Simplicity and centralized control
  - Computation-heavy processing can be offloaded to a powerful server
    - Clients can be “thin”
- Disadvantages
  - Single-point of failure at server
  - Scalability

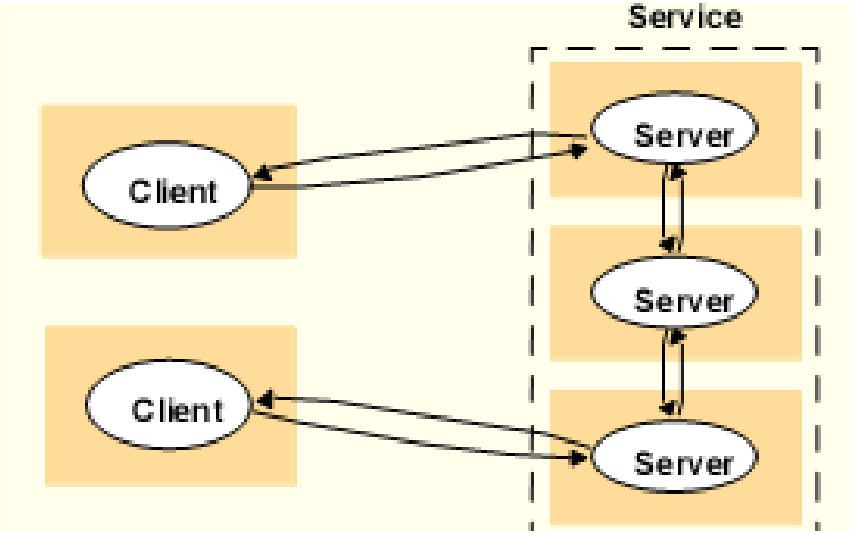
client/server ---单服务器

- 历史上最重要的结构之一，是**Internet**应用最常见的结构。



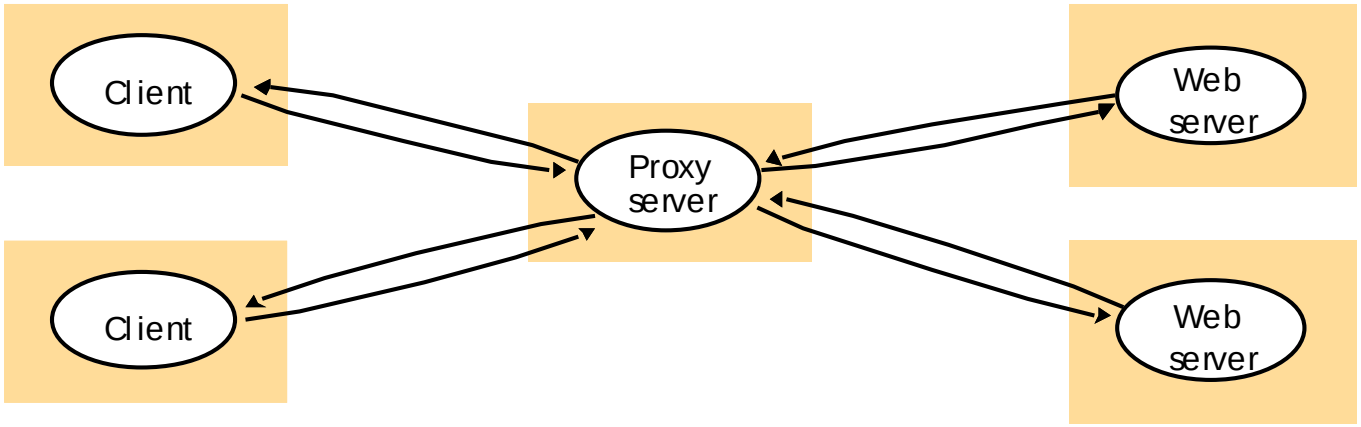
client/server --- 服务器组

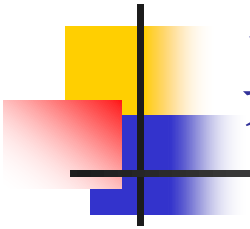
- 将不同的服务对象在不同的机器上实现
  - **e.g. Web**
- 在几个主机上维护副本服务
  - **e.g. Sun NIS Service**



client/server ---代理服务器和缓存

- 缓存（**Cache**）
  - 保存最近使用过的数据，可以在本地缓存，也可以在代理服务器上做缓存。
  - 缓存可以减少不必要的网络传输，减少服 务器负担，还可以代理其它用户透过防火 墙访问服务器。





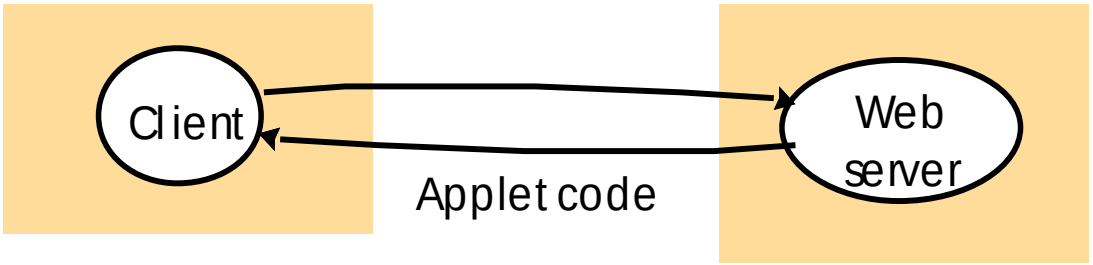
其它形式的变种

- 为什么要有一些变种？
  - 移动代码和移动代理
  - 用户需要低成本的计算机
  - 能方便的加入或移除移动设备

变种1---移动代码

- 将代码下载到客户端运行，可以提高交互的效率。

a) client request results in the downloading of applet code



b) client interacts with the applet

变种2---移动代理

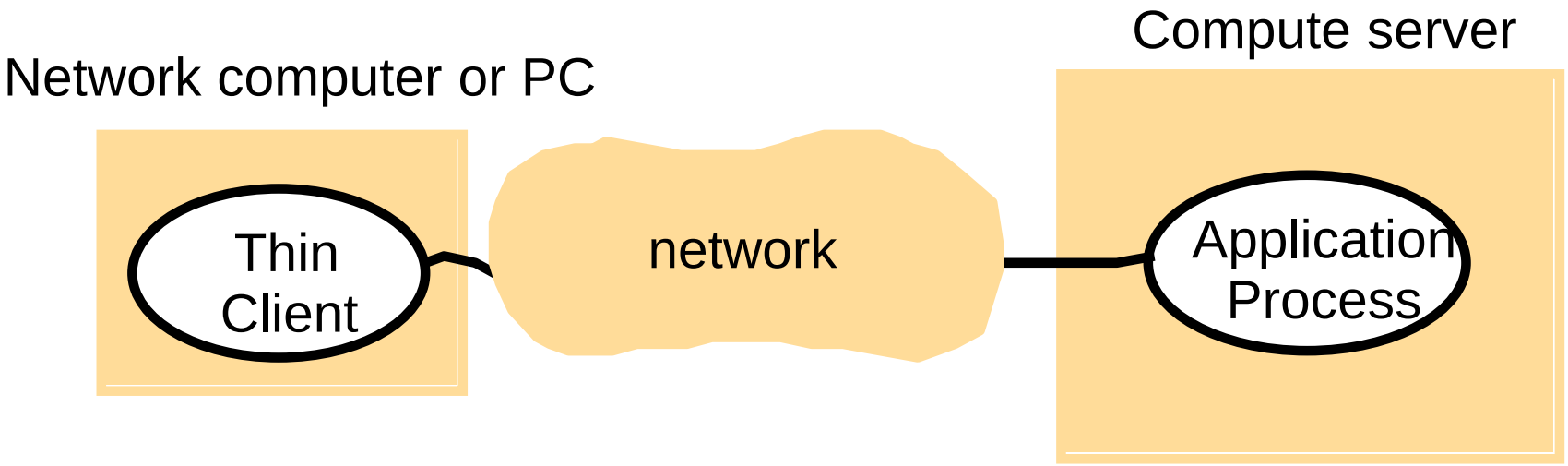
- 在网络上的服务器之间穿梭，并执行的代码。代替一些机器执行任务。例如：利用空闲计算机完成密集型计算，计算程序（连同数据）从一个机器移动到另一个机器，在目的机上运行。
  - 和**移动代码** 的区别：移动代码是服务器与客户端之间传递代码，后者是在服务器之间传递。
  - 和**远程调用**比较：减少数据传输，降低通信开销。安全威胁限制了它的使用。

变种3---网络计算机

- 本身不安装操作系统，而是从远程服务器上下载操作系统和应用软件
- 所有的应用数据和代码都存储在文件服务器上
- 用户可以移动

变种4---瘦客户

- 本地只是一个 **GUI**（图形用户界面），应用程序在远程计算机上执行。早期的大型机就有哑终端的概念，只不过那时大型机在哑终端在一个机房。现在可以是通过网络访问服务器。缺点：高延迟。



变种5--移动设备和自发互操作

- 设备带着软件组件在网络上移动
- 移动透明性
- 变化的连接性
- 自发互操作

自动驾驶　5G

Peer to Peer (P2P) Architecture

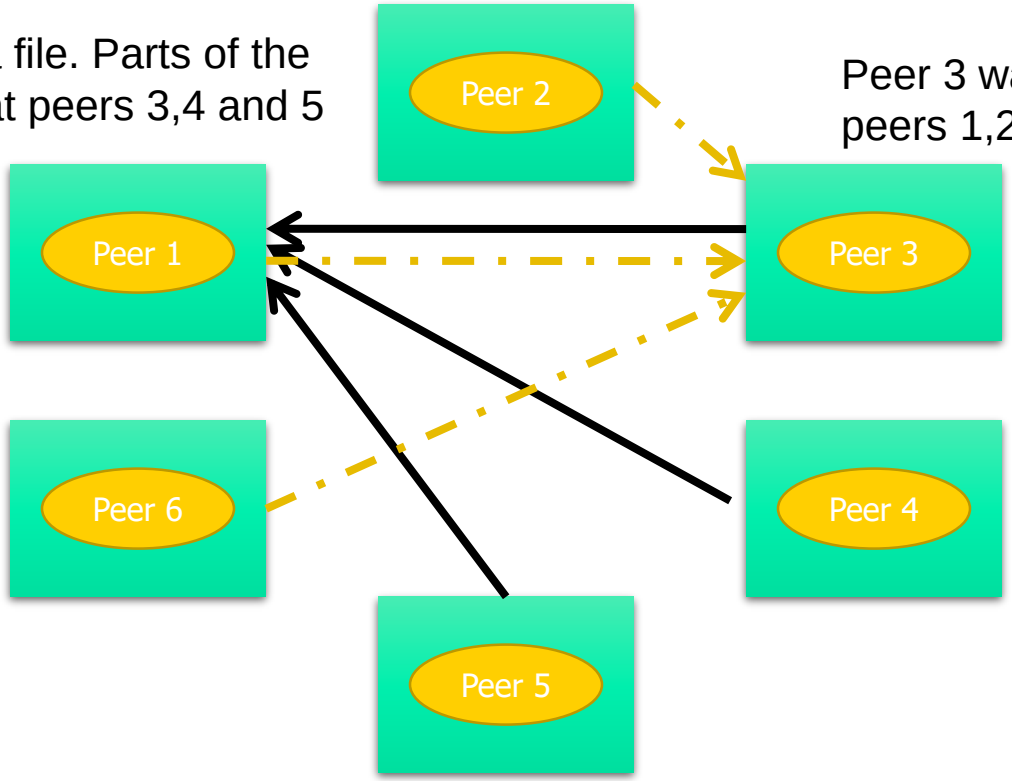
- In P2P, roles of all entities are identical
  - All nodes are peers
  - Peers are equally privileged participants in the application
- e.g.: Napster, Bit-torrent, Skype

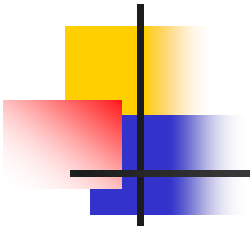
Peer to Peer Architecture

Example: Downloading files from bit-torrent

Peer 1 wants a file. Parts of the file is present at peers 3,4 and 5

Peer 3 wants a file stored at peers 1,2 and 6



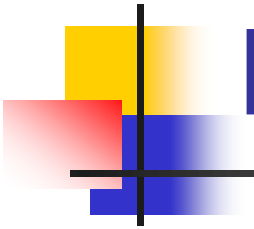


内容提要

- 引言
- 物理模型
- 架构模型
  - 通信实体是什么?
  - 使用什么通信范型?
  - 扮演什么(可能改变的)角色，承担什么责任?
  - 怎样被映射到物理基础设施上(它们被放置在哪里)?
- 架构风格
- 基础模型
- 总结

架构模型关键问题之四

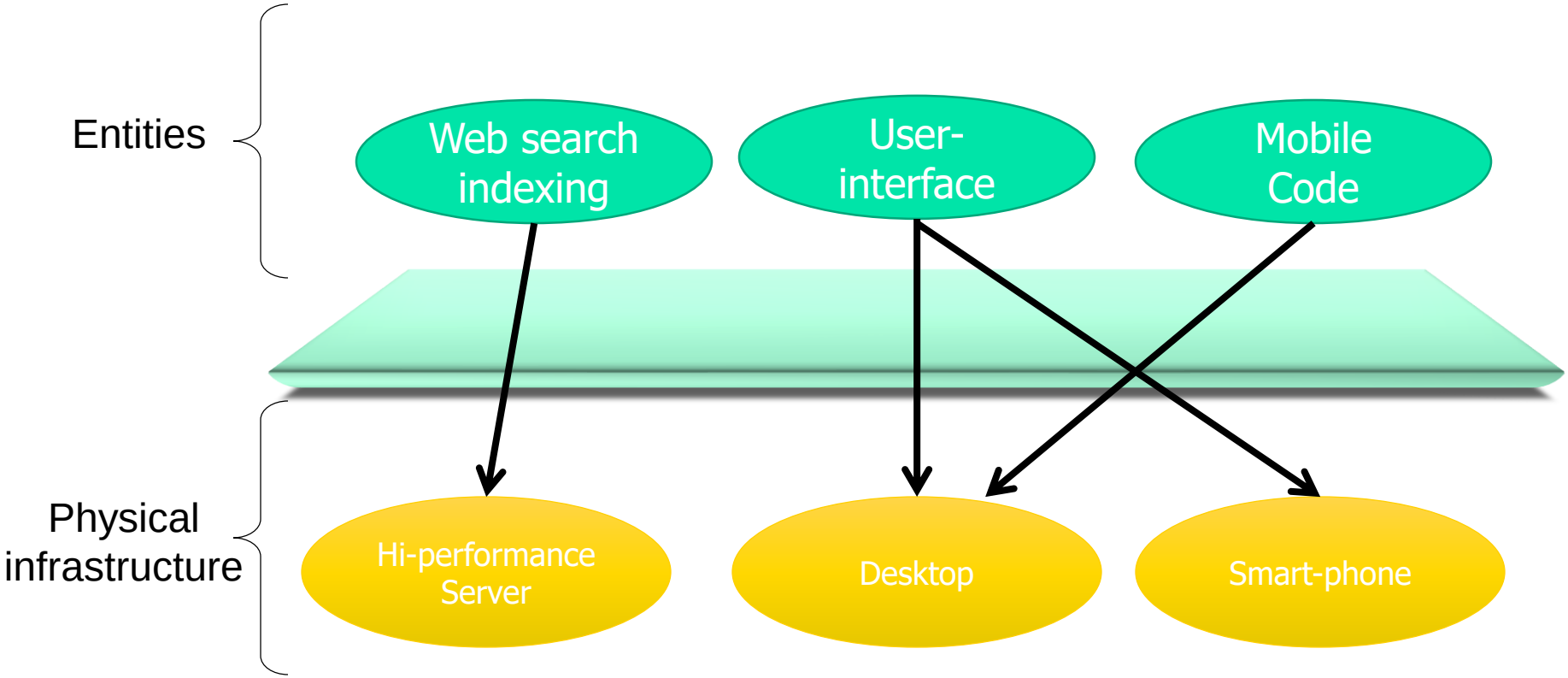
- 为了理解一个分布式系统的基础构建块，有必要考虑下面四个关键问题：
  - 在分布式系统中进行通信的实体是什么?
  - 它们如何通信，特别是使用什么通信范型?
  - 它们在整个体系结构中扮演什么(可能改变的)角色，承担什么责任?
  - 它们怎样被映射到物理分布式基础设施上(它们被放置在哪里)?

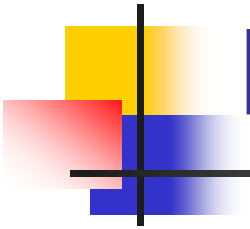


Placement

- Observation:
  - A large number of heterogonous hardware (machines, networks)
  - Smart mapping of entities (processes, objects) to hardware helps performance, security and fault-tolerance
- "Placement" maps entities to underlying physical distributed infrastructure
  - Placement should be decided after a careful study of application characteristics
  - Example strategies:
    - Mapping services to multiple servers
    - Moving the mobile code to the client

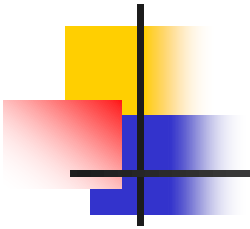
Placement


Entities
Web search indexing
User-interface
Mobile Code
Physical infrastructure
Hi-performance Server
Desktop
Smart-phone



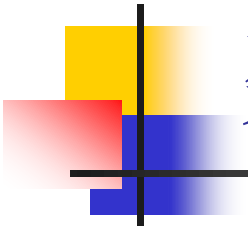
Recap

- We have covered primitive architectural elements
  - Communicating entities
  - Communication paradigms of entities
    - IPC, RMI, RPC, Indirect Communication
  - Roles and responsibilities that entities assume, and resulting architectures
    - Client-Server, Peer-to-Peer, Hybrid
  - Placement of entities



内容提要

- 引言
- 物理模型
- 架构模型
- 架构风格
  - layering architecture
  - tiered architecture
- 基础模型
- 总结

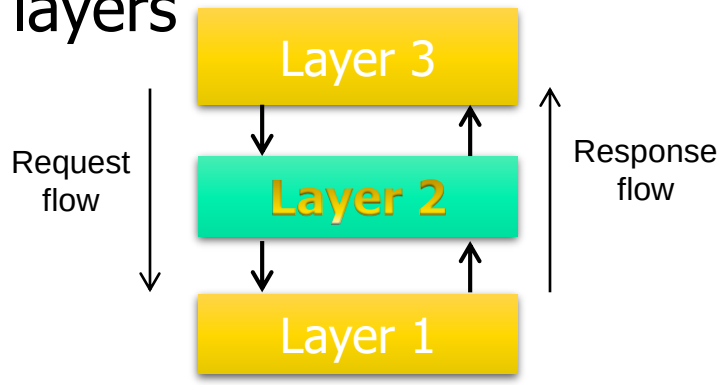


架构风格

- 架构风格构建在相对原始的体系结构元素之上
  - 提供组合的、重复出现的结构
  - 这些结构在给定的环境下能够运行良好
- 分层架构　layering architecture
- 层次化架构　tiered architecture

Layering

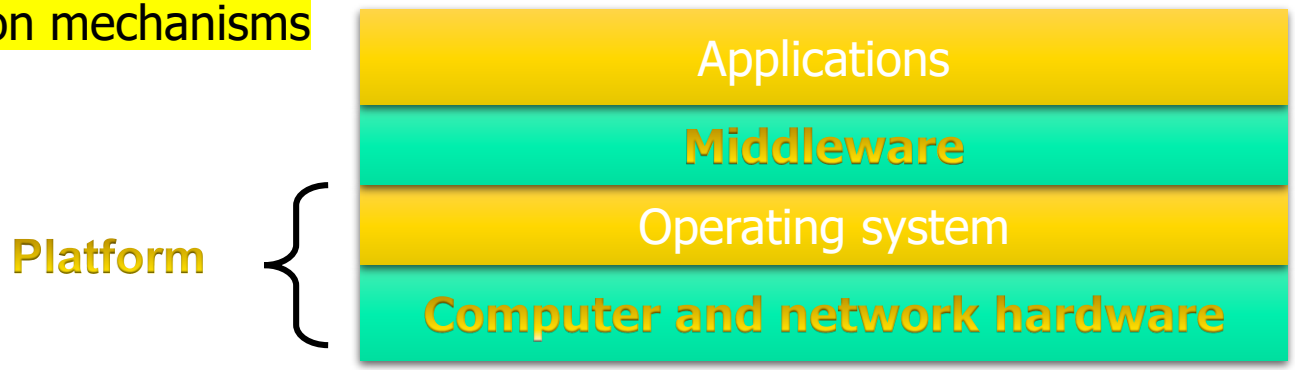
- A complex system is partitioned into layers
  - Upper layer utilizes the services of the lower layer
  - A vertical organization of services
- Layering simplifies design of complex distributed systems by hiding the complexity of below layers
- Control flows from layer to layer

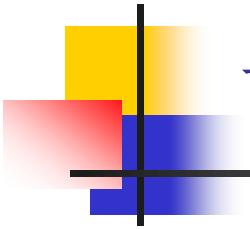


Layering – Platform and middleware

Distributed Systems can be organized into three layers

1. Platform
   - Low-level hardware and software layers
   - Provides common services for higher layers
2. Middleware
   - Mask heterogeneity and provide convenient programming models to application programmers
   - Typically, it simplifies application programming by abstracting communication mechanisms
3. Applications





平台Platform

最底层的软硬件，为上层提供服务。如：

- **Intel x86/Windows**
- **Intel x86/Linux**
- **Intel x86/Solaris**
- **SPARC/SunOS**
- **PowerPC/MacOS**

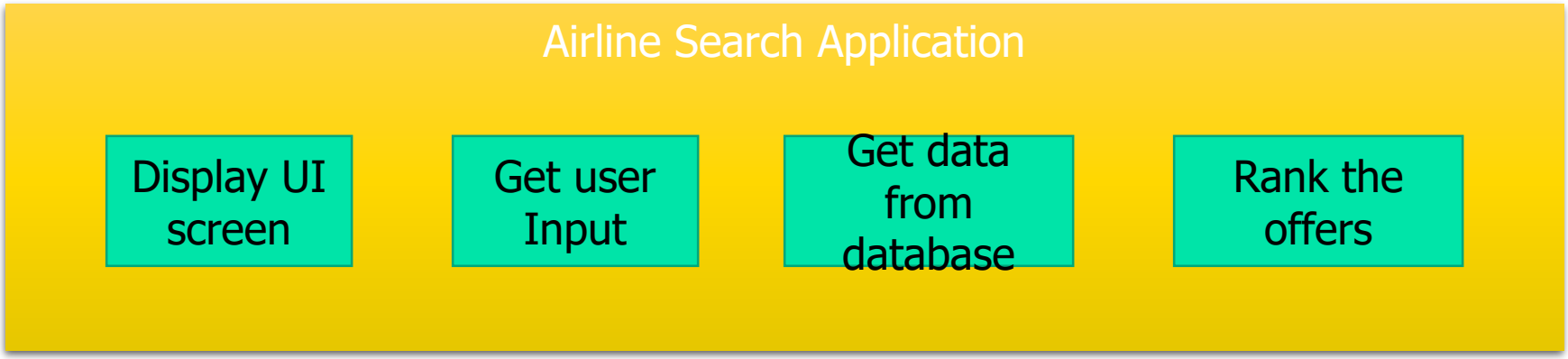
- **IAAS　PAAS　FAAS　……**

中间件

- 软件层，一组计算机上的进程和对象，它们互相交互，实现分布式系统的通信和资源共享。
- 用来对系统开发者屏蔽系统的异构性，提供更方便的编程模式
  - **e.g. OMG's CORBA, Java RMI, DCOM**
- 中间件可以提供这样一些对抽象的支持
  - **Remote method invocation: Sun RPC**
  - **Group communication: Isis**
  - **Notification of events: CORBA**
  - **The replication of shared data**
  - **Transmission of multimedia data**

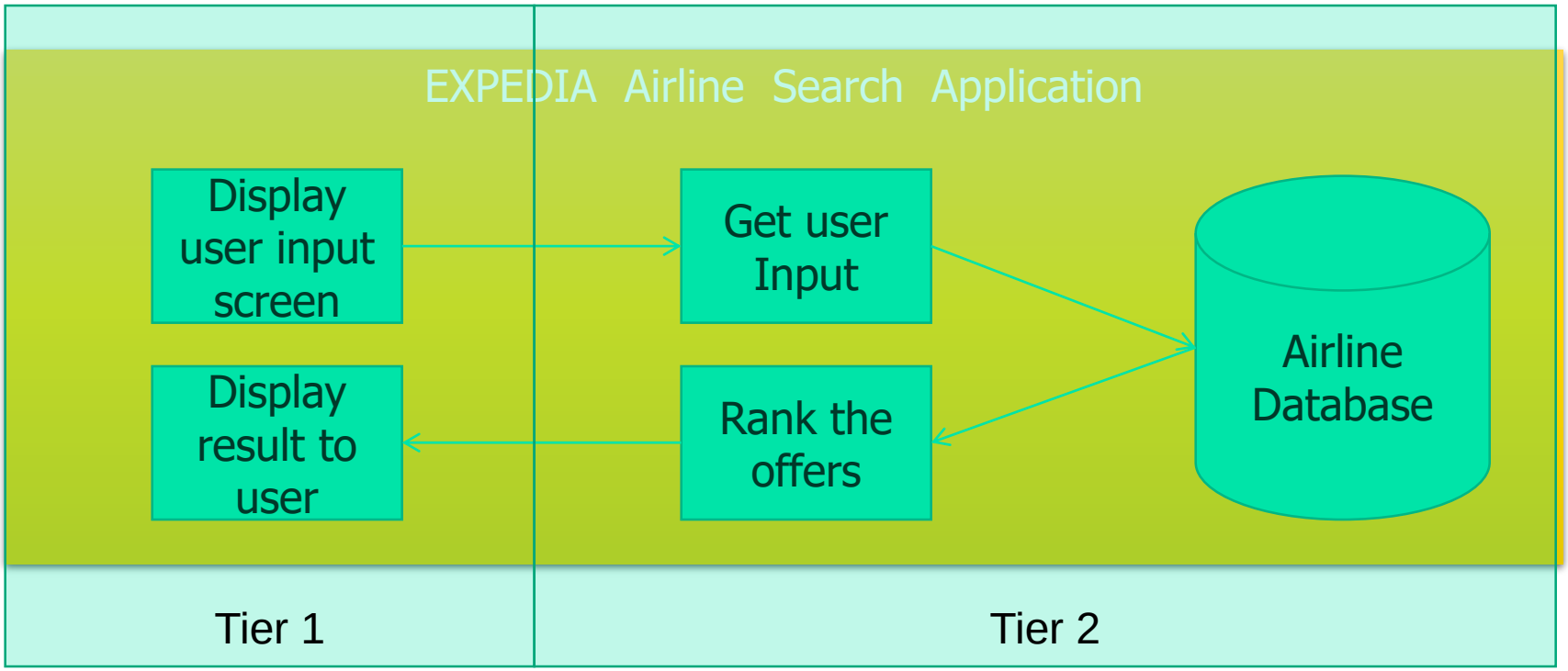
Tiered Architecture

- A technique to:
  1. Organize the functionality of a service, and
  2. Place the functionality into appropriate servers



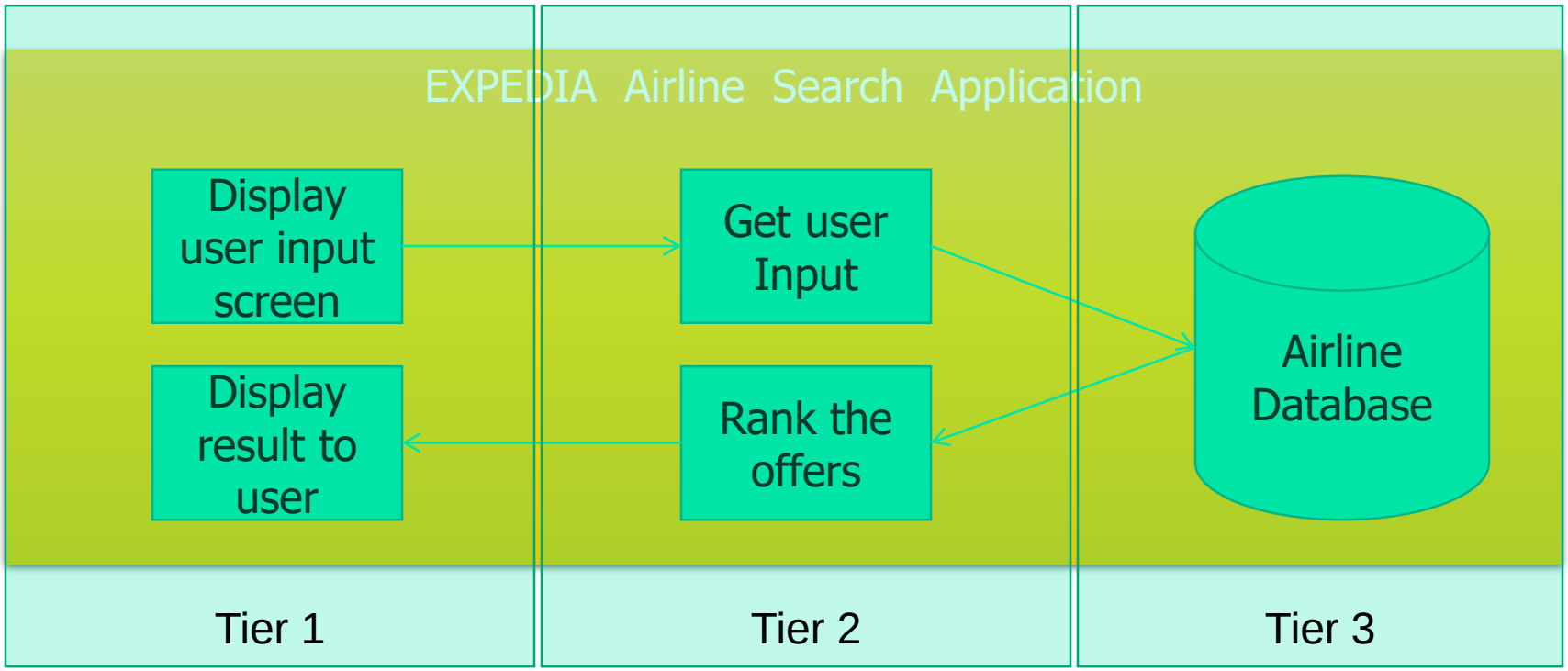
A Two-Tiered Architecture

- How do you design an airline search application:

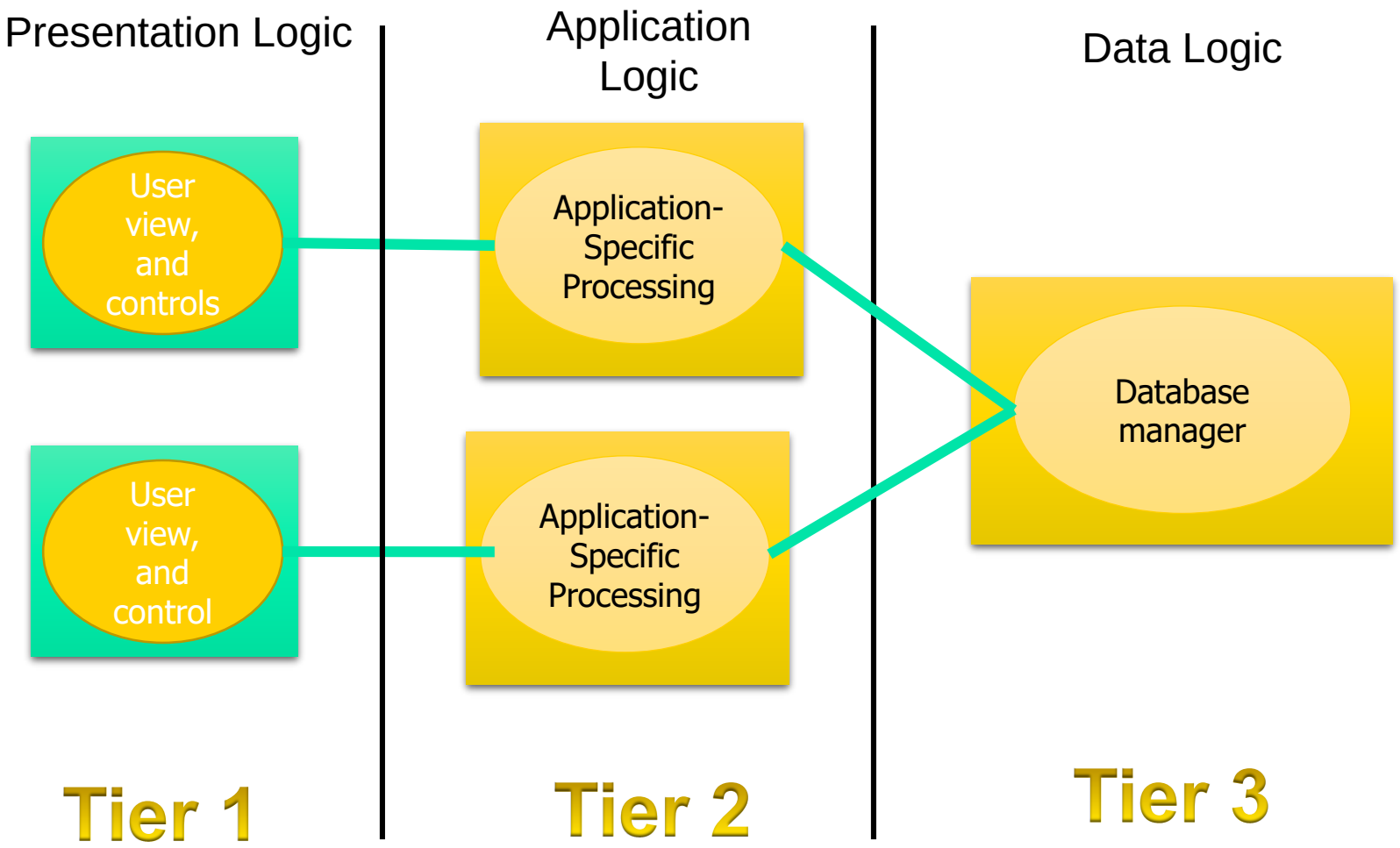


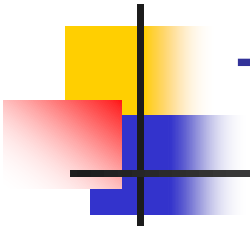
A Three-Tiered Architecture

- How do you design an airline search application:



A Three-Tiered Architecture





Three-Tiered Architecture: Pros and Cons

- Advantages:
  - Enhanced maintainability of the software (one-to-one mapping from logical elements to physical servers)
  - Each tier has a well-defined role
- Disadvantages
  - Added complexity due to managing multiple servers
  - Added network traffic
  - Added latency

架构模型---中间件的分类

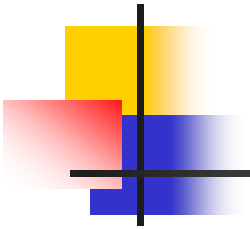
主类	子类	系统例子
分布式对象（第5、8章）	标准	RM-ODP
	平台	CORBA
	平台	Java RMI
分布式组件（第8章）	轻量级组件	Fractal
	轻量级组件	OpenCOM
	应用服务器	SUN EJB
	应用服务器	CORBA 组件模型
	应用服务器	JBoss
发布－订阅系统（第6章）		CORBA 事件服务
		Scribe
		JMS
消息队列（第6章）		Websphere MQ
		JMS
Web 服务（第9章）	Web 服务	Apache Axis
	网格服务	Globus Toolkit
对等（第10章）	路由覆盖网	Pastry
	路由覆盖网	Tapestry
	应用特定的	Squirrel
	应用特定的	OceanStore
	应用特定的	Ivy
	应用特定的	Gnutella

中间件的限制

- 许多分布式应用完全依赖中间件提供的服务来支持应用的通信和数据共享需求
  - 例如，一个适合客户-服务器模型的应用，如一个名字和地址的数据库，可以依赖只提供远程方法调用的中间件
  - 通过依靠中间件支持的开发，能大大简化分布式系统的编程，但系统可依赖性的一些方面要求应用层面的支持
- 考虑从发送邮件的主机传递电子邮件到接收邮件主机
  - 乍一看，这是一个TCP数据传输协议的简单应用。但问题是:
    - 用户试图在一个可能不可靠的网络上传递非常大的文件。TCP 提供一些错误检测和更正，但它不能从严重的网络中断中恢复。因此，邮件传递服务增加了另一层次的容错，维护一个进展记录，如果原来的TCP 连接断开了，用一个新的TCP 连接继续传递

端到端争论

- Saltzer, Reed 和Clarke 的一篇经典论文[Saltzer et al. 1984]，称为"端到端争论":
  - 一些与通信相关的功能，可以只依靠通信系统终点(end point)应用的知识和帮助即可完整、可靠地实现。
  - 将这些功能作为通信系统的特征不总是明智的(虽然由通信系统提供一个不完全版本的功能有时对性能提高是有用的)。
- 可以看出他们的论点与通过引入适当的中间件层将所有通信活动从应用编程中抽象出来的观点是相反的。
  - 争论的关键是分布式程序正确的行为在很多层面上依赖检查、错误校正机制和安全手段，其中有些要求访问应用的地址空间的数据。任何企图*在通信系统中*单独完成的检查将只能保证部分正确性。
  - 因此，可能在应用程序中重复同样的任务，降低了编程效率，更重要的是增加了不必要的复杂性并要执行冗余的计算。



内容提要

- 引言
- 物理模型
- 架构模型
- 架构风格
- 基础模型
- 总结

基础模型

- 所有的系统模型
  - 都由若干进程组成且通过在计算机网络上发送消息而相互通信
  - 所有的模型都共享下列设计需求:
    - 实现进程及网络的性能和可靠性特征，确保系统中资源的安全性。
- 利用这些基本特性的模型，更详细地描述系统可能展示的特征、故障和安全风险。
  - 交互模型
  - 故障模型
  - 安全模型

基础模型的目的

- 显式地表示有关我们正在建模的系统的假设。
- 给定这些假设，就什么是可能的、什么是不可能的给出结论。
  - 结论以通用算法或要确保的特性的形式给出。
  - 特性成立的保证依赖于逻辑分析和(适当时候的)数学证明。

基础模型

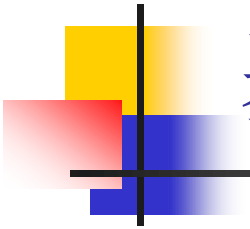
- 交互模型
  - 进程之间通过消息传递进行交互，实现系统的通信和协作功能。
  - 有较长时间的延迟。
  - 时间是进程间进行协调的基本的参照，在分布式系统中，很难有相同的时间概念。
  - 独立进程之间相互配合的准确性受限于上面两个因素。

基础模型

- 故障模型
  - 计算机或者网络发生故障，会影响服务的**正确性**
  - 故障模型的意义在于将定义可能出现的故障的形式，为分析故障带来的影响提供依据
  - 设计系统时，知道如何考虑到容错的需求。

基础模型

- 安全模型
  - 分布式系统的**模块特性**以及**开放性**，使得它们**暴露**在内部和外部的攻击之下
  - 安全模型的目的是提供依据，以此分析系统可能收到的侵害，并在设计系统时防止这些侵害的发生。



基础模型之交互模型

- 两个影响进程交互的因素
  - 通信性能
  - 不可能维护一个全局时间概念

基础模型之交互模型

通信通道的性能

- 延迟
  - 第一个比特流从发出到到达目的节点在网络中所花费的时间。
  - 访问网络的时间、操作系统通信服务的时间
- 带宽
  - 在单位时间内，网络上能够传输的信息的总量
- 抖动
  - 传输一系列信息所花费时间的变化值
  - **E.g.** 抖动会影响流媒体服务的质量，因为这类数据要求相对稳定的传输率。

基础模型之交互模型

计算机时钟和时间事件

- 时钟漂移率（**Clock drift rate**）
  - 局部时钟和一个精确的参考时钟的差值 **Timing event**
  - **e.g., GPS, Logical time**

基础模型之交互模型

交互模型的两个变体

- 同步分布式系统
  - 进程执行**每一步的时间**有一个上限和**下限**。
  - 每个在网络上**传输消息**可在**已知时间范围内接收到**。
  - 每个进程的**局部时钟**相对于实际时间的**漂移**是在已知的范围内。

基础模型之交互模型

交互模型的两个变体

- 异步分布式系统---没有可预测的时限：
  - 进程执行速度
    - 每一步都可能需要任意长的时间
  - 消息传递延迟
    - 收到一个消息的等待时间可能任意长
  - 时钟漂移率
    - 漂移率可能是任意的

基础模型之交互模型

同步分布时系统和异步分布时系统的例子

- 异步分布式系统
  - **Email**
  - **FTP**
- 同步分布式系统
  - **VOD（Video on demand)**
  - **Voice Conference System**

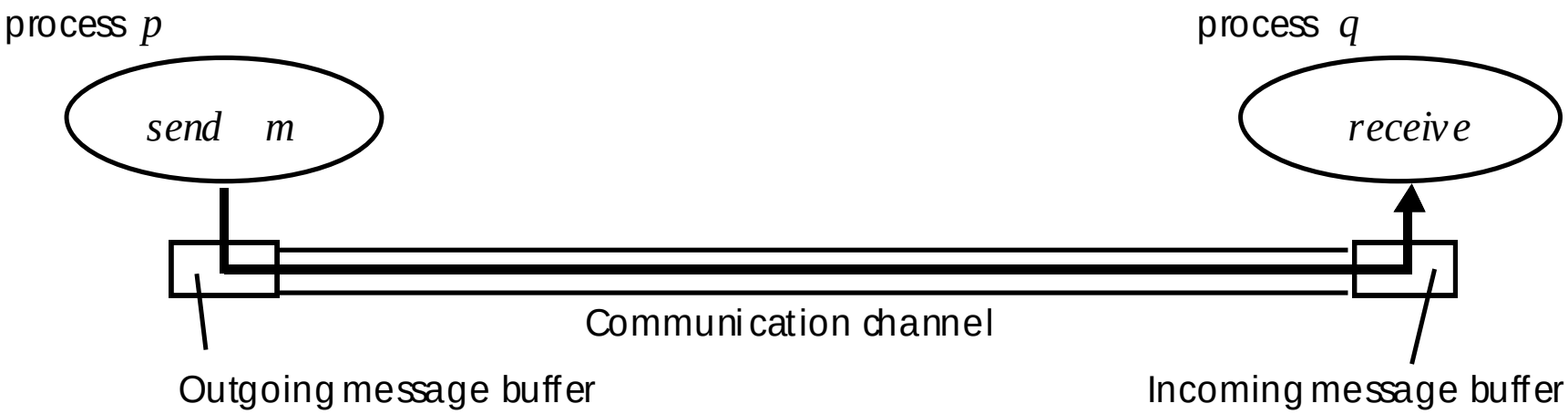
故障模型

- 定义故障发生的行为，帮助理解故障对分布式系统的影响。
- 分类**[Hadzilacos and Toueg, 1994]**
  - 遗漏故障Omission failures
  - 随机故障Arbitrary failures
  - 时序故障Timing failures

故障模型---遗漏故障

- 进程或者通信通道不能完成它应该做的动作
- 进程遗漏故障：进程崩溃
  - 进程停止：
    - 在同步系统中通过时限是可以检测出来的。
    - 但是在异步系统中，即使很长时间没有收到来自某个进程的消息，也不能判断是进程停止了。
- 通信遗漏故障：丢失信息
  - 发送丢失，接受丢失，通道丢失
- 遗漏故障是良性故障

故障模型---遗漏故障


process p
process q
send m
receive
Communication channel
Outgoing message buffer
Incoming message buffer

故障模型---随机故障

- 随机故障（拜占庭故障）：对系统影响最大的一种故障形式，而且错误很难探知。
- 发生在进程中的随机故障：随便遗漏应有的处理步骤或者进行不应有的处理步骤，该做的不做，不该做的却做了。
  - **E.g.** 对一个过程调用返回一个错误的信息
- 随机故障很少出现在通信信道。
  - **E.g.** 校验和，消息有序列号。

故障模型---时间故障

- 仅仅发生在同步分布式系统中

Class	*effects*	*Description*
Clock	**Process**	**Process's local clock exceeds the bounds on its rate of drift from real time.**
Performance	**Process**	**Process exceeds the bounds on the interval between two steps.**
Performance	**Channel**	**A message's transmission takes longer than the stated bound.**

故障模型---屏蔽故障

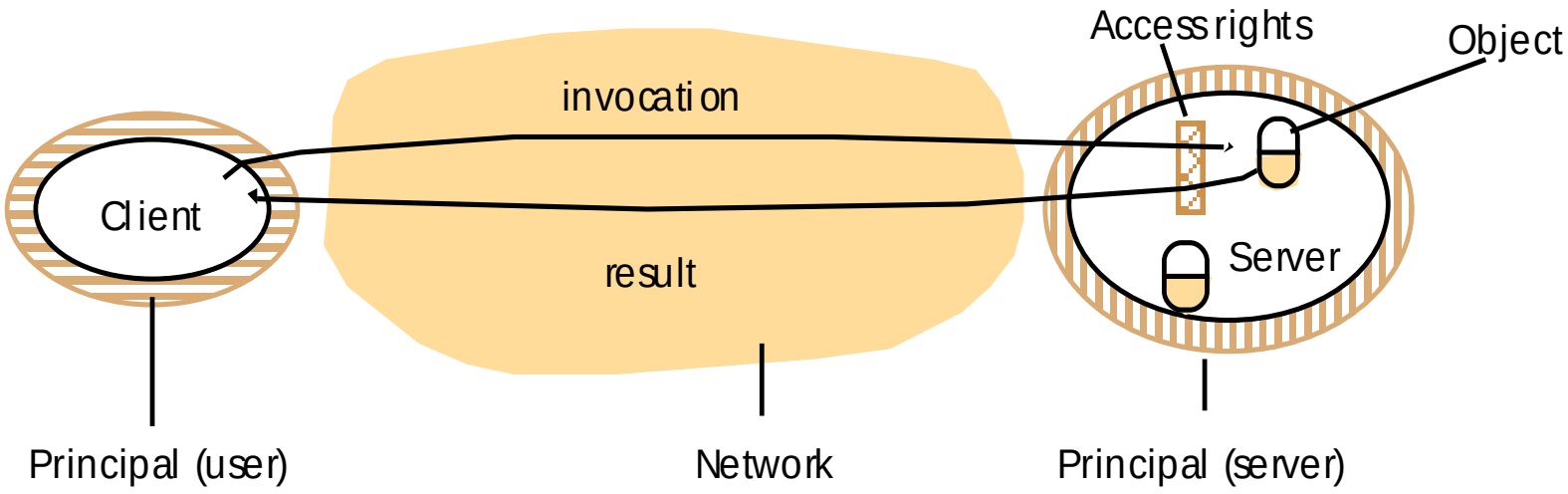
- 隐藏
  - 例如：备份服务器
- 转换
  - 例如：检验校验和发现出错信息，将其丢弃：**arbitrary failure -> omission failure**
- 一对一通信的可靠性
  - 有效性（**Validity**）
    - 在发送端缓冲区的消息最终能够到达接收端的缓冲区。
  - 完整性（**Integrity**）
    - 接收到的消息和发送的消息完全一样，没有消息被发送两次。

安全模型

- 分布式系统的安全性
  - 进程的安全性
  - 通信信道的安全性
  - 对象的安全性
- 对对象的保护措施
  - 访问权限：在**对象的访问控制表**中规定什么人具有访问一个对象的权限
  - 权能：**用户一方**所持有的访问那些对象的权限

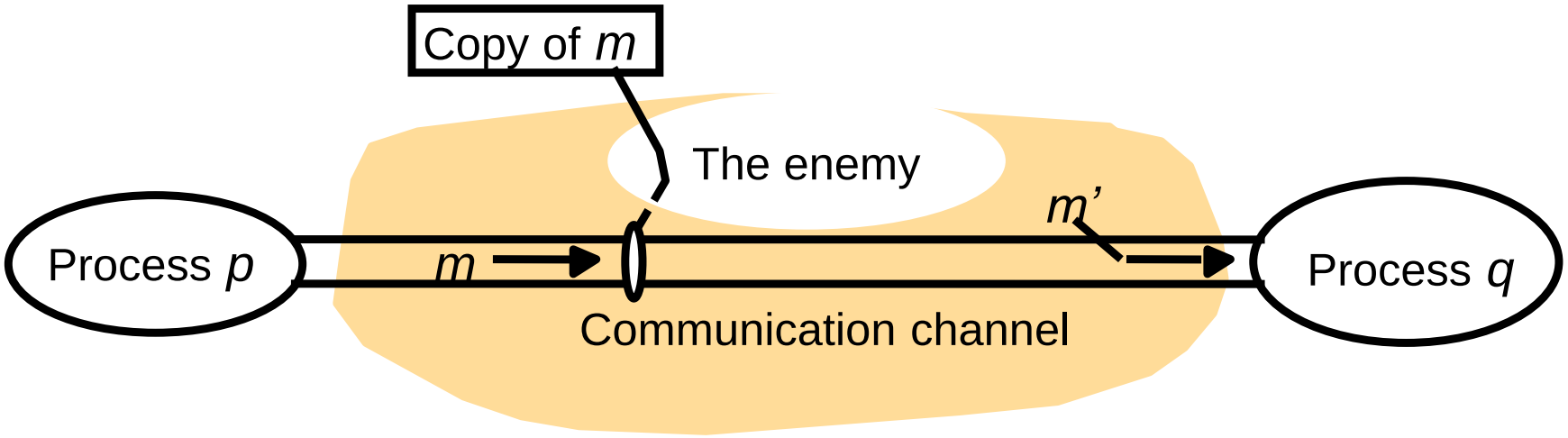
安全模型—威胁

- 对进程的威胁
  - 对服务器：来自一个伪造实体的调用请求,例如：欺骗 **mail server**
  - 对客户端：收到一个伪造的结果。例如：窃取密码



安全模型—威胁

- 对信道的威胁
  - 拷贝消息，篡改或者填充
  - 保存并重发：例如：从资金在两个账号间转移，如果重发这样的请求，将产生错误结果。



安全模型—威胁

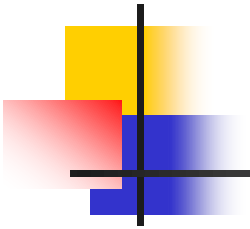
- **Dos**攻击
  - 发出大量的服务请求，造成服务器过载不能提供服务，或者在网上传送大量的消息，占用带宽，拥塞网络。
- 移动代码
  - 恶意代码入侵系统，例如：特洛依木马。

安全模型—对策

- 加密技术和共享密钥
  - 通过共享密钥相互确认对方
  - 密码学（**Cryptography**）是基础
- 认证（**Authentication**）
  - 对发送方提供确认身份进行认证。

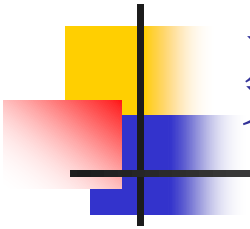
安全模型—对策

- 安全通道
  - 每个进程知道正在执行的进程所代表的委托方的身份。
  - 确保在其上传输的数据的私密性和完整性。
  - 每个消息包含物理的和逻辑的时间戳。



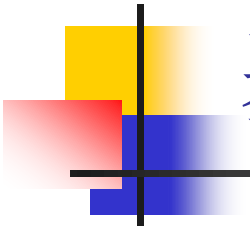
内容提要

- 引言
- 物理模型
- 架构模型
- 架构风格
- 基础模型
- 总结



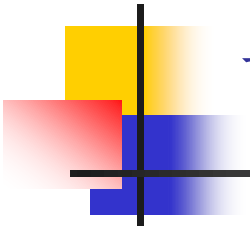
架构模型

- **Client / Server**
  - **e.g. Web, FTP, NEWS**
- **Multiple Servers**
  - **e.g. DNS**
- **Proxy and Cache**
  - **e.g. Web Cache**
- **Peer processes**
- **Variations of C/S**
  - **Mobile code, mobile agent, network computer, thin client, spontaneous networks**



基础模型

- **Interaction models**
  - **synchronous DS and asynchronous DS**
- **Failure models**
  - **omission failures**
  - **arbitrary failures**
  - **timing failures**
- **Security model**
  - **the enemies**
  - **the approaches of defeating them**



课堂讨论

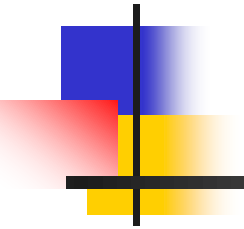
- 应用场景：多人参与的聊天室
- 什么样的结构模型适合这种应用？
- 对照基础模型中的几个情况，讨论可能出现的问题。

课堂讨论

- 结构模型**1**：完全对等
- 结构模型**2**：有中心的组管理服务器。
- 交互模型：时序问题，不能一个人看到的回答先于问题到达。
- 故障模型：消息丢失，保证每个组成员都要收到相同的消息。
- 安全模型：消息的加密。

思考题

- 分布式系统的物理模型有哪些？你觉得发展的趋势如何？
- 分布式系统常用的架构风格有哪几种？各有哪些特点？
- 随着物理模型的发展，以及5G、CPS等技术和需求的发展，你觉得分布式系统软件模式的发展趋势有哪些？



END