第2章 系统模型

第2章 系统模型

- 引言
- 物理模型
- 体系结构模型
- 基础模型
- 总结

引言

- 物理模型 (physical models)
  - 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；
- 体系结构模型 (architectural models)
  - 从系统的计算元素执行的计算和通信任务方面来描述系统；
- 基础模型 (fundamental models)
  - 采用抽象的观点描述大多数分布式系统面临的单个问题的方案。

物理模型

- 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。
- 早期的分布式系统
  - 80s, LAN
- 互联网规模的分布式系统
  - 90s, cluster

静态、分立和自治
Static, discrete, autonomous

- 当代的分布式系统
  - 移动计算——节点移动(service discovery)
  - 普适计算——嵌入日常物品和周围环境
  - 云计算——一组结点提供给定服务

物理模型

Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

体系结构模型

- 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**系统可靠性、适应性、可管理性和性价比。

体系结构模型

- 体系结构元素：分布式系统的基础构建块
- 体系结构模式：构建在体系结构元素之上，提供组合的、重复出现的结构

体系结构元素——关键问题

- 通信的实体是什么？
- 如何通信（通信范型）？
- 扮演的角色及承担的责任是什么？
- 如何映射到具体的基础设施上（放置）？

体系结构元素——通信实体

- 面向系统的角度
  - 进程
  - 线程
  - 节点
- 面向问题的角度
  - 对象：给定问题领域的自然单元（Java RMI, CORBA）
  - 组件：对象+需要的接口（JavaBeans）
    - 依赖关系/部署支持/复杂度
  - Web服务：经常跨越组织边界
    - 微服务架构

体系结构元素——通信范型

- 进程间通信：相对底层的支持
  - 套接字、多播、消息传递
- 远程调用：最常见的通信范型，双向交换
  - 请求—应答协议：一对消息的交换
  - 远程过程调用（Remote Procedure Call, RPC）：远程计算机上进程中的过程能被调用
  - 远程方法调用（Remote Method Invocation, RMI）：一个发起调用的对象能够调用一个远程对象中的方法。

体系结构元素——通信范型

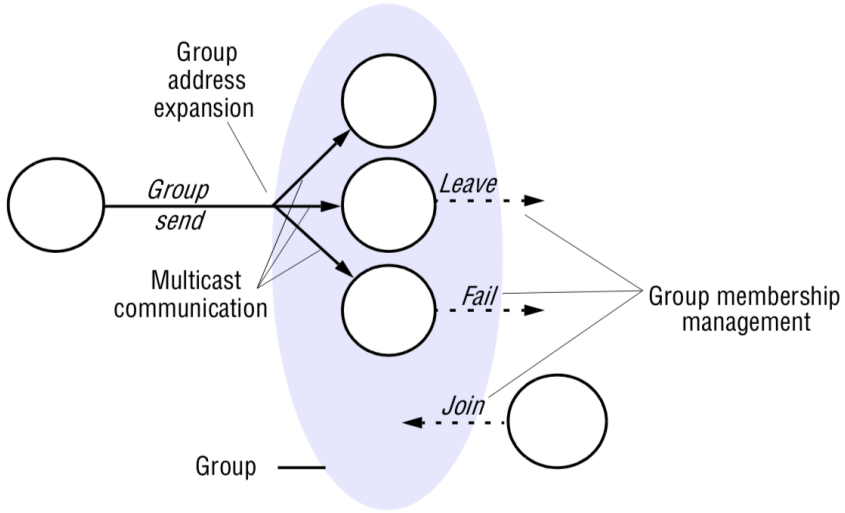
- 进程间通信及远程调用的限制：
  - 发送者需要知道接收者（空间耦合）
  - 发送者和接收者同时存在（时间耦合）

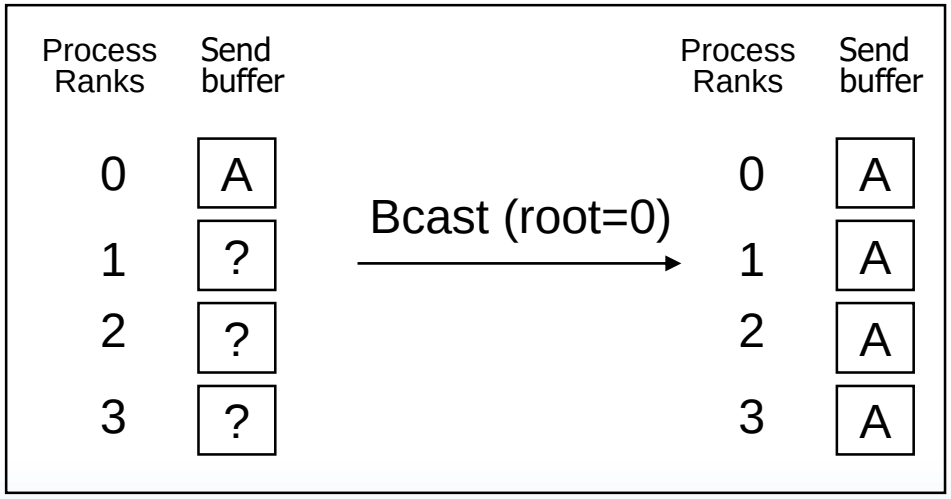
体系结构元素——通信范型

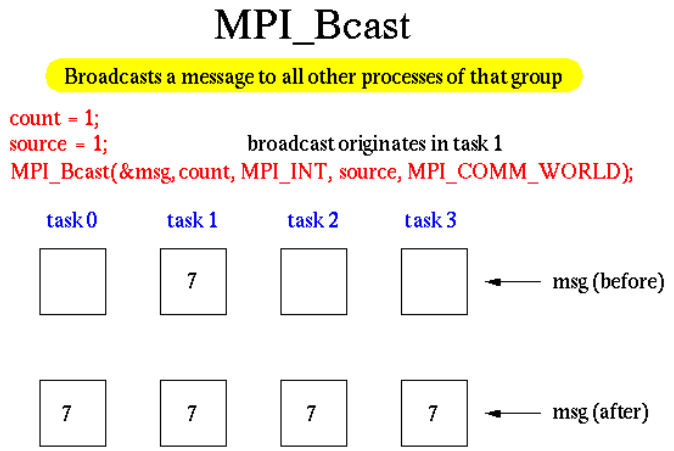
- 间接通信（空间、时间解耦合）
  - 组通信：一对多通信范型
  - 发布—订阅系统：提供一个中间服务，有效确保由生产者生成的消息被路由到需要这个消息的消费者
  - 消息队列：提供点到点服务
  - 分布式共享内存：用于支持在不共享物理内存的进程间共享数据。
  - 元组空间：进程可把任意结构化数据项存放在一个持久元组空间，其他进程可读或删除，类似 DSM

Group Communication

- Membership management
- Examples
  - Multicast
    - 224.0.0.0/4
  - Overlay network
    - MPI

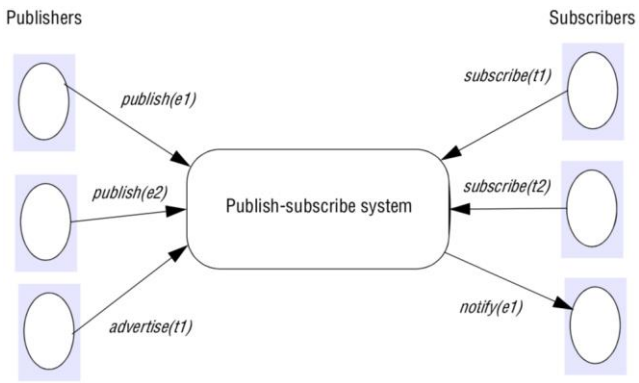


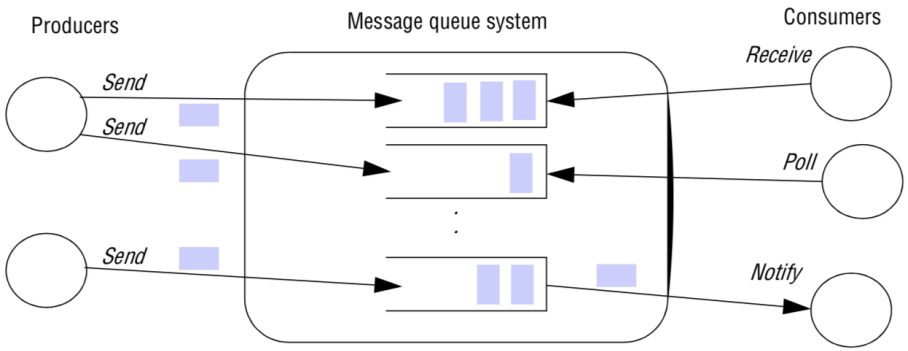


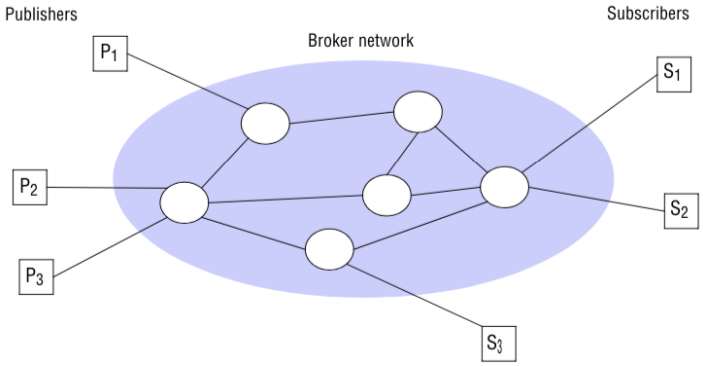


Publish-subscribe

- Event-driven: event notification
  - Channel-based, topic-based,
  - Centralized vs. Decentralized
- Examples
  - RabbitMQ, ActiveMQ, MQTT



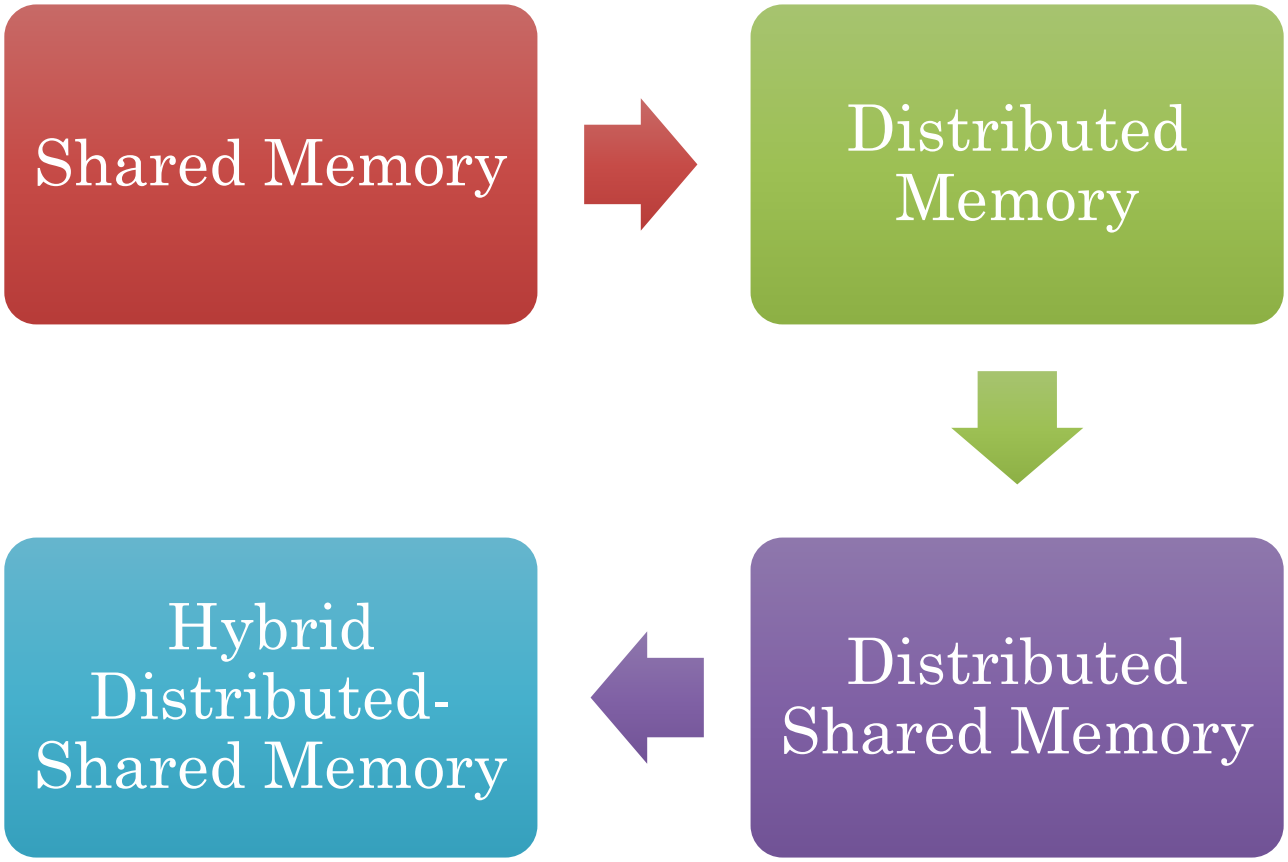




Exercise

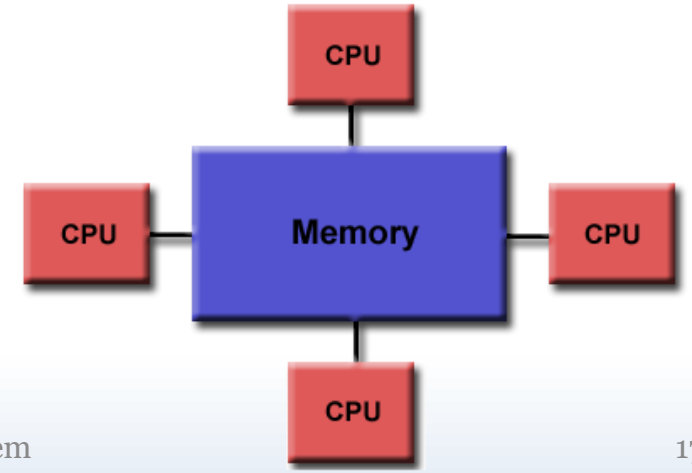
- MQ is one of the core components in distributed systems
- Read about SQS / RabbitMQ
- Write a program with RabbitMQ

Shared Memory



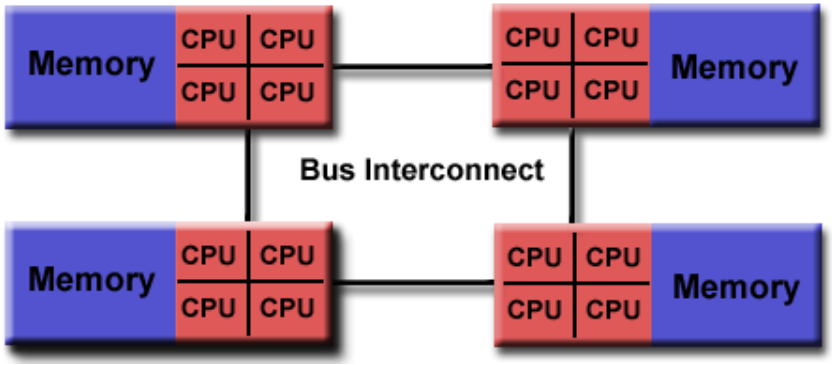
Shared Memory

- All processors share the same ***memory address (and inherently, address space)***
- Uniform Memory Access (UMA)
  - Symmetric Multiprocessor (SMP); typical multi-core CPU
  - Identical processors, equal access times to memory
  - Sometimes called CC-UMA - Cache Coherent UMA.
    - if one processor updates a location in shared memory, all the other processors know about the update → accomplished at the hardware level.



Non-Uniform Memory Access (NUMA)

- Often made by physically linking two or more SMPs
  - One SMP can directly access memory of another SMP
  - Not all processors have equal access time to all memories
  - Memory access across link is slower
  - If cache coherency is maintained, then may also be called CC-NUMA

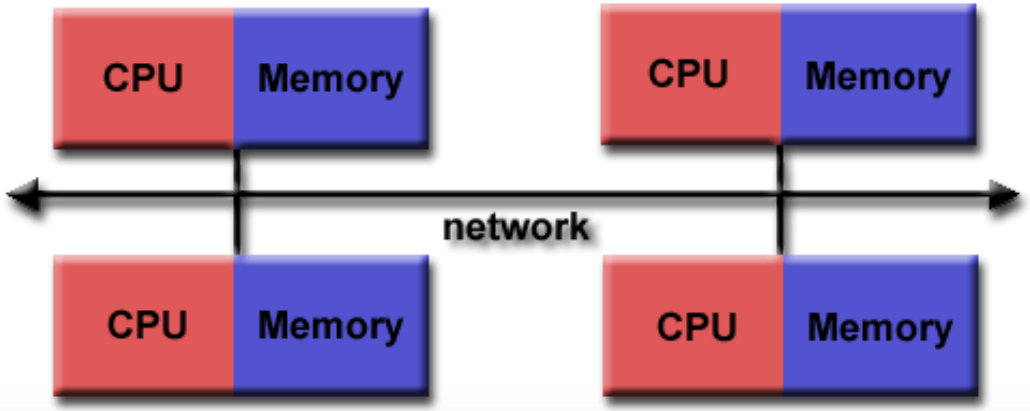


Shared Memory

- Advantages
  - Global address space provides a user-friendly programming perspective to memory
  - Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs
- Disadvantages
  - Lack of scalability between memory and CPUs.
    - Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
  - Programmer responsibility for synchronization constructs that ensure "correct" access of global memory.

Distributed Memory

- Local private memory for each processor → memory addresses in one processor do not map to another processor → *no concept of global address space*
  - computational tasks can only operate on local data (memory)
  - require a communication network to connect inter-processor memory.
  - local update → cache coherency does not apply.

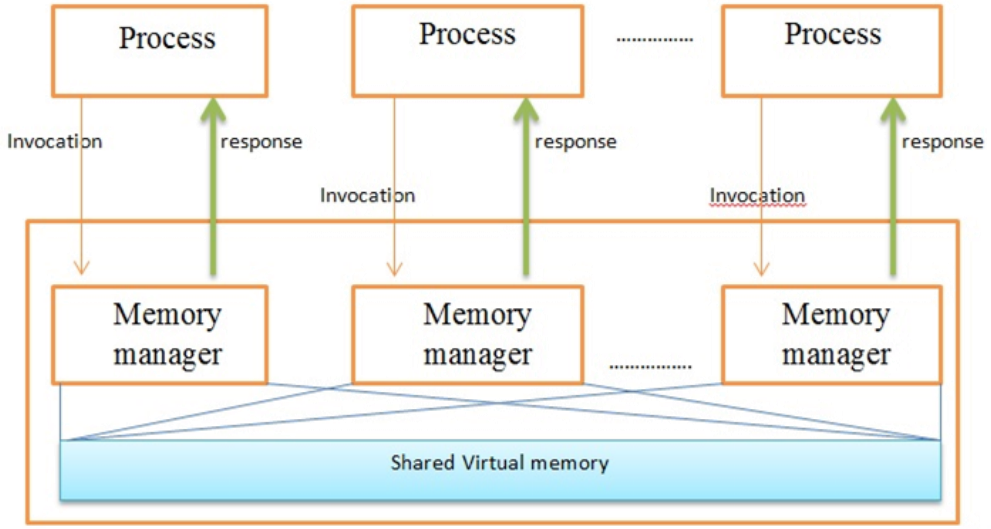


Distributed Memory

- Advantages
  - Memory is scalable with the number of processors. Increase the number of processors and the size of memory increases proportionately.
  - Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain global cache coherency.
  - Cost effectiveness: can use commodity, off-the-shelf processors and networking.
- Disadvantages
  - The programmer is responsible for many of the details associated with data communication between processors.
  - It may be difficult to map existing data structures, based on global memory, to this memory organization.
  - Non-uniform memory access times - data residing on a remote node takes longer to access than node local data.

Distributed Shared Memory (DSM)

- A form of memory architecture, the ***physically separate*** memories can be addressed as one ***logically shared address space***.
- it is shared memory, NUMA.
- Programming model is the same as shared memory.



DSM——Advantages

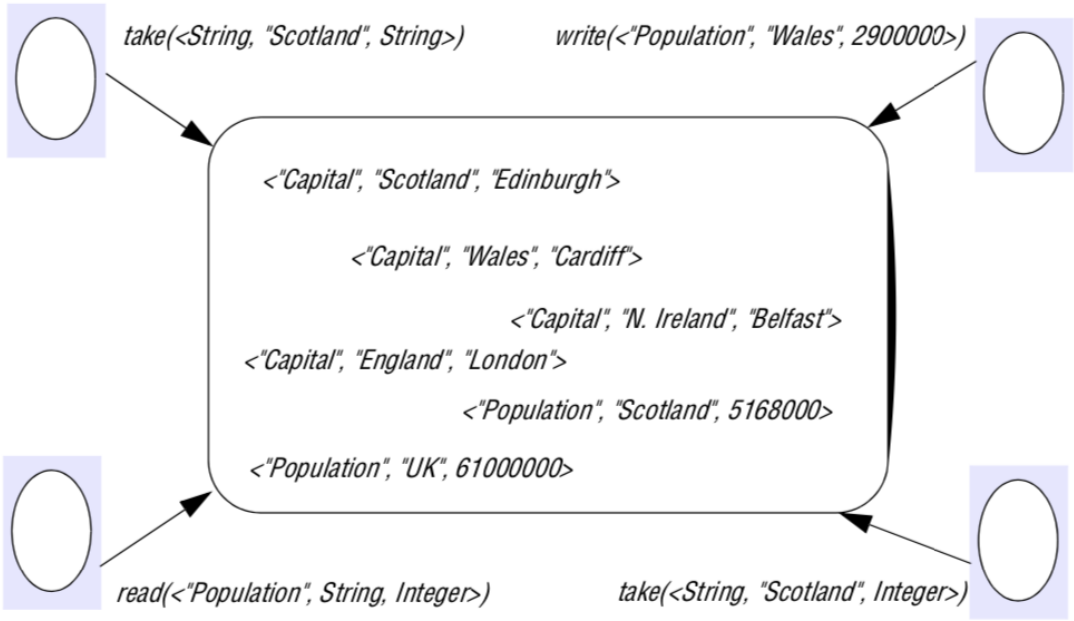
- Hides the message passing
- Can handle complex and large data bases without replication or sending the data to processes
- Provides large virtual memory space
- Programs are portable as they use common programming interface
- Shields programmer from sending or receive primitives

DSM——Issues

- Could cause a performance penalty
  - Like framework, difficult to optimize
- Should provide for protection against simultaneous access to shared data such as lock.
- Controlled by memory management software, operating system, language run-time system
- Granularity: how much data is transferred in a single operation?

Tuple Space

- Similar to DSM + encapsulation + MQ matching
- Implementation: centralized vs. decentralized
- Example
  - Javaspace, Linda



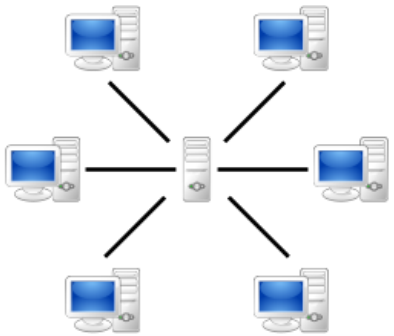
体系结构元素——角色及责任

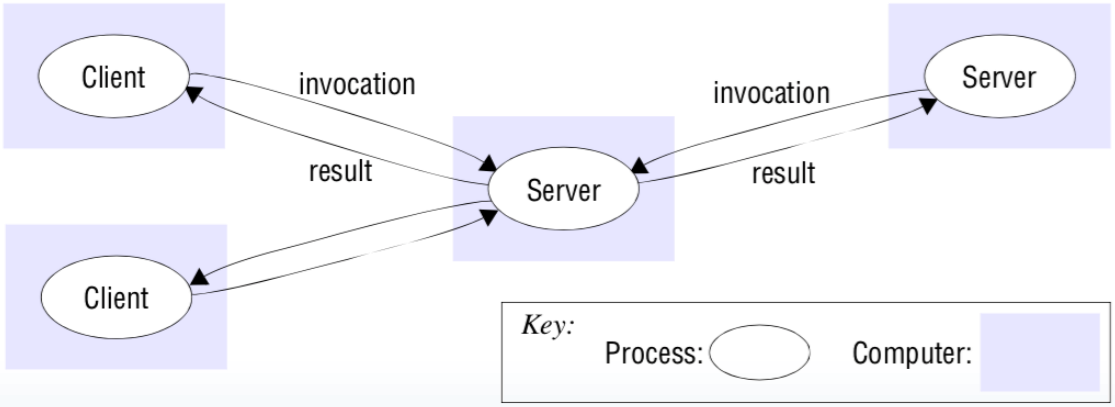
- 客户—服务器 Client-Server/CS/BS

- 对等体系结构 Peer-to-Peer/P2P

角色及责任——客户－服务器

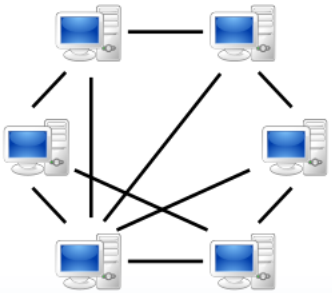
- 历史上最重要的结构之一，现在仍被广泛使用。
- 优点：
  - 简单、直接
- 缺点：
  - 伸缩性差，系统的伸缩性不会超过提供服务的计算机的能力和该计算机所处网络连接的带宽。
- 例子：web/ftp/email

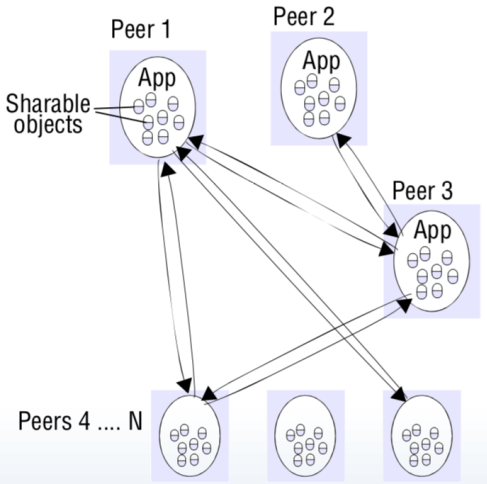




角色及责任——对等体系结构

- 观察
  - 普通节点已具备一定的能力，且多数具备随时可用的带宽。
  - 挖掘普通节点的能力可提升系统的服务能力
- 目的
  - 可用于运行服务的资源随用户数目的增加而增加。
- 特点
  - 系统应用中，完全由对等进程组成，进程间的通信模式完全依赖于应用的需求。
  - 管理难度大：节点动态性、路由、QoS、安全
- 例子
  - Bittorrent
  - eDonkey
  - IPFS



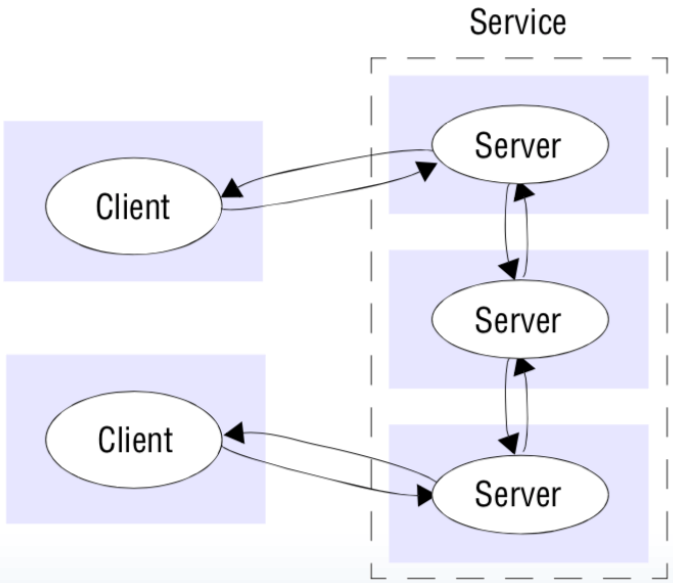


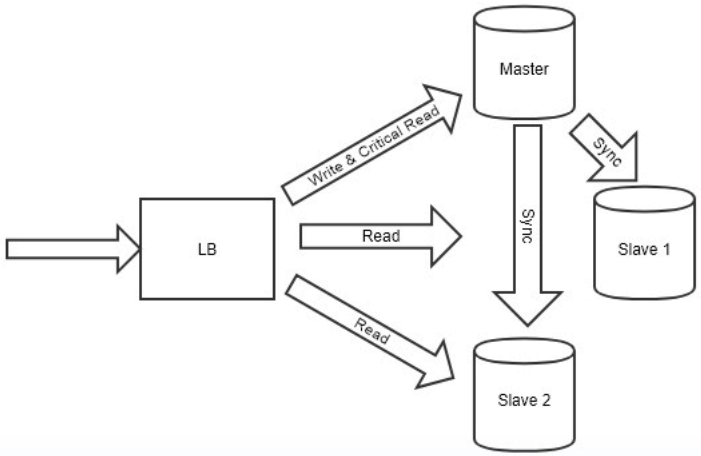
体系结构元素——放置

- 放置：对象或服务等实体如何映射到底层的物理分布式设施上。
  - E.g., Ceph hash algorithm
- 放置大多与性能相关，但也涉及到可靠性及安全性等
  - HDFS replication policy: 2 in same rack, 1 in different rack
  - DMZ
- 常见的放置策略
  - 将服务映射到多个服务器
  - 缓存
  - 移动代码
  - 移动代理

放置——服务器组

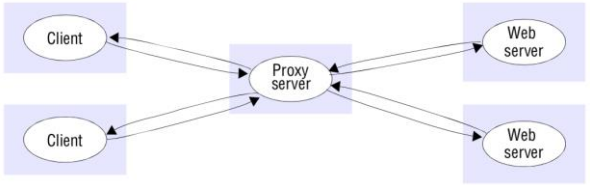
- 将不同的服务对象在不同的机器上实现
  - e.g. Web
- 在多个主机上维护副本服务
  - e.g. Sun NIS Service, MySQL replication
  - 衍生问题：如何支持节点动态变化？如何确认身份可信度？





放置——缓存

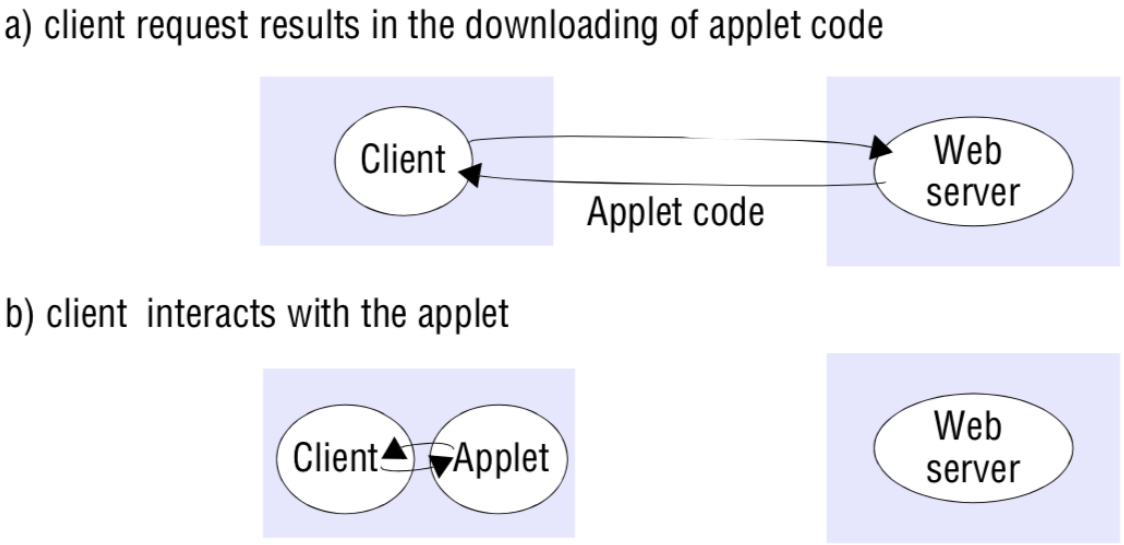
- 缓存（Cache）
  - 保存最近使用过的数据，在本地或代理服务器上。
  - 减少不必要的网络传输，减少服务器负担，还可以代理其它用户透过防火墙访问服务器。



- 两种类型：前向、反向
  - Forward position system (recipient or client side)
  - Reverse position system (content provider or web-server side)
- Examples
  - Squid / Vanish / CDN

放置——移动代码

- 将代码下载到客户端运行，可以提高交互的效率。
  - Java Applet
  - JavaScript / Actionscript



放置——移动代理

- 在网络上的计算机之间穿梭，并执行的代码。代替一些机器执行任务。
  - 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。
- 和移动代码比较：移动代码是服务器与客户端之间传递代码，代理则是在服务器之间传递。
- 和远程调用比较：松耦合，数据传输少，通信开销低。安全威胁限制了它的使用。

Examples

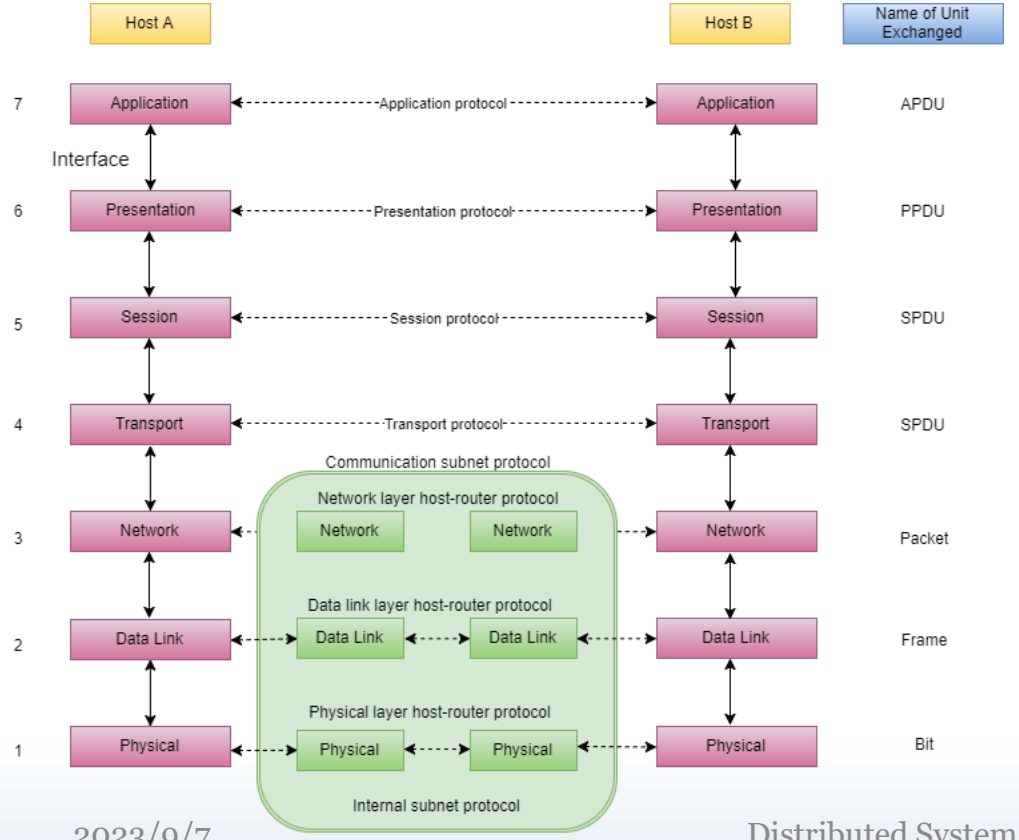
- Take advantage of non-local resources
  - Using compute resources on a wide area network, or even the Internet when local compute resources are scarce or insufficient.
  - Example: SETI@home over 165K hosts in nearly every country in the world (March, 2020).
  - Example: Folding@home uses over 4M computers globally (June, 2020)

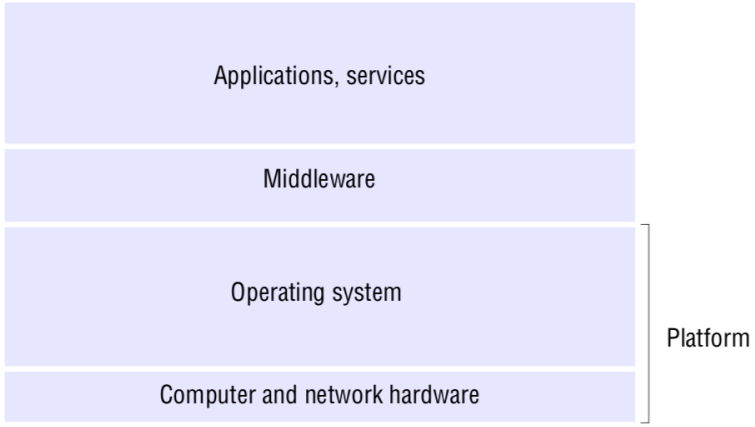
体系结构模式

- 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。
- 关键体系结构模型
  - 分层体系结构（layering architecture）
  - 层次化体系结构（tiered architecture）
  - 瘦客户（thin client）

体系结构模式——分层

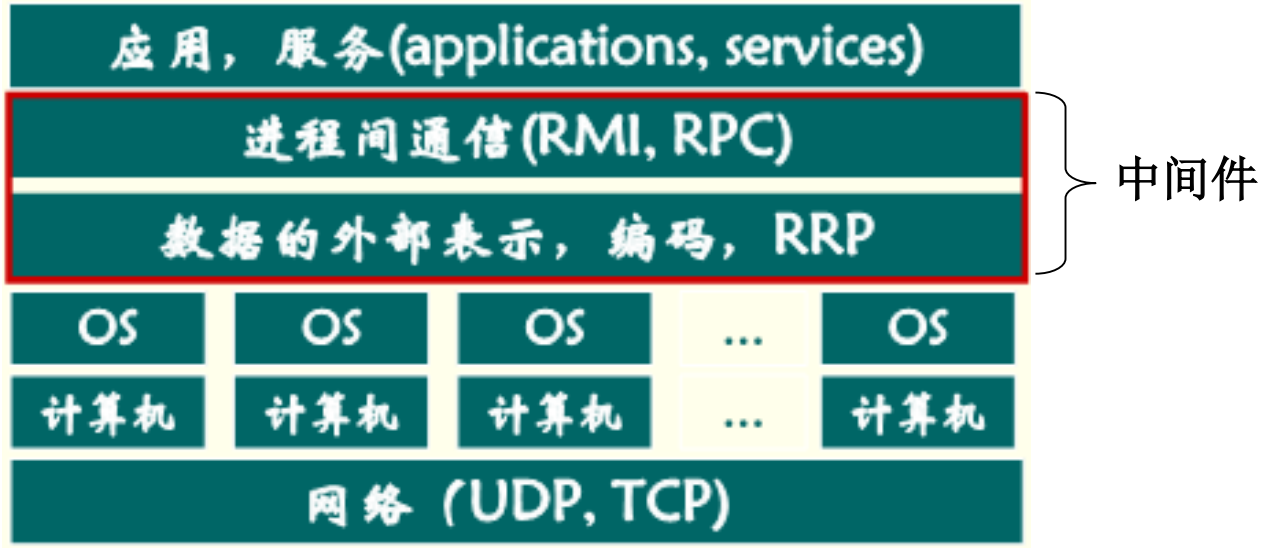
- 分层：一个复杂的系统被分成若干层，每层利用下层提供的服务。
- 在分布系统中，等同于把服务垂直组织成服务层。





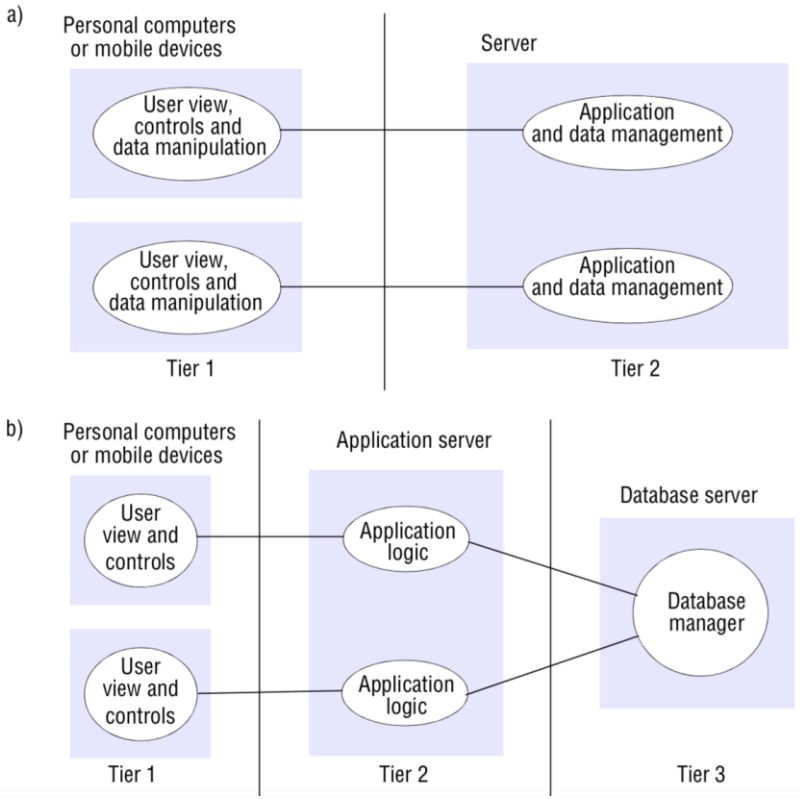
体系结构模式——分层

- 中间件层
  - 中间件是一种软件，它提供基本的通信模块和其他一些基础服务模块，为应用程序开发提供平台。



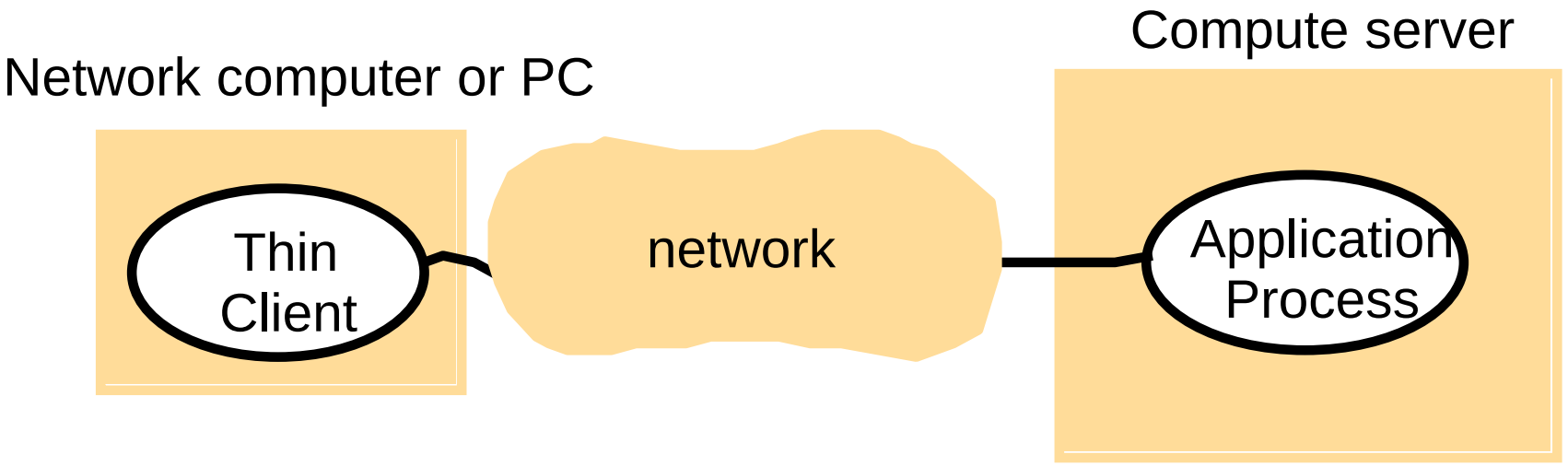
体系结构模式——层次化体系结构

- 层次化 Tiered
  - 与分层体系结构互补，是一项组织给定层功能的技术。
  - 应用角度理解系统
- 例子：web 应用



体系结构模式——瘦客户

- 本地只是一个 GUI，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，只不过那时大型机和哑终端在一个机房。现在可以是通过网络访问服务器。
  - Eg, VNC, Remote Desktop



第2章 系统模型

- 引言
- 物理模型
- 体系结构模型
- 基础模型
- 总结

基础模型

- 对体系结构模型中公共属性的一种更为形式化的描述
  - 交互模型：延迟的不确定性及缺乏全局时钟的影响。
    - 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：组件、网络等故障的定义、分类
    - 试图给出进程和信道故障的一个精确的约定。定义了什么是可靠通信和正确的进程。
  - 安全模型：内、外部攻击定义、分类
    - 讨论对进程的和信道的各种可能的威胁。引入安全通道的概念，它可以保证在存在各种威胁的情况下通信的安全。

交互模型

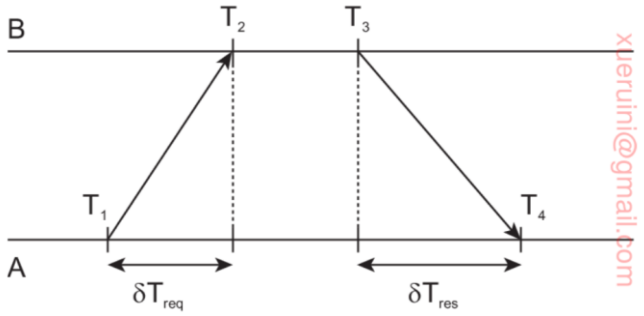
- 分布式系统由多个以复杂方式进行交互的进程组成。
- 进程之间通过消息传递进行交互，实现系统的通信和协作功能。
- 实例
  - 多个服务器进行能相互协作提供服务：域名服务
  - 对等进行能相互协作获得一个共同目标：视频会议
- 两个影响进程交互的因素
  - 通信性能
  - 难以维护全局时钟

交互模型——通信通道的性能

- 延迟 latency
  - 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 bandwidth
  - 在单位时间内，网络上能够传输的信息的总量
- 抖动 jitter
  - 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。

交互模型——计算机时钟

- 每台计算机具有自己的独立内部时钟
- 计算机时钟和绝对时间存在偏移
  - 时钟漂移率（Clock drift rate）：计算机时钟偏移绝对参考时钟的比率
  - 时钟校正方法
    - ntp
    - GPS时钟同步



$$\tilde{d}_i - c \cdot \Delta_r = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$$

交互模型——两个变体(variants)

- 分类依据
  - 是否对时间有严格的假设限制
- 两个变体
  - 同步分布式系统：有严格的时间限制假设
  - 异步分布式系统：无严格的时间限制假设，非常常见

交互模型——两个变体

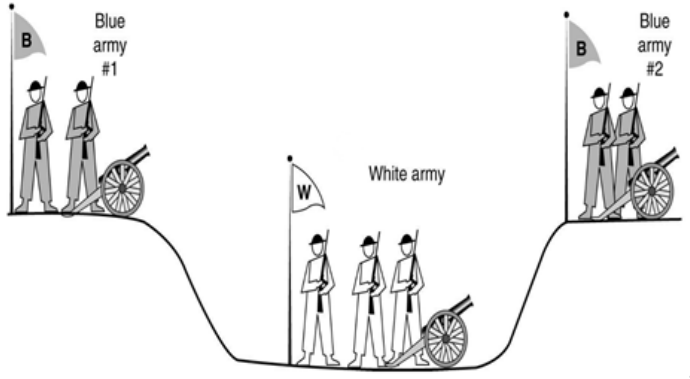
- 同步分布式系统
  - 进程执行每一步的时间有一个上限和下限。
  - 每个在网络上传输消息可在已知时间范围内接收到。
  - 每个进程的局部时钟相对于实际时间的漂移在已知的范围内

交互模型——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 Process execution speeds
    - 每一步都可能需要任意长的时间
  - 消息传递延迟 Message transmission delays
    - 收到一个消息的等待时间可能任意长
  - 时钟漂移率 Clock drift rates
    - 漂移率可能是任意的
- 异步分布式系统
  - Email
  - FTP

交互模型——两个变体

- 异步分布式系统达成协定非常困难
- Pepperland协定/(两军问题的区别)
  - 红师、蓝师分别驻扎在两个山顶，敌人位置他们之间的山谷；
  - 红、蓝师之间靠通信兵进行通信，假设通信兵不会被捕获；
  - 红、蓝师能否对是否冲锋达成一致？
    - **Yes**。只关心结论，等的时间长没关系
  - 红、蓝师能否对冲锋时间达成一致？
    - **No**。接收太久冲锋时间失效



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——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

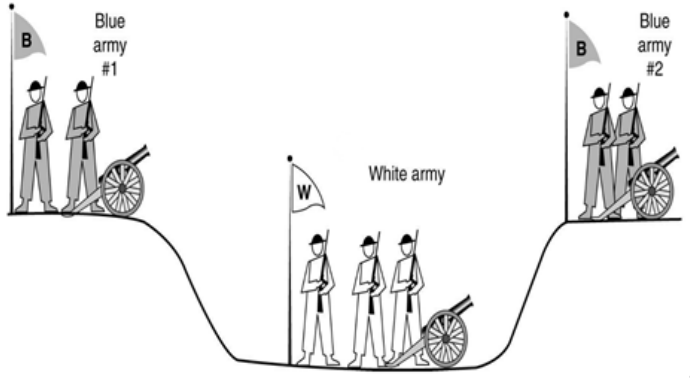
故障模型——遗漏故障

- 进程或者通信通道没有正常的工作
- 进程遗漏故障：进程崩溃 crash
  - Fail-stop：在同步系统中通过时限是可以检测出来的。
  - 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，接收丢失，通道丢失
- 丢失故障是**良性故障**

故障模型——遗漏故障

- 面对通信故障时达成协定的不可能性
- Pepperland协定
  - 红师、蓝师分别驻扎在两个山顶，敌人位于他们之间的山谷；
  - 敌人可单独击败任一师；
  - 红、蓝师之间靠通信兵进行通信，假设通信兵可能会被捕获，但不会叛变；
  - 红、蓝师能否对是否冲锋达成一致？

Never!



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——时序故障

- 仅仅发生在同步分布式系统中
- 异步系统无时间保证，故不存在此类故障

Class of failure	*Affects*	*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

- 分布式系统中每个组件通常基于其他一组组件构造，利用存在故障的组件构造可靠的服务是可能的。
- 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。

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，以此分析系统可能受到的侵害，并在设计系统时防止这些侵害的发生。

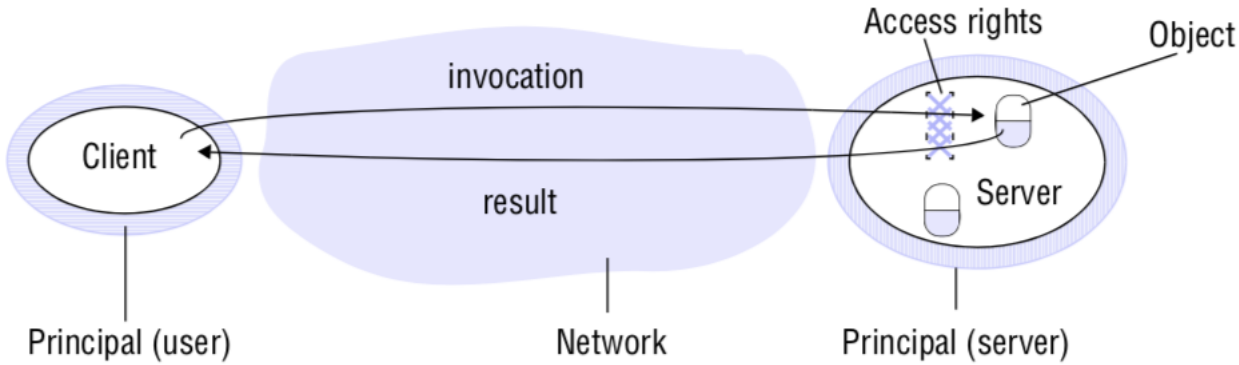
基础模型——安全模型

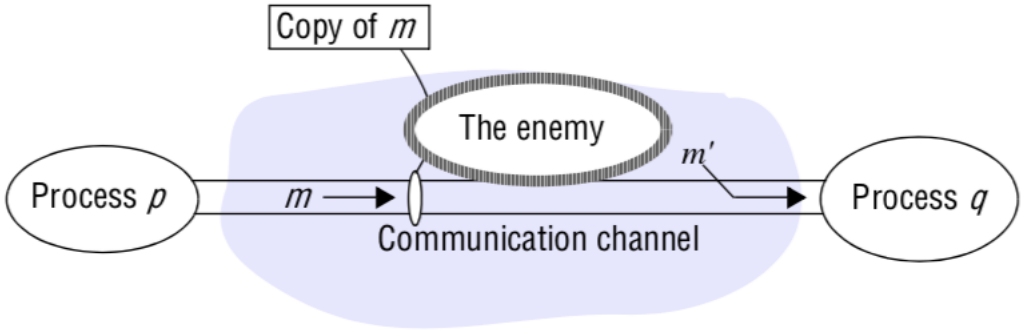
- 分布式系统的安全性
  - 进程的安全性
  - 通信信道的安全性
  - 对象的安全性
- 保护对象的措施
  - 访问权限：在对象的访问控制表中规定什么人具有访问对象的权限
  - 权能：用户一方所持有的访问那些对象的权限

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：CSRF
  - 对客户端：收到一个伪造的结果（钓鱼），例如：窃取密码
- 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


Access rights
Object
invocation
Client
result
Server
Principal (user)
Network
Principal (server)


Copy of m
The enemy
m'
Process p
m
Process q
Communication channel

安全模型——威胁

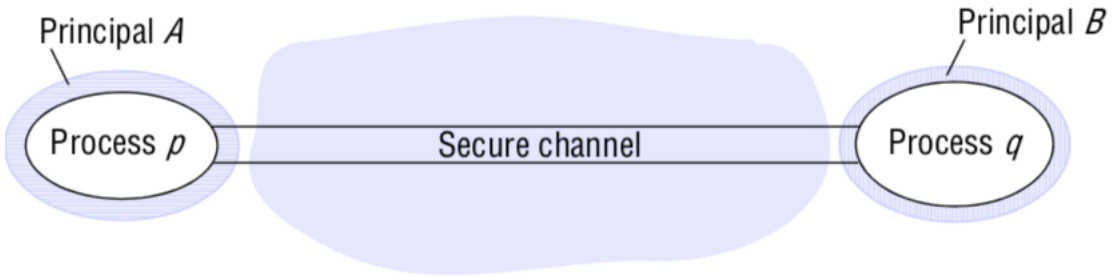
- DoS攻击
  - 发出大量的服务请求，造成服务器过载不能提供服务，或者在网上传送大量的消息，占用带宽，拥塞网络。
- 移动代码
  - 恶意代码入侵系统，例如：特洛伊木马。

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。

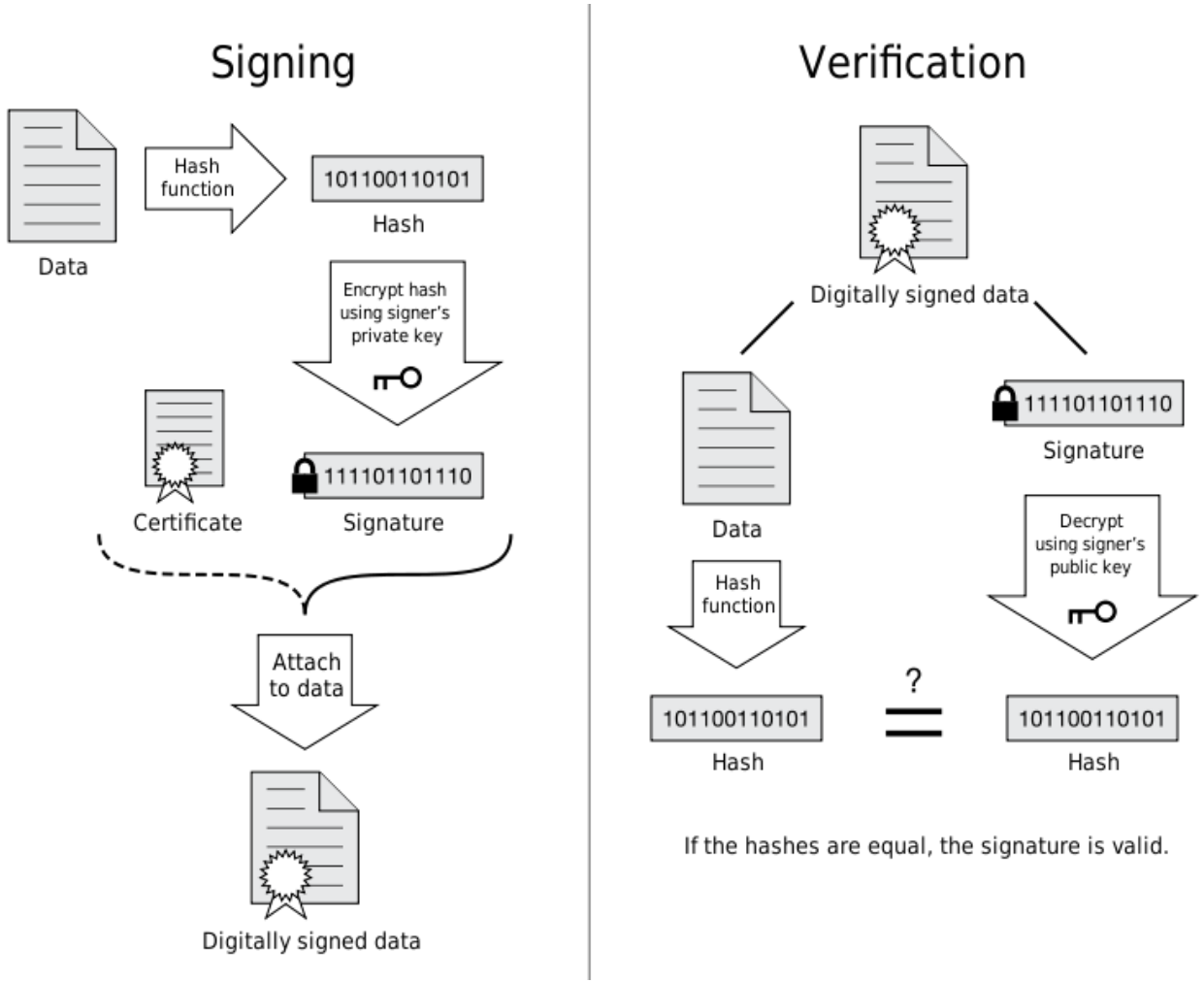
安全模型—对策

- 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。



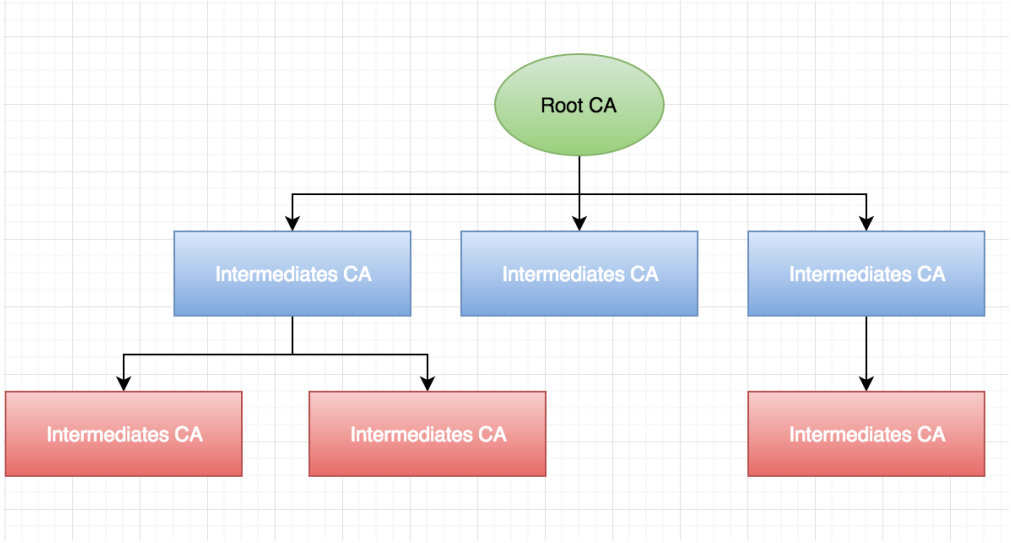
- Discussion: How does HTTPS work?

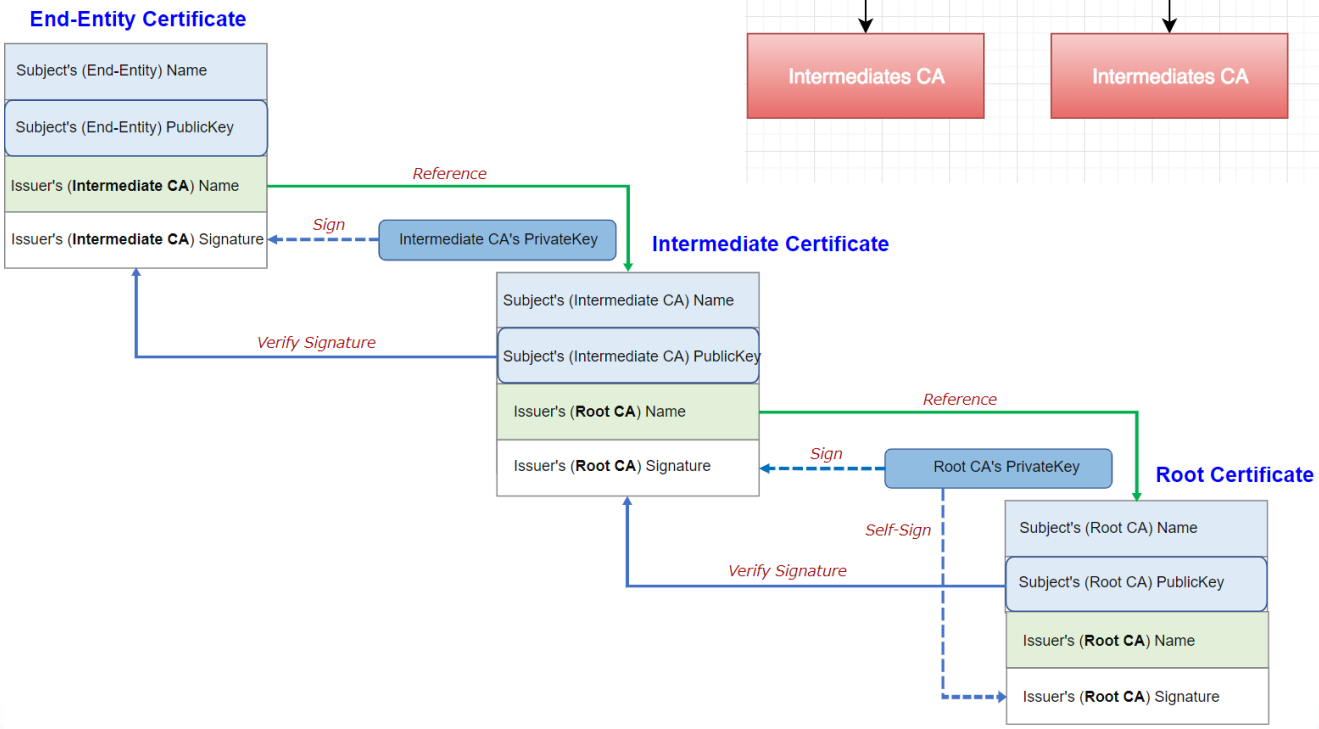
Signing and Verification



CAs

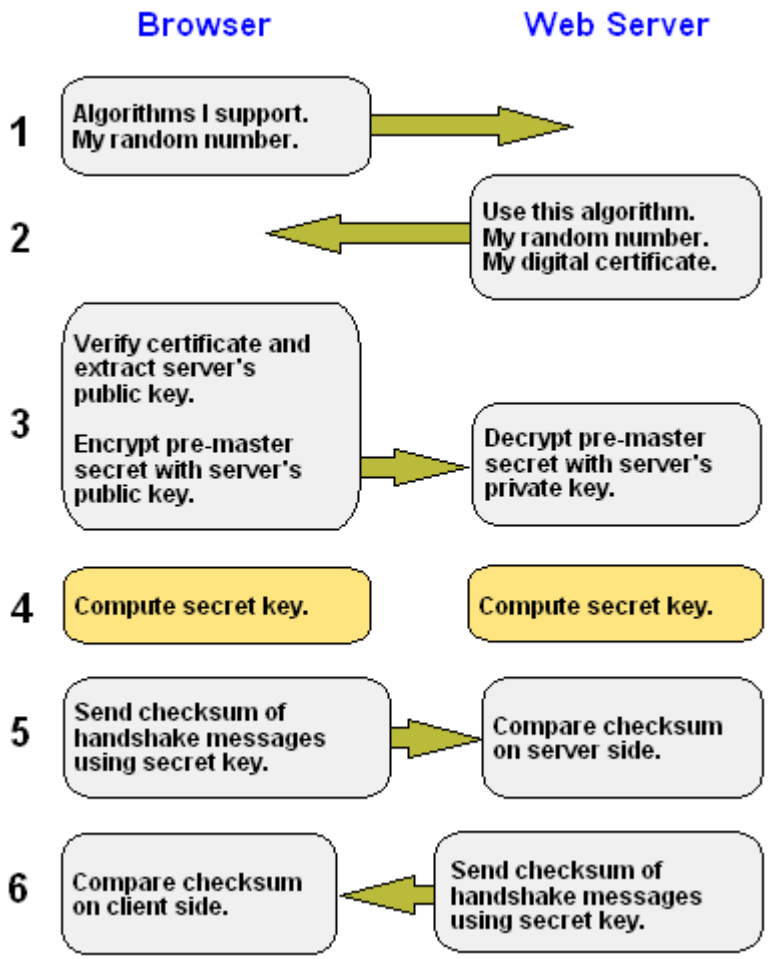
- End-user's certificate
- Intermediates CA
- Root CA





How to setup SSL link?

- negotiate symmetric key
  - pre-master, the 3rd random number is used to generate
- during the session
  - all data are encrypted
- recall the insecure GET and POST



第2章 系统模型

- 引言
- 体系结构模型
- 基础模型
- 总结

体系结构模型

- Architectural elements
  - Communicating entities
    - process, thread, object, component
  - Communication paradigms
    - inter-process communication, remote invocation, indirect communication
  - Roles and responsibilities
    - client/server, peer-to-peer
  - Placement
    - mapping of servers to multiple servers, caching, mobile code, mobile agent
- Architectural pattern
  - layering, tired architecture, thin client

基础模型及安全模型

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

Thanks