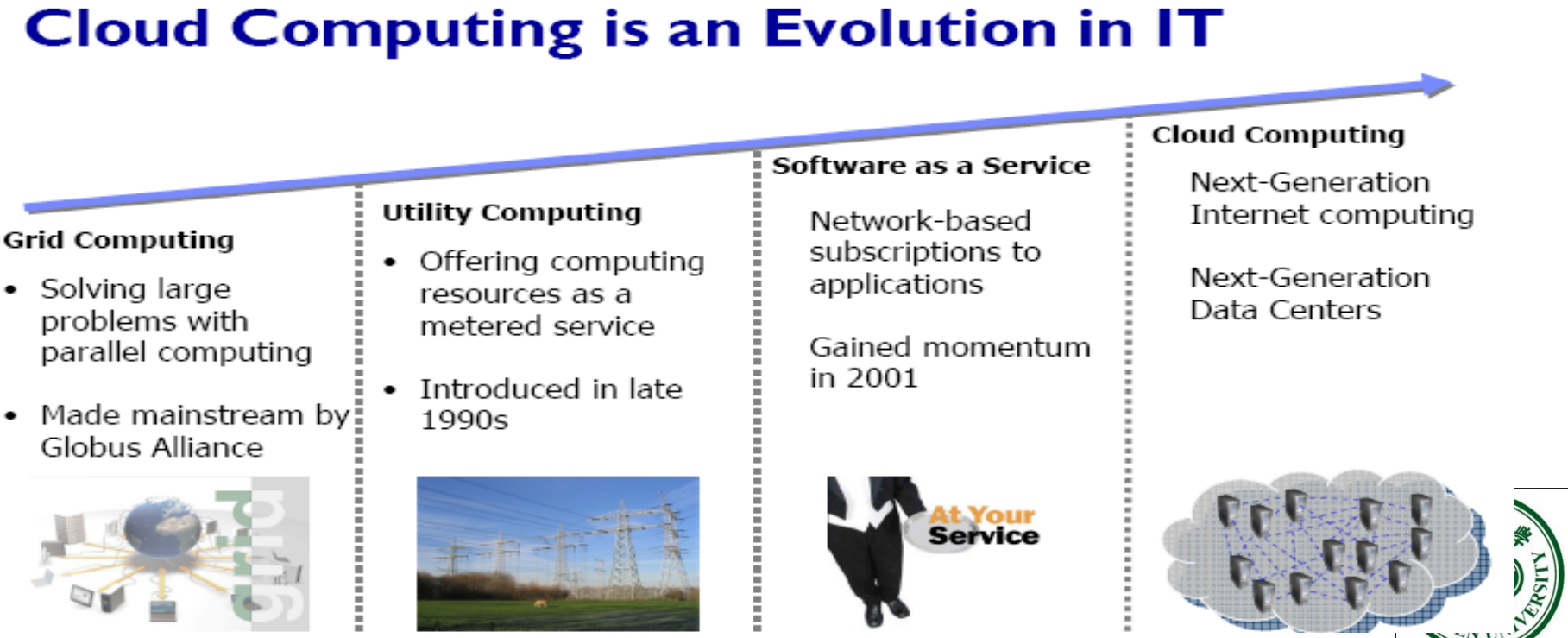
云计算概论期末复习

一、基本概念与模型
二、关键技术
三、主要机制和架构

一、基本概念与模型

- **Cloud computing** is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a utility (like the electricity grid) over a network (typically the Internet). –from NIST



云计算服务

云计算 =数据*（软件+平台+基础设施）* 服务

- 数据（Data）
  - 爆炸增长（传感器、物联网）：1.2ZB = 10^{21}B
  - 各个领域各个层面
- 软件（Software）
  - 检索、发现、关联、处理和创造数据
- 平台（Platform）：
  - “云计算”时代也会诞生自己的通用平台
- 基础设施（Infrastructure）
  - 存储资源、计算资源等
- 服务（Service）
  - IT服务化：产品→服务 XaaS

交付模型

- IaaS、PaaS、SaaS
- IaaS + PaaS
- IaaS + PaaS + SaaS
- PaaS + SaaS?!
- IaaS + SaaS?!

云部署模型

- 公有云Public
- 社区云Community
- 私有云Private
- 混合云Hybrid

两大技术流派

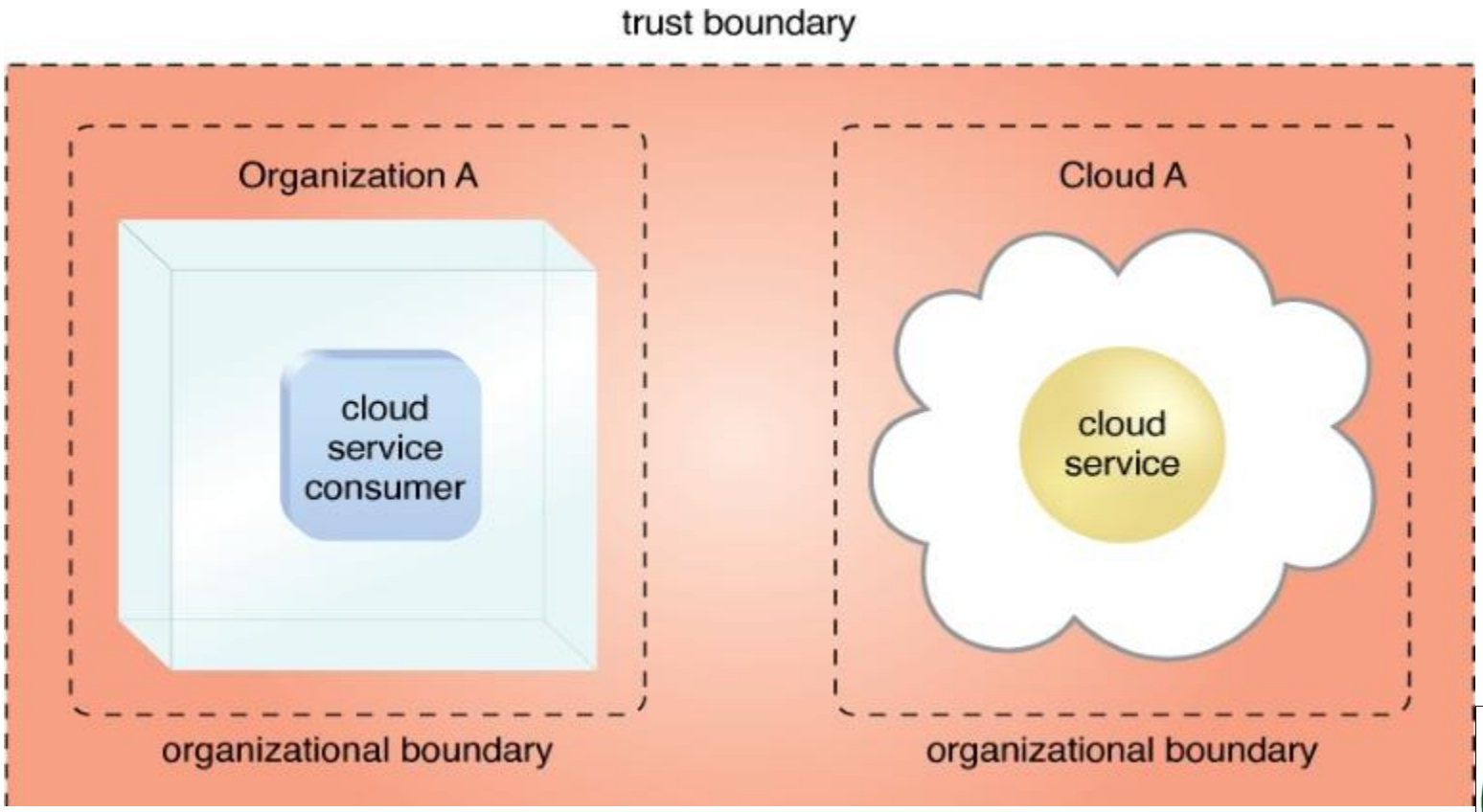
互联网云	IT云
分布式架构	虚拟化架构
公有云	私有云

基本概念和术语

- 可扩展性
  - 水平扩展（horizontal scaling）：内外扩展
  - 垂直扩展（vertical scaling）：上下扩展

角色与边界

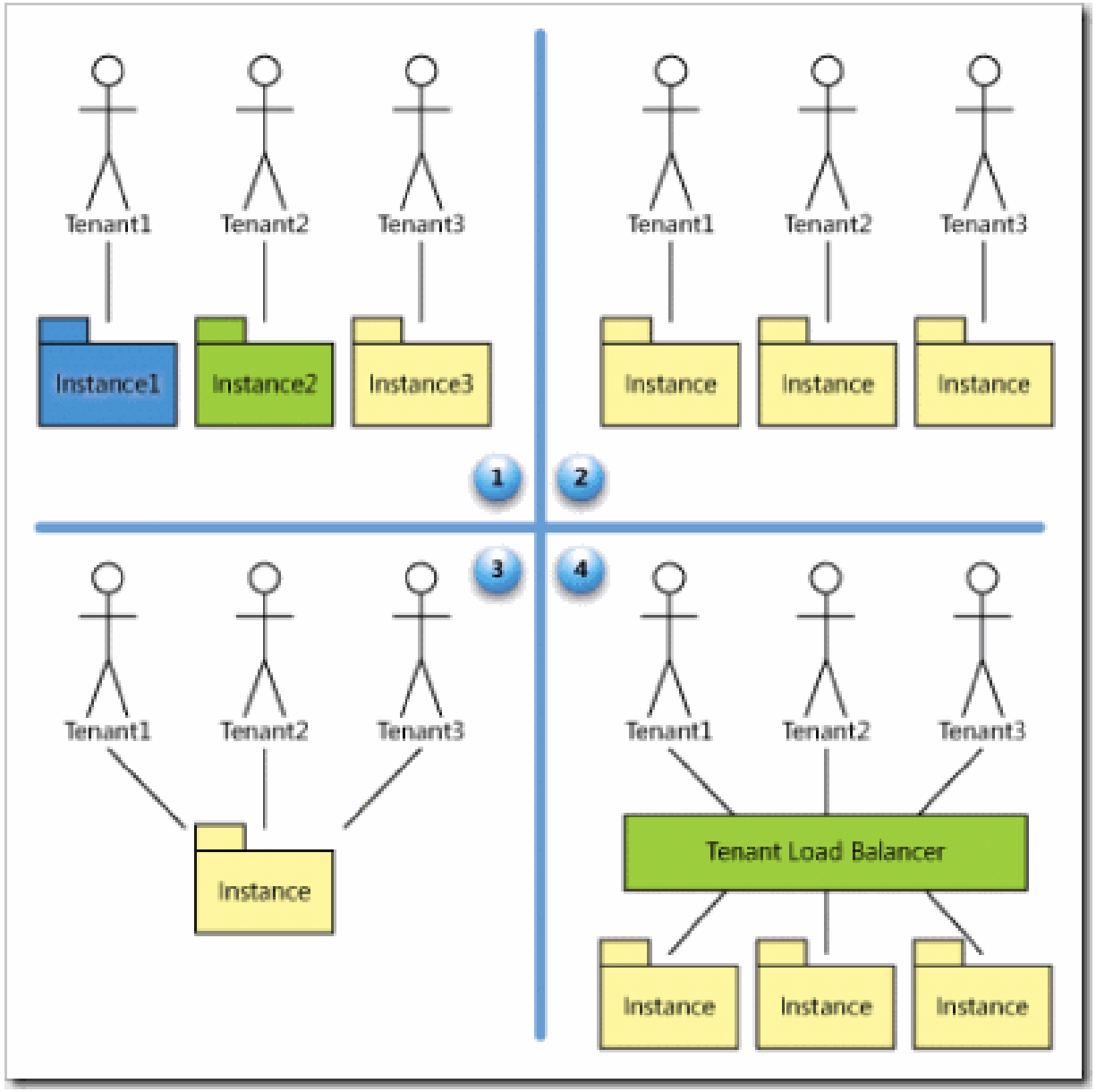
- 云用户、云提供者、云服务拥有者、云资源管理者
- 云组织边界、信任边界



云特性

- 按需使用On-demand usage
  - 基于服务的特性和使用驱动的特性。
- 随处访问Ubiquitous usage
  - 需要支持一组设备、传输协议、接口和安全技术。
- 多租户Multitenancy (and Resource Pooling)
  - 多个云用户共享软件和实例。
  - 可以根据云服务用户的需求动态分配IT资源。
- 弹性Elasticity
  - 与降低投资和与使用比例的成本这些好处紧密地联系在一起。
- 可测量的使用Measured Usage
  - 云平台对云用户使用的IT资源使用情况的记录能力。
- 可恢复性Resiliency
  - 是一种故障转移的形式。
  - 在多个物理位置分放IT资源的冗余实现。

多租户


Tenant1
Tenant2
Tenant3
Instance1
Instance2
Instance3
Tenant1
Tenant2
Tenant3
Instance
Instance
Instance
1
2
3
4
Tenant1
Tenant2
Tenant3
Instance
Tenant1
Tenant2
Tenant3
Tenant Load Balancer
Instance
Instance
Instance

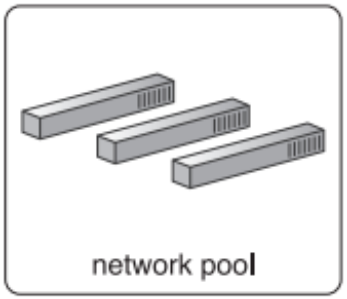
已就绪环境

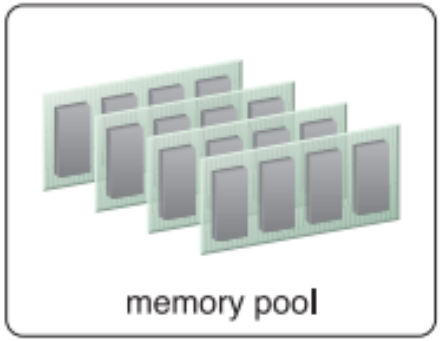
- 已就绪环境机制是PaaS云交付模型的定义组件，代表的是预定义的基于云的平台，该平台由一组已安装的IT资源组成，可以被云用户使用和定制。

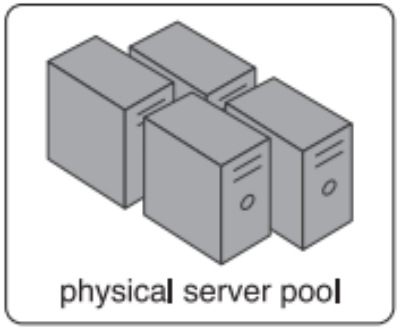
- 已就绪环境被云用户用来在云内远程开发和配置自身的服务与应用程序，其通常配备了一套完整的软件开发工具包（SDK）。

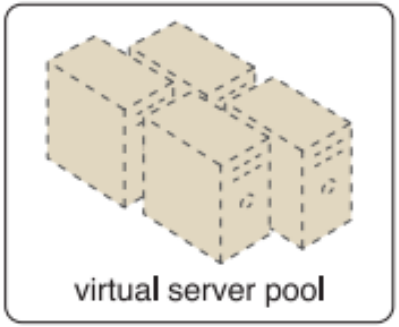
- 典型的已就绪环境包括预安装的IT资源，如数据库、中间件、开发工具和管理工具。

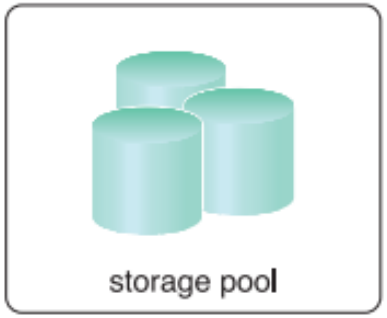
资源池

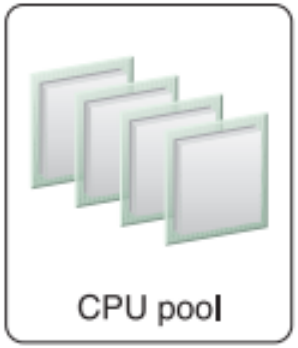


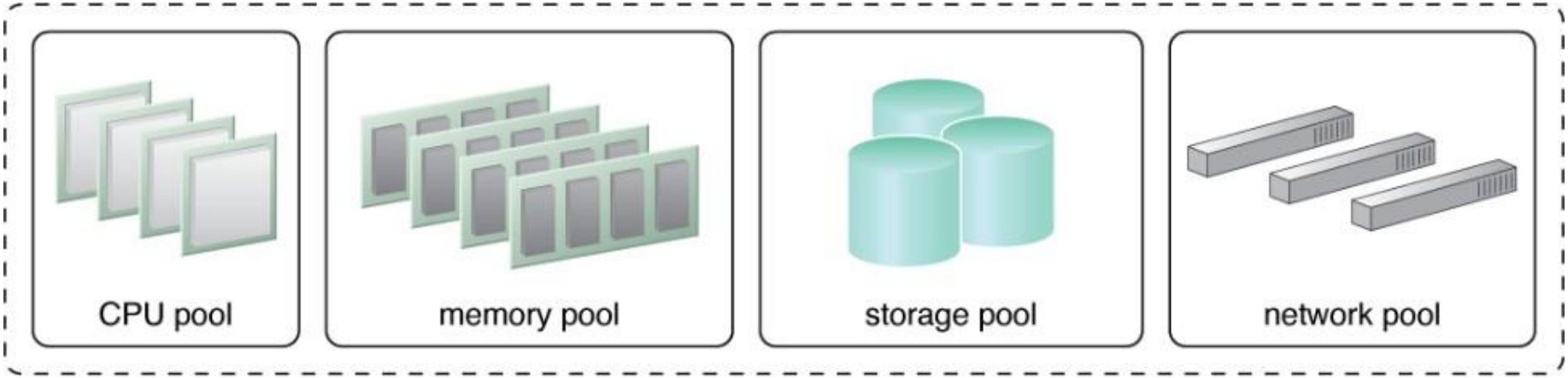












层次资源池

- 资源池可以建立层次结构，形成资源池之间的父子（parent）、兄弟（sibling）和嵌套（nested）关系，从而有利于构成不同的资源池需求。

- 同级资源池（Sibling pools）之间是互相隔离的，云用户只能访问各自的资源池。

- 嵌套资源池（Nested pools）可以用于向同一个云用户组织的不同部门或者不同组分配资源池。

云安全概念

- 保密性(confidentiality)、完整性(integrity)、真实性(authenticity)、可用性(availability)
- 威胁(threat)、漏洞(vulnerability)、风险(risk)
- 威胁作用者(threat agent)是引发威胁的实体
  - 匿名攻击者、恶意服务作用者、授信的攻击者、恶意的内部人员

云安全概念

- 授权不足(Insufficient Authorization)攻击是指错误地授予了攻击者访问权限或是授权太宽泛，导致攻击者能够访问到本应该受保护的IT资源
- 虚拟化攻击(Virtualization Attack)利用的是虚拟化平台中的漏洞来危害虚拟化平台的保密性完整性和可用性。
- 信任边界重叠：恶意的云服务用户可以吧目标设定为共享的IT资源，意图损害其他共享同样信任边界的云服务用户或IT资源

加密

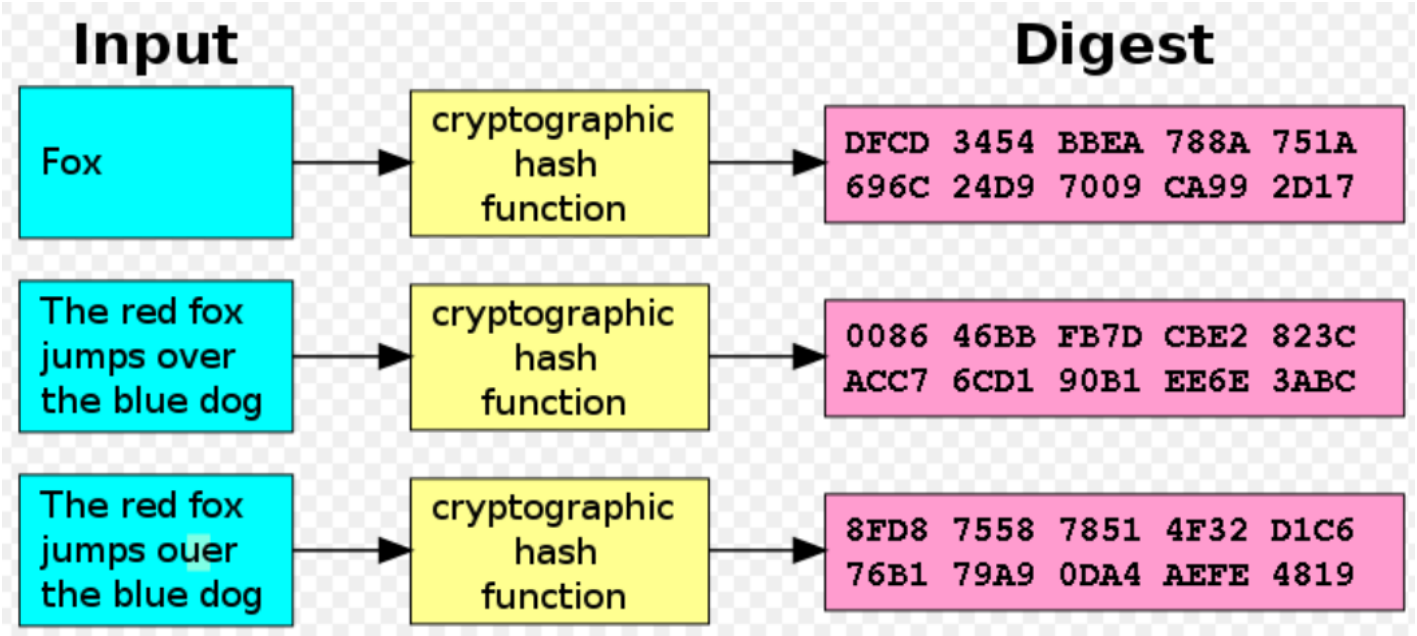
- 加密(encryption)
  - 是一种数字编码系统，专门用来保护数据的保密性和完整性。
  - 用于对抗流量窃听、恶意媒介、授权不足和信任边界重叠这样一些安全威胁。
- 加密部件(cipher)
  - 加密用的标准化算法，把原始的明文数据（Plaintext）转换成加密的密文数据（ciphertext）。
- Key – 密钥

加密方法

- **Symmetric 对称加密**
  - Same key for encryption and decryption
  - Key distribution problem
- **Asymmetric 非对称加密**
  - Mathematically related key pairs for encryption and decryption
  - Public and private keys
- **Hybrid**
  - Combines strengths of both methods
  - Asymmetric distributes symmetric key
    - Also known as a ***session key***
  - Symmetric provides bulk encryption
  - Example:
    - SSL negotiates a hybrid method

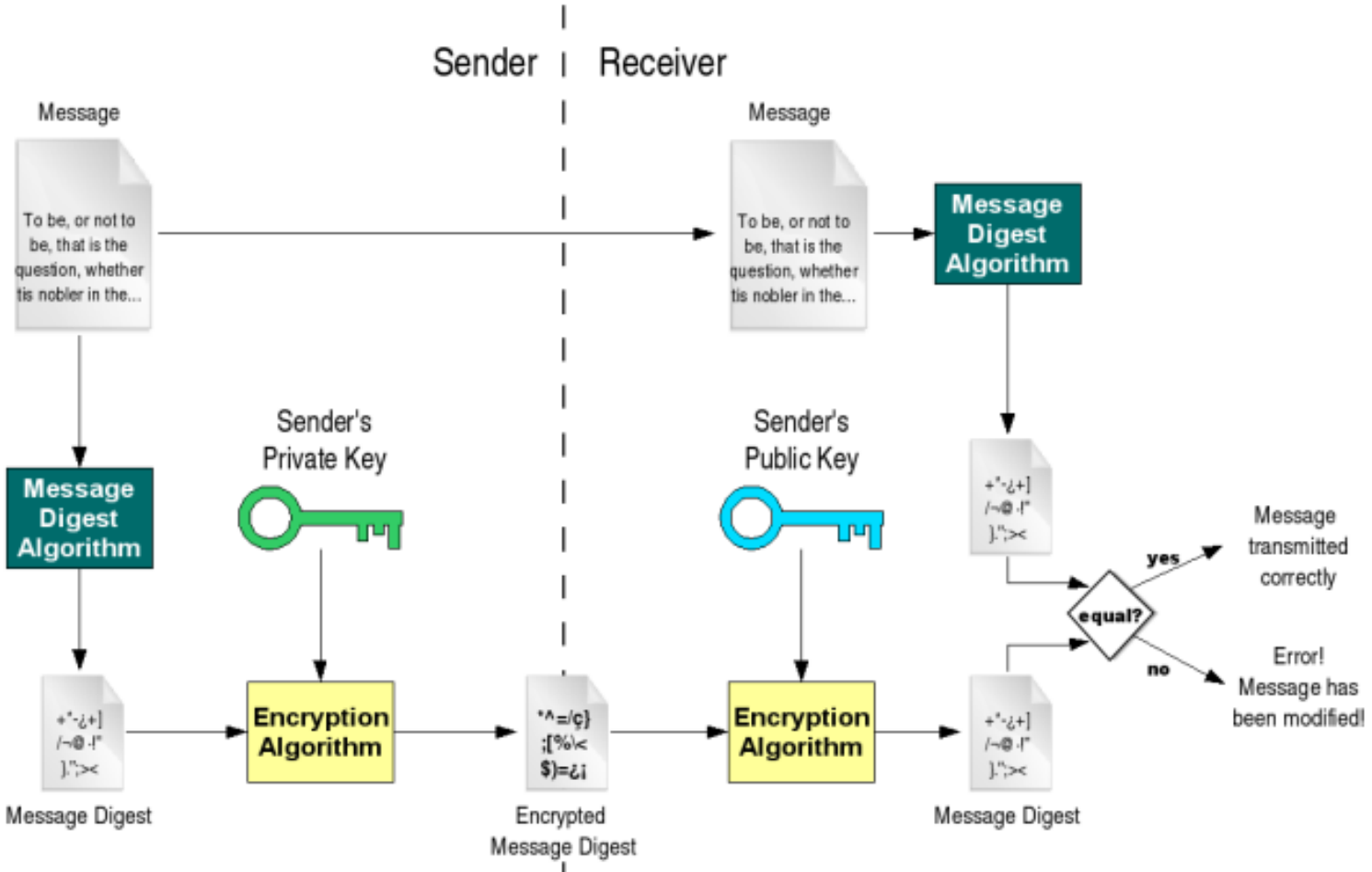
哈希 Hashing

- 哈希(Hashing)
  - 用来获得消息的哈希代码或消息摘要(message digest)，通常是固定的长度，小于原始的消息大小。
  - 单向的、不可逆转的数据保护。



数字签名

- 数字签名(digital signature)
  - 将消息进行哈希，然后用私钥进行加密。



数字签名

- 作用：
  - 提供不可否认性、数据真实性和数据完整性。
  - 类似于日常生活中的“签名”。
- 算法：
  - RSA是最常用的签名算法。
- 扩展应用：
  - 数字证书：个人安全邮件证书
  - 智能卡：将数字签名放入便携物理存储介质

商业成本指标

前期成本
up-front cost

- 获得IT资源的成本
- 部署及管理这些资源的开销

持续成本
on-going cost

- 运行IT资源的开销（软件（执照）费用、电费、人力开销等）
- 保持IT资源的开销（硬件费用、宽带费用、人力开销等）

附加成本

- 资金成本（cost of capital）
- 已支付成本（sunk cost）
- 集成成本（integration cost）
- 锁定成本（locked-in cost）

云使用成本指标

- 网络使用（Network Usage）
- 服务器使用（Server Usage）
- 云存储设备（Cloud Storage Device）
- 云服务（Cloud Service）

网络使用

网络使用定义为网络连接中传输的总数据量，包括以下指标：

	描述	测量值	频率	模型	示例
流入	流入网络流量	Σ，按字节计算	在预订的时间段内连续计算并进行累计	IaaS，PaaS，SaaS	1GB以下免费；10TB以下，一个月为$0.001/GB
流出	流出网络流量	同上	同上	同上	1GB以下免费一个月；10TB以下，一个月为$0.01/GB
云内WAN	同一云内不同地理位置的IT资源的网络流量	同上	同上	同上	每天小于500M免费；500M到1TB之间每月$0.01/GB；超过1TB为每月$0.005GB

服务器使用

根据虚拟服务器和已就绪环境的数量来进行量化。

指标	描述	测量值	频率	模型	示例
按需使用的虚拟机实例	实例的正常运行时间	Σ，启动时间到停止时间	在预订的时间段内，连续计算并进行累计	IaaS，PaaS	小型$0.10/小时，中型$0.20/小时，大型$0.90/小时
预留的虚拟机实例	预留一个实例的前期成本	Σ，预留开始时间到预留结束时间	每天，每月，每年	同上	小型$55.10/小时，中型$99.90/小时，大型$249.90/小时

云存储设备使用

云存储通常按预定义时间内的分配空间总量来收费。

	描述	测量值	频率	模型	示例
按需使用的存储空间指标	以字节为单位的按需使用的存储空间分配的时间与大小	Σ，存储空间释放/重分配时间到存储空间分配时间（当存储空间大小改变时清零）	连续	IaaS，PaaS，SaaS	每小时费用为$0.01/GB(通常表示为GB/月)
I/O数据传输指标	传输I/O数据总量	Σ，按字节计量I/O数据	连续	IaaS，PaaS	$0.10/TB

云服务使用

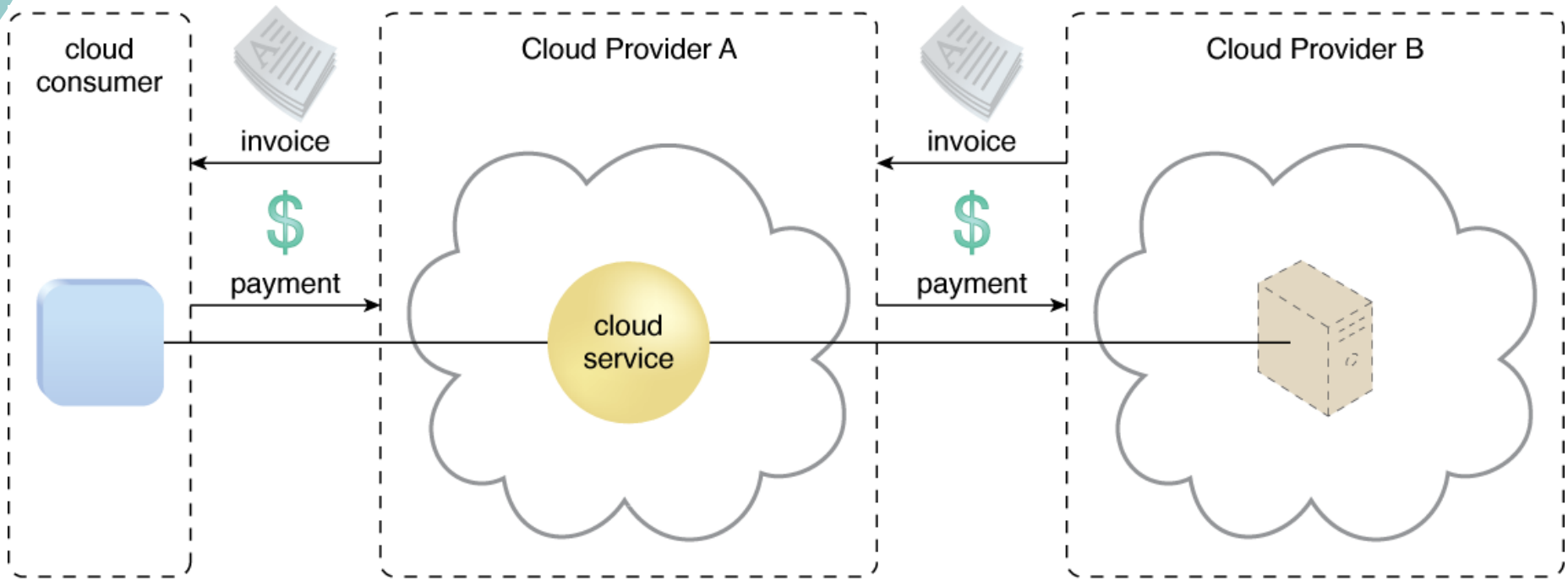
SaaS环境下的云服务使用通常使用下列3种指标来计量：

	描述	测量值	频率	模型	示例
应用订购持续时间指标	云服务使用订购的期限	Σ，订购期限开始时间到结束时间	每天，每月，每年	SaaS	每月费用为$69.90
制定用户数量指标	进行合法访问的注册用户数	用户数量	每月，每年	SaaS	每月每增加一个用户费用为$0.90
用户事务数量指标	云服务提供的事务数量	事务数量（交换请求-响应消息）	连续	PaaS，SaaS	每1000个事务的费用$0.05

定价模型

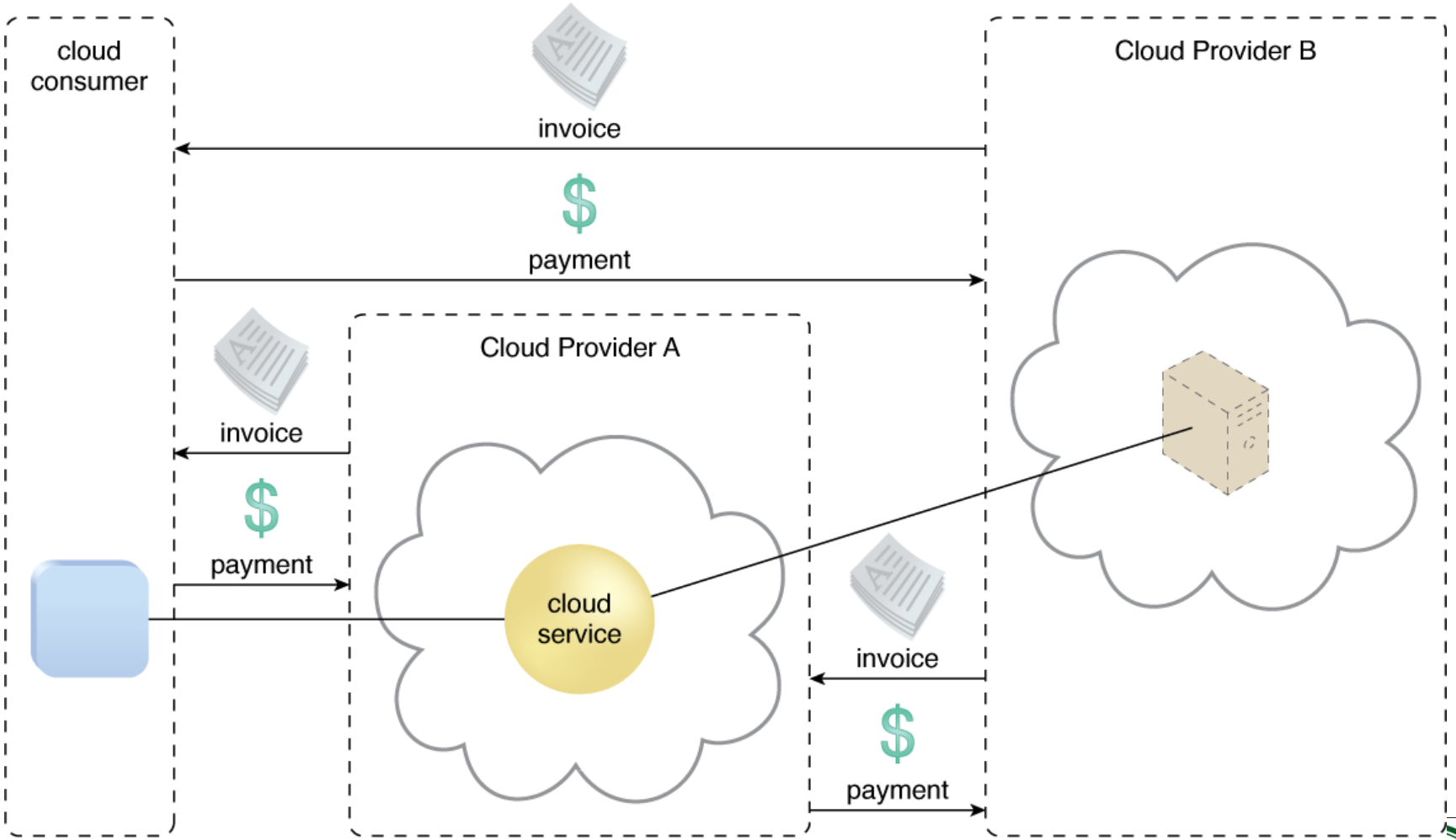
- 价格模板定义定价模型的结构
  - 通过设置各种计量、使用配额、折扣以及其他费用的单位。
- 一个定价模型可以包括多个价格模板，其具体内容由下列因素决定：
  - 成本指标和相关价格（Cost Metric And Associated Price）
  - 固定费用和浮动费用的定义（Fixed And Variable Rates Definition）
  - 使用量折扣（Volume Discount）
  - 成本与价格定制选项（Cost And Price Customization Option）

综合定价模型



SaaS + IaaS

独立定价模型



SLA与QoS

- SLA：
  - 服务质量(quality-of-service, QoS) 特性、保证
- QoS特性通过服务质量指标来表达：
  - 可用性(availability)：可用性比率、停用时间
  - 可靠性(reliability)：MTBF、成功%、
  - 性能(performance)：容量、时间
  - 可扩展性(scalability)：最大容量、规格
  - 可恢复性/弹性(resiliency)： MTSO、MTSR

弹性指标

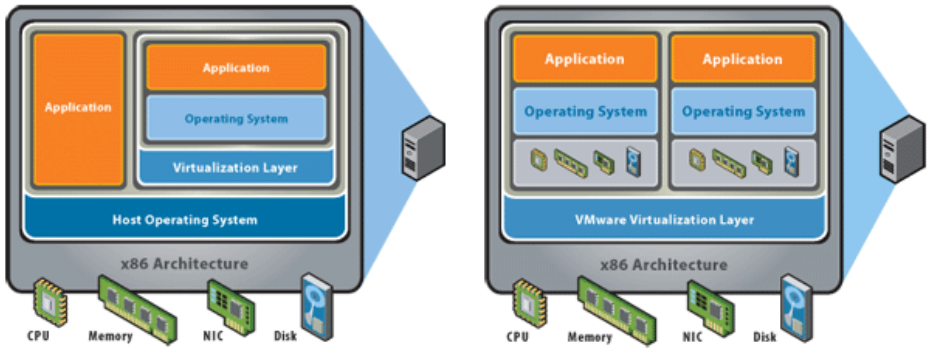
- IT资源从运行问题中恢复的能力。
- 通常是基于在不同物理位置上的冗余和资源复制以及各种灾难恢复系统
- 可以在三个不同的阶段中实施弹性指标，来解决可能威胁到常规服务水平的挑战和事件：
  - 设计阶段(design phase)
  - 运行阶段(operation phase)
  - 恢复阶段(recovery phase)

二、关键技术

- 虚拟化技术
- 分布并行计算技术
  - 大数据处理技术
- 呈现技术、服务技术

2.1 虚拟化技术

- 服务器、网络、存储设备、电源……
- 虚拟机架构

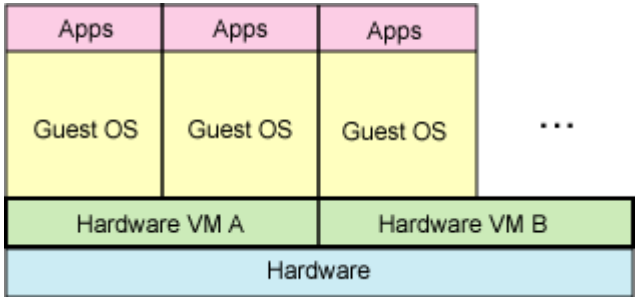


Bare-metal (Type I)	Hosted (Type II)
效率高 （不受Host OS影响、I/O优化）	
安全性高	
	使用方便
	功能丰富（e.g. 3D加速）
适于服务器系统	适于桌面系统

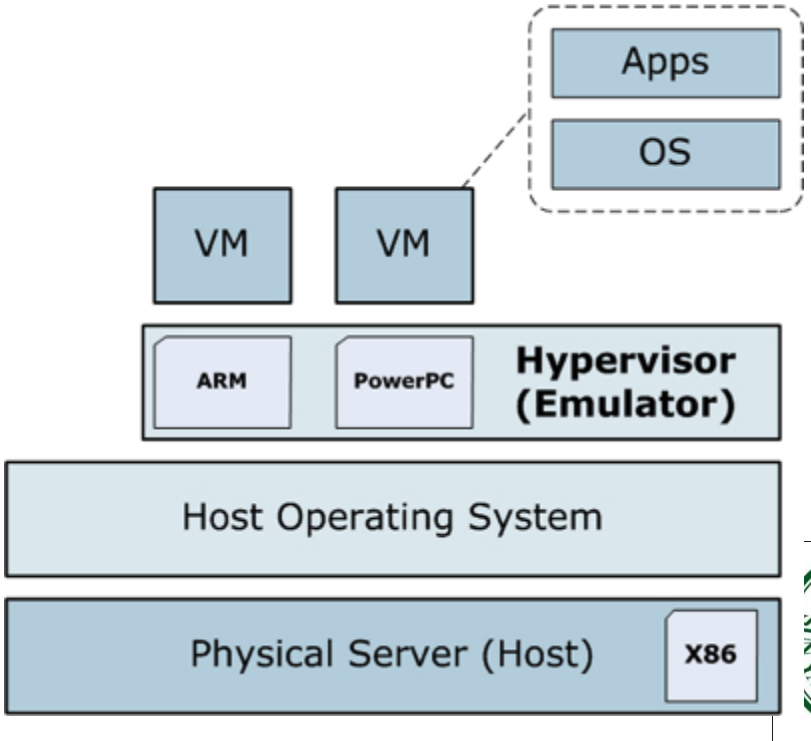
系统虚拟化技术

- 硬件仿真（**Emulation**）
- 全虚拟化（**Full Virtualization**）
- 半虚拟化（**Parairtulization**）
- 硬件辅助虚拟化（**Hardware Assisted Virtualization**）
- 操作系统级虚拟化（**Operating System Level Virtualization**）
  - 容器

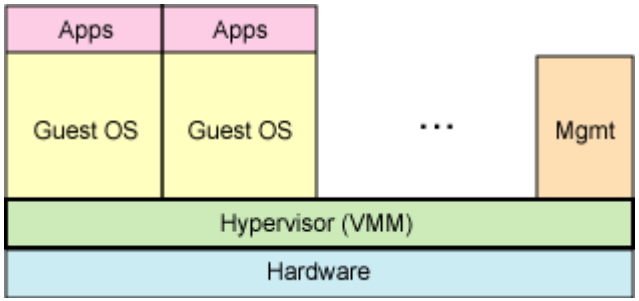
硬件仿真（Emulation）



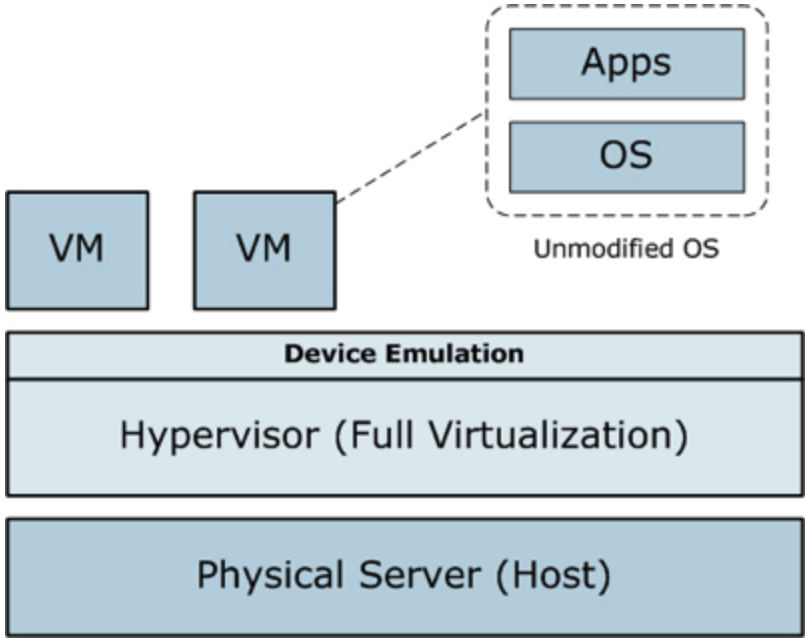
- 属于Hosted架构
- Host OS → VMM (Emulator) → Guest OS。
- 技术要点
  - 特权指令(Guest)>>陷入>>VMM>>模拟>>特权指令(Host)
- 知名产品
  - Bochs (open source)
  - QEMU (free)
  - Virtual PC (Microsoft)



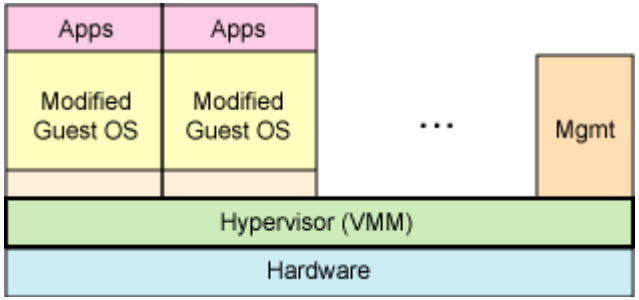
全虚拟化（Full Virtualization）



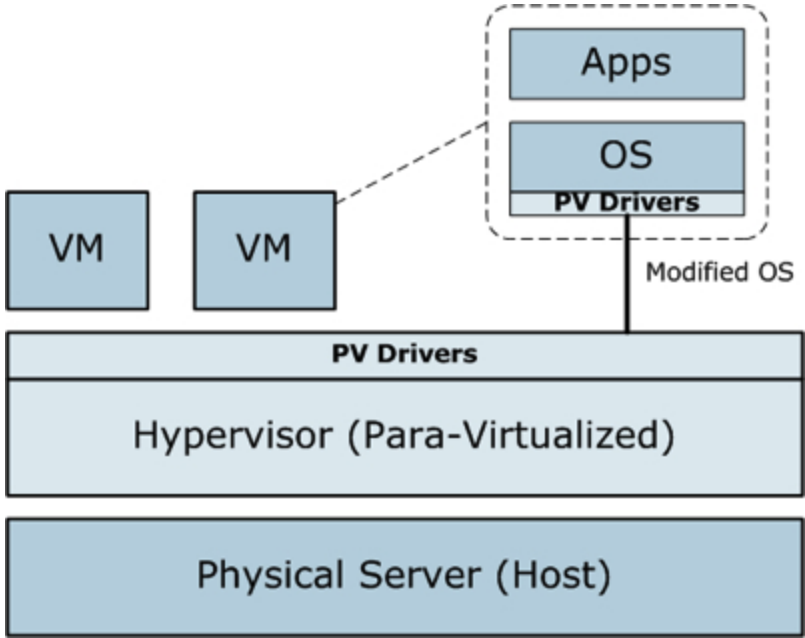
- 最常见、最成熟的虚拟化技术
- Hosted 和 Bare-metal都有
- 技术要点：
  - 在客户操作系统和硬件之间捕捉和处理那些对虚拟化敏感的特权指令，使客户操作系统无需修改就能运行
- 知名产品
  - IBM CP/CMS
  - Oracle VirtualBox
  - KVM
  - VMware ESX



半虚拟化（Parairtulization）



- Bare-metal模式
- 技术要点：
  - 与全虚拟化类似,利用VMM实现对底层硬件的访问
  - Guest OS集成与半虚拟化有关的代码，以配合VMM
  - 无需重新编译或捕获特权指令，性能非常接近物理机
- 知名产品
  - Xen
  - Microsoft Hyper-V

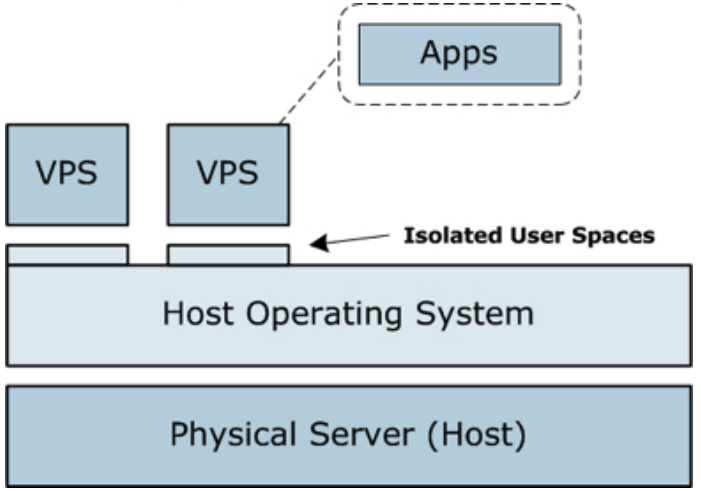


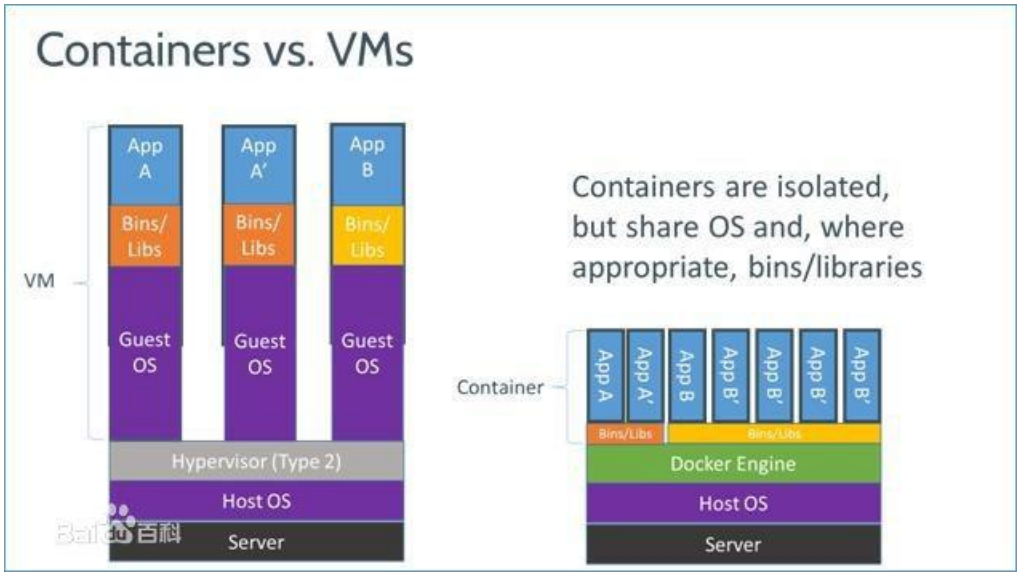
Hardware Assisted Virtualization

- 不是独立的虚拟化技术
- 结合到全/半虚拟化技术中
- 技术要点：
  - 通过对部分全虚拟化和半虚拟化使用到的软件技术进行硬件化来提高性能
- 知名产品
  - Intel VT-x
  - AMD-V (AMD SVM)

Operating System Level Virtualization

- Virtual Private Server、Container（容器）
- 技术要点：
  - 对操作系统的用户空间进行隔离
    (只有一份操作系统）
- 知名产品
  - Linux-Vserver
  - Docker





Comparisons

	硬件仿真虚拟化	全虚拟化	半虚拟化	硬件辅助虚拟化	操作系统级虚拟化
速度	<30%	30%~80%+	80%+	80%+	90%
模式	Hosted	Hosted/Bare-metal	Bare-metal	Hosted/Bare-metal	类Bare-metal
优点	Guest OS无需修改 非常适合硬件、固件及OS的开发	Guest OS无需修改，速度和功能都不错，使用非常简单	比全虚拟化架构更精简，速度上有优势	速度快	成本低 效率高
缺点	速度非常慢（有时速度比物理情况慢100倍以上）	基于Hosted模式时性能较差，特别是I/O方面	需要对Guest OS进行修改，用户体验较差	硬件实现不够优化	OS支持
趋势	颓势但仍存	主流	一定份额	普遍采用	越来越多

CPU全虚拟化（模拟执行）

- 优先级压缩（Ring Compression）技术
  - 优先级压缩能让VMM和Guest运行在不同的特权级下
  - 让VMM截获一部分在Guest上执行的特权指令，并对其进行虚拟化。
  - X86架构：VMM运行在特权级最高Ring 0下，Guest的内核代码运行在Ring 1下，Guest的应用代码运行在Ring 3下。
- 二进制代码翻译技术（Binary Translation）技术
  - 用于一些对虚拟化不友好的指令
  - 通过扫描并修改Guest的二进制代码来将那些难以虚拟化的指令转化为支持虚拟化的指令。

CPU半虚拟化（操作系统辅助）

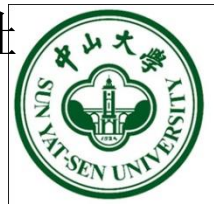
- 基于OS级的支持
- 通过修改Guest OS的代码，使其将那些和特权指令相关的操作都转换会发给VMM的Hypercall（超级调用）
- Hypercall支持Batch（批处理）和异步这两种优化方式，使Hypercall能得到接近物理机的速度。

CPU硬件辅助虚拟化

- 通过引入新的指令和运行模式，来让VMM和Guest OS能分别运行在其合适的模式下
- VT-x支持两种处理器工作方式
  - Root模式（Operation），VMM运行于此模式，用于处理特殊指令
  - Non-Root模式（Operation），Guest OS运行于此模
  - 当在Non-Root 模式Guest执行到特殊指令的时候，系统会切换到运行于Root模式VMM，让VMM来处理这个特殊指令。

内存虚拟化

- **目标：**
  - 做好虚拟机内存空间之间的隔离，使每个虚拟机都认为自己拥有了整个内存地址，并且效率也能接近物理机。
- **全虚拟化：**影子页表 Shadow Page Table
  - 为每个Guest都维护一个影子页表，写入虚拟化之后的内存地址映射关系，而Guest OS的页表则无需变动
  - VMM将影子页表交给MMU进行地址转换
- **半虚拟化：**页表写入法
  - 当Guest OS创建一个新的页表时，其会向VMM注册该页表
  - 之后在Guest运行的时候，VMM将不断地管理和维护这个表，使Guest上面的程序能直接访问到合适的地址
- **硬件辅助虚拟化：**扩展页表EPT（Extended Page Table）
  - EPT通过使用硬件技术，在原有的页表的基础上，增加了一个EPT页表
  - 通过这个页表能够将Guest的物理地址直接翻译为主机的物理地址

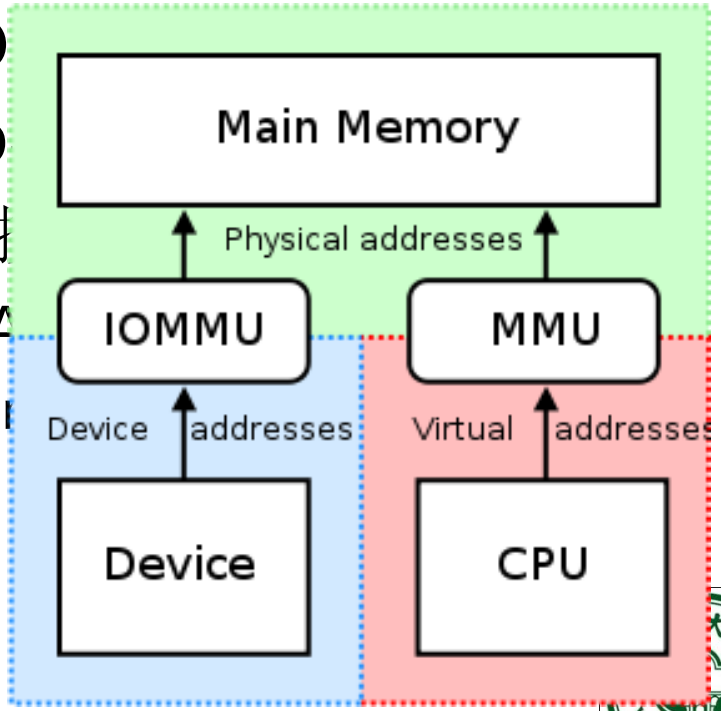


I/O虚拟化

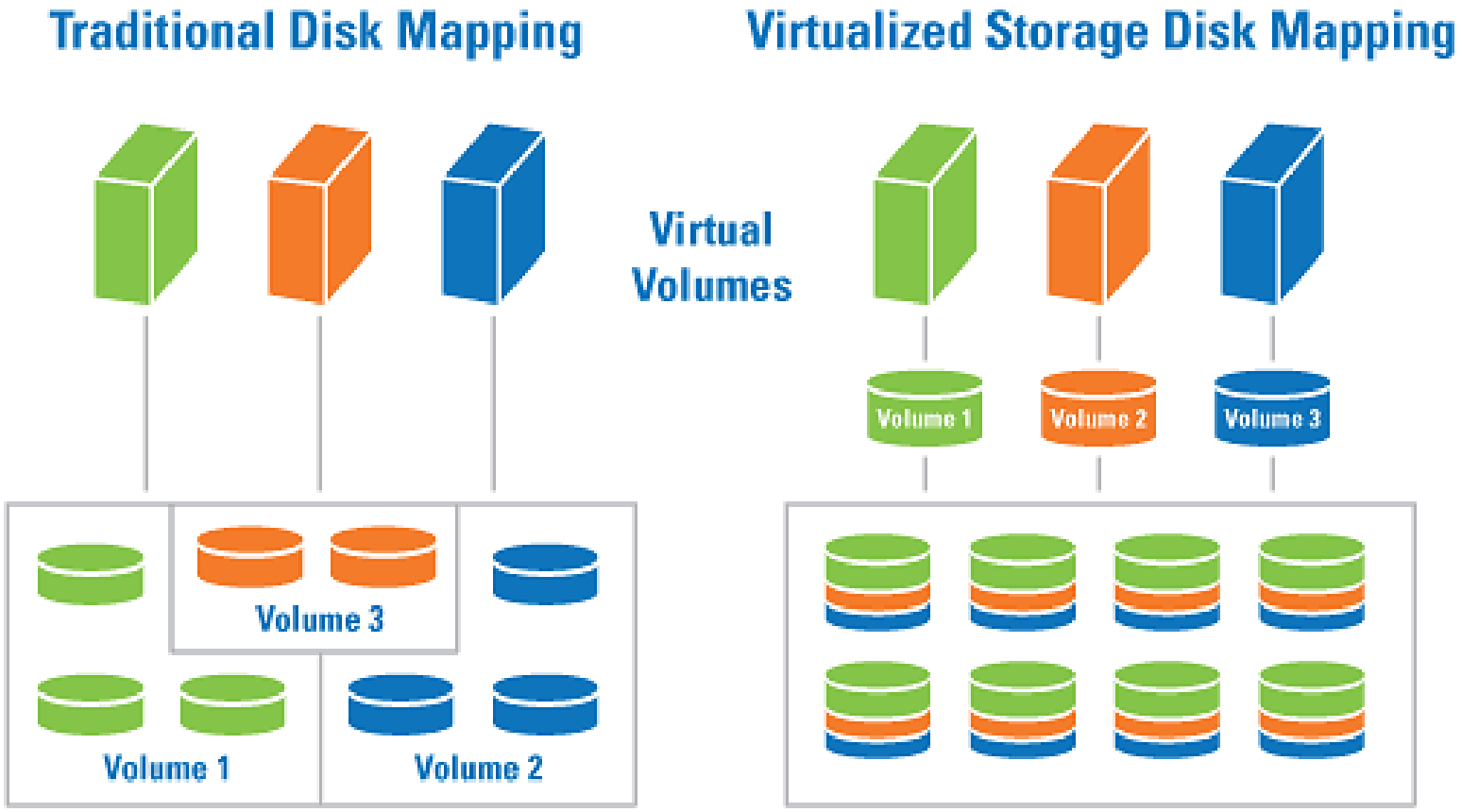
- 目标：
  - 不仅让虚拟机访问到它们所需要的I/O资源，而且要做好它们之间的隔离工作，更重要的是，减轻由于虚拟化所带来的开销。
- 全虚拟化：
  - 通过模拟I/O设备（磁盘和网卡等）来实现虚拟化。
  - Guest OS每次I/O操作都会陷入到VMM，让VMM来执行。
  - 对Guest OS而言，它看到就是一组统一的I/O设备，完全透明。
- 半虚拟化：
  - 通过前端（Front-End）/后端（Back-End）架构，将Guest的I/O请求传递到特权域（Privileged Domain，也被称为Domain-0）。

I/O虚拟化

- **软件模拟虚拟化：**
  - 用软件模拟I/O设备
  - Guest OS的操作被VMM捕获并交给Host OS的用户态进程，由其进行系统调用
- **硬件辅助虚拟化（直接划分）**
  - 代表产品：Intel的VT-d，AMD
  - 其核心思想就是让虚拟机能直
  - 技术要点：I/O地址访问和DMA
    - 通过采用DMA重映射（Remappi

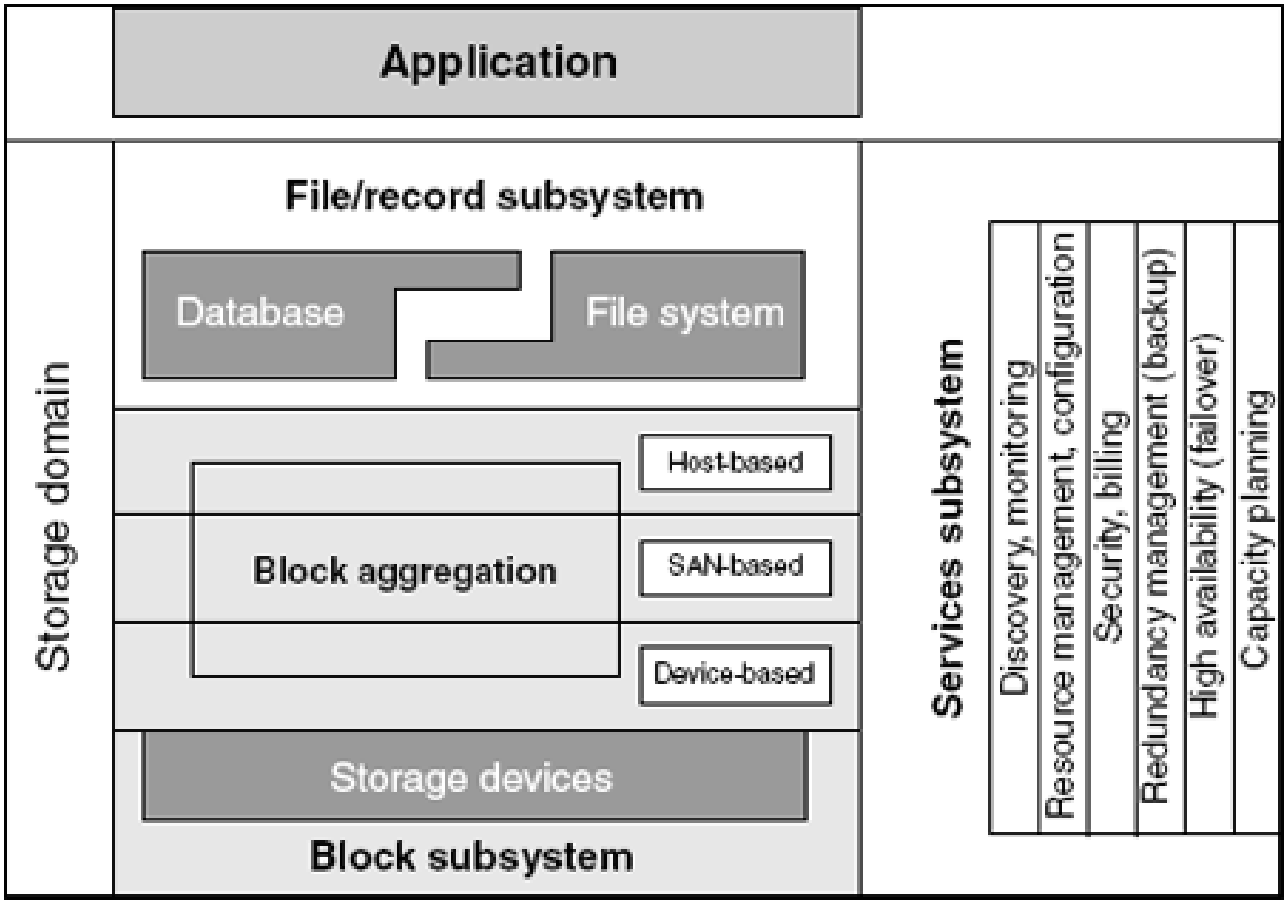


存储虚拟化


Traditional Disk Mapping
Virtualized Storage Disk Mapping
Virtual Volumes
Volume 1
Volume 2
Volume 3
Volume 3
Volume 1
Volume 2

存储系统的基本结构 (from SNIA)

- 文件/记录层：为上层应用提供存储访问接口
- 块聚合层：将底层存储设备聚合为统一存储资源
- 存储设备层：识别数据块存储的物理位置，执行物理设备的数据读写。

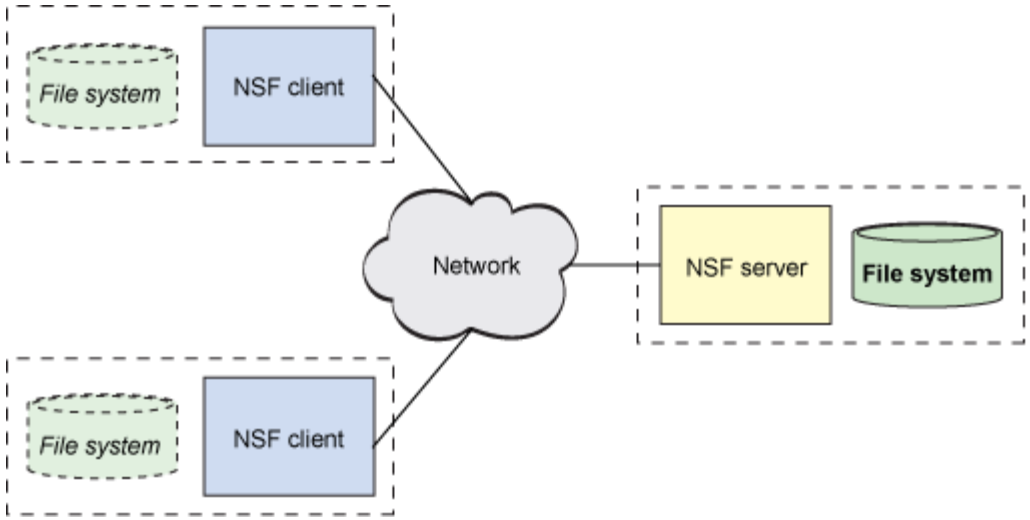


块聚合层虚拟化

- 目标：
  - 整合不同的物理存储设备
- 基本策略：
  - 先将多个物理存储器虚聚合成一个单一的虚拟存储器，提高容量、可靠性等；
  - 再将虚拟存储器划分成多个小存储器分配给用户。
- 虚拟化技术
  - 带内（In-band）虚拟化vs. 带外（Out-of-band）虚拟化

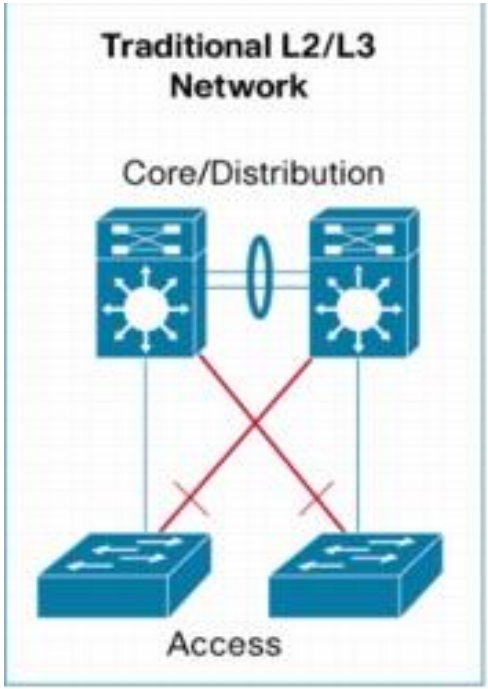
文件/记录层虚拟化

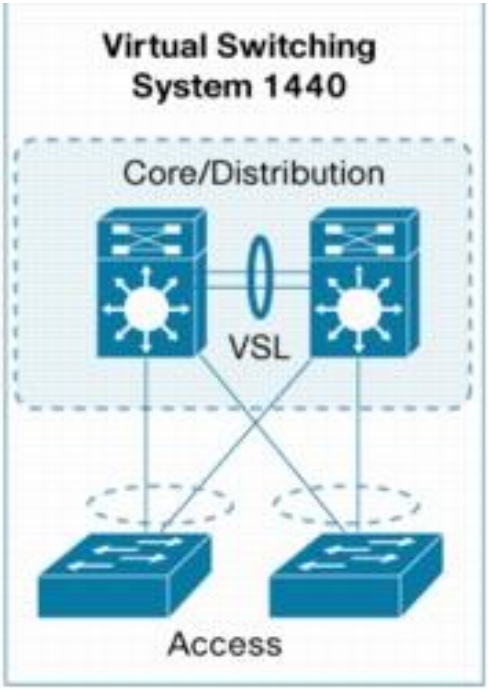
- 目标：
  - 整合不同的文件系统
  - 使上层用户能够透明、高效地访问存储在远程的文件。
- 实现技术
  - NFS、CIFS、NAS
  - 分布式文件系统



网络设备虚拟化

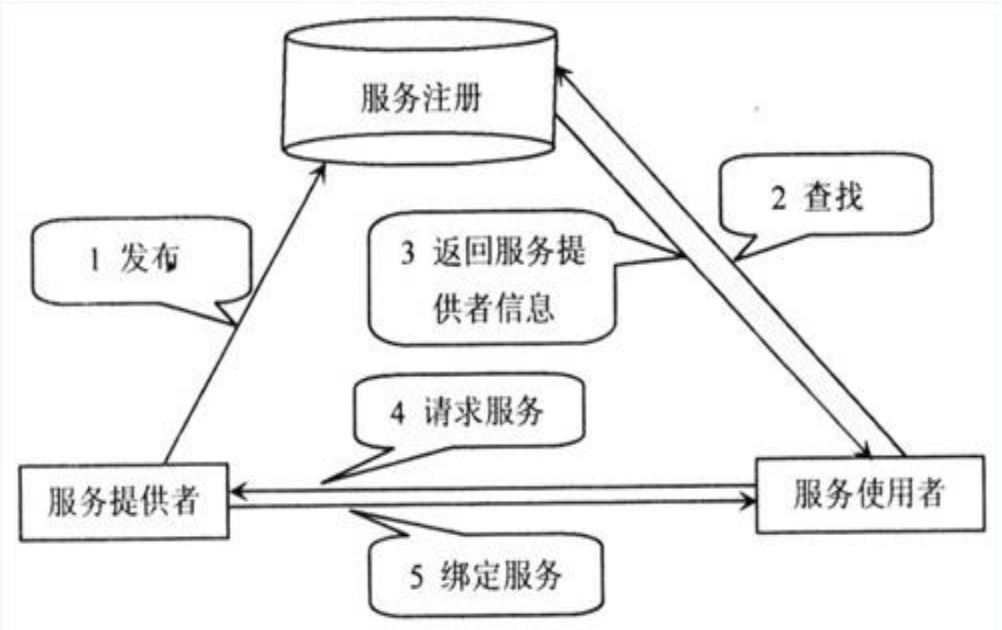
- 适应云计算环境的需求
  - 增加容量、应对横向流量、大缓存低延迟等。
- 技术要点
  - 网络交换设备层面实施虚拟化
- 两种形式
  - 横向整合：多个变一个
  - 纵向分割：VLAN等





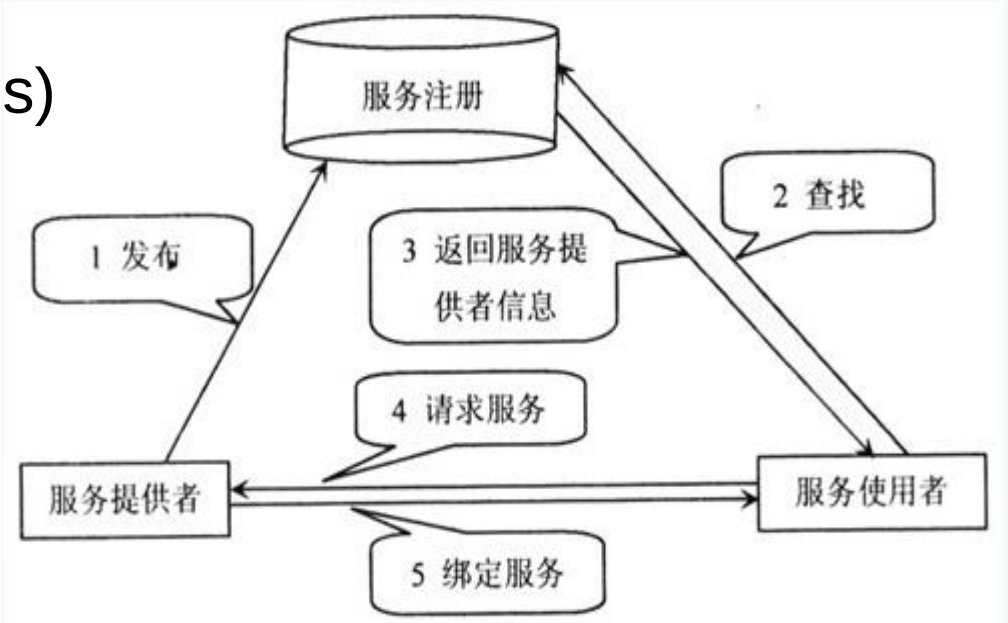
服务技术

- SOA模型
- 三种角色：
  - 服务提供者(Service Provider)
  - 服务请求者(Service Requester/Consumer)
  - 服务代理 (Service Registry/Broker/Repository
- 一种媒介：
  - 服务总线(Service Bus)
- 三个操作
  - 发布(Publish)
  - 查找(Discover)
  - 绑定(Bind)

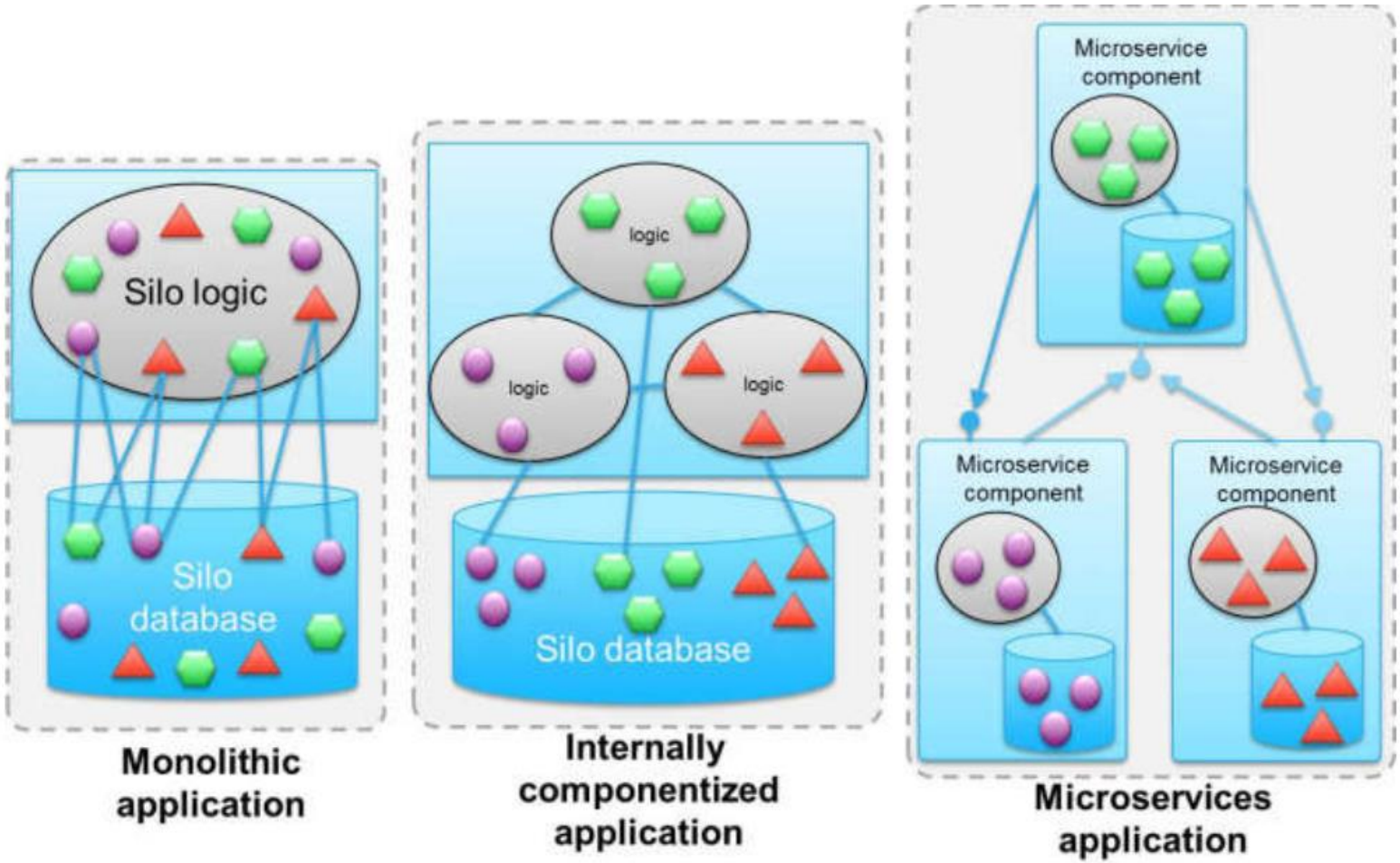


SOA模型

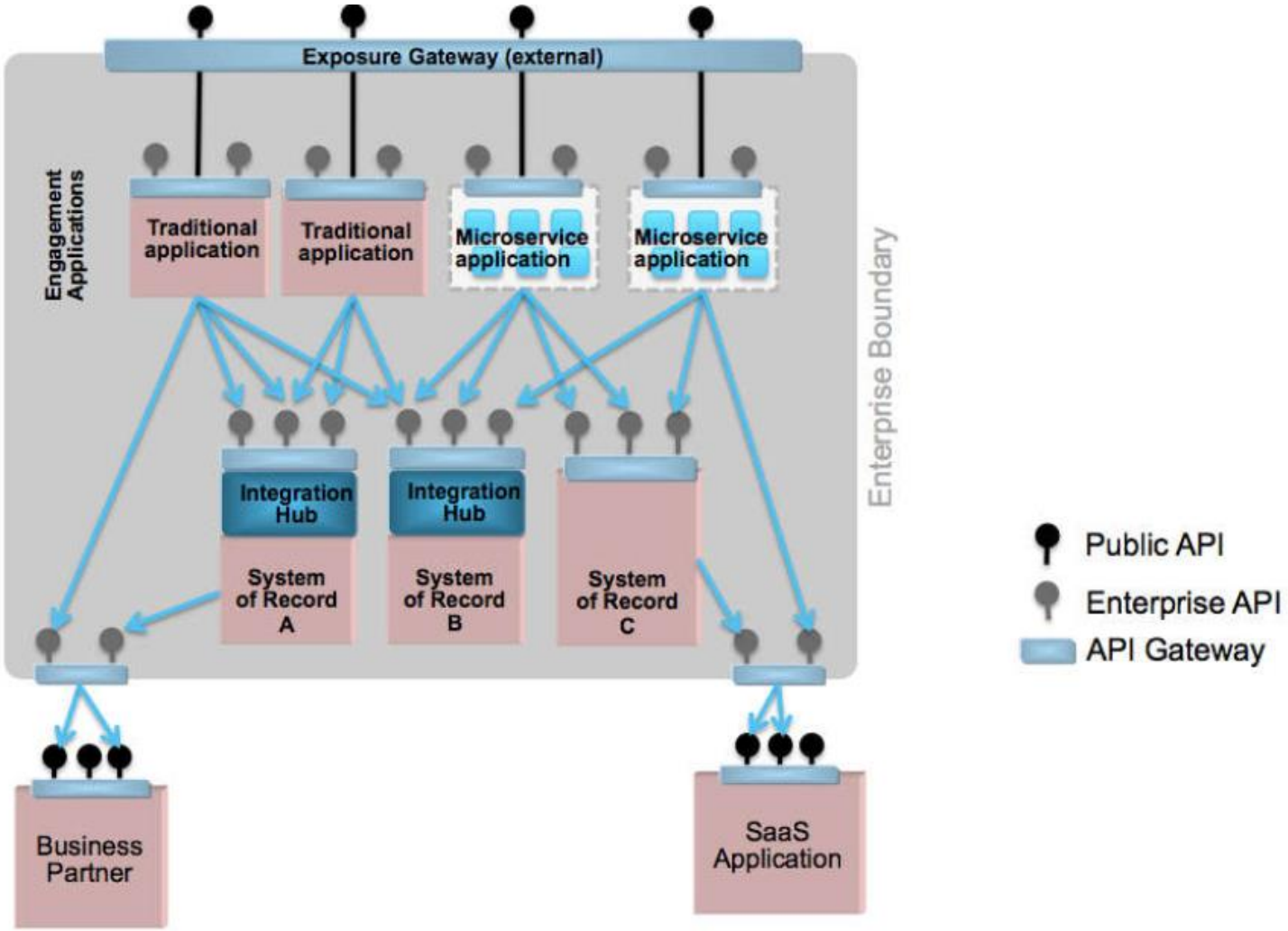
- 三种角色：
  - 服务提供者(Service Provider)
  - 服务请求者(Service Requester/Consumer)
  - 服务代理 (Service Registry/Broker/Repository
- 一种媒介：
  - 服务总线(Service Bus)
- 三个操作
  - 发布(Publish)
  - 查找(Discover)
  - 绑定(Bind)



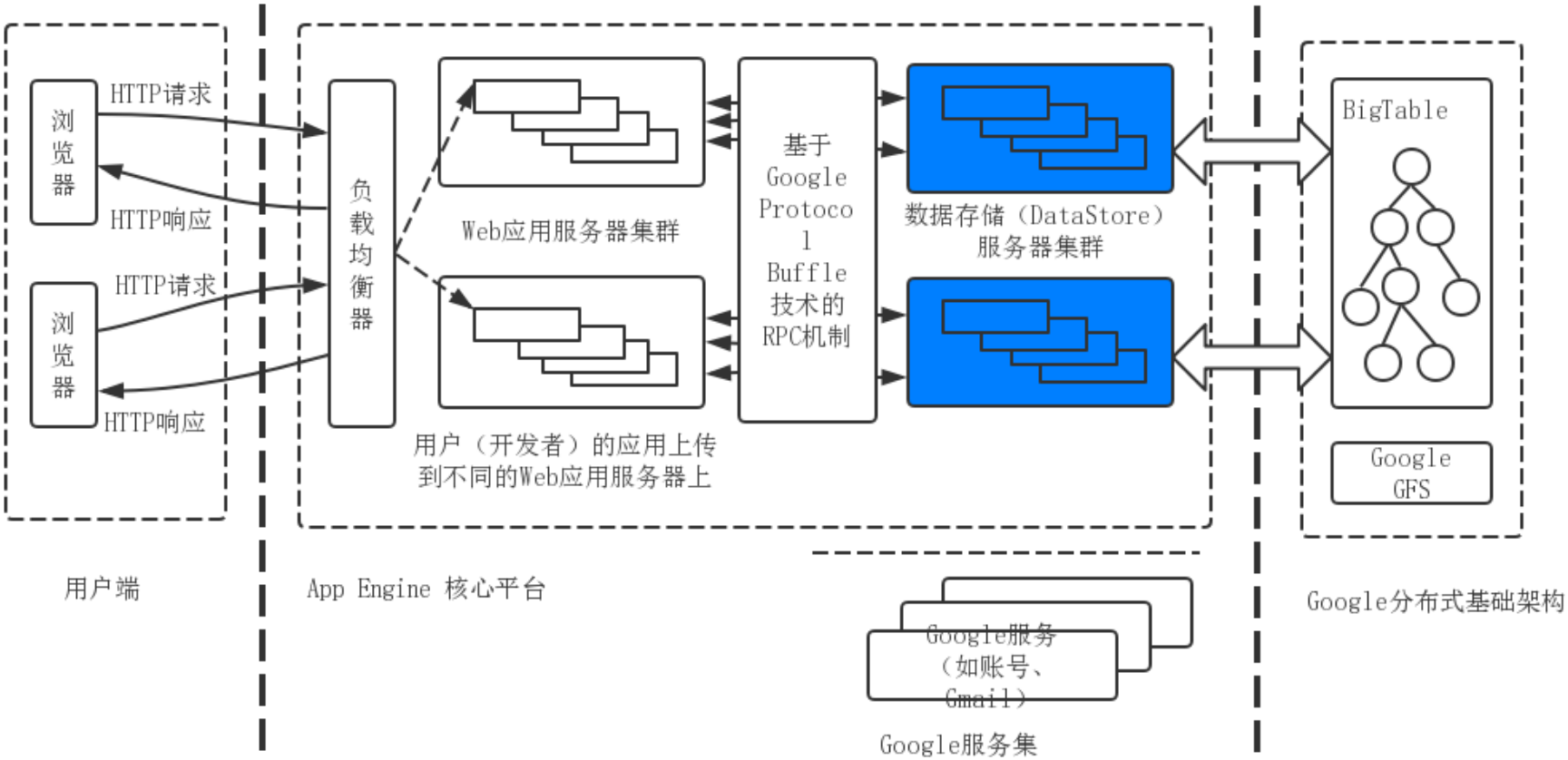
微服务架构


Silo logic
Silo database
Monolithic application
logic
logic
logic
Silo database
Internally componentized application
Microservice component
Microservice component
Microservice component
Microservices application

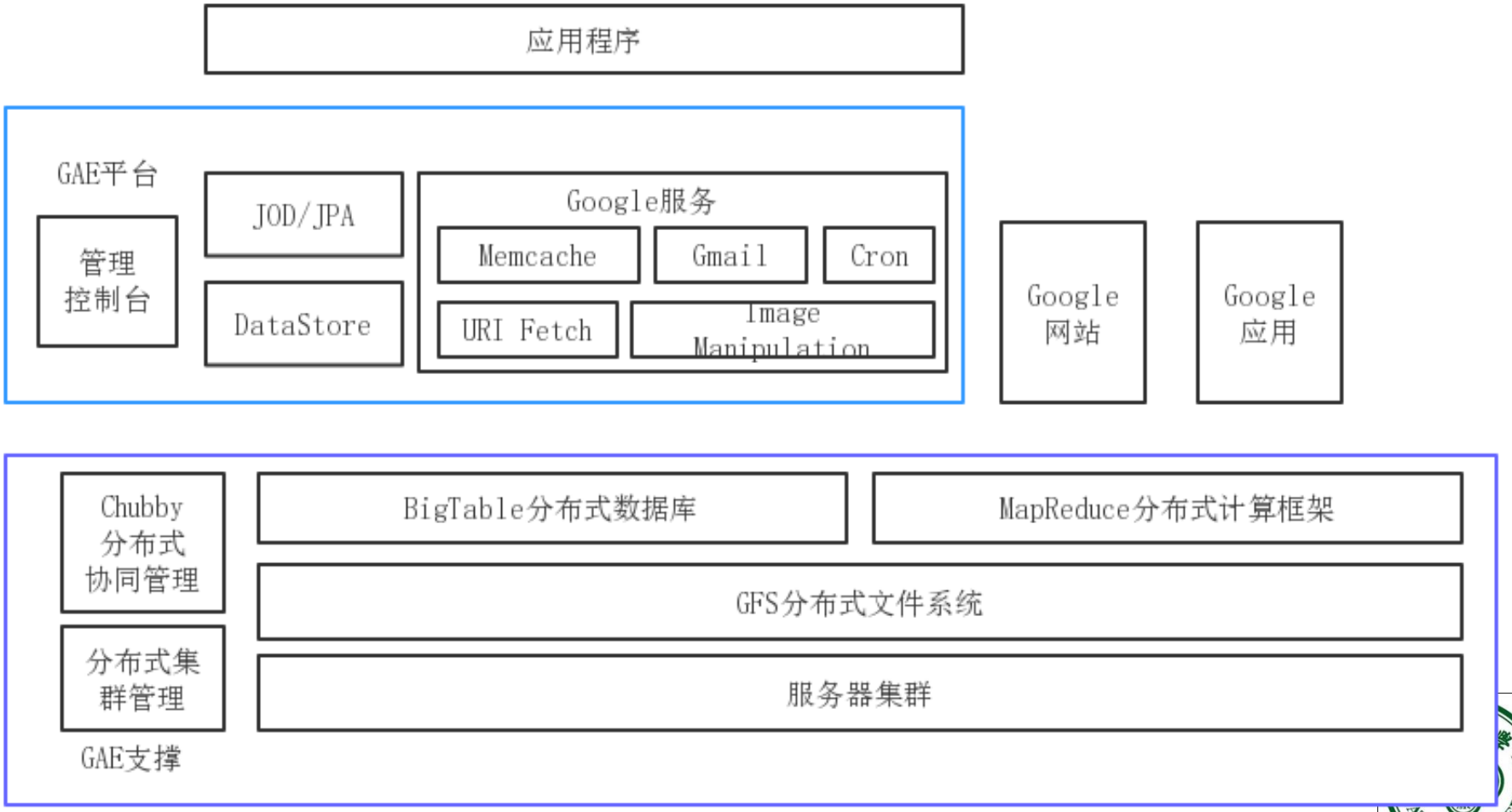
微服务、SOA 和 API


Exposure Gateway (external)
Engagement Applications
Traditional application
Traditional application
Microservice application
Microservice application
Enterprise Boundary
Integration Hub
Integration Hub
System of Record A
System of Record B
System of Record C
Business Partner
SaaS Application
Public API
Enterprise API
API Gateway

2.2 分布并行计算技术

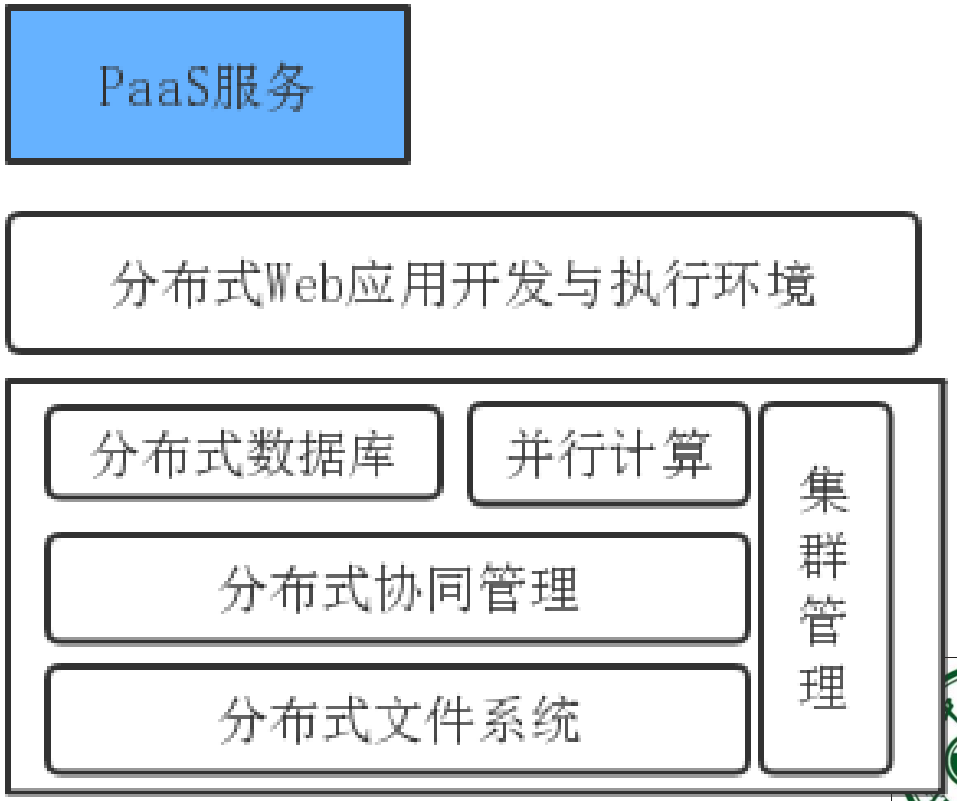

浏览器
HTTP请求
HTTP响应
浏览器
HTTP请求
HTTP响应
负载均衡器
Web应用服务器集群
用户（开发者）的应用上传到不同的Web应用服务器上
基于Google Protocol Buffle技术的RPC机制
数据存储（DataStore）服务器集群
BigTable
Google GFS
用户端
App Engine 核心平台
Google服务（如账号、Gmail）
Google服务集
Google分布式基础架构

Google 的大数据处理平台



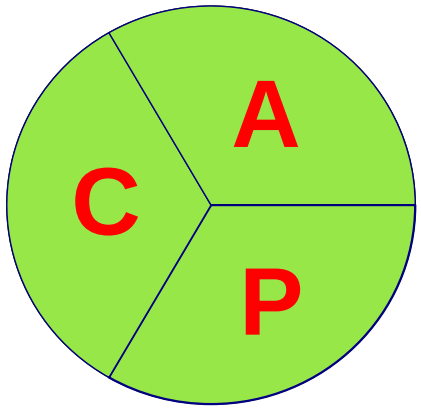
云数据处理关键技术

- 分布式数据管理
  - 分布式文件系统
  - 分布式数据库
- 云并行计算技术
- 分布式协同管理
- 集群管理



CAP Theorem

- Three properties in sharing data)
  - **Consistency:**
    - all copies have the same value
  - **Availability:**
    - reads and writes always succeed
  - **Partition-tolerance:**
    - consistency and/or availability hold even when network failures prevent communicating among some machines

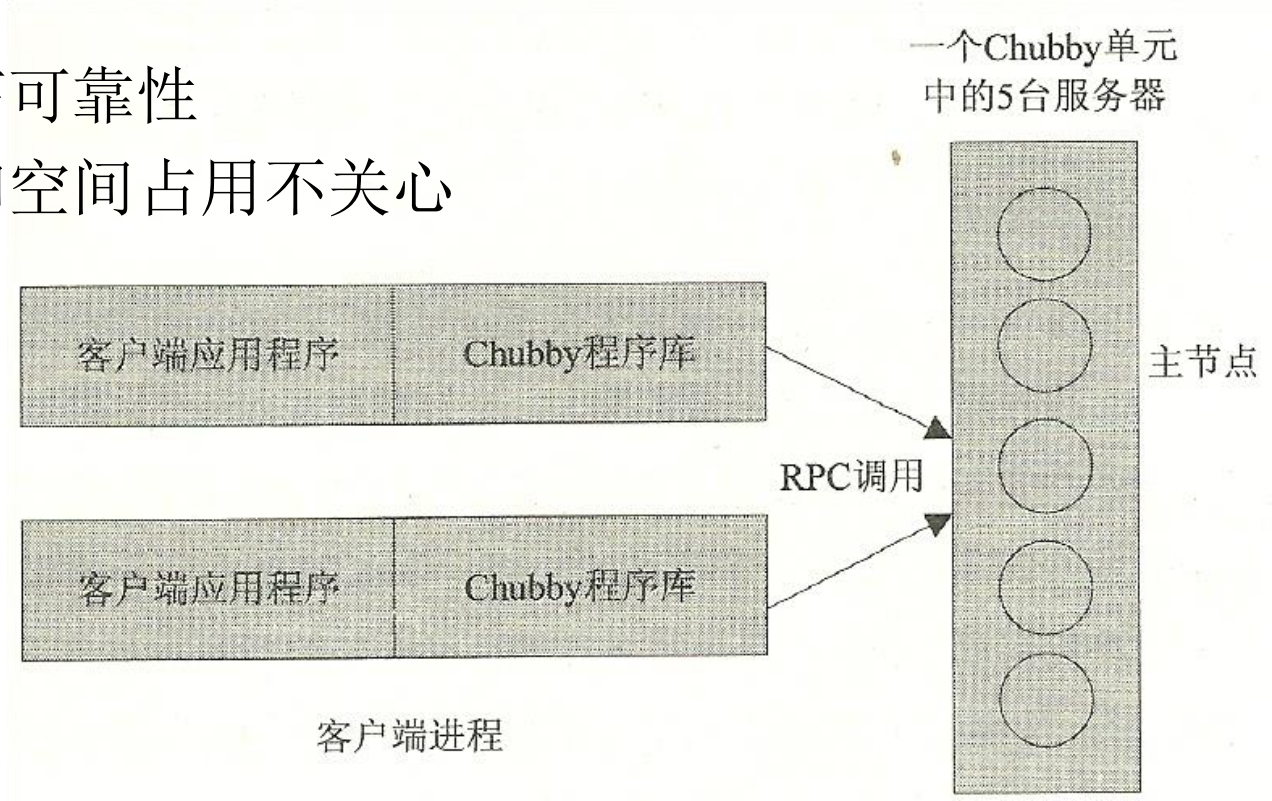


分布式协同管理

- 共享资源的并行操作→数据不一致
- 通过同步机制控制并发操作
- 常用的并发控制方法
  - 基于锁的并发控制
    - 两阶段锁协议
    - 多副本的锁机制：读-写全法、多数法、主副本法、单一管理法
  - 基于时间戳的并发控制
    - 基于全局唯一的时间戳
  - 乐观并发控制
  - 基于版本的并发控制

Google Chubby 并发控制

- 分布式所服务
- 通过文件操作实现锁操作
  - 文件代表锁
- 主要目标
  - 高可用性、高可靠性
  - 性能、吞吐和空间占用不关心

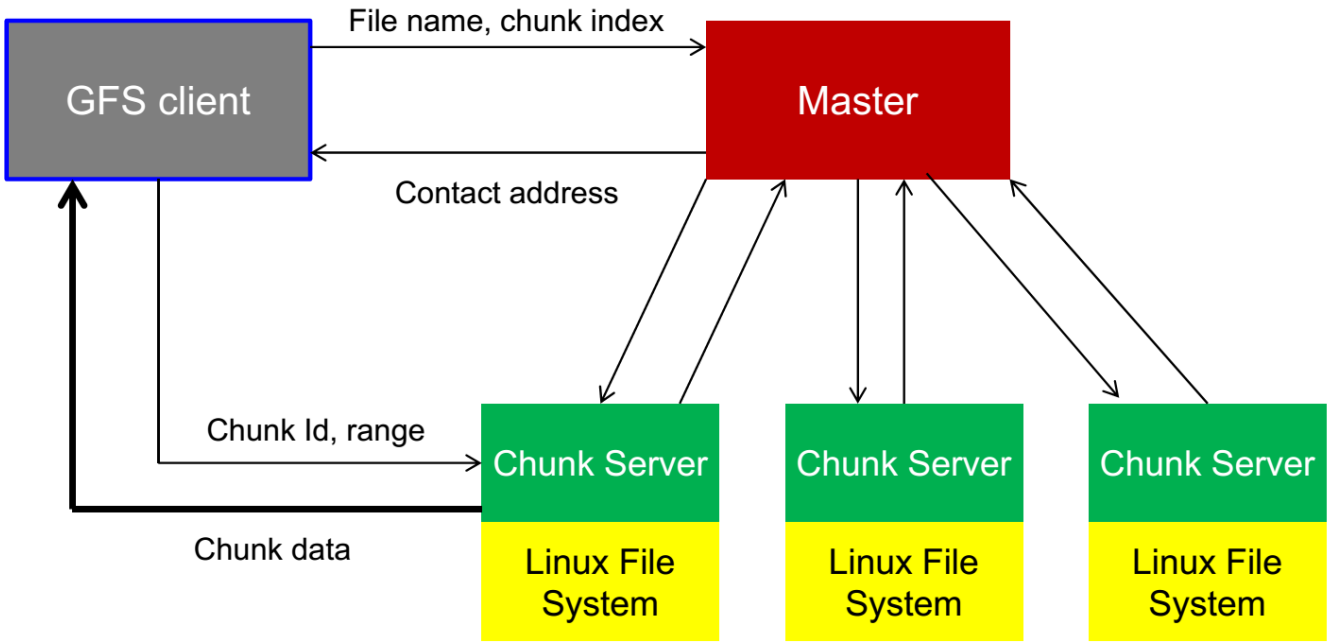


集群和平台管理

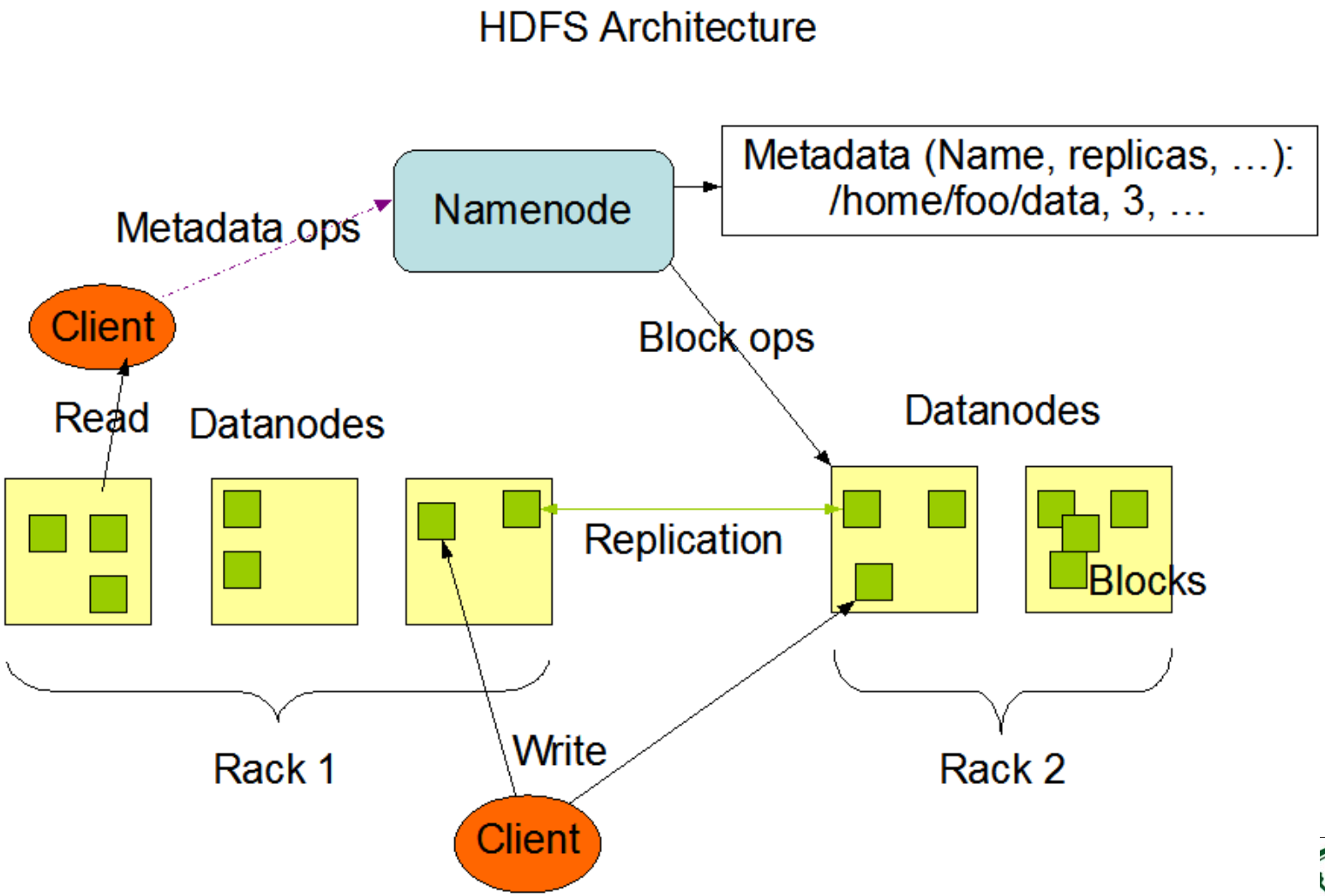
- 集群的自动化部署
  - 安装配置OS、DFS、分布式计算程序、作业管理软件、系统管理软件等。
- 集群作业调度
  - 基于资源管理器调度作业的运行节点和时间
  - Google Work Queue： 调度管理MapReduce任务
- 远程控制
  - 开关机控制
  - 远程控制台
- 应用管理与度量计费

分布式文件系统

- 基本特征
  - 透明性、并发访问、高可用性
- 基本需求
  - 数据冗余、异构性、一致性、高效性、安全性

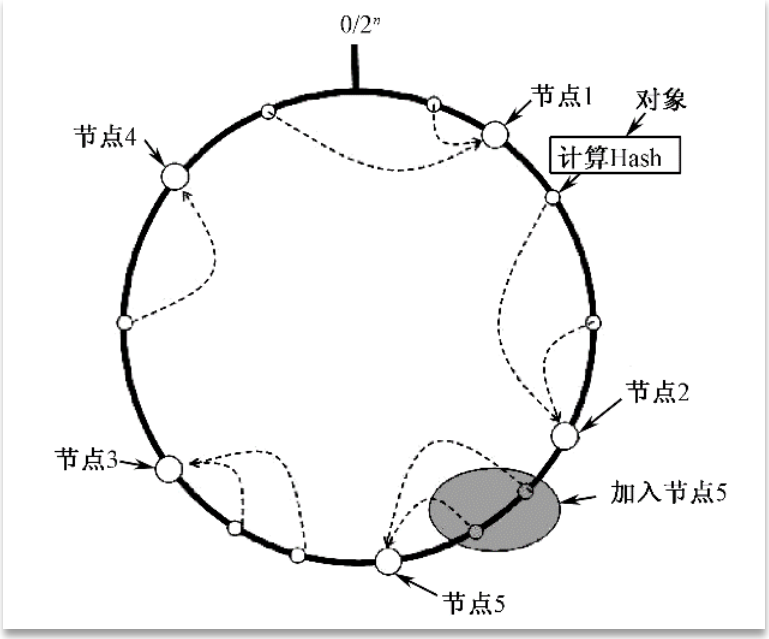


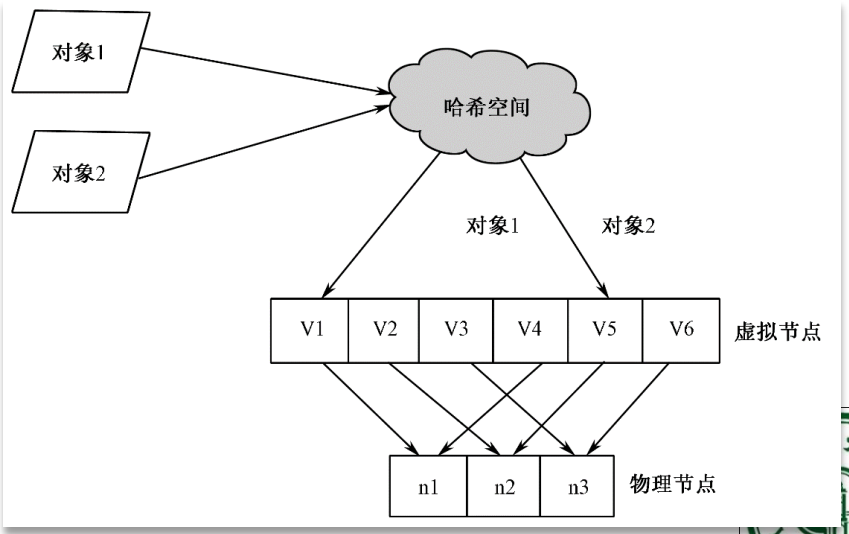
HDFS体系结构


HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

去中心化存储

- 文件、对象等数据类型
- 基于一致性哈希的元数据管理
  - 去中心化、数据ID哈希→Ring←节点ID（虚拟节点）





分布式数据库（NoSQL）

NoSQL数据库

- Key features (advantages):
  - non-relational, don’t require schema
  - data are replicated to multiple nodes and can be partitioned:
    - down nodes easily replaced
    - no single point of failure
  - horizontal scalable
  - cheap, easy to implement
  - massive write performance
  - fast key-value access

NoSQL数据库

- Disadvantages:
  - Don't fully support relational features
    - no join, group by, order by operations (except within partitions)
    - no referential integrity constraints across partitions
  - No declarative query language (e.g., SQL) → more programming
  - Relaxed ACID (see CAP theorem) → fewer guarantees
  - No easy integration with other applications that support SQL

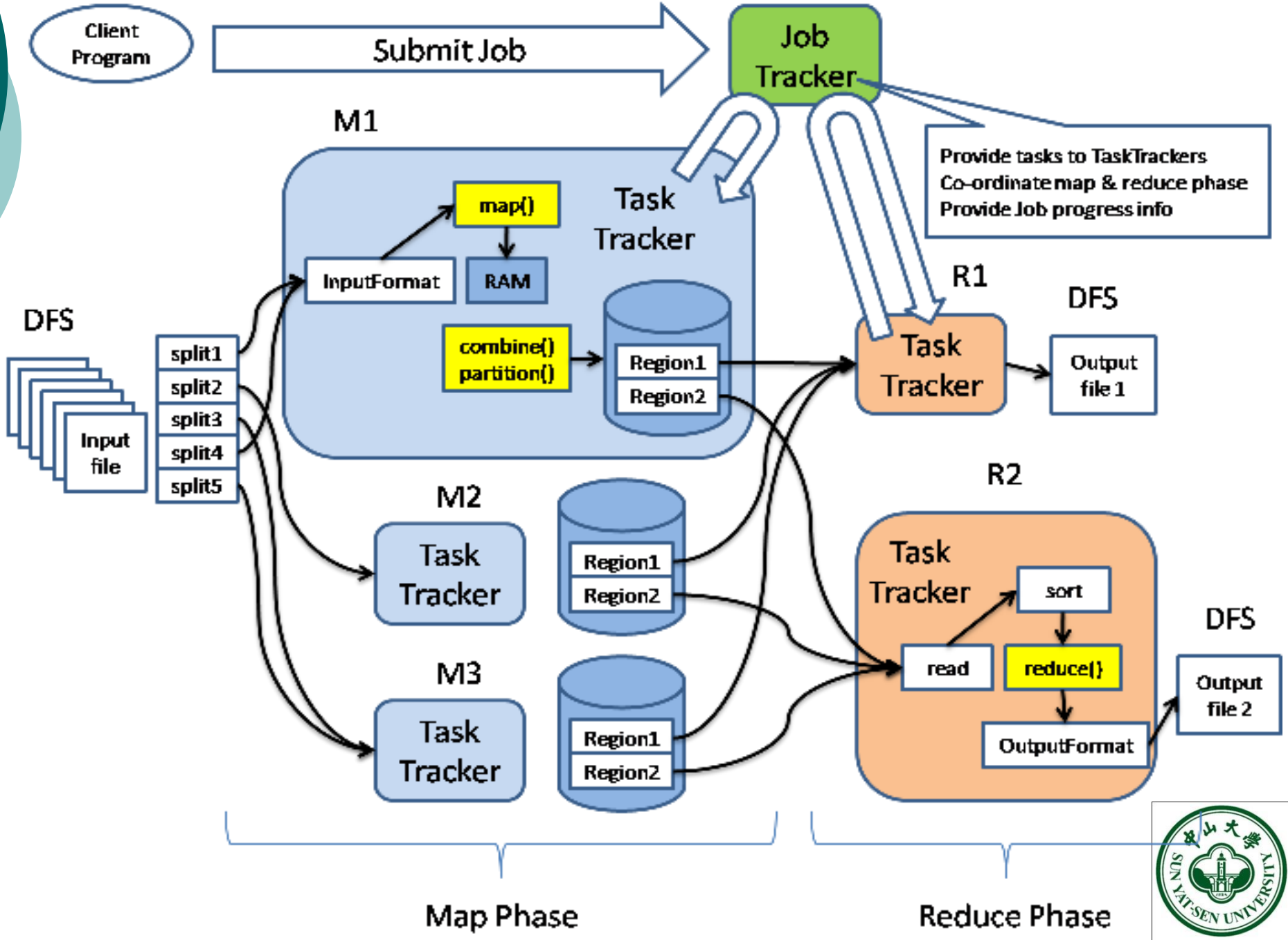
NoSQL categories

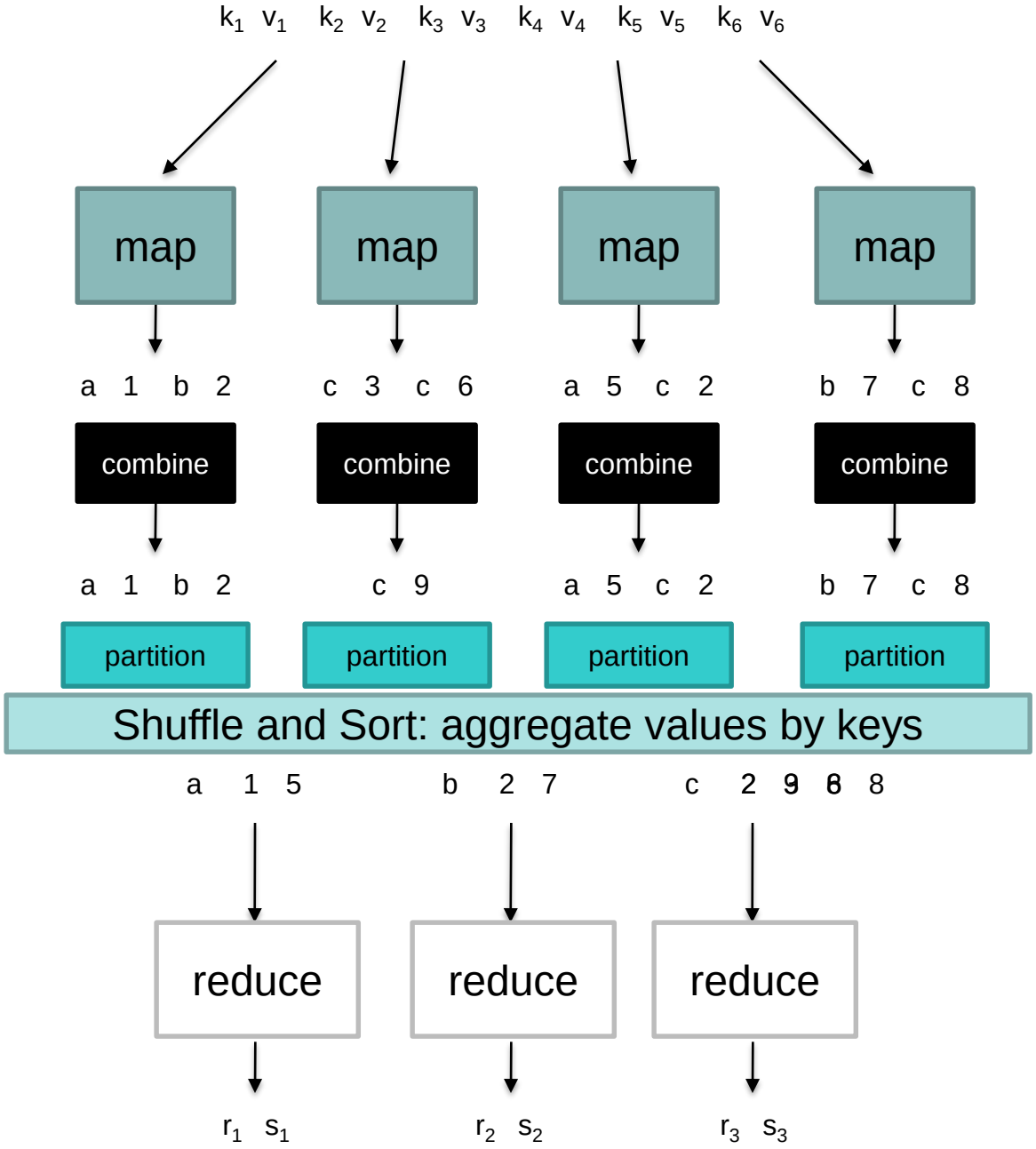
1. Key-value
   - Example: DynamoDB, Voldermort, Scalaris
2. Document-based
   - Example: MongoDB, CouchDB
3. Column-based
   - Example: BigTable, Cassandra, Hbased
4. Graph-based
   - Example: Neo4J, InfoGrid

- "No-schema" is a common characteristics of most NoSQL storage systems
- Provide "flexible" data types

MapReduce技术

- 一种新的编程框架/模型与运行环境
- 适于大数据分析处理
- 高度并行化、可扩展、透明、容错


Client Program
Submit Job
Job Tracker
Provide tasks to TaskTrackers
Co-ordinate map & reduce phase
Provide Job progress info
M1
Task Tracker
map()
InputFormat
RAM
combine()
partition()
Region1
Region2
DFS
Input file
split1
split2
split3
split4
split5
R1
Task Tracker
DFS
Output file 1
M2
Task Tracker
Region1
Region2
R2
Task Tracker
sort
read
reduce()
OutputFormat
DFS
Output file 2
M3
Task Tracker
Region1
Region2
Map Phase
Reduce Phase


k1 v1 k2 v2 k3 v3 k4 v4 k5 v5 k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combine
combine
combine
combine
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

Example Word Count: Map

```
public static class MapClass extends MapReduceBase
implements Mapper {
  private final static IntWritable one= new IntWritable(1);
  private Text word = new Text();

  public void map(WritableComparable key, Writable value,
  OutputCollector output, Reporter reporter)
  throws IOException {
    String line = ((Text)value).toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    }
  }
}
```

Example Word Count: Reduce

```
public static class Reduce extends MapReduceBase
implements Reducer {
  public void reduce(WritableComparable key, Iterator
  values, OutputCollector output, Reporter reporter)
  throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += ((IntWritable) values.next()).get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

Example Word Count: Main

```
public static void main(String[] args) throws IOException
 {
   //checking goes here
   JobConf conf = new JobConf();

   conf.setOutputKeyClass(Text.class);
   conf.setOutputValueClass(IntWritable.class);

   conf.setMapperClass(MapClass.class);
   conf.setCombinerClass(Reduce.class);
   conf.setReducerClass(Reduce.class);

   conf.setInputPath(new Path(args[0]));
   conf.setOutputPath(new Path(args[1]));

   JobClient.runJob(conf);
}
```

Example

- Page 1: the weather is good
- Page 2: today is good
- Page 3: good weather is good.

Map output

- Worker 1:
  - (the 1), (weather 1), (is 1), (good 1).
- Worker 2:
  - (today 1), (is 1), (good 1).
- Worker 3:
  - (good 1), (weather 1), (is 1), (good 1).

Reduce Input

- Worker 1:
  - (the 1)
- Worker 2:
  - (is 1), (is 1), (is 1)
- Worker 3:
  - (weather 1), (weather 1)
- Worker 4:
  - (today 1)
- Worker 5:
  - (good 1), (good 1), (good 1), (good 1)

Reduce Output

- Worker 1:
  - (the 1)
- Worker 2:
  - (is 3)
- Worker 3:
  - (weather 2)
- Worker 4:
  - (today 1)
- Worker 5:
  - (good 4)

三、云机制及云架构

- 监控代理
- 虚拟机监控器及虚拟机迁移
- 自动伸缩监听器及动态伸缩架构
- 负载均衡监控器及负载分布架构
- 故障转移系统与云服务容错
- SLA监控器及管理系统
- 计费监控器及管理系统
- 审计监控器
- 远程管理系统

监控机制

- 一种轻量级的自治软件程序
- 用于收集和处理IT资源的使用数据
- 数据发送到日志数据库，以便后续处理和报告。
- 三种种常见的实现形式：（基于代理）
  - 监控代理（monitoring agent）
  - 资源代理（resource agent）
  - 轮询代理（polling agent）

监控代理

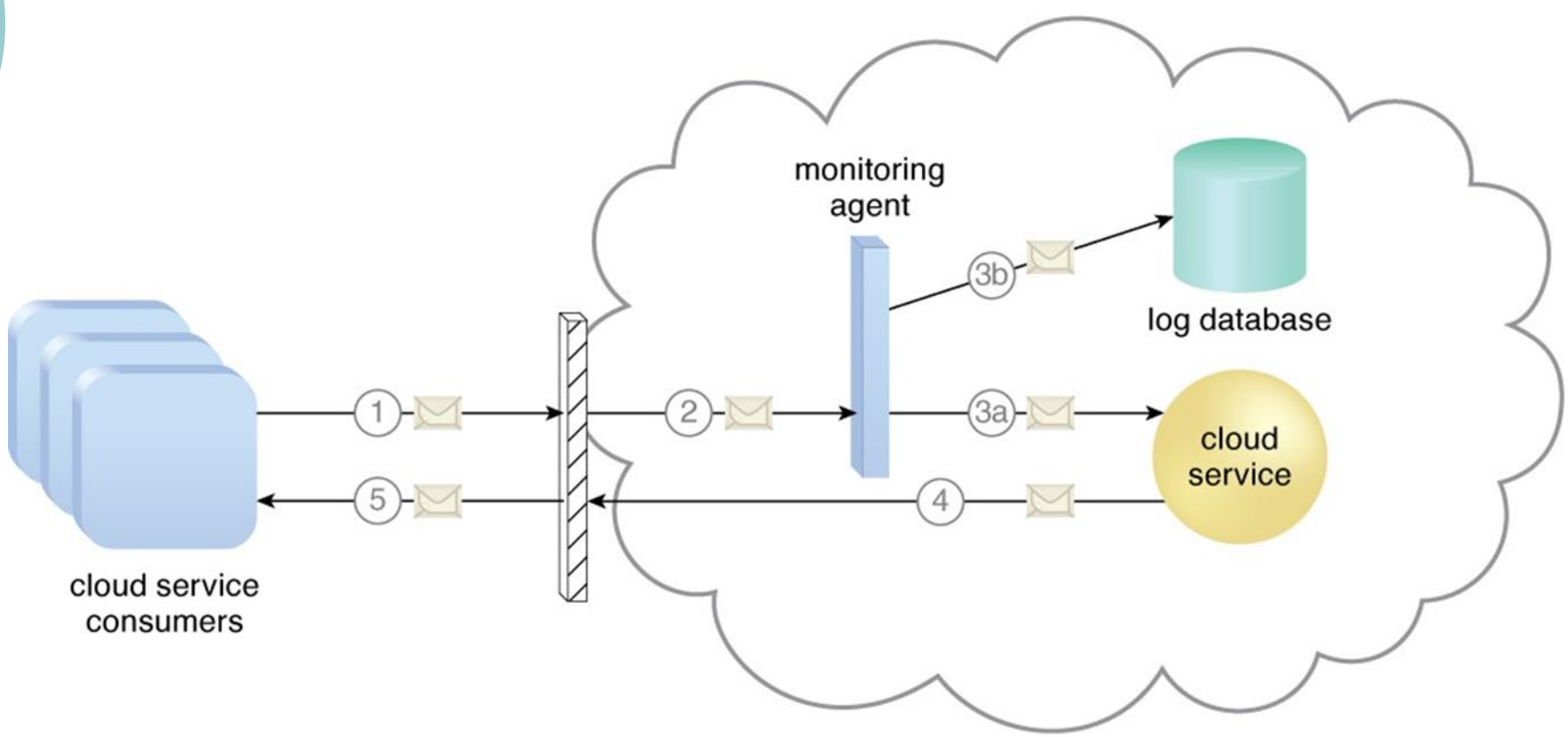


Figure7.12 — 云服务用户向云服务发送请求消息（1）监控代理拦截此消息，收集相关使用数据（2），然后将其继续发往云服务（3a）。监控代理将收集到的使用数据存入日志数据库（3b）。云服务产生应答消息（4），并将其发送回云服务用户，此时监控代理不会进行拦截（5）。

资源代理

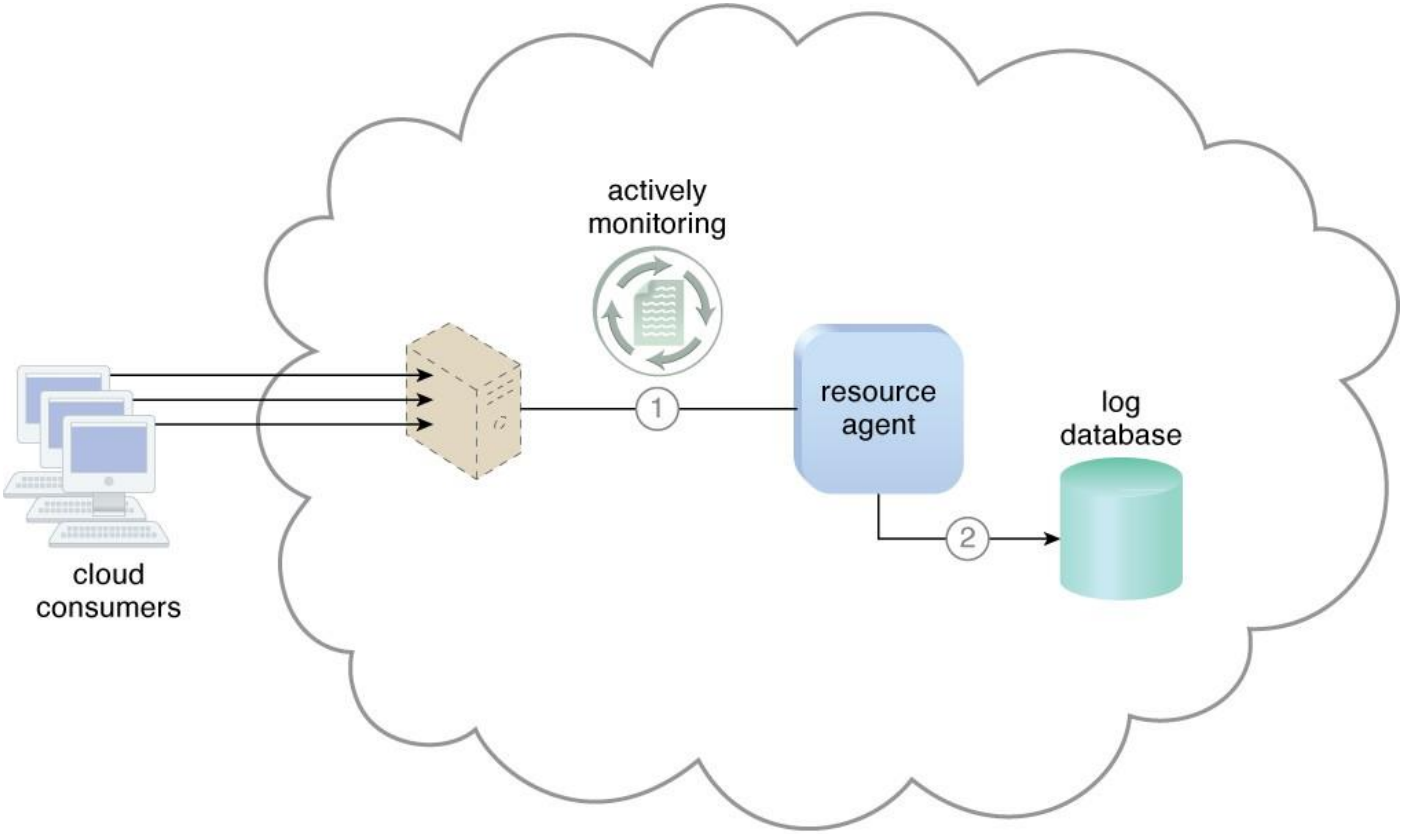


Figure7.13 — 资源代理主动监控虚拟服务器，并检测到使用的增加（1）。资源代理从底层资源管理程序收到通知，虚拟服务器正在进行扩展，按照其监控指标，资源代理将收集的使用数据存入日志数据库（2）。

轮询代理

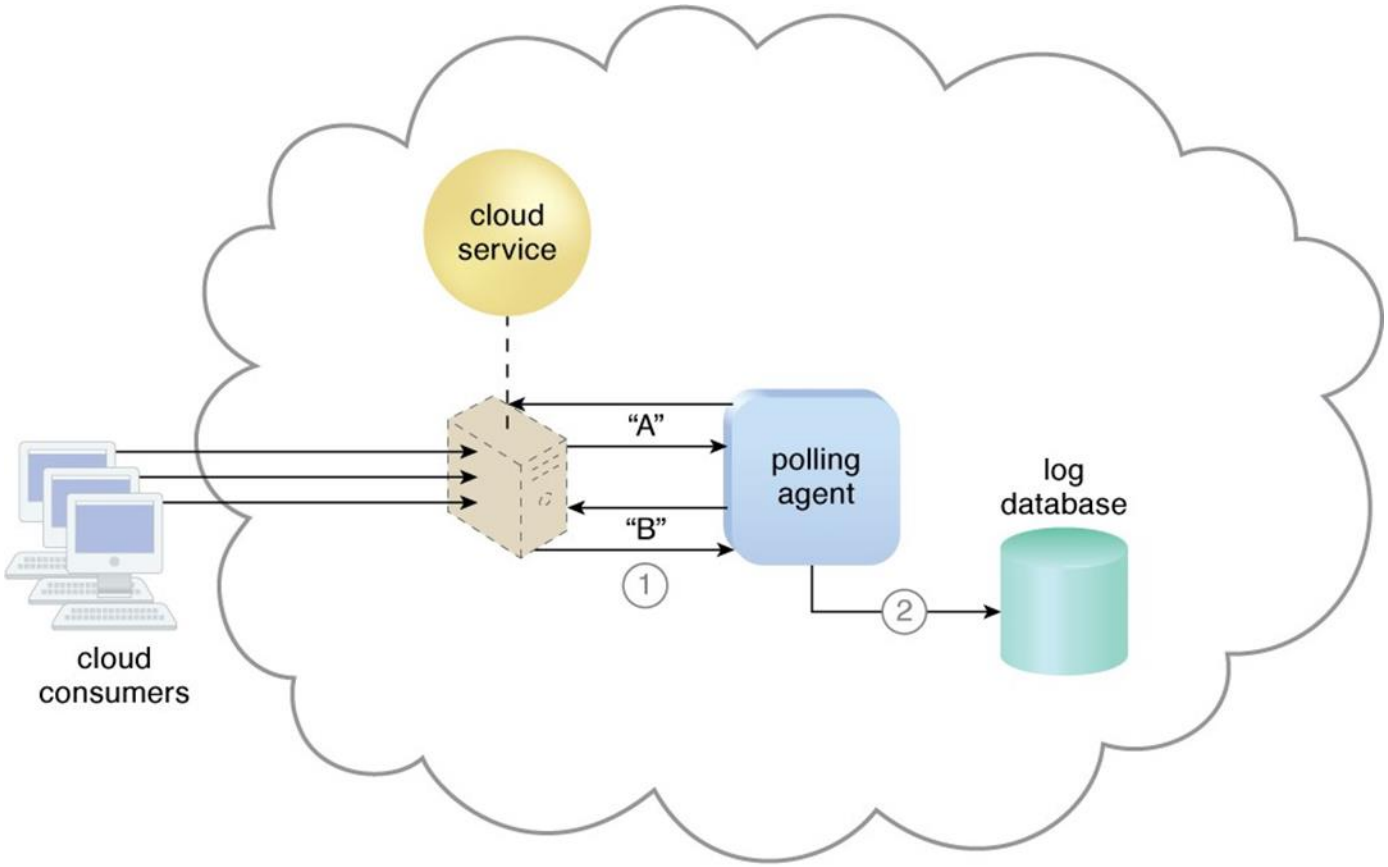
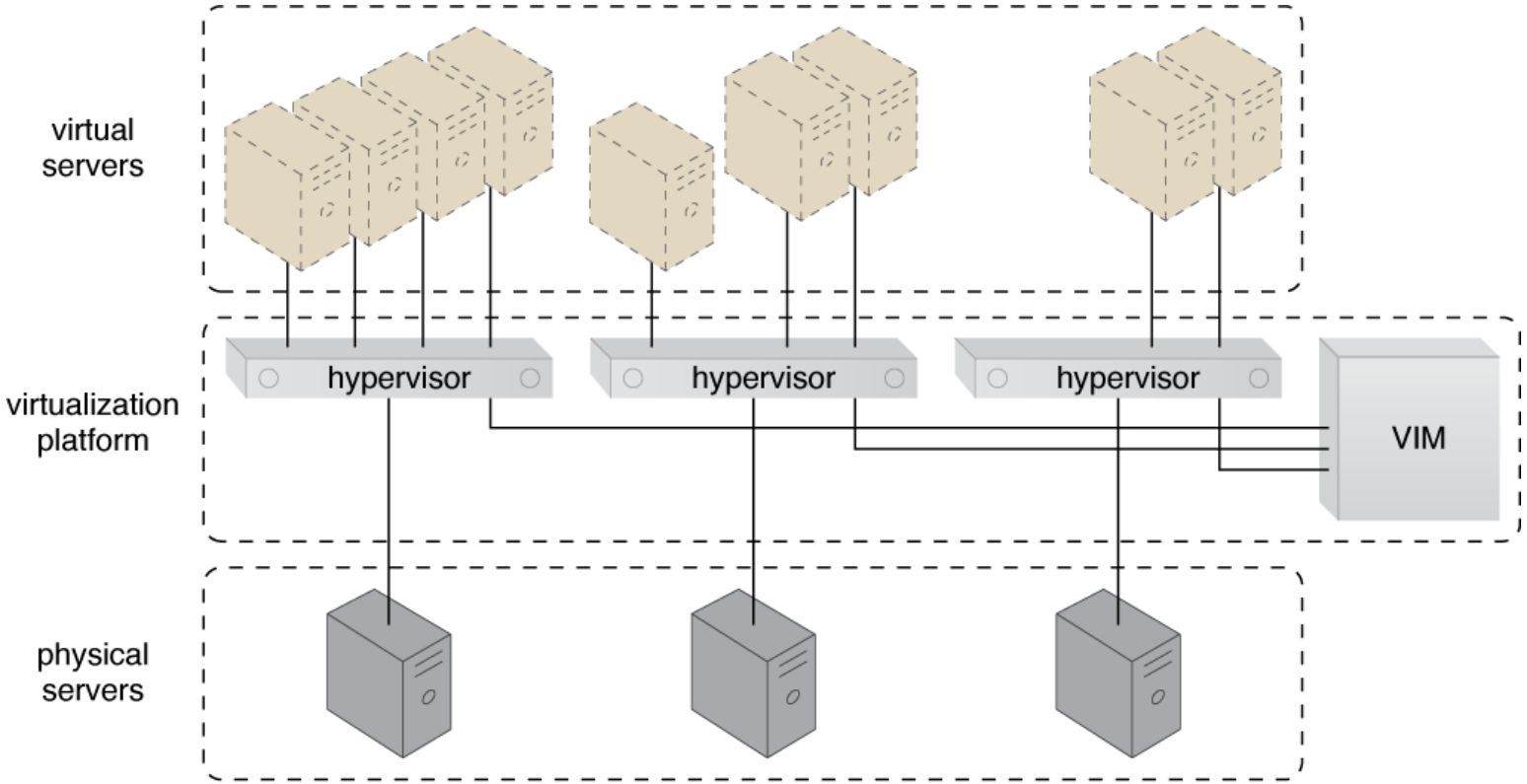


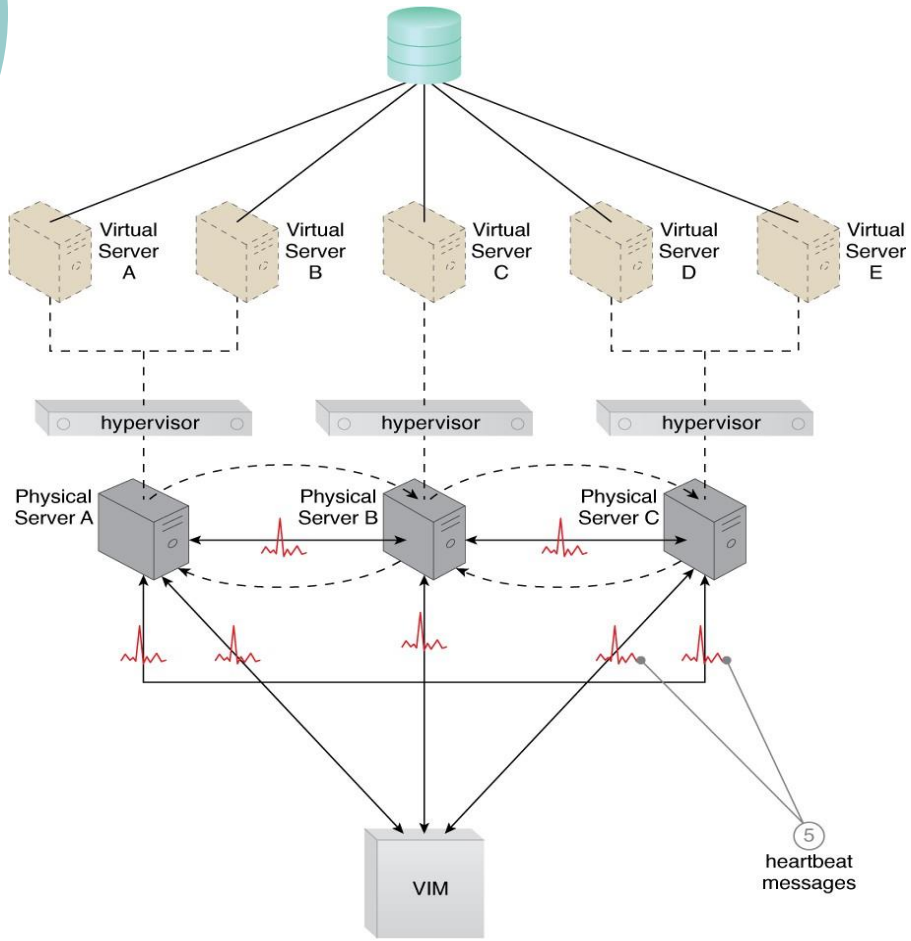
Figure7.14 — 轮询代理监控虚拟服务器上的云服务状态，它周期性地发送轮询消息，并在数个轮询周期后接收到使用状态为“A”的轮询响应消息。当代理接收到使用状态为“B”时（1），轮询代理就将新的使用状态记录到日志数据库中（2）。

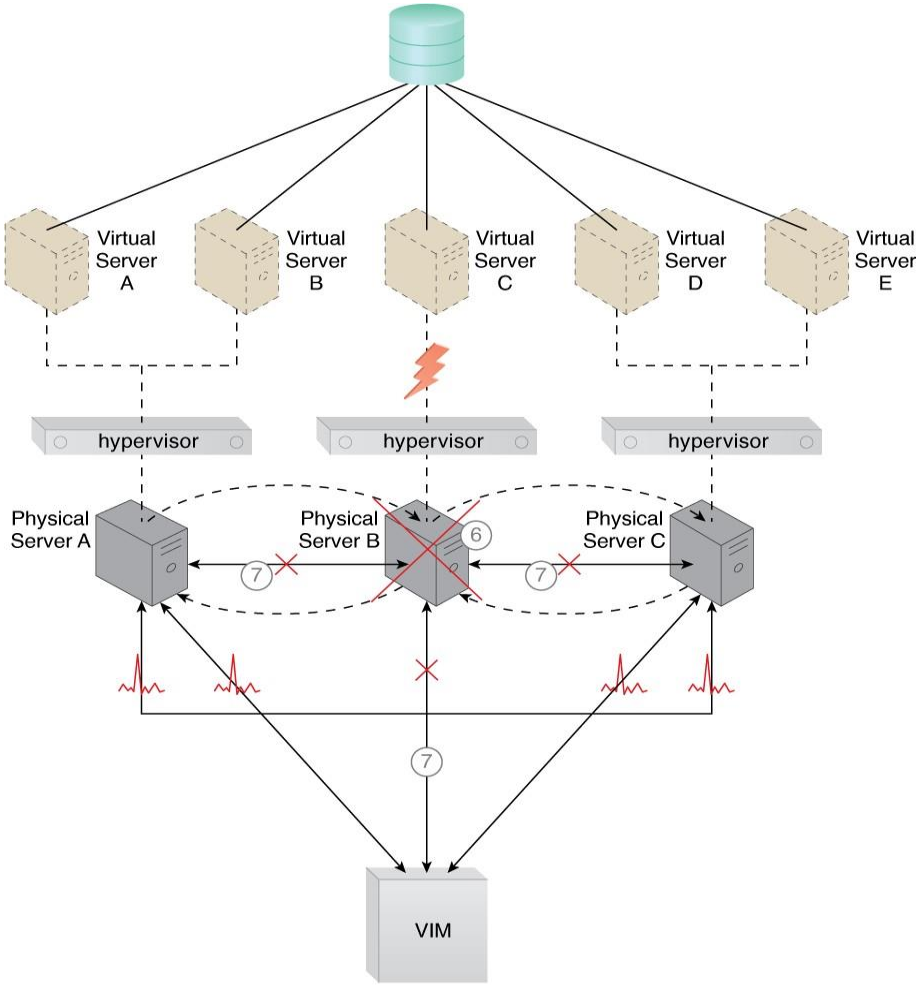
虚拟机监控器

- 虚拟基础设施管理器（VIM）



Hypervisor/VS 在线迁移

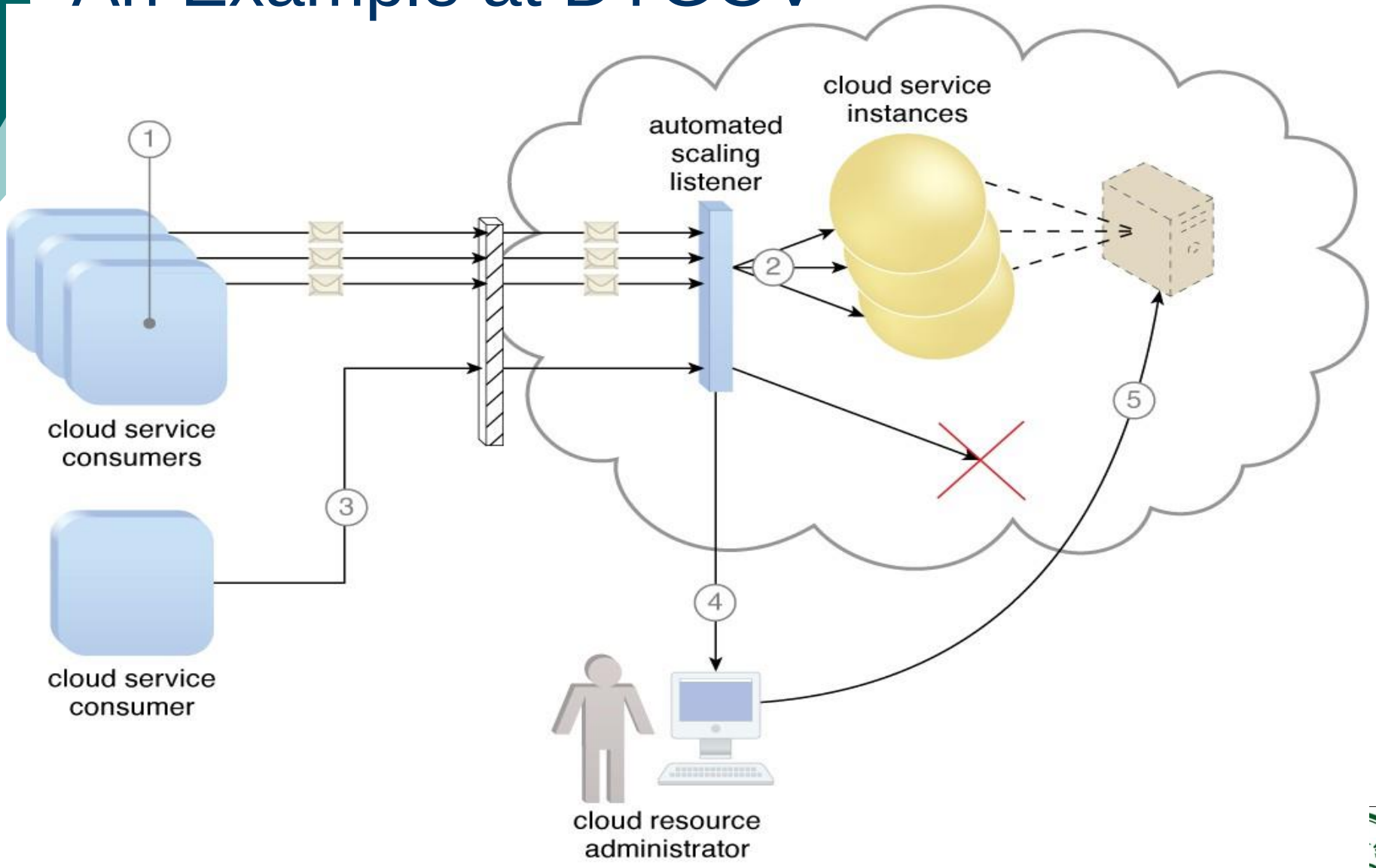




自动伸缩监听器

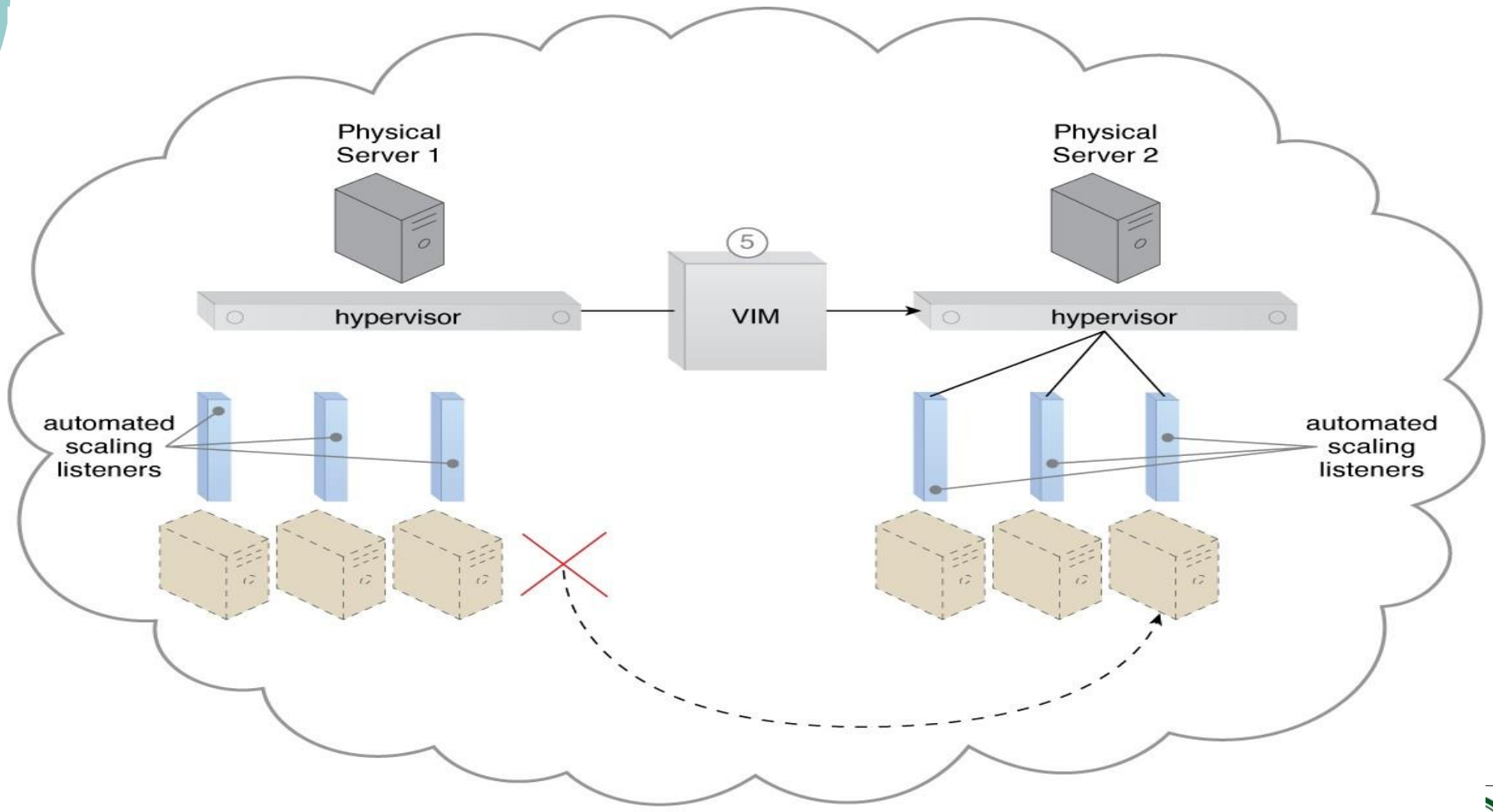
- 自动伸缩监听器(automated scaling listener mechanism)
  - 一个服务代理，监控和追踪云服务用户和云服务之间的通讯，用以动态自动伸缩。
- 通常部署在靠近防火墙的位置，来自动追踪负载状态信息。
- 对应负载波动的条件，可以提供不同类型的响应：
  - 自动伸缩IT资源(auto-scaling)：根据事先定义的参数
  - 自动通知云用户(auto-notification)：负载过高或过低时

An Example at DTGOV



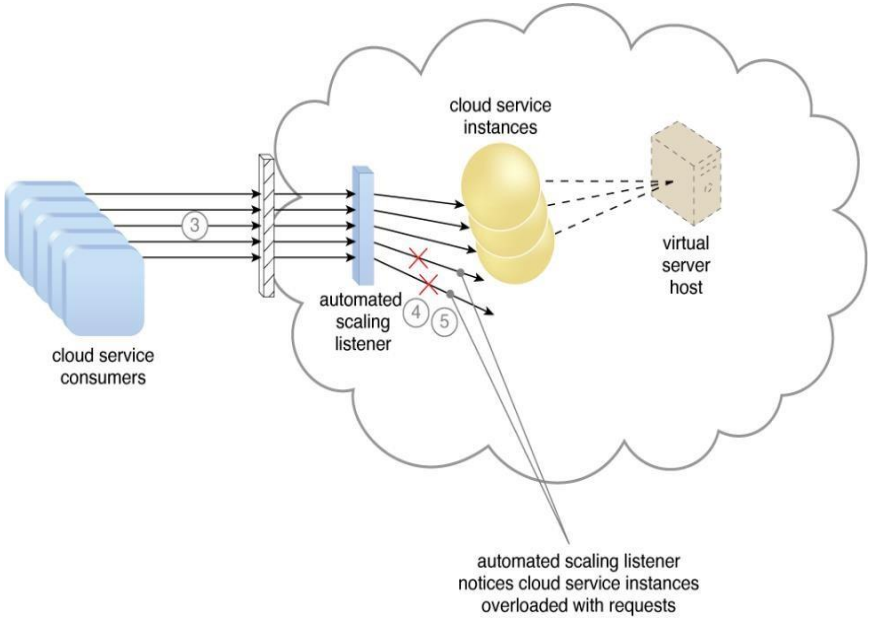
The fourth cloud service consumer is rejected due to workload constraints; Administrator can change the constraint upon notification.

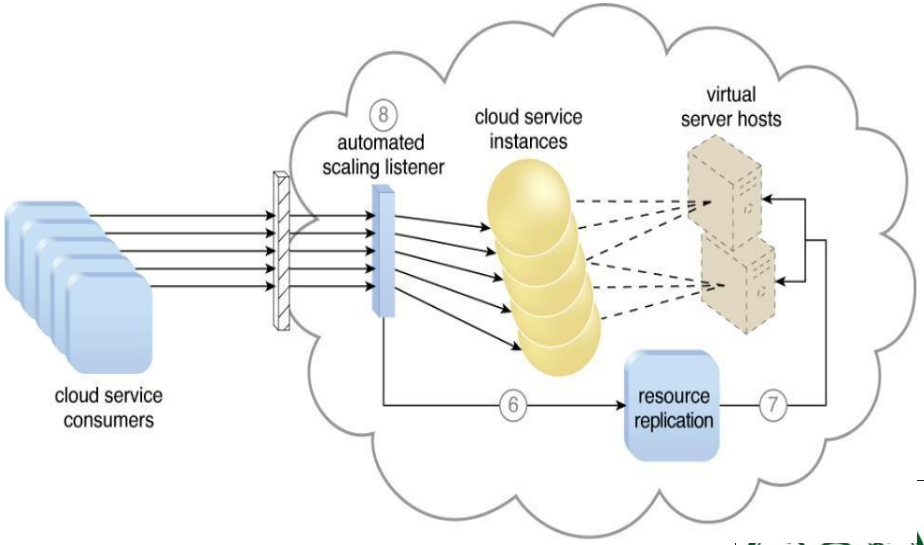
The VIM determines that scaling up on Physical Server 1 is not possible, and proceeds to live migrate it to Physical Server 2.



动态可扩展架构

- 动态可扩展架构
  - 基于预先定义的扩展条件从资源池中动态分配IT资源
  - 云资源弹性管理的核心机制。
  - 基于自动扩展监听器





弹性资源容量架构

- Elastic Resource Capacity Architecture
  - 主要与虚拟服务器的动态供给相关
  - 根据负载变化分配和回收CPU与RAM资源
- 扩展逻辑在智能自动化引擎（不在监听器）
  - 自动扩展监听器发信号给智能引擎（一种脚本）
  - 智能引擎运行工作流逻辑，触发扩展
- 扩展属于垂直扩展
- 依赖一些额外机制：
  - 云使用监控器（Cloud Usage Monitor）
  - 按使用付费监控器（Pay-Per-Use Monitor）
  - 资源复制（Resource Replication）

弹性资源容量架构工作流程（1）

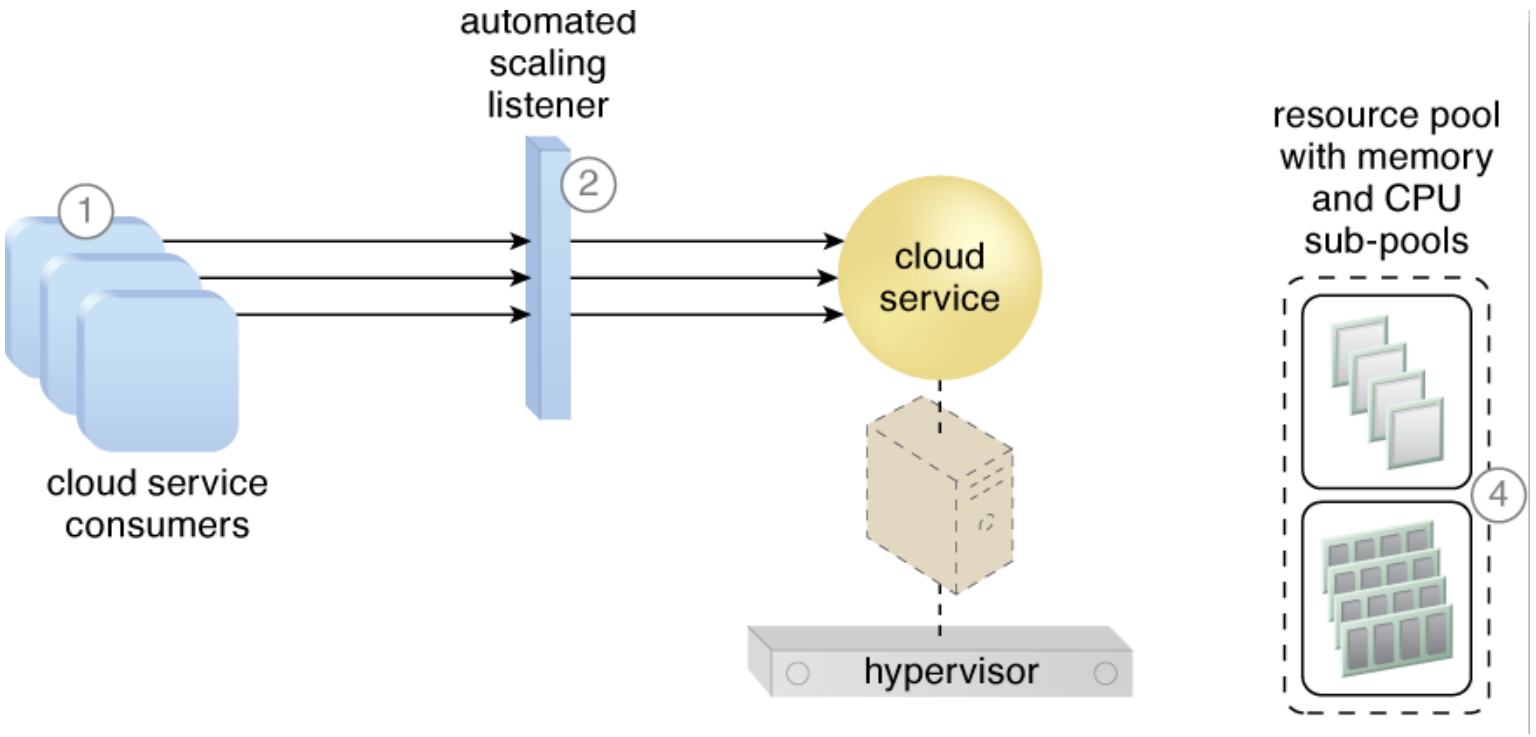
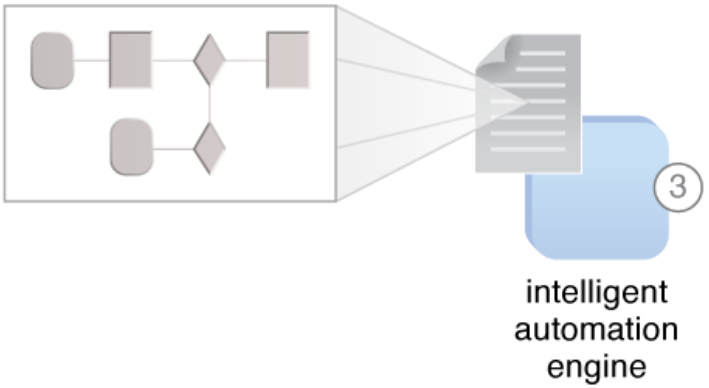


Figure11.8 云服务用户主动向云服务发送请求（1），自动扩展监听器对此进行监控（2）。智能自动化引擎脚本与工作流逻辑一起部署（3），能够通过非配请求通知资源池（4）。



弹性资源容量架构工作流程（2）

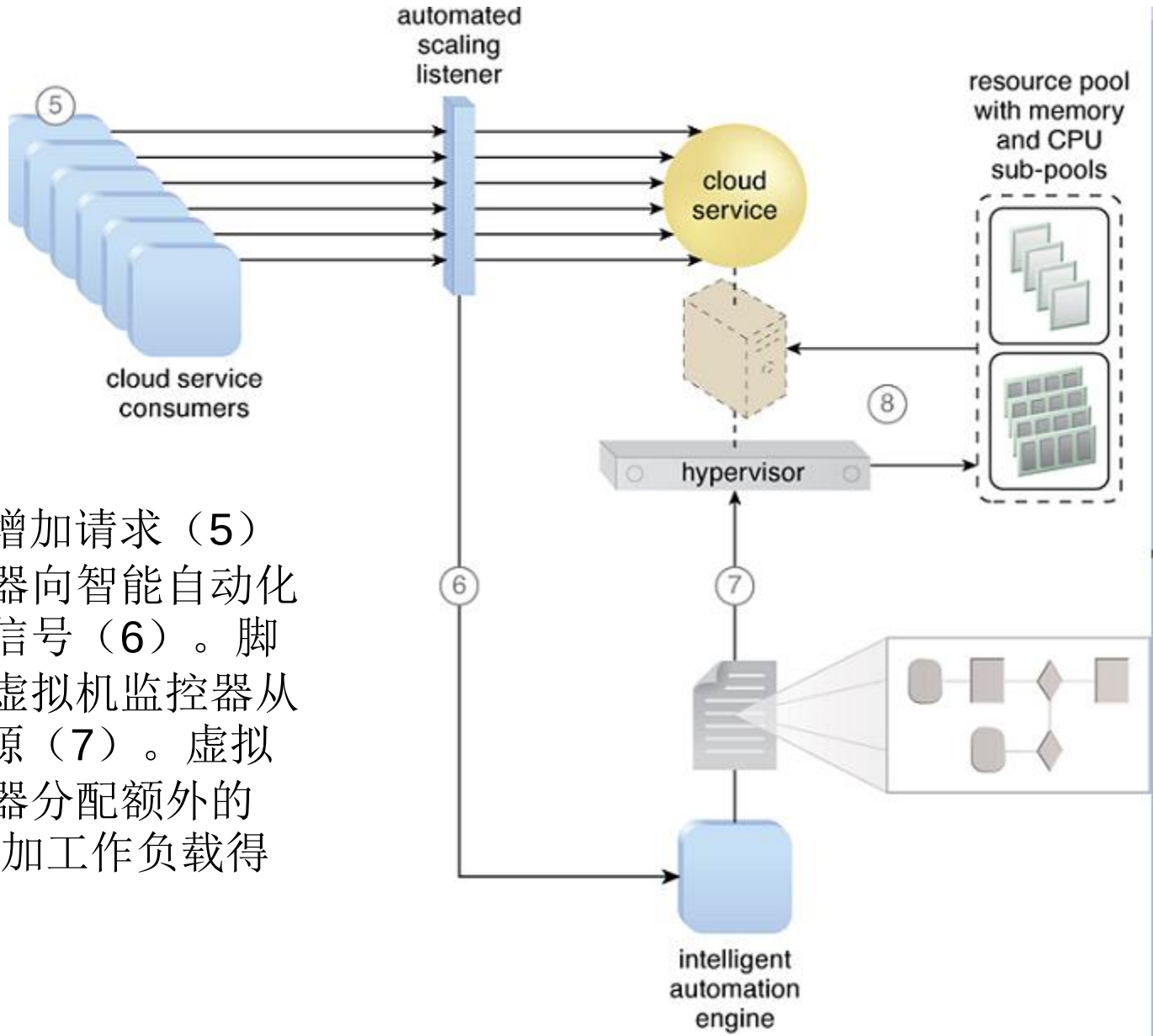
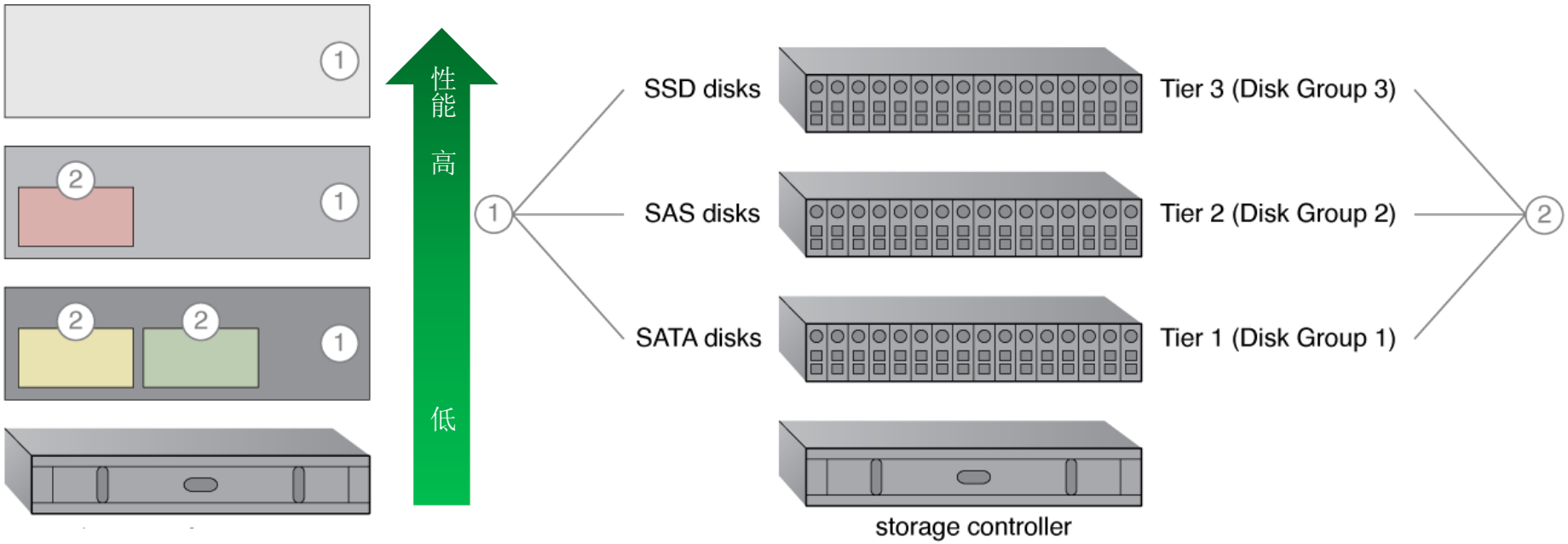
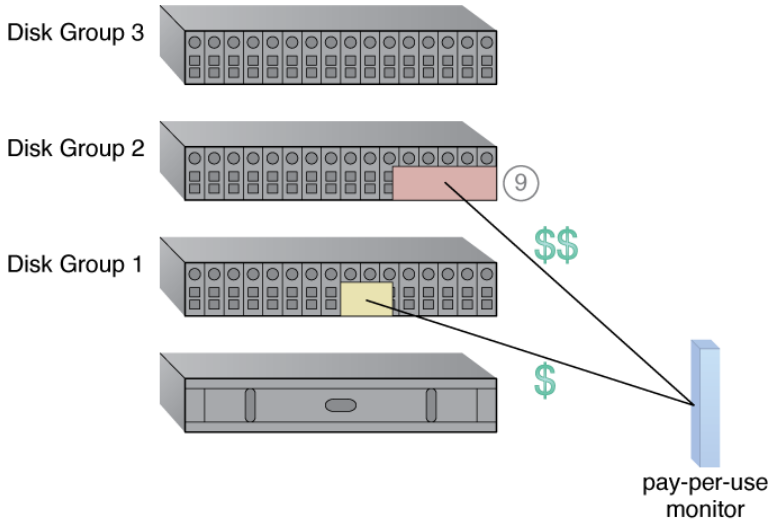


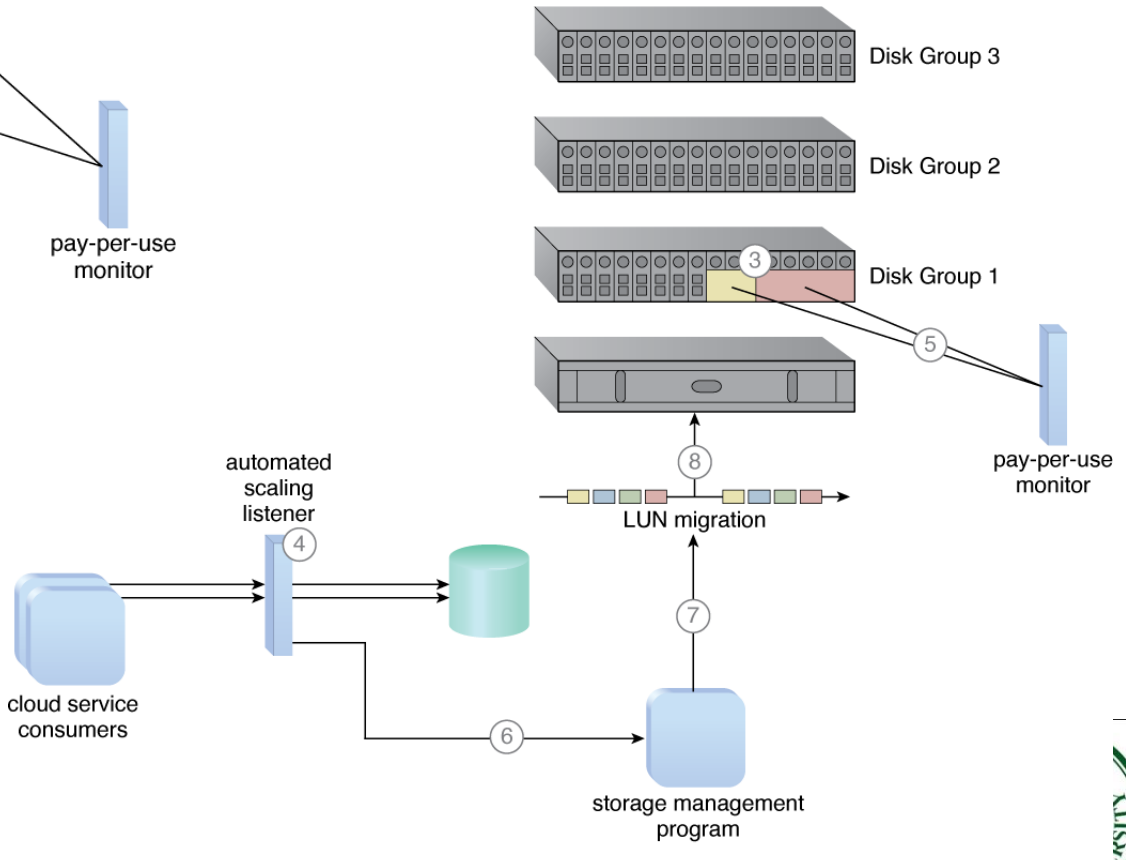
Figure11.9 云服务用增加请求（5），使得自动扩展监听器向智能自动化引擎发送执行脚本的信号（6）。脚本运行工作流逻辑，虚拟机监控器从资源池分配更多IT资源（7）。虚拟机监控器给虚拟服务器分配额外的CPU和RAM，使得增加工作负载得以处理（8）。

存储设备内部垂直扩展



存储设备内部垂直扩展



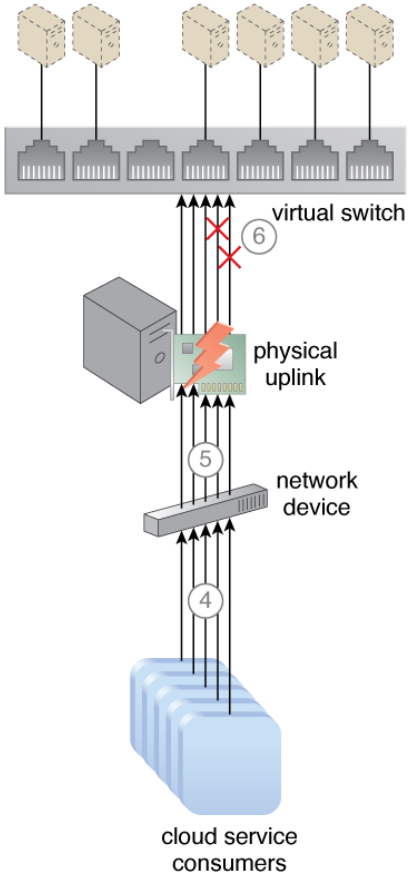


弹性网络容量架构

- Elastic Network Capacity Architecture
  - 用于给网络动态分配额外带宽，以避出现网络瓶颈
- 基本机制
  - 共享的网络端口资源池（物理/虚拟交换机）
  - 不同的云用户使用不同的网络端口实现隔离
  - 自动扩展监听器和智能自动化引擎脚本用于检测流量
  - 到达带宽阈值时动态分配额外带宽和网络端口

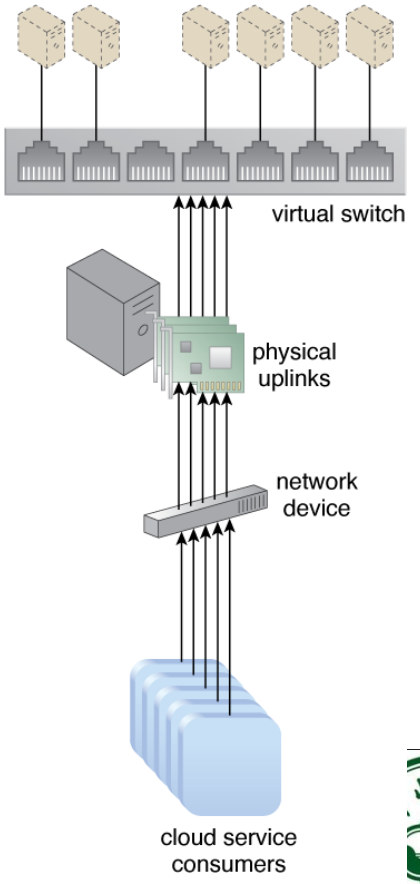
负载均衡的虚拟交换架构

- 虚拟交换机连接虚拟机和物理网络
- 可能会出现带宽瓶颈



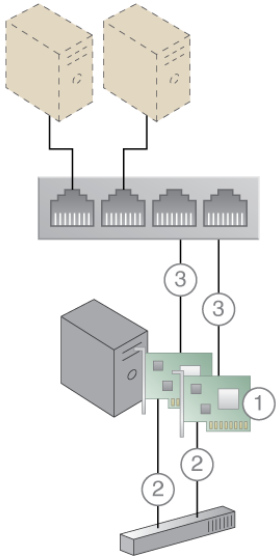
负载均衡的虚拟交换机架构

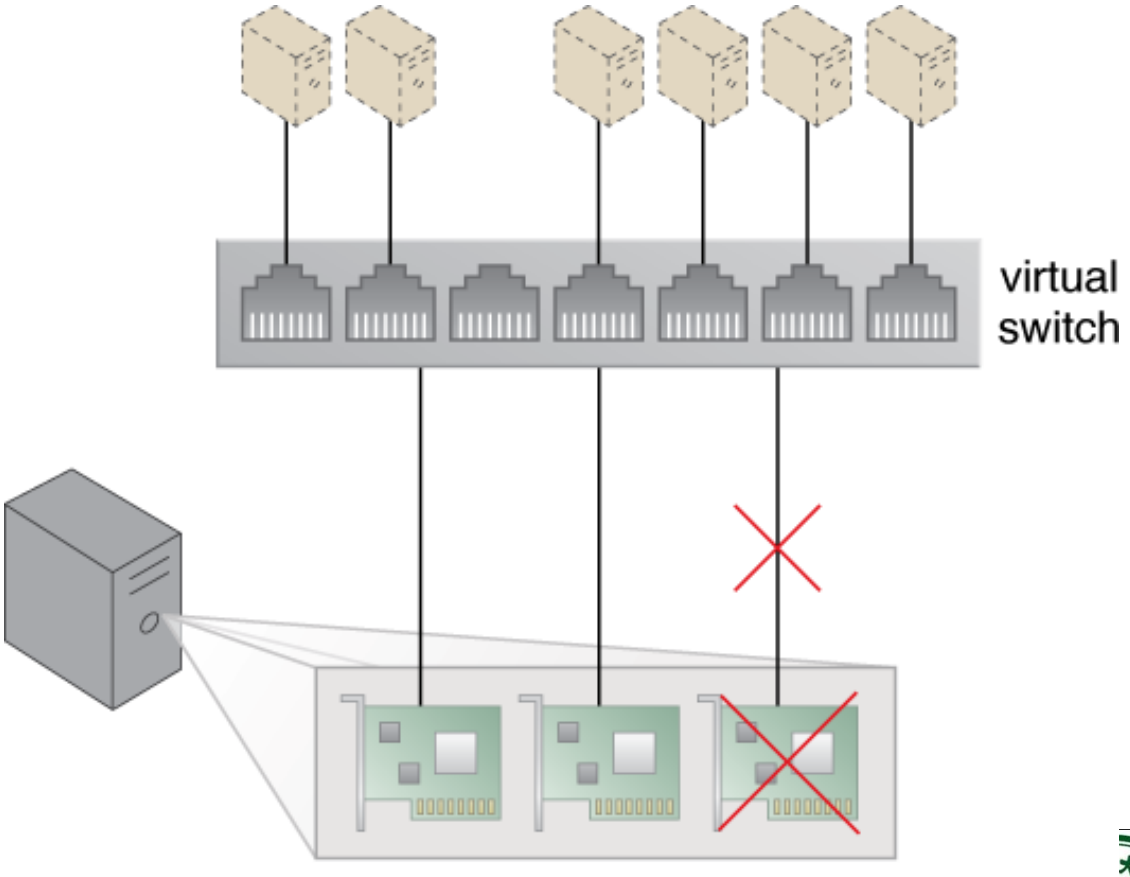
- Load balanced virtual switches architecture
  - 提供多条上行链路来平衡多条上行链路或冗余路径之间的网络流量负载
  - 有助于避免出现传输延迟和数据丢失。
- 多物理上行链路机制
  - 虚拟交换机需要进行配置
  - 支持多物理上行链路
    - 通常将其配置一个组。



虚拟机冗余物理连接

- 可以应对物理链路失效
  - 网卡
  - 网线





负载均衡器

- 负载均衡器(load balancer)机制是一个运行时代理
- 主要用于把负载在两个或更多的IT 资源上做负载
- 载体：
  - 多层网络交换机(multi-layer network switch)
  - 专门的硬件设备(dedicated hardware appliance)
  - 专门的基于软件的系统(dedicated software-based system (in server OS))
  - 服务代理(service agent (controlled by cloud management software))

负载分布架构

- Workload Distribution Architecture
- 多个构成部分

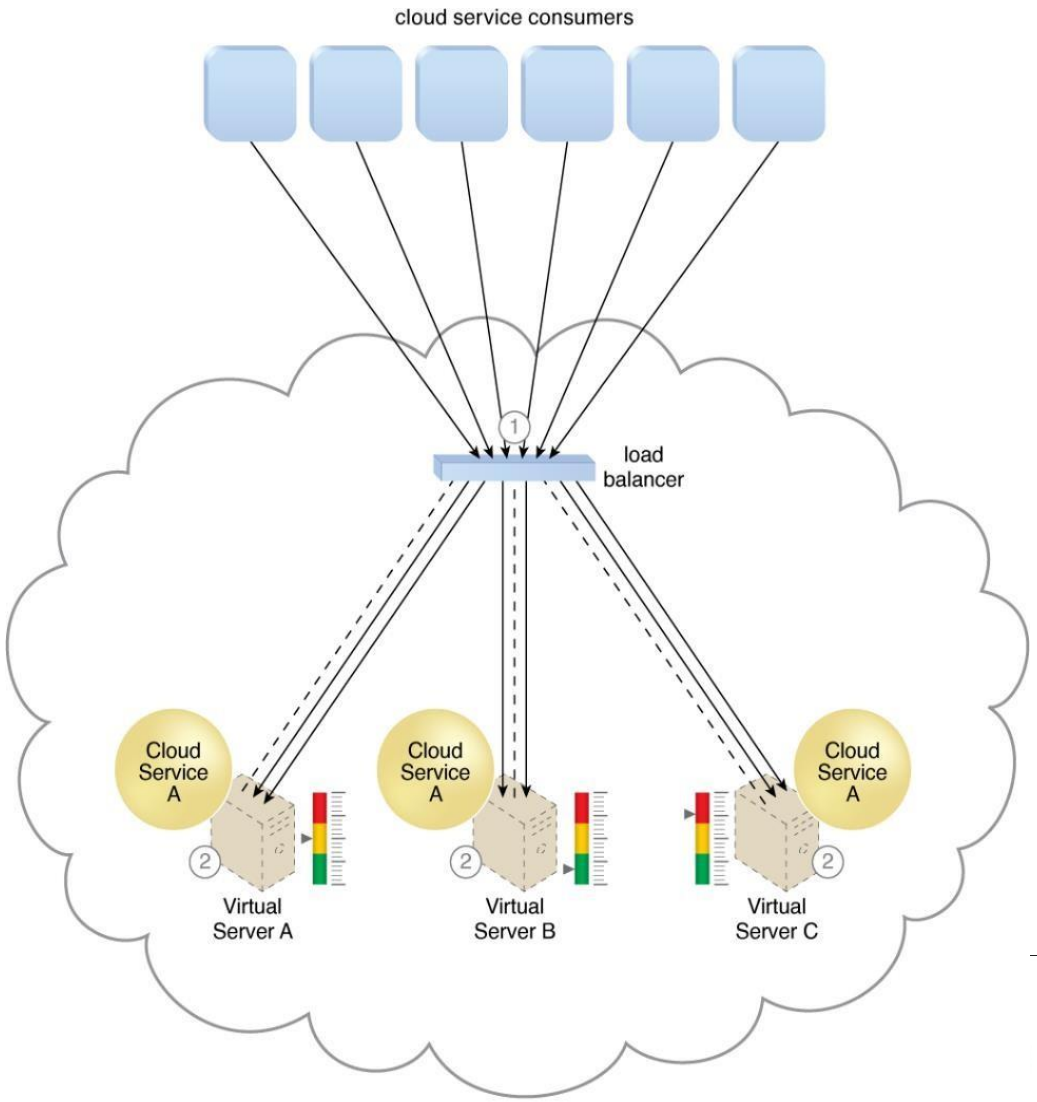
IT资源	服务入口	监控与审计
• 资源副本 • 资源集群	• 负载均衡器 • 逻辑网络边界	• 使用监控器 • VM监控器 • 审计监控器

服务负载均衡架构

- Service Load Balance Architecture
  - 工作负载分布架构的一个特殊变种
  - 专门针对扩展云服务实现的
- 基于云服务的冗余部署
  - 部署在虚拟机上或容器中的应用程序或软件
  - 多实例构成资源池
- 两种形式
  - 独立于云设备及主机服务器，或
  - 作为应用程序/服务器环境的内置组件

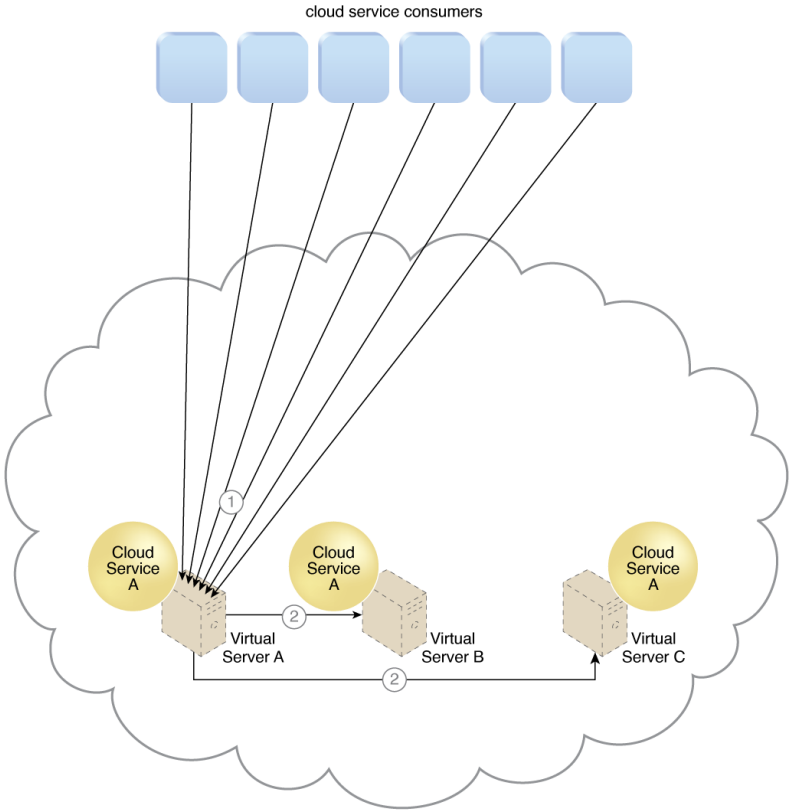
服务负载均衡架构两种工作方式（1/2）

Figure11.10 负载均衡器截获云服务用户发送的消息（1）并将其转发给虚拟服务器，从而使工作负载的处理得到水平扩展（2）。



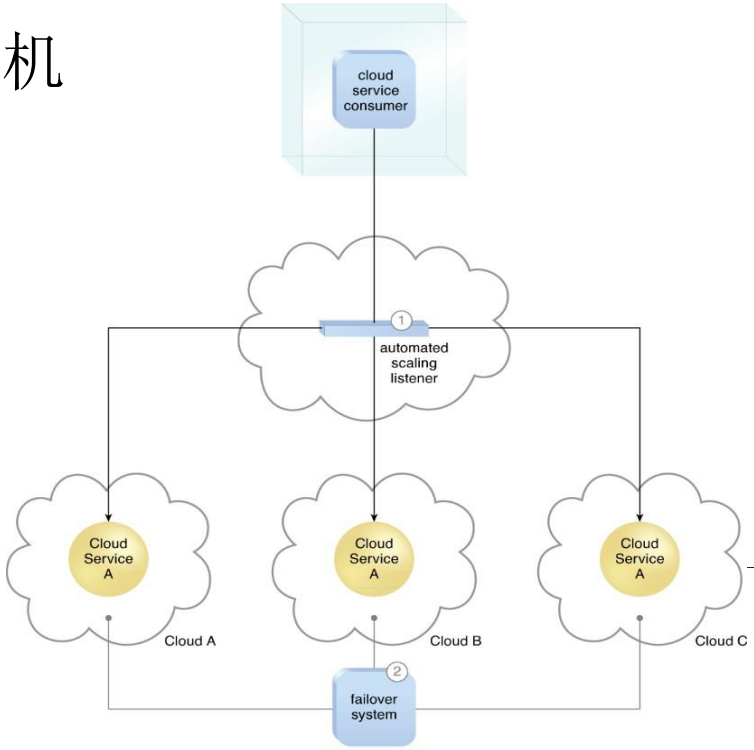
服务负载均衡架构两种工作方式（2/2）

Figure11.11 云服务用户的请求发送给虚拟服务器A上的云服务A（1）。内置负载均衡逻辑包含在云服务实现中，它可以将请求分配给相邻的云服务A，这些云服务A的实现位于虚拟服务器B和C上（2）。



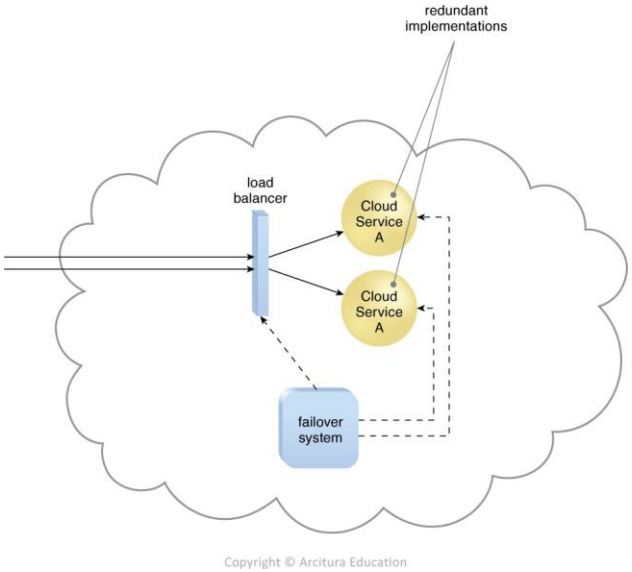
粗粒度负载均衡与资源预留

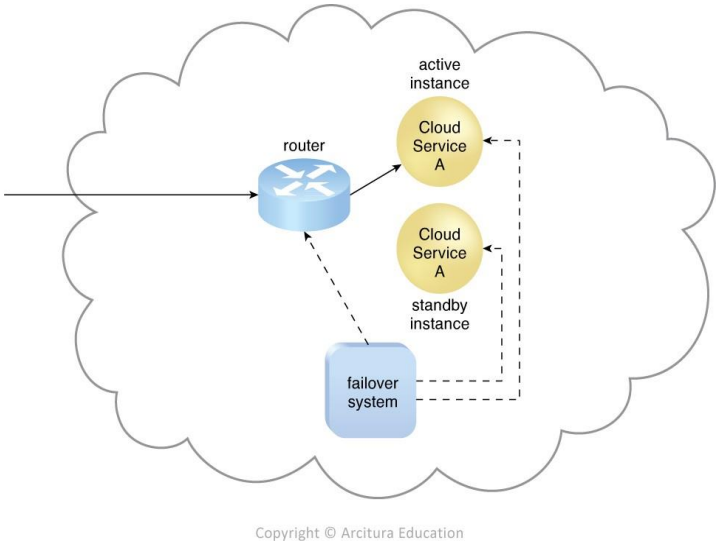
- 负载均衡的粒度有很多
- 用户请求
  - 前端LB把用户请求分发给不同的云服务实例
- 虚拟服务器实例
  - 把虚拟服务器放置到不同的物理机
- 云负载均衡
  - 数据中心间的负载均衡



故障转移系统

- 故障转移系统(failover system)
- 通过使用现有的集群技术提供冗余的实现来增加IT资源的可靠性和可用性。
- 只要当前活跃的IT资源变得不可用时，便会自动切换到冗余的或待机IT资源实例上。



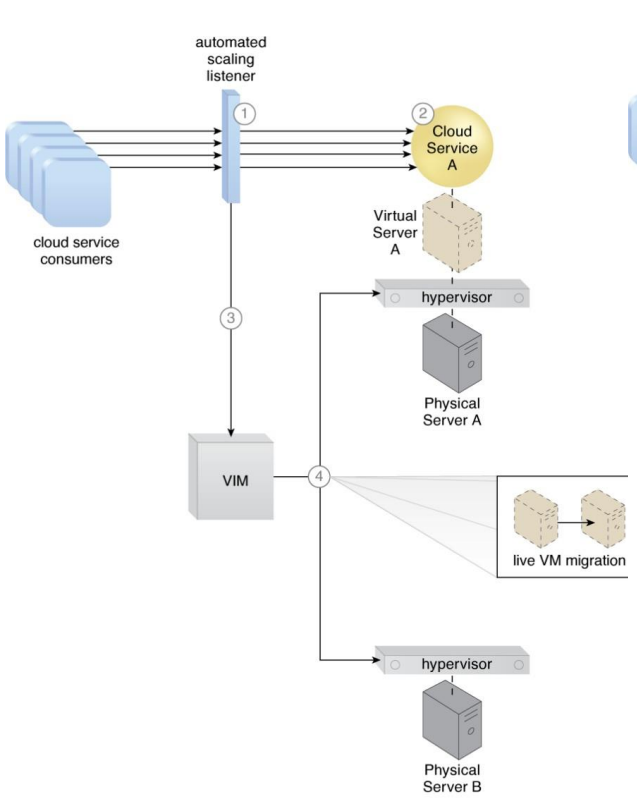


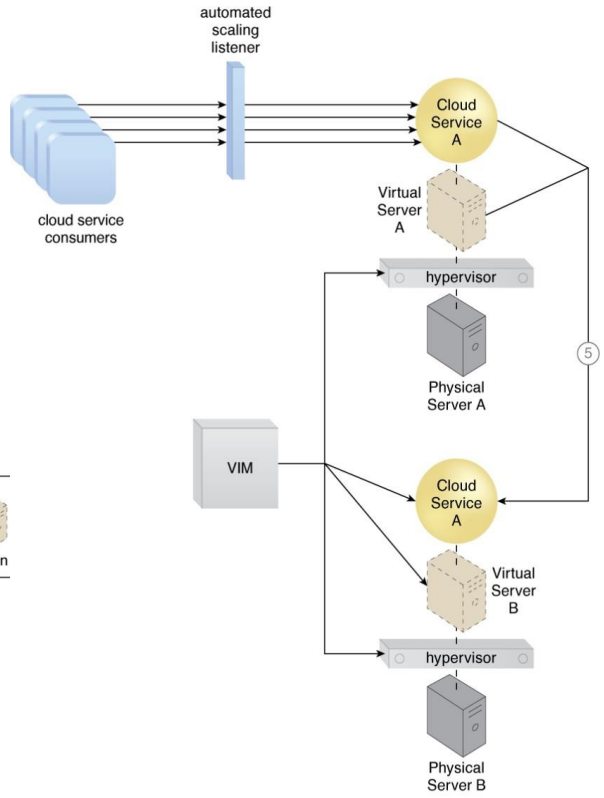
云服务容错架构

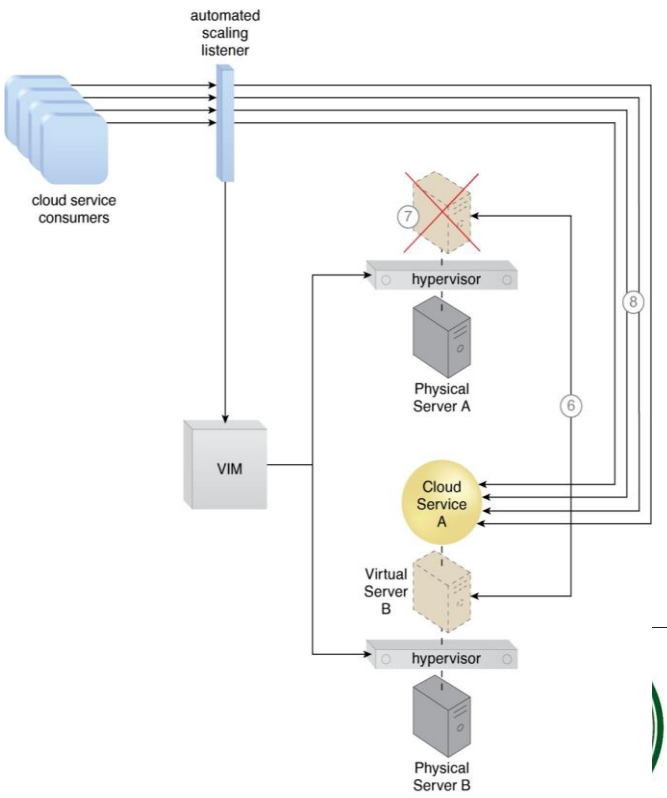
- 云服务不可用的原因很多：
  - 运行时需求超出处理能力
  - 维护更新导致的暂时中断
  - 云服务迁移
  - 物理机失效、宕机
- 应对机制
  - 服务迁移
  - 服务故障检测与恢复
  - 物理机容错

服务迁移

- 不中断服务重定位架构
- Non-disruptive service relocation architecture
  - 预先定义事件，触发云服务实现运行时复制或迁移

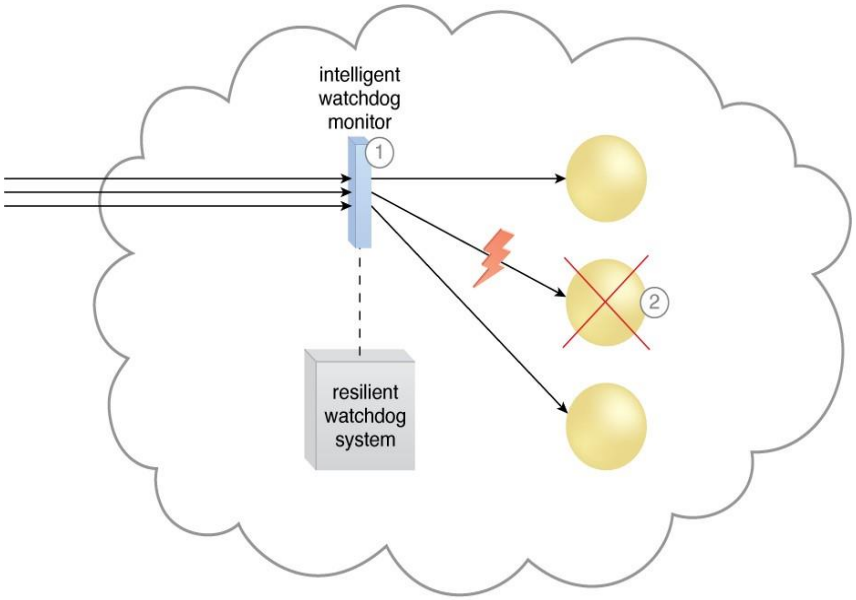


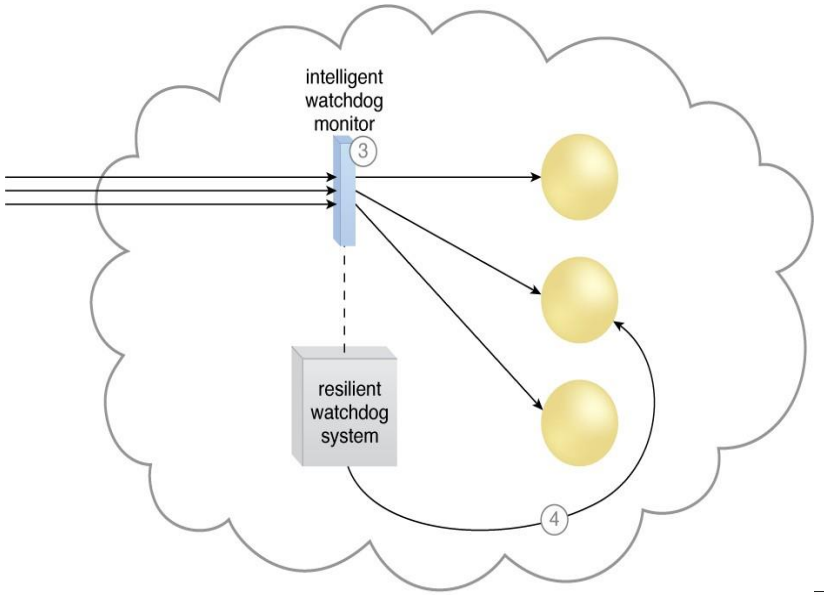




动态故障检测与恢复

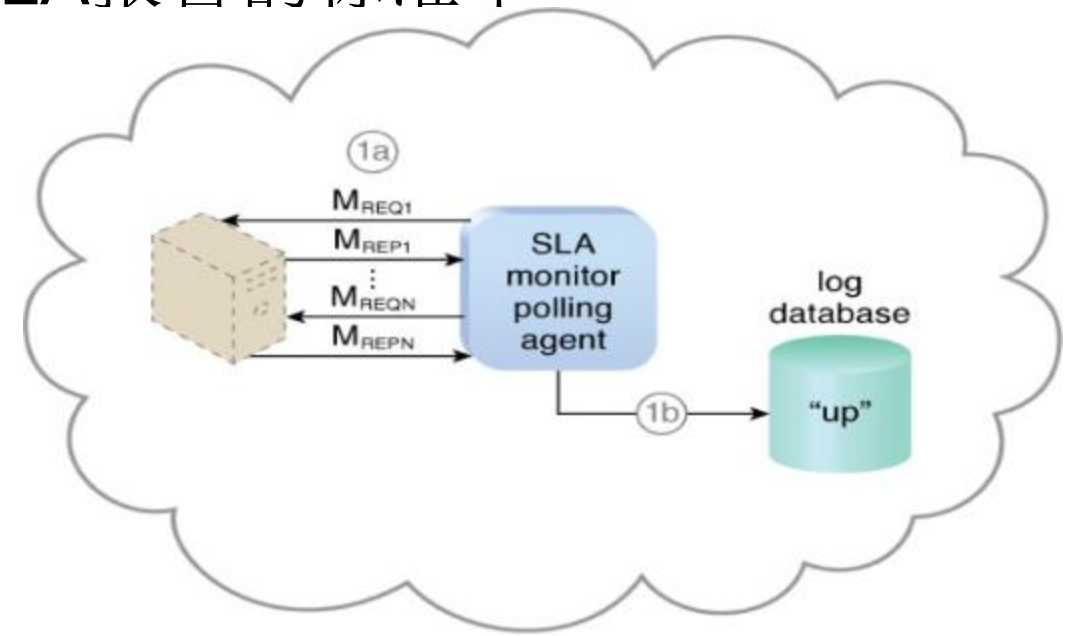
- Dynamic failure detection and recovery
  - 根据预先定义的故障场景
  - 通常基于弹性（Resilient）看门狗系统





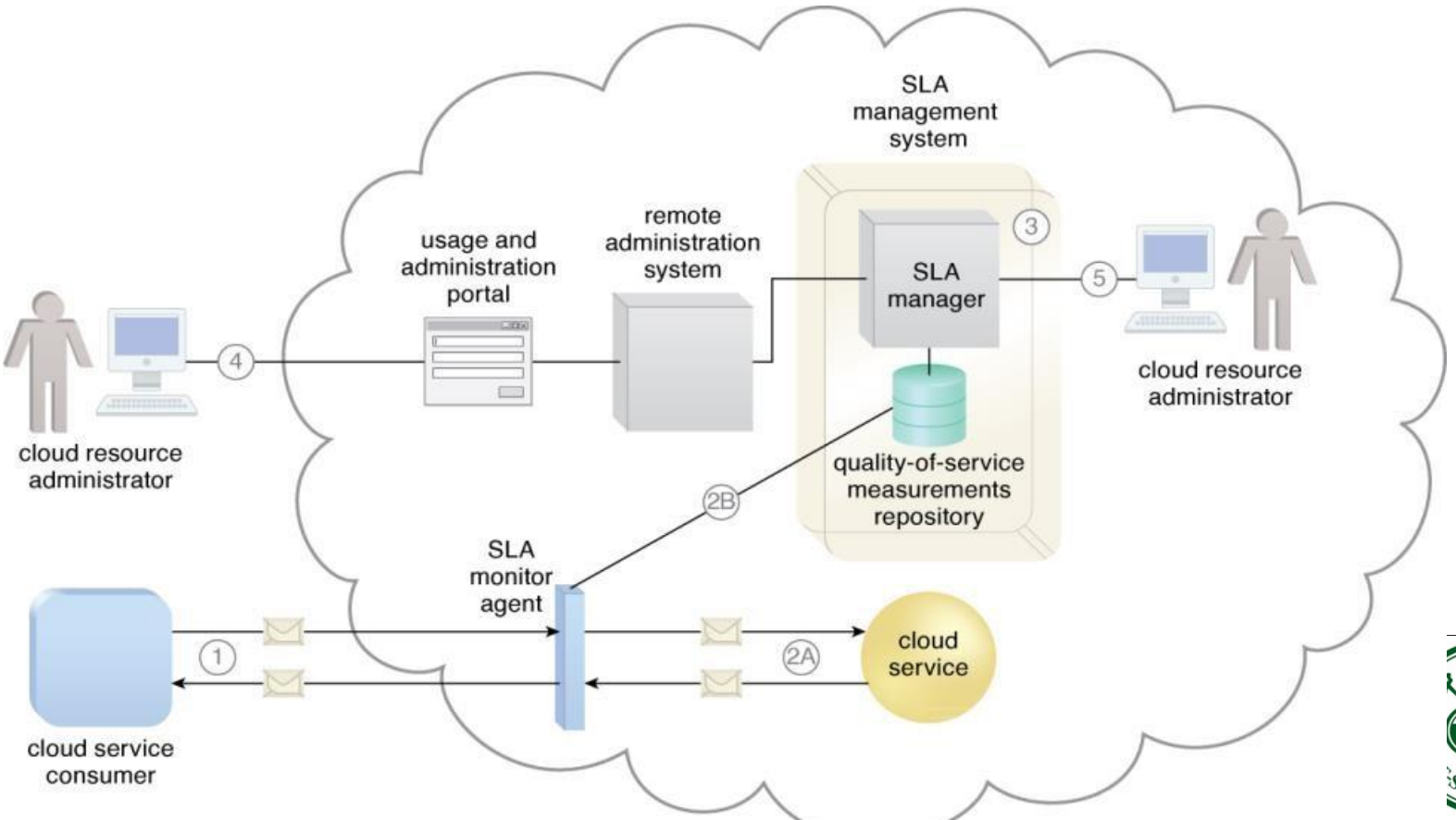
SLA监控器

- SLA监控器(SLA monitor)
- 用来专门观察云服务的运行时性能，确保它们履行了SLA中报告的约定的QoS需求。
- SLA监控器收集的数据由SLA管理系统处理并集成到SLA报告的标准中



SLA管理系统

- SLA管理系统两个任务：
  - 周期性测量QoS指标来验证是否与SLA保证相符合
  - 收集与SLA相关的数据，用于各种类型的统计分析

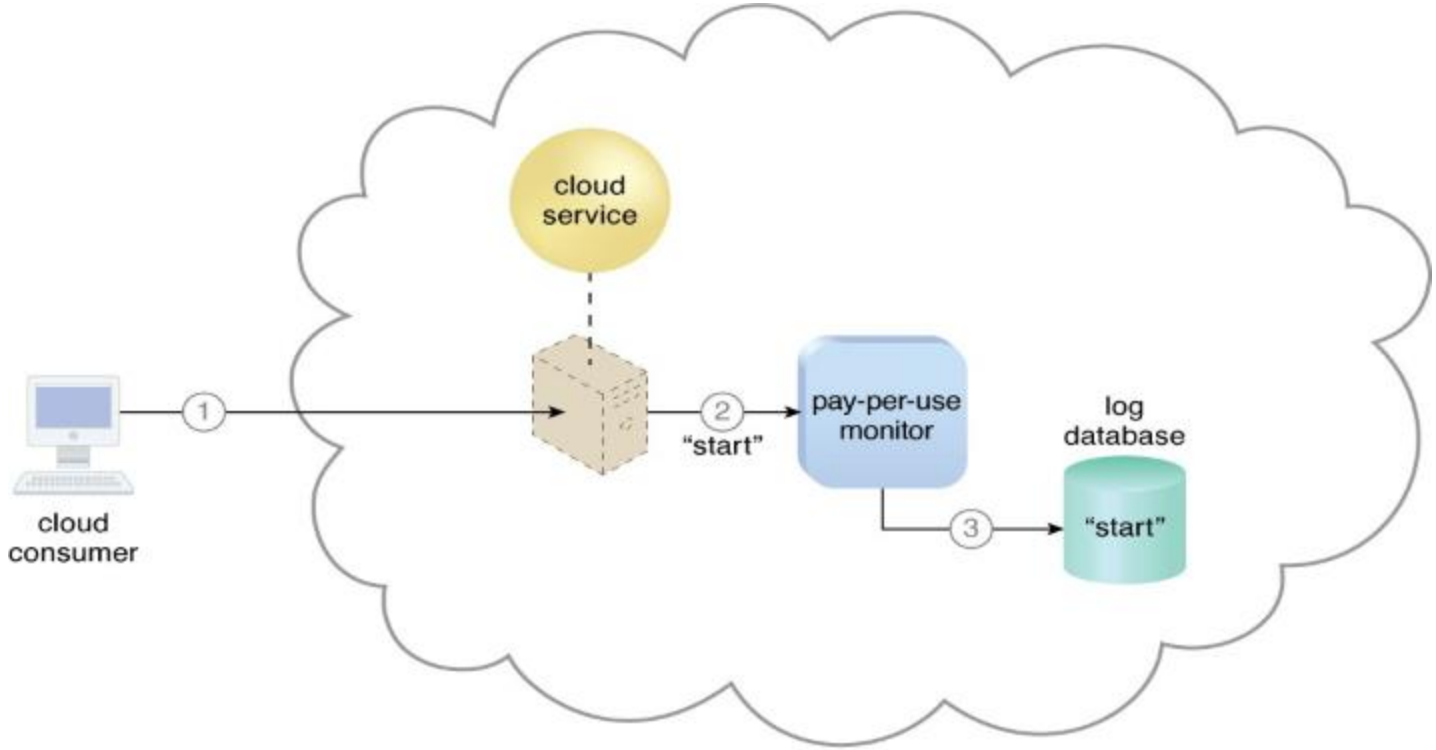


按使用付费监控器

- 按使用付费监控器(pay-per-use monitor)
- 按照预先定义好的定价参数测量云资源使用，并生成使用日志用于计算费用。
- 使用数据由计费管理系统(billing management system)进行处理。
- 实现方式
  - 资源代理
  - 监控代理

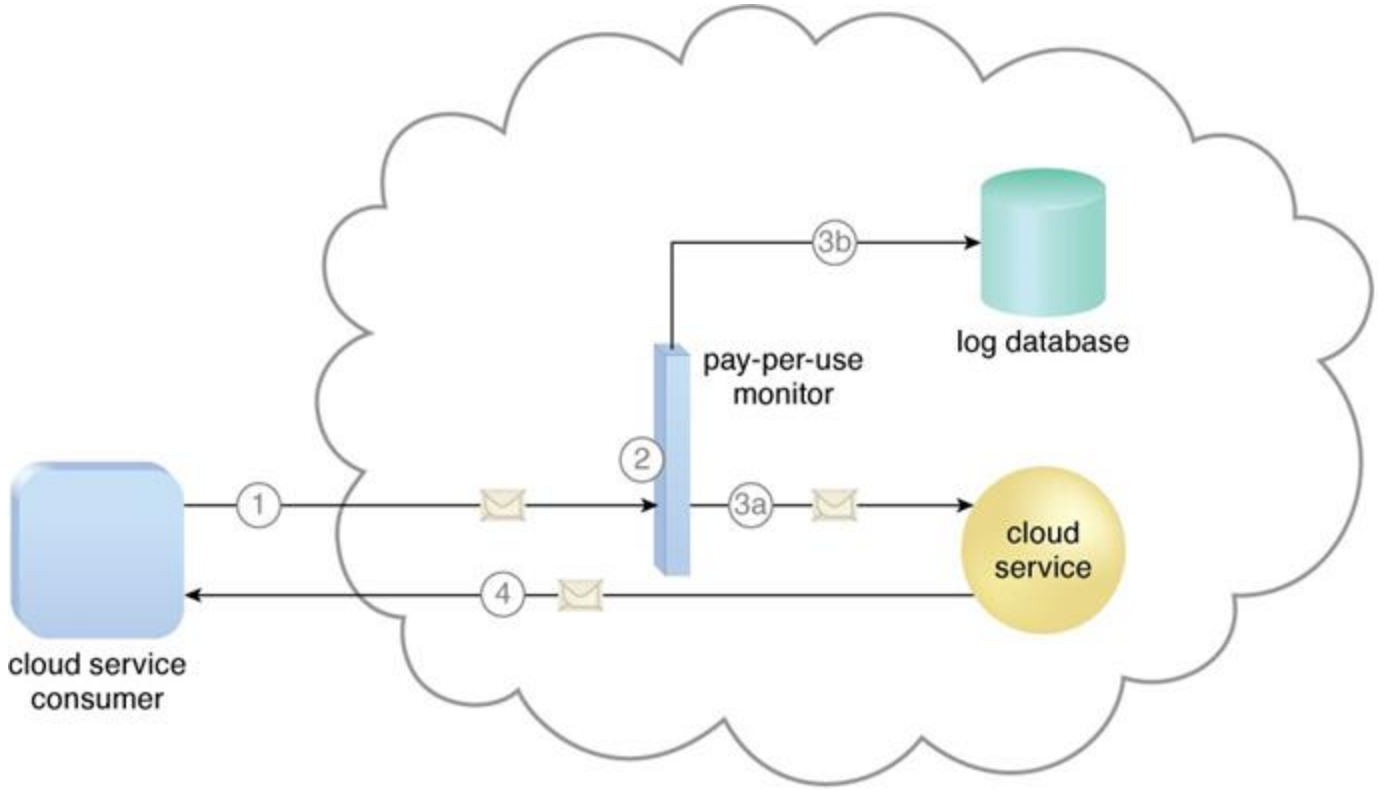
使用付费监控器作为资源代理

- A cloud consumer requests a new instance of a cloud service (1).
- The pay-per-use monitor receives a "start" event notification from the resource software (2).
- The pay-per-use monitor stores the value timestamp in the log database (3).



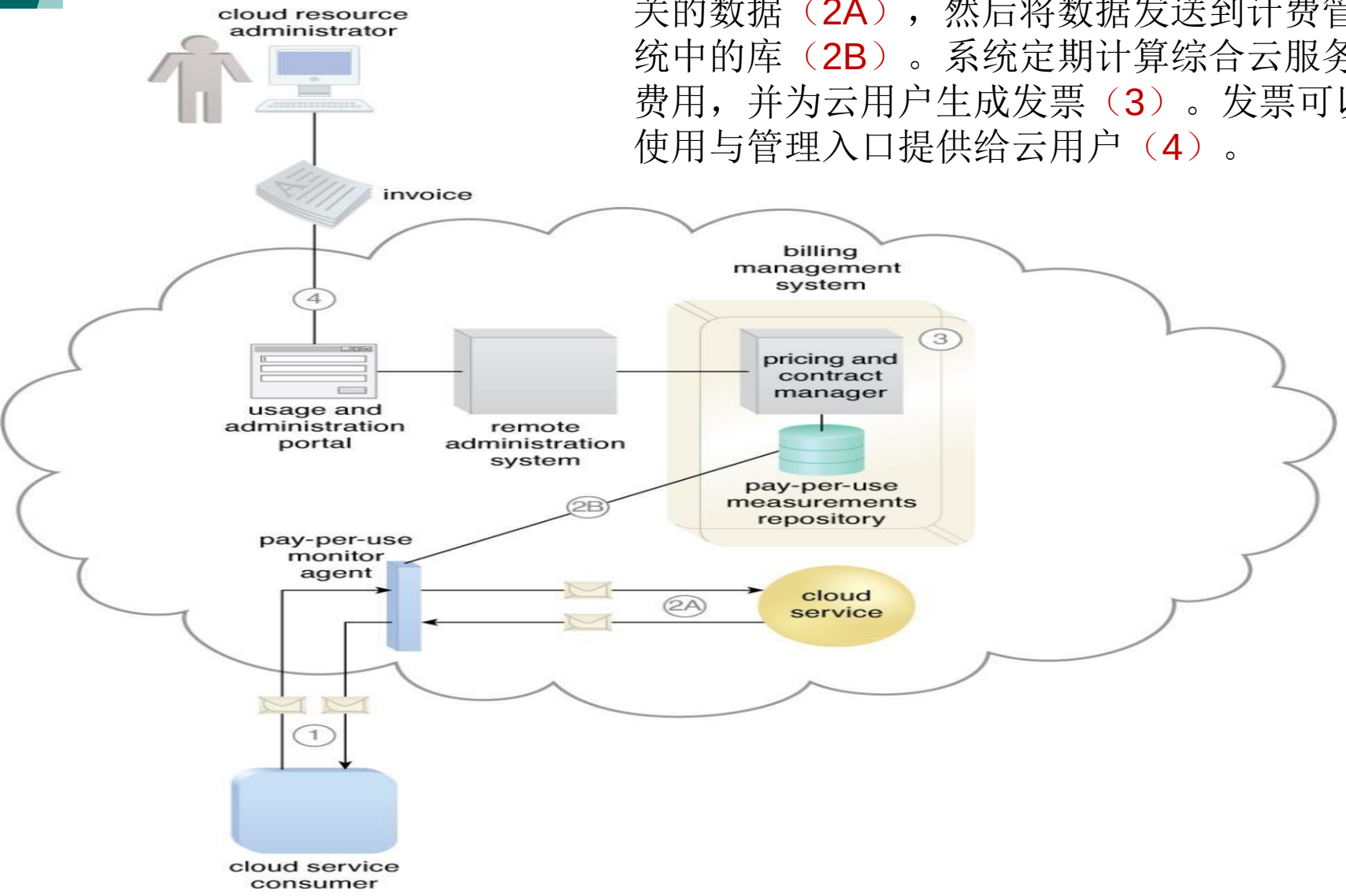
使用付费监控器作为监控代理

- 云服务用户向云服务发送请求消息（1）
- **按使用付费监控器**截获该消息（2），将他转发给云服务（3a），按照监控指标把使用信息存储起来（3b）
- 云服务讲响应消息转发回云服务用户，提供所请求的服务



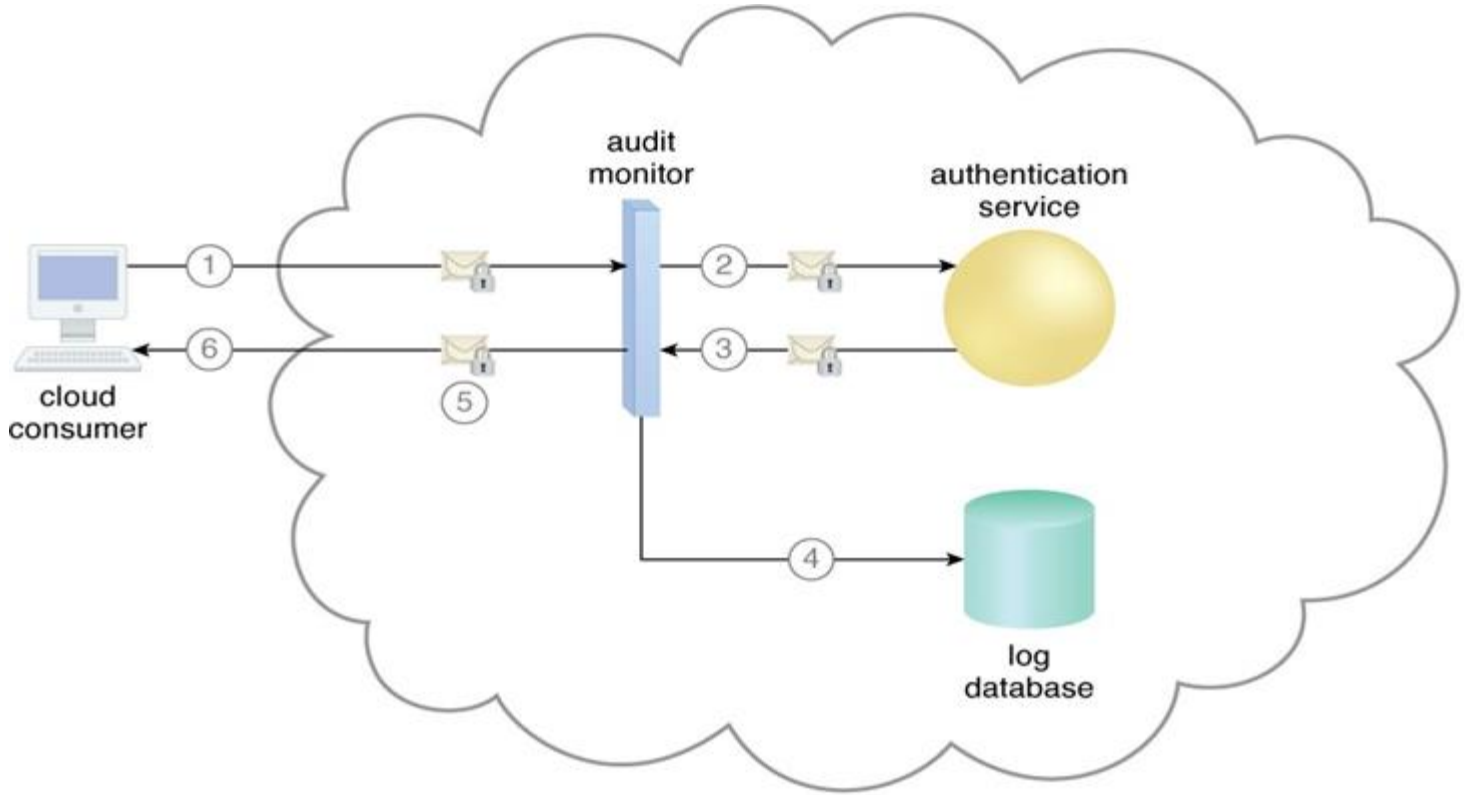
计费管理系统

Figure9.10 — 云服务用户与云服务交互（1）。按使用付费监控器跟踪使用情况，并收集与计费相关的数据（2A），然后将数据发送到计费管理系统中的库（2B）。系统定期计算综合云服务使用费用，并为云用户生成发票（3）。发票可以通过使用与管理入口提供给云用户（4）。



审计监控器

- 审计监控器(audit monitor)
- 用来收集网络和IT 资源的审计记录数据
- 用以满足管理需要或者合同义务。



远程管理系统

- 远程管理系统机制向外部云资源管理者提供工具和用户界面来配置并管理基于云的IT资源。
- 远程管理系统能够建立一个入口以便访问各种底层系统的控制与管理功能，包括资源管理、SLA管理和计费管理。
- 远程管理系统主要创建如下两种类型的入口：
  - 使用与管理入口（Usage And Administration Portal）——一种通用入口
  - 自助服务入口（Self-Service Portal）——本质上是一个购买门户，允许云用户搜索云提供者提供的最新云服务和IT资源列表

That’s NOT all!

考试：7月3日，周三，下午

地点：待定