2023-2024学年春季学期

计算机体系结构安全

Computer Architecture Security

授课团队：史岗，陈李维

计算机体系结构安全

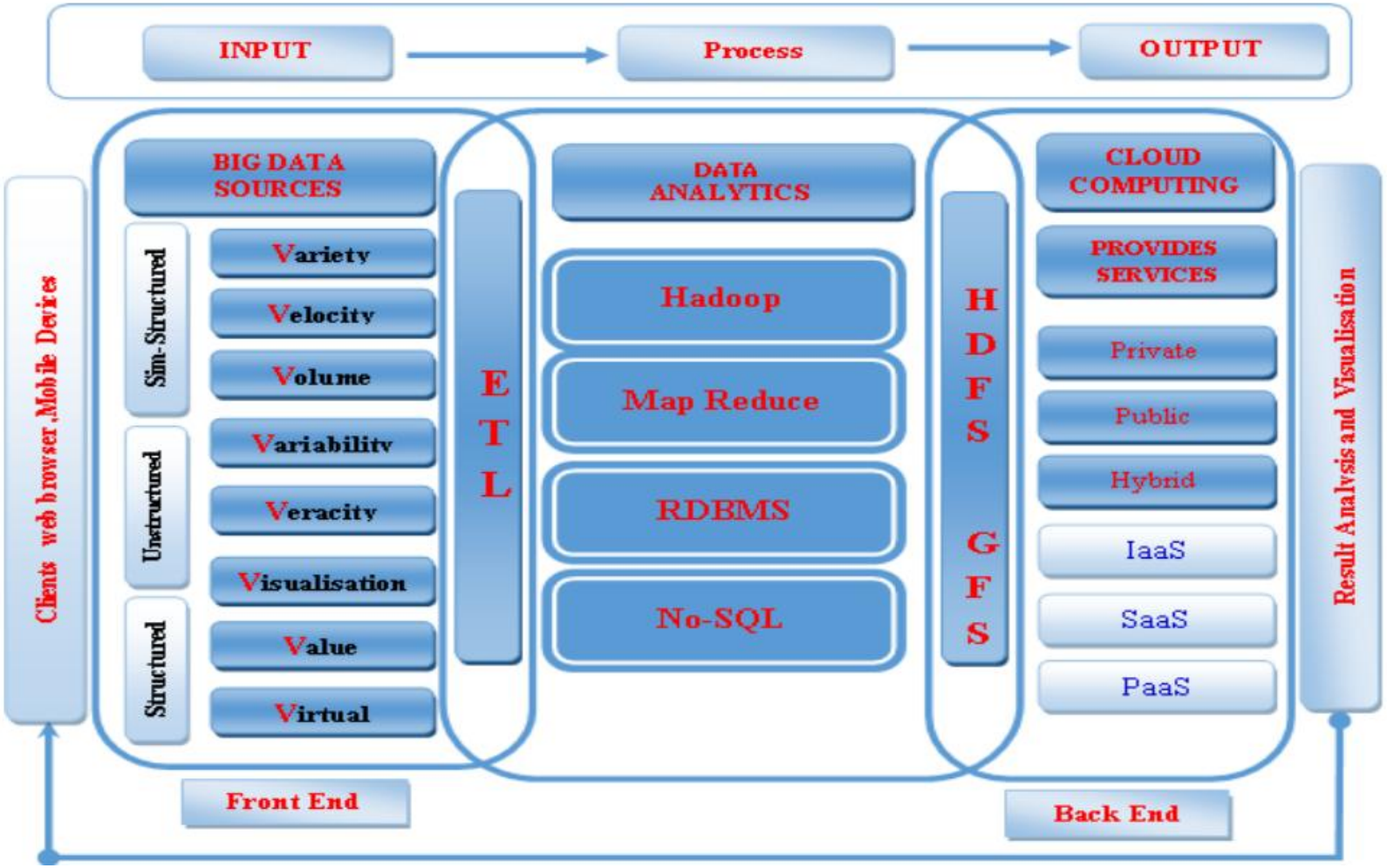
Computer Architecture Security

[第15次课] 云计算与大数据平台体系架构安全

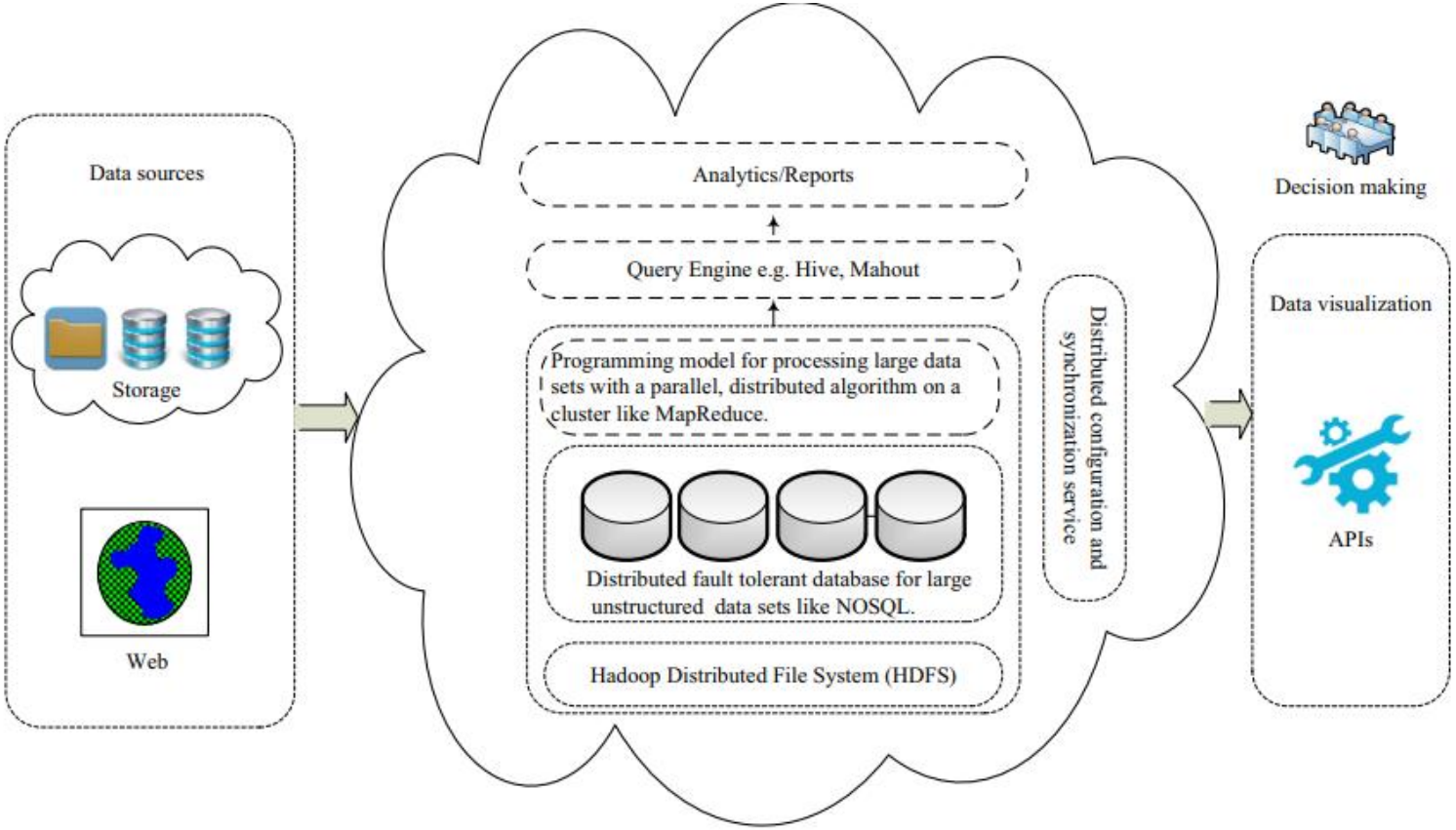
授课教师：史岗

授课时间：2024.6.3

云计算与大数据


INPUT
Process
OUTPUT
Clients web browser,Mobile Devices
BIG DATA SOURCES
Sim-Structured
Unstructured
Structured
Variety
Velocity
Volume
Variability
Veracity
Visualisation
Value
Virtual
E T L
DATA ANALYTICS
Hadoop
Map Reduce
RDBMS
No-SQL
H D F S
G F S
CLOUD COMPUTING
PROVIDES SERVICES
Private
Public
Hybrid
IaaS
SaaS
PaaS
Result Analysis and Visualisation
Front End
Back End

云计算与大数据



云计算在可用性、可扩展性和容错性上很好地满足了大数据的需求。

内容概要

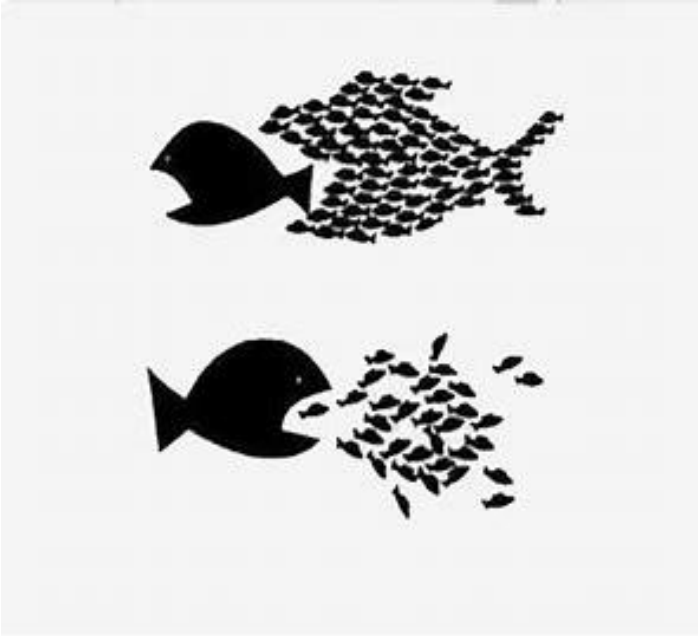
- 云计算平台
  - 云计算体系结构
  - 云计算安全风险
- 大数据平台
  - Hadoop体系结构
  - Hadoop安全风险
- 总结

内容概要

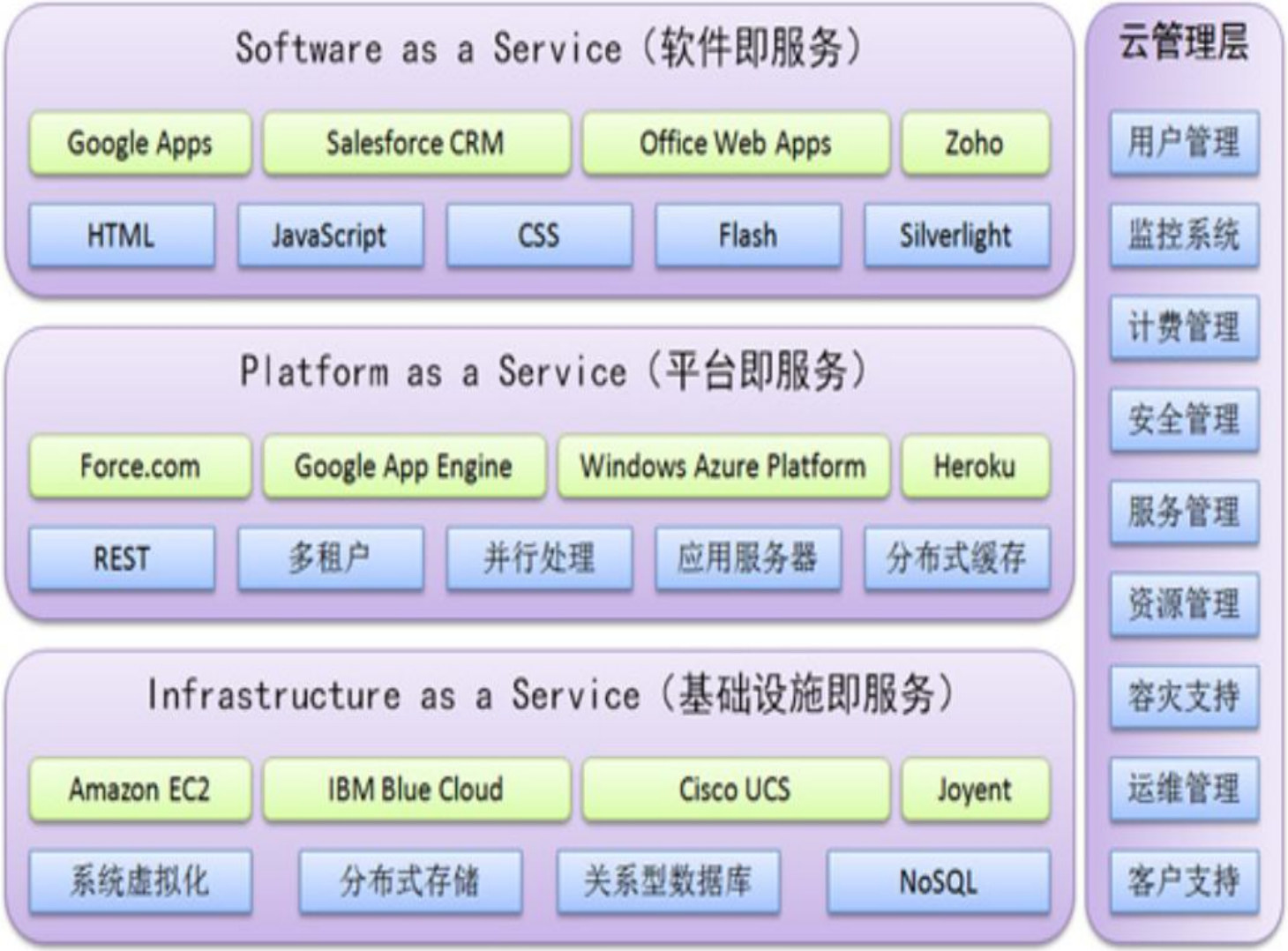
- 云计算平台
  - 云计算体系结构
  - 云计算安全风险
- 大数据平台
  - Hadoop体系结构
  - Hadoop安全风险
- 总结

云计算定义

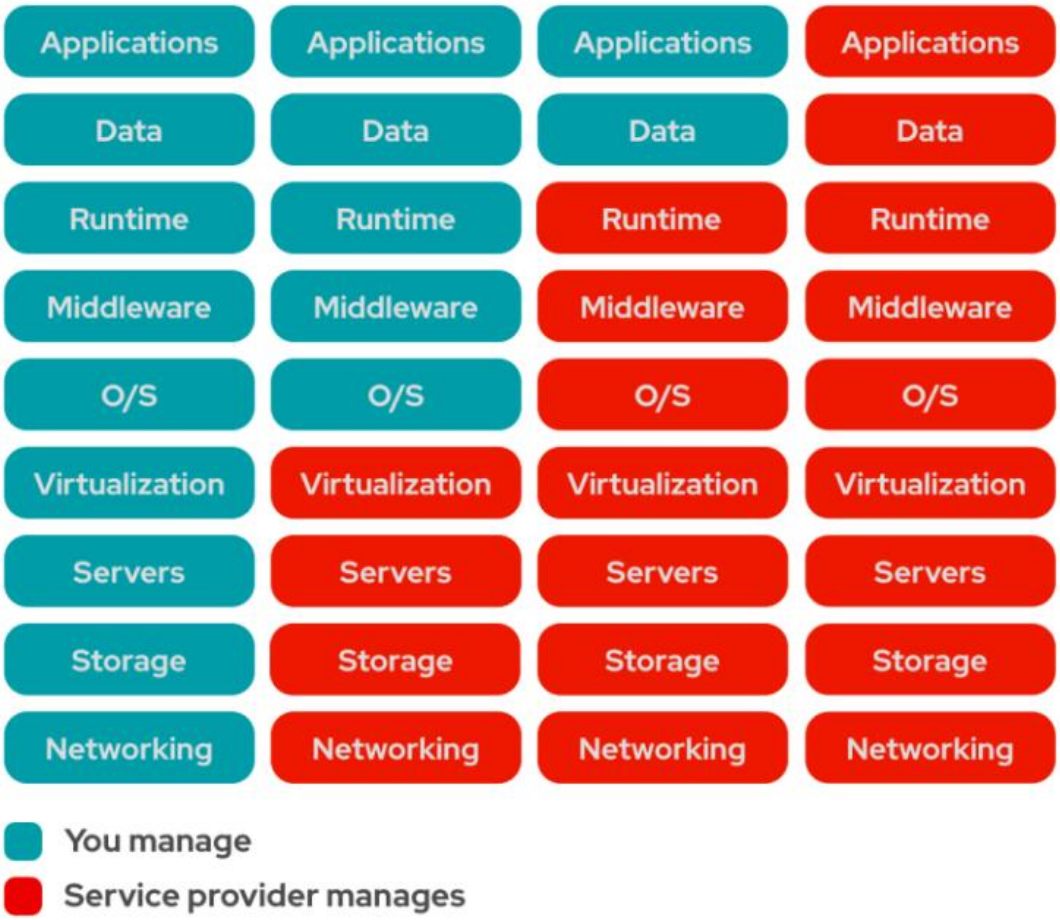
- 美国国家标准与技术研究院（NIST）：云计算是一种模型，它可以随时随地、方便地、随需应变地从可配置的共享计算资源池中获取资源（如，网络、服务器、存储、应用程序、服务），这些资源可以快速供应并释放，使管理资源的工作量与服务提供商的交互减小到最低程度。



云计算架构


Software as a Service（软件即服务）
Google Apps
Salesforce CRM
Office Web Apps
Zoho
HTML
JavaScript
CSS
Flash
Silverlight
Platform as a Service（平台即服务）
Force.com
Google App Engine
Windows Azure Platform
Heroku
REST
多租户
并行处理
应用服务器
分布式缓存
Infrastructure as a Service（基础设施即服务）
Amazon EC2
IBM Blue Cloud
Cisco UCS
Joyent
系统虚拟化
分布式存储
关系型数据库
NoSQL
云管理层
用户管理
监控系统
计费管理
安全管理
服务管理
资源管理
容灾支持
运维管理
客户支持

- 云计算架构
  - SaaS层涉及的技术
    - HTML、Javascript CSS、Flash
  - PaaS层涉及的技术
    - 并行处理
    - 分布式缓存
    - 多租户
  - IaaS层涉及的技术
    - 虚拟化
    - 分布式存储



虚拟化技术

- **核心思想：**利用软件或固件管理程序构成虚拟化层，把物理资源映射为虚拟资源，实现软硬件彻底解耦；在虚拟资源上可安装部署多个虚拟机，实现多用户共享物理资源

虚拟化级别	性能	应用程序灵活性	实现复杂度	应用程序隔离性
ISA级虚拟化	★	★★★★★	★★★	★★★
硬件级虚拟化	★★★	★★★	★★★★★	★★★★
操作系统级虚拟化	★★★★★	★★	★★★	★★
运行时库虚拟化	★★★	★★	★★	★★
应用程序级虚拟化	★★	★★	★★★★★	★★★★★

- 硬件虚拟化（架构维度）
  - **寄居虚拟化：** Hypervisor运行在基础操作系统上，构建出一整套虚拟硬件平台，支持创建各种操作系统类型虚拟机，主要面向桌面计算机。代表性产品有VMware WorkStation、Redhat KVM。（Type-II Hypervisor）
  - **裸金属虚拟化：** Hypervisor直接运行在硬件上，直接与硬件交互提升效率，主要面向服务器。代表性产品有VMware ESXServer、Citrix XenServer、Microsoft Hyper-V。（Type-I Hypervisor）
- 硬件虚拟化（技术维度）
  - **全虚拟化：** Guest OS通过Hypervisor来最终分享硬件，VM发出的指令需经过Hypervisor捕获并处理。
  - **半虚拟化：** Guest OS增加一个专门的API，这个API可以将VM发出的指令进行最优化，即不需要Hypervisor耗费一定的资源进行翻译操作。
  - **硬件支持虚拟化：** Hypervisor可以在部分功能上与硬件直接交互，提升性能。比如在CPU性能较差的网络IO方面与硬件直接交互。

- 计算虚拟化（CPU虚拟化）
  - 目的：提供一种方法，使得虚拟机能够在物理计算机上运行，并且能够在不同的虚拟机之间实现隔离和资源分配。
  - 方法1：VMM逐条读取虚拟机的指令，并根据指令模拟执行。

```
while(1){
    curr_instr = fetch(virtHw.PC);
    virtHw.PC += 4;
    switch(curr_instr){
        case ADD:
            int sum = virtHw.regs[curr_instr.reg0] +
                      virtHw.regs[curr_instr.reg1];
            virtHw.regs[curr_instr.reg0] = sum;
            break;
        case SUB:
            //...etc...
```

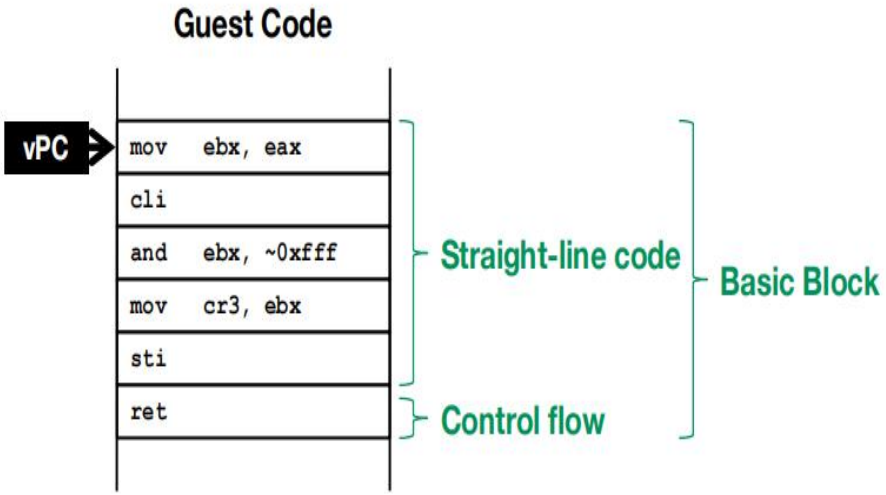
优点：可处理所有类型的指令，方法通用性好
缺点：实现很难且特别慢！

- 计算虚拟化（CPU虚拟化）
  - 方法2：部分硬件直接执行，部分模拟方法运行。
    - 直接执行：大多数非敏感指令直接在硬件上执行，无需VMM（虚拟机监视器）的干预，这样可以提高性能。
    - 陷阱-仿真：对于敏感指令，即那些可能影响虚拟化环境稳定性或安全性的指令，它们会被“陷阱”（trap），然后由VMM来模拟这些指令的效果。

优点：对于非特权指令，几乎没有额外性能开销。VM无法直接访问或影响其他VM或宿主机的硬件，隔离性也较好。

缺点：对于特权指令，由于需要陷阱和仿真，增加复杂性，开销大。

- 计算虚拟化（CPU虚拟化）
  - 方法3：二进制翻译。
    - VMM（虚拟机监视器）动态地重写客户操作系统的二进制指令，使得原本不能直接在硬件上执行的指令能够通过翻译后执行。



TU：基本代码块（包含12条指令或者以改变控制流结尾的指令为尾）

CCF(Compiled Code Fragment)：编译后的代码片段

TC(Translation Cache)：存储CCF的Cache

当Guest OS要执行TU时，会在TC中查找是否已有翻译后的CCF：

·如果没有，执行翻译过程，将TU翻译为CCF，并存储在TC中。

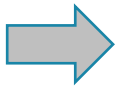
·如果有，可以直接从TC中获取CCF，从而提高执行效率。

当执行完CCF后，调用Translator（由VMM实现），来决定下一个要被翻译的TU。

内存虚拟化

- 目标：允许多个虚拟机（VMs）在单个物理服务器上运行，并且每个虚拟机都能够安全地访问内存资源，而不会相互干扰。
- 虚拟地址 GVA → 客户机物理地址 GPA → 宿主机物理地址 HPA

Process	Operating System
1. Load from memory: TLB miss: Trap	
	2. OS TLB miss handler: Extract VPN from VA; Do page table lookup; If present and valid: get PFN, update TLB; Return from trap
3. Resume execution (@PC of trapping instruction); Instruction is retried; Results in TLB hit	

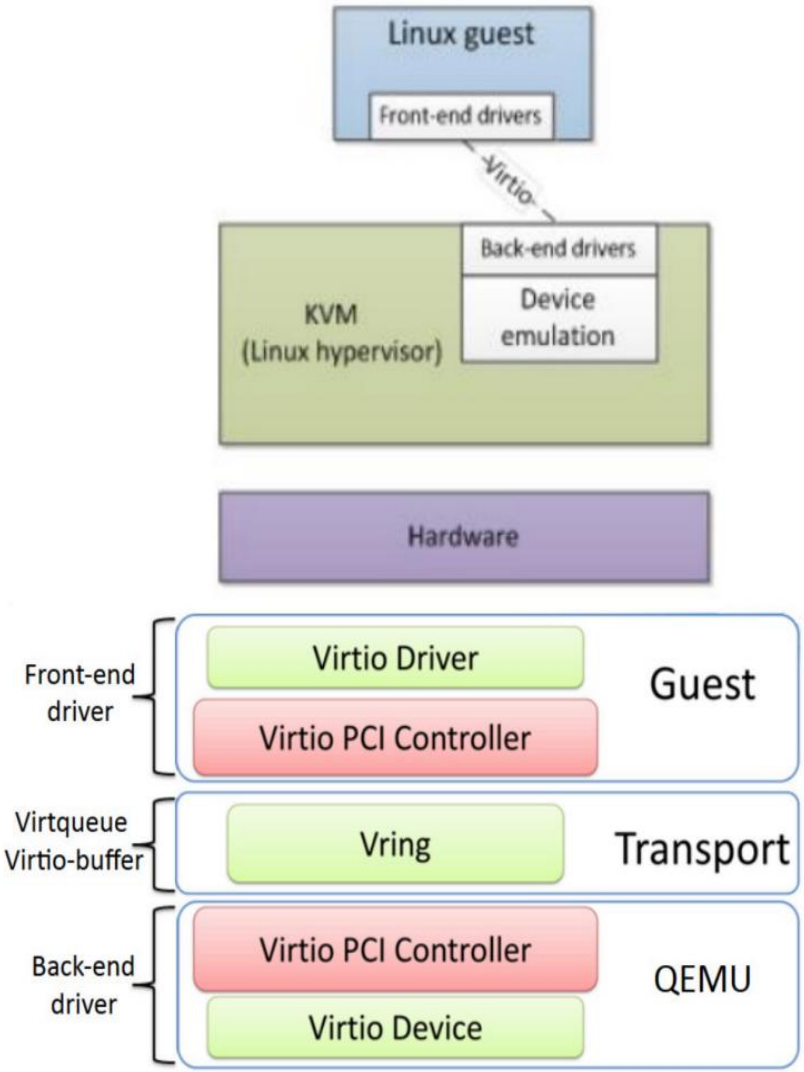


Process	Operating System	Virtual Machine Monitor
1. Load from mem TLB miss: Trap		
		2. VMM TLB miss handler: Call into OS TLB handler (reducing privilege)
	3. OS TLB miss handler: Extract VPN from VA; Do page table lookup; If present and valid, get PFN, update TLB	
		4. Trap handler: Unprivileged code trying to update the TLB; OS is trying to install VPN-to-PFN mapping; Update TLB instead with VPN-to-MFN (privileged); Jump back to OS (reducing privilege)
	5. Return from trap	
		6. Trap handler: Unprivileged code trying to return from a trap; Return from trap
7. Resume execution (@PC of instruction); Instruction is retried; Results in TLB hit		

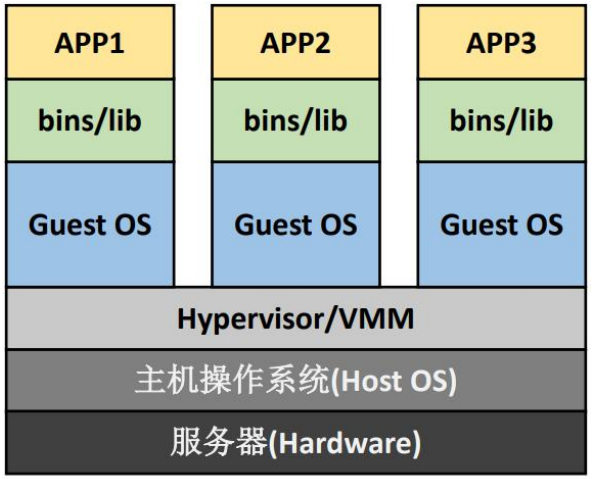
- VMM通过1次TLB Miss和2次Trap实现从Guest OS虚地址到宿主机物理地址的转化。

- I/O虚拟化
  - 目标：在客户虚拟机之间多路复用物理外围设备。
  - 解决方案：
    - 直接访问：虚拟机独占拥有一个设备
    - 设备仿真：虚拟机监视器在软件中模拟设备
    - 准虚拟化：将驱动程序拆分为客户部分（前端）和主机部分（后端），Guest OS运行新驱动程序；VMM为每个前端运行后端驱动程序，并最终运行真实设备驱动程序来驱动设备。
    - 硬件辅助：一个物理设备虚拟出多个虚拟设备给虚拟机使用。

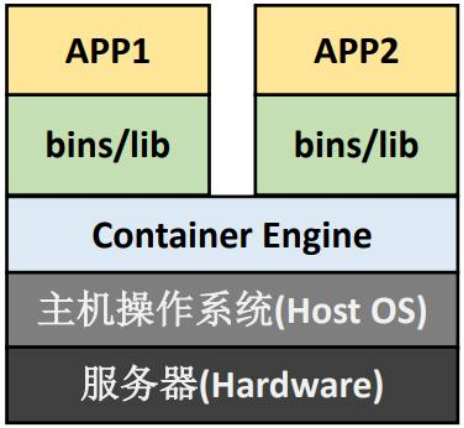
- virtio：Linux的准虚拟化I/O解决方案
  - Front-end Driver
    - Guset OS中的内核模块
    - 接受来自用户进程的I/O请求
    - 将I/O请求传输到Back-end Driver
  - Back-end Driver
    - 接受来自front-end driver的I/O请求
    - 通过物理设备执行I/O操作
  - Virtqueue
    - 一个可从guest和host操作系统访问的内存区域
    - 实现为vring的接口



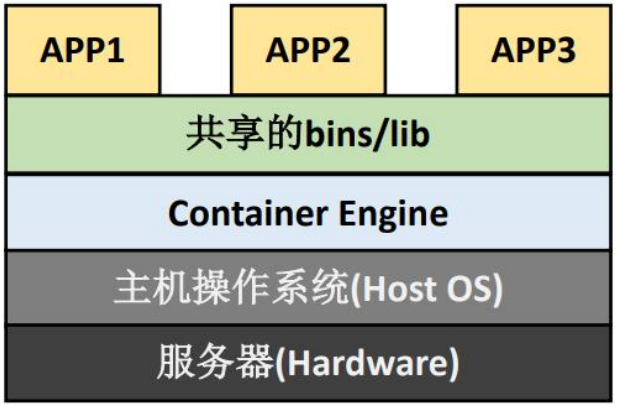
- 操作系统级虚拟化
  - 没有 Hypervisor / VMM，在主机操作系统中插入一个虚拟化层
  - 从安全的视角看，操作系统虚拟化一般给恶意和非可信的软件提供执行环境
  - 沙箱也是一种操作系统虚拟化技术



虚拟机



容器



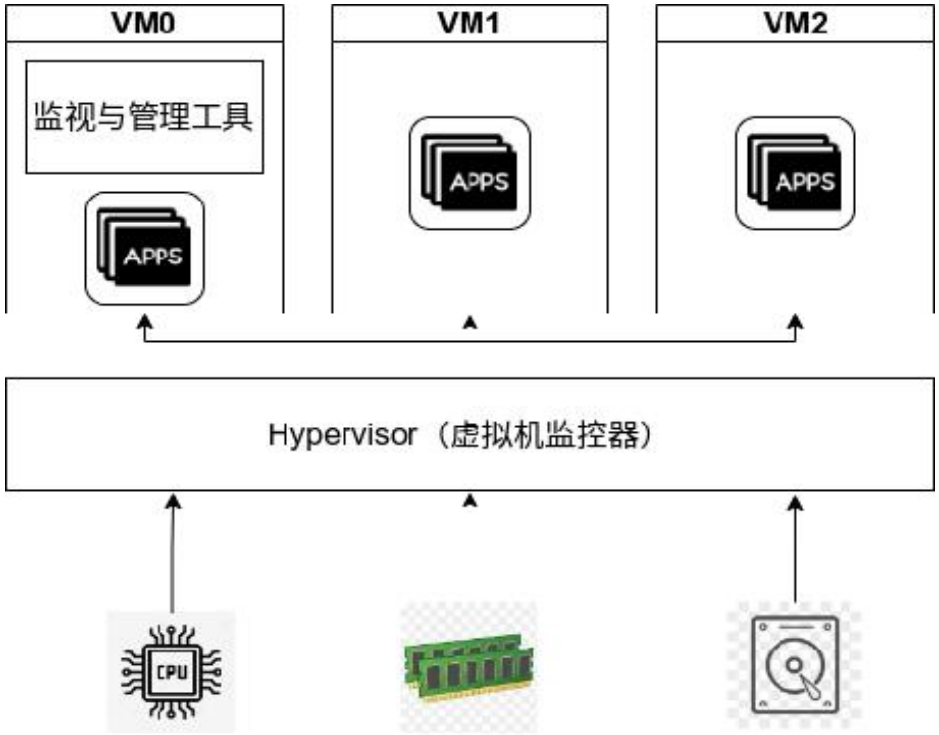
轻量级容器

内容概要

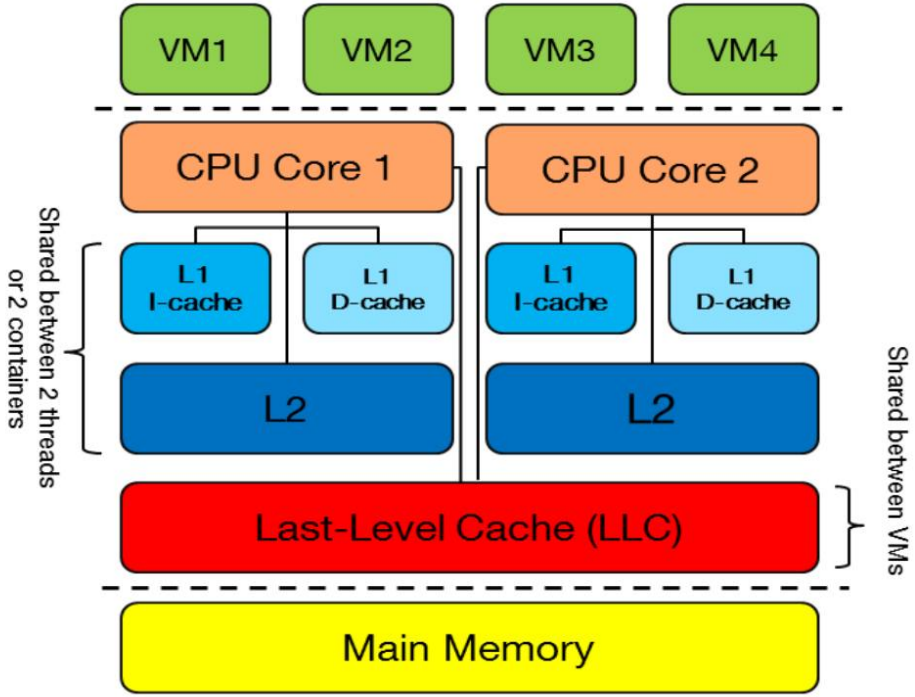
- 云计算平台
  - 云计算体系结构
  - 云计算安全风险
- 大数据平台
  - Hadoop体系结构
  - Hadoop安全风险
- 总结

云虚拟化安全

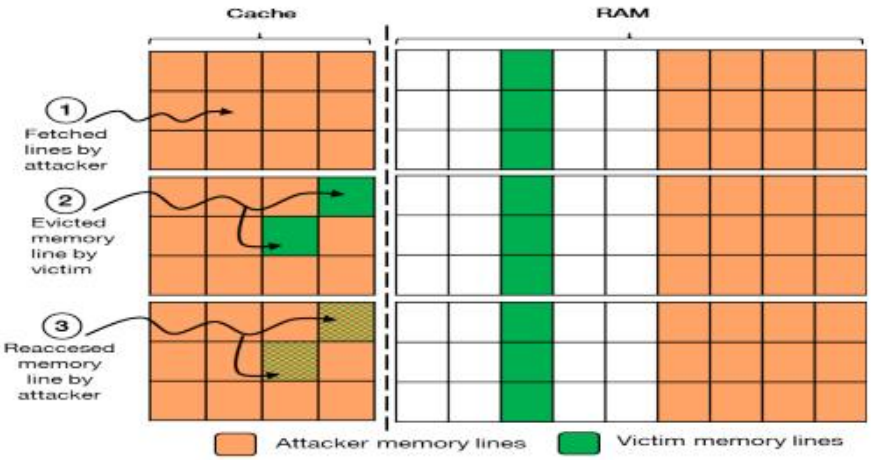
- 云虚拟化作为云计算的核心技术，其安全性至关重要
- 虚拟化引入了**虚拟化层**功能，导致了新的攻击面
- 不同的虚拟机可能会**共享一部分的物理内存和CPU资源**。共享资源有可能引入新的侧信道风险，为虚拟机窃取数据提供了可能。

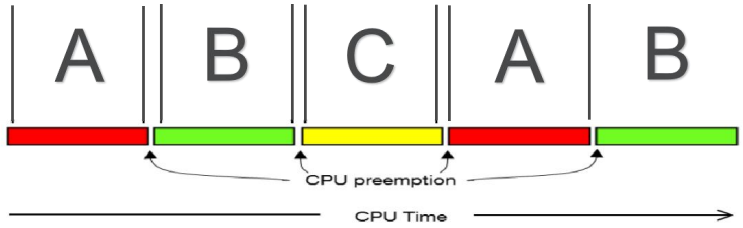


- 跨虚拟机侧信道攻击
  - 利用的脆弱性：CPU缓存
  - 攻击对象：虚拟机层级
  - 攻击场景：云环境中，攻击者和受害者的虚拟机使用相同的CPU资源
  - 虚拟化环境下，不同的VM会共享末级缓存，因为不同VM同时调度到同一个核的难度较大，但调度到同一个CPU的可能性较大（**资源共享条件**）



- 跨虚拟机侧信道攻击
  - CPU中不同任务的执行时间存在交叉，CPU抢占后的利用缓存布局可推测被观测任务的缓存使用情况。 **（状态变化条件）**
    - 攻击者在受害者抢占前填充缓存的全部内容
    - 受害者抢占，驱逐攻击者填充的部分缓存并写入自己的缓存
    - 攻击者抢占，探测缓存读取情况判断受害者填充缓存的地址
    - 重复进行上述步骤
    - “Prime + Probe”方式





*参考论文：
Side Channels in the Cloud: Isolation Challenges, Attacks, and Countermeasures

跨虚拟机侧信道攻击

高精度时钟获取（时钟同步条件）

- 从CPU中特定寄存器获取
- 观察读缓存前后时钟的变化，判断缓存是否被驱逐

```
volatile long long timer =0;
unsigned long long get_cpu_cycle()
{
    __asm__ __volatile__(
    "rdtsc" :
    "=d" (reth1),
    "=a" (retl0)
    );
    return ((reth1 << 32)|(retl0));
}
```

获取特殊寄存器值

```
int main()
{
    long long timestamp = 0;
    while (1)
        timestamp = get_cpu_cycle();

    ...
    // Analysis timestamp end
    ...
}
```

参考时钟

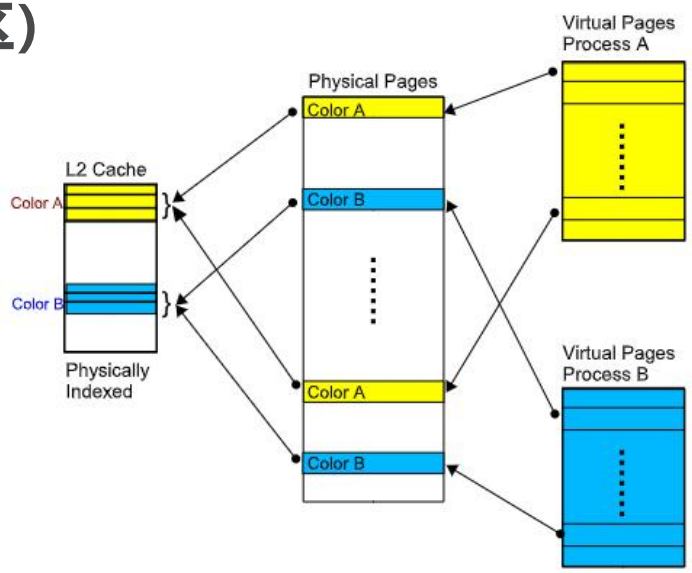
跨虚拟机侧信道攻击

缓解方案1：避免资源的同时托管（资源的粗粒度分区）

- 简单地禁止来自不同租户的虚拟机托管在同一物理资源上，从而防止攻击者和受害者虚拟机之间共享资源。
- 这种方法从根本上与云计算的核心动机相悖：通过共享提高资源利用率来降低成本。

缓解方案2：缓存着色（资源的细粒度分区）

- 通过染色的思路，将CPU中的缓存分配给不同的Guest VM，该方案可以通过Hypervisor配合CPU结构实现。

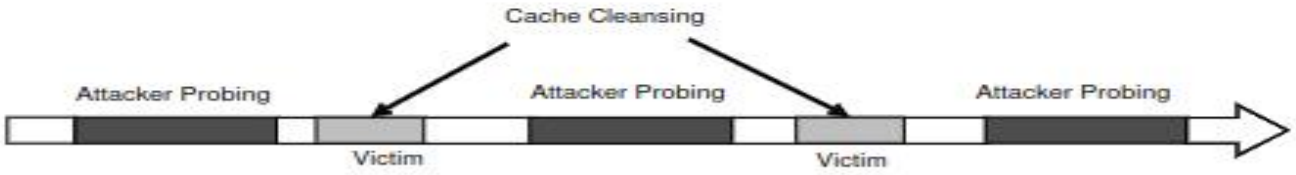


跨虚拟机侧信道攻击

缓解方案3：在缓存中添加噪声（OS中的防御措施）

- 正常用户可以在操作系统中部署内核线程，定时向缓存中存入无意义数据（噪声），清洗CPU中的已有缓存。
- 优点是原理简单，不需要修改Hypervisor层；缺点是会引入额外性能开销，破坏了缓存的设计初衷

(a) Attackers exploits time-shared caches by preempting victim VCPUs.

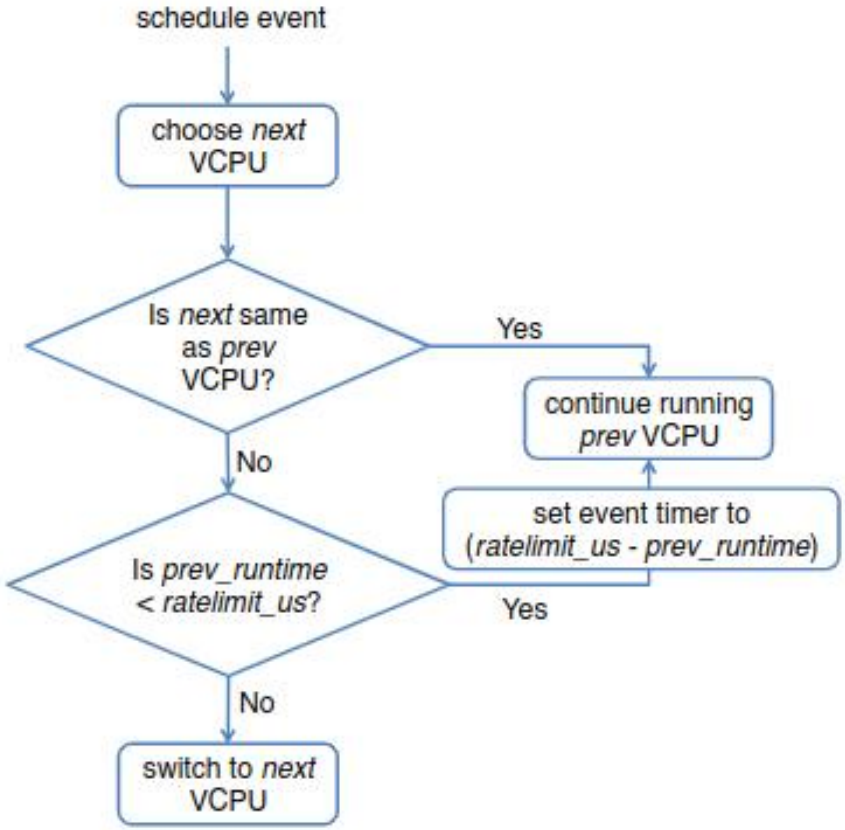


(b) Periodic time-shared cache cleansing.

跨虚拟机侧信道攻击

缓解方案4：事件调度机制修改（Hypervisor中的防御措施）

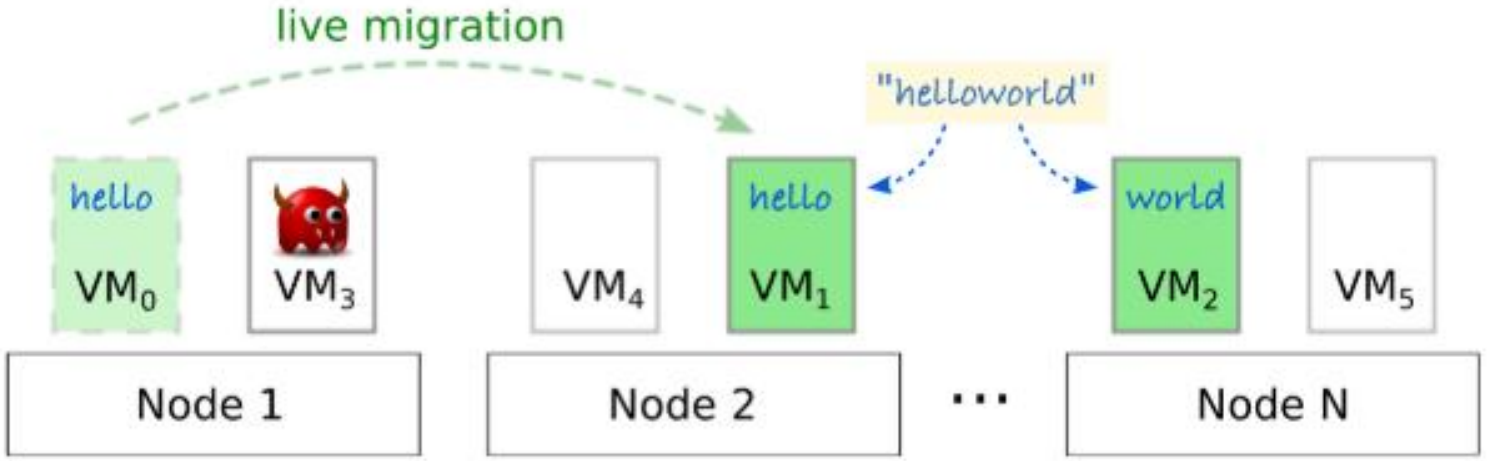
- 攻击者可以观察到受害者行为的频率越低，信息泄漏的风险就越小
- 优点是可以与其他缓解方案结合；缺点是额外的时间调度开销



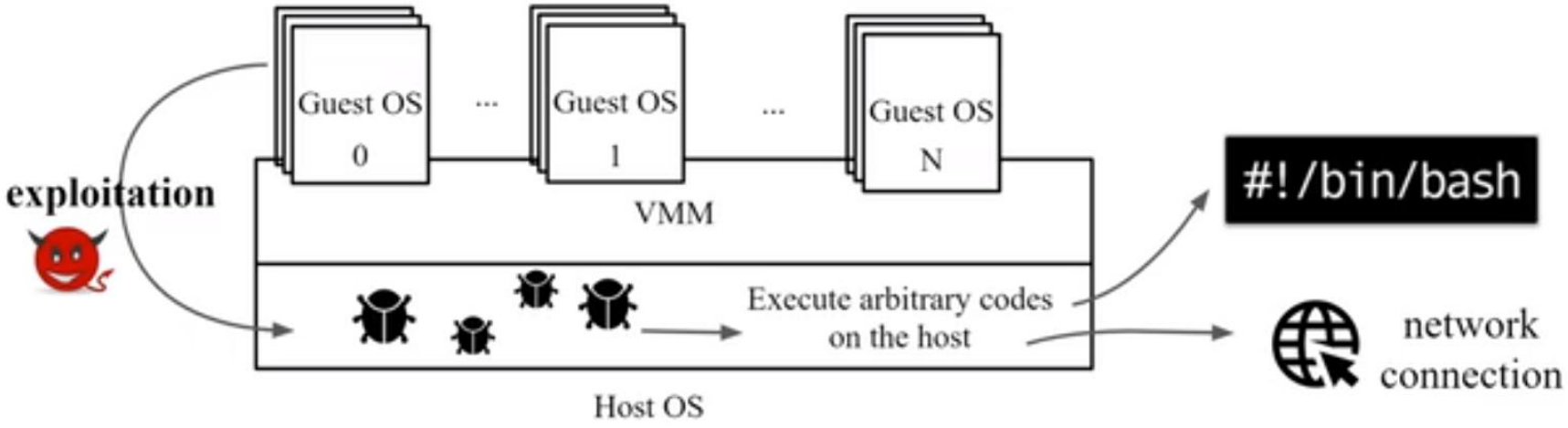
跨虚拟机侧信道攻击

缓解方案5：动态移动虚拟机（Hypervisor中的防御措施）

- 一种简单粗暴的解决思路，让所有的虚拟机在不同的物理节点上定时迁移。
- 优点是简单有效；缺点是迁移的额外开销并不总是可以接受的

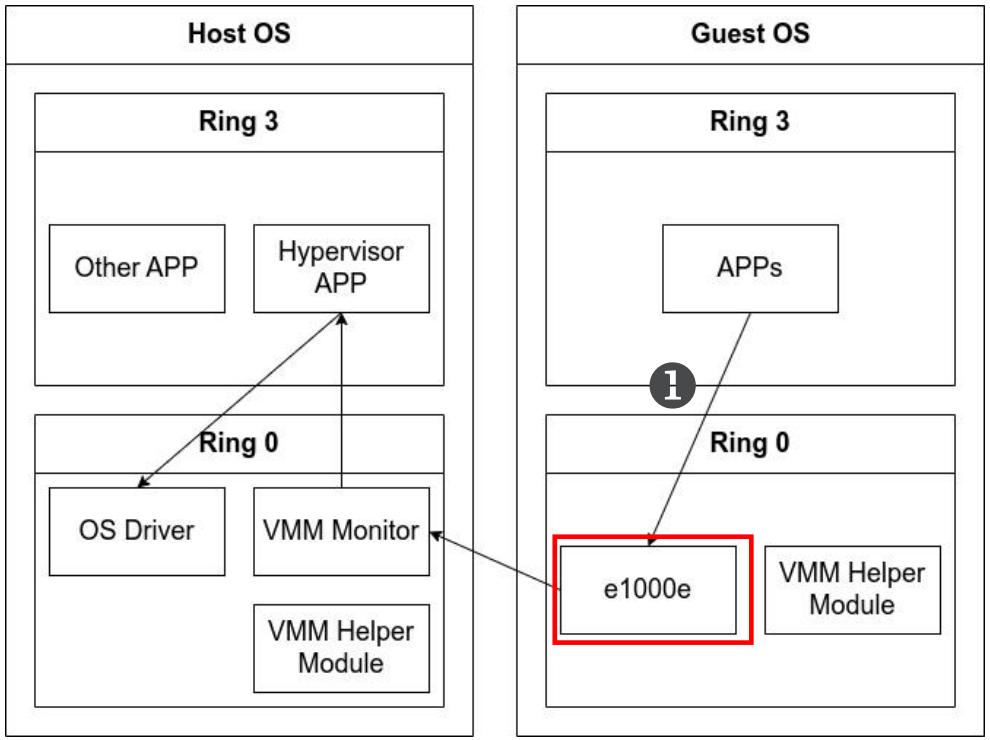


- 虚拟机逃逸
  - 原因：任何在Hypervisor中的漏洞都可能被攻击者利用，实现从虚拟机到宿主系统的逃逸。
  - 结果：攻击者可以在虚拟机上运行代码，突破Guest OS与Hypervisor，从而能够访问主机操作系统以及在该主机上运行的每个虚拟机。



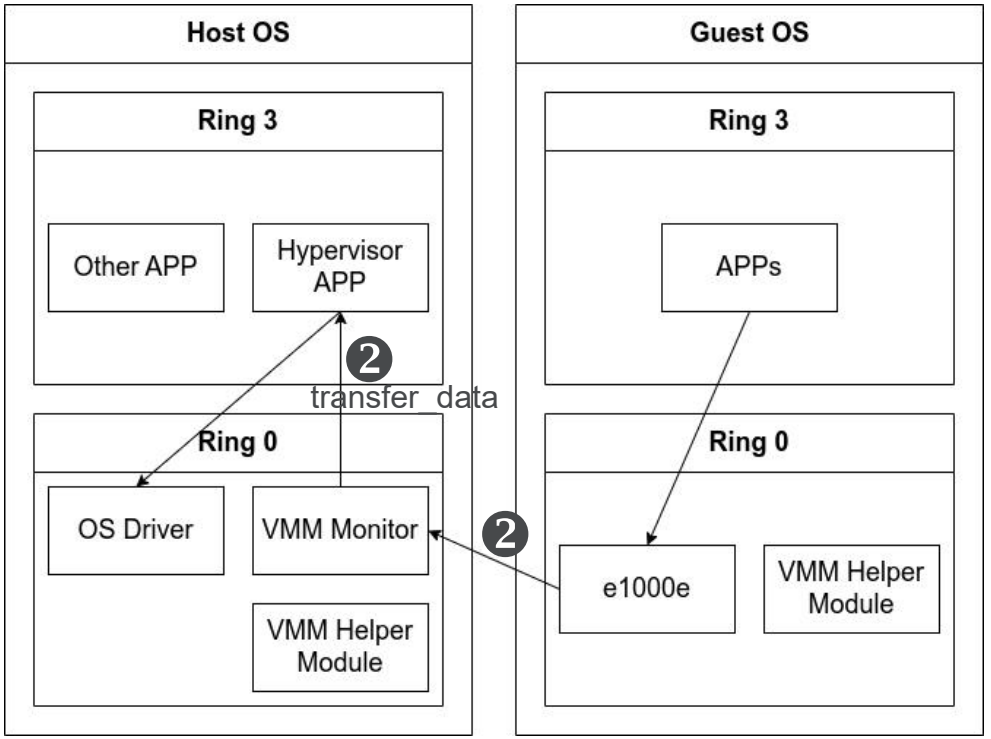
虚拟机逃逸流程

正常流程❶：应用使用e1000e网卡发送数据，该网卡是Guest OS的一个设备。




```
union{
        struct{
                uint64_t buf_addr;
                uint64_t size;
        }transfer_data;
        struct{
                uint8_t ipcss;          //IP checsum start
                uint8_t ipcso;          //IP checsum offset
                uint16_t ipcse;         //IP checsum end
                uint8_t tucss;          //TCP checsum start
                uint8_t tucso;          //TCP checsum offset
                uint16_t tucse;         //TCP checsum end
                uint32_t cmd_and_length;
                uint8_t status;         //Descriptor status
                uint8_t hdr_len;        //Header length
                uint16_t mss;           //Maximum segment
size
        }prop_desc;
}tranfer;
```

- 虚拟机逃逸流程
  - 正常流程❷：VMM Monitor监控到网卡被使用，将e1000e的数据结构体转送至hypervisor app中进行处理。



packet transfer

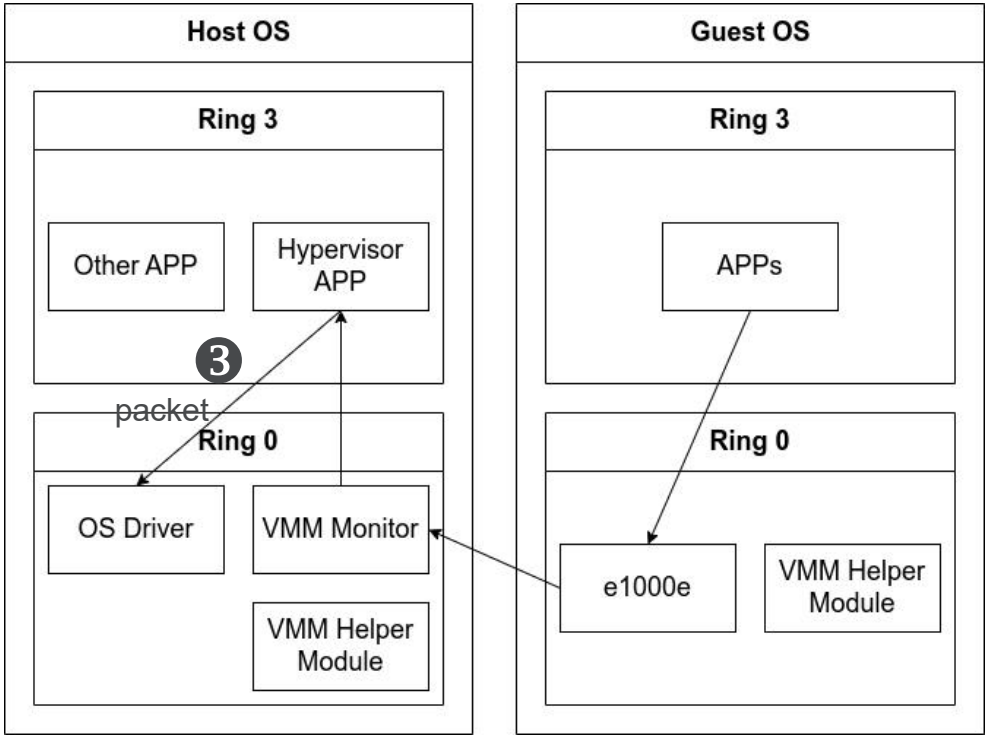
```
mem = registers[TDBAH]<<32|registers[TDBAL];
mem = mem + registers[TDH]*sizeof(transfer);
ReadGuestMem(mem,&transfer);
//Handle transfer struct
...
...
registers[TDH]++;
if(registers[TDH]==registers[TDT])
      return;
else
      loop;
```

```
if(transfer.length & E1000_TXD_CMD_DEXT)
      e1000_process_TXD_CMD_DEXT(...);
else
      //init e1000e property
      prop = &e1000e->prop;
      prop->ipcss = transfer.prop_desc.ipcss;
      ...
```

虚拟机逃逸流程

正常流程❸：

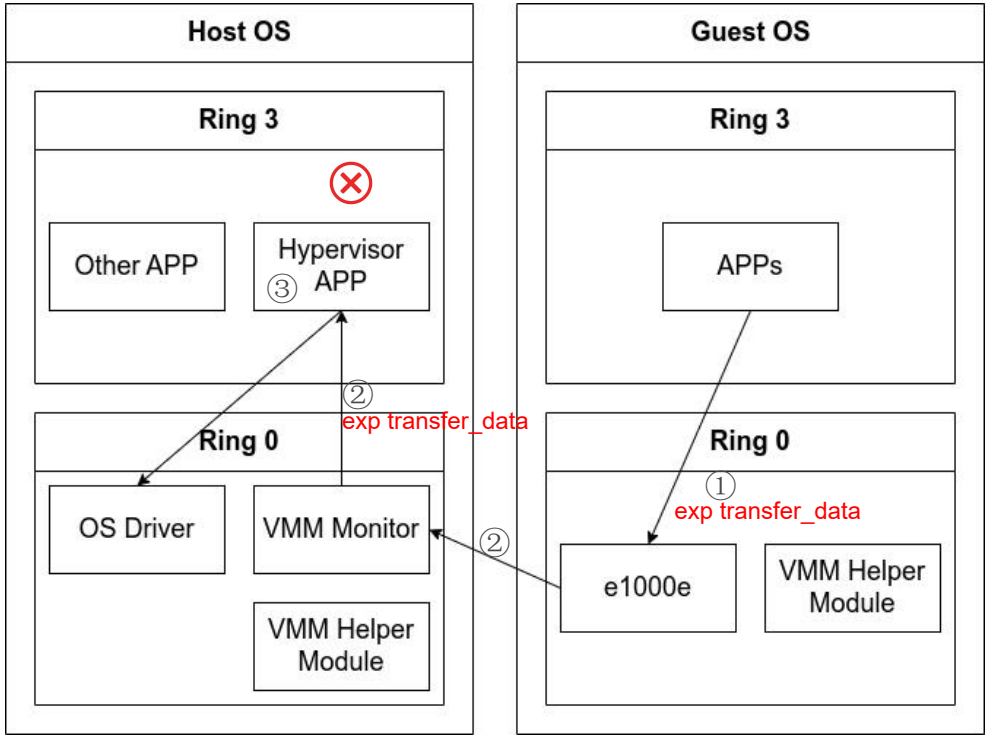
Hypervisor app向物理网卡发送包，完成网络传输。



虚拟机逃逸流程

示例：CVE-2019-5541

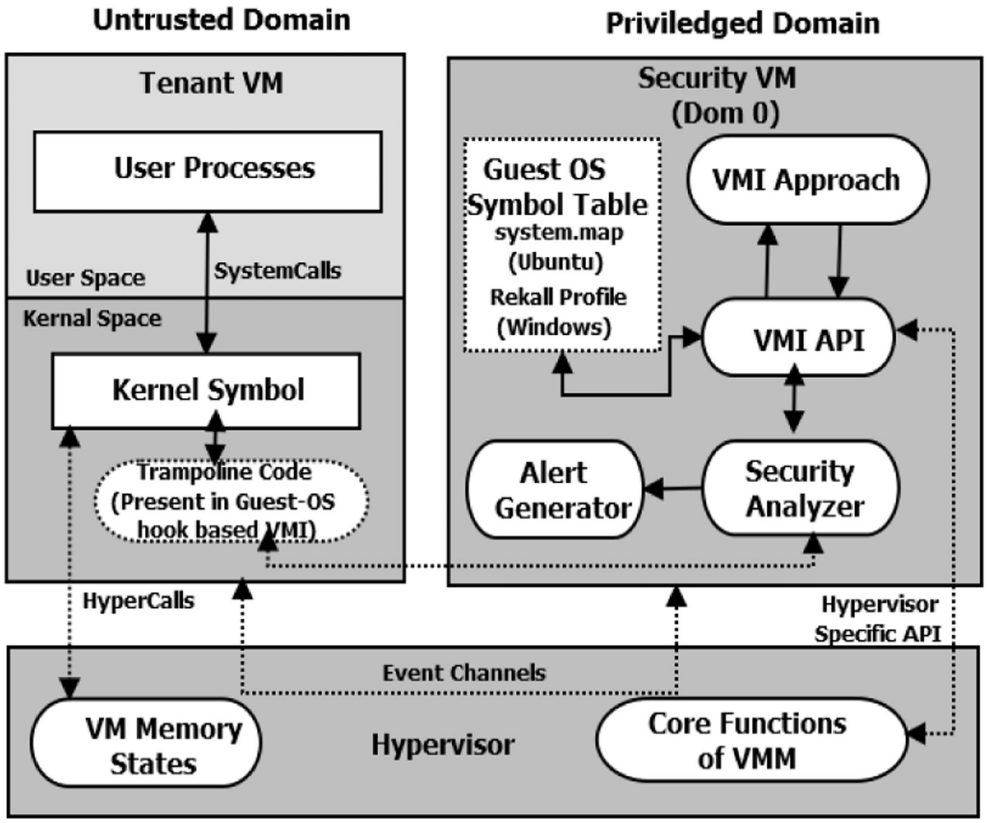
- 步骤1：Guest OS向网卡传输包含exp的ipcss的IPv6 Large Segmentation Offload packet
- 步骤2：VMM Monitor转送包
- 步骤3：Hypervisor解包，发生堆溢出，控制流被劫持



虚拟机逃逸缓解方案

虚拟机自检（VMI）技术

- 使用Guest OS中的符号表来访问虚拟机内存区域。
- VM0如果发现任何虚拟机内存区域可疑，安全分析器会向云管理员发出警报。



云数据安全

- **云存储作为云计算提供的核心服务，是不同终端设备间共享存储资源的一种解决方案，其中的数据安全已成为云安全的关键挑战之一。**
  - **访问控制技术（解决数据的机密性）**
    - 管理资源的授权访问范围
  - **数据分享算法（解决数据拥有者与使用者之间的身份差异）**
    - 代理重加密算法
    - 属性加密算法
  - **可搜索加密技术（决绝对密文数据的检索）**
    - 对称可搜索加密技术
    - 非对称可搜索加密技术
  - **可回收性与所有权证明**
    - 验证云端数据的完整性

- 访问控制技术
  - 基于角色的访问控制（RBAC）
    - 用户被分配一个或多个角色，每个角色具有特定的权限。
    - 优点：通过为用户分配角色并将权限与角色挂钩，简化权限管理，确保只有授权用户才能访问特定资源，减少了权限滥用的风险。
    - 缺点：角色定义和管理复杂，一旦角色过多，管理难度增大。
  - 基于属性的访问控制（ABAC）
    - 根据用户属性（如部门、职位、地点）和环境条件（如时间、设备类型）动态确定的。ABAC 提供了比 RBAC 更灵活的控制。
    - 优点：通过用户、资源、环境等多种属性来定义访问策略，实现了更细粒度的控制，适应动态和复杂的云环境需求。
    - 缺点：需要管理和维护大量的属性和策略，ABAC的实施和管理比较复杂，评估多种属性组合可能导致性能开销，特别是在大规模环境中。

- 访问控制技术
  - 云计算访问控制的需求
    - 动态性：云计算环境中的资源和用户需求动态变化，要求访问控制策略能够动态调整。
    - 多租户环境：多个用户共享同一个物理资源，访问控制策略必须确保彼此之间的隔离和安全。
    - 细粒度控制：需要对资源进行细粒度的权限控制，以防止数据泄露和未授权访问。
  - 基于任务-角色的访问控制（TRBAC）
    - 任务和角色关联：访问权限不是直接分配给用户，而是通过任务来分配。每个任务对应一组执行该任务所需的角色。
    - 动态任务分配：任务可以根据具体条件（如时间、地点、环境等）动态分配给用户。
    - 支持多租户环境：该模型旨在确保即使在潜在不可信的租户之间，也能安全共享资源。
    - 细粒度权限管理：TRBAC模型支持对同一云用户授予不同的访问权限，使其能够安全地使用多种服务。

基于任务-角色的访问控制（TRBAC）

- 用户（User）：起点，代表任何需要访问云计算资源的个体或实体。
- 角色分配（Role Assignment, RA）：用户根据其凭证或职位被分配特定角色。
- 会话（Sessions）：管理用户和角色之间的授权关系。
- 角色（Roles）：会话中包含多个角色，从角色A到角色n，表明会话可以涵盖多个角色，具体取决于用户的角色分配。

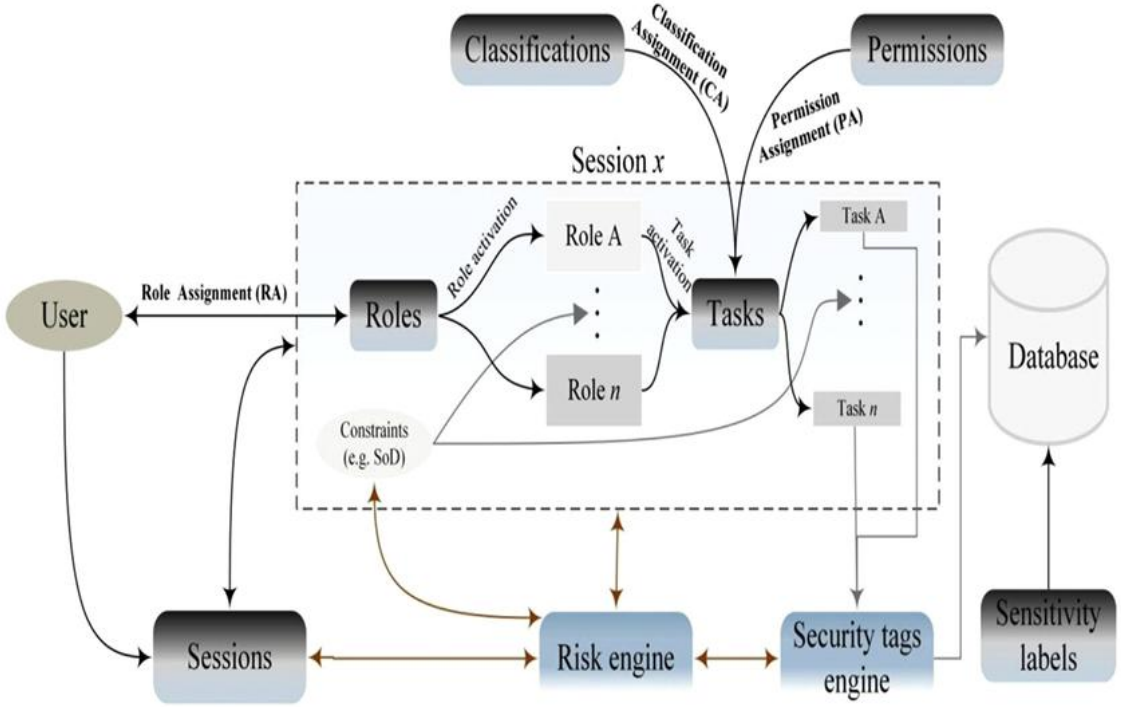


Fig. 2 – Access control for cloud computing (AC3) (level 2&3).

基于任务-角色的访问控制（TRBAC）

- 约束（Constraints）：例如"SoD"代表"Separation of Duties"（职责分离），是用于确保没有单一角色或用户拥有过多权限或产生冲突责任的约束。
- 任务（Tasks）：可以执行的具体工作相关联。从任务A到任务n，表明可以为角色分配的多个任务。权限赋予任务，而不是直接给角色。
- 权限（Permissions）：每个任务都附带特定权限，表示任务能够执行的操作，例如读取、写入或修改数据库。
- 数据库（Database）：表示任务可能会与之交互的数据存储或数据库。

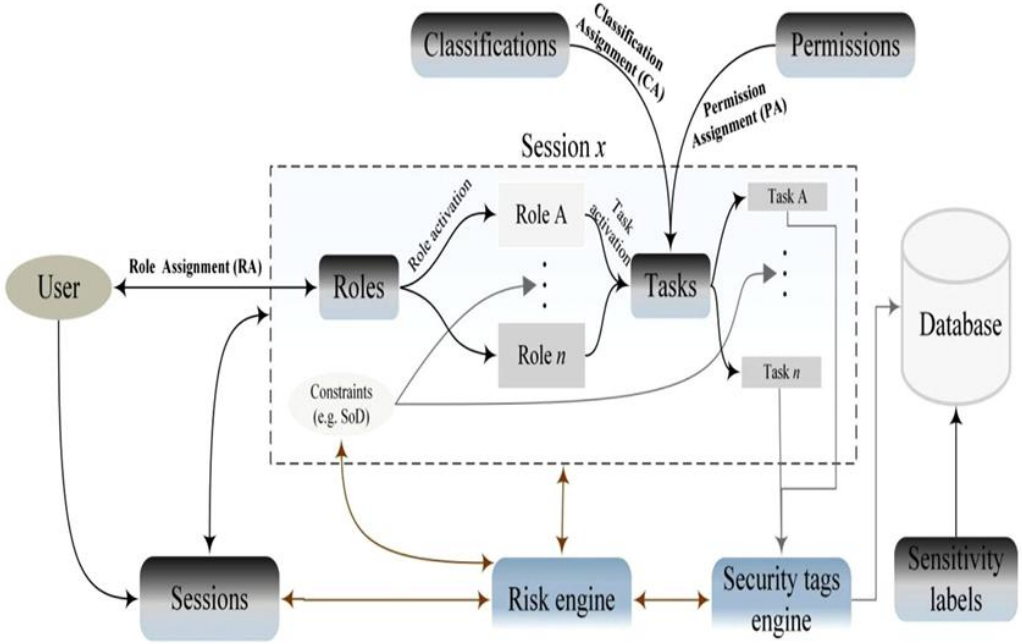


Fig. 2 – Access control for cloud computing (AC3) (level 2&3).

基于任务-角色的访问控制（TRBAC）

- 敏感性标签（Sensitivity Labels）：附加在数据库上，这些标签根据数据的敏感性限制对数据的访问。
- 风险引擎（Risk Engine）：该组件分析会话中的潜在风险，评估例如用户行为、任务的重要性或数据敏感性等因素。
- 安全标签引擎（Security Tags Engine）：生成或管理用于控制和细化访问的安全标签，动态调整会话权限。
- 分类（Classifications）：根据安全级别、数据类型等对任务进行分类。

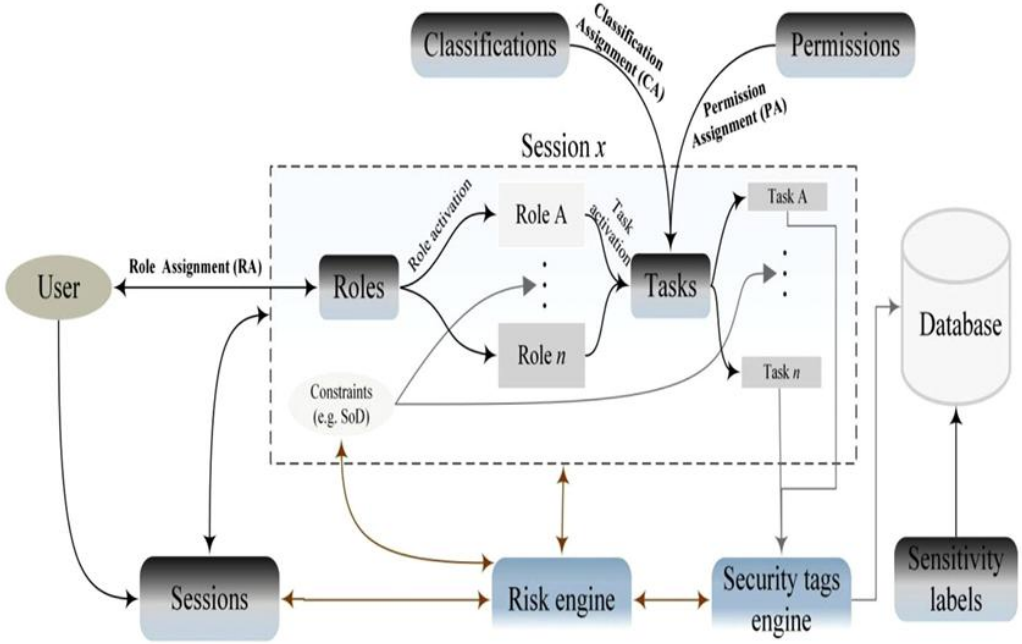


Fig. 2 – Access control for cloud computing (AC3) (level 2&3).

基于任务-角色的访问控制（TRBAC）

- 每个用户 u 都可以拥有一定数量的角色 {r1.rn}
- 每个角色都可以拥有一定数量的任务 {t1.tn}
- 每个任务 t 都会被分配完成其工作所需的确切权限 {p1.pn}，以及访问目标数据或资产的分类 cla。
- 任务分配后，用户执行任务所需的权限被激活，系统在用户执行操作时，动态检查权限，确保任务仅在授权范围内执行。
- 用户完成任务后，与任务相关的临时权限被收回，用户仅保留基础角色的权限，确保最小权限原则。

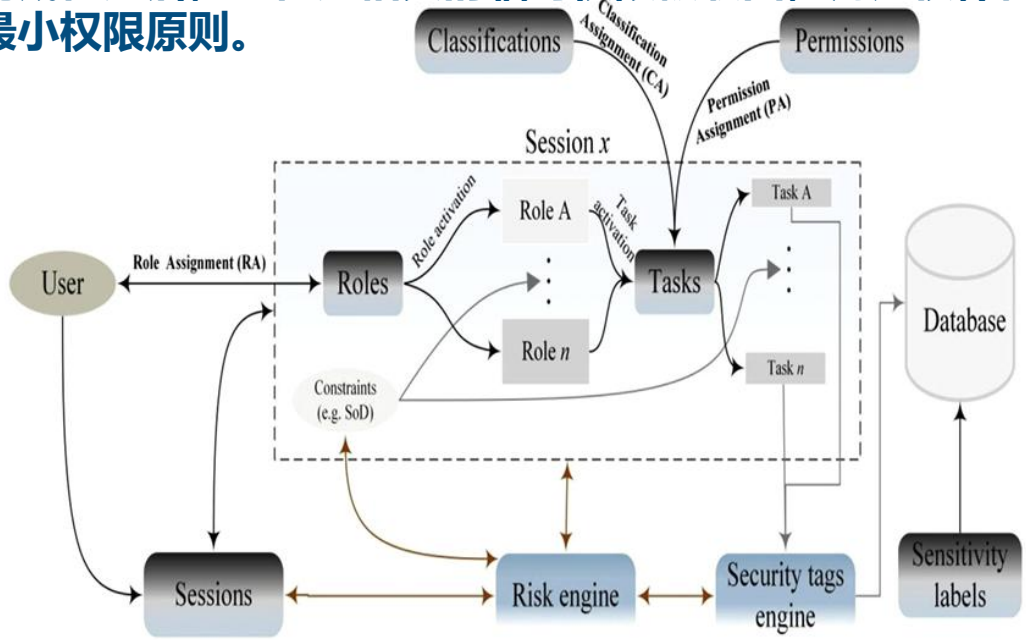
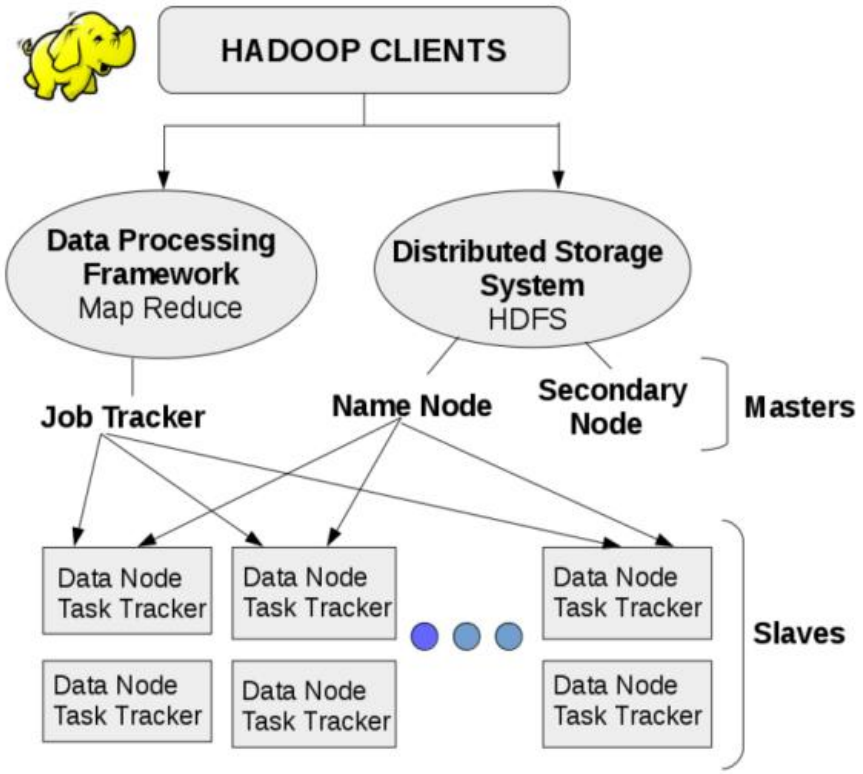


Fig. 2 – Access control for cloud computing (AC3) (level 2&3).

内容概要

- 云计算平台
  - 云计算体系结构
  - 云计算安全风险
- 大数据平台
  - Hadoop体系结构
  - Hadoop安全风险
- 总结

- Hadoop概述
  - Apache Hadoop是一个基于大数据存储分析的开源分布式系统，是一个完整的平台生态系统。用户可在不了解分布式结构细节的情况下，开发分布式应用程序，充分利用计算机集群（Cluster）的威力进行数据的高速运算和存储。
  - 基于Java语言开发，具有很好的跨平台特性。
  - 核心是分布式文件系统HDFS (Hadoop Distributed File System)和MapReduce。 HDFS可实现**大数据的存储**，MapReduce可实现**大数据复杂度计算**。

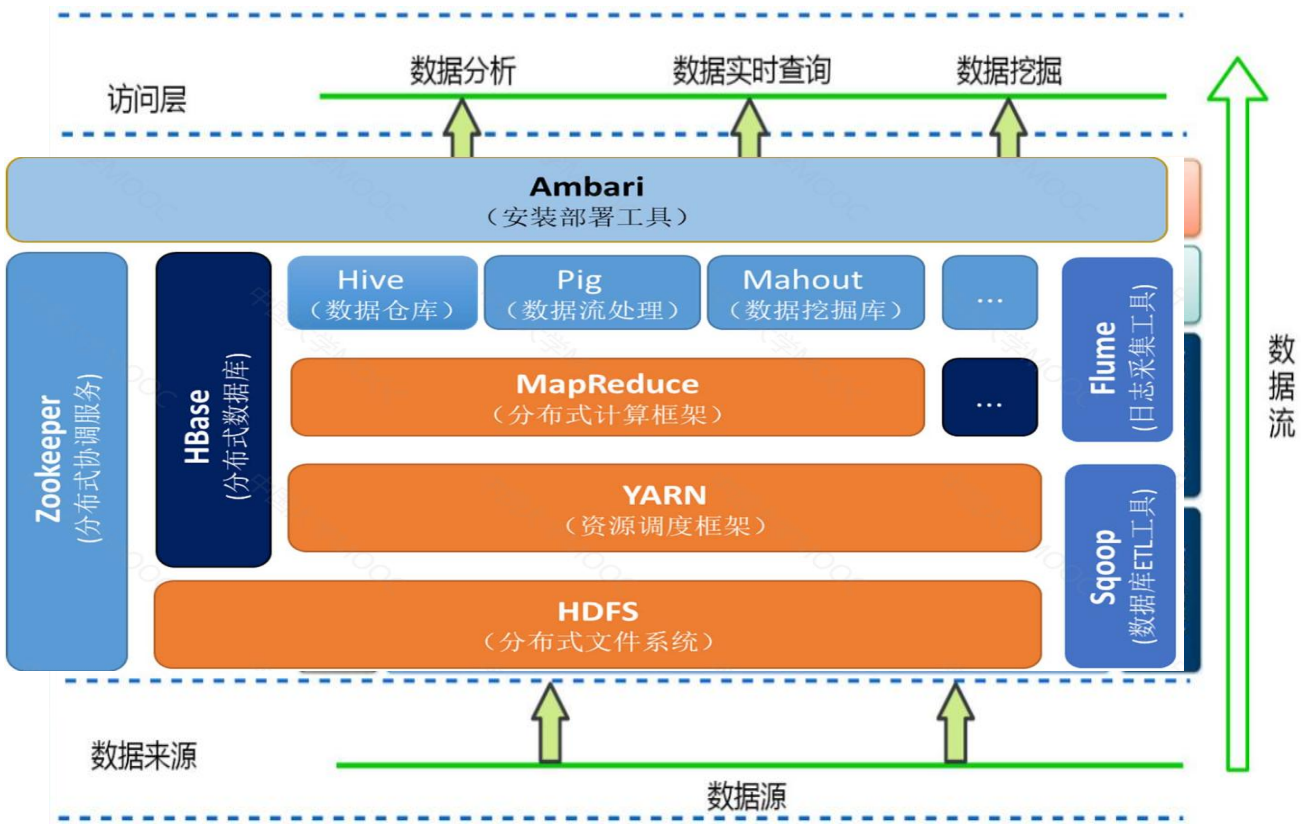


HADOOP应用系统由数据来源，大数据层和数据访问层构成。

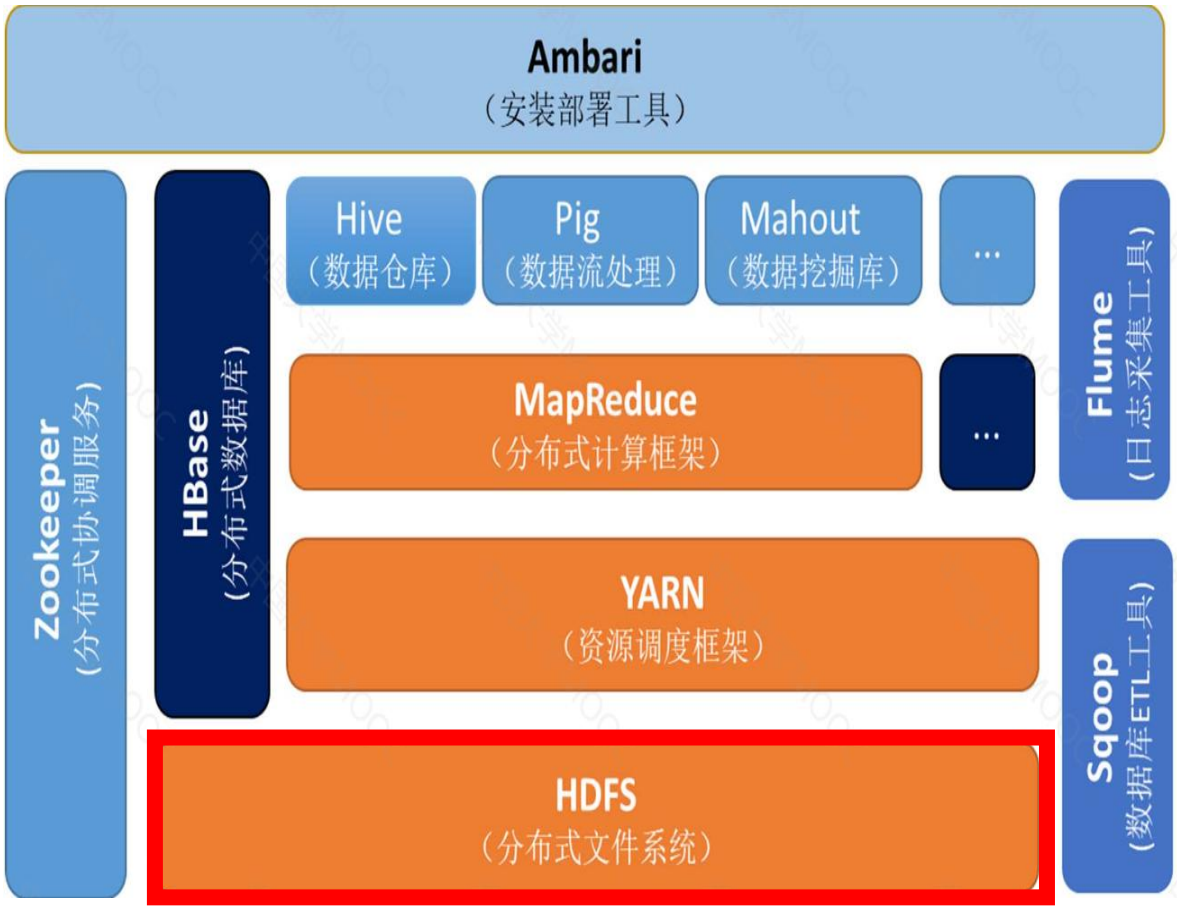
负责提供数据的最终分析、呈现和应用

大数据层包括各种数据处理工具和方法

负责数据采集



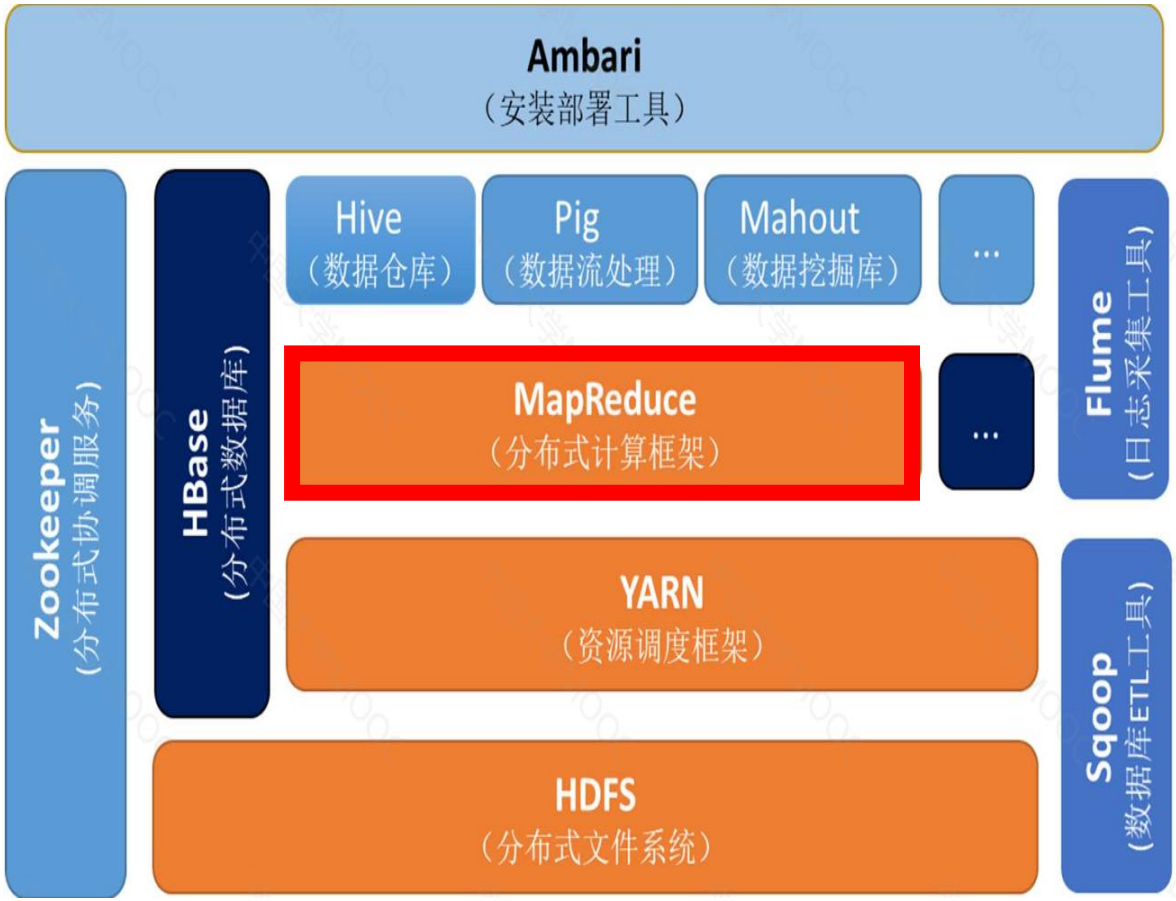
Hadoop体系结构



HDFS（Hadoop Distributed File System）：

核心组件，冗余存储系统，将文件分布式存储在很多服务器上，并高效地将数据以流式传输到用户应用程序。

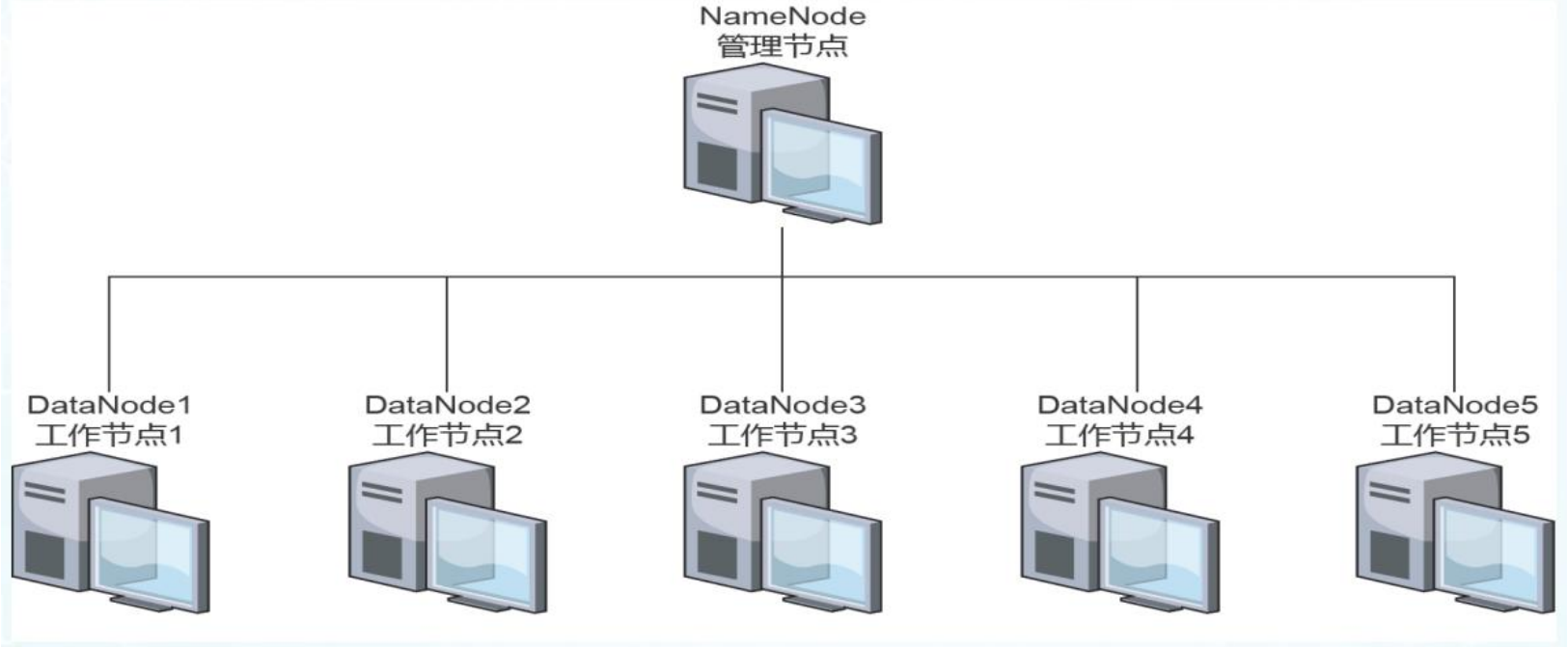
Hadoop体系结构



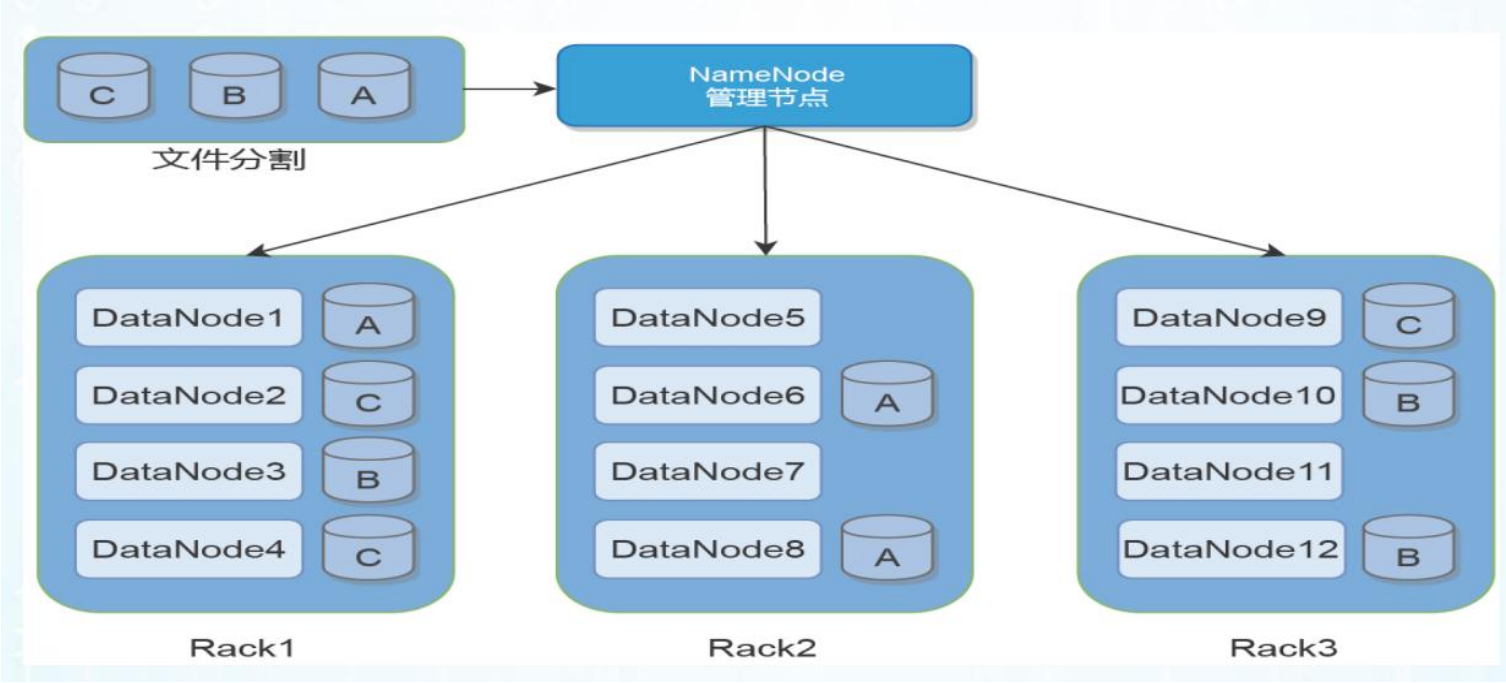
MapReduce：

分布式计算框架，负责分布式并行计算，可以将用户编写的业务逻辑代码和自带默认组件整合成一个完整的分布式运算程序，并发运行在一个Hadoop集群上。

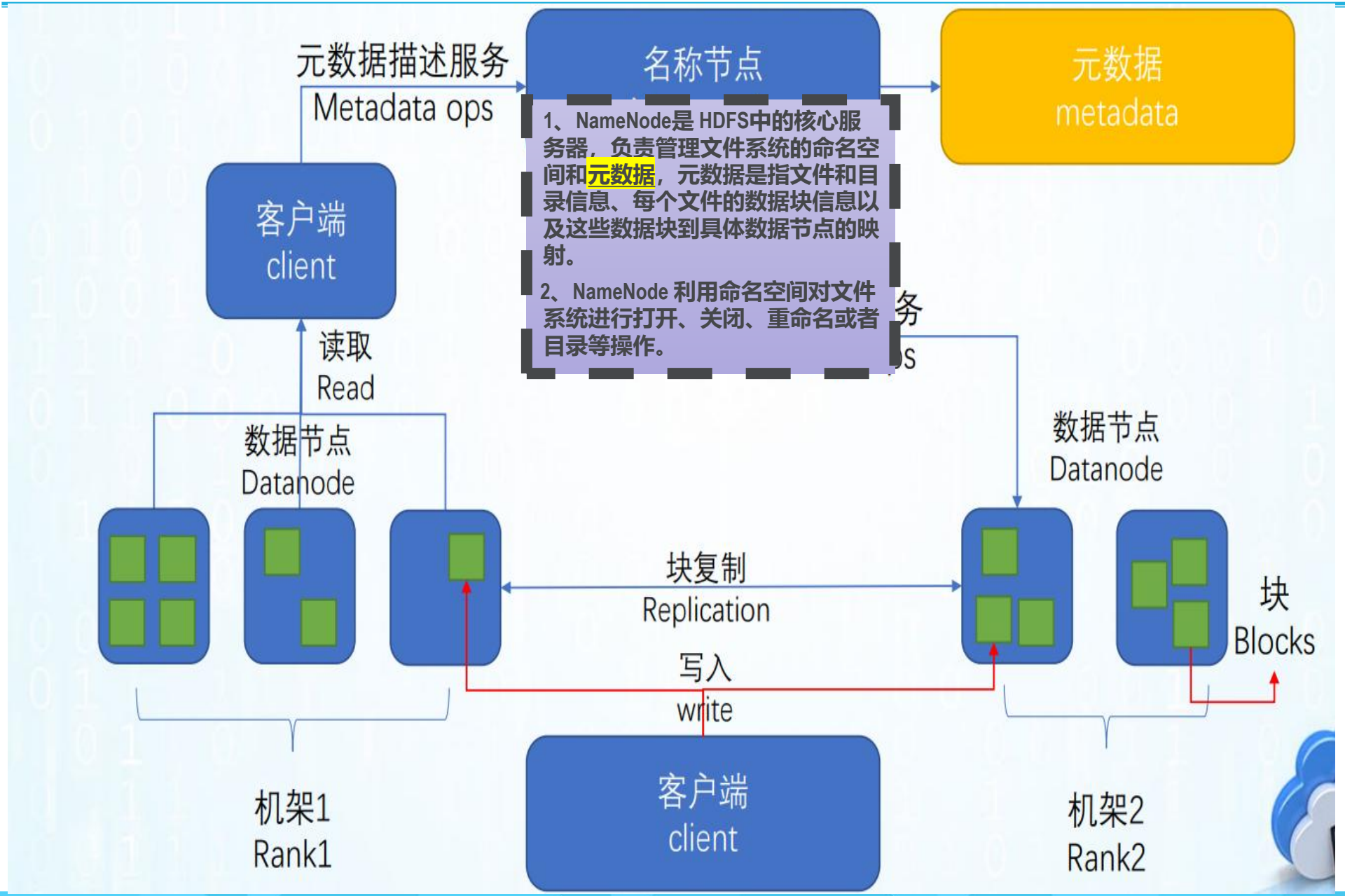
- HDFS （Hadoop Distributed File System）
  - 采用主从(Master/Slave)架构
  - HDFS集群由以下元素组成：
    - 一个名称节点（ NameNode ）
    - 若干数据节点（ DataNode）



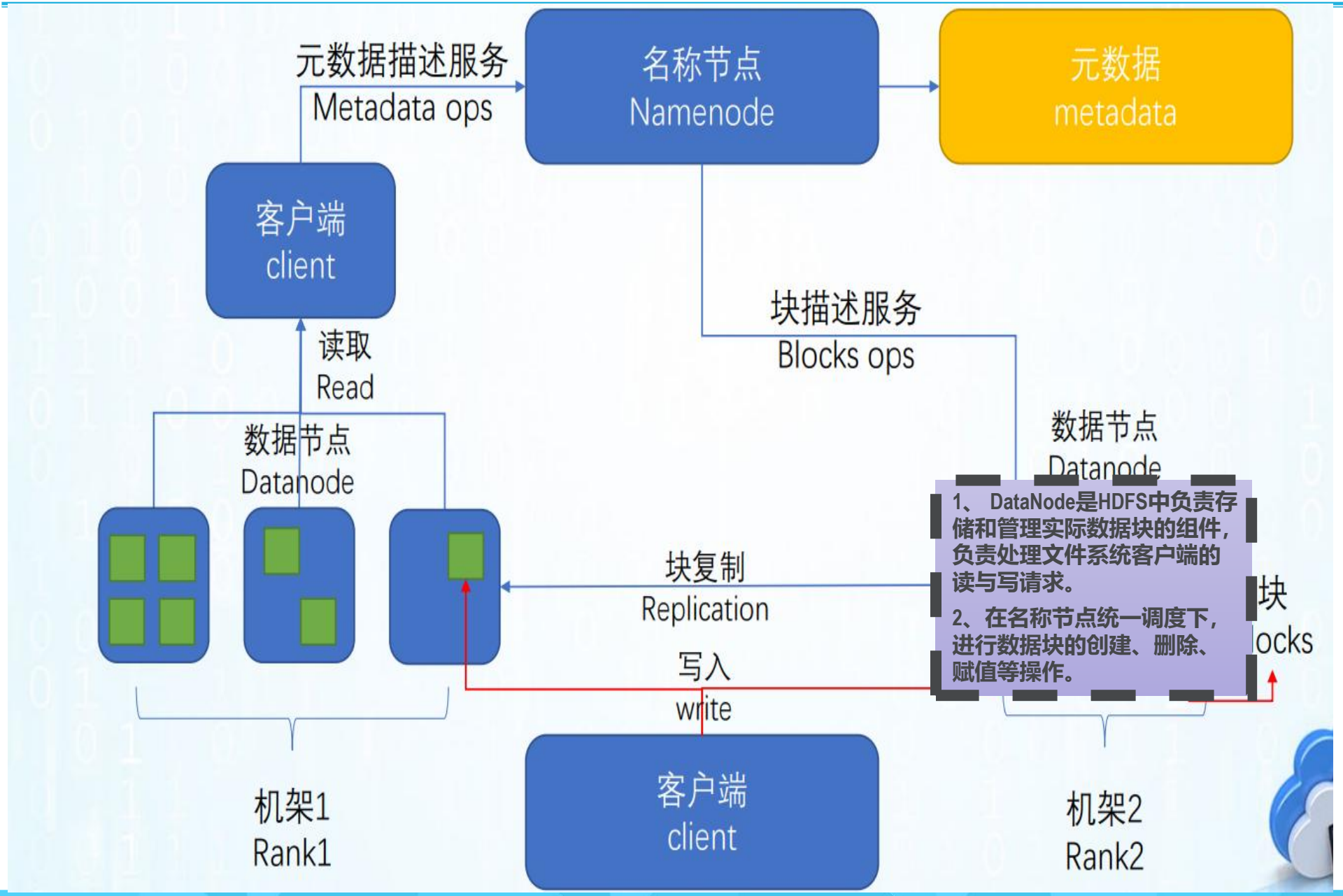
- HDFS中，一个文件通常被分割成若干个块（Block），存储在一组DataNode上，同时建立Block和DataNode之间的映射，如下图所示。



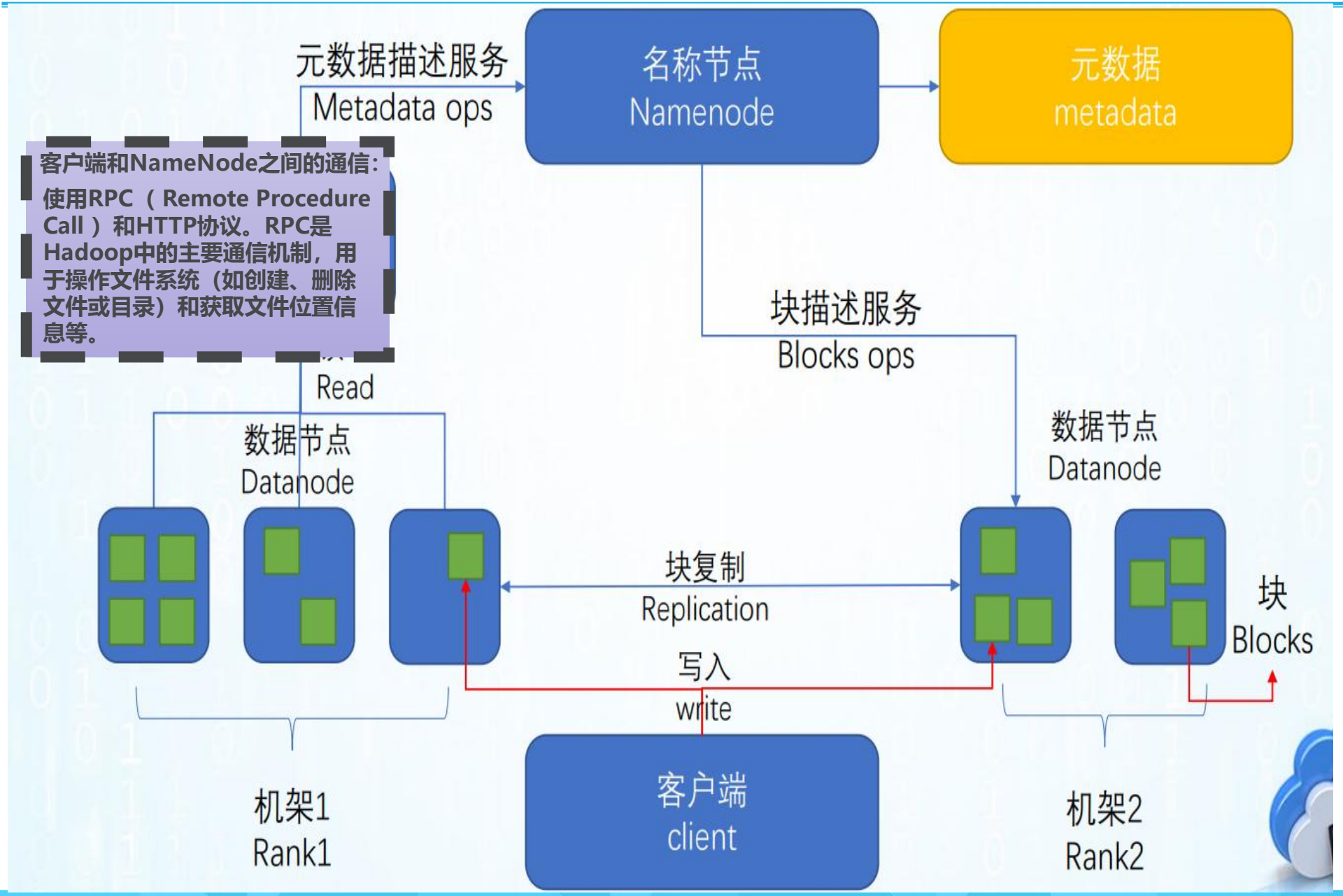
Hadoop体系结构


元数据描述服务
Metadata ops
名称节点
元数据
metadata
1、NameNode是 HDFS中的核心服务器，负责管理文件系统的命名空间和元数据，元数据是指文件和目录信息、每个文件的数据块信息以及这些数据块到具体数据节点的映射。
2、NameNode 利用命名空间对文件系统进行打开、关闭、重命名或者目录等操作。
客户端
client
读取
Read
数据节点
Datanode
数据节点
Datanode
块复制
Replication
块
Blocks
写入
write
客户端
client
机架1
Rank1
机架2
Rank2

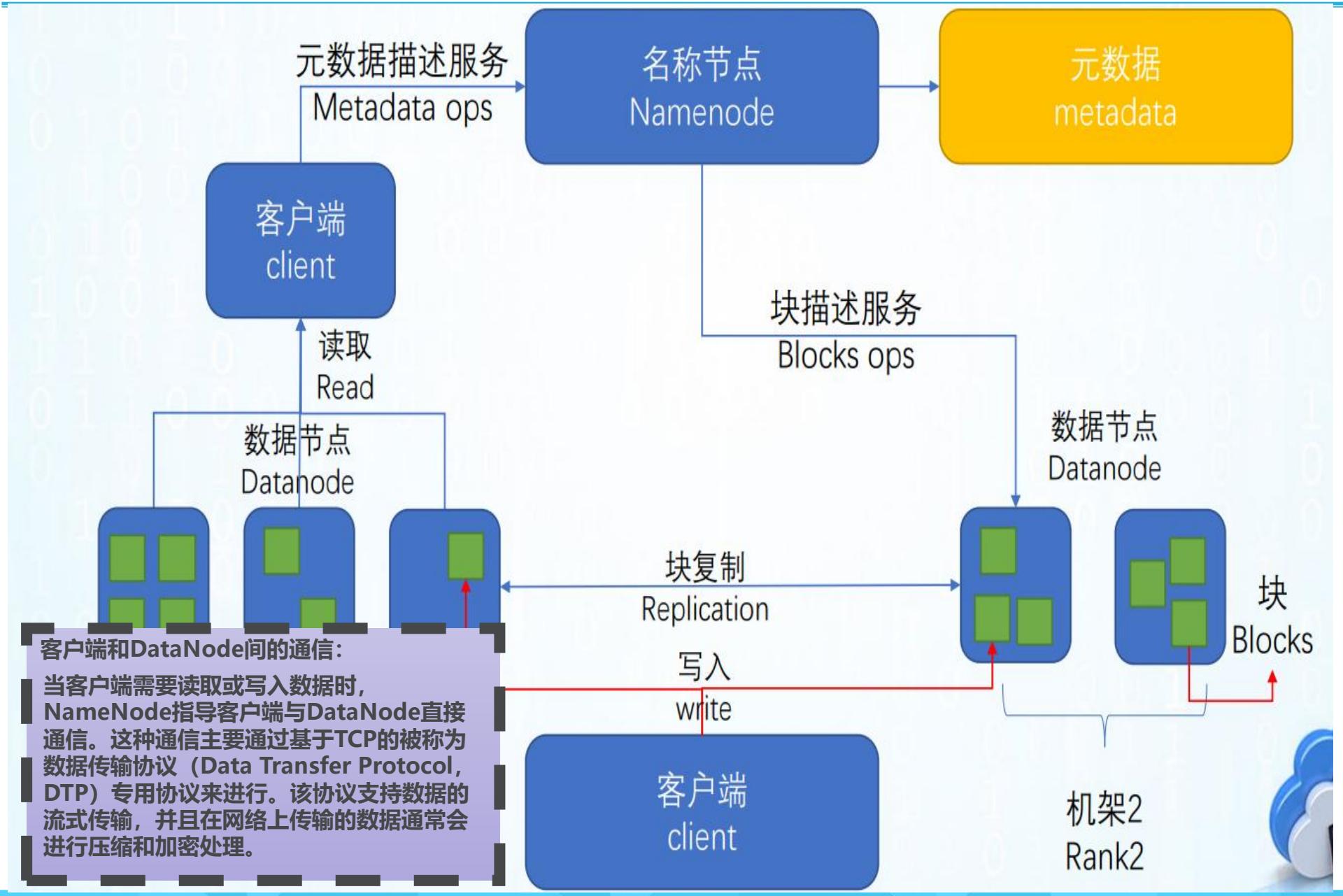
Hadoop体系结构


元数据描述服务
Metadata ops
名称节点
Namenode
元数据
metadata
客户端
client
读取
Read
块描述服务
Blocks ops
数据节点
Datanode
数据节点
Datanode
1、 DataNode是HDFS中负责存储和管理实际数据块的组件，负责处理文件系统客户端的读与写请求。
2、在名称节点统一调度下，进行数据块的创建、删除、赋值等操作。
块
ocks
块复制
Replication
写入
write
客户端
client
机架1
Rank1
机架2
Rank2

Hadoop体系结构


元数据描述服务
Metadata ops
名称节点
Namenode
元数据
metadata
客户端和NameNode之间的通信：
使用RPC（Remote Procedure Call）和HTTP协议。RPC是Hadoop中的主要通信机制，用于操作文件系统（如创建、删除文件或目录）和获取文件位置信息等。
Read
块描述服务
Blocks ops
数据节点
Datanode
数据节点
Datanode
块复制
Replication
块
Blocks
写入
write
客户端
client
机架1
Rank1
机架2
Rank2

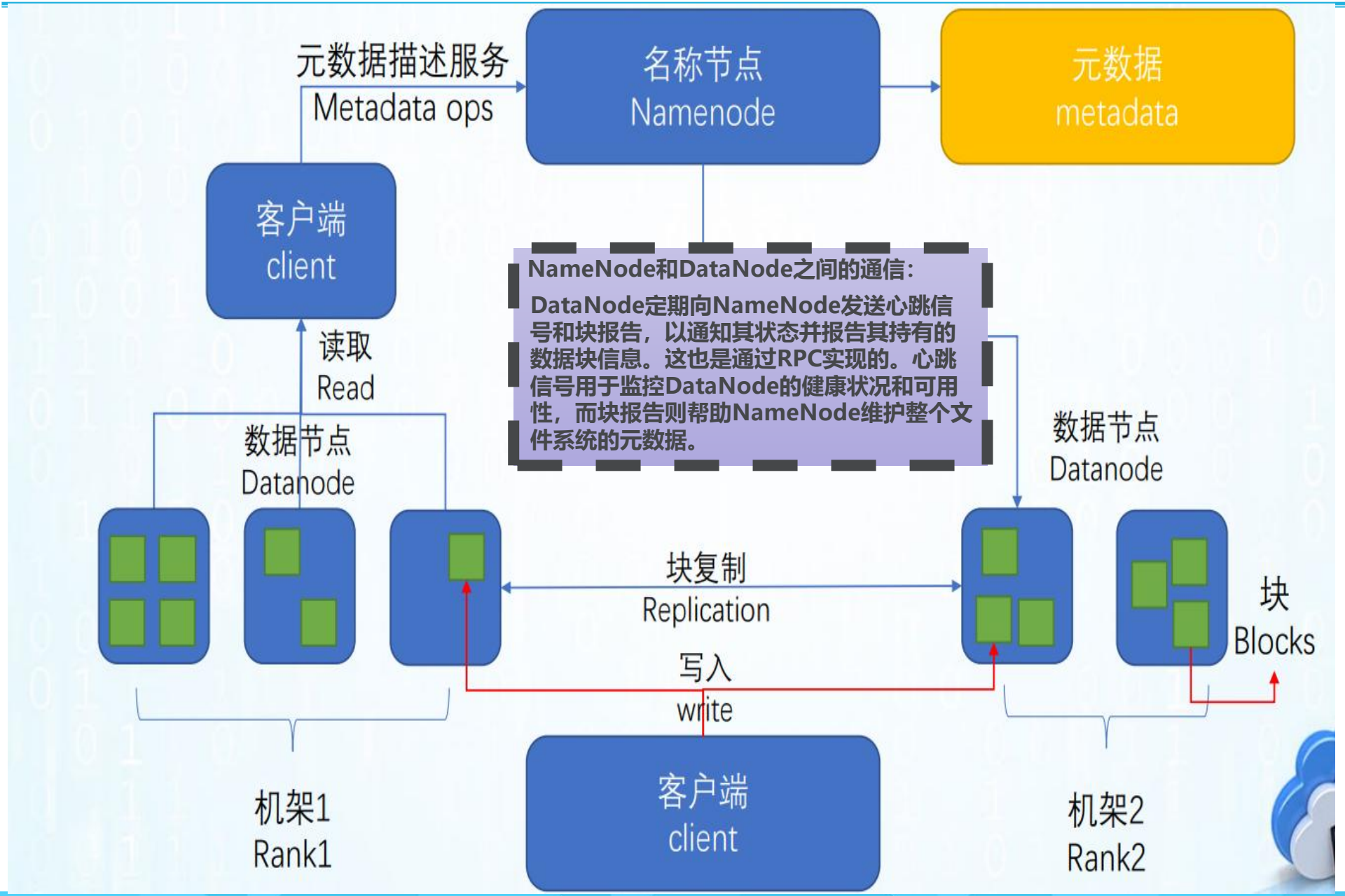
Hadoop体系结构



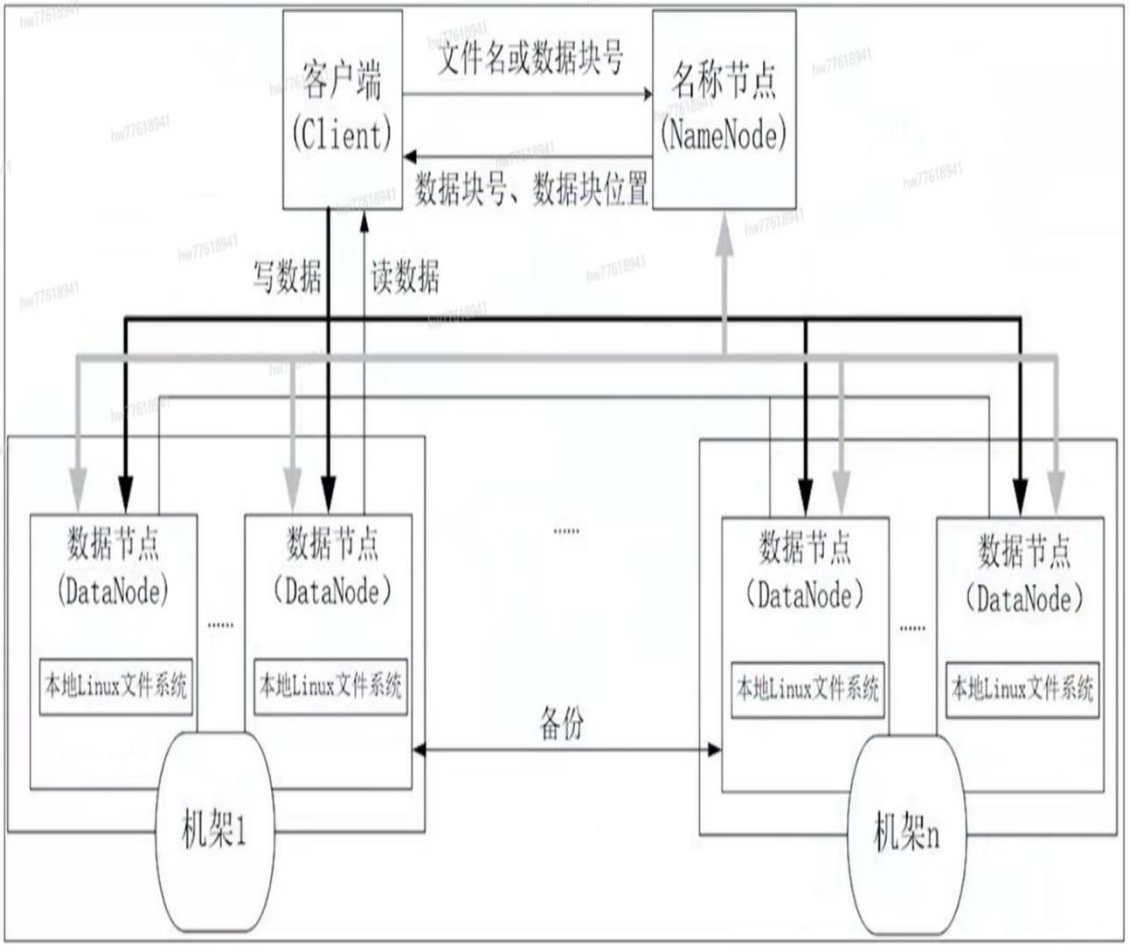
客户端和DataNode间的通信：

当客户端需要读取或写入数据时，NameNode指导客户端与DataNode直接通信。这种通信主要通过基于TCP的被称为数据传输协议（Data Transfer Protocol，DTP）专用协议来进行。该协议支持数据的流式传输，并且在网络上传输的数据通常会进行压缩和加密处理。

Hadoop体系结构


元数据描述服务
Metadata ops
名称节点
Namenode
元数据
metadata
客户端
client
NameNode和DataNode之间的通信：
DataNode定期向NameNode发送心跳信号和块报告，以通知其状态并报告其持有的数据块信息。这也是通过RPC实现的。心跳信号用于监控DataNode的健康状况和可用性，而块报告则帮助NameNode维护整个文件系统的元数据。
读取
Read
数据节点
Datanode
数据节点
Datanode
块复制
Replication
块
Blocks
写入
write
客户端
client
机架1
Rank1
机架2
Rank2

Hadoop体系结构



1、客户端从NameNode中获取文件的**元数据**，例如文件的权限和数据块的位置。
2、根据NameNode提供的块的位置信息，客户端请求从DataNode获取**实际数据**。如果某个DataNode无法提供数据，客户端会通过其他副本的DataNode获取数据，以确保数据的可用性。
3、客户端从DataNode获取数据块后，可以直接读取数据进行后续处理。
4、当客户端写入数据时，NameNode会告知可以存储文件的若干DataNode位置信息，用来存储文件的第一个数据块的副本，直到所有数据块被写入。

研究内容摘自hadoop官网。参考链接：
https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

- MapReduce架构
  - 包含Map函数和Reduce函数
  - Map函数将输入的键值对拆分为一个中间值键值对列表
  - Reduce函数将同一个中间键对应的中间值合并，输出一个键值对

Data type: Each record is (key, value)

任务分解

Map function:

$$(K_{in}, V_{in}) \rightarrow list(K_{inter}, V_{inter})$$

结果汇总

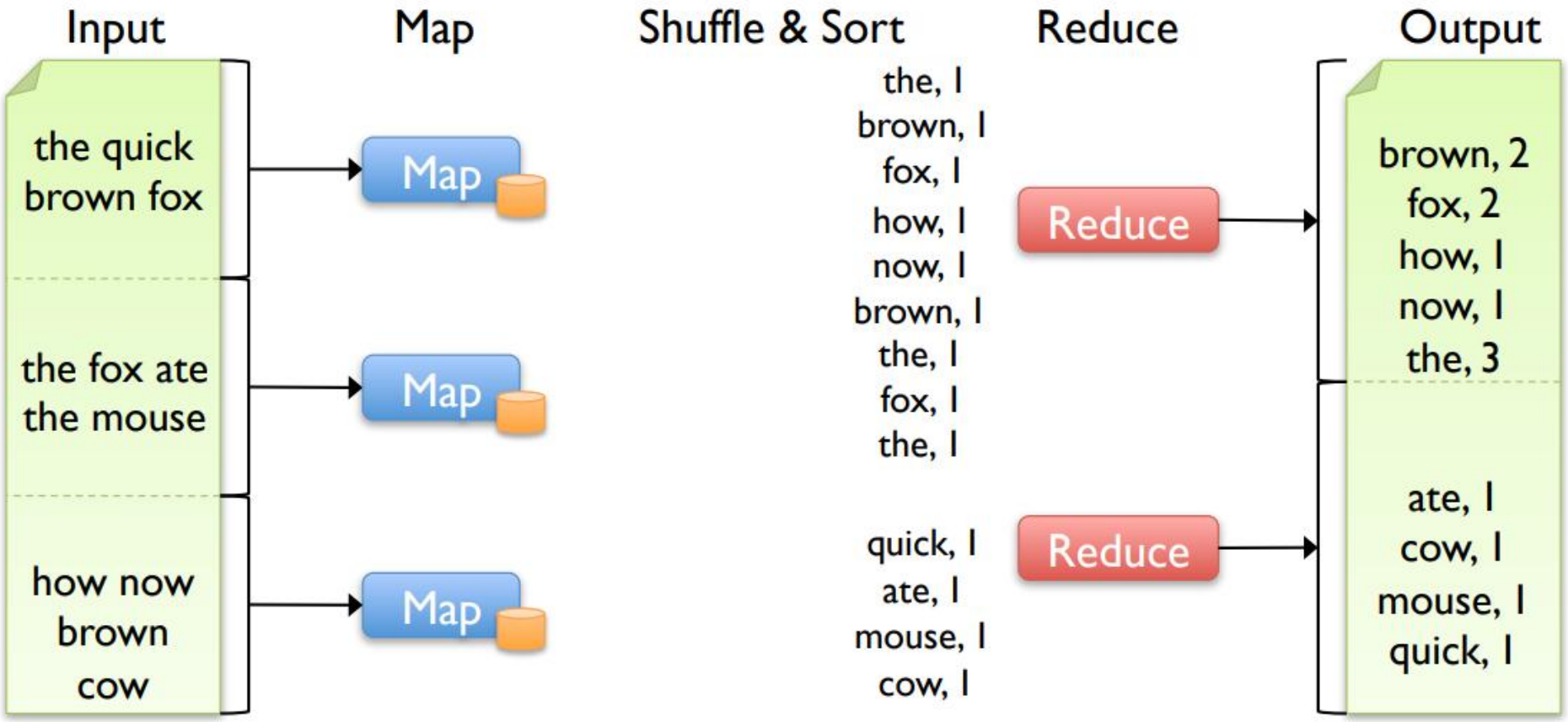
Reduce function:

$$(K_{inter}, list(V_{inter})) \rightarrow list(K_{out}, V_{out})$$

例子

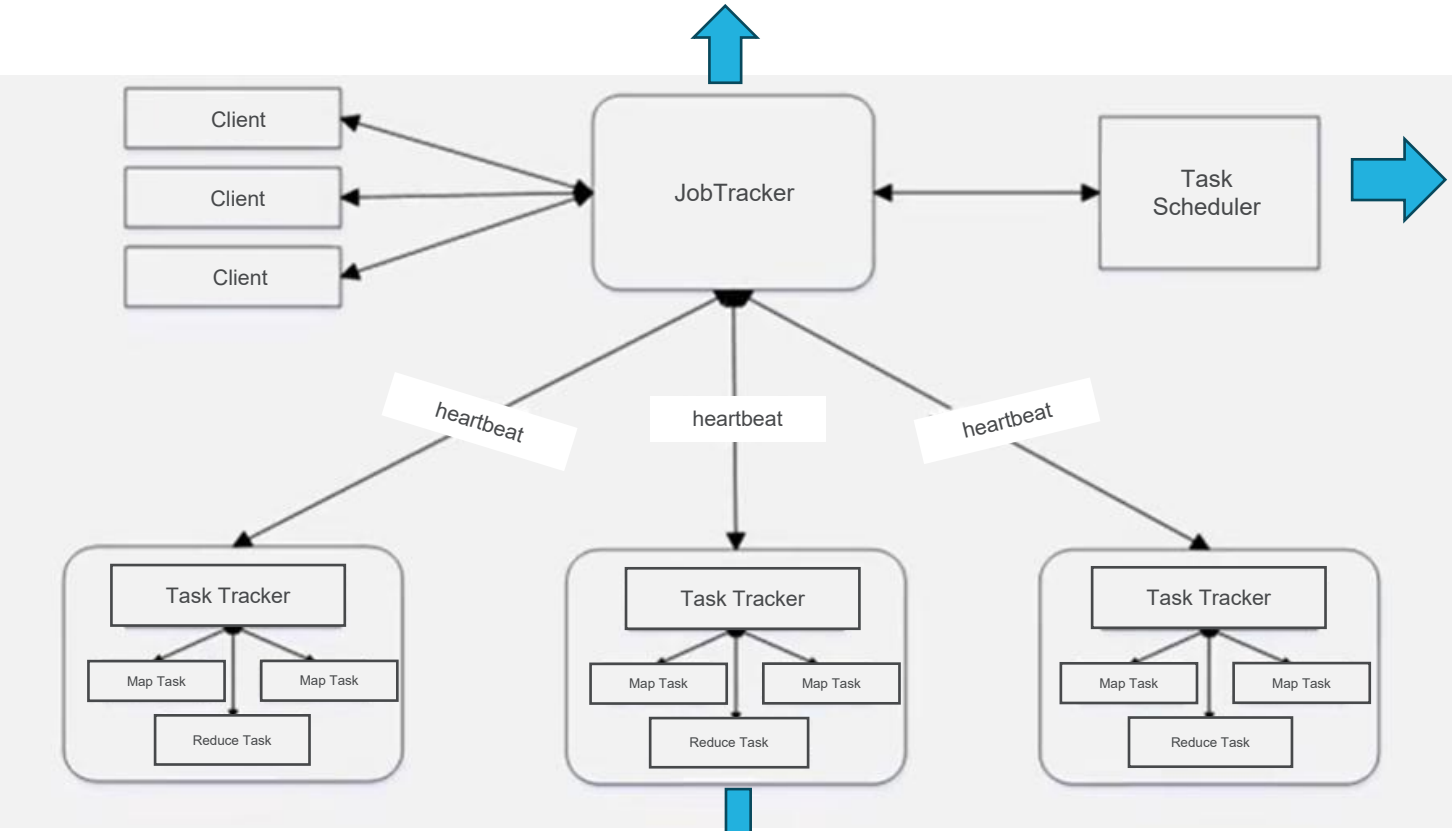
```
def mapper(line):
    for word in line.split():
        output(word, 1)

def reducer(key, values):
    output(key, sum(values))
```



Hadoop体系结构

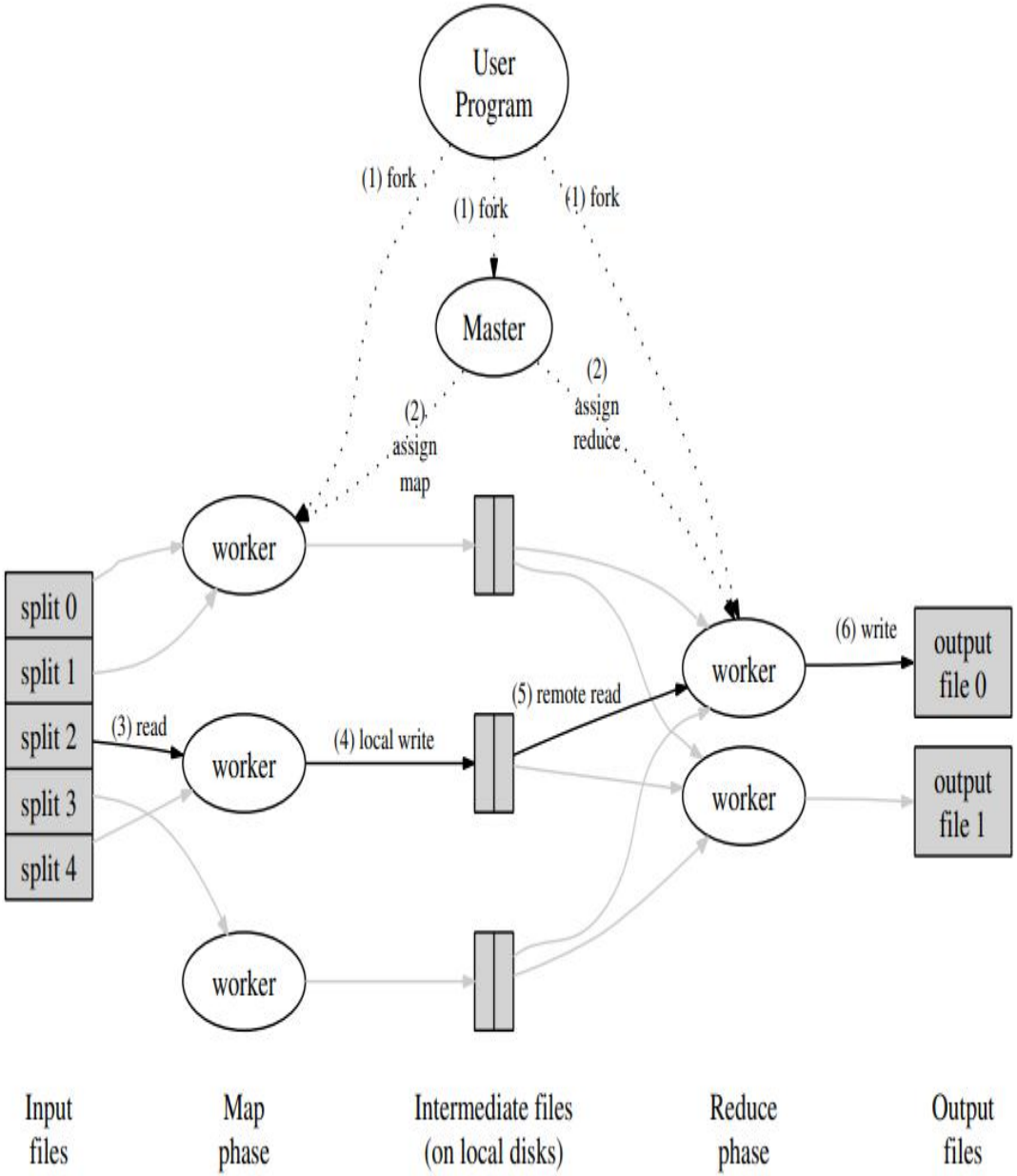
1、负责资源的监控和作业的调度
2、监控底层的其他的Task Tracker以及当前运行的job运行情况
3、一旦探测到失败的情况就把这个任务转移到其他节点继续执行跟踪任务执行进度和资源使用量



JobTracker的辅助模块，负责任务调度

1、接受JobTracker发送过来的命令，再根据命令给worker分发任务，Map Task 和Reduce Task可以同时执行。
2、把一些自己的资源使用情况，以及任务的运行进度通过Heartbeat，发送给JobTracker

Hadoop体系结构



1、MapReduce 将输入文件分割成多个片段，并在机器集群上创建多个输入文件的副本（一个Master副本+若干worker副本）

2、Master 将map或reduce task分配给worker

3、被分配map task的worker使用map function将输入片段处理为intermediate 键值对

4、 Intermediate键值对会被周期性的保存到磁盘的多个区域，并将它们的位置反馈给Master

5、Reduce worker收到master键值对位置信息的时候，则从内存中获取数据并对其进行排序和合并

6、Reduce Workers通过Reduce function处理排序后的数据，并将结果附加到输出文件中

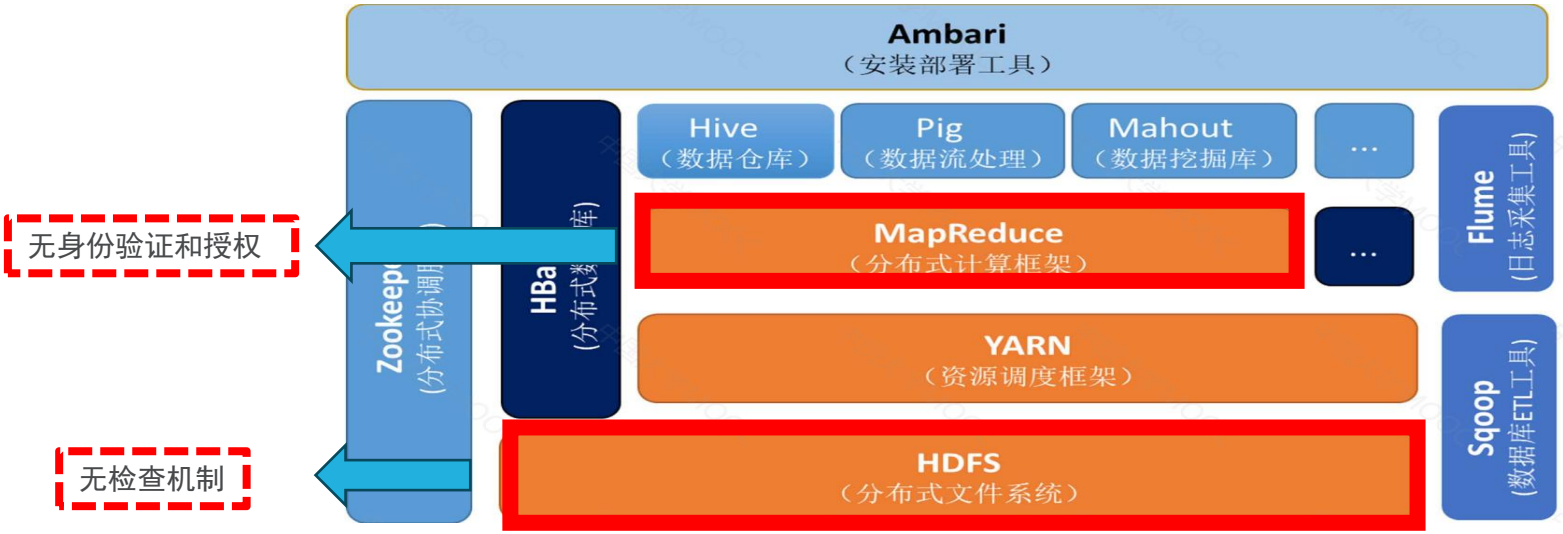
参考链接：Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107–113. https://doi.org/10.1145/1327452.1327492

内容概要

- 云计算平台
  - 云计算体系结构
  - 云计算安全风险
- 大数据平台
  - Hadoop体系结构
  - Hadoop安全风险
- 总结

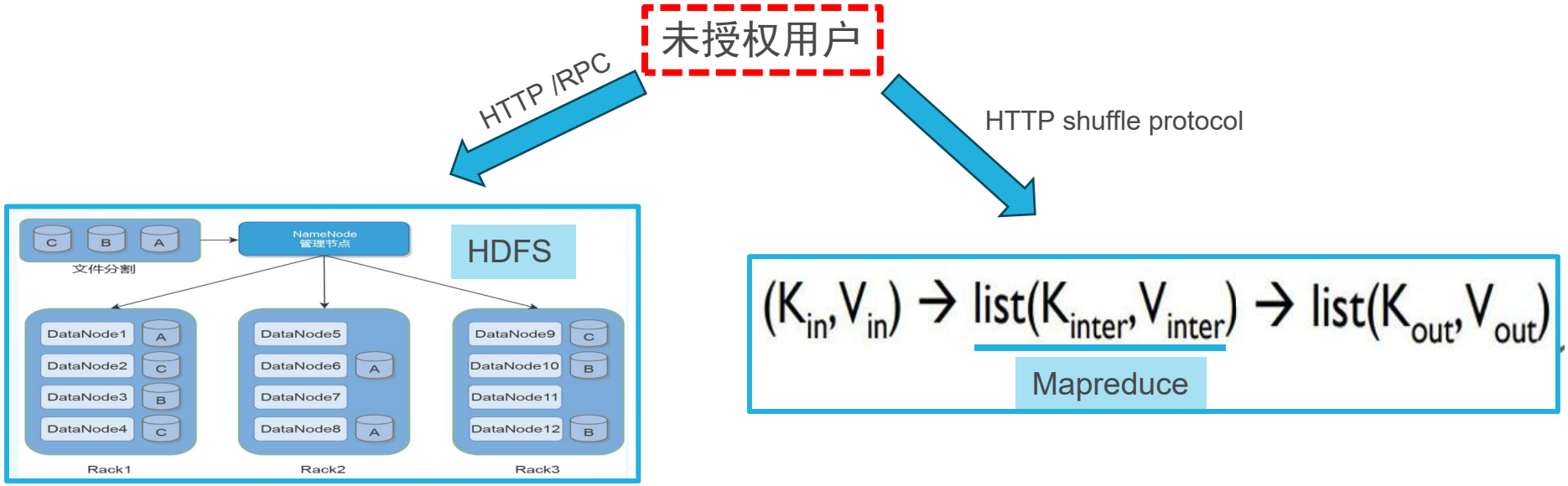
Hadoop安全风险

- HDFS 中无检查机制，用户身份可以随意申明，所以任意用户、程序均可以通过 Hadoop 客户端或编程方式访问到 Hadoop 集群内的全部数据
- MapReduce 任务没有身份验证和授权，所以任意用户均可以向集群提交任务、查看其他人的任务状态、修改任务优先级甚至强行杀死别人正在运行的程序



Hadoop安全风险

- 未授权的用户可以通过 RPC 或 HTTP 访问 HDFS 上的文件，并可以在集群内执行任意代码
- 未授权的用户可以通过 HTTP shuffle protocol直接访问一个 Map 任务的中间输出结果

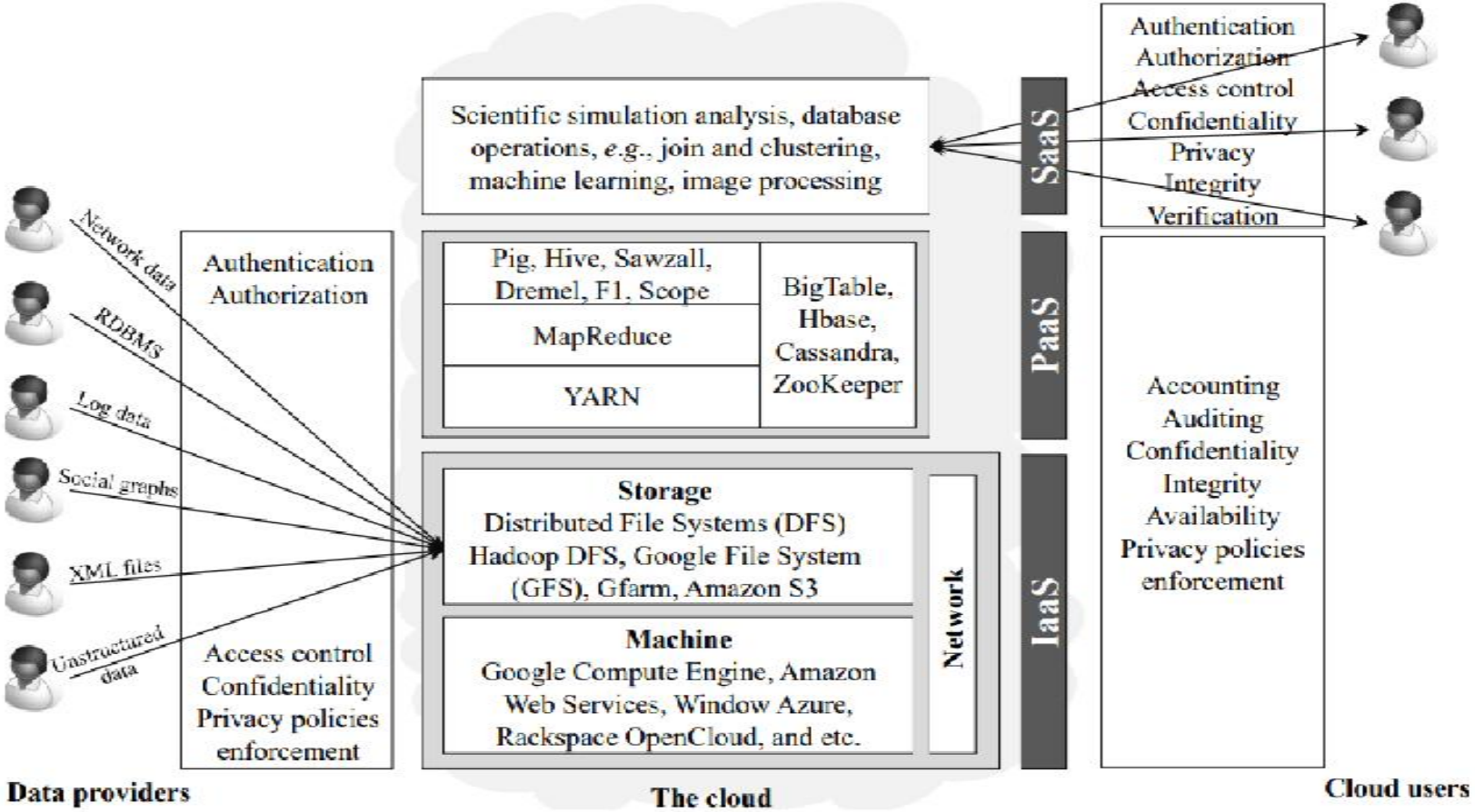


- 针对Mapreduce的主要攻击
  - Impersonation attack(冒充攻击)
    - **定义**：当对手通过对弱密码、弱加密方案或其他方式的暴力攻击成功地伪装成系统的合法用户时，就发生了冒充攻击。
    - **MapReduce中的情况**：在冒充攻击成功后，攻击者可以代表合法用户执行MapReduce作业，这些作业可能导致数据泄露、数据被篡改或数据计算错误。此外，在公有云上，受到冒充攻击时，攻击者可能会进行正常的 MapReduce 计算，而被冒名顶替的用户则需为数据存储、通信费用和计算时间支付费用。

- 针对Mapreduce的主要攻击
  - Replay 攻击
    - **定义**：当对手重新发送（或重放）一个被捕获的有效信息给节点时，就发生了重放攻击。
    - **MapReduce中的情况**：当对手向节点分配一些旧任务，使它们持续忙碌时，就发生重放攻击。
  - Eavesdropping（窃听攻击）
    - **定义**：当对手（被动地）未经同意监视网络和节点时，就发生了窃听攻击。
    - **MapReduce中的情况**：当对手未经数据和计算拥有者的同意，观察输入数据、中间输出、最终输出以及MapReduce计算过程时，发生窃听攻击。

- 针对Mapreduce的主要攻击
  - Man-in-the-Middle (MiM) attacks（中间人攻击）
    - **定义**:在MIM中，对手（主动地）修改、破坏或插入在系统的两个合法用户之间传递的数据。
    - **MapReduce中的情况**：当对手修改或破坏在框架的任何两个合法节点之间传递的计算代码、输入数据、中间输出或最终输出时，就会发生中间人攻击。
  - Repudiation（否认攻击）
    - **定义**：当一个节点错误地否认处理了一个发送的信息或任务执行时，就发生了否认攻击。
    - **MapReduce中的情况**：当一个mapper或reducer错误地否认它已经执行过的执行请求时，就会发生否认攻击。

Hadoop上的防御机制



数据提供者：

认证授权
访问控制
隐私策略

Iaas & PaaS：

账号管理
综合审计
隐私策略

SaaS：

认证授权
访问控制
多重验证

内容概要

- 云计算平台
  - 云计算体系结构
  - 云计算安全风险
- 大数据平台
  - Hadoop体系结构
  - Hadoop安全风险
- 总结

内容概要

- 介绍了云计算平台结构及其安全风险
- 介绍了大数据平台结构及其安全风险
- 结论：相比云计算平台，大数据底层的结构风险更大，尤其是Mapreduce自身安全问题比较严重。


Q&A