



中山大學 軟件工程學院
SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

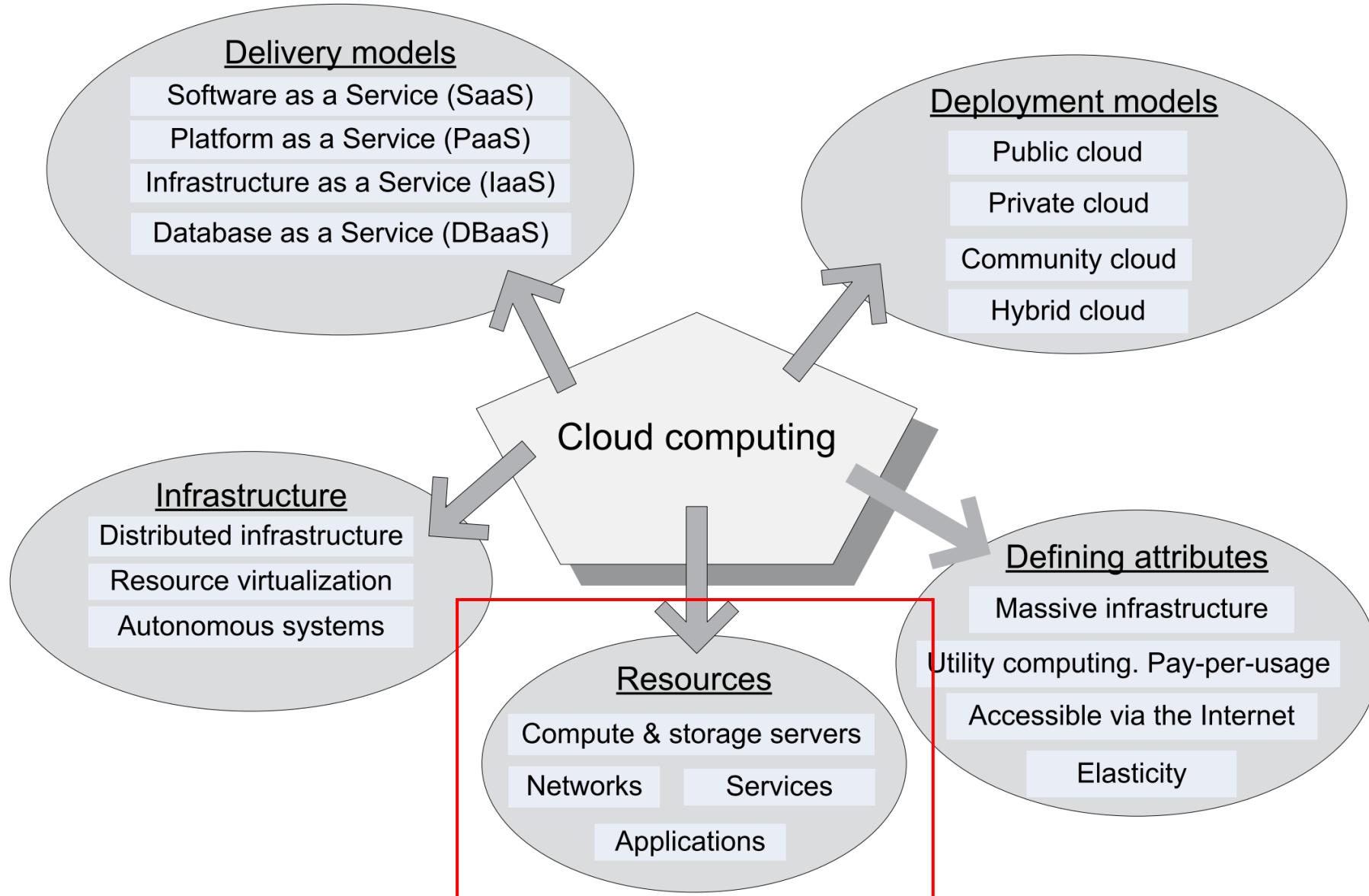
SSE316 : 云计算技术 Cloud Computing Technology

陈壮彬

软件工程学院

<https://zbchern.github.io/sse316.html>

云计算资源、特性和模型





云计算特性与模型

- ❖ 云特性
- ❖ 云部署和交付模型
- ❖ 云应用程序的开发与架构风格
- ❖ 微服务架构
- ❖ 分布式数据处理MapReduce



云计算特性与模型

- ❖ 云特性
- ❖ 云部署和交付模型
- ❖ 云应用程序的开发与架构风格
- ❖ 微服务架构
- ❖ 分布式数据处理MapReduce

云特性



云内的IT环境要求有一组特性，使其能以可扩展和可测量的方式远程提供给用户。

按需使用
on-demand usage

云用户可以单边访问基于云的IT资源，给予云用户自助提供IT资源的自由。

泛在接入
ubiquitous access

云服务被广泛访问的能力。

多租户
multitenancy

一个软件程序的实例能够服务不同的用户，且用户之间相互隔离。

云特性



云内的IT环境要求有一组特性，使其能以可扩展和可测量的方式远程提供给用户。

弹性
elasticity

云自动透明地增加或减少IT资源的能力。

可测量的使用
measured usage

云平台记录IT资源被云用户的使用情况的能力。

可恢复性
resilient computing

一种故障转移的形式，通过IT资源的冗余实现。



云计算特性与模型

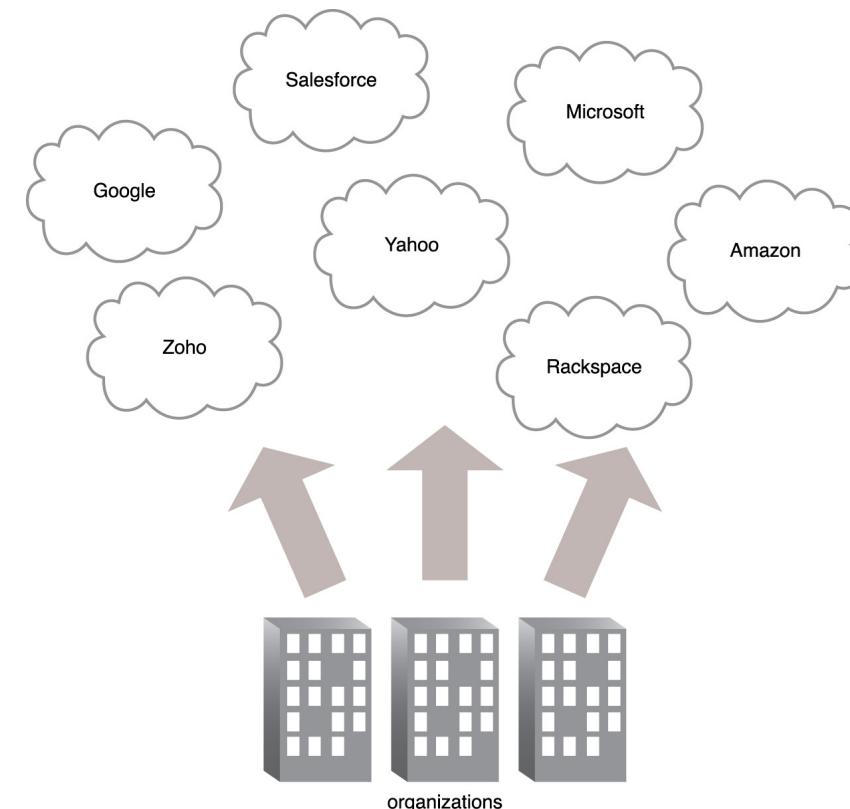
- ❖ 云特性
- ❖ 云部署和交付模型
- ❖ 云应用程序的开发与架构风格
- ❖ 微服务架构
- ❖ 分布式数据处理MapReduce



云部署模型 (cloud deployment model) 表示某种特定的云环境类型，主要以**所有权、大小和访问方式**来区别。

公有云 (public cloud)

由第三方云提供者拥有的可公共访问的云环境。

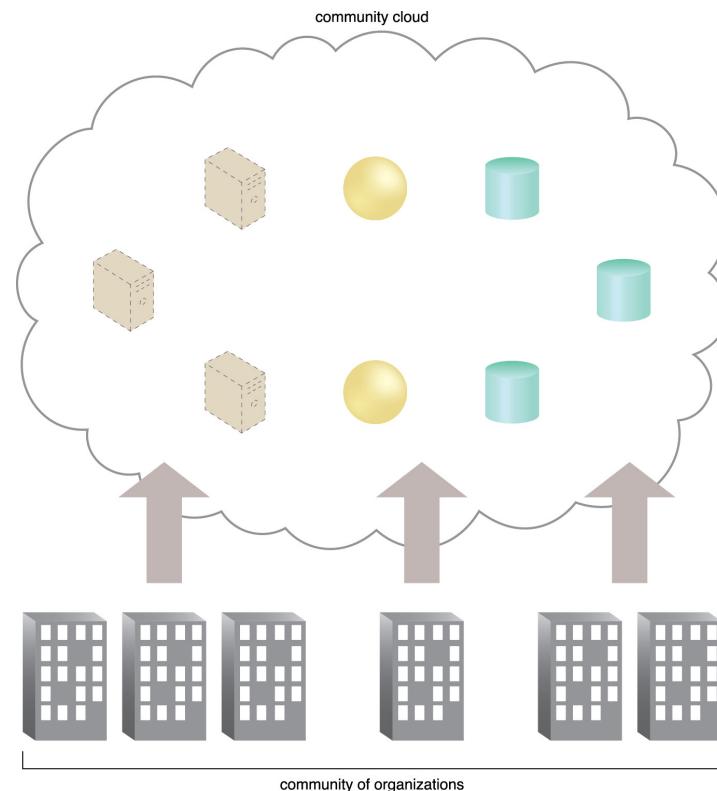




云部署模型 (cloud deployment model) 表示某种特定的云环境类型，主要以**所有权、大小和访问方式**来区别。

社区云 (community cloud)

类似公有云，只是其访问被限制为特定的云用户社区。

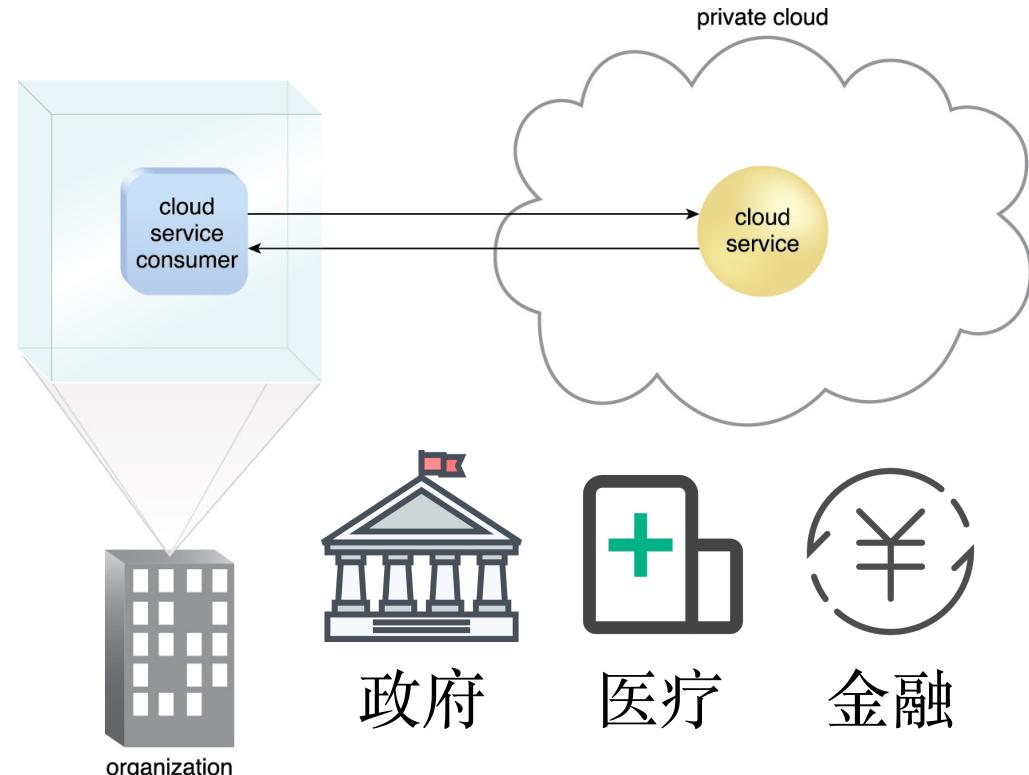




云部署模型 (cloud deployment model) 表示某种特定的云环境类型，主要以**所有权、大小和访问方式**来区别。

私有云 (private cloud)

私有云是由一家组织单独拥有的，通常由云提供者建设。

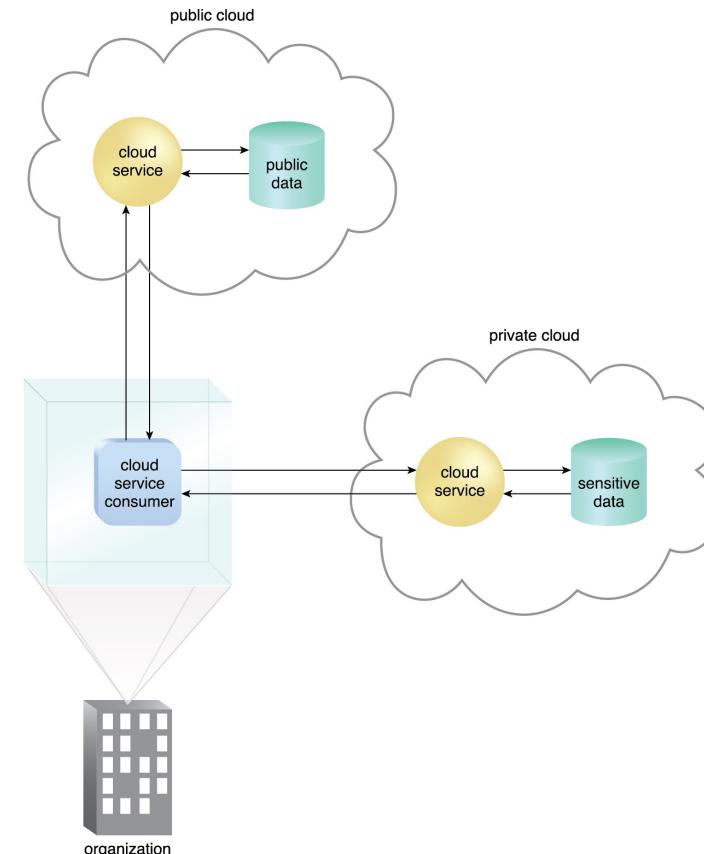




云部署模型 (cloud deployment model) 表示某种特定的云环境类型，主要以**所有权、大小和访问方式**来区别。

混合云 (hybrid cloud)

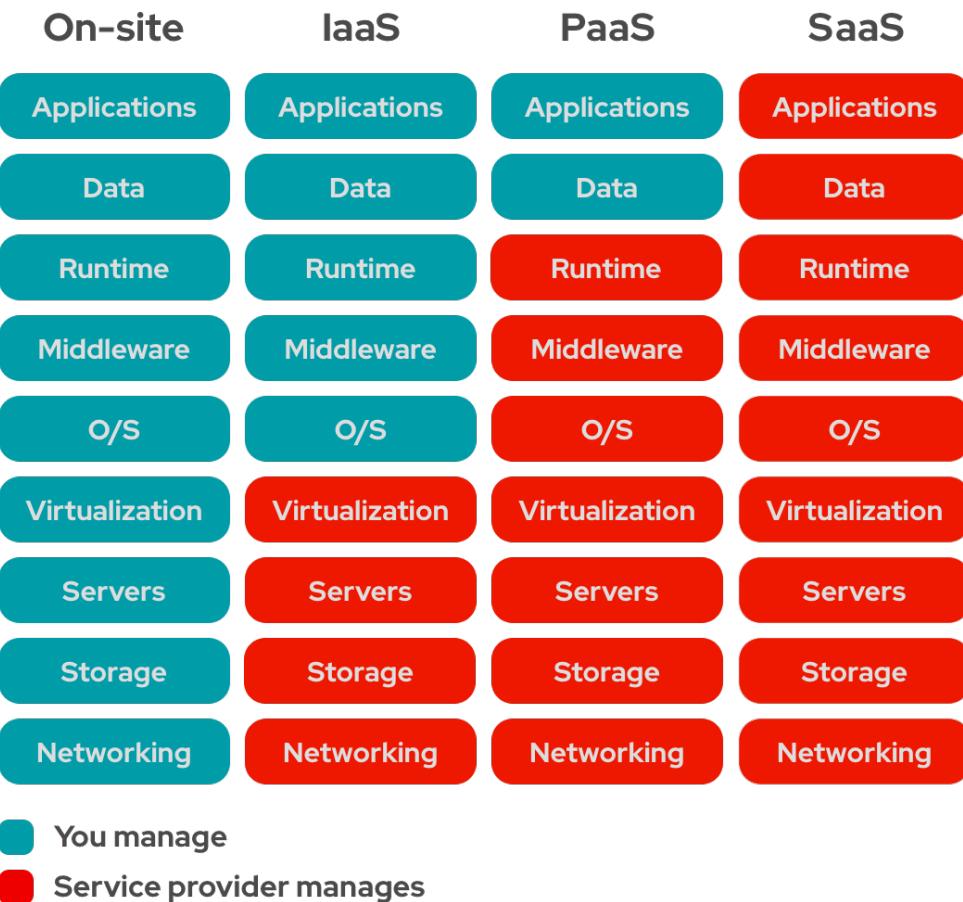
混合云是由两种或以上云部署模型组成的云环境。



云交付模型



云交付模型 (cloud delivery model) 是云提供者提供的具体的、事先打包好的IT资源组合。



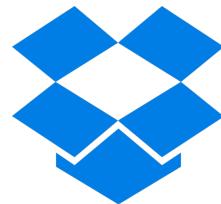
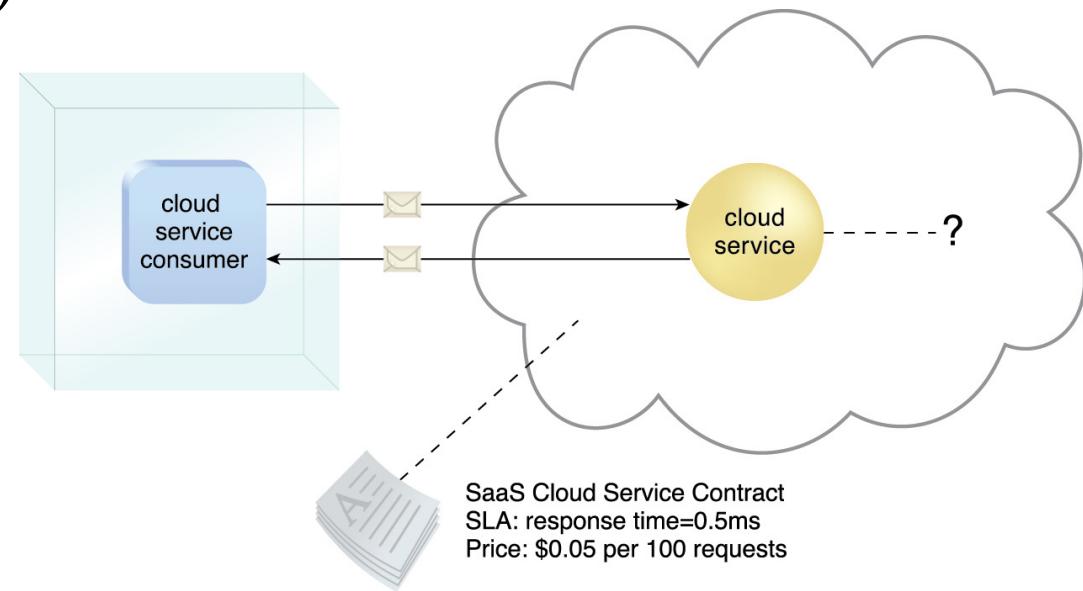
常见的三种
云交付模型





软件作为服务 (Software as a Service, SaaS)

- 把软件程序定位成共享的云服务，作为“产品”进行提供
- 用户可通过网页浏览器、桌面或客户端访问
- 基础设施由SaaS供应商托管和管理

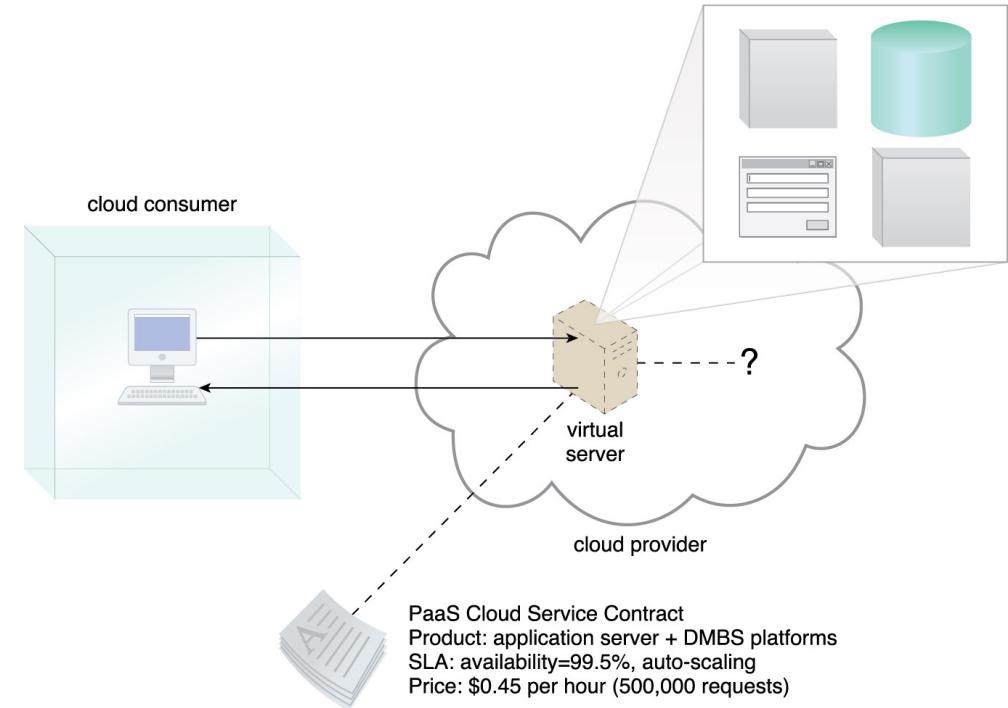


云交付模型



平台作为服务 (Platform as a Service, PaaS)

- 预先定义好的“就绪可用 (ready-to-use)”的环境
- 由已经部署好和配置好的IT资源（操作系统、中间件、开发环境等）组成
- 云用户可以进行应用开发、运行和管理



阿里云

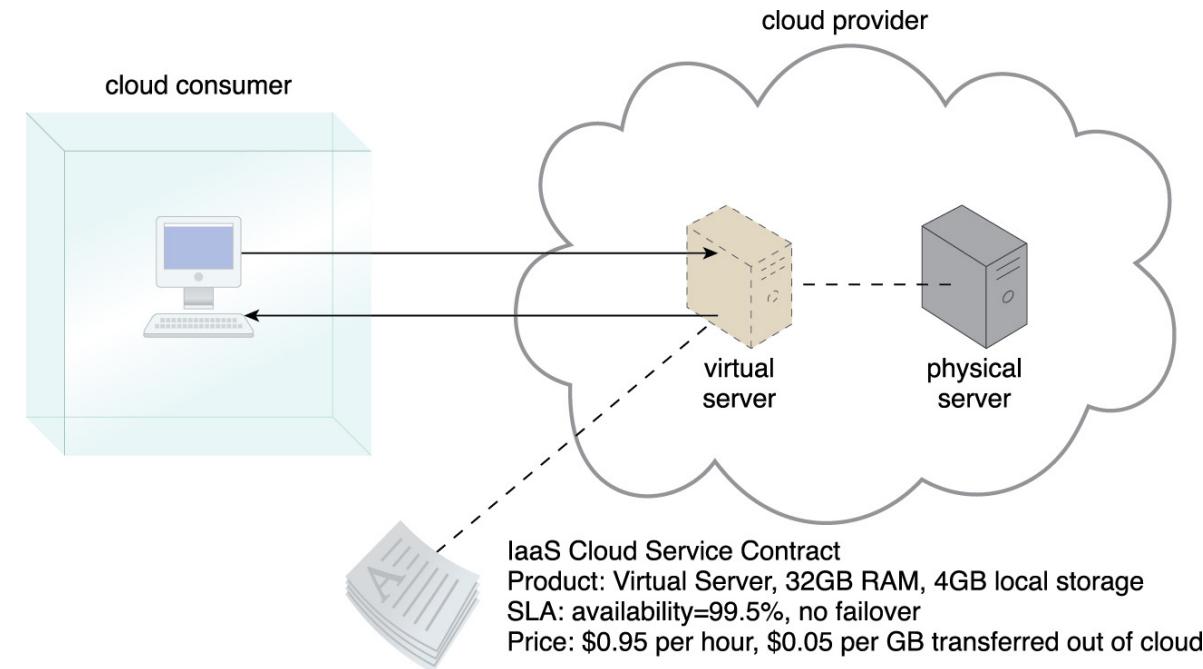
华为云

腾讯云



基础设施作为服务 (Infrastructure as a Service, IaaS)

- 自我包含的IT环境，由以基础设施为中心的IT资源（服务器、存储、网络资源）组成
- 云用户可以通过基于云服务的接口和工具访问和管理这些资源

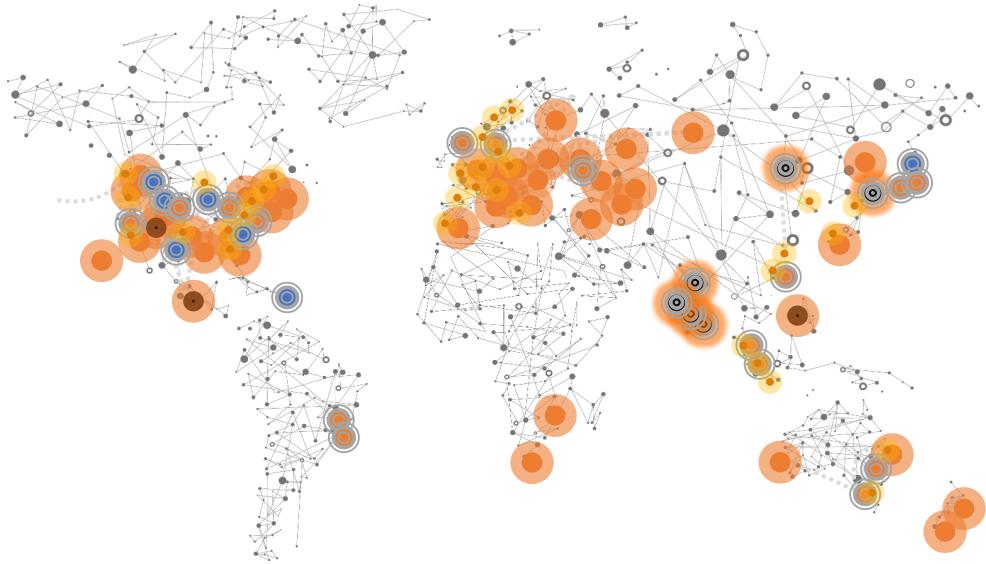


 阿里云

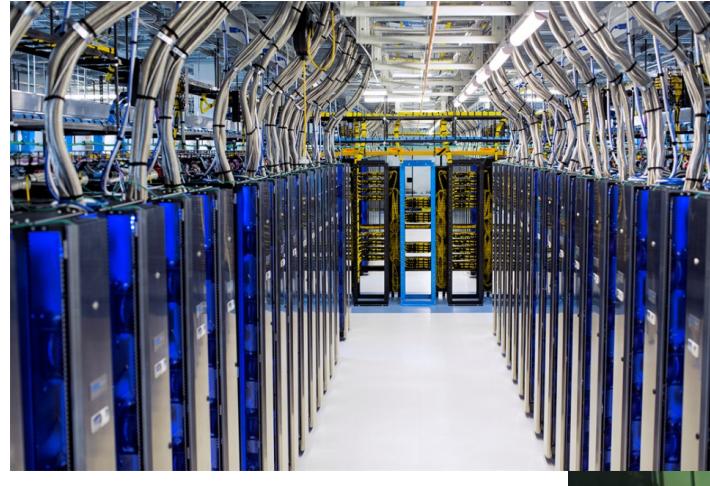
 华为云

 腾讯云

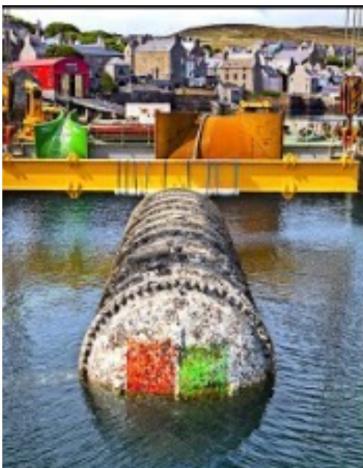
云基础设施—数据中心



微软云数据中心遍布全球



微软云数据中心内部



微软云水下数据中心

其他云交付模型



- ✓ 函数即服务 (Function as a Service)
- ✓ 存储作为服务 (Storage as a Service)
- ✓ 安全作为服务 (Security as a Service)
- ✓ 测试作为服务 (Testing as a Service)
- ✓ ...



云计算特性与模型

- ❖ 云特性
- ❖ 云部署和交付模型
- ❖ 云应用程序的开发与架构风格
- ❖ 微服务架构
- ❖ 分布式数据处理MapReduce



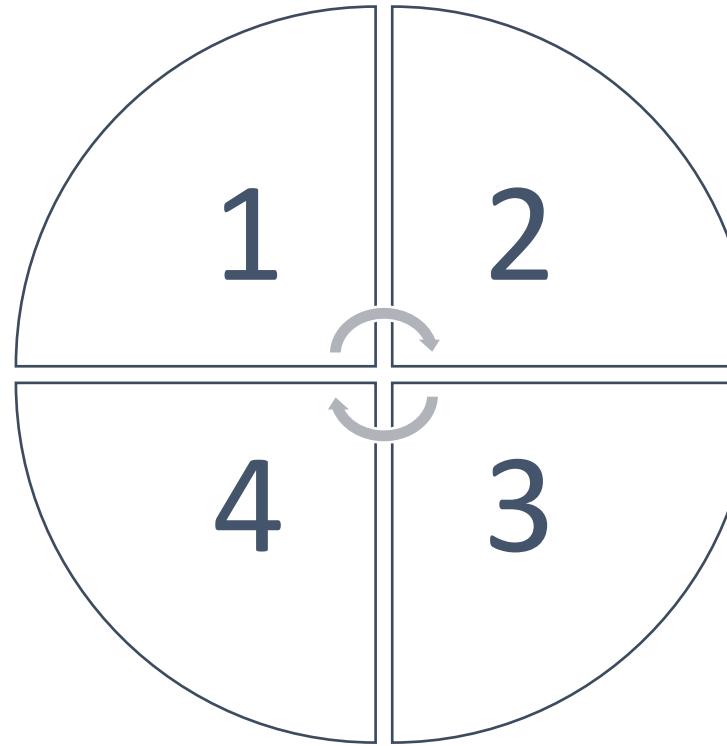
□多年来，应用开发人员和服务提供商面临巨大的挑战

应用开发困难

现有大型计算系统对开发高效的数据密集型或计算密集型应用不友好

经济效益低

资源集中提供的任何经济优势都会被昂贵资源的相对低的利用率低消



定制化的系统

应用程序移植困难，针对一个系统优化的应用在其他系统上表现不佳

系统资源管理困难

系统规模庞大且负载高度动态，故障发生后，安全性和快速恢复难以保证

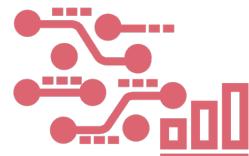
云应用的开发风格



云计算以低成本实现高效计算，提供即时基础设施，将应用开发与系统运行分离开。

适用于云计算的应用程序

工作负载可以划分成任意大小的段，并且可以由云中可用的服务器并行处理。



大数据分析
和处理



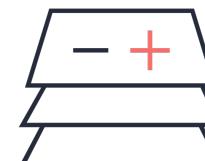
Web应用



云存储

不适用于云计算的应用程序

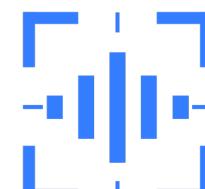
无法对工作负载任意划分或需要在并发实例之间进行密集通信的应用程序。



高性能计算



大规模数据库



实时应用



绝大多数云应用利用请求-响应方式与客户通信

有状态应用 (Stateful Apps)

有状态服务会在服务处理过程中维护某种状态信息，比如会话数据、用户配置信息等，需要维护一个持久化的状态存储，例如数据库或缓存。



电子商务



游戏



聊天

无状态应用 (Stateless Apps)

无状态服务在处理请求时不会维护任何状态信息，每个请求都是相互独立的，不需要额外的上下文信息。



Web 应用



API 服务



静态文件服务



云计算特性与模型

- ❖ 云特性
- ❖ 云部署和交付模型
- ❖ 云应用程序的开发与架构风格
- ❖ 微服务架构
- ❖ 分布式数据处理MapReduce

单体软件架构 (monolithic software)



- 一种软件架构模式，其中所有组件都紧密耦合并集成到单个可执行文件或模块中

Monolithic Architecture



单体软件架构 (monolithic software)



□ 单体软件架构的问题

难以扩展

架构难以维护及进化

缺乏敏
捷性

开发/测试/部
署周期长

新版本周期长达数月

缺乏创
新

维护困难

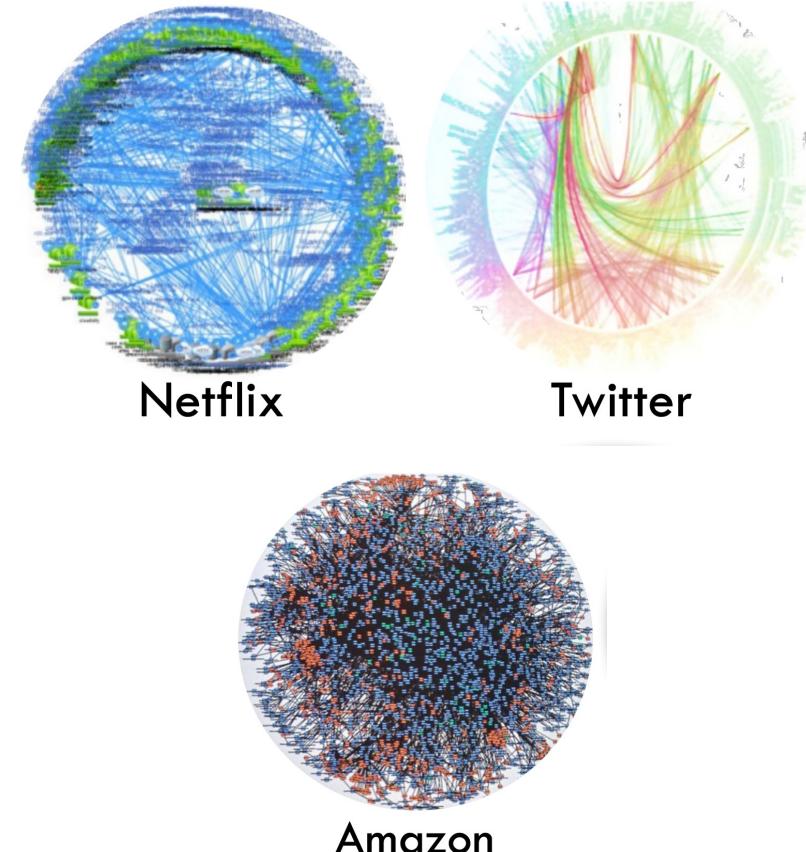
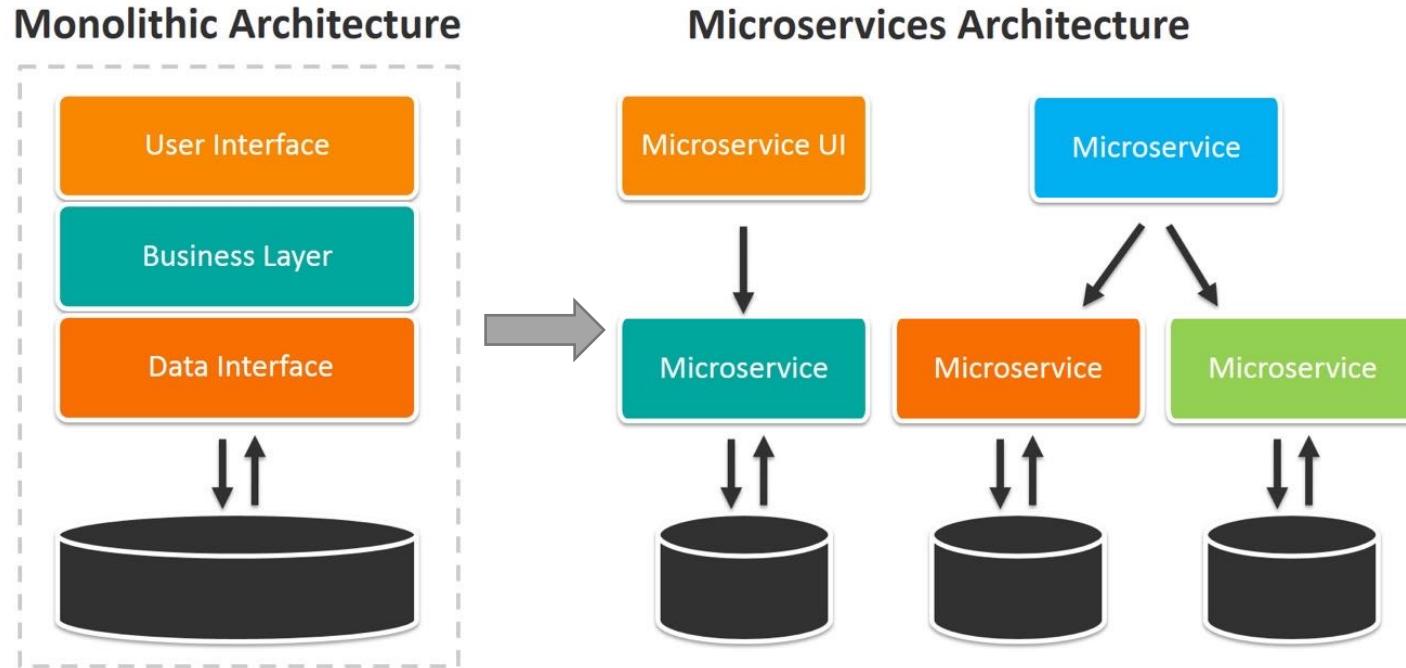
某个模块出现问题难以问责

缺乏可
恢复性

微服务架构 (microservices)



□ 微服务架构是一种软件架构模式，它将一个大型的软件应用程序划分成多个松耦合、可独立部署的服务



微服务架构的特点



“service-oriented architecture
composed of
loosely-coupled elements
that have
bounded contexts”

微服务架构的特点



“service-oriented architecture
composed of
loosely-coupled elements
that have
bounded contexts”

每个微服务通过轻量级的通信机制（如HTTP、RPC）进行通信。

微服务架构的特点



“service-oriented architecture
composed of
loosely-coupled elements
that have
bounded contexts”

每个服务有独立的开发、
测试和部署流程，不会
影响其他服务的功能。

微服务架构的特点



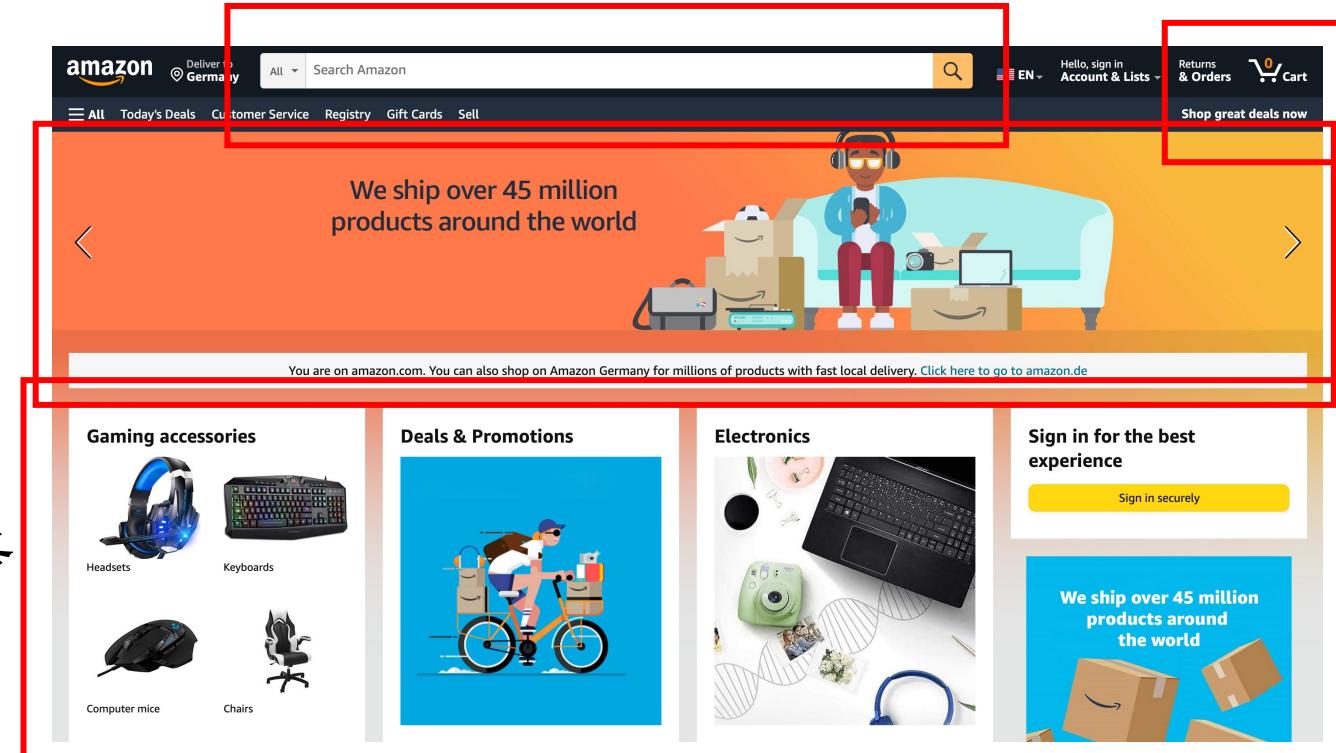
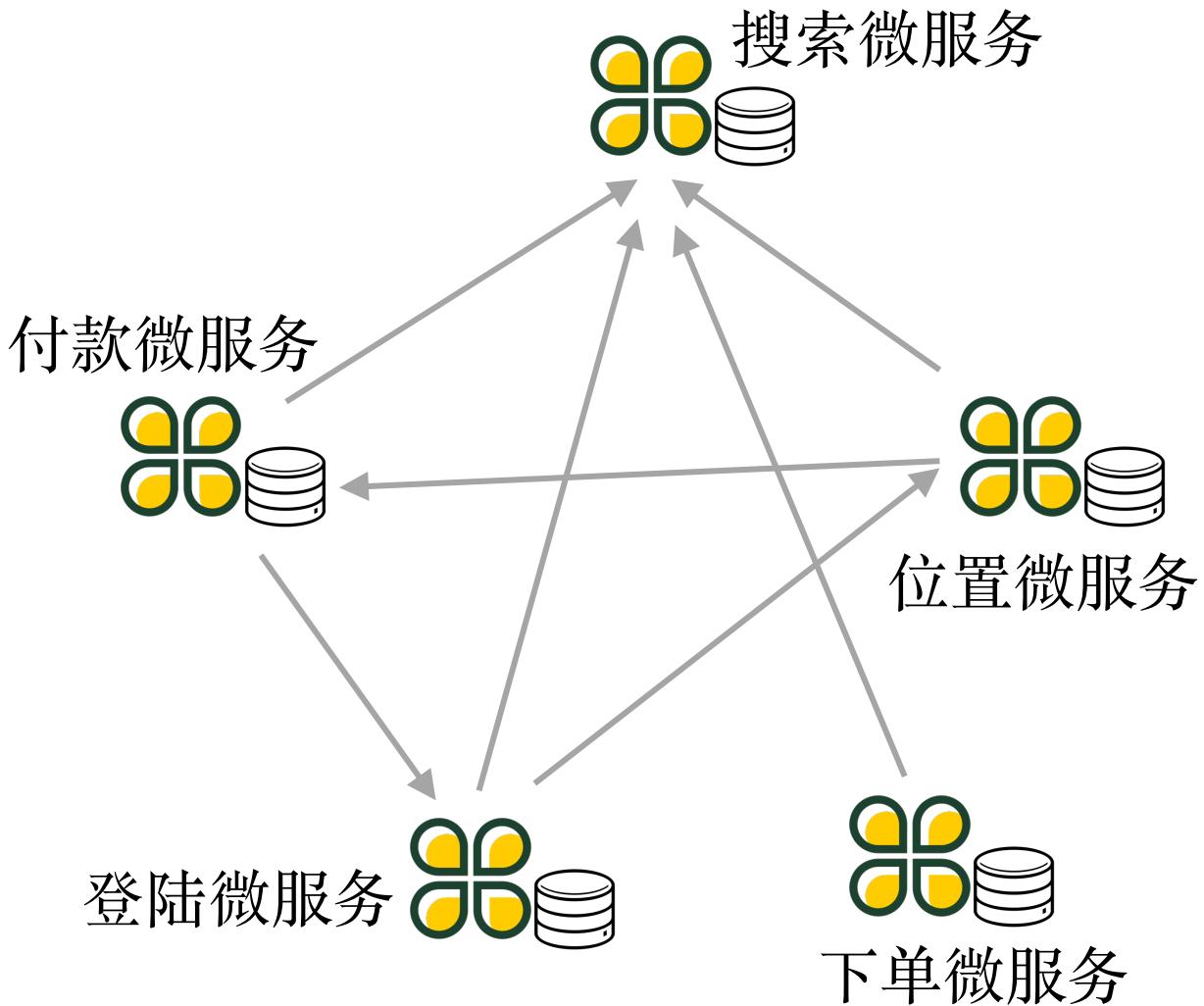
“service-oriented architecture
composed of
loosely-coupled elements
that have
bounded contexts”

每个服务都有自己独
立的代码库、数据存
储和数据库等。

微服务架构的可扩展性



□ 微服务架构实例



微服务架构的可扩展性



单体软件架构

单体软件进行扩展需要运行整个程序副本，包含所有的功能模块。



不同模块处理
请求所需的资
源/时间不一样



微服务架构

对瓶颈微服务进行精准扩容，更容易地适应不断变化的业务需求。



亚马逊微服务例子

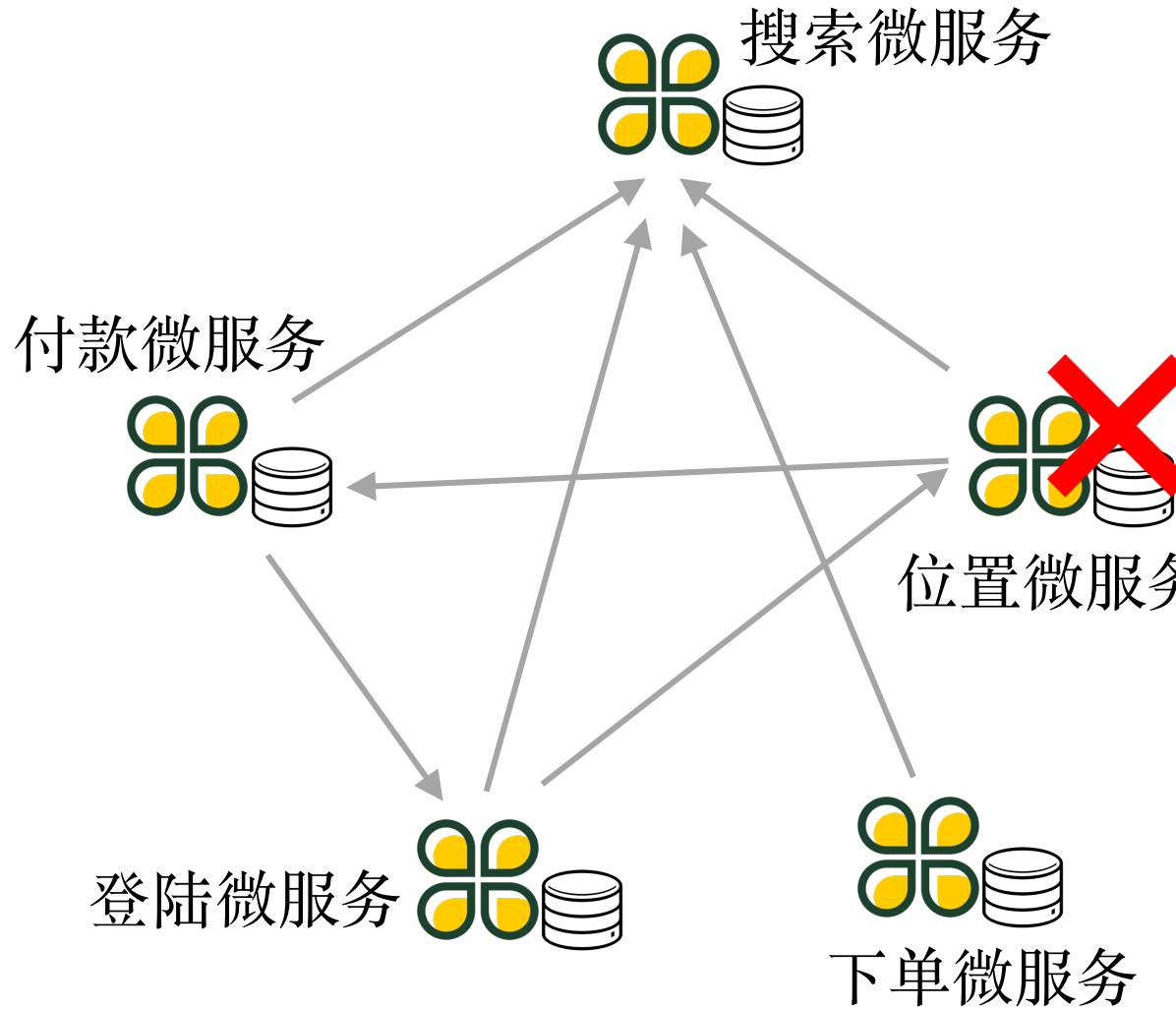
Thousands of teams

- × Microservices**
- × Continuous delivery**
- × Multiple environments**

= 50 million deployments a year

每小时 5708 个更新，或每0.63秒1个

微服务架构的可靠性



每个微服务都是独立的，发生故障时只会影响单个服务，而不会影响整个应用程序。

函数即服务 (Function as a Service)



- 函数即服务 (FaaS) 是一种无服务器计算模型
- 允许开发人员将单个函数或小代码片段部署到云端
- 云服务提供商负责管理运行这些函数所需的基础设施

FaaS vs. Microservices



特点	FaaS	微服务
架构模式	无服务计算	分布式架构
部署单元	函数	服务
部署方式	事件驱动	部署在虚拟机或容器中
依赖关系	无	通过API通信
可扩展性	自动扩展	手动扩展
管理复杂度	低	中等
适用场景	任务简单、数据处理	复杂应用、业务逻辑



云计算特性与模型

- ❖ 云特性
- ❖ 云部署和交付模型
- ❖ 云应用程序的开发与架构风格
- ❖ 微服务架构
- ❖ 分布式数据处理MapReduce

分布式数据处理MapReduce



云计算的一个主要优势是弹性。应用的工作负载被划分成任意数量的较小工作负载。

MapReduce基于一种简单的思想来并行处理数据密集型应用，支持任意可分割的负载。

分布式数据处理MapReduce



□产生背景



Jeffery Dean设计一个新的抽象模型，封装并行处理、容错处理、本地化计算、负载均衡的细节，还提供了一个简单而强大的接口。

这就是MapReduce。

分布式数据处理MapReduce



□产生背景

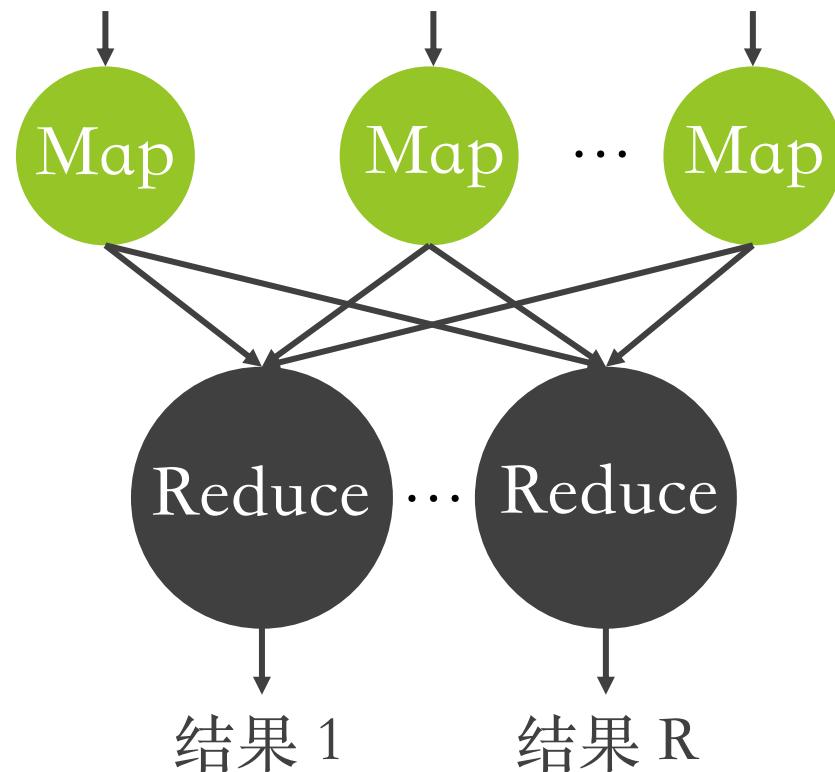
MapReduce这种并行编程模式思想最早是在1995年提出的。

MapReduce把对数据集的大规模操作，分发给一个主节点管理
下的各分节点共同完成，通过这种方式实现任务的可靠执行与容错机制。

MapReduce编程模型



原始数据 1 原始数据 2 原始数据 M



Map函数：对一部分原始数据进行指定的操作。每个Map操作都针对不同的原始数据，因此Map与Map之间是互相独立的，这使得它们可以充分并行化。

Reduce操作：对每个Map所产生的一部分中间结果进行合并操作，每个Reduce所处理的Map中间结果是互不交叉的，所有Reduce产生的最终结果经过简单连接就形成了完整的结果集。

MapReduce编程模型



Map: $(in_key, in_value) \rightarrow \{(key_j, value_j) \mid j = 1 \cdots k\}$

Reduce: $(key, [value_1, \dots, value_m]) \rightarrow (key, final_value)$

Map输入参数：in_key和in_value，它指明了Map需要处理的原始数据

Map输出结果：一组<key,value>对，这是经过Map操作后所产生的中间结果

MapReduce编程模型



Map: $(in_key, in_value) \rightarrow \{(key_j, value_j) \mid j = 1 \cdots k\}$

Reduce: $(key, [value_1, \dots, value_m]) \rightarrow (key, final_value)$

Reduce输入参数 : $(key, [value_1, \dots, value_m])$

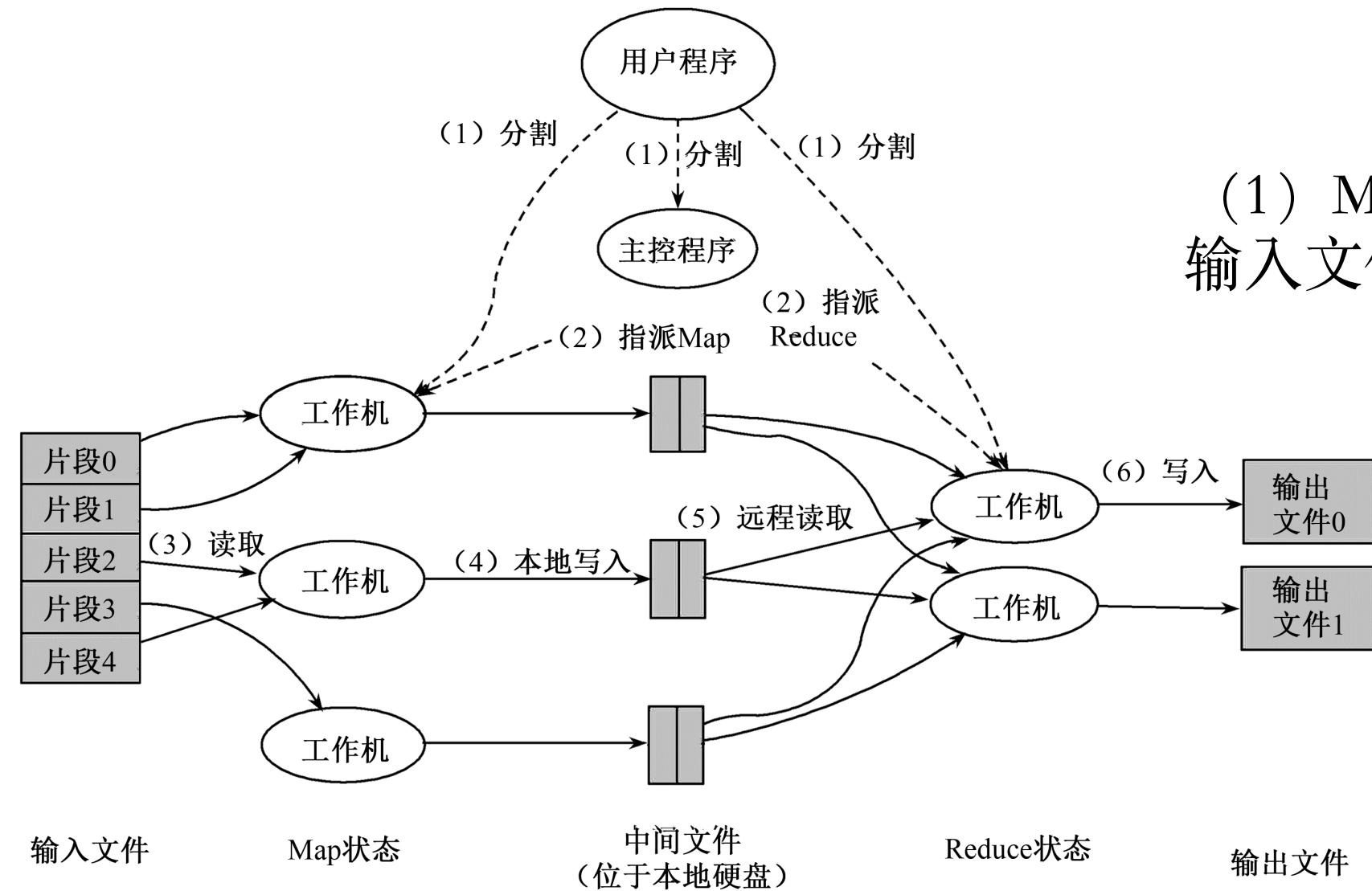
Reduce工作 :

对这些对应相同key的value值进行归并处理

Reduce输出结果 :

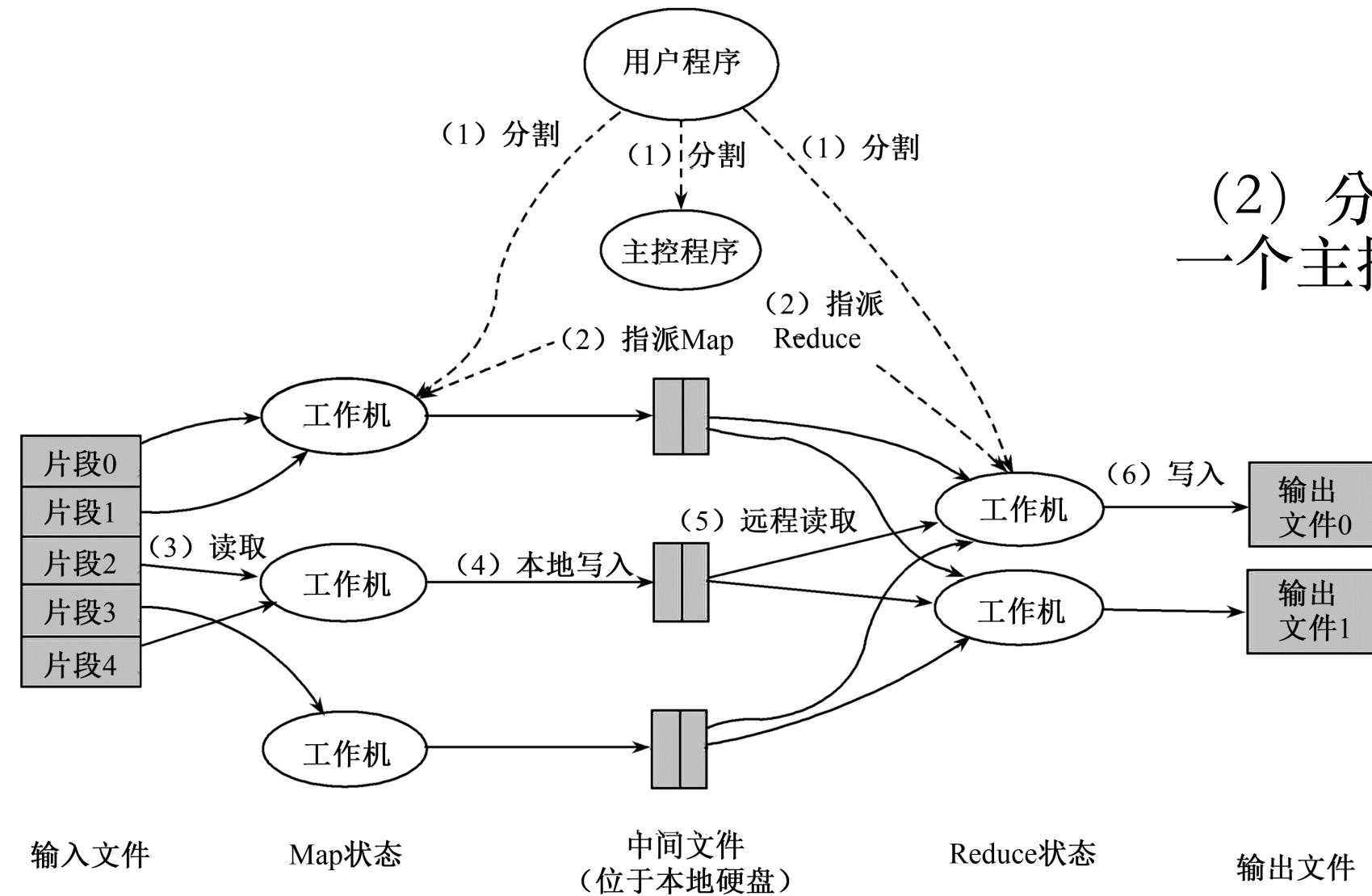
$(key, final_value)$, 所有Reduce的结果并在一起
就是最终结果

MapReduce实现机制



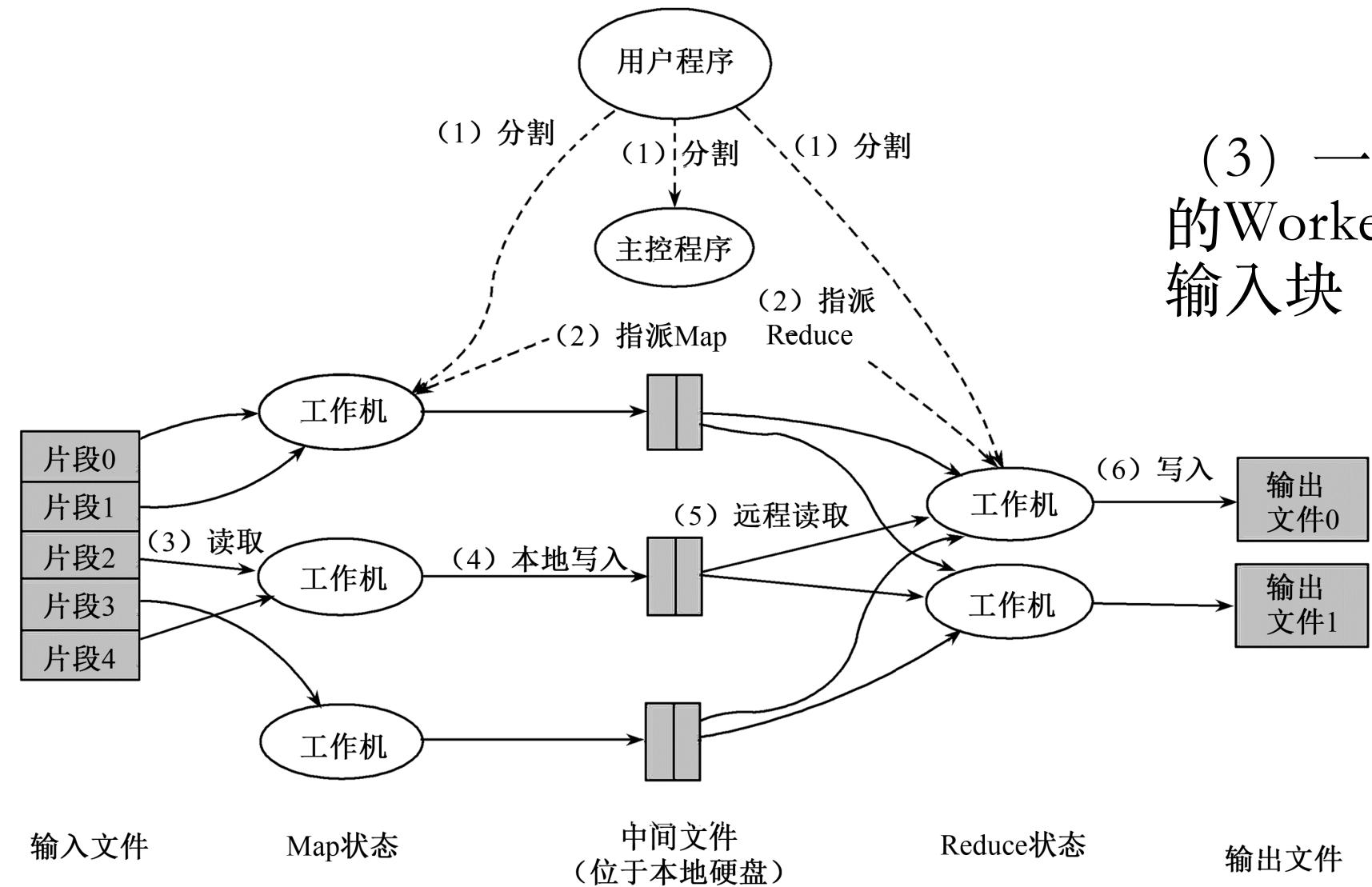
(1) MapReduce函数首先把输入文件分成M块

MapReduce实现机制



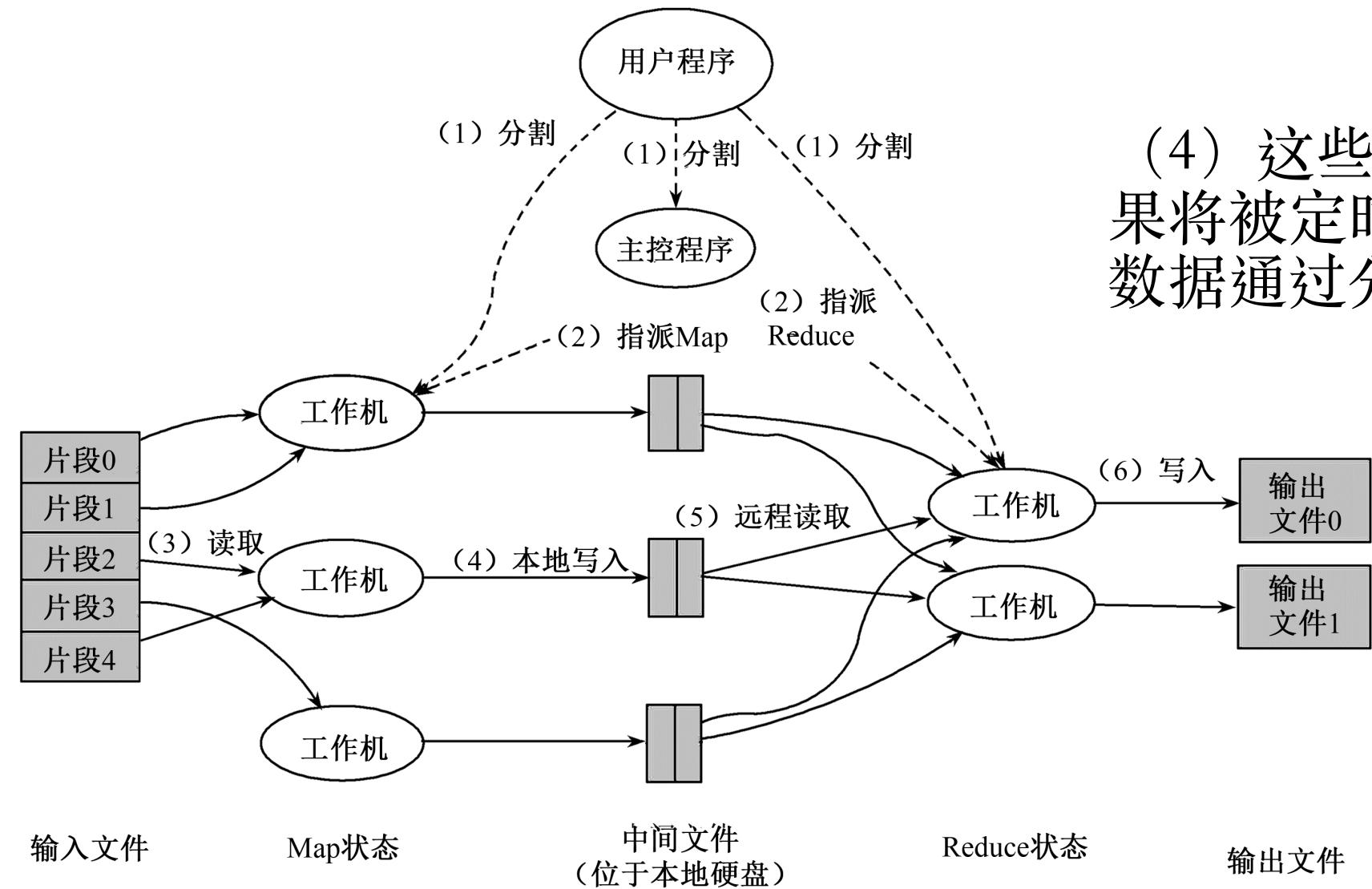
(2) 分派的执行程序中有一个主控程序Master

MapReduce实现机制

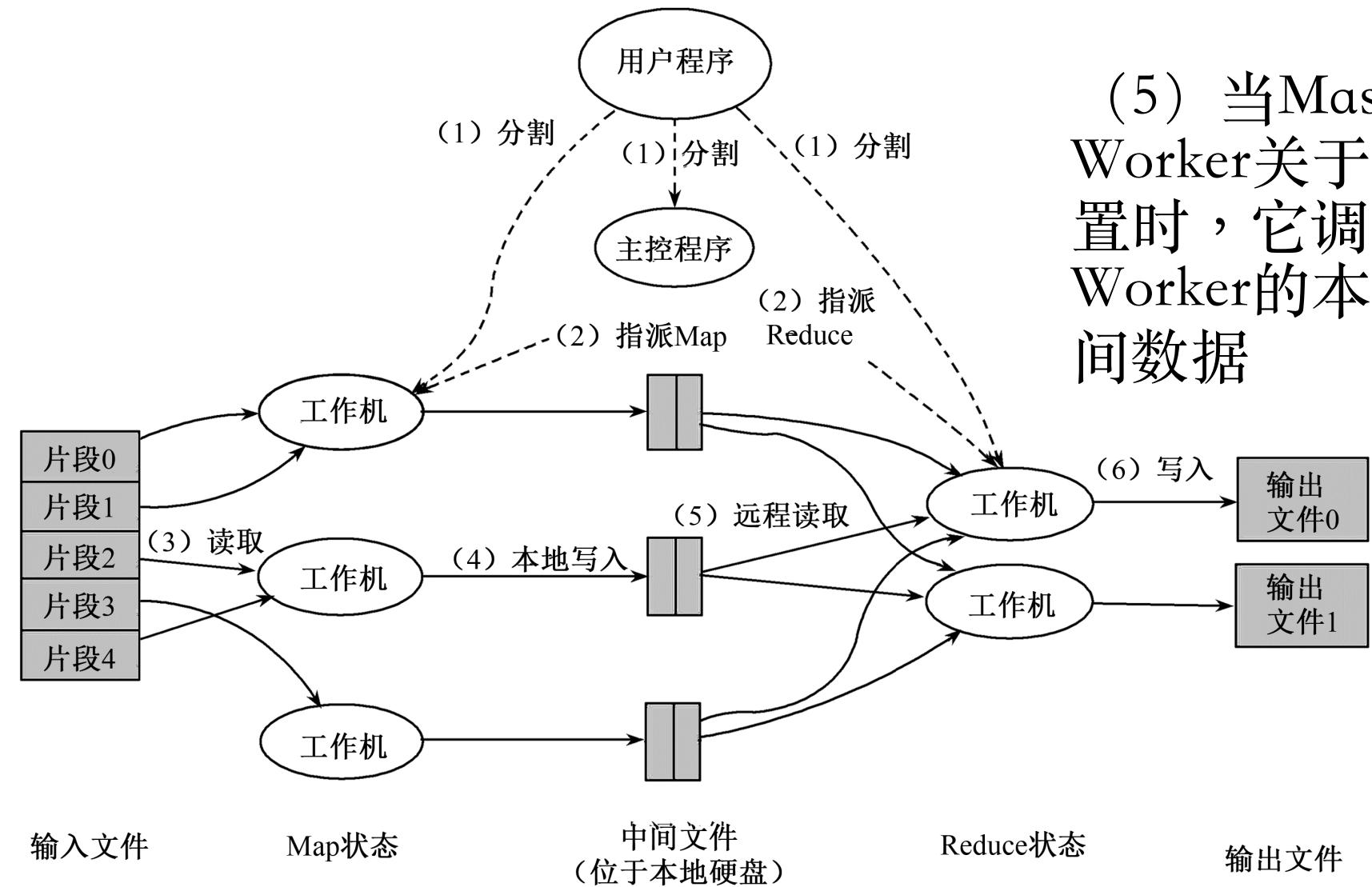


(3) 一个被分配了Map任务的Worker读取并处理相关的输入块

MapReduce实现机制

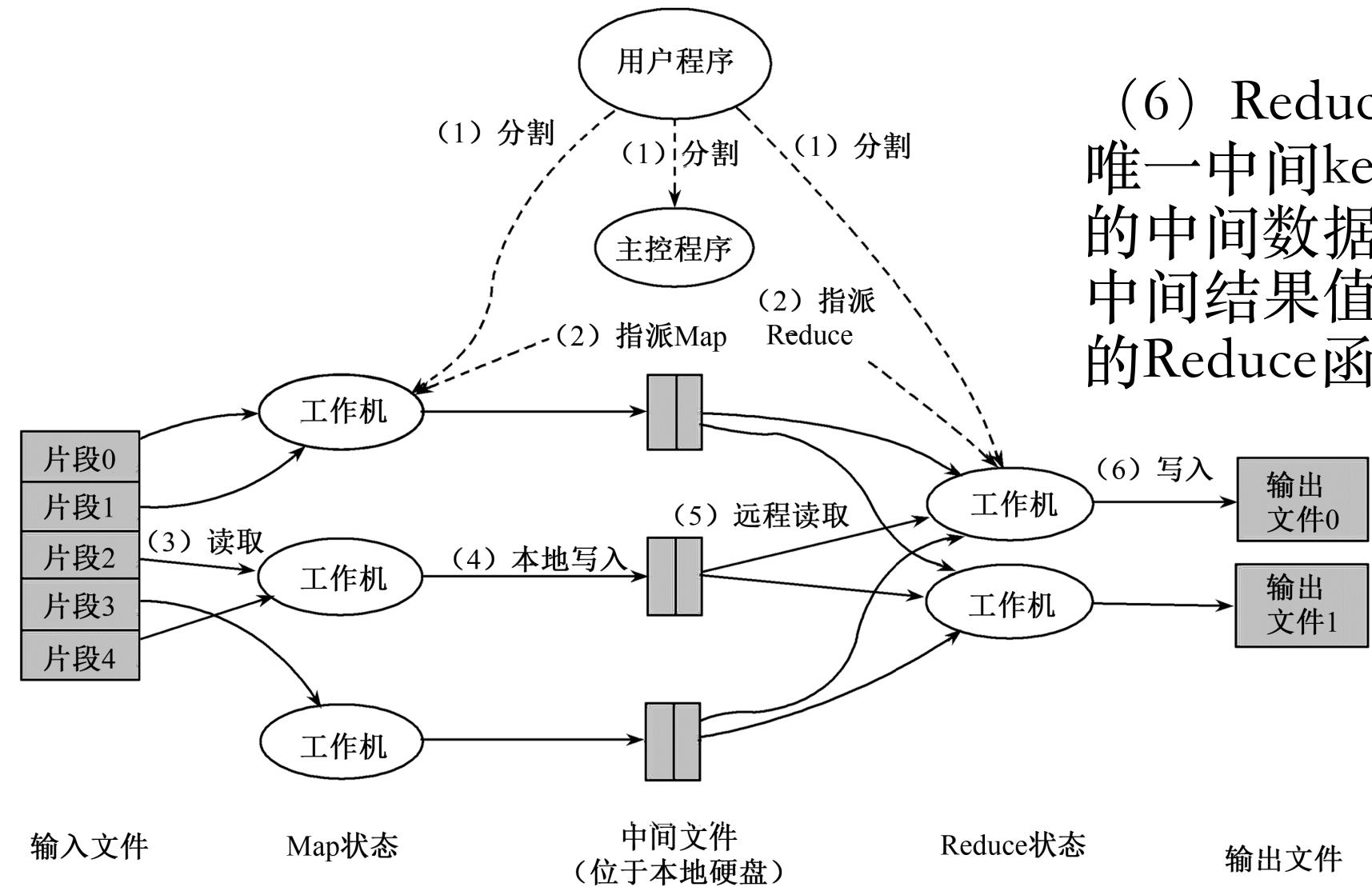


MapReduce实现机制



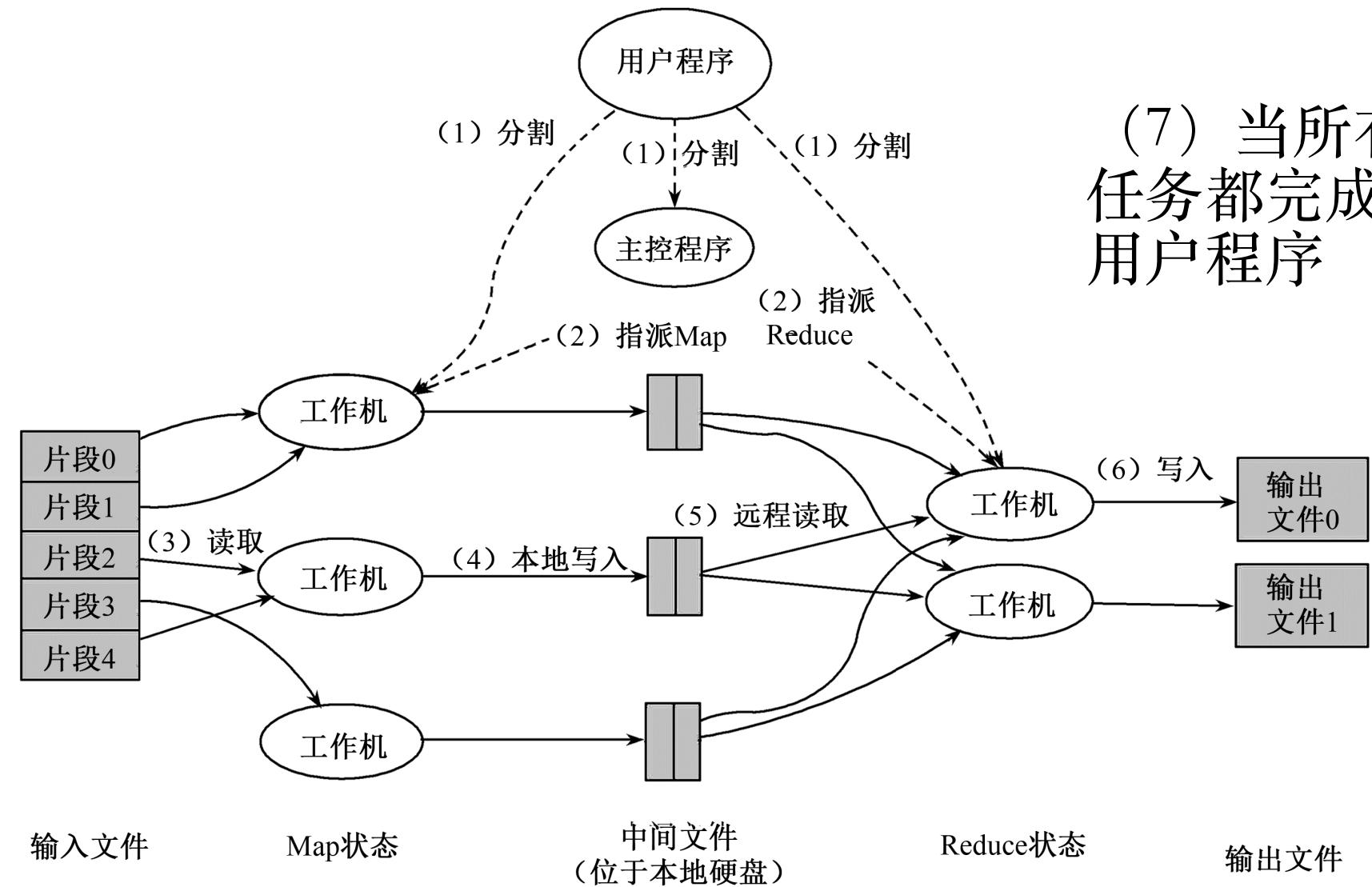
(5) 当Master通知执行Reduce的Worker关于中间对的位置时，它调用远程过程，从Map Worker的本地硬盘上读取缓冲的中间数据

MapReduce实现机制



(6) Reduce Worker根据每一个唯一中间key来遍历所有的排序后的中间数据，并且把key和相关的中间结果值集合传递给用户定义的Reduce函数

MapReduce实现机制



(7) 当所有的Map任务和Reduce任务都完成的时候，Master激活用户程序

MapReduce容错机制



由于MapReduce在成百上千台机器上处理海量数据，所以容错机制是不可或缺的。
MapReduce通过重新执行失效的地方来实现容错。

Master失效

Master会周期性地设置检查点（checkpoint），并导出Master的数据。一旦某个任务失效，系统就从最近的一个检查点恢复并重新执行。

由于只有一个Master在运行，如果Master失效了，则只能终止整个MapReduce程序的运行并重新开始。

MapReduce容错机制



由于MapReduce在成百上千台机器上处理海量数据，所以容错机制是不可或缺的。
MapReduce通过重新执行失效的地方来实现容错。

Worker失效

Master会周期性地给Worker发送ping命令，如果没有Worker的应答，则Master认为Worker失效，终止对这个Worker的任务调度，把失效Worker的任务调度到其他Worker上重新执行。

MapReduce案例分析



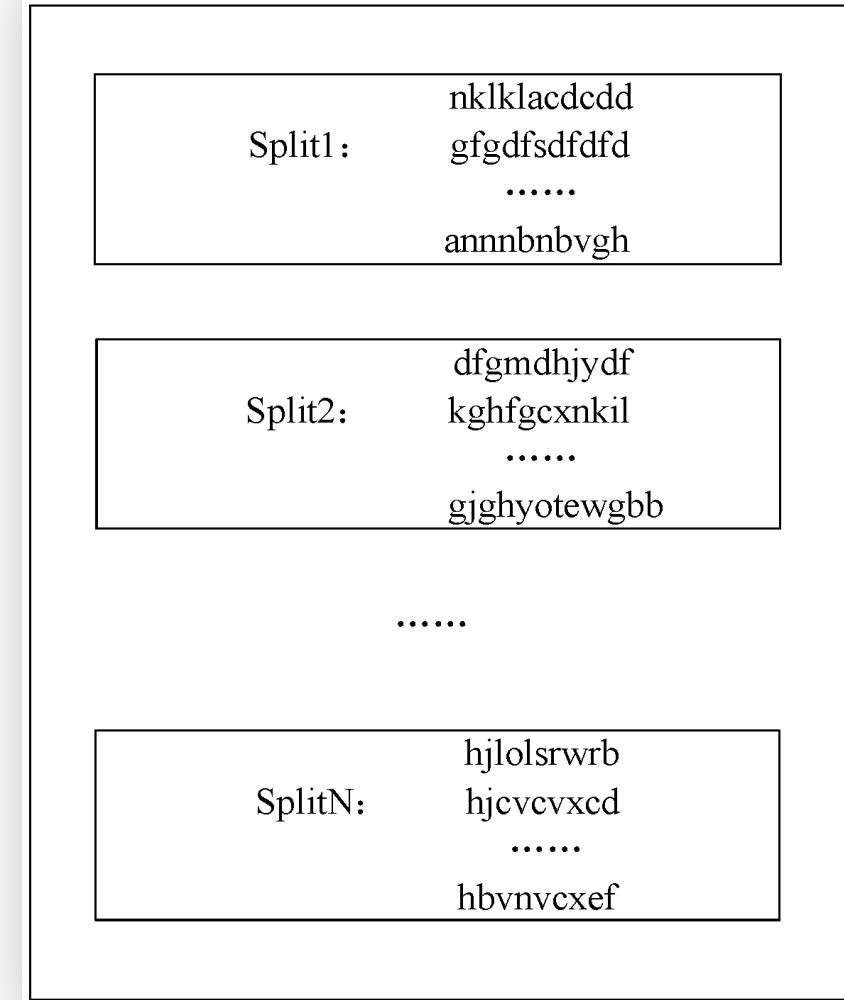
怎样通过MapReduce完成排序工作，
使其有序（字典序）呢？

MapReduce案例分析



□ 步骤一

对原始的数据进行分割（Split），
得到N个不同的数据分块。



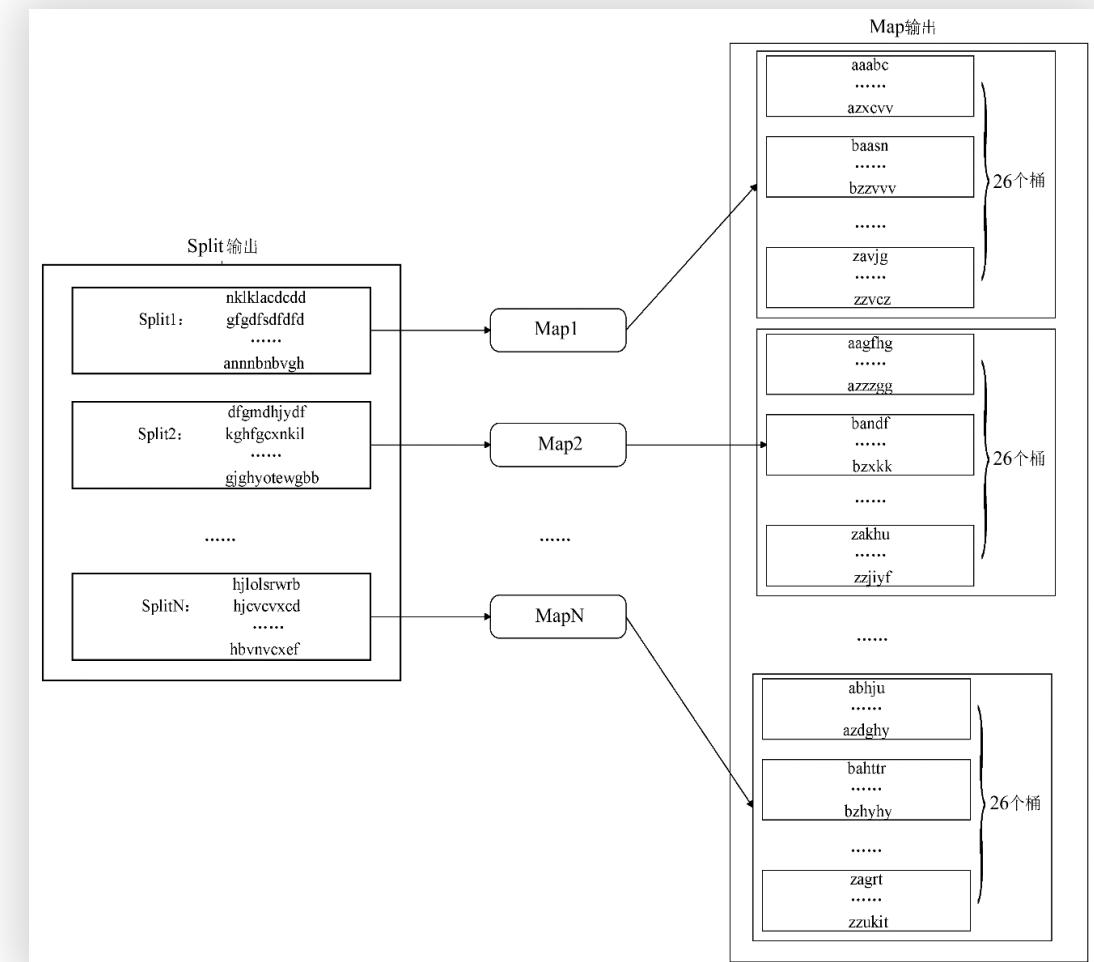
MapReduce案例分析



□ 步骤二

对每一个数据分块都启动一个Map进行处理。

采用桶排序的方法，每个Map中按照首字母将字符串分配到26个不同的桶中。



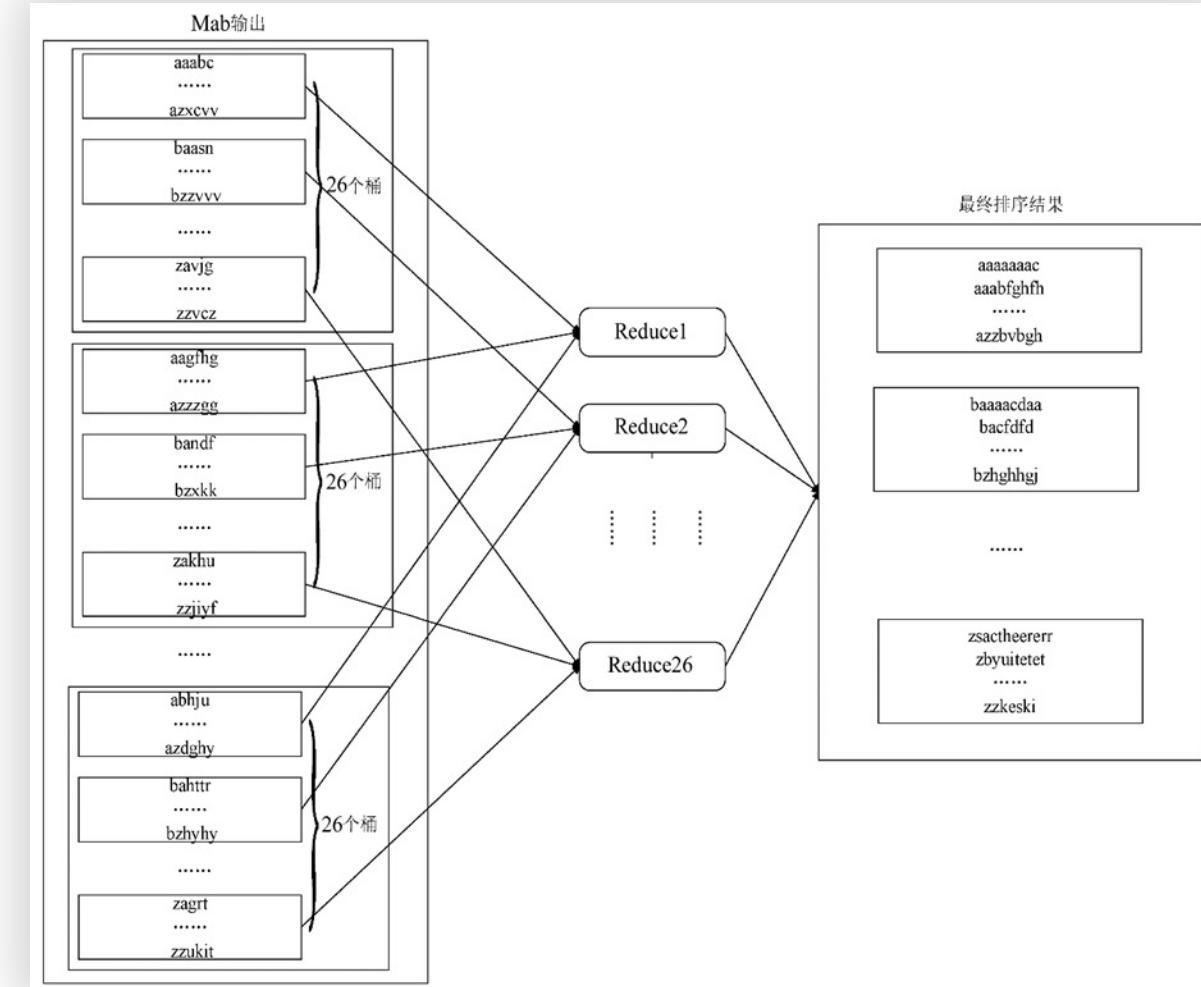
MapReduce案例分析



□ 步骤三

对于Map之后得到的中间结果，启动26个Reduce。

按照首字母将Map中不同桶中的字符串集合放置到相应的Reduce中进行处理。



Apache Spark和Apache Flink



MapReduce是“大数据”早期的关键技术，被用于在普通硬件集群上处理和分析大量数据。

现在，有其他分布式计算框架和数据处理技术变得越来越流行，比MapReduce更适合现代数据处理需求：



Spark和Flink相对于MapReduce的优点



□ 更快的数据处理速度

- Spark和Flink在处理数据时，采用了内存计算的方式，减少了数据从磁盘读取的次数

□ 更灵活的计算模型

- Spark和Flink采用了基于内存的计算模型，支持多种数据处理场景，包括批处理、交互式查询、流处理和图计算等，而MapReduce只适用于批处理场景

□ 更丰富的API和功能

- Spark和Flink提供了更多的API和函数库，这使得用户能够更方便地进行数据处理、机器学习、图计算等任务



中山大學 软件工程学院
SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

谢谢

陈壮彬
软件工程学院

<https://zbchern.github.io/sse316.html>