西安电子科技大学 XIDIAN UNIVERSITY
计算机科学与技术学院 School of Computer Science and Technology

分布式计算

Distributed Computing

主讲人：李龙海

第一讲 概述一

主讲教师信息

- 主讲人：李龙海
- Email：

lhli@xidian.edu.cn,

xdlilonghai@qq.com

- 课程交流群：面对面建群

■ 分布式系统定义

一个分布式系统由多个通过网络互联的独立自治的计算节点组成，这些计算节点基于消息传递机制进行相互协作，以完成共同的目标。

从普通用户角度看，分布式系统是计算节点内聚在一起的一个整体，用户在使用系统功能的时候，往往无法察觉分布式系统的内部构成和节点之间的协作关系。

■ 分布式计算

多个通过网络互联的计算节点通过相互协作共同完成计算任务

分布式系统定义

理解分布式系统定义的几个要点

1. 多个计算节点：

计算节点一般指单个计算机，也可以是计算机中的一个进程、线程或虚拟机。计算节点抽象为有限状态机(图灵机)。

2. 网络互联：

节点之间可以通过有线、无线等任意网络通信方式互联。对物理拓扑结构不做明确限定。

3. 独立自治：

每个节点都有自己独立的CPU、独立的时钟，发生错误的时机和模式也相互独立。并发：不同节点的动作是同时进行的。

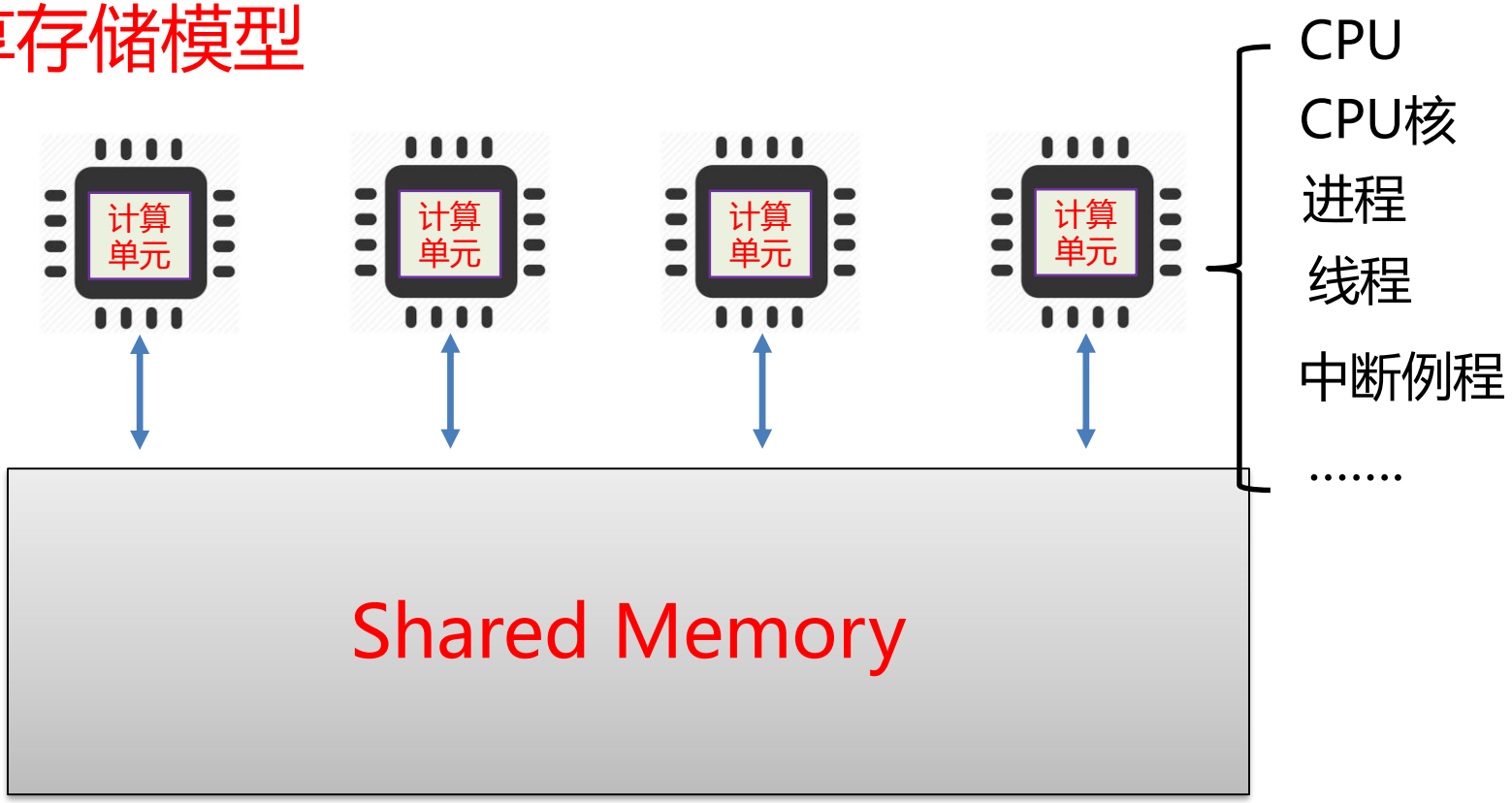
3. 相互协作以完成共同目标

4. 消息传递模型：

消息传递模型，并非内存共享模型

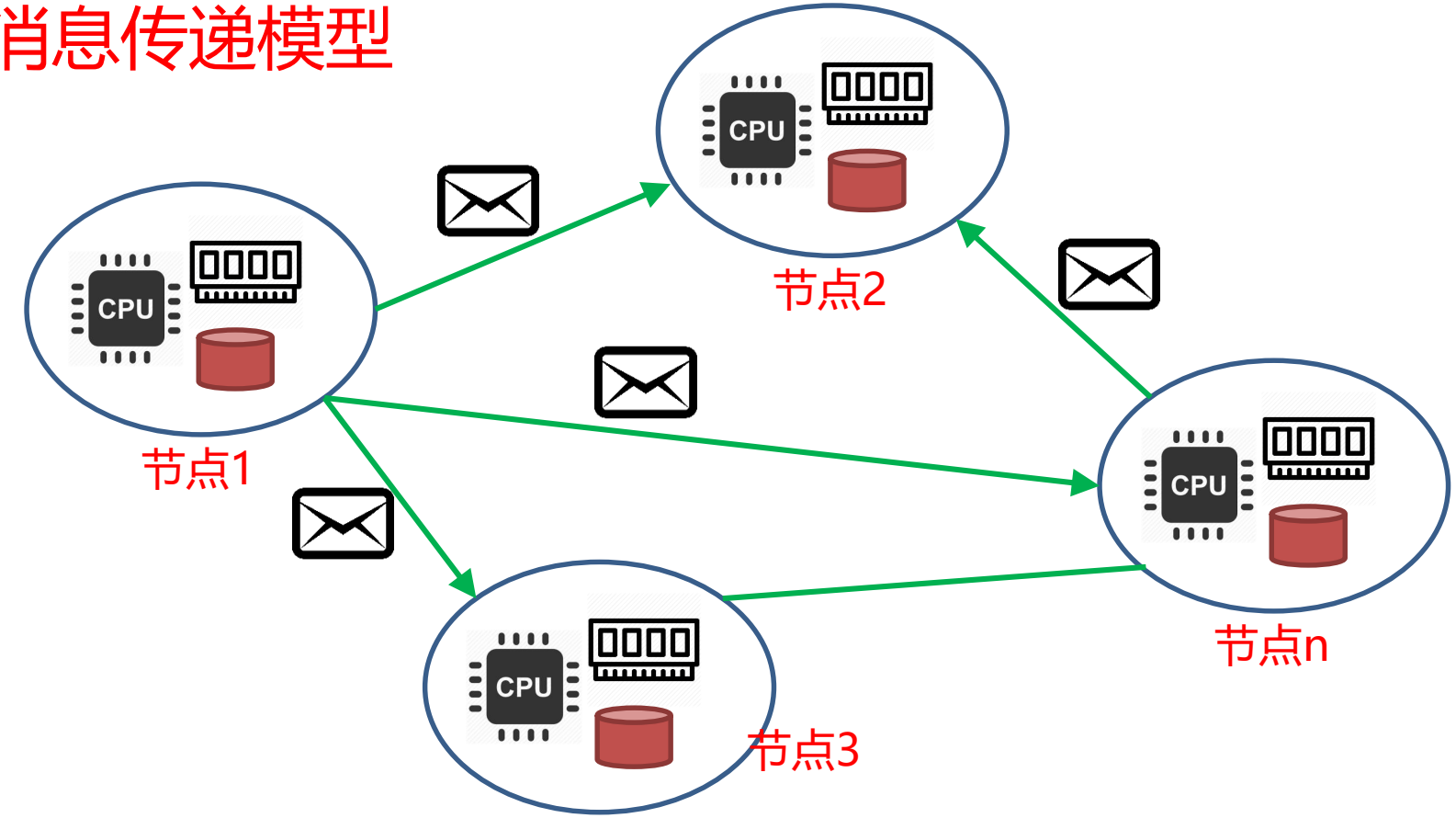
消息传递模型

共享存储模型



1. 不同单元之间共享公共状态；
2. 单元之间通过写-读共享存储中的公共状态来隐式地进行通信；
3. 难点：任务如何划分、不同计算单元的同步与协调、竞争处理(锁、信号量、优先级、死锁）、数据一致性（Serialisability、事务）、可扩展性、如何证明程序的正确性和可靠性

消息传递模型

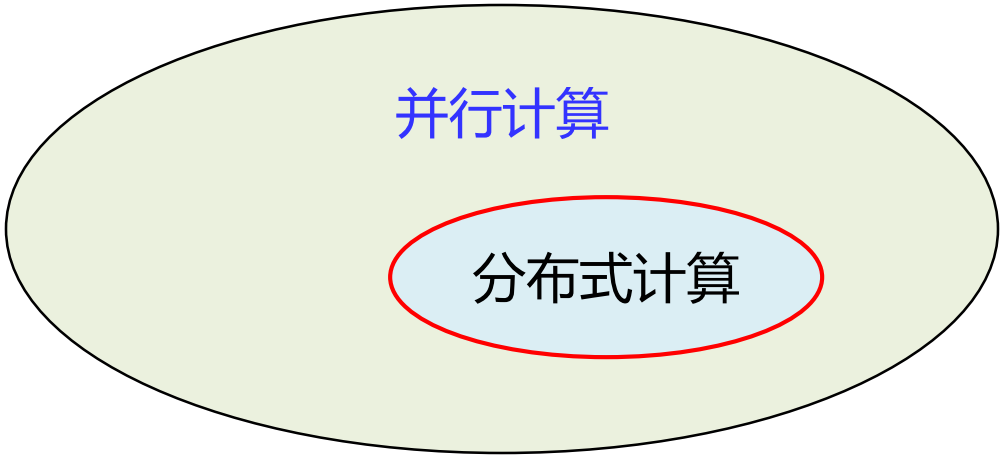


1. 节点之间没有公共状态，必须通过相互发送消息来进行协作。
2. 内部运算速度比消息传递速度高几个数量级。通信复杂度是影响效率的重要因素。
3. 系统设计时必须要应对局部失效、消息延迟/丢失等错误。
4. 也被称为“Share Nothing”架构
5. 难点：共享存储模型所面临的所有难点、消息延迟/丢失的处理、系统局部失效的处理、如何降低通信复杂度

与并行计算的关系

不同层次的并行计算：

- 指令级并行：多指令并行；单指多数并行（向量指令）
- CPU多核并行：多线程编程
- 多CPU并行（一致性内存访问）：多线程编程
- 多CPU并行（非致性内存访问）：超级计算机
- 基于GPU的并行：单指多数并行；CUDA、OpenCL
- 多机并行：基于消息传递的分布式计算（share nothing）



并行与并发

- **并发（Concurrency）：**
  - 多个计算任务在观察者（当前程序眼中）眼里 “似乎在同一个时间段内同时执行”，则称这些任务为“并发任务”
  - 多个并发任务有可能是交替执行，也有可能是真正同时执行
  - 并发程序设计更关注“并发任务管理与调度，如何高效地切换任务”
  - 例子：高并发Web服务器设计、多任务调度系统设计
- **并行（Parallelism）：**
  - 多个计算任务在一个时间段内同时执行
  - 并行可以视为并发的一种特殊情况
  - 并行程序设计更关注“对计算密集型任务的优化，通过分解大任务为多个小任务并同时执行，来加速任务的完成。”
  - 例子：计算密集型任务的并行优化（如图像处理、科学计算、大数据处理）

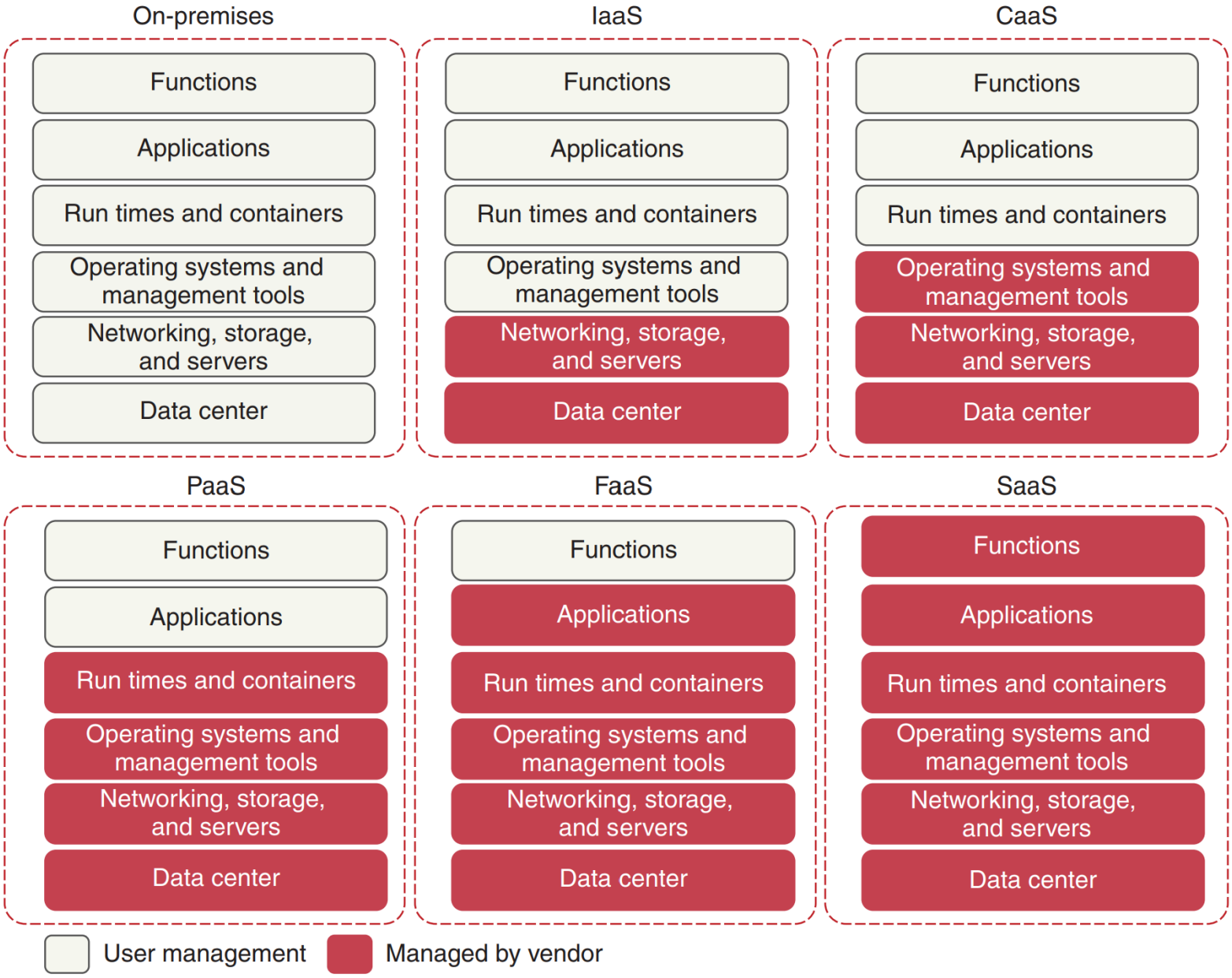
常用的并行计算框架(模型/工具/库)

- 多核并行：OpenMP、Pthreads、Intel TBB、Cilk Plus
- 分布式计算：MPI、MapReduce、Hadoop、Spark
- GPU并行：CUDA、OpenCL
- 图计算：Pregel、GraphX
- 深度学习：TensorFlow、PyTorch、MXNet
- 其他：Celery、Dask

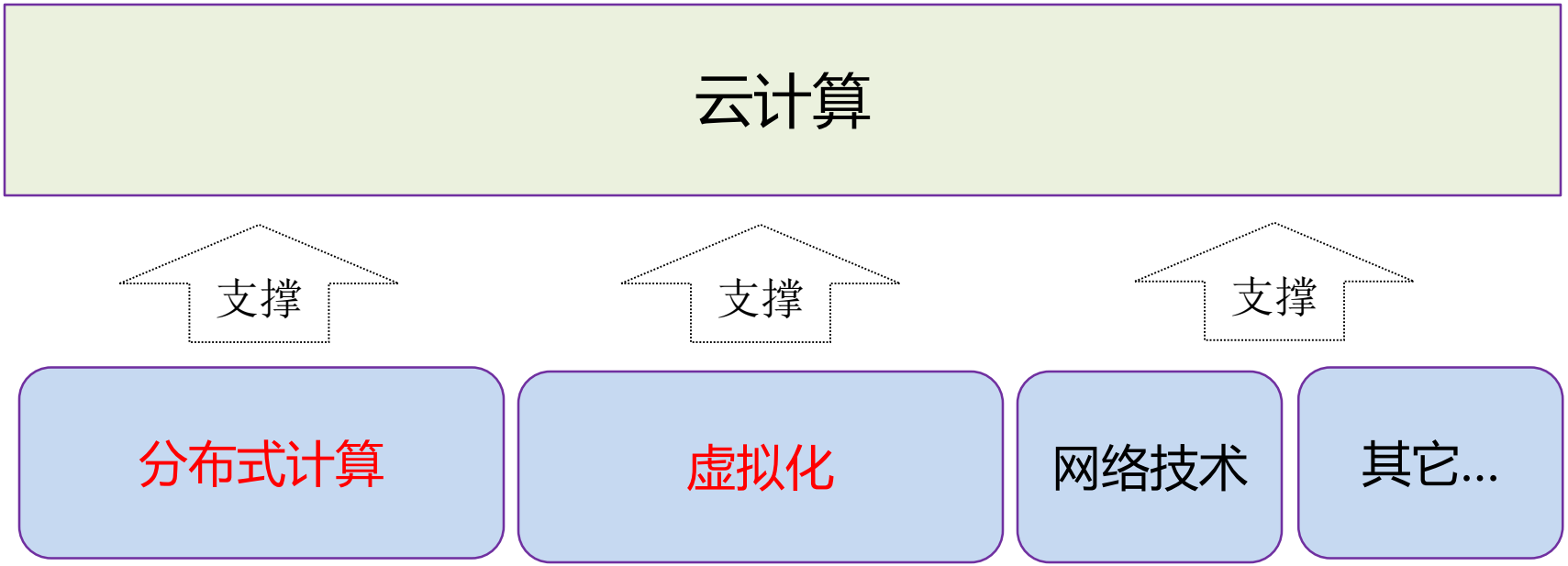
关于云计算

- 云计算是一种计算资源的应用模式/服务模式/交付模式（站在用户角度看），不是一种技术。
- 核心理念是使用户可以随时随地、便捷地、随需应变地从可配置计算资源共享池（云）中获取所需的资源（包括计算、网络、存储、软件应用等资源）。资源能够按需供应、按需释放。
- 共享资源池的管理主要由云服务提供商负责，用户可以专注于自己的业务逻辑。
- 云计算的不同服务模型：
  1. IaaS：基础设施即服务（Infrastructure as a Service）
  2. PaaS：平台即服务（Platform as a Service）
  3. SaaS：软件即服务（Software as a Service）
  4. CaaS：容器即服务（Container as a Service）
  5. FaaS：函数即服务（Function as a Service），Serverless Computing
  6. BaaS（Backend）、DaaS（Data）、NaaS（Network）

不同的云计算模型


On-premises
Functions
Applications
Run times and containers
Operating systems and management tools
Networking, storage, and servers
Data center
IaaS
Functions
Applications
Run times and containers
Operating systems and management tools
Networking, storage, and servers
Data center
CaaS
Functions
Applications
Run times and containers
Operating systems and management tools
Networking, storage, and servers
Data center
PaaS
Functions
Applications
Run times and containers
Operating systems and management tools
Networking, storage, and servers
Data center
FaaS
Functions
Applications
Run times and containers
Operating systems and management tools
Networking, storage, and servers
Data center
SaaS
Functions
Applications
Run times and containers
Operating systems and management tools
Networking, storage, and servers
Data center
User management
Managed by vendor

- 分布式计算是实现云计算的核心技术之一
- 云计算是目标，分布式计算是手段。



分布式系统出现的背景

团结起来，实现更大的胜利

- 大大小小的计算设备无处不在
- 网络通信技术高速发展、网络规模不断扩大
- 摩尔定律走到瓶颈，免费午餐已经结束
- 越来越多的计算任务需要由分布在不同区域的多个计算节点协作完成。

构建分布系统的目的

- 提高计算能力
- 提高存储能力
- 提高网络吞吐能力（并发访问能力）
- 提高可靠性（解决局部失效问题）
- 提高安全性（解决被局部攻击问题）
- 提高可扩展性（解决瓶颈问题）
- 实现资源共享
- 实现跨越时空的协同服务（发挥不同节点的优势）

衡量分布式系统优劣的特性

- 可扩展性/可伸缩性（Scalability）
  - 垂直可扩展性（Vertical Scalability）
  - 水平可扩展性（Horizontal Scalability） （节点热插拔）
- 容错性（Fault Tolerance/Reliability）
  - 可用性（Availability）
  - 可恢复性（Recoverability）
- 一致性（Consistency）
- 透明性（Transparency）
- 开放性（Openness）
- 安全性（Security）
- 可维护性（Maintainability）
- 可观测性（Observability）

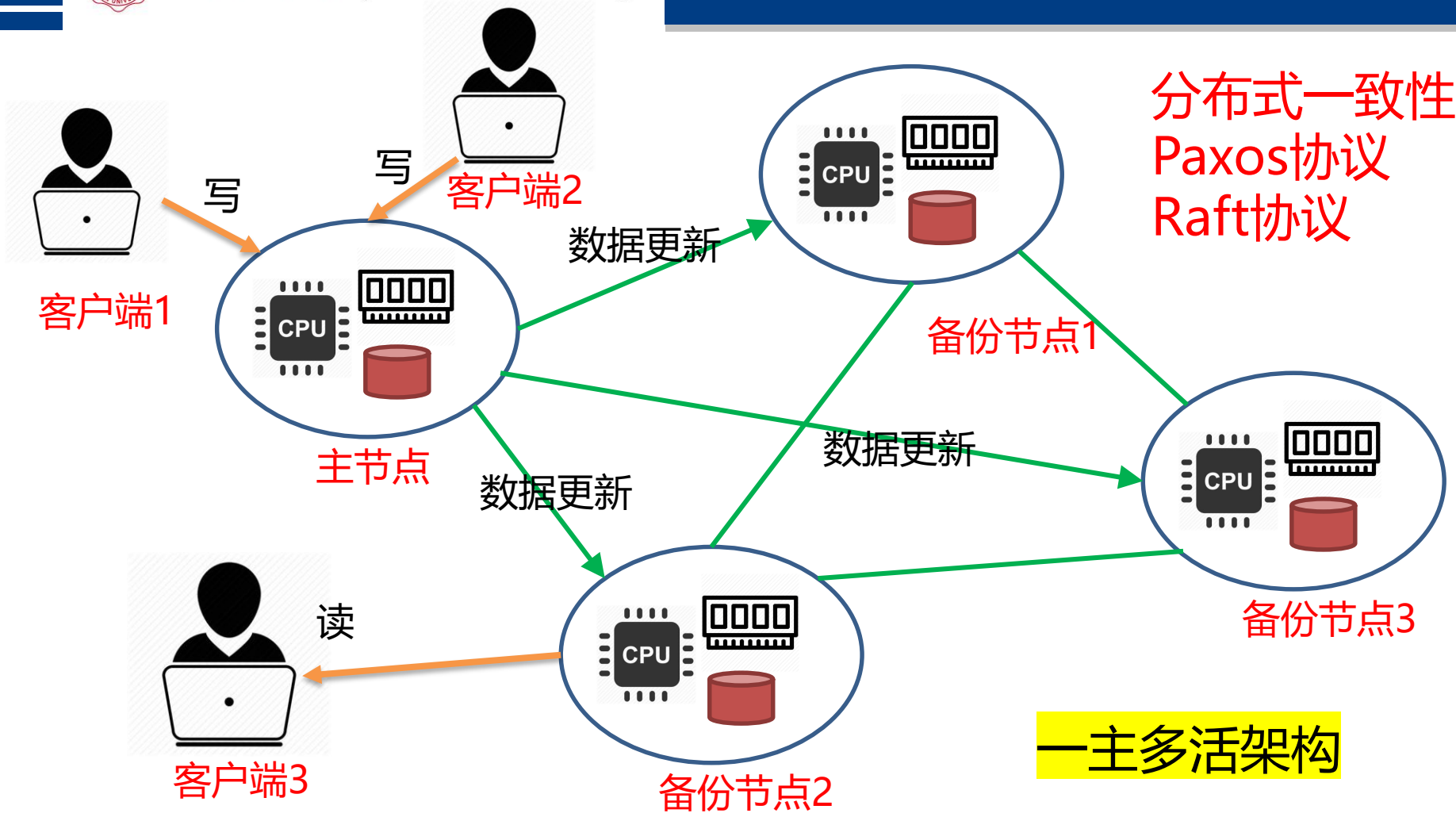
透明性分类

透明性	描述
访问透明性	数据的表示、存储和底层访问方式被隐藏
节点透明性	实际提供服务的物理计算机节点标识被隐藏
迁移透明性	用户感觉不到对象/资源的迁移过程
复制透明性	用户感觉不到对象/资源存在多个副本
并发透明性	多个用户能并发的使用共享资源而互不干扰
伸缩透明性	分布式系统内部节点的增减被隐藏
错误透明性	用户感觉不到分布式系统的局部错误
性能透明性	系统能够自动通过增减资源以适应变化负载变化
移动透明性	用户能够在系统内移动而不会影响到正在使用的功能

设计分布式系统的挑战

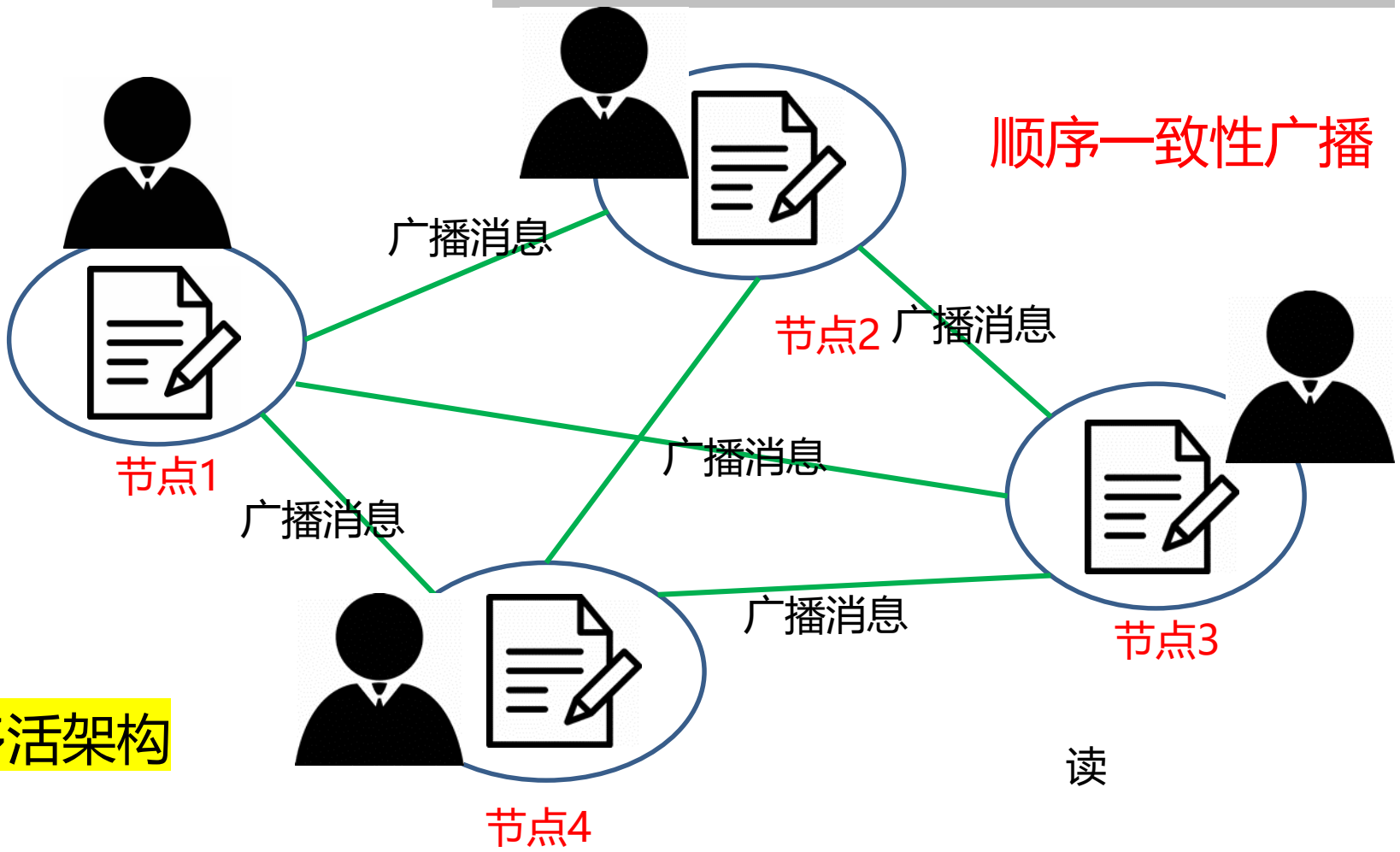
- 自治：各节点有自己独立的时钟、独立的内部状态
- 局部视图：节点只能看到整个系统的某个局部视图
- 故障处理：必须处理网络故障、局部节点故障等
- 开放性：节点数目在变动，网络情况在变动
- 可扩展性：节点增加时性能须合理增长
- 异构性：各个节点的软硬件差异性很大
- 安全性：保密性、完整性、认证性、隐私、可用性
- 透明性：应用层或用户无法察觉位置、并发、复制、故障、移动、伸缩、性能等变化
- 观测问题：运维者如何准确获取各个节点的运行状态和参数
- 维护问题：针对具体节点如何快速上线/下线/升级/更改配置
- 服务质量保证

设计挑战举例：分布式数据库



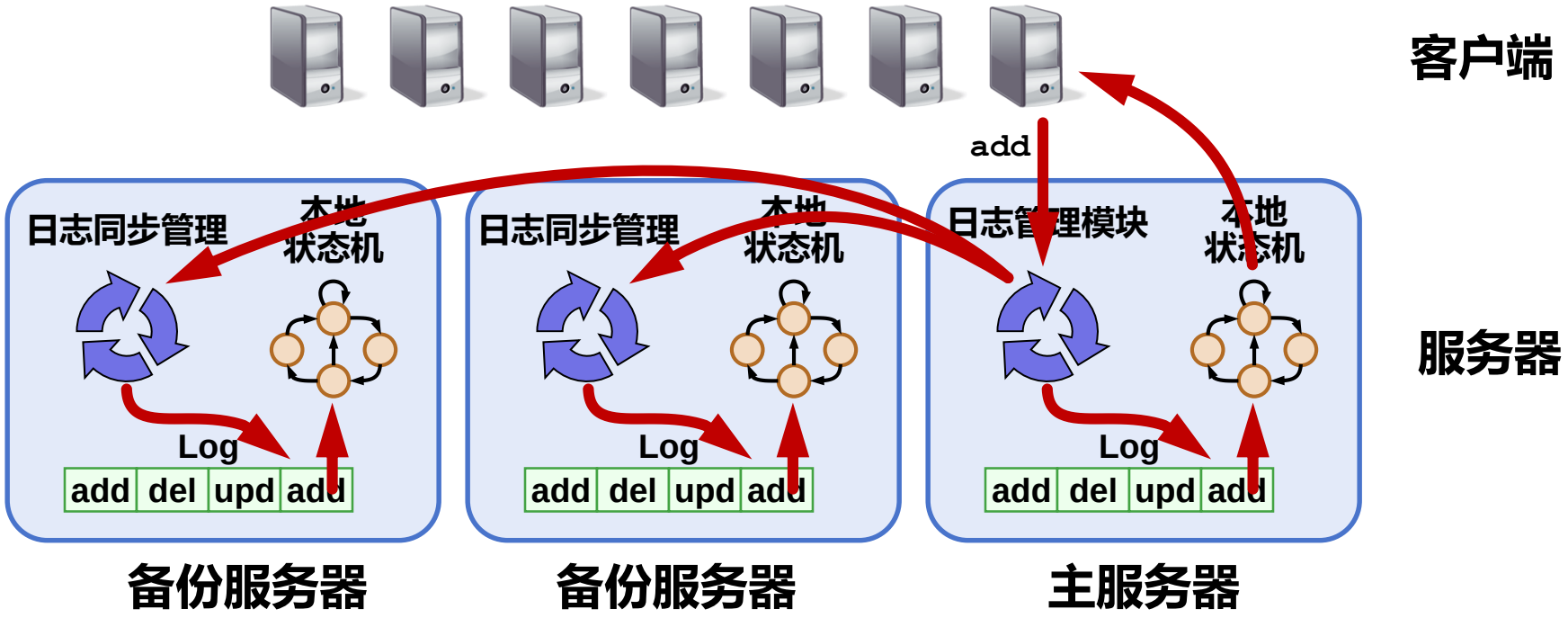
1. 只有主节点数据库接收写入请求。主节点再向各个备份节点发送数据更新命令。
2. 如果主备节点之间状态一致，则客户端可以从任意的节点读取数据。
3. 当主节点失效后，自动切换为某一个备份节点作为主节点。（也要求分布式一致性）

设计挑战举例：协同文档编辑



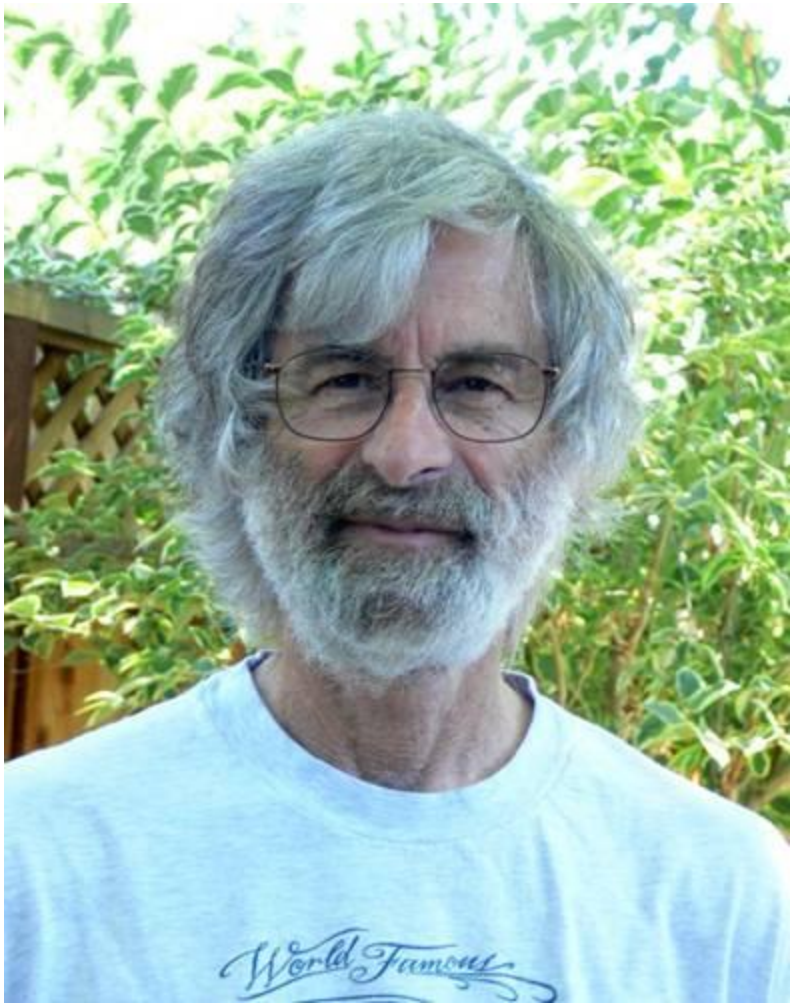
1. 多个用户同时编辑同一个文档。该文档在每个分布式节点上都有一本地备份。
2. 每个节点都把自己对文档的修改及时广播给其他所有节点。
3. 为了保持不同节点上文档的一致性，必须保证：每个节点都接收到了顺序一致、内容一致的广播消息。

挑战举例：复制状态机



1. 如何构建一个“永远在线”的高可用网站?
2. 常用解决方法：多机热备技术（配合动态DNS、负载均衡等技术）
3. 挑战：有状态服务如何处理?

Leslie Lamport



2013年图灵奖获得者

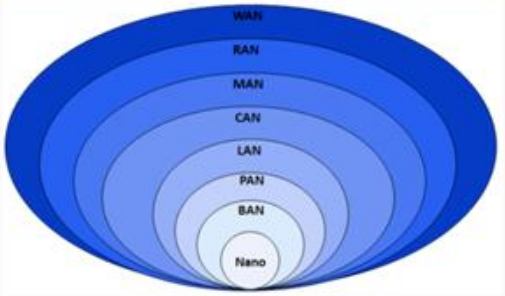
他的分布式计算理论奠定了这门学科的基础，被称为“分布式计算原理之父”

分布式系统的例子

- 实现资源共享的分布式系统（分布式存储）
  - Web系统、DNS系统
  - 网络文件系统：NFS、HDFS
  - P2P资源共享系统：BitTorrent、μTorrent 、eMule
  - 区块链、比特币
- 高性能计算系统：Map-Reduce、Spark、TensorFlow
- 云计算
- 网格计算
- 集群计算
- 分布式信息系统：跨企业应用系统、金融应用系统
- 泛在计算（Ubiquitous Computing）

What is Ubiquitous Computing?

Computer network types by spatial scope



- Nanoscale
- Near-field (NFC)
- Body (BAN)
- **Personal (PAN)**
- Near-me (NAN)
- Local (LAN)
  - Home (HAN)
  - Storage (SAN)
  - Wireless (WLAN)
- Campus (CAN)
- Backbone
- Metropolitan (MAN)
- Wide (WAN)
- Cloud (IAN)
- Internet
- Interplanetary Internet

V • T • E

学习本课程的意义

- 现实生活中分布式系统已经得到了广泛应用
- 近年来的热点技术背后都以分布式系统作为后盾
  - 云计算（边缘计算、雾计算）
  - 物联网
  - 大数据（分布式采集、存储、处理）
  - 人工智能（高性能分布式计算、GPU、分布式机器学习）
  - 区块链
  - 可能没听过的一些热点：SOA、微服务、P2P、网格计算、无处不在计算（Ubiquitous Computing）
  - 大型Web系统：HW/BAT的后台是什么？
- 计算机科学发展到今天，分布式计算应该成为和操作系统同等重要的专业基础课程。
- 分布式计算领域还大有可为

本课程主要学习内容

内家功(理论)

- 分布式系统的基本概念、模型和架构
- 基本思想：负载均衡、复制、切片、一致性、分布式事务
- 基本理论：时钟问题、一致性哈希、CAP理论、BASE理论
- 常用协议：分布式共识（Paxos、Raft）、Leader选举、失效检测、时钟同步、互斥、全序广播、Gossip等
- 分布式算法设计

外家功

- 分布式系统中的通信技术：RPC、RestfulAPI、消息中间件
- 容器与微服务
- 分布式存储系统：HDFS、HBase、NoSQL
- 分布式数据处理：MPI、Map-Reduce、Spark 、流处理
- 分布式协同平台：ZooKeeper/etcd

预备知识

- 操作系统（进程、线程、虚拟内存管理、API与系统调用、用户态、内核态、文件系统、时钟、中断）
- 计算机网络
- Java编程语言（优先）（Go、 Python、C++）
- 网络编程（Socket编程）

考核方式

- N次编程作业 20%
- 随堂实验（编程大作业） 20%
- 期末考试 60%

课下作业

自学如下内容：

- Java Socket编程
- Java多线程编程、Java线程池编程
- Maven系统的原理和使用方法
- Java NIO编程（多路复用、事件驱动）
- Java 虚拟线程