计算思维与计算科学

分布式思维

第八章 分布式思维

目录

1、并发原理与编程
2、单机的能力扩展
3、计算机网络
4、并发程序设计
5、分布式系统

如何充分发挥和提升计算系统的能力

一、并发的原理与编程

- 基本概念
  - 并行与并发
  - 同步和异步
- 中断
- 进程与线程
- 资源管理与死锁

并行和并发

- 并发(Concurrency)
  - 多个任务在同一时间段共同执行
  - 宏观
  - 时间段相互重叠
- 并行(Parallelism)
  - 多个事件在同一时间点同时发生
  - 微观

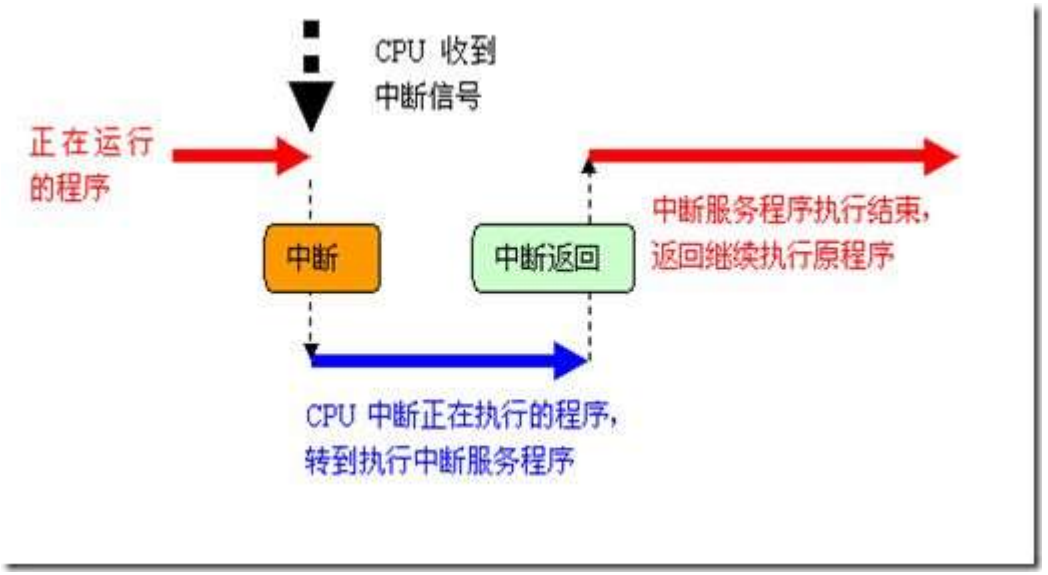
同步和异步

- 同步：多个协同工作的部件按照同一节拍，或按照可预测的时间点工作，可以随时交互
- 异步：多个协同工作的部件按照自身的节拍，交互时需要判断对方状态

- 同步时序电路 vs 异步时序电路
- 同步通信 vs 异步通信
- 同步设备接口 vs 异步设备接口
- 同步网络协议 vs 异步网络协议

中断

- 中断
  - CPU运行过程中，出现某些情况时，能自动中止正在运行的程序，并转入特定的处理新情况的程序，处理完毕后又返回原被暂停的程序继续运行
- CPU与外部慢速设备
  - 等待外部设备的响应
  - 查询外部设备的状态



中断的来源

- 硬中断
  - 外部中断：如时钟中断、输入输出设备中断
  - 硬件故障中断：如电源故障、内存错误
- 软中断
  - 异常：如浮点溢出、内存越界、用户态下使用了特权指令等
  - 访管中断：对操作系统提出某种请求时所发生的中断
- 可屏蔽中断、不可屏蔽中断

中断的作用

- 提高系统运行效率（CPU与外部设备并行）
  - 在外部I/O设备工作时，CPU可以处理其他事情，当I/O完成就绪后发出中断，CPU进行处理
- 提供实时处理
  - 当中断产生时，不管当前执行的是什么程序，都可以得到及时处理
- 进行系统调度
  - 结合时钟中断、软中断，进行现代OS的进程调度工作
- 系统故障检测和处理

中断的执行过程

1. 外部产生中断信号
2. 在每个指令的某一个时钟节拍，检查是否有中断，当中断发生，并允许中断时，进行处理
3. 自动保存基本的中断现场（PC、PSW）
4. 根据中断的来源信息，使用特定方式得到中断服务程序的起始地址，自动转到中断服务程序执行
5. 中断服务程序在程序头，要保存全部的中断现场（所有用到的内部寄存器值）
6. 中断服务程序运行结束前，恢复保存的中断现场
7. 根据之前保存的产生中断处的PC值，返回原程序

操作系统

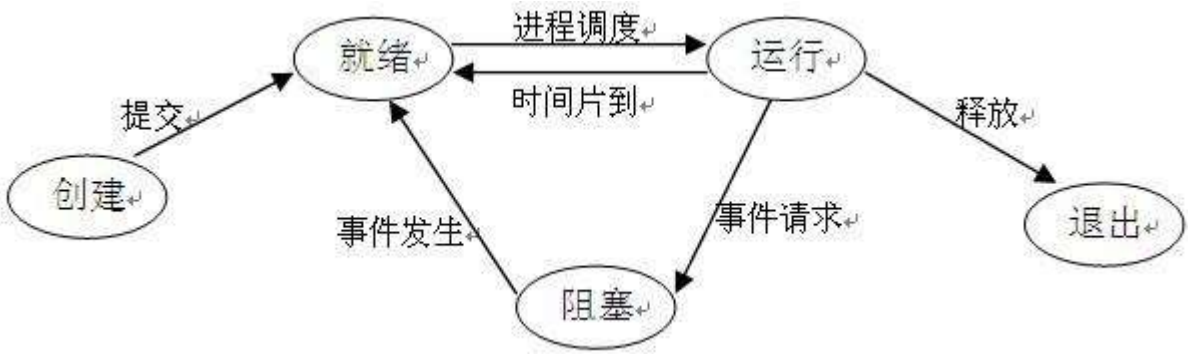
- 操作系统（Operation System，简称OS）是管理计算机硬件与软件资源的计算机程序
  - 进程管理
  - 存储管理
  - 设备管理
  - 文件管理
- 分类
  - 批处理系统、分时系统、实时系统、嵌入式系统
  - 单用户、多用户

进程

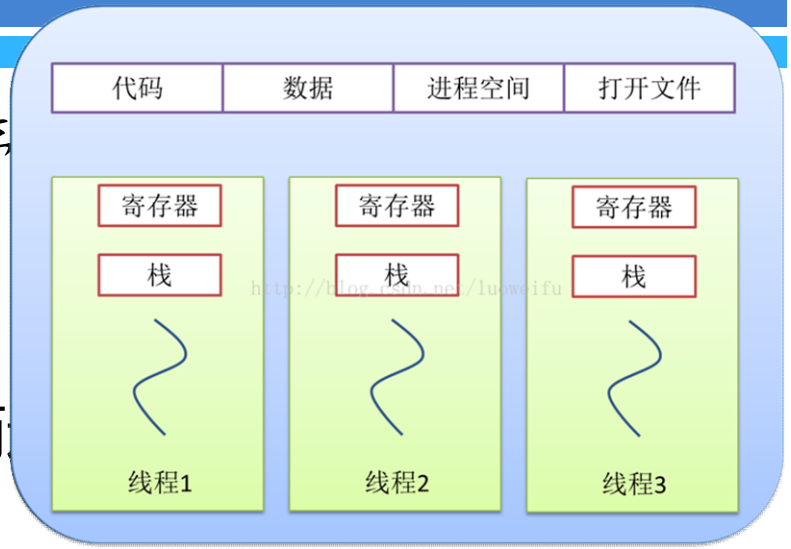
- 进程：应用程序在操作系统调度下的一次执行
- 进程与程序的关系
  - 进程是动态的，有生命周期，程序是静态的
  - 进程是资源竞争的单位
  - 一个程序可以有多个进程
  - 进程可以创建子进程
- 进程之间是相互隔离的，不能够直接访问，必须通过OS提供的进程间通信机制来交互

进程调度

- 进程的状态
  - 运行态 Running
  - 就绪态 Ready
  - 阻塞态 Blocked



线程



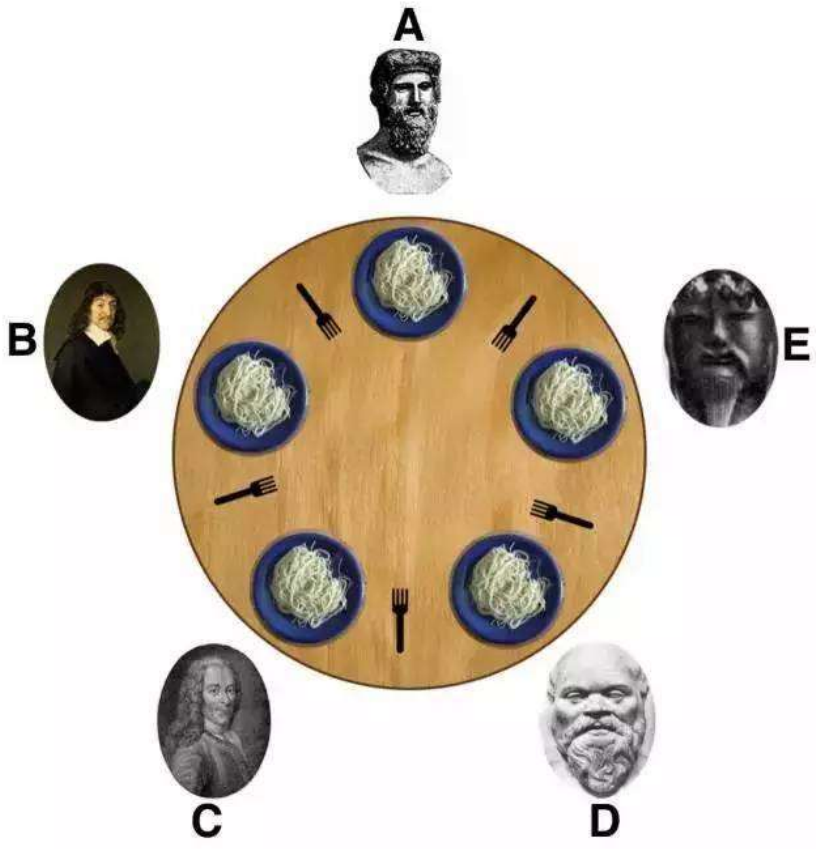
- 线程是进程内一个单一的程序
  器调度和分派的基本单位
- 进程与线程的关系
  - 1.线程是程序执行的最小单位，而
    资源的最小单位;
  - 2. 一个进程由一个或多个线程组成，线程是一个进程中代码的不同执行路线;
  - 3. 进程之间相互独立，但同一进程下的各个线程之间共享程序的内存空间(包括代码段，数据集，堆等)及一些进程级的资源(如打开文件和信号等)；
  - 4. 调度和切换：线程上下文切换比进程上下文切换要快得多

资源管理

- 各种可被多个任务使用的对象，如外部设备、文件、内存等
  - 共享性资源
  - 独占性资源（竞争性资源）
- 每个任务对于竞争性资源应当按照规则访问
  - 访问前获得许可（锁）
  - 完成访问之后要释放资源
  - 不应该长期占用资源
- 在进程结束后，OS会自动回收该进程申请的资源

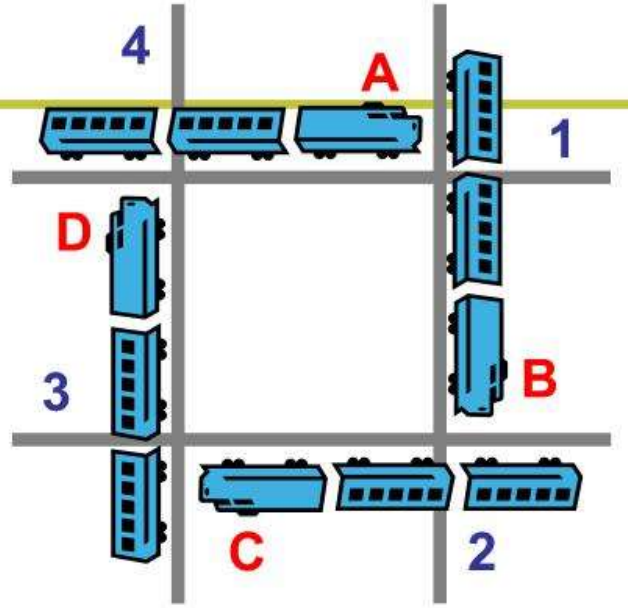
死锁（dead lock）

- 一组任务中，每个任务都持有一部分资源，同时试图申请被其他任务占据的资源，所有进程处于无限等待中，这种状态称为死锁
- 多个任务、多个资源

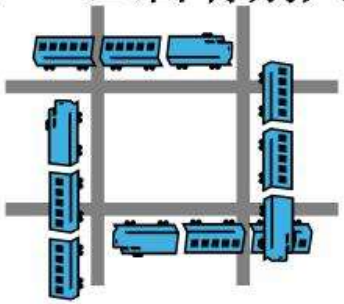


死锁

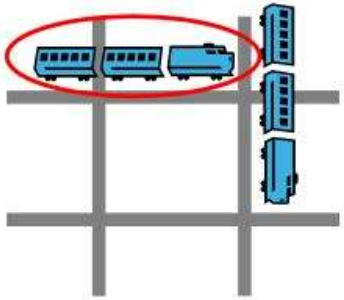
死锁的成因



- 资源互斥使用，一旦占有别人无法使用



- 进程占有了一些资源，又不释放，再去申请其他资源



- 各自占有的资源和互相申请的资源形成了环路等待

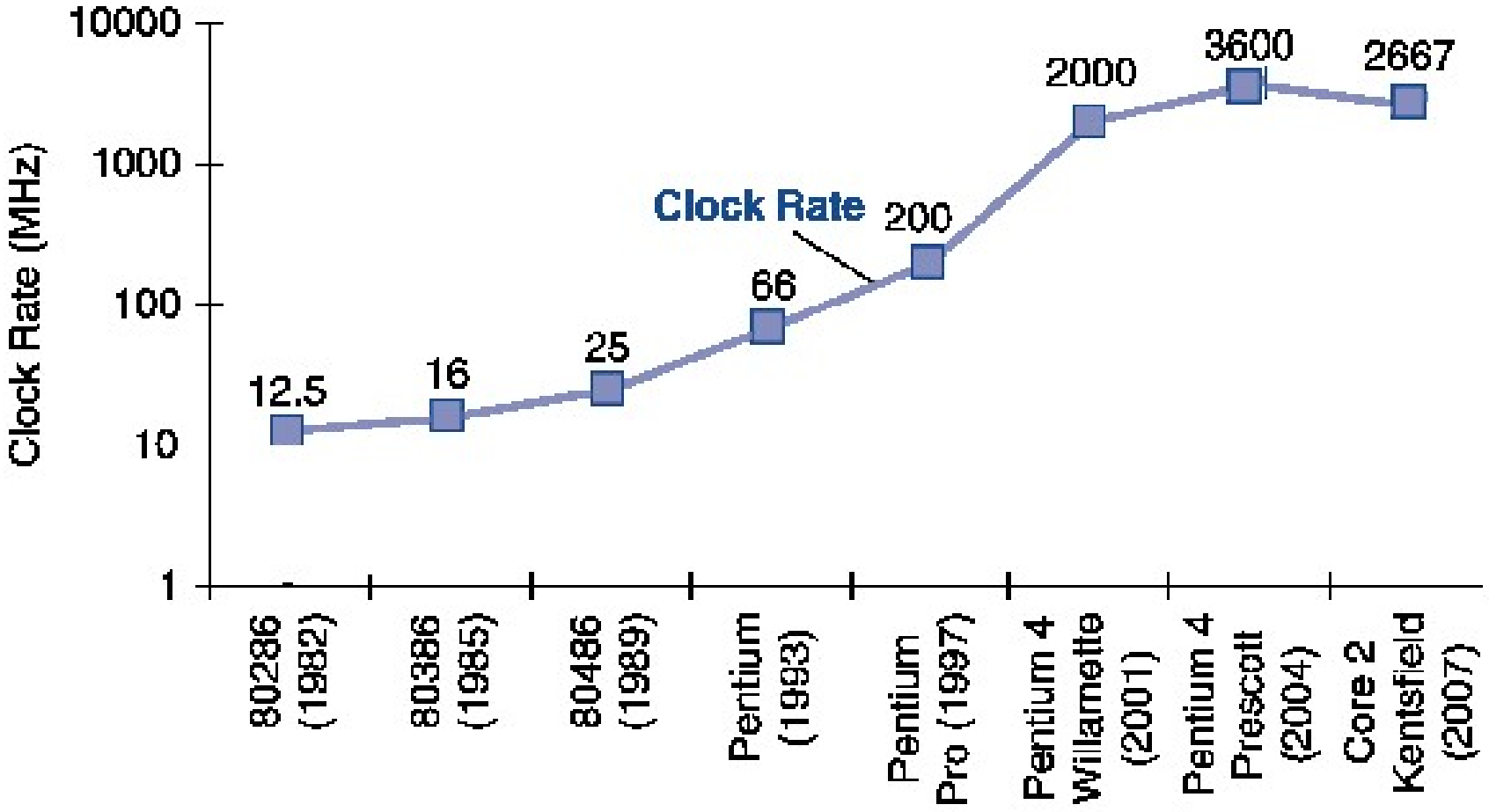
死锁的解决方案

- 预防死锁
  - 从机制上避免死锁的出现
- 避免死锁
  - 采取合适的资源分配方案，防止进入不安全状态
- 检测死锁
  - 检测资源分配图中的环路
- 解除死锁
  - 撤销任务、剥夺资源、进程回退、重新启动

二、单机的能力提升

- 单核CPU能力提升的途径
  - 提升主频、字长
  - 内存Cache
  - 指令流水
  - 单CPU的性能瓶颈
- 多核与多CPU
- 异构计算

提高处理器频率

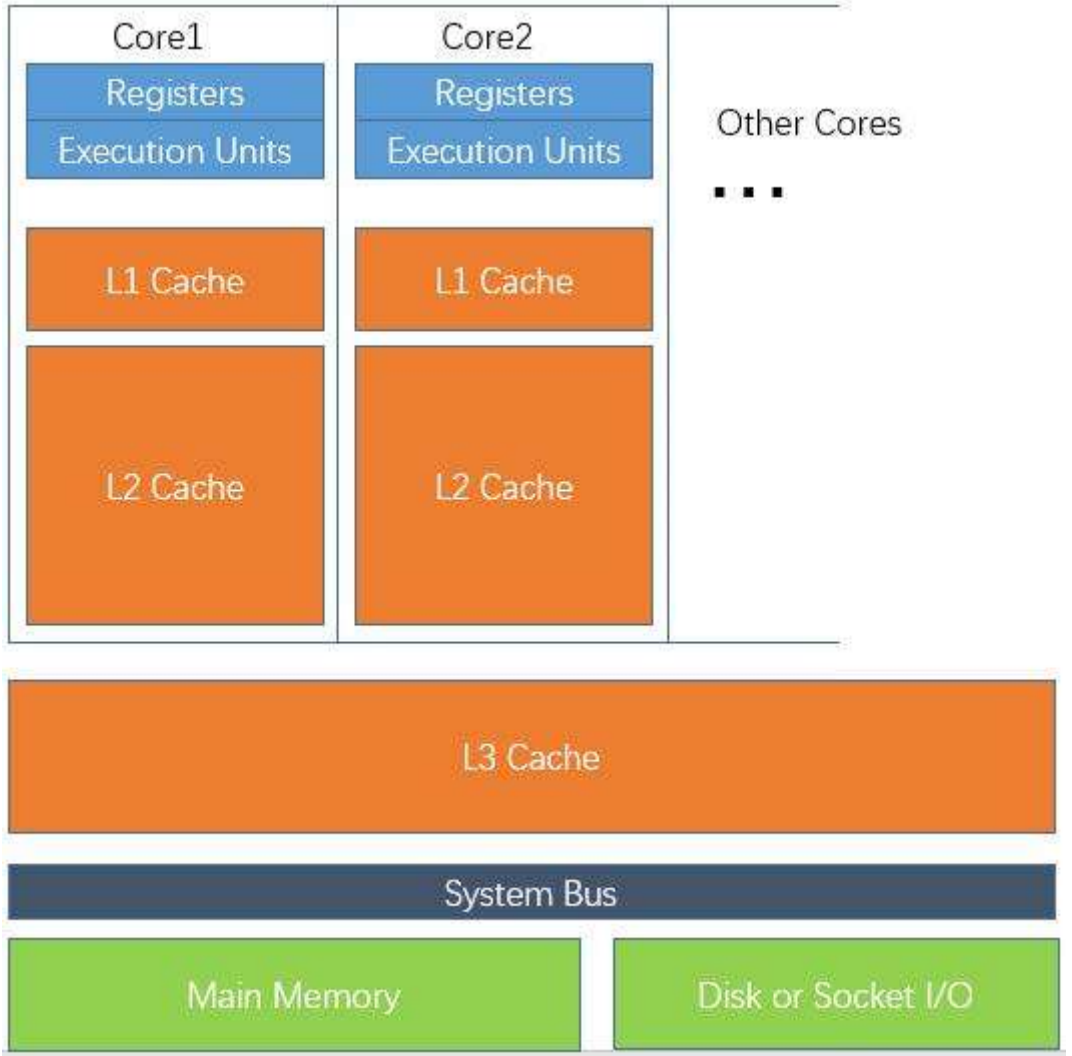

Clock Rate (MHz)
10000
1000
100
10
1
Clock Rate
12.5
16
25
66
200
2000
3600
2667
80286 (1982)
80386 (1985)
80486 (1989)
Pentium (1993)
Pentium Pro (1997)
Pentium 4 Willamette (2001)
Pentium 4 Prescott (2004)
Core 2 Kentsfield (2007)

提高处理器字长

年代	Intel代表CPU	字长
1971	4004	4
1978	8086	8
1982	80286	16
1985	80386，80486，Pentium	32
2005	Pentium D，Core	64

匹配慢速内存

- CPU主频远远高于内存
  - Register：1ns
  - L1Cache：1ns
  - L2Cache：3ns
  - L3Cache：12ns
  - DRAM：65ns
- 为适应不同部件的访问速度，在快速部件中加入慢速部件的映像
- Cache更新和写回算法



指令流水

- 一条指令的执行包括多个时钟节拍，每个时钟节拍的动作可以由一个功能部件完成
- 串行执行指令：
  - 每一个指令顺序地完成每个时钟节拍的工作，分别由不同的功能部件执行
  - 每个功能部件只在特定的时钟节拍工作，其他阶段空闲
- 流水执行指令
  - 只要功能部件完成当前指令的工作后，在下一个节拍，就执行下一条指令的相应工作，而不用等到当前指令的所有阶段执行完毕
  - 最佳情况下，所有节拍，所有功能部件同时工作

指令流水示意图

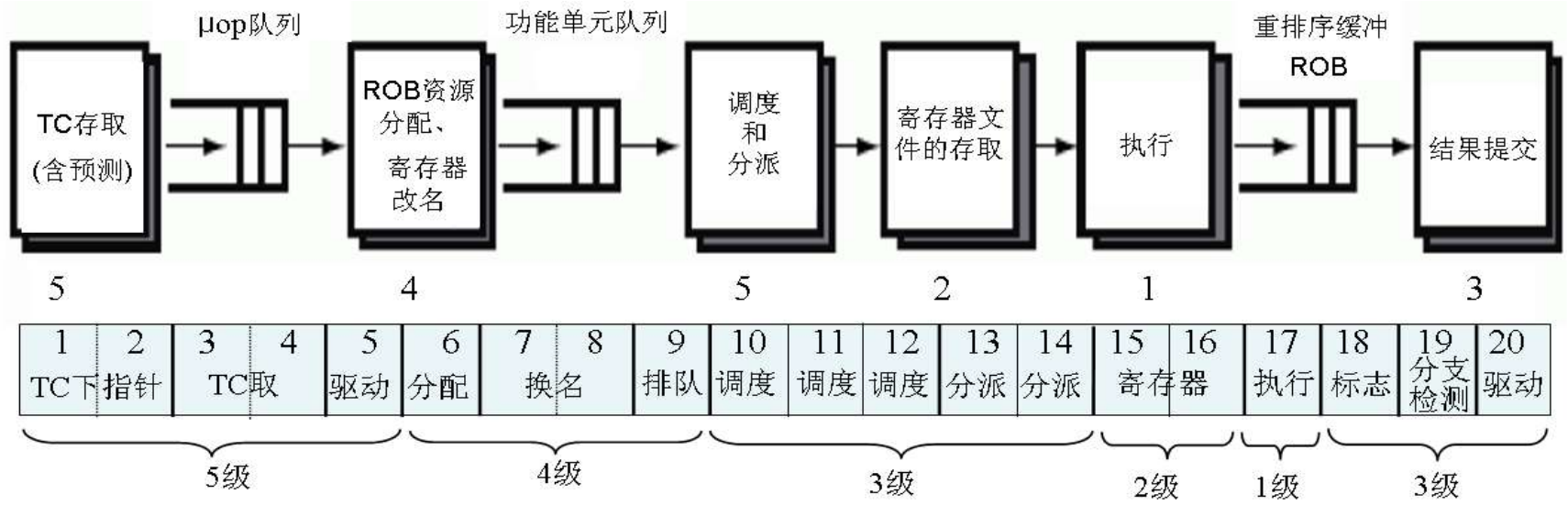
图 5　ARM9 的五级最佳流水线

指令流水的特点

- 指令级并行(Instruction-Level Parallelism, ILP)
- 多条指令同时并行运行，占用不同的内部资源
- 并没有缩短每条指令的执行时间，但提高了整个系统的吞吐率
- 连续不断的指令才能更好地提高运行效率
  - 结构相关：内部资源冲突
  - 数据相关：访问同一数据（寄存器、Cache、内存）
  - 控制相关：分支指令

流水线越发复杂

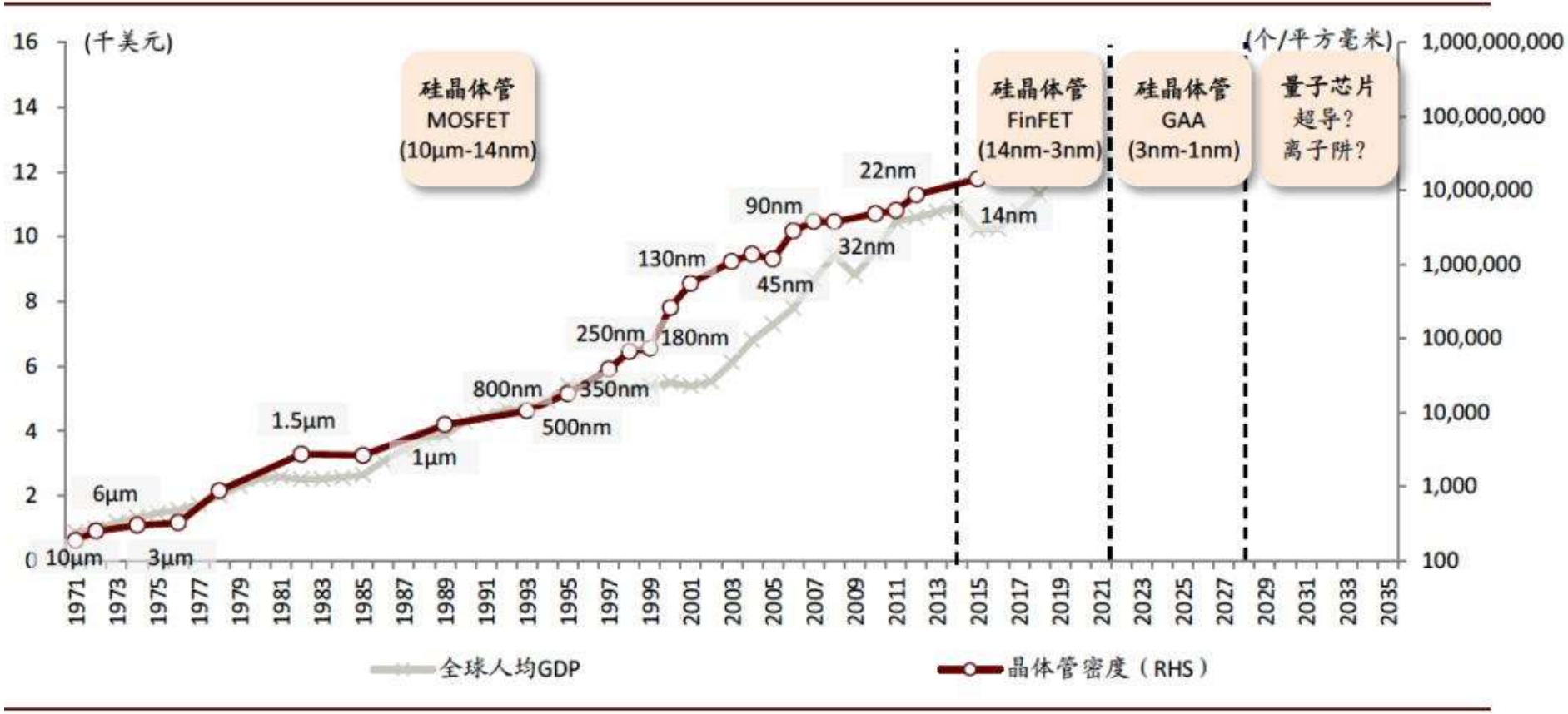
- 分支预测,寄存器重命名，超长指令字（VLIW），超标量(Superscalar)，乱序执行，Cache……
- Pentium 4(CISC结构）采用了20级复杂流水



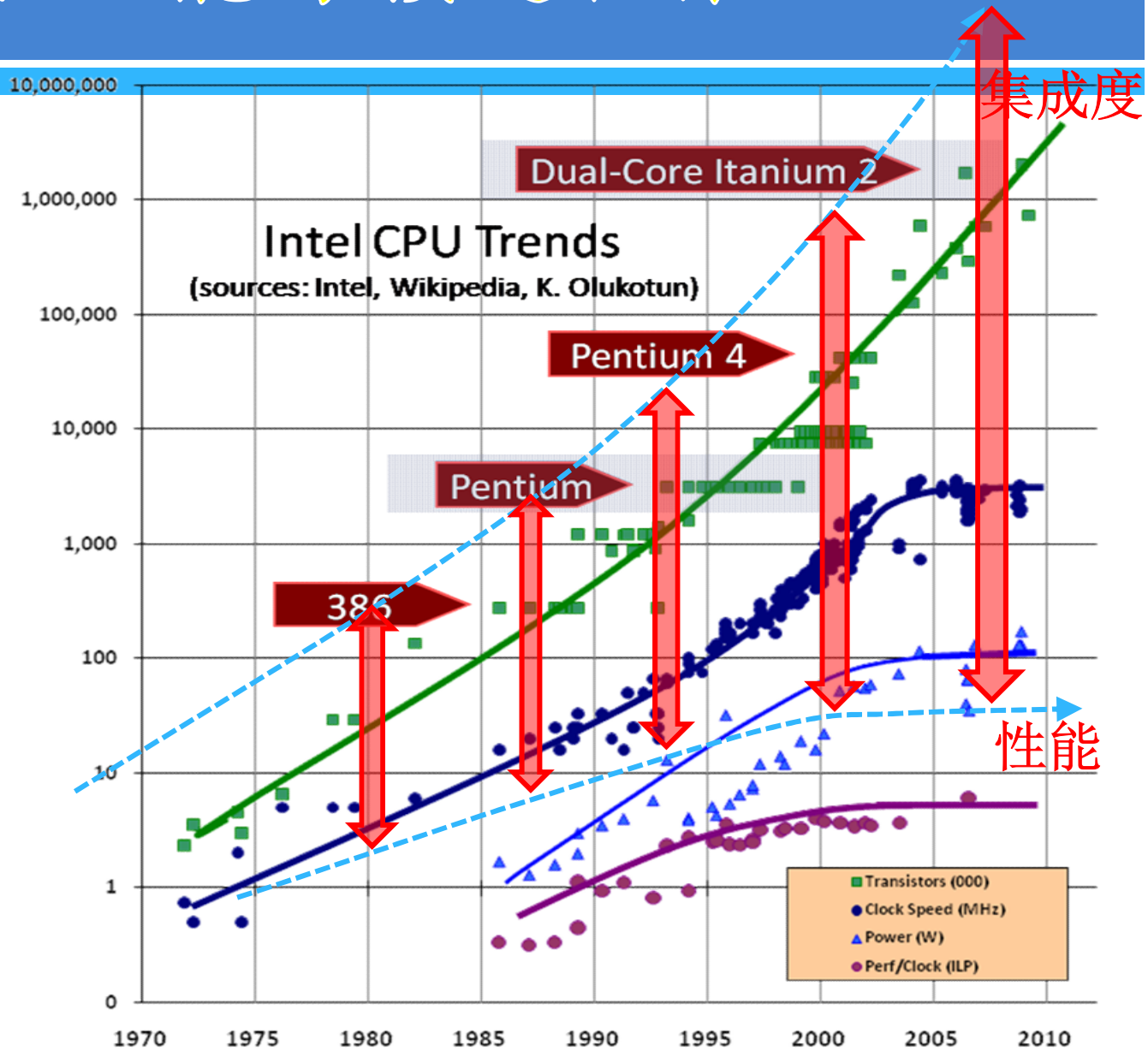
提高集成度

- 集成电路的发展
  - 1947年，晶体管诞生，BCS，贝尔实验室
  - 1958年，集成电路诞生，仙童公司
  - 1963年，发明CMOS工艺
  - 1971年，Intel 4004， 10μm工艺，集成度：2.25k
  - 1978年，VLSI，64kRAM，集成度：14万
  - 1989年，Intel486，1μm工艺，集成度：120万
  - 1999年，奔腾3，0.25μm工艺
  - 2003年，奔腾4，90nm工艺
  - 2012年，酷睿3，22nm工艺，集成度： 26亿
  - 2020年，麒麟9000，5nm工艺，集成度： 153亿

摩尔定律


(千美元)
(个/平方毫米)
硅晶体管
MOSFET
(10μm-14nm)
硅晶体管
FinFET
(14nm-3nm)
硅晶体管
GAA
(3nm-1nm)
量子芯片
超导?
离子阱?
10μm
6μm
3μm
1.5μm
1μm
800nm
500nm
350nm
250nm
180nm
130nm
90nm
45nm
32nm
22nm
14nm
全球人均GDP
晶体管密度（RHS）

单核CPU能力接近极限



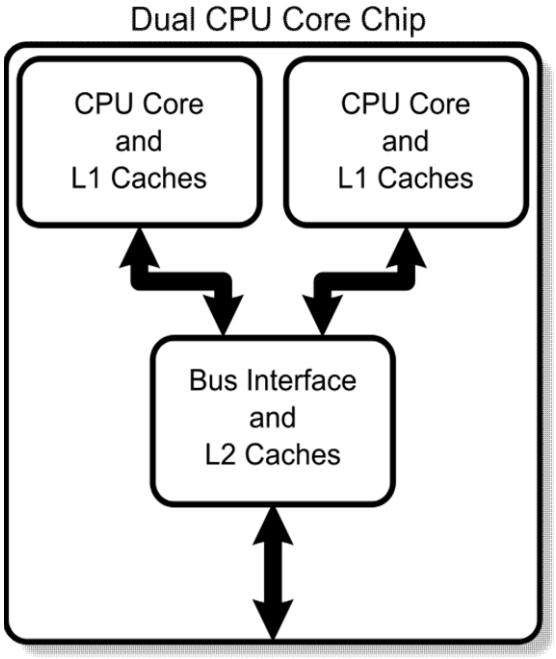
单核处理器性能提升接近极限！

单核CPU能力接近极限

1. VLSI集成度不可能无限制提高，目前已经接近量子极限
2. 处理器的指令级并行度提升接近极限
3. 处理器速度和存储器速度差异越来越大
   - 存储器速度已经比处理器速度慢2个数量级
   - 内部Cache越来越大
4. 功耗和散热大幅增加超过芯片承受能力
   - 处理器功耗与主频的3次方成正比

多核与多CPU

- 简化单处理器的复杂设计，代之以单个芯片上设计多个简化的处理器核，以多核/众核并行计算提升计算性能
- 双核、4核、8核…
- 多CPU：
  - 在一个计算机内安装多个CPU



GPU

- 图形处理的大部分需求，数据之间是独立的
- GPU含有大量计算单元，可对每个数据进行独立的并行计算
- 控制电路相对简单，而且对Cache的需求较小

异构计算

- 能协调地使用性能、结构各异的计算部件，满足不同的计算需求，使代码能以获取最大总体性能方式来执行的体系结构
- CPU：作为控制核心，可以完成复杂的控制逻辑
- GPU：适于大范围、多任务的简单运算
- ASIC：直接构建在物理硬件上的程序逻辑
- FPGA：现场可编程门阵列，可以将通用芯片定制为特定逻辑的计算设备

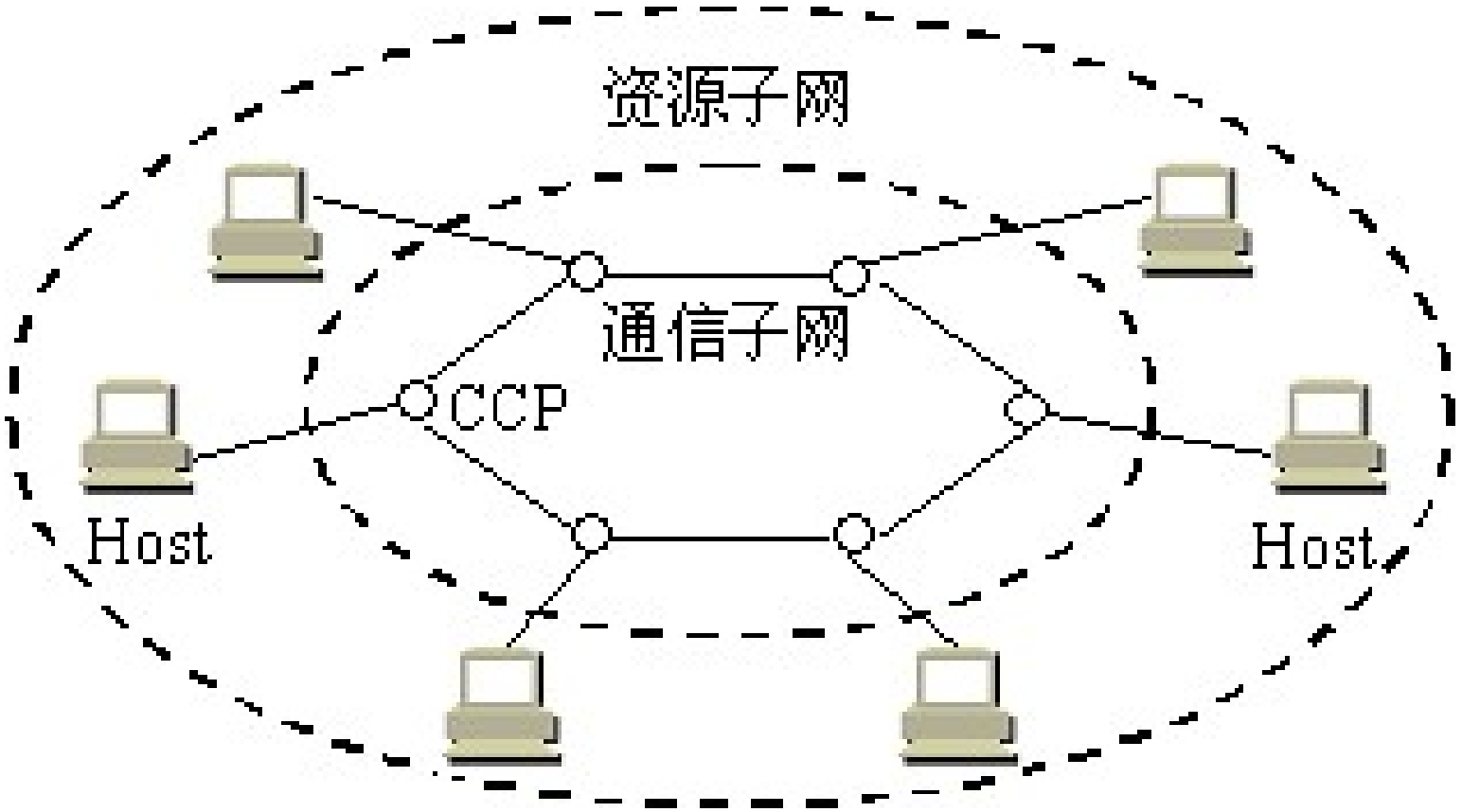
三、计算机网络

- 基本概念
- 网络协议
- 命名与转换
- 网络通信编程

计算机网络的定义与组成

- 计算机网络：
  - 指把若干台地理位置不同，且具有独立功能的计算机，用通信线路和通信设备互相连接起来，以实现彼此之间的数据通信和资源共享的一种计算机系统。
  - 通信技术与计算机技术的结合是产生计算机网络的基本条件。一方面，通信网为计算机之间的数据传输和交换提供了必要的手段；另一方面，计算机技术的发展渗透到通信技术中，又提高了通信网的各种性能。
- 计算机网络的组成
  - 一个计算机网络是由资源子网和通信子网构成。资源子网负责信息处理，通信子网负责网中的信息传递。

计算机网络的组成



数据通信

- 数据通信是指数字计算机或其它数字终端设备之间通过通信信道进行的数据交换。
- 数据通信的质量有两个最主要的指标。
  - 数据传输速率，用“比特位/秒”（b/s）来表示；
  - 误码率，它表示一段时间内接收到的错误比特数与转输的总比特数之比。
- 按照信道中传输的是模拟信号还是数字信号，可以相应地把通信信道分为模拟信道和数字信道。

数据交换

- 在数据通信系统中，网络中的两个设备之间需要经过中间节点转发数据，这种中间节点参与的通信称为交换
- 电路交换
- 分组交换
- 将计算机网络看作图
  - 终端：度很小的节点
  - 交换设备：度比较高的节点
  - 通信线路：边

网络传输协议

- 网络协议（Protocol），是使网络中的通信双方能顺利进行信息交换而双方预先约定好并遵循的规程和规则。
- 一个网络协议主要由以下三个要素组成：
  - 语义：规定通信双方彼此“讲什么”；
  - 语法：规定通信双方彼此“如何讲”；
  - 同步：语法同步规定事件执行的顺序。

分层次的体系结构

- 分层模型（Layering model）：是一种用于开发网络协议的设计方法。采用在协议中划分层次的方法，把要实现的功能划分为若干层次，较高层次建立在较低层次基础上，同时又为更高层次提供必要的服务功能。
- 分层的好处：高层次只要调用低层次提供的功能，而无需了解低层的技术细节；只要保证接口不变，低层功能具体实现办法的变更也不会影响较高一层所执行的功能。

开放系统互联参考模型（OSI/RM）

应用层	• 根据网络数据的语义，满足用户的需要
表示层	• 确定在两个通信系统中交换信息的表示方式
会话层	• 建立、组织和协调两个互相通信的应用进程之间的交互
传输层	• 向上一层提供可靠的端到端（End-to-End）服务，确保“报文段（Segment）”无差错、有序、不丢失、无重复地传输。
网络层	• 完成网络中主机间“分组”(Packet)交换，从点对点的传输变成网络传输
数据链路层	• 在通信的实体之间建立数据链路连接，传送以“帧（Frame）”为单位的数据，采用差错控制、流量控制方法，使有差错的物理线路变成无差错的数据链路。
物理层	• 利用物理传输介质为数据链路层提供物理连接，以便透明地传送比特流

协议分层结构的优点

- 1. 各层之间相互独立，复杂程度下降。
- 2. 结构上可分隔开：各层都可以采用最合适的技术来实现。
- 3. 易于实现和维护：系统已被分解为若干个相对独立的子系统。
- 4. 灵活性好：一层发生变化其他各层不受影响。
- 5. 能促进标准化工作：每一层的功能及所提供的服务都有精确的说明。

TCP/IP协议族

OSI模型	TCP/IP	代表协议
应用层	应用层	HTTP、FTP、SMTP
表示层		
会话层		
传输层	传输层	TCP、UDP
网络层	网络层	IP、ICMP
数据链路层	网络接口层	以太网、无线以太网
物理层		

IP地址

- IP地址可以视为网络上的门牌号码，使用它，可以唯一地得到主机所在的局部网络和网络中的位置
- IPv4地址由32位二进制数组成，为了表示方便一般写成由4个十进数构成，每个十进制数取值0到255
- 为解决IPv4地址不够的问题，IPv6地址由128个bit构成,提供更好的路由方法、更好的安全性，移动连接更加方便

因特网（Internet）

- 是一个由全世界许许多多的网络互联组成的一个网络集合
- Internet使用TCP/IP协议族
- 重要的应用包括
  - 万维网WWW（World Wide Web）
  - 电子邮件（E-mail）
  - 远程登录（Telnet、SSH）
  - 文件传输（FTP）等等

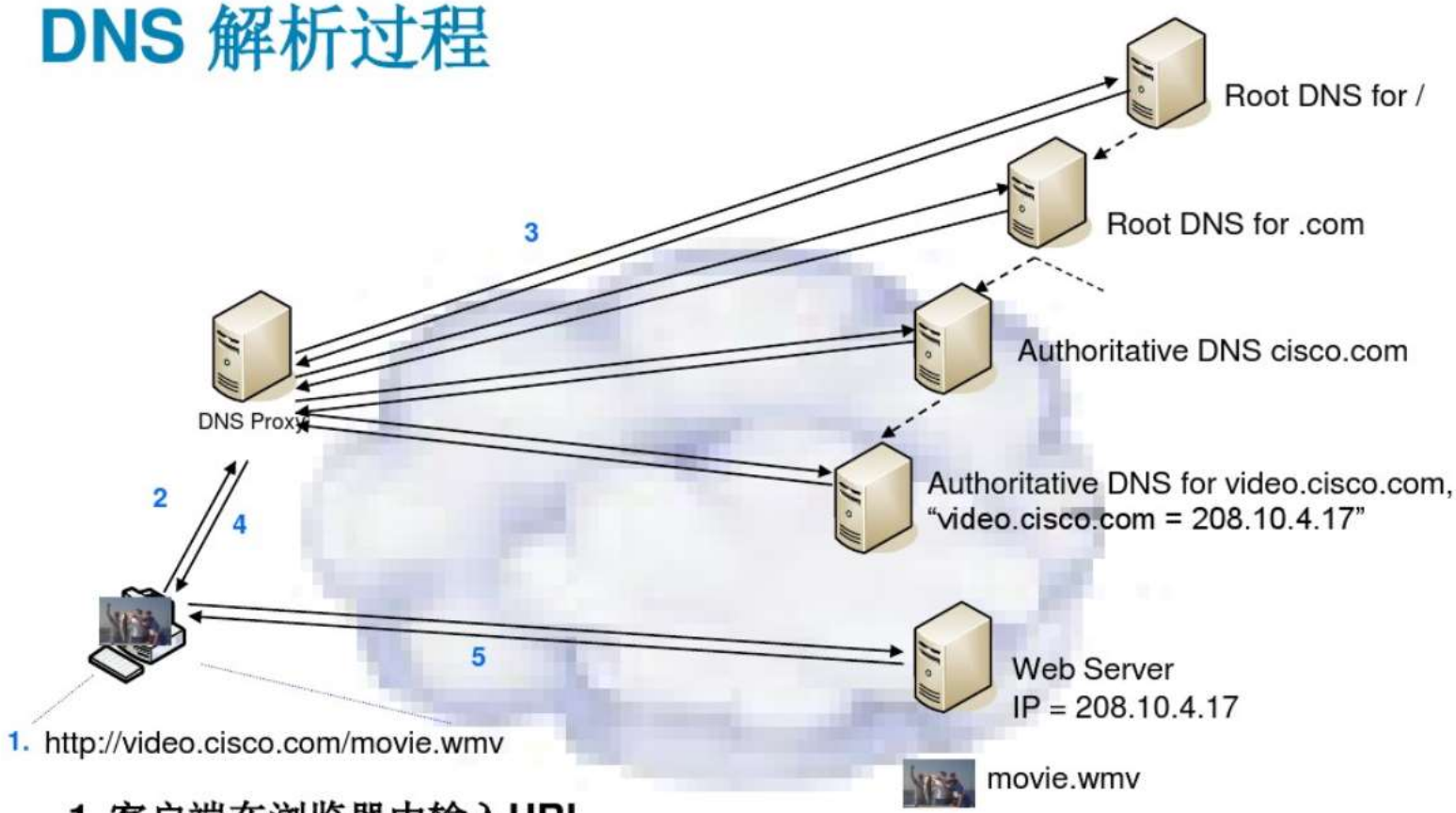
命名与转换

- 在每一层，为了将信息从源发送到目的地，可能需要一个编码来对不同的端进行区分，称为地址或名字
  - MAC地址（数据链路层）、IP地址（网络层）、域名（应用层）
- 在同一层传输时，需要一些方法找到目标名字的通信路线
- 上一层协议调用下一层时，一般需要将上一层名字转换为下一层的名字

域名系统

- 域名（Domain Name）：由一串用点分隔的名字组成的Internet上某一台计算机或计算机组的名称，用于在数据传输时对计算机的定位标识
- 对Internet上主机的命名，一般必须考虑以下问题：
  - 主机名字在全局的惟一性，即能在整个Internet上通用
  - 要便于管理
  - 便于映射
- 分级域名，从右往左
  - WWW.JLU.EDU.CN
- 访问Internet上的主机时，既可以使用域名，也可以直接使用IP访问
- 在使用域名时，需要通过DNS协议，将域名转换为对应的IP地址

DNS 解析过程



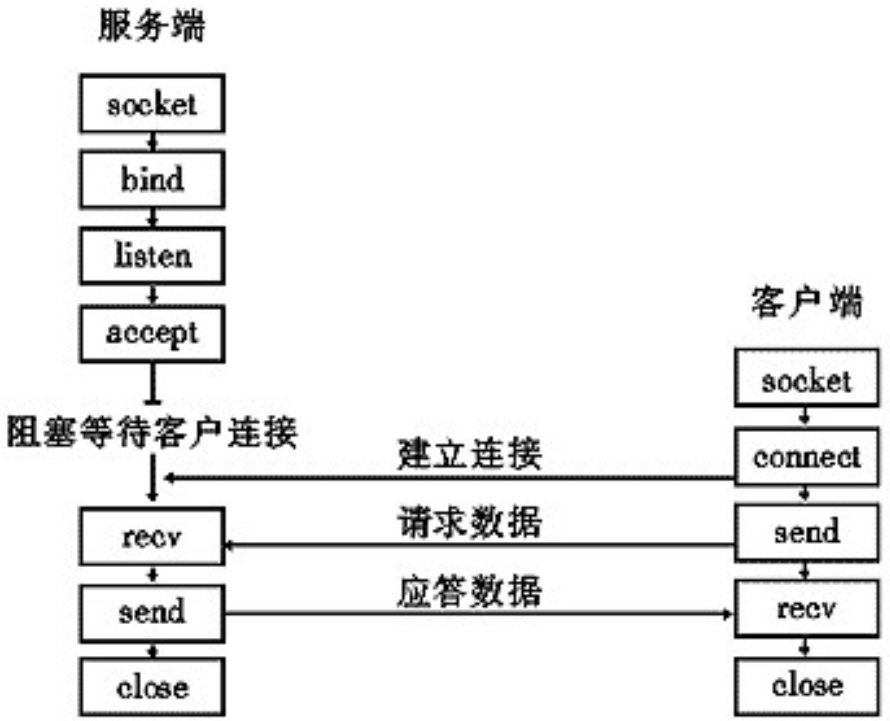
1. 客户端在浏览器中输入URL.
2. 浏览器向DNS代理服务器发送递归查询请求，请求解析 video.example.com.
3. DNS代理服务器向其他授权服务器发送迭代请求，直到返回一个A类记录
4. DNS代理服务器向客户端返回一个A类记录.
5. 浏览器向 208.10.4.17 发送HTTP请求，请求 "movie.wmv"

DNS服务器的层次和缓存

- 全球根服务器
  - 1台主根服务器，美国
  - 12台辅根服务器，9台在美国，欧洲2台，日本1台
  - 镜像服务器：1342台
  - 企业服务器
  - 局域网服务器
- 缓存
  - 局部服务器缓存
  - 本地缓存

网络通信编程

- 客户/服务器模型
  - 传输层之上
  - 面向域名/IP:端口
- Socket编程



四、并发程序设计

- 并发编程简介
- 互斥和临界区
- 并发I/O编程
- 多线程并发编程
- Cuda编程
- 大规模并发编程框架

并发编程简介

- 基本概念
  - 加速比
- 硬件体系结构
  - I/O并发
  - 单机：多线程/多进程
  - 异构计算：GPU + Cuda
  - 多机：基于网络通信
- 低耦合任务并发

互斥与临界区

- 在多线程和多进程情况下，多个任务（进程或线程）并发执行
- 多个任务会访问同一个资源（内存、文件、外部设备…）
- 任务之间的相对执行速度是不可预测的
- 可能出现各种情况，即并发异常

互斥与临界区

- 为了避免出现并发异常，保证程序并发执行的正确性
- 临界资源：一次只允许一个任务访问的资源
- 临界区：一个任务中，完整操作临界资源的代码片段

- 互斥：多个任务因访问同一个临界资源，而相互制约、相互等待的过程和机制

互斥的执行原则

- 不能假设各个任务的相对执行速度
- （一个临界资源）任意时刻最多只能有一个任务处于临界区
- 在临界区外的任务不能干扰其他任务进入临界区
- 当没有任务处于临界区时，想进入临界区的任务能够立即进入
- 不能出现试图进入临界区的任务被无限推迟的情况

实现互斥的方法

- 指令级
  - 关中断
  - Test And Set指令
- 操作系统级
  - 锁
  - 信号量
  - 管程
- 高级机制
  - 消息队列等模型

并发I/O编程

- I/O的特点和分类
  - 直接I/O vs 操作系统管理的I/O
  - 共享性I/O vs 独占性I/O
  - 带缓存的I/O
- 并发I/O编程
  - 轮询式
  - 阻塞式
  - 事件响应式
  - 异步IO

并发I/O编程模式总结

模式	同步/异步	运行时间	CPU占用	线程
轮询	同步	计算+I/O	多	单
阻塞	同步	计算+I/O	少	单
事件响应	异步	Max(计算，I/O)	少	单
异步I/O	异步	Max(计算，I/O)	少	多

多线程并发编程

- 线程安全
  - 可以在多线程环境下被同时执行，调用者无需其他操作，不会带来安全问题
  - 全局变量、静态变量的耦合
- 分类：
  - 类
  - 数据结构
  - 函数（可重入）

Cuda编程

- Nvidia推出的GPU并行计算编程框架
- 使用C语言来为支持CUDA架构的GPU编写程序
- 基本流程
  - 将数据拷贝到GPU
  - 切分成多个数据块（与GPU的线程数匹配）
  - 同时执行多个线程的处理
  - 将结果拷贝回内存

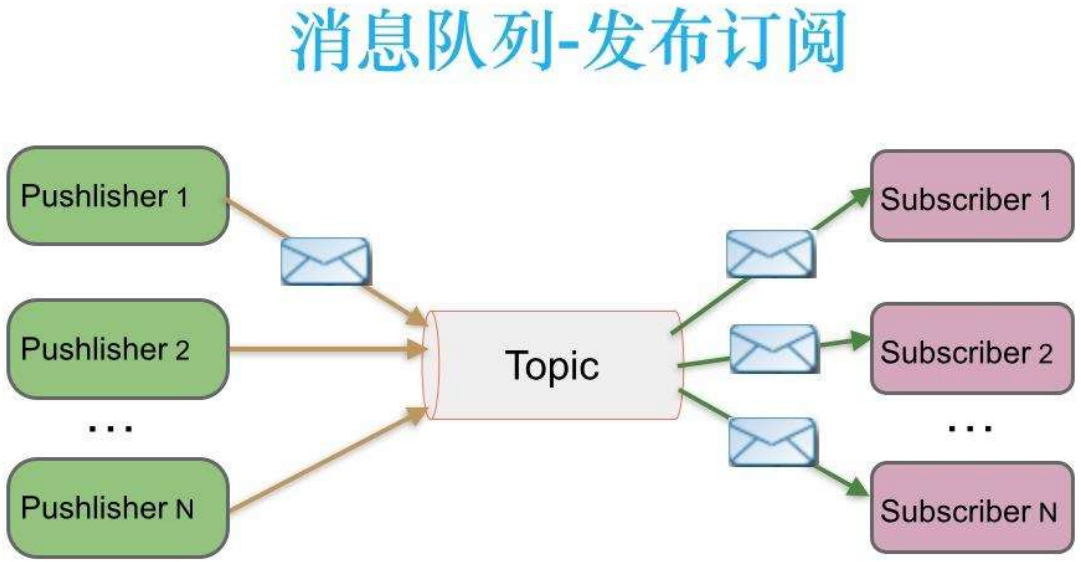
```
__global__ void Plus(float A[], float B[], float C[], int n){
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    C[i] = A[i] + B[i];
}

int cuda_plus(){
    float*A, *Ad, *B, *Bd, *C, *Cd;
    int n = 1024 * 1024;
    int size = n * sizeof(float);
    // 初始化数组A,B
    // GPU端分配内存
    cudaMalloc((void**)&Ad, size);
    cudaMalloc((void**)&Bd, size);
    cudaMalloc((void**)&Cd, size);
    // CPU的数据拷贝到GPU端
    cudaMemcpy(Ad, A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(Bd, B, size, cudaMemcpyHostToDevice);
    // 定义kernel执行配置，（1024*1024/512）个block，每个block里面有
512个线程
    dim3 dimBlock(512);
    dim3 dimGrid(n/512);
    // 执行kernel
    Plus<<<dimGrid, dimBlock>>>(Ad, Bd, Cd, n);
    // 将在GPU端计算好的结果拷贝回CPU端
    cudaMemcpy(C, Cd, size, cudaMemcpyDeviceToHost);
}
```

Map-Reduce

- 总体思路：
  - Map：将待处理的数据集分解成许多个小的数据集
  - 并行：每一个小数据集都可以完全并行地进行处理
  - Reduce：将小数据集的处理结果进行合并
- 所有的数据交换都是通过 MapReduce框架自身去实现
- 使用分布式文件系统保存数据

消息队列

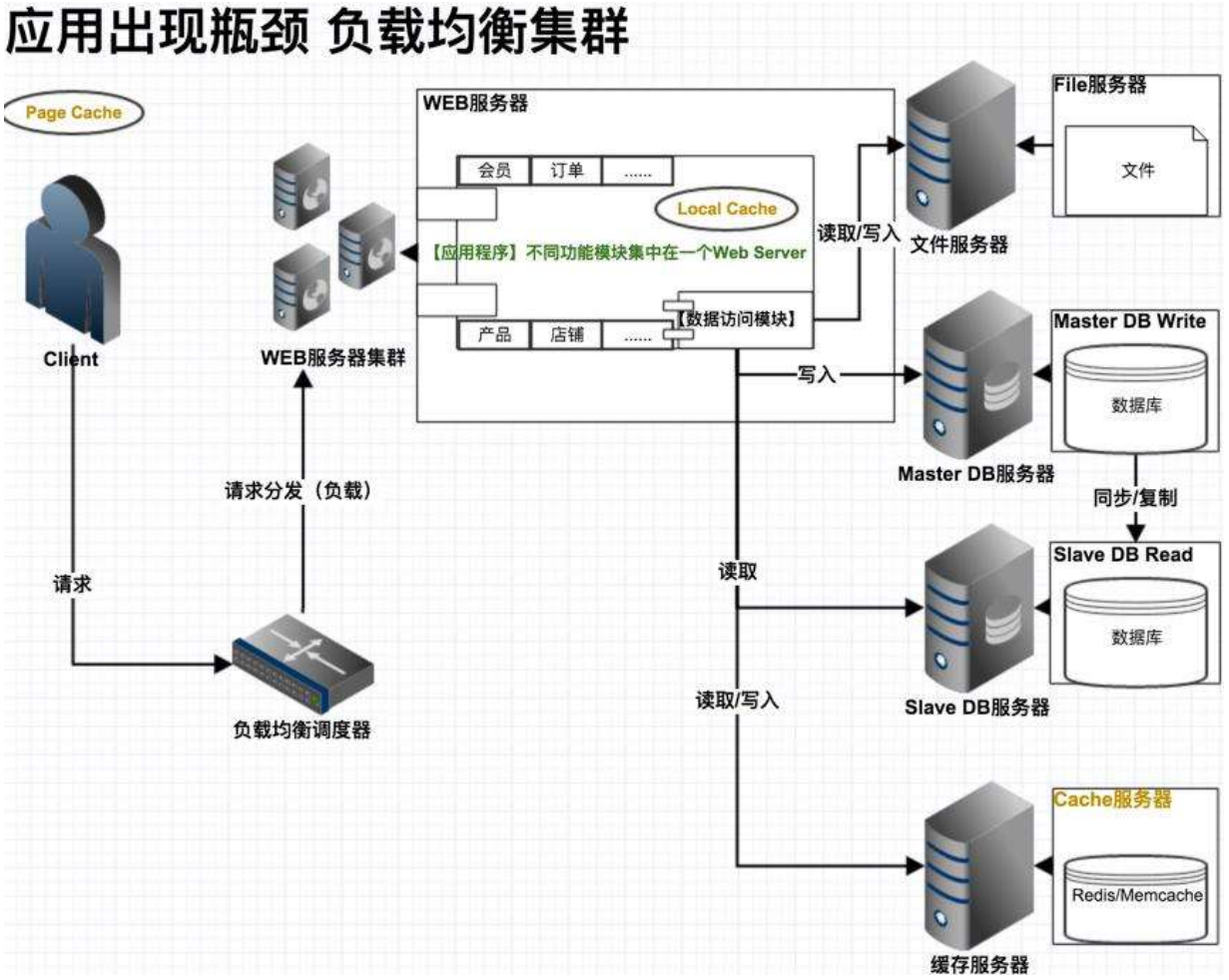
- 消费者-生产者模型的实现
- 在复杂场景下
  - 系统解耦
  - 异步通信（同步转异步）
  - 流量削峰



五、分布式系统

- 由若干互联的计算机构成一个完整的系统，完成一个共同的目标
  - 分布式计算系统
  - 分布式信息处理系统
  - 分布式控制系统
  - 分布式普适系统（嵌入式，可穿戴）
- 特征
  - 分布性
  - 自治性
  - 并行性
  - 全局性

分布式信息处理系统


应用出现瓶颈 负载均衡集群
Page Cache
Client
WEB服务器集群
WEB服务器
会员
订单
.....
Local Cache
【应用程序】不同功能模块集中在一个Web Server
产品
店铺
.....
【数据访问模块】
读取/写入
文件服务器
File服务器
文件
写入
Master DB服务器
Master DB Write
数据库
同步/复制
Slave DB Read
数据库
Slave DB服务器
读取
读取/写入
请求分发（负载）
请求
负载均衡调度器
Cache服务器
Redis/Memcache
缓存服务器