Chapter 1 Introduction to Computer System Architecture

Feng Li

fli@sdu.edu.cn

https://funglee.github.io

Outline

- 计算机系统的多级层次结构
- 计算机系统结构、组成与实现
- 计算机系统的软硬取舍、性能评测及定量设计原理
- 软件、应用、器件对系统结构的影响
- 系统结构中的并行性发展和计算机系统的分类

什么是计算机？

- 计算机是一种不需要人的直观干预而能自动完成各种算术和逻辑运算的工具。

Supercomputer

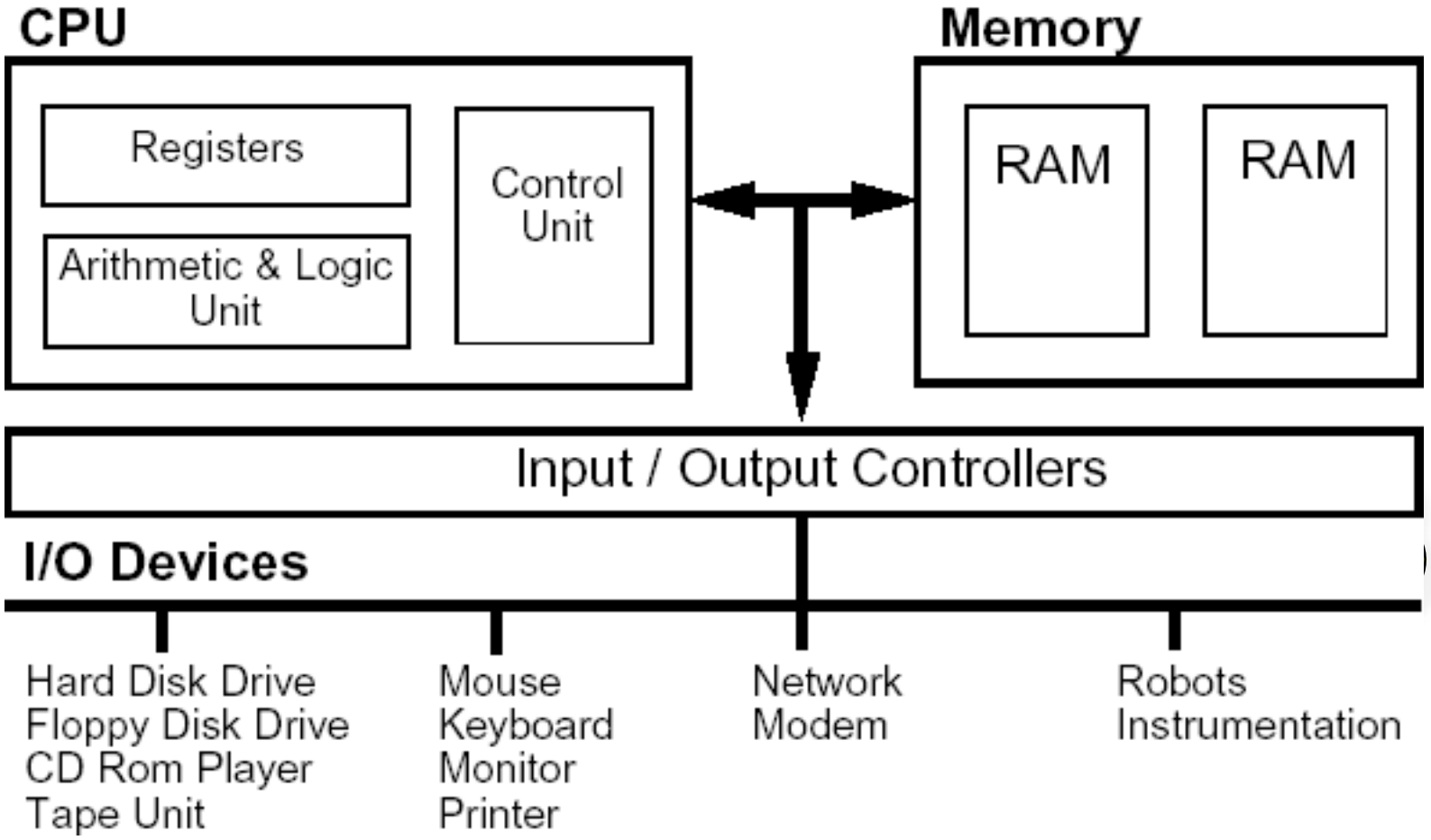
Mobile Phone

Laptop

Wearable Device

Sensor Mote

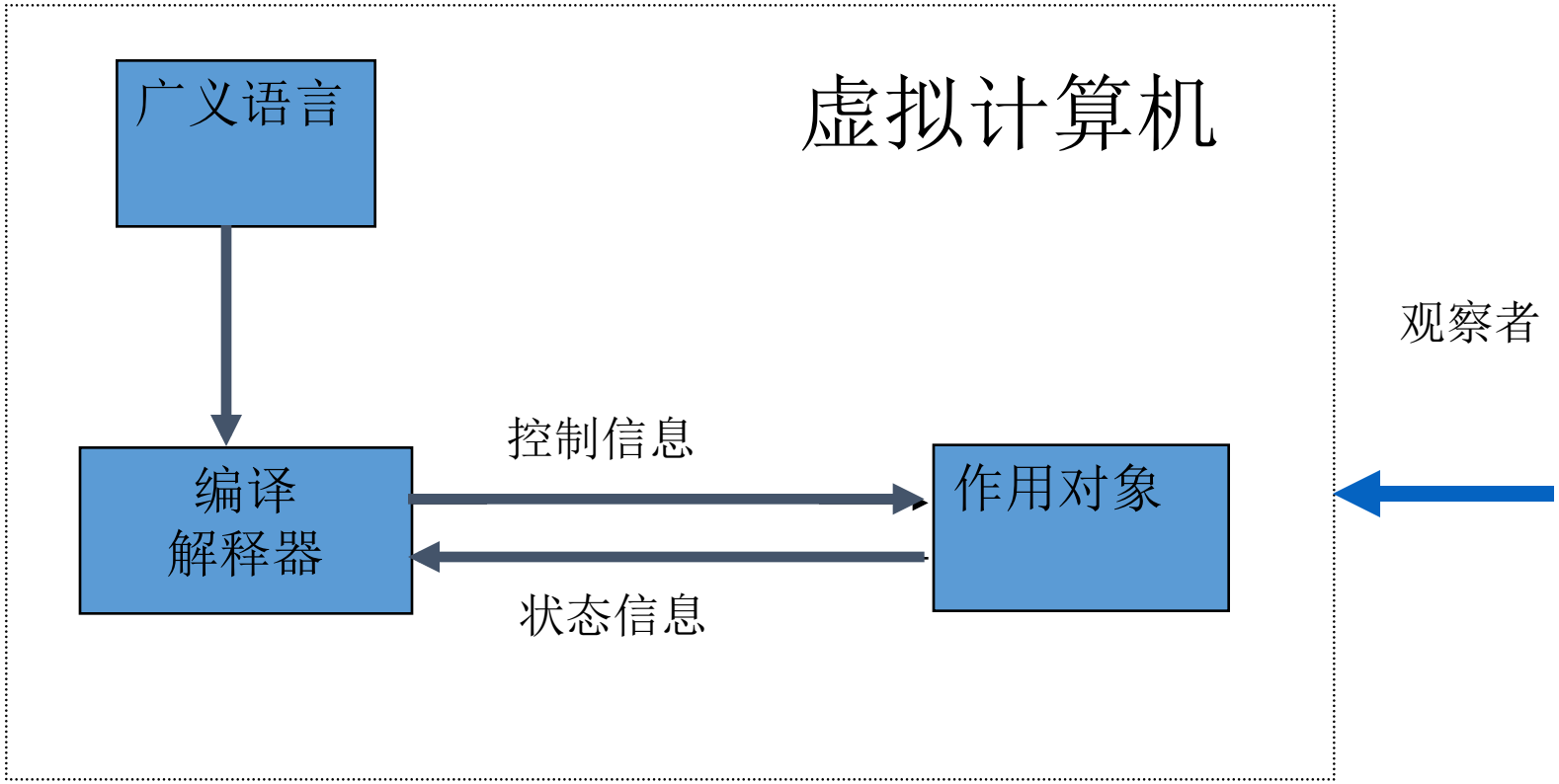
A Generic Architecture of Computer



计算机系统的多级层次结构

- 机器
  - 能存储和执行相应语言程序的算法和数据结构的执行体。
- 计算机语言
  - 是用以描述控制流程的、有一定规则的字符集合。
  - 语言不是专属软件范畴，可以分属于计算机系统的各个层次，具有不同作用。
- 多层次结构
  - 从使用语言的角度，一台由软、硬件组成的通用计算机系统可以被看成是按功能划分的多层机器级组成的层次结构。

从观察者看到的虚拟计算机



Preliminary Knowledge

- 微指令（Micro-code）
  - 一系列底层操作（即硬件控制信号）的集合称之为微指令，可以用来执行复杂的机器指令
  - 微指令一般由硬件完成，可以将机器指令与相关的电路做分离。
  - 微指令通常由CPU工程师在设计阶段编写，并存储在ROM或PLA中
  - 例如，寄存器之间的数据传递
- 微程序（Micro-program）
  - 一系列微指令的集合，对应于一条机器指令
- 机器语言（Machine Language）
  - 用二进制代码表示的计算机能直接识别和执行的一系列机器指令的集合
- 汇编语言（Assembly Language）
  - 依赖于符号的底层编程语言。
  - 用助记符代替机器指令的操作码，用地址符号（或标号）代替指令或操作数的地址

The advantages of layering computer systems

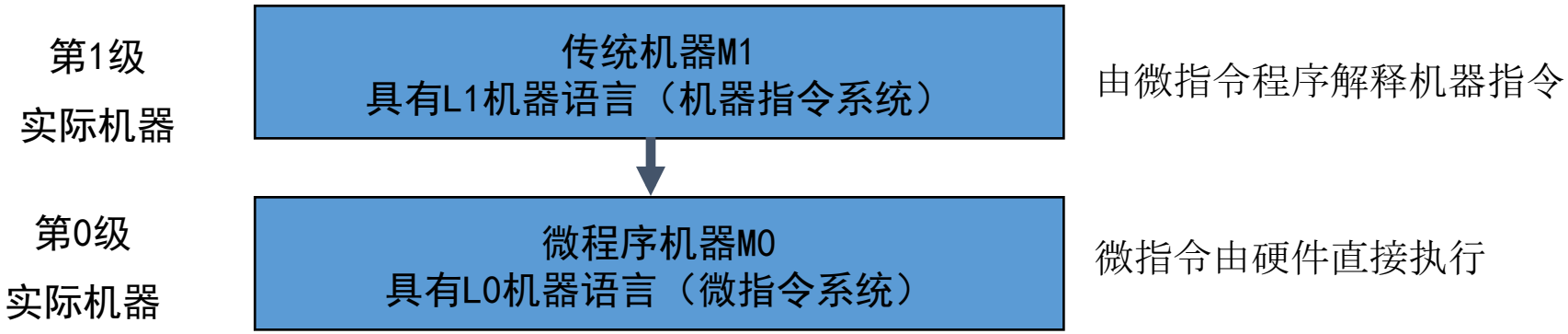
- 有利于人们正确理解计算机系统的工作，明确软硬件和固件在计算机系统的地位和作用
- 理解各种语言的实质及其实现
- 有利于探索虚拟机器新的实现途径，便于设计新的系统
- 有助于理解计算机体系结构的含义，从而合理地进行计算机系统的开发和设计。

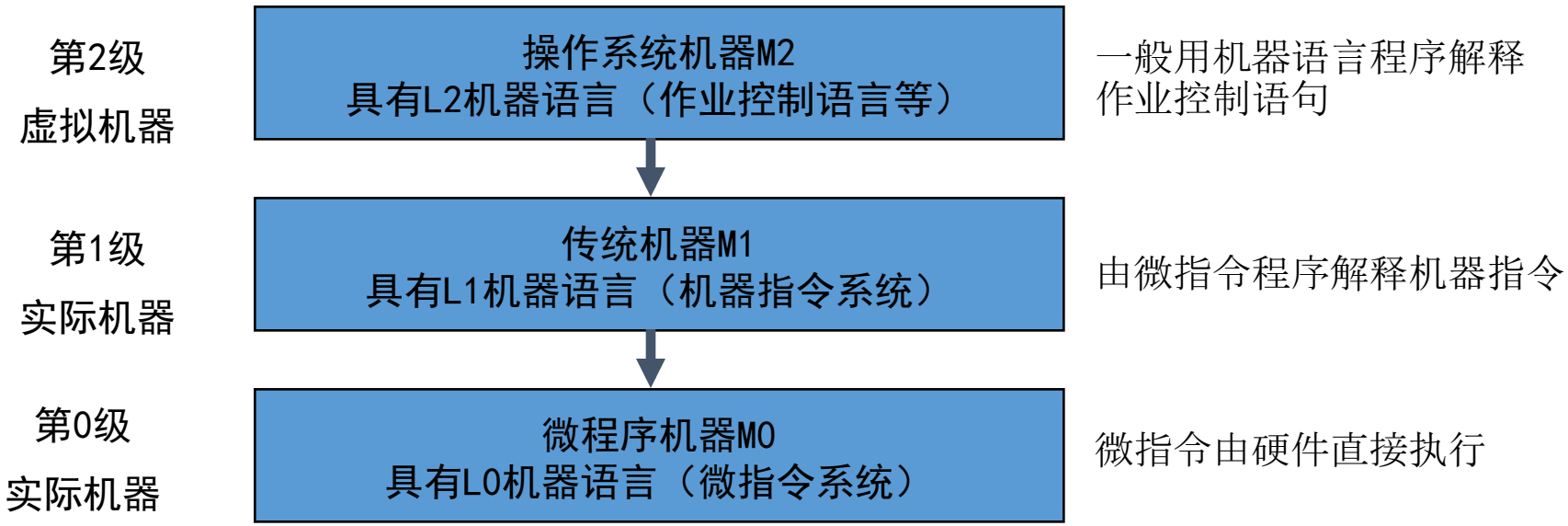
第0级

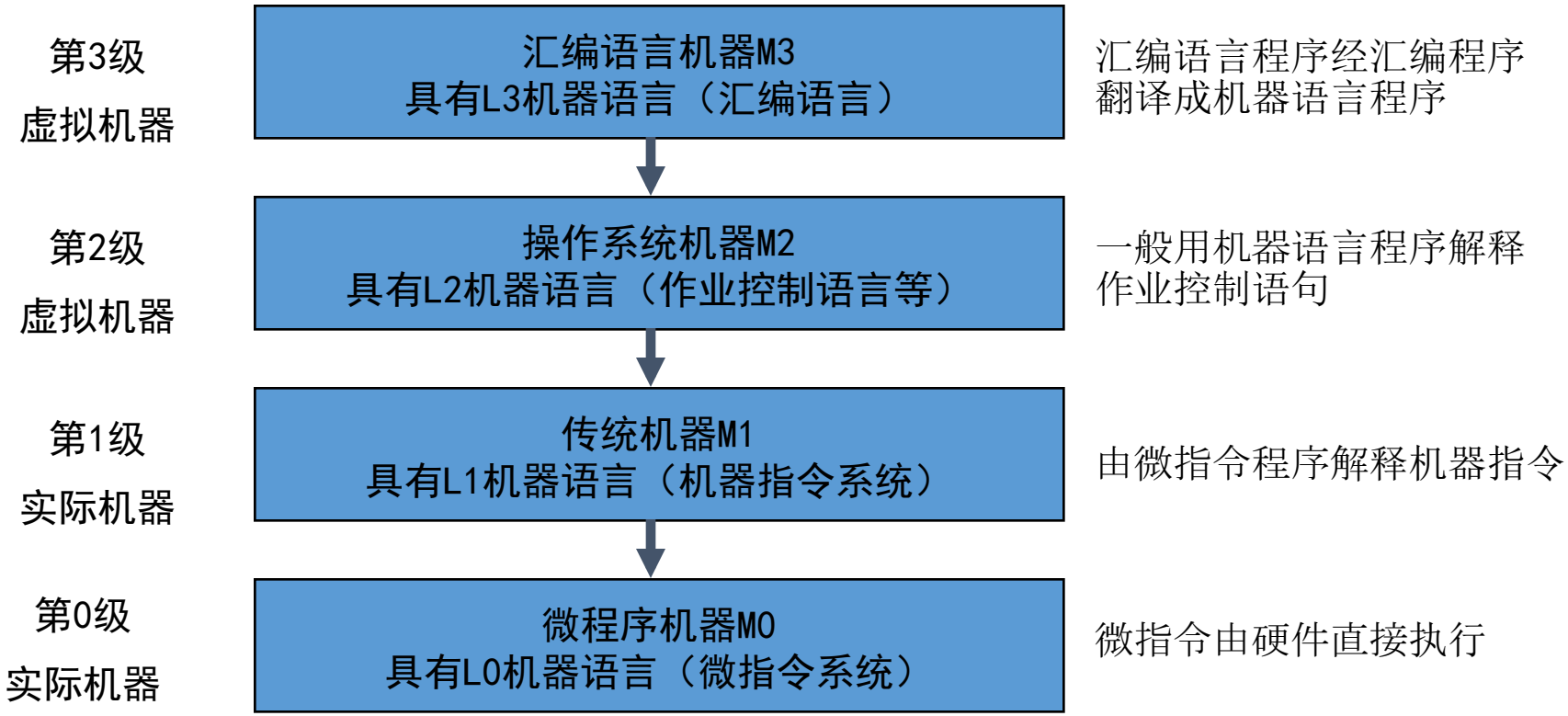
实际机器

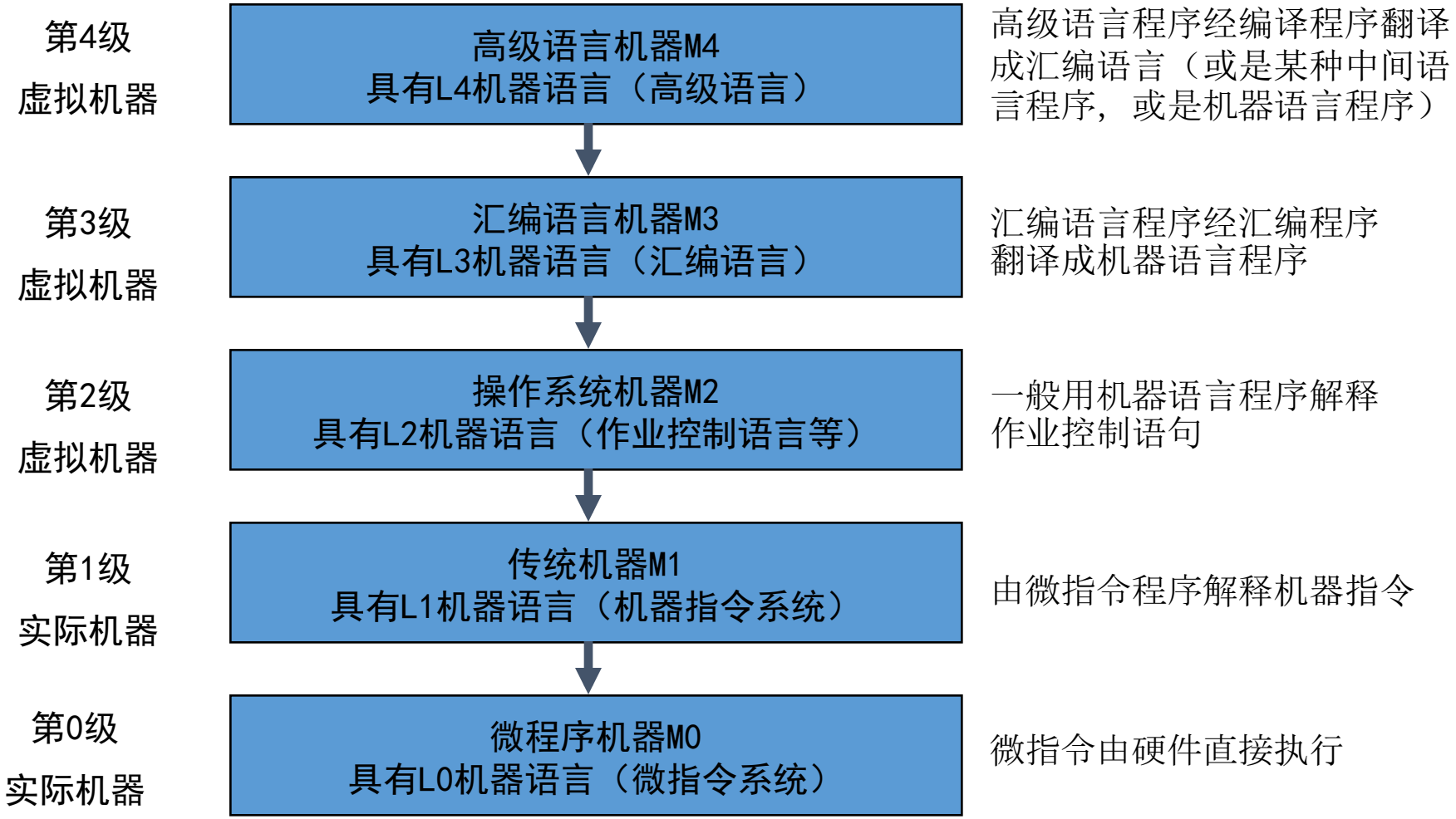
微程序机器M0
具有L0机器语言（微指令系统）

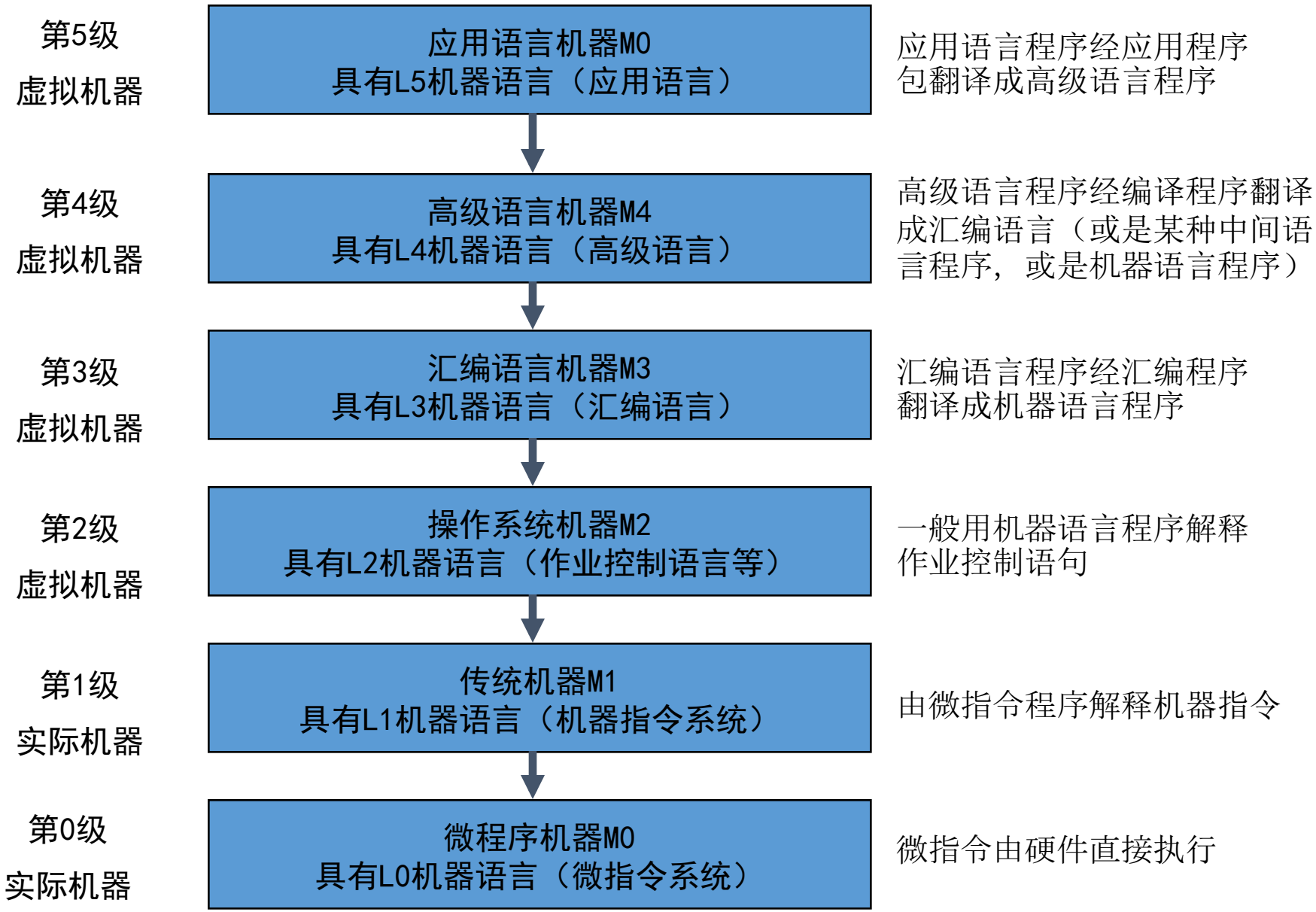
微指令由硬件直接执行


第1级
实际机器
传统机器M1
具有L1机器语言（机器指令系统）
由微指令程序解释机器指令
第0级
实际机器
微程序机器M0
具有L0机器语言（微指令系统）
微指令由硬件直接执行


第2级
虚拟机器
操作系统机器M2
具有L2机器语言（作业控制语言等）
一般用机器语言程序解释
作业控制语句
第1级
实际机器
传统机器M1
具有L1机器语言（机器指令系统）
由微指令程序解释机器指令
第0级
实际机器
微程序机器M0
具有L0机器语言（微指令系统）
微指令由硬件直接执行


第3级
虚拟机器
汇编语言机器M3
具有L3机器语言（汇编语言）
汇编语言程序经汇编程序翻译成机器语言程序
第2级
虚拟机器
操作系统机器M2
具有L2机器语言（作业控制语言等）
一般用机器语言程序解释作业控制语句
第1级
实际机器
传统机器M1
具有L1机器语言（机器指令系统）
由微指令程序解释机器指令
第0级
实际机器
微程序机器M0
具有L0机器语言（微指令系统）
微指令由硬件直接执行

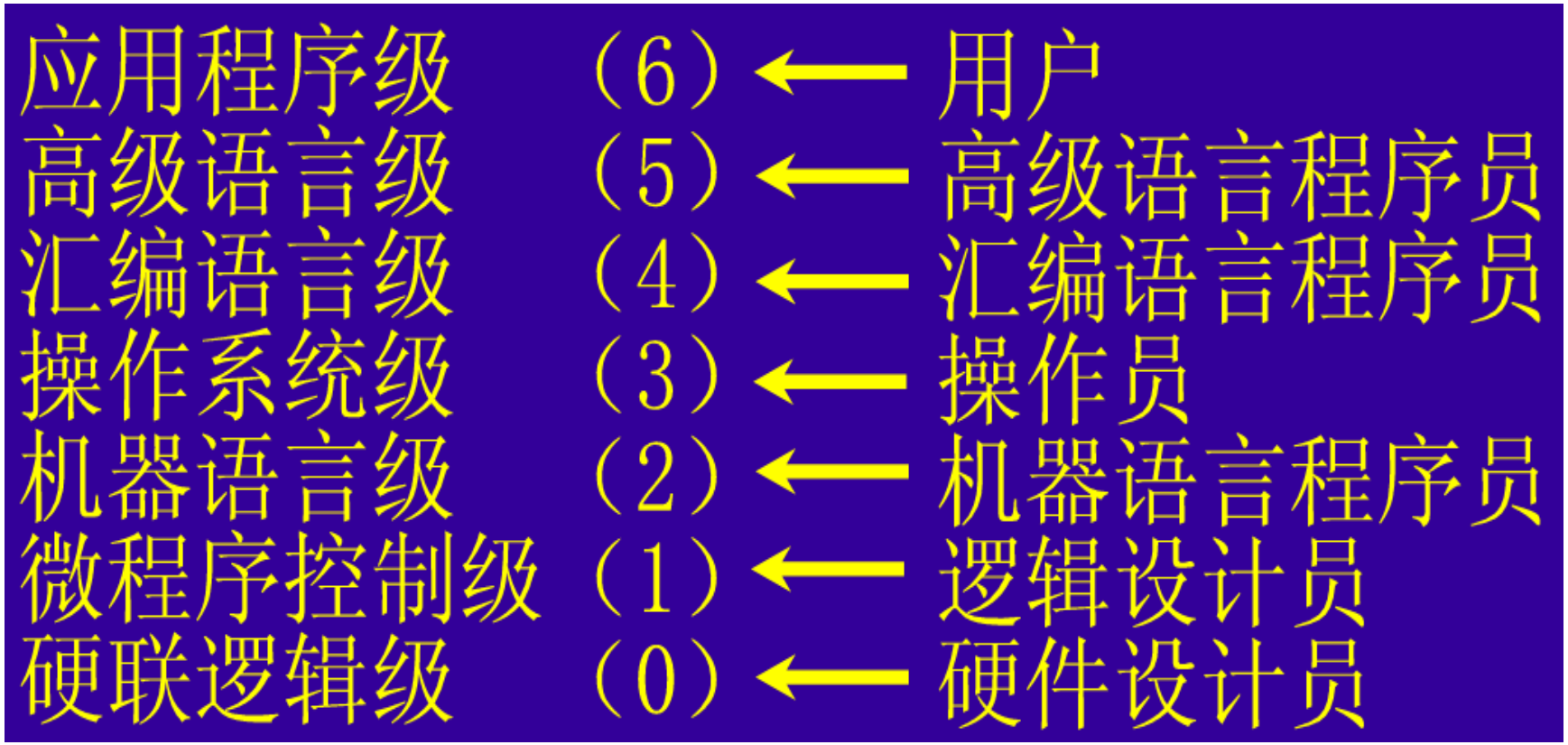

第4级
虚拟机器
高级语言机器M4
具有L4机器语言（高级语言）
高级语言程序经编译程序翻译成汇编语言（或是某种中间语言程序，或是机器语言程序）
第3级
虚拟机器
汇编语言机器M3
具有L3机器语言（汇编语言）
汇编语言程序经汇编程序翻译成机器语言程序
第2级
虚拟机器
操作系统机器M2
具有L2机器语言（作业控制语言等）
一般用机器语言程序解释作业控制语句
第1级
实际机器
传统机器M1
具有L1机器语言（机器指令系统）
由微指令程序解释机器指令
第0级
实际机器
微程序机器M0
具有L0机器语言（微指令系统）
微指令由硬件直接执行

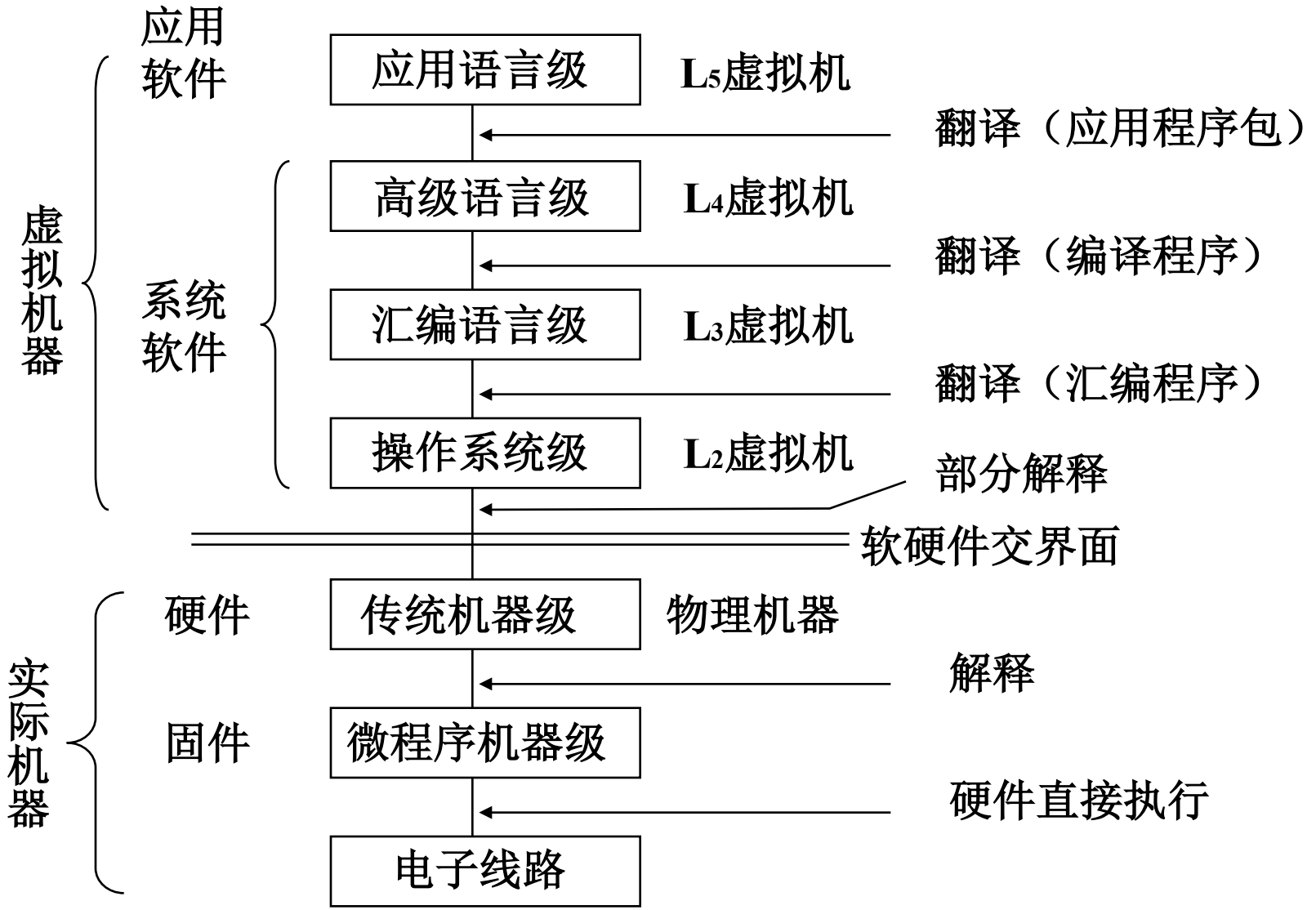

第5级
虚拟机器
应用语言机器M0
具有L5机器语言（应用语言）
应用语言程序经应用程序包翻译成高级语言程序
第4级
虚拟机器
高级语言机器M4
具有L4机器语言（高级语言）
高级语言程序经编译程序翻译成汇编语言（或是某种中间语言程序，或是机器语言程序）
第3级
虚拟机器
汇编语言机器M3
具有L3机器语言（汇编语言）
汇编语言程序经汇编程序翻译成机器语言程序
第2级
虚拟机器
操作系统机器M2
具有L2机器语言（作业控制语言等）
一般用机器语言程序解释作业控制语句
第1级
实际机器
传统机器M1
具有L1机器语言（机器指令系统）
由微指令程序解释机器指令
第0级
实际机器
微程序机器M0
具有L0机器语言（微指令系统）
微指令由硬件直接执行

多级层次结构（机器---语言）

- M5：应用语言机器--------应用语言
- M4：高级语言机器--------高级语言
- M3：汇编语言机器--------汇编语言
- M2：操作系统机器-------作业控制语言
- M1：传统机器------------机器指令系统
- M0：微程序机器----------微指令系统

从设计人员看到的层次


应用程序级 (6) ← 用户
高级语言级 (5) ← 高级语言程序员
汇编语言级 (4) ← 汇编语言程序员
操作系统级 (3) ← 操作员
机器语言级 (2) ← 机器语言程序员
微程序控制级 (1) ← 逻辑设计员
硬联逻辑级 (0) ← 硬件设计员


虚拟机器
实际机器
应用软件
系统软件
硬件
固件
应用语言级
高级语言级
汇编语言级
操作系统级
传统机器级
微程序机器级
电子线路
L5虚拟机
L4虚拟机
L3虚拟机
L2虚拟机
物理机器
翻译（应用程序包）
翻译（编译程序）
翻译（汇编程序）
部分解释
软硬件交界面
解释
硬件直接执行

编译 Vs 解释

- 编译（Compilation）
  - The source code in high-level language is transformed by a compiler into another (low-level) computer language which usually has an executable binary form
  - High efficiency, platform-dependent
  - E.g., C, C++
- 解释（Interpretation）
  - The instructions written by programming or script language is directly interpreted and performed by a interpreter during the execution.
  - Low efficiency, platform-independent
  - E.g., Perl, Python

就目前的情况来讲

- M0 用硬将实现，M1用微程序（固件）实现，M2-M5大多用软件实现
- 固件 Firmware：一种具有软件功能的硬件，指存储在计算机ROM和其它集成电路中的系统软件，固件不能随意改变。
- 以软件为主实现的机器称为虚拟机，以区别由硬件或者固件实现的实际机器
- 计算机系统作为一个整体，包括软、硬件，之间无固定界面。

计算机系统结构、组成与实现

- 计算机系统结构、组成与实现的定义和内涵
- 计算机系统结构、组成和实现的相互关系

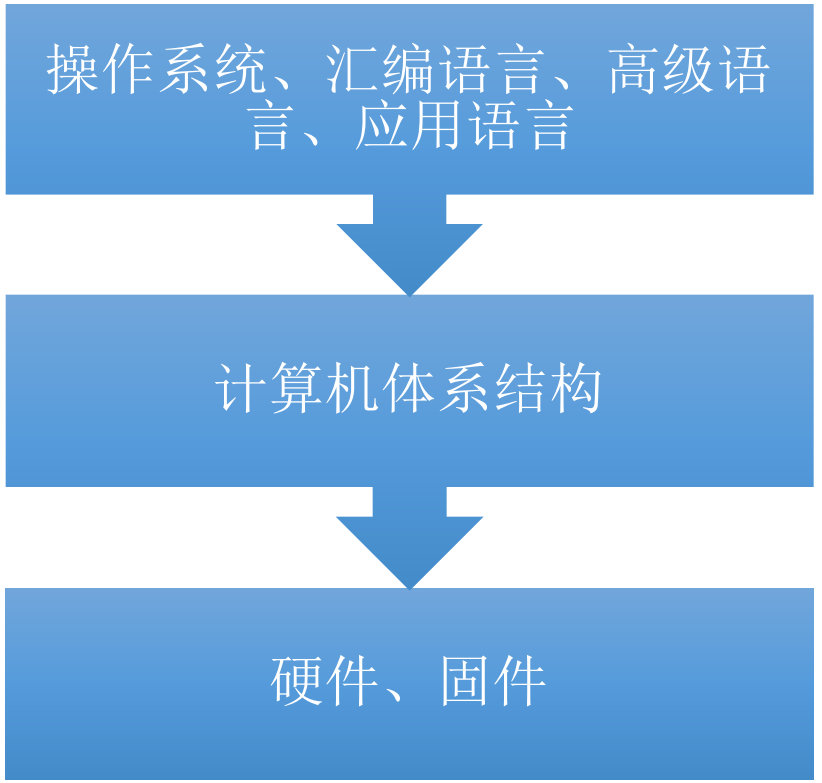
What's computer architecture?

- The attributes of a computer system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation.

Amdahl, Blaaw, and Brooks, 1964

计算机系统结构

- 系统结构 System Architecture
  - 计算机系统中各级界面的定义及其上下功能的分配
  - 透明（Transparent）:各个层级的程序员所看到的计算机属性是不同的，系统结构的研究内容之一就是要确定属性的透明性。
- 计算机系统结构 （计算机系统的体系结构，Computer Architecture）
  - 传统机器级（M1）的系统结构
  - 软件和硬件 / 固件的交接面
  - 机器语言、汇编语言程序设计者，或者编译程序设计者看到的机器物理系统的抽象



计算机系统结构的属性

- 软、硬件之间的功能分配以及对传统机器级界面的确定
- 属性包括
  - 硬件级的数据类型和格式
  - 寻址功能的单位、种类和计算
  - 通用 / 专用寄存器的组织（设置、数量、字长、使用规则等）
  - 指令系统（二进制或汇编指令）
  - 存储系统组织（编址单位、编址方式、容量、最大可编址空间等）
  - 中断机制（中断的分类与分级、中断处理程序功能和入口地址等）
  - 系统机器级的管态和用户态的定义与切换
  - 机器级I/O结构（I/O设备的连接、使用、流量、容错等）
  - 系统各个部分的信息保护方式和保护机构等属性

计算机组成 Computer Organization

- **计算机系统结构的逻辑实现，**包括机器级内部的**数据流**和**控制流**的组成以及逻辑设计等
- 研究内容：机器级内部各事件（Events）的排序方式和控制机构、各部件的功能和部件之间的联系。
- 研究目标：如何将各种设备和部件组织成计算机，实现所确定系统结构，达到既定的性价比。
- 属性包括：
  - 数据通路宽度
  - 专用部件的设置
  - 部件的共享机制和并行机制
  - 控制机构的组成方式 （硬件 / 固件，单机 / 多机）
  - 缓冲（Caching）和排队（Queuing）技术
  - 预估、预判技术
  - 可靠性技术（冗余和容错机制）

计算机实现 Computer Implementation

- 计算机组成的物理实现
- 研究各部件的物理结构、机器的制造技术和工艺等，是计算机组成的物理实现。它着眼于器件技术和微组装技术。
- 主存的物理实现，如存储器采用什么样器件，逻辑电路设计和微组装技术均属计算机实现

举例1

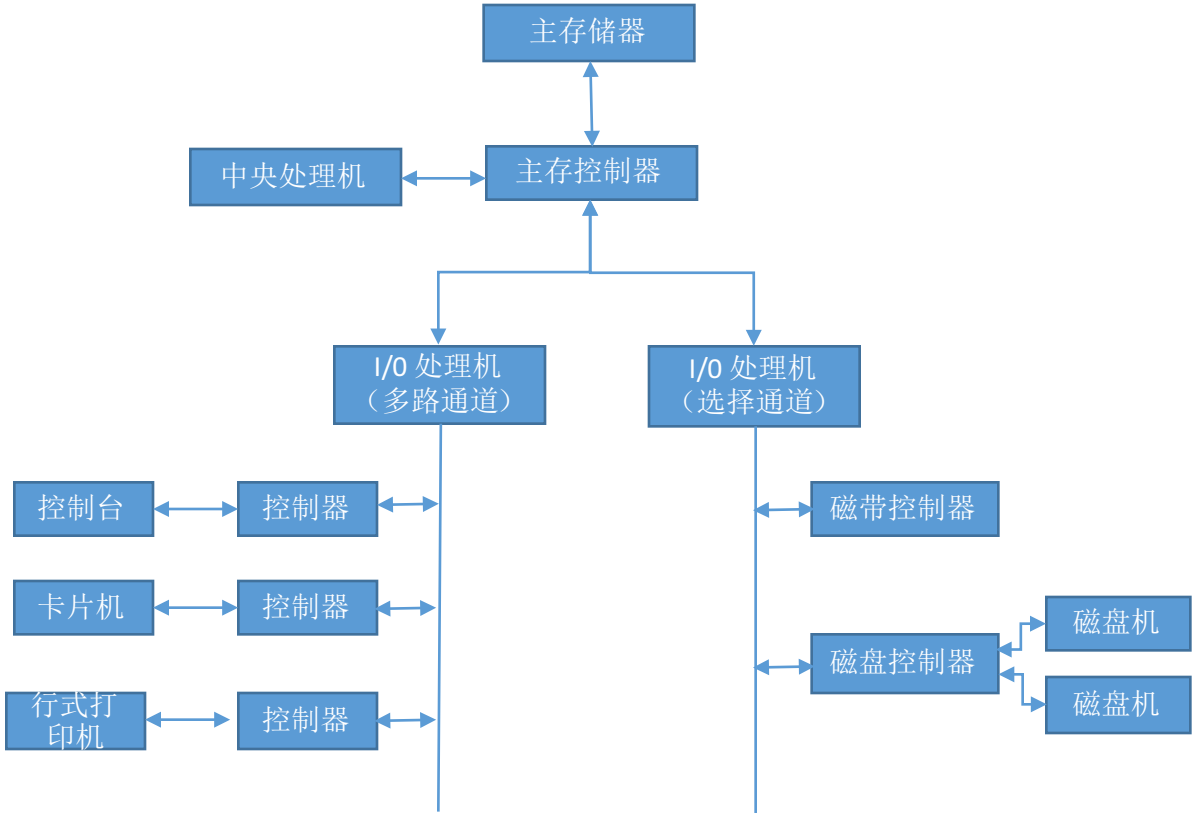
- 指令系统
  - 指令系统的确定----系统结构
  - 指令的实现---------组成
  - 具体电路、器件设计及装配技术---实现
- 乘法指令
  - 是否设乘法指令---系统结构
  - 用高速乘法器还是加法器移位器实现---组成
  - 器件的类型、数量及组装技术的确定---实现

举例2

- 主存系统
  - 主存容量与编址方式的确定----系统结构
  - 主存速度的确定、逻辑结构的模式---组成
  - 器件的选定、电路的设计、组装技术---实现

举例3

- IBM系列机包含多个型号，如115，125，135，145，158，168等
- 这些系列机基于同样的系统结构（例如指令系统，数据表示，是否采用通道机制等），但不同型号的机器使用了不同的组成和实现（如数据通路宽度，指令的解释方式等）
- 设计何种系列机属于计算机系统结构研究范畴，而系列内不同型号计算机的组织属于计算机组成研究的范畴



举例4

- IBM 370 系列机具有相同的机器指令和汇编指令系统
  - 低档机采用较低的数据通路宽度，而高档机采用较高的数据通路宽度
  - 指令的分析和执行在低档机上采用顺序执行，在高档机上采用重叠、流水或其他并行处理方式进行
  - 低档机采用结合型通道，而高档机采用独立型通道
- 系统结构：机器 / 汇编指令系统、数据表示、是否采用通道方式
- 计算机组成：指令采用顺序、重叠、流水还是其他方式解释，数据通路宽度的确定，通道采用结合型还是独立型

透明性 Transparence

- 本来存在的事务或属性，从某个角度上看不到。反之，不透明。
- 在一个计算机系统中，低层机器的属性往往对高层机器的程序员是透明的
- 计算机组成设计的内容，对传统机器程序员来讲一般是透明的。

举例1

- 在多级层次结构的计算机系统中，传统机器级的概念性结构和功能特性，对高级语言的程序员来说是透明的，而对汇编语言的程序员来说不是透明的。这说明高级语言的程序员不必知道机器的指令系统、中断机构等，这些本来存在的属性，对高级语言的程序员来说好象不存在一样，所以说是透明的。
- 对计算机系统结构来说，存储器采用交叉存取还是并行存取、CPU内部的数据通路的宽度是8位还是16位，这些都是透明的，而对计算机组成来说这些不是透明的。

举例2

- 指令执行采用串行、重叠还是流水控制方式，对系统结构来说是透明的，但对计算机组成来说不是透明的。
- 乘法指令采用专用乘法器实现。对系统结构来说是透明的，而对计算机组成来说不是透明的。
- 存储器采用哪种芯片，对计算机系统结构和组成来说是透明的，而对计算机实现来说不是透明的。

系统结构、组成和实现三者的相互关系

- 具有相同系统结构的计算机可以采用不同的组成，一种计算机组成可以采用多种不同的计算机实现；
- 采用不同的系统结构会使可以采用的组成技术产生差异，计算机组成也会影响系统结构；

一种计算机系统结构，可以采用不同的组成

一种计算机系统结构	可以采用不同的组成	性能与价格	考虑因素
1。设计指令系统	1。指令间顺序执行	速度慢、价格低	性能价格比
	2。指令间重叠执行	速度快、价格高	
2。乘法指令	1。用加法器、移位器	速度慢、价格低	性能价格比、乘法指令使用频度
	2。用专门乘法器	速度快、价格高	

结构对组成的影响

A:=B+C　　D:=E*F

面向寄存器：

```
LOAD   R1,B
ADD    R1,C
STORE  A, R1
LOAD   R2,E
MPY    R2,F
STORE D, R2
```

面向主存的三地址寻址：

```
ADD   B,C,A
MPY   E,F,D
```

- 若要提高运算速度，可以让加法运算和乘法运算并行，需要设置独立的加法器和乘法器
- 面向寄存器的系统结构，需要让R1和R2同时被访问
- 面向主存的三地址寻址，需要同时形成多个访存操作数地址，并能够同时访存

组成对结构的影响

- 微程序控制
  - 通过改变控制存储器中的微程序，就可以改变机器指令，从而改变结构
  - 将复杂指令利用微程序实现，可以减少大量访问主存机器指令的次数，提高速度

系统结构、组成和实现三者的相互关系

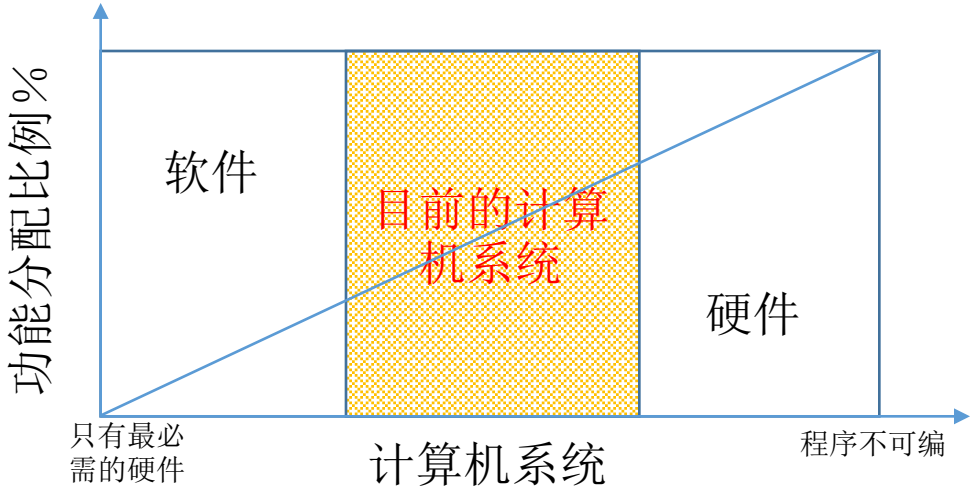
- 系统结构的设计需要考虑到应用的需求，为软件和算法的实现提供更多、更好的支持，同时要考虑对不同组成和实现技术的兼容性
- 计算机组成的设计，其上决定于计算机系统结构，其下又受限于所用的实现技术。它的发展促进了实现技术的发展，也促进了结构的发展；
- 组成与实现相互权衡
  - 复杂组成简单实现：专用芯片
  - 简单组成复杂实现：高性能器件
- 计算机实现，特别是器件技术的发展是计算机系统结构和组成的基础，促进了组成与结构的发展；
- 结构、组成和实现的相互转换和融合
  - 随着技术的发展，三者关系融合于一体，难以分开，在相互促进中发展。

计算机系统的软硬取舍、性能评测及定量设计原理

- 软、硬取舍的基本原则
- 计算机系统的性能评测
- 计算机系统的设计原则
- 计算机系统设计者的主要任务
- 计算机系统的设计思路
- 计算机系统的设计步骤

软硬件的取舍

- 软件和硬件的逻辑等效性
  - 硬件实现：速度快、成本高；灵活性差、占用内存少
  - 软件实现：速度慢、复制费用低；灵活性好、占用内存多、易设计、可改性强、适应性强、设计周期短；
- 软件和硬件的分配
  - 在满足应用的前提下，软硬件功能分配的比例主要看能否成分利用硬件、器件技术的进展，使系统具有高的性价比



软件和硬件的取舍原则

- 原则1：应考虑在现有硬件、器件（主要是逻辑器件和存储器件下）条件下，系统要有高的性价比，主要从实现费用、速度和其他性能要求来综合考虑。

设计费用D_s，D_h，$D_h \approx 100D_s$，每次重复生产费用M_s，M_h，$M_h \approx 100M_s$，软件重新设计次数C，软件重复生产次数R，计算机生产数量V。那么每台计算机硬件费用$=\frac{D_h}{V}+M_h$，软件费用$=\frac{CD_s}{V}+M_sR$。*只有当*

$$\frac{D_h}{V}+M_H<\frac{CD_s}{V}+M_sR，即\frac{100D_s}{V}+100M_s<\frac{CD_s}{V}+M_sR$$

才适合使用硬件使用该功能。

$$\frac{100D_s}{V} + 100M_s < \frac{CD_s}{V} + M_sR$$

- 结论1：生产一定数量的计算机，C和R越大，即该功能是要经常使用的基本功能，才适合用硬件实现
- 结论2：假设$D_s \approx 10^4 \times M_s$,

$$\frac{10^6}{V} + 100 < \frac{C}{V} + R$$

由于C一般小于100，因此V越大，即该计算机系统产量比较大，增大硬件功能实现的比例才是合适的。

软件和硬件的取舍原则

- 原则2：要考虑准备采用和可能采用的组成技术，使之尽可能不要过多或者不合理地限制各种组成、实现技术的采用
- 原则3：不能仅从“硬”的角度考虑如何 便于应用组成技术的成果和便于发挥器件技术的进展，还应从“软”的角度把如何为编译和操作系统的实现以及如何为高级语言程序的设计提供更多、更好的硬件支持放在首位。
- 结论：要缩短底层系统结构和机器语言与上层高级语言、操作系统和程序设计环境之间的语义差距，加强系统结构对软件设计的支持。

软硬取舍的基本原则（续）

- 考虑用户的应用领域：专用—硬件
- 设计周期长的硬件不宜采用
- 常用的功能尽量采用硬件实现
- 实现功能的成本性能比（或价格性能比）要低
- 尽量采用新技术实现超前设计

举例1：

- 某一计算机用于商业外贸的事务处理，有大量的字符串操作。由于这种事务处理很普遍，有较大的市场，故而设计人员决定在下一代此类计算机的CPU中加入字符串操作的功能。经测试应用软件调查发现，字符串操作的使用占整个程序运行时间的50%，而增加此功能如用软件（如微程序）实现，则快5倍，增加CPU成本1/5倍；如果用硬件实现，则快100倍，CPU成本增加到5倍。问设计人员提出增加此功能是否恰当？是否用软件还是硬件？
- 设CPU成本占整机成本的1/3。

- 硬件实现

$$T_{new} = T_{old}(1-50\%) + \frac{50\%T_{old}}{100}$$

$$\text{性能变化} = \frac{T_{old}}{T_{new}} = \frac{T_{old}}{T_{old}(1-50\%) + \frac{50\%T_{old}}{100}} = \frac{1}{1-50\% + \frac{50\%}{100}} = 1.98\text{倍}$$

$$\text{成本增加} = \frac{2}{3} * 1 + \frac{1}{3} * 5 = 2.33\text{倍}$$

$$\text{成本性能比} = \frac{2.33}{1.98} = 1.18\text{倍}$$

- 软件实现

$$T_{new} = T_{old}(1-50\%) + \frac{50\%T_{old}}{5}$$

$$\text{性能变化} = \frac{T_{old}}{T_{new}} = \frac{T_{old}}{T_{old}(1-50\%) + \frac{50\%T_{old}}{5}} = \frac{1}{1-50\% + \frac{50\%}{5}} = 1.66\text{倍}$$

$$\text{成本增加} = \frac{2}{3} * 1 + \frac{1}{3} * (1 + \frac{1}{5}) = 1.07\text{倍}$$

$$\text{成本性能比} = \frac{1.07}{1.66} = 0.64\text{倍}$$

举例2：

- 如果上例中，字符串操作功能的使用时间占整个程序运行时间的90%，则情况如何？
- 硬件实现

$$\text{性能变化} = \frac{T_{old}}{T_{new}} = \frac{T_{old}}{T_{old}(1-90\%) + \frac{90\%T_{old}}{100}} = \frac{1}{1-90\% + \frac{90\%}{100}} = 9.17\text{倍} \qquad \text{成本性能比} = \frac{2.33}{9.17} = 0.25\text{倍}$$

- 软件实现

$$\text{性能变化} = \frac{T_{old}}{T_{new}} = \frac{T_{old}}{T_{old}(1-90\%) + \frac{90\%T_{old}}{5}} = \frac{1}{1-90\% + \frac{90\%}{5}} = 3.57\text{倍} \qquad \text{成本性能比} = \frac{1.07}{3.57} = 0.30\text{倍}$$

https://funglee.github.io/csa.html

计算机系统的评测

- 在系统上程序实际运行的时间是衡量机器时间（速度）性能最可靠的标准
- 峰值性能：在理想情况下计算机系统可获得的最高理论性能值，它不能反映出系统的实际性能
- 实际性能（持续性能）：只是峰值性能的5%-30%

持续性能的表示

- 算术性能平均值： $A_m = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{T_i} = \frac{1}{n}(\frac{1}{T_1} + \frac{1}{T_2} + ... + \frac{1}{T_n})$
- 调和性能平均值： $H_m = \frac{n}{\sum_{i=1}^{n} \frac{1}{R_i}} = \frac{n}{\sum_{i=1}^{n} T_i} = \frac{n}{T_1 + T_2 + ... + T_n}$
- 几何性能平均值： $G_m = \sqrt[n]{(\prod_{i=1}^{n} R_i)} = \sqrt[n]{(\prod_{i=1}^{n} \frac{1}{T_i})}$

- 加权算术平均值：$A_m = \sum_{i=1}^{n} \alpha_i R_i = \sum_{i=1}^{n} \alpha_i \frac{1}{T_i}$
- 加权调和平均值：$H_m = (\sum_{i=1}^{n} \alpha_i T_i)^{-1} = (\sum_{i=1}^{n} \frac{\alpha_i}{R_i})^{-1}$
- 加权几何平均值：$G_m = \prod_{i=1}^{n} (R_i)^{\alpha_i} = R_1^{\alpha_1} \times R_2^{\alpha_2} \times \ldots \times R_n^{\alpha_n}$

CPU性能公式

- CPU的程序执行时间T_{CPU}
  - 程序执行的总指令条数IC（Instruction Counter）
  - 平均每条指令的时钟周期数CPI（Cycles Per Instruction）
  - 时钟主频 f_c

$$T_{CPU} = IC \times CPI \times \frac{1}{f_c}$$

- n种指令，每种指令的时钟周期数CPI_i，出现次数I_i

$$T_{CPU} = \frac{\sum_{i=1}^{n}(CPI_i \times I_i)}{f_c}$$

- 平均指令时钟周期数

$$CPI = \frac{\sum_{i=1}^{n}(CPI_i \times I_i)}{IC} = \sum_{i=1}^{n}(CPI_i \times \frac{I_i}{IC})$$

MIPS (Million Instructions Per Second)

$$MIPS = \frac{IC}{T_{CPU} \times 10^6} = \frac{f_c}{CPI} \times 10^{-6},$$

$$T_{CPU} = \frac{IC}{MIPS} \times 10^{-6}$$

- MIPS越高，在一定程度上反映机器性能越好
- MIPS很大程度依赖于指令系统，只能用于相同指令系统的计算机之间的性能
- MIPS还与机器硬件的实现有关

MFLOPS（Million Floating Point Operations Per Second）

$$MFLOPS = \frac{I_{FN}}{T_{CPU}} \times 10^{-6}$$

- I_{FN}表示程序运行中的浮点运算总次数
- 只能反映机器执行浮点操作的性能，并不能反映机器的整体性能
- 适用于衡量处理机中向量运算性能
- 一般认为$1\ MFLOPS \approx 3\ MIPS$

基于工作负荷的评测方法

- 采用实际的应用程序测试
  - 如：C语言的编译程序，CAD应用：Spice
- 采用核心程序测试
  - 从实际程序中抽出关键部分组合而成，例如循环部分
- 合成测试程序
  - 人为写的核心程序，规模小，结果预知
- 综合基准测试程序
  - 考虑了各种可能的操作和各种程序的比例，人为地平衡编制的基准测试程序

计算机系统的定量设计原理

- 哈夫曼压缩原理
- Amdahl定律
- 程序访问的局部性规律

哈夫曼压缩原理

- 尽可能加速高概率事件远比加速处理概率很低的事件对性能提高要显著。
- 例如，CPU在运算中发生溢出的概率很低，所以设计时主要考虑加快不溢出时候的运算速度

Amdahl（安达尔）定律

- 系统中某一部件由于采用某种更快的执行方式后整个系统性能的提高与这种执行方式的执行频率或占总执行时间的比例有关
- 1967年由IBM公司的Amdahl提出
- 对系统中的性能瓶颈部分采取措施提高速度
- 系统加速比$S_p = \frac{T_{old}}{T_{new}} = \frac{1}{(1-f_{new})+f_{new}/r_{new}}$
  - 性能可改进比f_{new}
  - 部件加速比r_{new}

例1：假设将某系统的某一部件的处理速度加快到10倍,但该部件的原处理时间仅为整个运行时间的40%，则采用加快措施后能使整个系统的性能提高多少？

$$S_p = \frac{1}{(1-0.4)+0.4/10} = \frac{1}{0.64} \approx 1.56$$

例2：采用哪种实现技术来求浮点数平方根FPSQR的操作对系统的性能影响较大。假设FPSQR操作占整个测试程序执行时间的20%。一种实现方法是采用FPSQR硬件，使FPSQR操作的速度加快到10倍。另一种实现方法是使所有浮点数据指令的速度加快，使FP指令的速度加快到2倍，还假设FP指令占整个执行时间的50%。请比较这两种设计方案。

- $S_{FPSQR} = \frac{1}{(1-0.2)+0.2/10} = \frac{1}{0.82} = 1.22$
- $S_{FP} = \frac{1}{(1-0.5)+0.5/2} = \frac{1}{0.75} = 1.33$

例3：如果FP操作的比例为25%，FP操作的平均CPI=4.0，其它指令的平均CPI为1.33，FPSQR操作的比例为2%， FPSQR的CPI为20。假设有两种设计方案，分别把FPSQR操作的CPI和所有FP操作的CPI减为2。试利用CPU性能公式比较这两种设计方案哪一个更好(只改变CPI而时钟频率和指令条数保持不变)。

原系统的CPI=25%×4+75%×1.33=2

方案1（使FPSQR操作的CPI为2）系统

CPI=CPI原-2%×（20-2）=2-2%×18=1.64

方案2（提高所有FP指令的处理速度）系统

CPI=CPI原-25%×（4-2）=2-25%×2=1.5

显然，提高所有FP指令处理速度的方案要比提高FPSQR处理速度的方案要好。

方案2的加速比=2/1.5= 1.33

程序访问的局部性规律

- 局部性分时间上的局部性和空间上的局部性
  - 时间局部性：程序中近期被访问的信息项很可能马上将被再次访问。
  - 空间局部性：指那些在访问地址上相邻近的信息项很可能会被一起访问。
- 存储器体系的构成就是以访问的局部性原理为基础的

计算机系统设计

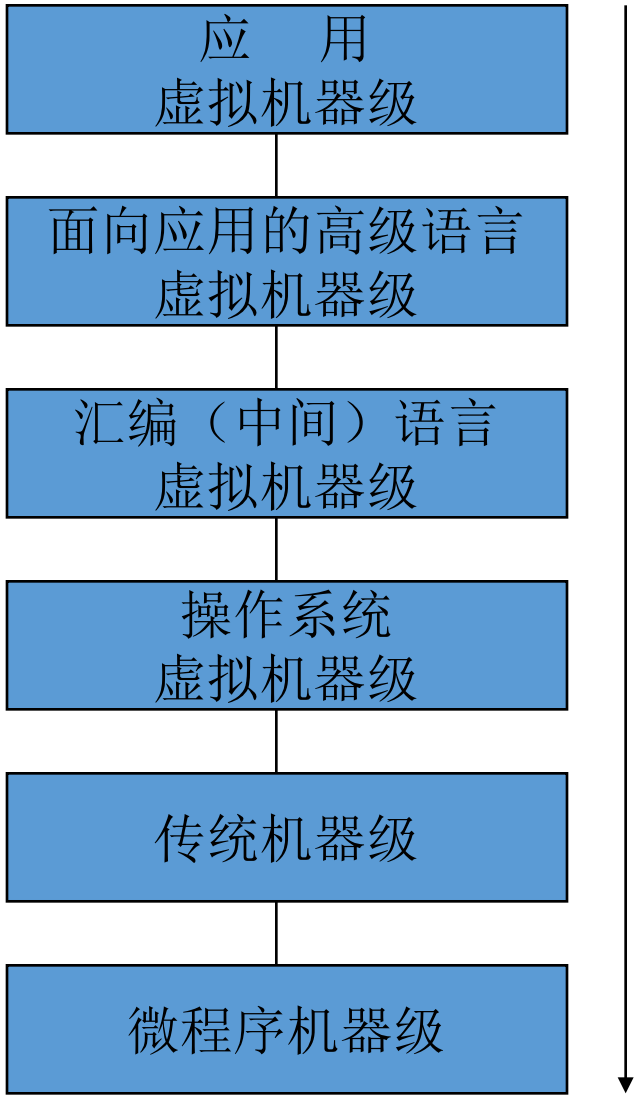
- 主要包括**系统结构设计、计算机组成设计、计算机实现设计**
- 确定用户对于功能、性能和价格的需求
  - 应用领域是专用还是通用
  - 软件兼容属于哪个层级（高级语言级还是传统机器语言级）
  - 对操作系统的要求（主要是存储系统的设计）
  - 保证标准化程度
- 优化系统设计，提高性价比
  - 应用程序测试评价量化性能
  - 软件硬件的分配
- 降低设计复杂性
- 适应未来硬件技术、软件技术、器件技术和应用需求的发展变化
- 易扩展性、兼容性

计算机系统设计思路

- 由上往下(Top-down)
- 由下往上(Bottom-up)
- 由中间开始(Middle-out)

由上往下(Top-down)

- 设计过程：由上向下
  - 面向应用的数学模型
  - 面向应用的高级语言
  - 面向这种应用的操作系统
  - 面向操作系统和高级语言的机器语言
  - 面向机器语言的微指令系统和硬件实现
- 应用场合：专用计算机的设计（早期计算机的设计）
- 特点：对于所面向的应用领域，性能和性能价格比很高，应用对象变了，难以适应。
- 随着通用计算机价格降低，目前已经很少采用



第一步：确定这一级的基本特性

第二步：设计或选择面向这种应用的高级语言

第三步：设计适于所用高级语言编译的中间语言

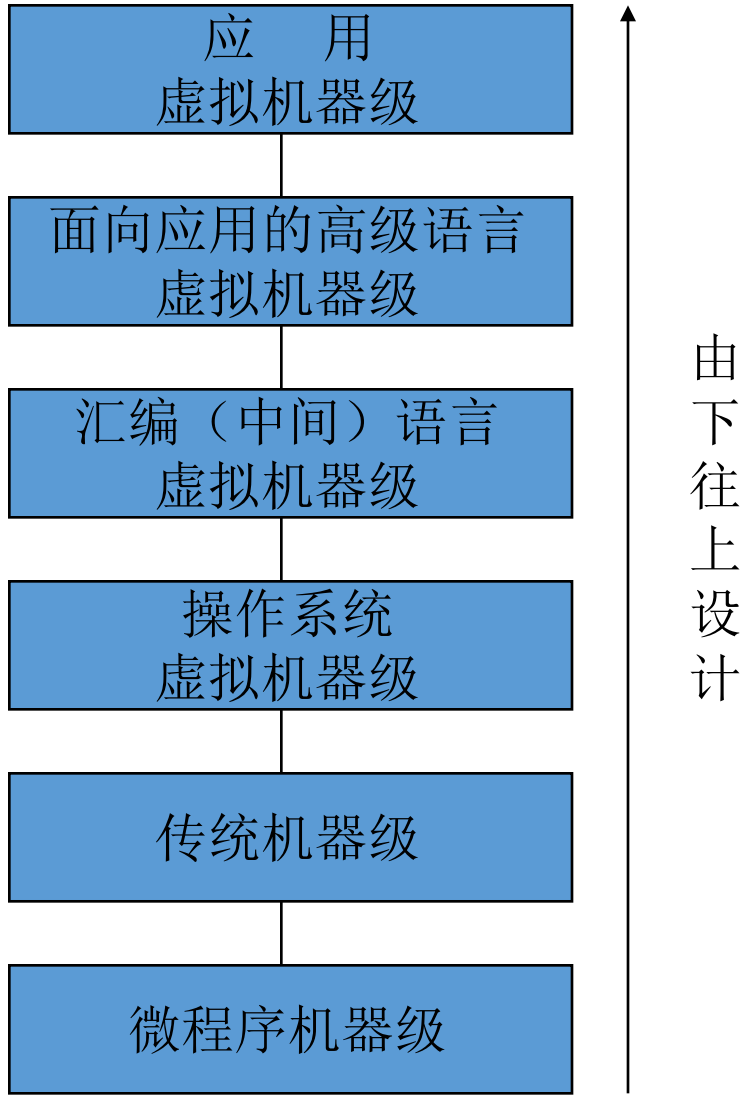
第四步：设计面向这种应用的操作系统

第五步：设计面向所用编译程序和操作系统的机器语言

第六步：设计面向机器语言的微指令机器硬件实现

由下往上(Bottom-up)

- 设计过程：
  - 根据当时的器件水平，设计微程序机器级和传统机器级。
  - 根据不同的应用领域设计多种操作系统、汇编语言、高级语言编译器等。
  - 最后设计面向应用的虚拟机器级。
- 应用场合：在计算机早期设计中（60～70年代）广为采用
- 特点：
  - 容易使软件和硬件脱节
  - 软件被动，某些性能指标不确切
  - 整个计算机系统的效率降低。



由中间开始(Middle-out)

- 设计过程：
  - 首先定义软硬件的分界面，包括：指令系统、存储系统、输入输出系统、中断系统、硬件对操作系统和编译系统的支持等
  - 然后各个层次分别进行设计：
    - 软件设计人员设计操作系统、高级语言、汇编语言、应用程序等；
    - 硬件设计人员设计传统机器、微程序、硬联逻辑等
- 应用场合：用于系列机的设计
- 特点：
  - 软硬件的分界面在上升；
  - 硬件价格下降，软件价格上升；
  - 软硬件人员结合共同设计，有利于缩短周期
  - 随着硬软件技术的发展，设计的中间点由上升的趋势。

计算机系统的设计步骤

- 需求分析
  - 主要在应用环境、所用语言的种类及特性、对OS的特殊要求
  - 所用外设特性、技术经济指标、市场分析等方面。
- 需求说明
  - 主要包括设计准则、功能说明、器件性能说明等。
- 概念性设计
  - 进行软、硬件功能分析
  - 确定机器级界面，如数据表示、指令系统、寻址方式、存储机构、中断系统、输入输出系统、总线结构等
  - 可考虑几种方案
- 反复进行优化设计及评价

软件、应用、器件对系统结构的影响

- 软件对系统结构的影响
- 应用系统对系统结构的影响
- 器件发展对系统结构的影响

软件的可移植性（Probability）

- 是指软件不用修改或只需经少量加工就能由一台机器搬到另一台机器上运行。
  - 统一高级语言
  - 采用系列机思想
  - 模拟与仿真

统一高级语言

- 采用与硬件平台无关的高级程序设计语言标准如FORTRAN、COBOL等
- 不同用途的高级语言有不同的语法结构和语义
- 人们对语言的基本结构看法不一（例如 GOTO 语句）
- 即使同一种高级语言在不同厂家的机器上也不能完全通用（例如不同的操作系统）
- 受习惯势力阻扰，不愿放弃惯用的语言

采用系列机

- 在软、硬件界面上确定好一种系统结构，之后软件设计者按此设计软件，硬件设计者根据机器速度、性能、价格的不同，选择不同的器件，在用不同的硬件技术和组成、实现技术，研制并提供不同档次的机器。
  - IBM/360,370
  - DEC PDP-11 VAX-11/780,750
  - Intel 80x86系列：8086,80286,80386,PI,P2,P3,P4
  - IBM P系列Unix服务器,p690,p670,p660,p665等
- 系列内各档机器具有相同的机器属性，因此按此属性编制的机器语言程序及编译程序能不加修改的通用于各档机器。
- 向上兼容 V.S. 向下兼容
- 向前兼容 V.S. 向后兼容

举例1

- 增加字符数据类型和指令，支持事务处理
  - 判断：不改变原有系统结构，满足软件向后兼容。
  - 结论：可采用
- 为增强中断处理功能，将中断分级4级改为5级，并重新调整中断响应的优先次序
  - 判断：中断系统属系统结构，改变
  - 结论：不可采用

举例2

- 在CPU和主存之间增设Cache，克服因主存访问速度过低而造成性能瓶颈
  - 判断：是否增设Cache，不属于
  - 结论：可采用
- 为解决计算误差大，将机器中浮点数的下溢处理方法由原来的恒置“1”法，改为ROM存放下溢处理结果的查表舍入法
  - 判断：不属于
  - 结论：可采用

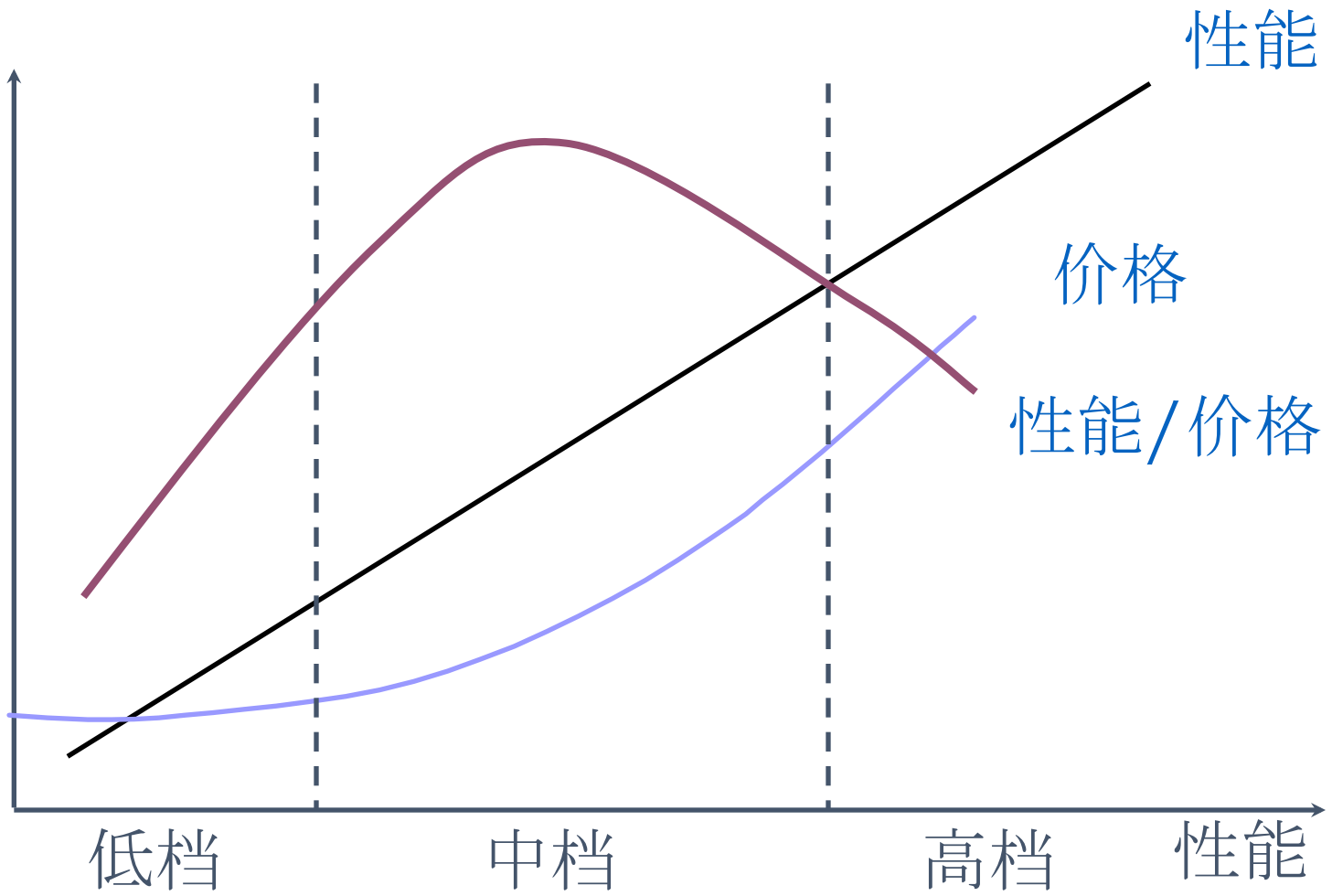
举例3

- 为增加寻址灵活性和减少平均指令字长，将原等长操作码改为3类不同码长的扩展操作码，将原操作数寻址方式由操作码指明改为如VAX-11那种寻址方式位字段指明
  - 判断：改变系统结构
  - 结论：不可采用
- 把原0号通用寄存器改为堆栈指示器
  - 判断：属于系统结构
  - 结论：不可采用

举例4

- 将CPU与主存间的数据通路宽度由16位扩展成32位，加快主机内部信息的传送
  - 判断：不属于
  - 结论：可采用
- 为减少公用总线的使用冲突，将单总线改为双总线
  - 判断：不属于
  - 结论：可采用

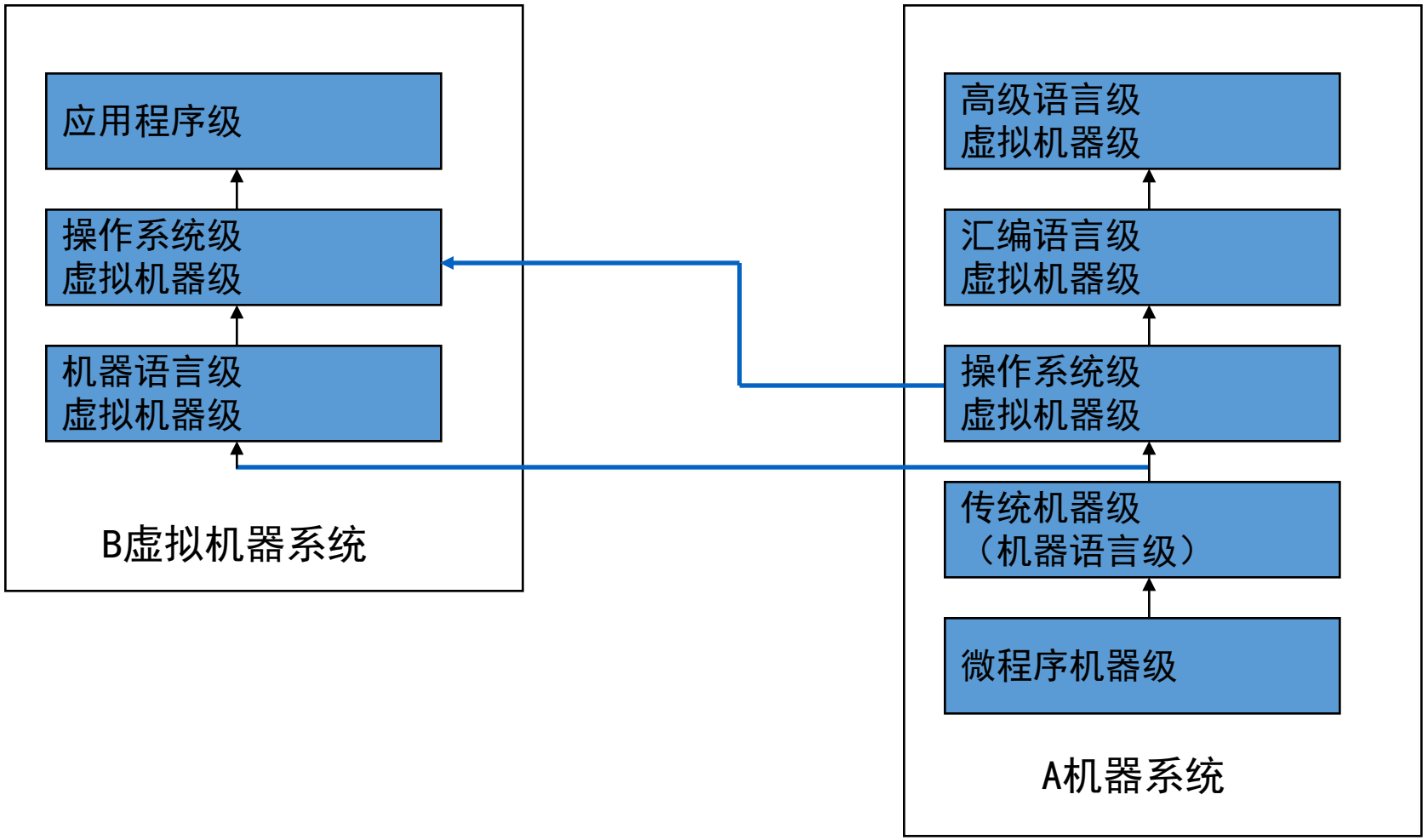
系列机的性能价格比



模拟 Simulation

- 用机器语言程序解释实现软件移植的方法。
  - 进行模拟工作的A机称为宿主机（Host Machine）
  - 被模拟的B机称为虚拟机（Virtual Machine）
  - 所有为各种模拟所编制的解释程序通称为模拟程序，编制非常复杂和费时
  - 只适合于移植运行时间短，使用次数少，而且在时间关系上没有约束和限制的软件；

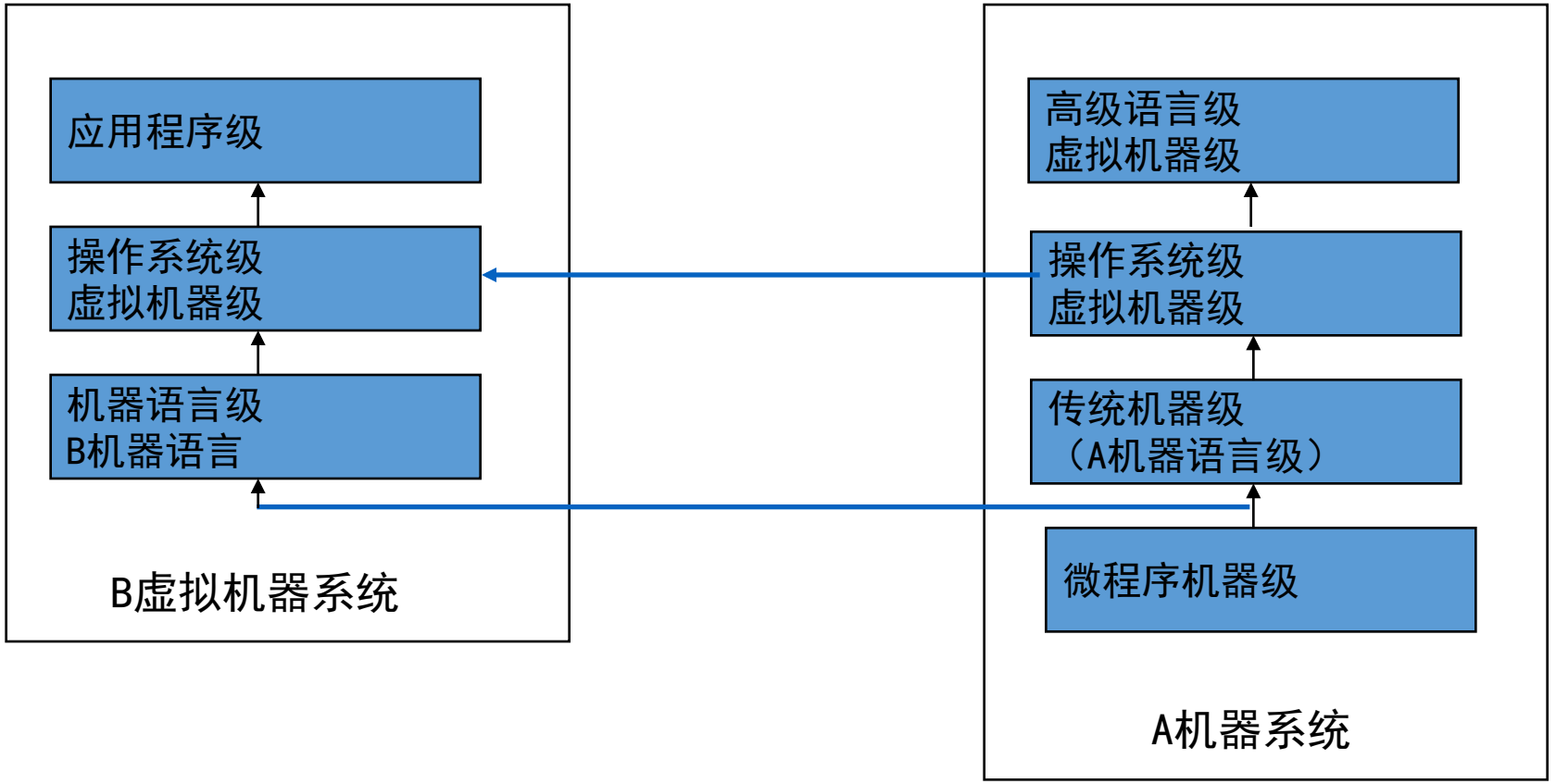
用模拟方法实现应用软件的移植



仿真 Emulation

- 用微程序直接解释另一种机器指令的方法。
  - 进行仿真工作的A机称为宿主机
  - 被仿真的B机称为目标机（Target Machine）
  - 所有为仿真所编制的解释微程序通称为仿真微程序；

用仿真方法实现应用软件的移植



仿真与模拟的区别

- 解释用的语言不同
- 解释程序所存的位置不同：仿真存在控制寄存器，模拟存在主存中
- 说明：
  - 模拟适用于运行时间不长、使用次数不多的程序
  - 仿真提高速度，但难以仿真存储系统、I/O系统，只能适用于系统结构差异不大的机器间；
  - 在开发系统中，两种方法共用

模拟与仿真的比较

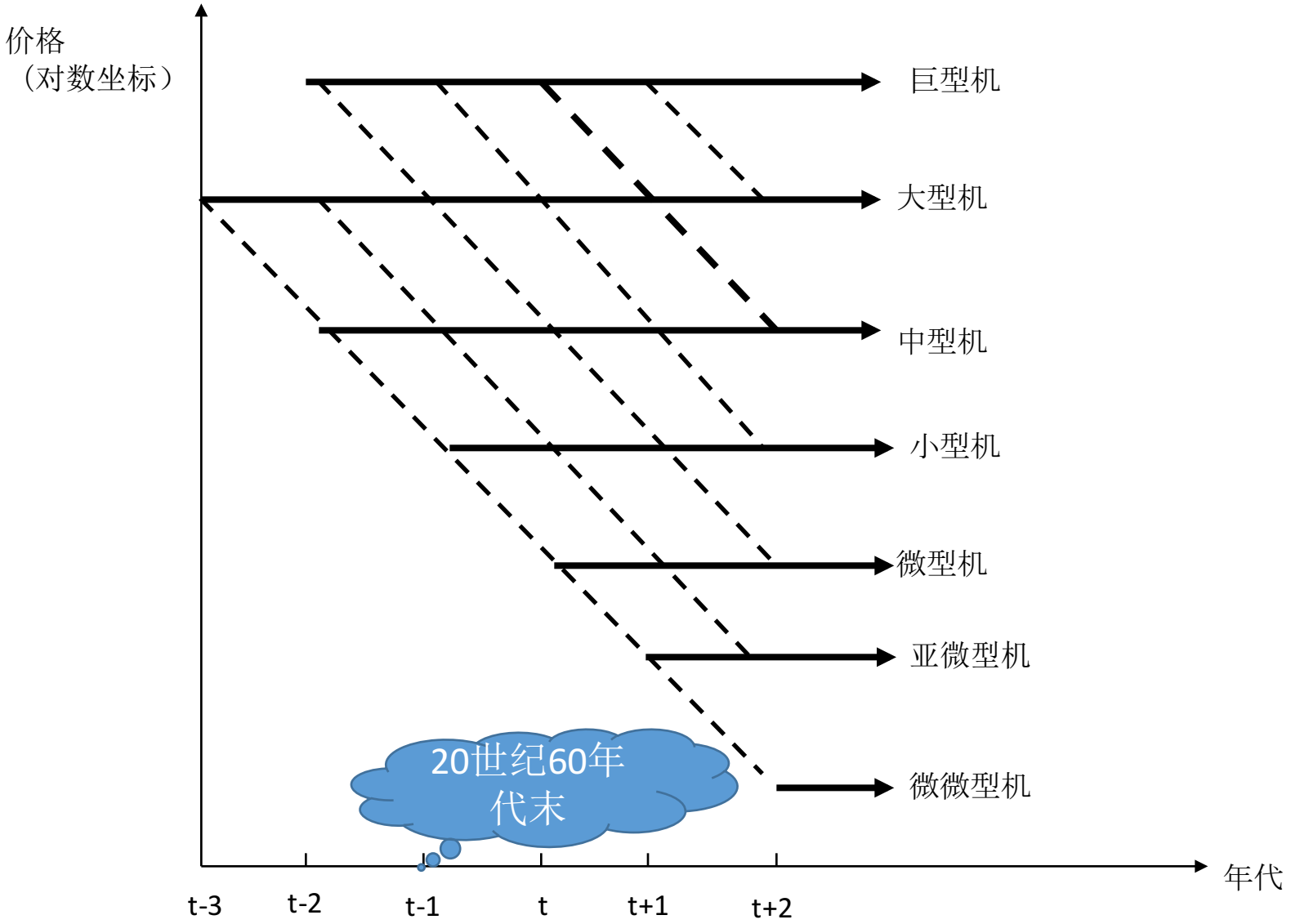
项目	模拟	仿真
优点	可实现结构差别大的机器间软件移植	速度较快
缺点	运行速度低，实时性差，模拟程序复杂	机器结构差别大时，仿真困难
适用场合	运行时间短，使用次数少，无时间关系约束的软件	频繁使用且易于仿真的指令

三种方法的比较

- 采用统一高级语言最好，是努力的目标
- 系列机是暂时性方法，也是目前最好的方法
- 仿真的速度低，芯片设计的负担重
  - 目前用于同一系列机内的兼容
  - 1/10～1/2的芯片面积用于仿真
- 发展异种机通过网络互联是实现软件移植的新途径

应用系统对系统结构的影响

- 应用要求：高速度、大容量、大吞吐率
- 应用场合：大、中、小、巨、微型机
- 大、巨型机趋势：研究新的系统结构、组成技术，并推广，向通用结构发展；
- 中、小、微型机趋势：保持价格基本不变，提高性能；保持性能基本不变，降低价格。
- 从系统结构的观点来看，各型（档）计算机的性能随时间下移，实质上就是在低档（型）机上引用，甚至照搬高档（型）机的系统结构和组成。



各型机器价格性能随时间变化的趋势

价格和应用对系统结构的影响

- 要全面评价一个系统结构，既要考虑性能又要考虑价格。
- 当两个系统的功能类似或性能接近时，性价比的比较才有意义。
- 改进系统结构应使性能或价格产生较小变化，以获得更好的性价比。
- 改进系统结构可提高系统的绝对性能，并使价格的增加比较合理。
- 针对特殊负载（特殊应用）的专用计算机系统结构往往具有高效率，但缺乏通用性，市场面小。
- 通用系统结构可适应各种应用场合，市场面大，但效率低。
- 设计的出发点是使专用系统结构的高效率与通用系统结构的广泛市场成均势。

计算机的四类应用

- 数据处理（Data Processing）
  - 可计算性
- 信息处理（Information Processing）
  - 数据的可管理性
- 知识处理（Knowledge Processing）
  - 数据的可组织性、可理解性
- 智能处理（Intelligence Processing）
  - 可智能性

非用户片、现场片和用户片

- 非用户片：也称通用片，其功能是由器件厂家生产时已确定的，器件的用户（即机器设计者）只能使用，不能改变器件内部功能
- 现场片：是用户根据需要可改变器件内部功能的芯片
- 用户片：是专门按用户要求生产的高集成度VLSI器件
  - 全用户片：是完全按用户要求设计的用户片
  - 半用户片：是基本按通用片进行生产，最后按用户要求再制作的用户片，如门阵列、门-触发器阵列等

器件发展对系统结构的影响

- 器件发展过程：
  - 通用片→现场片→半用户片→用户片
- 器件的发展推动系统结构与组成技术的发展，同样系统结构的发展要求器件不断发展。
- 新结构的使用，取决于器件发展能否提供可能
  - 器件性能/价格提高，使新结构、组成下移速度更快
  - 器件的发展，推动算法、语言的发展
  - 器件的发展，改变了逻辑设计方法。

综述

- 软件是促使计算机系统结构发展的最重要的因素
  - 没有软件，机器就不能运行，所以为了能方便地使用现有软件，就必须考虑系统结构的设计
  - **软件最重要**
- 应用需求是促使计算机系统结构发展的最根本的动力
  - 机器是给人用的，我们追求更快更好，机器就要做得更快更好
  - **需求最根本**
- 器件是促使计算机系统结构发展最活跃的因素
  - 没有器件就产不出电脑，器件的每一次升级就带来计算机系统结构的改进
  - **器件最活跃**

并发性发展及计算机系统的分类

- 并行性概念
- 计算机系统的并行性发展
- 并行处理系统的结构与多机系统的耦合度
- 计算机系统的分类

并行性（Parallelism）

- 只要在同一时刻或是在同一时间间隔内完成两种或两种以上性质相同或不同的操作或运算，它们在时间上能互相重叠。
  - 同时性（Simultaneity）：两个或多个事件在同一时刻发生。
  - 并发性（Concurrency）：两个或多个事件在同一时间间隔内发生。

并行性的不同等级

- 从计算机系统执行程序的角度看
  - 指令内部：一条指令内部的各个微操作之间的并行
  - 指令之间：多条指令的并行
  - 任务或进程之间：多个任务或程序段的并行执行
  - 作业或程序之间：多个作业或多道程序的并行

并行性的不同等级

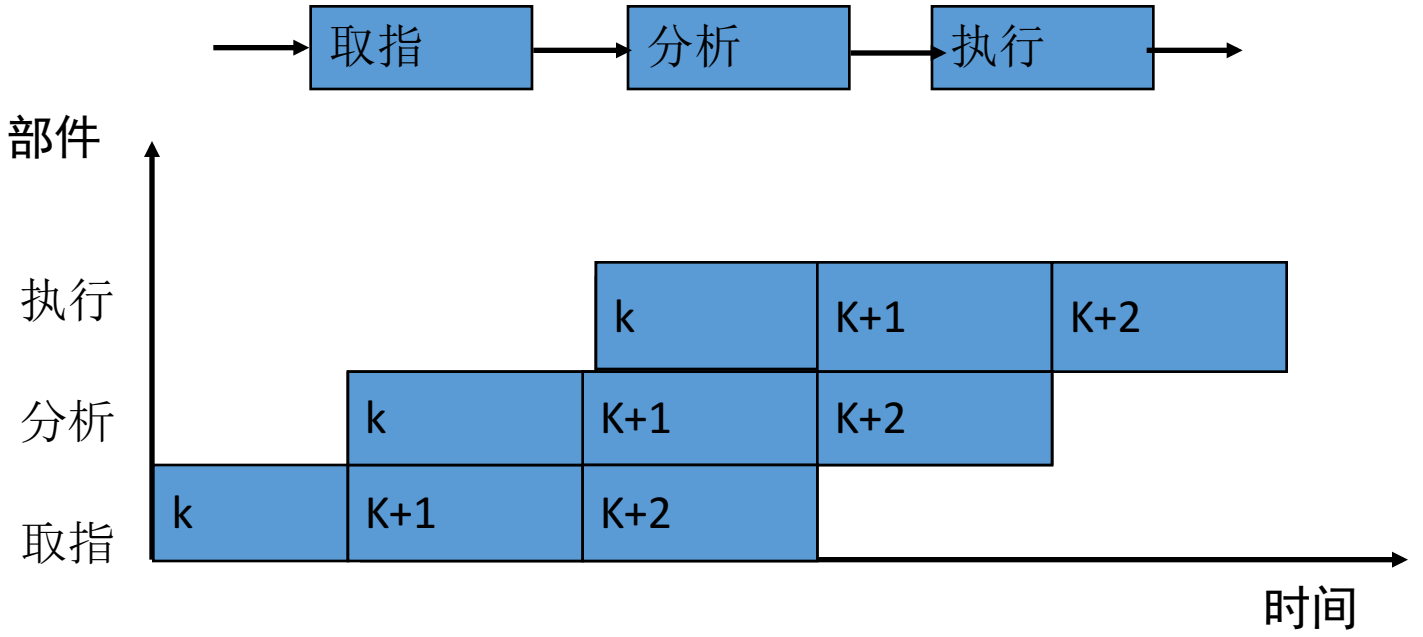
- 从计算机系统中数据处理的角度看
  - 位串字符：同时只对一个字的一位进行处理
  - 位并字符：同时对一个字的全部位进行处理
  - 位片串字并：同时对许多字的同一位（称位片）进行处理
  - 全并行：同时对许多字的全部或部分位组进行处理

并行性的不同等级

- 从计算机信息加工的不同步骤和阶段来看
  - 存储器操作的并行：存储器中多字并行比较、检索、更新、变换等操作
  - 处理器操作步骤并行：指令处理和运算操作的执行步骤在时间上重叠流水地进行。
  - 处理器操作并行：通过重复设置大量处理单元，在同一控制器的控制下按同一指令要求对向量、数组中的元素同时操作
  - 指令、任务、作业并行：多个处理机同时对多条指令和相关数据进行处理

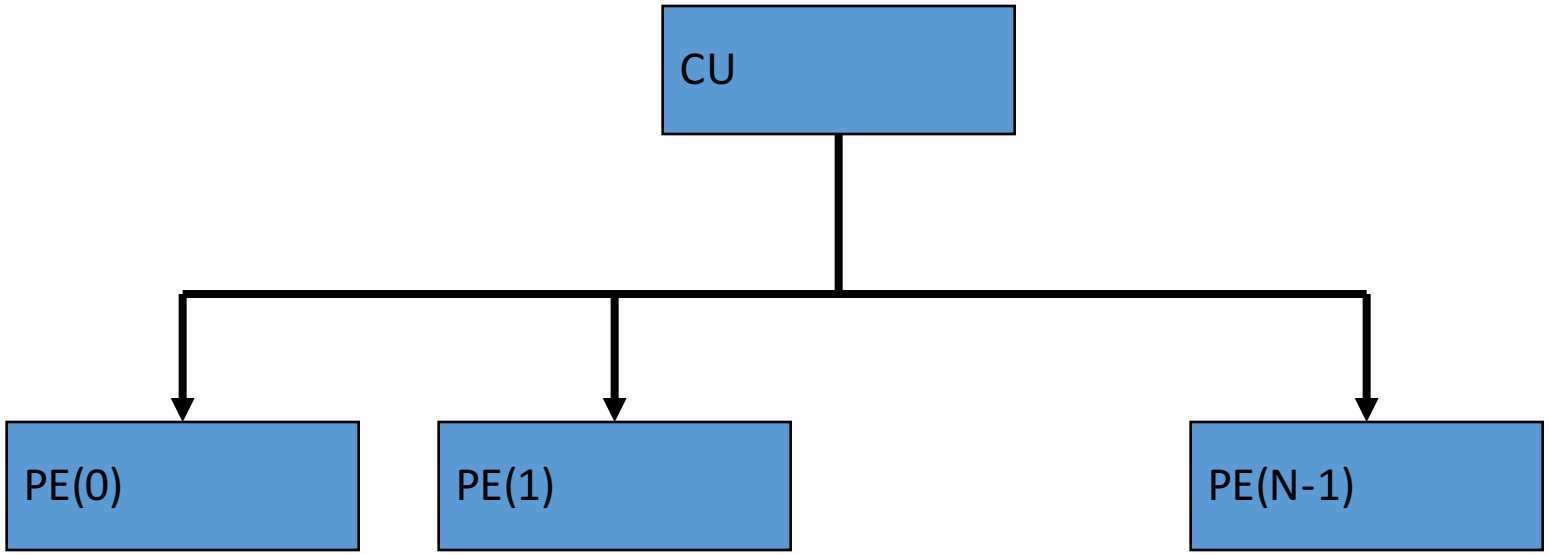
并行性开发途径1

- 时间重叠（Time Interleaving）是在并行性概念中引入时间因素，让多个处理过程在时间上相互错开，轮流重叠地使用同一套硬件设备的各个部分，以加快硬件周转而赢得速度。
- 举例：指令流水线系统



并行性开发途径2

- 资源重复（Resource Replication）：是在并行性概念中引入空间因素，通过重复设置硬件资源来提高可靠性或性能。



并行性开发途径3

- 资源共享（Resource Sharing）：是利用软件的方法让多个用户按一定时间顺序轮流地使用同一套资源，以提高其利用率，这样也可以提高整个系统的性能。

并行处理计算机的结构

- 流水线计算机（时间重叠）
  - 主要通过时间重叠，让多个部件在时间上交错重叠地并行执行运算和处理，以实现时间上的并行。
- 阵列处理机（资源重复）
  - 主要通过资源重复，设置大量算术逻辑单元，在同一控制部件作用下同时运算和处理，以实现空间上的并行。
- 多处理机系统（资源共享）
  - 主要通过资源共享，让共享输入/输出子系统、数据库资源及共享或不共享贮存的一组处理机在统一的操作系统全盘控制下，实现软件和硬件各级上相互作用，达到时间和空间上的异步并行。

多机系统

- 指的是多处理机系统和多计算机系统
- 多处理机系统：是由多台处理机组成的单一计算机系统，各处理机都可有自己的控制部件，可带自己的局部存储器，能执行各自的程序，但受逻辑上统一的操作系统控制。处理机之间以文件、单一数据或向量、数组等形式进行交互作用，实现各级别的并行。
- 多计算机系统：是由多台独立的计算机组成的系统，各计算机分别在逻辑上独立的操作系统控制下运行，机间可以互不通信，即使通信也只是经通道或通信线路以文件或数据集形式进行，实现多个作业的并行。

多机系统的耦合度

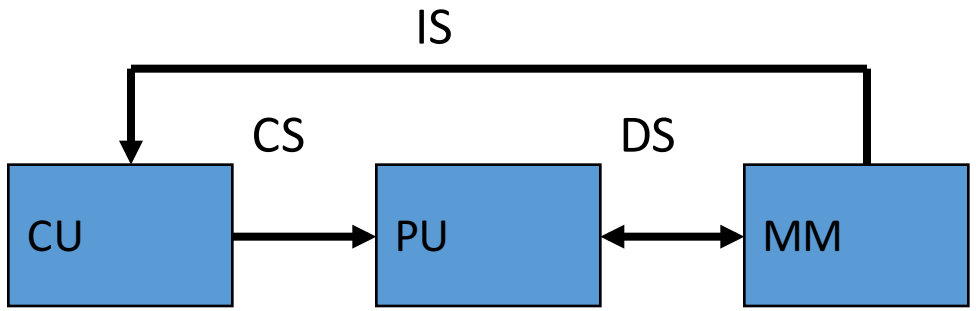
反映多机系统中各级机器之间物理连接的紧密程度和交叉作用能力的强弱。

- 最低耦合系统（Least Coupled System）：各种脱机系统
- 松散耦和系统（Loosely Coupled System）：如果多台计算机通过通道或通信线路实现互连，共享某些磁带、磁盘等外围设备，以较低频带在文件或数据集一级相互作用。间接耦合系统
- 紧密耦合系统（Tightly Coupled System）：如果多台机器之间通过总线或高速开关互连，共享主存，并有较高的信息传输速度，可以实现数据集一级、任务级、作业级的并行。

Michael J,Flynn分类

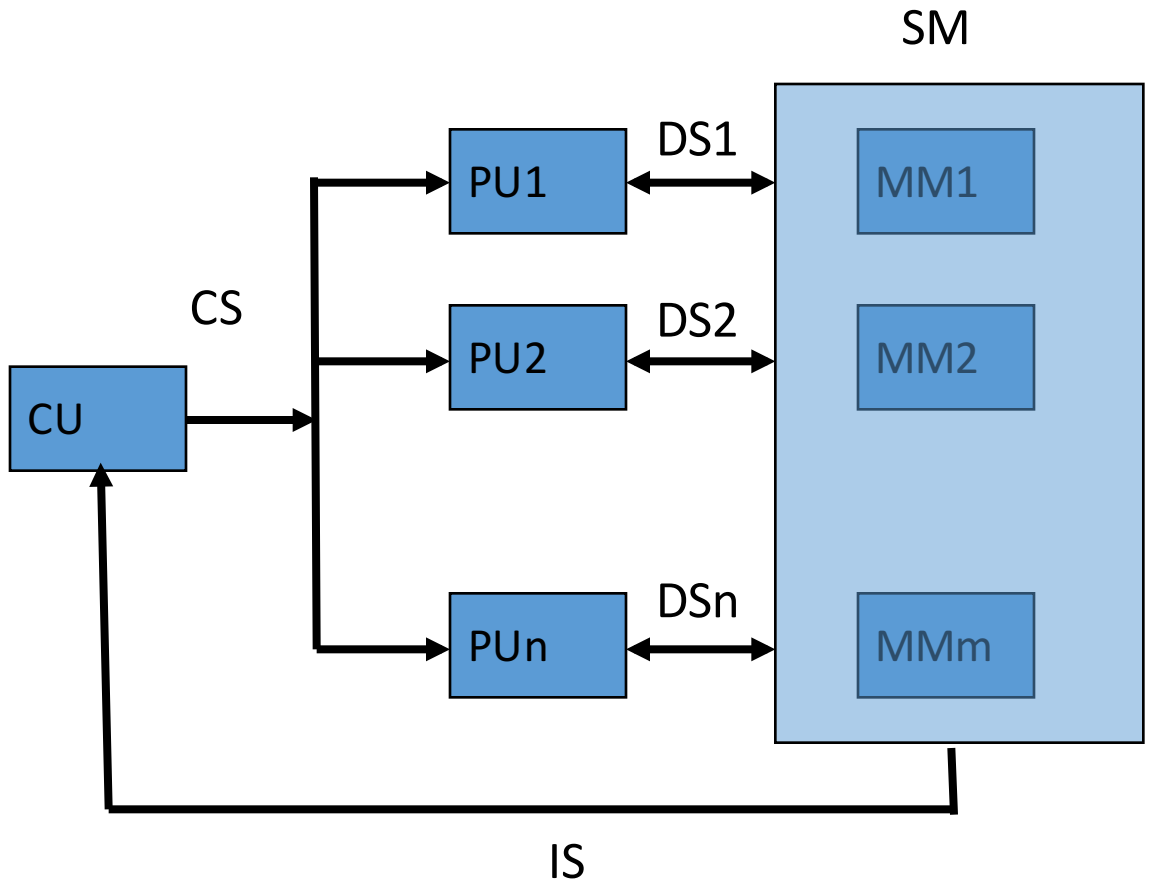
- 指令流：是指机器执行的指令序列。
- 数据流：是指指令流调用的数据序列，包括输入数据和中间结果。
- 多倍性：是指在系统性能瓶颈部件上处于同一执行阶段的指令或数据的最大可能个数。

- 单指令流单数据流（SISD, Single Instruction Stream Single Data Stream）
- 指令部件每次只对一条指令译码，只对一个操作部件分配数据
- 例如流水方式的单处理机



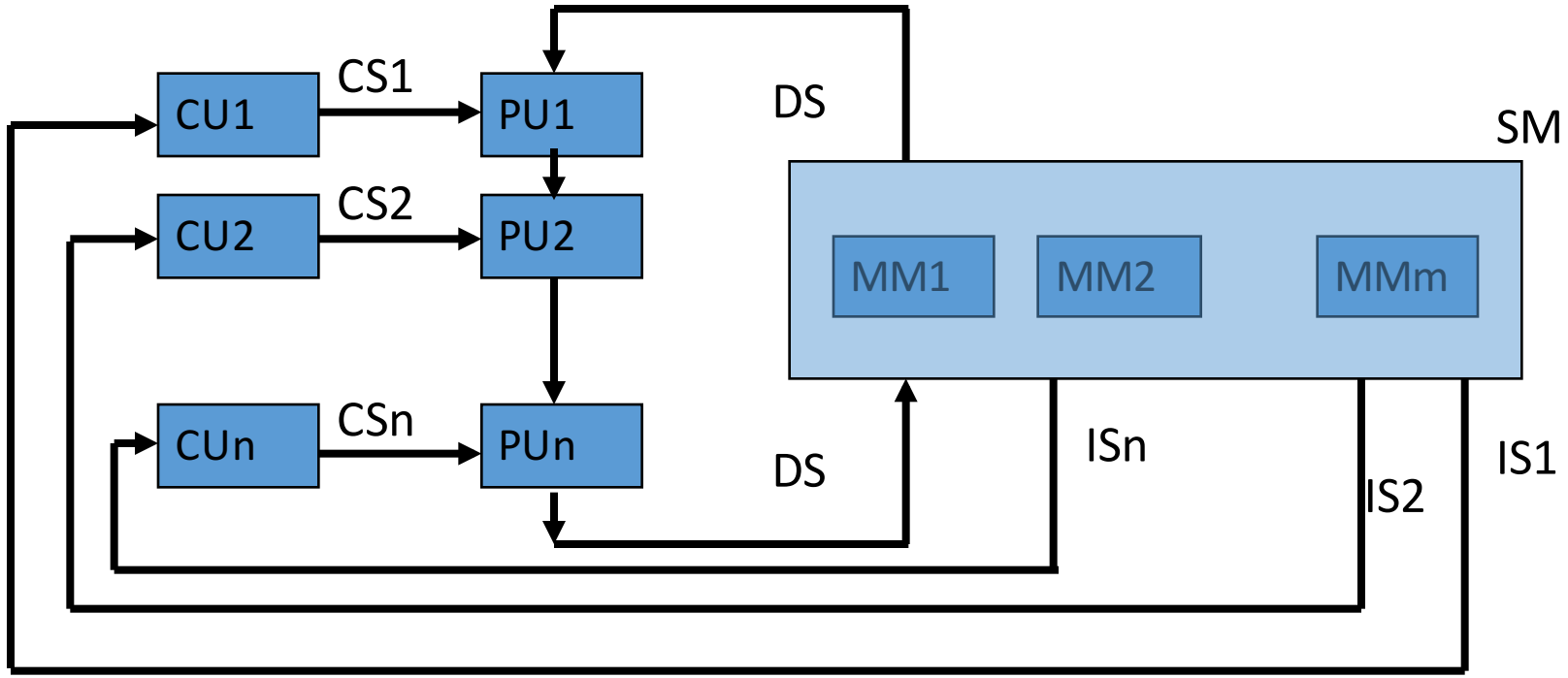
CU：控制部件，PU：处理机，MM：贮存模块，SM：共享主存
IS：指令流，CS：控制流，DS：数据流

- 单指令流多数据流（SIMD, Single Instruction Stream Multiple Data Stream）
- 例如阵列处理机和相联处理机



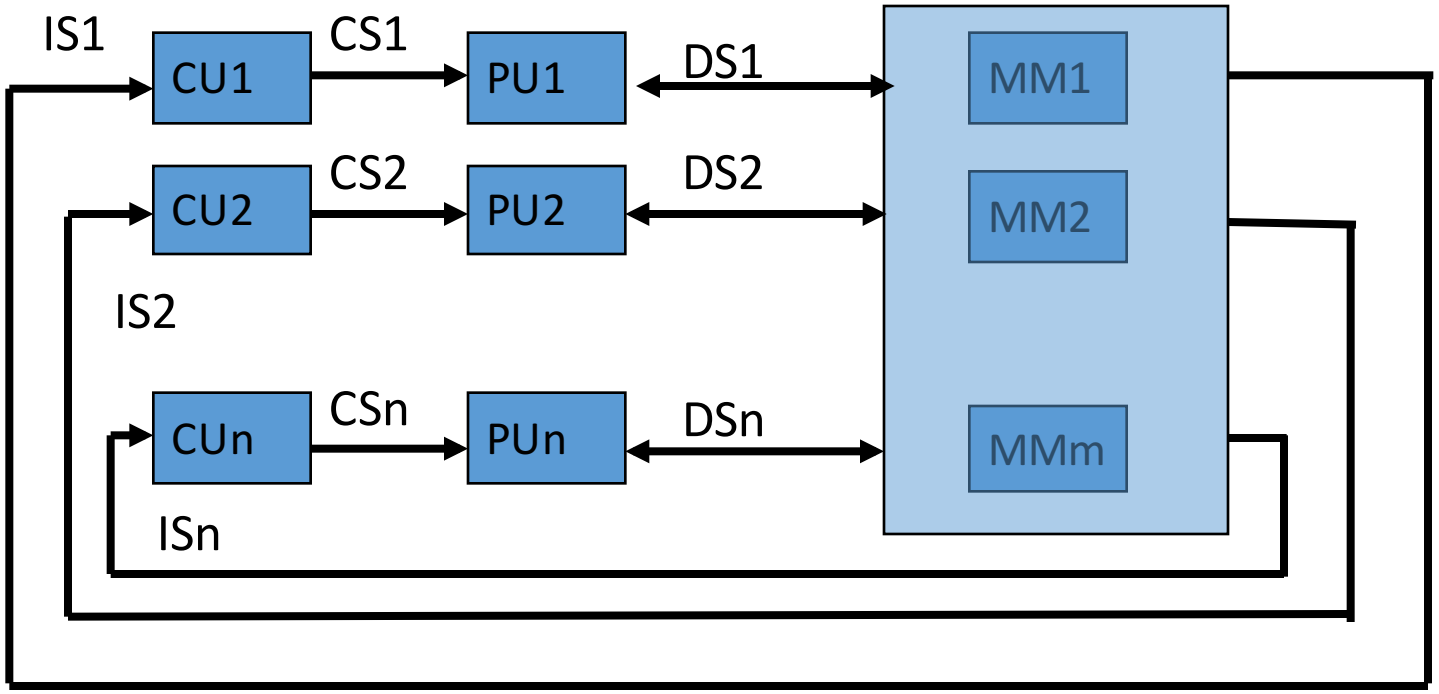
CU：控制部件，
PU：处理机，
MM：贮存模块，
SM：共享主存
IS：指令流，
CS：控制流，
DS：数据流

- 多指令流单数据流（MISD, Multiple Instruction Stream Single Data Stream）
- 例如脉动阵列流水机



CU：控制部件，
PU：处理机，
MM：贮存模块，
SM：共享主存
IS：指令流，
CS：控制流，
DS：数据流

- 多指令流多数据流（MIMD, Multiple Instruction Stream Multiple Data Stream）
- 在作业、任务、指令和数组等各个层级实现全面并行



CU：控制部件，
PU：处理机，
MM：贮存模块，
SM：共享主存
IS：指令流，
CS：控制流，
DS：数据流

David J.Kuck分类

- 用指令流和执行流（Execution Stream）及其多倍性来描述计算机系统总控制器的结构特征。
  - SISE：单处理机系统
  - SIME：多操作部件的处理机
  - MISE：带指令级多道程序的单处理机
  - MIME：多处理机

冯泽云分类

- 提出用数据处理的并行度来定量地描述各种计算机系统特性。
  - WSBS（字串位串）：位串处理方式，每次只处理一个字中的一个位
  - WSBP（字串位并）：字（字片）处理方式，每次处理一个字中的n位
  - WPBS（字并位串）：位（位片）处理方式，一次处理m个字中的1位
  - WPBP（字并位并）：全并行处理方式，一次处理m个字，每个字为n位