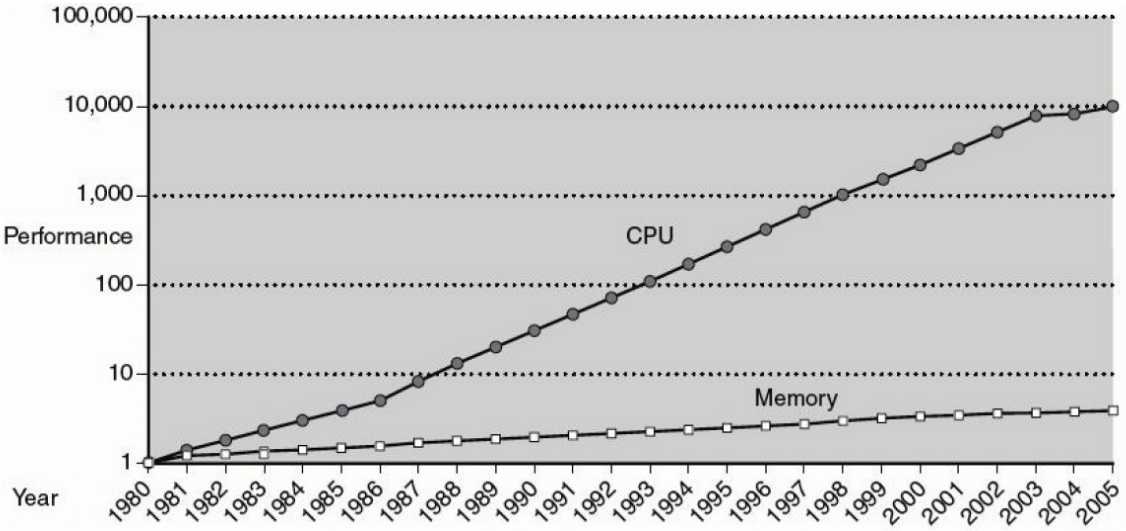
# 计算机组成 Lecture Note 5

本文主要探讨存储器相关的内容。

## 构想和局部性原理

对于程序员而言，最理想的存储器应当*速度又快*、*容量又大*。随着 CPU 的处理能力和速度不断提升，为了让存储器的速度能与之协调，这样的需求正在持续增加。

很难直接造出这样的存储器。原因在于和 CPU 相比，半导体存储器技术的发展速度严重滞后。




幸好，通过给存储器分层次和运用**局部性原理（Principle of Locality）**，我们能够近似地做到这一点。

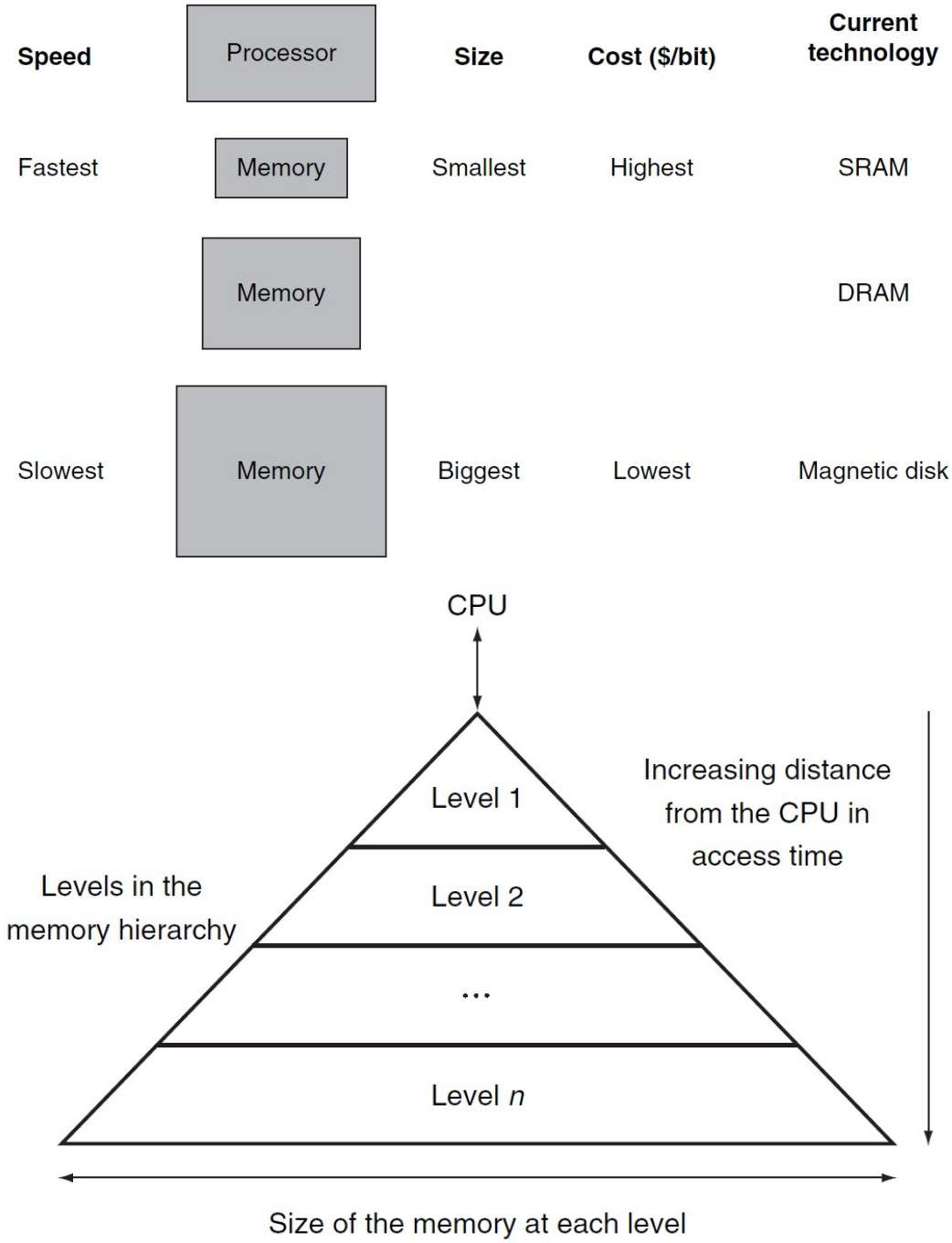
局部性原理表明了在任何时间内，程序访问的只是地址空间相对较小的一部分内容。有两种不同类型的局部性：

1. **时间局部性（Temporal Locality）**：如果某个数据项被访问，那么在不久的将来它可能再次被访问。
2. **空间局部性（Spatial Locality）**：如果某个数据项被访问，与它地址相邻的数据项可能很快将被访问。

程序的局部性起源于简单自然的程序结构。例如循环语句体现了时间局部性，而串行执行语句体现了空间局部性。因此，用局部性原理来指导存储器的设计是一定程度上合理的。

应用局部性原理，我们将存储器组织成为**存储器层次结构（Memory Hierarchy）**。存储器层次结构由不同速度和容量的多级存储器构成。更快的存储器单位存储空间的造价比慢的要高，同时总存储量上也要更小。

层次结构可以用两张图来表述：




存储器层次结构利用了时间局部性，将最近被访问的数据项放在靠近处理器的地方；同时也利用了空间局部性，将一些包含连续字的块移至较高层次。

在大多数系统中，存储器确实以层次结构实现，这意味着上一层包含的数据是下一层的子集。完整的数据包含在最低的一级，而处理器只访问最高的一级。

通过适当的层次间的调度，我们就能实现存储器速度又快（就最高一级而言）、容量又大（就最低一级而言）的构想。

# 基础概念

下面罗列一些会用到的术语：

- 两级层次进行信息交换的最小单元称为**块（Block）**或**行（Line）**。本文偏向于用“块”。
- 处理器对数据的访问也可更正式地称为**引用（Reference）**。
- 对于上下两层，如果存储器要访问的数据在上一层的某个块里。就称为**命中（Hit）**。否则称为**缺失（Miss）**。相应地可以定义**命中率（Hit Rate）**和**缺失率（Miss Rate）**。两者可以用来衡量存储器层次结构的性能。

# 存储技术概览

存储器层次结构常用的四种存储技术为：**SRAM（Static Random Access Memory，静态随机存储器）**、**DRAM（Dynamic Random Access Memory，动态随机存储器）**、**闪存（Flash Memory）**、**磁盘（Magnetic Memory）**。

从前向后，单位存储空间造假逐步下降，访问时间逐步升高。

## SRAM 和 DRAM

SRAM 是一种集成电路，数据以阵列（array）的形式存储，一般有一个端口用于读或者写。SRAM 对于任意数据项的访问耗时固定，但对于读和写耗时可能会有区别。

DRAM 的造价比 SRAM 便宜。其单位存储空间所占用的物理意义上的空间要比 SRAM 少很多，因用相同量的硅去造，DRAM 的总存储量会更大。

两者在存储上的一个重要区别在于是否有**刷新机制**。

SRAM 中，一个 bit 通常由 6-8 个晶体管来存储，只要给 SRAM 上电，其中的数据就可以无限期存储。

DRAM 的一个 bit 由一个电容存储，数据会以电荷的形式保存在电容中，用电势高低表示 0/1。由一个晶体管来对保存的电荷进行读或写。由于电容的特殊性，数据不能无限期地存储，而是需要定期地刷新。

在刷新的过程中，DRAM 不可读不可写。DRAM 的“动态”就反映了需要刷新这一点，与之相对的是 SRAM 的“静态”。

## 磁盘

磁盘是一种能存放海量数据的存储工具。结构：

- 盘片（Disk）：每一个盘面被分成多个磁道（Track），每一个磁道被分成多个扇区（Sectors）。扇区是保存信息的最小单元。多个盘面的同半径磁道构成柱面（Cylinder）。
- 读写头：可移动，用于读数据。每一个盘面上都有一个。读写头合并在一起，同时移动。

读写时间耗费来源：

- 进入读写队列的等待时间。
- 移动读写头到正确磁道需要的时间。
- 磁道转到正确扇区的时间。计算时一般认为*平均*转半圈可以到。
- 数据传输到处理器的时间。
- 从处理器到存储器的时间。

使用 ATA、SCSI 等智能接口可以进行优化。

## 闪存

**闪存（Flash Memory）**是电可擦写、可编程的 ROM。分为 NOR Flash 和 NAND Memory。

HHD 为闪存和磁盘结合的产物。

# 高速缓存 cache

cache 是位于处理器和主存中的一块容量并不大的 SRAM，但它在层次结构中可以起到重要的作用。

核心是四个问题：

- 怎么放？（块从下一级来，放在哪？）
- 怎么找？（CPU 请求，怎么找块是否存在？）

- 怎么换？（缺失了，块怎么换出去？）
- 怎么写？（CPU 写到 cache，怎么保持一致性？）

## 怎么放?

几种映射方式：

- 直接映射：块地址 $\mathrm{mod}$ 块数。
- 全相联映射：任何位置均可。
- 组相联映射：*直接映射*到某个组，组内放在某个块。组内 $n$ 个块就称为 $n$ 路组相联映射。

## 怎么找?

块有附属信息：

- 标志位 Tag：块地址的高位。
- 脏位：这个块是不是被 CPU 修改过。
- 有效位：这个块是不是有效的。如刚上电，所有块都无效。

Tag 长度可以根据组的大小、块的大小计算。

一般在硬件层面会并行检查。

## 怎么换?

有多种替换策略，如随机替换、先进先出替换、**最近最少使用（LRU）**替换等。一般 LRU 最复杂，但也性能最好。

LRU 一般可以通过设置**引用位（Reference Bit）**的方法来近似。

## 怎么写?

两种写 cache 的基本策略：

- **写直达（Write Through）**：同时改 cache 和低一级部分，一气呵成。
- **写回法（Write Back）**：只有 cache 块被替换时更新低一级部分。

两种写不在 cache 中（写缺失）的基本策略：

- 写分配：先读到 cache 中，再写。即为写的内容分配一个块。
- 不按写分配：只改低一级，不放到 cache 中。

一般是写回搭配写分派、写直达搭配不按写分配。即 lazy 和不 lazy 相互搭配。

## 多级 cache

cache 可以有多级。概念：

- 局部缺失率：本级 cache 缺失数除以对*本级 cache* 的访问总数。
- 全局缺失率：本级 cache 缺失数除以对*存储器*的访问总数。

缺失代价可以递归计算。

## 计算相关

**AMAT（平均存储器访问时间）**为平均每次访问存储器需要的时间。

*平均存储器停顿时间*为由于缺失导致的惩罚时间。

*平均指令访存时间*为平均每条指令，花费在访存上的时间。

> 一般而言缺失代价并不是常数，读写的缺失率、缺失代价也常常不同。因此计算可以看作是高度简化的。

# 虚拟内存

虚拟内存技术可以将磁盘的一部分当作“存储器”使用。动因：

- 消除小而受限的主存容量对程序运行及编程造成的影响。
- 允许在多个程序间有效而安全地共享内存。

虚拟内存中，块被称为**页（Page）**，访问缺失称为**缺页（Page Fault）**。

将 CPU 请求的虚拟地址转换为物理地址的机制为**地址转换（Address Translation）**。地址被划分为**虚页号（Virtual Page Number）**和**页偏移（Page Offset）**。相应地有*物理页号*，物理地址的高位为物理页号，低位为页偏移。**页偏移的位数决定了页的大小**。

依然，回答 4 个问题。

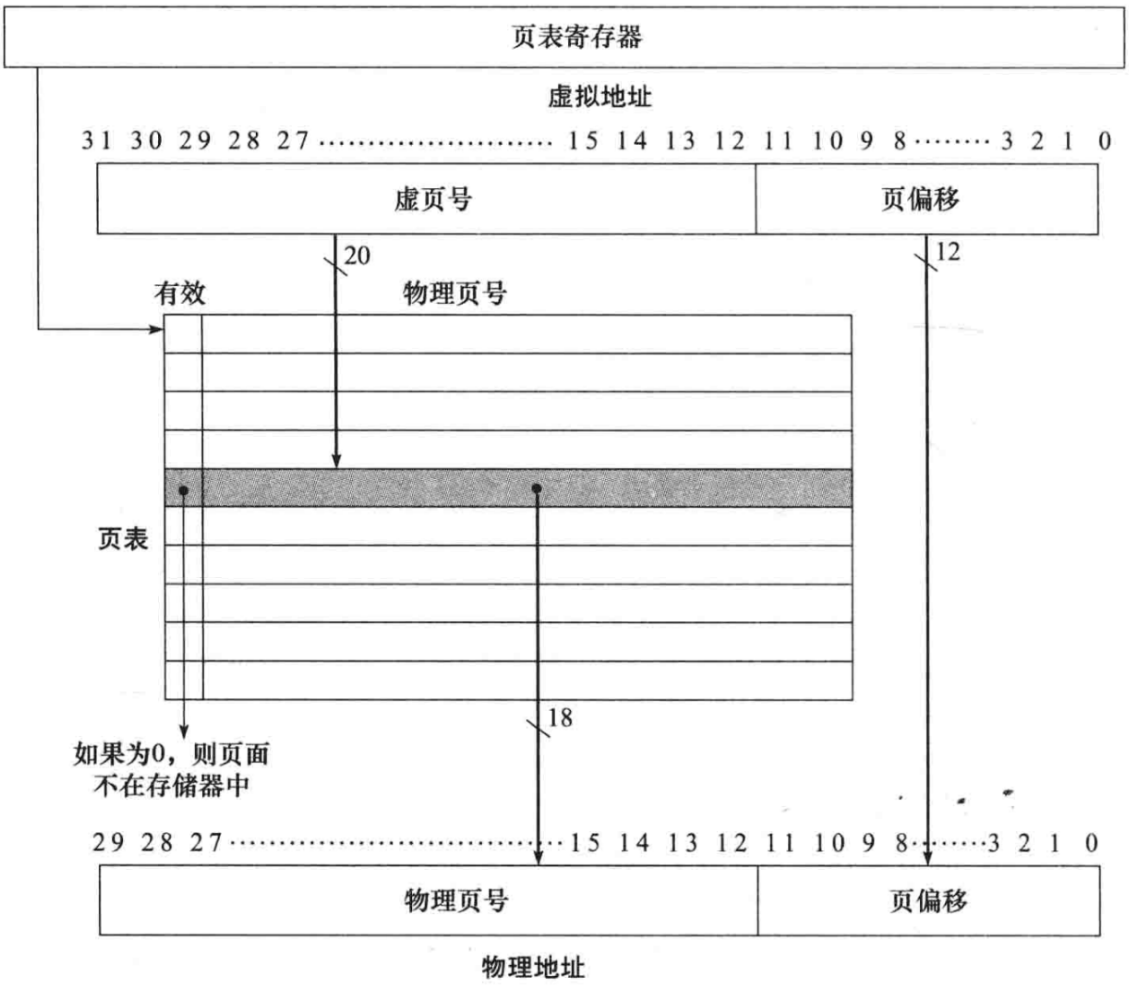
## 怎么放?

全相联。为了减小缺页率。

## 怎么找?

使用**页表（Page Table）**。其被放在存储器中，使用*虚拟页号做索引*，映射到物理页号。每个程序都有自己的页表。

硬件包含一个指向页表首地址的寄存器，即页表寄存器。图示：




页表不需要标记位，因为虚拟页号就是索引。

## 怎么换?

操作系统一开始在磁盘空间会建立**交换区（Swap Space）**，为进程中所有的页创建空间。同时定义某个数据结构，记录每个虚拟页在磁盘的存放位置。

缺页发生时，操作系统获得控制权，从磁盘中找到缺失的页，并且放到存储器中。替换方法使用 LRU。

## 怎么写?

使用写回法，写直达的代价太大了。

## 快表

页表本身也很大。有可能页表本身还要分页，被分走的放在磁盘上。同时页表放在主存上会导致每次访存都实际需要至少 2 次：一次转换出物理地址，一次读数据。

**快表（Translation-Lookaside Buffer，TLB）**是一种用于快速转换地址的 cache。它应用了时间局部性，记录了最近使用地址的映射信息。

- TLB 的表项包含有效位、脏位和引用位。
- TLB 实现的为全相联映射。一般使用随即替换。
- TLB 的标志位即为虚拟页号。
- 如果引用了 TLB 中的虚页号，那么引用位置位。
- 如果是写，那么脏位要置位。
- TLB 缺失可以用硬件或者软件处理。由于 TLB 中的页表项很少，因此缺页也会非常频繁。

TLB 可以和 cache 搭配使用。如果 TLB 命中，就可以立即用找到的物理页号和请求的页偏移拼出物理地址，然后在 cache 中寻找相关页。

搭配使用，如果缺失时的所有可能情况：

| TLB | 页表 | cache | 这种情况可能发生么？如果可能，在什么情况下发生？ |
|---|---|---|---|
| 命中 | 命中 | 缺失 | 可能，但若 TLB 命中就不可能检查页表 |
| 缺失 | 命中 | 命中 | TLB 缺失，但在页表中找到表项；重试后在 cache 中找到数据 |
| 缺失 | 命中 | 缺失 | TLB 缺失，但在页表中找到表项；重试后在 cache 中未找到数据 |
| 缺失 | 缺失 | 缺失 | TLB 缺失，随后发生缺页；重试后在 cache 中必找不到数据 |
| 命中 | 缺失 | 缺失 | 不可能：如果页不在主存中，TLB 中没有此转换 |
| 命中 | 缺失 | 命中 | 不可能：如果页不在主存中，TLB 中没有此转换 |
| 缺失 | 缺失 | 命中 | 不可能：如果页不在主存中，数据不允许在 cache 中存在 |

这按照结构层次来推理（TLB -> 页表，cache 服务于存储器）是自然的。

## 多级页表和反置页表

虚拟空间很大时，理论页表也很大。

- 页大小除以页项大小得到一个页表的总的页项数。
- 总的页项数等于虚拟页号的数目。

两种节省空间的手段：

- 多级页表：如果一个页表可以有 $t$ 项，而虚拟页号总共有 $N$ 个，那么至少需要 $\lceil \log_t N \rceil$ 级页表。从而一个页号通过多级页表拼出。
- 反置页表：物理空间是有限的。只保存物理空间实际可能装得下的页数的映射数目就是反置页表。可以通过哈希表实现地址转换。

# MMU

事实上，计算机系统设计中的几乎所有事情都是硬件和操作系统密切配合的结果。硬件层面有**MMU（存储器管理单元）**。虚拟存储器是 MMU 和 OS 协调工作的结果。

# 小结

对编程者而言，必须理解了存储器层次结构的工作方式，才能获得良好的性能。将存储器看作是一个线性的随机访问的存储设备的视角已经过时了。

# 致谢

本文的主要内容参考自：

- 上海交通大学《计算机组成》（课程代号：EI332）一课的课程材料。
- 《Computer Organization and Design: Hardware/Software Interface (5th edition)》

本文的图片来自

- 《Computer Organization and Design: Hardware/Software Interface (5th edition)》
- 《计算机组成与设计：硬件/软件接口（第 5 版）》