

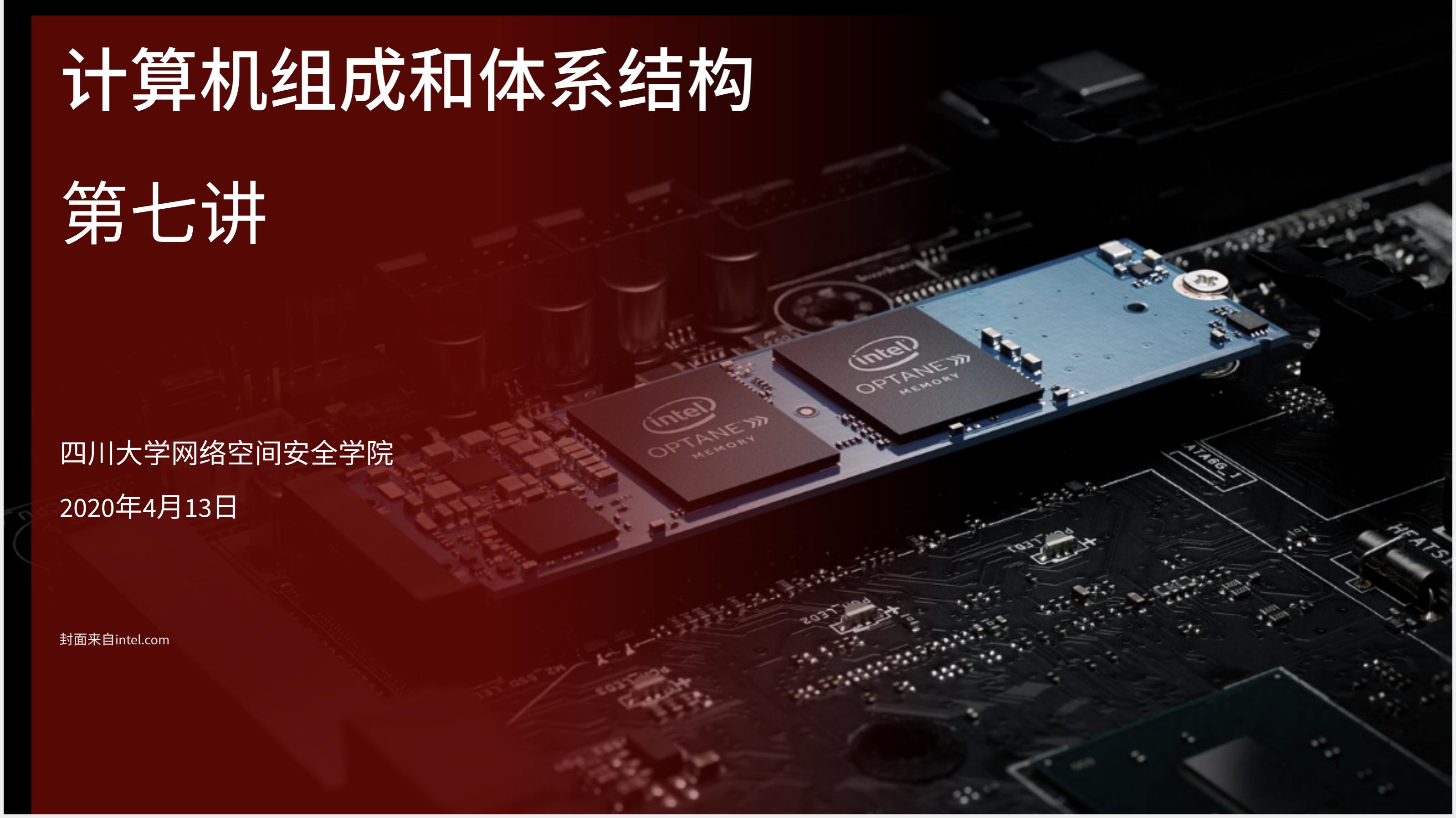
计算机组成和体系结构

第七讲

四川大学网络空间安全学院

2020年4月13日

封面来自intel.com



版权声明

课件中所使用的图片、视频等资源版权归原作者所有。

课件原创内容采用 [创作共用署名—非商业使用—相同方式共享4.0国际版许可证\(Creative Commons BY-NC-SA 4.0 International License\)](#) 授权使用。

Copyright@四川大学网络空间安全学院计算机组成与体系结构课程组，2020



上期内容回顾

- 指令流水线
 - 指令流水线的性能分析
 - 流水线冒险
 - 流水线对于指令集架构设计的影响
- 现实中的指令集架构

本期学习目标

- 掌握常见的存储器类型及特点
- 掌握存储器层次结构
- 高速缓存
 - 缓存策略的性能指标和缓存失效
 - 缓存的映射策略
 - 缓存的替换策略

中英文缩写对照表

英文缩写	英文全称	中文全称
RAM	Random Access Memory	随机存取存储器
ROM	Read-Only Memory	只读存储器
SRAM	Static Random Access Memory	静态随机存取存储器
DRAM	Dynamic Random Access Memory	动态随机存取存储器
CAM	Content-Addressable Memory	内容寻址存储器
FIFO	First-in First-out	先入先出策略
LRU	Least Recently Used	最近最少访问
EAT	Effective Access Time	有效访问时间

存储器1: 层次化存储结构

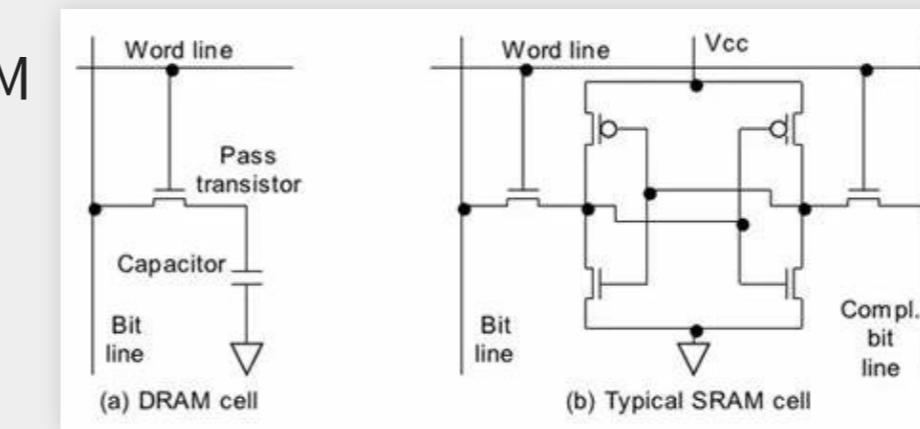
存储器类型(1)

- ROM (Read-Only Memory)
 - 数据读取速度快、存储时间长，只需要很少的电能用于维持数据
 - 例子： BIOS， CD-ROM， 闪存
- RAM (Random Access Memory)
 - 可以对随机位置的数据进行读写，需要通电才能保存数据
 - 例子： 主存

存储器类型(2)

RAM按照实现方式的不同又可以分为两类：

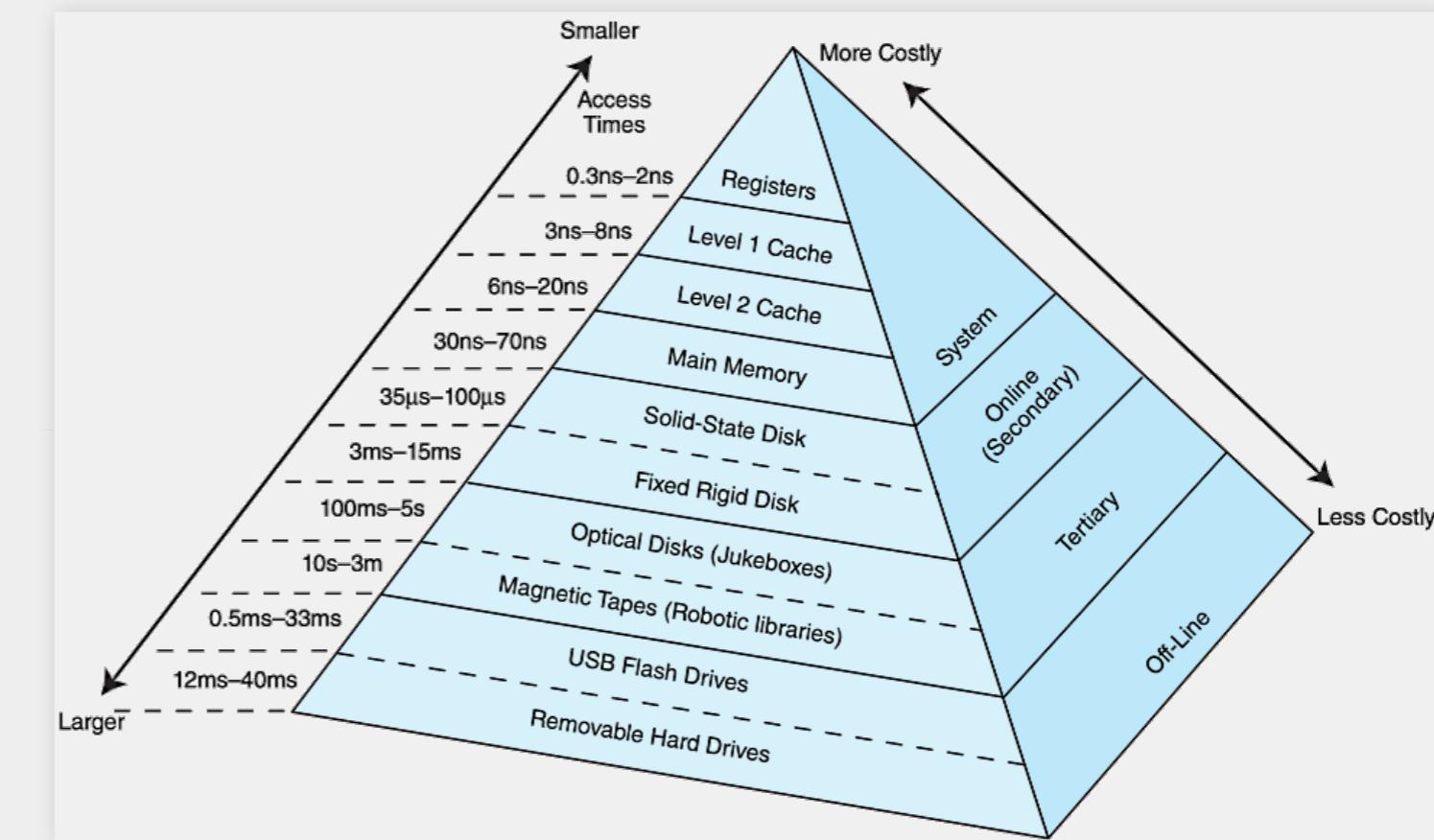
- 静态随机访问存储器 (Static Random Access Memory, SRAM)
 - 采用类似D触发器的电路实现
- 动态随机访问存储器 (Dynamic Random Access Memory, DRAM)
 - 采用电容实现，使用过程中电容会漏电因此每隔几毫秒需要充电
- SRAM和DRAM的性能比较
 - 访问速度：SRAM (~10ns) < DRAM (~50 ns)
 - 能耗：SRAM < DRAM
 - 电路复杂度：SRAM > DRAM
 - 密度：SRAM < DRAM
 - 造价：SRAM > DRAM



图片来源：<https://www.quora.com/What-is-the-difference-between-SRAM-and-DRAM>

通常采用DRAM实现主存储器，采用
SRAM实现高速缓存

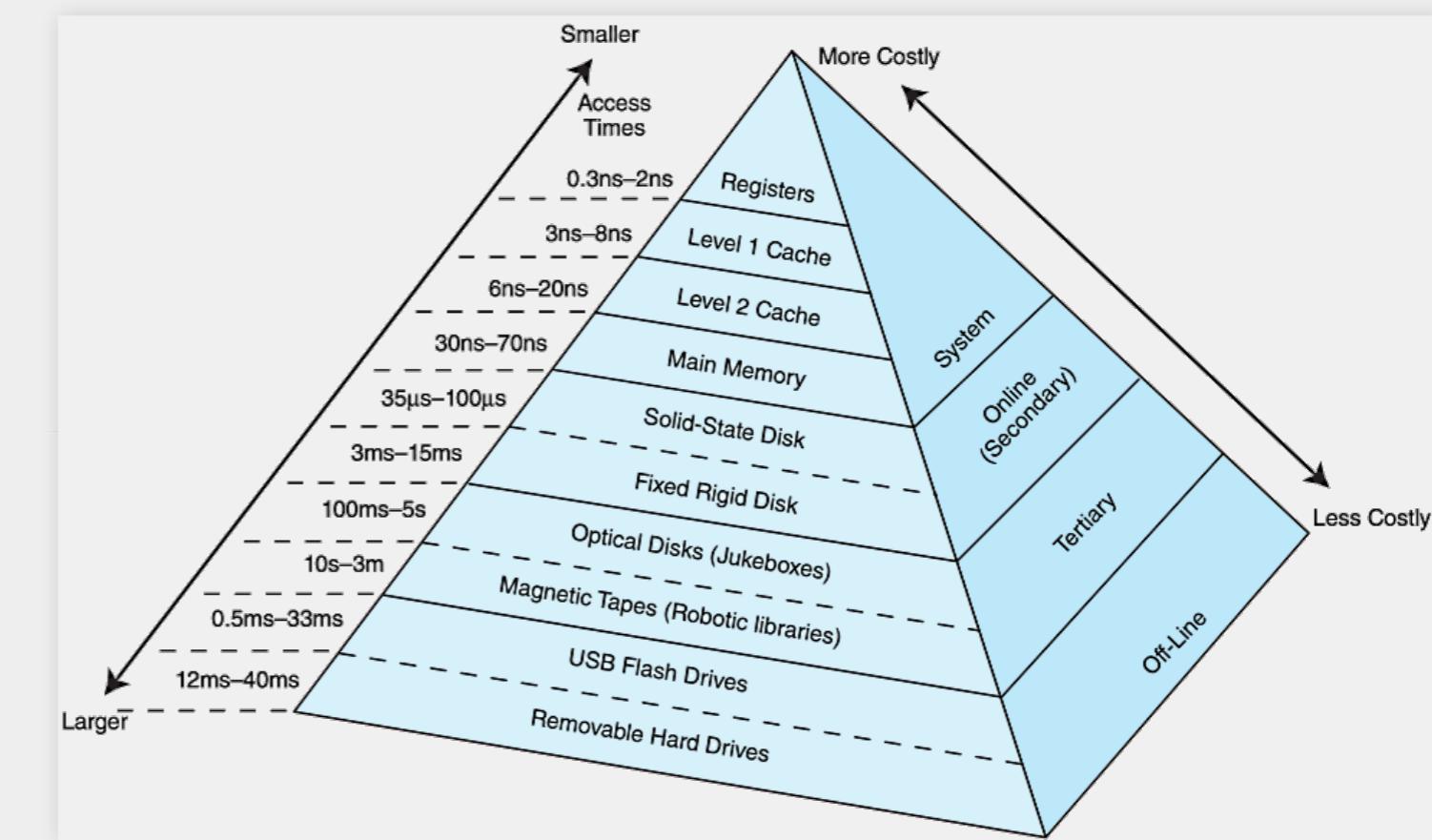
存储器层次化结构 (1)



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

存储器层次化结构 (1)

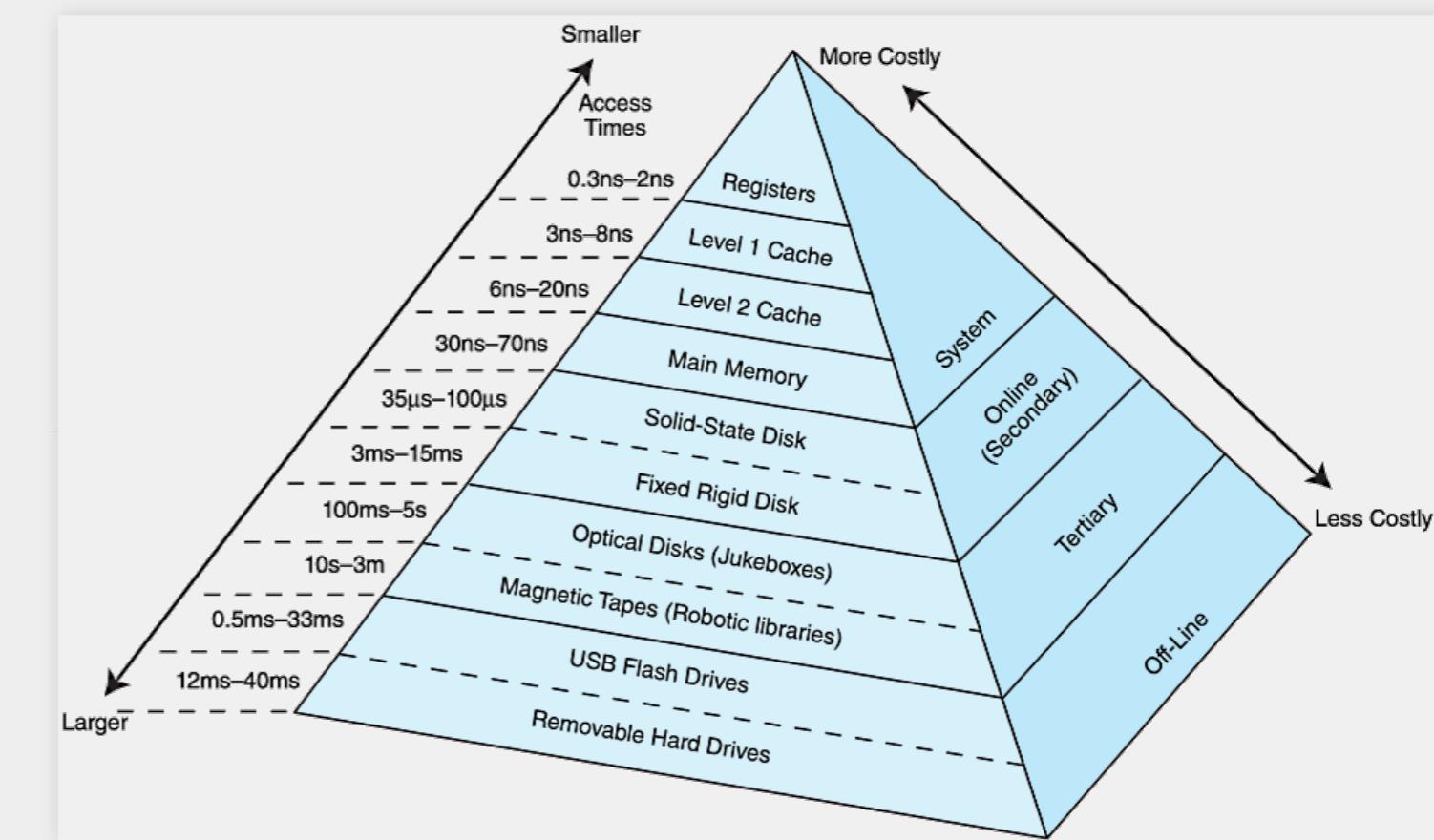
- 不同类型的存储器访问速度和存储成本不同



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

存储器层次化结构 (1)

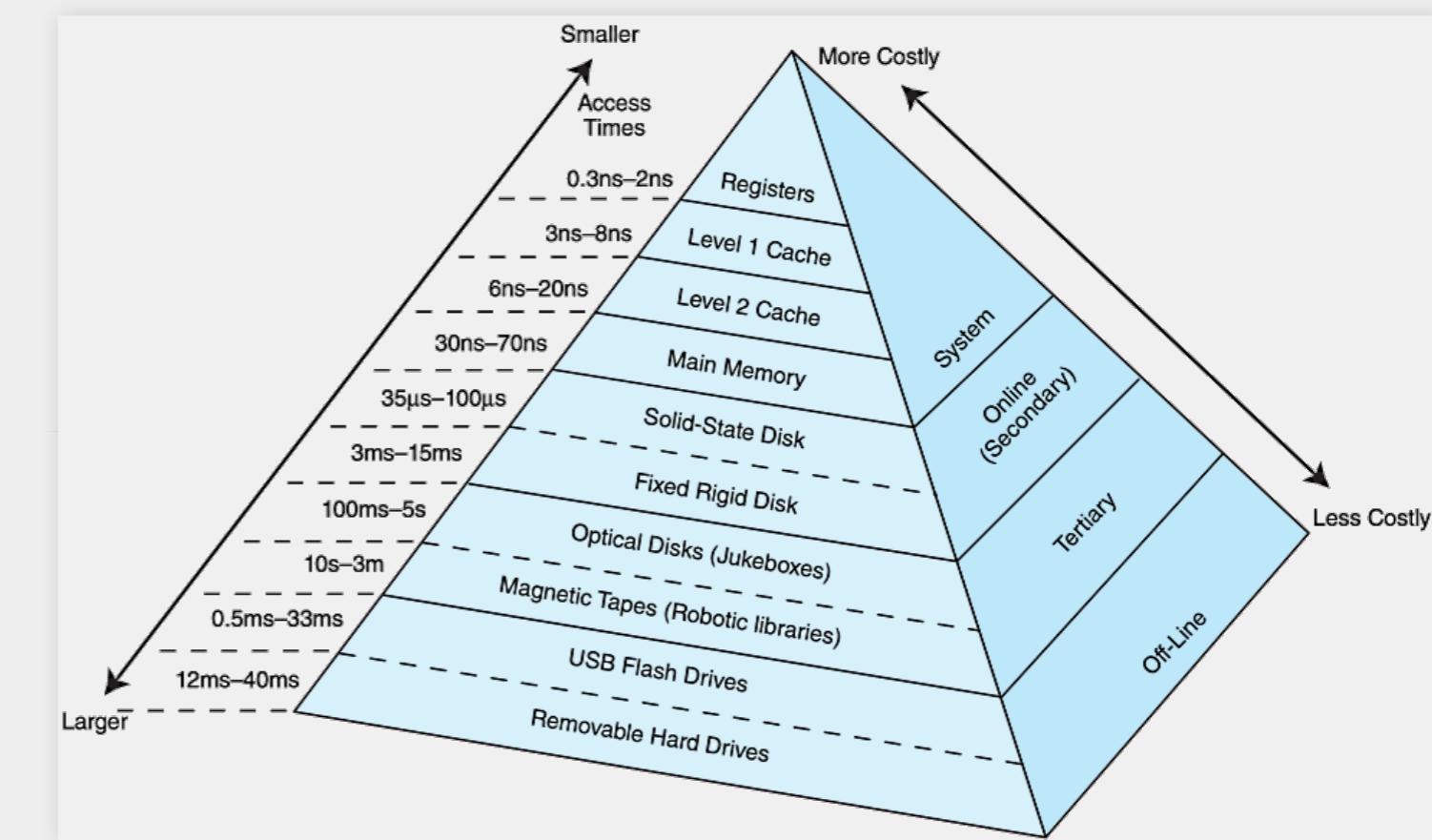
- 不同类型的存储器访问速度和存储成本不同
- 一般来说，访问速度越快的存储器价格越高、存储空间大小越小



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

存储器层次化结构 (1)

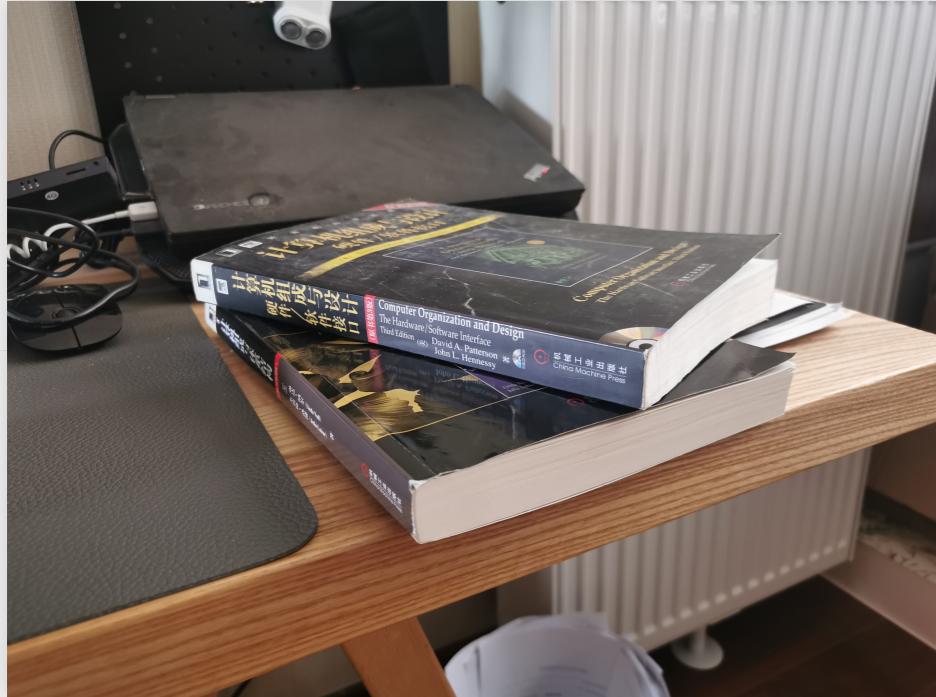
- 不同类型的存储器访问速度和存储成本不同
- 一般来说，访问速度越快的存储器价格越高、存储空间大小越小
- 计算机在访问数据的时候**优先使用访问速度快的存储器**



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

存储器层次化结构 (2)

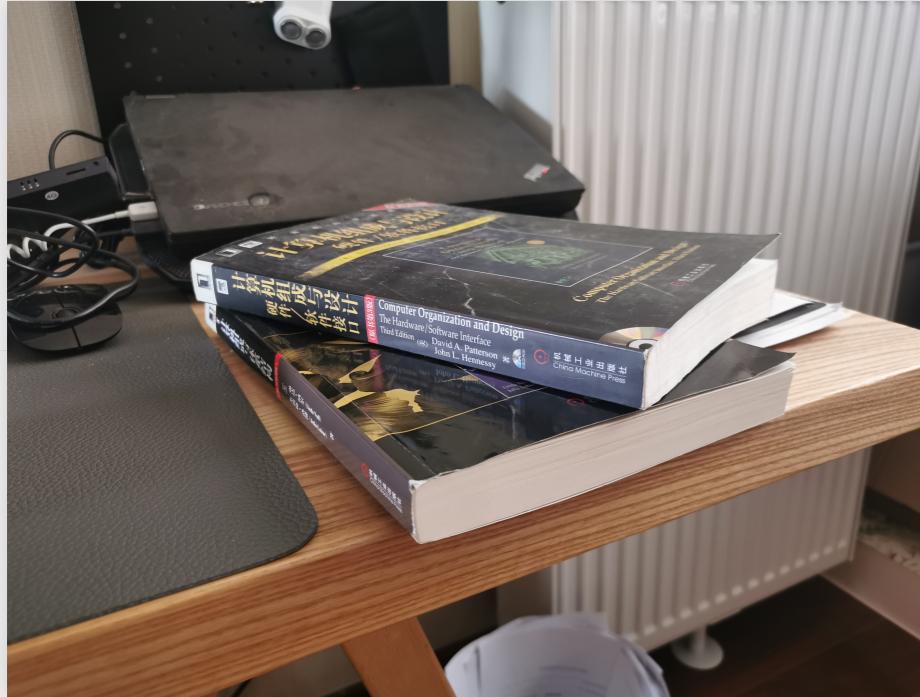
存储器层次化结构 (2)



书桌

- 存储大小: $O(1) \sim O(10)$
- 访问速度: 秒级

存储器层次化结构 (2)



书桌

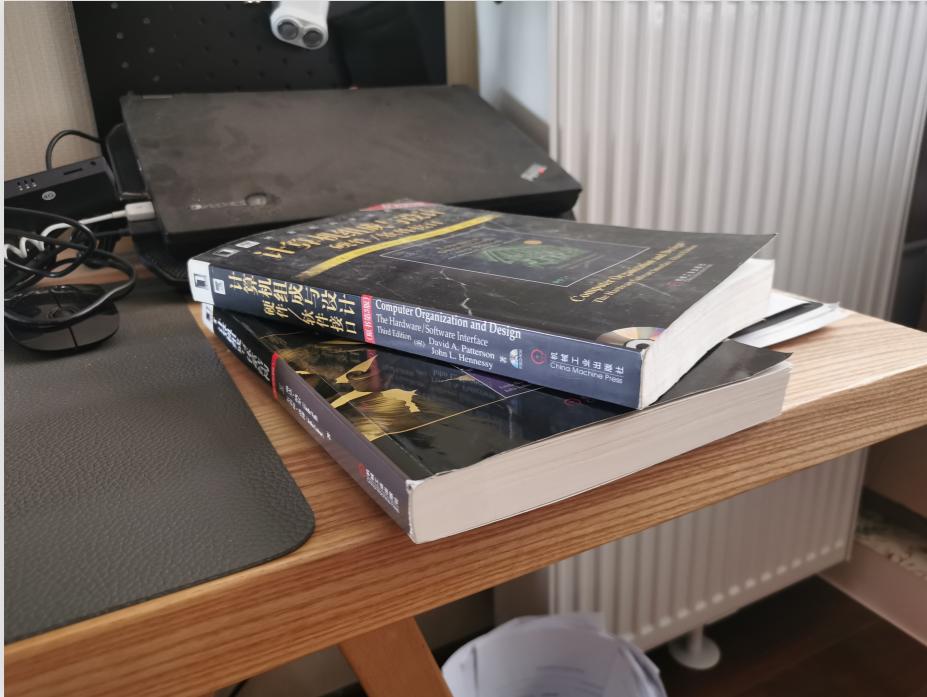
- 存储大小: $O(1) \sim O(10)$
- 访问速度: 秒级



书架

- 存储大小: $O(10) \sim O(100)$
- 访问速度: 分钟级

存储器层次化结构 (2)



书桌

- 存储大小: $O(1) \sim O(10)$
- 访问速度: 秒级



书架

- 存储大小: $O(10) \sim O(100)$
- 访问速度: 分钟级



图书馆

- 存储大小: $O(10^4) \sim O(10^6)$
- 访问速度: 十分钟级、小时级

图片来源: <http://gxjd.scu.edu.cn>

存储器层次化结构的性能 (1)

对于每一层存储结构，我们用如下的指标来描述其性能：

存储器层次化结构的性能 (1)

对于每一层存储结构，我们用如下的指标来描述其性能：

- 命中 (hit)：所需信息存储在该层存储结构中
 - 命中率 (hit rate)：发生命中的次数占总查询次数的比例
 - 命中时间 (hit time)：在该层存储结构中进行一次查找的时间

存储器层次化结构的性能 (1)

对于每一层存储结构，我们用如下的指标来描述其性能：

- 命中 (hit)：所需信息存储在该层存储结构中
 - 命中率 (hit rate)：发生命中的次数占总查询次数的比例
 - 命中时间 (hit time)：在该层存储结构中进行一次查找的时间
- 失效 (miss)：所需信息没有存储在该层存储结构中
 - 失效率 (miss rate)：发生失效的次数占总查询次数的比例
 - 失效惩罚 (miss penalty)：处理一次失效事件所需的时间开销，包括在下一层存储器中查询、将所需数据块传输到当前存储器中的时间开销

存储器层次化结构的性能 (1)

对于每一层存储结构，我们用如下的指标来描述其性能：

- 命中 (hit)：所需信息存储在该层存储结构中
 - 命中率 (hit rate)：发生命中的次数占总查询次数的比例
 - 命中时间 (hit time)：在该层存储结构中进行一次查找的时间
- 失效 (miss)：所需信息没有存储在该层存储结构中
 - 失效率 (miss rate)：发生失效的次数占总查询次数的比例
 - 失效惩罚 (miss penalty)：处理一次失效事件所需的时间开销，包括在下一层存储器中查询、将所需数据块传输到当前存储器中的时间开销
- 平均访问时间 = 命中率 × 命中时间 + 失效率 × 失效惩罚
 - 失效率 + 命中率 = 1
 - 命中时间 << 失效惩罚 = (下一层平均访问时间 + 当前层次处理数据更新的时间)

存储器层次化结构的性能 (1)

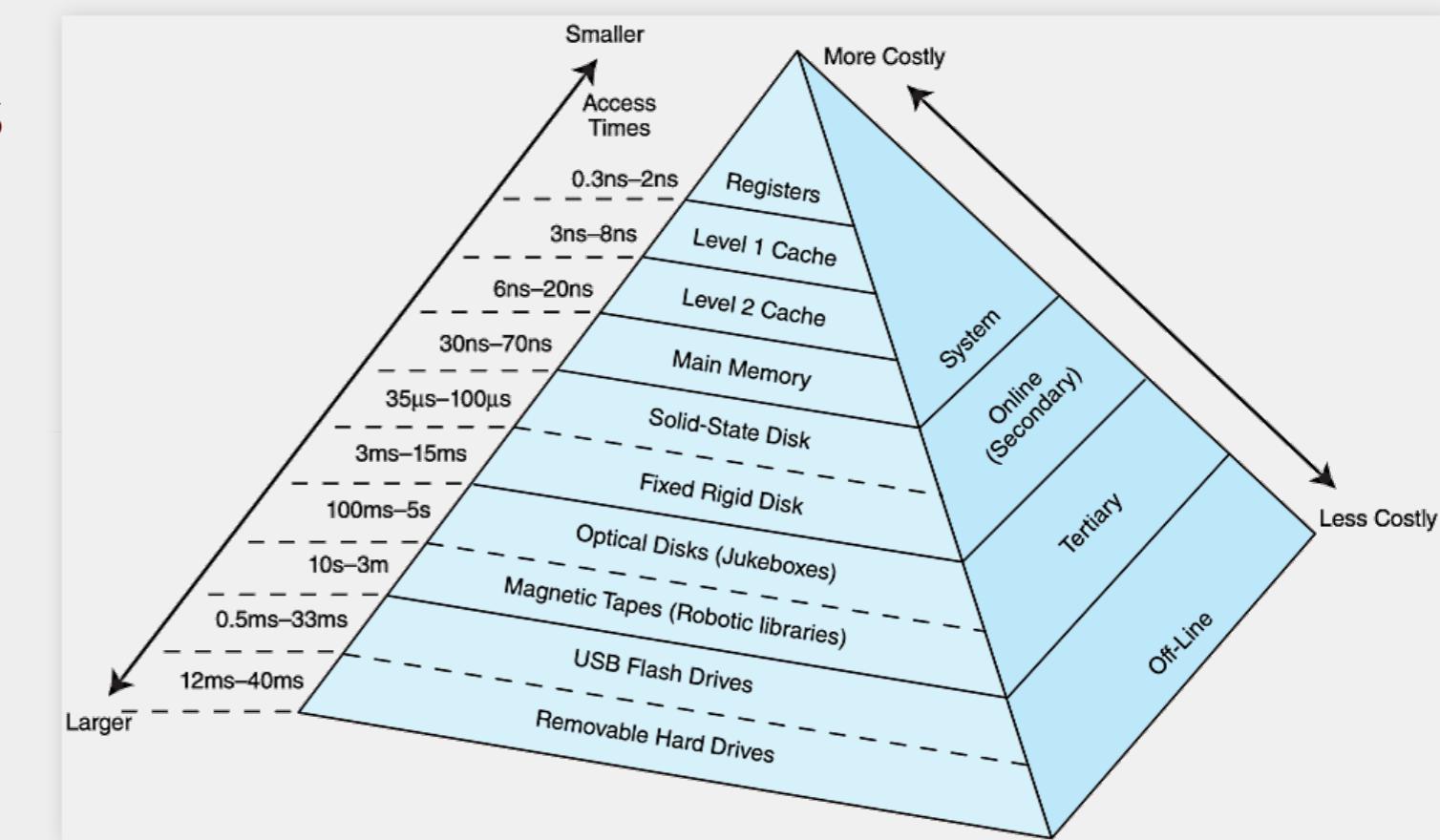
对于每一层存储结构，我们用如下的指标来描述其性能：

- 命中 (hit)：所需信息存储在该层存储结构中
 - 命中率 (hit rate)：发生命中的次数占总查询次数的比例
 - 命中时间 (hit time)：在该层存储结构中进行一次查找的时间
- 失效 (miss)：所需信息没有存储在该层存储结构中
 - 失效率 (miss rate)：发生失效的次数占总查询次数的比例
 - 失效惩罚 (miss penalty)：处理一次失效事件所需的时间开销，包括在下一层存储器中查询、将所需数据块传输到当前存储器中的时间开销
- 平均访问时间 = 命中率 × 命中时间 + 失效率 × 失效惩罚
 - 失效率 + 命中率 = 1
 - 命中时间 << 失效惩罚 = (下一层平均访问时间 + 当前层次处理数据更新的时间)

如何提高一个存储层次的性能？

存储器层次化结构的性能 (2)

通常采用**有效访问时间(Effective Access Time, EAT)**来描述一个层次化存储系统的性能

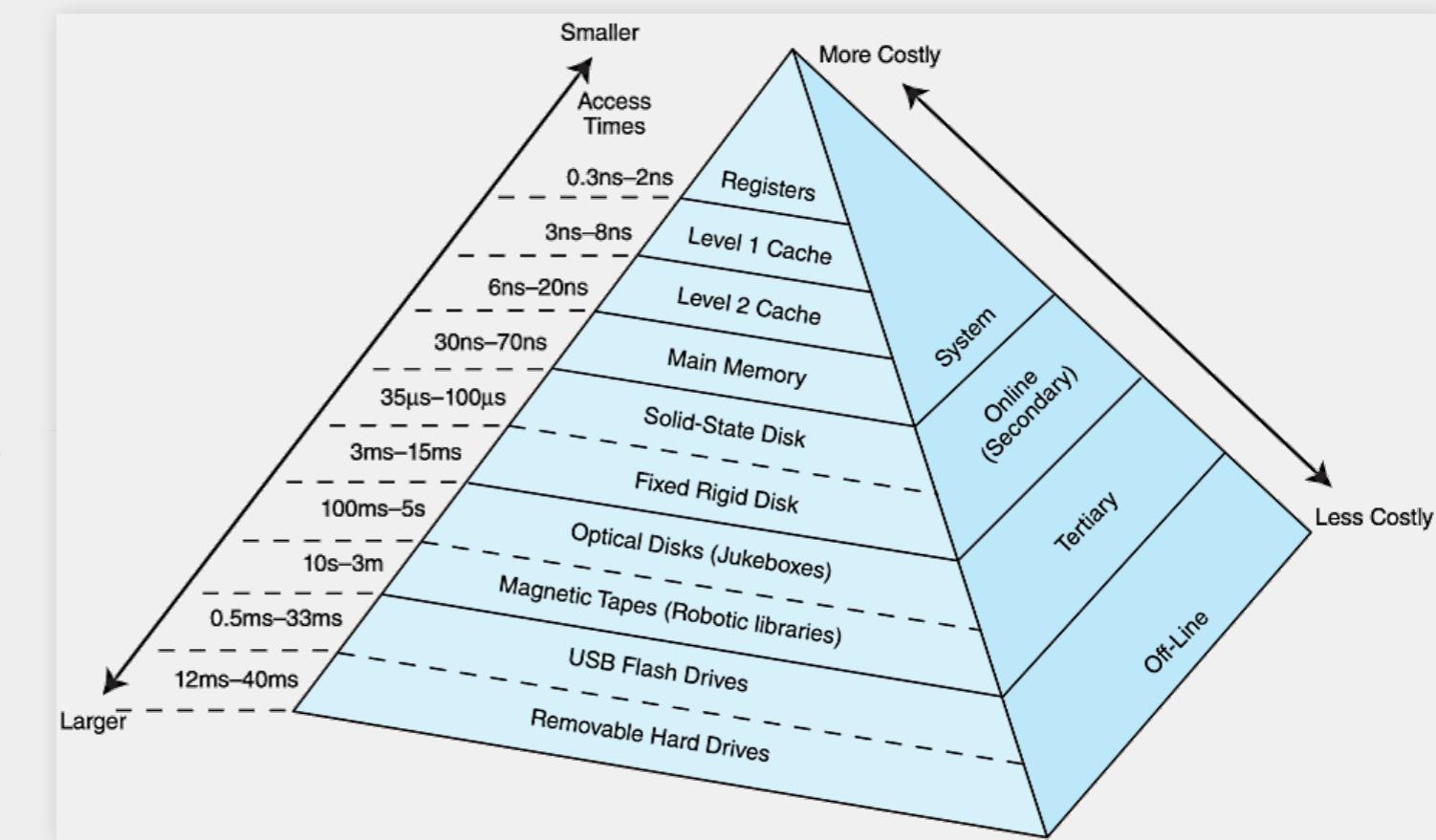


图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

存储器层次化结构的性能 (2)

通常采用**有效访问时间(Effective Access Time, EAT)**来描述一个层次化存储系统的性能

- 有效访问时间：层次化存储系统数据查找的期望时间

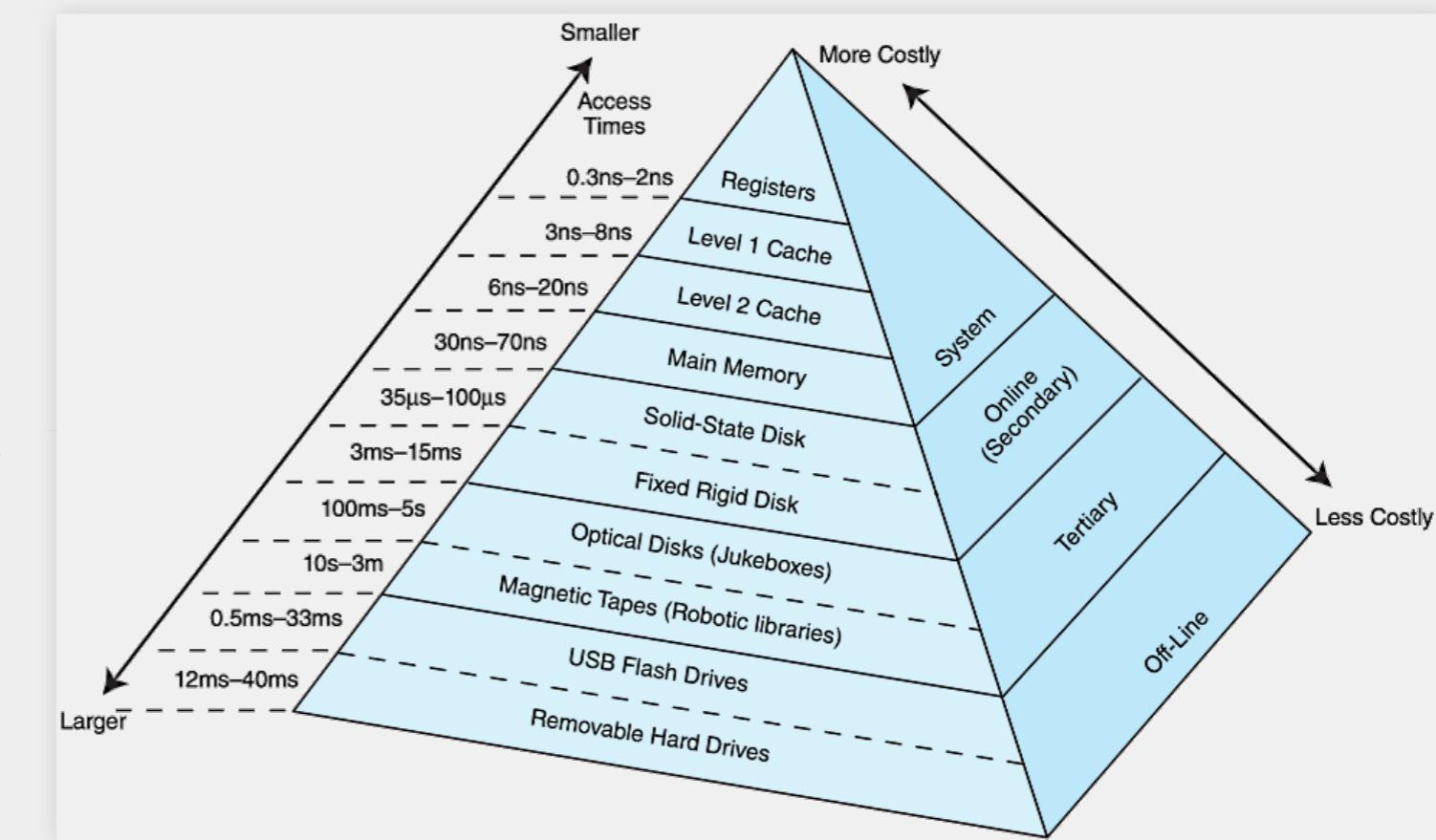


图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

存储器层次化结构的性能 (2)

通常采用**有效访问时间(Effective Access Time, EAT)**来描述一个层次化存储系统的性能

- 有效访问时间：层次化存储系统数据查找的期望时间
- 决定因素：各存储层性能、**查找方法**

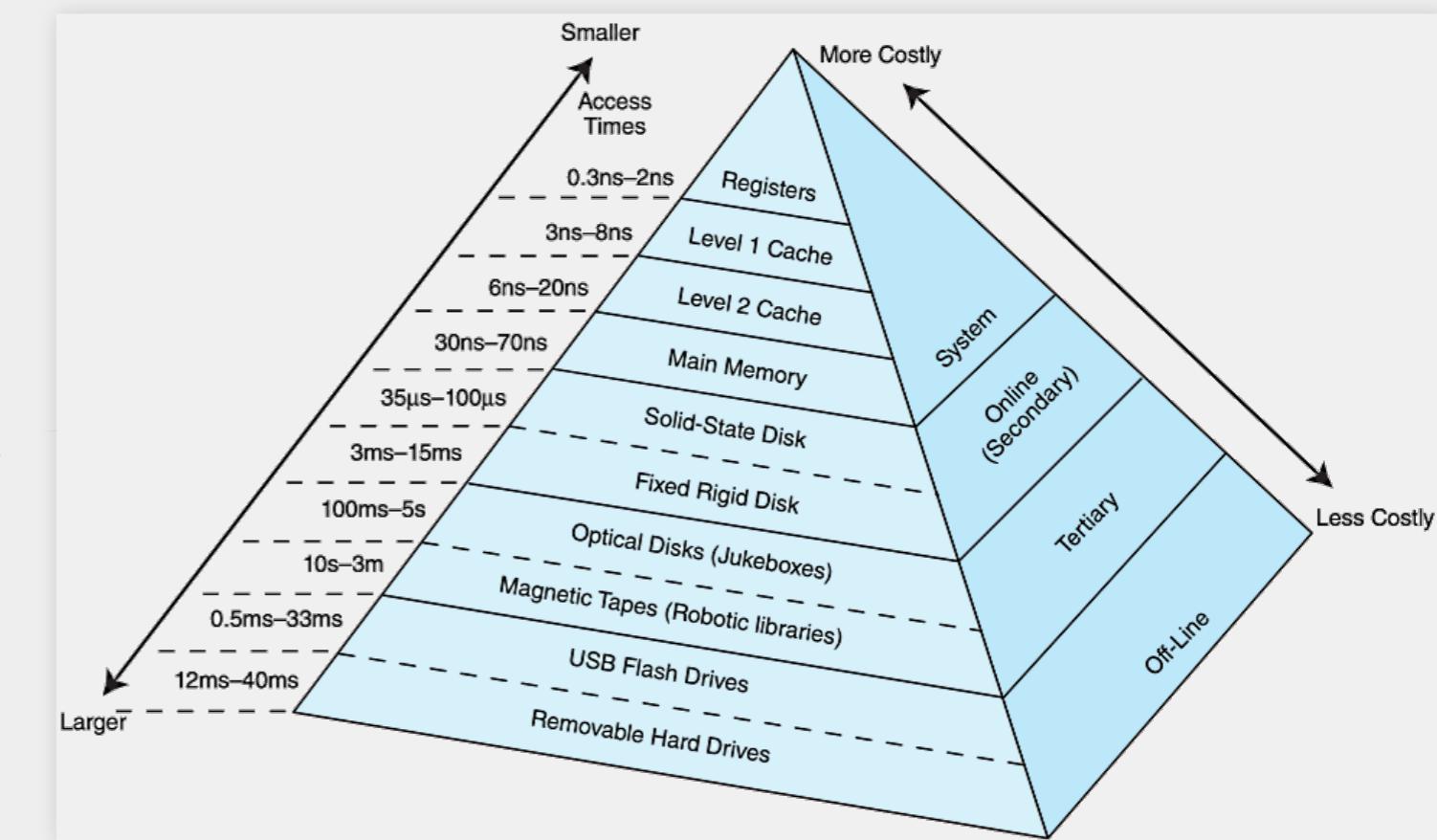


图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

存储器层次化结构的性能 (2)

通常采用**有效访问时间(Effective Access Time, EAT)**来描述一个层次化存储系统的性能

- 有效访问时间：层次化存储系统数据查找的期望时间
- 决定因素：各存储层性能、**查找方法**
 - 并行访问：多个不同层次的存储器同时开始访问数据

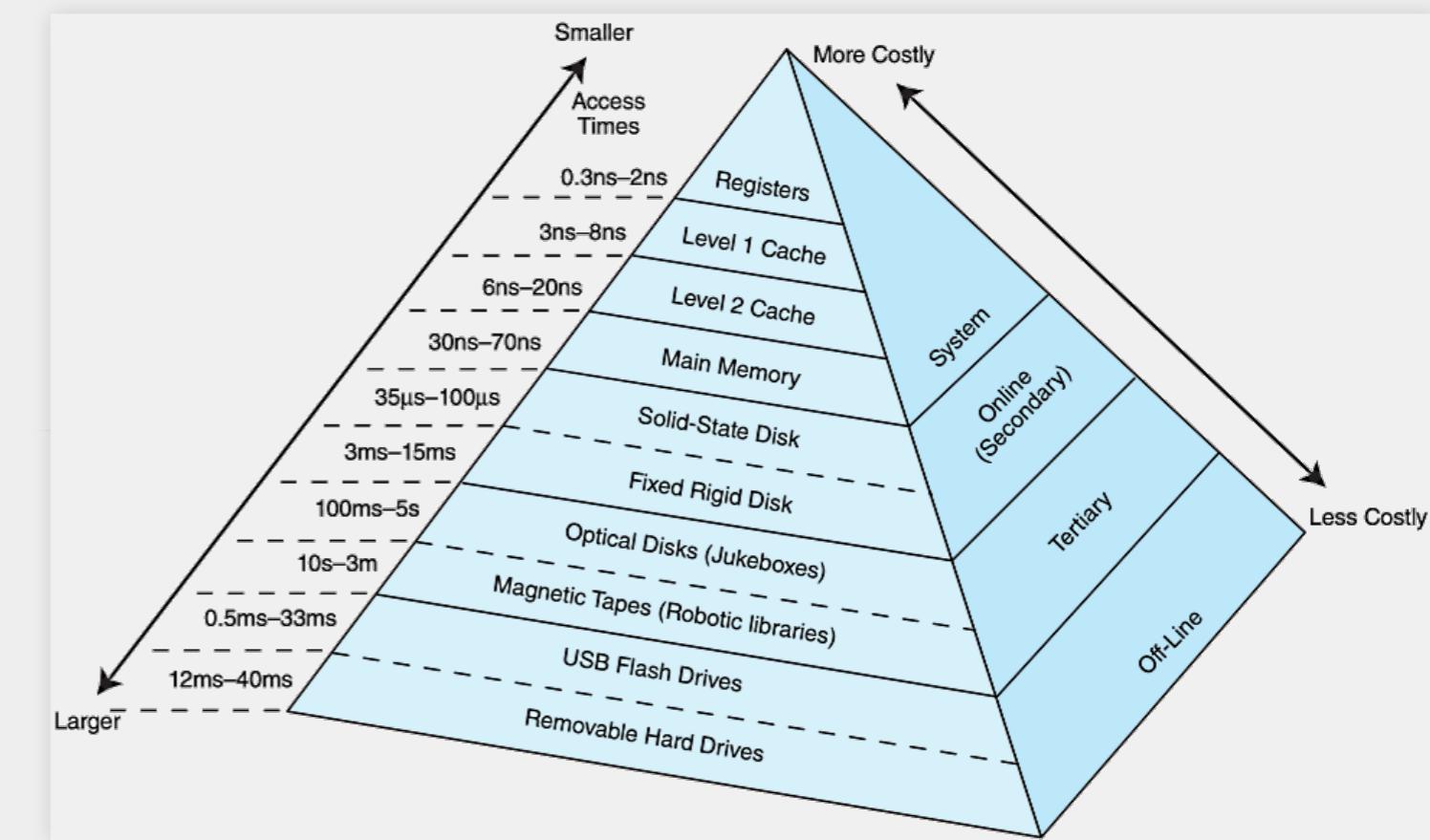


图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

存储器层次化结构的性能 (2)

通常采用**有效访问时间(Effective Access Time, EAT)**来描述一个层次化存储系统的性能

- 有效访问时间：层次化存储系统数据查找的期望时间
- 决定因素：各存储层性能、**查找方法**
 - 并行访问：多个不同层次的存储器同时开始访问数据
 - 顺序访问：访问低层次的存储器失效才访问高层次的存储器



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

存储器层次化结构的性能 (3)

例：假设一个两级层次化存储系统，其中第一级访问时间为
10ns，第二级访问时间为100ns，其中第一级访问命中率为90%

- 采用并行访问： $EAT = 0.9 \times 10ns + 0.1 \times 100ns = 19ns$
- 采用顺序访问：

$$EAT = 0.9 \times 10ns + 0.1 \times (10ns + 100ns) = 20ns$$

存储器层次化结构的性能 (3)

例：假设一个两级层次化存储系统，其中第一级访问时间为10ns，第二级访问时间为100ns，其中第一级访问命中率为90%

- 采用并行访问： $EAT = 0.9 \times 10\text{ns} + 0.1 \times 100\text{ns} = 19\text{ns}$
- 采用顺序访问：

$$EAT = 0.9 \times 10\text{ns} + 0.1 \times (10\text{ns} + 100\text{ns}) = 20\text{ns}$$

对于采用并行访问的N级层次化存储器

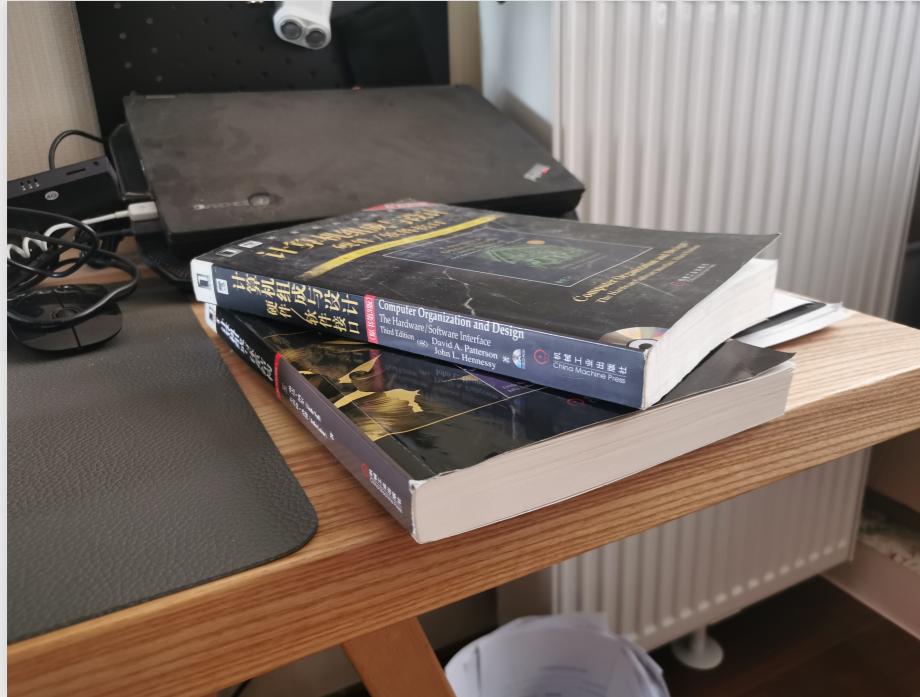
- 假设第*i*层命中率为 h_i ，访问时间为 t_i
- 我们可以推导其通用EAT计算公式为：

$$EAT = \sum_{i=1}^N \prod_{j=1}^{i-1} (1 - h_j) h_i t_i + \left(\prod_{j=1}^N (1 - h_j) \right) t_N$$

存储器的数据局部性

- 通常当发生一次数据失效的时候，层次存储器会把失效数据附近的数据块一齐拷贝并存储。这种处理方法的原理是**利用数据局部性提高命中率**
- 数据局部性体现为三种形式：
 - 时间局部性(temporal locality)：最近访问的数据在不远的将来会再次访问
 - 空间局部性(spatial locality)：即将访问的数据和最近访问的数据地址空间接近
 - 顺序局部性(sequential locality)：数据按顺序访问

数据局部性的实例



书桌

- 存储大小: $O(1) \sim O(10)$
- 访问速度: 秒级
- 假设我现在要做课件, 但是教材在书架上, 怎么办? 为什么?



书架

- 存储大小: $O(10) \sim O(100)$
- 访问速度: 分钟级



图书馆

- 存储大小: $O(10^4) \sim O(10^6)$
- 访问速度: 十分钟级、小时级

图片来源: <http://gxjd.scu.edu.cn>

典型的层次化存储系统

典型的层次化存储系统

- 一些层次化存储系统之间的局部性需要程序员显示指定

典型的层次化存储系统

- 一些层次化存储系统之间的局部性需要程序员显示指定
 - 例：从内存加载数据到寄存器、读取文件到内存等

典型的层次化存储系统

- 一些层次化存储系统之间的局部性需要程序员显示指定
 - 例：从内存加载数据到寄存器、读取文件到内存等
- 一些层次化存储系统之间的局部性对于程序员来说是透明的

典型的层次化存储系统

- 一些层次化存储系统之间的局部性需要程序员显示指定
 - 例：从内存加载数据到寄存器、读取文件到内存等
- 一些层次化存储系统之间的局部性对于程序员来说是透明的
 - 例：高速缓存、主存和虚拟内存组成的层次化存储系统

典型的层次化存储系统

- 一些层次化存储系统之间的局部性需要程序员显示指定
 - 例：从内存加载数据到寄存器、读取文件到内存等
- 一些层次化存储系统之间的局部性对于程序员来说是透明的
 - 例：高速缓存、主存和虚拟内存组成的层次化存储系统
 - 高速缓存提高内存访问速度

典型的层次化存储系统

- 一些层次化存储系统之间的局部性需要程序员显示指定
 - 例：从内存加载数据到寄存器、读取文件到内存等
- 一些层次化存储系统之间的局部性对于程序员来说是透明的
 - 例：高速缓存、主存和虚拟内存组成的层次化存储系统
 - 高速缓存提高内存访问速度
 - 虚拟内存扩大了内存访问的空间

存储器2: 高速缓存

高速缓存的基本原理

高速缓存的基本原理

- 利用数据局部性，用较小的高速访问存储器提高访问内存的性能

高速缓存的基本原理

- 利用数据局部性，用较小的高速访问存储器提高访问内存的性能
 - 现代计算机一般采用多级缓存

高速缓存的基本原理

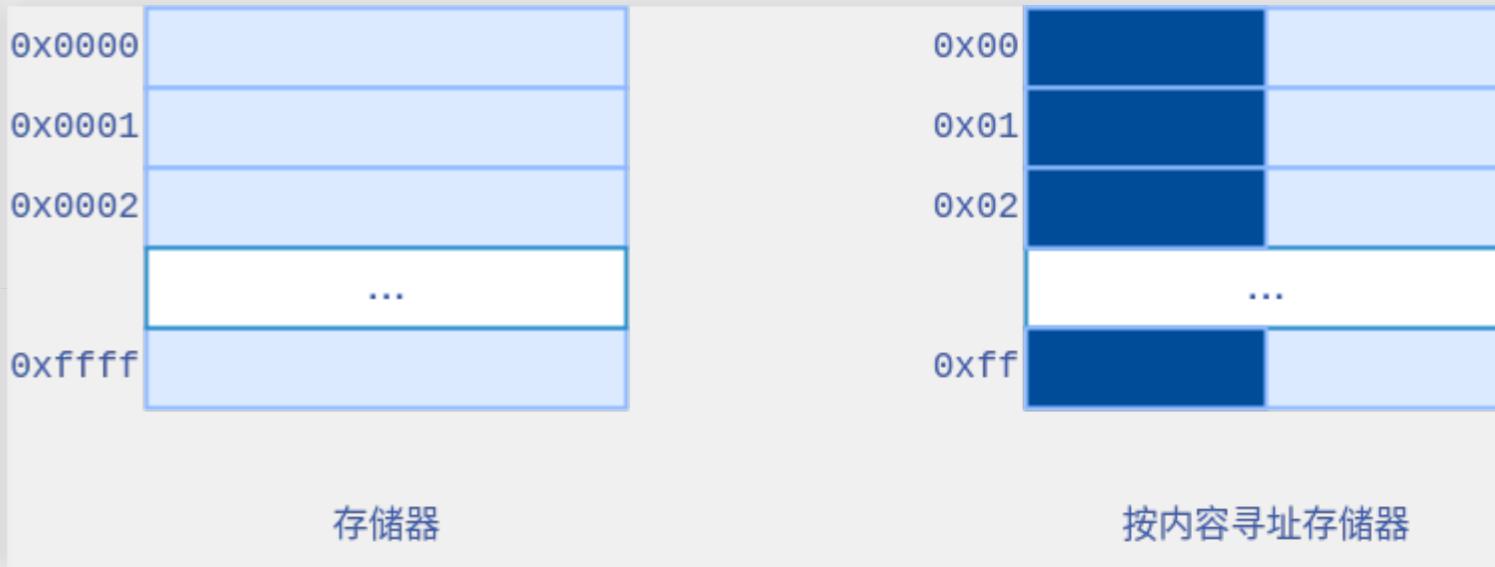
- 利用数据局部性，用较小的高速访问存储器提高访问内存的性能
 - 现代计算机一般采用多级缓存
 - 典型的高速缓存大小为：L1 (8K ~ 64K)， L2 (256K ~ 512K)， L3 (4M ~ 16M)

Vendor ID:	GenuineIntel
CPU family:	6
Model:	78
Model name:	Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz
Stepping:	3
CPU MHz:	800.057
CPU max MHz:	3400.0000
CPU min MHz:	400.0000
BogoMIPS:	5602.18
Virtualization:	VT-x
L1d cache:	64 KiB
L1i cache:	64 KiB
L2 cache:	512 KiB
L3 cache:	4 MiB

高速缓存的基本原理

- 利用数据局部性，用较小的高速访问存储器提高访问内存的性能
 - 现代计算机一般采用多级缓存
 - 典型的高速缓存大小为：L1 (8K ~ 64K)， L2 (256K ~ 512K)， L3 (4M ~ 16M)
- 高速缓存中的寻址依赖根据存储的内容，因此也被称为**按内容寻址存储器 (Content Addressable Memory, CAM)**

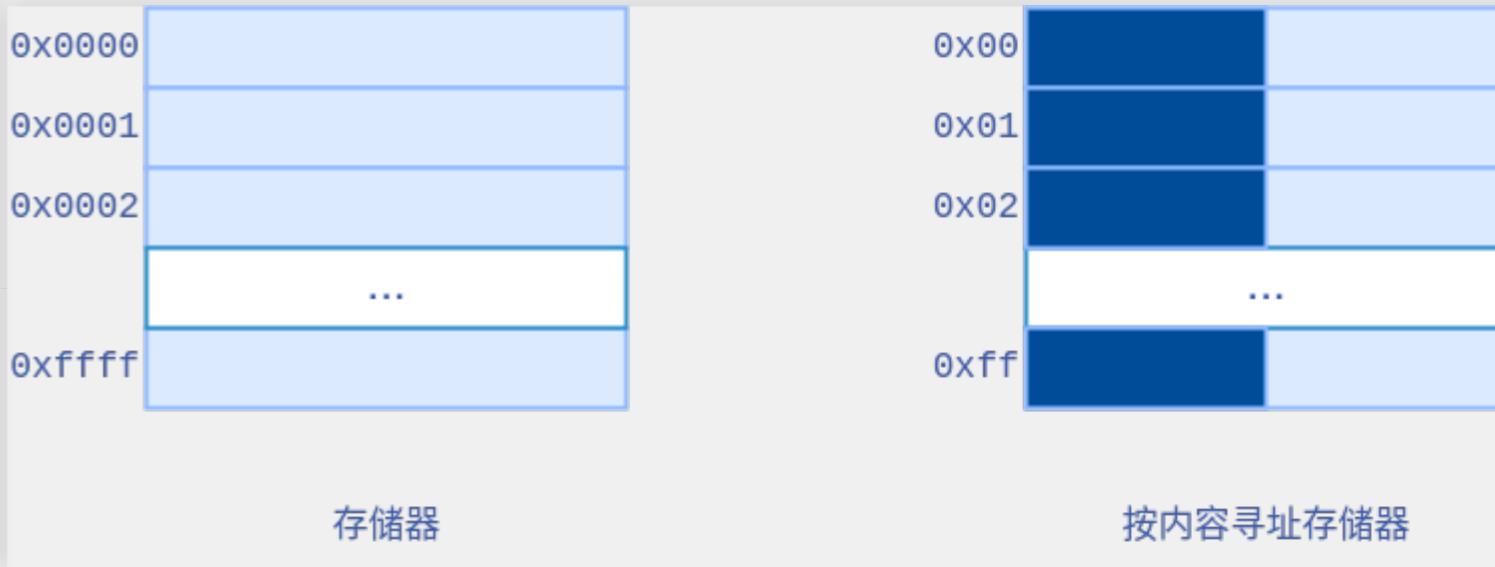
Vendor ID: GenuineIntel
CPU family: 6
Model: 78
Model name: Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz
Stepping: 3
CPU MHz: 800.057
CPU max MHz: 3400.0000
CPU min MHz: 400.0000
BogoMIPS: 5602.18
Virtualization: VT-x
L1d cache: 64 KiB
L1i cache: 64 KiB
L2 cache: 512 KiB
L3 cache: 4 MiB



高速缓存的基本原理

- 利用数据局部性，用较小的高速访问存储器提高访问内存的性能
 - 现代计算机一般采用多级缓存
 - 典型的高速缓存大小为：L1 (8K ~ 64K)， L2 (256K ~ 512K)， L3 (4M ~ 16M)
- 高速缓存中的寻址依赖根据存储的内容，因此也被称为**按内容寻址存储器 (Content Addressable Memory, CAM)**
- 高速缓存中的寻址过程需要将主存地址映射为缓存中的地址，因此也称为**缓存映射**。缓存通常采用和主存同样的存储单元(字节、字)，因此映射方法只取决于地址空间大小，和内存寻址方法无关

Vendor ID: GenuineIntel
CPU family: 6
Model: 78
Model name: Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz
Stepping: 3
CPU MHz: 800.057
CPU max MHz: 3400.0000
CPU min MHz: 400.0000
BogoMIPS: 5602.18
Virtualization: VT-x
L1d cache: 64 KiB
L1i cache: 64 KiB
L2 cache: 512 KiB
L3 cache: 4 MiB



缓存寻址方式类比

- 主存的寻址方式类比数组
 - 假设主存用数组a表示
 - 给定一个地址x，数据保存在a[x]

缓存寻址方式类比

- 主存的寻址方式类比数组
 - 假设主存用数组a表示
 - 给定一个地址x，数据保存在a[x]
- 高速缓存的**直接映射**寻址方式类比哈希表
 - 假设高速缓存用表a存储，总共有N个存储单元
 - 每个存储单元存储了一对数据(x, y)
 - 对于给定的x，高速缓存计算一个哈希值 $z = h(x) \bmod N$
 - 如果 $a[z].x == x$ ，则返回数据 $a[z].y$ ，否则查找失败

缓存寻址方式类比

- 主存的寻址方式类比数组
 - 假设主存用数组a表示
 - 给定一个地址x，数据保存在a[x]
- 高速缓存的**直接映射**寻址方式类比哈希表
 - 假设高速缓存用表a存储，总共有N个存储单元
 - 每个存储单元存储了一对数据(x, y)
 - 对于给定的x，高速缓存计算一个哈希值 $z = h(x) \bmod N$
 - 如果 $a[z].x == x$ ，则返回数据 $a[z].y$ ，否则查找失败
- 高速缓存的**全相联映射**寻址方式类比数组
 - 假设高速缓存用数组a存储，总共有N个存储单元
 - 每个存储单元存储了一对数据(x, y)
 - 对于给定的x，高速缓存对于第i个存储单元进行判断
 - 如果 $a[i].x == x$ ，则返回数据 $a[i].y$
 - 如果没有找到满足条件的数据则查找失败

缓存映射

- 高速缓存和主存采用大小相同的块(block)
- 缓存采用**有效位**标识一个缓存块是否保存了有效的数据
- 当发生缓存失效时，需要将整个块从主存中载入缓存
- 由于同一份数据在块内的偏移值是一致的，因此缓存映射策略主要体现在块的映射策略

缓存映射

- 高速缓存和主存采用大小相同的块(block)
- 缓存采用**有效位**标识一个缓存块是否保存了有效的数据
- 当发生缓存失效时，需要将整个块从主存中载入缓存
- 由于同一份数据在块内的偏移值是一致的，因此缓存映射策略主要体现在块的映射策略

为什么要采用块？

缓存映射

- 高速缓存和主存采用大小相同的块(block)
- 缓存采用**有效位**标识一个缓存块是否保存了有效的数据
- 当发生缓存失效时，需要将整个块从主存中载入缓存
- 由于同一份数据在块内的偏移值是一致的，因此缓存映射策略主要体现在块的映射策略

为什么要采用块？

- 利用数据的局部性提高命中率

缓存映射

- 高速缓存和主存采用大小相同的块(block)
- 缓存采用**有效位**标识一个缓存块是否保存了有效的数据
- 当发生缓存失效时，需要将整个块从主存中载入缓存
- 由于同一份数据在块内的偏移值是一致的，因此缓存映射策略主要体现在块的映射策略

为什么要采用块？

- 利用数据的局部性提高命中率
- 对地址、状态信息进行压缩，提高缓存存储能力

缓存映射

- 高速缓存和主存采用大小相同的块(block)
- 缓存采用**有效位**标识一个缓存块是否保存了有效的数据
- 当发生缓存失效时，需要将整个块从主存中载入缓存
- 由于同一份数据在块内的偏移值是一致的，因此缓存映射策略主要体现在块的映射策略

为什么要采用块？

- 利用数据的局部性提高命中率
- 对地址、状态信息进行压缩，提高缓存存储能力
 - 例：假设采用64位按字寻址的存储器和高速缓存，每个缓存单元需要3个状态标识位

缓存映射

- 高速缓存和主存采用大小相同的块(block)
- 缓存采用**有效位**标识一个缓存块是否保存了有效的数据
- 当发生缓存失效时，需要将整个块从主存中载入缓存
- 由于同一份数据在块内的偏移值是一致的，因此缓存映射策略主要体现在块的映射策略

为什么要采用块？

- 利用数据的局部性提高命中率
- 对地址、状态信息进行压缩，提高缓存存储能力
 - 例：假设采用64位按字寻址的存储器和高速缓存，每个缓存单元需要3个状态标识位
 - 如果缓存按存储单元处理，8个单元需要 $8 \times (64 + 64 + 3) = 1048$ 位

缓存映射

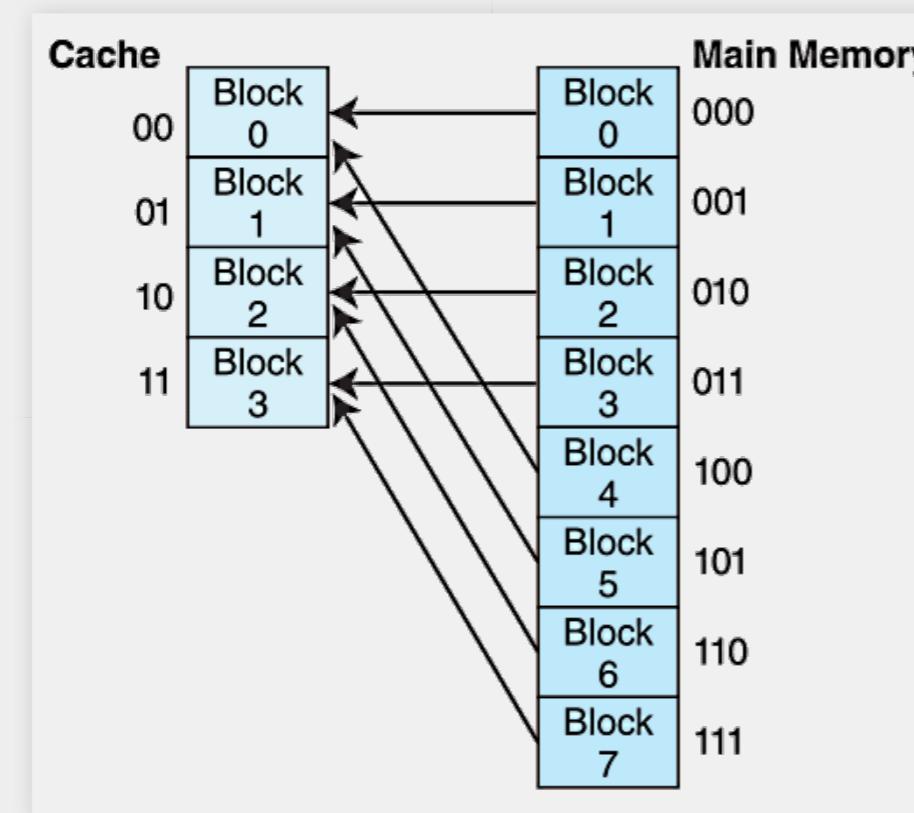
- 高速缓存和主存采用大小相同的块(block)
- 缓存采用**有效位**标识一个缓存块是否保存了有效的数据
- 当发生缓存失效时，需要将整个块从主存中载入缓存
- 由于同一份数据在块内的偏移值是一致的，因此缓存映射策略主要体现在块的映射策略

为什么要采用块？

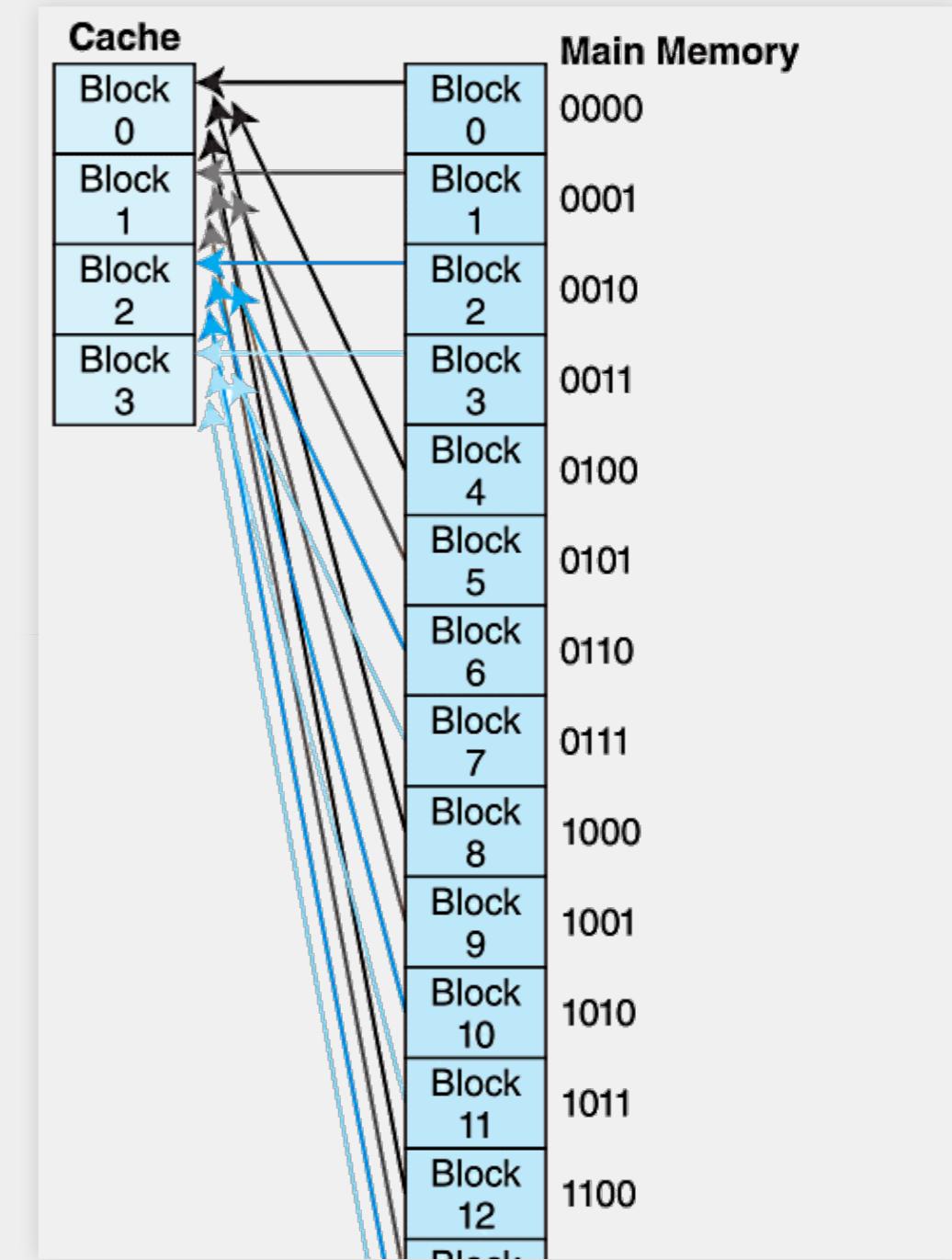
- 利用数据的局部性提高命中率
- 对地址、状态信息进行压缩，提高缓存存储能力
 - 例：假设采用64位按字寻址的存储器和高速缓存，每个缓存单元需要3个状态标识位
 - 如果缓存按存储单元处理，8个单元需要 $8 \times (64 + 64 + 3) = 1048$ 位
 - 如果按照8个单元组成的块，8个单元需要 $8 \times 64 + (64 - \log_2(8) + 3) = 576$ 位

直接映射

- 假设缓存包含 N 个块，内存中编号为 X 的块映射到缓存中编号为 Y 的块，则 $Y = X \bmod N$
- 例1：假设缓存有4个块，内存有8个块，则内存与缓存的映射关系如下图所示
- 例2：假设缓存有4个块，内存有16个块，则内存和缓存的映射关系如右图所示



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

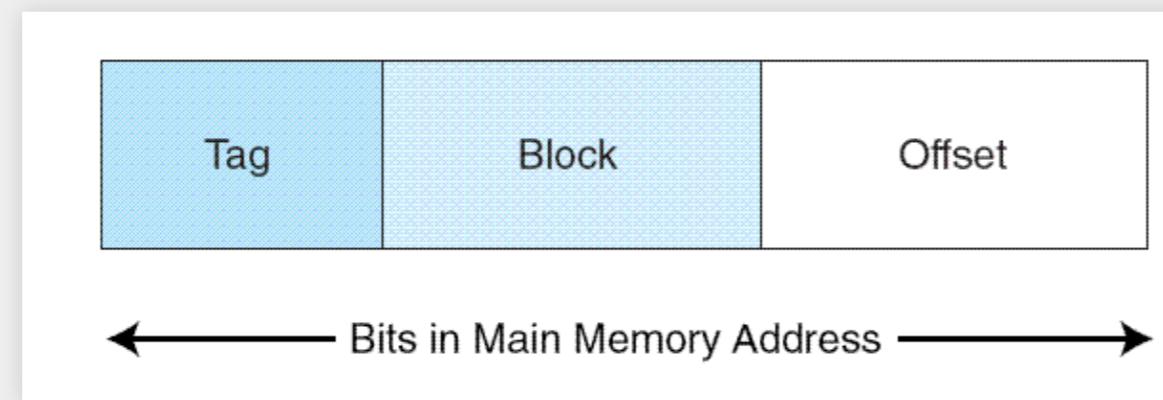


图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

直接映射的地址结构

直接映射的地址如下图所示，分为三个字段：

- 块内偏移值 (offset)：代表数据在块内的地址偏移值
- 块号 (block)：代表数据在缓存内的存储块编号
- 标记 (tag)：标记和块号共同代表数据在主存中的存储块编号

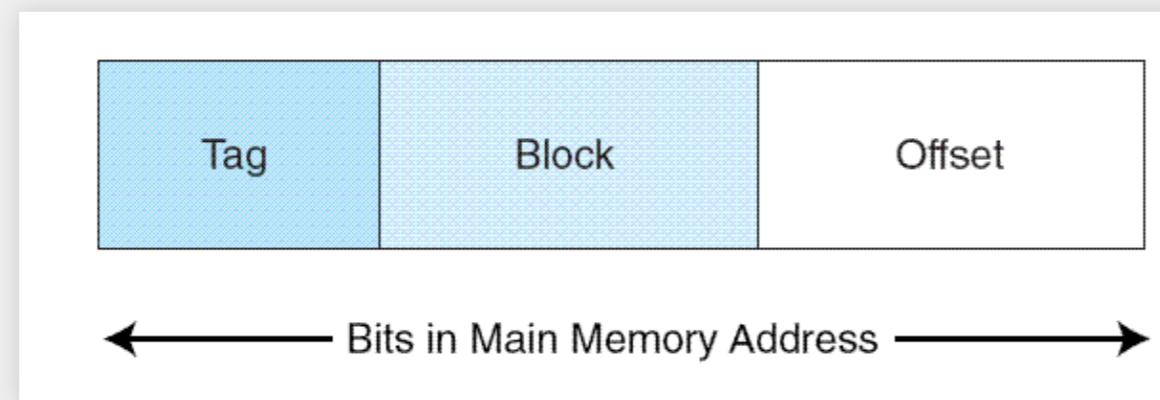


图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

直接映射的地址结构

直接映射的地址如下图所示，分为三个字段：

- 块内偏移值 (offset)：代表数据在块内的地址偏移值
- 块号 (block)：代表数据在缓存内的存储块编号
- 标记 (tag)：标记和块号共同代表数据在主存中的存储块编号



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

如何确定各字段的大小？

直接映射示例1

假设块大小为4, 缓存容量为2个块, 主存地
址空间为4个块

直接映射示例1

假设块大小为4, 缓存容量为2个块, 主存地址空间为4个块

- 地址长度为4位

直接映射示例1

假设块大小为4, 缓存容量为2个块, 主存地址空间为4个块

- 地址长度为4位
- 偏移值所需长度为 \log_2 (块大小) = 2

直接映射示例1

假设块大小为4, 缓存容量为2个块, 主存地址空间为4个块

- 地址长度为4位
- 偏移值所需长度为 \log_2 (块大小) = 2
- 块号所需长度为 \log_2 (缓存容量) = 1

直接映射示例1

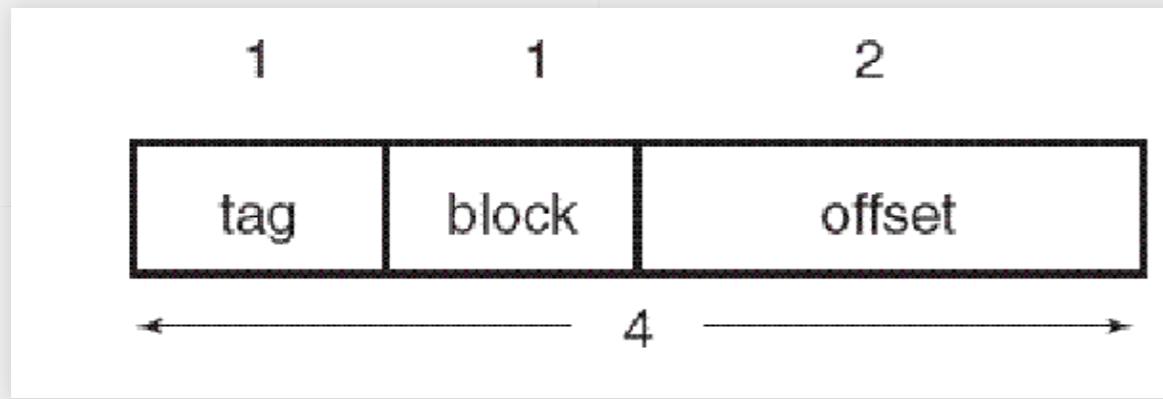
假设块大小为4, 缓存容量为2个块, 主存地址空间为4个块

- 地址长度为4位
- 偏移值所需长度为 \log_2 (块大小) = 2
- 块号所需长度为 \log_2 (缓存容量) = 1
- 标记所需长度为
地址长度 – 偏移值长度 – 块号长度 = 1

直接映射示例1

假设块大小为4, 缓存容量为2个块, 主存地址空间为4个块

- 地址长度为4位
- 偏移值所需长度为 \log_2 (块大小) = 2
- 块号所需长度为 \log_2 (缓存容量) = 1
- 标记所需长度为
地址长度 – 偏移值长度 – 块号长度 = 1



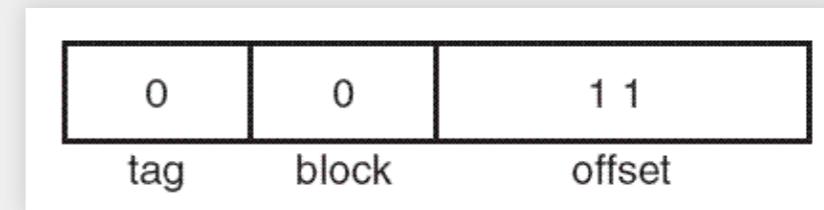
图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

直接映射示例1

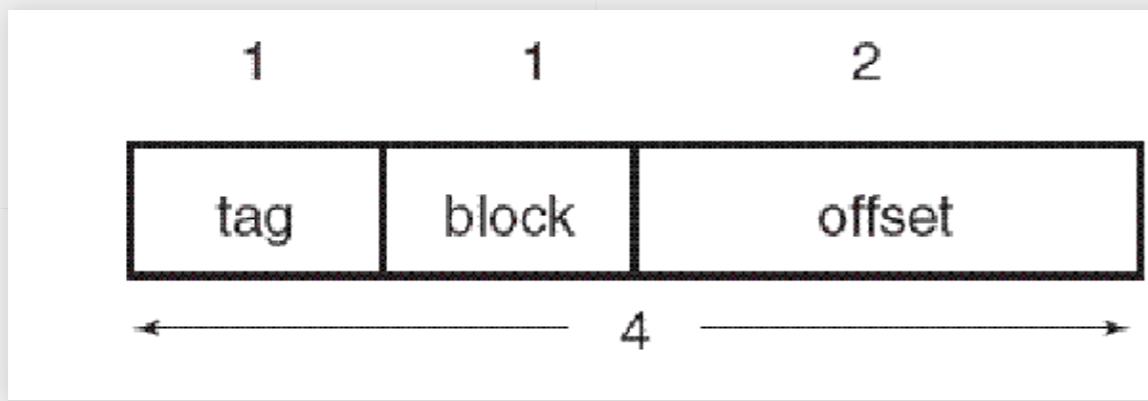
假设块大小为4, 缓存容量为2个块, 主存地址空间为4个块

- 地址长度为4位
- 偏移值所需长度为 \log_2 (块大小) = 2
- 块号所需长度为 \log_2 (缓存容量) = 1
- 标记所需长度为
地址长度 – 偏移值长度 – 块号长度 = 1

地址为0x3对应的字段



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

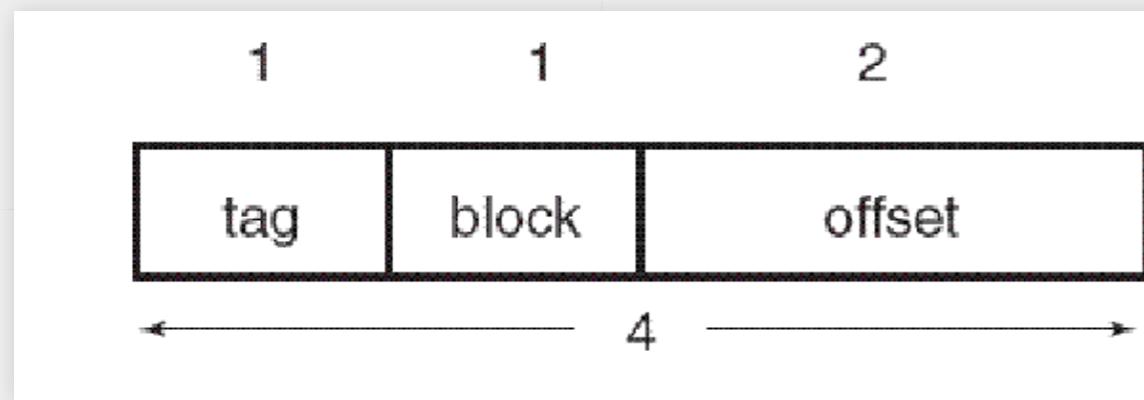


图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

直接映射示例1

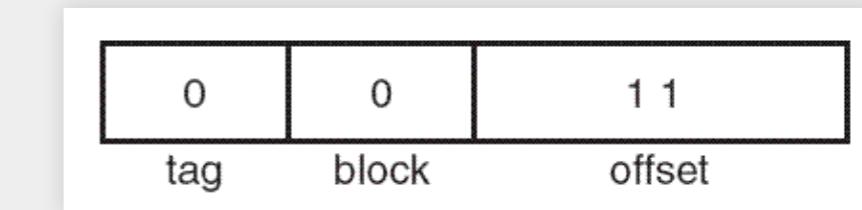
假设块大小为4, 缓存容量为2个块, 主存地址空间为4个块

- 地址长度为4位
- 偏移值所需长度为 \log_2 (块大小) = 2
- 块号所需长度为 \log_2 (缓存容量) = 1
- 标记所需长度为
地址长度 - 偏移值长度 - 块号长度 = 1



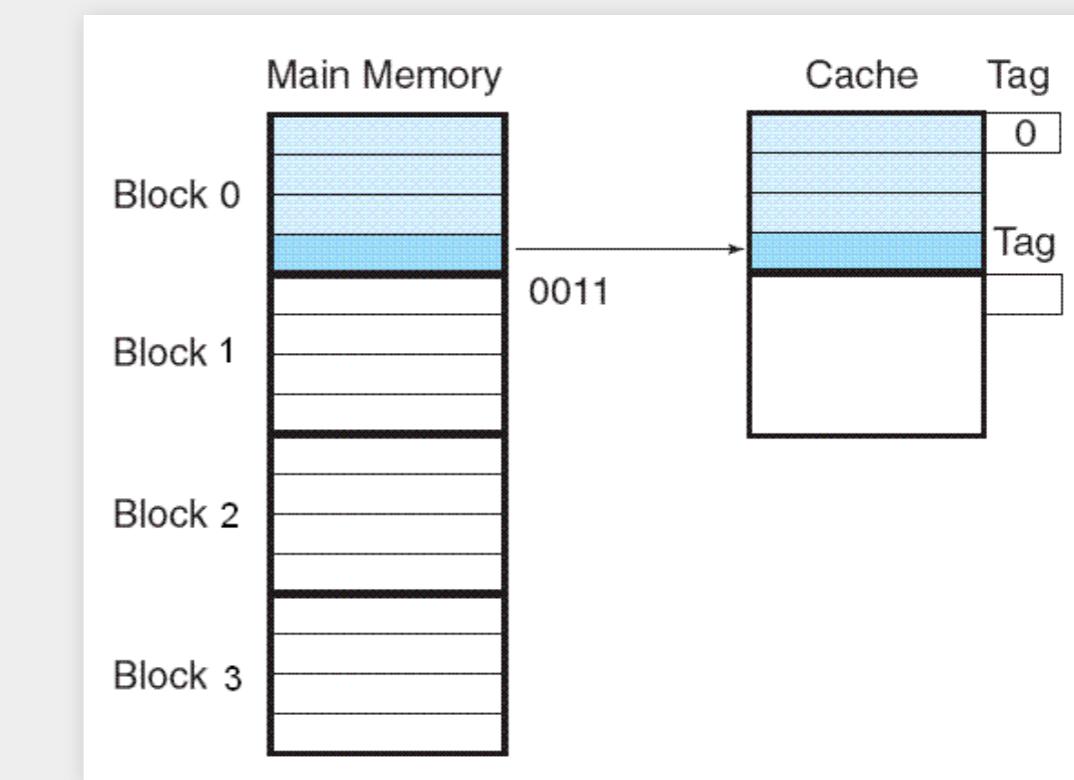
图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

地址为0x3对应的字段



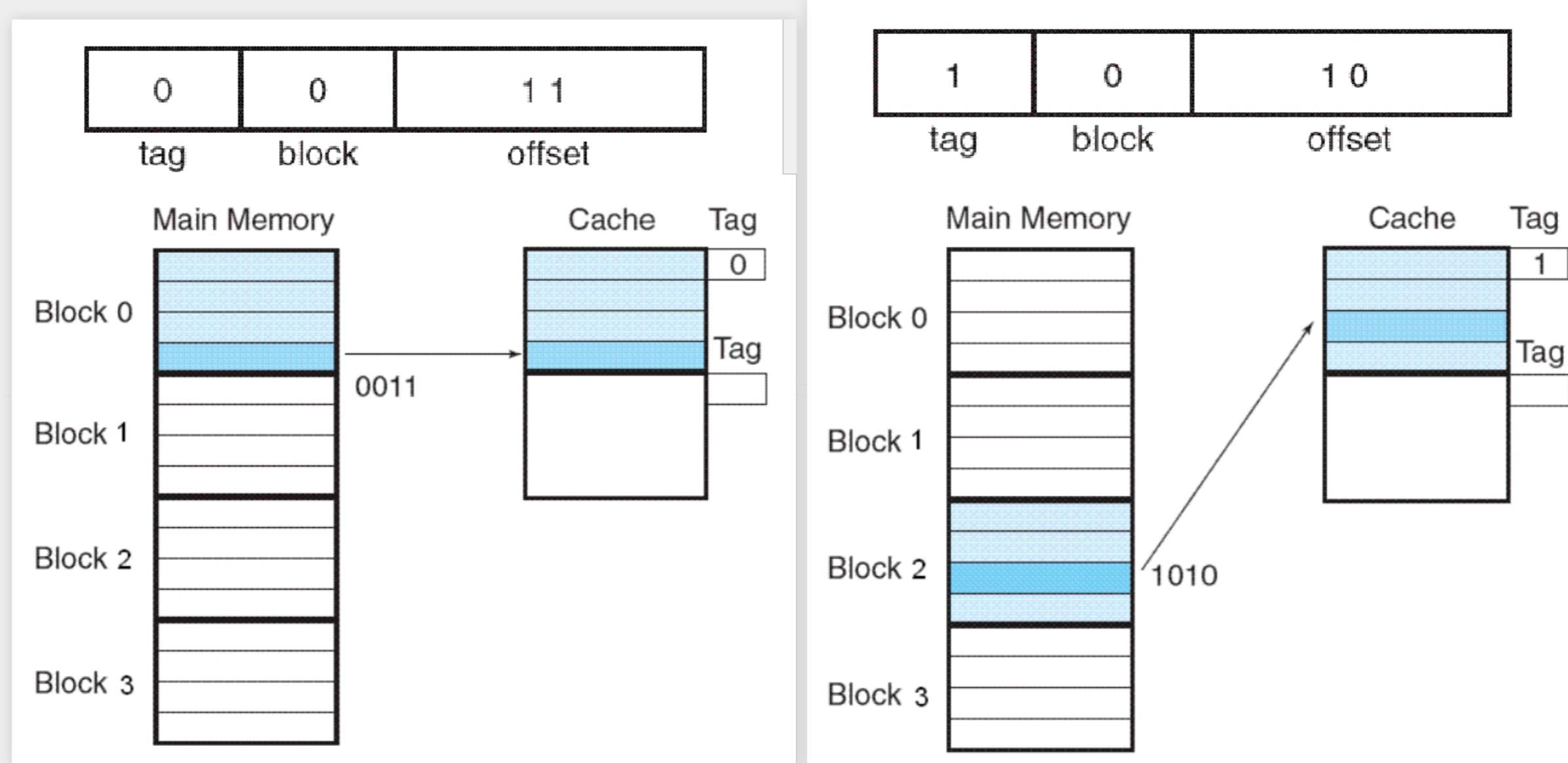
图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

地址为0x3对应的缓存映射



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

直接映射示例1



图片来源： Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

直接映射示例2

假设一个按字节寻址的存储器容量为 2^{14} 字节，缓存有16个块，每个块8个字节

- 地址总长度=
- 偏移值字段长度=
- 块号字段长度=
- 标记字段长度=

直接映射示例2

假设一个按字节寻址的存储器容量为 2^{14} 字节，缓存有16个块，每个块8个字节

- 地址总长度= $\log_2 (2^{14}) = 14$
- 偏移值字段长度=
- 块号字段长度=
- 标记字段长度=

直接映射示例2

假设一个按字节寻址的存储器容量为 2^{14} 字节，缓存有16个块，每个块8个字节

- 地址总长度= $\log_2 (2^{14}) = 14$
- 偏移值字段长度= $\log_2 (8) = 3$
- 块号字段长度=
- 标记字段长度=

直接映射示例2

假设一个按字节寻址的存储器容量为 2^{14} 字节，缓存有16个块，每个块8个字节

- 地址总长度= $\log_2 (2^{14}) = 14$
- 偏移值字段长度= $\log_2 (8) = 3$
- 块号字段长度= $\log_2 (16) = 4$
- 标记字段长度=

直接映射示例2

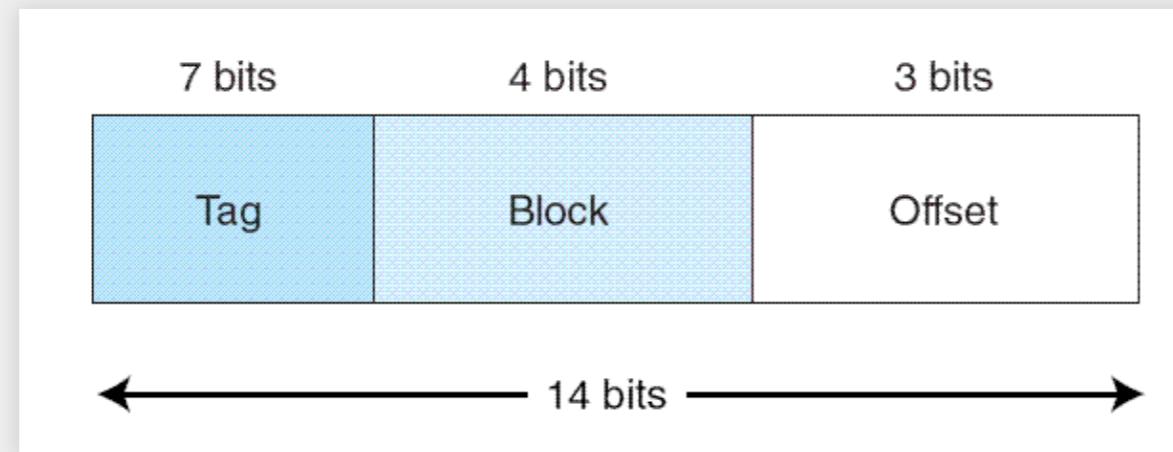
假设一个按字节寻址的存储器容量为 2^{14} 字节，缓存有16个块，每个块8个字节

- 地址总长度= $\log_2 (2^{14}) = 14$
- 偏移值字段长度= $\log_2 (8) = 3$
- 块号字段长度= $\log_2 (16) = 4$
- 标记字段长度= $14 - 3 - 4 = 7$

直接映射示例2

假设一个按字节寻址的存储器容量为 2^{14} 字节，缓存有16个块，每个块8个字节

- 地址总长度= $\log_2 (2^{14}) = 14$
- 偏移值字段长度= $\log_2 (8) = 3$
- 块号字段长度= $\log_2 (16) = 4$
- 标记字段长度= $14 - 3 - 4 = 7$



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

直接映射示例3

假设一个按字节寻址的存储器使用16位地址，缓存有64个块，每个块由8个字节构成

- 地址总长度=16
- 偏移值字段长度=
- 块号字段长度=
- 标记字段长度=
- 如果地址为0x0404 (0000 0100 0000 0100)的内容缓存命中，该内容的缓存块号是 ，标记号为

图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

直接映射示例3

假设一个按字节寻址的存储器使用16位地址，缓存有64个块，每个块由8个字节构成

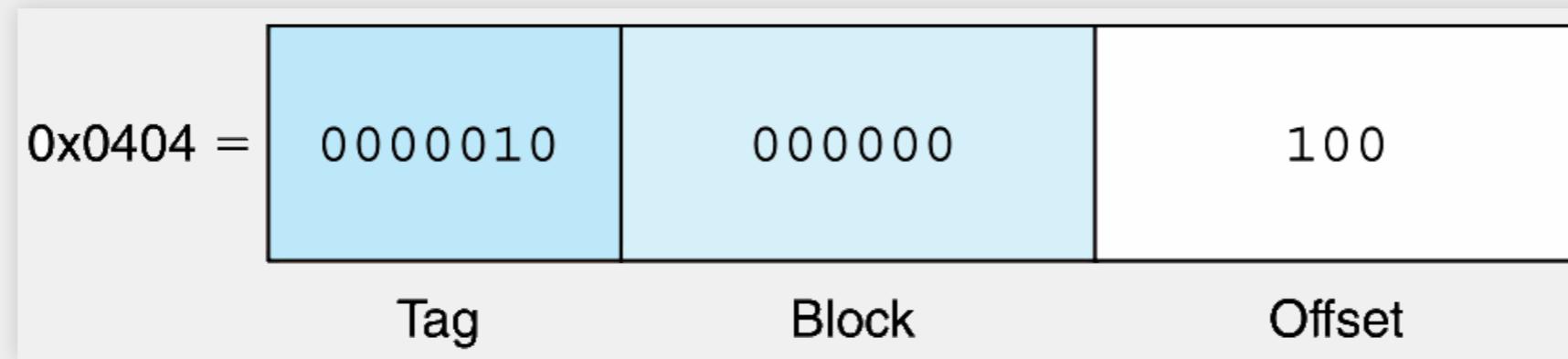
- 地址总长度=16
- 偏移值字段长度= $\log_2 (8) = 3$
- 块号字段长度= $\log_2 (64) = 6$
- 标记字段长度= $16 - 3 - 6 = 7$
- 如果地址为0x0404 (0000 0100 0000 0100)的内容缓存命中，该内容的缓存块号是 ，标记号为

图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

直接映射示例3

假设一个按字节寻址的存储器使用16位地址，缓存有64个块，每个块由8个字节构成

- 地址总长度=16
- 偏移值字段长度= $\log_2 (8) = 3$
- 块号字段长度= $\log_2 (64) = 6$
- 标记字段长度= $16 - 3 - 6 = 7$
- 如果地址为0x0404 (0000 0100 0000 0100)的内容缓存命中，该内容的缓存块号是0x00，标记号为0x02



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

直接映射小结

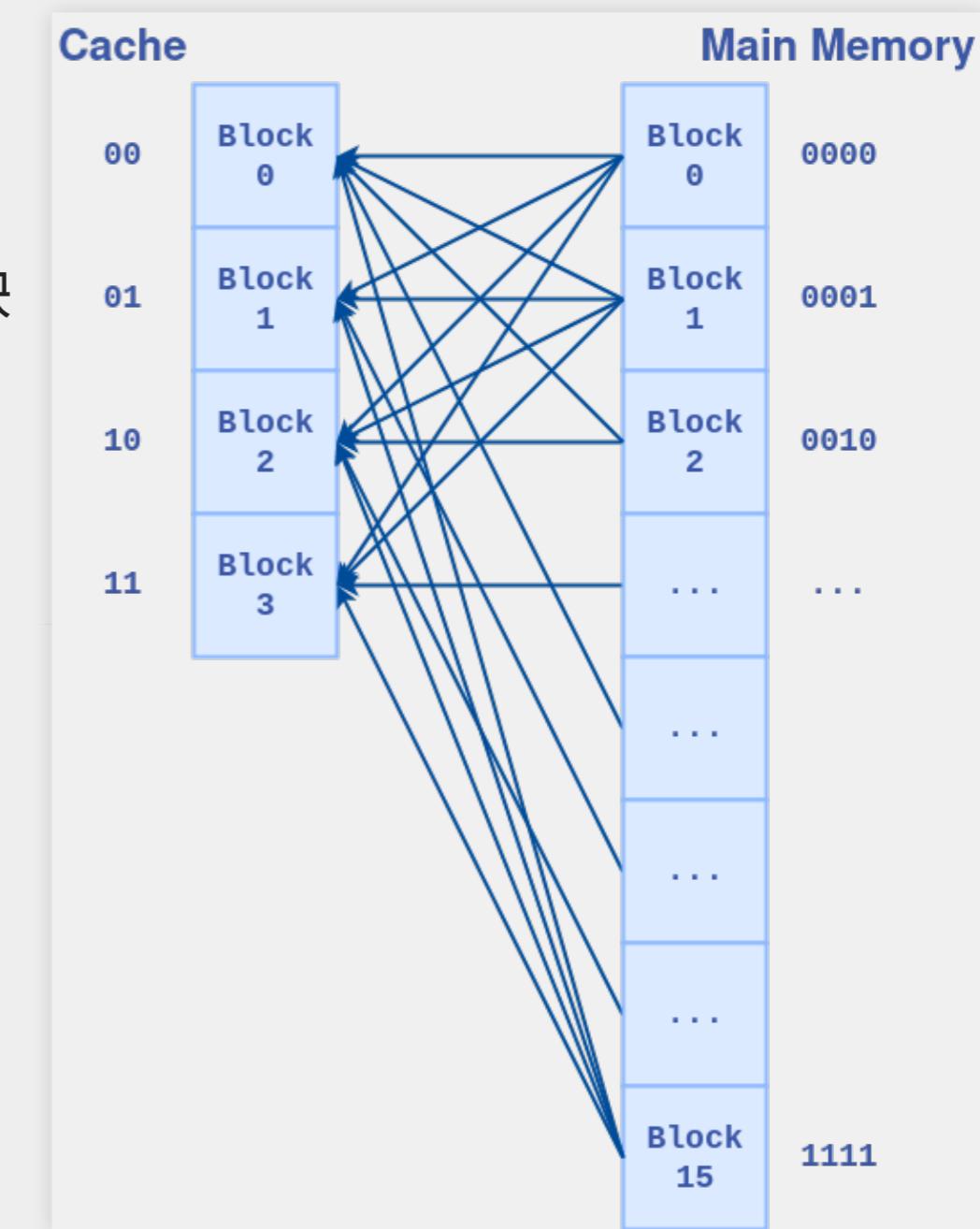
为了正确计算直接映射的结果，我们需要

- 根据内存地址寻址空间大小决定地址长度
- 根据块大小决定偏移值字段长度
- 根据缓存块数量决定块号字段长度
- 根据上述信息计算标记字段长度
- 根据上述信息，计算给定地址各字段的值
- 根据块号查找对应的块，检查标记字段是否匹配以及其它状态位判断是否缓存命中

全相联映射

全相联映射(Fully Associative Mapping)

- 假设缓存包含 N 个块，内存中编号为 X 的块可以映射到缓存中任一编号的块
- 例：假设缓存包含4个块，内存包含16个块，则内存和缓存的映射关系如右图所示



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th E

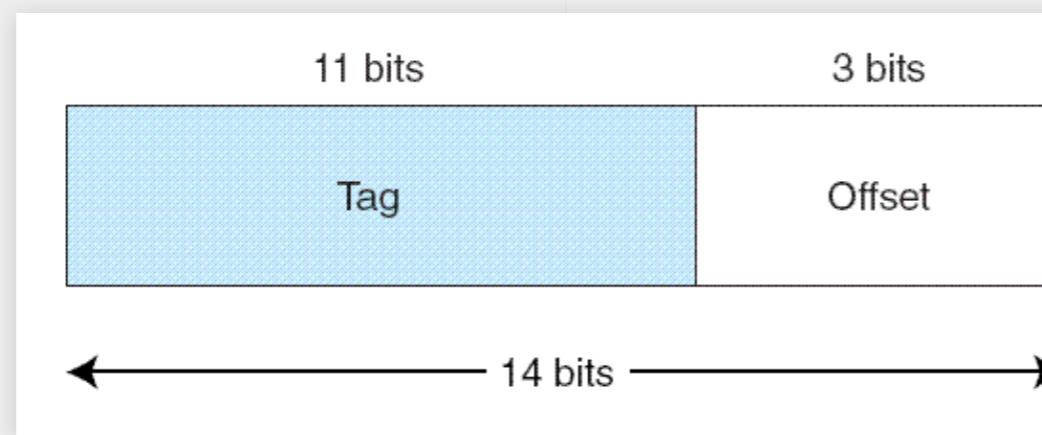
全相联映射地址结构

全相联映射的地址如下图所示，分为两个字段：

- 块内偏移值：代表数据在块内的地址偏移值
- 标记：代表数据在主存中的存储块编号

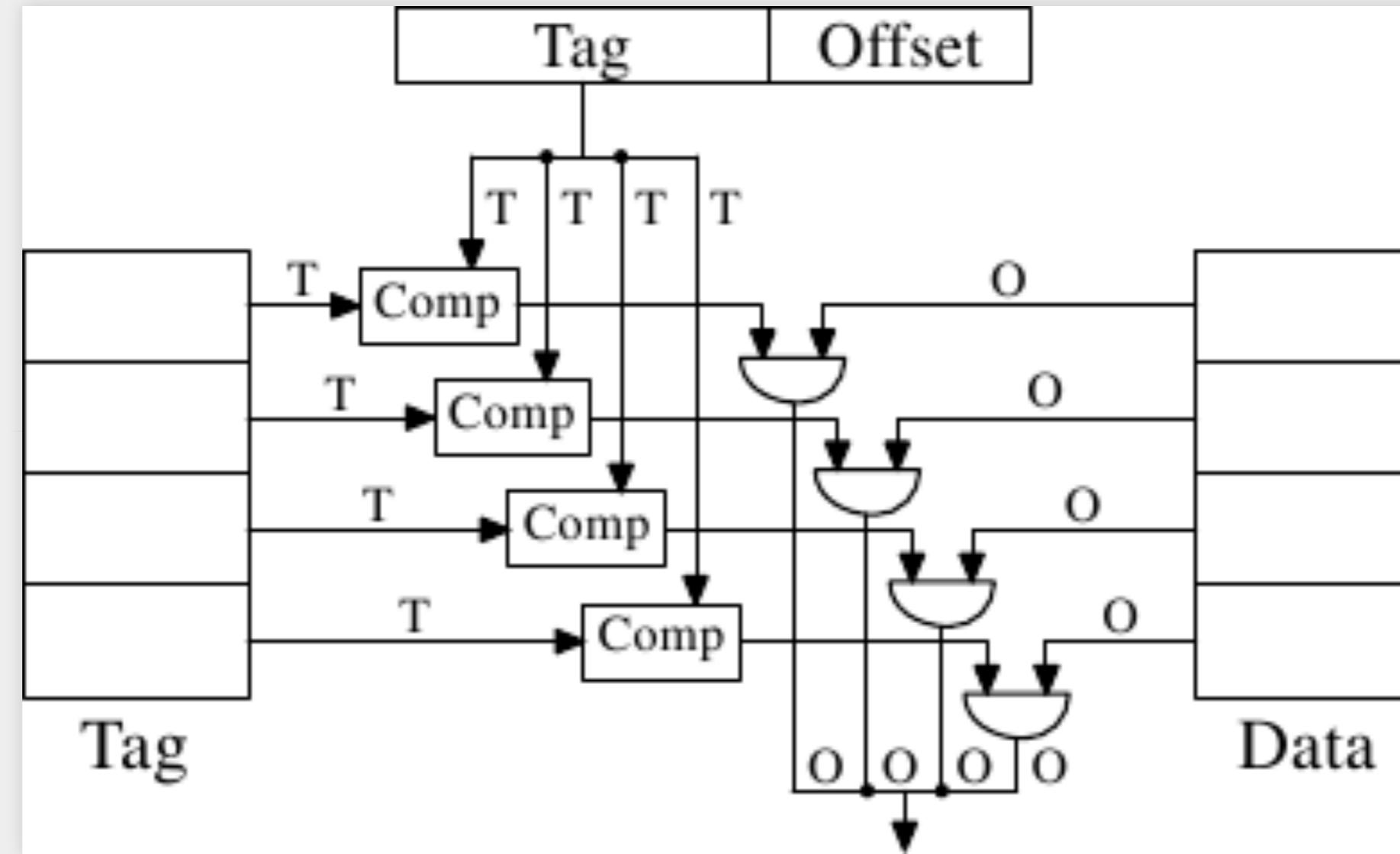
示例：假设一个按字节寻址的主存采用14位地址，块大小为8个字节，缓存共有16个块

- 地址长度=14
- 偏移值字段长度= $\log_2 (8) = 3$
- 标记字段长度=地址长度-偏移值字段长度=11



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

全相联映射的电路实现



图片来源：<https://lwn.net/Articles/252125/>

全相联映射的特点

全相联映射和直接映射相比：

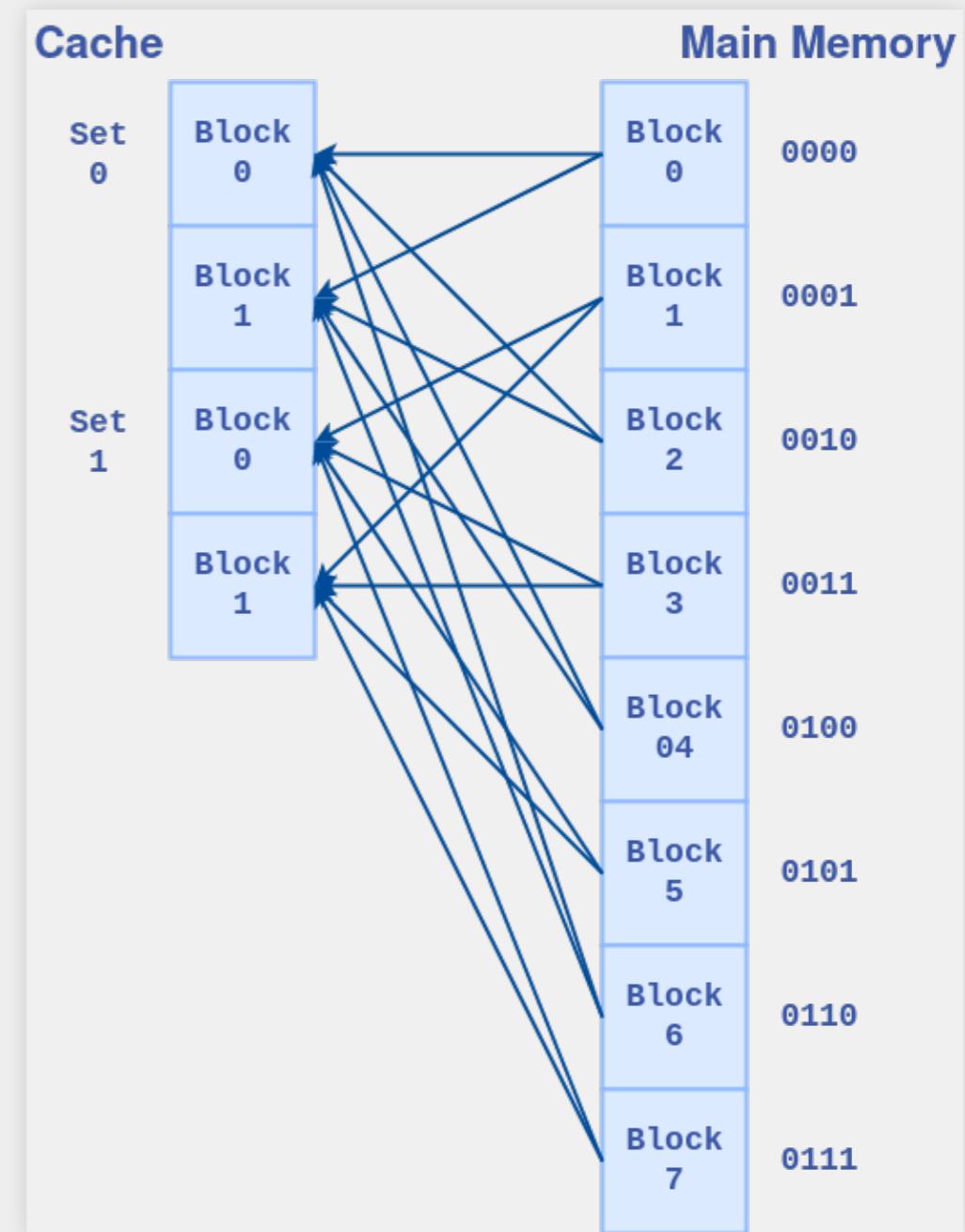
- 缓存利用率高
- 命中率更高
- 标识位较长，占用存储更多
- 成本更高

在实际中，采用两种映射方式结合的折中方案：组相联映射(Set Associative Mapping)

组相联映射

N路组相联映射(N-way Set Associative Mapping)映射方法如下：

- 缓存由 M 个组构成，每个组包含 N 个块
- 内存中编号为 X 的块映射到缓存中编号为 Y 的组，且 $Y = X \bmod M$
- 在编号为 Y 的组中，通过相联存储电路，根据 X 的标记位找到对应的数据块
- 例：假设缓存采用2组2路组相联映射，内存有8个块，则内存和缓存的映射关系如右图所示



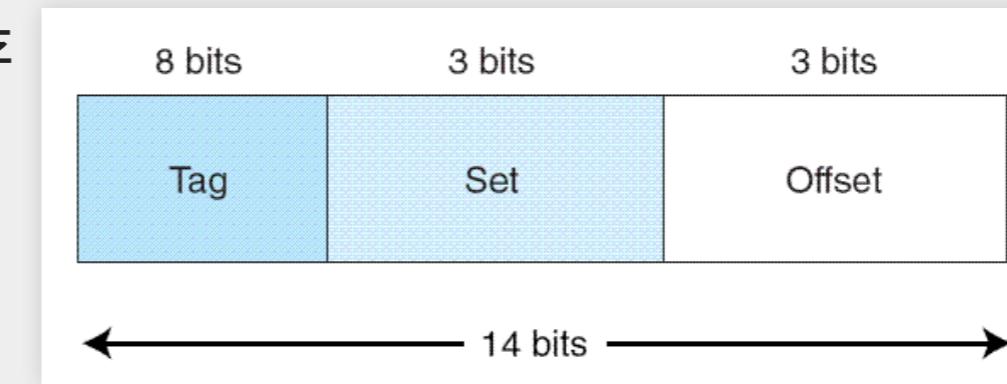
组相联映射的地址结构

组相联映射的地址如下图所示，分为三个字段：

- 块内偏移值 (offset)：代表数据在块内的地址偏移值
- 组号 (set)：代表数据在缓存内的组编号
- 标记 (tag)：标记和组号共同代表数据在主存中的存储块编号

示例：假设一个按字节寻址的主存采用14位地址，块大小为8个字节，缓存采用8组2路组相联映射

- 偏移值字段长度= $\log_2 (8) = 3$
- 组编号字段长度= $\log_2 (8) = 3$
- 标记字段长度=地址长度-偏移值字段长度-组编号字段长度=8



图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

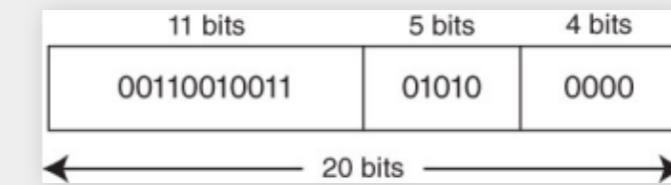
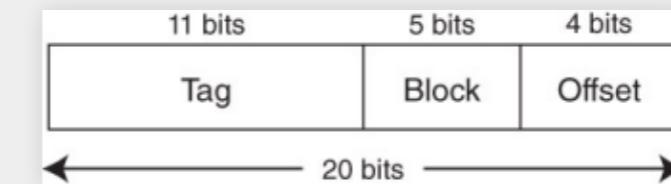
缓存映射示例

假设一个按字节寻址的存储器总容量为1MB，缓存有32个块，每块有16个字节，分析采用直接映射、全相联映射和4路组相联映射时，地址为0x326A0(0011 0010 0110 1010 0000)在缓存中的位置

缓存映射示例

假设一个按字节寻址的存储器总容量为1MB，缓存有32个块，每块有16个字节，分析采用直接映射、全相联映射和4路组相联映射时，地址为0x326A0(0011 0010 0110 1010 0000)在缓存中的位置

- 直接映射：地址总位数20,偏移值字段长度4,块编号字段长度5,标记字段长度11



位于编号为0x0A的缓存块

图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

缓存映射示例

假设一个按字节寻址的存储器总容量为1MB，缓存有32个块，每块有16个字节，分析采用直接映射、全相联映射和4路组相联映射时，地址为0x326A0(0011 0010 0110 1010 0000)在缓存中的位置



无法确定具体位置

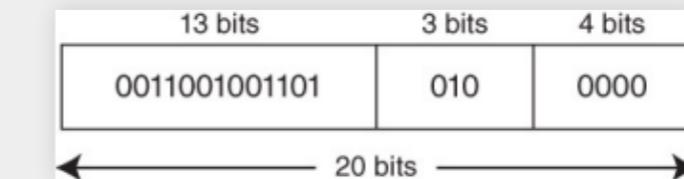
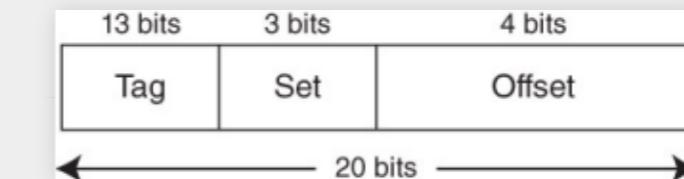
图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

- 直接映射：地址总位数20,偏移值字段长度4,块编号字段长度5,标记字段长度11
- 全相联映射：地址总位数20,偏移值字段长度4,标记字段长度16

缓存映射示例

假设一个按字节寻址的存储器总容量为1MB，缓存有32个块，每块有16个字节，分析采用直接映射、全相联映射和4路组相联映射时，地址为0x326A0(0011 0010 0110 1010 0000)在缓存中的位置

- 直接映射：地址总位数20,偏移值字段长度4,块编号字段长度5,标记字段长度11
- 全相联映射：地址总位数20,偏移值字段长度4,标记字段长度16
- 4路组相联映射：地址总位数20,偏移值字段长度4,组编号字段长度3,标记字段长度



位于编号为0x02的组中，无法确定具体位置

图片来源：Linda Null and Julia Lobur, *Computer Organization and Architecture*, 4th Edition

缓存替换策略

- 对于全相联映射和N路组相联映射，当缓存空间不足时需要用新的数据块替换旧的数据块，选择被替换数据块的方法称为替换策略
- 理论最优的替换策略：将未来一段时间内访问频率最低的块换出(evict)
- 常见替换策略：
 - 最近最少使用(Least Recently Used, LRU)：根据过去一段时间采样的结果，将使用频率最低的块换出
 - 先入先出(First-in First-out, FIFO)：将缓存中存在时间最长的数据块换出
 - 随机替换：随机选择数据块换出，可能会导致经常使用的块被换出，但是可以避免同一个数据块被频繁换入换出

缓存综合示例

我们用一个例子来综合展示缓存的映射、查找和替换

假设一个按字节寻址的存储器使用8位地址，一个块包含4个字节，缓存共有8个块。假设初始状态缓存为空，写出依次访问0x01/0x04/0x05/0x21/0x01/0x41的缓存命中结果

直接映射

访问地址	地址字段	是否命中
0x01	000 000 01	
0x04	000 001 00	
0x05	000 001 01	
0x21	001 000 01	
0x01	000 000 01	
0x41	010 000 01	

直接映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000 000 01		000	-	-
0x04	000 001 00		001	-	-
0x05	000 001 01		010	-	-
0x21	001 000 01		011	-	-
0x01	000 000 01		100	-	-
0x41	010 000 01		101	-	-
			110	-	-
			111	-	-

直接映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000 000 01	否	000	000	0x00 - 0x03
0x04	000 001 00		001	-	-
0x05	000 001 01		010	-	-
0x21	001 000 01		011	-	-
0x01	000 000 01		100	-	-
0x41	010 000 01		101	-	-
			110	-	-
			111	-	-

直接映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000 000 01	否	000	000	0x00 - 0x03
0x04	000 001 00	否	001	000	0x04 - 0x07
0x05	000 001 01		010	-	-
0x21	001 000 01		011	-	-
0x01	000 000 01		100	-	-
0x41	010 000 01		101	-	-
			110	-	-
			111	-	-

直接映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000 000 01	否	000	000	0x00 - 0x03
0x04	000 001 00	否	001	000	0x04 - 0x07
0x05	000 001 01	是	010	-	-
0x21	001 000 01		011	-	-
0x01	000 000 01		100	-	-
0x41	010 000 01		101	-	-
			110	-	-
			111	-	-

直接映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000 000 01	否	000	001	0x20 - 0x23
0x04	000 001 00	否	001	000	0x04 - 0x07
0x05	000 001 01	是	010	-	-
0x21	001 000 01	否	011	-	-
0x01	000 000 01		100	-	-
0x41	010 000 01		101	-	-
			110	-	-
			111	-	-

直接映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000 000 01	否	000	000	0x00 - 0x03
0x04	000 001 00	否	001	000	0x04 - 0x07
0x05	000 001 01	是	010	-	-
0x21	001 000 01	否	011	-	-
0x01	000 000 01	否	100	-	-
0x41	010 000 01		101	-	-
			110	-	-
			111	-	-

直接映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000 000 01	否	000	010	0x40 - 0x43
0x04	000 001 00	否	001	000	0x04 - 0x07
0x05	000 001 01	是	010	-	-
0x21	001 000 01	否	011	-	-
0x01	000 000 01	否	100	-	-
0x41	010 000 01	否	101	-	-
			110	-	-
			111	-	-

全相联映射

访问地址	地址字段	是否命中
0x01	000000 01	
0x04	000001 00	
0x05	000001 01	
0x21	001000 01	
0x01	000000 01	
0x41	010000 01	

全相联映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000000 01		000	-	-
0x04	000001 00		001	-	-
0x05	000001 01		010	-	-
0x21	001000 01		011	-	-
0x01	000000 01		100	-	-
0x41	010000 01		101	-	-
			110	-	-
			111	-	-

全相联映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000000 01	否	000	000000	0x00 - 0x03
0x04	000001 00		001	-	-
0x05	000001 01		010	-	-
0x21	001000 01		011	-	-
0x01	000000 01		100	-	-
0x41	010000 01		101	-	-
			110	-	-
			111	-	-

全相联映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000000 01	否	000	000000	0x00 - 0x03
0x04	000001 00	否	001	000001	0x04 - 0x07
0x05	000001 01		010	-	-
0x21	001000 01		011	-	-
0x01	000000 01		100	-	-
0x41	010000 01		101	-	-
			110	-	-
			111	-	-

全相联映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000000 01	否	000	000000	0x00 - 0x03
0x04	000001 00	否	001	000001	0x04 - 0x07
0x05	000001 01	是	010	-	-
0x21	001000 01		011	-	-
0x01	000000 01		100	-	-
0x41	010000 01		101	-	-
			110	-	-
			111	-	-

全相联映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000000 01	否	000	000000	0x00 - 0x03
0x04	000001 00	否	001	000001	0x04 - 0x07
0x05	000001 01	是	010	001000	0x20 - 0x23
0x21	001000 01	否	011	-	-
0x01	000000 01		100	-	-
0x41	010000 01		101	-	-
			110	-	-
			111	-	-

全相联映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000000 01	否	000	000000	0x00 - 0x03
0x04	000001 00	否	001	000001	0x04 - 0x07
0x05	000001 01	是	010	001000	0x20 - 0x23
0x21	001000 01	否	011	-	-
0x01	000000 01	是	100	-	-
0x41	010000 01		101	-	-
			110	-	-
			111	-	-

全相联映射

访问地址	地址字段	是否命中	缓存块编号	缓存标记	内存地址范围
0x01	000000 01	否	000	000000	0x00 - 0x03
0x04	000001 00	否	001	000001	0x04 - 0x07
0x05	000001 01	是	010	001000	0x20 - 0x23
0x21	001000 01	否	011	010000	0x40 - 0x43
0x01	000000 01	是	100	-	-
0x41	010000 01	否	101	-	-
			110	-	-
			111	-	-

采用FIFO的2路组相联映射

访问地址	地址字段	是否命中
0x01	0000 00 01	
0x04	0000 01 00	
0x05	0000 01 01	
0x21	0010 00 01	
0x01	0000 00 01	
0x41	0100 00 01	

采用FIFO的2路组相联映射

访问地址	地址字段	是否命中	缓存组编号	第一路标记	第二路标记	内存地址范围
0x01	0000 00 01		00	-	-	-
0x04	0000 01 00		01	-	-	-
0x05	0000 01 01		10	-	-	-
0x21	0010 00 01		11	-	-	-
0x01	0000 00 01					
0x41	0100 00 01					

采用FIFO的2路组相联映射

访问地址	地址字段	是否命中	缓存组编号	第一路标记	第二路标记	内存地址范围
0x01	0000 00 01	否	00	0000	-	0x00-0x03
0x04	0000 01 00		01	-	-	-
0x05	0000 01 01		10	-	-	-
0x21	0010 00 01		11	-	-	-
0x01	0000 00 01					
0x41	0100 00 01					

采用FIFO的2路组相联映射

访问地址	地址字段	是否命中	缓存组编号	第一路标记	第二路标记	内存地址范围
0x01	0000 00 01	否	00	0000	-	0x00-0x03
0x04	0000 01 00	否	01	0000	-	0x04-0x07
0x05	0000 01 01		10	-	-	-
0x21	0010 00 01		11	-	-	-
0x01	0000 00 01					
0x41	0100 00 01					

采用FIFO的2路组相联映射

访问地址	地址字段	是否命中	缓存组编号	第一路标记	第二路标记	内存地址范围
0x01	0000 00 01	否	00	0000	-	0x00-0x03
0x04	0000 01 00	否	01	0000	-	0x04-0x07
0x05	0000 01 01	是	10	-	-	-
0x21	0010 00 01		11	-	-	-
0x01	0000 00 01					
0x41	0100 00 01					

采用FIFO的2路组相联映射

访问地址	地址字段	是否命中	缓存组编号	第一路标记	第二路标记	内存地址范围
0x01	0000 00 01	否	00	0000	0010	0x00-0x03, 0x20-0x23
0x04	0000 01 00	否	01	0000	-	0x04-0x07
0x05	0000 01 01	是	10	-	-	-
0x21	0010 00 01	否	11	-	-	-
0x01	0000 00 01					
0x41	0100 00 01					

采用FIFO的2路组相联映射

访问地址	地址字段	是否命中	缓存组编号	第一路标记	第二路标记	内存地址范围
0x01	0000 00 01	否	00	0000	0010	0x00-0x03, 0x20-0x23
0x04	0000 01 00	否	01	0000	-	0x04-0x07
0x05	0000 01 01	是	10	-	-	-
0x21	0010 00 01	否	11	-	-	-
0x01	0000 00 01	是				
0x41	0100 00 01					

采用FIFO的2路组相联映射

访问地址	地址字段	是否命中	缓存组编号	第一路标记	第二路标记	内存地址范围
0x01	0000 00 01	否	00	0100	0010	0x40-0x43, 0x20-0x23
0x04	0000 01 00	否	01	0000	-	0x04-0x07
0x05	0000 01 01	是	10	-	-	-
0x21	0010 00 01	否	11	-	-	-
0x01	0000 00 01	是				
0x41	0100 00 01	否				



第七讲结束

本期内容回顾

- 掌握常见的存储器类型及特点
- 掌握存储器层次结构
- 高速缓存
 - 缓存策略的性能指标和缓存失效
 - 缓存的映射策略
 - 缓存的替换策略



Q & A