



# LoongArch CPU 设计实战

## 第4天：访存系统与缓存设计

教师：杨建磊、万寒

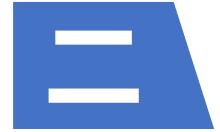
助教：苏阳、周振源

北京航空航天大学 计算机学院

2025年7月31日 四川 成都



## 存储管理



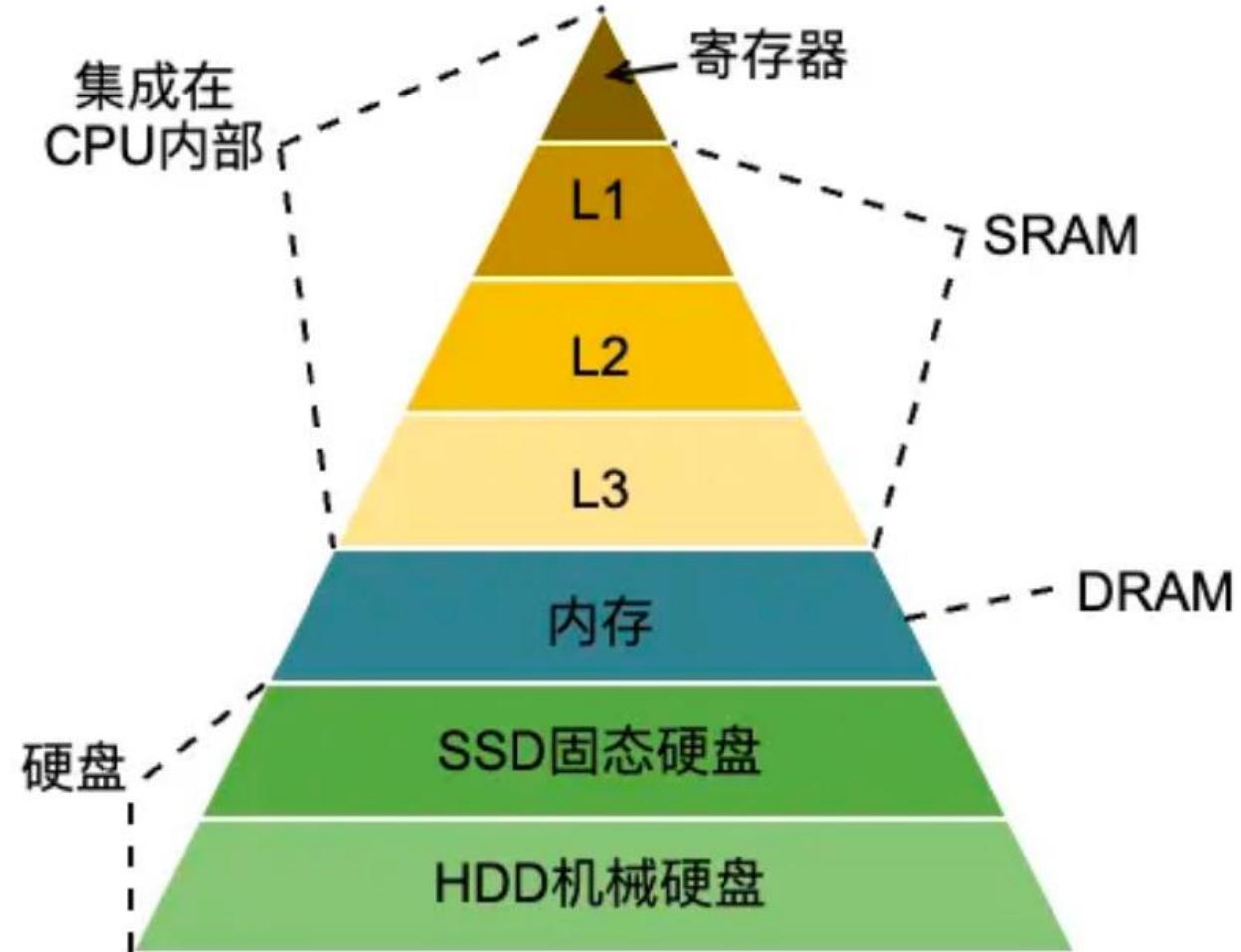
## Cache设计



## 实验

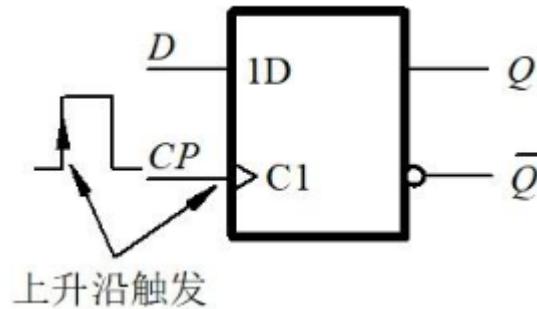
# 1.1 访存系统结构

- 由于物理实现上存在差异，CPU和内存的速度提升幅度一直存在差距，而且这种差距随着时间的推移越来越大。



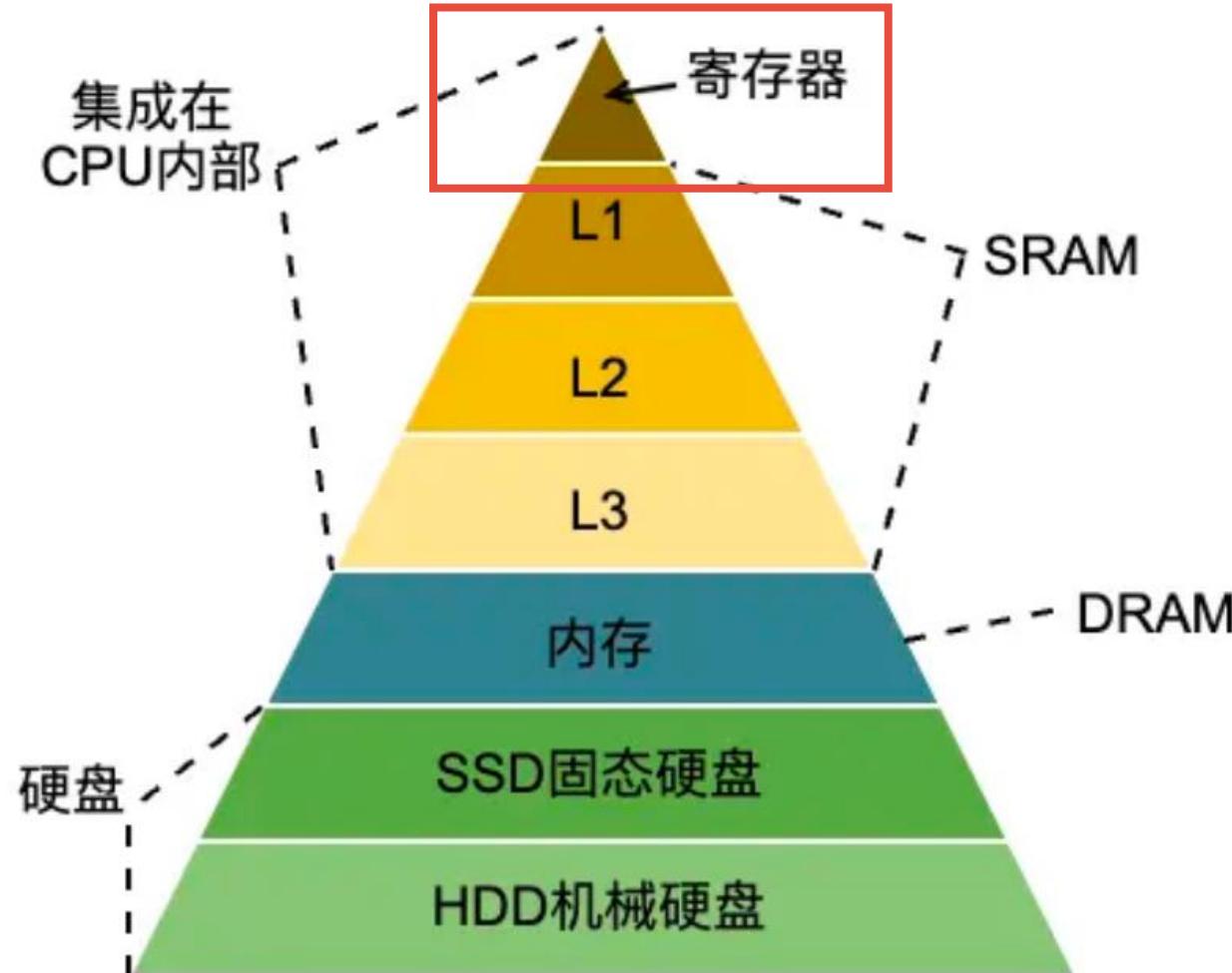
# 1.1 访存系统结构

- 寄存器是处理器内部的高速存储单元，直接与处理器核心相连，用于临时存储数据和指令

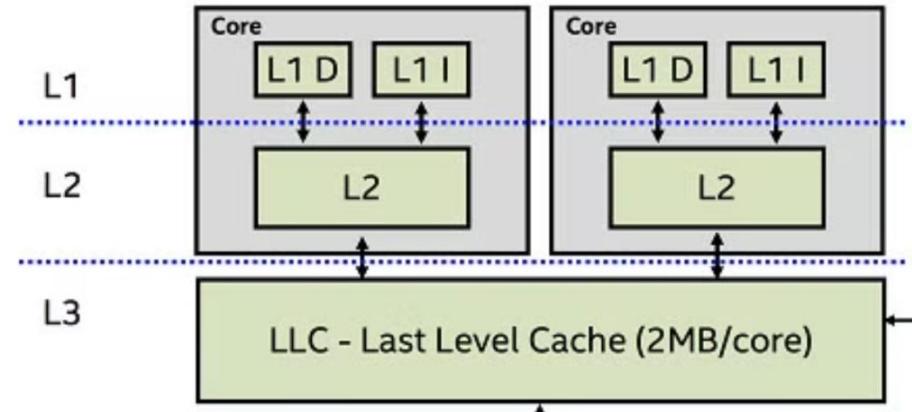
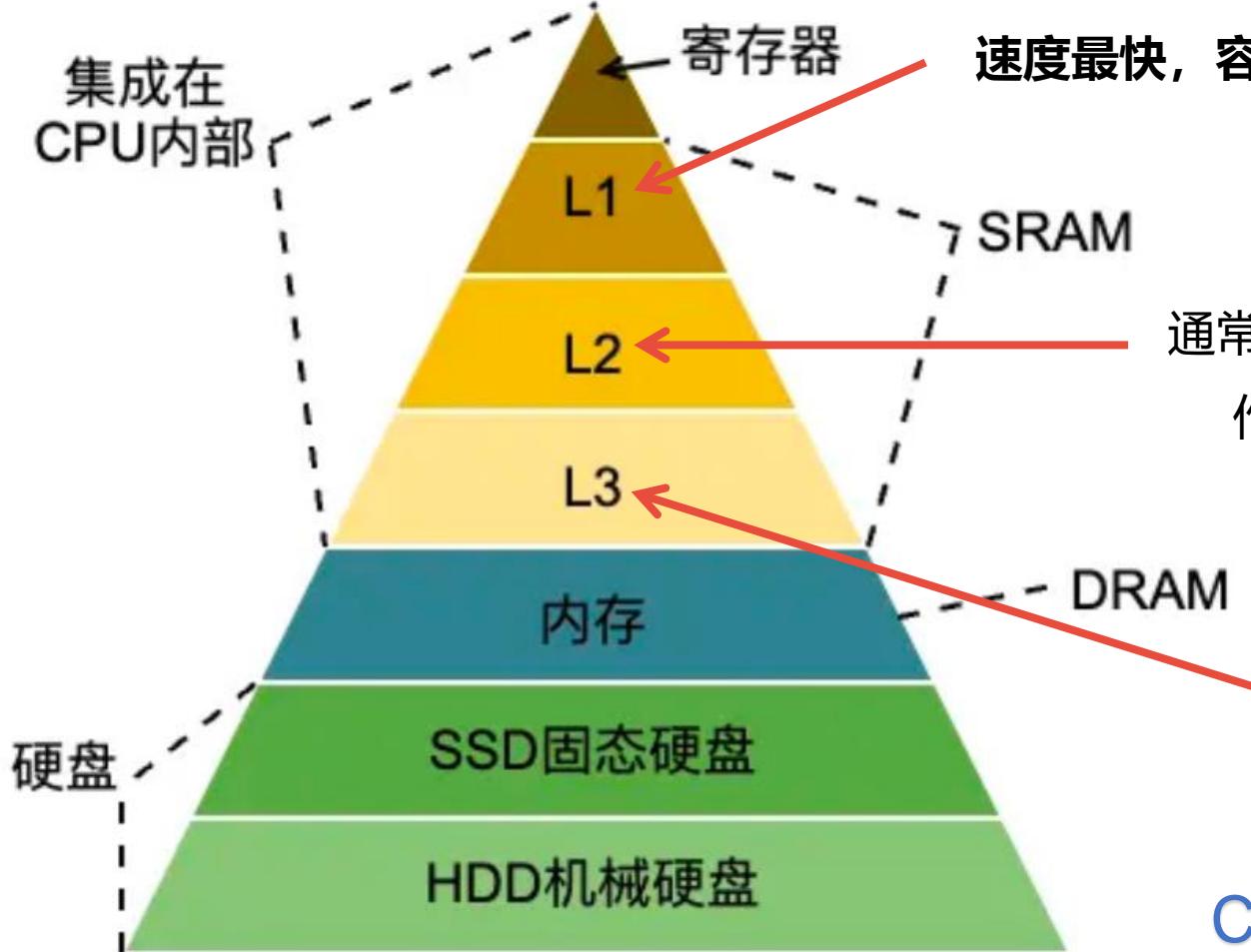


- D触发器是寄存器的实现方式之一
  - CP上升沿到来之间是旧值
  - CP上升沿之后，电路记录新值，抛弃旧值

寄存器是CPU直接可以访问的单元

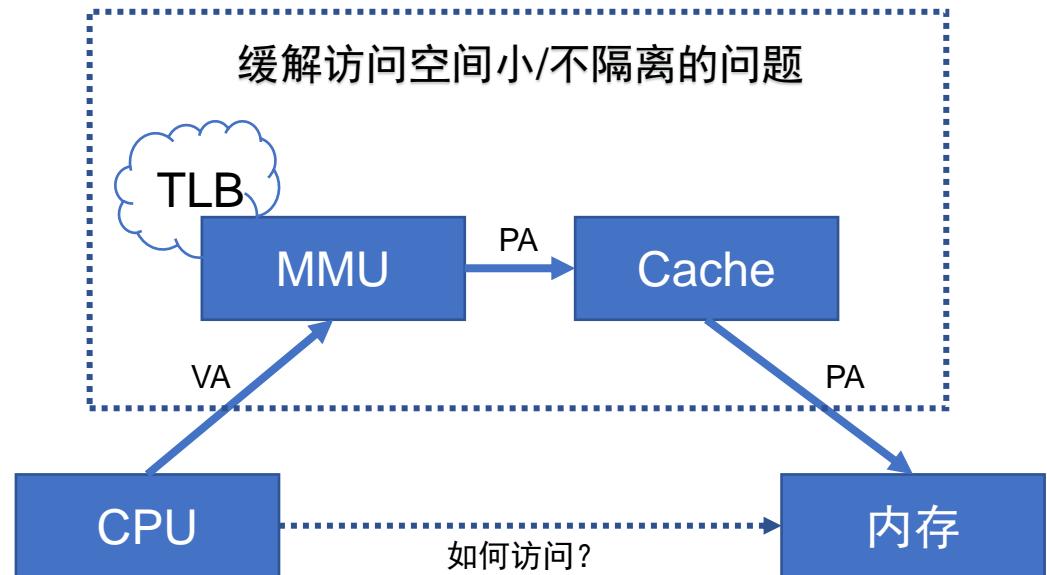
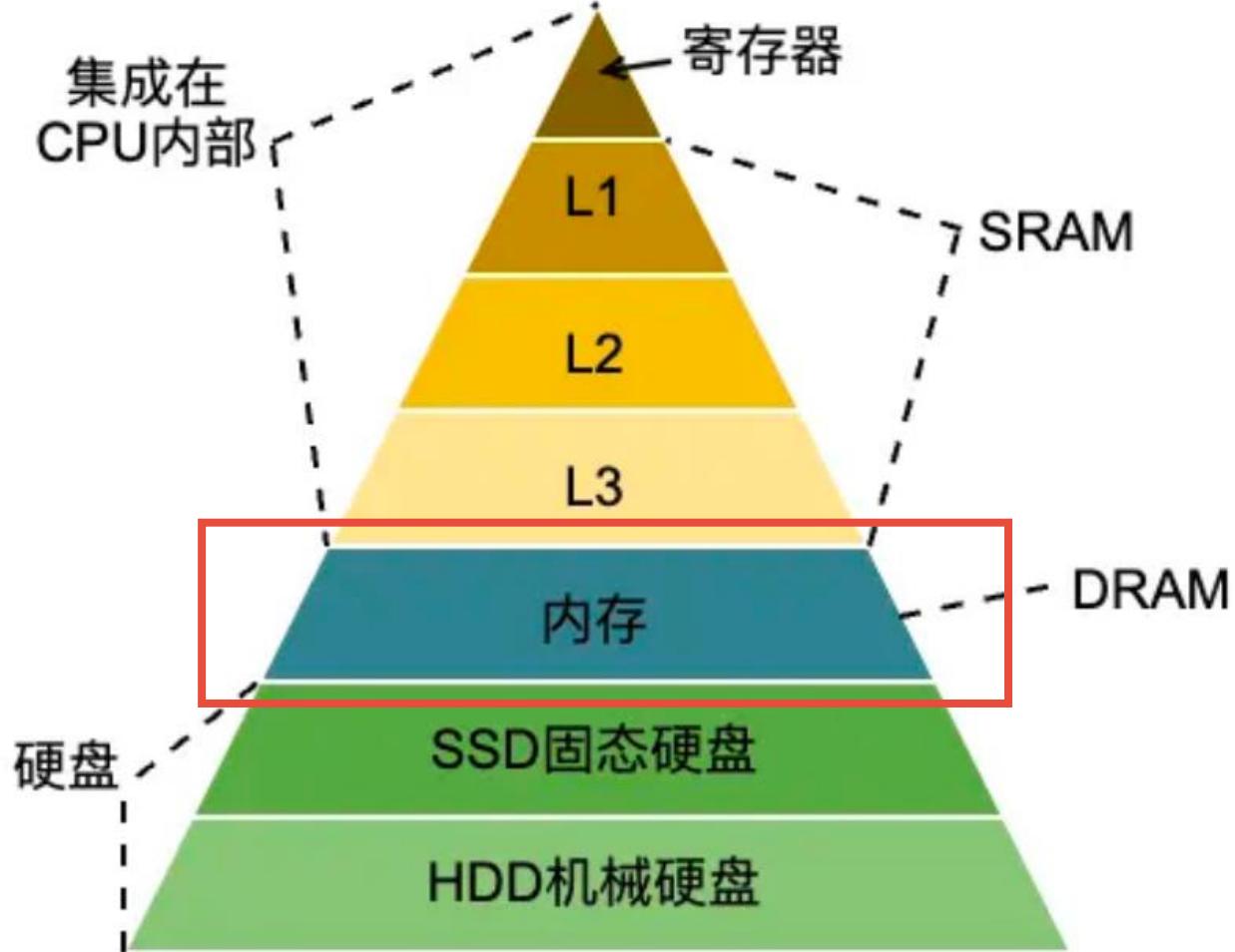


# 1.1 访存系统结构



# 1.1 访存系统结构

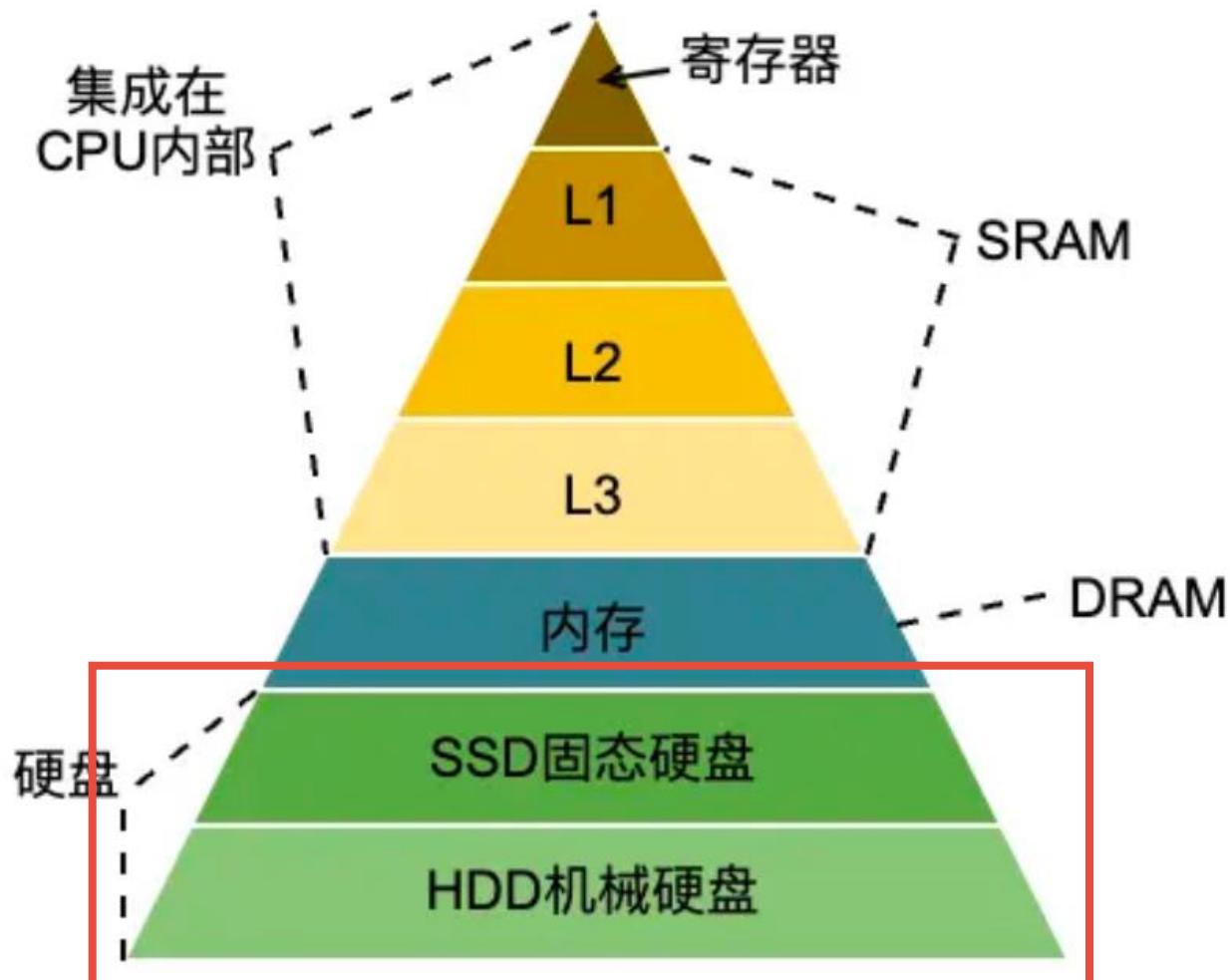
内存是计算机内临时存放正在使用数据和程序的部件



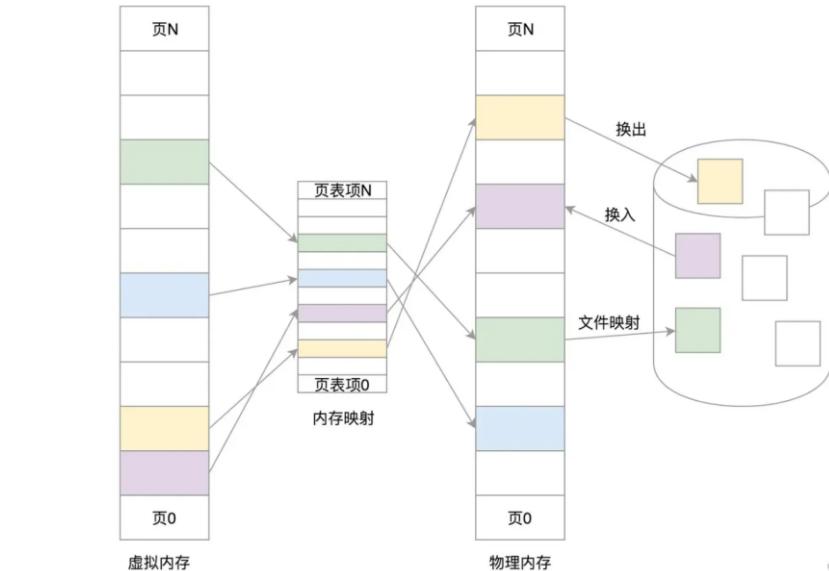
内存是CPU能够通过指令  
直接访问的存储器

# 1.1 访存系统结构

口 辅助存储器，包括固态硬盘、机械硬盘，由操作系统负责管理维护



辅存用于扩大处理器存储空间  
而为了使用辅存——>虚拟地址访问机制



# 1.2 存储器类型

## □ 静态随机存储器 (SRAM)

- 存储器只需要保持通电，数据就可以永远保持
- CPU 的高速缓存通常采用 SRAM



## □ 动态随机存储器 (DRAM)

- DRAM 每隔一段时间刷新一次以保持数据
- 微架构中相对大型存储器可以采用 DRAM



## □ 闪存 (Flash)

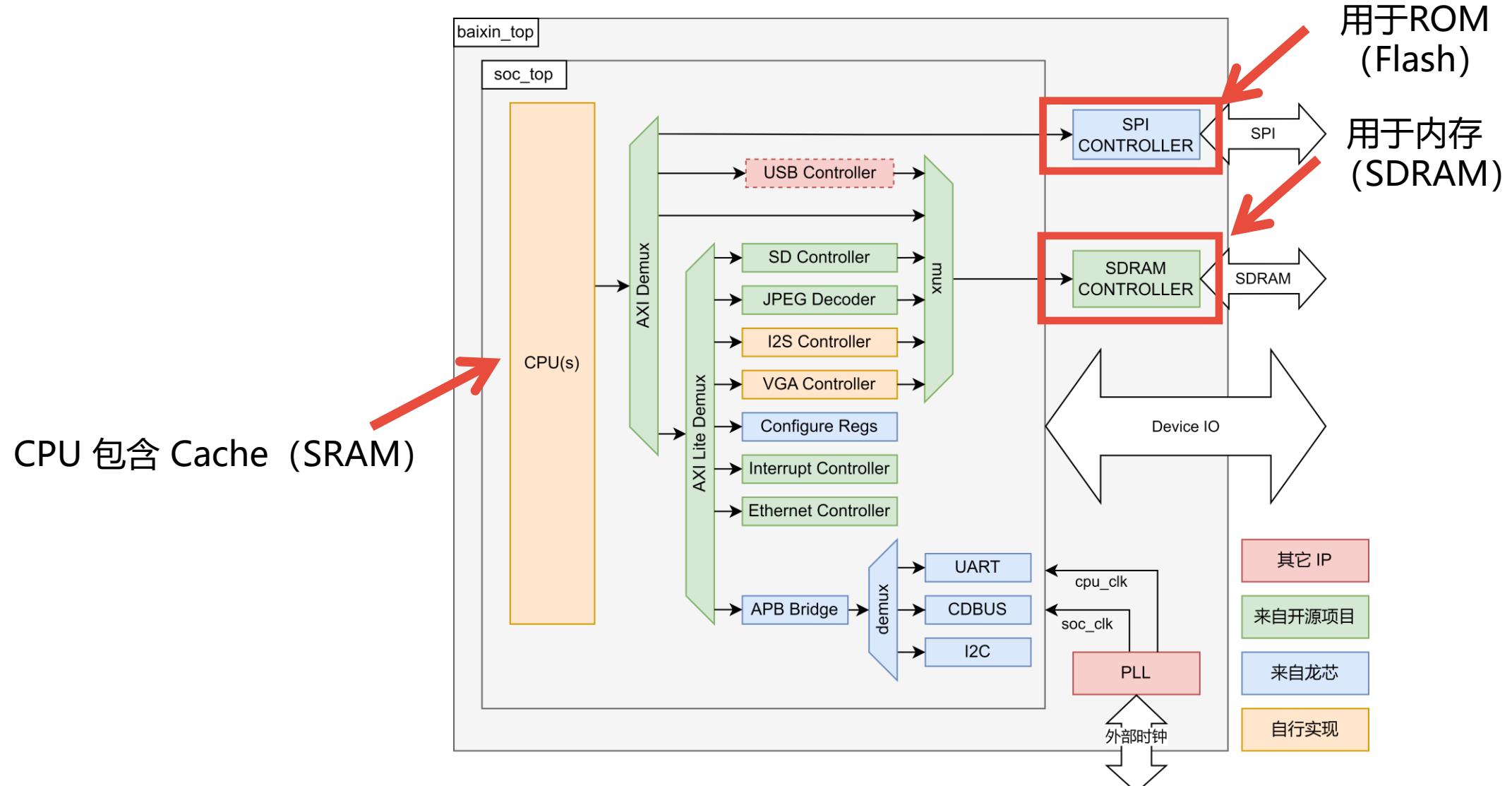
- 长寿命的非易失性存储器
- BootLoader 通常存储在 ROM 中（只读的闪存）

## □ 双倍速率同步动态随机存储器 (DDR)

- 由 SDRAM 发展而来 (SDRAM 相较 DRAM 多了一个同步接口)

# 1.2 存储器类型

## □ SoC中的存储器



# 1.3 地址翻译模式

## □ 虚拟地址空间与地址翻译模式

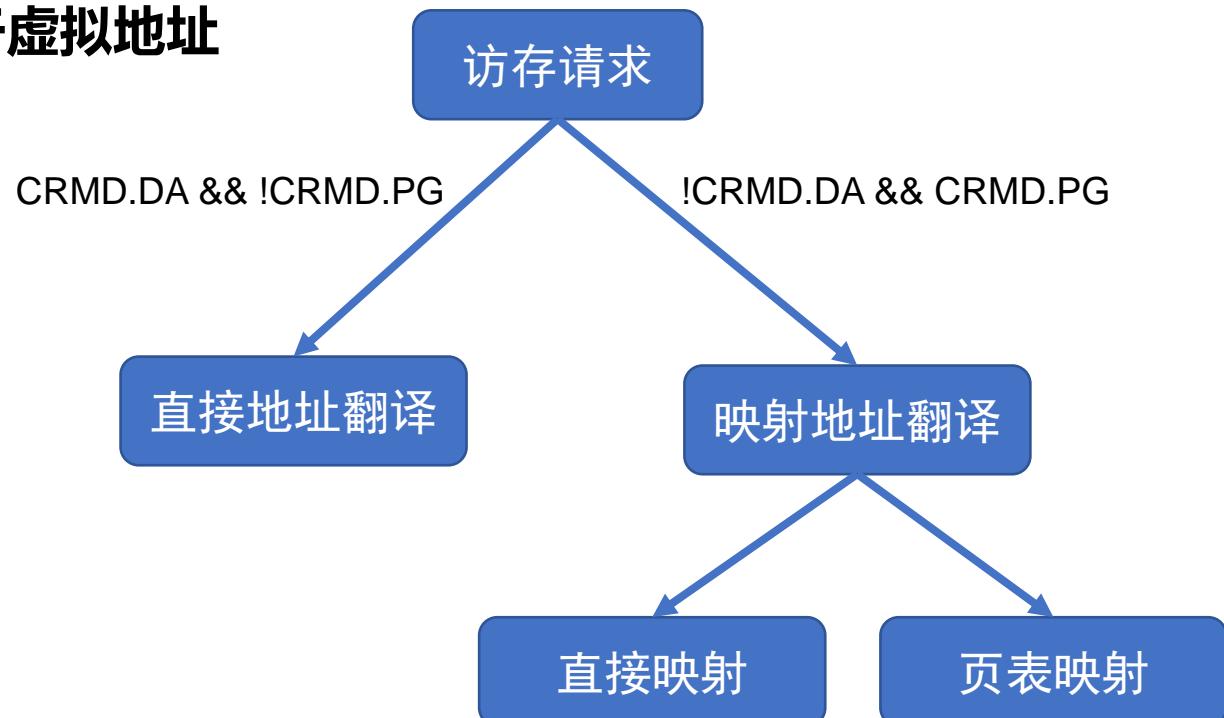
- LA32R的MMU支持两种虚实地址翻译模式

- 直接地址翻译模式
- 映射地址翻译模式

- 直接地址翻译模式——物理地址直接等于虚拟地址

- 映射地址翻译模式

- 直接映射
- 页表映射

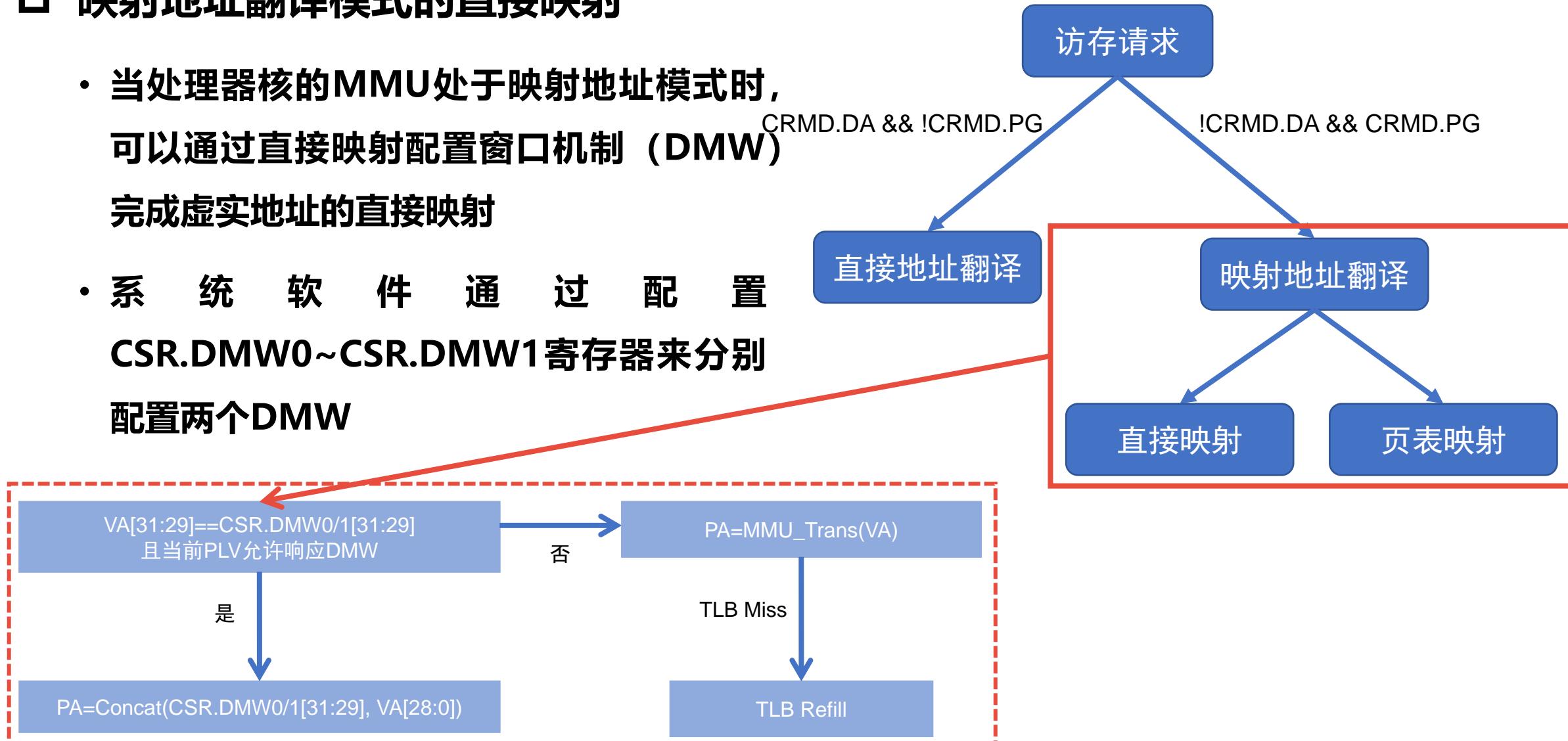


# 1.3 地址翻译模式

## 映射地址翻译模式的直接映射

- 当处理器核的MMU处于映射地址模式时，可以通过直接映射配置窗口机制（DMW）完成虚实地址的直接映射

- 系统软件通过配置 CSR.DMW0~CSR.DMW1 寄存器来分别配置两个DMW



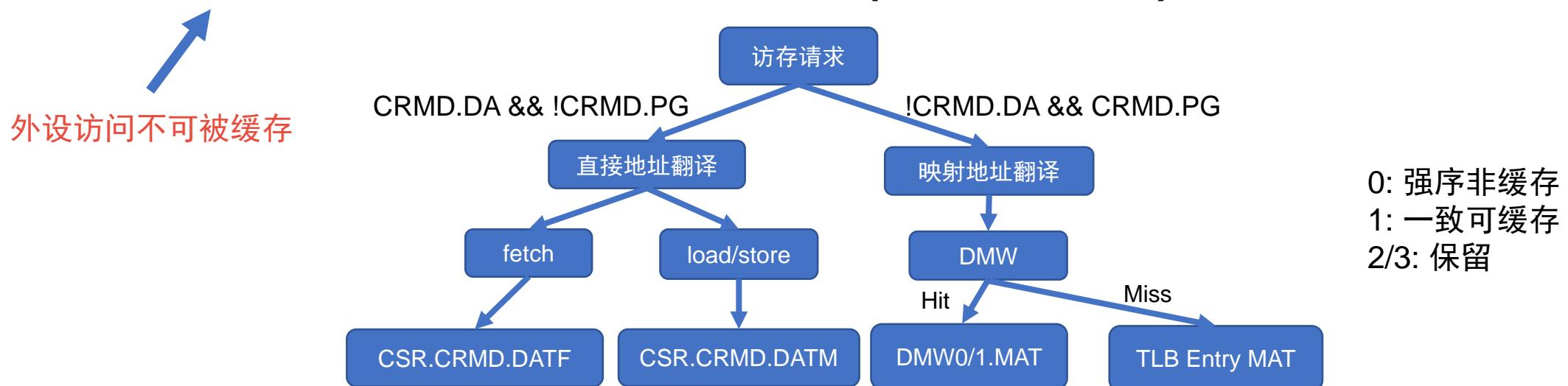
# 1.4 存储访问类型

## □ 一致可缓存 (CC)

- 访问对象既可以是最终存储对象，也可以是处理器中维护有效缓存一致性的缓存
- 通常采用这种访问类型访问内存以获得高性能

## □ 强序非缓存 (SUC)

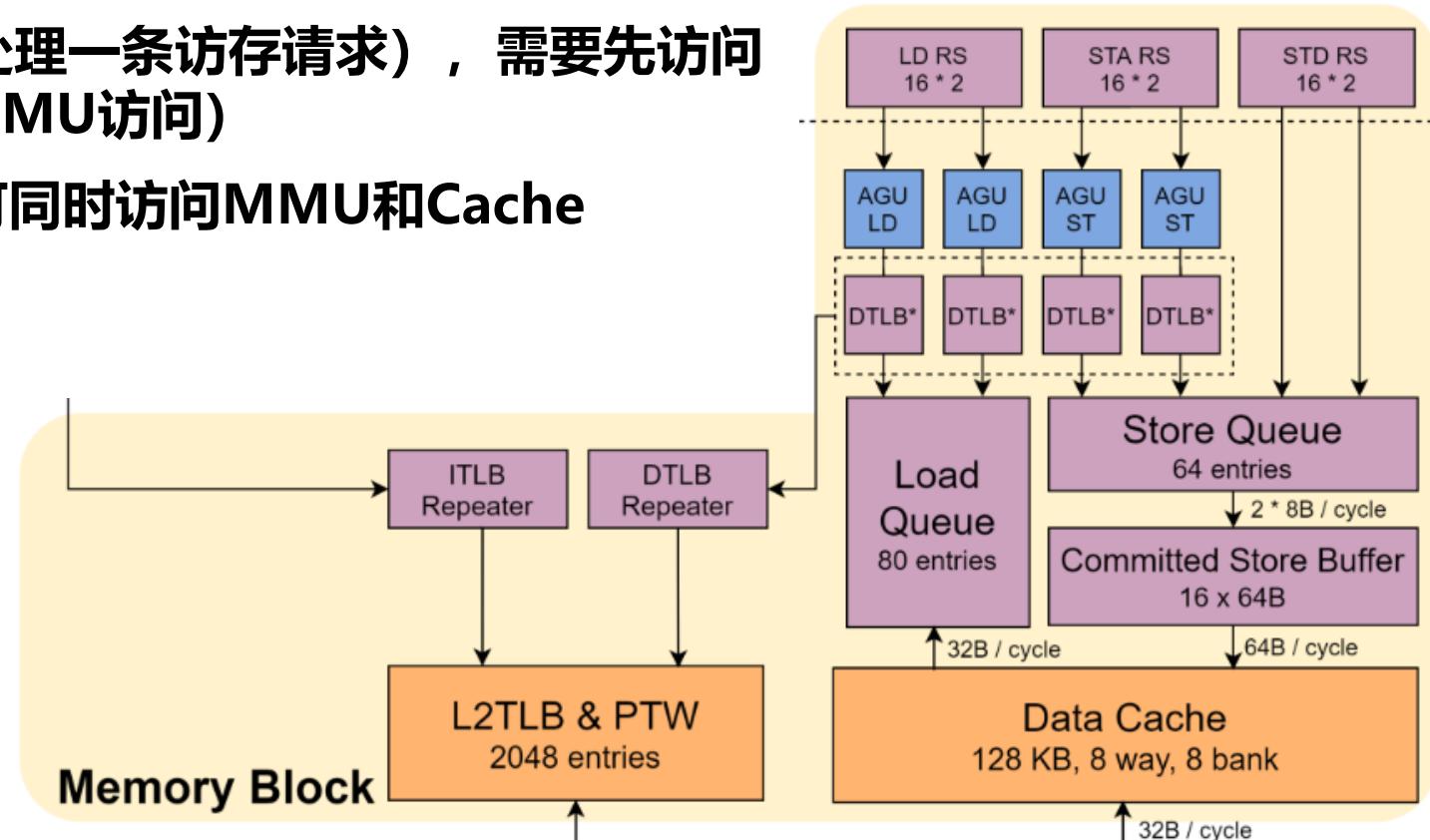
- 所有访问严格按照程序中的次序且当前访存操作彻底完成前不能开始执行下一个访存操作
- 强序非缓存类型的取指操作可以具有副作用（如分支预测失败），其余不行



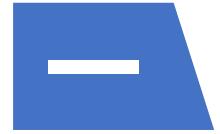
# 1.5 访存单元设计

## 口 访存单元 (LSU)

- 访存单元用于接收源于发射队列的load/store指令
- 对于串行访存单元（即单次仅能处理一条访存请求），需要先访问MMU，再访问Cache（或者由MMU访问）
- 对于流水访存单元，不同的请求可同时访问MMU和Cache
- 建议实现Store Buffer



# Day4 目录



存储管理



Cache设计



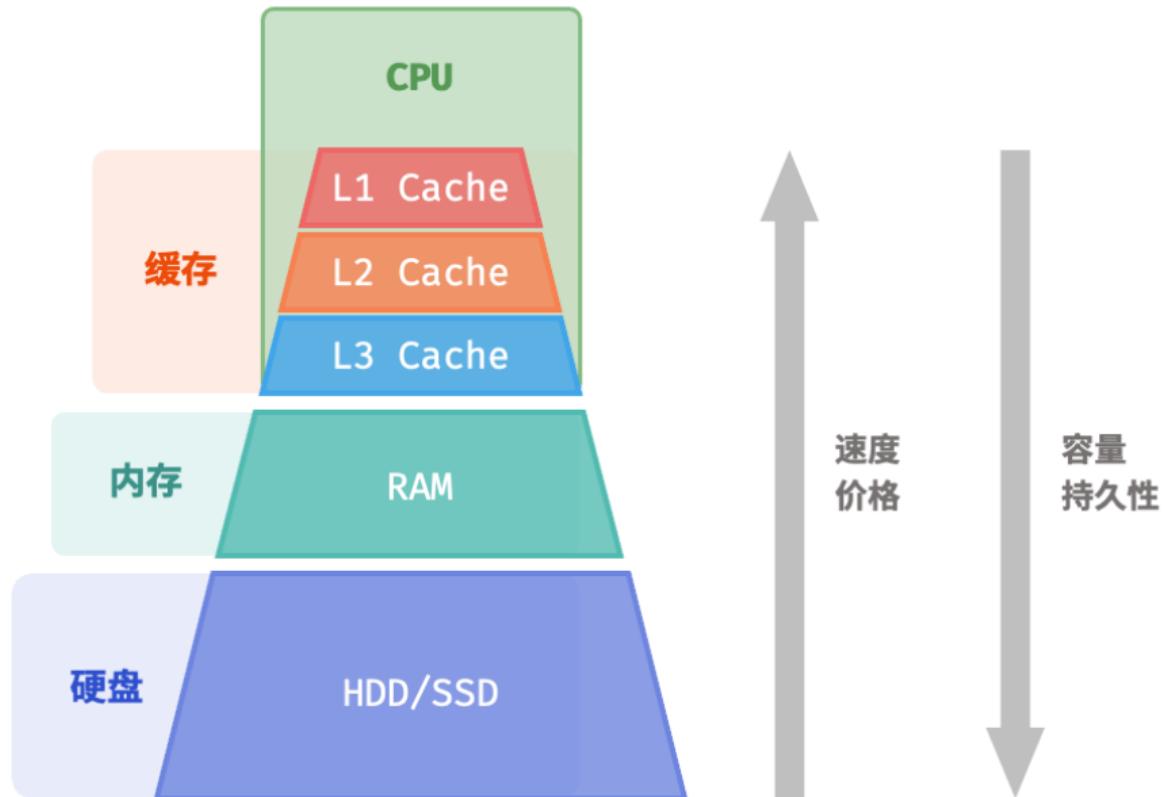
实验

# 2.1 Cache简介

## □ Cache用于弥补内存速度的不足——内存的子集

- 如何划分子集?
- 如何组织子集?
- 如何保持子集和全集的一致性?

Cache的访问速度是内存的 $1/10 \sim 1/100$   
空间大小是内存的 $1/1000 \sim 1/100000$



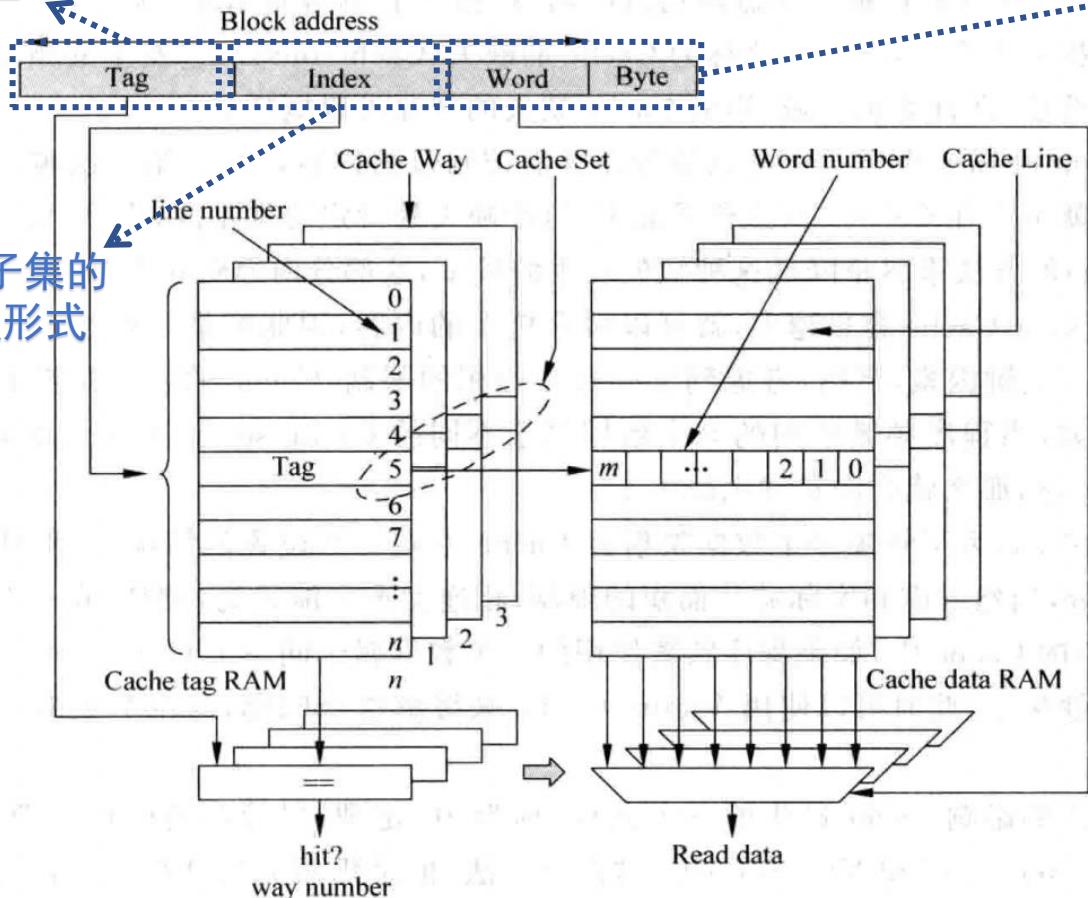
# 2.1 Cache简介

## Cache主要由两部分组成，Tag和Data

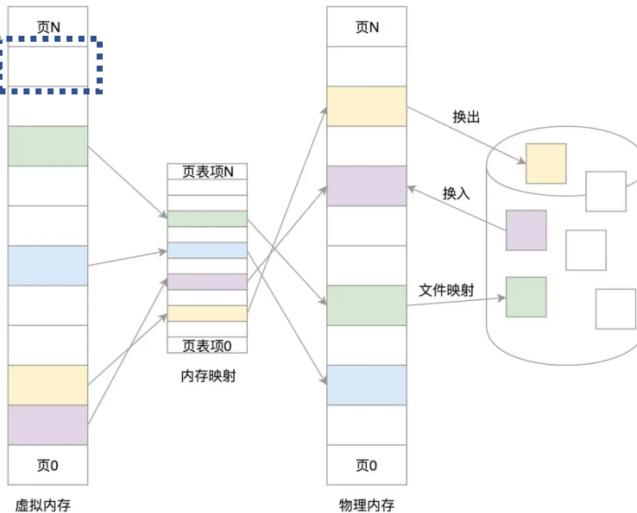
- Tag部分用于存储连续数据的公共地址
- Data部分用于存储实际数据

地址的  
“身份证”

决定子集的  
组织形式



还记得吗，VA/PA都是分页的

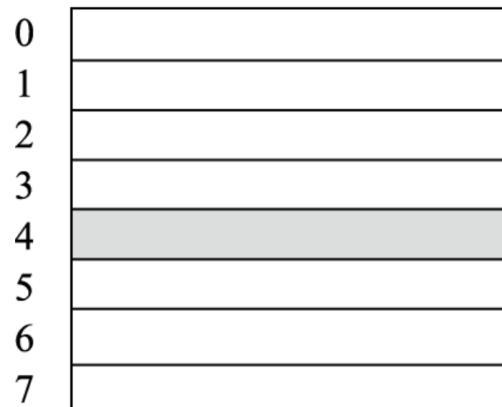


# 2.2 Cache组织形式

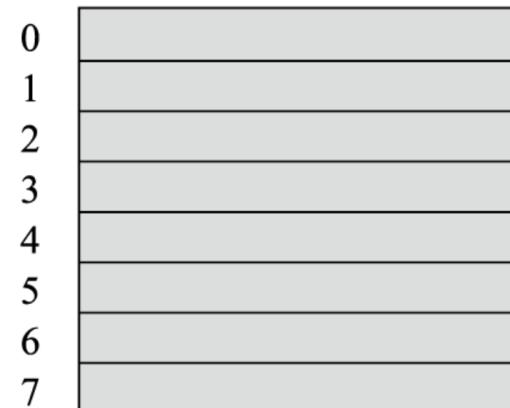
## □ Cache组织形式

根据Cache元数据的Index划分出.....

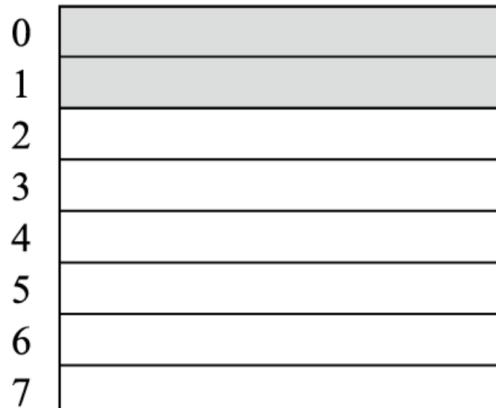
- **直接映射**: 每一个内存块映射到Cache的特定位置
- **组相连** (常用于现代Cache) : 每一个内存块映射到Cache特定组的任意位置
- **全相连** (常用于Victim Cache、TLB) : 任何内存块可映射到Cache的任意位置



直接相连



全相连

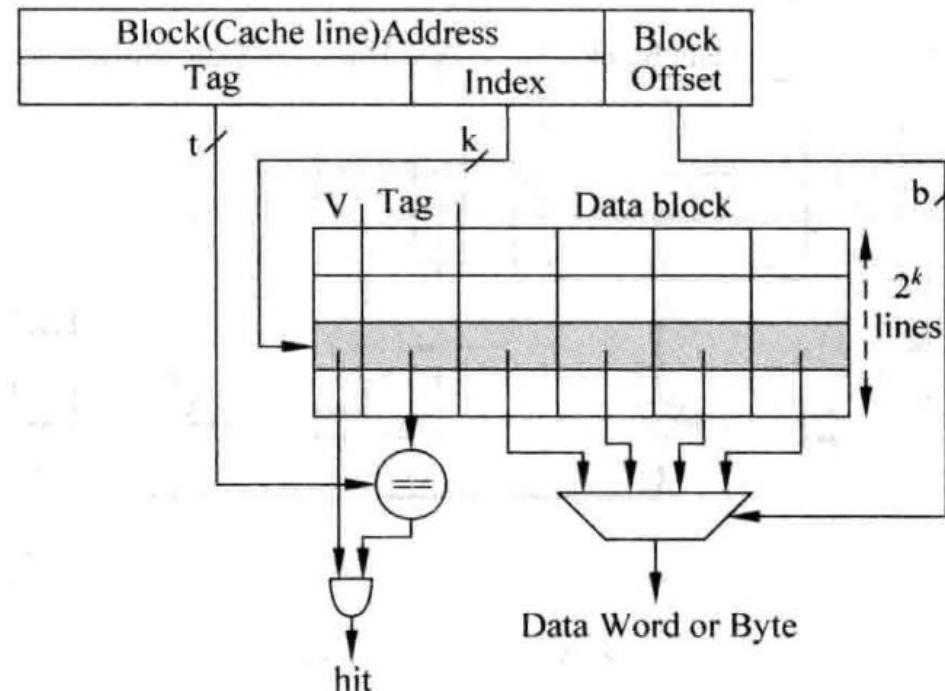
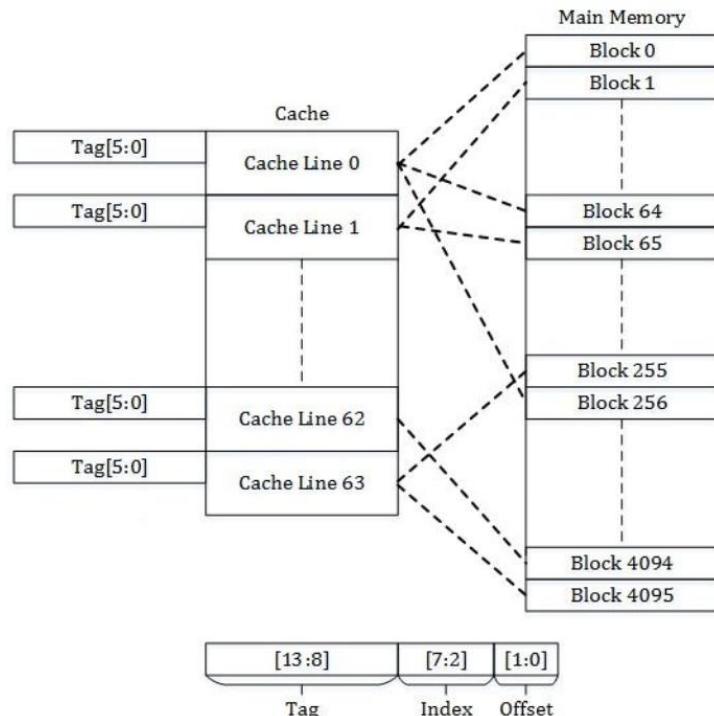


组相连

# 2.2 Cache组织形式

## □ 直接映射

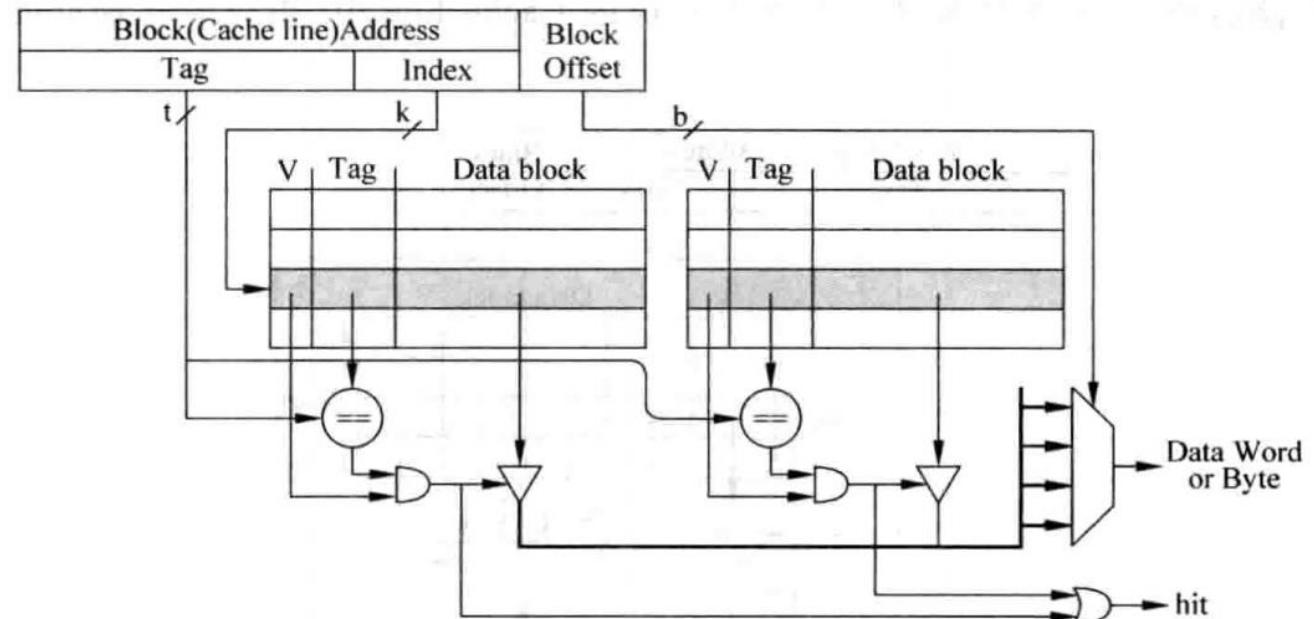
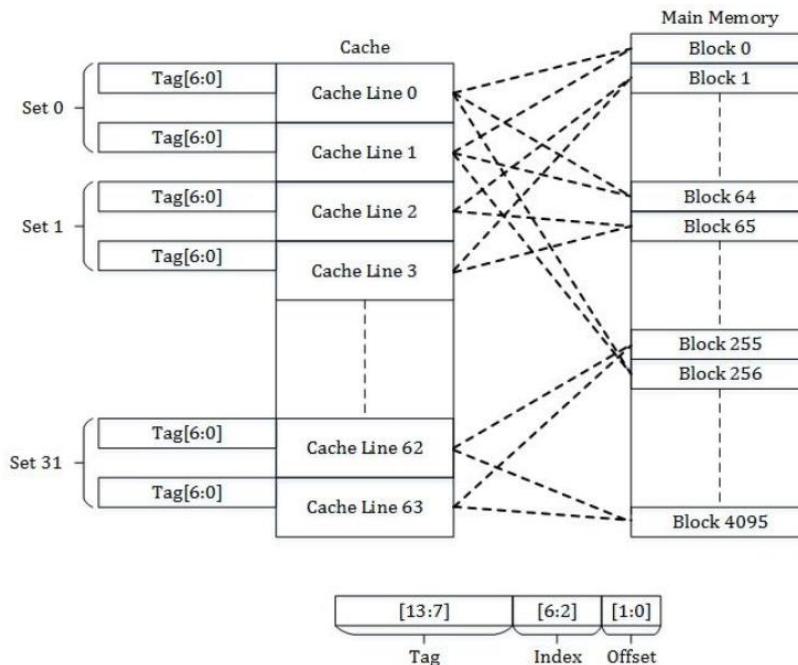
- 指令被划分为Tag、Index、Offset
  - Index用于从Cache中找到对应Cache行
  - Tag用于判断命中情况
  - Offset用于定位字节



# 2.2 Cache组织形式

## 口 组相连

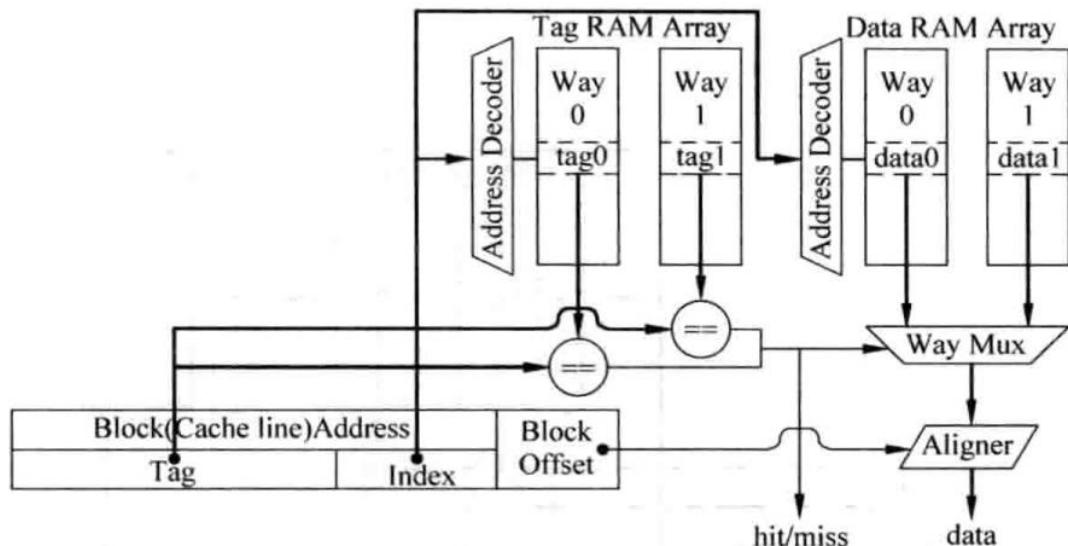
- 指令被划分为Tag、Index、Offset
  - Index用于从Cache中找到对应多个Cache行
  - Tag用于判断命中情况
  - Offset用于定位字节



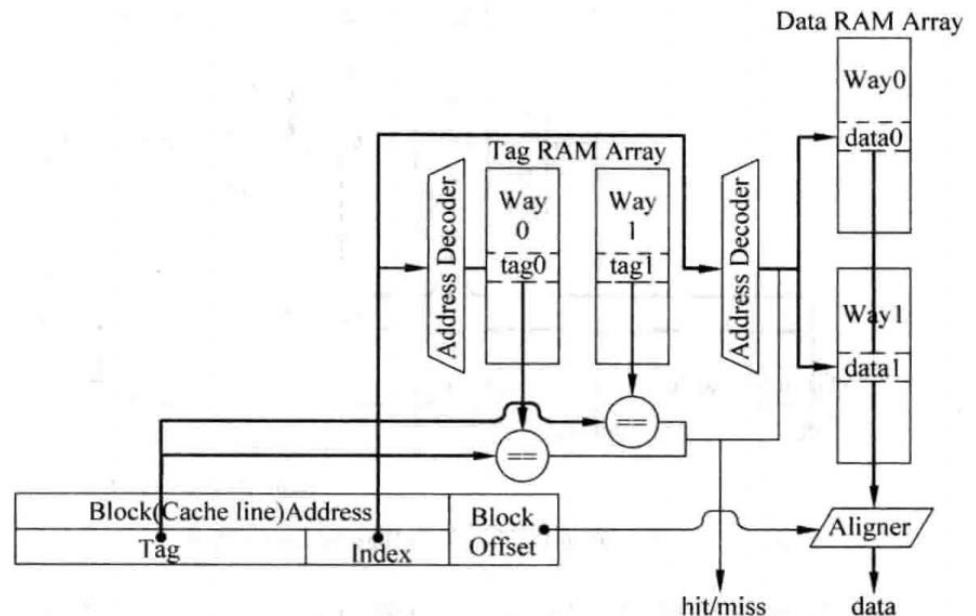
# 2.2 Cache组织形式

## □ 组相连

- 实际中Cache的Tag和Data部分是放开放置的，称为Tag SRAM和Data SRAM
- 同时访问Tag SRAM和Data SRAM的方式称为并行访问（速度快）
- 先访问Tag SRAM，根据Tag比较结果访问Data SRAM的方式称为串行访问（能耗低）



并行访问

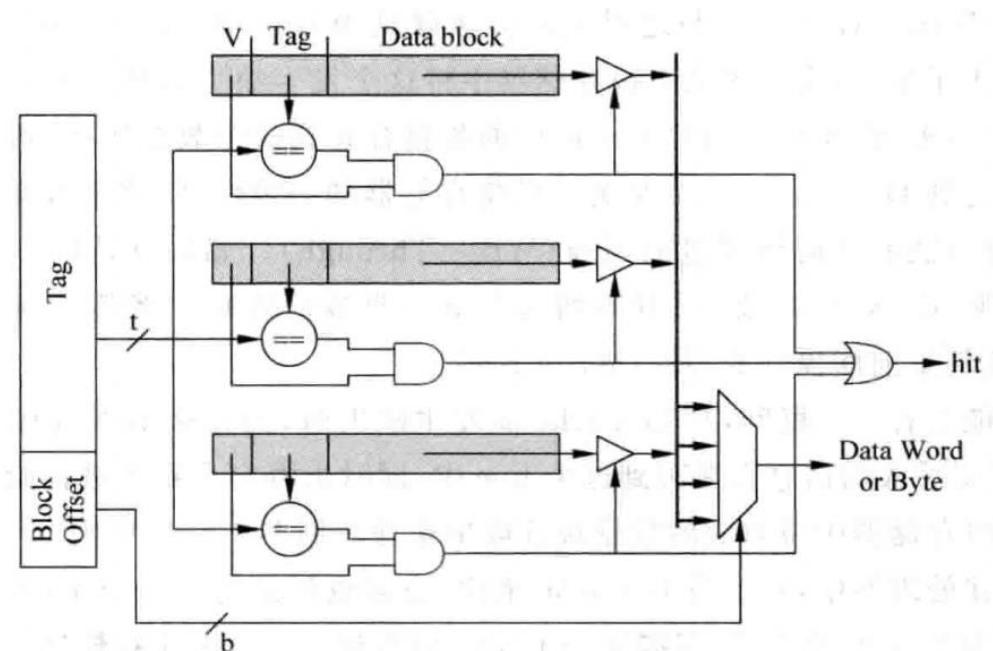
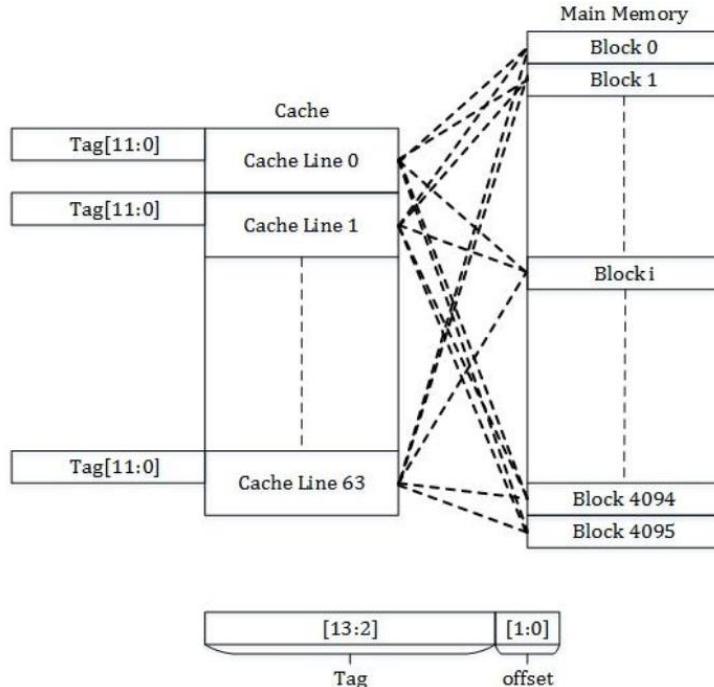


串行访问

# 2.2 Cache组织形式

## □ 全相连

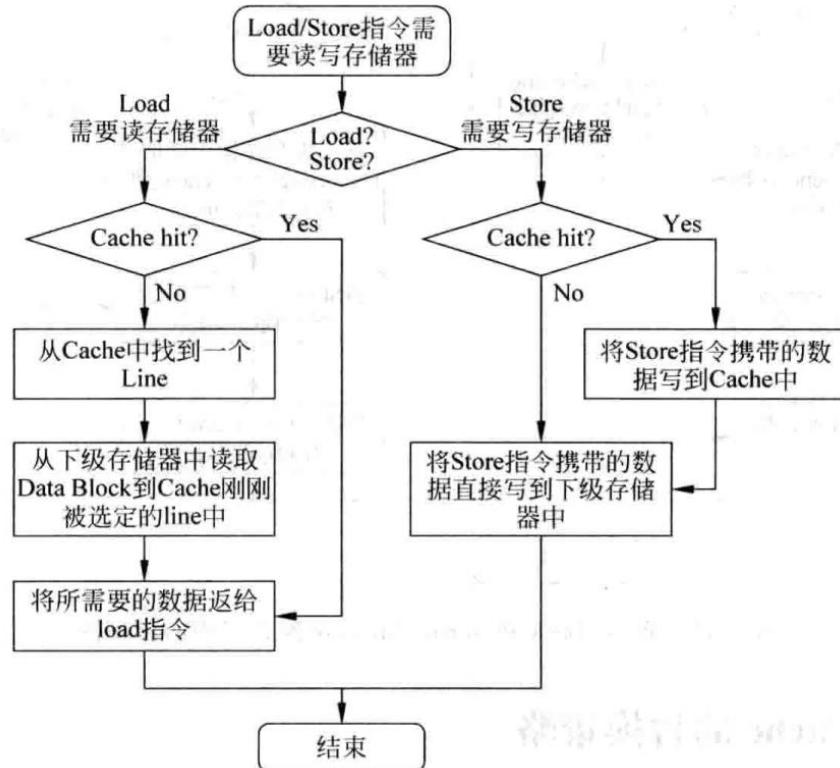
- 指令被划分为 Tag、Offset (无Index)
  - Tag 用于判断命中情况
  - Offset 用于定位字节
  - 缺失率最低，延迟最高



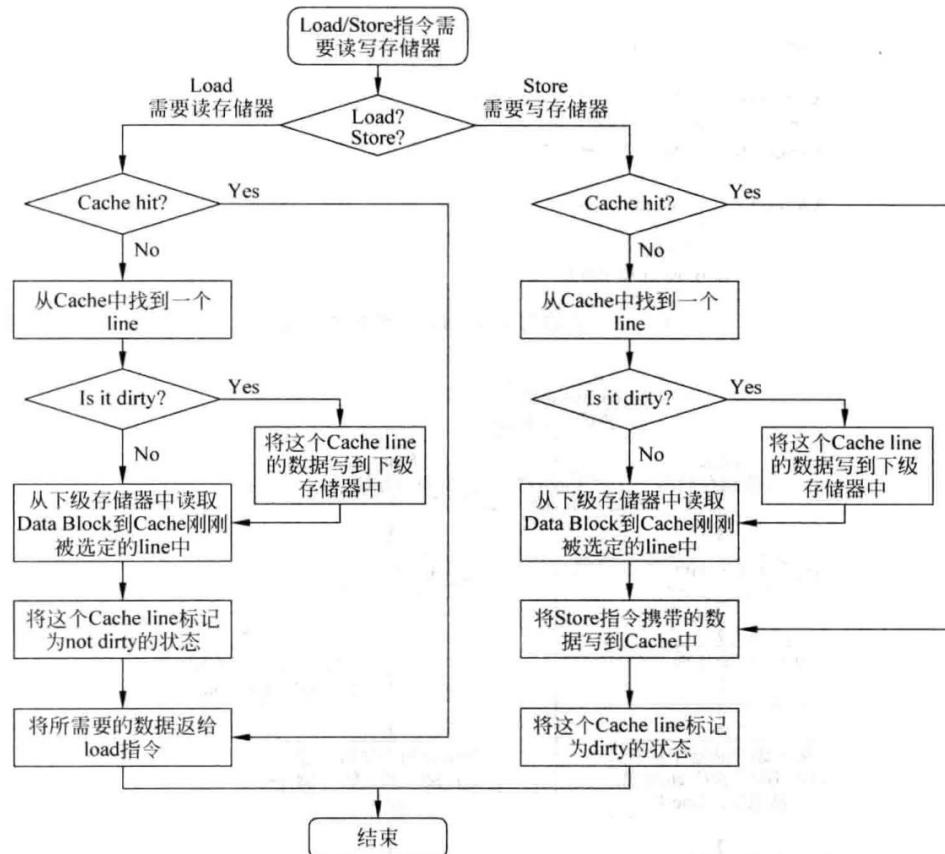
## 2.3 Cache写策略

为了保持Cache和下级存储器的一致性？需要设计写策略，包含：

- 写通（Write Through）：数据写到Cache的同时，更新到下级存储器
- 写回（Write Back）：数据写到Cache的同时，标记相应Cache行，替换时将脏行更新到下级存储器



写通

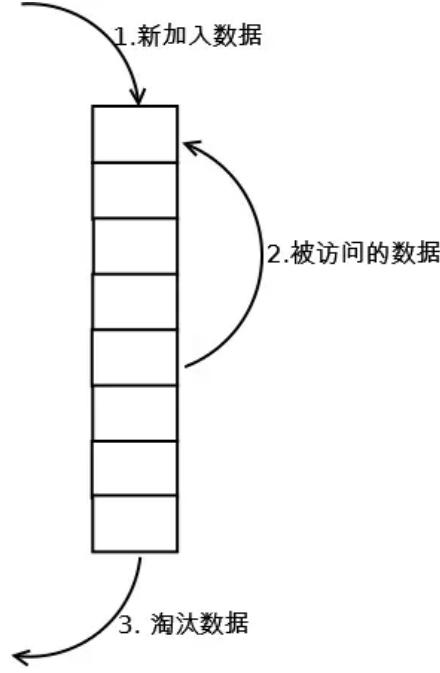


写回

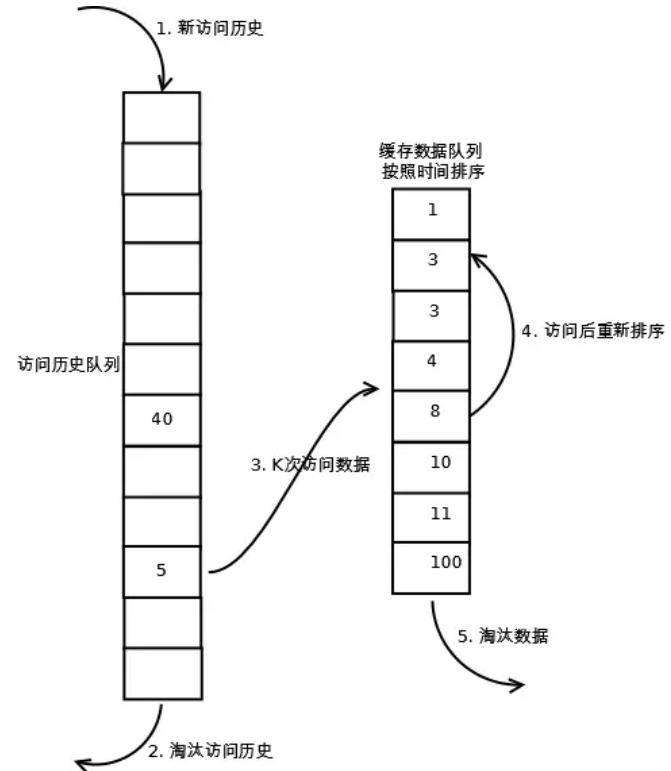
## 2.4 Cache替换策略

□ Cache缺失时，需要从对应的集合中找到可以替换的Cache行

- 最近最少使用法 (LRU) : 选择最近被使用次数最少的Cache行
- 最近最少使用K次法 (LRU-K) : 为了解决LRU缓存污染，将LRU最近使用1次的标准扩展为最近使用过K次



LRU算法

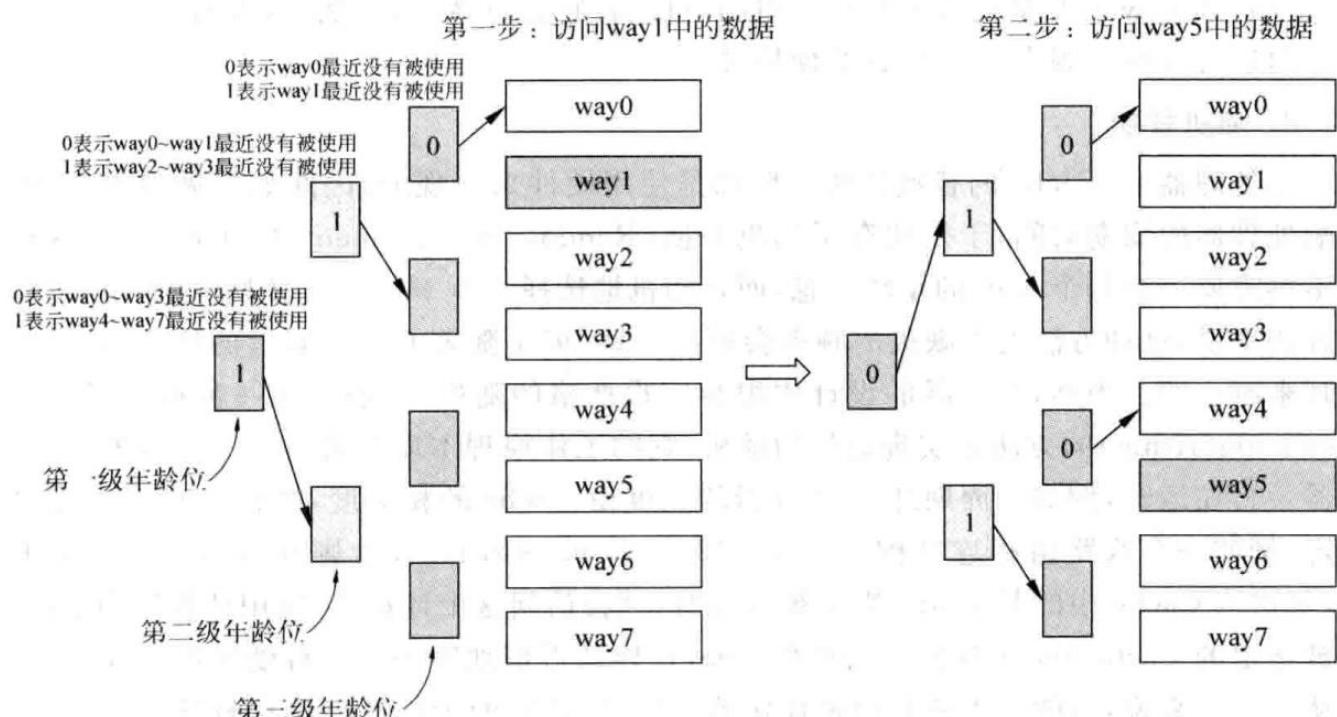


LRU-K算法

## 2.4 Cache替换策略

□ Cache缺失时，需要从对应的集合中找到可以替换的Cache行

- 先入先出法 (FIFO)：最先进入Cache的数据最先被替换出去
- 随机替换法：随机一路Cache行进行替换
- 伪最近最少使用法 (PLRU)：采用分级年龄位选择Cache行进行替换

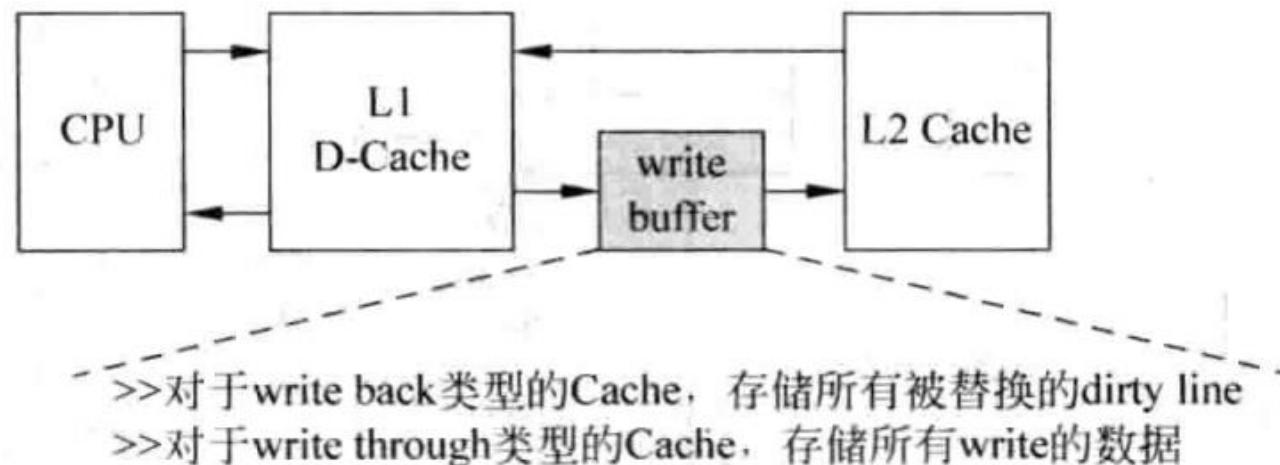


PLRU算法

# 2.5 Cache性能优化

## 口 写缓存

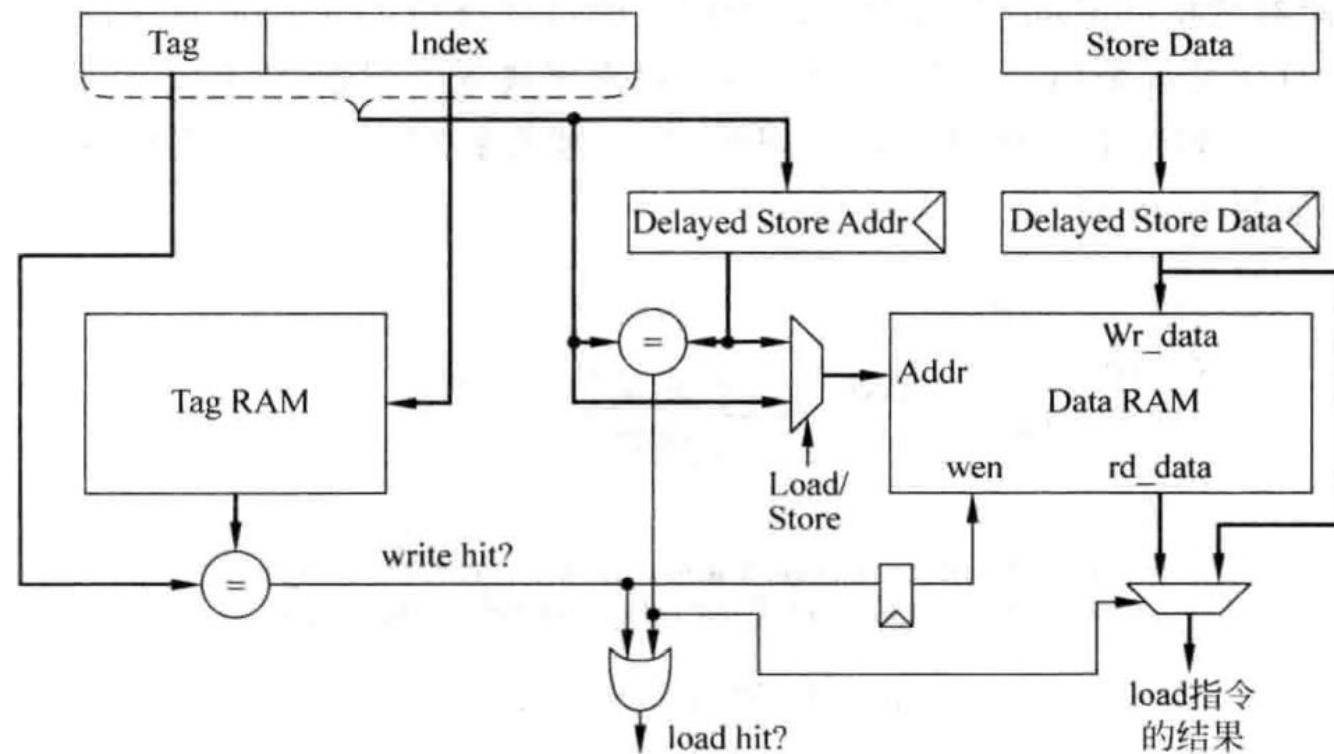
- 当Cache发生缺失时，需要选定Cache行进行替换，若选中的Cache行是脏状态，则需要先将数据先回下级存储器。为了缓存写回时间过长的问题，引入写缓存，等到下级存储器有空闲的时候，才会将写缓存中的数据写到下级存储器中。
- 对于写通类型的Cache，同样可以先将数据放到写缓存中。



# 2.5 Cache性能优化

## 口 流水线

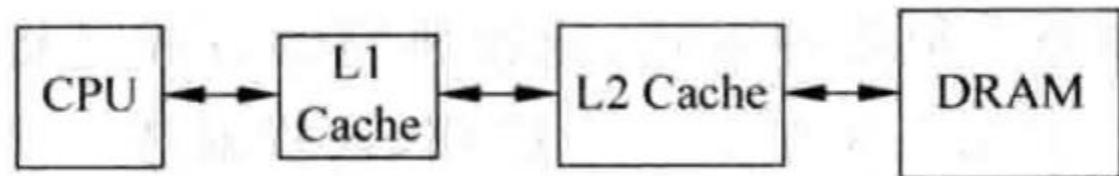
- 对于主频比较高的处理器，Cache访问难以在一个周期内完成，因此可以对Cache采用流水线架构。
- 典型方式是将Tag SRAM的读取和比较放在一个周期，写Data SRAM放在下一个周期。



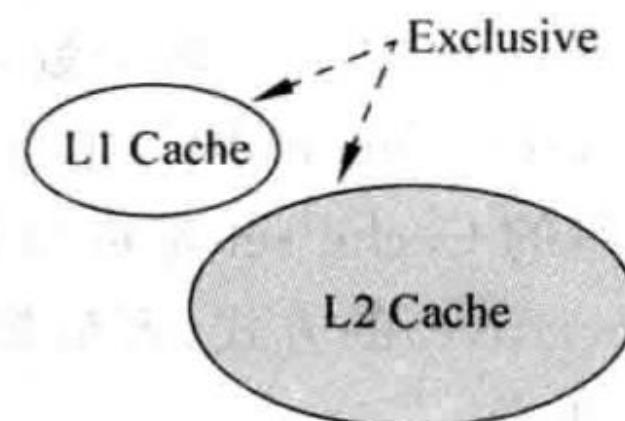
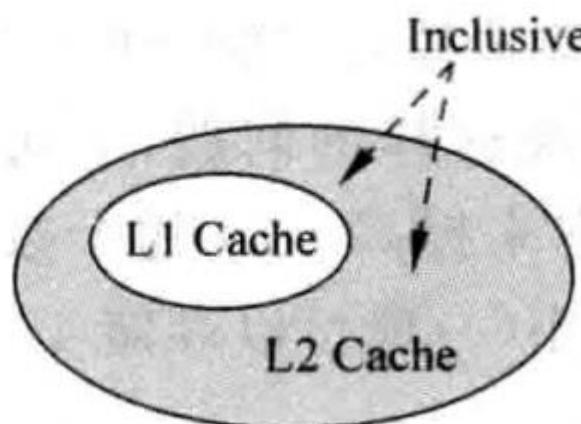
# 2.5 Cache性能优化

## 口 多级结构

- 现代处理器常采用多级架构。



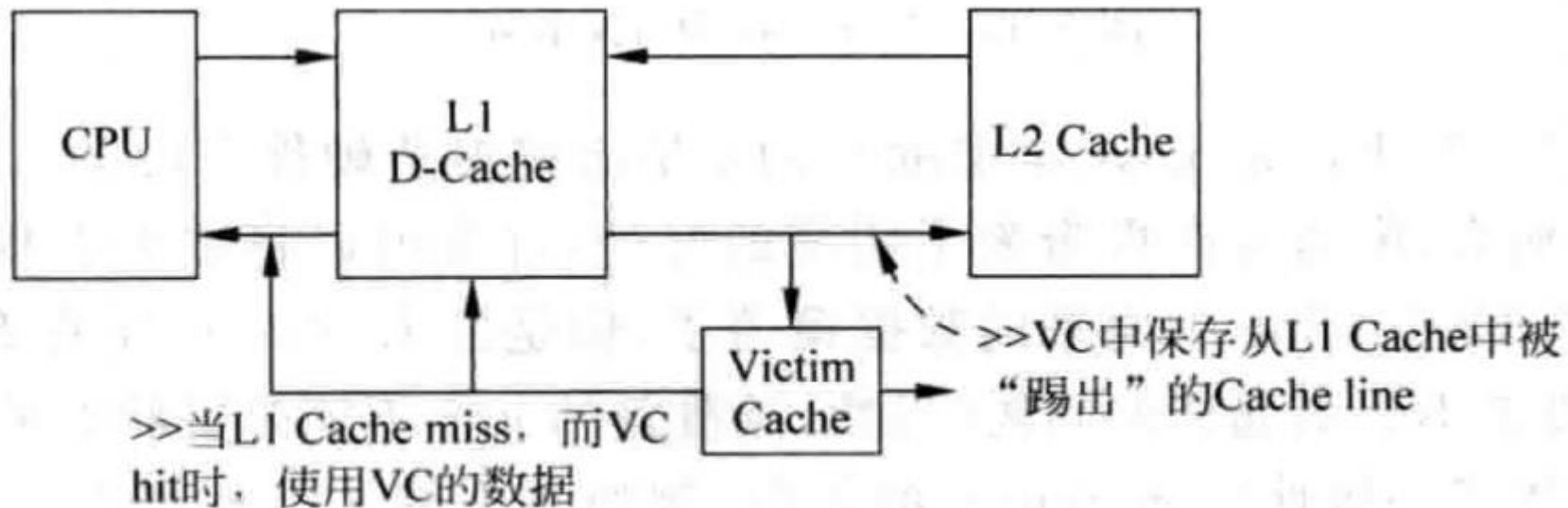
- 包容性多级结构 (Inclusive) : 如果L2 Cache包括了L1 Cache中的所有内容。
- 非包容性多级结构 (Exclusive) : 如果L2 Cache和L1 Cache的内容互不相同。



# 2.5 Cache性能优化

## □ Victim Cache

- Vimtim Cache可以保存最近被替换出Cache的数据
- Victim Cache的容量通常较小，采用全相连的方式

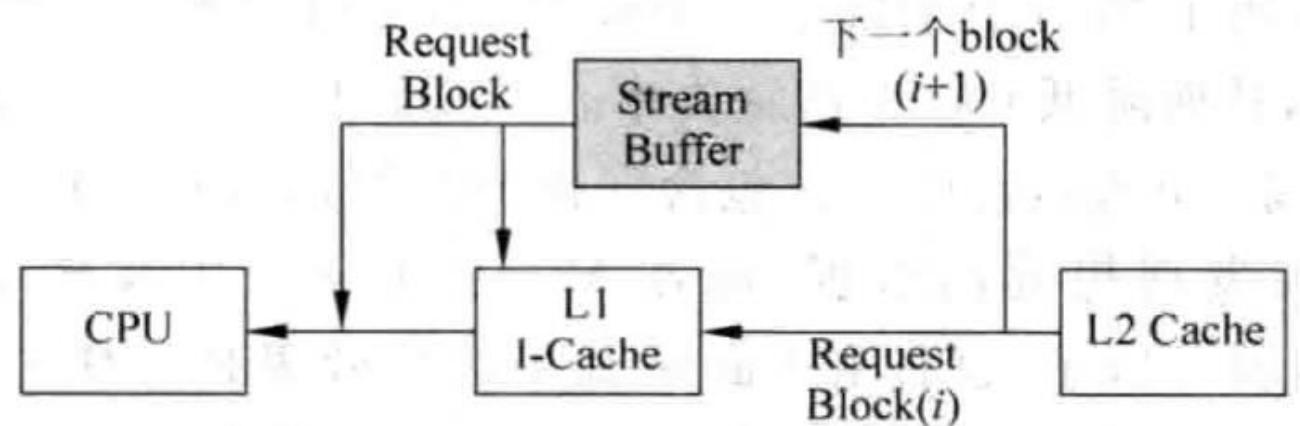


- Vimtim Cache的本质相当于增加了Cache的路数，能够避免多个数据竞争Cache中的有限位置，从而降低Cache的缺失率

# 2.5 Cache性能优化

## 口 预取

- 预取技术猜测处理器在以后可能使用什么指令或数据，然后提前将其放到Cache中，这个过程可以使用硬件完成，也可以使用软件完成
- 硬件预取：对于串行程序，当Cache发生缺失时，除了将需要的数据块从下级存储器取出来并放到Cache中，还会将下一个数据块也读取出来；非串行程序的预取可以相当困难



- 软件预取：在程序编译阶段对程序进行分析，由编译器插入预取指令。通常软件预取的Cache是非阻塞的，即预取时可以正常执行其他请求

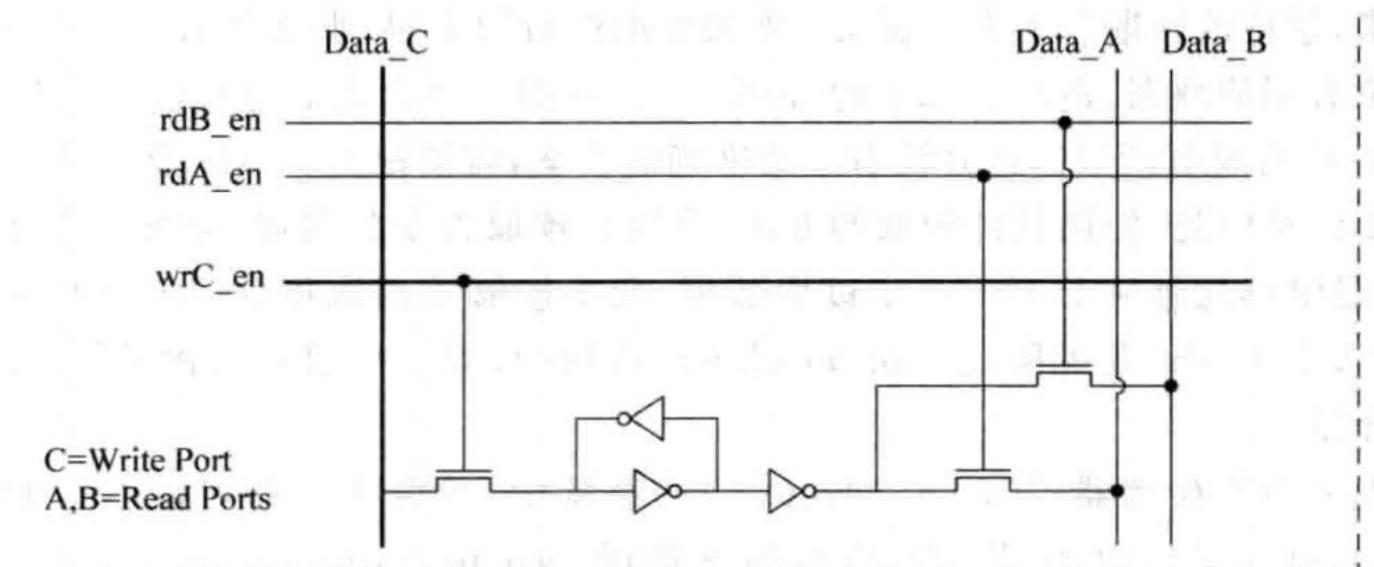
# 2.5 Cache性能优化

## □ 多端口Cache

- 在超标量处理器中，要求处理器能够每周期执行多条访存请求，这就需要多端口的Cache
- 多端口Cache主要采用多端口、多副本和多片技术实现

## □ 多端口 (True Multi-port)

- 通过多端口SRAM实现，所有在Cache中的控制通路和数据通路都需要复制
- 由于多端口的SRAM需要驱动多个读端口，访问时间更长



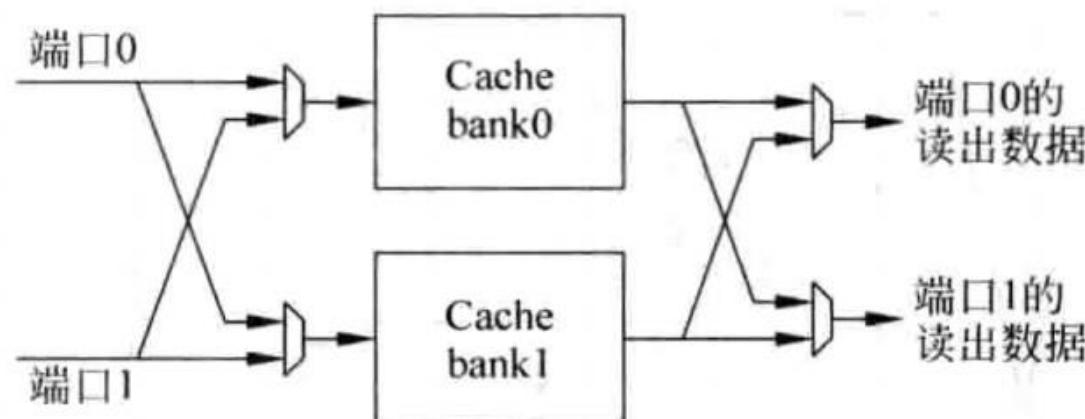
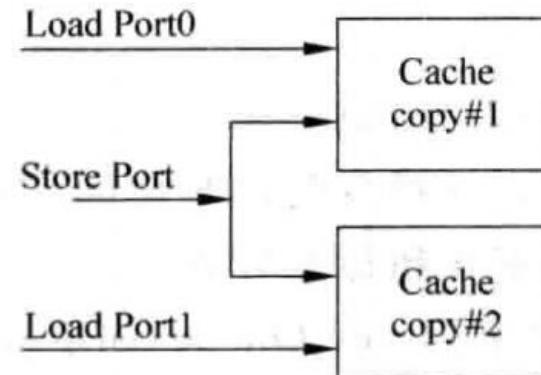
# 2.5 Cache性能优化

## □ 多副本 (Multi Cache Copies)

- 复制多份Tag SRAM和Data SRAM
- 需要保持Cache间同步，设计复杂，面积浪费

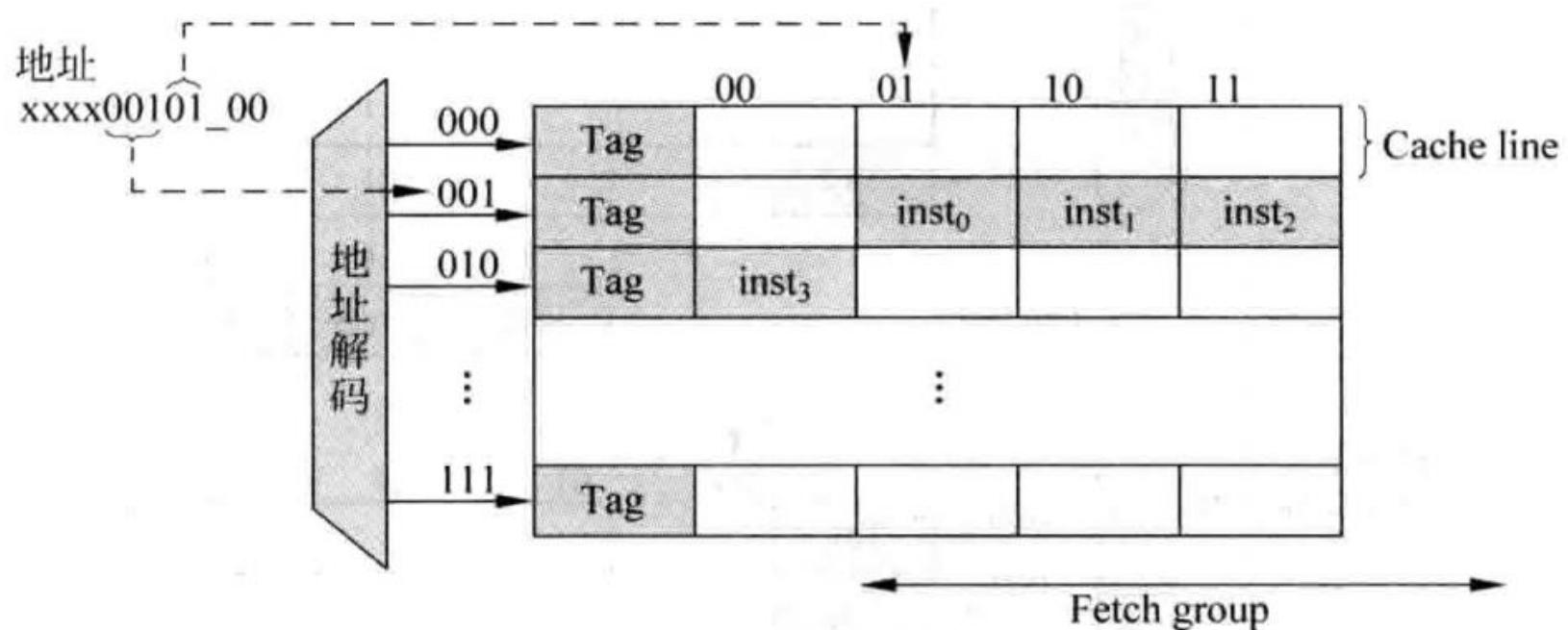
## □ 多片 (Multi-banking)

- 将Cache分为很多小的bank，每个bank都只有一个端口
- bank可能发生冲突
- 仅Tag SRAM需要多端口设计

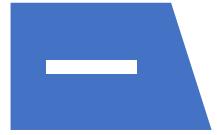


## 2.6 超标量处理器中的Cache

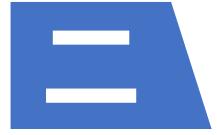
- 对于一个多发射的超标量处理器，其发出一个取指令的地址后，Cache至少需要送出发射数的指令
- 由于跳转指令的存在，处理器送出的取指令地址不一定总是字对齐的，可以将多余指令缓存



# Day4 目录



存储管理



Cache设计



实验



# 实验说明

## □ Cache设计

- 要求: 通过chiplab: func/func\_lab9、coremark/drystone性能测试
- 代码内容: Cache存储系统和地址转换机制
  - 数据缓存(DCache)控制逻辑
  - 直接映射窗口(DMW)地址转换
- 实验报告链接:
  - <https://bhpan.buaa.edu.cn/link/AA79AE16003B4A452C861C0CF1605716E3>
  - 文件夹名: Lab3
  - 报告提交期限: 2025-08-03 23:59
  - 提取码: uestc