**Cache技术**

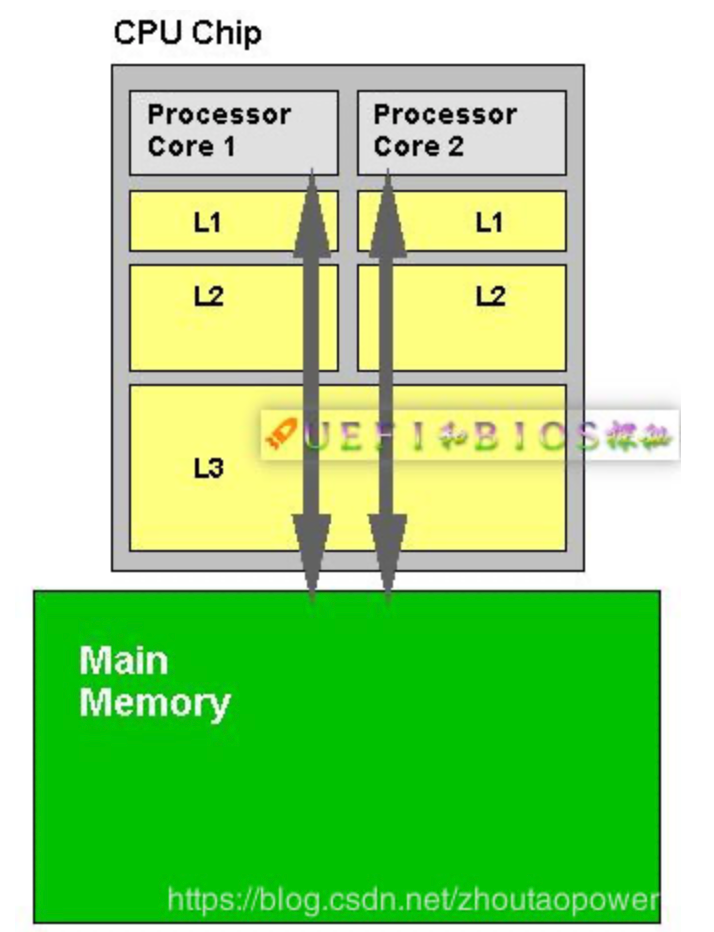
先看一下百度给出的定义：Cache技术是一种高速缓冲存储器，是为了解决CPU和主存之间速度不匹配而采用的一项重要技术。

**先思考这样的一个问题，为什么我们需要Cache技术呢？**

第一个原因，程序访问的局部性。这也是我们程序设计时会设计到的内容一般，任务的耗时点都在我们的循环内容上。

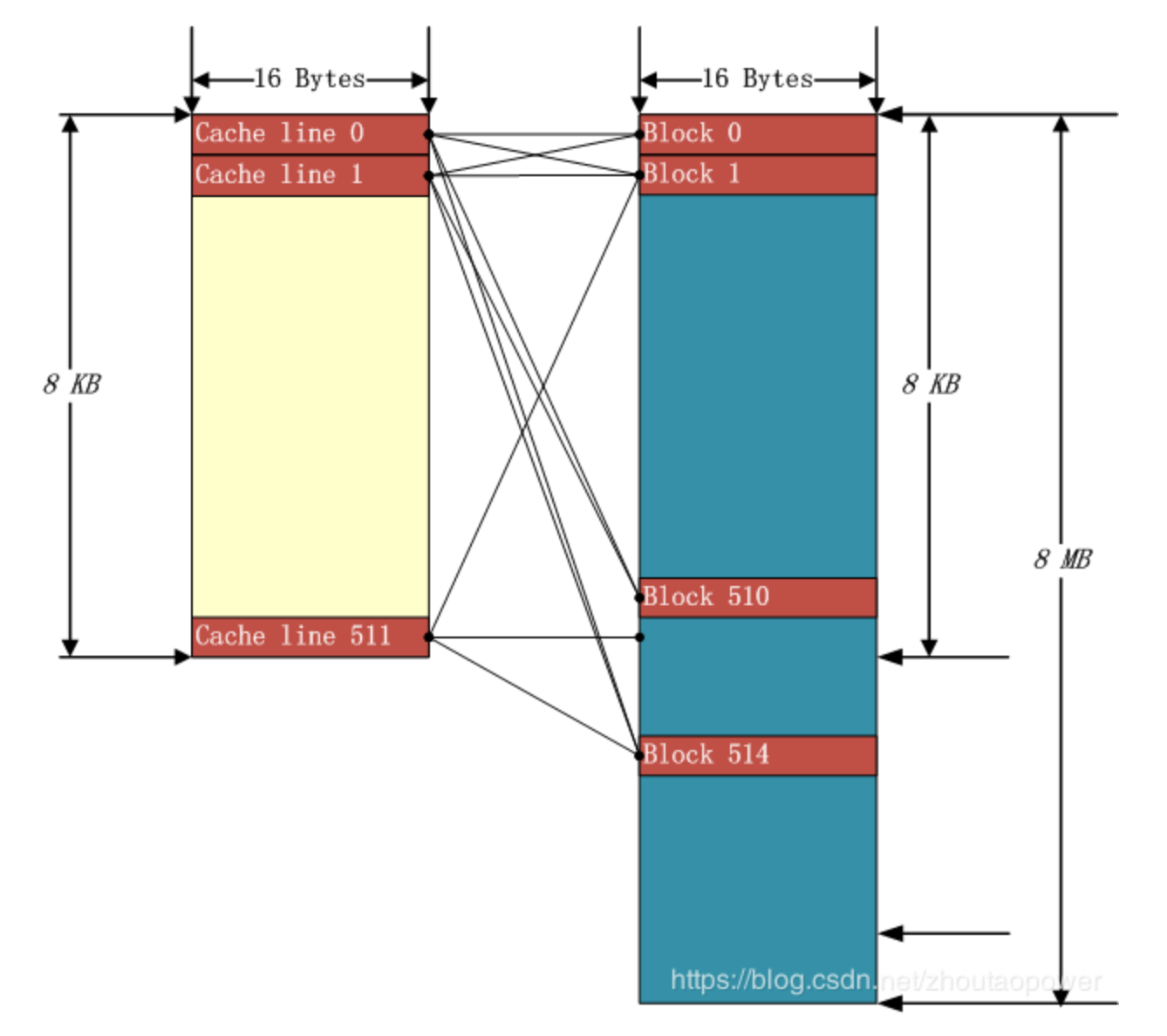
第二个原因，CPU的速度远远高于，甚至说甩飞内存，就好像高铁和摩托车的速度区别一样。当CPU直接从内存中存取数据时是需要一定的时间周期等待的，而Cache则可以保存CPU刚用过或循环使用的一部分数据，如果CPU需要再次使用该部分数据时就又可以从Cache中直接调用，这样就避免了重复存取数据，减少了CPU的等待时间，因而提高了系统的效率。

不过现在的计算机一般的Cache不会只有一层，可能会做到3-4层。

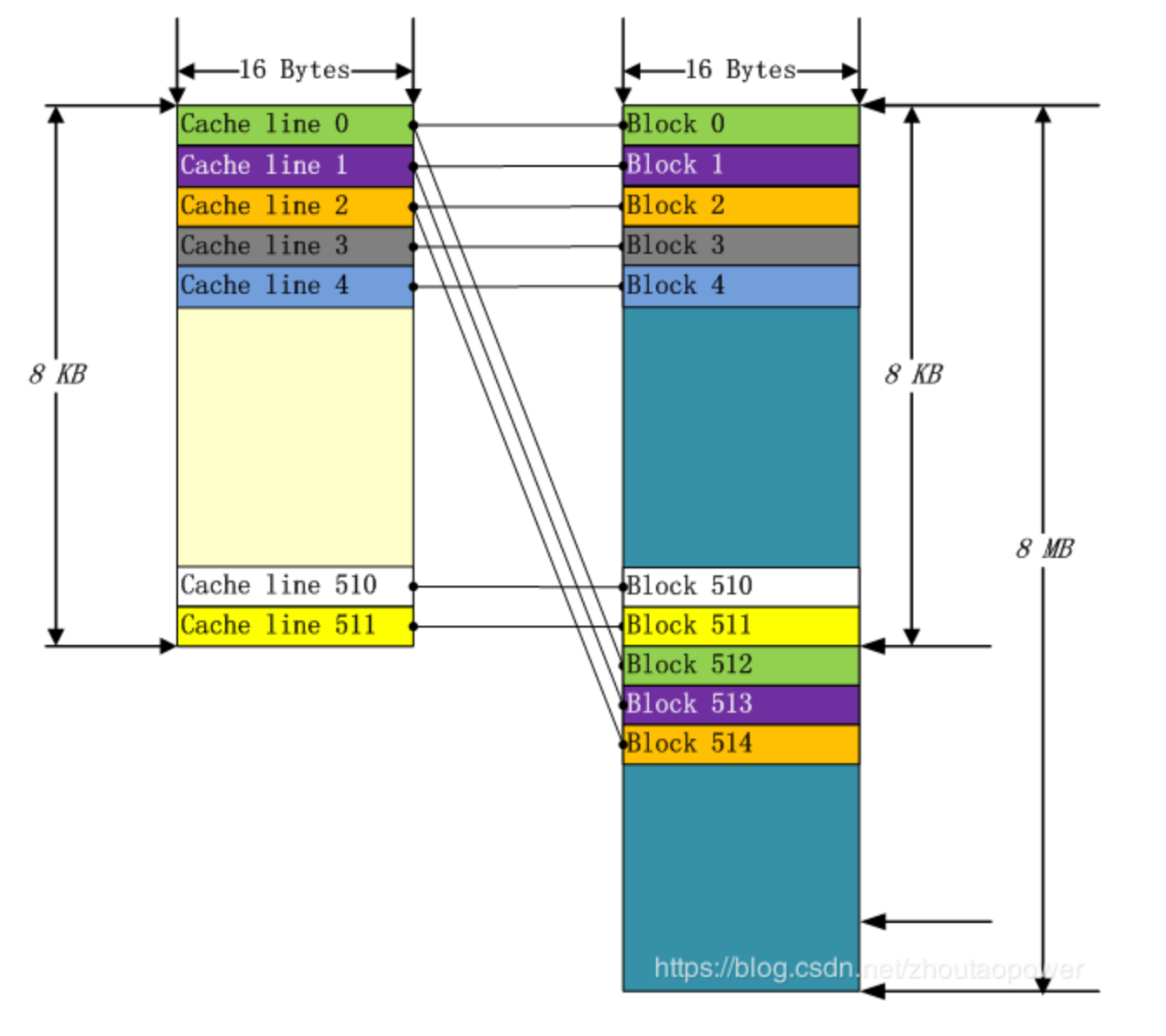


**第二个问题，他是如何进行设计的呢？**Cache通常由相联存储器实现。相联存储器的每一个存储块都具有额外的存储信息，称为标签。当访问相联存储器时，将地址和每一个标签同时进行比较，从而对标签相同的存储块进行访问。一般分成三种形式：

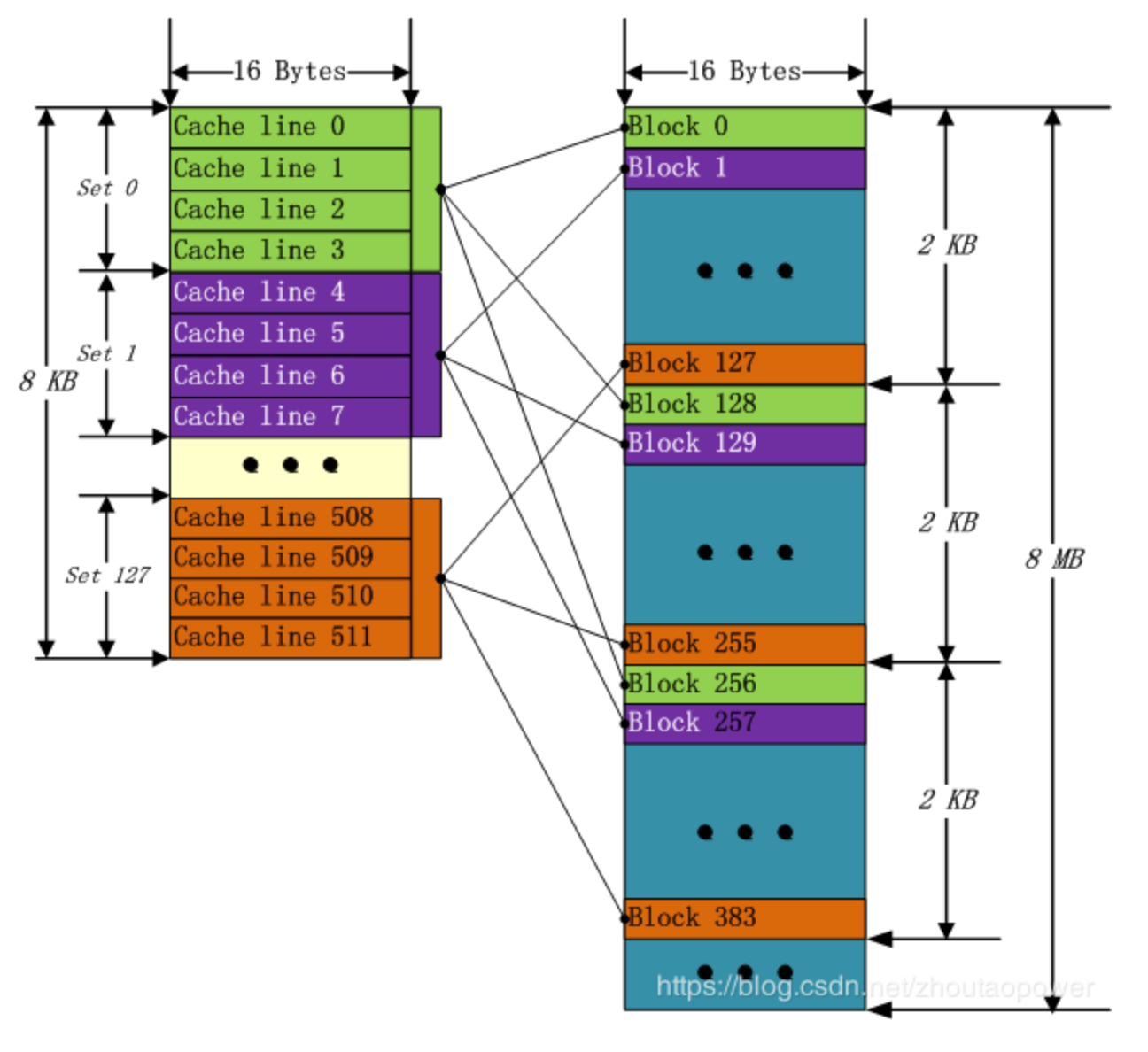
1. 全相联Cache：它能够在给定的时间内去存储主存器中的不同的块，命中率高，但是查找速度慢，因为他要进行全部的查询后才能完成任务。



1. 直接映像Cache：其实是类似于HashMap的一个构造形式了，基于地址直接找到。但是因为数据的相关性和局限性，同一时刻需要用的数据大部分都在附近，会造成Cache频繁换进换出，造成颠簸。

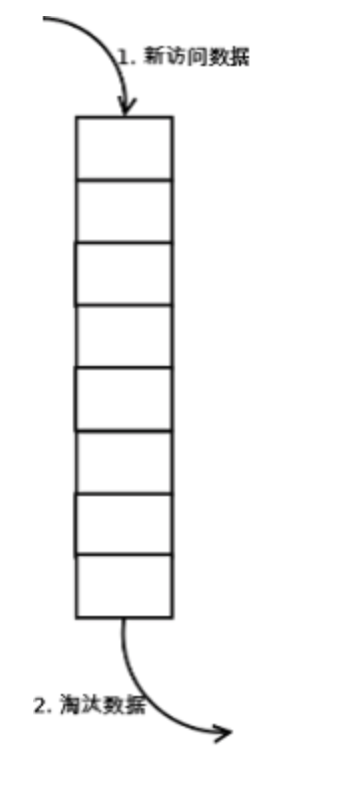


1. 组相联Cache：上述两种的这种方案，有点类似于jdk7的ConcurrentHashMap设计的分段锁形式，是为了效率和命中率的两者同时考虑

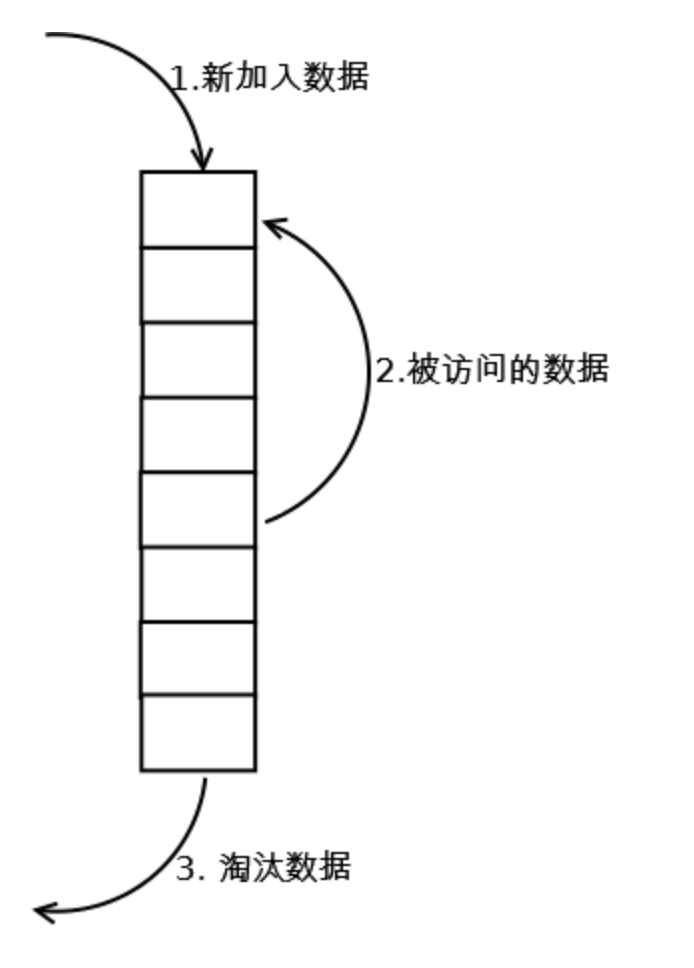


Cache的算法设计常见的有FIFO(先进先出)、LRU(最近最少使用)和LFU（最不经常使用)。

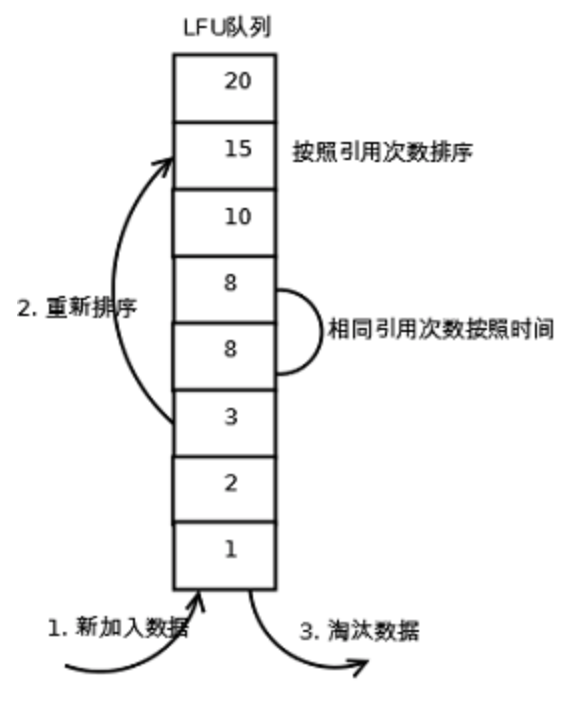
1. FIFO(先进先出)，这是一个最简单的实现方案，但是并不好使，因为并不会满足我们上述说过的程序访问的局部性。



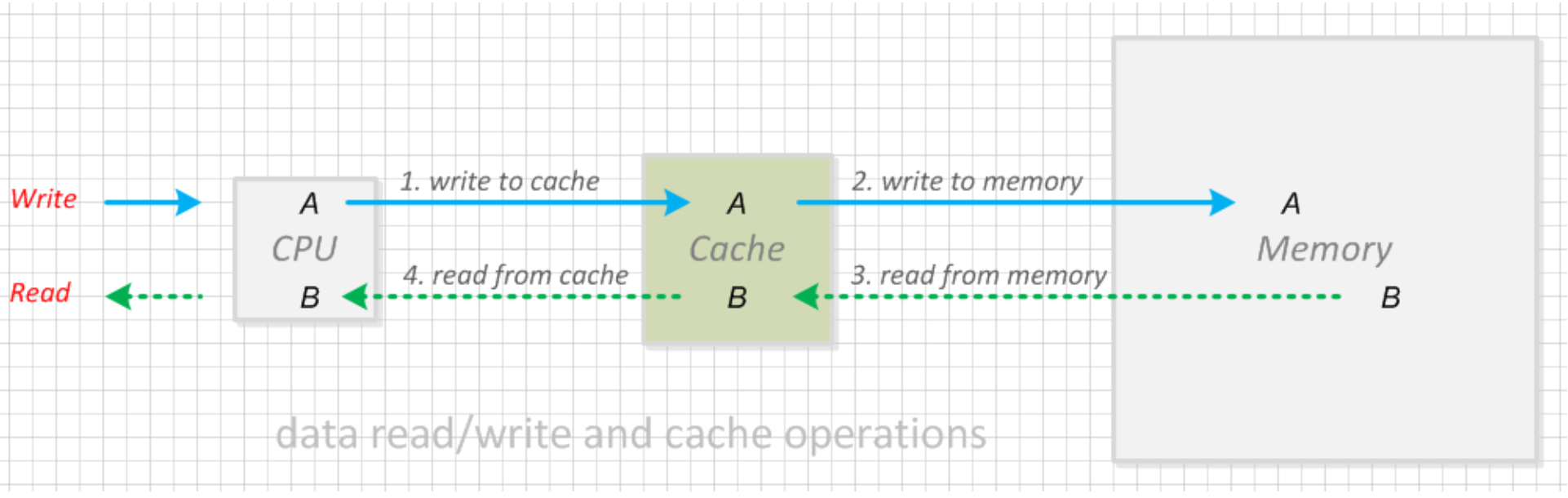
1. LRU(最近最少使用)，根据数据的历史访问记录来进行淘汰数据，其核心思想是“如果数据最近被访问过，那么将来被访问的几率也更高”。



1. LFU（最不经常使用)，根据数据的历史访问频率来淘汰数据，其核心思想是“如果数据过去被访问多次，那么将来被访问的频率也更高”。



**最后一个就是关于Cache最重要一个任务了 —— 读写。**

****

**从图上可以看出，**

写入数据时： 第一步，CPU将数据写入Cache； 第二步，将Cache数据传送到Memory中相应的位置；   
读取数据时： 第一步，将Memory中的数据传送到Cache中； 第二步，CPU从Cache中读取数据；

**在具体的硬件实现上，Cache有写操作有透写（Write-Through）和回写（Write-Back）两种方式：**

在透写式Cache中，CPU的数据总是写入到内存中，如果对应内存位置的数据在Cache中有一个备份，那么这个备份也要更新，保证内存和Cache中的数据永远同步。所以每次操作总会执行图一中的步骤1和2。

在回写式Cache中，把要写的数据只写到Cache中，并对Cache对应的位置做一个标记，只在必要的时候才会将数据更新到内存中。所以每次写操作都会执行步骤中的图1，但并不是每次执行步骤1后都执行步骤2操作。