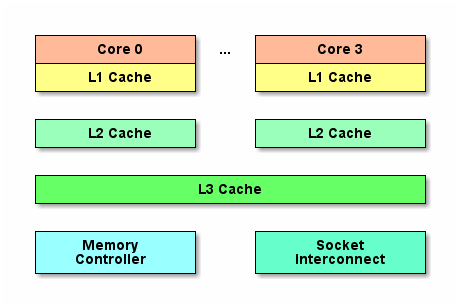
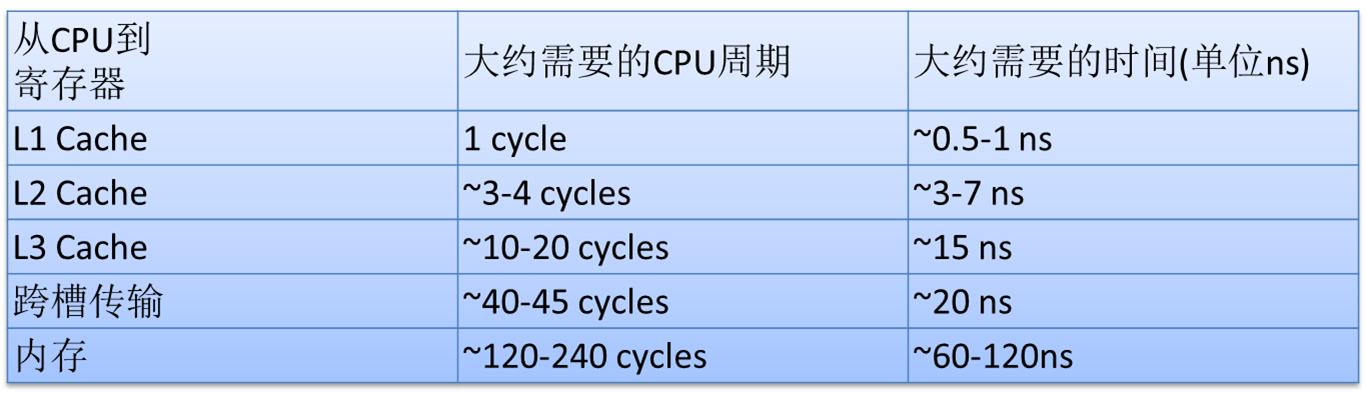
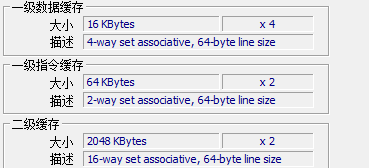
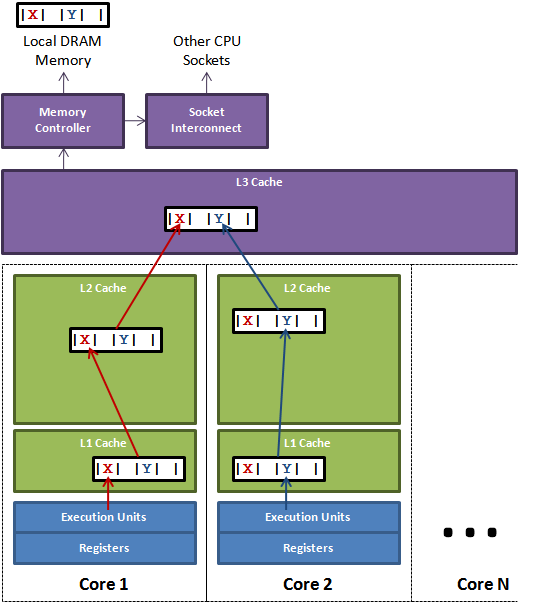
**CPU缓存结构：**

   
CPU为了加速访问，就像数据库和缓存关系相似，存在L1缓存，L2缓存，L3缓存来缓存内存中的数据。   
级别越小，CPU访问越快：   
   
\*\*缓存行：\*\*CPU缓存并不是将内存数据一个一个的缓存起来，而是每次从内存中取出一行内存，称为缓存行（Cache Line），对于我的电脑，缓存行长度是 64Bytes。对于Java，也就是说，假设X和Y对象，他们两个内存相邻，而且加起来的长度小于64 Bytes   
   
这里会涉及到缓存行失效的问题，如下图：   
   
假设有两个线程分别访问并修改X和Y这两个变量，X和Y恰好在同一个缓存行上，这两个线程分别在不同的CPU上执行。那么每个CPU分别更新好X和Y时将缓存行刷入内存时，发现有别的修改了各自缓存行内的数据，这时缓存行会失效，从L3中重新获取。这样的话，程序执行效率明显下降。为了减少这种情况的发生，其实就是避免X和Y在同一个缓存行中，可以主动添加一些无关变量将缓存行填充满，比如在X对象中添加一些变量，让它有64 Byte那么大，正好占满一个缓存行。

**Java对象占用内存：**

对于一个Java对象，按顺序排列内存中元素：

* 对象头，保存一些锁的信息和其他对象信息，64位Hotspot JVM占用12 Bytes，如果是数组的话还有4Bytes的长度位
* 基本类型占用：Java8种基本类型的内存，按下面的顺序排列   
  + long，double：8字节
  + int，float：4字节
  + char，short：2字节
  + boolean，byte：1字节
* 引用类型占用：4字节
* 补齐：最后整个长度必须为8的倍数，不足补位，如果补齐长度是4字节，而且对象头是12字节，就在对象头后补齐4字节。

最后我们看内存中Sequence的状态，我们这里只关心变量部分，常量部分不考虑。Sequence的核心就是value这个值，我们要并发的修改不同Sequence的value的值，我们要保证每个缓存行里面只有一个Sequence的value。一般现在的电脑CPU缓存行是64字节，如下图所示，即使最极端的情况下（Cache line开头或者结尾是value），也能保证每个CacheLine中只有一个Disruptor中的Sequence，这样我们能明显减少false sharing带来的性能损耗   
