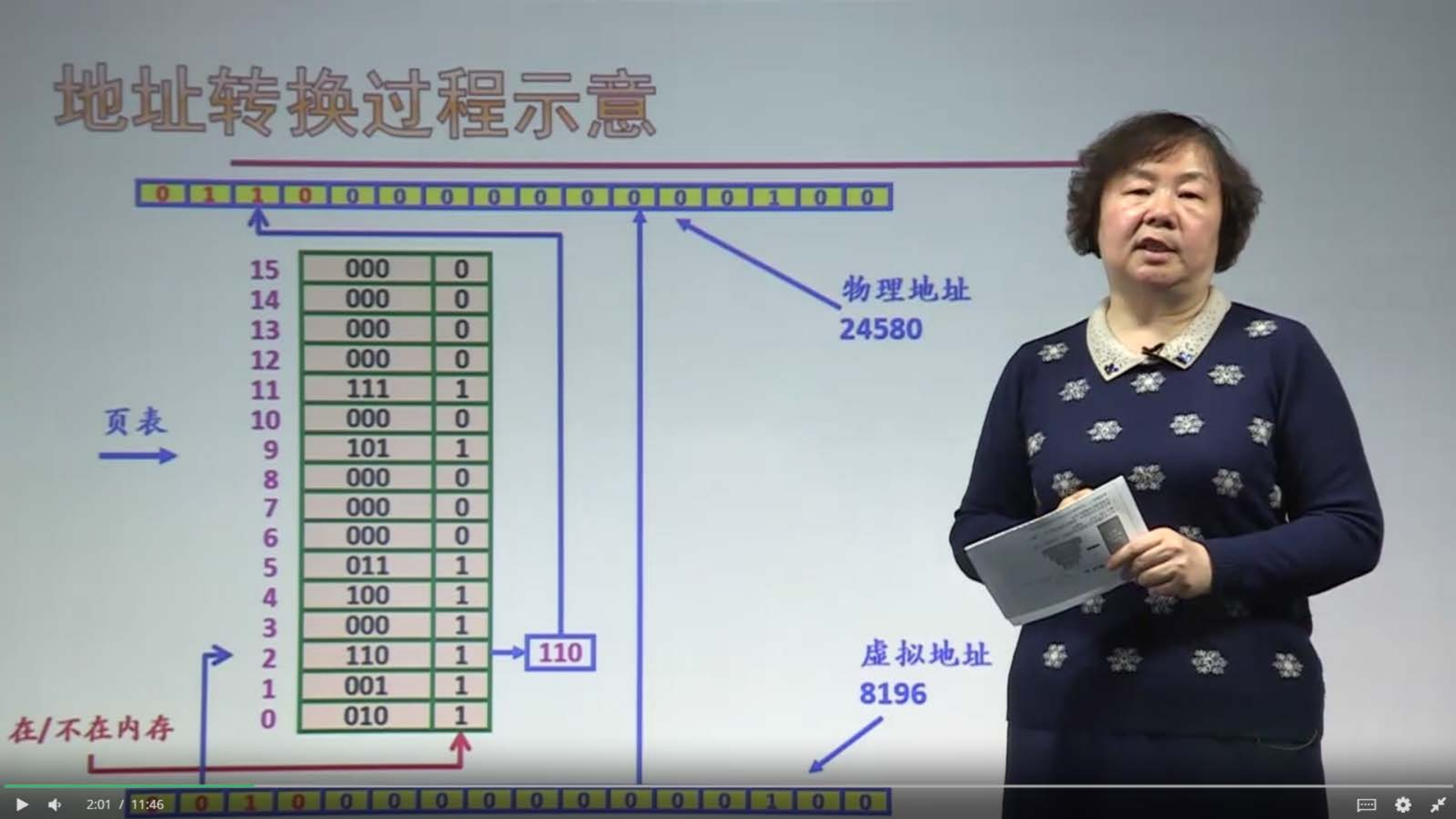
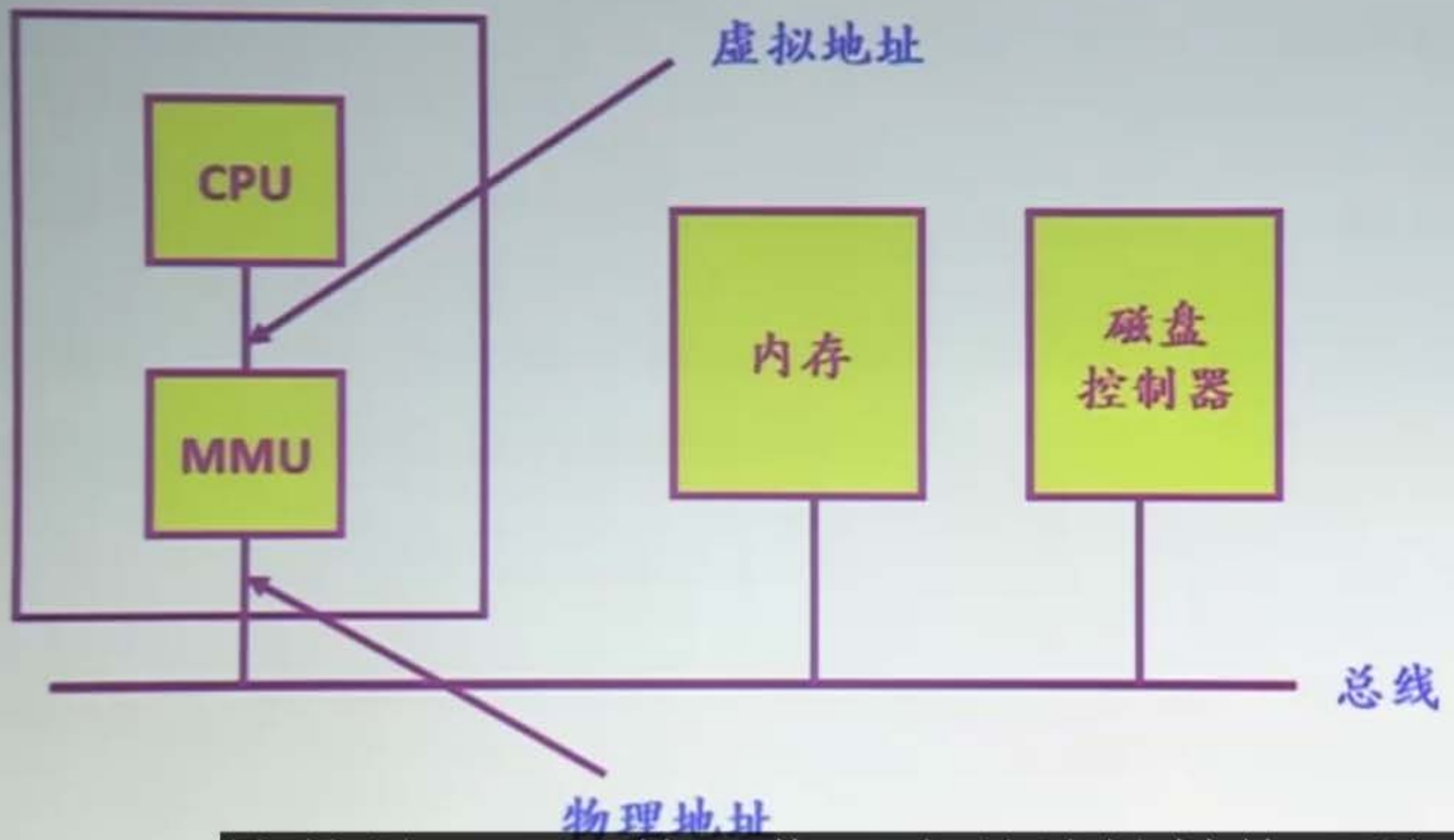
MMU、快表(TLB)

地址转换过程及TLB引入

地址转换过程示意

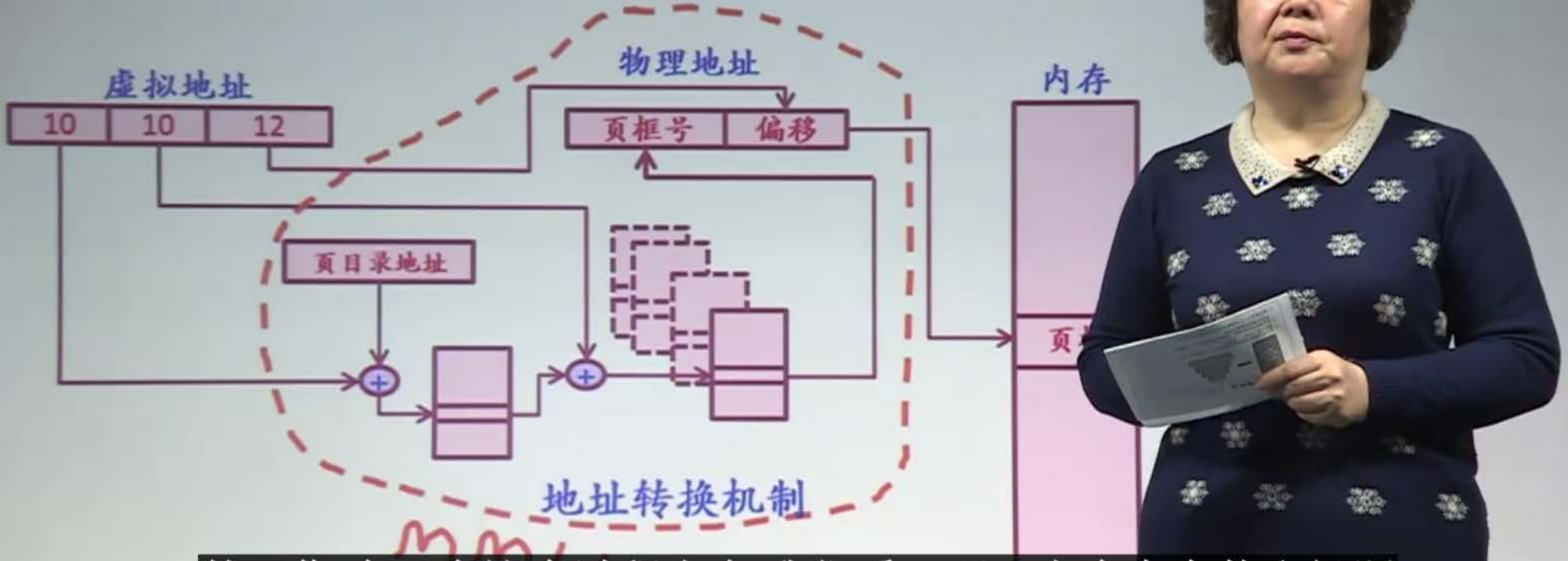


MMU：内存管理单元



这就是 MMU 的工作，也是地址转换啊是它的一个主要工作

地址转换(映射)



的工作 在，在这个过程当中 我们看一下，它会存在什么问题

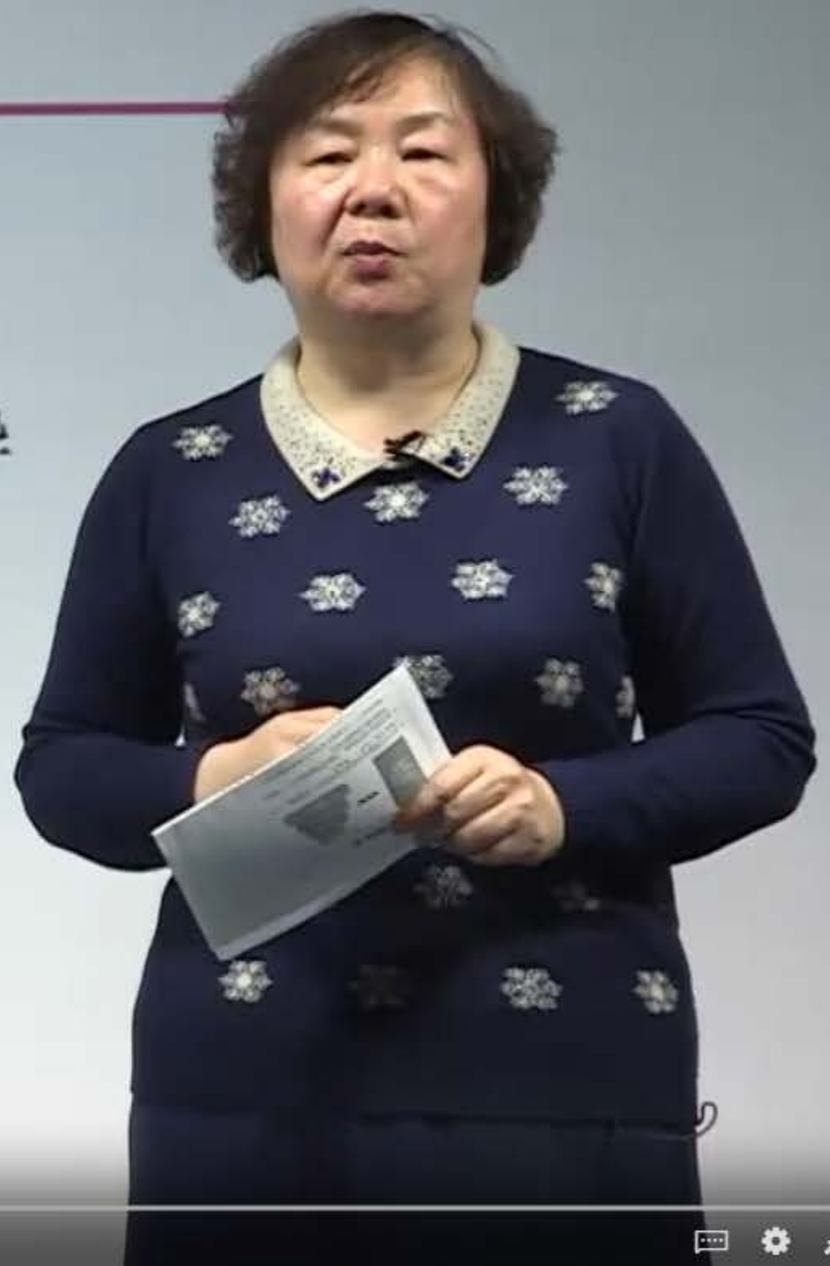
快表(TLB)的引入

问题

- 页表 → 两次或两次以上的内存访问
- CPU的指令处理速度与内存指令的访问速度差异大，CPU的速度得不到充分利用

如何加快地址映射速度，以改善系统性能？

程序访问的局部性原理 → 引入快表(TLB)



快表是什么？

快表的大小、位置

快表的置换问题?

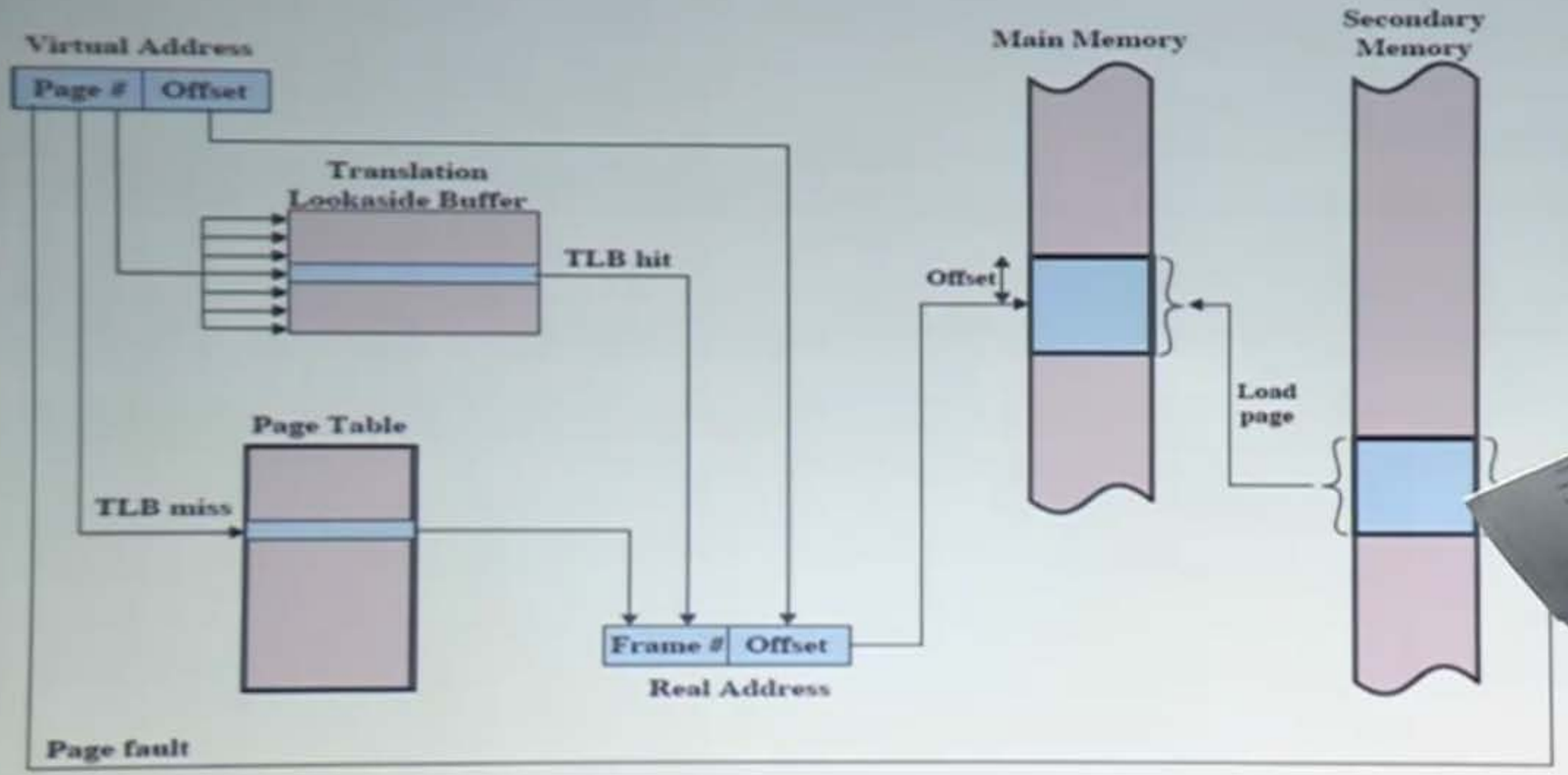
- TLB — Translation Look-aside Buffers

在CPU中引入的高速缓存（Cache），可以匹配CPU的处理速率和内存的访问速度

一种随机存取型存储器，除连线寻址机制外，还有接线逻辑，能按特定的匹配标志在一个存储周期内对所有的字同时进行比较

- 相联存储器（associative memory）
  特点：按内容并行查找
- 保存正在运行进程的页表的子集(部分页表项)

加入TLB后地址转换过程示意



选自William Stallings

0:00 下面我们介绍一下，虚拟页式存储管理方案当中，地址转换的过程以及 TLB 引入的问题。 我们首先回顾一下，地址转换过程 这里有一个虚拟地址 那么虚拟地址当中，我们可以看到 我们后面 12 位是页内偏移前面呢，我们只写了 4 位，啊，我们只列出来 4 位 那么要把虚拟地址转换成物理地址，需要页表的支持我们来看一下，这是啊，这个进程当前的一个部分的页表项 那么这个页表项当中呢，主要记录了页框号和有效位。 那么实际上我们知道 其它还有一些位，这里头就省略没有画出来 那么，这个有效位啊 0 就表示这个页框啊，这个页面还没有调入内存 那这个页框号是无效的。 如果是 1 就表示这个页面已经调入内存了，这个页框号就是它在 物理内存中的啊，位置。 好，那么我们来看一下 在进行地址转换过程当中，我们需要把虚页号取出来，当然啦，这个 取出来是由硬件啊自动地取出的，那么就是这里头的 0010 ，那么 0010 呢，实际上是虚页号是 2 啊，我们来看 2 对应的页表项，这页表项当中呢， 1 表示它已经啊 进入内存了，那么进入内存的页框号呢是 110 ，所以呢我们来 得到了 110 ，然后这个 110 呢实际上就和原来的 页内的偏移，页内地址拼接成了一个新的地址，这就是我们最终的 物理地址，那么这个过程 是由谁来完成的呢？首先，这个过程完成的部件呢是叫 MMU 叫内存管理单元。 那么它是一个硬件的部件 内存管理单元的作用是，将虚拟地址转换成物理地址 CPU 取到的呢是一个虚拟地址，然后它交给了 MMU MMU 呢做相应的工作，比如说查页表啊，先把虚拟地址 页号取出来，划分成不同的几段。 然后呢去，分别去查对应的页表 然后呢，得到了页框号，再去拼接成物理地址 那么，在这个中间过程中呢，还要去检查页表项 当中的一些位啊，是不是满足条件？ 所以， MMU 做了这些工作之后呢，最终啊，得到了物理地址，然后呢，再用这个物理地址呢， CPU 再去到 内存或者什么地去取相应的数据啊。这就是 MMU 的工作，也是地址转换啊是它的一个主要工作 我们再把这个图看一遍，MMU 的位置呢就是 说在这个地方，也就是这里头都是 MMU 要做的事情 那么，前面是虚拟地址，得到的是物理地址，中间的这个过程都是 MMU 的工作 在，在这个过程当中 我们看一下，它会存在什么问题 你要进行地址转换就要访问页表 那么，如果页表是 2 级页表，就要访问啊两次内存，如果是 4 级页表，就要访问四次内存 所以，我们可以看一下，1，访问 页表至少是两次或者两次以上的内存访问，因此， 当一个非常频繁的啊，因为我们知道这个 CPU 执行指令啊，要频繁地进行这项工作。 那么几百万条指令的执行都要 double 或者是好多倍的内存访问 这样的话呢， CPU 的指令的处理速度 由于和内存的指令的这种访问的

速度差异还是很大的。 因此，CPU 的这个 性能就得不到提升， CPU 的速度其实上得不到充分地利用 所以，这是如果引入了，如果有页表 的存在，那么我们要访问页表就会带来这样一个问题 那么怎么解决这个问题呢？也就是说，怎么样来加快 这样一个地址转换的过程，地址映射的速度呢？通过 这样一个速度的提升，实际上对整个系统的性能是得到一个改善的 那么，我们就要利用这样一个原理，也就是程序访问的局部性原理 那么这就相当于我们的一个公理一样，是吧？就是说，程序在执行的 时候，进程在执行的时候，它会呈现出一个高度的局部性，它总是集中 在一段时间里头，总是集中访问一些页面 那么，由于程序访问的局部性原理，所以，我们引入了一种新的啊 数据结构或者新的一个部件，叫做快表，强调它是快 那么快表的这个啊，英文呢实际上就是 TLB，所以我们经常说快表 那就中文一般说快表，英文就是 TLB，好，我们来看看 快表是什么？那么快表的英文呢是 Translation Look-aside Buffers，那么它实际上是一个啊缓冲区啊，缓冲区，可以放 一些内容的一个存储区域，那么它的特点是什么呢？ 这个存储区域呢，它的 组成呢实际上是高速缓存，是通过一种 高速缓存，或者我们就说的是一种 Srun [iii]，这样一种机制来构造的这个快表 那么这个速度是比较快的，它可以匹配 CPU 的处理速度 也可以匹配 CPU 的处理速度和内存的这样一个速度的一个矛盾可以解决 那么快表的具体的组织呢是这样的，那么它是由随机存取型 存储器组成，那么把除了连线的这种寻址机制之外，其实最重要的 还有一个接线的逻辑。 这个接线的逻辑可以让 这个快表能够按特定的匹配算法，在一个 存储周期内把所有的字进行同时的比较 就是我在这个快表里，查找一个数值，那么我可以 同时对快表的所有的单元同时查找，因为我有这样一个接线逻辑 而且由于快，一个存储周期就可以把相应的内容进行比较了 这是快表的一个构成。 那么快表呢，通常我们称之为相联存储器 那么它的特点就是刚才我们已经说过的，它是按内容并行查找 我们要找一个序列号是不是在快表里头，我们可以在一 个存储周期内同时进行比较，得到结论在还是不在 但是呢快表呢，大家可以看到它是由 cache 这部分组成 因此它的成本比较高，因此快表的大小容量是有限的 通常呢由于快表的成本啊比较大，所以我们只能够用一个比较小的 存储空间啊来做快表，那么它保存的内容呢是正在运行进程的 它的页表的一个子集，也就是一部分的页表项 那么快表一般的有多大呢？ 那么假设我们按啊这个页表项来算，那么可以快表的这个存，存放 的内容大致上可以是 64 个，放 64 项，或者是

128 项 那么也有啊分成不同的层次，就像高速缓存一样，可以分成几级高速缓存 所以快表有的时候也可以分成几级，那么每一级的大小又不太一样。 最大可以到，比如说，512 啊，512，那么它的位置呢实际上呢是在 CPU 的这个片上 那么 CPU、 MMU 和快表，就和这个 TLB 实际上都是在一个 啊片子上。 那么快表由于它的容量有限，因此呢它放的内容不多 所以在运行一段时间之后，那么快表内容就会满了，如果要想进入新的啊 页表项呢，就需要有一个替换的问题，置换的问题 那么这个呢后面我们还会介绍相应的算法，这里头我们只是把问题提出来 所以快表引入就是为了能够加快地址转换的速度 因此我们来看一下，当引入了快表之后 地址转换过程应该是什么样子的？那么这个过程当中呢我们 用这张图简化了一下，因为我们的页表就 做了一个一级的页表，就不要做，再做成二级或者是三级四级了 我们看一下，首先这里有一个虚拟地址啊，虚拟地址，那么虚拟地址呢那么 硬件或者是 MMU 就把虚拟地址给它分成啊几部分，两部分 吧，比如说在这里，那么一个是虚页号，一个呢是偏移 那么用虚页号呢，首先第一件事是先查 快表 TLB，我们可以看到这张图里头示意出是 并行地去进行比较啊确定 你要访问的这个页表项是不是在 快表里头，当然有两种结果，一种就是命中 叫做 TLB hit，就是命中了。 如果命中了，那我就可以 得到了这个虚页号所对应的页框号，我就可以直接拼接出 物理地址了。 当然也可能会出现没有命中的情况，比如说如果没有命中，那么下面 这个 MMU 继续用这个虚页号去查页表 啊查页表，这就是 TLB miss 的时候就去查页表 那么查了页表，找到了对应的页表项，我们还要看它的这个有效位是不是 0 或还是 1 如果是 1，那么说明这个页框啊内容已经是这个读入内存的内容了，那么这个时候 就需要从页表项当中得到了页框号，也是可以形成物理地址 那么也可能这个有效位是 0，表示这个页面还没有读入 内存，这个时候就要产生一个啊异常，叫做 Page fault 异常 那么 page fault 异常就是转入了操作系统，操作系统就会 做相应的工作。 比如说，到磁盘上把相应的页面 先调入内存，就得到了页框号，再去把页表填好，然后再重新再进行地址转换 啊，得到了相应的物理地址，来去访问内存 所以这就是加入了快表之后的地址转换的一个过程的一个示意