物理地址就是真实的地址嘛，这种寻址方式很容易破坏操作系统，而且使得操作系统中同时运行两个或以上的程序几乎是不可能的（此处可以举个例子，第一个程序给物理内存地址赋值 10，第二个程序也同样给这个地址赋值为 100，那么第二个程序的赋值会覆盖掉第一个程序所赋的值，这会造成两个程序同时崩溃）。

当然，也不是完全不可能，有一种方式可以实现比较粗糙的并发

就是说，我们将空闲的进程存储在磁盘上，这样当它们不运行时就不会占用内存，当进程需要运行的时候再从磁盘上转到内存上来，不过很显然这种方式比较浪费时间。

于是，我们考虑，把所有进程对应的内存一直留在物理内存中，给每个进程分别划分各自的区域，这样，发生上下文切换的时候就切换到特定的区域

那问题还是很明显的，就是仍然没法避免破坏操作系统，因为各个进程之间可以随意读取、写入内容。

所以，我们需要一种机制对每个进程使用的地址进行保护，因此操作系统创造了一个新的内存模型，那就是虚拟地址空间

就是说，每个进程都拥有一个自己的虚拟地址空间，并且独立于其他进程的地址空间，然后每个进程包含的栈、堆、代码段这些都会从这个地址空间中被分配一个地址，这个地址就被称为虚拟地址。底层指令写入的地址也是虚拟地址。

有了虚拟地址空间后，CPU 就可以通过虚拟地址转换成物理地址这样一个过程，来间接访问物理内存了。

地址转换需要两个东西，一个是 CPU 上的内存管理单元 MMU，另一个是内存中的页表，页表中存的虚拟地址到物理地址的映射

但是呢，每次访问内存，都需要进行虚拟地址到物理地址的转换，对吧，这样的话，页表就会被频繁地访问，而页表又是存在于内存中的。所以说，访问页表（内存）次数太多导致其成为了操作系统地一个性能瓶颈。

于是，引入了转换检测缓冲区 TLB，也就是快表，其实就是一个缓存，把经常访问到的内存地址映射存在 TLB 中，因为 TLB 是在 CPU 的 MMU 中的嘛，所以访问起来非常快。

然后，正是因为 TLB 这个东西，导致了进程切换比线程切换慢。

由于进程切换会涉及到虚拟地址空间的切换，这就导致内存中的页表也需要进行切换，一个进程对应一个页表是不假，但是 CPU 中的 TLB 只有一个，页表切换后这个 TLB 就失效了。这样，TLB 在一段时间内肯定是无法被命中的，操作系统就必须去访问内存，那么虚拟地址转换为物理地址就会变慢，表现出来的就是程序运行会变慢。

而线程切换呢，由于不涉及虚拟地址空间的切换，所以也就不存在这个问题了。